Tracking People and Arms in Robot Surgery Workcell Using Multiple RGB-D Sensors

Antonio Verdone Sánchez

School of Informatics

University of Edinburgh

This dissertation is submitted for the degree of

Master of Science

Artificial Intelligence

August 2017

Abstract

The tracking of human body parts is an essential requirement for a safe and valuable interaction between autonomous robot assistants and humans. In this project, we present a markerless tracking program capable of detecting a person in a cluttered environment and locating the position of his or her arms over a sequence of frames, using color and 3D information, or only the 3D information, coming from 4 RGB-D sensors (Kinects version 2).

The project is divided into three parts: a pre-processing stage, a detection part and a tracking part. In the pre-processing stage, the algorithm gathers the data recorded with the sensors and corrects the small delays that could occur between the color and depth images. In the detection part, the algorithm segments the person from the background in each frame using a region growing algorithm and the depth information of the scene. In the tracking part, the program fits the 3D data to a human model consisting of a kinematic tree model and a shape model, necessary for the robot assistant to interact efficiently with the human and avoid collisions.

One of the main contributions of this project is a new dataset for human tracking, called Edinburgh 5six (5 people performing 6 actions). The dataset consists of 30 videos of 5 different people recorded from the waist up approximately with 4 RGB-D sensors while performing 6 different actions. Some of the subjects are not wearing special clothes, they are not located in the center of the camera view and are not isolated from other objects in the scene. Our algorithm obtained an accuracy of 91% (localization within 10cm) on our 5six dataset.

Acknowledgements

First of all, I would like to thank my supervisor Bob Fisher for all of his advices and for being so welcoming from the first day of my arrival at the university. His immense knowledge is second only to his kindness and humbleness. Second, I would like to thank my co-supervisors Marcelo Saval and Radim Tylecek for their invaluable guidance throughout my dissertation and the assistance with the data collection. Also, thank you to Alasdair, Diogo, Marcelo and Gonçalo for being part of my experiments. Finally, thanks to Arturo for helping me edit the video showing the results of the project.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Antonio Verdone Sánchez*)

Research your own experience; absorb what is useful, reject what is useless and add what is essentially your own.

Table of Contents

Chapter 1

Introduction

Some of the ideas and examples presented in this introduction were taken from the introduction in the Informatics Research Proposal (IRP) [1].

1.1 Motivation

The topic of tracking human motion is one of the most actively studied areas of computer vision, with annual conferences and specialized journals devoted to the subject [2]. The solution of this problem is a starting point for many real-world applications in a variety of areas. Some of the domains where human motion capture and analysis are relevant are the following ones [3, 4, 5]: computer animation, virtual reality and video games where precise records of tracked human motions are used to render a cartoon or avatar performing actions that could be dangerous or infeasible for a real person; surveillance of highly-trafficked areas such as airports where an alarm could trigger upon detection of strange behavior of people in the tarmac; human computer and human robot (HCI/HRI) interaction; telepresence, for more realistic and vivid distance communications; home assistance of elderly and infants tracking anomalous behavior; targeting, mainly for military usage where radar or infrared sensors are used to decide what to hit.

Another field that has experienced a rapid growth in recent years and in which this project is framed is robotic surgery. This term includes many applications already implemented in surgical operations nowadays. However, most of them refers to procedures where the doctor controls the end-effectors of a robotic arm through a telemanipulator or a computer control as in the Da Vinci Surgical System [6]. The arrival of these new

technologies revolutionized laparoscopic surgery[1] as well as enhanced open surgery, where traditional steel tools are replaced by autonomous instruments performing specific actions in a more precise way than a human hand could.

Nevertheless, this project focuses its attention in a newer type of machine: autonomous surgical robot assistants that can aid the doctor in different manners such as holding surgical tools or handling equipment. These kinds of robots should be able to independently perform a sequence of actions without a human guiding them through every step. The doctors should be able to interact with the robot assistant in an intuitive way using gestures or their own voice as if they were speaking with a human assistant. In order for this new technology to be implemented, it is crucial that the surgical theater remain a safe environment for the doctors and the patient at all times. Hence, it is necessary to use tracking algorithms in order to track the humans, model their shapes and recognize their actions. In this way, the robot assistant will know at every moment the location of all the humans and their limbs allowing it to handle tools in the right place and prevent collisions. Moreover, the robot should also be able to predict the future movements of the doctors in order to preemptively avoid accidents.

Even though the actual project is presented on the frame of robotic surgery, it is not hard to conceive the use of these robot assistants or enhanced versions of them in completely different environments. They could be used as home helpers for elderly or disabled people, or perhaps as educational tools in school, or even as kitchen assistants in restaurants or at home. The applications are really unlimited and the actual utility of these and other robots will depend on the quality and generalization of the algorithms implemented on them.

1.2 Challenges

The problem of tracking the different parts of the human body is not a new one. Efforts on analyzing the human motion have been made since the early 1960s [7]. Thus, many of the issues that arise when tracking a person remain the same. Despite the advent of new and better technologies such as sensors with higher resolution, most of the factors that turn tracking into a difficult task are intrinsic to human nature and they will never disappear. To begin with, the human body can appear in many different shapes and

[1] "A modern surgical technique in which operations are performed far from their location through small incisions elsewhere in the body". Quotation taken verbatim from Wikipedia. Link: `https://en.wikipedia.org/wiki/Laparoscopic_surgery`. Last accessed: 24 of July, 2017.

sizes, and can take numerous poses. Furthermore, different people behave in different ways depending on their culture, personalities and temporal mood. For example, two individuals could use different signs to express the same feeling. Or perhaps, the same person could perform the same action in different ways depending on the context of the situation. It is clear then, that the tracking algorithms have to deal with a huge amount of variability.

Nonetheless, there are also other difficulties of a more technical nature that need to be dealt with. Fast motions and occlusions are two elements that add complexity to the tracking problem [7, 8]. One object, such as a hand, moving faster than the frame rate of the capture device may not be recognized as the same one on a different time step. Identical outcomes could occur for objects disappearing behind an occluder and reappearing frames later in a video sequence. Self-occlusions will be a frequent problem on our project where one arm of the person will occlude the other arm.

Finally, other kinds of problems are inherent to the type of data used in computer vision, that is videos, or sequences of images. Collecting and sharing this data is sometimes troublesome, especially when using 3D sensors, due to the cost of the equipment needed to capture the data and the large amount of information obtained as a result. Moreover, the sequences of images are captured in different environmental settings, with varying lightning conditions and camera settings [7]. In addition, since the existing datasets cover only a set of specific actions, the corresponding algorithms are very restricted and tend to generalize poorly. In fact, most of the commercial programs available target a narrow set of actions in order to achieve high accuracy rates. Also, we should not forget that obtaining ground truth for every set of data is a laborious and time consuming task.

1.3 Problem Definition and Objectives

The goal of this project is to detect, locate and track people and more specifically their upper limbs using simultaneously up to 4 RGB-D sensors, namely Microsoft Kinects 2, located at different viewpoints as can be seen in Figure 1.1. Each Kinect has been allocated a number for an easier referencing: the two closest sensors to the person are number 1 and 2, and the two further ones are number 3 and 4 (Figure 1.1). Chapter 2 will describe in detail this and other types of 3-dimensional (3D) sensors as well as their benefits and drawbacks with respect to the more common 2-dimensional (2D) sensors (i.e., regular cameras).

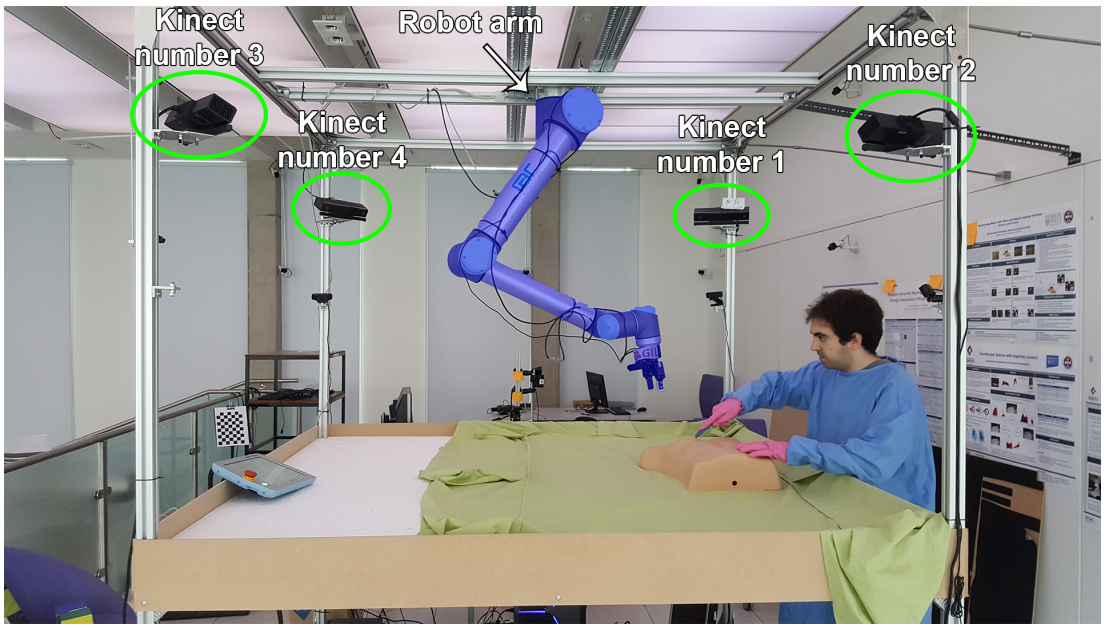


Figure 1.1: Workcell simulating a surgical theater. The Kinect RGB-D sensors are outlined in green. The robot arm is filled in blue.

This thesis is part of a larger plan with the final goal of building an assistant robot arm for a surgical theater. The assistant will be capable of tracking the people present in the simulated operating room (with the help of external sensors), recognizing their hand gestures and distinguishing between different tools. Ultimately, the robot should be able to autonomously understand and follow different instructions such as staying in place, holding a tool or reaching for one and placing it in the hand of the doctor.

Thus, this thesis is the starting point from where the rest of the projects will branch out. Our job will be to identify the actual position of the human and predict his or her future location so as to avoid the robot moving to the same place and colliding with the person. In order to do so, we will have to solve the problem of tracking the different parts of the body across a sequence of frames while dealing with the multiple self-occlusions that could occur. The methodology followed in order to solve this problem will be presented in Chapter 3.

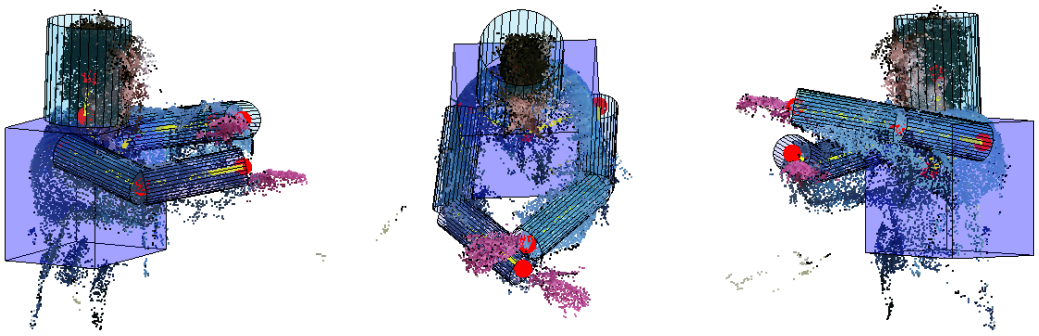
One particular difficulty that we will have to face and that does not seem largely considered in the literature, is the fact of having the subject to be tracked not isolated from the rest of the elements in the environment. In our case, the person will stand at one of the four sides of the simulated operating room as seen in Figure 1.1, in direct contact with the surface simulating the operating table and with the lower part of the body occluded by it at every moment. Normally, the training data used in the literature

has the subject to be tracked far away from any object and in the center of the scene [9, 10], or even in a green screen background [11], facilitating immensely the detection and tracking of the person. Moreover, many of the datasets used in the literature consist of people wearing either special markers or tight clothes designed specifically for the recordings. In our case, the subjects will be wearing either a loose blue gown, simulating a real medical outfit, or their own regular clothes. As we will see, this will lead to recordings with huge amounts of noise but also with a more realistic feeling.

Our solution to the tracking problem is a model-based method, described in Chapter 3, based on some of the approaches presented in the literature, some of which are reviewed in Chapter 2. Nonetheless, our framework exhibits necessary modifications so as to adapt to the peculiarities of the project, such as not having the subject in the center of the scene or having sequences of frames with a lot of noise and not very precise at times.

1.4 Achieved Results

We implemented a program that successfully keeps track of the arms of a person over time for the majority of frames recorded. We conducted a series of experiments and evaluated them quantitatively, as we will see in Chapter 4. After all the tests, we obtained a final accuracy of 91.16% (localization within 10cm) on our dataset. Figure 1.2 shows a final result of the tracking algorithm implemented for this project. Moreover, we found that using four cameras simultaneously achieved from 12% up to 20% more accuracy than using each Kinect separately (Figure 4.23). Final remarks and recommended future steps will be presented in Chapter 5. Some of the final results of this project are available at `https://youtu.be/RZFBXTjE8Ik`.



(a) Left view. (b) Center view. (c) Right view.

Figure 1.2: Three different views of a final tracking result.

1.5 Contributions

The contributions of this project are the following ones:

- The implementation of an end-to-end tracking framework using 4 RGB-D sensors, a configuration not very explored in the literature. In addition, the program created is scalable to any number of 3D sensors. Every step of the process will be explain in detail to assure reproducibility of the results. Furthermore, a fully functional Matlab code will be released along with the thesis.

- The investigation and evaluation of different detection and tracking techniques on markerless realistic noisy data of people wearing loose and regular clothes.

- A new dataset, called Edinburgh 5six, publicly available for the computer vision community, containing 30 videos of 5 different people performing 6 different actions. Every sequence contains 2D and 3D information.

- The application of computer vision algorithms to push the boundaries of the field of robotic surgery. This thesis is part of the first series of projects conducted in the simulated surgical theater built at the University of Edinburgh that make use of an assistant robot arm.

Chapter 2

Background

Various parts of this chapter were taken from the Informatics Research Proposal (IRP), some of them being verbatim or almost verbatim [1].

2.1 A Brief Overview of the Field

The field of computer vision is very broad and relatively in its early stage compared with other more established ones (in the scope of computer vision or outside). As such, it is hard to find a unified taxonomy of all the constituents of the field. Several are the occasions in which, depending on the source of information, the same concept takes on different names or different ideas are called in the same way. The quick appearance of new publications all around the world makes it very difficult to maintain a consolidated nomenclature and the assimilation of new ideas does not come without effort.

This thesis studies one particular area of computer vision: articulated human body tracking. The problem of estimating the configuration of body parts (i.e., their position and orientation relative to some coordinate system) from a sensor input in one time step, that is from a single image frame, is usually called human body pose estimation. When poses are estimated over time, using a sequence of frames (i.e., a video), we call it human motion analysis [4]. Other authors call this last approach tracking [12], while others describe tracking as a segmentation step previous to the pose estimation [3]. This segmentation process is sometimes called detection [10]. In order to not confuse the reader, in the context of this thesis we will consider pose estimation, motion analysis and tracking as synonyms, and the preprocessing stage of locating the person in every frame will be addressed as segmentation or detection part. For a clearer understanding of all these different steps, refer to Figure 3.1. Pose estimation and motion analysis

(or tracking) study only the configuration of body parts, in one time step and over time respectively. If they were also to classify the poses over a set of classes in one frame and interpret the motion over time, they would respectively take the name of pose recognition and gesture recognition [4]. These two approaches will not be discussed in this thesis. However, they are part of parallel projects being conducted as part of the bigger plan of creating a fully functional robot assistant. More precisely, here we will find the position of the hands of a person (among other body parts), while a separate project will be responsible for interpreting their specific movements.

2.2 From 2D to 3D Data

Historically, the task of estimating the articulated motion of the human body has been studied using 2D data [3, 13], more specifically sequences of RGB images (videos). Various were the descriptors used to try to find the configuration of a person, such as silhouettes, edges, motion, color or 3D reconstruction [4]. This last feature was initially extracted using multiple cameras. In recent years, with the arrival of new types of 3D sensors, the depth information of a scene can be extracted directly from the sensor without the need to estimate it.

With the advent of new and accessible technologies such as Microsoft Kinect [14] or Asus Xtion [15], that allow for the direct capture of 3D data, the field of human body tracking has flourished with new and exciting techniques (nonetheless, 3D sensors have also transformed many other areas). The depth imagery coming from 3D sensors is not only a more natural representation of the 3-dimensional universe in which we live, but it also helps to solve real worldly problems such as illumination changes [7], model drifting or occlusion [8]. In fact, 3D data coming from a point cloud is more robust to light changes as it is only composed by the 3D coordinates of every point in the scene. Furthermore, the appearance of objects is more sensitive to rotations and deformations in a 2D environment than in a 3D, where, as a consequence, the issue of the model drifting is mitigated [8]. Moreover, having a complete 3D structure of the scene could help solve the problem of occlusion in certain situations.

2.3 Depth Sensors

In this section, we will briefly describe the 3 main available sensor technologies to capture depth information of a scene. The data obtained from these sensors can be

represented in two different ways: as a 2D range image, where each pixel stores a depth value (instead of a grayscale intensity value or three RGB intensity values), or as a point cloud where each point in the scene is defined by a 3-dimensional vector of X, Y and Z coordinates, where the distance from the sensor to each point (the depth distance) is stored in the Z coordinate [16].

2.3.1 Stereo cameras

This technology imitates the human visual system and uses two regular 2D cameras, usually located vertically or horizontally, to capture intensity images of the desired scene. Subsequently, the depth distance is inferred using geometric properties. A drawback of this method is that stereo images are sensitive to light changes for being captured by regular RGB sensors [7]. They also fail in lack of texture since it is not possible to obtain reliable correspondences between images. Thus, they are not suitable for many kinds of real world situations. Moreover, some parts of the scene will not be seen by both cameras at the same time, preventing a correct 3D reconstruction of those areas. However, these parts will only be located at the edges of the reconstructed surfaces.

2.3.2 Structured light sensors

This second option also uses two devices: one intensity camera and one light source that projects a known pattern that can be recognized by the camera. The depth measurement is based again on triangulation [7]. The first version of the Kinect sensor used this exact technology; in its case, the light source was an infrared sensor emitting a random speckle pattern known by the system. One disadvantage of this method is similar to the one encountered in stereo cameras: some parts of the scene will not be seen by both the camera and the light source. We will lose data at the edges as with stereo cameras and we could get *holes* due to sensing or matching features. This last issue could apply to stereo cameras as well. Another problem of this type of sensors occurs when the pattern light/color is covered by the ambient light or the scene color. In the specific case of Kinect version one, the projected pattern is in the infrared spectrum and it cannot be used outdoors due to the sunlight.

2.3.3 Time of flight

One technology that deals with the problems of light changes in stereo images and holes in structured light images (and also in the edges of stereo images) is the *time of flight* (ToF) method. In this case, we only need one sensor that projects active light pulses towards the objects in the scene. Since the speed of light is known, we only need to measure the time that a pulse takes to reach a particular object (*time of flight*) in order to obtain the distance from the sensor to that object. Alternatively, a sinusoidally modulated infra-red light signal could be projected to the scene and the distance estimated using the phase delay of the reflected signal on the sensor [7].

The second version of the Kinect, that will be used for this project, implements this type of technology (light pulses). With this method, all the pixels in the image will have a depth value (i.e. the depth map will be dense). One drawback of this method, that we will face on our project, is the low resolution of the sensors, especially on an inexpensive commodity hardware such as the Kinect. As a consequence, the person to be tracked will be unable to exit a predefined zone in the scene, otherwise some of the sensors will not be able to capture his or her body, namely the sensors that are closer to the person. Even though the project could be performed using less sensors, we will try to use the four cameras available at all times to achieve higher accuracy.

This kind of sensor normally works in association with a regular RGB camera in order to get color along with the depth information. For this reason, these sensors are usually called RGB-D. This is the case for the Kinect version 2. A caveat should be noted in this regard: the RGB sensor and the depth sensor will not be able to record data at the exact same time. Hence, there will be always a short delay between the color and depth images that, in our case, will be magnified by the fact of having four Kinects recording at the same time from a single computer. This will introduce a high latency caused by having one computer being under the stress of simultaneously recording color and depth images from four different Kinects. We will deal with this issue creating a synchronization algorithm as the first preprocessing step of our data.

2.4 Previous Work for Human Tracking

As already mentioned, it is hard to devise a general pipeline of a computer vision problem. Nevertheless, a generic one could comprise the following three stages:

1. *Pre-processing*: filtering, mesh estimation, registration, segmentation, region of

interest detection, etc.

2. *Processing*: region of interest tracking, occlusion handling, etc.

3. *Post-processing*: action estimation, feedback with recent information, etc.

With regard to the problem of human tracking, the research community has not come up yet with a standard and generally accepted solution, not in 2D and even less in 3D, which is logical due to the higher-dimensionality of the task. One of the approaches that has been usually proposed to simplify the whole tracking problem is the use of fiducial markers on the subjects or the environment of the scene [17]. Other solutions make use of green screens in the background of the scene to facilitate just the preprocessing step of segmenting the person [11]. Nonetheless, these solutions, apart from requiring the addition of obtrusive elements to the scene, lack of generalization power and interfere with the scene itself. That is why markerless approaches are much preferred in most of the situations [9].

Several surveys and taxonomies exist within the domain of human motion analysis [3, 4, 7, 8]. Some of them separate 2D from 3D approaches. Others differentiate methods that use a shape model with those that do not. There are also distinctions between tracking from single and multiple perspectives.

A subdivision that seems to be common to most of the studies is one that separates them in two main categories that can take different names:

- *Bottom-up* designs, also known as model-free, discriminative models or holistic approaches, among other denominations.

- *Top-down* designs, also known as model-based, generative models, analysis-by-synthesis or part-based approaches, among others.

Some authors classify all methodologies into model-based and model-free, and consider both bottom-up and top-down approaches as part of only the model-based methodologies [4]. To keep things simple, we will categorize markerless human motion analysis studies into the two aforementioned classes (from the list).

2.4.1 Bottom-up designs

Bottom-up methods try to detect the parts of a body directly from input sensory data, without making use of a model or using just a very simple one. These methods need a

huge amount of training data that captures many different poses of the body in order to work efficiently. Thus, the computational time to train these models is very high, but the runtime of test examples is very efficient. These methods are very dependent on the input data and on the mapping algorithm that tries to link some created features to the human pose space.

Plagemann, Ganapathi et al. [18] built a model to detect and identify body parts that could be used as a pre-processing stage for a full body tracking algorithm. This model captured depth data from a single ToF camera and created a 3D mesh. From this, the model was able to detect interest points as geodesic extrema in the surface mesh. Subsequently, the points, that corresponded with salient parts of the body, were classified as head, feet or hands, as seen in Figure 2.1, using a boosting algorithm and local shape descriptors. Moreover, the proposed system was able to detect the 3D orientation of the body.

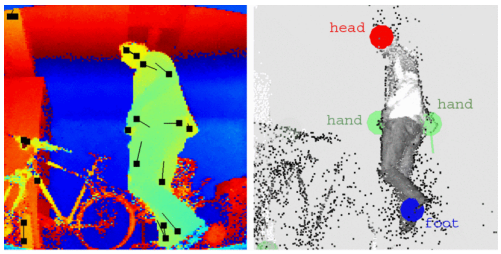


Figure 2.1: The left image shows the depth image with the interest points in black calculated as geodesic extrema. The right image shows the final points classified as head, feet and hands using local shape descriptors. Image taken from [18] showing the approach of the model of Plagemann, Ganapathi et al.

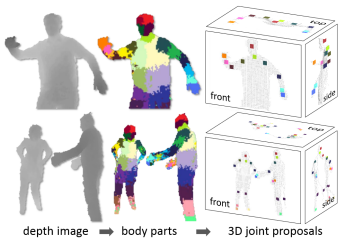


Figure 2.2: From left to right: depth image of the person (left); the result of the random forest on one single frame where every color corresponds to a different body part (center); the final output of the program after calculating the mode of each body part using mean shift (right). Image taken from [19] showing the process followed by the model of Shotton et al.

Shotton et al. [19] proposed a method to predict the positions of body joints using a single depth image. They divided the problem into two parts. First, they segmented an input depth image into 31 possible different body parts assigning a probability to every pixel in the image, as shown in Figure 2.2 (middle). Different colors represent different body parts. This per-pixel classification task is performed by a deep randomized decision forest that avoids overfitting by using hundreds of thousands of training examples, a common trait on bottom-up methods. Once the per-pixel distributions are inferred by

the forest, the mode of every group is calculated using mean shift, obtaining the 3D joints as shown in Figure 2.2 (right). As we can see, this model does not use temporal information from previous depth images. However, the output of the model could be used as a starting or recovery point for any tracking algorithm and temporal information or kinematic contraints to the joints could be added. This algorithm was implemented on the Xbox 360 and obtained robust invariant results to different body poses, shapes and clothing thanks to the huge training set used. The data used for this model had the person (or persons) always unobstructed. Hence, since no external occluders were present, it was enough to recognize the pose of the person on a per-frame manner. Nevertheless, if occlusions were to occur, a more thorough tracking over frames would be needed. Due to time constraints, it is not feasible for us to record the amount of data necessary to follow this approach. Moreover, the person in our project will be in the middle of a much cluttered scene.

The projects reviewed until now are framed within the area of human pose estimation, but not in the specific scope of detecting clinicians in an operating room. Beyl et al. [20] presented an environment very similar to the one we built. They used four Kinect cameras (version 1) in order to detect and track a person in an operating room. Additionally, they installed seven low latency ToF sensors (six PMD[vision] S3 and one PMD[vision] CamCube 2.0) for collision avoidance. The Kinect sensors were unable to conduct scene supervision due to the high latency introduced by them into the whole processing chain. On the other hand, the PMD sensors could not be used for human detection because of their low resolution. All the sensors were ceiling-mounted around the operating table with the layout seen in Figure 2.3. For the human motion analysis performed by the Kinects, the authors used directly the Primesense OpenNI package in conjunction with some cleaning algorithms such as Sobel-based edge detection. The OpenNI package is not specialized in handling occlusions in a robust manner but it is capable to recover the pose of the person after a failure [9]. The occlusion handling problem was mitigated by the setup of four cameras. The paper claims that up to 16 users can be detected per camera, using a correspondence based on the centroid of each subject. This project differs from ours by having a mobile robot arm instead of a fixed one. Thus, the authors came up with two different methods to locate the robot: one based on its relative position with respect to the operating table and the other using the convex hull of the robot in previous time steps. In our case, the location of the robot arm is more simple because the base is fixed and a region growing algorithm could be started from there, in case it would be needed.

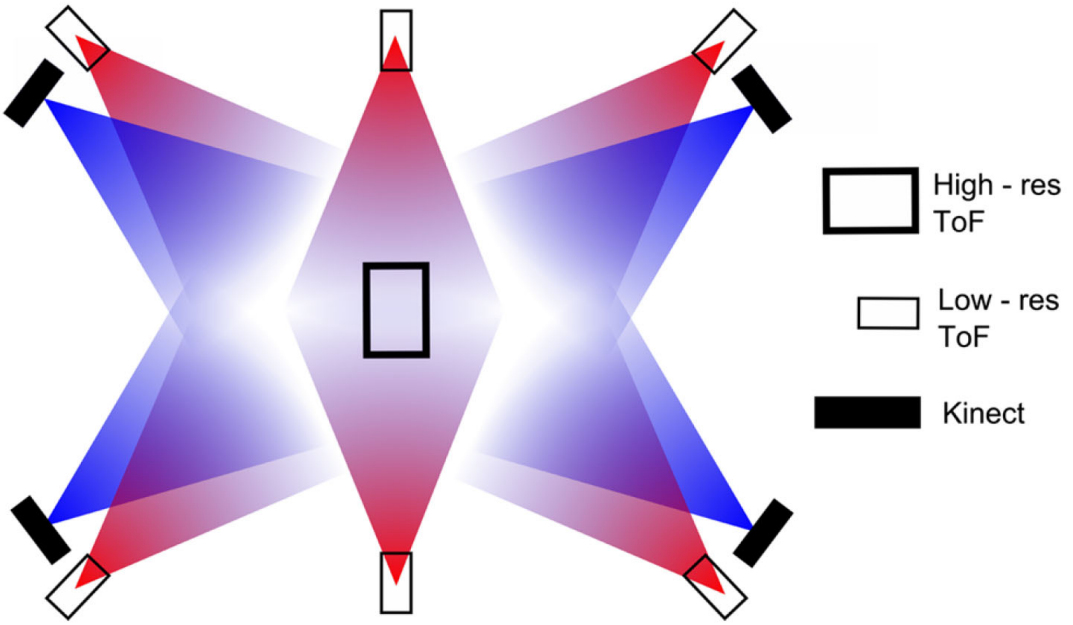


Figure 2.3: Layout of the camera setup from a bird's view from the Beyl et al project. Image taken from [20].

2.4.2 Top-down designs

Top-down methods use an explicit fully articulated model of the human body and try to estimate the angles of each joint that would best fit this model to real data. The model is usually made by a skeleton with dots corresponding to the joints and a surface covering the skeleton. The surface can be constructed with 3-dimensional shapes, such as rectangular cuboids representing the upper body, cylinders representing the arms and forearms, and spheres for the different joints. The model is usually created with kinematic constraints that prevent the skeleton from having unrealistic poses.

The objective of a top-down method is to solve an optimization problem in which the difference between a hypothesized pose of the human body and the 3D reconstruction from some real input data is minimized using a particular objective function. Since the minimization problem can be hard to solve, these kinds of methods are usually computationally expensive, but also quite accurate [9]. Some of the techniques used to track the human pose are the Kalman filter or the particle filter (or Sequential Monte Carlo (SMC)) [10], known in the computer vision community as condensation algorithm [21]. One of the prerequisites of this kind of algorithm is the need for an initial 3D pose in the first frame of the sequence that usually is set manually. One of the advantages of top-down methods is that they do not require a lot of data to work. This was one of the

reasons for which we chose to build a model-based algorithm for our project.

Ganapathi et al. [22] extended their interest point model in [18] by adding a probabilistic filtering algorithm to track the body pose. Hence, they combined a new generative model with their previous discriminative model to detect body parts (described in Section 2.4.1). This model was capable of estimating the joint angles of a human body model of 48 degrees of freedom (DOF) at 4 to 10 frames per second using an efficient GPU implementation.

All the models presented until now used a single depth camera to gather the input data. In contrast, Zhang et al. [10] made use of two Kinect sensors facing each other to track a human model with 22 degrees of freedom. They named their model Multiple Depth Camera Approach (MDCA) and it was claimed that it could be extended to multiple cameras. They fused the data from the two sensors into one point cloud as shown in Figure 2.4. We will follow the same strategy for our 4 point clouds, with the difference that our data will be much noisier. In order to model the human body, they first created a skeleton of 22 DOF with kinematic constraints. Then, they attached to it a surface model using an extended cylinder model in which the circular base was divided into 4 quarter ellipses that could be modified independently to enhance flexibility. The authors decided to exclude regions near the joints for being difficult to model. We will use a similar kinematic model but only for the upper part of the body. In order to estimate the pose of the model based on input sensory data, the authors used an annealed particle filtering algorithm. As in any regular model based tracking algorithm, they defined a motion model to update the state over time and an observation model that relates measured data to the current state. The state in this case is represented by a vector $\mathbf{x} \in \mathbb{R}^{22}$ where every element is a 3-dimensional point in the point cloud. Furthermore, they used a coarse-to-fine search paradigm to solve the optimization problem common on top-down approaches. We will build a similar particle filter but with a different observation model: while the authors of the paper were able to use a signed distance as the likelihood to minimize (the observation model), we will not be able to do the same because our data is too noisy and most of the times the person will be missing the back part of the limbs (since the person is not located at the center of the scene). Instead, we will use the count of the points in the point cloud as our likelihood function (3.3.2.2). The model built in [10] was compared to two state of the art systems: the Microsoft Kinect SDK and the Primesense OpenNI. Four joint points available in every system were selected and were manually tracked every 10 frames in order to evaluate the accuracy of all the models. Similar results were obtained when no

occlusions were present; however, when occlusions arised, the MDCA outperformed the other two models.

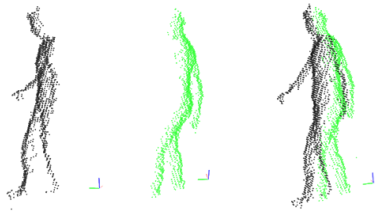
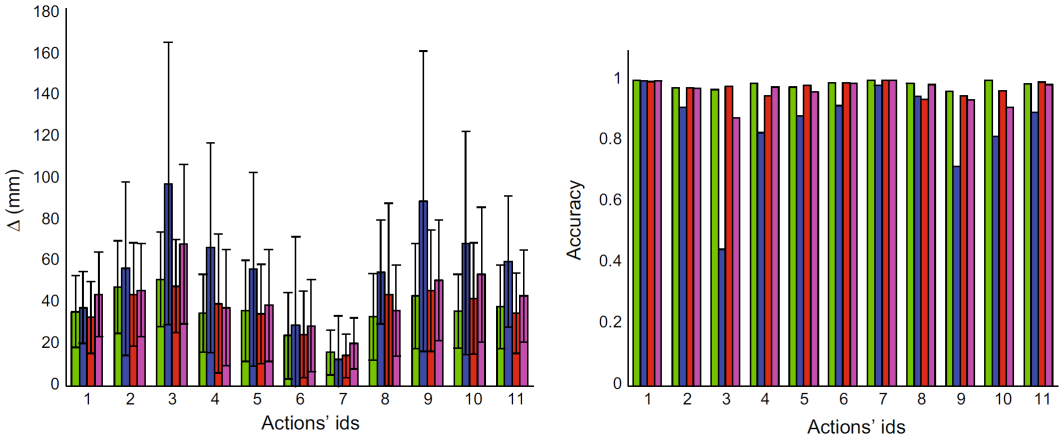


Figure 2.4: The figures on the left and center show the individual point clouds of each Kinect. The figure on the right shows the fused point cloud. Image taken from from [10].

Figure 2.5: From left to right: depth image of the reconstructed person after fusing the data from the two sensors (left); skeleton of 35 DOF and the surface attached to it (center); skeleton fitted to a sensory input example (right). Image taken from [9].

Michel et al. [9] proposed a model similar to the one in [10] but implementing a different observation model and a different optimization algorithm. The set up was also comprised of two Kinect sensors facing each other and positioned in opposite corners of a room. The volumetric representation of the human shown in Figure 2.5 (left) was obtained dividing the space in voxels of $150 \times 150 \times 150$ and assigning the value of 1 to every voxel where the human was present. This process was repeated for both sensors. The body model was created with 35 DOF and used more shapes than in [10] such as elliptic cylinders, ellipsoids, spheres and truncated cones. In this case, the joints were not ignored and were modeled with spheres as shown in Figure 2.5 (center). One of the main contributions of [9] was the optimization algorithm used, namely *perturbed Particle Swarm Optimization* (pPSO) that was a modification of the basic Particle Swarm Optimization (bPSO). The model built was compared with the Primesense OpenNI system that estimates a skeleton without the need of a starting pose. The pPSO model outperformed the OpenNI model in the experiments realized as can be seen in Figure 2.6. The accuracy corresponds to the percentage of distances between the predicted and the real position of the joints within a threshold of 10 centimeters (see Section 4.4). Moreover, the authors created another model called HYBRID mixing their pPSO and the OpenNI in order to combine the merits of both models. The main advantage of HYBRID was that it was completely automatic because it used the OpenNI to estimate the human body model, removing the necessity of having a starting pose. Nonetheless, this same reason was the cause for HYBRID to perform worse than pPSO

that resulted to be the most accurate model. The use of pPSO or HYBRID would then be dependent on the actual application.



(a) Average error.

(b) Accuracy.

Figure 2.6: Quantitative evaluation of 11 different actions of the models from [9]. The pPSO model is in green, the bPSO is in red, the OpenNI in blue and the HYBRID model in purple. Images taken from [9].

Another recent project that studied tracking in the scope of an operating room is the one from Kadkhodamohammadi et al. [23]. The authors of this paper presented a novel approach based on the pictorial structures (PS) models first introduced by Fischler and Elschlager [24] and then expanded with a computationally tractable approach by Felzenszwalb and Huttenlocher [25]. The pictorial structure framework relies on 2D intensity images and uses color-based body part appearance models to segment a person. Kadkhodamohammadi et al. made use of an extended version of the pictorial structure framework, called flexible mixtures of parts (FMP) that employs also depth data. The problem representation is similar to the kinematic models previously seen. Pictorial structures define an energy function over a tree-structured graph where nodes represent body parts and edges represent kinematic constraints. Some final results can be seen in Figure 2.7. As we can see, this approach seems promising to capture different people on a cluttered environment of a real operating room. However, the model does not seem to handle self-occlusions as we can see on the second image from the left in the bottom row where the occluded lower part of the arms of the clinician (at the bottom of the image) are not detected. Furthermore, the human models do not have a surface simulating the skin and flesh, which is necessary to avoid collisions with the robot in our project. It seems logical because the goal of this paper does not seem to have a human robot interaction but just to locate the clinicians in the image. Moreover, they use

only one RGB-D camera. For future work, it appears plausible to mix the 3D pictorial structure approach used by Kadkhodamohammadi et al., that allows for the detection of multiple targets in a cluttered environment, and the approach used in this thesis, inspired by the work of Zhang et al. [10] and Michel et al. [9], where an extended 3D kinematic human model and the use of more than one 3D sensor permits to achieve more precise results.

Figure 2.7: Some examples of pose estimation results from the 3D pictorial structures used by Kadkhodamohammadi et al. Image taken from [23].

As we just saw, bottom-up and top-down are different ways of approaching a problem. Nevertheless, one is not limited to one path or the other. In fact, as we just reported, Michel et al. [9] used a combination of both methods in their HYBRID algorithm. Furthermore, Ganapathi et al. [22] also extended their method that learned from data to use a human body model. Thus, the choice of a solution is dependent on the actual problem that is been treated.

For this thesis, a top-down approach was preferred due to the complexity of the scene designed. Moreover, without a human model to give us prior information, we would have needed a huge database as the ones used in bottom-up strategies with different people in several different positions. Our solution differs from the top-down methods reviewed by the fact of having the human in a cluttered environment where a more thorough segmentation will be needed in order to detect the person in every frame.

Chapter 3

Methodology

In this chapter we will describe the algorithms and methods used to create our tracking project. We designed a plan divided into 3 parts: a *pre-processing* stage, a *detection* part and a *tracking* part. Figure 3.1 shows the workflow of the whole process and the main components of each step. Next, we will explain each stage in detail.

All the programs were implemented in Matlab R2017a as separate blocks increasing the reusability of the code. For example, it is very easy to substitute our tracking part with a completely different one, but still using the point clouds output by our detection part.

3.1 Pre-Processing

The basic thing we need to start the tracking process is data. All the videos used in this thesis were captured using the ROS operating system [26]. The outputs of the recordings are bag files, a file format in ROS for storing ROS message data, containing the color and depth information extracted from the Kinects.

For each frame recorded, we have a color image, a depth image and a point cloud. The Kinect allows for three different resolutions [14]: FullHD resolution (HD - 1920×1080), quarter FullHD resolution (QHD - 960×540) and standard definition (SD - 512×424). The native resolution of the depth sensor is SD and the native resolution of the color camera is HD. Hence, for the correct registration of the depth images to the color images (i.e., to transform them into the same coordinate system) in HD and QHD a calibration is needed. Instead, in SD the registration parameters are provided by the sensor and a calibration is not needed. Therefore, the point clouds in HD and QHD have artificial points generated using upsampling from the original point cloud in SD. For

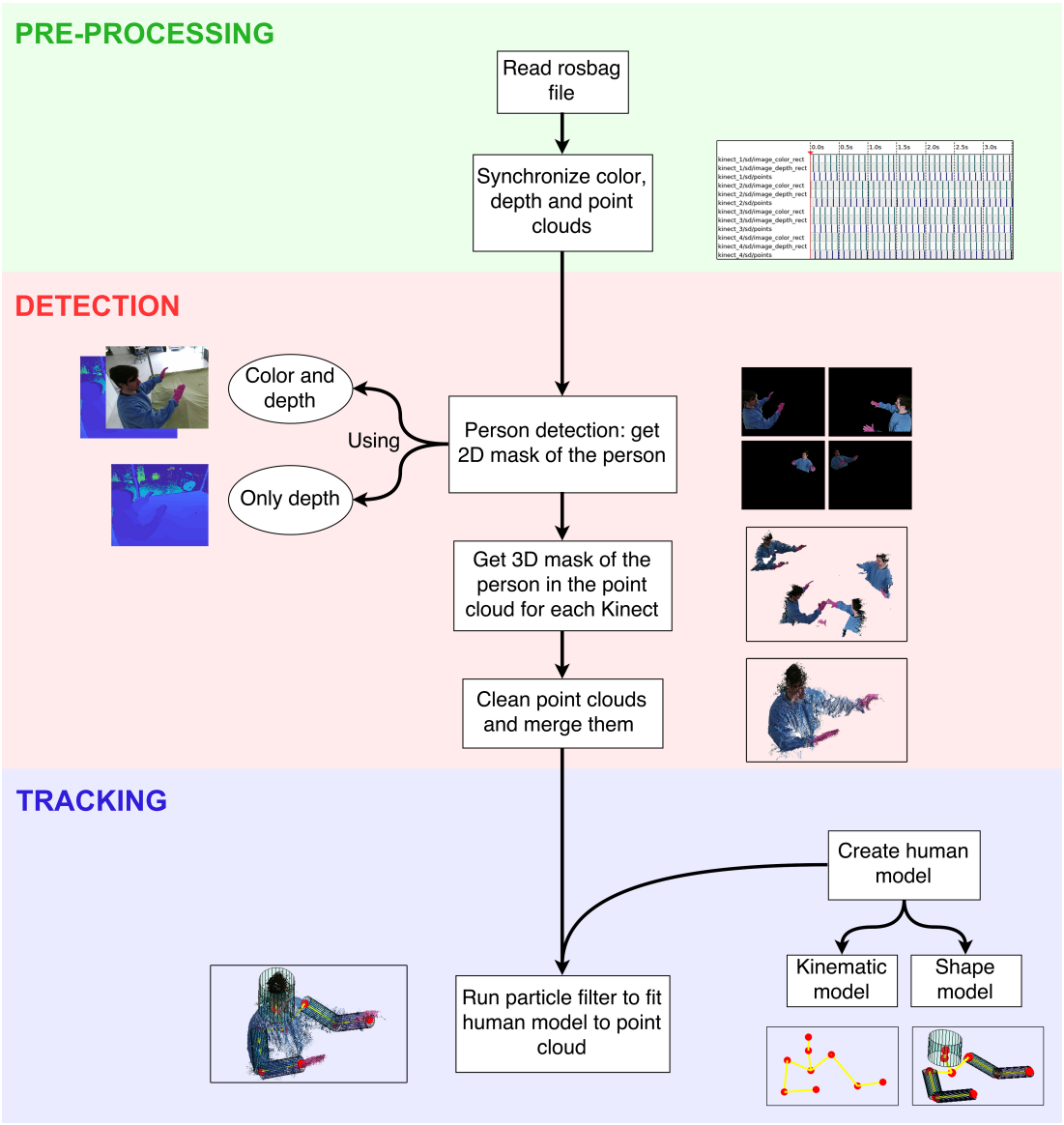


Figure 3.1: Flowchart of the project.

the reasons set forth above and since our algorithms make use mainly of depth data, we chose to use only data recorded in SD. This also allowed for smaller data sets and less computational time of our algorithms due to fewer pixels and 3D points to compute.

As already mentioned when we introduced the time of flight sensors (Section 2.3.3), the color and depth images captured by the Kinect are asynchronous, meaning that they correspond to representations of the scene at slightly different times. Most of the times, when using a single Kinect, this difference is so insignificant that can be ignored. Nonetheless, in our case, since we record data simultaneously from four Kinects, not only the delay between color and depth images within a single Kinect is more noticeable, but also the data between different Kinects is recorded at slightly different moments

in time. Thus, it is necessary to synchronize the data within each Kinect and between them if we want to use both color and depth images.

In order to achieve this synchronization, we programmed a heuristic algorithm that takes two vectors as input and outputs them after dropping the necessary elements in each of them. We call data as synchronized if they are elements with the same index whose absolute time difference is less than a threshold (selected by the user). Figure 3.2 illustrates the functioning of the program with two examples where the threshold has been set to 0.3. As we observe, in the first example 1 and 1.1 are close to each other as well as 3 and 3.2. Thus, the 2 in the first vector ends up without a match and needs to be dropped from the final output (analogously for the 4). On the other hand, in the second example we changed the 4.8 in the second vector for a 4.6. Since $|5-4.6| = 0.4 > 0.3$ (where 0.3 is the maximum time difference allowed), those two elements are not considered close anymore and are dropped. The 4 in the first vector still gets dropped because also $|4-4.6| = 0.6 > 0.3$. As we can see, the algorithm always produces vectors with the same number of elements which is a necessary feature (or at least a very desired one) for the remaining programs to work properly.

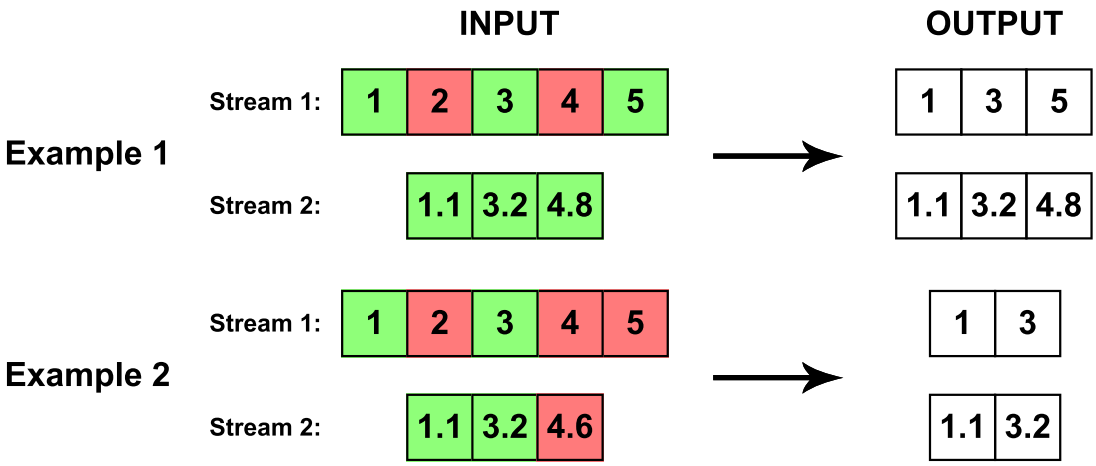


Figure 3.2: Two examples of the functioning of the synchronization algorithm. The numbers in the boxes are the timestamps. The elements in red are dropped for not having a close couterpart in the other vector. The threshold for these examples was set to 0.3.

Now that we have the program, we simply need to use it on the timestamps of the images captured by the Kinects. First, we use it within each Kinect between the point clouds and the depth images. Even though the depth information in the depth images and the point clouds is the same (but in different scales), the Kinect saves them separately

incurring in a time delay. Although the number of depth images and point clouds should always be the same, since they come from the same data, if the computer recording the sequences is under too much stress, then some depth images or point clouds may be skipped and not stored. Thus, we decide to synchronize the depth images and the point clouds first because our project relies more heavily on these than in the color images and hence, we want the best possible synchronization between them. Notice that if we have the same number of depth images and point clouds and they are perfectly paired, our synchronization algorithm will not introduce any change in this step. Subsequently, we calculate the average between elements with the same index of the synchronized vectors, and run the program on this vector and the one with the timestamps of the color images. We compute again the average between the two resulting vectors and repeat the same process but now using the four resulting vectors of each Kinect. For all this process we used a threshold of 0.3 that, in this case, corresponds to seconds. That is, we considered images recorded within a period of 0.3 seconds to be captured at the same time. Figure 3.3 shows a final result of the synchronization step, where vertical lines correspond to color and depth images (in green), and point clouds (in blue).

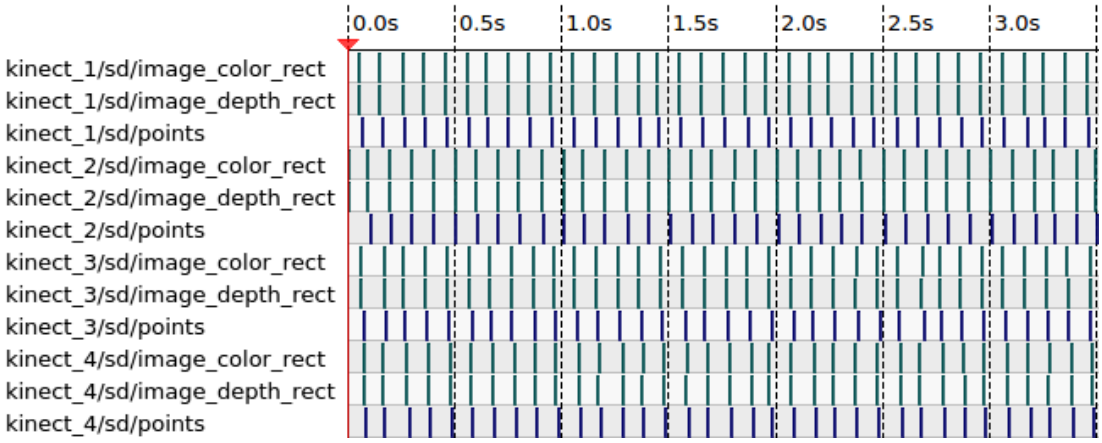


Figure 3.3: Example of captured data after synchronization. The green vertical lines correspond to color or depth frames, while the blue lines are point clouds. The *rect* stands for rectified, which means that color and depth images have already been registered (transformed to the same coordinate system). The space between two vertical dotted lines corresponds to a 0.5 seconds interval.

The latency introduced to the PC by the four Kinects increases as we increase the frame rate of the recordings. The Kinect is capable of capturing video at a maximum frame rate of 30 frames per seconds (fps) [14]. However, the computer used for the project was able to capture data only at a maximum rate of approximately 20 fps. Even

at this rate, the latency was too high and our program had to drop too many frames in order to find an acceptable synchronization. That is why we decided to record all the videos of our final dataset at 5 fps, a frequency that could be handled by the computer in a stable manner. In this way, our heuristic program didn't have to intervene in all the videos, but in case it had, we were confident of having a fairly synchronized data.

3.2 Detection

One of the distinguishable traits of our project is the cluttered scene in which the person to be tracked is located. The human subject is not in the center of the camera views and is not isolated from other objects. For instance, the person is always touching the simulated operating table and his or her lower body part is not seen by the Kinects at any time. Therefore, the foreground segmentation of the person in our project needs careful consideration.

Zhang et al. [10] and Michel et al. [9] used background subtraction on the depth information to attribute the person to the scene foreground. This approach is simple and computationally inexpensive; however, it lacks of generalization power. In fact, every background subtraction algorithm is very sensitive to changes in the environment such as objects appearing and disappearing from the scene. This is the case in an operating room, where surgical tools are constantly moved from one place to another in the table. Moreover, a background subtraction approach needs a view of the environment without the human and this may not always be available.

In order to build a more robust detection algorithm and taking advantage of the fact that the subjects would be wearing a medical outfit, we designed and implemented a region growing algorithm that uses depth information to grow and color data as a starting point.

3.2.1 Person detection: 2D mask

3.2.1.1 Region growing algorithm using color and depth

The medical outfit for this project consists of a blue medical gown and pink gloves. At the same time, the table as well as a mannequin torso simulating the patient are covered with a green cloth. A small opening in the cloth was cut to simulate the access from which the doctor would operate on the patient. The fact of having the mannequin covered by the green cloth ensures a more precise detection of the person.

However, as we will see in Section 3.3, the tracking algorithm is practically unaffected by other objects being part of the foreground and so, the mannequin could have also been uncovered. As long as the person is part of the foreground, the tracking part will most likely succeed. Figure 3.4 shows the outfit of the simulated clinician as well as the environment, from Kinect number 1.

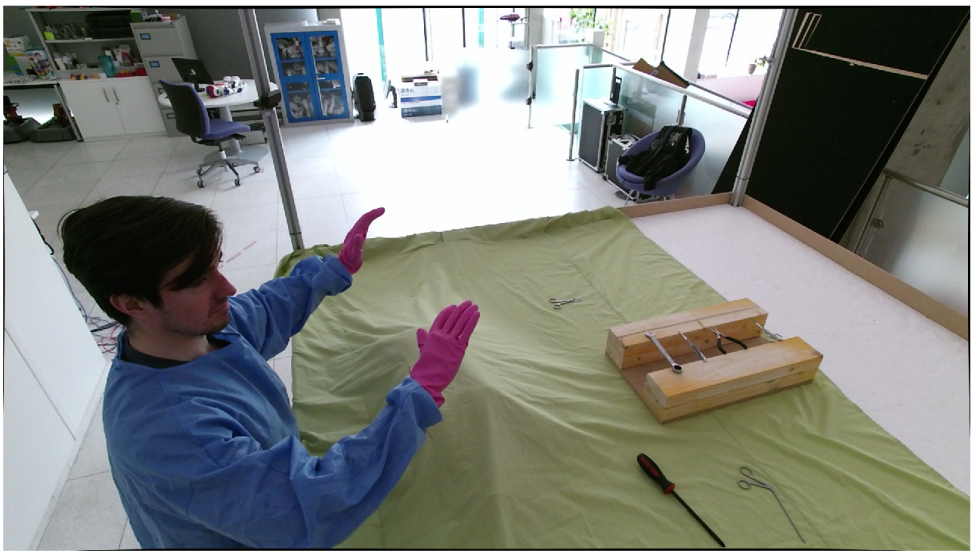


Figure 3.4: View from Kinects number 1 of the simulated surgical environment and the simulated clinician wearing a medical outfit.

Region growing is a segmentation method that combines pixels into larger regions based on their resemblance according to a predefined similarity criteria [27]. This criteria is usually based on color or intensity, but can also be related to depth information. The region growing approach needs a number of pixels, usually called *seeds* from where to start. Then, the algorithm will visit the pixels in the neighborhood of each seed and will add them to the region if they satisfy the similarity criteria with respect to the corresponding seed. In our case, the algorithm will add a pixel if its depth distance is similar to the depth of the seed; that is, if the absolute difference between both depth values does not exceed a certain threshold. For the cases in which the person is wearing the medical outfit, the seed pixels will be the ones corresponding to the hands and the body of the person because they can be easily segmented using color.

3.2.1.1.1 Detecting the gown and the hands We start segmenting all the pink and blue pixels for every frame in every Kinect, using the color images. This process is

accomplished independently for each frame in the dataset; thus, no temporal relationship is used in the detection part. We decided to convert the RGB colormap, output by the Kinect, to the HSV colormap. However, any other color model such as normalized RGB could be used, as long as they discriminate the wanted colors appropriately. Next, we threshold the hue, saturation and value of brightness in HSV with the right threshold values in order to first, select the blue of the gown and second, in a separate process, the pink of the gloves. The threshold values were selected interactively with the *Color Thresholder* app in Matlab. After this segmentation, we had to use morphological operations on the binary masks obtained in order to remove the remaining noise. More precisely, we first removed groups with less than 10 pixels, then used a close operation (dilate and then erode) and finally removed groups with less than 50 pixels. These parameters were chosen experimentally, but any similar values or procedures could work as well. As a final step, for the blue color, we selected the largest blob in the mask that should correspond to the blue gown (if we observe, the blue cupboard in the background of Figure 3.4 should always be smaller than the person); for the pink color, we selected the two largest blobs, representing the hands, if they were available (sometimes only one hand could be visible). Figure 3.5 shows the binary mask before and after morphological operations, and the mask in the color image corresponding to the blue gown, in one frame captured from the Kinect number 1. Analogously, Figure 3.6 shows the whole segmentation process to extract the pink gloves. Finally, Figure 3.7 shows the final color masks corresponding to the blue gown and the pink gloves of the same frame from the four different views of each Kinect.

(a) Binary mask before cleaning the noise.

(b) Binary mask after cleaning the noise and selecting the largest blob.

(c) Color mask corresponding to the pixels of the binary mask.

Figure 3.5: Process of segmenting the medical gown using its blue color in one frame of the Kinect number 1.

(a) Binary mask before cleaning the noise.

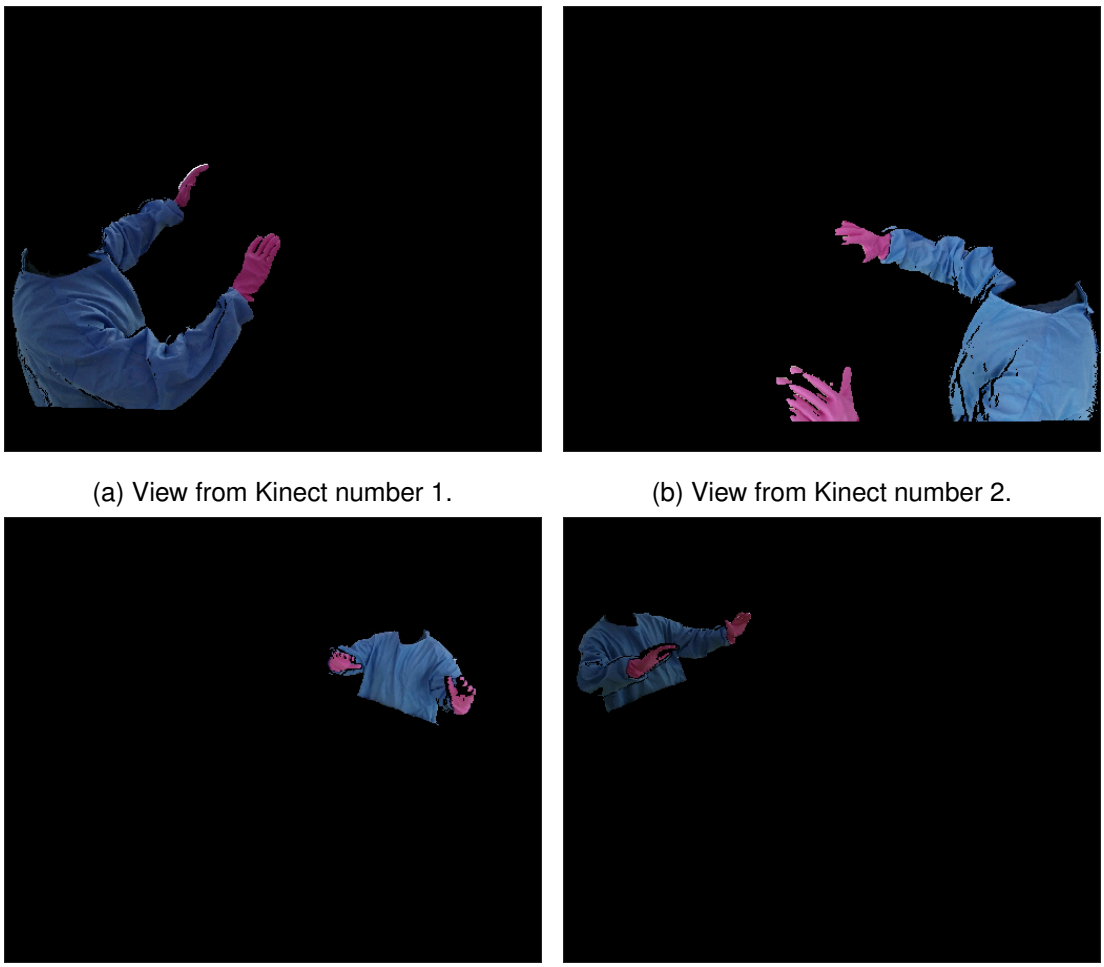
(b) Binary mask after cleaning the noise and selecting the two largest blobs.

(c) Color mask corresponding to the pixels of the binary mask.

Figure 3.6: Process of segmenting the gloves using their pink color in one frame of the Kinect number 1.

Since we are using the same threshold parameters for the blue segmentation and the same for the pink one in order to have a more generalizable algorithm, this process is sensitive to lighting changes in the environment. However, since the masks obtained will be just the starting point for our region growing algorithm, there is no need for these masks to detect exactly the whole gown and the full gloves surface. In any case, if needed, different parameters could be set for each Kinect or even for different lightning conditions (e.g. natural or artificial light). After a visual inspection of all the frames computed, we found that no other object apart from the gown and the gloves were detected by the program. Furthermore, very rarely was the occasion in which parts of the gloves were missing in the view of one Kinect due to the lightness of the scene being slightly different. Since this is kind of a pre-processing step for the tracking stage of our program, we will not present any evaluation metrics for this part. All the metrics will be presented as final scores of the whole program in the experiments chapter (Section 4.5).

3.2.1.1.2 Detecting the table In addition to detecting the gown and the gloves, it is advisable to use the green color to detect the table cloths too. Otherwise, the region growing algorithm will continue growing along the table, since the person is in direct contact with it (the depth values between the table and the person will be very similar). Even though this may not pose a problem for the tracking part in most of the situations, there could be occasions in which it could affect the outcome of the model, especially where the arms or hands of the person are very close to the table, or even in direct contact with it. In this case, the tracking algorithm will receive as input a point cloud where it could be difficult to differentiate between the actual arm and the table. For



(a) View from Kinect number 1.

(b) View from Kinect number 2.

(c) View from Kinect number 3.

(d) View from Kinect number 4.

Figure 3.7: Views from the four Kinects of the final color masks resulting from segmenting the blue gown and the pink gloves.

this reason and since we are already using the color in this part of the program, we will segment the table using the green color in order to know beforehand the pixels to which the region growing algorithm is not allowed to grow.

We follow the same strategy used to segment the blue gown and the pink gloves with the difference that, in this occasion, it is not strictly necessary to perform morphological operations to clean the noise. We decided to only remove small groups of pixels in order to have a clearer illustration for the reader. However, the algorithm would be completely unaffected by the noise because noisy pixels would mainly correspond to points very far from the person where we would not want the region algorithm to grow anyway. Figure 3.8 shows the binary mask containing the table and the corresponding pixels in the color image of one frame captured from Kinect number 1.

(a) Binary mask. (b) Color mask.

Figure 3.8: Results from the process of segmenting the table using its green color in one frame of the Kinect number 1.

3.2.1.1.3 Detecting the whole person The last step of this process is actually running the region growing algorithm in order to detect the remaining parts of the person not segmented by the blue and pink colors. In this case, these parts will be mainly the neck and the head. However, it is not hard to see that the region growing algorithm could also detect other body parts if they were not already part of the mask segmented with color. For example, if the person was not wearing pink gloves, the growing algorithm would naturally grow along the hands; analogously for the case of the person wearing short sleeves. On the other hand, the more we can segment with color, the less time will the region growing algorithm take to detect the whole person.

In order to start the algorithm, we need to identify the seed pixels in the depth image shown in Figure 3.9[1]. For this, we only need to apply the mask in Figure 3.7a (the binary mask rather than the color one shown) to the depth frame corresponding to the same time step and take its perimeter. These will be the seed pixels. In fact, the rest of the points in the mask as well as their adjacent neighbors are already part of the region of interest. All the points in the mask will be added to a list of visited pixels so the algorithm does not waste time going through them. Figure 3.10 shows the depth image with the initial seed pixels outlined in green and the rest of the pixels in the mask in red.

Furthermore, we remove the white pixels in Figure 3.8a, corresponding to the green table, from the depth image in order for the growing algorithm not to grow along them. In addition, we also remove a strip of pixels on the top and bottom part of the image

[1]White pixels correspond to parts in the scene for which the depth sensor has no information for being too far, too shiny or too dark. They take zero value by default.

that were not captured by the color sensor, as seen for example in Figure 3.5. Finally, we had to deal again with the color and depth images not representing exactly the same moment in time. Despite having the best synchronization possible, the mask coming from the color segmentation might not overlap exactly the pixels representing the person in the depth image, especially when the person is moving. Thus, when applying the mask to the depth frame, some seed pixels that should correspond only to the person may actually belong to the background, leading the region growing algorithm to grow throughout the whole image and failing to recognize the person. To avoid this behavior, we shrink the mask in the depth image, using the morphological operation of erode, to ensure that all the pixels truly correspond to the person. In the same way, we enlarge the mask corresponding to the table, using the dilate operation, to remove possible trailing frames of the table caused again by the asynchronous data. The resulting image that will be used as a starting point for the region growing algorithm can be seen in Figure 3.11. As we observe, we may be missing part of the hands and the body initially, but we are making sure that all the seed points are part of the person.

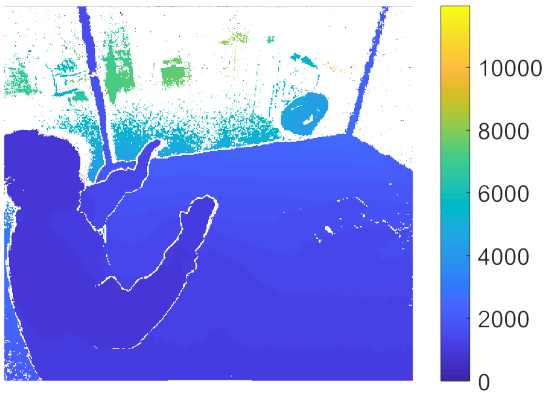


Figure 3.9: Depth image of a frame captured from Kinect number 1. The scale is set in millimeters. Pixels with an exact value of zero correspond to points in the scene with no depth information and are drawn in white.

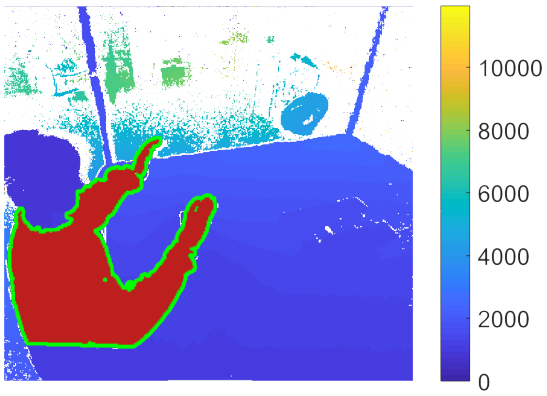


Figure 3.10: Depth image of a frame captured from Kinect number 1 with the seed pixels outlined in green and the rest of the mask coming from the color segmentation in red.

3.2.1.1.4 Region growing Once we have identified the starting points in the depth image, we iterate through all the seed pixels. In every iteration we check if the pixels in the neighborhood of the current seed pixel satisfy the predefined similarity criteria, which in our case is fulfilled if the absolute difference between the depth values of

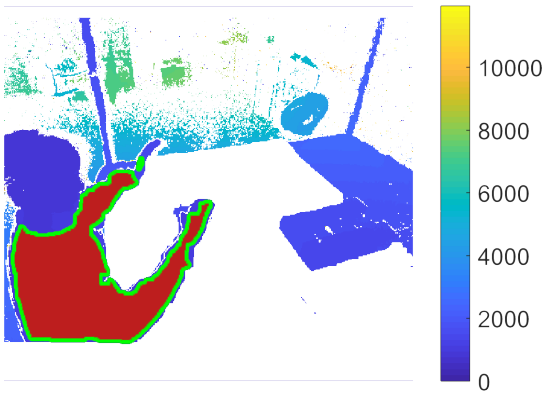


Figure 3.11: Same depth image as in Figure 3.10 after removing the undesired points and correcting the mask.

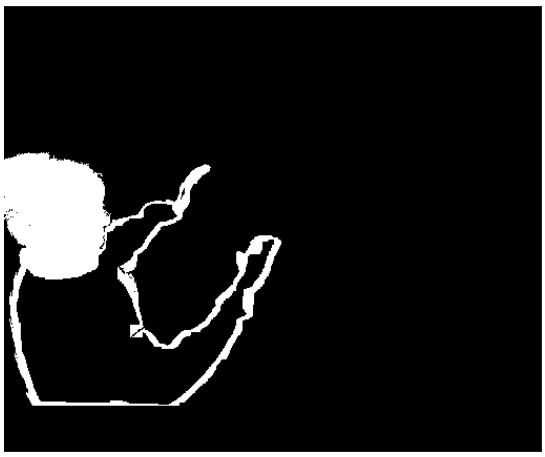


Figure 3.12: Resulting mask of the region growing algorithm.

the seed and neighbor pixels is smaller than a threshold, set experimentally to 100 millimeters. If the condition is satisfied, the neighbor pixel is added to the pool of seed pixels and also to the group of visited ones, meaning that it is part of the person. The neighbors that do not satisfy the criteria are not added to the seed pixels but neither to the visited ones because they could become a member of the region of interest for being neighbors of a different seed pixel with a similar depth value (however this is unlikely and it could happen only when the difference in depth is very close to the threshold). After every iteration, the previous seed pixel is removed from the list of seed pixels. Thus, the algorithm finishes when there are no remaining seed pixels. As in any algorithm of this type, a maximum number of iterations is set, after which the program will stop processing the current frame. This is set in case something went wrong and the algorithm started growing through the background. Figure 3.12 shows the resulting binary mask after running the region growing algorithm on the frame that we have been using as example. If we add this mask to the previously obtained color-based mask, we get the result in Figure 3.13. As a final step, we use the morphological operation of closing (dilate and then erode) in order to fill the small gaps remaining. The final color mask of the detection part for the example frame considered can be seen from the four different views of each Kinect in Figure 3.15. Algorithm 1 shows the pseudocode of the region growing algorithm.

The algorithm implemented is highly customizable. First, we can customize the type of lattice to use for the neighborhood of the seed pixels: we can choose between a 4-connected von Neumann neighborhood or an 8-connected Moore neighborhood. The

Algorithm 1 Region growing algorithm to detect the person in one frame using depth

Precondition: dm is the depth map, as in Figure 3.11, with the pixels corresponding to the table excluded

Precondition: $seeds$ are the pixels located at the perimeter of $mask$, the binary color mask

Precondition: $visited$ are all the pixels in the color $mask$

```
Initialize newMask with same size of mask to 0
iter ← 0
while seeds ≠ ∅ do
    currentSeed ← seeds(1)
    allNeighbors ← findNeighbors(currentSeed)
    validNeighbor ← allNeighbors \ visited
    for every currentNeighbor ∈ validNeighbor do
        if |dm(currentSeed) − dm(currentNeighbor)| < threshold then
            Add currentNeighbor to newMask
            Add currentNeighbor to visited
            Add currentNeighbor to seeds
        end if
    end for
    Remove currentSeed from seeds
    iter ← iter + 1
    if iter > MAX_ITER then
        Exit while loop
    end if
end while
```

difference can be seen in Figure 3.16. The results between the both should not differ a lot in this case, but we implemented it for being a regular option in computer vision algorithms and in order to make a program easily adaptable to different situations. We used an 8-connected neighborhood for all of our data. The second parameter that can be personalized is the level of adjacency of the neighborhood as seen in Figure 3.17. Increasing this value can decrease the running time of the region growing algorithm dramatically. An example of using a level equal to 10 can be seen in Figure 3.14. As we observe, there are gaps in the mask that will be easily filled by the closing operation.

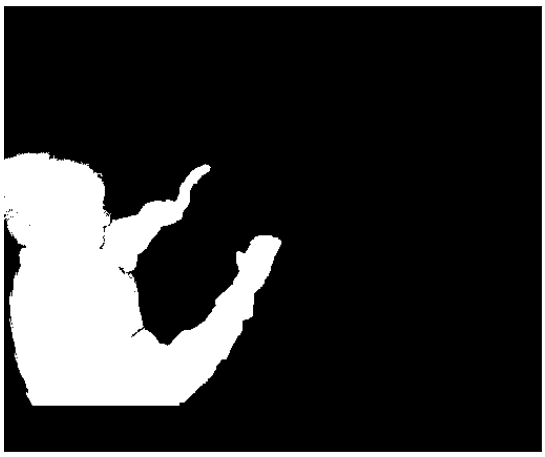


Figure 3.13: Mask resulting from the addition of the color-based mask and the mask obtained with the region growing algorithm.

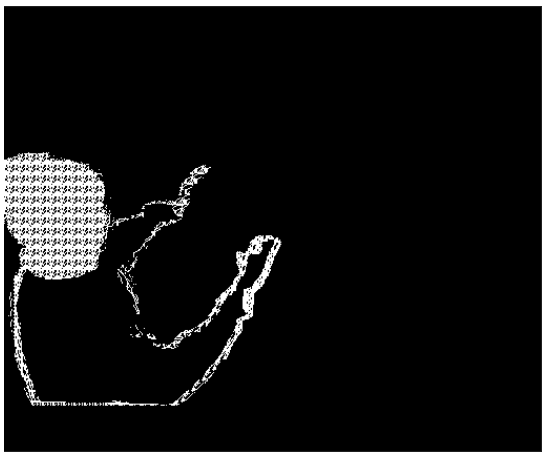


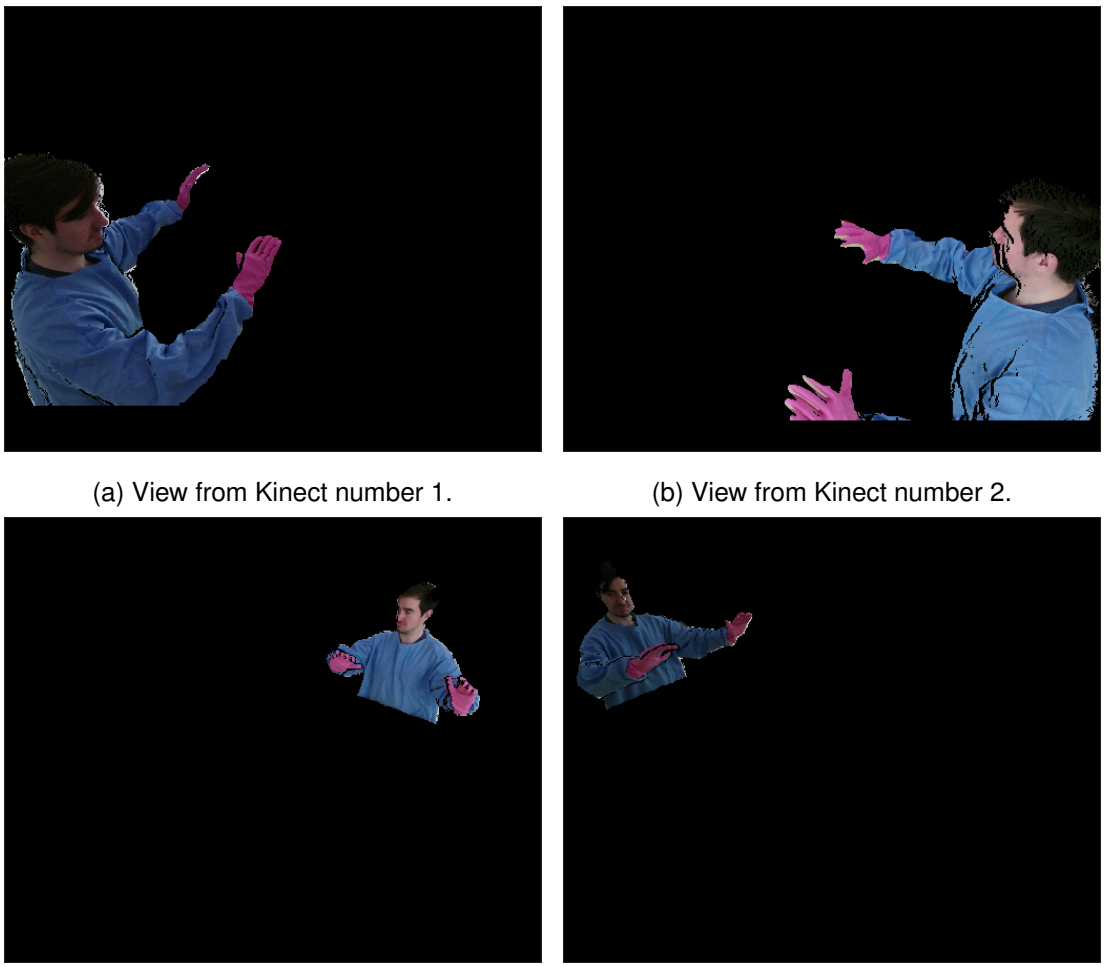
Figure 3.14: Resulting mask of the region growing algorithm using an adjacency level equal to 10.

We don't show the final result of this example because it is virtually the same as the mask in Figure 3.15a. This parameter was initially intended to speed up the process for HD images. Since we are using SD images and because a trade-off between speed and accuracy exists, we found acceptable to leave the value equal to 1. After various tests, we have noticed an approximate increase of 30% in speed for every 10 levels. Nonetheless, there is clearly an upper bound for this level of adjacency; after reaching this limit the algorithm will start comparing pixels too far apart, thus failing to correctly detect the person.

In this project, we will focus our attention on the person in the scene. However, the region growing algorithm implemented can be easily adapted to detect other relevant objects in the scene, such as the robot arm. Appendix A shows how the region growing algorithm can be used to detect the robotic arm.

3.2.1.2 Background subtraction using only depth

Until now, we have only considered the case in which the person to be tracked is wearing a medical outfit. Nevertheless, in order to test the generalization power of our tracking algorithm and also to create a richer dataset for the computer vision community, we decided to also record videos where the subjects are wearing their regular clothes as seen in Figure 3.18. This image contains both color and depth information. More precisely, the black pixels represent parts where the depth sensors could not capture any information. These same parts take the value of zero in the depth map as seen in Figure



(a) View from Kinect number 1.

(b) View from Kinect number 2.

(c) View from Kinect number 3.

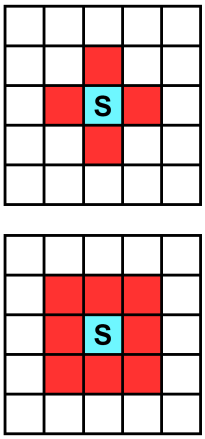
(d) View from Kinect number 4.

Figure 3.15: Views from the four Kinects of the color masks obtained as a result of the detection part of the project.

3.19 and are represented in white.

In this occasion, it will not be possible to start the segmentation process using color because every person will wear different outfits with different colors. And even if we knew the color in advance, it is not optimal to modify the thresholds on the color segmentation programs for every person and every different outfit. Furthermore, people could also wear multicolor garment. Hence, we will need a different strategy that makes use only of depth information to find initial seeds for the region growing algorithm.

A simple approach would be to start from a predefined area where we expect the person to be. If the depth values of the pixels in this area are between some experimentally defined thresholds, it means that a person is present in the scene and thus, the region-based algorithm can start growing from there. However, we quickly



4-connected von Neumann neighborhood

8-connected Moore neighborhood

Figure 3.16: Difference between the 4-connected von Neumann neighborhood (top) and the 8-connected Moore neighborhood (bottom). The cyan cell labeled as S is the seed pixel and the red cells are the neighbors pixels considered.

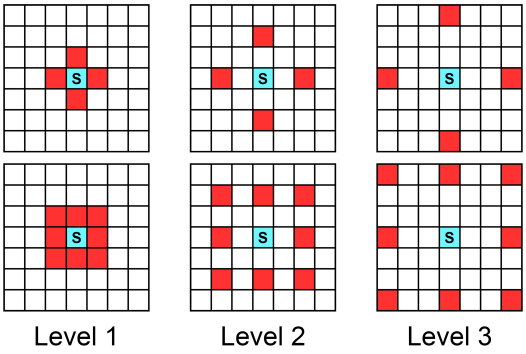


Figure 3.17: Different levels of adjacency. The top row correspond to the von Neumann neighborhood and the bottom row to the Moore neighborhood. The cyan cell labeled as S is the seed pixel and the red cells are the neighbors pixels considered.

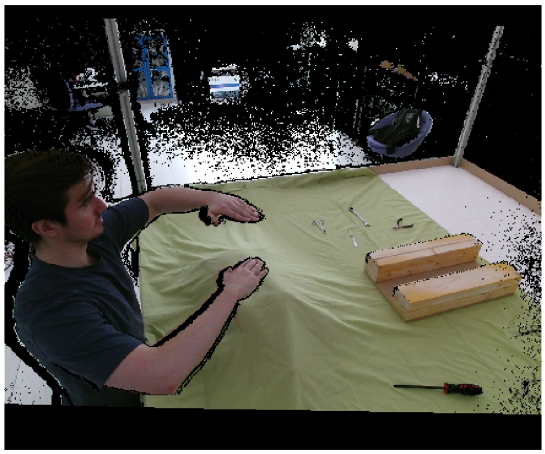


Figure 3.18: View from one of the Kinects of the simulated surgical environment and the person wearing regular clothes. Fusion of color and depth information.

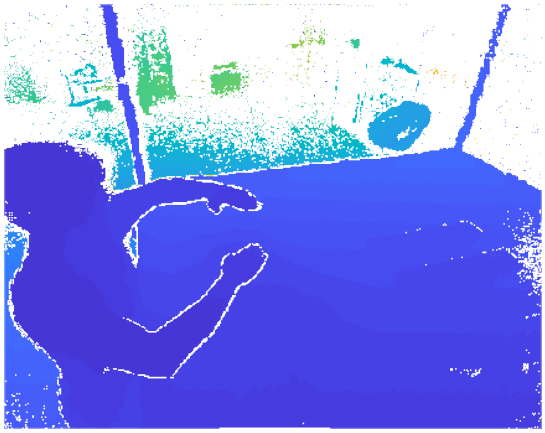


Figure 3.19: View of the depth map from one of the Kinects of the simulated surgical environment and the person wearing regular clothes.

realized that the low resolution of the depth sensors and the fact of having the person located on one side of the scene posed a problem: in some frames the person was not captured in a continuous way, meaning that the elbows, for example, were excluded from the image and the lower arms were not directly connected to the person. This prevented the region growing algorithm to detect the body parts that were isolated from

the main part of the body. This issue may not have caused serious problems to the tracking part because we will eventually merge the point clouds coming from the four Kinects into one and even if one Kinect is missing one arm, some other will have it (especially the two Kinects situated further away). Nonetheless, this strategy would have probably been much less precise than in the previous section due to the poor starting area, and surely would have taken much longer to compute (as the starting area is smaller).

For the above mentioned reasons and in order to have more precise masks to facilitate the work of the tracking part, we decided to use a similar approach to the works of Zhang et al. [10] and Michel et al. [9]. Therefore, the method to segment the body in the videos where the person is wearing regular clothing will be a background subtraction approach that will use only depth information. On the one hand, this method needs only depth data to work. In fact, for these videos we will never use the color images, not here, nor in the tracking part of the program. Thus, for this set of videos, we will not have to deal with the issues caused by the asynchronicity of the color and depth data. Another advantage of this solution is the computational time which is almost instant. On the other hand, for every video, we will need a depth view of the scene without the human. Fortunately, this issue is easily solved because these recordings start with the person out of the vision of the cameras or very far from them. Hence, we can simply use the first frame of each video as representation of the empty background. Then, we only need to subtract every pixel of every frame to the corresponding pixels in the background model and consider as foreground values those above some threshold. That is, for every pixel, if $background - frame > threshold$, then that pixel will be part of the foreground scene. The thresholds are set experimentally after a few tests and are different for each Kinect. For Kinects closer to the person, the number 1 and 2, the threshold is set to 100 millimeters. For Kinects number 3 and 4, the further ones to the human, the thresholds are set to 50 millimeters. After the subtraction, in order to clean the noise, some morphological operations are performed and small groups of pixels removed. Finally, the largest blobs are selected. We do not choose only one because, after all, the whole point of using this strategy is that the person will not appear connected in some frames. Figure 3.20 shows a final binary mask of the background subtraction algorithm, before and after the use of morphological operations.

(a) Mask before cleaning. (b) Mask after cleaning.

Figure 3.20: Binary mask obtained using the background subtraction algorithm.

3.2.2 Person detection: 3D mask

Once we have all the binary masks representing the person for every frame, we just need to transform that information from a 2D image into a 3D point cloud. Remember that a point cloud is nothing but a matrix with 3 columns, corresponding to the X, Y and Z coordinates of the points in the scene, and a number of rows equal to the number of pixels in the depth map, as the one shown in Figure 3.9. Thus, since we are using an SD resolution of 512×424 pixels, the number of rows of a point cloud coming from a single frame is $512 \times 424 = 217088$. However, not every row will hold information. In fact, we already saw that many pixels in the depth images did not have any depth data (and took the default value of 0) and so, those points will correspond to empty rows in the point cloud. Every software handles this situation differently, either assigning to the X, Y and Z coordinates of those rows a numeric data type value representing an undefined or unrepresentable value such as NaN (not a number), or just removing them from the array.

Hence, getting a point cloud from a binary mask consists simply in saving the linear indices corresponding to the white pixels in the mask (i.e., those with a value equal to 1) and selecting the rows in the corresponding point cloud with those same indices. We can think of this process as unraveling the binary mask into a vector and putting it side by side with the point cloud to check what rows have a value of 1 in the mask and thus, are part of the point cloud representing the person. However, not all the points labeled with a 1 will be actually part of the person; some of them will be noise and correspond to points in the scene very far from the person. These points will be removed from the

point clouds. The whole process is performed almost instantly to all the frames in our dataset, since it consists only in indexing arrays and no new data needs to be created. Figure 3.21 shows the four point clouds corresponding to the masks in Figure 3.15. As we can see, the amount of noise is huge and there are plenty of holes (some of them caused by self-occlusions). Sometimes, the same part of the body appears in different positions or it is completely missing from one of the point clouds. This could be caused by several reasons: the person being out of the view of the sensor, natural noise in the image, or again the fact of having asynchronous data. For all these reasons, for every time step, we will merge the four point clouds obtained into one.

Figure 3.21: Point clouds corresponding to each Kinect calculated using binary masks.

3.2.3 Merging the point clouds

One of the most interesting features of this project is the fact of having four Kinects recording data simultaneously. It seems reasonable then to utilize this leverage in our favor. As we saw, it is not uncommon to have missing body parts in certain point clouds, due to occlusions, asynchronicity or low resolution of the sensors. Nevertheless, it is likely that these missing parts of the scene appear in some of the other point clouds

captured at the same moment in time. Therefore, for every time step, we will merge the four point clouds into one.

In order to transform all the point clouds into the same coordinate system, we need to know the corresponding rotation matrices and translation vectors. These values were obtained as a result of a calibration performed after installing the Kinects in the workcell simulating the operating room. More precisely, for the rotation parameters, the system produced a quaternion that we transformed into a rotation matrix. One of the four point clouds, the one coming from Kinect number 1, remained untouched and the other three were rotated and translated in order to overlap the first one. To speed up the process, we combined the rotation and translation values into a single transformation matrix. In order to apply this matrix to the corresponding point cloud, we used homogenous coordinates, obtained by simply adding a column of ones to the point clouds[2]. After merging the point clouds, a standard procedure in Matlab (function *pcdenoise*) to remove outliers was performed. Figure 3.22 shows the resulting merged point cloud. Even though we cannot discern the features of the face of the person, there are no missing body parts and the shape obtained seems reasonably human. The high amount of noise is not only caused by our detection part, but also by the calibration of the four Kinects being off by approximately 5 millimeters in some parts of the scene.

3.3 Tracking

For the detection part (Section 3.2) we used the information coming from the recorded data in a per frame manner, that is, we segmented the region of interest (the person) independently in each frame. In this section we will use the temporal information of the data in a frame to frame fashion, that is, we will estimate the pose of the person over a sequence of frames.

The final goal of the tracking part is to decide what is the most likely position of each body part in a 3-dimensional space. In order to do so, we need a representation of the person and of his or her body parts. We will then try to find the optimal fit of this human model to the observed data using a particle filter method, also known as condensation algorithm in the computer vision community [28].

[2]A point cloud is a matrix A of size $n \times 3$, where n is the number of points in the point cloud. If P is the 4×4 transformation matrix and $B = [A\ \mathbf{1}]$ is the $n \times 4$ augmented point cloud, with a vector of ones as the last column, then the transformation will be $C^T = PB^T$. The final transformed point cloud is C, after removing the last column of ones.

Figure 3.22: Point cloud obtained by merging the four point clouds in a single frame.

3.3.1 Human model

A common 3D representation of the human consists of a kinematic model representing the bone structure of the person, and of a shape model depicting the flesh and skin.

3.3.1.1 Kinematic model

The kinematic model not only portrays a simplified version of the skeleton of a human body, but also describes the kinematic restrictions that exclude non-realistic poses [9]. For this project, we chose to use a kinematic tree model consisting of 9 joints or body parts of the upper body, as seen in Figure 3.23. The root of the tree corresponds to the centroid of the torso and we decided to call it *chest*. An alternative and equivalent nomenclature for the tree would be to report the different arm parts (lower and upper arm) instead of the joints [29]. Figure 3.24 shows a simplified skeleton model corresponding to the tree in Figure 3.23.

For the set up designed in this project, due to the low resolution of the depth sensors and since the subject will not be standing in the center of the camera views, the person

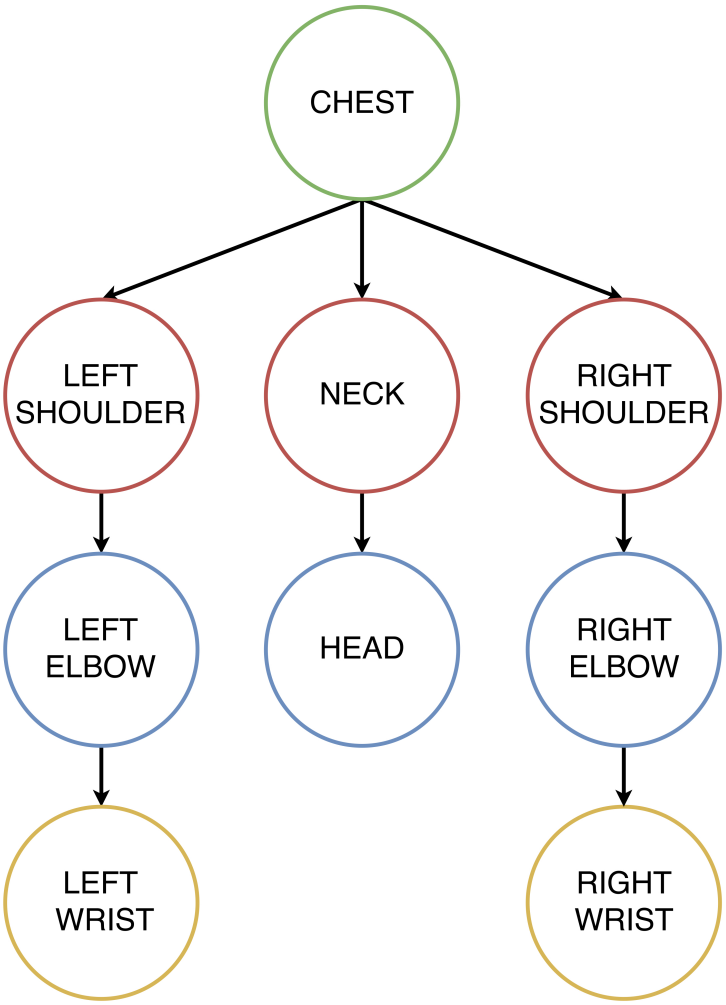


Figure 3.23: Kinematic tree representation of the upper part of the human body.

and his or her torso will remain approximately in the same position at every moment. For this reason, we will consider the chest and the shoulders fixed. In the same way, the neck and head will have a predefined relation with respect to the chest. Hence, we will focus our attention on the analysis of the motion of the arms, more specifically the elbows and wrists.

Therefore, our kinematic tree model will have 8 degrees of freedom (DoF): 4 corresponding to the right arm and 4 to the left one. For each arm, 3 DoF correspond to the shoulder (from which we obtain the elbow's position) and one to the elbow (from which we obtain the wrist's position) [29, 30].

A possible state representation of the joints would be their 3D location, where each joint would be a 3-dimensional vector containing X, Y and Z coordinates. Nonetheless, this definition would be too unconstrained and would not allow us to specify kinematic

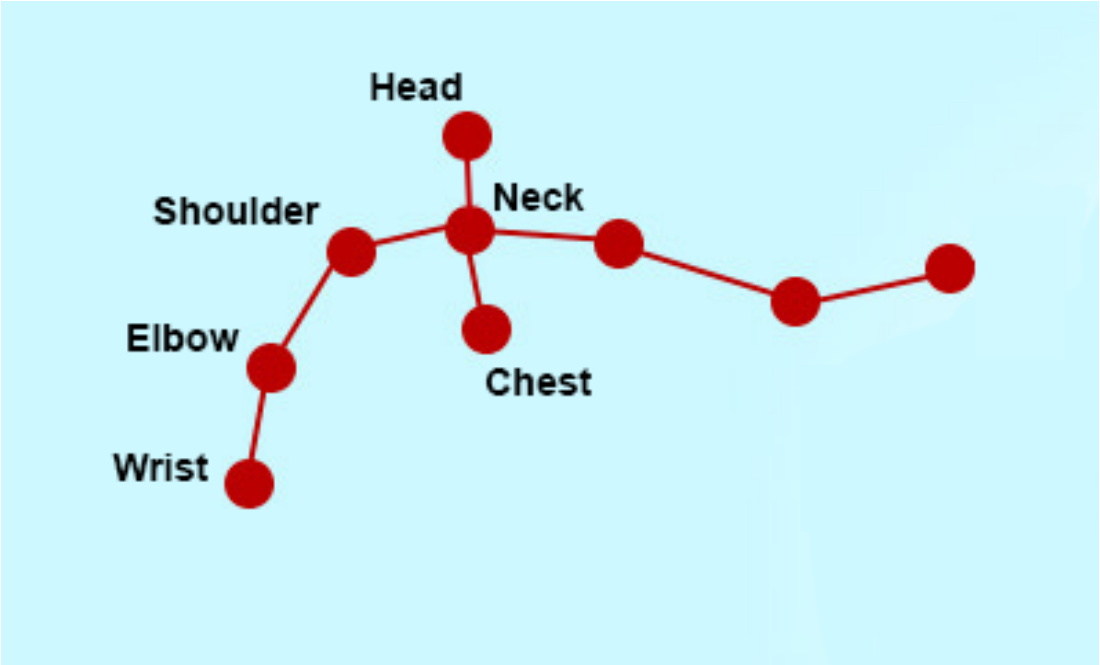


Figure 3.24: Simplified skeleton model of the upper part of the human body.

restrictions in an easy way. A more common way of illustrating the possible rotations of the joints is using Euler angles because they are a natural representation of the DoF of each joint [30] (3 Euler angles are needed for the shoulder to specify the elbow's position and 1 for the elbow to specify the wrist's position). Nonetheless, Euler angles may not be the most natural computational representation of 3D points. For this reason and since the length of the different bones (i.e., the distance between the joints) will be fixed in our model, we decided to use spherical coordinates instead. We will still have 4 DoF for each arm: the azimuth and elevation to find the elbow given the shoulder and the same angles to find the wrist given the elbow.

More specifically, in order to find the right elbow's position, for example, we will fix the origin of a local coordinate system to the right shoulder. Since the distance between the two joints is fixed, we only need the elevation and azimuth to find the 3D location of the elbow. Subsequently, this newly found position will be the origin of a new coordinate system constructed to find the right wrist in an analogous way. Fixing the distance of the bones is a sensible and common approach [10] because it reduces notably the search space: given a shoulder, the corresponding elbow will be on the surface of a sphere with fixed radius (the bone distance) and the shoulder as its center. The distance of all bones was set to 27 centimeters for being the approximate average of the bones of all the subjects (calculated on the merged point cloud). Moreover, with

the state representation chosen, it is straightforward to define kinematic constraints bounding the elevation and azimuth of each joint, reducing even further the search space (the sphere surface). For instance, the elevation of the right elbow with respect to the right shoulder is limited between -90 and 90 degrees, and its azimuth is between -90 and 60 degrees, as seen in Figure 3.25. This Figure also shows a helpful way of defining the axis; a different set of directions would complicate the definition of the kinematic restrictions. The restrictions on the position of the rest of the joints are analogous to the ones just shown. The bounding values are based on the physiology of a regular person. While it is true that these values could be different depending on the person, we selected values that are suitable for all subjects in our experiments (for instance, although a person could reach more negative values than -90 in the azimuth of Figure 3.25, all the relevant movements in the experiments are going to happen in front of the person). Moreover, we assumed that the person is always facing the table simulating the operating room.

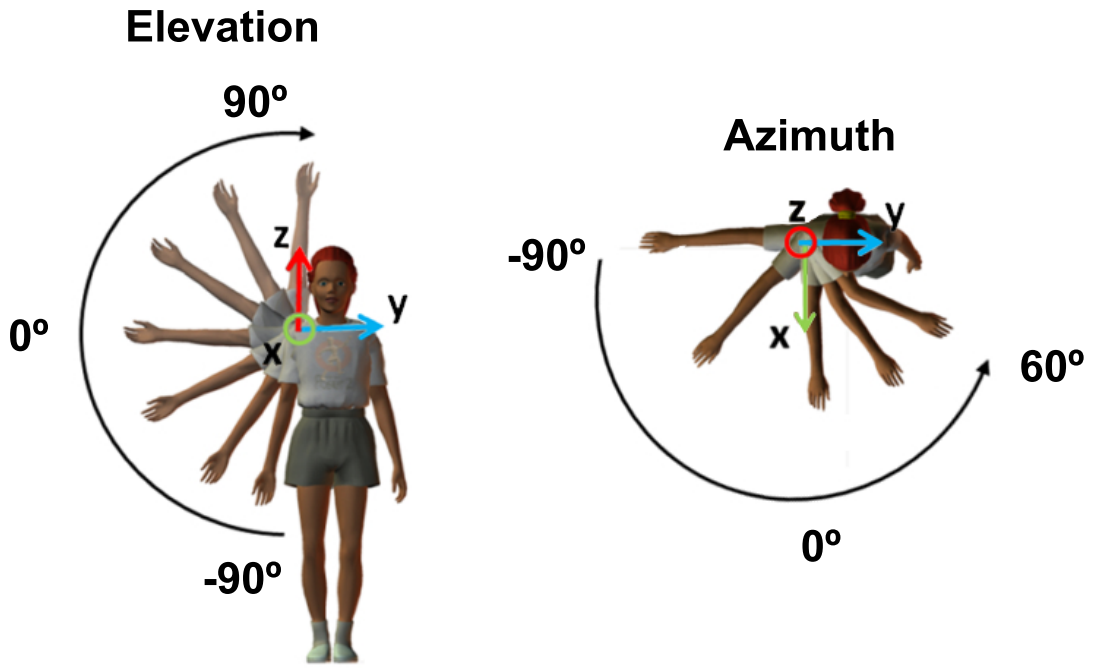


Figure 3.25: Kinematic restrictions on the elevation and azimuth of the right elbow with respect to the right shoulder. Modified image taken from [30].

The complete state of the kinematic model is then a 23-dimensional vector with the fixed positions of the chest, neck, head and shoulders ($3 \cdot 5 = 15$ dimensions), and with the pairs of elevation and azimuth for the elbows and wrists ($2 \cdot 4 = 8$ dimensions). In reality, the only true position is the one of the chest. For the neck, head

and shoulders we define translation vectors to get from the chest to the rest of the nodes in the tree. More formally, the state vector for this project is a vector $\mathbf{x} \in \mathbb{R}^{23}$, such that $\mathbf{x} = (\mathbf{x}_{fixed}, \mathbf{x}_{angles})$, where $\mathbf{x}_{fixed}$ comprises of the fixed position and translation vectors of the chest, neck, head and shoulders (different for every subject), and $\mathbf{x}_{angles} = (el_{rs}, az_{rs}, el_{re}, az_{re}, el_{ls}, az_{ls}, el_{le}, az_{le})$[3] are the angles that the particle filters will estimate.

3.3.1.2 Shape model

The shape model represents the flesh and skin of the human, and it is usually defined by 3-dimensional geometrical shapes such as spheres, cylinders or cones [9, 10]. The shape model plays a significant role in the tracking algorithm. Thus, it is especially important to define proper shapes for the arms, since they will be the parts tracked. We will also define shapes for the torso and the head, but they will not be used in the tracking program because the torso and head are considered fixed. However, it is crucial to define shapes for every part of the upper body in order to avoid collisions with the robot arm in the workcell. These shapes will be the limits from which the robot should stay away. If we only had a kinematic model consisting of 3D points, the robot would not know how close it could move from these points.

We decided to use cylinders to model the upper and lower arms. In the case of the upper arm, the shoulder and the elbow are the centers of the cylinder (see Figure 3.26). For the lower arm, the elbow and the wrist are the corresponding centers. For each cylinder, we assigned a local coordinate system to the point clouds where one of the axes (the Z one) traverses the centers of the cylinder (this will facilitate the job of the observation model in the particle filter). We used a radius of 5 centimeters in both cases. It is common to have human models with different sizes (bone distances and shape widths); however, even though our program is ready to implement this, we decided to use a single human model for all of our experiments. The torso is modeled as a parallelepiped (i.e., a 3D box) and the neck and head as a cylinder. The sizes of all the shapes are determined in a way that the points in the point clouds corresponding to each part are contained inside them. Figure 3.26 shows the shape model with the kinematic model underneath.

[3] $el \equiv elevation$; $az \equiv azimuth$; $rs \equiv right\ shoulder$; $ls \equiv left\ shoulder$; $re \equiv right\ elbow$; $le \equiv left\ elbow$.

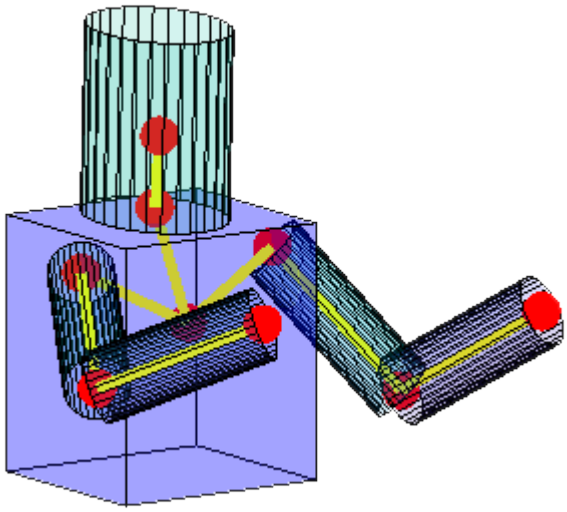


Figure 3.26: Shape model of the upper part of the human body.

3.3.2 Particle filter

The final objective of the tracking part is to estimate the optimal configuration of the human model in every frame given some sensor data. This is equivalent to find the best possible values for the elements of the state vector $\mathbf{x}_{angles} \in \mathbb{R}^8$, already presented in the kinematic model section (Section 3.3.1.1).

In order to solve this problem, we will use particle filtering. The sensor data are the point clouds output by the detection part (Section 3.2) and will be denoted by $\mathbf{z}$. The state is represented by a set of particles $\mathbf{x} = \{\mathbf{x}^{(1)}_{angles}, \ldots, \mathbf{x}^{(m)}_{angles}\}$ where each particle has an associated weight $w^{(i)} > 0$ with $\sum_{i=1}^{m} w^{(i)} = 1$, for $i = 1, \ldots, m$ being m the number of particles. Each particle $\mathbf{x}^{(i)}_{angles}$ represents one hypothesis of the human model configuration; more specifically, each particle holds the azimuth and elevation values from which the elbows' and wrists' locations are determined. Particles with high weight values will be considered more likely to correspond to the true body configuration of the person.

One of the issues of the particle filter is the high dimensionality of the state vector [31]. In our case, 8 dimensions are probably too many to handle for the filter. In order to tackle this issue, we use partitioned sampling [32], which means that we compute a separate particle filter for each of the four joints that we are trying to track (right elbow, right wrist, left elbow, left wrist). In this way, each filter will estimate only vectors with two elements, the azimuth and elevation of the corresponding joint.

The particle filter is a recursive method, applied iteratively to successive images,

in which, in every step, the particles are propagated further using a motion model and the importance weights are corrected using an observation model [28]. More formally, the tracking problem can be seen as a probabilistic inference problem [33] in which we are trying to estimate the posterior probability $p(\mathbf{x}_t|\mathbf{z}_1,\ldots,\mathbf{z}_s)$ of a state vector $\mathbf{x}_t$ given some observations $\{\mathbf{z}_1,\ldots,\mathbf{z}_s\}$, where t and s are time steps (frame indices in our case). If $t < s$, the problem is called smoothing (estimating the past); if $t > s$, it is called predicting (estimating the future); and if $t = s$, the problem is called filtering (estimating the present) [34]. In our case, the problem at hand is a filtering one, more precisely a Bayesian filtering because we are using prior knowledge coming from the human model.

It is customary in the Bayesian filtering framework to make two assumptions that are not very restrictive, but that simplify the problem enormously [33]:

- *Only the immediate past matters* or alternatively have a first order Markov process:

$$p(\mathbf{x}_t|\mathbf{x}_1,\ldots,\mathbf{x}_{t-1}) = p(\mathbf{x}_t|\mathbf{x}_{t-1}).$$

- *Observations depend only on the current state*:

$$p(\mathbf{z}_t|\mathbf{x}_1,\ldots,\mathbf{x}_t,\mathbf{z}_1,\ldots,\mathbf{z}_{t-1}) = p(\mathbf{z}_t|\mathbf{x}_t).$$

With these two assumptions, it is easy to compute the posterior density wanted using Bayes' rule as follows:

$$p(\mathbf{x}_t|\mathbf{z}_1,\ldots,\mathbf{z}_t) \propto p(\mathbf{z}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{z}_1,\ldots,\mathbf{z}_{t-1}). \tag{3.1}$$

The prior density is computed recursively from the previous time step:

$$p(\mathbf{x}_t|\mathbf{z}_1,\ldots,\mathbf{z}_{t-1}) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_{t-1}|\mathbf{z}_1,\ldots,\mathbf{z}_{t-1})\,\mathrm{d}\mathbf{x}_{t-1}. \tag{3.2}$$

As we can see, the structure of the tracking problem is equivalent to the one of a Hidden Markov Model (HMM), where the point clouds $\mathbf{z}_t$ correspond to the observed states and the configurations $\mathbf{x}_t$ of the human model play the role of the hidden states in the HMM.

There is no general solution for the expressions in (3.1) and (3.2). However, restricted solutions exist if we assume all the densities to be Gaussian[4]. This is the case of

[4]Also, we only consider the case in which the state space is continuous, as it is our case. For discrete scenarios, other techniques such as histogram filters are required [35].

the Kalman filter in which the dynamics of the system are considered linear. We deemed Kalman filter as not the most adequate method for our problem because, as already mentioned, it can only represents unimodal hypothesis. The dynamics of our system, on the other hand, are non-linear and need a multimodal representation. In fact, the hands and elbows of the person will change direction (thus the non-linearity) too rapidly for the Kalman filter to keep track of them [21]. For this reason we used particle filtering, which is capable of representing simultaneous alternative hypotheses (corresponding to each particle) that can be seen as a mixture of Gaussians degenerated to Dirac delta functions. Another advantage of the particle filter is its simple implementation compared to other types of filters.

The particle filter approximates the posterior probability $p(\mathbf{x}_t|\mathbf{z}_1,\ldots,\mathbf{z}_t)$ by taking samples from the prior distribution $p(\mathbf{x}_t)$[5]. These samples are the particles that, in our case, are 2-dimensional vectors consisting of the azimuth and elevation of a joint. As the number of samples tend to infinity, the filter approaches the optimal Bayesian solution [36]. In practice, increasing the number of particles will increase the precision of the model, but will also incur larger computational time. The likelihood $p(\mathbf{z}_t|\mathbf{x}_t)$ in (3.1) will play the role of the importance weights that will sum to one for being probabilities of the same density.

Hence, the particle filter will complete the following steps iteratively:

1. Assign an importance weight to each particle using the observation model. For the first frame, all weights take the same value ($\frac{1}{m}$, being m the number of particles) and the particles are randomly initialized (more details in Section 3.3.4).

2. Resample the particles based on the weights assigned, weeding out the particles with lower weights, corresponding to less probable configurations. At this point, many particles will be duplicate.

3. Move the particles using the motion model to avoid duplicates and to simulate a prediction of the future configuration. In our case, the motion model is only adding noise and it is not actually performing a real prediction; however, other models, such as the Newtonian model [21], could use additional information from the state space and would perform a more educated guess. For this reason, the motion model is also known as prediction stage.

[5]Here $\mathbf{x}_t$ should be written as $\mathbf{x}^{(i)}_{angles,t}$ for $i = 1,\ldots,m$ being m the number of particles. However, in the remainder of this section we will refer to a generic particle at time t as $\mathbf{x}_t$ to improve readability.

4. Re-assign weights for the next step using the observation model.

A graphical visualization of these steps can be seen in Figure 3.27. The curves outlined are just an illustration of the unknown distribution that we are trying to approximate by sampling particles. The sizes of the gray circles in the resampling stage represent the magnitudes of the importance weights: bigger circles correspond to particles with larger weights that will have more probability of being resampled. For example, the biggest gray circle in the resampling step corresponds to a particle with high probability that is resampled three times. These three resamples are then moved by the motion model and subsequently analyzed by the observation model. Now that we went through all the particle filter theory it is not hard to see that the motion model corresponds to the expression $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ in (3.2), from which we sample, and the observation model is the likelihood $p(\mathbf{z}_t|\mathbf{x}_t)$ in (3.1). We still need to decide the specific form of these two expressions for our current problem of tracking the articulated human body.

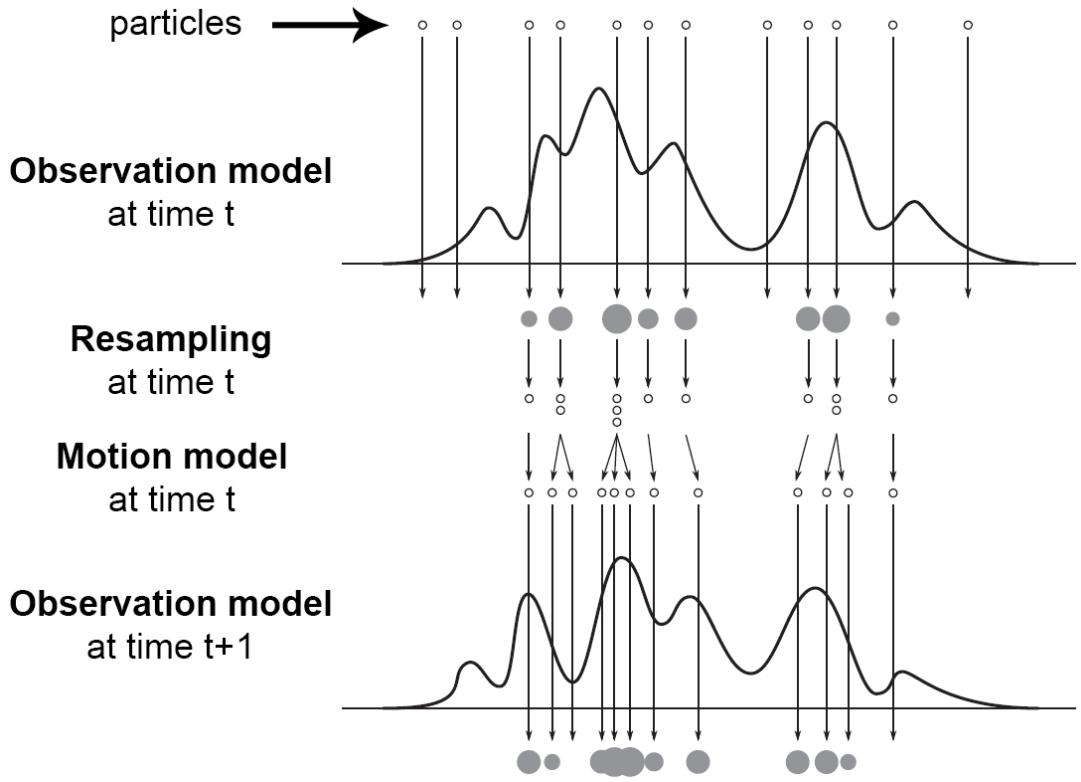


Figure 3.27: Iterative process computed by the particle filter. Modified image taken from [37].

Since we are using a different particle filter for each joint, we need to concatenate the corresponding filters according to the kinematic tree in Figure 3.23. For every new

frame, we will have different hypothesis (particles) of the position of the elbow; then, we will take the one with the highest weight as the starting point of the particle filter responsible for locating the wrist. That is, the best particle from the particle filter of the elbow will be the origin of the coordinate system from which the wrist's position will be found, estimating its azimuth and elevation.

It can be argued that decoupling each arm into two different particle filters may not be the most optimal strategy. In fact, an error in the filter responsible for tracking the elbow will spread to the filter in charge of locating the wrist, leading to not only a wrong predicted position for the elbow, but also for the wrist. Sometimes this error can be small and the effects on the wrist can be negligible; at other times the consequence can be a complete failure on both the elbow and the wrist tracking. Even though it is true that these kinds of errors can occur, we believe that defining a proper observation model would limit their frequency of occurrence and would justify the use of partitioned sampling. In fact, Zhang et al. [10] and MacCormick and Isard [32] already implemented successful tracking algorithms using partitioned sampling[6]. Remember that we used partitioned sampling to reduce the curse of dimensionality in the particle filtering. Thus, a trade-off exists between the quality of the model and the time needed to compute it, that is proportional to the number of particles in the model: the more particles used, the better the model should be (but not always due to the randomness involved in the process), but also the computational time will increase. The curse of dimensionality in the particle filter arises from the fact that, in order to maintain the same level of performance when the state space increases, we need to also increase the number of particles. This can be seen using the following expression [32]:

$$D \approx \alpha m, \tag{3.3}$$

where D is called *survival diagnostic* and represents the number of particles that would survive after the resampling process shown in Figure 3.27, α is called *survival rate* and m is the number of particles in the filter. A low value of D could mean that a filter is unreliable. If we want to achieve a survival diagnostic that is equal or greater than a certain value D that assures good performance, we would need $m \geq D/\alpha$ particles. If α corresponds to the tracking of a single object, it can be proven [32] that α^2 is the

[6]Zhang et al. partitioned a human body into 5 parts, the 4 limbs and the torso and MacCormick and Isard built a hand tracker divided into 4 parts, the fist, the first joint of the thumb, the second joint of the thumb and the index finger. In Zhang's program, for example, a bad tracking of the torso could have ruined the predictions of the other particle filters, but the overall results were successful. In our project, we are splitting the human body into different parts, but the outcome will also be positive as we will see in Chapter 4

survival rate for two objects in the same filter (two joints in our case). If we want to achieve the same value D in this case, we would need $m \geq D/\alpha^2$ particles. Since α is usually a very small positive value, the number of particles added is substantial. We considered that the addition of two dimensions to our particle filters (in order to have a single particle filter for each arm) would have increased the number of particles and the computational time too much to obtain satisfactory results in an acceptable amount of time.

3.3.2.1 Motion model

As we just saw, in every iteration the particles are propagated from one time step to another using the motion model $p(\mathbf{x}_t|\mathbf{x}_{t-1})$. In our case, this means literally moving the particles some distance in the 3D space. However, the particles do not hold the actual positional information (X, Y and Z coordinates), but instead they consist of the azimuth and elevation of a properly defined coordinate system. Hence, the motion model has to introduce some variation on these two values of each particle in order to simulate movement along the surface of the sphere representing the valid space of the particles. We decided to use the following motion model for each particle:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \varepsilon, \tag{3.4}$$

where $\varepsilon \sim \mathcal{N}(0,\Sigma)$ is Gaussian distributed and Σ is a 2-dimensional diagonal matrix. The elements in the diagonal represent how much the azimuth and elevation angles are allowed to vary. Individually, each part of the motion model for every particle, where $\mathbf{x}_t = (el_t, az_t)$, is equal to:

$$el_{t+1} = el_t + \varepsilon_{el}, \tag{3.5}$$

$$az_{t+1} = az_t + \varepsilon_{az}, \tag{3.6}$$

where $\varepsilon_{el} \sim \mathcal{N}(0,\sigma_{el})$ and $\varepsilon_{az} \sim \mathcal{N}(0,\sigma_{az})$ are now univariate Gaussian distributions. The standard deviations σ_{el} and σ_{az} represent the agility of each joint. We consider all the joints to have the same agility and thus, they have the same standard deviations in the motion model. Also, we believe that the variation of azimuth and elevation is very similar and they take the same values as well (nonetheless, our model is ready to tune all these values separately). The fact of using the same value for all the angles and all the joints not only allowed us to have easier comparisons between different values, as we will see in the experimental part (Chapter 4), but also led us to good results. Despite the simplicity of the motion model, the noise value (i.e., σ_{el} and σ_{az}) influences

the quality of the model to a large degree. In fact, as seen in Figure 3.28, too much noise would move the particles too far away from the actual observations (bottom), losing track of the joint, and too little would keep them almost still (middle), preventing them to follow the person's movements. As we observe, the particles are subject to the kinematic restrictions defined previously because they are all representations of joints in the human model.

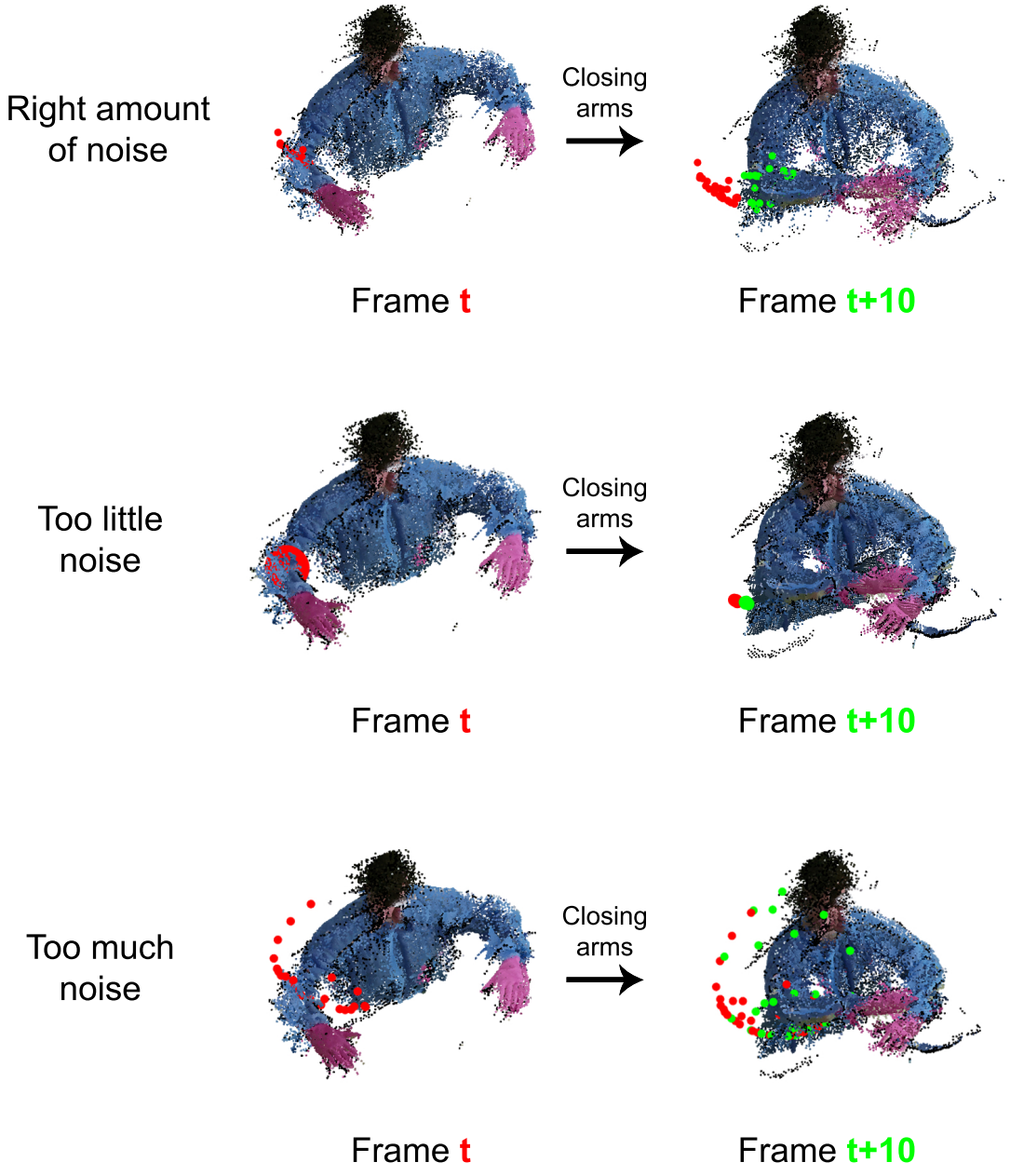


Figure 3.28: Effect of noise in the motion model of particles representing the right elbow. The red dots are particles in a frame at time t and the green dots are particles at time $t + 10$. The subject is closing his arms from one frame to the other.

In the experiments conducted we found that the best value for the noise was 0.2 (see Section 4.5). This value is in radians and corresponds to approximately 11 degrees. Since we know that in a Gaussian distribution roughly 68%, 95% and 99% of the values lie within one, two and three standard deviations of the mean respectively, we can infer that 68%, 95% and 99% of the particles will move less than 11, 22 and 33 degrees in their azimuth and elevation planes from one frame to another respectively.

3.3.2.2 Observation model

The goal of the observation model is to compare the pose estimates coming from the motion model with the point clouds output by the detection part (Section 3.2), in order to determine a likelihood (or importance weight) $p(\mathbf{z}_t|\mathbf{x}_t)$ for every particle $\mathbf{x}_t$. We will use the shape model from the previous section (Section 3.3.2.1) to calculate a score for each hypothesis. Remember that, since the m particles are actual representations of the joints, we will have m different configurations of the shape model (with the corresponding kinematic model underneath). Thus, for the right upper arm, for instance, we will have m different cylinders, each of them corresponding to a different hypothesis of the human model.

A common way of doing this in the literature is minimizing a signed distance [10]. For example, consider a cylinder corresponding to the right upper arm (constructed with the fixed point of the right shoulder and a particle representing the right elbow): a point in the point cloud will have a signed distance equal to zero if it lays exactly on the surface of the cylinder, it will have a negative distance if it lays inside the cylinder and a positive one if it stands outside. Figure 3.29 shows a schematic representation of the signed distance where a cylinder is seen from one of the bases. The green area represents a predicted cylinder coming from one particle (thicker than in reality for illustrative purposes). The points in the green area are laying on the surface of the cylinder and will have a signed distance equal to zero. The points in the orange and red areas are outside the cylinder and will have a positive distance and the ones in the cyan and blue zones are inside the cylinder and will have a negative signed distance. The signed distance is normally bounded between two predefined values (one negative and one positive).

This is a sensible approach for when the person stands in the center of the scene and ideally wears tight clothes that prevents excessive noise. Nevertheless, that is not the case for the frames captured in our project. As we can see in Figure 3.30, the right arm of the person is missing one part that is not captured by any Kinect. For this reason, if

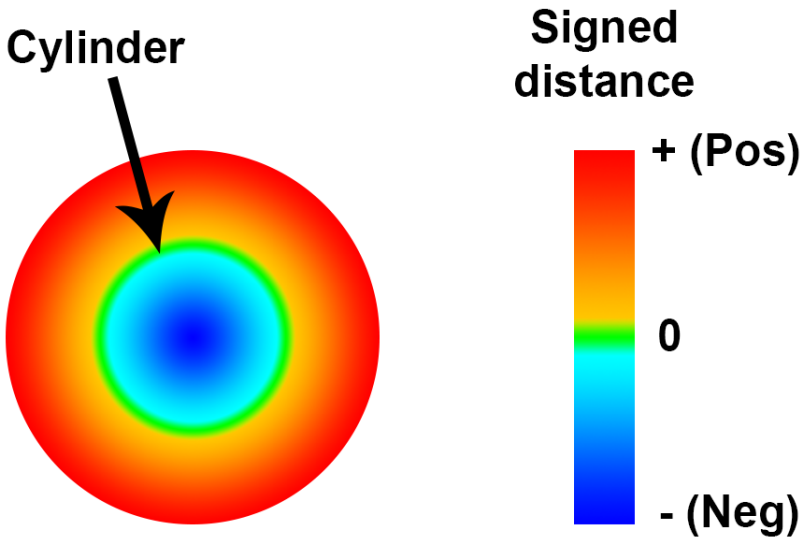


Figure 3.29: Schematic representation of the signed distance. The image shows a view of the predicted cylinder from one of the two bases. The green area represents the cylinder; points in this zone have zero signed distance. Points in the orange and red areas are outside the cylinder and have a positive signed distance. Points in the cyan and blue areas are inside the cylinder and have negative signed distance.

we use the signed distance to fit a cylinder to the lower arm in this Figure, the cylinder with the lowest signed distance will probably be wrongly positioned, closer to the left arm. In fact, this cylinder will include almost all the points corresponding to the lower arm, assigning to them a negative value, and will not have enough close[7] points outside (that would have a high positive value) to counteract the negative score of the inner points. Moreover, the high amount of noise in our recordings and the use of loose clothes makes the use of a signed distance even more difficult.

For the above mentioned reasons, we decided to use a measure more robust to noise, holes and missing parts: the density of the point cloud or, in other words, the number of points in the point cloud. Therefore, when trying to fit a cylinder to the lower right arm, for instance, the particle corresponding to the cylinder with more inner points will be the one with the largest weight. This calculation is very simple because in the shape model we constructed a local coordinate system with the origin on one of the cylinder's centers and the Z axes traversing both centers. Thus, we only need to convert the number of inner points of each cylinder to probabilities, which means dividing the number of inner points in each cylinder by the sum of all the inner points in every cylinder, so all the

[7]Points further than some distance would not be considered.

Figure 3.30: Point cloud where the person is missing one part of the right arm.

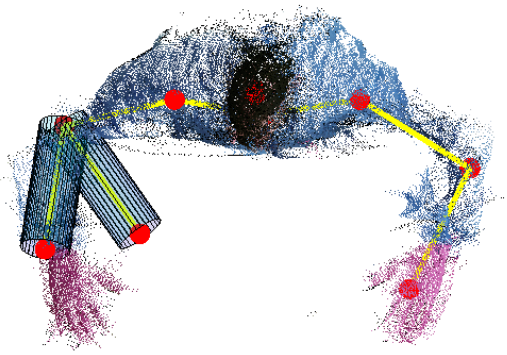


Figure 3.31: Example of a good fit (left cylinder) and a bad one (right cylinder) for the lower right arm. The person is seen from above.

normalized values sum to one and are valid likelihoods. As an instructive example, consider a model with 3 particles where each of the three cylinders, originated from said particles, contain 20, 30 and 50 points respectively (we do not care about the rest of the points in the point cloud, the outer ones). Then, the normalized weights assigned to each of these particles will respectively be $20/(20+30+50) = 20/100 = 0.2$, $30/100 = 0.3$ and $50/100 = 0.5$.

Figure 3.31 shows an example of a cylinder that fits the point cloud correctly and one that it does not. As we can observe, the cylinder on the left contains more points of the point cloud than the one on the right and hence, the particle corresponding to the left cylinder gets a higher weight than the one on the right. Since the arms may not be seen entirely by the cameras in some frames, we also added a lower bound to the number of points inside a cylinder. If the number of inner points of some cylinder is less than a predefined threshold fixed experimentally, then that particle will correspond to a missing joint. Figure 3.32 shows an example where the cameras did not captured the right arm of the person, meaning that both the best particle of the elbow and the wrist did not have enough inner points in their corresponding cylinders and thus, they are correctly not plotted. Additionally, Figure 3.33 shows a different final example of the tracking algorithm where all the body parts seem to be correctly tracked (a quantitative evaluation will be done in the experimental part in Section 4.5). Algorithm 2 shows a simplified version of the pseudocode of the particle filter for a generic joint.

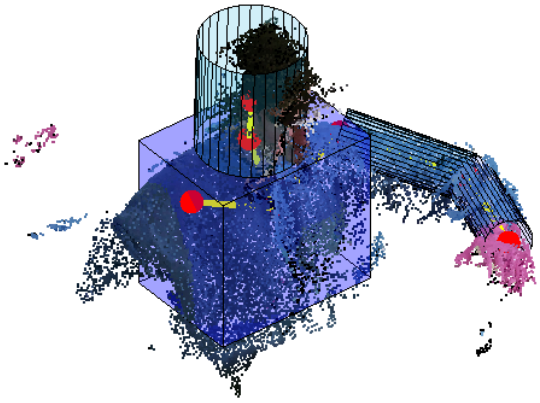


Figure 3.32: Final example of a frame where the right arm is not captured by the cameras. The model correctly does not plot the missing arm.

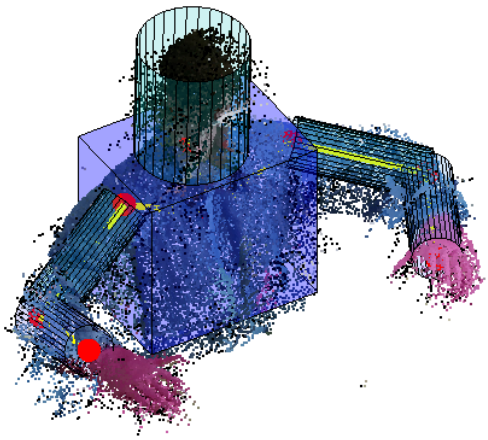


Figure 3.33: Final example of a person correctly tracked.

Algorithm 2 Particle filter to track a generic joint

Precondition: $pcl(t)$ is the point cloud at time t with the origin being the best hypothesis of the parent joint in the kinematic tree at every time step.

Initialize $particles(1)$ with their kinematic restrictions
Initialize $weights(1) \leftarrow 1/NUMBER\_PARTICLES$
for $t \leftarrow 2, NUMBER\_FRAMES$ **do**
  $particles(t) \leftarrow$ RESAMPLE$(particles(t-1))$ with $weights(t-1)$
  ▷ Motion model
  $particles(t) \leftarrow particles(t) + motionNoise$
  for $p \leftarrow 1, NUMBER\_PARTICLES$ **do**
    ▷ Observation model
    $innerPoints(p) \leftarrow$ #points $\in pcl(t)$ inside $CYLINDER(t,p)$
  end for
  $weights(t) \leftarrow$ NORMALIZE$(innerPoints(p))$ ▷ So that $\sum_t weights(t) = 1$
end for

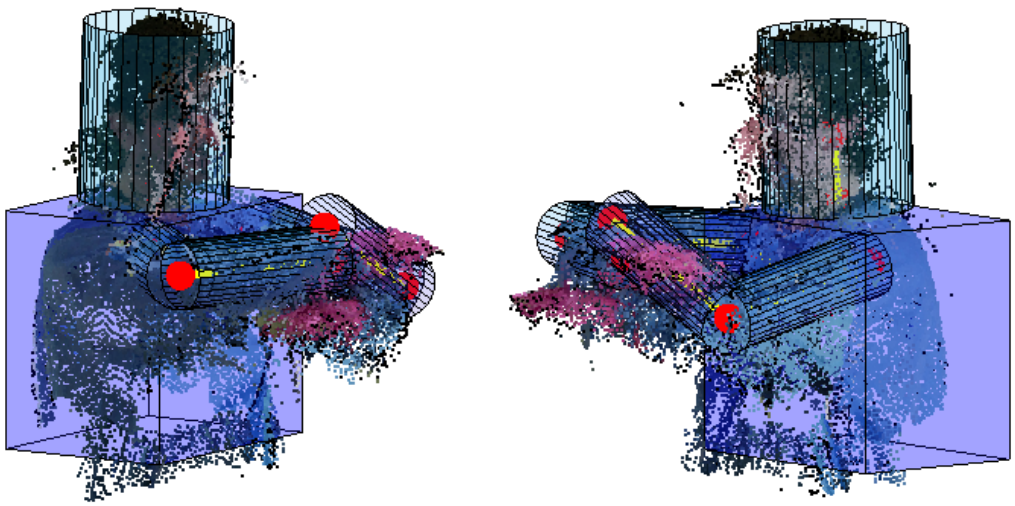
3.3.3 Data association

One of the problems encountered in the strategy followed in this project is related to the procedure known as data association. This process refers to the task of determining which points in the point cloud are informative and which are not (usually called *clutter*)

for a specific particle filter [33]. Consider the case in which we are trying to track the lower right arm and the point clouds at our disposal contain only points corresponding to the lower right arm. In this case, all the points available would be informative and we would be fairly sure that the cylinder with more inner points is the best possible configuration for the arm model. Nonetheless, this case is unrealistic unless some previous process segmented the point clouds in different body parts (e.g. Shotton et al. [19]), and even in that case some noise would exist.

In our case, the point clouds are not labeled and so, some cylinders of the lower right arm could be counting some points that actually belong to the torso or to the left arm as inner points. Avoiding the torso is a simpler task achieved by proper kinematic restrictions that prevent the arm cylinders to end up inside of the torso. However, there is not an easy way to differentiate between points of the left and right arm when both limbs are close together. Hence, if the two arms are close we could have failures as the one seen in Figure 3.34 where the density of points in the right arm is higher that the one in the left, leading the predicted joint of the left wrist to end up in the right hand instead.

A solution to this problem is not trivial. We could introduce some heuristic rule that prevents points in the point cloud to be counted as inner points by two cylinders at the same time. However, it is not clear which filter should be computed first. Furthermore, if a point is assigned incorrectly to one cylinder, later it could not be counted in the correct filter. Perhaps a different observation model could help solve the problem, however we could not find an appropriate one that deals with this problem as well as with the other peculiarities of our dataset (noise, loose clothes, holes). Maybe an entirely different optimization algorithm such as the particle swarm used by Michel et al. [9] could help solve the problem. Nonetheless, they did not test any movement with both forearms close together and we cannot be sure. Even though we will find occasional failures in this particular movement (with the arms close together), it was an expected behavior and the action was knowingly designed as probably the hardest one for our model. In any case, the errors detected in this set of actions will not be very high because the two arms are close together anyway. Moreover, when the two limbs are so close it is hard even for a human to tell exactly where one arm starts and the other one ends. For these reasons, we consider acceptable the actual model and leave further improvements in this regard for future work.



(a) Left view. (b) Right view.

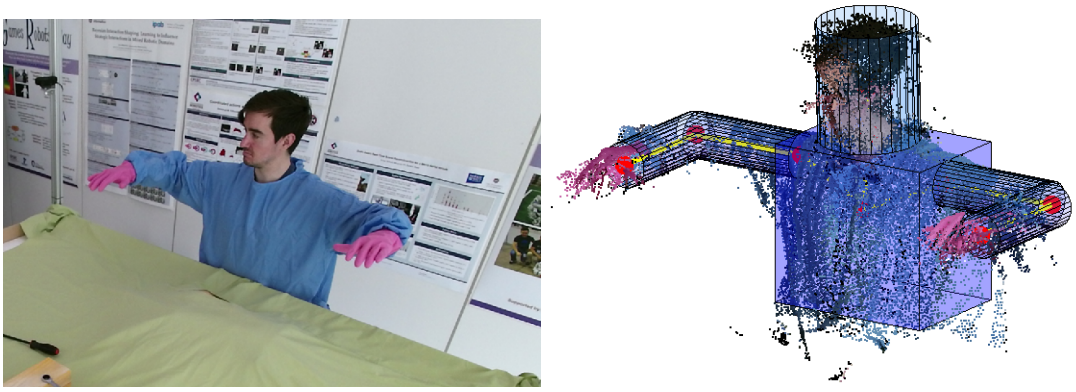
Figure 3.34: Example of a failure where the left wrist of the human model ends up in the right hand of the observed data.

3.3.4 Tracking initialization

As we mentioned already, one of the inevitable drawbacks of top-down algorithms is the need of an initial 3D pose in the first frame of the image sequence. Until now, we have assumed that in every step of the particle filters the observations were close to the predictions of the motion models. If this was not the case (imagine all the cylinders corresponding to the lower arm spawning in the back of the person), the filter would assign high probability to configurations that do not correspond to the actual data and it would continue to do so for the remainder of the calculations. For this reason, in the first frame of every sequence we need to have the human model close to the actual point cloud.

In order to do so, we asked the subjects to start the set of actions in a specific pose. We then initialized all the particles in a way to match the initial pose. Every particle was initialized randomly around the area where the actual point clouds were supposed to be. Thus, the particle filters start their actual computations from the second frame, where they compare the point clouds in the second time step with the particles initialized at random in the first frame and moved by the motion model. The initial pose selected has the human standing with both arms raised and parallel to the ground, and the elbows bent 90 degrees as shown in Figure 3.35. Notice that assigning the best model on the first frame was done retroactively, mainly for illustrative purposes and it is not actually

part of the particle filter. In fact, the importance weights of the particles take the same value in the first frame and the filter starts to act in the second frame.



(a) Color image. (b) Point cloud and best human model.

Figure 3.35: View of the color image and the corresponding point cloud with the best human model of the first pose.

The chest position and the translation vectors to get the neck, head and shoulders are set manually for each person based on the first frame of one of their videos. More exactly, we manually find the torso of the person and calculate the chest as the centroid of the points in the torso. Then, we determine the translation vectors to get the rest of the joints based on the chest position. Normally, since the sequences were recorded one after the other, it is enough to examine just one video and use the same values for the rest, since the person is staying in the same position. However, for the recordings where the subjects are wearing their own clothes, a new set of initial parameters was calculated because the person starts far from the table and then walks towards it, reaching most of the times a slightly different position than in the rest of the videos.

Chapter 4

Experiments

In this chapter, we will report all the experiments realized using the methodology presented in Chapter 3. We will describe the scene composition as well as the designed data set that will be publicly available to the scientific community along with this thesis. Finally, we will present a quantitative evaluation of our research to assess the validity and quality of the program built.

4.1 Environmental Setup

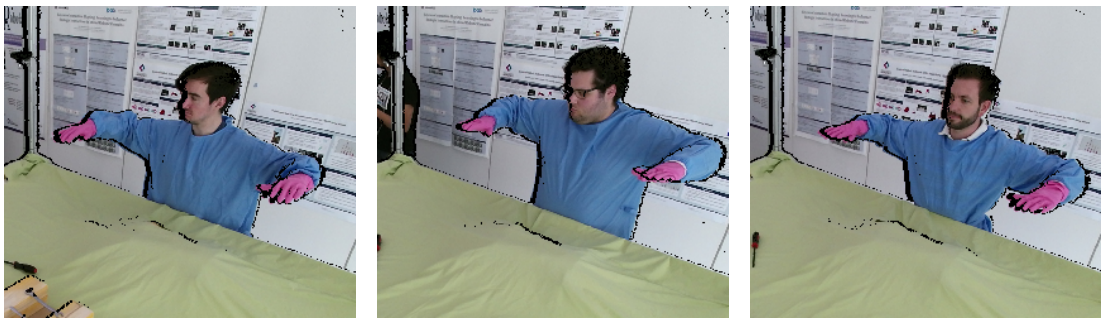
This section is similar to text in the Informatics Research Proposal (IRP) [1].

For the purpose of this and similar projects, a cubical workcell simulating an operating theater has been built in one of the labs in the School of Informatics of the University of Edinburgh. A robotic arm has been ceiling-mounted at the center of the structure as seen in Figure 1.1. In addition, 4 RGB-D Kinect version 2 sensors have been positioned at the four corners of the structure, facing downward towards one of the sides of the structure where the person simulating the clinician will stand. The simulated operating table will be covered with a green cloth and on top of it there will be a mannequin, simulating the patient, and several simulated surgical tools.

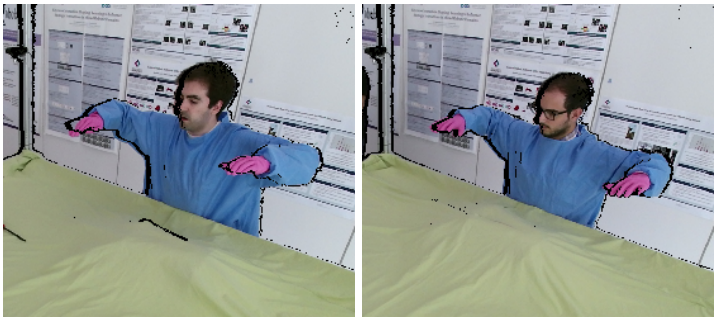
4.2 Data Set

One of the main contributions of this thesis is a new dataset, which we call 5six (5 people, 6 actions) conceived for the problem of articulated human motion tracking using multiple RGB-D sensors. We designed various sets of actions trying to cover a wide range of movement variability. We recorded 6 videos at 5 frames per second of 5

different people with different heights and constitutions that can be seen in Figure 4.1. The dataset consists of a total of 2398 frames.



(a) Subject 1. (b) Subject 2. (c) Subject 3.



(d) Subject 4. (e) Subject 5.

Figure 4.1: The 5 subjects of our 5six dataset.

The 6 actions were conceptualized with an increasing level of difficulty, from actions with almost no occlusions, to movements with several occlusions, arms very close together and no distinguishable colors. The five first sets of actions were recorded with the subject wearing a blue medical gown and pink gloves. The last set of videos of each person was recorded with the subject wearing his own clothes in order to assess the generalization power of the tracking algorithm. Every video starts with the person in the initial pose shown in Figure 3.35, except for the sixth action where the person first walks towards the table and then performs the initial pose before starting the set of actions. Next, we describe in detail and show examples of the actions designed. In every video the same action is repeated between 2 and 4 times.

Action 1: The subject extends his arms in front of his body and waves them simultaneously from left to right. This action does not involve any crossing or occlusions. Figure 4.2 shows the movements in action 1.

Action 2: The subject extends his arms and crosses them horizontally in front of his body. On the way back while uncrossing them, the person moves one of

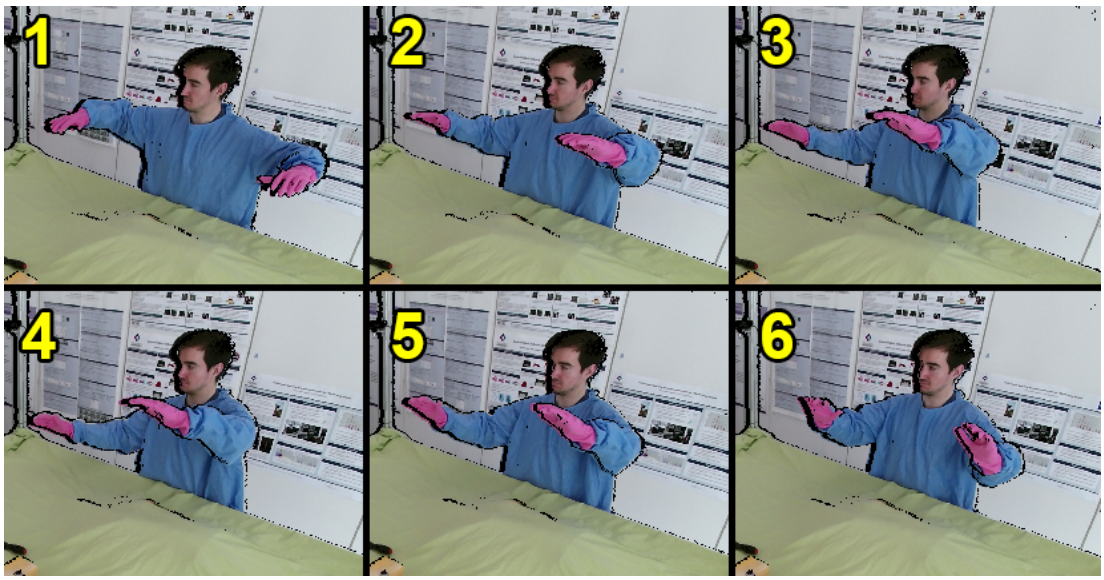


Figure 4.2: Movements of action 1.

the arms towards his back, while leaving the other arm in front of his body. This movement is performed alternatively with one arm and then with the other until the end of the recording. Figure 4.3 shows the movements in action 2.

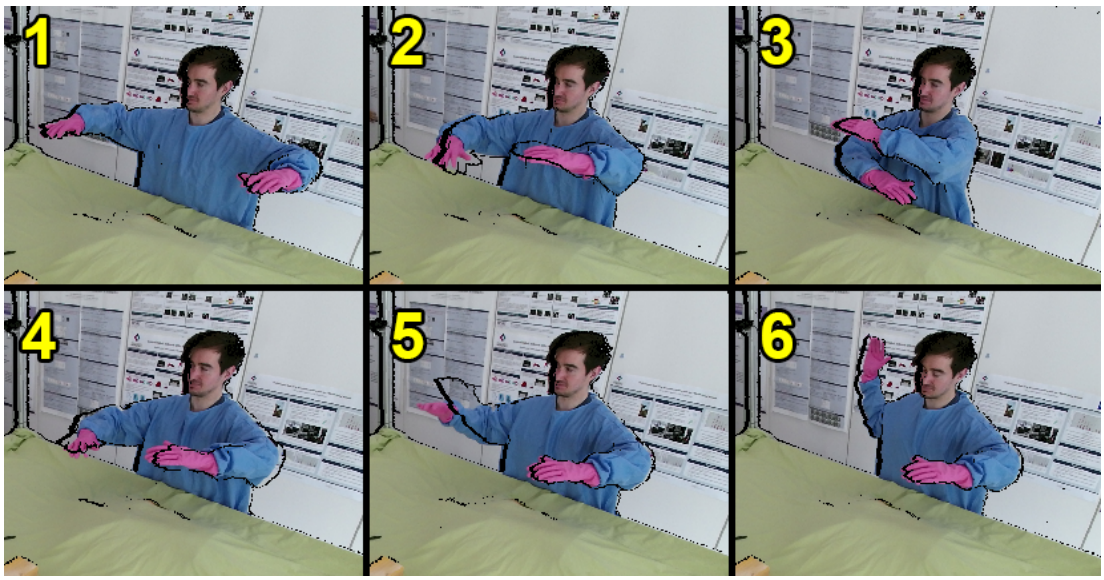


Figure 4.3: Movements of action 2.

Action 3: The subject extends his arms and crosses them vertically in front of his body. Figure 4.4 shows the movements in action 3.

Action 4: The subject extends his arms and put the hands together like in a *high five*.

Figure 4.4: Movements of action 3.

Right after, the person touches the table with one hand, alternating between right and left throughout the whole recording, and then go back to the *high five* position. Figure 4.5 shows the movements in action 4.

Figure 4.5: Movements of action 4.

Action 5: The person extends his arms and puts one forearm on top of the other, grabbing the elbows with the opposite hand. Then, the person separates the arms. The left and right arms alternate being on top for the duration of the recordings. Figure 4.6 shows the movements in action 5.

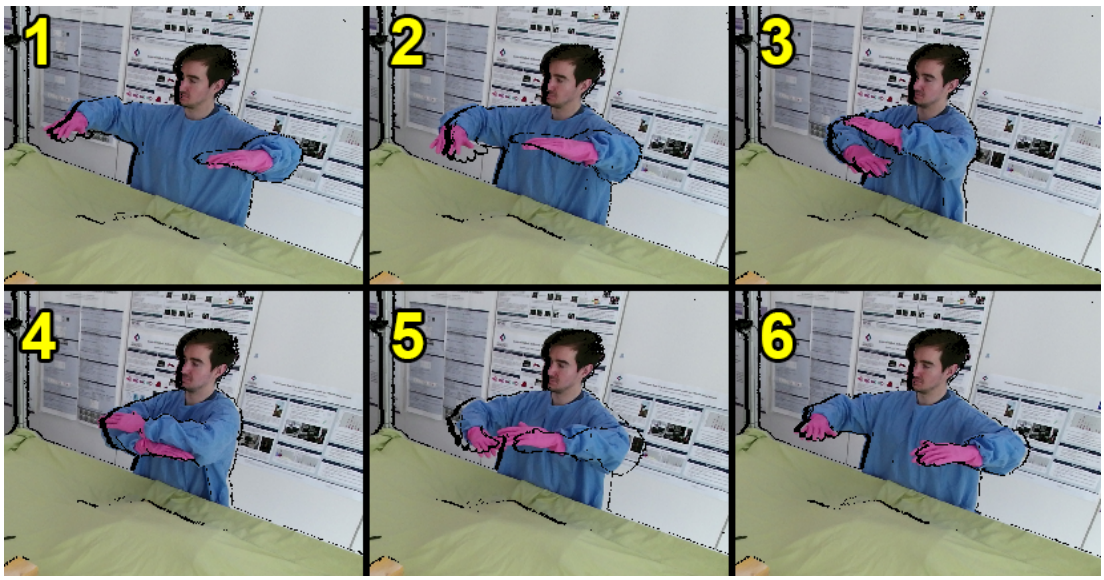


Figure 4.6: Movements of action 5.

Action 6: The person extends his arms and crosses them horizontally. The person in this set of videos is wearing regular clothes. Figure 4.7 shows the movements in action 6.

A total of 2398 frames were collected. Table 4.1 shows the number of frames recorded in every video. The average number of frames recorded for a single video is equal to 80 frames. Since the frequency of the videos was set to 5 fps, the average duration of a video is equal to $80/5 = 16$ seconds.

	Action 1	Action 2	Action 3	Action 4	Action 5	Action 6	All actions
Subject 1	118	136	94	92	78	115	633
Subject 2	80	58	80	70	51	62	401
Subject 3	66	84	80	70	51	62	413
Subject 4	76	123	87	78	80	88	532
Subject 5	61	69	82	69	47	91	419
All subjects	401	470	423	370	313	421	2398

Table 4.1: Contingency table of the number of frames recorded in every video.

All the subjects gave their permission to be recorded and for their images to appear in this thesis and on the web.

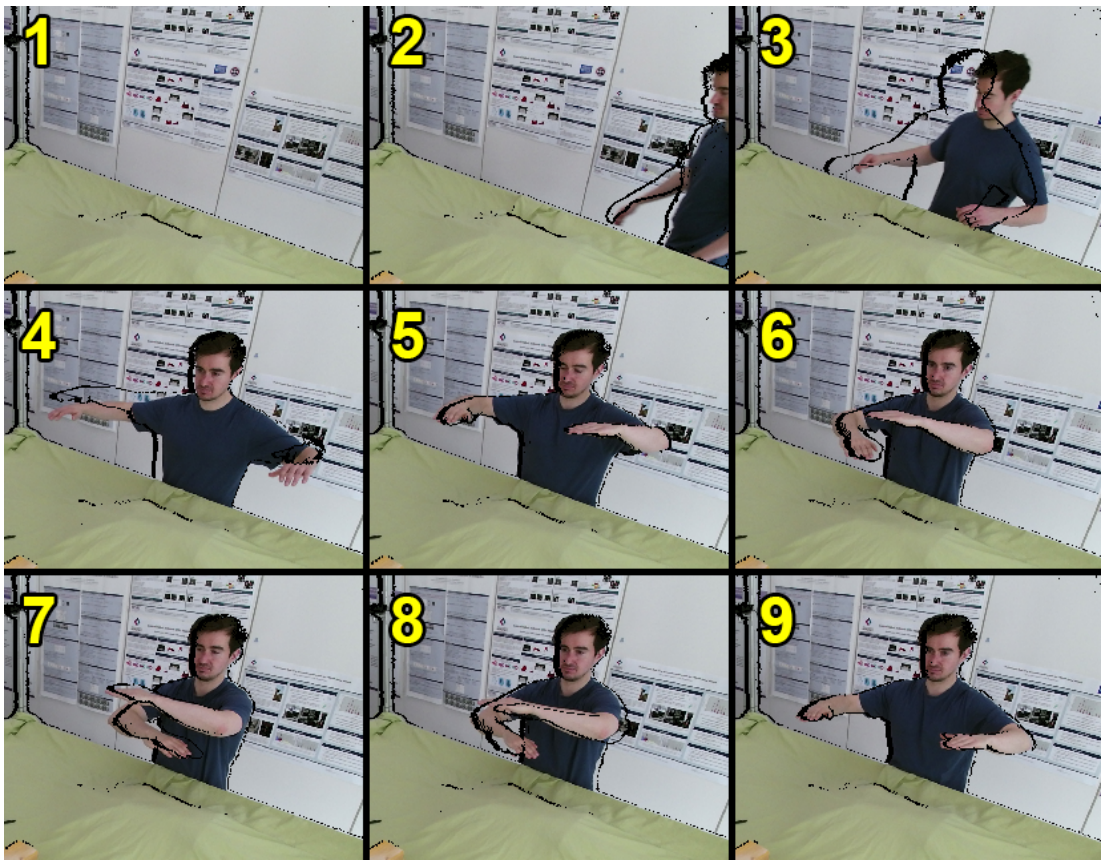


Figure 4.7: Movements of action 6.

4.3 Ground Truth

In order to assess the performance of the tracking program implemented, we need to have empirical evidence to compare the predictions with. For this purpose, we manually labeled the 3D position of the elbows and wrists of the person in every tenth frame of each video. We chose not to label all frames due to time constraints and because we believed that having one frame from every ten frames labeled was enough. We avoided labeling the first few frames of each video because we would be assessing the ability of the person to position himself in the initial pose instead of the quality of the model.

Finding the X, Y and Z coordinates of each joint was a straightforward process because the global coordinate system of the merged point clouds has the axes appropriately defined. More specifically, the X axis is parallel to the ground and crosses the person from the back to the chest; the Y axis is parallel to the ground and traverses both shoulders of the person; the Z axis is perpendicular to the ground and its direction crosses the person vertically.

Figure 4.8 shows an example of two views that were used to label the joints. As we

can see, we only need to use the vertical and horizontal lines in the grid to determine the 3D coordinates. We tried to find the 3D point corresponding to the center of each joint. For this reason, we could not use an actual point in the point cloud as an approximation of the joints: the point clouds only capture the external surface of the body, thus no point corresponds to the center of the joint. The views shown in Figure 4.8 are just two examples. We used more views from other angles and we also zoomed in when needed to have more precise values for the 3D positions.

Ideally, a ground truth would be the result of a joined effort of various people: each of them would manually label every joint in the dataset and the final 3D position of a joint would be, for example, the mean of the positions labeled by each person. In this way, we could also calculate the standard deviation for each labeled joint in order to assess the quality of the ground truth. As an example, consider three people labeling a joint in a point cloud and let's call them person A, person B and person C. If person A determines that the X coordinate of the joint is equal to 0.2 and person B and C determine that it is 0.3 and 0.4 respectively, then the final X coordinate of the joint would be 0.3, the mean between the three values, and the standard deviation would be 0.1.

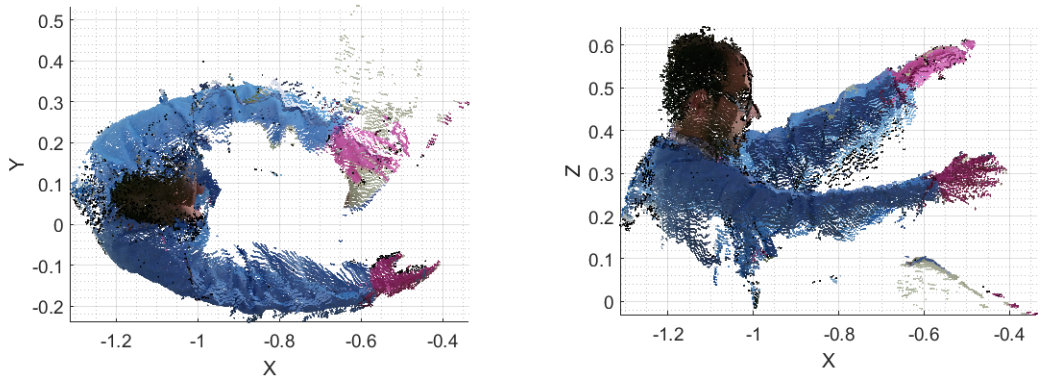
In our case, we could not afford to have other people labeling the dataset. For this reason, in order to asses the quality and validity of our ground truth, we repeated the label process on a subset of 50 joints and calculated the euclidean distance between the new labeled positions and the positions labeled the first time. The average of these distances is equal to 3.28 centimeters. Since we will use an accuracy with a 10 centimeters threshold (Section 4.5) as our evaluation metric, we can say that our ground truth is valid to assess the performance of our tracking program.

4.4 Evaluation metrics

This section is similar to text in the Informatics Research Proposal (IRP) because we will use the same metrics already reported there [1].

When comparing the predictions of the model (coming from the position found using the predicted azimuth and elevation and transforming it to the original coordinate system) and the ground truth, a metric is needed to assess the quality of the results obtained. For this project, we will use two common metrics in the human body pose estimation field that were used also by Michel et al. [9] in their research.

The first metric calculates the distance between the position of an estimated joint



(a) View from above.

(b) View from one side.

Figure 4.8: Two example views of a merged point cloud used to label the ground truth.

and the position of the corresponding labeled joint in the ground truth. The average of all these distances over all the frames available will be the final estimate error E of our system. Clearly, we can get estimated errors for separate videos, actions or subjects, selecting only the appropriate frames. The average error for one frame at time t is as follows:

$$E_t = \frac{1}{n}\sum_{i=1}^{n} |r_i - e_i| = \frac{1}{n}\sum_{i=1}^{n} d_i, \tag{4.1}$$

where n is the number of joints (4 in our case for the two wrists and the two elbows) and r_i and e_i are the ground truth position and the estimated position respectively of joint i. Hence, the final estimation error would be:

$$E = \frac{1}{T}\sum_{t=1}^{T} E_t, \tag{4.2}$$

where T is the number of frames in the ground truth. T can also refer to the total number of labeled frames with subject 1 on them, for example, leading to an estimate error of the tracking algorithm for said subject.

The second metric is the percentage of distances d_i that are within a predefined threshold. Consider a threshold of 10 centimeters. If 80% of the distances between the predicted and labeled position of the joints are smaller than 10cm, then the metric will be $A = 80\%$ and we will refer to it as the accuracy in human tracking. This metric is very similar to the classic statistical accuracy in the sense that an example with a distance d_i smaller than the predefined threshold is considered a correct classification. Our metric then counts all the correct examples and divides the result by the total number of examples, as in the regular accuracy. The expression for this metric is the

following:

$$A = \frac{1}{T}\sum_{t=1}^{T} A_t = \frac{1}{T}\sum_{t=1}^{T}\left(\frac{1}{n}\sum_{i=1}^{n}\mathbf{1}_{|r_i - e_i|<\gamma}\right) = \frac{1}{T}\sum_{t=1}^{T}\left(\frac{1}{n}\sum_{i=1}^{n}\mathbf{1}_{d_i<\gamma}\right), \tag{4.3}$$

where γ is the threshold and $\mathbf{1}_x$ is the indicator function that is equal to 1 if x is true and 0 otherwise. This second metric not only has a more direct interpretation (a percentage of correct examples instead of an error value that needs additional knowledge to be interpreted), but also compensates for the small inaccuracies in both the predicted results and the ground truth.

These two metrics are valid when there are estimated and ground truth joints available. However, they do not take into account frames in which the tracking algorithm fails to output a joint that is actually in the image (missed detection), or frames in which a joint is not in the image but the tracker produces a model of it anyway (false detection). We will calculate the percentage of frames in which these two errors happen. Thus, the errors E and accuracies A will only be computed in the frames where the program does not miss or falsely detect a joint.

To summarize, we will be using the four following different criteria to assess the validity of our tracking model:

- Average error E.
- Human tracking accuracy A.
- Missed detection rate.
- False detection rate.

4.5 Quantitative Evaluation

4.5.1 Parameter Tuning: Motion Noise and Number of Particles

Some of the parameters used by the tracking algorithm were determined based on empirical observations. For example, we set the distance of the bones in the kinematic model (Section 3.3.1.1) to 27 centimeters for being the approximate length average of the subjects in the 5six dataset. The width of the cylinders in the shape model (Section 3.3.1.2) was set to 10 centimeters (5cm radius) in an analogous way. A similar situation holds for the minimum number of inner points in a cylinder to activate a specific joint, or for the various thresholds used in the region growing algorithm (Section 3.2.1.1) in

the detection part. Some of these parameters could have been modified in different runs of the program, but the difference in the final results would have been minimal, such as changing the bone length (adult humans have usually similar bone lengths); some other parameters, such as the thresholds in the detection part (Section 3.2.1.1.4), can take a range of values that will output roughly the same results and that can be easily found after a few manual tests.

Nevertheless, there are other parameters that cannot be determined so easily and that influence the final results significantly. In our program, we found that the two parameters with the highest impact on the final predictions are the noise of the motion model (Section 3.3.2.1) and the number of particles in the particle filters (Section 3.3.2). We already examined the behavior of both parameters (see Figure 3.28 and final paragraph of Section 3.3.2). Too much noise would spread the particles too far apart from the point cloud and too little noise would prevent the particles from following the movement of the person. However, we cannot know the corresponding values for "too much" and "too little" noise unless we run the whole program several times using different values for the standard deviation of the Gaussian noise in the motion model. As we already mentioned, the fact of using the same noise for all the joints will allow us to have easier comparisons. In regards to the number of particles, a higher volume of particles increases the performance of the model but also the time needed to run the program. Nonetheless, the performance improvement is not linear with respect to the number of particles: as we increase this number, the marginal improvement of the model decreases (effect of diminishing returns). This is easily seen by looking at how the particle filter works: if there is one particle, it is very unlike that it corresponds to a good configuration of the human model; hence, adding four more particles, for example, will most likely improve the accuracy of the predictions. On the other hand, if we already have a million particles, it is very improbable that four new particles will correspond to better configurations than the best one from the million particles.

In order to have a rigorous tuning of the two parameters[1] at hand, we will perform a grid search based on some manually selected values. Normally, random searches of the parameters have higher performance than manually ones [38]. Nonetheless, a grid search is still reliable in low dimensional spaces like ours (2 dimensions) and gives a deeper insight than a random search. Thus, in order to have a better understanding of the effect of the parameters and taking into account the time constraints of the project,

[1] Normally this process is called hyper-parameter tuning, but since the particle filter is a non-parametric model, we will call it simply parameter tuning.

we chose the following values for our two parameters:

- For the *motion noise*: 0.01, 0.1, 0.2 and 0.3 radians, corresponding approximately to 0.57, 5.72, 11.45 and 17.18 degrees respectively.
- For the *number of particles*: 10, 30 and 50 particles.

We have a total of 12 different cases. For each of those we run our tracking program on the 30 videos of our 5six database for a total of $12 \times 30 = 360$ executions of the program.

Figure 4.9 shows the human tracking accuracy defined in Section 4.4 of the four models that use 10 particles against the threshold in the definition of the accuracy. Figures 4.10 and 4.11 show analogous results for the models with 30 and 50 particles respectively. Finally, Figure 4.12 compares the best models from each of the previous Figures. Each of the points in these curves correspond to the total accuracy of all the videos in the dataset.

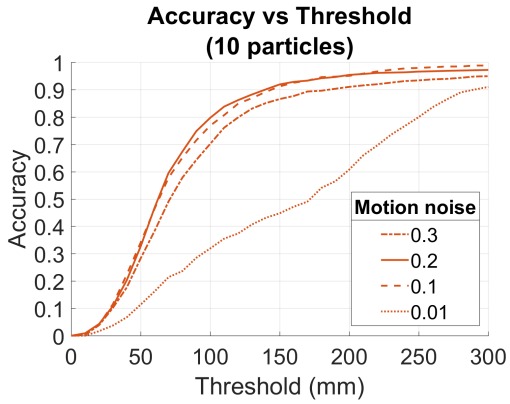


Figure 4.9: Accuracy against the threshold in the accuracy definition of models using 10 particles.

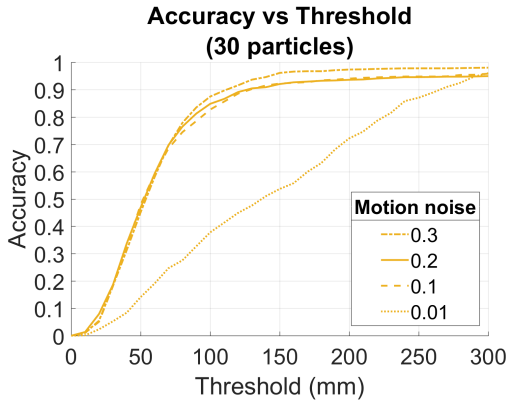


Figure 4.10: Accuracy against the threshold in the accuracy definition of models using 30 particles.

As we observe, the models that use a motion noise equal to 0.01 perform poorly until reaching a threshold of 30 centimeters, which seems too high to be acceptable. It is logical that the accuracy for any model will tend to 1 as we increase the threshold to consider a prediction as correct. Therefore, we are interested in getting high accuracy values with the lowest possible threshold, meaning that the precision of our model is very high. However, this situation is hard to achieve as can be seen in the previous Figures by the low accuracies and high steepness of the curves when the threshold is lower than 10 centimeters. Obtaining high accuracies with low thresholds is difficult

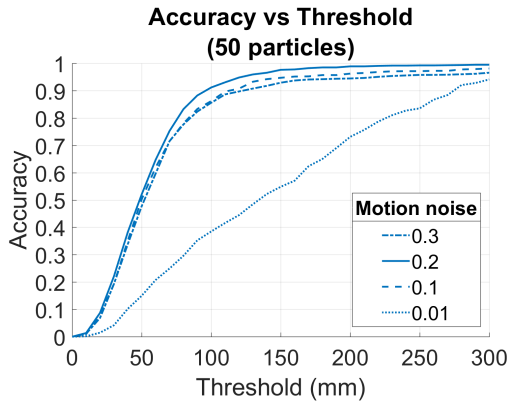


Figure 4.11: Accuracy against the threshold in the accuracy definition of models using 50 particles.

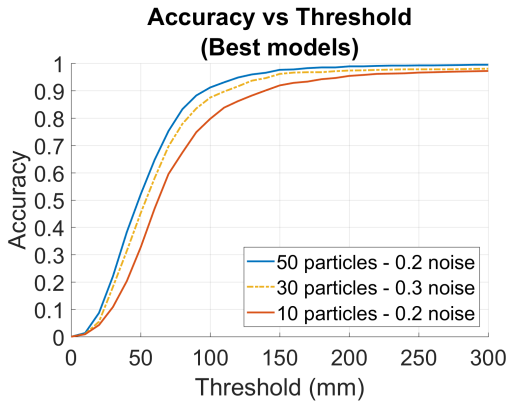


Figure 4.12: Accuracy against the threshold in the accuracy definition of the best model with 10 particles, the best model with 30 particles and the best model with 50 particles.

due to the high variability in the model caused by the low resolution of the depth sensors, the imprecisions of the tracking algorithm and the possible inaccuracies in the ground truth. For these reasons, it seems sensible to choose a threshold equal to 10 centimeters for the remainder of the comparisons. Moreover, since Michel et al. [9] used the same threshold value, it will be easier to compare their results with ours.

Figure 4.12 shows that the best model out of the 12 considered is the one that uses particle filters with 50 particles and a motion noise equal to 0.2 radians. In fact, this model seems to be superior to the rest for any threshold selected (as the blue curve is always on top of the other two). We can also start noticing the diminishing return effect in the bigger gap that exists between the red and yellow curves than between the yellow curve and the blue one. It is possible that models with more than 50 particles would achieve slightly higher accuracies. Nevertheless, we considered that the extra computational time needed for those models was not worth the minimal expected increase in accuracy.

Figures 4.13 and 4.14 show a simultaneous comparison between the 12 computed models. As we would expect, the model with lower average error and highest accuracy is the model with 50 particles and 0.2 radians of motion noise, with a final accuracy of 91.16% (localization within a threshold of 10cm). Table 4.2 shows the exact error and accuracy values obtained in the 12 different models.

The false detection rate is equal to zero for all the 12 models. This is not unusual

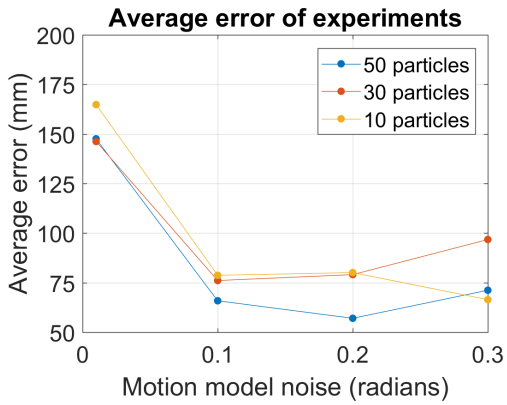


Figure 4.13: Average error against the noise in the motion models.

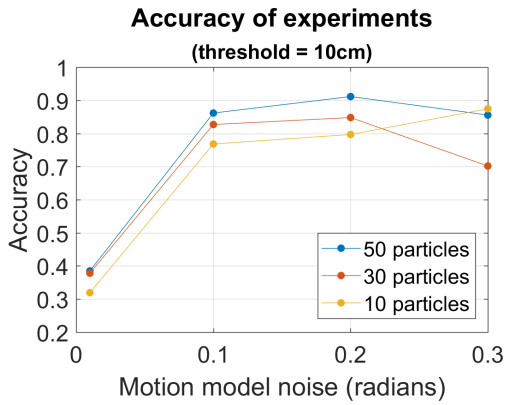


Figure 4.14: Accuracy against the noise in the motion models.

	10 particles		30 particles		50 particles	
Motion noise	Error (mm)	Accuracy	Error (mm)	Accuracy	Error (mm)	Accuracy
0.01 radians	164.84	0.3191	146.22	0.3780	147.55	0.3849
0.1 radians	78.68	0.7682	76.03	0.8271	65.82	0.8617
0.2 radians	80.04	0.7969	79.03	0.8481	56.99	0.9116
0.3 radians	66.39	0.8747	96.74	0.7018	71.15	0.8557

Table 4.2: Average errors and accuracy values of the 12 models computed. The highlighted cells corresponds to the lowest error and highest accuracy.

due to the threshold of minimum number of points necessary to output a joint that we introduced in the observation model (Section 3.3.2.2). On the other hand, Figure 4.15 shows the miss rate detection of the 12 models computed. As we observe, this percentage is negligible and all the failures come from examples such as the one seen in Figure 4.16. In this case, the right arm is labeled as present in the scene. However, the number of points in the point cloud representing the lower arm are too few for the model to output the limb. Moreover, the arm seems to be behind the person, in which case the model would not output a joint due to the kinematic restrictions. We decided to leave these frames as they are because these kinds of situations happen very rarely and when they occur, they do not affect the final error and accuracy values.

4.5.2 Evaluation of the best model

Now that we have determined that the best model is the one where the four particle filters are using 50 particles and a noise in the motion model equal to 0.2 radians, we

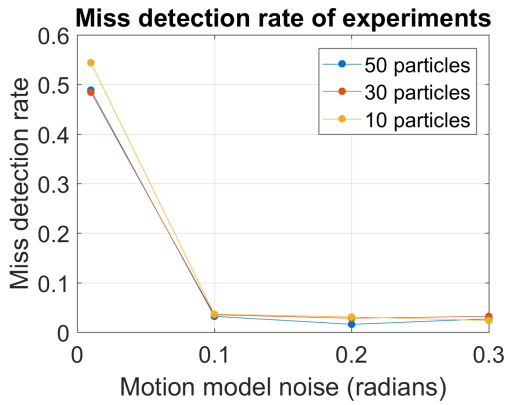


Figure 4.15: Miss detection rate against the noise in the motion models.

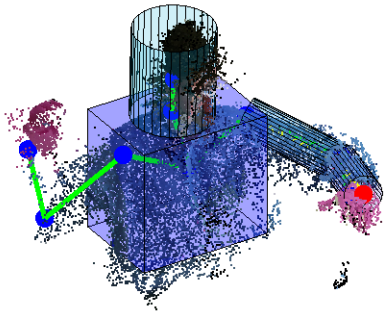


Figure 4.16: Missed joint example where the right arm has been missed by the model. The green bones and blue joints represent the ground truth. The yellow bones and red joints are the predictions from the model. This example comes from the model with 50 particles and 0.2 motion noise.

can compare the error and accuracy of the individual videos in the dataset for this model. Figures 4.17 and 4.18 show respectively the average error and accuracy with a threshold of 10cm of every video in the 5six dataset. The error plot shows also confidence intervals of 2 standard deviations, meaning that, for each bar, approximately 95% of the errors lie within each interval. As we can see, the results are not far from the ones of Michel et al. [9], shown in Figure 2.6, where they use a more advanced technique such as particle swarm optimization [9]. In our results, there is no evidence of an action or a person with worse results than the rest. Nonetheless, it seems that the action 6 achieves slightly better results than the rest of the actions, obtaining a perfect accuracy in three different people. Even though the perfect accuracy is probably caused by the few frames available in the ground truth, it is true that our model seems to perform better in the videos of action 6. This is understandable because in the recordings of action 6 the subjects are wearing tighter clothes and even short sleeves. As a consequence, the resulting point clouds have less noise and are a more clear representation of the person in the scene.

It is also interesting to evaluate the performance of each joint separately. First, Figure 4.19 shows the histograms of all the estimated errors of each joint. We can see that the majority of the errors, with the exception of some outliers, are distributed

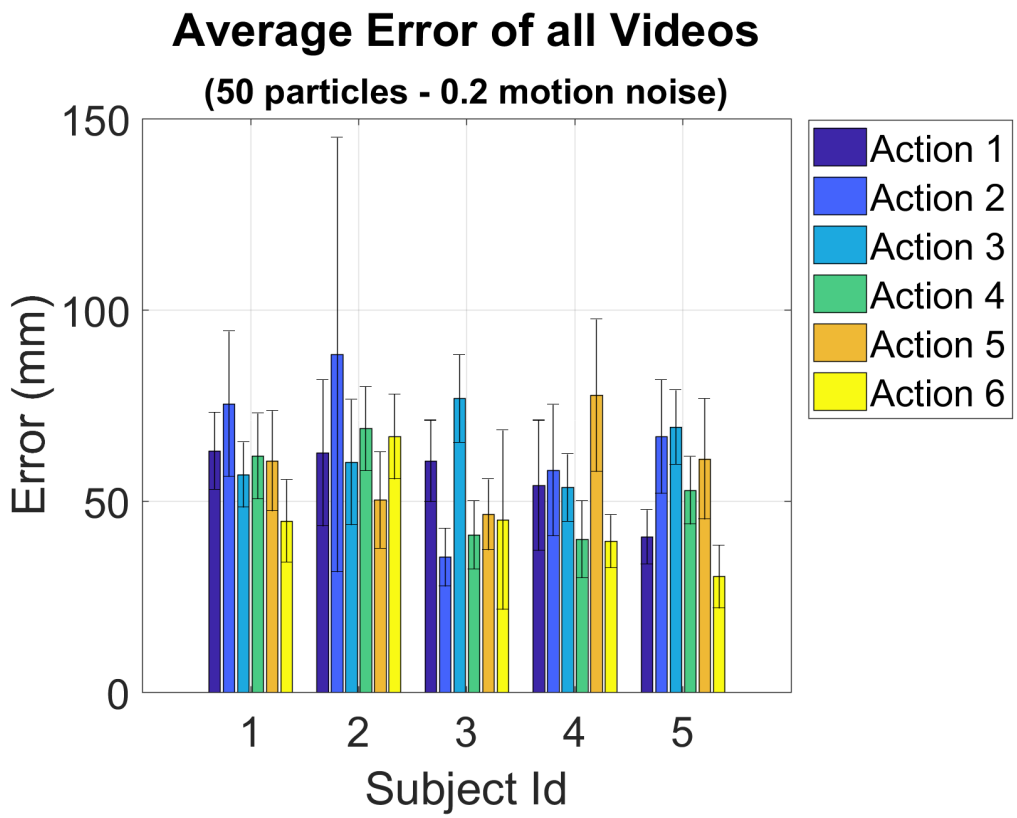


Figure 4.17: Average error and confidence intervals (of 2 standard deviations) of every video in the dataset of the best model with 50 particles and 0.2 motion noise.

between zero and a bit more than 10 centimeters. Thus, it seems sensible to use 10cm as the threshold for the accuracy metric. Figure 4.20 shows the average error of each joint with a confidence interval of 2 standard deviations. Figure 4.21 shows the accuracy of every joint in the model of 50 particles and 0.2 motion noise. As we can see, the wrists' predictions are less accurate than the elbows' ones (both have smaller accuracies, higher errors and larger confidence intervals). This is expectable because the wrists nodes in the kinematic tree in Figure 3.23 are the children of the elbows nodes, meaning that every inaccuracy in the position of the elbows will most likely be transmitted to the wrists (discussed in the last paragraph of Section 3.3.2). Furthermore, we can notice a small difference in performance between the left and right wrists, while both elbows seem to have almost the same accuracy and error. This could be caused by a difference in density of the point clouds between the left and right wrists. In fact, while performing our experiments, we observed that sometimes the left arm of the person was less dense than the right one, probably due to the orientation of the cameras. This fact seems to affect more the particle filters responsible for locating the wrists than the ones for

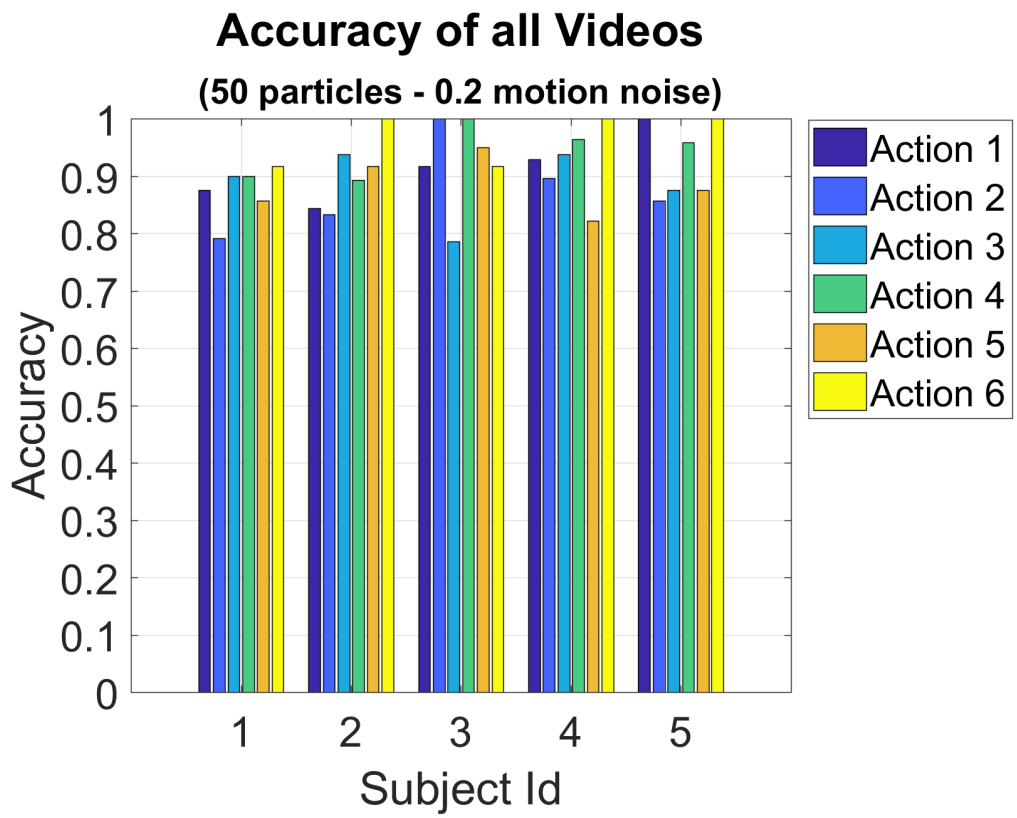
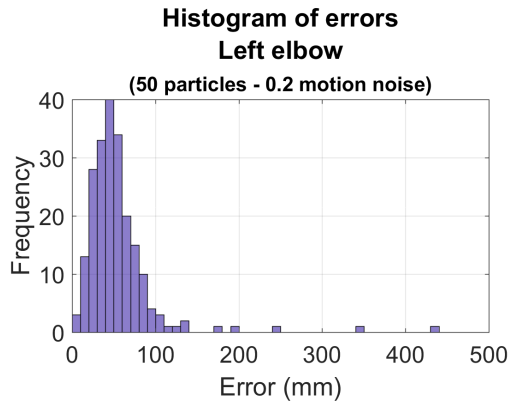


Figure 4.18: Accuracy with a 10cm threshold of every video in the dataset of the best model with 50 particles and 0.2 motion noise.

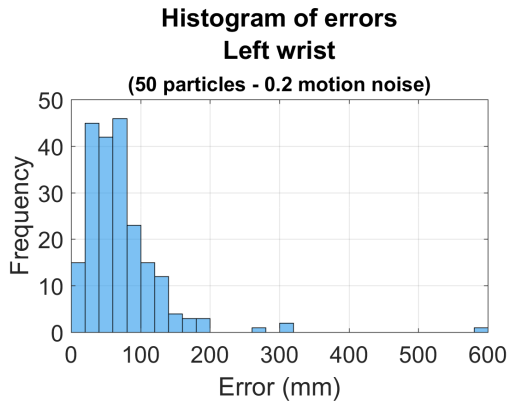
locating the elbows. An explanation could be that the upper arms are in general more dense than the lower arms and so, the particle filters that use the lower arms in the observation model (the ones locating the wrists) could be more affected by low densities of the point clouds.

4.5.3 Merged vs separate point clouds

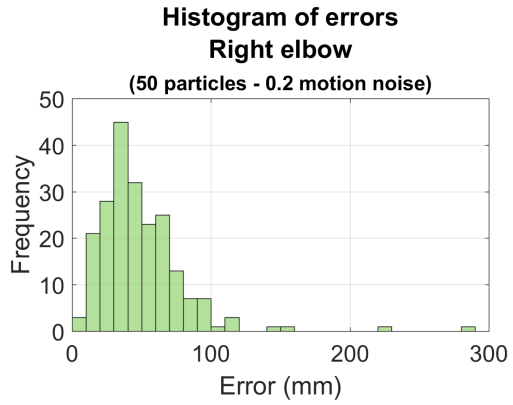
All the results presented until now in this section (Section 4.5) come from particle filters that use as input point clouds merged using the four point clouds of the four Kinects, as seen in Section 3.2.3. The reason for this was that certain Kinects could miss some body parts due to occlusions, the low resolution of the depth sensors or the position from which they are facing the scene. In this section, we are going to investigate the usefulness of this fusion strategy in a quantitative manner. In order to do this, we run the tracking part of the algorithm four additional times using the parameters of the best model found in Section 4.5.1 (50 particles and 0.2 radians as motion noise). For each of these runs, we used the separate point clouds output by each Kinect (before the merging



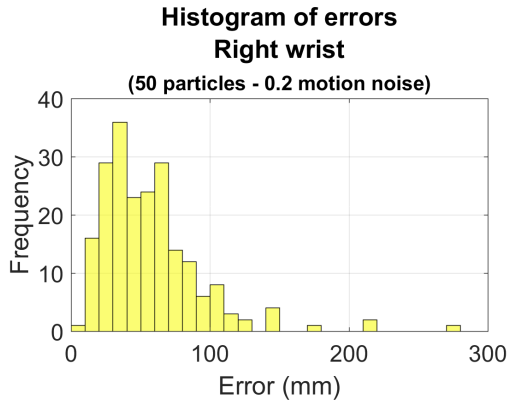
(a) Left elbow.



(b) Left wrist.



(c) Right elbow.



(d) Right wrist.

Figure 4.19: Histograms of the estimated errors of each joint.

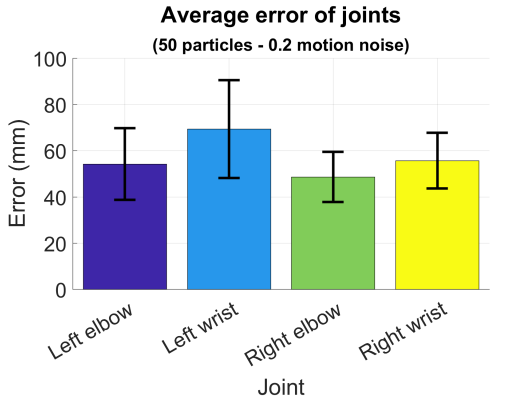


Figure 4.20: Average error and confidence intervals (of 2 standard deviations) of each joint in the model with 50 particles and 0.2 motion noise.

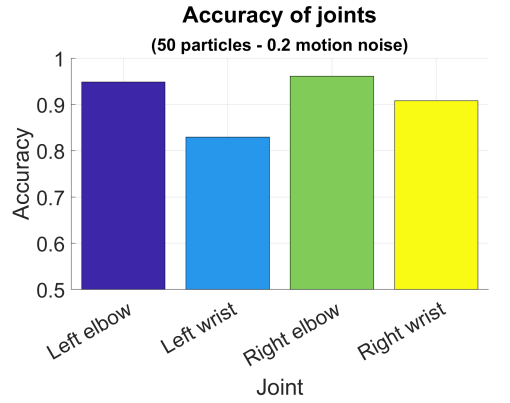


Figure 4.21: Accuracy of each joint in the model with 50 particles and 0.2 motion noise. The threshold for the accuracy is set to 10cm.

step in Section 3.2.3) as input. Notice that no changes are needed to the tracking part code because this part just takes as input any point cloud, but it is unaware of where those point clouds came from or what they represent.

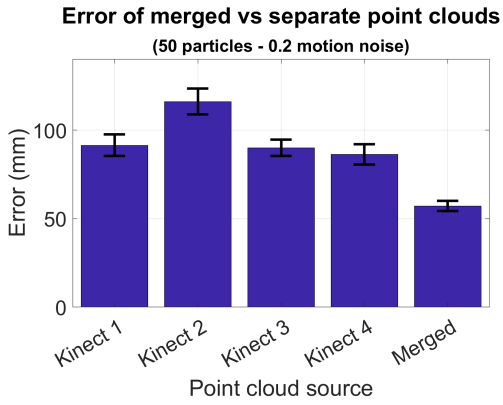


Figure 4.22: Total average error in all 30 videos and confidence intervals (of 2 standard deviations) of the best model using different point clouds as input.

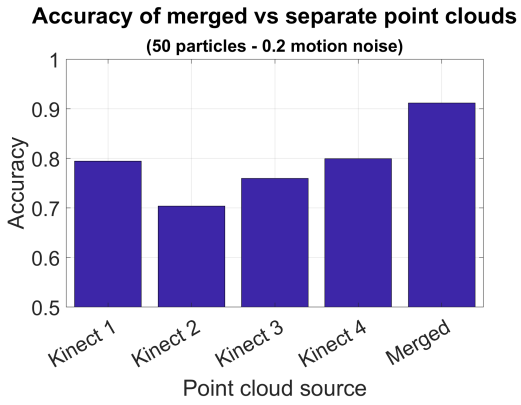


Figure 4.23: Total accuracy in all 30 videos of the best model using different point clouds as input.

Figures 4.22 and 4.23 show the total average error and the accuracy in all the 30 videos of the dataset when using as input the merged point clouds and the separate ones that come from each Kinect. The confidence intervals in Figure 4.22 are shorter than the ones in Figure 4.20, which in turn are shorter than the ones in Figure 4.17, due to the higher amount of information condensed in each bar. The bars in Figure 4.22 corresponds to all the joints in the 30 videos, thus they have lower variance than the bars in other Figures where they correspond to fewer joints. As we observe in Figures 4.22 and 4.23, the model that uses the merged point clouds is superior to the models that utilize the separate point clouds. The highest performance by a separate point cloud is achieved by the Kinect number 4 with an accuracy of 79.91%. If we compare this number with the ones in Table 4.2 (corresponding to Figure 4.14), we can see that it is almost as good to run a model with 10 particles (and 0.2 motion noise) and use the merged point cloud as input (79.69% accuracy) as running it with 50 particles and using the separate point clouds (79.91% accuracy in the best Kinect). However, if we look at Figure 4.24, we can see that the model using the point clouds from the Kinect number 4 misses 20.53% of the joints. In fact, the four models that use the separate Kinects have much higher miss detection rates than the 12 models that use the merged point clouds (as seen in Figure 4.15). Therefore, the model with 10 particles and 0.2 motion noise

that uses the merged point clouds is superior to all the models that use 50 particles, 0.2 motion noise and the separate point clouds.

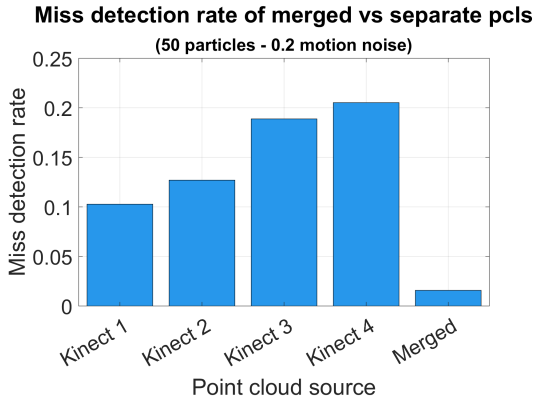


Figure 4.24: Miss detection rate in all 30 videos of the best model using different point clouds as input.

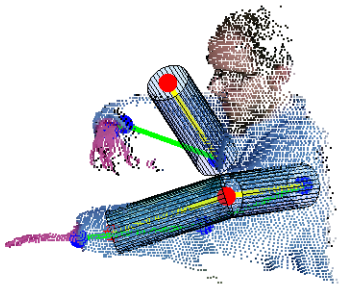


Figure 4.25: Final tracking example of a model using a separate point cloud as input. The yellow bones and red joints are the predictions and the green bones and blue joints are the ground truth.

Figure 4.25 shows a final example of a tracking model that used only the point clouds coming from Kinect number 3. Only the arms are shown for a clearer view. The yellow bones and red joints are the predictions and the green bones and blue joints are the ground truth. As we can see, the right lower arm is missed because it is almost completely occluded by the right hand as seen in Figure 4.26. Moreover, the right elbow's prediction is not precise as the right upper arm is also partially occluded. Furthermore, we can notice that the density of points in this Kinect (one of the two far ones) is very low compared to the merged point cloud in Figure 4.27, where we even are showing only 40% of the points in the point cloud for a better visualization. Hence, we can conclude that merging the four point clouds of each Kinect is a sensible approach that improves significantly the performance of the whole model and helps to deal with occlusions.

Additional graphs for the models using separate point clouds, analogous to the ones presented in Section 4.5.2 for the merged point clouds, can be seen in Appendix B, as well as supplementary final tracking examples.

Figure 4.26: Color image of the frame corresponding to the point clouds in Figures 4.25 and 4.27.

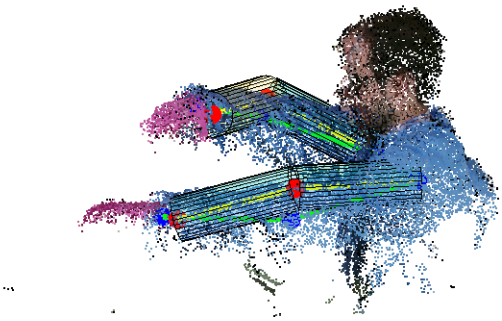


Figure 4.27: Final tracking example corresponding to the same frame as in Figure 4.25 of a model using the merged point cloud as input. The yellow bones and red joints are the predictions and the green bones and blue joints are the ground truth.

4.6 Final Results

In this section we present some final examples of the best tracking model, the one that uses 50 particles in the four particle filters and 0.2 radians as the noise in the motion models. The yellow lines and red points represent the bones and joints output by the tracking algorithm. The green lines and blue points represent the bones and joints in the ground truth, and will be shown when available. For a clearer visualization, three different views of the point clouds will be displayed for each example: one from the left, one center view from above and one from the right. Additionally, the color image from Kinect number 3 will be shown. We chose to show always the color images from Kinect number 3 for consistency and because it is the one that captures the scene without cutting any limbs more regularly.

4.6.1 Successful results

Next, we present five successful results of the tracking algorithm. The details of each example are the following:

- Figure 4.28 shows a successful result of subject 1 performing action 1.

- Figure 4.29 shows a successful result of subject 4 performing action 2.
- Figure 4.30 shows a successful result of subject 3 performing action 3.
- Figure 4.31 shows a successful result of subject 2 performing action 4.
- Figure 4.32 shows a successful result of subject 5 performing action 6.

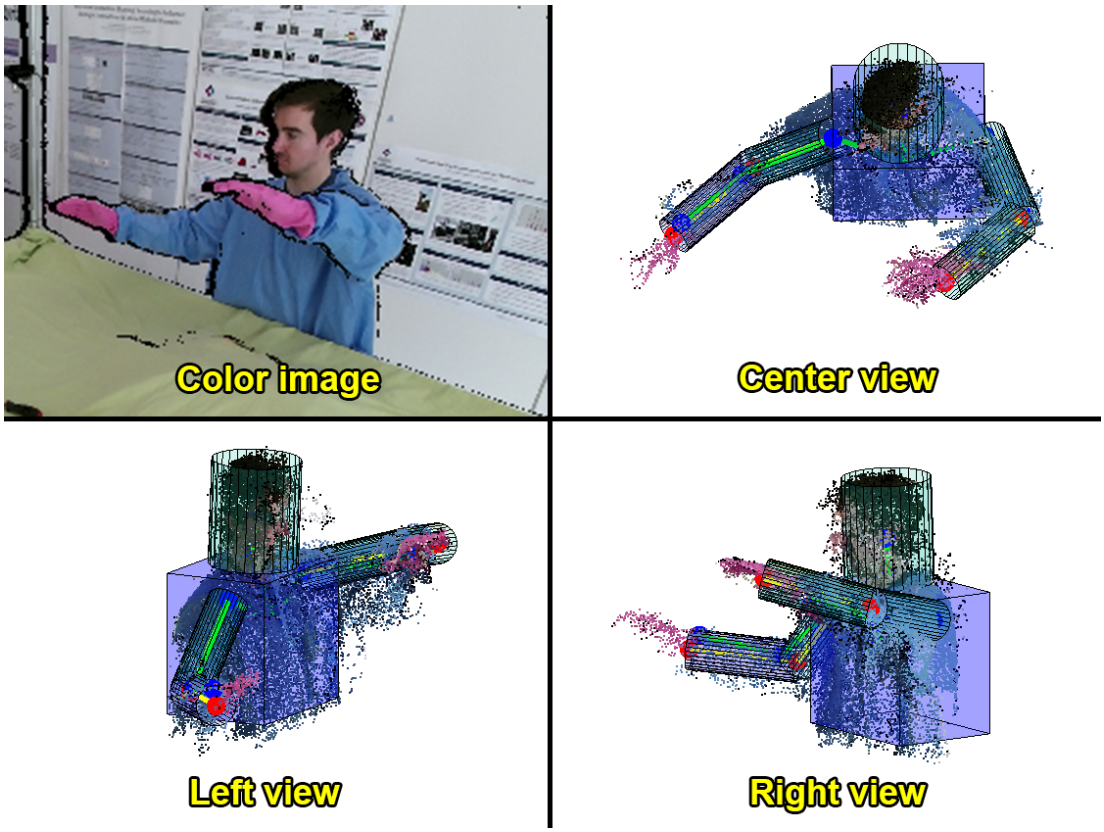


Figure 4.28: Successful final tracking example of the best model using the merged point cloud as input. Frame number 31 of subject 1 performing action 1.

We do not present here any results for action 5 (when the arms are close together) because most of the examples are either unsuccessful or not entirely successful. For the latter, there are frames in which the cylinders corresponding to the lower arms are approximately in the correct position, but the final part of one cylinder goes inside the other one; thus, we do not consider this outcome as a complete success.

4.6.2 Unsuccessful results

Here, we present three results where the tracking algorithm could not predict the position of the joints in a precise way and thus, we consider them unsuccessful results. The

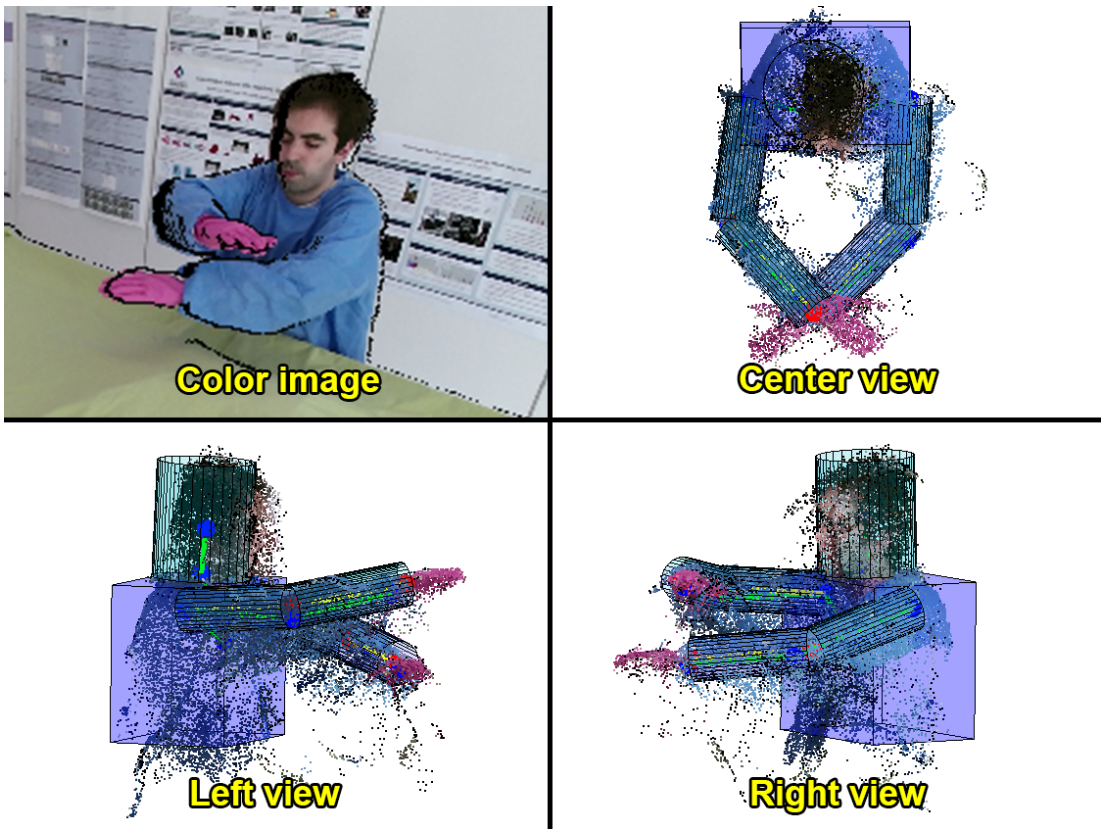


Figure 4.29: Successful final tracking example of the best model using the merged point cloud as input. Frame number 81 of subject 4 performing action 2.

details of each example are the following:

- Figure 4.33 shows an unsuccessful result of subject 5 performing action 5.
- Figure 4.34 shows an unsuccessful result of subject 2 performing action 2.
- Figure 4.35 shows an unsuccessful result of subject 1 performing action 5.

As we expected, action 5 turned out to be the most difficult one for our program to deal with. In fact, when the two arms are very close together, the algorithm gets confused between the both. As discussed in Section 3.3.3, the solution for this problem is not trivial. On the other hand, Figure 4.34 shows a failure that can be corrected much easily. This failure is caused by a kinematic restriction set too loosely; in this case, the inferior limit of the elevation most likely. Hence, in order to correct this error, we would only need to set a more restrictive limit on the elevation for the human model of this video, or for all the videos of this person in general.

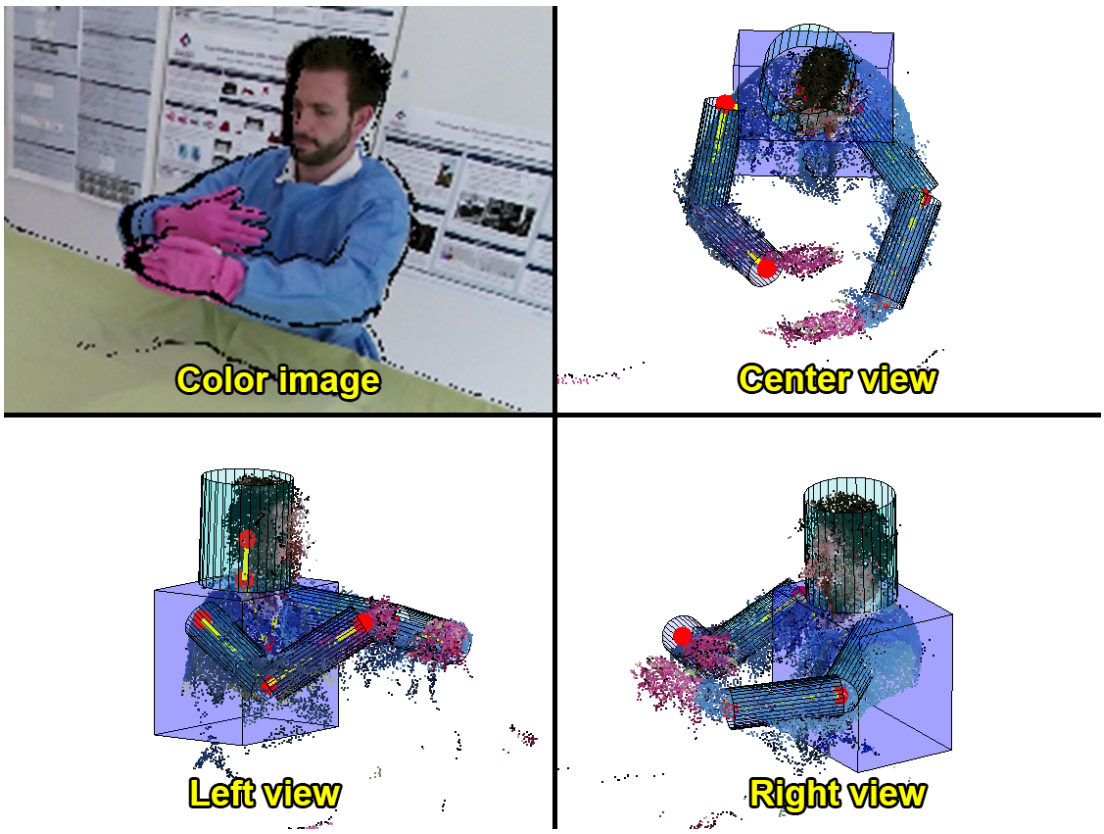


Figure 4.30: Successful final tracking example of the best model using the merged point cloud as input. Frame number 55 of subject 3 performing action 3.

4.7 Computational Time

All the computations in this project, except for the data acquisition that was carried out using a computer directly connected to the four Kinects, were performed using Matlab R2017a on a Windows machine with an Intel i7-2600 CPU and 16GB of RAM. The average time for the detection part to run on a single frame was of approximately 5 seconds and the whole detection was computed in roughly 200 minutes. For the tracking part, the approximate time to compute a single frame using 50 particles was 7 seconds. Since there are almost 2400 frames in the dataset (2398 exactly) and each frame needs to be computed by four different particle filters, the approximate time of computation for the 30 videos in the 5six dataset when using 50 particles in each filter is equal to 1120 minutes or 18 hours and 40 minutes.

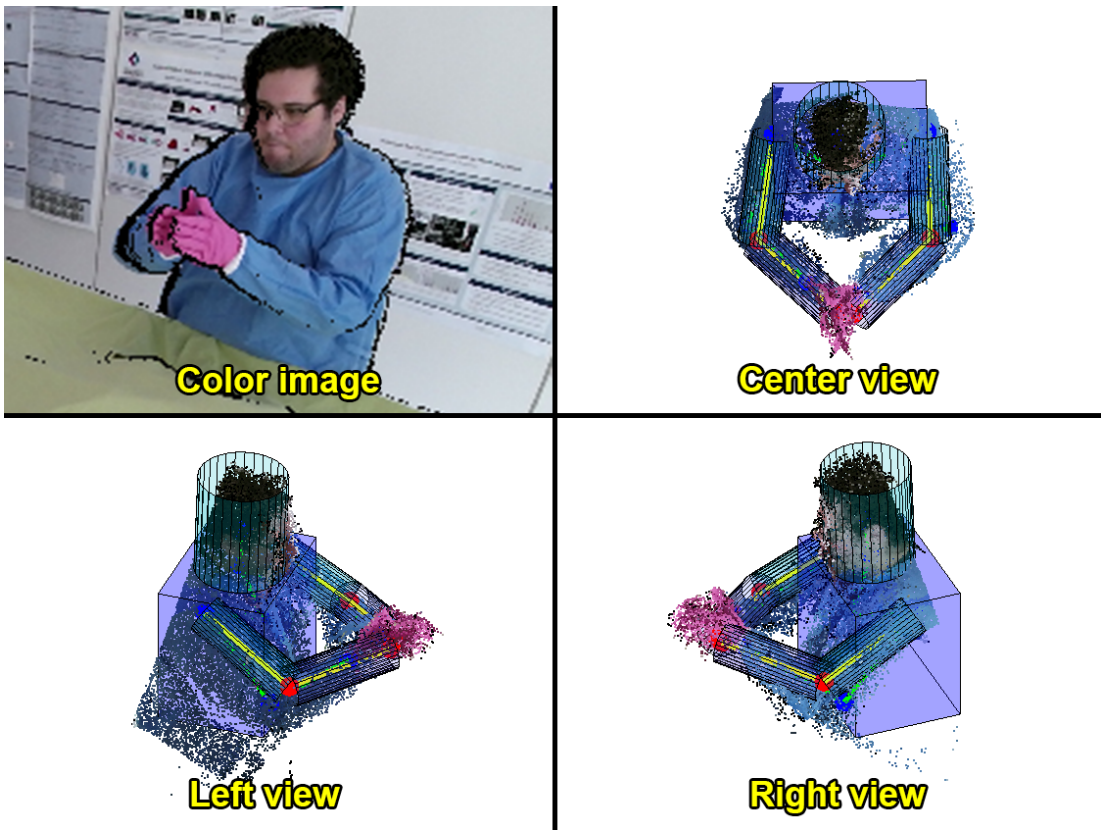


Figure 4.31: Successful final tracking example of the best model using the merged point cloud as input. Frame number 41 of subject 2 performing action 4.

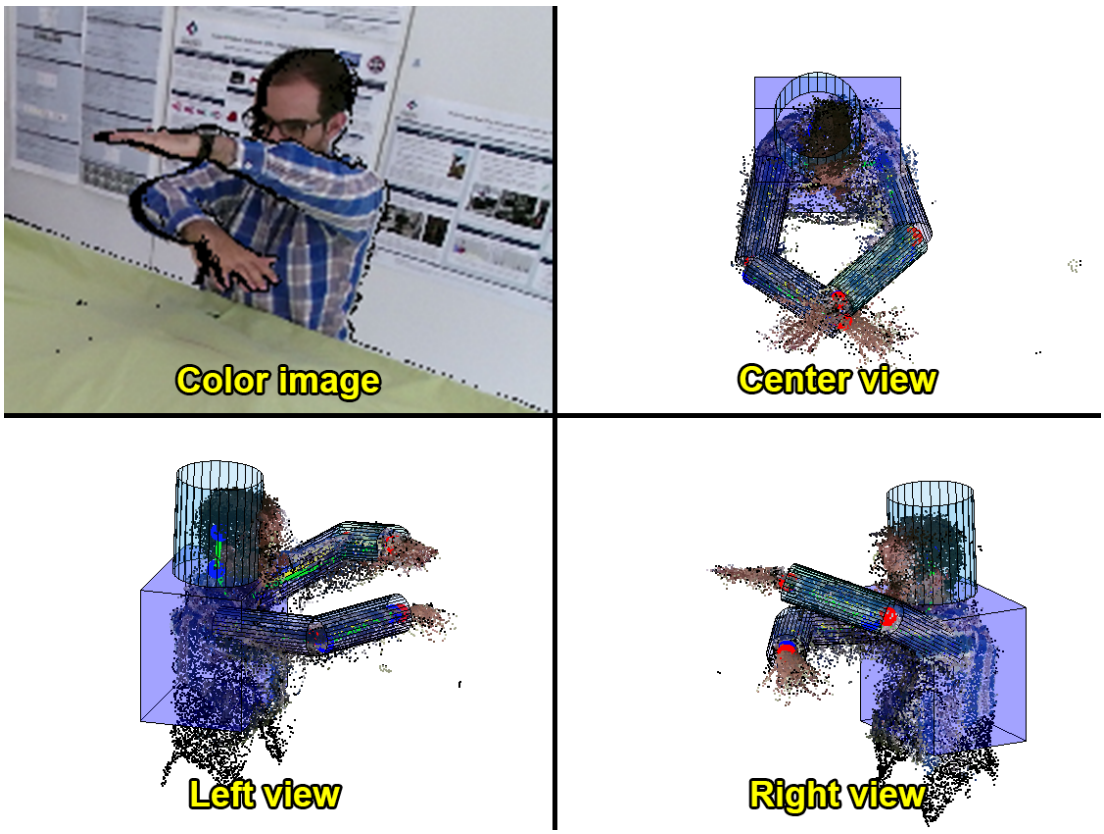


Figure 4.32: Successful final tracking example of the best model using the merged point cloud as input. Frame number 41 of subject 5 performing action 6.

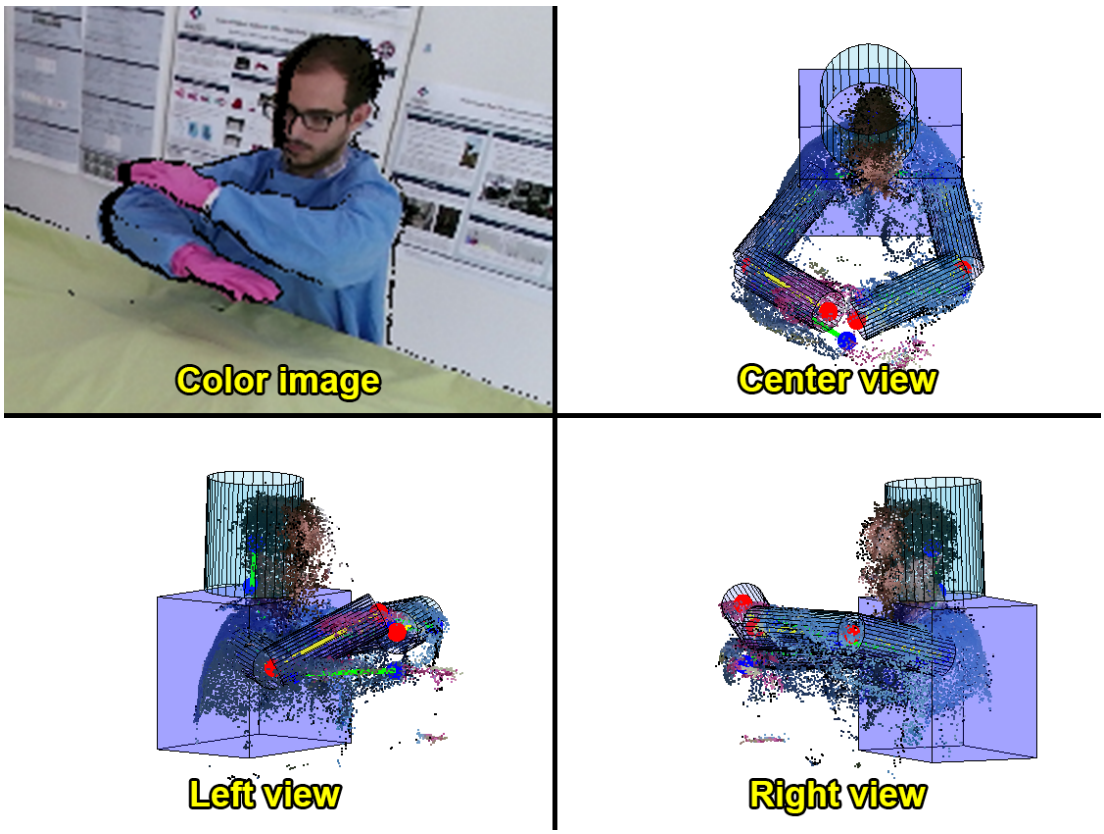


Figure 4.33: Unsuccessful final tracking example of the best model using the merged point cloud as input. Frame number 41 of subject 5 performing action 5.

Figure 4.34: Unsuccessful final tracking example of the best model using the merged point cloud as input. Frame number 51 of subject 2 performing action 2.

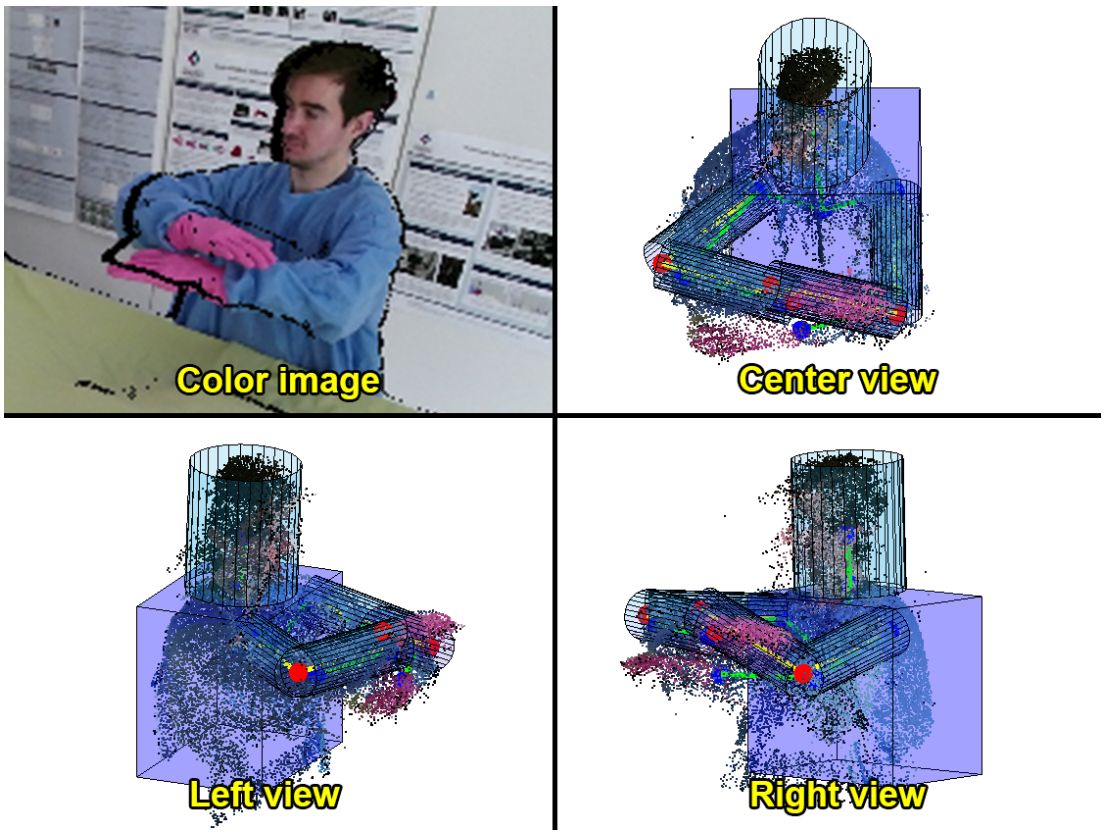


Figure 4.35: Unsuccessful final tracking example of the best model using the merged point cloud as input. Frame number 61 of subject 1 performing action 5.

Chapter 5

Conclusion

5.1 Discussion

In this thesis, we investigated the problem of articulated human body tracking using multiple RGB-D sensors in the context of surgical robotics. We built an end-to-end project that detects a person and locates his or her arms over a sequence of frames, in order for an assistant robotic arm to interact with the human in a safe manner and aid him or her in tasks such as holding or handling tools. We examined the advantages of using data coming from four time of flight RGB-D sensors (Kinect version 2) simultaneously in a cluttered and noisy environment. As a result of this project, we also contribute to the computer vision community with the new 5six dataset for human tracking on which our program obtained a 91.16% position accuracy (localization within a 10 centimeters threshold). Some final results of our program computed on this new dataset can be found at `https://youtu.be/RZFBXTjE8Ik`.

We demonstrated the power of two basic techniques, region growing and particle filtering, on depth data after the appropriate modifications were performed. Moreover, we quantitatively evaluated the improvement obtained from using multiple cameras. The use of the merged 3D information of the objects in the scene was essential to deal with the several occlusions that occurred as a result of having a non trivial environmental setup where the human is not in the center and is not isolated from the rest of the objects in the scene. Furthermore, we implemented a synchronization algorithm to deal with the latency introduced into the system by having four RGB-D sensors recording data at the same time.

The majority of the research in the human motion analysis field has been evaluated using datasets where the subjects are easily detected and where they are wearing special

tight clothes. In order to explore the potential of tracking algorithms in more realistic scenarios, we gathered a dataset of people wearing loose medical gowns or regular street clothing while standing on the limit of the scene. As a consequence, our experiments had to deal with noisy and irregular point clouds where many parts of the human body were not being captured by the sensors.

The performance obtained by our program was very close to state of the art results from the literature (Michel et al. [9]). However, it needs to be considered that the work in [9] makes use of a more advanced technique such as particle swarm optimization and evaluates the results in an easier dataset.

This thesis is one of the starting points of a larger project with the final objective of building a completely autonomous robot assistant. As such, the results obtained in this work can be of great utility to other projects framed in the same context. For example, a parallel project is being conducted with the goal of interpreting the hand gestures performed by a human. Our predicted position of the wrists using only depth data could be used to find the hands of the person even when he or she is not wearing any gloves or the gloves do not have an easily recognizable color. In the same way, reducing the search space for the hands (since they are attached to the wrists) could help find any tools that the person is holding or that are at least in the proximity of the hands.

We believe that in the coming years the interest on human tracking will continue growing thanks to the increasing quality and resolution of affordable 3D sensors, and also due to the high impact that positive results could have on our daily lives. The advent of autonomous robot assistants is not limited to the operating room: many other areas could benefit from these technologies such as elderly or infant care, home assistance or any form of recreational activity such as cooking or painting. For this kind of automation to become reality, it is necessary to have algorithms with high levels of accuracy in realistic scenarios in order for a human to interact in a safe and useful way with an autonomous robot. We consider that current state of the art algorithms are close to those levels but not enough for robots to start interacting with people without human supervision. Nevertheless, we are confident that the continuous scientific research and the appearance of more algorithms and datasets will bring autonomous robot assistants to reality in a not too distant future.

5.2 Future Work

Many are the improvements that could be made to the work in this project and that were not investigated or implemented due to constraints in time or resources. To start with, a more thorough and quantitative comparison with work in the literature could be performed. Of special interest would be to examine the performance of state of the art algorithms (such as the one in [9]) on our 5six dataset and see if they can deal with the huge amount of noise and holes in the 3D data.

The algorithm that we designed and implemented was tested offline. However, some of the parts could be parallelized in order to speed the process and run the program in real time. Perhaps the more clear part to parallelize is the one with the particle filters. Also, each arm could be run simultaneously in parallel because they do not share any information.

We believe that in order to produce realistic algorithms the human interaction should be minimized or removed completely from the execution of the tracking program. Since we followed a top-down approach, we had to interact with the program in order to define an initial pose for the particle filter to start working. Hence, it would be advisable to use an hybrid methodology between bottom-up and top-down (such as the one in [9]). In this way, we would have a semantic classification of the different parts of the body that could be used as a starting or recovery point for the particle filters or any analogous tracking algorithm. Nonetheless, the drawback of this approach is the need for a huge dataset and for a large amount of computational time, that most certainly will decrease as technology evolves.

Bibliography

[1] Antonio Verdone. Informatics research proposal. *University of Edinburgh*, 2017.

[2] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.

[3] Thomas B Moeslund, Adrian Hilton, and Volker Krüger. A survey of advances in vision-based human motion capture and analysis. *Computer vision and image understanding*, 104(2):90–126, 2006.

[4] Ronald Poppe. Vision-based human motion analysis: An overview. *Computer vision and image understanding*, 108(1):4–18, 2007.

[5] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002.

[6] Rishi Nayyar, Siddharth Yadav, Prabhjot Singh, and Prem Nath Dogra. Impact of assistant surgeon on outcomes in robotic surgery. *Indian journal of urology: IJU: journal of the Urological Society of India*, 32(3):204, 2016.

[7] Lulu Chen, Hong Wei, and James Ferryman. A survey of human motion analysis using depth imagery. *Pattern Recognition Letters*, 34(15):1995–2006, 2013.

[8] Shuran Song and Jianxiong Xiao. Tracking revisited using rgbd camera: Unified benchmark and baselines. In *Proceedings of the IEEE international conference on computer vision*, pages 233–240, 2013.

[9] Damien Michel, Costas Panagiotakis, and Antonis A Argyros. Tracking the articulated motion of the human body with two rgbd cameras. *Machine Vision and Applications*, 26(1):41–54, 2015.

[10] Licong Zhang, Jürgen Sturm, Daniel Cremers, and Dongheui Lee. Real-time human motion tracking using multiple depth cameras. In *Intelligent Robots and*

Systems (IROS), 2012 IEEE/RSJ International Conference on, pages 2389–2395. IEEE, 2012.

[11] Zhiquan Gao, Yao Yu, Yu Zhou, and Sidan Du. Leveraging two kinect sensors for accurate full-body motion capture. *Sensors*, 15(9):24297–24317, 2015.

[12] Leonid Sigal, Alexandru O Balan, and Michael J Black. Humaneva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion. *International journal of computer vision*, 87(1):4–27, 2010.

[13] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Online object tracking: A benchmark. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2411–2418, 2013.

[14] MICROSOFT. Kinect for windows v2. *Available: https://developer.microsoft.com/en-us/windows/kinect*, July 2014.

[15] Asus. Asus xtion. *Available: https://www.asus.com/3D-Sensor/Xtion*, 2012.

[16] Bob Fisher. Advanced vision course, slides on "sources of range data". 2017.

[17] Motion Capture Systems. Vicon. *https://www.vicon.com*, 2012.

[18] Christian Plagemann, Varun Ganapathi, Daphne Koller, and Sebastian Thrun. Real-time identification and localization of body parts from depth images. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 3108–3113. IEEE, 2010.

[19] Jamie Shotton, Toby Sharp, Alex Kipman, Andrew Fitzgibbon, Mark Finocchio, Andrew Blake, Mat Cook, and Richard Moore. Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 56(1):116–124, 2013.

[20] Tim Beyl, Philip Nicolai, Mirko D Comparetti, Jörg Raczkowsky, Elena De Momi, and Heinz Wörn. Time-of-flight-assisted kinect camera-based people detection for intuitive human robot cooperation in the surgical operating room. *International journal of computer assisted radiology and surgery*, 11(7):1329–1345, 2016.

[21] Bob Fisher. Advanced vision course, "system 3, detecting and tracking objects in video". 2017.

[22] Varun Ganapathi, Christian Plagemann, Daphne Koller, and Sebastian Thrun. Real time motion capture using a single time-of-flight camera. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 755–762. IEEE, 2010.

[23] Abdolrahim Kadkhodamohammadi, Afshin Gangi, Michel de Mathelin, and Nicolas Padoy. Articulated clinician detection using 3d pictorial structures on rgb-d data. *Medical image analysis*, 35:215–224, 2017.

[24] Martin A Fischler and Robert A Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on computers*, 100(1):67–92, 1973.

[25] Pedro F Felzenszwalb and Daniel P Huttenlocher. Pictorial structures for object recognition. *International journal of computer vision*, 61(1):55–79, 2005.

[26] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. 2009.

[27] Chris Solomon and Toby Breckon. *Fundamentals of Digital Image Processing: A practical approach with examples in Matlab*. John Wiley & Sons, 2011.

[28] M Isard and A Blake. Condensation–conditional density propagation for visual tracking,(1998). *International Journal of Computer Vision Publ*, pages 5–28, 1998.

[29] Sanjay Saini, Nordin Zakaria, Dayang Rohaya Awang Rambli, and Suziah Sulaiman. Markerless human motion tracking using hierarchical multi-swarm cooperative particle swarm optimization. *PloS one*, 10(5):e0127833, 2015.

[30] Diane Haering, Maxime Raison, and Mickael Begon. Measurement and description of three-dimensional shoulder range of motion with degrees of freedom interactions. *Journal of biomechanical engineering*, 136(8):084502, 2014.

[31] Chris Snyder, Thomas Bengtsson, Peter Bickel, and Jeff Anderson. Obstacles to high-dimensional particle filtering. *Monthly Weather Review*, 136(12):4629–4640, 2008.

[32] John MacCormick and Michael Isard. Partitioned sampling, articulated objects, and interface-quality hand tracking. In *European Conference on Computer Vision*, pages 3–19. Springer, 2000.

[33] David Forsyth and Jean Ponce. *Computer vision: a modern approach*. Upper Saddle River, NJ; London: Prentice Hall, 2011.

[34] GA Einicke. Smoothing, filtering and prediction: Estimating the past, present and future. rijeka, croatia: Intech. Technical report, ISBN 978-953-307-752-9, 2012.

[35] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[36] M Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on signal processing*, 50(2):174–188, 2002.

[37] E Roy Davies. *Computer and machine vision: theory, algorithms, practicalities*. Academic Press, 2012.

[38] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

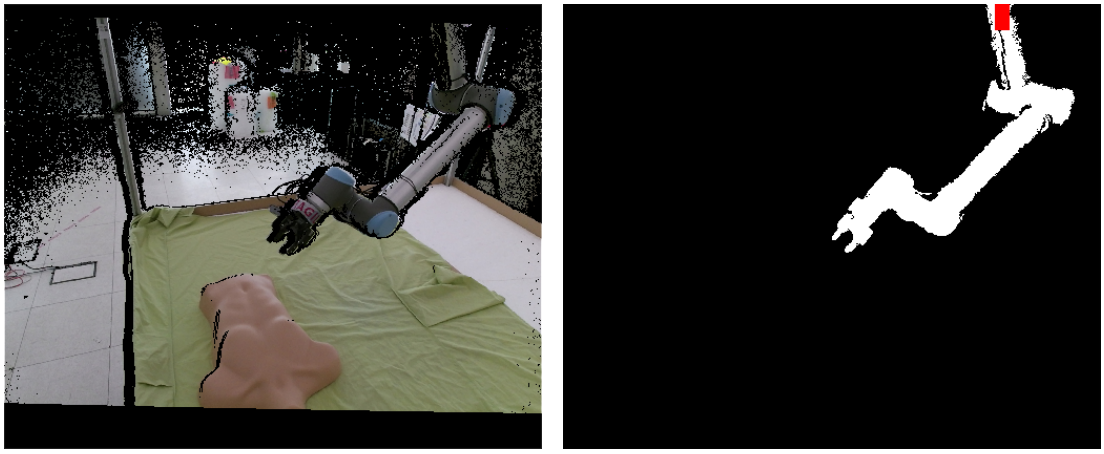
Appendix A

Detecting the robot arm using the region growing algorithm

The region growing algorithm implemented in this project is generalizable enough to detect many objects of interest in the scene. One of these objects could be the robot arm. In order to detect the arm, we will use the exact same region growing algorithm that we used to detect the person. The only difference is that, in the case of the robot, we cannot use the color to identify the initial pixels in the region growing algorithm. However, for the robot arm this problem can be overcome in a simple way by manually specifying the initial seed pixels from where the algorithm will start growing. Since the robot is ceiling mounted, its base will remain in the same position at all times. Thus, we only need to manually define the region in the depth images from where the starting seeds will be selected. This initialization process is done only once because the base of the robot does not move. In our case, some cameras do not capture exactly the base of the robot because they are pointing towards the person, but for the sake of this example, we will consider that they see the base, even though we will be choosing the seed pixels from a part of the robot slightly below the actual base. Once we have the seed pixels selected, we run the algorithm in the same way that we did for the person. The difference is that instead of having seed pixels that come from the blue gown or the pink gloves, we have them selected manually from the base of the robot.

Figure A.1 shows a color image of the robot arm in the scene and the corresponding binary mask of the robot after computing the region growing algorithm on it. The red rectangle in the binary mask is the manually chosen area from where the seed pixels were selected. Figure A.2 shows views from the four Kinects of the final color masks resulting from segmenting the robot arm. Finally, Figure A.3 shows two different views

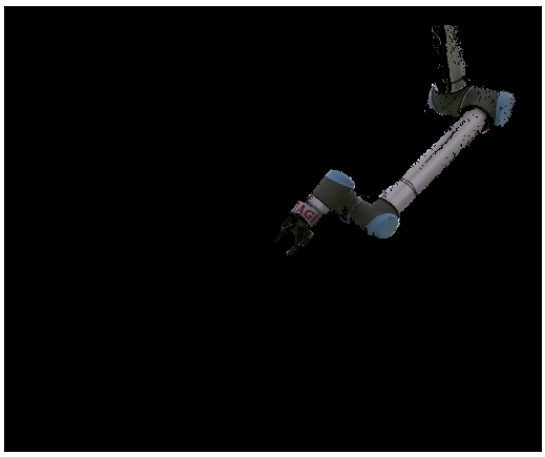
of the same point cloud resulting from merging the four point clouds coming from each Kinect. As we can see, the calibration of the Kinects seems a bit off. This is caused by the fact that the calibration was performed focusing on the area of the scene where the person would be standing. For future work, the calibration should be corrected.



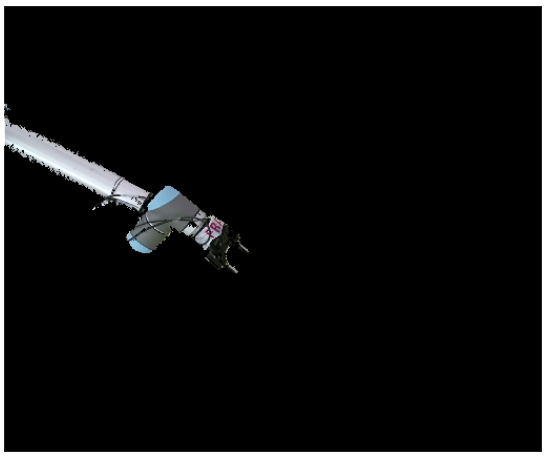
(a) Color image.

(b) Binary mask of the robot arm.

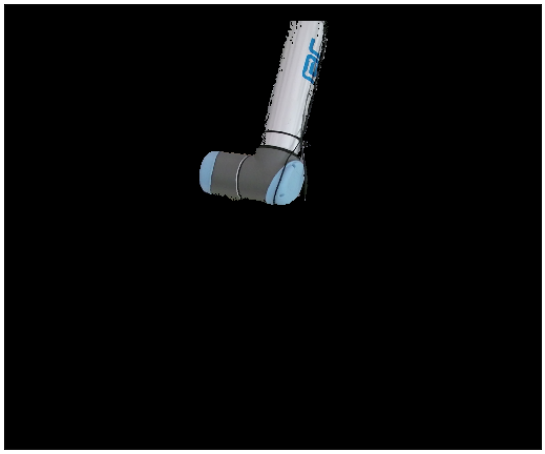
Figure A.1: Color image of a frame with the robot arm in the scene viewed from Kinect number 1 and the corresponding binary mask of the robot arm. The red rectangle in the binary mask corresponds to the seed pixels.



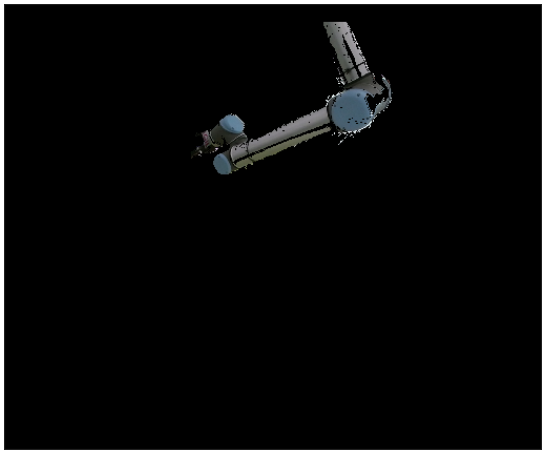
(a) View from Kinect number 1.



(b) View from Kinect number 2.



(c) View from Kinect number 3.



(d) View from Kinect number 4.

Figure A.2: Views from the four Kinects of the final color masks resulting from segmenting the robot arm.

(a) View from one side.

(b) View from the other side.

Figure A.3: Two views of the final merged point cloud of the robot arm.

Appendix B

Merged vs separate point clouds

B.1 Additional graphs

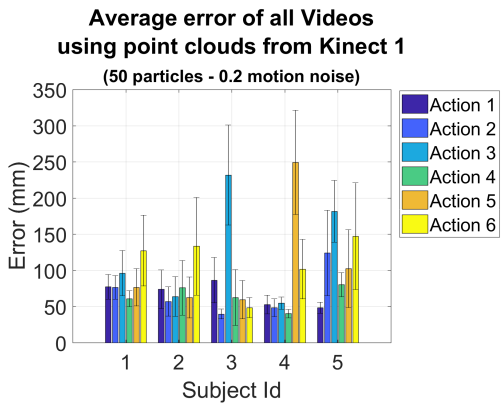


Figure B.1: Average error and confidence intervals (of 2 standard deviations) of every video in the dataset of the best model with 50 particles and 0.2 radians as motion noise, using the separate point clouds coming Kinect number 1.

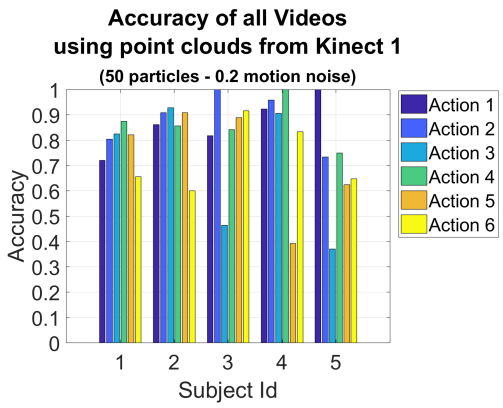


Figure B.2: Accuracy with a 10cm threshold of every video in the dataset of the best model with 50 particles and 0.2 radians as motion noise, using the separate point clouds coming Kinect number 1.

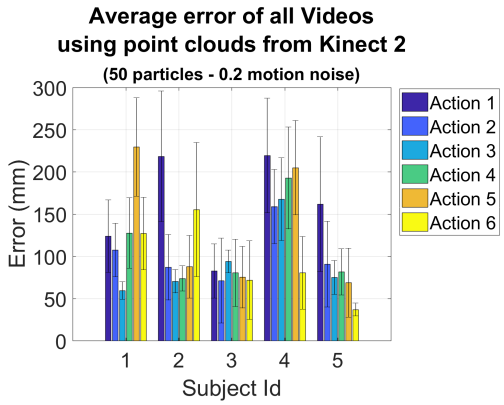


Figure B.3: Average error and confidence intervals (of 2 standard deviations) of every video in the dataset of the best model with 50 particles and 0.2 radians as motion noise, using the separate point clouds coming Kinect number 2.

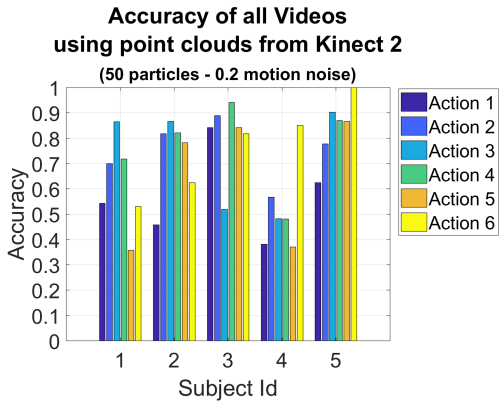


Figure B.4: Accuracy with a 10cm threshold of every video in the dataset of the best model with 50 particles and 0.2 radians as motion noise, using the separate point clouds coming Kinect number 2.

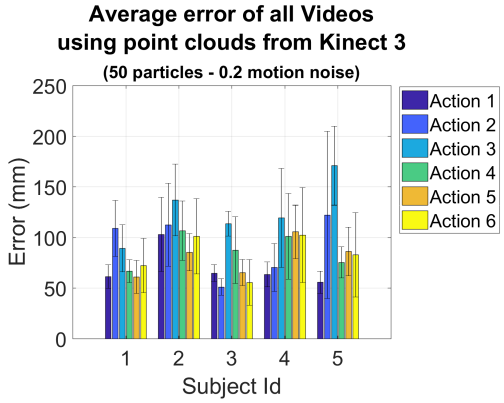


Figure B.5: Average error and confidence intervals (of 2 standard deviations) of every video in the dataset of the best model with 50 particles and 0.2 radians as motion noise, using the separate point clouds coming Kinect number 3.

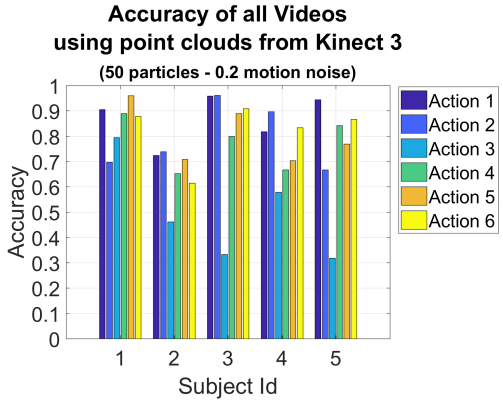


Figure B.6: Accuracy with a 10cm threshold of every video in the dataset of the best model with 50 particles and 0.2 radians as motion noise, using the separate point clouds coming Kinect number 3.

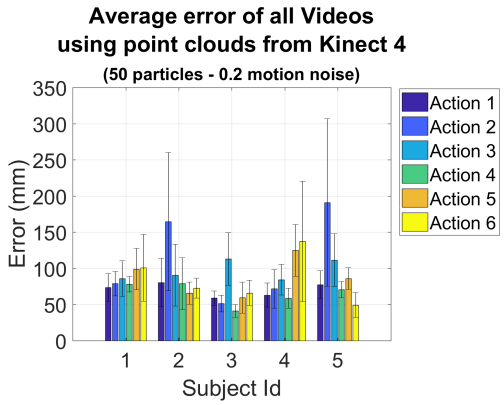


Figure B.7: Average error and confidence intervals (of 2 standard deviations) of every video in the dataset of the best model with 50 particles and 0.2 radians as motion noise, using the separate point clouds coming Kinect number 4.

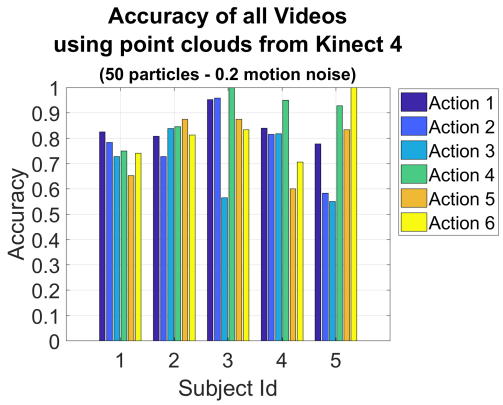


Figure B.8: Accuracy with a 10cm threshold of every video in the dataset of the best model with 50 particles and 0.2 radians as motion noise, using the separate point clouds coming Kinect number 4.

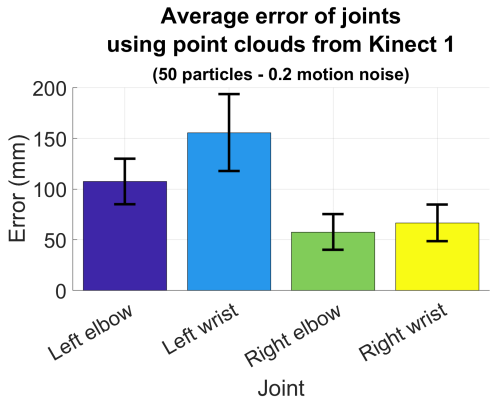


Figure B.9: Average error and confidence intervals (of 2 standard deviations) of each joint of the best model with 50 particles and 0.2 radians as motion noise, using the separate point clouds coming Kinect number 1.

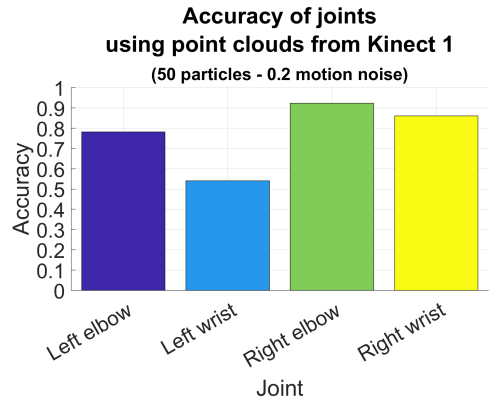


Figure B.10: Accuracy with a 10cm threshold of each joint of the best model with 50 particles and 0.2 radians as motion noise, using the separate point clouds coming Kinect number 1.

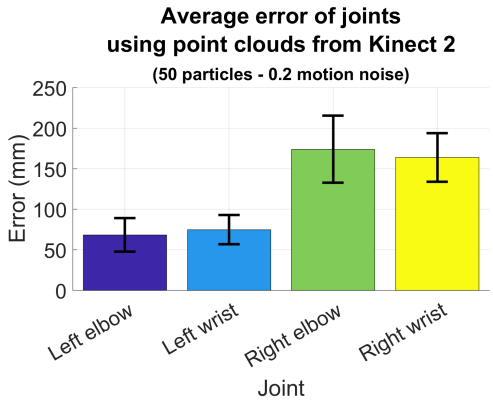


Figure B.11: Average error and confidence intervals (of 2 standard deviations) of each joint of the best model with 50 particles and 0.2 radians as motion noise, using the separate point clouds coming Kinect number 2.

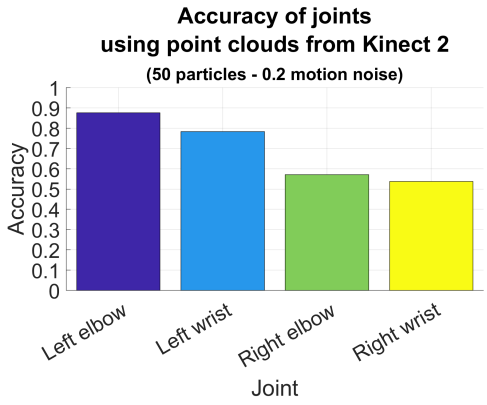


Figure B.12: Accuracy with a 10cm threshold of each joint of the best model with 50 particles and 0.2 radians as motion noise, using the separate point clouds coming Kinect number 2.

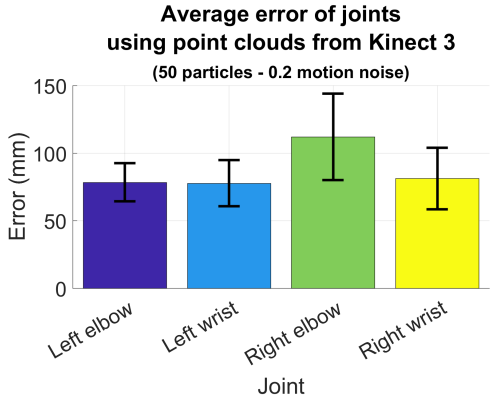


Figure B.13: Average error and confidence intervals (of 2 standard deviations) of each joint of the best model with 50 particles and 0.2 radians as motion noise, using the separate point clouds coming Kinect number 3.

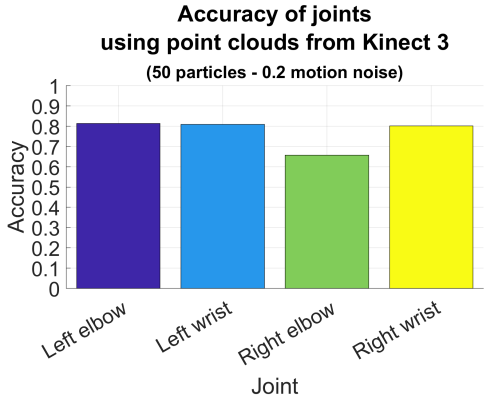


Figure B.14: Accuracy with a 10cm threshold of each joint of the best model with 50 particles and 0.2 radians as motion noise, using the separate point clouds coming Kinect number 3.

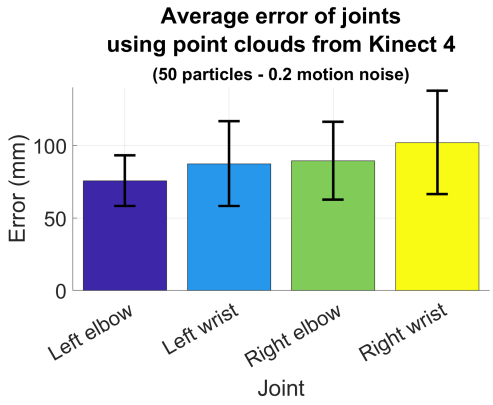


Figure B.15: Average error and confidence intervals (of 2 standard deviations) of each joint of the best model with 50 particles and 0.2 radians as motion noise, using the separate point clouds coming Kinect number 4.

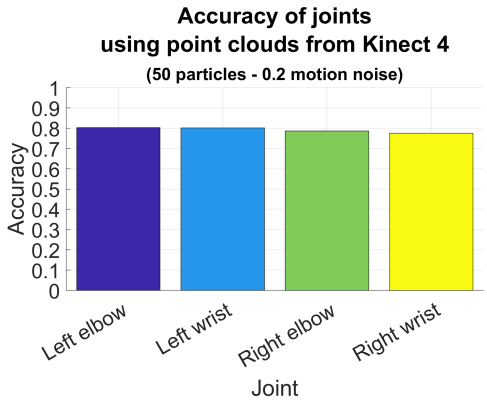


Figure B.16: Accuracy with a 10cm threshold of each joint of the best model with 50 particles and 0.2 radians as motion noise, using the separate point clouds coming Kinect number 4.

B.2 Additional tracking examples

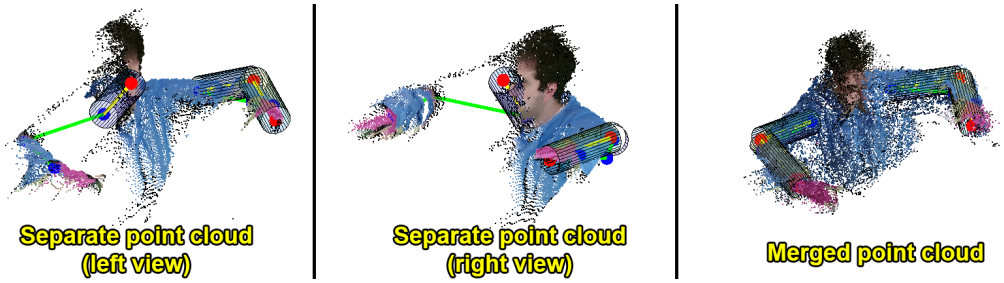


Figure B.17: Final tracking example of the best model using a separate point cloud (left and center) and a merged one (right) as input. Frame number 11 of subject 4 performing action 2. The separate point clouds come from Kinect number 2.

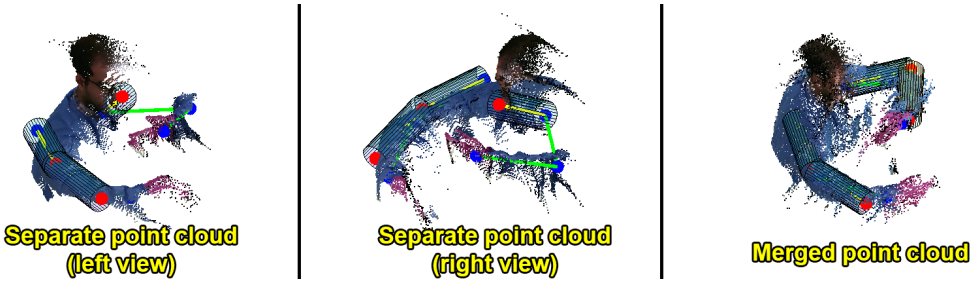


Figure B.18: Final tracking example of the best model using a separate point cloud (left and center) and a merged one (right) as input. Frame number 21 of subject 5 performing action 3. The separate point clouds come from Kinect number 1.

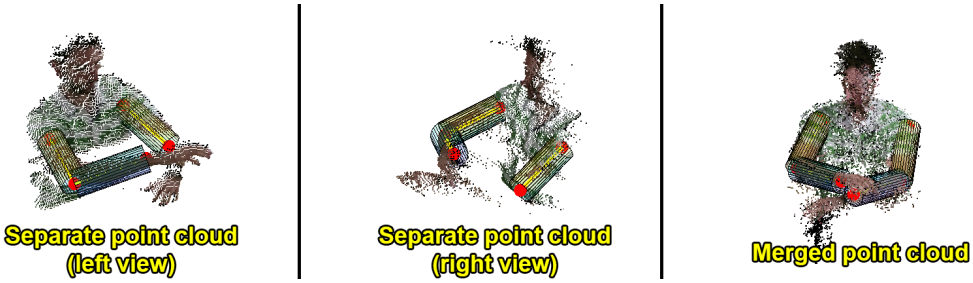


Figure B.19: Final tracking example of the best model using a separate point cloud (left and center) and a merged one (right) as input. Frame number 35 of subject 3 performing action 6. The separate point clouds come from Kinect number 4.