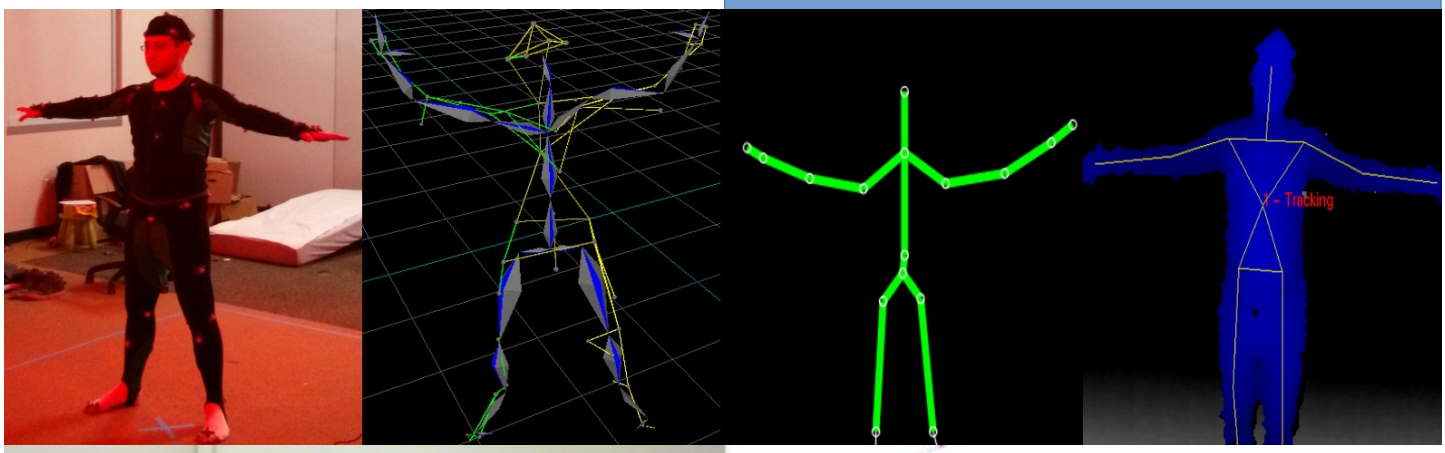
Evaluation of Skeleton Trackers and Gesture Recognition for Human-robot Interaction



Martin Bünger
Fall 2012 – Spring 2013
Vision, Graphics and Interactive Systems, AAU
Robotics and Intelligent Machines, Georgia Tech

AALBORG UNIVERSITY

Department of Architecture, Design & Media Technology

Fredrik Bajers Vej 7, 9220 Aalborg, Denmark

Telephone 99 40 86 00

http://www.sict.aau.dk/

Title: Evaluation of Skeleton Trackers and Gesture Recognition for Human-robot Interaction.

Theme: Master thesis

Project period:
P10, fall 2012 - spring 2013

Project group:
group nr. 1049

Participants:
Martin Bünger

Supervisor:
Thomas B. Moeslund
Henrik I. Christensen

Issue count: 3
Page count: 83
Enclosed: 5 Appendices
Finish date: June 6., 2013

Abstract:

This report is a master thesis in the field Vision, Graphics and Interactive System. It documents the work done during a two semesters visit at Georgia Institute of Technology. The report consists of two parts: An evaluation of skeleton trackers and Gesture recognition.

The evaluation of skeleton trackers compares the two Kinect skeleton trackers from Microsoft and Primesense. The results of the tests shows that their performance is very similar. It is also tested how a simple multi Kinect setup with data fusion can work, which shows slight improvements.

The gesture recognition is based on hidden Markov models and is mainly about recognizing pointing gestures. The test includes a two-class and a multi-class setup, which both yields good results.

This page is intentionally left blank.

This report is the documentation for the project: *Evaluation of Skeleton Trackers and Gesture Recognition for Human-robot Interaction*, that was made during a trip to the Robotics and Intelligent Machines (RIM) laboratory at Georgia Institute of Technology(GT), Atlanta, USA. The trip lasted from September 2012 to May 2013. The report represents a master thesis in computer engineering in the area of Vision, Graphics and Interactive Systems from the School of Information and Communication Technology at Aalborg University in Denmark.

The project was supervised by Dr. Henrik I. Christensen, who is the director of the Center for Robotics and Intelligent Machines, during the visit. In that period Thomas B. Moeslund functioned as a secondary supervisor, after returning home he functioned as the main supervisor.

The two main parts of the report, were done from January to May 2013, while two demonstrations were worked on from October to January. The two main parts are: An evaluation of Kinect API's, see part I, which lead to the article *Evaluation of OpenNI NITE and Kinect SDK Skeleton Trackers for Human-Robot Interaction*, see appendix E and a pointing gesture recognition system, see part II. The two demonstrations were done at GT, which can be seen in appendix B and C.

Acknowledgement

First, I wish to thank Dr. Henrik I. Christensen, the director of the Center for Robotics and Intelligent Machines at Georgia Institute of Technology, for making my visit possible. Also for being a great supervisor and for hosting me in the RIM-lab. I also want to thank my supervisor Thomas B. Moeslund for giving me great advice, while I was abroad.

I met many incredibly welcoming and helpful people at the RIM-lab. One of them was Akansel Cosgun, who is a Graduate Research Assistant at GT. We worked together on the article: *Evaluation of OpenNI NITE and Kinect SDK Skeleton Trackers for Human-Robot Interaction* and on the Kinect part of the second demonstration for the BMW-project. I also want to thank Aaron Bobick, who was the supervisor, with Henrik, on the BMW-project, for advices and guidance.

I also want to thank the people from the RIM-lab: Alexander Trevor, Kimoon Lee, Jake Huckaby, Joan Devassy, Alexander Lambert and Carlos Nieto for always being helpful and made Atlanta feel like home. A thank also goes to the Danish people I met on the trip, especially Søren and Mads, who is also students at Aalborg University.

Last, I really appreciate the people who took time to visit me during my stay: My mom, dad, Søren, Christian and Anders.

Contents

This page is intentionally left blank.

1 Introduction

The assembly line has been under development since the start of the Industrial Revolution, because of the higher and higher demands in the manufacturing process, but also the sheer amount of production. Now a days these high production factories, relies heavily on robots, which can do physical and precision demanding work really well. Still, there is some tasks, that humans can do better and humans are also still required for supervising the robots. Since these high torque robots can be extremely dangerous for people, they are often very strictly divided.

This segregation is made for two reasons. First of all, a robot often does not know how to behave safely around people. Secondly, robots work in controlled environments with very few unknown variables, people tend to introduce these unknown variables.

In modern automotive factories, like the BMW assembly line, there is some tasks, that requires precision and adaptivity, which only a human worker can do. These tasks also requires strength, which raises the physical demands for the worker. Therefore it would be efficient to introduce collaboration between robots and human workers.

The idea of this master thesis is to discover the opportunities of improving the manufacturing process, by letting robots and human workers collaborate along the assembly line.

1.1 Scenario

The BMW-project is a research project at GT, which includes five students and two supervisors. The main purpose of the project is to make a preliminary analysis of how to allow closer collaboration between robots and humans in an industrial environment.

The primary use case from BMW is the installation of the battery in the cars, where workers have to handle a 20 kilogram mass in physical demanding poses. This is where the robots enters the field. The notion is, that the robot will pick up a battery and drive it to the worker, when it is needed. Then the robot needs to give the battery to the worker or assist in installing it, where the worker can provide the fine movement. Having a high powered robot so close to a human

requires high safety, which can be achieved by having complete knowledge of what the worker is doing and the intentions. The scenario is illustrated on figure 1.1, where the workspace can be seen and the stream of cars coming from right.

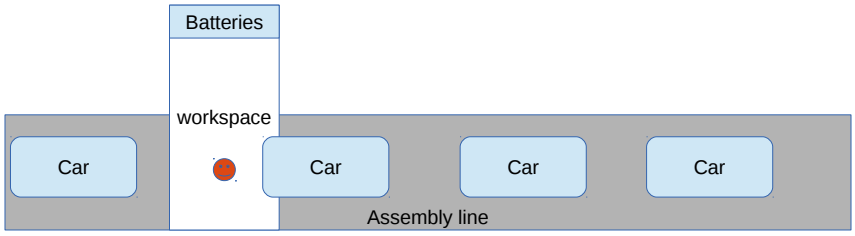


Figure 1.1: The scenario at the BMW factory.

One of the essential parts for making this work, is to have a comprehensive sensor system, which is capable of tracking the worker at all times. Two of the other important parts is the control of the robot and the handling of the information from the sensors.

As mentioned above the scenario contains a lot of really interesting engineering problems. This report will be looking at two problems. First, how to gather information about the area and the worker. Second, how to utilize this information to enable collaboration between the worker and the robot.

1.2 Reading guide

The reading guide consists of multiple sections. First a description of the report structure. Then how citations are made and last, descriptions of the abbreviations used in the report.

1.2.1 Report structure

The report is divided into two parts. The first part is about the data acquisition from the Kinect, which unfolds into a comparison between the skeleton trackers from Microsoft and Primesense. Next is the gesture recognition part, which leads to a pointing gesture recognition system. Both of these parts includes: Analysis, design, implementation and test. The report ends with a conclusion and future work.

Enclosed to the report is a CD, see appendix D, which includes all the code and scripts used in the project. It also includes all data used for the tests. A video of the second demonstration is also included.

1.2.2 Citations

The citations are made after the IEEE standard, where they are represented as a number in brackets, [1]. The references can be found in the end of the report before the appendices.

1.2.3 Abbreviations

The following abbreviations are used frequently throughout this report. When each abbreviation is first introduced into the report, it is presented by this syntax:

...using Point Clouds (PC)..

PC Point Cloud

ROS Robot Operating System

API Application Programming Interface

NITE Natural Interface Middleware

HMM Hidden Markov Model

ICP Iterative Closest Point

SDK Software Development Kit

MS-SDK Microsoft Software Development Kit

FSM Finite State Machine

PDF Probability Density Function

This page is intentionally left blank.

Part I

Kinect Fusion

This page is intentionally left blank.

2 Analysis

This chapter consists of an introduction, an analysis and a discussion about how to gather the data. The first section is an introduction to data gathering and the choice of sensors are made. Next up is an analysis of the chosen sensor and methods for handling the data.

2.1 Introduction

Data acquisition can be done in a variety of ways. There is optical sensors, laser range sensors, RGB sensors, sound sensors etc. In order to determine which sensor to use, it is required to do an analysis of the scenario. This analysis can be seen in section 1.1. Basically the data needed for the system can be defined as *where is it safe for the robot to move*, *where is the worker* and *what is the worker doing*. The main problem for a vision based sensing system is occlusion in this scenario, because there is multiple moving objects in the detection area. This counts the robot, workers and an assembly line with a vehicle. A way to solve this problem is to use multiple cameras in the workspace. For instance if the view of the worker is blocked from one camera, it might be possible to get a view of the worker from another camera. A multiple sensor setup introduces a whole new problem, data fusion.

Data fusion is simply put, to combine measurements from different sensors to achieve a better measurement. Depending on the application, it can be done in different ways. An application can use data fusion to combine data from two different types of sensors or by merging data from multiple sensors of the same type. For example an application could be using both microphone and camera to find the position of a person. Data fusion in this way is nothing new. Humans and animals have always been doing this, combining the view from each eye to get a 3D view. And also combine information from multiple sensors, senses. While it is impossible to see what is behind us, we can hear it.

There is a few issues when designing a data fusion system, which is highly dependable on the choice of sensor. There is quite a few choices of visual sensors, standard RGB cameras, Time of Flight (ToF) cameras and RGB-D cameras. RGB is widely available with a variety of different resolutions, where as a RGB-D sensor usually is stock with a low resolution. The same

statement can be noted between the ToF and the RGB-D sensor, ToF has a better precision than the other. Thus, RGB-D does both jobs worse, than the individual sensors. Work has been done to combine a ToF and RGB sensor into one application [1]. For this system it has been chosen to work with a RGB-D camera, since the combination of RGB and depth information gives a wider range of opportunities.

In this system the sensor will be the RGB-D sensor Microsoft Kinect, and is described further in section 2.2. One of the first problems with multiple sensors is how they affect each other and how to actually fuse the data from them. These problems are a part of the analysis below.

2.2 The Kinect sensor

The Kinect sensor was introduced in November 2010 [2] as a new input method to the Xbox 360. It was introduced as a competitor to the Nintendo Wii and later the PlayStation Move from Sony. Apart from being a gaming console peripheral the Kinect has also served as a depth sensing device for developers. It did not take long for developers to hack the Kinect and start making their own applications with the Kinect [3]. Microsoft apparently realized the potential of the Kinect and launched their own SDK [4]. The reason for the Kinect to be so popular among developers and academics were the low price. Previous to the Kinect, depth sensors were more expensive and therefore were a bigger investment. With the Kinect, everyone can have a depth sensor.

The Kinect technology was developed by the Israeli PrimeSense, under the codename "Project Natal" [5]. PrimeSense have since launched its own SDK and middleware called OpenNI and NITE [6], and is also selling depth sensors.

The Microsoft Kinect is not just a depth sensor, it also has a microphone array for voice recognition and a standard RGB camera. The depth sensing is done by projecting a infrared grid of dots out in the room, called *structured light*. Then a infrared sensor is measuring the shift between the dots and computing the depth [7] [8]. This method has also been tested on visible light, by using a projector and a normal RGB-camera [9].

When using multiple Kinect at the same time, the use of infrared light is causing problems. Interference caused by the multiple projected infrared grid will cause noise in the computed depth [10]. Tests have shown that the interference can be avoided by placing the Kinects in the right way. The interference is greatly reduced by placing the Kinects with a perpendicular viewing angle. The worst scenarios is when the Kinects are facing each other or parallel to each other [11]. Different solutions to the problem have been proposed, in [12] [13] a small mechanical device is shaking the Kinect sensor. The shaking will give the infrared light motion blur, but since the infrared projector and the sensor is both moving simultaneous, the motion

blur is reduced for that exact sensor, but still making it blurry for the other sensors. Another solution [14] is using both the RGB and the Depth images to filter the noise, but the method is only tested on a single Kinect. Another approach to reducing the noise from multiple Kinects is to use *Time division multiple access*(TDMA), where multiple signals can be transmitted on the same channel, by scheduling a specific time slot for each signal to be transmitted in. TDMA is possible to use with multiple Kinects by turning the infrared projector on and off. Unfortunately it is not possible to do this with the current API's as it is, since it is too slow. In [15] [10] a small mechanical modification is made to make it possible to toggle the transmitter.

One really interesting feature on the Kinect platform is the built-in skeleton tracker. Microsoft and OpenNI have their own implementation of this feature. The Microsoft skeleton tracker works by classifying each pixel of the depth image as being part of a joint using heavy trained decision forests [16]. Each decision forest is a gathering of multi-class decision tree classifiers. After this classification the joint positions are computed by using a weighted Gaussian kernel on the joint pixels. The result of this procedure can be seen on figure 4.2. Primesense have not released any information about how their tracker works, but a main difference is that it uses temporal information. Another approach to skeleton tracking from depth imaging is presented in [17]. This approach is based on having a human model, that consists of a kinematic model and a shape model. The depth data is then fitted to the model using a particle filter.

An evaluation of the two skeleton trackers has never been done on the same dataset, because of platform and driver problems, thus the first step of this thesis, is to evaluate which of these sensors gives the best data. This data is the information that needs to be fused. Before looking at the actual fusing, another problem will be covered in section 2.3.

2.3 Multiple Kinect calibration

This section is a discussion about how to calibrate multiple Kinects. The camera calibration process consists of finding the right parameter for the camera. These parameters are divided into two groups, the intrinsic parameters and the extrinsic parameters. The intrinsic parameters includes the focal length and the lens distortion, while the extrinsic parameters tell about the transformation between the camera coordinate system and the world coordinate system. For the intrinsic parameters, the usual checkerboard calibration can be used, both for the RGB and the depth camera.

The extrinsic calibration is the first step in the process of fusing the data, and is defined by being the process of computing the 6D pose transformation between multiple Kinects. This is done by capturing images of an object from all the cameras, and then extracting the same point in each frame. This will give a position of an object from multiple poses and it is then possible to use those positions to find the pose of the cameras. This can be done with the traditional

checkerboard or a QR-code.

In a setup with perfect conditions the extrinsic calibration would yield multiple skeleton joints laying exactly on top of each other. The following section 2.4, Data fusion, will discuss method for solving the problem when the skeleton joints positions is not perfect.

2.4 Data fusion

This section will present different forms of fusion methods. First in form of graphical models, which is based on seeing each joint as a node in a tree-structure. Then a Point cloud (PC) based approach, where the idea is to merge the two PC from each Kinect, and fed it to a skeleton tracker. Last, apply a probabilistic filter, such as the Kalman filter, to the joint data.

2.4.1 Graphical model

A graphical model for solving the multiple Kinect fusion could be made by the same method as in [16]. The method is based on a large amount of decisions trees, which forms decision forests. These forests are fed with a large amount a data in order to optimize the parameters. All this data will adapt all the threshold values in all the branches, in all the trees, in all the forests, to be able to distinguish between different poses. The downside of using this machine learning approach is that it needs a lot of training. The Microsoft system in [16] is using approximately 500.000 samples.

The method mention above can not directly be applied to the skeleton fusion problem, because is it based on pixel intensities. But, by using more human-like features, such as length of limbs and other physical constraints it is possible to make a fusion model, with the capabilities of both figuring out which points to fuse and how to fuse them.

Another approach to the fusion problem is to use the information about the position and the orientation of the person. For instance, it is known [18], that the skeleton information from a Kinect is best if you are facing towards it. Thus, could a model be to weight the values from the facing Kinect higher than others.

2.4.2 Point cloud based

The essential step in data fusion by using Point Cloud is to merge the clouds. This can be done in various ways. The KinectFusion by Microsoft Research [19] [20] is using the *Iterative Closest Point*(ICP) algorithm to merge Point Clouds and compute the transformation between two frames. The downside of the ICP algorithm is, that the Point Clouds must be roughly align to begin with, for it to work. But, if the Kinects is calibrated to operate in the same coordinate frame, then the joints will already be roughly aligned. In this scenario the ICP can be used for

two purposes to iteratively improve the Kinects calibration and to finish the alignment of the joints.

By using the perfectly aligned Point Clouds as input to the skeleton tracker, NITE, it is possible to get a merged skeleton model from multiple Kinects. A PC based method also offers the ability to filter the PC and remove noise or objects in the scene.

The problem with this method is, that the fusion just happens without any considerations about which Kinect actually gives the correct data.

2.4.3 Probabilistic filter

Recursive Bayesian estimation is a probabilistic method to recursively estimate a *Probability Density Function*(PDF). Two well known *Bayes filters* is the *Kalman filter* and the *particle filter*.

Particle filters have been used with the Kinect for tracking before [17] [21], and it also has the capabilities to merge the data in the process. Though particle filters can be used for filtering the Kinect data, it is more viable as a method, when the system is *Non-Gaussian*. Which means that the PDF converges towards more than one local maximum.

The Kalman filter can also be used for data fusion [22]. In [23] two different approaches to Kalman Filter fusion is tested. The two methods are based on filtering before or after the fusion process. If the measurement matrices are identically the methods functions equally, but the second method, filtering after, is more efficient computational-wise. It is also noted, that the first method is best if the measurement matrices are not equally.

2.5 Discussion

After the analysis of the problems regarding data fusion, it is possible to make decisions about what will work best for this specific scenario. The first task is to figure out a way to handle the interference between multiple sensors. Results have shown that the interference can be reduced with a simple *vibrating* device mounted on the sensors. This method will be implemented in the system. Next step is to *prepare* the data for the fusion algorithm, which is done by calibrating the extrinsic parameters of the sensors. This calibration will transform the joints coordinates from each Kinect to a common coordinate system.

To fuse the system three different approaches has been analyzed. The graphical methods offer some very interesting ways to make data fusion based on the model of the human. The point cloud method also has a unique feature in being able to, for instance, remove objects from the the scene, which could reduce the effect of unavoidable occlusions. These occlusions could be

the robot blocking the view of the worker or other objects. The probabilistic methods offers a fusion based on filters. The particle filter is not ideal for tracking and is a better choice for parameter estimation. The Kalman filter is the state of the art filter for tracking, but might be too demanding for tracking 24 joints in a 3D space. Therefor it is chosen to work with both PC and a graphical method.

3 Design

This chapter includes all the thoughts and design decisions made for the skeleton fusion part of the system. First a method for solving the interference problem is presented. Next, the calibration of the Kinects are done. Then the skeleton fusing is described. Last, the test setup and test scenarios are designed.

3.1 Kinect interference

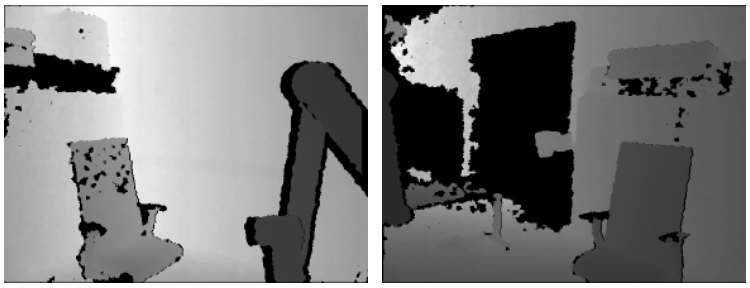
The Kinect interference problem occurs when there is more Kinects pointing at the same scene. The root of the problem is that the IR patterns emitted by the infrared projectors are overlapping and that causes noise in the depth data from the Kinects. On figure 3.1 it can be seen that there is overlap, noise, in the left side of frame 3.1a.

(a) Depth image 0. (b) Depth image 1.

Figure 3.1: The depth frames from two Kinects.

As mentioned in section 2.2 it is possible to reduce the interference by shaking the Kinects. In figure 3.2 the results of a constant tapping on one of the Kinects can be seen. The tapping adds vibrations to the Kinect and thereby adds motion blur to the grid seen by the other Kinect. By comparing the frame in 3.1a and the frame in 3.2a, it is easy to see, that some of the noise is gone.

The Shake'n'Sense [12] introduced in section 2.1 have tested different frequencies for the vibrations, but also suggests a simple solution. The simple solution is to mount a little motor on



(a) Depth image 0. (b) Depth image 1.

Figure 3.2: The depth frames from two Kinects while adding vibrations to one of them.

top of the Kinect with an off-center wheel. This approach will be used for this system, since it is fairly simple to implement and should be able to reduce the interference.

3.2 Kinect calibration

As described in 2.3 the Kinect calibration is two-fold, first the intrinsic parameters are found and then the extrinsic parameters.

Even though the Kinect has a built in intrinsic calibration [24], it is possible to get a better performance by doing a calibration. The intrinsic parameters is found using a camera calibration toolbox for Matlab [25]. The toolbox uses 20 images, of a checkerboard, from a camera to compute the focal length, principal point, skew coefficient and the distortion.

The RGB cameras and the depth cameras can be calibrated using the checkerboard method. When calibrating the depth camera, IR sensor, it is a good idea to use an external source of IR light [24]. This could be sunlight or a halogen lamp.

The extrinsic calibration is the mapping from camera coordinates to world coordinates. It consists of a translation and a rotation matrix. These to matrices can be found by recording different poses of a checkerboard with frames from both Kinects, which gives two set of 3D points. This gives two point sets and the transformation between such two sets can be computed by using the *Singular Value Decomposition* (SVD) [26]. The first step in the process is to find the covariance matrix between the two point sets, *A* and *B*. The *centroid* of each point set is computed by averaging all the points. The computation of covariance matrix *H* can be seen in equation 3.1.

$$H = \sum_{i=1}^{H} \left(P_A^i - centroid_A\right)\left(P_B^i - centroid_B\right)^T \tag{3.1}$$

Next, the SVD is applied to *H* for computing the rotation. Equation 3.2 and 3.3 shows this, *R* is the rotation matrix, describing the rotation between the two sets of points.

$$[U, S, V] = \mathbf{SVD}\,(H) \tag{3.2}$$

$$R = VU^T \tag{3.3}$$

The translation is then computed by adding the rotation to the *centroid* of *A*, and then moved to point set *B* by adding the *centroid* of *B*. This procedure is shown in equation 3.4

$$\mathbf{T} = -\mathbf{R} \cdot centroid_A + centroid_B \tag{3.4}$$

3.2.1 Skeleton fusion

This section will provide two different methods to the skeleton fusion problem.

The NITE framework and Microsoft SDK gives an interesting *confidence* value for each joint. This value can be used for fusion purposes, since we are interested in figuring out, which data is reliable. The *confidence* value is 1, when it gives a reliable position of the joint and 0, when the data is unreliable. Thus, it is possible to make a simple model for the joint fusion by trusting the Kinect with the best values based on confidence. Two different approaches is suggested for using the confidence value, with a common basis, if only one Kinect gives 1, trust that one. If both Kinects gives the value of one, then the first approach is to count the sum of confidence for each Kinect and then trust the Kinect with the highest sum. The other approach is to average the two joints.

The next method is to merge the point clouds and run the NITE skeleton tracker on the new *improved* PC. There is one major disadvantage by using this method, the system does not know if the data is reliable or not. On the other side, a combined PC must be more complete for the person because of the added points from different angles. It is also worth noticing, that it is possible to do some filtering on the new point cloud. The filtering could for instance be to remove the background and other noise-components.

3.3 Test design

This section describes the test setup for the Kinect fusion. As mentioned in section 2.2 there is a platform problem, when comparing Microsoft SDK and NITE to each other. The Microsoft SDK is currently only supported on Windows, while OpenNI and NITE works on all major platforms. This limits the test to be executed on the Windows platform. Choosing the right platform does not solve all the constrictions of this setup, the drivers from Microsoft and OpenNI cannot stream data from a Kinect at the same time. Therefore it is necessary to use a third party software called *kinect-mssdk-openni-bridge* [27], which allows OpenNI to stream data from the Microsoft driver. OpenNI works by creating production notes, these notes could for instance be

a *depth generator* or a *user generator*, with the bridge, it is possible to create a depth node from either Microsoft or OpenNI. In the same way it is possible to create a user node using NITE or using the Microsoft SDK, tough it is not possible to do this at the same time.

Therefore it is chosen to use a setup, where the skeleton tracker from Microsoft is running live and in the same process the depth node is saved to a OpenNI recording, a *.oni*-file. Then in the post-processing the depth node can be streamed and the NITE skeleton tracker can be applied to that data. Thus, both skeleton trackers are running on the same data. This approach can also been used on multiple Kinects, by running the same program multiply times.

To test the performance a sort of ground truth is needed. In the evaluation of another skeleton tracker they used a motion capture system to acquire a ground truth [28]. This will also be the way of acquiring a ground truth for this test, by using a Vicon Motion Capture system [29]. The Vicon system triangulates the position of passive reflective markers, which is usually placed on a suit. When a person is wearing the suite, it is possible calibrate a character in the motion capture software and then compute the position and rotation of each joint. The joint informations can then be streamed to a user application using their SDK.

The whole system then consists of two Kinect applications and the motion capture data. These three processes needs to be synchronized so the frames are matching. This can be achieved by implementing a network connection between the processes. The network connection will send a message out to each processes to tell if it should start or stop capturing data.

As mentioned earlier the test will be conducted in a motion capture lab. The setup of the Kinect will be done as shown on figure 3.3, which shows the Kinect being perpendicular to each other. The test scenarios will be performed at the red dot in the figure, which is 2 meters from each Kinect. This distance is based on the optimal range which is from 0.8-4 meters [30].

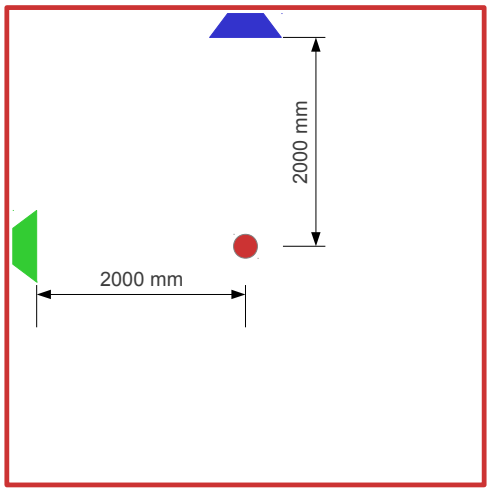


Figure 3.3: The relation between the position of the Kinect.

3.3.1 Test scenarios

This section consists of descriptions of each test scenario. There will be an explanation of how each scenario is played out, and then there will be a reasoning for the scenario.

Walking Walk in a circle in front of the Kinect facing the Kinect with the torso and head all the time. This scenario is for testing basic movement in front of Kinect, without any occlusions.

360 This full body rotation is done with arms out in a T-pose and with the arms a long the side of the body. This is done to evaluate which of the two skeleton trackers are best at tracking under rotation both in range of motion and in picking up the skeleton again.

Hiding arms This scenario is played out by standing in front of the Kinect facing it. Then hiding both arms one by one, and last both at the same time. This will test which system handles hand occlusion best.

Box pick up The box pick up is where the person is standing facing the Kinect and then picking up a package from the floor. The person must then show the package to the Kinect by stretching out both arms. This test will discover what happens when both system are affected by partial occlusions.

Full body occlusion In this scenario the person is going behind a wall to create full body occlusions. The person will go back and forth a few times. This will test how fast the systems will pick up a skeleton again after losing it.

Sitting This scenario is conducted by having a person standing beside a chair and then sitting down, which will be done a few times. This will test which system is best handling a case of a person sitting.

Basic arms Basic arms is done by standing with a 45 degrees angle to the Kinect and then move the arms from a T-pose down along the side and then stretching forward. The arm movement is repeated for every 45 degree in a 90 degrees range for the Kinect.

And for the dual Kinect setup the following scenarios are covered.

360 This scenario is exactly like the one conducted for one Kinect.

Box pick up In this box pick up the box is reached over to the second Kinect.

Basic arms This is done for 180 degrees so the person reaches 45 degrees for both Kinects.

This page is intentionally left blank.

4 Implementation

This section describes how the system designed in 3 is implemented. First a small overview is given with a system flowchart, then each subsystem is described.

4.1 Overview

This section's purpose is to give an overview of the implementation, which can be seen on figure 4.1. As noted in section 3.2.1 the data acquisition has to be implemented in Visual C++, rest of the system is implemented in Matlab.

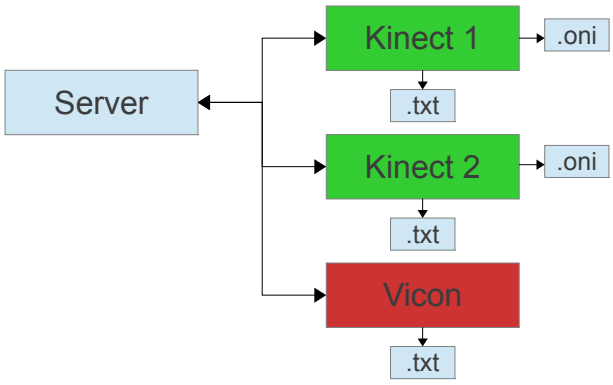


Figure 4.1: Flowchart of the system.

On figure 4.1 it can be seen that the Kinects are delivering data in two ways, writing to a *txt*-file and saving an *oni.*-file. The *txt*-file is the MS-SDK skeleton tracker and the *oni.*-file is meant for the NITE tracker. The two different models and the Vicon model can be seen on figure 4.2

4.2 Multiple Kinect calibration

The camera calibration is done in two steps, as described in 3.2. First the intrinsic parameters is computed for each Kinect using the *camera calibration toolbox* [25]. In 3.2 the extrinsic calibration is explained as finding the transformation between the two Kinects using images of

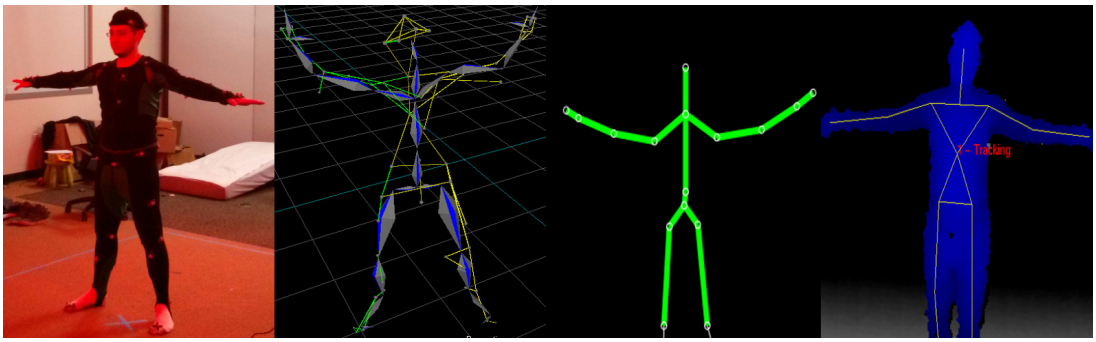


Figure 4.2: The different skeleton models in the system. From left: The Vicon motion capture suit, The skeleton model in the Vicon Blade software, Microsoft skeleton tracker and OpenNI skeleton tracker.

a checkerboard seen by both Kinects. Since a third system, the Vicon motion capture, is introduced another approach is used for the calibration. Instead of calibrating the two Kinects to operate in the same coordinate frame, the two Kinects are separately calibrated to operate in the coordinate frame of the Vicon system. This is achieved by using a checkerboard with reflective markers on it, the checkerboard can be seen in figure 4.3.

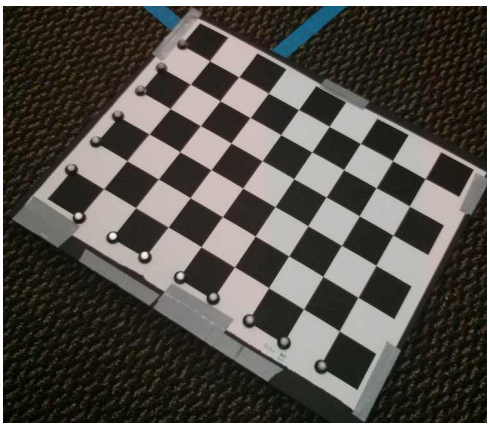


Figure 4.3: Checkerboard used for extrinsic calibration.

The corners can be detected by the Kinect and the markers by the Vicon system. To find the checkerboard corners in the Vicon frame, the markers are used for inferring the positions of the corners, which can be done because of the known dimensions of the checkerboard. Then the SVD-method, as explained in section 3.2, can be applied to get the transformation between the coordinate frames.

4.3 Kinect interference

This section explains how the interference problem from section 3.1 is solved. The solution is implemented by mounting a DC motor on top of the Kinect, with an off-centered weight. The

off-centered weight is tape fastened on the axis, which is sufficient for the desired vibration. A 9V battery is mounted for powering the motor. Figure 4.4 shows the implementation.

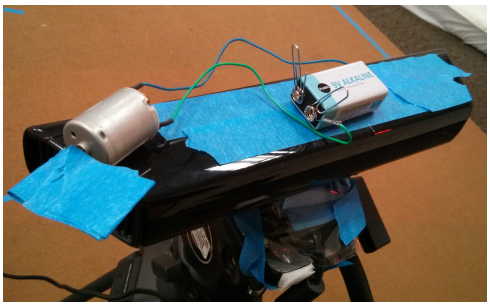


Figure 4.4: The DC motor with off-centered weight and battery.

4.4 System synchronization

This section covers the system synchronization as mentioned in 3.3, and seen in figure . The synchronization is using TCP sockets by using the *Winsock2* API.

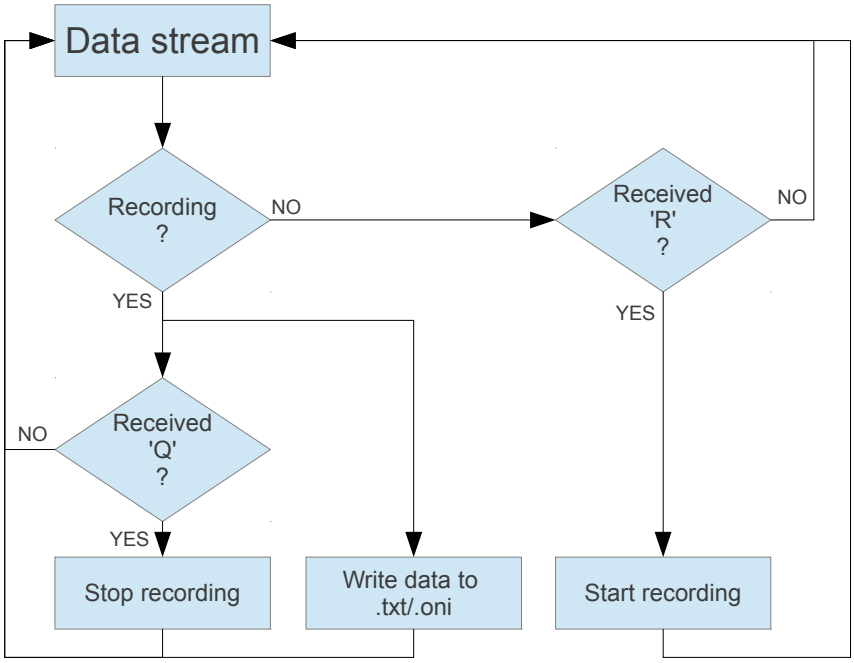


Figure 4.5: The system flow for the synchronization on the clients.

The system consists of one server and three clients. Figure 4.5 shows the system flow for one *client*. The data stream represents the incoming TCP packages from the *server*. The server can send two different messages: *Q*, for stopping the recording, *R* for starting the recording and *A* for saving data to file. The *Q* and *R* is prompted from a user interface, while *A* is continuously send each 33ms, while recording is active.

4.5 Skeleton fusion

In figure 4.6 it is shown how the skeleton fusion is implemented. The data from both Kinects is received, and then for each joint the confidence values are compared. If the values are equal then there is two choices. First, the two joint positions are averaged. Second, the sum of confidence for all joints is computed, and the Kinect with the highest sum is chosen. If the confidences is not equal, the highest one is chosen.

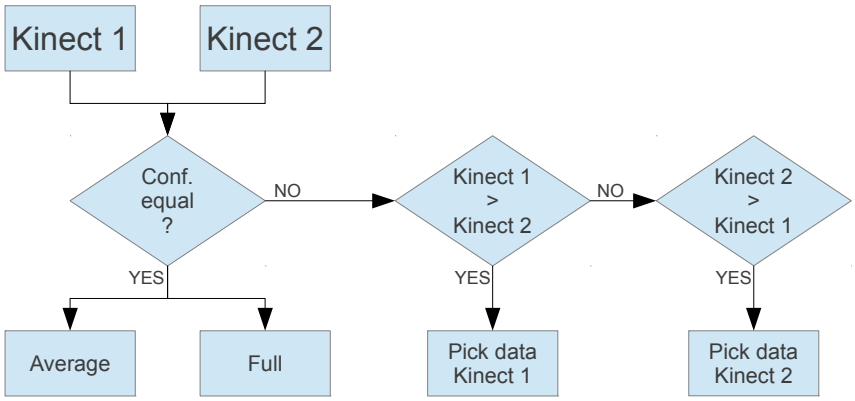


Figure 4.6: The kinect fusioning system.

The point cloud based method described in section 3.2.1 is not implemented due to time constraint and the choice of focusing on comparing the NITE skeleton tracker with the MS-SDK tracker.

5 System test

This chapter is divided into two parts. The evaluation of the Kinect fusion and of the human behavior prediction. The test will be conducted as described in section 3.3. Figure 5.1 shows how the test was conducted, where a person is wearing the motion capture suit, the two Kinects can be seen and some of the motion capture sensors are present at the top.

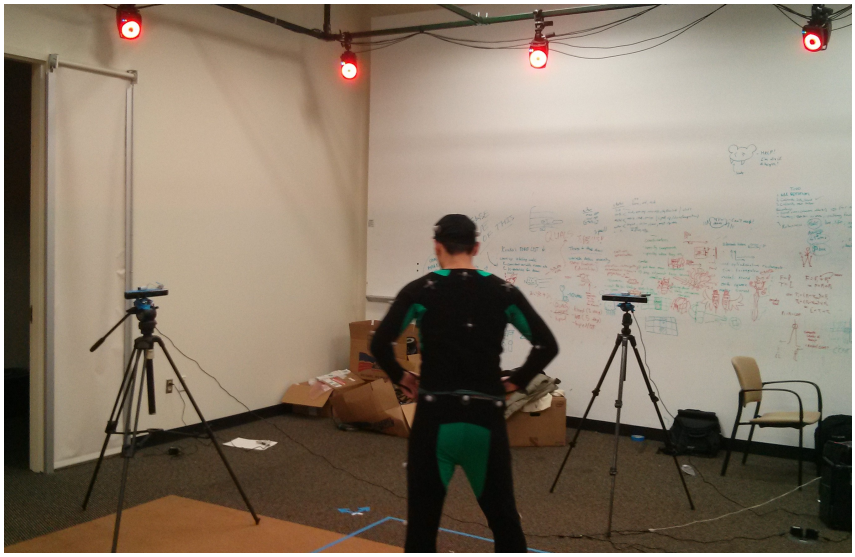


Figure 5.1: A picture of how the test was conducted.

5.1 Kinect fusion

The test of the Kinect fusion is two fold. First the two skeleton trackers, NITE and Microsoft SDK (MS-SDK) is compared using one Kinect. Then the dual Kinect fusion is tested. In the tables below, the name for each scenario is defined as:

- **Walking:** Walk
- **360:** 360
- **Hiding arms:** Hide

Joint	System	Walk	360	Hide	Box	Occ	Sit	Arms
Head	MSSDK	100	100	100	100	66	100	100
	NITE	91	99	93	82	83	68	99
Neck	MSSDK	100	100	100	100	67	100	100
	NITE	100	100	100	100	83	99	100
Torso	MSSDK	100	100	100	100	67	100	100
	NITE	100	100	100	100	83	99	100
Shoulders	MSSDK	100	73	100	100	60	99	96
	NITE	100	100	100	100	83	99	100
Elbows	MSSDK	100	75	100	96	57	74	96
	NITE	99	90	100	71	59	95	100
Hands	MSSDK	100	76	58	48	47	86	95
	NITE	99	88	100	70	58	94	100

Table 5.1: Percentage of tracked position

- **Box pick up:** Box
- **Full body occlusion:** Occ
- **Sitting:** Sit
- **Basic arms:** Arms

5.1.1 Single Kinect

In table 5.1 a large amount of percentages are shown. These values are telling how large of the percentage of the joints are tracked all the time, where tracked means have a confidence of *1*. Joints with a value of 100% are tracked all the time for that scenario. By comparing the values for NITE and MS-SDK it can be noticed that except for the head values, MS-SDK is more prone to lose track of joints. Especially in occlusion scenario.

In table 5.2 the average error of each joint for each scenario is listed, the **bold** values are the best ones. These errors are based on the ground truth from the Vicon motion capture system. By inspecting the values in table 5.2 is can be noticed, that MS-SDK seems better on shoulder, elbows and hands, while NITE is better on head, neck and torso. Also, it is worth noticing at the bad accuracy for the hands when occlusions are present.

A comparison between number in the two tables 5.1 and 5.2 leads to some interesting discoveries. For instance MS-SDK has better accuracy for the occlusion scenario, where it is more prone to throw away values. This is also true for the 360 scenario, which means that the confidence values from MS-SDK is more reliable.

Figure 5.2 shows a comparison of the error rate for the shoulder joint for the walking scenario. It can be seen that the error is changing over time between 6-8 centimeters. For this joint and

Joint	System	Walk	360	Hide	Box	Occ	Sit	Arms
Head	MSSDK	15.8	17.2	16.1	13.2	**32.5**	14.8	16.3
	NITE	**10.6**	**11.8**	**13.3**	**12.2**	76.2	**11.0**	**13.5**
Neck	MSSDK	11.2	14.5	10.9	**8.5**	**31.8**	7.7	11.2
	NITE	**4.6**	**4.9**	**3.2**	10.5	76.6	**5.7**	**5.2**
Torso	MSSDK	**4.4**	**5.9**	**3.9**	**10.1**	**30.7**	**7.5**	**4.2**
	NITE	6.7	6.7	6.8	15.5	82.0	12.1	7.5
Shoulders	MSSDK	7.8	**16.8**	**5.8**	9.3	**34.6**	**7.9**	**6.7**
	NITE	**5.6**	18.6	7.1	**8.7**	82.4	9.2	7.2
Elbows	MSSDK	9.6	**28.6**	7.6	**6.4**	**42.5**	**8.7**	**7.4**
	NITE	**9.0**	32.0	**7.4**	9.1	78.5	11.1	8.3
Hands	MSSDK	14.8	**47.3**	15.6	**12.2**	**52.9**	**14.1**	**12.7**
	NITE	14.8	50.2	**11.0**	15.9	84.7	14.2	14.2

Table 5.2: Error of the two models MS-SDK and NITE. All values are in centimeters.

scenario NITE is performing better.

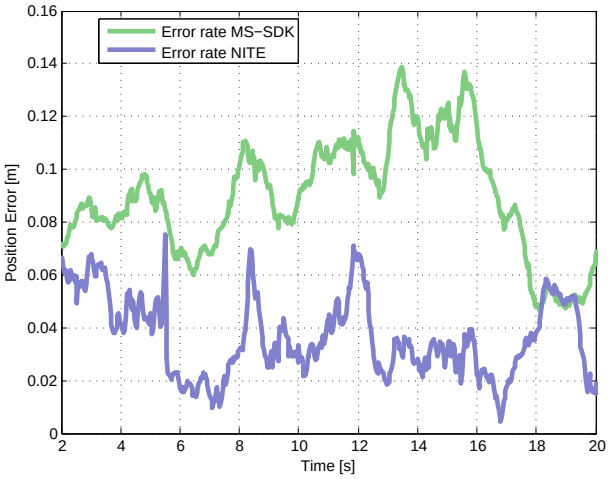


Figure 5.2: Average error in shoulder joint positions for the walking scenario.

Figure 5.3 demonstrates the left hand joint positions in x,y and z coordinates over time in the 360 turn scenario. It is noteworthy that it takes a few seconds for NITE to start tracking and this is because of our data capture procedure. NITE runs on the .oni file offline whereas MS-SDK always runs on the live stream. Therefore it takes some time for the NITE tracker to initiate tracking. First at around t=3s, the person starts to turn around self and this turn ends around t=12s, where another rotation is performed. The second 360 rotation is done with the arms along the side of the body, that is why we can see the Y-position decrease at t=12s. In figure 5.3 the X-position and the Z-position should be similar since it is a rotation around the Y-axis. For the X-position around t=6s, it seems like NITE is in doubt and is giving some really big errors, while MS-SDK stays closer to the real value. Whats happening at around t=7s on the X-position is that the skeleton flips and both MS-SDK and NITE thinks it sees the front of the person instead of the back, that is why the slope looks like the one at around t=10s, where the person is

actually facing the Kinect again. The same phenomenon can also be seen on the second rotation and for the Z-position. For this scenario NITE and MS-SDK seems to be performing equally, although NITE seems to return to the right position faster than MS-SDK. This can be seen on figure 5.3 around t=9s.

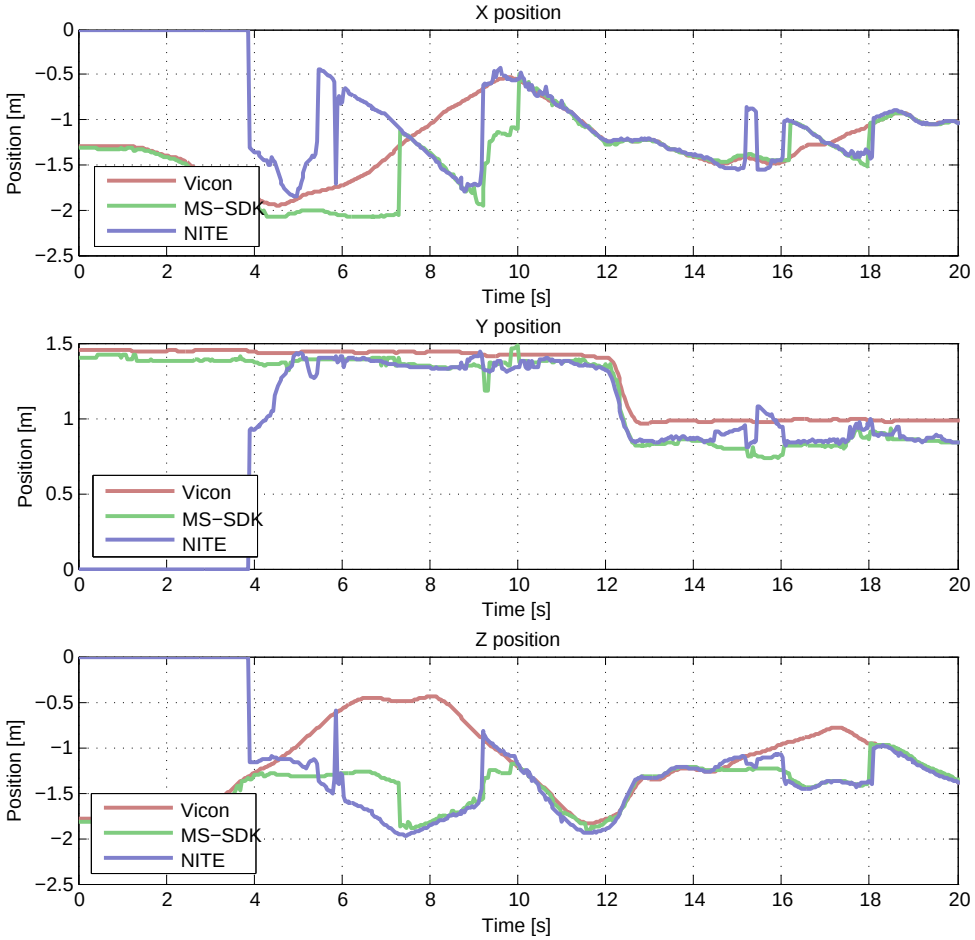


Figure 5.3: Absolute position for the left hand joint.

5.1.2 Dual Kinect

In this section the Kinect fusion as described in section 4.5 is evaluated. The two fusion methods are *Average* and *Sum* as seen in table 5.3, which shows the performance of the two fusion methods and and a single Kinect in the 360 scenario. One of the goals of have Kinect fusion is to increase the range of motion for the tracked person. Therefore it is interesting to compare the values in table 5.1 with the ones in table 5.3. A evaluation of those values shows that their is an increase in the range of rotation that the person can do, while being tracked.

Table 5.4 shows the accuracy of the two Kinect fusion methods, and it is clear that the Average methods is best. While the methods is not clearly better than the single Kinect on accuracy, it

Joint	System	Single Kinect	Dual Avg.	Dual Sum
Head	MSSDK	100	100	100
	NITE	95.5	100	100
Neck	MSSDK	100	100	100
	NITE	100	100	100
Torso	MSSDK	100	100	100
	NITE	100	100	100
Shoulders	MSSDK	76.6	99.4	99.4
	NITE	100	100	100
Elbows	MSSDK	76.4	99.6	99.6
	NITE	92.8	100	100
Hands	MSSDK	83	98.8	98.8
	NITE	91.6	100	100

Table 5.3: Percentage of tracked position

Joint	System	Single Kinect	Dual Avg.	Dual Sum
Head	MSSDK	**18.7**	19.3	19.5
	NITE	**12.97**	14.77	13.75
Neck	MSSDK	**14.57**	16.06	15.89
	NITE	5.59	**5.11**	6.13
Torso	MSSDK	6.44	**6.09**	6.75
	NITE	6.94	**6.24**	6.91
Shoulders	MSSDK	**16.63**	16.75	18.63
	NITE	**15.01**	16.17	17.03
Elbows	MSSDK	**29.03**	29.96	34.29
	NITE	**29.64**	29.93	31.63
Hands	MSSDK	46.24	**44.06**	52.08
	NITE	48.22	**44.79**	50.09

Table 5.4: Error of the two models MS-SDK and NITE for different fusion methods. All values are in centimeters.

has better values for some of the joints.

Figure 5.4 shows the X-position of the left hand joint for the 360 scenario just as in figure 5.3. The upper graph is using a single Kinect while the bottom graph is using the avg. fusion method. Up to around t=7s both systems looks noisy, that is because of the confidences are changing when the person is rotating away from both Kinect. MS-SDK seems to be doing a better job, without filtering the bad confidences. This changes though at around t=11s, where MS-SDK seems to do a better job while using the fusion method, and again around t=23s. With the fusion method NITE still seems to do a better job than MS-SDK on the tracking.

As mentioned in section 4.3 two small shake devices are mounted on each device. They were evaluated by running a test with them on and with them off. The result of this test can be seen on figure 5.5 and it is clear that the vibration actually makes the accuracy worse.

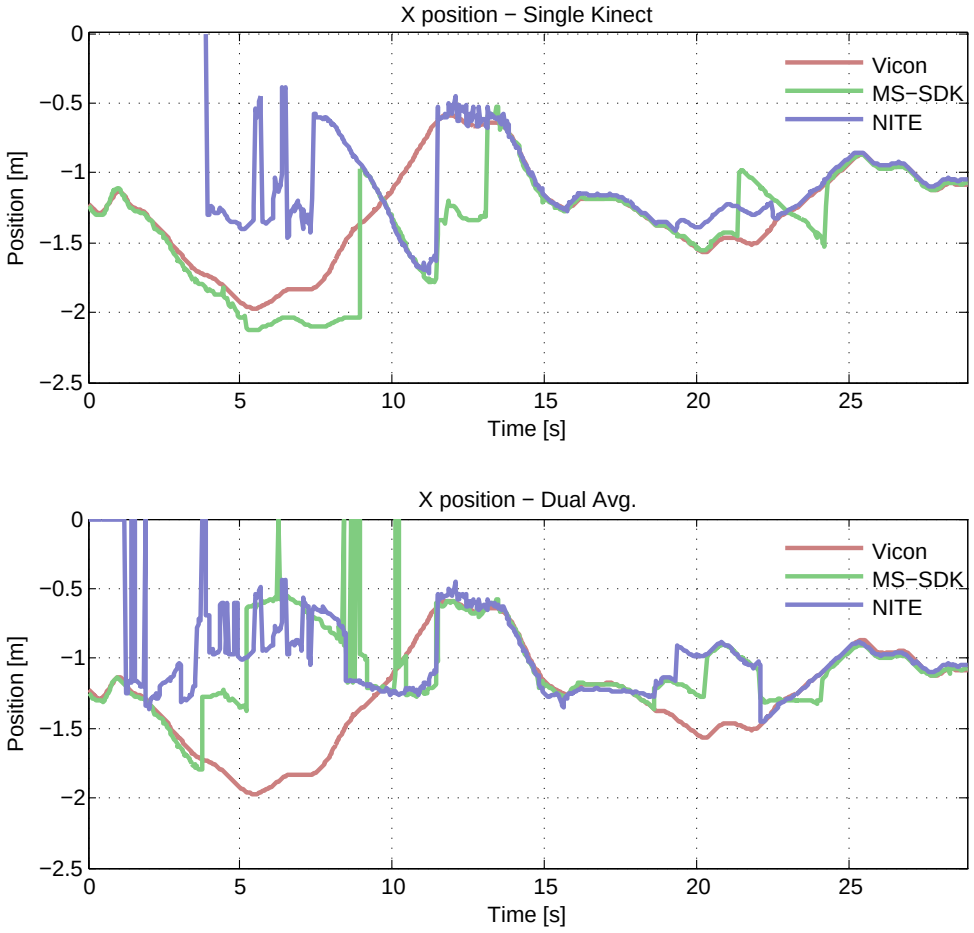


Figure 5.4: Single Kinect and Dual Kinect on the 360 scenario. The x-position tracked is from the left hand.

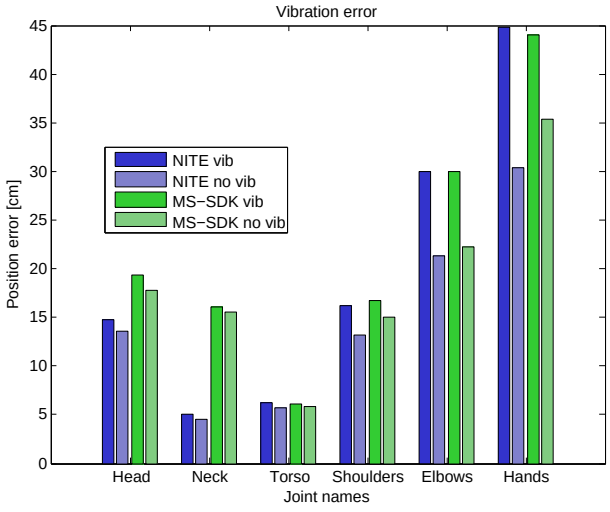


Figure 5.5: Accuracy measured with and without vibration.

5.2 Discussion

The test in this chapter was done as specified in section 3.3. In the test for a single Kinect it was discovered that the tracking performance was highly dependent on the joint. For instance the

NITE performed best tracking on all joint except for the head. Looking at accuracies, it is noted that MS-SDK performance better in some scenarios. For instance the *occlusion* scenario, where the whole body is temporal occluded, this might be caused by the fact, that MS-SDK reads the situation better and stops the tracking before NITE. With this conclusion made, it is interesting that the opposite seems to be happening in figure 5.3, where NITE looks faster at picking up the tracking again. NITE also performs better on the 360 rotation, which can be seen in table 5.1 and 5.3. So, while MS-SDK stops tracking faster and by that achieving higher accuracy, NITE seems to be faster at returning to tracking mode.

For dual Kinects NITE showed better tracking, see table 5.3, but this is also in the 360 scenario, where NITE also was best on a single Kinect. In figure 5.4 it can be seen that both tracking systems is improved by the fusion. The lines that goes up to zero is caused by the fact that both Kinect acquires bad data and the value is then set to zero. This tells that expect for the start the joints are tracked all the time.

The last test was the test of the vibration modules mounted on the Kinects. Here it was discovered that the accuracy was worse with the vibration modules. It is an interesting discovery, because it has been documented, that the vibration effect reduces the noise on the depth images. This noise reduction is apparently not enough to make up for the fact, that a lot of error can occur, if the sensors are moving. First, the calibration will become less accurate, and then the moving sensors will certainly add more noise to the acquired positions.

Another important thing is the different skeleton models in the system. Figure 4.2 in section 4.1 gives an impression on the different models. The problem is, that these three companies, Vicon, Microsoft and PrimeSense, have different definitions of a human skeleton. This is a major concern, when the data from the models is compared to each other, with the idea of computing accuracy. A way to solve this, is to gather the data, and the compute the offset from model to model. If a shoulder point is defined in two different ways, the difference must be measurable. This problem was tried to solve, but it was not possible to find any meaningful offsets, which could be caused by too noisy data.

This page is intentionally left blank.

Part II

Gesture recognition

This page is intentionally left blank.

6 Analysis

In this chapter starts out with an introduction to *human behavior prediction*, which is leading to the topic of gesture recognition. Next up is a discussion of what a gesture is and how it is defined in this report. Next is an analysis of how to detect gestures. Last, a discussion, where the analysis leads to a basis for a solution to the problem.

6.1 Introduction

The human behavior estimation is directly based on the data from the Kinect fusion. The goal is to deliver a model of the human to the robot. This model can basically be anything from just a position of the human, to a more advanced model of where the person is looking and what the person is doing. The first part of this chapter about human behavior prediction is a analysis of that kind of model the system should be using. To figure out how to model the system, one should ask questions about the scenario.

1. Where is the persons attention ?
2. Where is it safe for the person to move ?
3. What is the person doing ?
4. What action can the robot do, to be most helpful ?

All these questions are based on estimating information or behavior about the person. But even if it is possible to answer all the questions above, the robot still might be uncertain about a situation. This uncertainty could be solved by introducing a way for the worker to communicate with the robot. Ways of doing this could be some sort of physical interface, voice commands or gestures. Gestures is an interesting communication channel, since the Kinect gives a lot of informations about the state of the arms of the worker. Also, it will be more robust than voice commands in a loud workspace, like the assembly line at a car factory.

In Human-robot interaction gestures can be really useful. For instance to tell the robot to *stop*, *rotate* or *move*.

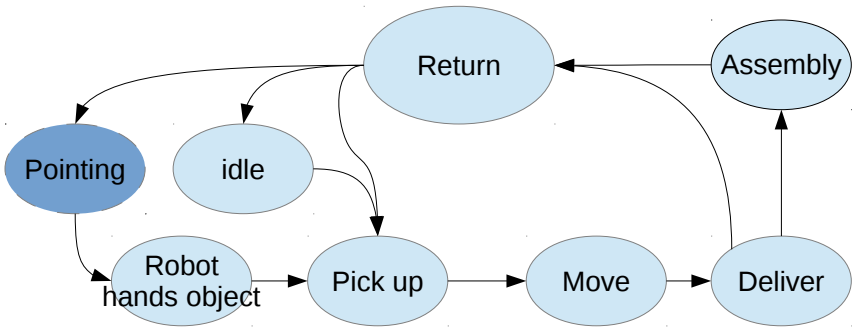


Figure 6.1: The states that the worker can be in.

Figure 6.1 shows a scenario taken from the description in section 1.1, where a gesture is incorporated. In this scenario the worker has two different ways to acquire the object, a battery, and then the workers proceeds to move to the car and do the assembling. The worker can acquire the battery by picking it up, or pointing to the object to signal the robot to deliver it.

The next section will elaborate on what a pointing gesture is and how others have defined it. After that methods for general gesture recognition and pointing gestures will be analyzed.

6.2 Gestures

Gestures is the standard non-verbal communication form for humans and have existed since the modern humans were born. Gestures are used all the time while communicating with others, in form of either facial expressions or movement with hands. The sign language is a complete language with grammar and a vocabulary, and is purely based on gestures. It has also been proven, that gestures is on of the first steps for learning to speak [31]. The skeleton tracked by the Kinect gives a good model for arms movement and is a great foundation for gesture-based communication for the system.

A gesture can be decomposed into three stages: *preparation*, *stroke* and *retraction*, with *stroke* as the most important stage [32]. It might be true, that the *stroke* is important because of the amount of information, but a gesture is still the sum of the stages. For instance, a kick, if the gesture is stopped after the *stroke*, the leg is raised, it will be ambiguous. First, how can it be known at which height to stop the gesture. This is why the *retraction* is important.

Gestures are, just like for instance hand writing, unique for an individual person. One of the most well known gestures, the wave, can be done in many different ways. Some people only bend the fingers, while others are only rotating the wrist.

This is also very true for pointing gestures, and is also what makes gesture recognition a non-trivial problem. A pointing gesture can be done in many ways, straight arm or bended elbow joint, and some people also use the entire hand instead of only the index finger. Three different ways of pointing is described in [33], which is: pointing with forearm, using straight arm or by lining up eyes and fingertip.

To narrow down the range of motions for a pointing gesture, it is decided that a pointing gesture is done with an almost straight arm, close to no bending in the elbow joint. Figure 6.2 shows what the pointing gesture can look like. The three stages then becomes: *arms in resting position*, *moving arm* and *holding arm in pointing position*.

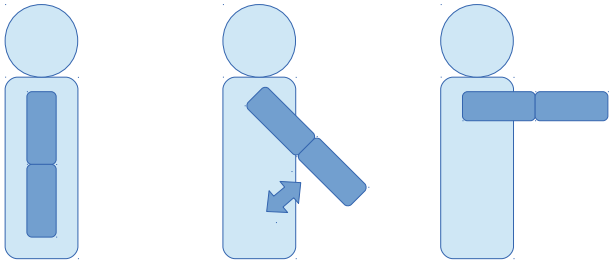


Figure 6.2: The three stage pointing gesture.

6.3 Methods

Detecting human activities from video streams is one of the most researched topics in Computer Vision, which have concluded in a few surveys [34] [35]. As mentioned earlier gestures are an interesting activity for recognition systems, because of the high variety in the execution of the gesture. A few surveys on the topics has been written [36], where it is noted that many have been using methods as Hidden Markov Models and finite state machines. Many well known methods: *Principal Component Analysis*, *Kalman filtering* and *Particle Filtering*, can be coupled with discrimination algorithms to achieve gesture recognition.

Hidden Markov Models (HMM) is a very popular state-space model. The hidden states in HMM is what makes it different from the standard Markov model. Instead of observing the states, output for the states is observed. HMM works by training a model with sequences and then it gives a likelihood, for how good a new sequence fits the model. This makes it able to adapt to almost any gestures and sequences, given the optimal observed values for the gesture. HMM is very well documented as on of the best methods for gesture recognition [37] [38] [39] [40] [41] [42].

One of the first examples of HMM recognizing human action is in [41], where the HMM's are recognizing different tennis strokes. Others have been using HMM's to train different vocabularies for communication [42], which has lead to a real-time American sign language recognition system [40]. In [39] complex *Tai Chi*, a Chinese martial art form, actions are recognized

using standard, linked, and coupled HMM's. These merged models are made by introducing a conditional probability between their hidden state variable. Another type is parametric HMM's, which is based on the idea of extracting information from the gesture. This is done with output densities, which are functions of a gesture parameter vector. This enables the model to return for instance the pointing direction [37]. One of the difficulties with HMM's are the recognition of a non-gesture, since the models usually are only based on actual gesture examples. A method for modeling the threshold of the likelihood value, is applied to solve this problem [38].

Another state based method is the finite state machines (FSM) [36], actually the HMM is a type of FSM. This implies that they are familiar. For FSM each state is represented as a parameter vector, which includes probabilistic parameters and a duration interval. One of the main differences is the fact that the states are hidden in the HMM, while everything is observable for FSM [43] [44].

Conditional random fields (CRF) have also been applied to gesture recognition tasks. The CRF is based on discriminative learning, which based on the idea of learning to distinguish between multiple models. This is very different from the generative learning in HMM, where the models are taught how a model looks, but not how it compares to other models. In [45] [46] different gesture recognition tasks are tested on both HMM and CRF. HMM performance better in both and is also stated faster in one of them. In [47] a hidden CRF is modeled and performance better than a HMM.

As mentioned earlier a lot of popular feature reduction methods and filtering methods can be combined with the discrimination methods above. The principal component analysis is a way of transforming a feature set onto a lower-dimensional manifold. In gesture recognition it works by creating templates for each gesture and new gestures can then be compared and recognized [48]. The Kalman filter is a state of the art tracking algorithm for multiple purposes, it works by estimating parameters by using previous noisy inputs. The Kalman filter is not useful alone for gesture recognition, but works great in combination with for instance HMM's [49]. The particle filter is a probabilistic filter like the Kalman filter. The idea of the filter is to represent probability densities with an amount of particles. These particles are sampled and weighted based on the observations, which in tracking scenarios often are representing location and velocity [50].

6.4 Discussion

Based on the analysis of gesture recognition and pointing gestures in the previous sections, it is possible to decided how to solve the problems for this system. First, it was decided to look for gestures to improve the collaboration between robots and workers in at the BMW assembly line. It was decided to define a pointing gesture as starting from a resting position, to raise the arm without bending in the elbow joint and the return to the resting position.

For detection the gesture a few similar solutions were described. FSM, HMM and CRF are all state based discrimination algorithms, which have all proven to be able to do gesture recognition. In tests comparing the algorithms it seems like HMM's are best, which is the reason for choosing it for this system.

This page is intentionally left blank.

7 Design

As described in chapter 6 the human behavior estimation is based on gesture recognition by using HMM's. The gesture, the system should be designed to recognize, is a pointing gesture described in section 6.2. HMM's are based on observations and transitions probabilities, that triggers the action of changing states. The states that a person is going through doing a pointing gesture can be loosely defined previous to the training of the HMM, but there is no guarantee that the HMM will fit the data like that. This concept and a lot more about HMM's will be discussed in section 7.2.

The observations are based on the skeleton data computed by the Kinect fusion mentioned in part I. To pick out the best observations a few questions should be answered. Which joints are interesting for the given gesture? Which information from each joint is interesting for the gesture? Are these informations robust from person to person? These questions will be answered in section 7.1.

7.1 Data analysis

A HMM is using observations to compute which state to be in. These observations are the inputs values from the sensor in the system, the Kinects. With the skeleton tracker for the Kinects it is possible to receive precise information about each joint. This information is the orientation, the position and the confidence of the values. Before deciding which informations to look for, it is important to make an analysis the given gesture and discover which informations are relevant.

As mentioned in section 6.2, the pointing gesture is defined as a straight arm lift to reduce the complexity of modeling the variety of pointing with a bended arm. Even though the modeling of the arms should be equal, it is decided to only use right arm gestures for this system. It is still important to observe both arms, since movement of the left arm might indicate another gesture than pointing. As mentioned in section 6.3, a HMM is using states, which is interesting to analyze, in order to figure out, which types of observations would work best. The states are based on the gesture definition from section 6.2, where the gesture was decomposed into three stages: *preparation*, *stroke* and *retraction*.

To be able to detect when the gesture is happens, when it is ongoing, and when it is done, the observations is divided into three states. So, the gesture might consist of these three states:

- **State 1:** Low angle for the arm, describing that the arm is in a resting position, along the side of the body. Close to zero velocity for the angles since the arm should be steady.
- **State 2:** High variety in the angles and a high velocity. This indicates that the arm is moving to the pointing position.
- **State 3:** High angle for the arm, which means that the arm is in a non-resting position. The velocities should be close to zero.

Figure 7.1 shows how those three states would look in a state machine. The arrows symbolizes the state transitions. A gesture is recognized if the state sequence is in form of: *State 1*, *State 2*, *State 3*, *State 2* and *State 1*.

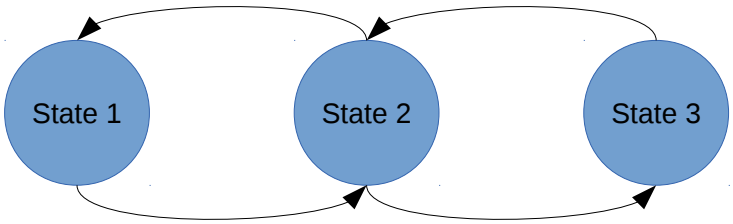


Figure 7.1: The three states and the transitions.

In the state description angles and velocities are mentioned, which is also recommended in a survey, studying features for gesture recognition, where usage of velocities and angles are recommended for recognition rather than raw Cartesian coordinates [51]. Using velocities instead of Cartesian coordinates makes the observations invariant to rotations, which makes the system able to detect pointing gestures in a variety of directions. The angles are still important since it keeps track of whether the arm is raised or not. Also, it might be useful to add the angle between the overarm and forearm to detect if the arm is straight.

It is now known which informations to look for, and as described in chapter 7 a Kinect is delivering the information. Figure 7.2 shows a sequence of multiple pointing gestures and the *xyz*-orientations for the shoulder joint. The orientations of the joint is described in quaternions, which is a common way of representing rotations in robotics. A quaternions consists of the three usual components *xyz* and *w*, *xyz* defines the rotational axis and *w* defines the amount. The figure 7.2 shows seven pointing gestures, which can be observed on the graph by looking at the *z* and *w* values. Graphs for the other joints looks similar to this one, which leads to the notion of computing angles from the joint positions instead. The problem with the values in figure 7.2 is, that they are not completely clear and also, they do not give the best representation for a

pointing gesture.

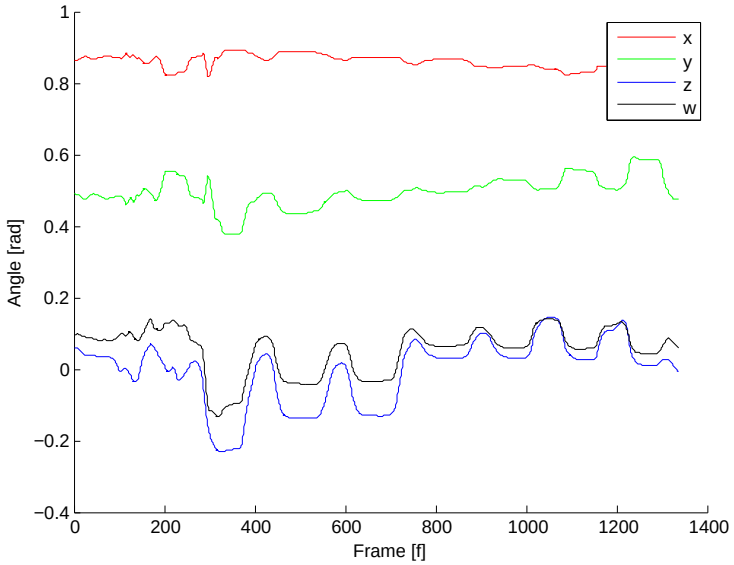


Figure 7.2: Orientations for the shoulder joint. The seven pointing gestures happens approximately at: $f = 300$, $f = 450$, $f = 600$, $f = 750$, $f = 900$, $f = 1050$ and $f = 1200$.

As mentioned above the pointing gesture consist of a straight arm raise from resting position. A good way of representing this gesture could be to compute the arm angle in respect to the human body e.g. a vertical line. This would make the angle invariant to different direction of pointing, but variant to how high the person is pointing. The computation of this angle can be seen in equation 7.1, where p_S and p_E is to 3D position of the shoulder and elbow respectively. $\vec{v}$ is a vertical vector defined as $[0, 1, 0]^T$. To verify that the elbow joint is straight, an angle between the upper arm and forearm is computed, which is done by computing the forearm angle the same way as the upper arm and then the two are subtracted. The $\vec{j}$ is in this case computed by subtracting the wrist point from the elbow point.

$$\begin{aligned} \vec{j} &= p_S - p_E \\ \alpha &= acos\left(\frac{\vec{v} \cdot \vec{j}^T}{|\vec{v}||\vec{j}|}\right) \end{aligned} \tag{7.1}$$

The new angles can be seen on the graph in figure 7.3 and it can clearly be seen, when the gestures are happening. The blue curve is the upper arm angle, which is the one showing how high the arm is raised. Up to frame number 200 the person is doing some calibration movements to ensure, that the Kinect is tracking the person. The figure shows seven pointing gestures, where the four first is in front of the person in similar heights. The last three gestures are all in ascending heights, which can be seen on the graph. During a pointing gesture it is expected that the entire left arm are still, which is seen on the graph. The positions for the right arm gets noisy during movement, and that is why the graph shows that the right elbow joint is not stretched.

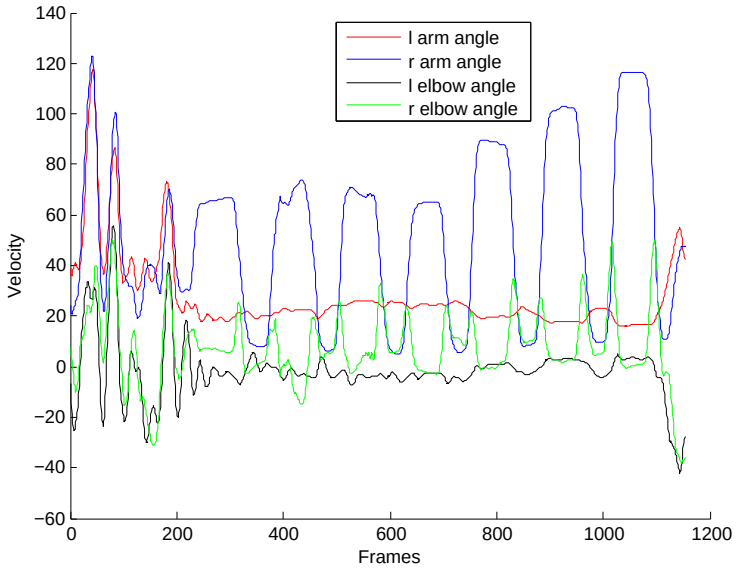


Figure 7.3: New arm angle.

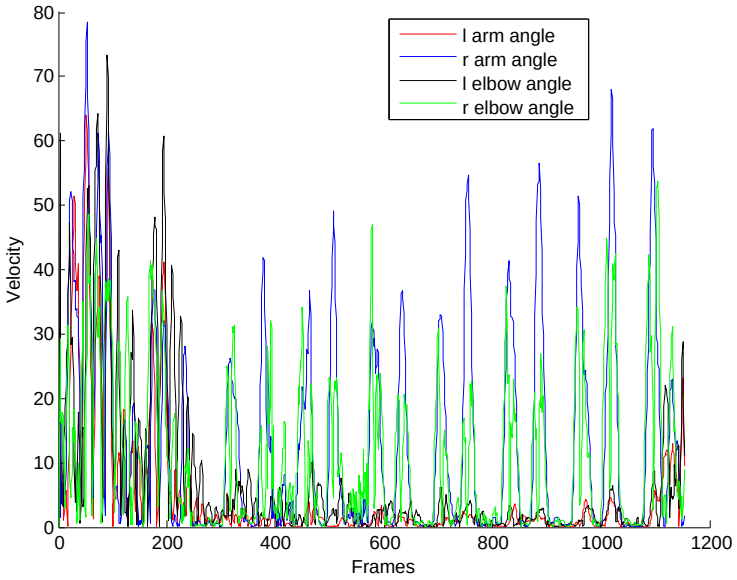


Figure 7.4: Velocity of the arm angles.

The velocities of the angles are computed with the formula in equation 7.2 and can be seen on the graph in figure 7.4. The graph shows that the velocities greatly increases when the arm is moving, which can be seen by comparing figure 7.4 with figure 7.3. Equation 7.2 is a simple average computation coupled with an amplification by an multiplying with ten.

$$v(n) = \frac{|\alpha(n) - \alpha(n-1)|}{0,2} \tag{7.2}$$

To summarize, it was investigated which observations would be best for detecting a pointing gesture. It was discovered that the orientations directly from the Kinect could not be used. Instead a new angle, based on the dot product between the arm and the vertical axis, is used.

This method is also used for computing the angle of the elbow joint. These angle makes a basis for the computation of velocities.

7.2 Hidden Markov Models

As mentioned in chapter 6.3, Hidden Markov Models, are popular models for temporal and sequence data. A simple way to explain how the HMM works is to think of a black box. First, an amount of sequences are shown to the box, which teaches the box what the sequences can look like. Then, a new sequence can be shown to the box and the box will give a value of how similar the new sequence is to the ones shown.

The HMM is derived from the standard Markov models, which are finite-state machines with transition probabilities. Therefore the HMM inherits the Markov property, which is, that the estimation for the next state only depends on the current state. The HMM is introduced when it is not possible to directly observe those states, but instead it is possible to observe some output from those states. The description in this section is based on *An introduction to hidden Markov models* by L. Rabiner [52].

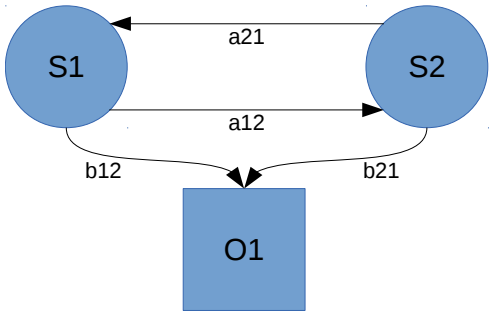


Figure 7.5: A basic HMM.

The HMM is defined by the equation seen in 7.3, where *N* is the number of states. *M* is the number of observation signals per state, for instance a coin would have two, heads and tails. *A* and *B* are both matrices. *A* is the state transition probabilities and *B* is the observation symbol probabilities. π is the initial state distribution. Though it is usually reduced to the compact notation seen in equation 7.4. A simple HMM is illustrated on figure 7.5, where a_{12} is the probability of a transition to S_2. The observation probability b_{12} is the probability of observing O_1, while in S_1.

$$\lambda = (N, M, A, B, \pi) \tag{7.3}$$

$$\lambda = (A, B, \pi) \tag{7.4}$$

When applying the HMM to real applications three problems occurs, which has to be solved. The three problems are:

1. **Evaluation:** Given a HMM model, what is the probability that a output sequence is produced by the model. In other words if the output sequence is given as $O = O_1 O_2 \dots O_T$ and the model is $\lambda = (A, B, \pi)$, how can $P(O|\lambda)$ be computed. This problem is solved by using a form for dynamical programming, namely the *Forward algorithm*.

2. **Decoding:** Given a model λ and a observation sequence O, how is it possible to find the optimal state sequence, $Q = Q_1 Q_2 \dots Q_T$. In other words, find the state sequence which is most likely to have generated the observation sequence O. This can be solved using the *Viterbi algorithm*.

3. **Learning:** Given a observation sequence or a set of those, how can the model parameters A, B and π be adjusted to maximize $P(O|\lambda)$. This is solved using the *Baum-Welch algorithm*

The three problems is what defines the features of a Hidden Markov Model. First the model is taught how the sequence should look like (Problem 3), and then the model can be analyzed (Problem 2). Finally, a new sequence can be classified (Problem 1).

7.2.1 Evaluation

The evaluation problem is to compute the probability, that the new observation was produced by the model. The problem can also be described as, how well the model fits the new observation, which is a better description of the problem given multiple models. The problem is to compute the probability of a given observation sequence, $O = O_1 O_2 \cdots O_T$, given a model, λ, $P(O|\lambda)$.

The easiest way to do this is to look through all the state sequences, with length the same as the observations sequence, which is given by $Q = q_1 q_2 \cdots q_T$. Then the probability of O given Q and λ is computed with the equation in 7.5

$$P(O|Q, \lambda) = \prod_{t=1}^{T} P(O_t|q_t, \lambda) \tag{7.5}$$

In 7.5 it is assumed that the observations are statistically independent, which gives equation 7.6. Here it can be seen, that the probability of a observation sequence given a state sequence and a model, is given by the probability of observing O_n while being in state q_n.

$$P(O|Q, \lambda) = b_{q_1}(O_1) \cdot b_{q_2}(O_2) \cdots b_{q_T}(O_T) \tag{7.6}$$

The probability that one specific state sequence will occur can be computed by equation 7.7, where π is the initial probability e.g. which state to start in and a is the transition probabilities.

$$P\left(Q|\lambda\right) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} \cdots a_{q_{T-1} q_T} \tag{7.7}$$

To get the probability that a given observation sequence O and a state sequence Q is happening at the same time, the product between 7.6 and 7.7 is computed. This is shown in equation 7.8.

$$P\left(O, Q|\lambda\right) = P\left(O|Q, \lambda\right) P\left(Q|\lambda\right) \tag{7.8}$$

Now, the probability for a observation sequence given a model can be computed as listen in equation 7.9.

$$\begin{aligned} P\left(O|\lambda\right) &= \sum_{allQ} P\left(O|Q, \lambda\right) P\left(Q|\lambda\right) \\ &= \sum_{q_1, q_2, \cdots, q_T} \pi_{q_1} b_{q_1}(O_1) a_{q_1 q_2} b_{q_2}(O_2) \cdots a_{q_{T-1} q_T} b_{q_T}(O_T) \end{aligned} \tag{7.9}$$

Equation 7.9 is a sum of joint probability over all possible state sequences, which is a lot of computations. For instance for a model with 5 states and a sequence length of 100, $2 \cdot 100 \cdot 5^{100} \approx 10^{72}$ computations are needed. This is clearly not a good solution, luckily an algorithm called Forward-backward can do this better. The algorithm is using the forward variable called $\alpha_t(i)$, which is defined as seen in equation 7.10.

$$\alpha_t(i) = P\left(O_1 O_2 \cdots O_t, q_t = S_i|\lambda\right) \tag{7.10}$$

The forward variable is the probability for being in state i at time t given the model λ. This can be solved inductively by going through the three steps:

1. Initialization

$$\alpha_t(i) = \pi_i b_i(O)_1, \qquad 1 \leq i \leq N \tag{7.11}$$

2. Induction

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}), \qquad 1 \leq t \leq T-1 \qquad 1 \leq j \leq N \tag{7.12}$$

3. Termination

$$P\left(O|\lambda\right) = \sum_{i=1}^{N} \alpha_T(i) \tag{7.13}$$

In equation 7.11 the forward variable is initialized, with the initial probabilities and the probability of observing O_1. Next up is equation 7.12, which is the induction step. Here the term $\alpha_t(i) a_{ij}$ is the probability of joint event that the sequence $O_1 O_2 \cdots O_T$ is observed and that the state S_j is reached at time $t+1$, via state S_i at time t. The sum of this term results in the

probability of S_j at time $t+1$, with respect to all the previous observations. Then, for computing $\alpha_{t+1}(j)$, the sum is multiplied with the probability of observing O_{t+1} in state j. The final probability is computed by summing the forward variables as seen in equation 7.13. With the forward variable it is possible to solve the problem with 3000 computations instead of $\approx 10^{72}$.

7.2.2 Decoding

The decoding problem is different from the evaluation problem, because there is no exact solution to the problem. It can be solved in many different ways and it all depends on what is meant by the *optimal* state sequence given a observation sequence. One possible solution could be to pick the states q_t that are individually most likely. This optimal criterion is then based on maximizing the expected number of correct individual states. To solve this problem the γ-variable is needed, which is defined in equation 7.14.

$$\gamma_t(i) = P\left(q_t = S_i | O, \lambda\right) \tag{7.14}$$

The γ-variable is the probability of at time t being in state S_i given the observation sequence O and the model λ. Equation 7.14 can be rewritten using the forward-backward variables as seen in equation 7.15.

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P\left(O|\lambda\right)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \tag{7.15}$$

In equation 7.15 the forward variable α includes the partial observation probabilities for $O_1 O_2 \cdots O_T$ and state S_i at t. The backward variable β is including probability of the observation sequence $O_{t+1}O_{t+2}\cdots O_T$ given state S_i at t. $P(O|\lambda)$ is a normalization factor to ensure that equation 7.16 is true.

$$\sum_{i=1}^{N} \gamma_t(i) = 1 \tag{7.16}$$

Before using the γ-variable, the backward variable is defined as seen in equation 7.17, where it can be noted, that it is the probability for the partial observation sequence from $t+1$ to the end, given the model λ and the state S_i at time t.

$$\beta_t^i = P\left(O_{t+1}O_{t+2}\cdots O_T | q_t = S_i, \lambda\right) \tag{7.17}$$

Like the forward variable, the backward variable is based on induction. Before that, the variable is initialized. The procedure can be seen in equation 7.18 and 7.19.

1. Initialization

$$\beta_T(i) = 1, \qquad 1 \leq i \leq N \tag{7.18}$$

2. Induction

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t-1}) \beta_{t+1}(j), \qquad t = T-1, T-2, \cdots, 1, \quad 1 \leq i \leq N \tag{7.19}$$

In the initialization step, equation 7.18, $\beta_T(i)$ is defined to be 1 for all i. The induction step in equation 7.19 computes the probability of have been in S_i at time t taking into account the observation sequence from time $t+1$. This is done by considering all possible state transitions at time $t+1$, the state transition from S_i to S_j, the observation O_{t+1} in state j, and accounting for the remaining partial observation sequences, $\beta_{t+1}(j)$.

Back to the γ-variable and the decoding problem. To find the most likely, individually, state q_t at time t equation 7.20 can be applied.

$$q_t = \underset{1 \leq i \leq N}{\arg\max} \left[\gamma_t(i)\right], \qquad 1 \leq t \leq T \tag{7.20}$$

Even though this method maximizes the expected number of correct states, the result might not be valid. This is because the algorithm does not validate the resulting state sequence and it can be wrong, since it is not always possible to transition from a given state to another. The solution to this is to change the optimal criterion to maximize $P\left(Q|O, \lambda\right)$ i.e. find the best state sequence given the observations and the model. This is equal to maximizing $P\left(Q, O|\lambda\right)$, which can be solved by the Viterbi algorithm. The Viterbi algorithm can find the best state sequence given a observations sequence. It is done using the quantity δ as seen in equation 7.21.

$$\delta_t(i) = \max_{q_1, q_2 \cdots, q_{t-1}} P\left[q_1 q_2 \cdots q_t = i, O_1 O_2 \cdots O_t | \lambda\right] \tag{7.21}$$

It can be seen that $\delta_i(t)$ is the high probability for a single path at time t, which includes the first t observations and ends in state S_i. Induction gives the equation 7.22.

$$\delta_{t+1}(j) = \left[\max_i \quad \delta_t(i) a_{ij}\right] \cdot b_j\left(O_{t+1}\right) \tag{7.22}$$

Then a variable is needed for saving each of the state sequences for each t and j, which is the array $\psi_t(j)$. The Viterbi algorithm consists of the following four steps:

1. Initialization

$$\delta_1(i) = \pi_i b_i(O_1), \qquad 1 \leq i \leq N \tag{7.23}$$

$$\psi_1(i) = 0 \tag{7.24}$$

2. Recursion

$$\delta_t(j) = \max_{1 \le i \le N} \left[\delta_{t-1}(i) a_{ij}\right] \cdot b_j\left(O_t\right), \qquad 2 \le t \le T, \quad 1 \le j \le N \tag{7.25}$$

$$\psi_t(j) = \arg\max_{1 \le i \le N} \left[\delta_{t-1}(i) a_{ij}\right], \qquad 2 \le t \le T, \quad 1 \le j \le N \tag{7.26}$$

3. Termination

$$P^* = \max_{1 \le i \le N} \left[\delta_T(i)\right] \tag{7.27}$$

$$q_T^* = \arg\max_{1 \le i \le N} \left[\delta_T(i)\right] \tag{7.28}$$

4. Path backtracking

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \qquad t = T-1, T-2, \cdots, 1 \tag{7.29}$$

The Viterbi algorithm is similar to the forward algorithm seen in equation 7.11-7.13 expect for the maximization, where the forward algorithm is using summing.

7.2.3 Learning

The last problem to solve is the learning problem. The problem is to optimize the model parameters (A, B, π) to maximize the probability of a observation sequence given the model. There is no known analytical way of solving this problem, instead it can be solved for a local maxima for $P(O|\lambda)$ with the Baum-Welch algorithm. First step is to define the ξ-variable, which is the probability of being in state S_i at time t and state S_j at time $t+1$, given the model and the observation sequence, see equation 7.30.

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j | O, \lambda) \tag{7.30}$$

This can be rewritten in a form using the forward and backward variables, see equation 7.31.

$$\begin{aligned} \xi_t(i,j) &= \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)} \\ &= \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)} \end{aligned} \tag{7.31}$$

The γ-variable was earlier defined as the probability of being in state S_i at time t given a observation sequence and a model. This can be related to ξ by summing over j, which gives equation 7.32

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \tag{7.32}$$

The sum of γ can be interpreted as the expected times a state S_i is visited or the number of transitions if $t = T$ is excluded. The sum of ξ can be seen as the expected number of transitions from S_i to S_j. These two summations can be seen in equation 7.33-7.34

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{expected number of transitions from } S_i \tag{7.33}$$

$$\sum_{t=1}^{T-1} \xi_t(i,j) = \text{expected number of transitions from } S_i \text{ to } S_j \tag{7.34}$$

Formulas 7.33-7.34 can be applied to give method for estimating the model parameters, π, A and B.

$$\begin{aligned} \bar{\pi_i} &= \text{expected number of times in } S_i \text{ at time } t = 1 \\ &= \gamma_1(i) \end{aligned} \tag{7.35}$$

$$\begin{aligned} \bar{a_{ij}} &= \frac{\text{expected number of transitions from } S_i \text{ to } S_j}{\text{expected number of transitions from } S_i} \\ &= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \end{aligned} \tag{7.36}$$

$$\begin{aligned} \bar{b_j(k)} &= \frac{\text{expected number of times in state j and observing symbol } v_k}{\text{expected number of times in state } j} \\ &= \frac{\sum_{t=1}^{T} \gamma_t(j), \quad O_t = v_k}{\sum_{t=1}^{T} \gamma_t(j)} \end{aligned} \tag{7.37}$$

Consider a current model $\lambda = (A, B, \pi)$, then equation 7.35-7.37 can be applied to obtain the reestimated model $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$, which is proven to be a better model than λ. This is done iteratively through the observation sequences to improve the model. The last step in procedure is to take the maximum likelihood estimate of the HMM, which is done through equation 7.38.

$$Q(\lambda, \bar{\lambda} = \sum_{Q} P(Q|O,\lambda) \log \left[P(O,Q|\bar{\lambda})\right] \tag{7.38}$$

$$\max_{\bar{\lambda}} \left[Q(\lambda, \bar{\lambda}\right] \Rightarrow P(O|\bar{\lambda}) \geq P(O|\lambda) \tag{7.39}$$

Equation 7.39 states what *Baum* proved, that the maximization of $Q(\lambda, \bar{\lambda}$ increases the likelihood.

7.3 Continuous HMM

The description of the HMM so far, have been based on the assumption, that the observation were discrete values from a finite alphabet with a probability mass function. Usually, when working on most real processes the values will take the form of continuous signals. Therefore it is necessary to modify the HMM to work with probability density functions (PDF) instead, which is done through equation 7.40.

$$b_j(\mathbf{O}) = \sum_{m=1}^{M} c_{jm}\Theta[\mathbf{O}, \boldsymbol{\mu}_{jm}, \mathbf{U}_{jm}], \qquad 1 \leq j \leq N \tag{7.40}$$

The c_{jm} is the mixture coefficient for the mth mixture in state j and $\mathbf{O}$ is the modeled vector. Θ is for instance a Gaussian density, where $\boldsymbol{\mu}_{jm}$ is the mean vector and $\mathbf{U}_{jm}$ is the covariance matrix. The mixture coefficient covers the stochastic constraints, that it sums to one and all entries are larger or equal than zero, which makes the PDF normalized as seen in equation 7.41

$$\int_{-\infty}^{\infty} b_j(x)dx = 1, \qquad 1 \leq j \leq N \tag{7.41}$$

The reestimation formulas for the coefficient for the mixture density are on the form shown in equation 7.42-7.44.

$$\bar{c}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)}{\sum_{t=1}^{T}\sum_{k=1}^{M} \gamma_t(j,k)} \tag{7.42}$$

$$\bar{\boldsymbol{\mu}}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k) \cdot \mathbf{O}_t}{\sum_{t=1}^{T} \gamma_t(j,k)} \tag{7.43}$$

$$\bar{\mathbf{U}}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k) \cdot (\mathbf{O}_t - \boldsymbol{\mu}_{jk})(\mathbf{O}_t - \boldsymbol{\mu}_{jk})'}{\sum_{t=1}^{T} \gamma_t(j,k)} \tag{7.44}$$

The γ function is earlier defined in equation 7.15. In this way it is possible to estimate the parameters of a probability density function and by that, using continuous signals in the HMM.

7.4 Real time

For the offline training part of using HMM it is possible to extract the sequence so they fit perfectly. This can not be done on real-time data from the sensors. Therefore it is necessary to add a window function to *cut* the live data stream into sequences. The length of this window is complicated to compute. One might base it on training data and choose a length, that fits the data. But what would happen if the gesture is done slower, faster or the actual pointing is longer that usual.

These concerns can be handled by reducing the input data to shorter sequences, only including

the important parts. This could be done by looking at the sequence and removing all the redundant frames. For instance the pointing gesture consists of three states, going through one to three, and then back again, in which case the sequence can be reduced to five frames. Another approach could be to view states as events. For instance, stay in state one until the right arm have a high velocity, when the transition happens the window could start and include an amount of previous frames.

7.5 Test design

This section will describe two different test scenarios conducted on the data gathered. The evaluation of a HMM is giving a likelihood value, this value can not be used without having something to compare it to. Usually, a system would look for more than one gesture, and then having multiple models to compare with the observed data. Another way is to have a *garbage* model, which is a model based on data, that is not the pointing gesture. Both of these test approaches will be used to evaluate the pointing gesture recognition.

The test will be based on the K-fold cross-validation method, where all the data is divided into K-parts. Then one of the parts are saved for testing, while the rest is applied to the learning algorithm. This is done for all of the K-parts, so every single sample have been used as training and test, but not at the same time. For this specific test, the data is divided into 5 parts, which gives 80% data for training and 20% for test.

The data gathered is a variety of pointing gestures done by ten different people. The gestures are done as specified in 6.2. In figure 7.6 it can be seen how the data is captured. Each person would stand on a mark on the floor and point at each marker. This add variety to the training set in form of different persons and different pointing directions.

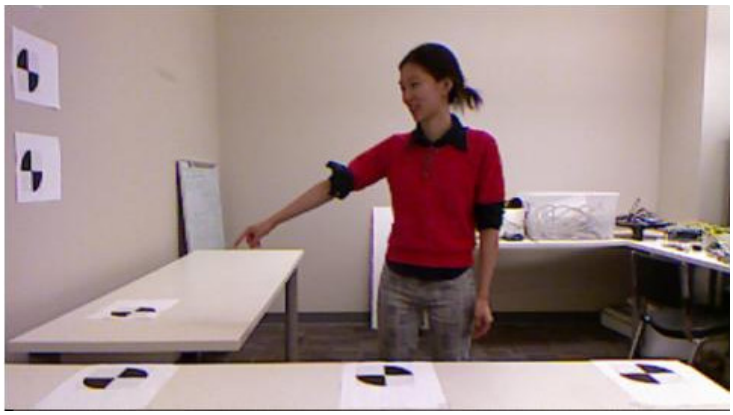


Figure 7.6: The acquisition of the training data.

Further more, there is a pool of data with four different *gestures*: *Stretching arms toward an object*, *Walking*, *Stretching arms over head* and *Wave both hands*. These extra gestures are sup-

pose to imitate gestures that might happen in a production facility. The gesture were recorded for the *garbage* model, but will also be used for a multi-model gesture recognition evaluation. Since the gestures were not a part of the pointing gesture plan, there is only a very limited amount of samples. This decreases the chance of the system working, but also works as an experiment, for how few samples that is actually needed.

8 Implementation

The implementation of the pointing gesture recognition is done in Matlab using the *probabilistic modeling toolkit* [53]. The toolkit allows for the evaluation, decoding and learning, as described in section 7.2. In this chapter the details of the implementations is specified. First a section, that gives an overview of how implementation is done. Last, a section that will go through the implementation and how it works more deeply.

8.1 Overview

The system flow is seen in figure 8.1, where it can be seen that it consists of three parts. The *data* is different gestures in trimmed sequences.

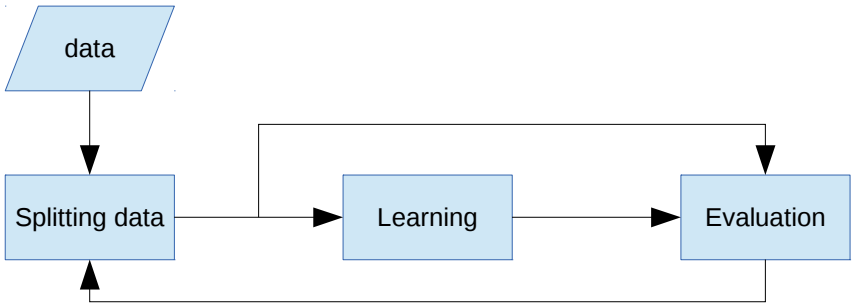


Figure 8.1: The system flow for the gesture recognition.

In the first step, *Splitting data*, all the data is divided into K-parts for each class, as described in section 7.5. Then 20% is send directly to the evaluation, while the remaining 80% is send to the learning algorithm. In the *learning*, all the training data is applied to obtain a HMM for each class. For the *Evaluation* with a *garbage* model the test sequence is compared to two models. For the multi-model gesture recognition evaluation, the test sequence is tested on all six models. The evaluation function from the toolkit returns a likelihood value, that describes how well the data fits the model. This likelihood is compared for all models and the model with the highest value is chosen. Due to time constraint the real time improvements mentioned in section 7.4 are not implemented. The evaluation is instead applied directly to fitted observed sequences.

This page is intentionally left blank.

9 System test

The gesture recognition system is tested as described in section 7.5. First, the training is evaluated for the two test approaches. Then the results are presented and then the results are discussed.

9.1 Training

The training is computed with the Baum-Welch algorithm. One of the interesting things about training HMM is deciding the amount of states. As written in 7.1, one might design the system with a certain number of states. But, this approach does not work for HMM. Instead one should try the trial and error method, and in that way obtain the best amount of states. After testing it was discovered that *7* states was optimal for the pointing gesture. To investigate the model the prior probabilities, the transition probabilities and the mean value for the observations, are listed in equation 9.1-9.3. In equation 9.1 it can be seen that the model is most likely to start in state S_5 and then S_6 and S_1. In equation 9.2 it can be noticed that the diagonal, the probability of staying in the same state, is very high. It can also be seen, that there is some state transitions that are very unlikely.

$$\pi = \begin{pmatrix} 0.195 & 0.012 & 0.098 & 0.012 & 0.445 & 0.223 & 0.015 \end{pmatrix} \tag{9.1}$$

$$A = \begin{pmatrix} 0.911 & 0.014 & 0.007 & 0.029 & 0.006 & 0.011 & 0.022 \\ 0.006 & 0.960 & 0.001 & 0.001 & 0.001 & 0.001 & 0.032 \\ 0.018 & 0.001 & 0.919 & 0.001 & 0.037 & 0.001 & 0.025 \\ 0.013 & 0.000 & 0.000 & 0.959 & 0.000 & 0.000 & 0.026 \\ 0.002 & 0.001 & 0.037 & 0.001 & 0.957 & 0.001 & 0.002 \\ 0.008 & 0.002 & 0.002 & 0.002 & 0.005 & 0.959 & 0.023 \\ 0.023 & 0.023 & 0.026 & 0.025 & 0.001 & 0.003 & 0.899 \end{pmatrix} \tag{9.2}$$

By examining equation 9.3 it is possible to understand what each state represents. For state S_1, which is the first column, the velocity is decent and a high angle. This indicates that this

state is the one where the hand goes from resting position towards the pointing position. State S_3 and S_7 indicates the same thing, but with different angle and velocities, which is properly because the range of that motion is large, i.e. high variance on the velocity, and therefore the state is divided into three. State S_2 and S_4 both have low velocity and high angles, which tells that these states represents when the arm is in the pointing position. Again, this is caused by the high variance in the angle, and therefore the state is divided into two. State S_5 and S_6 is representing the resting position of the arms, where the major difference seems to lie in the *left upper arm angle*. This could be because the two states represents the start of the gesture and the ending, where one of them involves movement in the left shoulder area.

$$\bar{\mu} = \begin{pmatrix} \text{L Upper angle} \\ \text{R Upper angle} \\ \text{L Under angle} \\ \text{R Under angle} \\ \text{L Upper velo} \\ \text{R Upper velo} \\ \text{L Under velo} \\ \text{R Under velo} \end{pmatrix} = \begin{pmatrix} 13.59 & 10.09 & 8.78 & 12.93 & 9.00 & 20.51 & 14.19 \\ 50.16 & 90.79 & 25.38 & 54.23 & 10.82 & 11.11 & 59.88 \\ 2.19 & 2.77 & 3.07 & 1.16 & 3.26 & -1.96 & 0.97 \\ 3.67 & 6.62 & 13.42 & 4.19 & 1.16 & 3.72 & 10.16 \\ 2.55 & 0.26 & 0.75 & 0.39 & 0.58 & 0.82 & 0.54 \\ 10.32 & 0.42 & 26.07 & 0.49 & 1.37 & 1.52 & 19.04 \\ 6.06 & 0.57 & 1.59 & 1.08 & 1.67 & 2.36 & 0.91 \\ 7.15 & 0.82 & 17.36 & 0.88 & 2.73 & 1.87 & 9.92 \end{pmatrix} \tag{9.3}$$

Now that the states are evaluated, it is possible to see if the transition probabilities makes sense. Lets assume the sequence starts in S_5, hence equation 9.1, which is one of the states where arm is in resting position. From S_5 the highest probability is to go to state S_3, $a_{35} = 0,037$, which is a state with high velocity, i.e. the arm is moving toward the pointing position. From state S_3 there are two probable moves either going back to state S_5 or state S_7. State S_7 leads to S_1-S_4, which indicates movement or being in the pointing position. By the same method it is possible to find the state sequence to move back to the resting position.

The analysis of the model above, is a simplification of the workings of a HMM, that excludes the emission probabilities, which indicates what the current state is. The emission probabilities are represented by a 8×8 matrix for each state. In the next section the results will show if it works on real sequences.

9.2 Results

The results for the two-class recognition can be seen in table 9.1. The table is designed as a confusion matrix, where the 91 is the *true positives* and 30 is *true negatives*. There is also 1 *false positive* and 1 *false negative*.

The results shows, that the pointing recognition system works pretty well with the *garbage* model. In table 9.2 the results for the multi-class gesture recognition is shown. It can easily be

Actual class	Predicted class: Pointing	Predicted class: Garbage
Pointing	91	1
Garbage	1	30

Table 9.1: Confusion matrix for two-class gesture recognition.

Actual class	Predicted class: Class 1	Class 2	Class 3	Class 4	Class 5
Class 1	92	0	0	0	0
Class 2	0	4	0	6	1
Class 3	0	0	5	0	0
Class 4	0	1	0	4	0
Class 5	0	1	0	1	3

Table 9.2: Confusion matrix for multi-class gesture recognition. Class 1: *Pointing*, Class 2: *Stretching arms toward an object*, Class 3: *Walking*, Class 4: *Stretching arms over head* and Class 5: *Wave both hands*.

seen by looking in the diagonal, correct predictions, that even though the sample number is low, the recognition system works well. The class with the biggest problems is class 2 with class 4. This is because how related these two gestures are. Class 2 is the *Stretching arms toward an object* and class 4 is *Stretching arms over head*, so these two classes are both using both arms and stretching. Class 1 is the pointing gesture, which passes the test perfectly. Class 3 is *Walking*, which is very different from the other gestures, and that is properly why it works so well. Class 5 *Wave both hands* is in some instances classified as Class 2 and Class 3, this might be caused by the fact that all three classes are having motion with both arms.

From table 9.2 the diagonal can be extracted and summed to get the total number of true positives: 108. The values in the non-diagonal entries are false positives: 10.

9.3 Discussion

The test is divided using two different approaches: The two-class and the multi-class. The *garbage* model approach is easy, but also flawed. A *garbage* model can easily be modeled to be so ambiguous, that every observed sequence is *closer* to the model of the pointing gesture. The other possibility can also be true, the *garbage* model might have so much variety, that each observed sequence easily can be accepted. Regarding the multi-class recognition one problem is clear, the amount of training data is very low. Only five samples for each, which is *one* for testing and *four* for training. The models for these gestures might be too undiscriminating for recognizing anything.

In the two-class there is 2 failed and 121 correct classifications, which is a classification rate of 98.3%. For the multi-class test there were 108 correct and 10 failed classifications, which gives

a classification rate of 90.7%. From the results it is clear that the lack of training data causes problem for the models, especially because all the errors occurs for the worst trained models. This is properly not the only problem all the errors are made by Class 2, 4 and 5, which are classes that are pretty similar. This might indicate that the observations are not good enough, but it is hard to judge, because of the lack of training data.

Another problem with the tests, is the way the likelihood values are compared. When just comparing values the system will classify everything, which means that some sequences might have had a low likelihood and yet still been classified. It would make more sense to make a threshold for the likelihood values and classify those below the threshold as an error.

With these concerns in mind the results are good. The results of 98.3% and 90.7% are good, and the improvements will make them better.

Part III

Conclusion

This page is intentionally left blank.

10 Conclusion

This report represent the work I did during my visit to the RIM lab at Georgia Institute of Technology. Working in an environment, which have a more research orientated approach, have been an interesting experience and definitely different from how I have been working on projects at Aalborg University.

In the first part of the report, an evaluation of the skeleton trackers from Primesense and Microsoft, was done. The purpose of this was to understand, which tracker would work best and how introducing multiple Kinects with a simple fusion algorithm would work, for a Human-robot collaboration in an industrial environment. It was discovered that there is only a little difference between the two trackers, one would be best for one scenario, while the other would be best in another scenario. Small vibration modules were also testes on the Kinects, which should reduce the noise in the depth images. The results of those tests were worse than without the modules, so the vibration might be a bad idea for skeleton tracking, while still being viable for point cloud capturing. While using two Kinects, two simple fusion algorithms were tested. The results from on of them was at times better than a single Kinect, while the other was worse. From the experience obtained during these test, with the BMW-project in mind, I would not recommend using any of these trackers. During the tests it was clear, that the data is only accurate, when a person is facing the Kinect, which is hard to achieve in an industrial environment while moving around.

The next part of the report is the gesture recognition, which was done using Hidden Markov models. Two different evaluation approaches were tested: Two-class and Multi-class. The two-class approach yielded great results, but are based on having a *garbage* model, which might not be the optimal choice. The multi-class test showed promising results for recognizing multiple gestures, even though the amount of training data was low. The report specifies some improvements that might improve the performance.

Bibliography

[1] M. Van den Bergh and L. Van Gool, "Combining rgb and tof cameras for real-time 3d hand gesture interaction," in *Applications of Computer Vision (WACV), 2011 IEEE Workshop on*. IEEE, 2011, pp. 66–72.

[2] Wikipedia. (2013) Kinect. [Online]. Available: http://en.wikipedia.org/wiki/Kinect

[3] Gizmodo. (2010) Microsoft kinect hacked? already?! [Online]. Available: http://gizmodo.com/5683744/was-microsoft-kinect-hacked-already

[4] Microsoft Reseach. (2011) Academics, enthusiasts to get kinect sdk. [Online]. Available: http://research.microsoft.com/en-us/news/features

[5] Microsoft. (2010) Primesense supplies 3-d-sensing technology to "project natal" for xbox 360. [Online]. Available: http://www.microsoft.com/en-us/news/press/2010/mar10/03-31PrimeSensePR.aspx

[6] PrimeSense. (2013) Nite middleware. [Online]. Available: http://www.primesense.com/solutions/nite-middleware/

[7] Optoelectronic notes. (2010) How kinect works with primesense. [Online]. Available: http://ntuzhchen.blogspot.com/2010/12/how-kinect-works-prime-sense.html

[8] K. Khoshelham, "Accuracy analysis of kinect depth data," in *ISPRS workshop laser scanning*, vol. 38, 2011, p. 1.

[9] D. TV. (2012) Measuring objects in 3d using only a camera and projector. [Online]. Available: http://www.diginfo.tv/v/12-0159-r-en.php

[10] F. Faion, S. Friedberger, A. Zea, and U. D. Hanebeck, "Intelligent sensor-scheduling for multi-kinect-tracking," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 3993–3999.

[11] L. Sumar and A. Bainbridge-Smith, "Feasability of fast image processing using multiple kinect cameras on a portable platform," 2011, unpublished.

[12] D. A. Butler, S. Izadi, O. Hilliges, D. Molyneaux, S. Hodges, and D. Kim, "Shake'n'sense: reducing interference for overlapping structured light depth cameras," in *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*. ACM, 2012, pp. 1933–1936.

[13] A. Maimone and H. Fuchs, "Reducing interference between multiple structured light depth sensors using motion," in *Virtual Reality Workshops (VR), 2012 IEEE*. IEEE, 2012, pp. 51–54.

[14] S. Matyunin, D. Vatolin, Y. Berdnikov, and M. Smirnov, "Temporal filtering for depth maps generated by kinect depth camera," in *3DTV Conference: The True Vision-Capture, Transmission and Display of 3D Video (3DTV-CON), 2011*. IEEE, 2011, pp. 1–4.

[15] Y. Schröder, A. Scholz, K. Berger, K. Ruhl, S. Guthe, and M. Magnor, "Multiple kinect studies," *Computer Graphics*, 2011.

[16] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore, "Real-time human pose recognition in parts from single depth images," *Communications of the ACM*, vol. 56, no. 1, pp. 116–124, 2013.

[17] L. Zhang, J. Sturm, D. Cremers, and D. Lee, "Real-time human motion tracking using multiple depth cameras," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 2389–2395.

[18] Xbox. Placering af kinect-sensoren. [Online]. Available: http://support.xbox.com/da-DK/xbox-360/kinect/sensor-placement

[19] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*. IEEE, 2011, pp. 127–136.

[20] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, *et al.*, "Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 2011, pp. 559–568.

[21] M. F. Fallon, H. Johannsson, and J. J. Leonard, "Efficient scene simulation for robust monte carlo localization using an rgb-d camera," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 1663–1670.

[22] S.-l. Sun, "Multi-sensor optimal information fusion kalman filters with applications," *Aerospace Science and Technology*, vol. 8, no. 1, pp. 57–62, 2004.

[23] Q. Gan and C. J. Harris, "Comparison of two measurement fusion methods for kalman-filter-based multisensor data fusion," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 37, no. 1, pp. 273–279, 2001.

[24] ROS. Intrinsic calibration of the kinect. [Online]. Available: http://ros.org/wiki/openni_launch/Tutorials/IntrinsicCalibration

[25] J.-Y. Bouguet. Camera calibration toolbox for matlab. [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib_doc/

[26] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 13, no. 4, pp. 376–380, 1991.

[27] T. S. Washio. (2012) Kinect-mssdk-openni-bridge: Experimental module to connect kinect sdk to openni. [Online]. Available: https://code.google.com/p/kinect-mssdk-openni-bridge/

[28] L. Sigal, S. Bhatia, S. Roth, M. J. Black, and M. Isard, "Tracking loose-limbed people," in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 1. IEEE, 2004, pp. I–421.

[29] Vicon.com. Motion capture systems from vicon. [Online]. Available: http://www.vicon.com

[30] Microsoft. Depth space range. [Online]. Available: http://msdn.microsoft.com/en-us/library/hh973078.aspx#Depth_Ranges

[31] J. M. Iverson and S. Goldin-Meadow, "Gesture paves the way for language development," *Psychological Science*, vol. 16, no. 5, pp. 367–371, 2005.

[32] Y. Wu and T. S. Huang, "Vision-based gesture recognition: A review," *Urbana*, vol. 51, p. 61801, 1999.

[33] K. Cheng and M. Takatsuka, "Hand pointing accuracy for vision-based interactive systems," in *Human-Computer Interaction–INTERACT 2009*. Springer, 2009, pp. 13–16.

[34] P. Turaga, R. Chellappa, V. S. Subrahmanian, and O. Udrea, "Machine recognition of human activities: A survey," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 18, no. 11, pp. 1473–1488, 2008.

[35] T. B. Moeslund, A. Hilton, and V. Krüger, "A survey of advances in vision-based human motion capture and analysis," *Computer vision and image understanding*, vol. 104, no. 2, pp. 90–126, 2006.

[36] S. Mitra and T. Acharya, "Gesture recognition: A survey," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 37, no. 3, pp. 311–324, 2007.

[37] A. D. Wilson and A. F. Bobick, "Parametric hidden markov models for gesture recognition," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 21, no. 9, pp. 884–900, 1999.

[38] H.-K. Lee and J.-H. Kim, "An hmm-based threshold model approach for gesture recognition," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 21, no. 10, pp. 961–973, 1999.

[39] M. Brand, N. Oliver, and A. Pentland, "Coupled hidden markov models for complex action recognition," in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*. IEEE, 1997, pp. 994–999.

[40] T. Starner, J. Weaver, and A. Pentland, "Real-time american sign language recognition using desk and wearable computer based video," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 20, no. 12, pp. 1371–1375, 1998.

[41] J. Yamato, J. Ohya, and K. Ishii, "Recognizing human action in time-sequential images using hidden markov model," in *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on*. IEEE, 1992, pp. 379–385.

[42] A. F. Bobick and Y. A. Ivanov, "Action recognition using probabilistic parsing," in *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on*. IEEE, 1998, pp. 196–202.

[43] P. Hong, M. Turk, and T. S. Huang, "Gesture modeling and recognition using finite state machines," in *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*. IEEE, 2000, pp. 410–415.

[44] A. F. Bobick and A. D. Wilson, "A state-based approach to the representation and recognition of gesture," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 19, no. 12, pp. 1325–1337, 1997.

[45] M. Elmezain, A. Al-Hamadi, S. Sadek, and B. Michaelis, "Robust methods for hand gesture spotting and recognition using hidden markov models and conditional random fields," in *Signal Processing and Information Technology (ISSPIT), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 131–136.

[46] D. Kelly, J. McDonald, and C. Markham, "Evaluation of threshold model hmms and conditional random fields for recognition of spatiotemporal gestures in sign language," in *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 490–497.

[47] S. B. Wang, A. Quattoni, L.-P. Morency, D. Demirdjian, and T. Darrell, "Hidden conditional random fields for gesture recognition," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 2. IEEE, 2006, pp. 1521–1527.

[48] S. Calinon and A. Billard, "Recognition and reproduction of gestures using a probabilistic framework combining pca, ica and hmm," in *Proceedings of the 22nd international conference on Machine learning*. ACM, 2005, pp. 105–112.

[49] C. Keskin, A. Erkan, and L. Akarun, "Real time hand tracking and 3d gesture recognition for interactive interfaces using hmm," *ICANN/ICONIPP*, vol. 2003, pp. 26–29, 2003.

[50] C. Shan, T. Tan, and Y. Wei, "Real-time hand tracking using a mean shift embedded particle filter," *Pattern Recognition*, vol. 40, no. 7, pp. 1958–1970, 2007.

[51] L. W. Campbell, D. A. Becker, A. Azarbayejani, A. F. Bobick, and A. Pentland, "Invariant features for 3-d gesture recognition," in *Automatic Face and Gesture Recognition, 1996., Proceedings of the Second International Conference on*. IEEE, 1996, pp. 157–162.

[52] L. Rabiner and B. Juang, "An introduction to hidden markov models," *ASSP Magazine, IEEE*, vol. 3, no. 1, pp. 4–16, 1986.

[53] K. Murphy. probabilistic modeling toolkit for matlab/octave, version 3. [Online]. Available: https://code.google.com/p/pmtk3/

[54] Microsoft. Kinect for windows sensor components and specifications. [Online]. Available: http://msdn.microsoft.com/en-us/library/jj131033.aspx

[55] ROS. Camera pose calibration. [Online]. Available: http://ros.org/wiki/camera_pose_calibration

Part IV

Appendices

This page is intentionally left blank.

A

Kinect specifications

This appendix is a description of the technical specifications of the Microsoft Kinect.

Overview

Figure A.1 shows the Kinect and where all the sensors are placed.

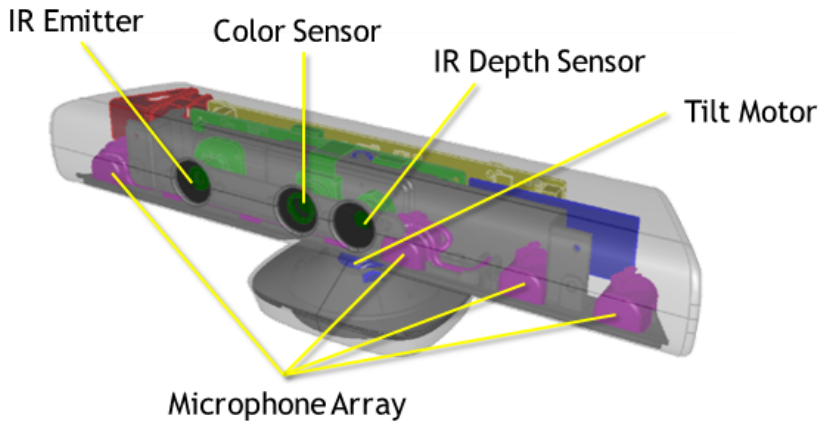


Figure A.1: The Kinect sensors and their position. Picture is taken from the official documentation for the Kinect [54].

Sensors

The list of sensors and other functionalities on the Kinect. The information is taken from the Kinect wikipedia site and the official documentation [54] [2].

- **RGB sensor** The RGB sensor is capable of recording in a resolution up to 1280x1024 at a low frame rate, but is usually running in 640x480 at 30 Hz. The view angle for both sensors are 43° vertically and 57° horizontally field of view, and the practical ranging limit is 1.2–3.5 meters.

- **Infrared emitter and sensor** The depth sensing consist of the the IR emitter and the IR sensor, which combined can generate the depth of the image. The IR sensor can stream the IR video directly in 640x480 with 30 Hz, this is also the resolution and framerate for the depth sensing. The range of the depth sensor is optimal at 0.8-4 meters, with the default range mode. For the near range mode it is 0.4-3 meters [30].
- **Multi-array microphone** The microphone array consists of four microphones, which enables the Kinect to capture sound and detect where the sound is coming from.
- **3-axis accelerometer** The accelerometer works for a 2G range and can be used for measuring the orientation of the Kinect.
- **Tilt motor** The built in motor can tilt the Kinect, which can be necessary when the Kinect is integrated into a system or if a higher or shorter person is using it. The tilt range is $\pm 27^{\circ}$

B

First demonstration

This demonstration was conducted in mid December 2013 for representatives from BMW, who were visiting Georgia Institute of Technology. In this demonstration there was also a setup with the robotic arm moving around, which was made by another student. This chapter will start with an introduction of what is done in this demonstration and the purpose of it. Next up is the description of the implementation of the demonstration. Last, the results and conclusions for the demonstration will be discussed.

Introduction

The purpose for this demonstration is to test a basic way of, intelligently, using multiple Kinects in a scene. This can be done by choosing the Kinect, that produces the best data. The best data must be received, when the person is in optimal position for the Kinect, which is directly in front of it and the person should be facing the Kinect. This must be true, since this is what the Kinect is invented for and therefore trained for.

Implementation

The implementation of this demonstration was done in ROS. The system consists of three nodes, two nodes which acquires data from the two Kinects and one notes that receives data and does the computation and visualization. For this demonstration the visualization tool, Rviz was used. This can be seen on figure B.1. Nodes seen on the figure are communicating through the *Topic* system that ROS uses. It works by enabling nodes to advertise and subscribe to topics. For instance, the two Kinect nodes are advertising the data from the Kinects. The listening node is subscribing to the Kinect nodes and is also advertising the information for the visualization.

To compute which Kinect the person was facing the angle of the torso was computed for both skeletons, if the angle was smaller than a threshold, the Kinect was chosen. This angle was taken directly from the Kinect data stream.

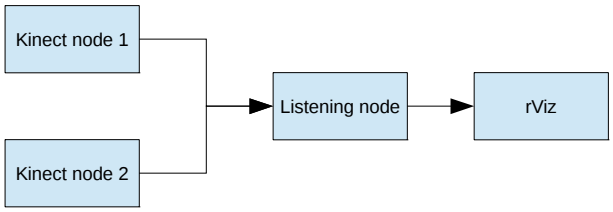


Figure B.1: The system flow for first demonstration.

Discussion

This demonstration confirmed that it is possible to choose Kinect based on the orientation of the person, which might be the optimal solutions for some scenarios. Even though the demonstration worked well, it was decided afterwards, that this approach was not the optimal choice for the system. This is caused by the fact, that it is not always guaranteed, that the person is somehow close to facing a Kinect, so in those cases. This approach would perform bad.

C

Second demonstration

This was an intern demonstration for the BMW project group at the Robotic and Intelligent Machines (RIM) laboratory at Georgia Institute of Technology(GT), supervised by Prof. Aaron Bobick and Dr. Henrik I. Christensen. This chapter will start with an introduction of what is done in this demonstration and the purpose of it. Next up is the description of the implementation of the demonstration. Last, the results and conclusions for the demonstration will be discussed.

Introduction

The purpose for this demonstration was to integrate the sensor setup with the robot. The sensor setup was in this case two Kinects standing with a 90 degrees angle, pointing at the workspace. The viewpoint for both Kinects can be seen on figure C.1. Integration meant, that both system should be aware of each other i.e. the Kinects were calibrated to function in the same coordinate space as the robot. This allowed the robot to know where the person in the workspace was, and therefore could make an action based on the position.

Figure C.1: The view from the to Kinects.

To demonstrate this the robot was programmed to follow the right hand of the person, a video clip of this can be seen on the CD, see appendix D. The data from the Kinects in the system were fused together, to make the system robust against missing data from one of them. This was achieved by using the *confidence value* from each joint from each Kinect. If one of the joints

had a *confidence value* smaller than 1, the data would be omitted. If both of the Kinects gave a *confidence value* of 1 for the same joint, the data was averaged.

Implementation

The robot and the Kinects were communication through the build in *Topic*-system, which is shortly explained in B. For doing the fusing of the data, the data from the Kinects was first streamed to a listening node, which fused the data and transmitted it to the robot. The calibration of the Kinects in this system was done using the *camera pose calibration* package from ROS [55].

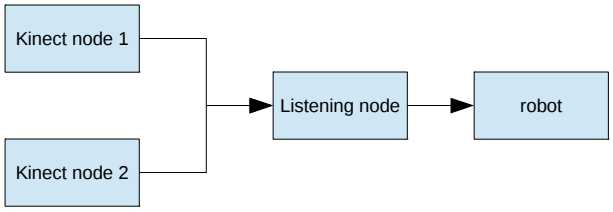


Figure C.2: The system flow for second demonstration.

On figure C.2, the node structure of the system can be seen. The skeleton fusing here is equal to the one as described in section 4.5.

Discussion

This demonstration was a test of integrating the robot and a vision system. The system worked well, and the robot could follow then hand. The fusion algorithm made it possible, that the system would still work, if one of the Kinect was completely covered, which might happen in a manufacturing facility. The success of this fusion algorithm also led to it being used in the article *Evaluation of OpenNI NITE and Kinect SDK Skeleton Trackers for Human-Robot Interaction*

D
CD index

This section describes what can be found on the enclosed CD.

- **code**
  - **Kinect fusion**
    C++ code for data acquisition and Matlab script for data analysis.
  - **Gesture recogntion**
    All the Matlab script written to extract the data, train and evaluate the HMM.
- **data**
  - **Kinect fusion**
    All data acquired during the evaluation. Includes data from OpenNI, Microsoft and the Vicon system. All scenarios are included in differet files.
  - **Gesture recogntion**
    All the data collected for the testing of the gesture recognition system. The data can also be found compressed in a *.mat* file in the *code/Gesture recognition/Matlab*-folder.
- **report**
  - report.pdf
  - evaluation of skeleton trackers.pdf
- **video**
  - BMW-Demo-02-2013-02-04.MOV

This page is intentionally left blank.

E

Article

Evaluation of OpenNI NITE and Kinect SDK Skeleton Trackers for Human-Robot Interaction

Akansel Cosgun[1], Martin Bünger[2] and Henrik I. Christensen[1]

***Abstract*— In this paper we evaluate the two well-known skeleton trackers OpenNI NITE and Kinect SDK. The choice of tracker is usually based on required platform instead of performance, so this paper gives an overview, through different scenarios, of which tracker to use. In our experiments we use a software bridge to run NITE and MS-SDK on the same data stream, and uses a professional motion capture system for ground truth. We also evaluate a simple dual Kinect fusion and mechanical vibration for interference reduction. Our experiments shows, that MS-SDK handles occlusion better, while NITE resumes tracking faster after an occlusion. MS-SDK has in general better accuracy on shoulders, elbows and hands, while NITE is better on head, neck and torso. The simple Kinect fusion improves the tracked range of motion, while no improvement in accuracy was measured.**

I. Introduction

Detection and tracking humans have been a very active research areas, with potential applications in different domains such as human-computer interaction, gaming, surveillance, medical therapy and robotics. Finding only the human position and orientation is enough for these applications. There has been successes in the area of person position tracking using monocular cameras [1], laser scanners and more recently Time-of-Flight cameras [2]. Motion capture systems aim to find the joint positions of the human body by requiring the user to wear markers. Markerless motion capture systems has been an area after the introduction of the RGB-D cameras. There are currently two state-of-the-art skeleton trackers that are used by researchers and practitioners: Microsoft Kinect SDK (MS-SDK) [3] and OpenNI (NITE) [4]. Kinect SDK is only available in Windows whereas NITE is available on both Windows and Linux. We have observed that the choice of the skeleton tracking system is dictated by the operating system and not the performance of the skeleton tracker. We aim to compare the accuracy and detection rates of the joint positions reported by the two trackers. The trackers cannot run on the same data streams due to the limitations at the driver level, therefore the datasets used for the comparison for MS-SDK and NITE has been different from each other in recent studies. In this work, we run both systems on the same data streams, which leads to a fair comparison.

We are specifically interested in scenarios that is likely to occur in human-robot interaction scenarios. We evaluate both systems for basic scenarios like walking, turning around and sitting. We further tested scenarios that included partial occlusion and full body occlusion. One other scenario involves the person manipulation a box-like object in front of the sensor. These scenarios are likely to happen in daily life when the robot is around people. In an evaluation of the skeleton tracking systems it is important to estimate how the reliability and accuracy are for different types of scenarios. By knowing the limitations of its perception system, a robot can then alter its actions accordingly. For example, the robot might hand-off an object more carefully if the person is sitting in a chair rather than standing. The accuracy and detection rates of the joints would also be important with regards to the safety of humans for scenarios where the robot operates in close proximity to humans.

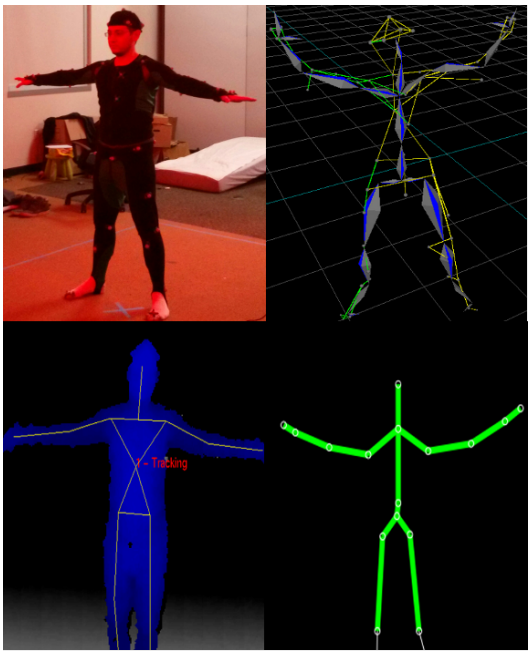


Fig. 1. The different conventions for skeleton models.

First, we examine the relevant literature on person tracking and motion capture in Section II. Section III describes our experimental setup in detail. We report the accuracy errors and detection rates and discuss the results in Section IV, before concluding in Section V.

II. Related Works

Estimation of human poses has been researched for many years and several surveys address the topic [5][6][7]. A variety of different ways of estimating a human model has been suggested for instance tracking the person from the

[1]A. Cosgun and H. Christensen are with Center for Robotics and Intelligent Machines, Georgia Tech, Atlanta, GA, USA.

[2]M. Bünger is with the Department of Computer Engineering at Aalborg University, Denmark.

*A.Cosgun and M. Bünger contributed equally to this work.

bottom and up [8][9], which uses probabilistic methods to fit models on the body, and have good occlusion recovery results. Poselets have also been used for tracking and annotating human body parts [10], where each poselet is trained by a SVM classifier. Another approach with a single Time-of-Flight (ToF) camera [2] using a GPU-accelerated filtering method, has given some good results, .05-.1 meter error rate. The downside of this method is that it only runs around 6 frames per second. Another approach with a ToF sensor and a stereo camera setup [11], which is based on Iterative Closest Point (ICP) and a human body model, and works by fitting the data to the model. Others have tried tracking humans by using multiple gray scale cameras [1]. Work has also been done by using the human silhouette to capture the 3d pose from a human [12]. Their algorithm is based on trained relevance vector machines, which allows it not to use any particular 3d human body model. This enables the algorithm to work on individuals with different body types. The mean angular error for this method is 6-7 degrees. The Microsoft Kinect platform is a common platform for extracting depth maps for skeleton tracking, and have also been used for identifying body parts [13]. In [14] a method uses multiple depth cameras to create a 22-DOF human model, which performs better than NITE and MS-SDK in speed, robustness and accuracy. This method uses a fused depth stream from multiple sensors and tracks the body using a annealed particle filters on the GPU. Others have been using decision trees to compute and track the human pose [15][3], which is the method used on the Microsoft Xbox 360 Kinect.

When multiple Kinects are facing the same scene, overlapping IR patterns cause interference problems for the Kinects. In [16] the RGB and depth stream is combined to filter the depth stream to remove noise and occlusions. Others have been using a Time Divided Multiple Access approach, where they schedule time for each Kinect [17]. Also, in [18] a small mechanical modification is made to the Kinect to make the sensor shake. The vibration adds motion blur to the pattern, which makes the individual sensors able to detect only its own pattern.

III. Data Acquisiton

A. Skeleton Tracking Systems

Our experimental setup is depicted in Figure 2. Skeleton data was acquired from MS-SDK and NITE by running the skeleton trackers on the depth streams obtained by the Kinect for Xbox 360 sensor at 30 Hz. MS-SDK and NITE use different sensor drivers and therefore cannot be run at the same time. Some researchers compared the two systems by acquiring different datasets [14]. We use a software bridge that allows us to run both the NITE and the MS-SDK on the same depth streams [19]. This way, we can do a direct comparison of tracking performances between the two systems. NITE and MS-SDK used in this paper are both versions 1.5.

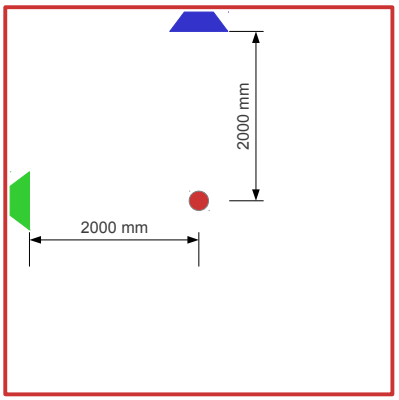


Fig. 2. The position of the Kinects in the experiment.

We used a commercial 3D motion capture system (Vicon [20]) to obtain the ground truth of the joint positions. Vicon finds the 3D positions of passive reflective markers in the scene by triangulation at up to 200Hz. We use the Vicon Blade software to output the skeleton joint poses after a calibration phase on the user who wears a suit with markers.

During data acquisition, Vicon and MS-SDK, which is running in OpenNI framework through bridge, logs the joint poses in real-time at 30Hz. In the meanwhile, the depth stream from the Kinect is saved to an .oni file in OpenNI. This .oni file is later streamed to the NITE skeleton tracker, and the joint positions are logged. This procedure ensures that the exact same motion is captured for the three different systems. Since it takes some time for the NITE tracker to initialize at the beginning of the .oni recording, some frames are lost in the beginning of each recording.

The reported joint positions on the human body model is slightly different in the 3 skeleton tracking systems. See figure 1 for an illustration of the joint position conventions of these systems.

B. Calibration and Synchronization

Before capturing data the sensors have to be calibrated both intrinsically and extrinsically. The intrinsic calibration for the Kinect RGB camera is done using the Camera Calibration Toolbox [21]. Extrinsic calibration is required to find the transformation between the Kinect RGB camera frame and the Vicon global frame, so that the joint positions can be described in the same coordinate frame. A checkerboard with 14 reflective markers on its edges, see figure 3, is used for extrinsic calibration. The markers are placed so that their translation to the interior corners and the markers are exactly known. 3D Corner positions are found in the checkerboard frame by utilizing the intrinsic parameters found. The neighboring corner positions too are used to infer the reflective marker positions in the RGB camera frame. This gives us 14 point correspondences. We then estimate the transformation between the two sets of points by minimizing the least squares error [22].

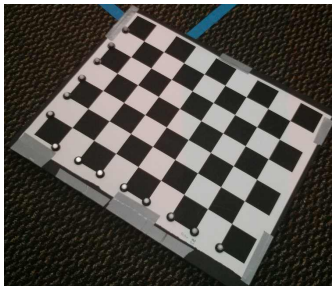


Fig. 3. Checkerboard used for extrinsic calibration.

Caused by software license issues, we used different PC's for the Kinect and Vicon. Although both the Kinect depth stream both run at 30Hz, sometimes the frame rate changes and system clock drifts due to high load CPU. Therefore time synchronization is necessary to associate the skeleton frames. We implemented a time server on the Kinect server where the Vicon PC inquire the current system clock of the Kinect PC after it receives a new data. Time association between NITE and MS-SDK skeletons were not necessary because the number of frames for both trackers were equal after the .oni file processing.

C. Dual Kinects

The experiment is divided into two parts: *single Kinect* and *dual Kinects*. In the dual Kinect experiments, we acquire time synchronized skeleton data from both Kinects. The Kinects are placed so that there is approximately 90° between them.

When multiple Kinects are facing the same area, the patterns from the IR projectors cause interference. We implemented the simple method described in [18], where mechanical vibration was applied by tying a simple DC motor with an off-balance load to the sensor. This practical solution improved the quality of the depth stream.

D. Dataset

We acquired a dataset under different scenarios in order to test the capabilities of Kinect-based skeleton trackers. We focused on practical scenarios for common tasks. The scenarios are as followed:

- **Walking** Walking around in a circle while facing the Kinect for measuring performance in basic movement in the scene.
- **360** Two full rotations around self with arms up and down, to understand occlusion issues when person is seen from different angles.
- **Hide** Hiding one arm at the time behind the back and then hiding both at the same time, to explore how the two systems handles partial occlusions.
- **Box** Picking up a box from the floor and extending it towards the sensor, for understanding how the system reacts when a object is introduced to the scene.
- **Occlusion** Full body occlusion by hiding behind a wall and coming back, to measure how fast each system can detect people.

Joint	System	Walk	360	Hide	Box	Occ	Sit	Arms
Head	MSSDK	100	100	100	100	66	100	100
	NITE	91	99	93	82	83	68	99
Neck	MSSDK	100	100	100	100	67	100	100
	NITE	100	100	100	100	83	99	100
Torso	MSSDK	100	100	100	100	67	100	100
	NITE	100	100	100	100	83	99	100
Shoulders	MSSDK	100	73	100	100	60	99	96
	NITE	100	100	100	100	83	99	100
Elbows	MSSDK	100	75	100	96	57	74	96
	NITE	99	90	100	71	59	95	100
Hands	MSSDK	100	76	58	48	47	86	95
	NITE	99	88	100	70	58	94	100

TABLE I

PERCENTAGE OF TRACKED POSITION

- **Sitting** Sitting down for several seconds, and then standing up 2 times. raise afterwards. For measuring how each system performs, when the subject is sitting.
- **Arms** Rotating in steps of 45 degrees with arms fully stretched forward and to the side, for measuring basic arm movement for different orientations.

For the setup with two Kinect one scenario was captured:

- **360 rotation** A slow 360 degrees rotation in T-pose for exploring occlusions caused by rotation in a dual Kinect setup.

IV. Results and Discussion

We measured the joint positional error and detection rate of the joints in the upper body for all scenarios. Joint positional error is the average distance to the ground truth joint position. Detection rate is the percentage of time the joint is tracked.

NITE reports a confidence value for each and every joint, which describe how reliable the data from the tracker is. Three discrete confidences are reported by NITE: 1 if the joint is tracked, 0.5 if in doubt and 0 if not tracking. MS-SDK has a similar variable for each joint of interest: tracking, inferred and not-tracking, which we interpret as 1, 0.5 and 0. We define the metric detection rate as the percentage of frames that the skeleton tracker returns a confidence value of 1 for the run. Table I and III shows the detection rate of different joints for all scenarios. The values in this table are percentages of time that the joint is tracked. This means that joints with a detection rate of $\%100$ is tracked all the time according to that particular skeleton tracker.

Table II and IV are the accuracy in centimeters of both models, compared to the ground truth. The accuracy is only measured for joints that are being tracked i.e. with a confidence of 1.

A. Single Kinect

For a single Kinect, it can be seen, in Table I that for the arms scenario, MS-SDK is more prone to stop tracking. The same observation can be made for the occlusion and

Joint	System	Walk	360	Hide	Box	Occ	Sit	Arms
Head	MSSDK	15.8	17.2	16.1	13.2	**32.5**	14.8	16.3
	NITE	**10.6**	**11.8**	**13.3**	**12.2**	76.2	**11.0**	**13.5**
Neck	MSSDK	11.2	14.5	10.9	**8.5**	**31.8**	7.7	11.2
	NITE	**4.6**	**4.9**	**3.2**	10.5	76.6	**5.7**	**5.2**
Torso	MSSDK	**4.4**	**5.9**	**3.9**	**10.1**	**30.7**	**7.5**	**4.2**
	NITE	6.7	6.7	6.8	15.5	82.0	12.1	7.5
Shoulders	MSSDK	7.8	**16.8**	**5.8**	9.3	**34.6**	**7.9**	**6.7**
	NITE	**5.6**	18.6	7.1	**8.7**	82.4	9.2	7.2
Elbows	MSSDK	9.6	**28.6**	7.6	**6.4**	**42.5**	**8.7**	**7.4**
	NITE	**9.0**	32.0	**7.4**	9.1	78.5	11.1	8.3
Hands	MSSDK	14.8	**47.3**	15.6	**12.2**	**52.9**	**14.1**	**12.7**
	NITE	14.8	50.2	**11.0**	15.9	84.7	14.2	14.2

TABLE II

ACCURACY OF THE TWO MODELS MS-SDK AND NITE

the 360 scenarios. Accuracy-wise NITE performs better for the head and neck, while MS-SDK is better for the rest of the joints as can be seen in Table II. The two basic scenarios Walk and Arms have acceptable accuracies for both systems. Moreover, NITE has better accuracy for Walk scenario and MS-SDK performs better in Arms scenario. In the 360 scenario the biggest errors are for elbows and hands, this is because these are the joints with most motion. The interesting values for the Box scenario is the torso and hands. The torso gets occluded when the box is picked up, which affects the accuracy. The hands have good values, with MS-SDK being better.

The full body occlusion scenario yields the worst accuracy, which was expected. The interesting result here, is that table I shows that MS-SDK is adjusting the confidence value more than NITE, which results in a better accuracy, shown in table II. The sitting scenario has overall good accuracy, but MS-SDK performs best.

Figure 4 shows the error of the position of the shoulder joint in the walking scenario. The figure includes the error of NITE and MS-SDK with respect to the ground truth. It can be noted that this is one of the joints where NITE is better than MS-SDK. The positional error varies between 0.05m and 0.14m in a run of 20 seconds, which may be considered as a large variation in the positional error.

Figure 5 demonstrates the left hand joint positions in x,y and z coordinates over time in the 360 turn scenario. It is noteworthy that it takes a few seconds for NITE to start tracking and this is because of our data capture procedure. NITE runs on the .oni file offline whereas MS-SDK always runs on the live stream. Therefore it takes some time for the NITE tracker to initiate tracking. First at around t=3s, the person starts to turn around self and this turn ends around t=12s, where another rotation is performed. The second 360 rotation is done with the arms along the side of the body, that is why we can see the Y-position decrease at t=12s. In figure 5 the X-position and the Z-position should be similar since it is a rotation around the Y-axis. For the X-position around 6, it seems like NITE is in doubt and is giving some really big errors, while MS-SDK stays closer to the real value. Whats happening at around 7 on the X-position is that the skeleton flips and both MS-SDK and NITE thinks it sees the front of the person instead of the back, that is why the slope looks like the one at around t=10s, where the person is actually facing the Kinect again. The same phenomenon can also be seen on the second rotation and for the Z-position. For this scenario NITE and MS-SDK seems to be performing equally, although NITE seems to return to the right position faster than MS-SDK. This can be seen on figure 5 around t=9s.

It is noteworthy to report that the inaccuracies are also dependent on the joint coordinate conventions for different skeleton trackers. While the errors might be systematic, we do not compensate for the joint offsets and report the raw errors in position.

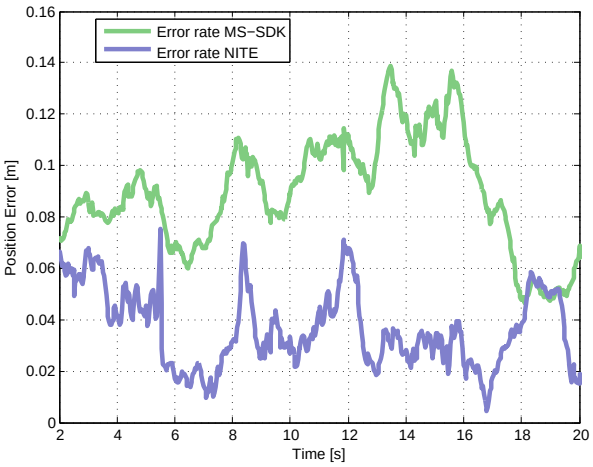


Fig. 4. Average error in shoulder joint positions for the walking scenario.

B. Dual Kinect

For testing the Dual Kinect setup two simple fusion approaches is used. First we look at the confidences, from each Kinect, if both is 1, then we average the result. If the confidence is only 1 for one of the Kinect, we trust the data from that Kinect. This method is called *Avg*. The other method is similar, instead of averaging the result we pick the data from the Kinect, which has the highest *sum* of confidences. The results from each of these fusion methods can be seen in table III and table IV. One of the goals of having multiple Kinects is to increase the range of motion for a person in a scene. It can be seen by comparing table I and III, that the dual Kinect setup has increased the percentage of tracked joints.

In table IV the accuracy for the two fusion approaches and a single Kinect, is shown. The single Kinect performs better on most joint, while the dual average is best on the rest. It can also be noted that the single Kinect and dual avg. is pretty close, while the dual full has a larger error.

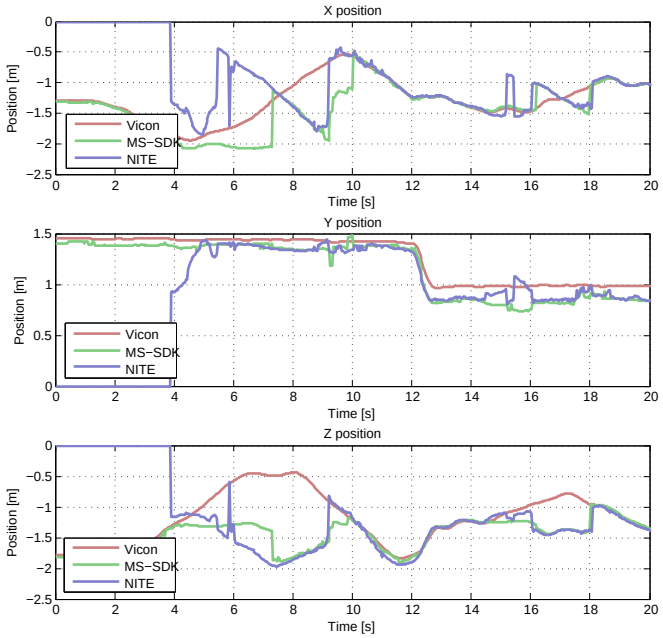


Fig. 5. Absolute position for the left hand joint.

Joint	System	Single Kinect	Dual Avg.	Dual Sum
Head	MSSDK	100	100	100
	NITE	95.5	100	100
Neck	MSSDK	100	100	100
	NITE	100	100	100
Torso	MSSDK	100	100	100
	NITE	100	100	100
Shoulders	MSSDK	76.6	99.4	99.4
	NITE	100	100	100
Elbows	MSSDK	76.4	99.6	99.6
	NITE	92.8	100	100
Hands	MSSDK	83	98.8	98.8
	NITE	91.6	100	100

TABLE III

PERCENTAGE OF TRACKED POSITION

Figure 6 shows the X-position of the left hand joint for the 360 scenario. The top graph is the values from a single Kinect, while the bottom graph is from the dual avg. method. Until around t=7s the data looks noise, this is because the confidence is 0. Next, the same phenomenon as seen in figure 5 can be seen, where the Kinect flips the skeleton. At around t=12s, we can detect the first improvement of the dual Kinect setup, where the MS-SDK finds the skeleton faster than before. This can also slightly be seen at around t=23s. It is worth noting that except for the start, it seems like OpenNI's NITE is doing a better job, but MS-SDK does a better job on the second rotation, where the arms are down.

As mentioned we applied mechanical vibration [18] to each of the Kinect to reduce the interference. In one of the scenarios data was recorded without the vibration. Figure 7 shows the result, the recordings *without* the vibration has better precision, even though the depth stream is improved. This might be because the vibration ruins the calibration.

Joint	System	Single Kinect	Dual Avg.	Dual Sum
Head	MSSDK	**18.7**	19.3	19.5
	NITE	**12.97**	14.77	13.75
Neck	MSSDK	**14.57**	16.06	15.89
	NITE	5.59	**5.11**	6.13
Torso	MSSDK	6.44	**6.09**	6.75
	NITE	6.94	**6.24**	6.91
Shoulders	MSSDK	**16.63**	16.75	18.63
	NITE	**15.01**	16.17	17.03
Elbows	MSSDK	**29.03**	29.96	34.29
	NITE	**29.64**	29.93	31.63
Hands	MSSDK	46.24	**44.06**	52.08
	NITE	48.22	**44.79**	50.09

TABLE IV

ACCURACY OF THE TWO MODELS MS-SDK AND NITE FOR DIFFERENT FUSION METHODS

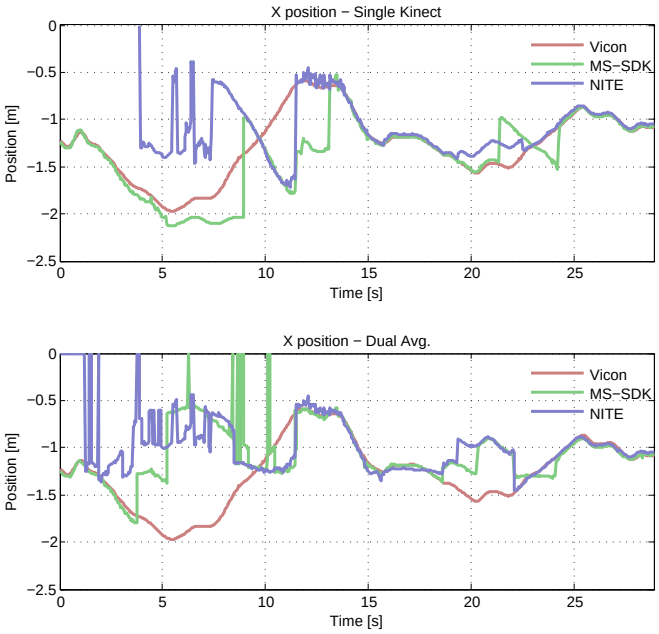


Fig. 6. Absolute position for the left hand joint.

V. CONCLUSION

In this paper, we evaluate the two skeleton trackers MS-SDK and OpenNI NITE. The trackers has been tested in different scenarios, and results concluded that the best tracker should be picked based on the scenario. MS-SDK has better accuracy when occlusion is present and is general better on shoulder, elbows and hands. NITE needs several frames before the tracking starts, but can resume skeleton tracking faster than MS-SDK, after an occlusion. A setup with dual Kinects has also been investigated with two simple fusion methods, results shows an increase in the tracked range of motion, but close to non improvements in accuracy. It has also been tested how mechanical vibration can improve dual Kinect performance, it was discovered that the vibration reduces the accuracy.

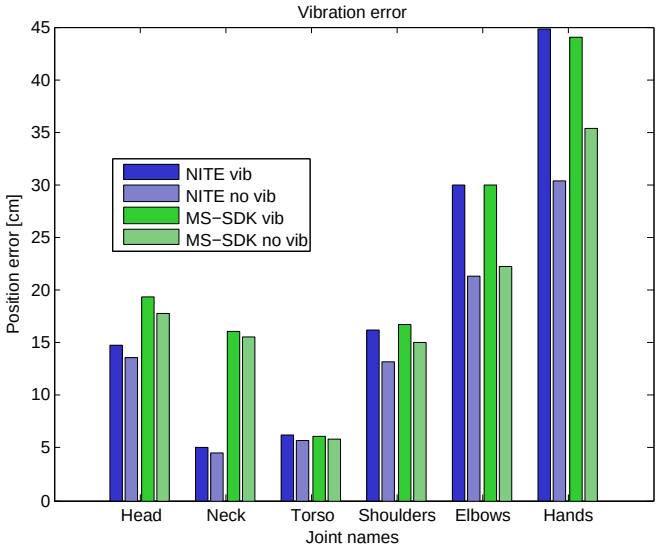


Fig. 7. The error rate comparison with and without vibration.

ACKNOWLEDGMENT

This work was made possible through financial support from the BMW corporation.

REFERENCES

[1] Q. Cai and J. Aggarwal, "Tracking human motion using multiple cameras," in *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*, vol. 3. IEEE, 1996, pp. 68–72.

[2] V. Ganapathi, C. Plagemann, D. Koller, and S. Thrun, "Real time motion capture using a single time-of-flight camera," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 755–762.

[3] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE, 2011, pp. 1297–1304.

[4] [Online]. Available: http://www.openni.org/files/nite

[5] R. Poppe, "Vision-based human motion analysis: An overview," *Computer Vision and Image Understanding*, vol. 108, no. 1, pp. 4–18, 2007.

[6] T. B. Moeslund and E. Granum, "A survey of computer vision-based human motion capture," *Computer Vision and Image Understanding*, vol. 81, no. 3, pp. 231–268, 2001.

[7] T. B. Moeslund, A. Hilton, and V. Krüger, "A survey of advances in vision-based human motion capture and analysis," *Computer vision and image understanding*, vol. 104, no. 2, pp. 90–126, 2006.

[8] L. Sigal, S. Bhatia, S. Roth, M. J. Black, and M. Isard, "Tracking loose-limbed people," in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 1. IEEE, 2004, pp. I–421.

[9] D. Ramanan and D. A. Forsyth, "Finding and tracking people from the bottom up," in *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, vol. 2. IEEE, 2003, pp. II–467.

[10] L. Bourdev and J. Malik, "Poselets: Body part detectors trained using 3d human pose annotations," in *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 1365–1372.

[11] S. Knoop, S. Vacek, and R. Dillmann, "Sensor fusion for 3d human body tracking with an articulated 3d body model," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. IEEE, 2006, pp. 1686–1691.

[12] A. Agarwal and B. Triggs, "3d human pose from silhouettes by relevance vector regression," in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 2. IEEE, 2004, pp. II–882.

[13] C. Plagemann, V. Ganapathi, D. Koller, and S. Thrun, "Real-time identification and localization of body parts from depth images," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 3108–3113.

[14] L. Zhang, J. Sturm, D. Cremers, and D. Lee, "Real-time human motion tracking using multiple depth cameras," *Quadrant*, vol. 1, no. a1, p. a2.

[15] G. Rogez, J. Rihan, S. Ramalingam, C. Orrite, and P. H. Torr, "Randomized trees for human pose detection," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 2008, pp. 1–8.

[16] S. Matyunin, D. Vatolin, Y. Berdnikov, and M. Smirnov, "Temporal filtering for depth maps generated by kinect depth camera," in *3DTV Conference: The True Vision-Capture, Transmission and Display of 3D Video (3DTV-CON), 2011*. IEEE, 2011, pp. 1–4.

[17] F. Faion, S. Friedberger, A. Zea, and U. D. Hanebeck, "Intelligent sensor-scheduling for multi-kinect-tracking," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 3993–3999.

[18] D. A. Butler, S. Izadi, O. Hilliges, D. Molyneaux, S. Hodges, and D. Kim, "Shake'n'sense: Reducing interference for overlapping structured light depth cameras," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '12. New York, NY, USA: ACM, 2012, pp. 1933–1936. [Online]. Available: http://doi.acm.org/10.1145/2207676.2208335

[19] T. S. Washio. (2012) Kinect-mssdk-openni-bridge: Experimental module to connect kinect sdk to openni. [Online]. Available: https://code.google.com/p/kinect-mssdk-openni-bridge/

[20] V. M. Systems and P. P. Inc. (2012) Vicon motion capture systems. [Online]. Available: http://www.vicon.com/

[21] J.-Y. Bouguet. (2010) Camera calibration toolbox for matlab. [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib_doc/

[22] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 13, no. 4, pp. 376–380, 1991.