Technical Report

IRI-TR-16-05

Autor:
Miguel Arduengo

Supervisors:
Guillem Alenyà
Francesc Moreno

July, 2016

Institut de Robòtica i Informàtica Industrial

Institut de Robòtica i Informàtica Industrial (IRI)
Consejo Superior de Investigaciones Científicas (CSIC)
Universitat Politècnica de Catalunya (UPC)
Llorens i Artigas 4-6, 08028, Barcelona, Spain

Tel (fax): +34 93 401 5750 (5751)
`http://www.iri.upc.edu`

Corresponding author:

G. Alenyà
tel: +34 93 405 4490
`galenya@iri.upc.edu`
`http://www.iri.upc.edu/staff/galenya`

INDEX

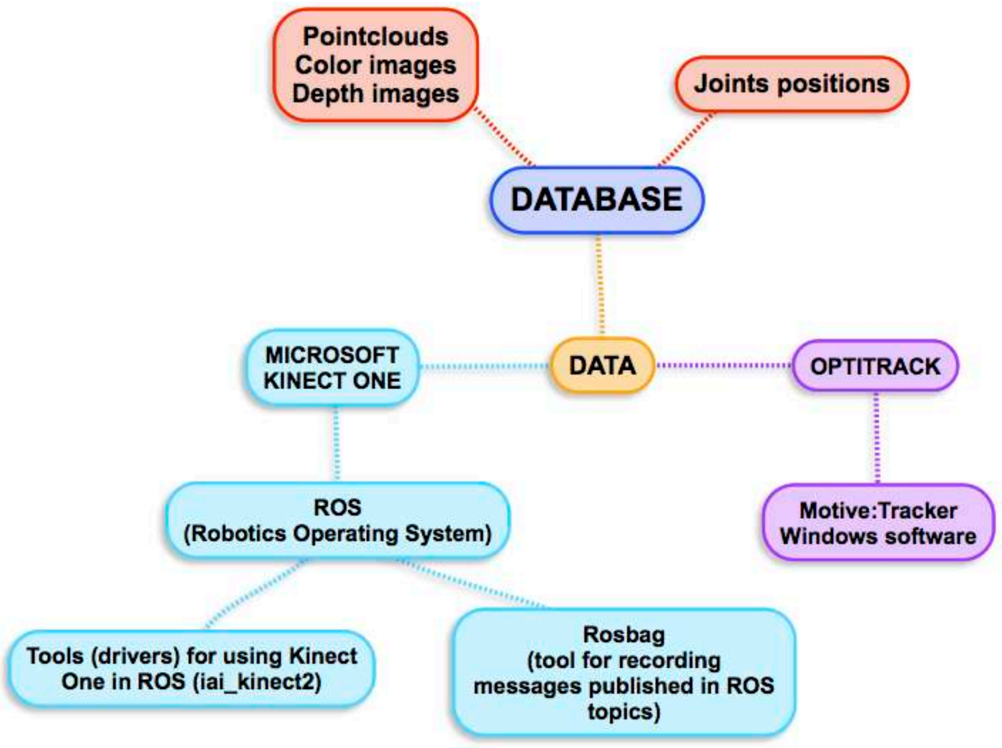

Pointclouds
Color images
Depth images
Joints positions
DATABASE
MICROSOFT KINECT ONE
DATA
OPTITRACK
ROS (Robotics Operating System)
Motive:Tracker Windows software
Tools (drivers) for using Kinect One in ROS (iai_kinect2)
Rosbag (tool for recording messages published in ROS topics)

Fig.0: Relations between the different elements of hardware and software in this work.

1.- Introduction.

This work is part of the project I-DRESS (Assistive interactive robotic system for support in dressing). I-DRESS is a project conducted by a Consortium consisting of: Institut de Robòtica i Informàtica Industrial, CSIC-UPC (Coordinator); Bristol Robotics Lab, University of West of England; and IDIAP Research Institute. The main aim of this project is "to develop a system that will provide proactive assistance with dressing to disabled users or users such as high-risk health-care workers, whose physical contact with the garments must be limited to avoid contamination" **[2]**. The system to be developed will consist of three main components, each of which has a significant impact in the field of robot-assisted services: "(a) intelligent algorithms for user and garment detection and tracking, specifically designed for close and physical human-robot interaction; (b) cognitive functions based on the multi-modal user input, environment modelling and safety, allowing the robot to decide when and how to assist the user; and (c) advanced user interface that facilitates intuitive and safe physical and cognitive interaction for support in dressing" **[8].**

The part of the project I-DRESS, within the section (Work Packages) called WP2 "Perception and Tracking", must be carried out by IRI **[2].** The specific objective is the detection of human body postures and the tracking of their movements. To this end, this work aims to create the image database needed for the training of the algorithms of pose estimation for the artificial vision of the robotic system, based on the depth images obtained by a sensor Time-of-Flight (ToF) depth camera type, such as the incorporated by the Kinect One (Kinect v2) device.

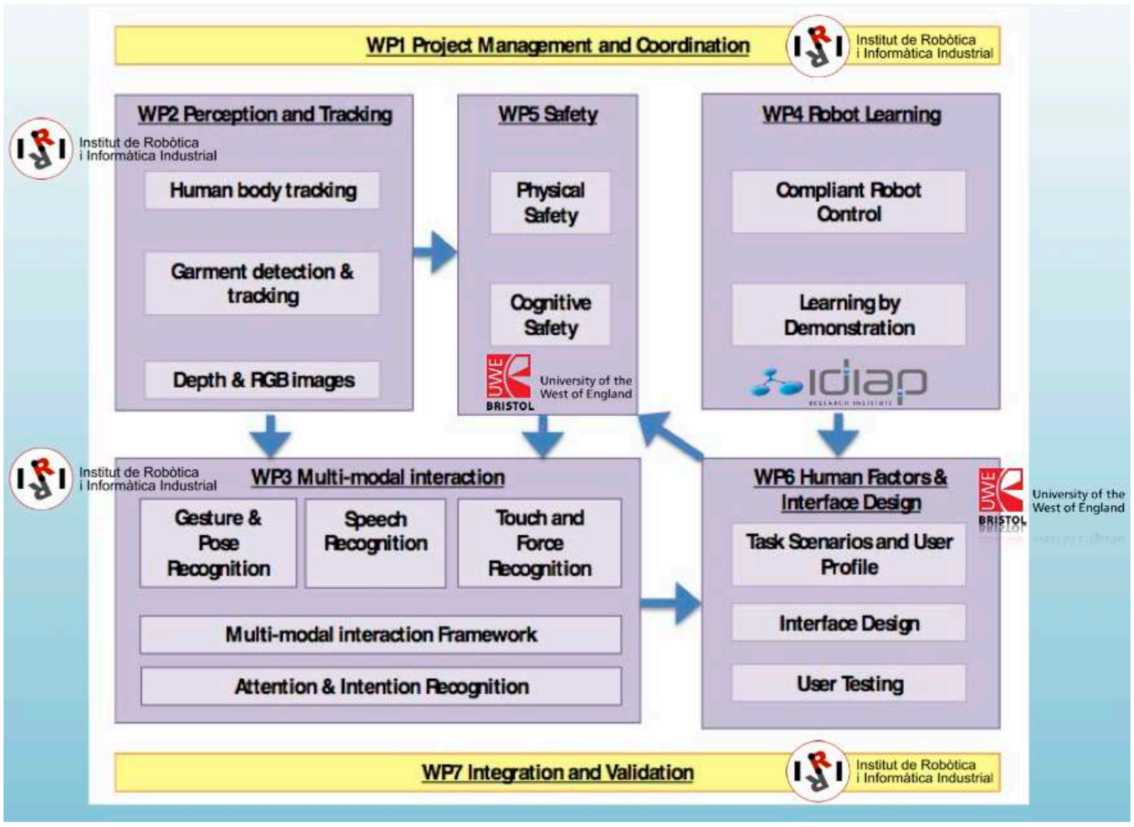

WP1 Project Management and Coordination
Institut de Robòtica i Informàtica Industrial
WP2 Perception and Tracking
Institut de Robòtica i Informàtica Industrial
Human body tracking
Garment detection & tracking
Depth & RGB images
WP5 Safety
Physical Safety
Cognitive Safety
University of the West of England
WP4 Robot Learning
Compliant Robot Control
Learning by Demonstration
idiap
WP3 Multi-modal interaction
Institut de Robòtica i Informàtica Industrial
Gesture & Pose Recognition
Speech Recognition
Touch and Force Recognition
Multi-modal interaction Framework
Attention & Intention Recognition
WP6 Human Factors & Interface Design
University of the West of England
Task Scenarios and User Profile
Interface Design
User Testing
WP7 Integration and Validation
Institut de Robòtica i Informàtica Industrial

Fig.1: I-DRESS – Work Packages Description [2]

The human pose estimation consists in identifying a human pose in an image scene using the human pose model that is given as input. The pose recognition should be possible in conditions where the human body has moved or when different points of view are considered. The first step for doing human pose estimation consists in finding features in the image. A feature is defined as a piece of information deriving from a specific point of an image. The set of features of the global image defines the image itself. The feature descriptors are the algorithms that extract this information from the image. Depending on the feature descriptor used, the information is extracted in a different way. Then the information extracted is saved into vectors called descriptors **[1, 7, 15].**

There are several methods for human pose estimation in a 2D dimension space where the information is extracted taking into account the pixel color information (RGB). But, since the development of the 3D dimension techniques, we can add a new data to the recognition in order to get better results, because we can know the distance from the human body to the robot without the need of elaborating data to get this new information, like some human pose estimation methods do. This is essential if the robot needs, for example, to pick up clothings. With the additional depth information, we no longer need to worry about the scale ambiguity, which is always a problem with single color images. The device used to acquire scenes with the depth information is the Kinect One, which is a video-game tool integrated with the Microsoft XBOX One. With this tool it is possible to capture 3D images with depth. This last parameter is the most important information we need in order to use a 3D model in a human pose descriptor **[22].**

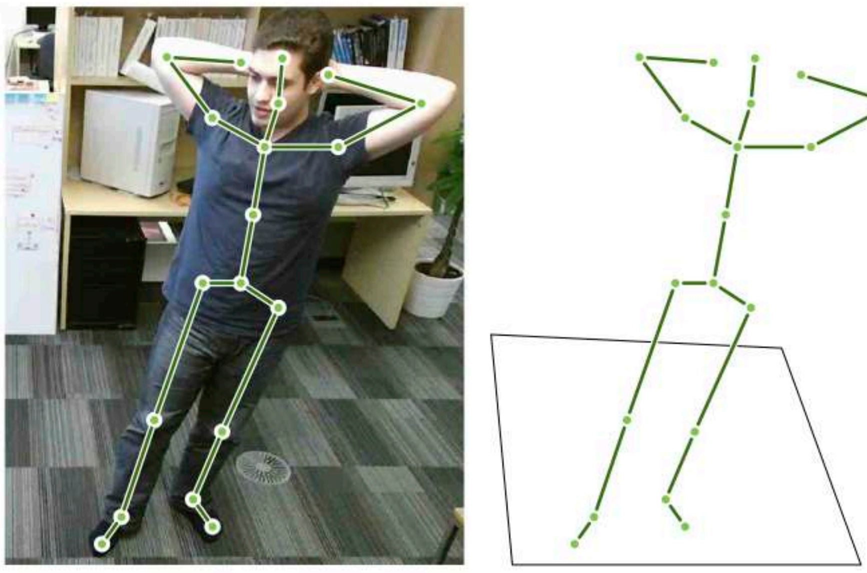


Fig.2: The goal of pose estimation is to learn to represent the postural information of the left image abstractly as shown in the right image [22].

This paper is structured as follows: the first part (sections 2 to 5) describes the tools, both hardware (Kinect One and Optitrack) and software (ROS, Rviz, Tf and PCL) used. The second part (sections 6 to 9) deals with the practical realization of the work (sequence and image recording). This database will be processed with comparison algorithms, classification and learning in a posterior phase.

2.- ROS (Robot Operating System)

3D perception is one of the most important parts in the robotics field and nowadays there is availability of new hardware that researchers can use. This creates an increasingly need for software in order to work with this new hardware.

The Robot Operating System (ROS) [http://wiki.ros.org] **[17]** is open-source software that tries to follow hardware's footsteps being daily updated in most of its libraries. It can be installed on a Robot and contains the drivers that allow the connection between the computer and the Kinect One. It also includes the tools that were used in this work.

ROS provides standard operating system services such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes and package management. Running sets of ROS-based processes are represented in a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor, control, state, planning, actuator and other messages.

The main ROS client libraries (C++, Phyton, LISP) are geared toward a UNIX-like system, primarily because of their dependence on large collections of open-source software. For these client libraries, Ubuntu Linux is listed as "supported", while other variants such as Fedora Linux, Mac OS X, and Microsoft Windows are designated "experimental".

ROS is released under the terms of the BSD license, and as such is free for both commercial and research use **[9, 10, 13, 16]**.

2.1.- ROS Structure

ROS programs have a modular structure, and each application or program (called node by ROS) runs within another executable program that works as core system (roscore) and it serves as a platform by which the different nodes communicate by topics or services.

The file system in ROS is divided into two levels:

(a) Packages, which are the lowest level of organization of the file system and can contain both executable (nodes) and messages, services, tools or libraries; and
(b) Stacks, which are groups of packages that form a high-level library. Each stack groups several packages that are complementary.

Both packages and stacks incorporate information files that specify the content, functionality and dependencies. They are called information files stack.xml in the stacks and manifest.xml in the packages.

For download, the stacks are grouped into repositories, which are equivalent to Linux repositories. In fact, ROS repositories can be downloaded directly from the command line, just like in Linux.

CMake is used to compile a package in ROS, an open source platform for generating, compiling and testing software packages. Its use is quite simple because the ROS package contains a file called CMakeList.txt composed of a series of macros that allow creating and compiling what you want.

A node is a module or individual process within the ROS system that performs a computation, can send and receive information from other nodes and use and offer services. Thus, these nodes are simply executable programs from the ROS packages.

The nodes can be written with C++ or Phyton using the corresponding library offered by ROS that are also ROS packages:

(a) roscpp: client library of C++; and
(b) rospy: client library of Phyton.

The running and communication between nodes is independent of the language they are written in. Nodes communicate between them publishing and receiving messages.

A node that is interested in a certain type of data subscribes to the corresponding topic, obtaining information from another node that published the item. There may be several concurrent publishers and subscribers to the same topic, and a single node can publish and / or subscribe to multiple topics.

Topics are buses the nodes send or receive messages throught. A node must inform the Master that it wants to publish on to send it information or subscribe to receive information. There can exist several nodes simultaneously publishing or subscribing on the same topic. They are anonymous, in the sense that the nodes publish or subscribe to them but they never know which other nodes are publishing or are subscribed on the topic, which decouples the production of information consumption.

All topics are unidirectional, so a node that sends information for a topic never gets an answer by the same topic or know if their messages are being received. When declaring a topic, in addition to its name (which must be unique), it is necessary to define the type of message that is being sent and, therefore, the type of message to be received.

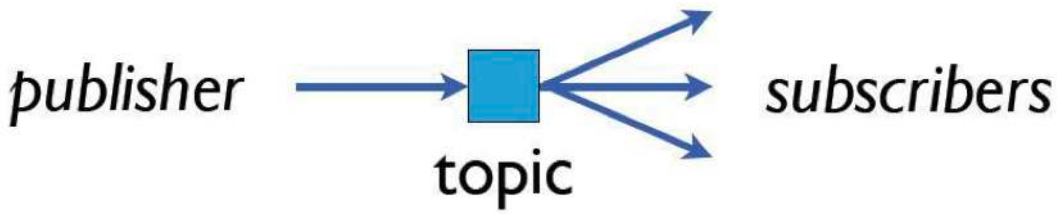


Fig.3: Topic in ROS [12]

When you want to send information and receive an answer the other communication between nodes, services should be used. ROS services are the way a message (request) is sent from one node to another and it replies with another message (response). Thus a node can provide a service (server) and use any other node (client) calling with a message and waiting for a response from it, i.e., with another message.

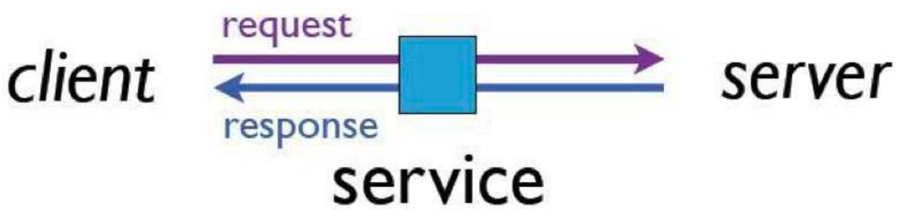


Fig.4: Services in ROS [12]

The service offered by a node is completely independent of the clients. It is possible, for example, that a node offers a service and the user calls this service from a terminal. Services are defined by the *.srv* files, which define the type of services they offer, that is, the kind of message that will be received as request and the type of message that will be returned as response.

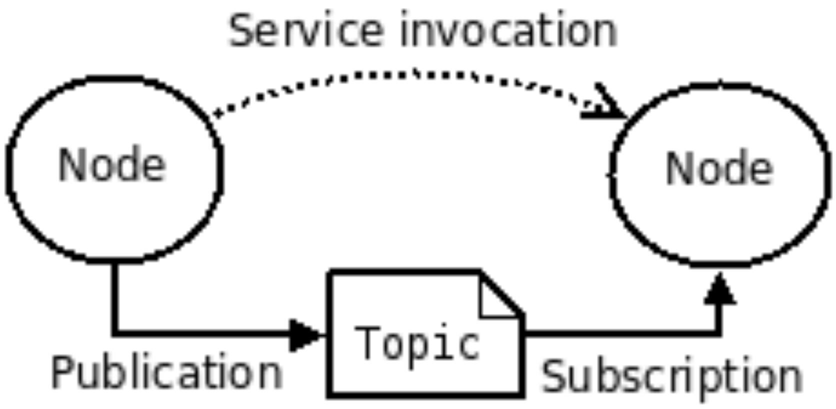


Fig.5: Basic concepts in ROS [17]

The master (roscore) is a node that runs as the system kernel, providing a platform by recording names, services and parameters thanks to which, the communication and sending data between the individual nodes of the system becomes possible. Without the master, nodes would not be able to exchange messages or invoke services, which makes it absolutely essential when running any type of program.

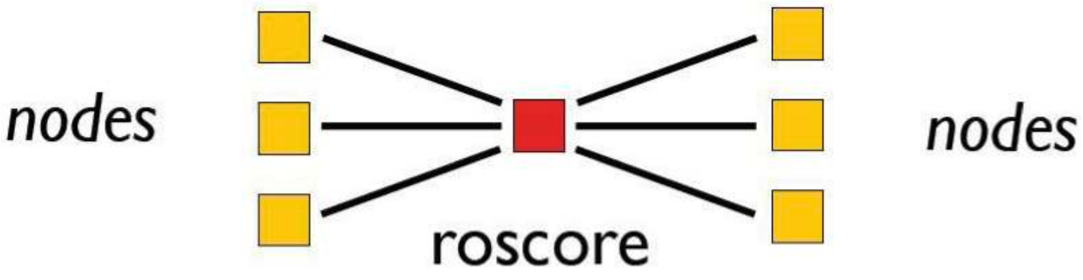


Fig.6: Roscore in ROS [12]

Roscore is a tool that provides three elements:

a) The master, understood as a communication platform for nodes.
b) The server of parameters.
c) The output register where all nodes send information, "/rosout".

The parameter server is used by the nodes to store or read parameters at runtime, besides enabling the user to view or modify these parameters at runtime terminal.

The roslaunch is a tool for easily launching multiple ROS nodes as well as setting the necessary parameters specified to create a complete application. Roslaunch takes in one or more XML configuration files (with the .launch extension) that specify the parameters to set and nodes to launch, as well as the machines where they should be run on.

The execution of the master is mandatory before launching the nodes that need to communicate with each other or need to read parameters, that is, always. To do this we run roscore in terminal.

A bag is a file format in ROS for storing ROS message data. That is, bags are a files for storing and reproducing data from a message of ROS, allowing us to store a series of commands and then repeat them sequencially. These bags are an important data storage mechanism, such as those recorded by a sensor that require further processing (for instance, to develop and test algorithms).

A tool like rosbag typically creates bags, which subscribe to one or more ROS topics, and store the serialized message data in a file as it is received. These bag files can also be played back in ROS to the same topics they were recorded from, or even remapped to new topics.

Using bag files within a ROS Computation Graph is generally no different from having ROS nodes send the same data, though you can run into issues with timestamped data stored inside of message data. For this reason, the rosbag tool includes an option to publish a simulated clock that corresponds to the time the data was recorded in the file.

Bags are the primary mechanism in ROS for data logging, which means that they have a variety of offline uses. Tools like rqt_bag allow you to visualize the data in a bag file, including plotting fields and displaying images. You can also quickly inspect bag file data from the console using the rostopic command (while you are playing it) Rostopic supports listing bag file topics as well as echoing data to screen (by typing rostopic echo).

The data stored within bag files is often very valuable, so bag files are also designed to be easily migrated when msg files are updated. The bag file format stores the msg file of the corresponding message data, and tools like rosbagmigration let you write rules to automatically update bag files when they become out of date.

The command rosbag record subscribes to topics and writes a bag file with the contents of all messages published on those topics. The file contains interlaced, serialized ROS messages dumped directly to a single file as they come in over the wire. This is the most performance and disk-friendly recording format possible. To further reduce disk usage, you can compress bag files as they are created.

For recording messages at a high bandwidth, such as from cameras, it is strongly recommended to run rosbag record on the same machine as the camera, and specify the file destination as being on the local machine disk.

The command rosbag info displays a human-readable summary of the contents of the bag files, including start and end times, topics with their types, message counts and median frequency, and compression statistics.

2.2.- Point Cloud Library (PCL)

Point Cloud Libray (PCL) [http://pointclouds.org/] is a comprehensive free, BSD licensed, library for n-D Point Clouds and 3D geometry processing **[14].** PCL is cross-platform, and has been successfully compiled and deployed on Linux, MacOS, Windows, and Android/iOS.

Point clouds can be acquired from hardware sensors such as stereo cameras, 3D scanners, or time-of-flight cameras, or generated from a computer program synthetically. PCL supports natively the OpenNi 3D interfaces, and can thus acquire and process data from the Microsoft Kinect **[18].**

PCL is fully integrated with ROS, the Robot Operating System, and has been already used in a variety of projects in the robotics community.

Several tools run as ROS nodes. They convert ROS messages or bags to and from Point Cloud Data (*.pcd*) file format. The node bag_to_pcd reads a bag file, saving all ROS point cloud messages on a specified topic as *.pcd* files. For the visualization of point clouds and images captured by the camera Kinect One, we use the package called pcl_ros.

Point clouds extracted from the rosbag can be viewed and analyzed with Point Cloud Library (PCL) **[17].** The pcl_viewer command lets you select the *.pcd* file and visualize the recorded point clouds. An example is shown in Fig. 7. Using the keys 1, 2, 3, 4 and 5 we can change hues to visualize the corresponding image. The PCL tool pcl_pcd2png allows us to extract color images and depth images from the cloud of points. pcl_pcd2png is a terminal command and we can obtain the parametres of this command with pcl_pcd2png–help**.**

This way we can record directly all the desired information to a rosbag, automatically and then extract it in a manipulable and friendly with other software format *(.png*, *.pcd*).

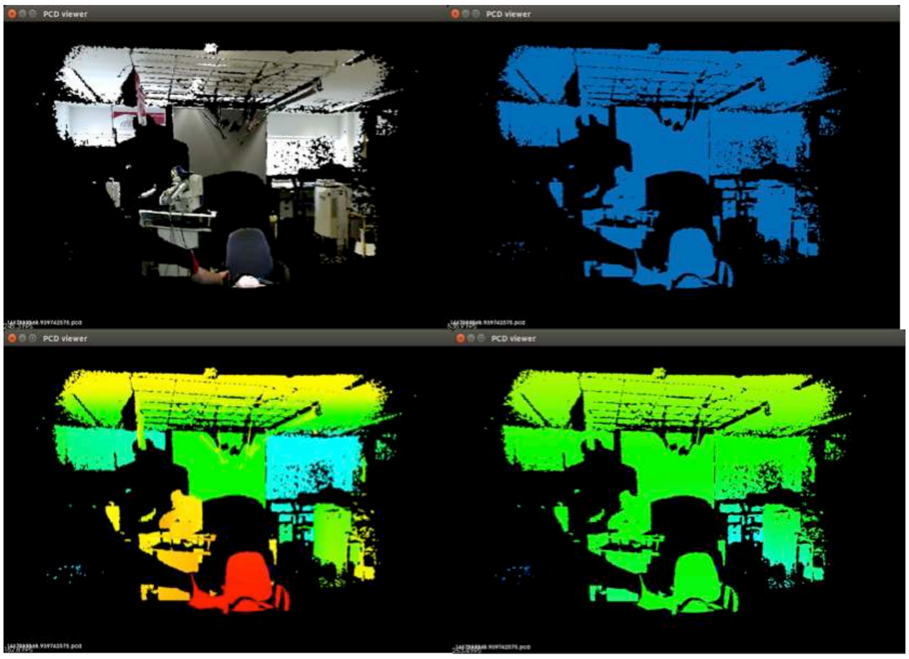


Fig.7: Images obtained with pcl_viewer

2.3.- RViz

ROS provides a graphical tool called RViz that can display a wide variety of information (naturally, by subscribing to appropriately-typed topics that a user selects). RViz is a 3D visualization environment for ROS and it is tightly integrated with the rest of the ROS tools **[19].**

Examples are point clouds, robot models, maps, transforms, camera/image displays and markers. A lot of different display types can then be combined in one view, to create a complete picture of a specific scenario.

RViz allows us to exploit information from point clouds recorded by the sensor Kinect One. Once RViz is opened, you define the image you wish to recover, i.e., a cloud of points and the topic that records the image is defined, resulting in the display shown in Fig. 8 **[3].**

When we wanted to visualize the cloud points recorded by the camera Kinect One, a problem araised: RViz could not extract, from the information provided by the camera, the position of the origin of coordinates. The solution to this problem has been found in the support forum of https://github.com/code-iai/iai_kinect2 that is, the repository of the drivers we have used for the Kinect One. To solve the problem, we initialize the camera:

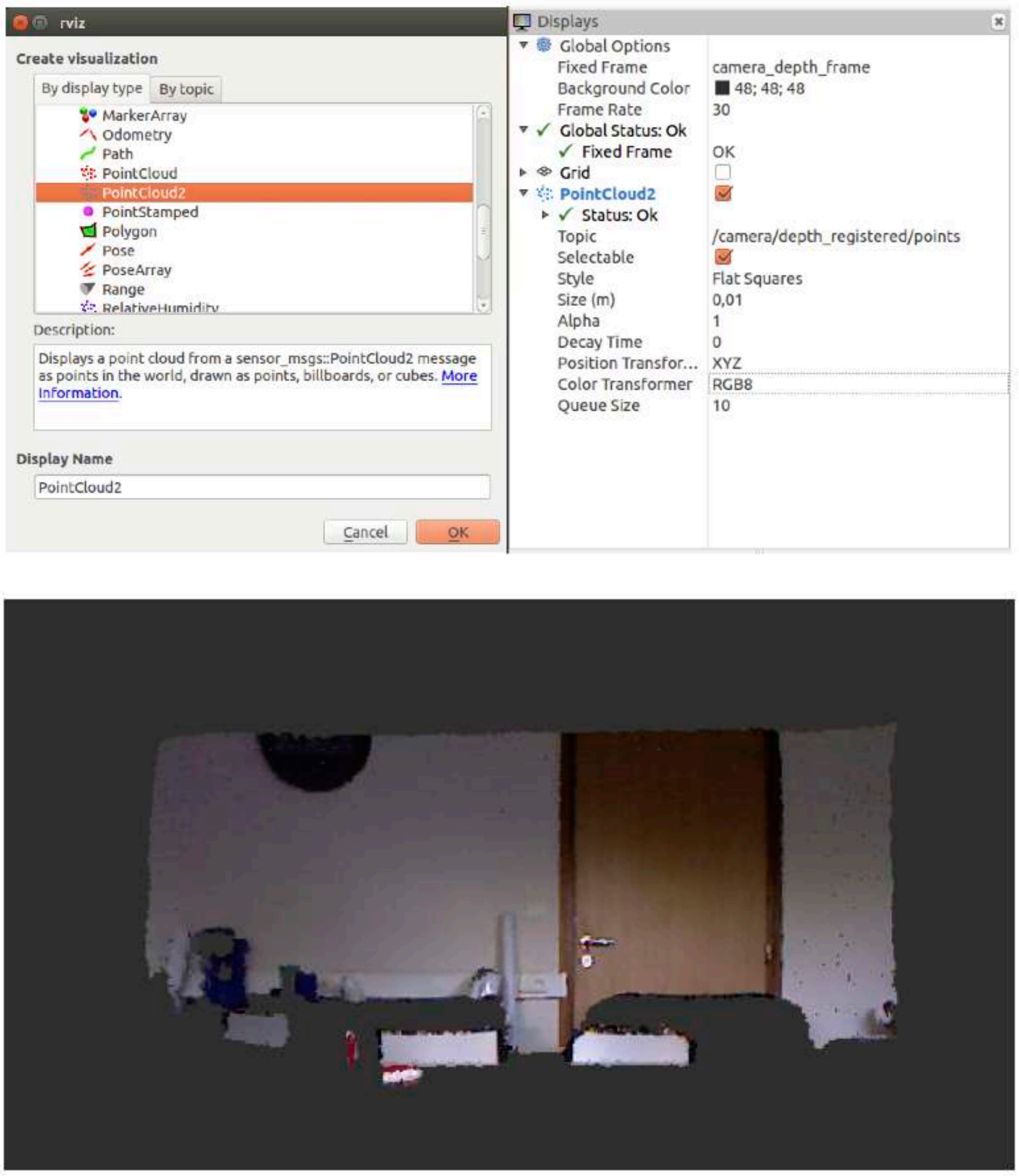
```
roslaunch kinect2_bridge kinect2_bridge.launch publish_tf: = true
```

instead of:

```
roslaunch kinect2_bridge kinect2_bridge.launch
```

for the camera to publish the information necessary in ROS to locate the origin of coordinates.

After this we can set as "fixed frame" in "global options", kinect2_link, and now we can visualize the desired information from the camera.



Description:

Displays a point cloud from a sensor_msgs::PointCloud2 message as points in the world, drawn as points, billboards, or cubes. More Information.

Display Name: PointCloud2

Global Options	
Fixed Frame	camera_depth_frame
Background Color	■ 48; 48; 48
Frame Rate	30
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	☐
PointCloud2	☑
✓ Status: Ok	
Topic	/camera/depth_registered/points
Selectable	☑
Style	Flat Squares
Size (m)	0,01
Alpha	1
Decay Time	0
Position Transfor...	XYZ
Color Transformer	RGB8
Queue Size	10

Fig.8: Visualization of the point cloud with RViz [3].

2.4.- Tf library

When combining views of different display types in RViz, it is important to consider how the different coordinate systems relate to each other **[4]**.

RViz uses the tf transform system **[23]**. By using tf, RViz is able to transform the coordinate frames of the different displays into a global reference frame. Choosing which frame to use as the global reference frame, can easily be done as a part of the configuration.

The tf library was developed as ROS package and has been adopted by the ROS community as the primary way to keep track of positional information.

The tf library is a standard way to keep track of coordinate frames and transform data within an entire system such that individual component users can be confident that the data is in the coordinate frame that they want without requiring knowledge of all the coordinate frames in the system.

The tf library allows, therefore, performing the transformation (translation and rotation) between two different reference frames in the required time. That is to say, it establishes the relationships between different coordinate systems.

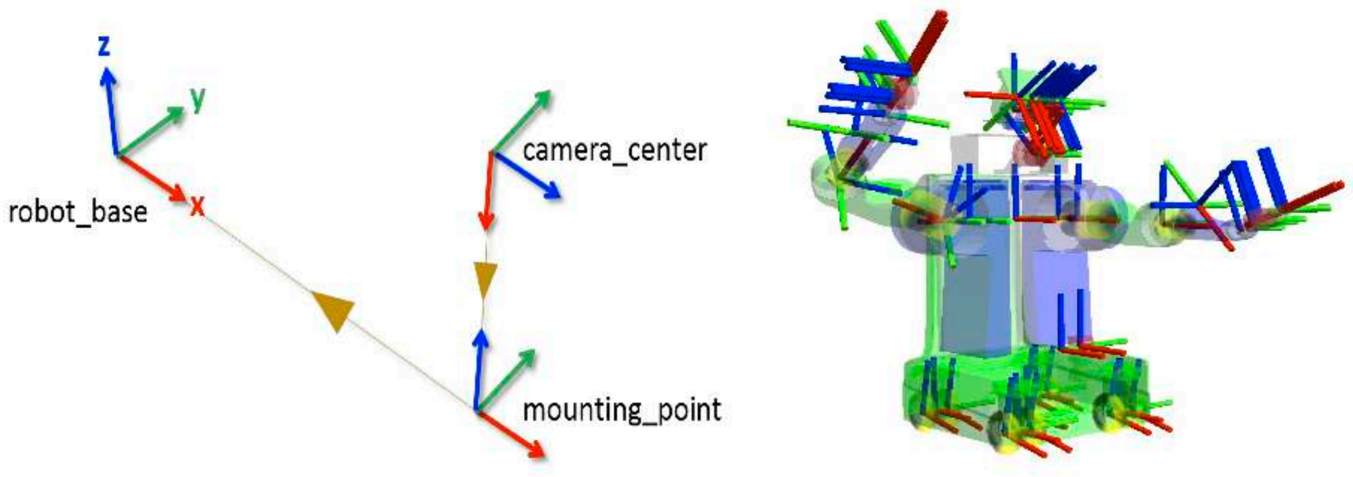


Fig.9: A robotic system typically has many 3D coordinates frames that change over time [23]

The tf library can operate in a distributed system. This means all the information about the coordinate frames of a robot is available to all ROS components on any computer in the system. There is no central server for transform information.

Although tf is mainly a code library meant to be used within ROS nodes, it comes with a large set of command-line tools that assist in the debugging and creation of tf coordinate frames. These tools include:

- view_frames: visualizes the full tree of coordinate transforms.
- tf_monitor: monitors transforms between frames.
- tf_echo: prints specified transform to screen.
- static_transform_publisher: is a command line tool for sending static transforms.
- tf_remap: is a utility node for remapping coordinate transforms.
- roswtf: with the tfwtf plugin, helps you track down problems with tf.

3.- Microsoft Kinect XBOX One.

The Kinect One (Kinect v2) is a Time of Flight (ToF) camera that uses the back trip time of light for the calculation of the depth of an object **[5, 6, 11, 20, 21, 25].** The Kinect One has the same number and type of sensors as Kinect 360: a color camera, an infrared camera and an array of microphones. A strobed infrared light illuminates the scene, obstacles reflect the light, and the infrared camera registers the time of flight for each pixel. Internally, wave modulation and phase detection is used to estimate the distance to obstacles (indirect ToF).

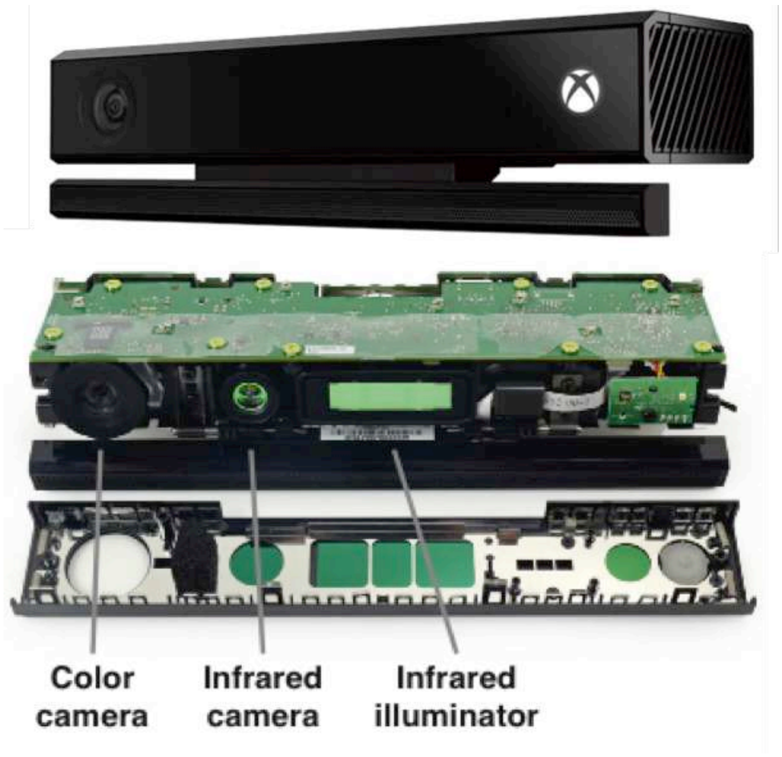


Fig.10: Kinect XBOX One [5, 25]

The colour camera has a resolution of 1920x1080; a field of view of 84.1°and 53.8°; and focal distances of 1036.32±3.72 and 1030.4±3.87. The infrared camera has a resolution of 512x424, a field of view of 70.6° and 60.0°; and focal distances of 364.15±1.45 and 362.40±1.45. All values in the preceding enumeration were respectively the horizontal and vertical directions. Due to these differences and other distortion coefficients, the camera has to be calibrated to obtain appropriate transformations to be applied in order to align the color, infrared and depth images.

The Kinect infrared camera detects infrared light with a wavelength between approximately 827 and 850 nm^2. This is a much smaller range than the one used by conventional infrared cameras for night vision. This makes sense since the Kinect uses the camera to record the projected dots of a certain wavelength to construct the depth map, while night visions cameras aim to capture as much light as possible to record the scene in low-light conditions **[25].**

The Kinect One uses several algorithms to create a depth map, most notable a time-of-flight algorithm. This results in much higher resolution depth data compared to the structured light method of the Kinect 360, which is very similar to triangulation used by in stereovision cameras. A new feature in the Kinect One is the self-adaptation of the exposure time of the RGB image; the Kinect, in fact, by automatically adapting this parameter, limits the number of frames that can be captured in a certain time, thus reaching a maximum frame rate of 15 fps. However, the captured images are brighter.

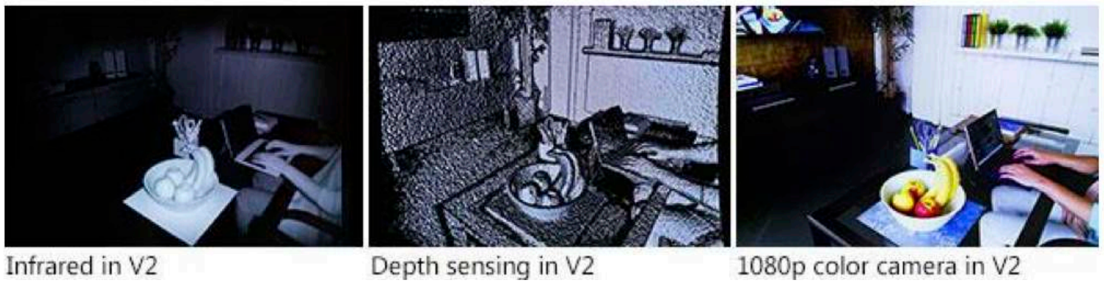


Fig.11: Images obtained with the three sensors available in the Kinect One (Kinect v2) [25]

Figure 12 **[21]** shows the 3D image sensor system of Kinect One. The system consists of the sensor chip, a camera SoC, illumination, and sensor optics. The SoC manages the sensor and communications with the Xbox One console. The time-of-flight system modulates a camera light source with a square wave. It uses phase detection to measure the time it takes light to travel from the light source to the object and back to the sensor, and calculates distance from the results.

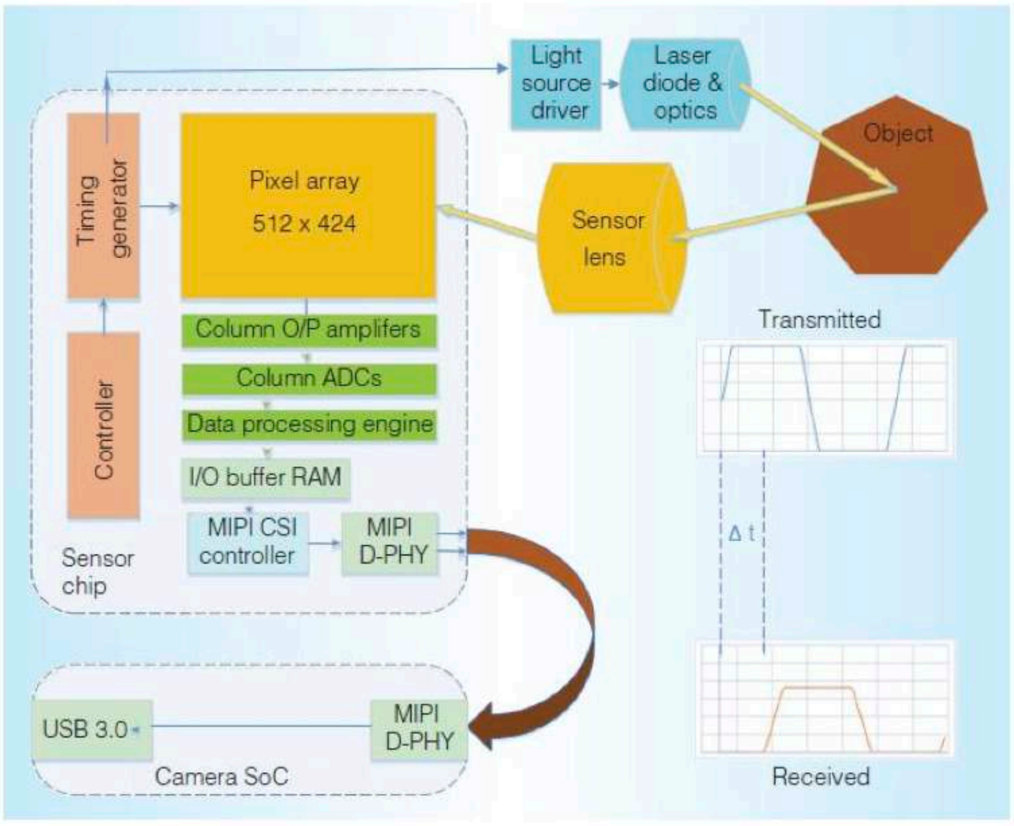


Fig.12: 3D image sensor system of Kinect One. The system comprises the sensor chip, a camera SoC, illumination, and sensor optics [21].

The timing generator creates a modulated square wave. The system uses this signal to modulate both the local light source (transmitter) and the pixel (receiver). The light travels to the object and back in time Δt. The system calculates Δt by estimating the received light phase at each pixel knowing the modulation frequency. The system calculates depth from the speed of light in air: 1 cm in 33 picoseconds. Figure 13 **[21]** shows the time-of-flight sensor and signal waveforms. A laser diode illuminates the subjects and then the time-of-flight differential pixel array receives the reflected light. A differential pixel distinguishes the time-of-flight sensor from a classic camera sensor. The modulation input controls conversion of incoming light to charge in the differential pixel's two outputs. The timing generator creates clock signals to control the pixel array and a synchronous signal to modulate the light source. The waveforms illustrate phase determination. The light source transmits the light signal that travels from the camera, reflects off any object in the field of view, and returns to the sensor lens with some delay (phase shift) and attenuation. The lens focuses the light on the sensor pixels. A synchronous clock modulates the pixel receiver. When the clock is high, photons falling on the pixel contribute charge to the A-out side of the pixel. When the clock is low, photons contribute charge to the B-out side of the pixel. The (A – B) differential signal provides a pixel output whose value depends on both the returning light level and the time it arrives with respect to the pixel clock. This is the essence of time-of-flight phase detection **[21].** Thus, the system measures the phase shift of a modulated signal (Fig.14) **[25].**

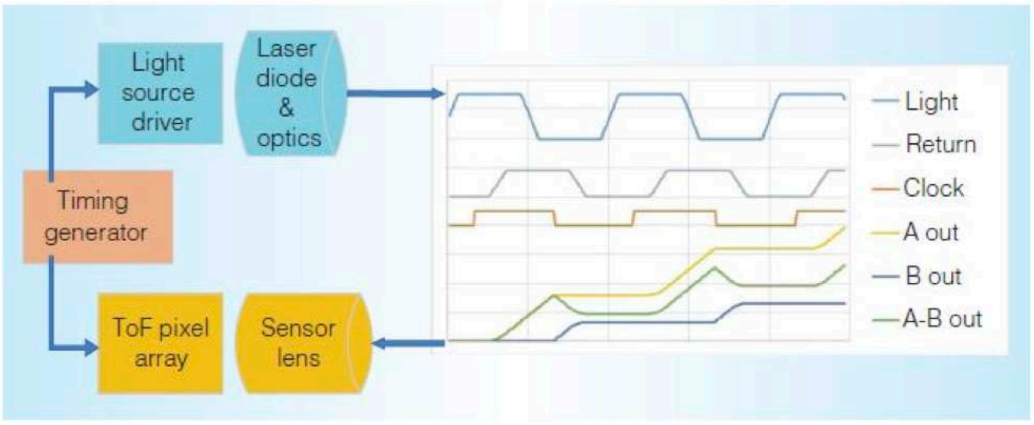


Fig.13: Time-of-flight sensor and signal waveforms. Signals "Light" and "Return" denote the envelope of the transmitted and received modulated light. "Clock" is the local gating clock at the pixel, while "A out" and "B out" are the voltaje output waveforms from the pixel [21].

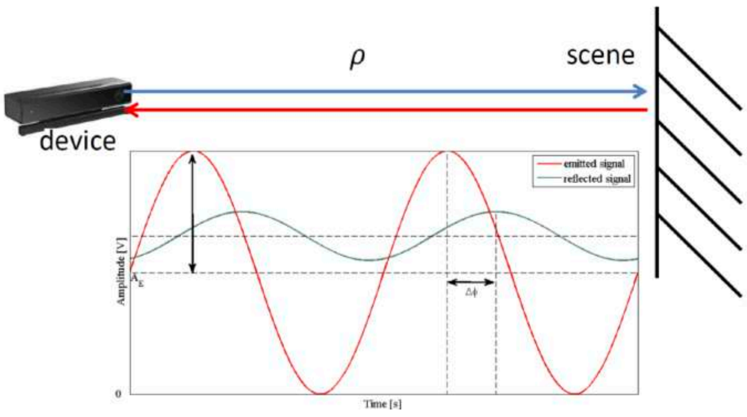


Fig.14: Basic operation principle of a Time-of-Flight camera [25]

4.- Kinect One Control with ROS.

In September 2014, Thiemo Wiedemeyer (Institute for Artificial Intelligence at the University of Bremen) developed a collection of tools and libraries for the Kinect v2 (https://github.com/code-iai/iai_kinect2) **[24].** The project is still under development but it aims to make available all the features that were usable with the first Kinect and it depends on the evolution of the libfreenect2 driver, which is under development too. The libfreenect2 driver is developed by OpenKinect, and open source community with the main purpose of enabling the use of the Xbox Kinect (v1 and v2) on Linux, Mac and Windows. The toolkit "iai_kinect2" contains:

a) A ROS interface to the device (driver) using libfreenect2;
b) A calibration tool using OpenCV for calibrating the IR sensor of the Kinect One to the RGB sensor and the depth measurements;
c) A library for depth registration with OpenCL support;
d) The bridge between libfreenect2 and ROS; and
e) A viewer for the images/point clouds.

The system has been developed for and tested in both ROS Hydro and Indigo (Ubuntu 12.04 and 14.04). The driver has been improved to reach high performance, meaning to be able to process the sensor's information at full frame rate (30Hz) on acceptable hardware (not only high-end machines). This was achieved through parallelization of the image pipeline **[24].**

Kinect2 bridge is mainly involved in acquiring the data from the Kinect through the libfreenect2 driver and in publishing them in the topics. The structure follows ROS standard for the cameras: camera/sensor/image. So, the following images are published **[24, 25]:**

- RGB/image: the colour image obtained by the Kinect2 with a resolution of 1920x1080 pixels;
- RGB_lowres/image: the rectified and undistorted color image, rescaled with a resolution of 960x540 pixels, Full HD's half;
- RGB_rect/image: the rectified and undistorted color image with the same size as the original;
- Depth/image: the depth's image obtained from the Kinect2 with a resolution of 512x424 pixels;
- Depth_rect/image: the depth's image overhauled and warmed up with a resolution of 960x540 pixels;
- Depth_lowres/image: the overhauled depth's image with the same size as the original;
- Ir/image: the infrared image reached from the Kinect2 with a resolution of 512x424 pixels;
- Ir_lowres/image: the rectified and undistorted color image, rescaled with a resolution of 960x540 pixels;
- Ir_rect/image: the rectified and undistorted infrared image with the same size as the original.

To launch Kinect2 Bridge use a command like:

```
roslaunch kinect2_bridge kinect2_bridge.launch
```

or

```
rosrun kinect2_bridge kinect2_bridge
```

You can also specify some parameters (only available with rosrun) as:

- fps, which limits the frames per second published;
- calib, to indicate the folder that contains the calibration parameters, intrinsic and extrinsic;
- raw, to specify to post pictures of depth as 512x424 instead of 960x540.

The calibration data used are those that can be estimated using the calibration package available in the iai_kinect2 repository.

The calibration of the Kinect v2 camera has been carried out following the guidelines in the tutorial of the package iai_kinect2:

- https://github.com/code-iai/iai_kinect2/tree/master/kinect2_calibration.

It is posible to choose different calibration patterns and we have selected chess5x7x0.03.pdf **[24].** After executing the commands in the tutorial, we have recorded the images required for calibration, in Fig. 15 a few examples are shown.

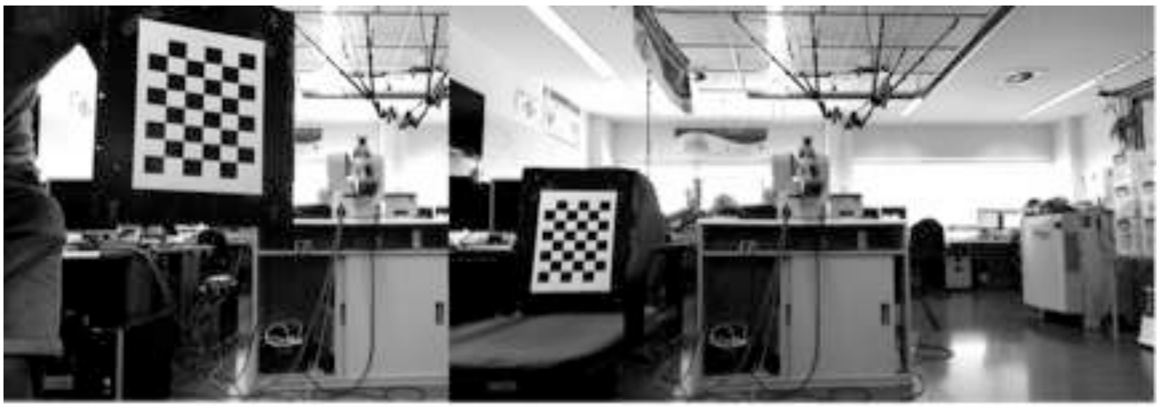


Fig.15: Calibration images for Kinect v2

The images necessary for calibration have been recorded with the two built-in cameras: the infrared camera and the RGB camera. We have recorded around 100 pictures, approximately, with each camera in order to ensure that the pattern was present in the entire view range of the camera.

Once calibration is completed, an example of image is shown in Fig. 16, where it can be seen that the image color (RGB camera) fits quite well to the point cloud (chamber depth) because it allows a 3D reconstruction, so we conclude that the calibration has been carried out correctly.

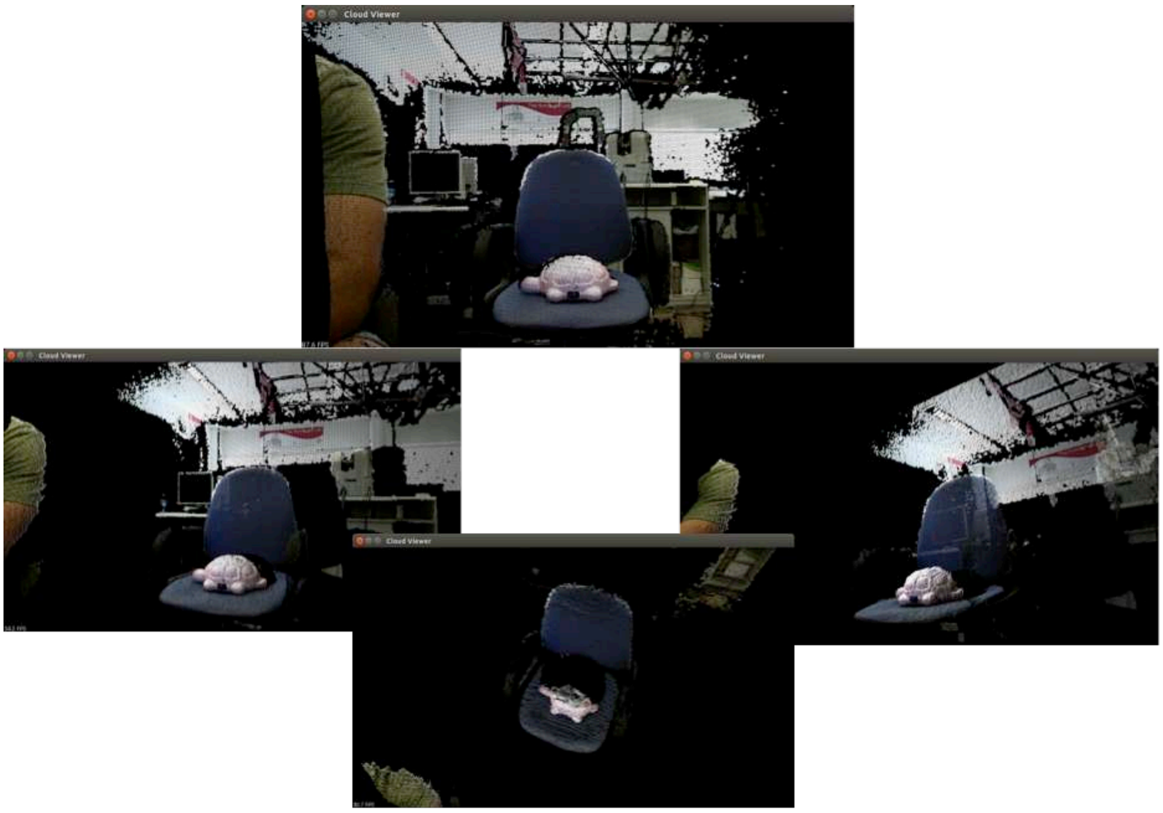


Fig.16: Front view image (top) that allows a 3D reconstruction (middel and bottom) after Kinect v2 calibration

To record images with Kinect v2, the package developed by T. Wiedemeyer offers the following alternative: We must activate the node kinect2_viewer to project the video in streaming on the computer. And then, when we want to perform the image capture, we can press the space bar or the "s" key. This process generates three images in *.jpg,* and *.png* format (the color image, the depth image and the colored depth image) and a point cloud in *.pcd* format.

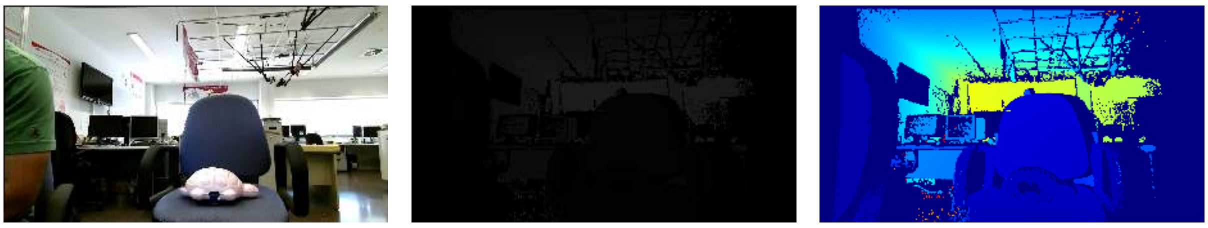


Fig.17: From left to right the color image, the depth image and the colored depth image

This process is entirely manual, but the keystroke is a process that can be automated by sending, to the current program, a simulated keyboard-event, using the Ubuntu xdotool package, which is a tool that can simulate the actions of the mouse or keyboard by commands in the terminal. Thus, to achieve our purpose of automating the recording of images captured by the Kinect v2, we have created the following *.sh* executable in terminal:

```
while sleep 2 #seconds between recordings
do
      xdotool key s
done
```

We run it by typing ./file_name.sh in terminal. The computer simulates pressing the "s" key to the cadence marked. Once this script has been executed we must click on the corresponding window of the Kinect One and it starts to take snapshots with the cadence we have set. However, it is not very precise for high frequencies, as it cannot reach a snapshot cadence bigger than 0.2 Hz.

Nevertheless, the best tool for the recording of images by Kinect One sensor is rosbag that allows us to extract and record directly in a .bag file the information that is being published on the wished topics. As we mentioned previously, bags are the primary mechanism in ROS for data logging, which means that they have a variety of offline uses. Tools like rqt_bag allow the visualization of the data in a bag file, including plotting fields and displaying images. You can also quickly inspect a bag file data from the console using the rostopic command.

The method chosen for the recording affects the recording rate of Kinect One camera, that also depends on the amount of information required (image resolution). Taking into account that images can be generated with two different resolutions: 960x540 and 1920x1080, we have checked the possibilities with the following results:

a) The recording using rosbag allows a faster recording rate than the recording of images from the viewer.
b) If the color, depth and point clouds images are recorded with a 960x540 resolution, the rosbag records them with a rate of 10 fps, while the manual method from the viewer can achieve a maximum rate recording of 4 fps.
c) If the color, depth and point clouds images are recorded with a 1920x1080, resolution, the rosbag records them with a rate of 2 fps while the manual method from the viewer can achieve a maximum rate recording of 1 fps.

In this work we used the rosbag as a method for storing data captured by the Kinect One camera. Point clouds are extracted from the rosbag in point cloud (*.pcd*) format and all the information obtained by the Kinect One sensor is present in them, as they are generated from the overlapping of color images and depth images.

A point cloud is a data structure used to represent a collection of multi-dimensional points and is commonly used to represent three-dimensional data. In 3D point cloud, the X, Y, and Z geometric coordinates usually represent the points of an underlying sampled surface. When color information is present, the point cloud becomes 4D.

5.- OptiTrack System

OptiTrack is an optical motion capture system. The system is used to determine the position and orientation of an object in the three dimensional space. A set of infrared cameras is adequately placed and infrared light reflectors are attached to the object whose position we want to ascertain. The cameras will perceive these reflectors (markers) with great intensity and among all of them, triangulating the position of the markers, they are able to determine the position and orientation of the body at all times (Fig. 18).

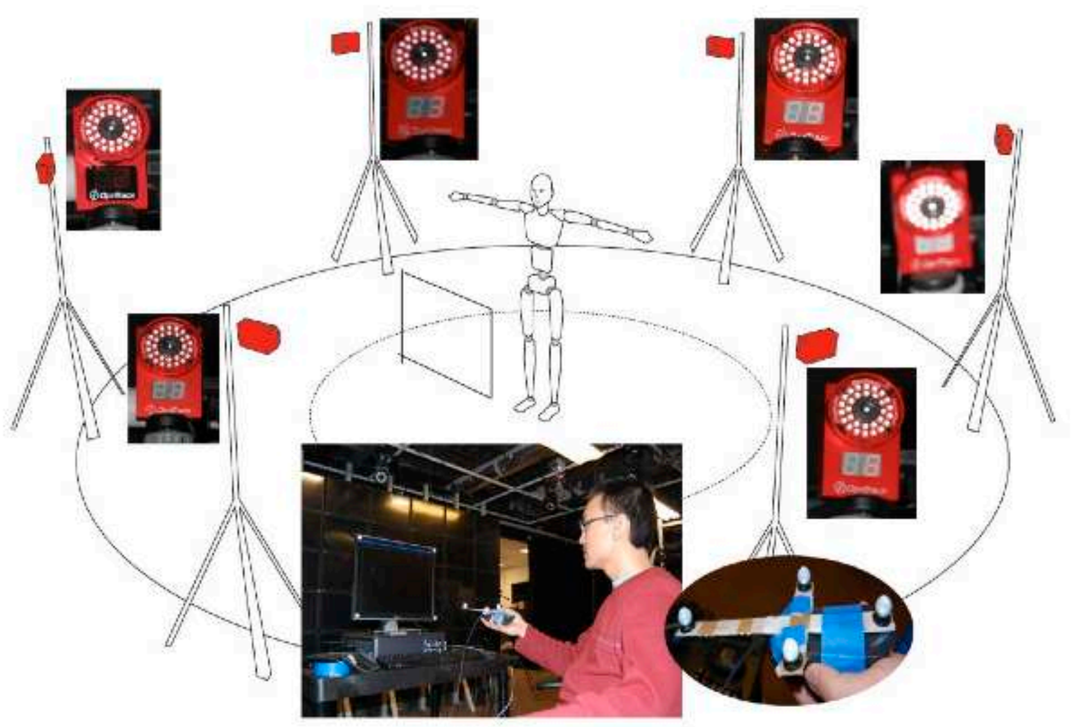
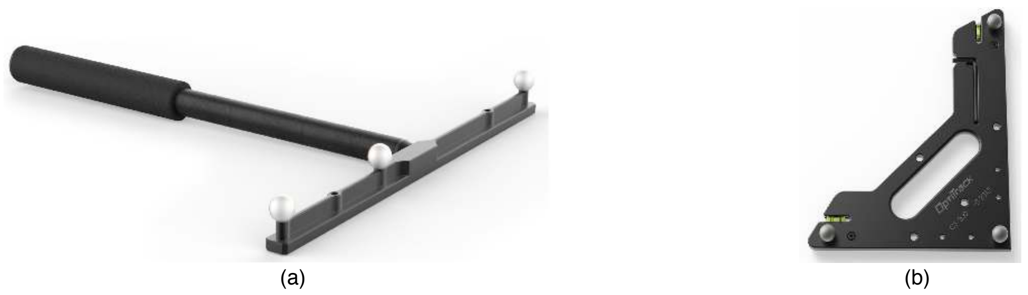


Fig.18: Outline of the OptiTrack system

An initial calibration of the system must be performed before taking measurements with the OptiTrack. For this, we use the instrument shown in Fig. 19 (a), whose dimensions are already known. It allows us to place several reflectors at predetermined distances:



(a) (b)

Fig.19: Instruments for the Optitrack Calibration and reference system setup

This tool is moved across the field of vision of each one of the cameras, in order to accomplish the greatest number of possible measures. The result, in the control screen, is shown in Fig. 20.

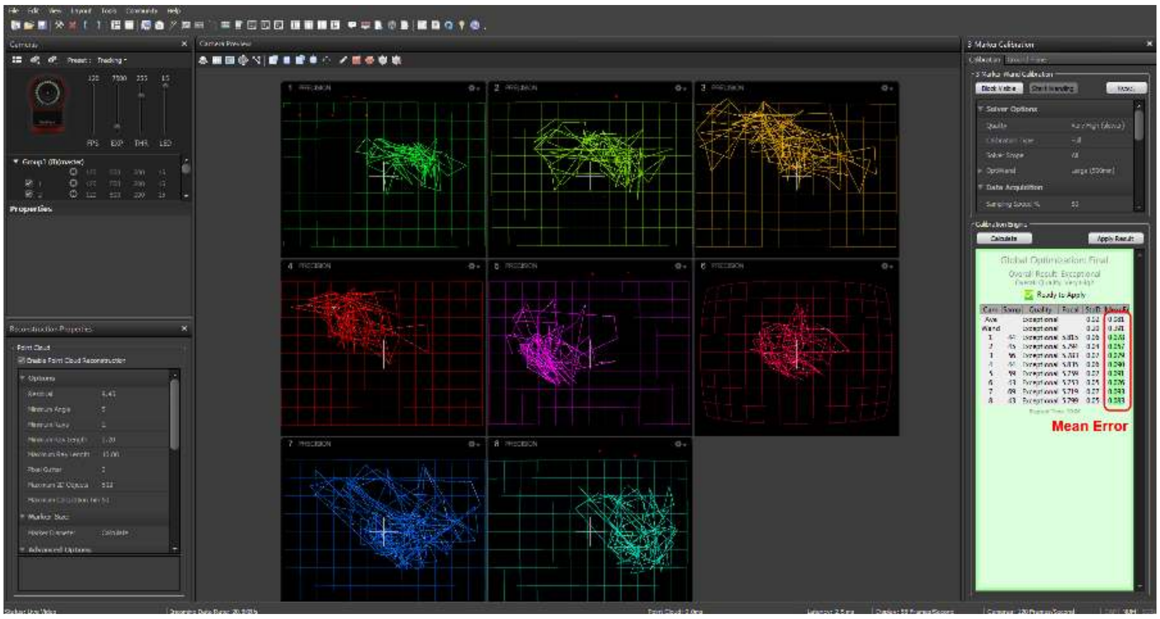


Fig.20: Sample images for calibration OptiTrack

Once the Optitrack is properly calibrated we place the origin of coordinates and orientation of the axes of the cartesian coordinates using the instrument shown in Fig. 19 (b), which should be placed at the chosen origin of coordinates and oriented according to the direction we intend to have for our reference system. However, the orientation of the reference system can be modified later.

The equipment installed in the IRI has a Windows control software called Motive:Tracker (Motion capture & 6 DOF object tracking). Among the features of this software is the ability to track the individual trajectory of each reflector, storing in a table (in *.csv* format) the data of the temporal evolution of the spatial position of each marker. That is, you can follow the trajectory of non-rigid groups of markers.

To explore the possibilities offered by this software Motive:Tracker the following urls can be consulted:

- https://www.youtube.com/playlist?list=PLdKrdVGpQ5OZSlQDCyYFT-iOo1QmfzI7Z
- http://wiki.optitrack.com/index.php?title=OptiTrack_Documentation_Wiki

The first one redirects us to a few short tutorials for the software Motive:Tracker, for tracking rigid bodies and markers individually, that is the one in the IRI laboratory. The software Motive:Body for tracking human bodies is not available at the IRI. The second url provides some basic references.

Among the tutorials, Manual Marker Labeling will be of special interest as it shows how to edit the sequence, once is recorded, and then exporting the tracking data from the Optitrack.

After recording the sequence the layout must be edited. First markersets for the different objects that we want to track must be created. For tracking a rigid body, a rigid body asset can be defined by clicking the right button when the markers that define the rigid body are selected and then, option Rigid Body and finally Create From Selected Markers. The software will automatically define a centroid for the rigid body but we can change its position if we want. The position of this centroid will also be exported as tracking data (Fig. 21).

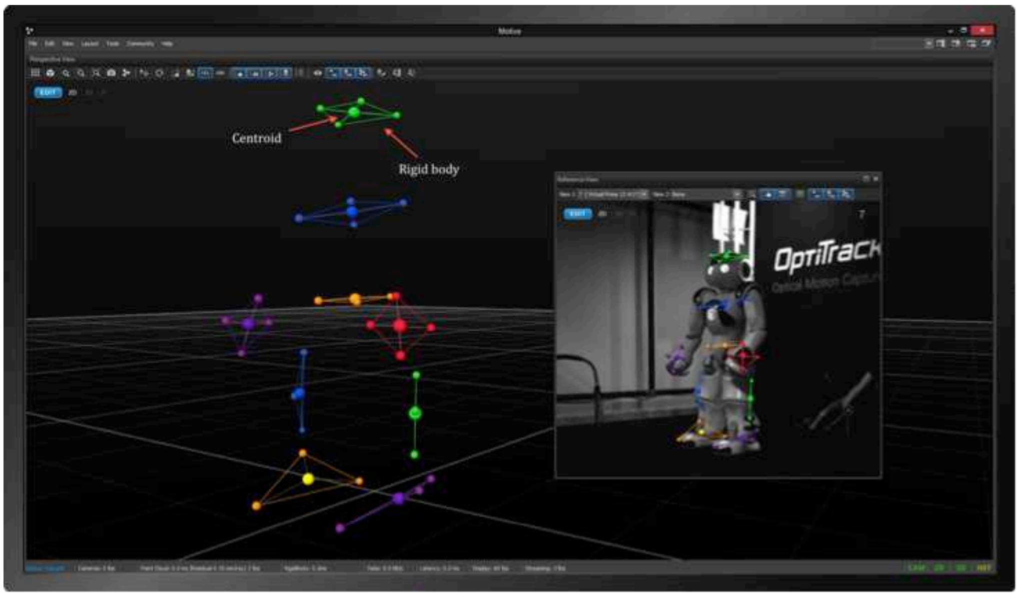


Fig. 21: Rigid bodies in Motive:Tracker

If we want to track the markers individually the process is a bit more complex. For example, if we want to track a non-rigid body such as the human body we have to create an asset first. In this asset we should define the name of the markers that will compose it. After defining our asset we now can label the markers one by one by selecting them, clicking right button and in the option Label Marker select one of the names that we have previously defined in the asset. By doing this when we export the tracking data we can identify each marker's trajectory looking for its name (Fig. 22).

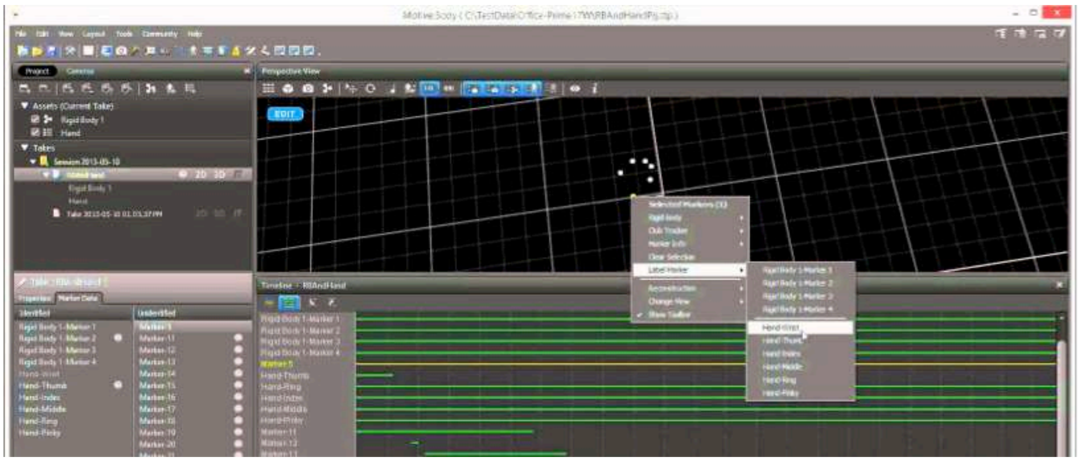


Fig. 22: Labelling markers manually

However OptiTrack tracking is not always perfect. Sometimes we can have some oclusions while recording the sequence, that is, that the OptiTrack loses the position of a marker for a few instants, maybe because it has been covered. When this happens, the OptiTrack stops the tracking of the marker and when it appears again the software identifies it as a new marker and it is not labelled and it must be relabelled again. If we do this, the trajectories of the marker before the occlusion and after will be automatically joined (Fig. 23).

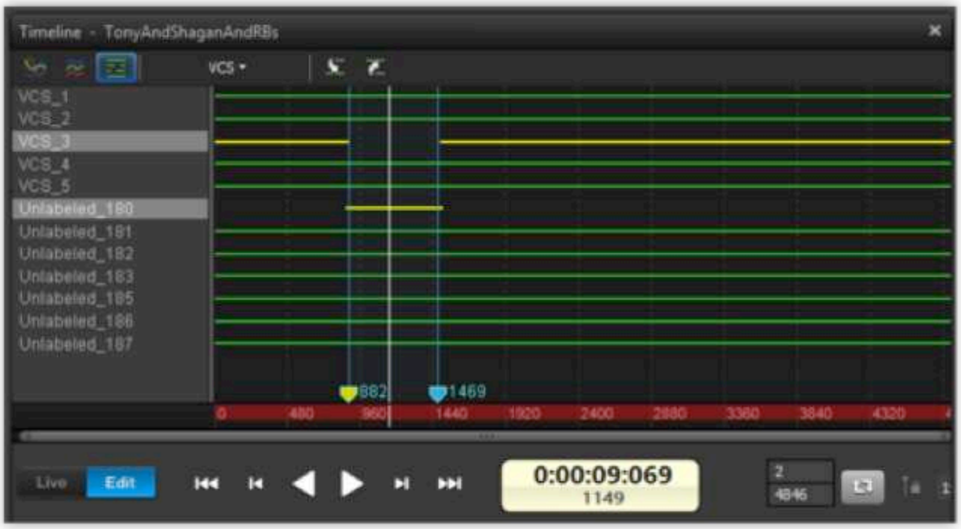


Fig. 23: What happens when we have some occlusion

Nevertheless after joining the trajectories it can still remain some small gaps in the marker's trajectory. They can be big if the marker has been covered for a long time or very small if it only has been covered an instant. If we want to fill this small gaps we can interpolate its trajectory in multiple ways selecting the option that we consider best, in window Gap Fill on the right size of the edit layout.

All in all, we can see the OptiTrack system with its control software allows us to assign a name to each reflector and to record its sequence of movements. The Optitrack records the time evolution of the coordinates x, y, z for each marker individually, being able to reconstruct their trajectories. After editing our sequence we can export the tracking data in *.csv* or in *.c3d* format. In this file, we will have the x, y, z coordinates of each marker in each OptiTrack's capture. Now, we can plot the trajectory of a marker. We have done it with Matlab and Figure 24 shows the results.

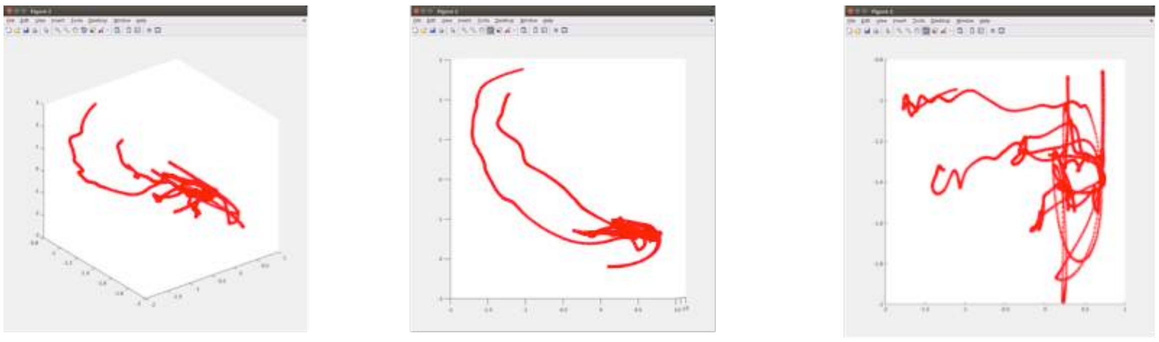


Fig. 24: Representation of the trajectory of a marker from its tracking data

6.- Tracking the Human Body

The aim of this work is to create a database with different body positions recording the depth images captured by the camera Kinect One and the position of a number of markers strategically placed to define precisely the position of the body at all times. Now that we know how to exploit the Motive:Tracker software, we have to think a way to track the human body position. In this section, the placing of the reference points used to determine the position of the human body will be described.

In principle, it is intended that the recording of the information obtained by the OptiTrack allows individual monitoring of each one of the reflectors, considering that they are not attached to a rigid solid. The position of the corresponding human body can be determined from the individual position of each reflector at a given time, taking into account their strategic location.

The initial approach is to adjust the placement of the reflectors to the model as shown in Figure 25.

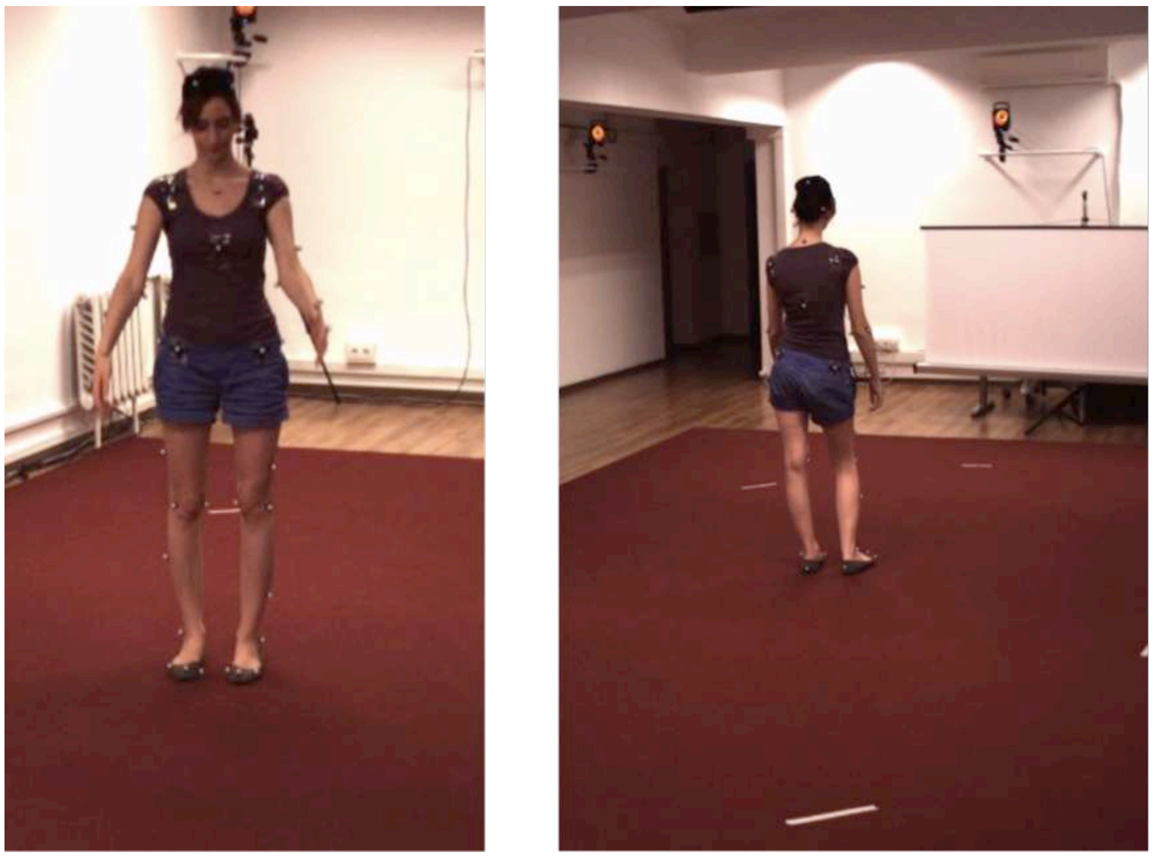


Fig.25: Placement of the reflectors over the human body.

Given the means and resources available in the IRI (we have 25 markers for the Optitrack), it has been outlined a placement scheme of the markers shown in Fig. 26.

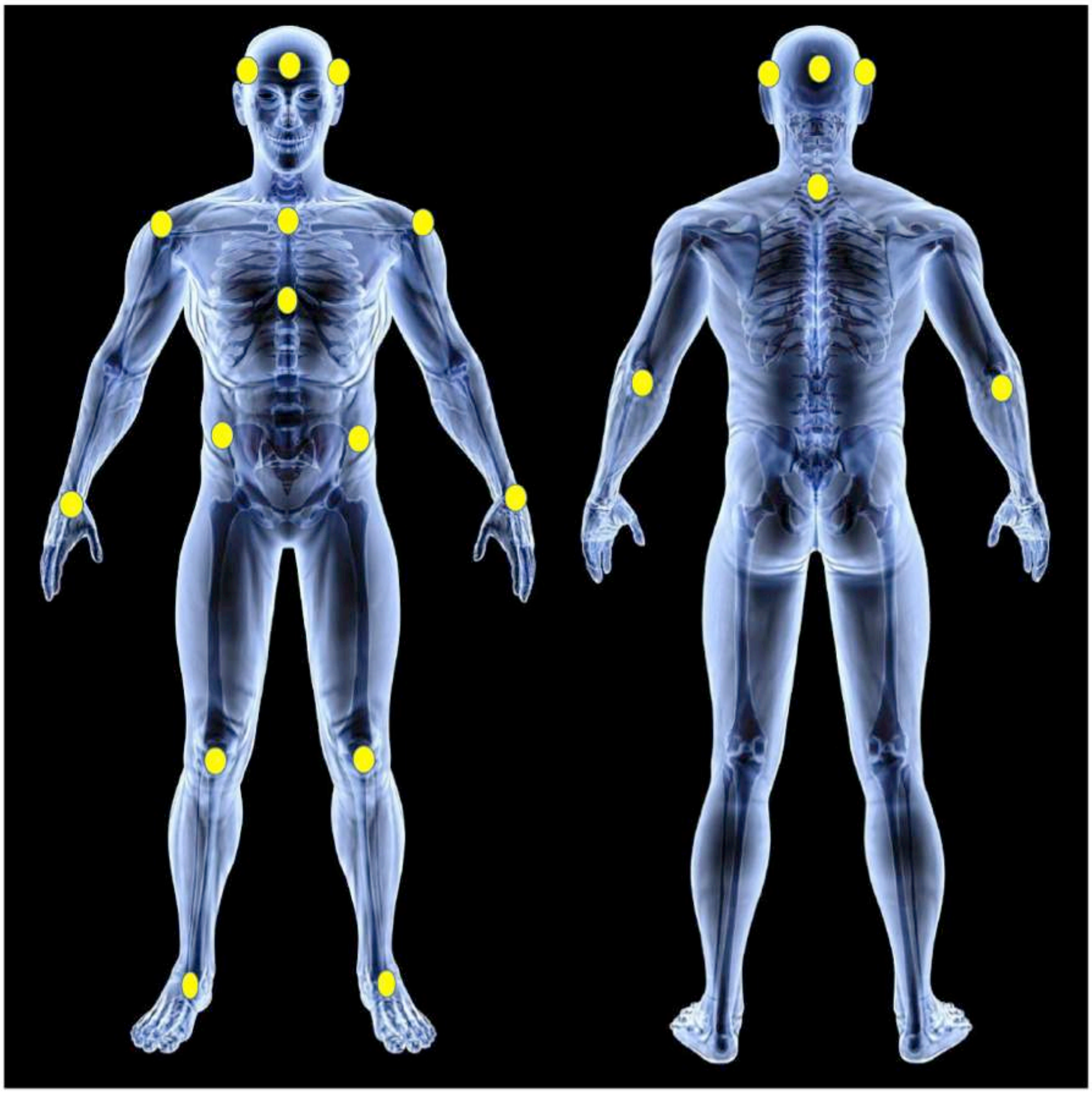


Fig.26: Definitive placement of the reflectores over the human body

From top to bottom, the reflectors are placed in the following way:

- 4 on the head: They must be placed in the line above the eyebrows forming a rhombus between them, i.e., facing two to two, and the side markers are at the same distance from the ones placed on the back and on the front of the head. The one placed on the front must be between the eyebrows.
- 2 on the neck: the one located in the front must be placed in the sternal notch and the one located at the rear in the vertical vertebrae.
- 1 in each shoulder: they must be placed on the acromion.
- 1 in each elbow: they must be placed on the olecranon.
- 1 on each wrist: they are placed on top of the wrist with the palm facing down.
- 1 on the chest: It must be placed on the xiphoid process.
- 1 on each hip: They must be placed on the ASIS.
- 1 on each knee: They must be placed on the kneecap.
- 1 in each ankle: They must be placed on the distal portion of the tibia and fibula.

By this placement of the markers, we intend to segmentate the body in such a way that we can estimate the position of the full body only having the positions of some specific points.

To ensure a correct position and firm grip to the body during the execution of the movements that the model must perform in each sequence, we have decided to make shirts to attach the markers on them. To make the OptiTrack suits we first bought three long sleeve T-shirts in three different sizes for the variety of models that we can use for recording the sequences. The markers that we have in the IRI stick to the surface by Velcro, so we prepared the shirts we bought for sticking the markers on them by attaching some Velcro straps on the shirts on the desired points with textile glue (Fig. 27). The markers in the knees and in the ankles have been attached to the body with some elastic rubbers.

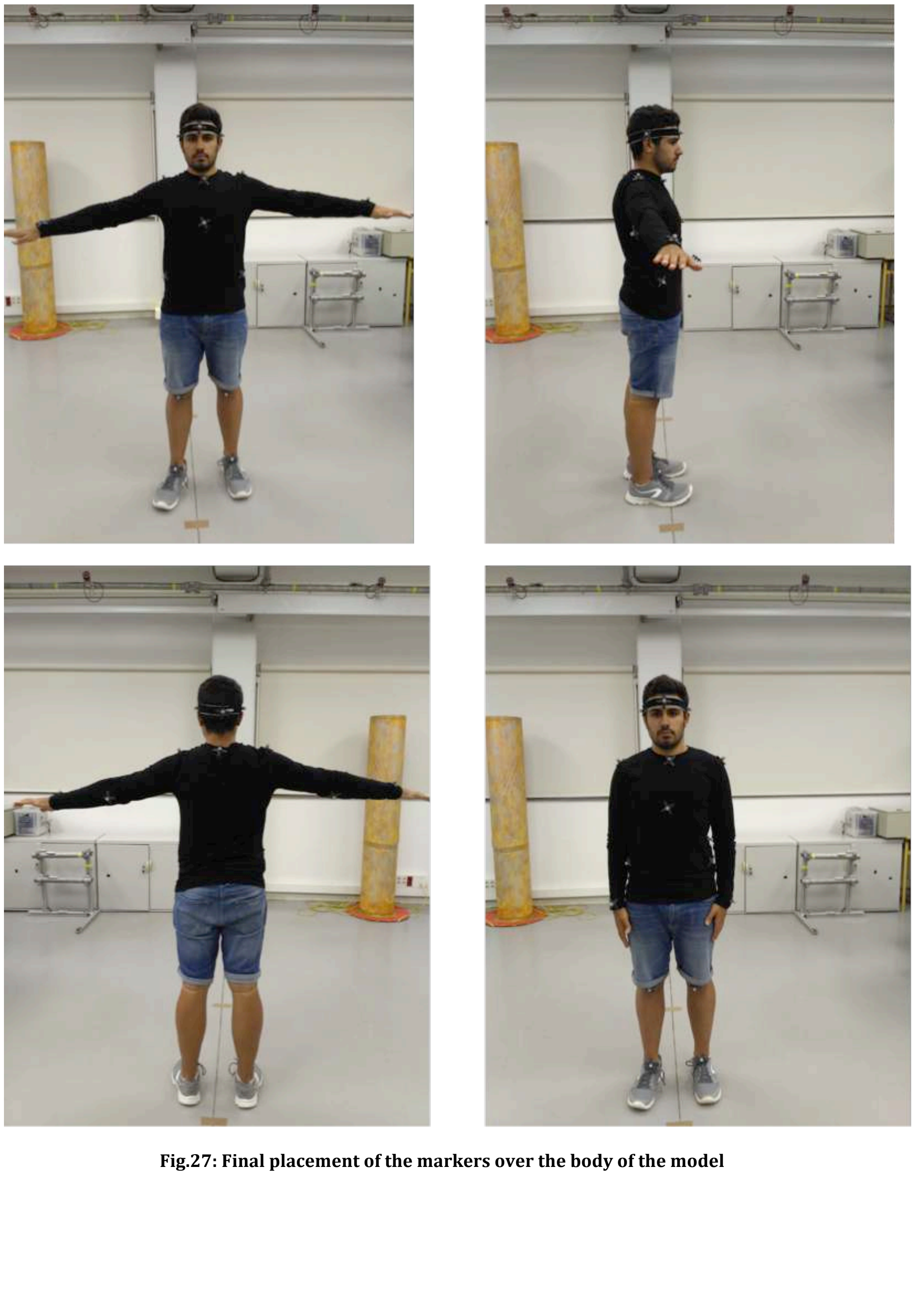


Fig.27: Final placement of the markers over the body of the model

7.- Movement Sequences

Once the reflectors are placed on the model's body, it is necessary to establish the sequences of movements that must be performed by the model. In this section, the experimental setup and the movements made by the models are described.

The Kinect camera is placed on a tripod within the vision field of the Optitrack, so that it visualizes the whole body of the model that performs the programmed movement sequences. It has been placed a few centimetres above the eye line and looking a few degrees down. As we have explained previously we also have to set the OptiTrack reference system. In our case we have defined the y-axis in vertical direction and the z-axis in the direction where the Kinect is looking.

The recordings were performed in the IRI laboratory (with the OptiTrack system assembled), as shown in Figure 28.

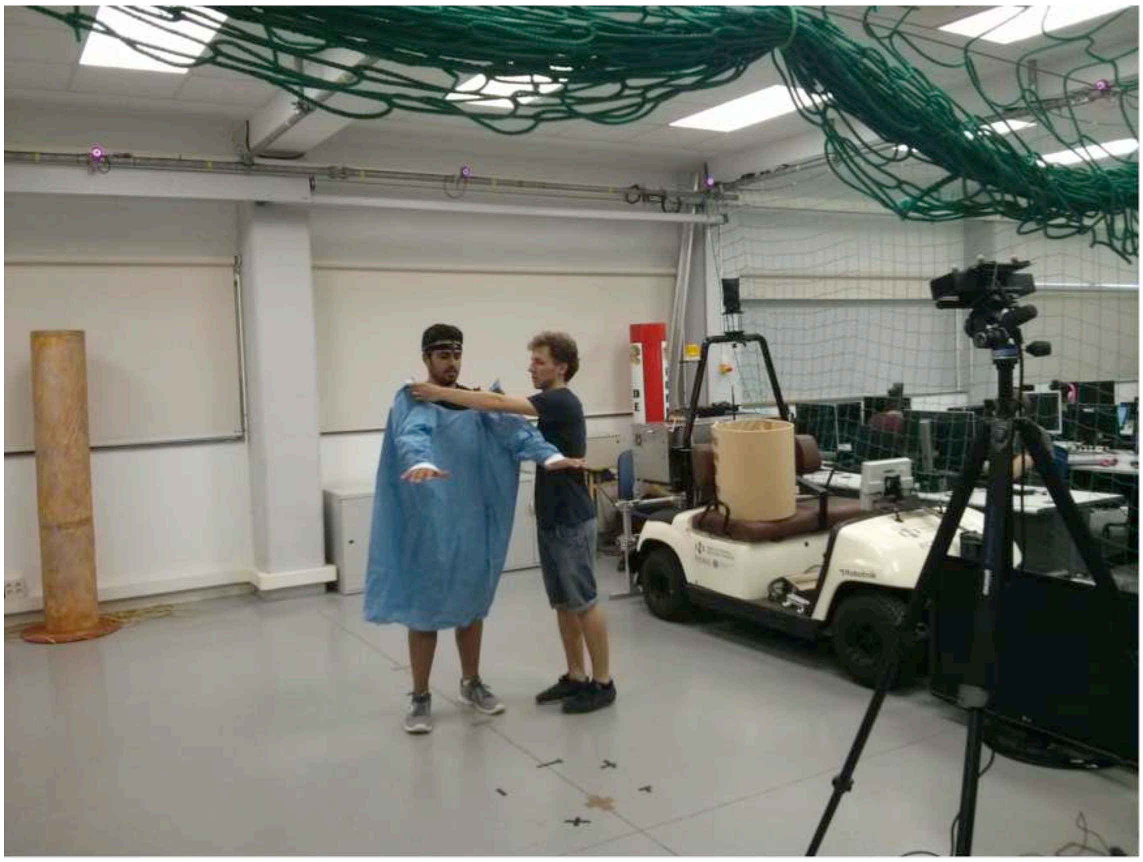


Fig.28: Arrangement of the elements in the laboratory, in which the placement of Kinect One camera is observed

One of the objectives of the project I-DRESS is to avoid the direct contact of the medical workers with their work clothes so the risk of infection is minimized. Therefore, it was considered to be appropriate to record sequences of movements when a person is assisted to change into a surgeon gown (Fig. 29).

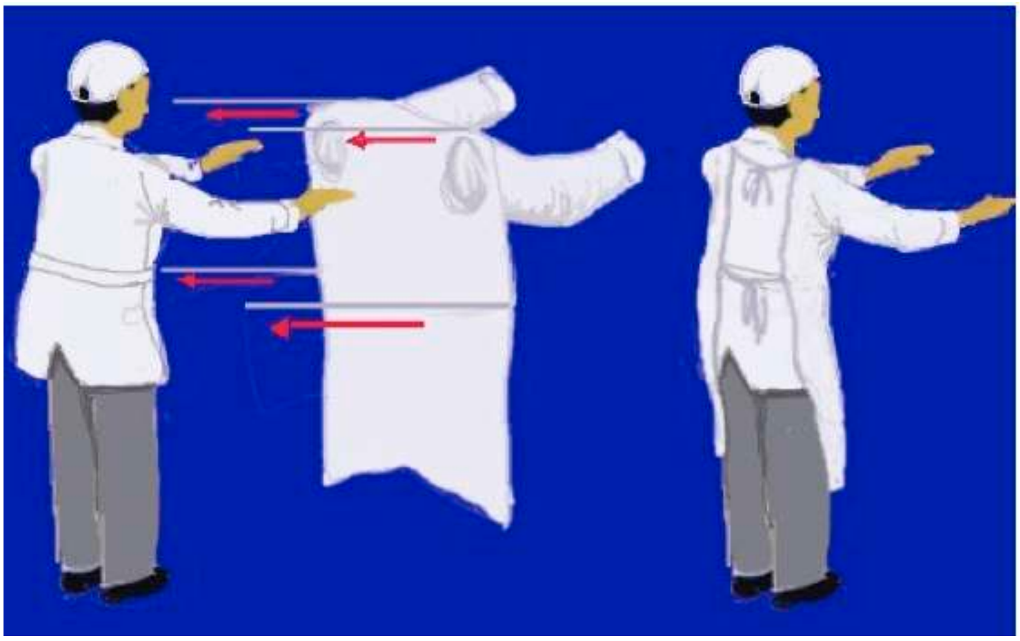


Fig.29: Person assisted wearing of a surgical gown

The sequences of movements of a person being assisted to put on a jacket will also be recorded. These sequences of movements will additionally be performed by models simulating that are being assisted to put these garments on, without using the garment because, by placing some clothes on the model, the markers are covered by it and the reflector positions are lost (Fig. 30). On the other hand, recordings made simulating the placement of the garments permit the viewing the position of all reflectors.

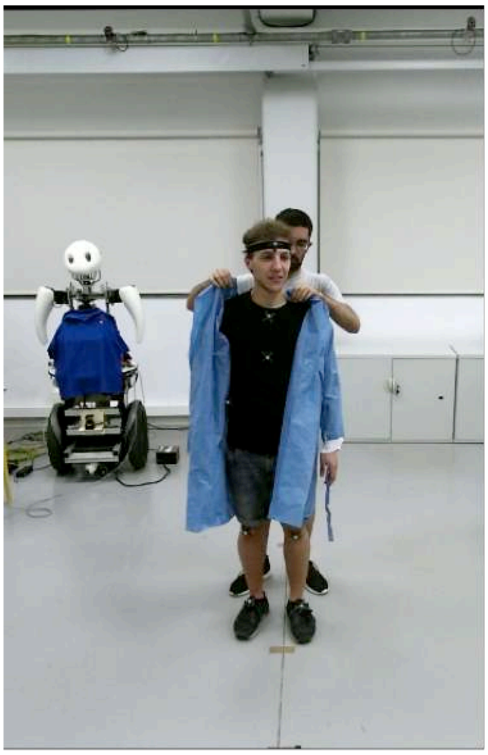
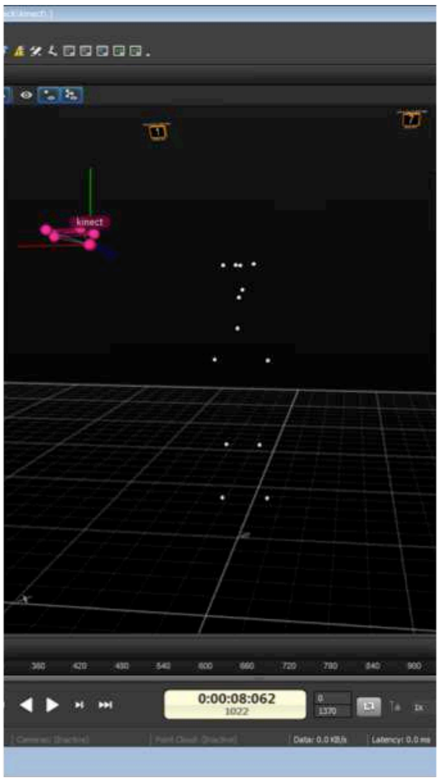



Fig 30: Example where we can see the oclusions in the OptiTrack when the jacket covers the arms

The following sequence of movements will be performed and recorded:

1. Random moves to validate (Fig. 31).

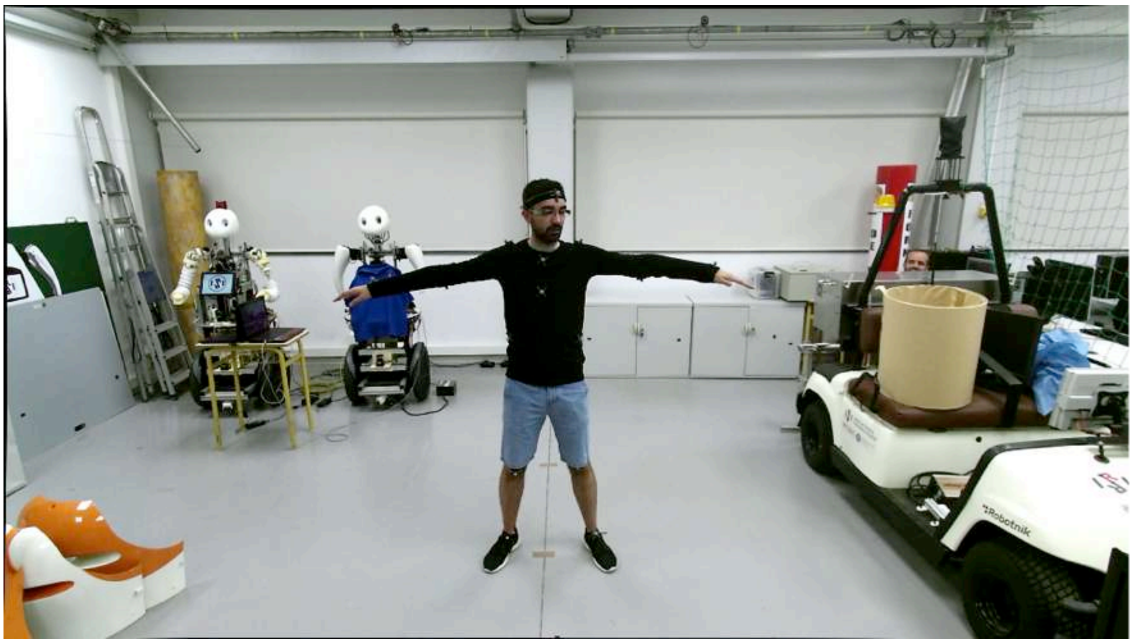


Fig.31: Random moves to validate

2. Person being assisted to wear a surgical gown:

   2a. A sleeve is first introduced to the shoulder and then the other sleeve.
   2b. The sleeve is introduced to the elbow and then the other, and finally the robe is placed in the correct position.
   2c. Both sleeves are placed at the same time (Fig. 32).

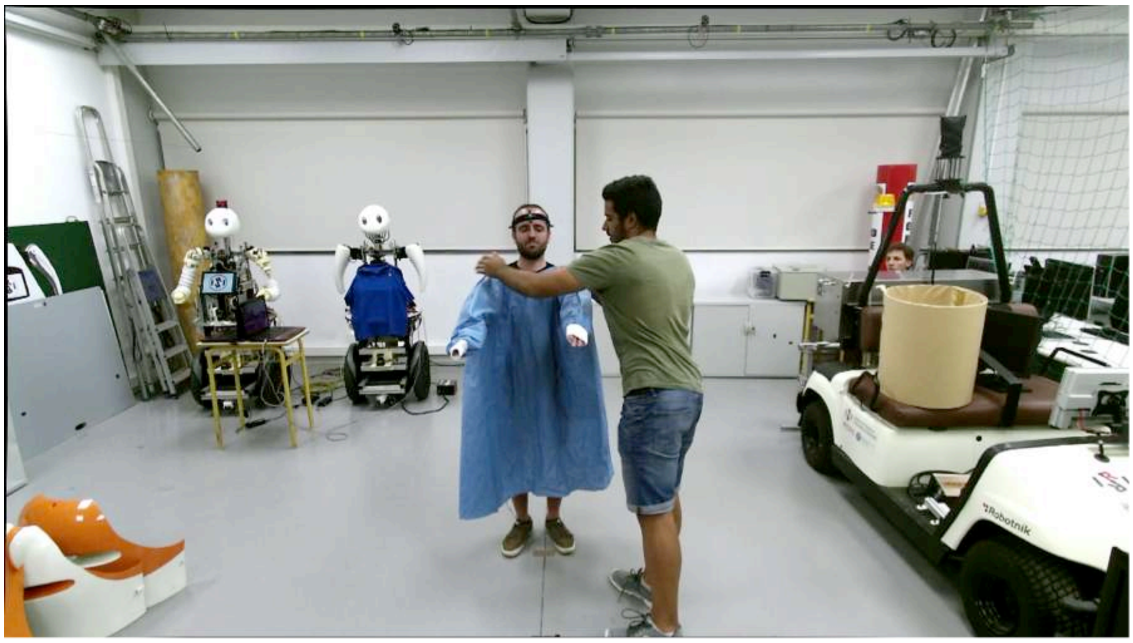


Fig.32: Person being assisted to wear a surgical gown (both sleeves are placed at the same time)

3. Person being assisted to wear a surgical gown, simulated movement (without the gown):

   3a. A sleeve is first introduced to the shoulder and then the other sleeve.
   3b. The sleeve is introduced to the elbow and then the other, and finally the robe is placed in the correct position.
   3c. Both sleeves are placed at the same time.

4. Person being assisted to put on a jacket:

   4a. A sleeve is first introduced to the shoulder, then the other sleeve (Fig. 33).

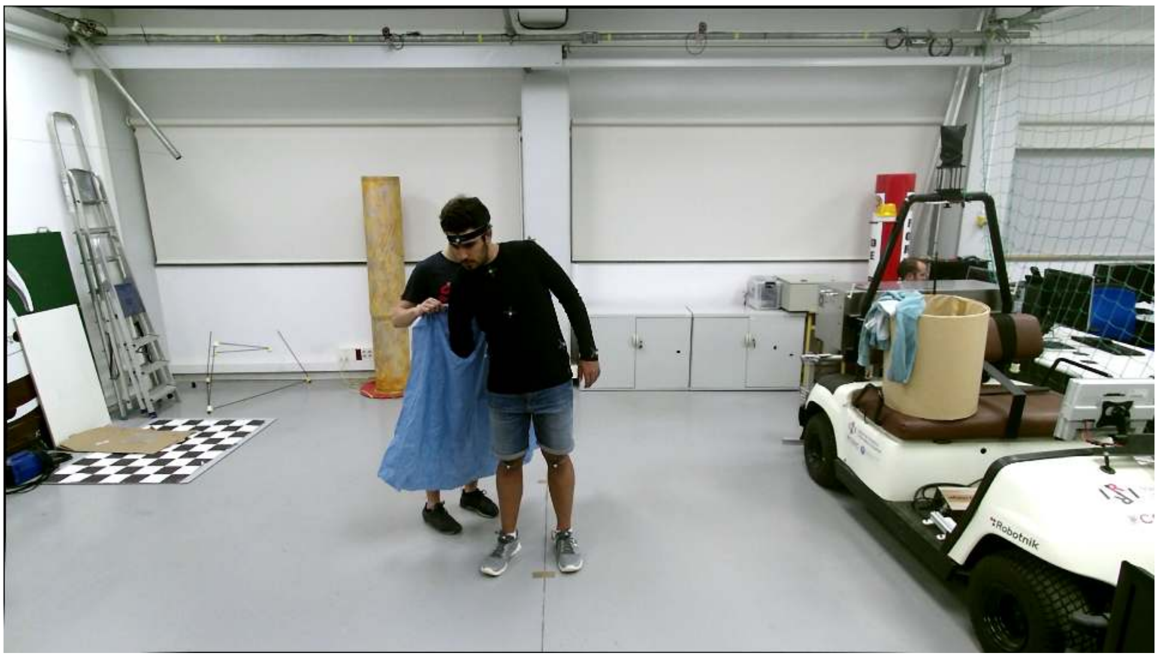


Fig.33: Person being assisted to put on a jacket (a sleeve is first introduced to the shoulder)

   4b. The sleeve is introduced to the elbow and then the other, and finally the robe is placed in the correct position.
   4c. Both sleeves are placed at the same time.

5. Person being assisted to wear a jacket, simulated movement (without the jacket):

   5a. A sleeve is first introduced to the shoulder and then the other sleeve.
   5b. The sleeve is introduced to the elbow and then the other, and finally the robe is placed in the correct position.
   5c. Both sleeves are placed at the same time.

8.- Recording and Data Proccesing

For the recording of the images two equipments have been used, each one (OptiTrack and Kinect) with its own control software: Motive:Tracker on Windows for the first and ROS (iai_kinect2 package) on Ubuntu 14.04 for the second. In this section, the procedure for the collection and processing of data is discussed. In particular, two aspects were considered: (a) The problem of synchronization between these recording systems, which is key for generating the database of images because the depth images of the human body captured by the Kinect and the instantaneous positions of reflectors (positioned on the body), registered by the OptiTrack system, must be univocally associated; and (b) We need to refer the images captured by the two devices to a single reference system (the one used by the Kinect).

8.1.- Data Recording

First of all, we initialize the Kinect with the previous explained command roslaunch kinect2_bridge kinect2_bridge.launch, but this time we will have to add at the end publish_tf:=true to get the tf information from the Kinect. We are not going to use it, but it could be useful for future uses of the data so we better record it (we will need the tf information to reproduce the point clouds in RViz for example). For the Kinect One camera recording, we begin recording in a rosbag the published messages in the following topics:

/tf, contains useful information, to reproduce the rosbag in rviz, for instance.

/kinect2/qhd/camera_info, contains information about the camera.

/kinect2/qhd/image_color_rect/compressed, contains the color images in a compressed format.

/kinect2/qhd/points, contains the point cloud data.

The drivers that we have installed for the Kinect One publish messages in many other topics, but we should try to record in the rosbag the minimum amount of information because the record process can exceed the buffer of the computer and it could start drop some messages while recording. That is why we record the compressed color images instead of the normal ones.

The software Motive:Tracker is used for the OptiTrack system and after editing we export the tracking data to a *.csv* file. The information for the edition has been obtained in the YouTube tutorial:

https://www.youtube.com/watch?v=iMPh7JhlCqc&index=4&list=PLdKrdVGpQ5OZSlQDCyYFT-iOo1Qmfzl7Z.

The images captured by the Kinect One camera must be perfectly associated with the map (instantaneous positions of reflectors) recorded by the OptiTrack system.

Some synchronization must be achieved between the two recording equipments, to ensure the adequate association of images between the two systems. We should be able, for example to associate the RGB and depth images from the Kinect and the map from the Optitrack (Fig. 34).

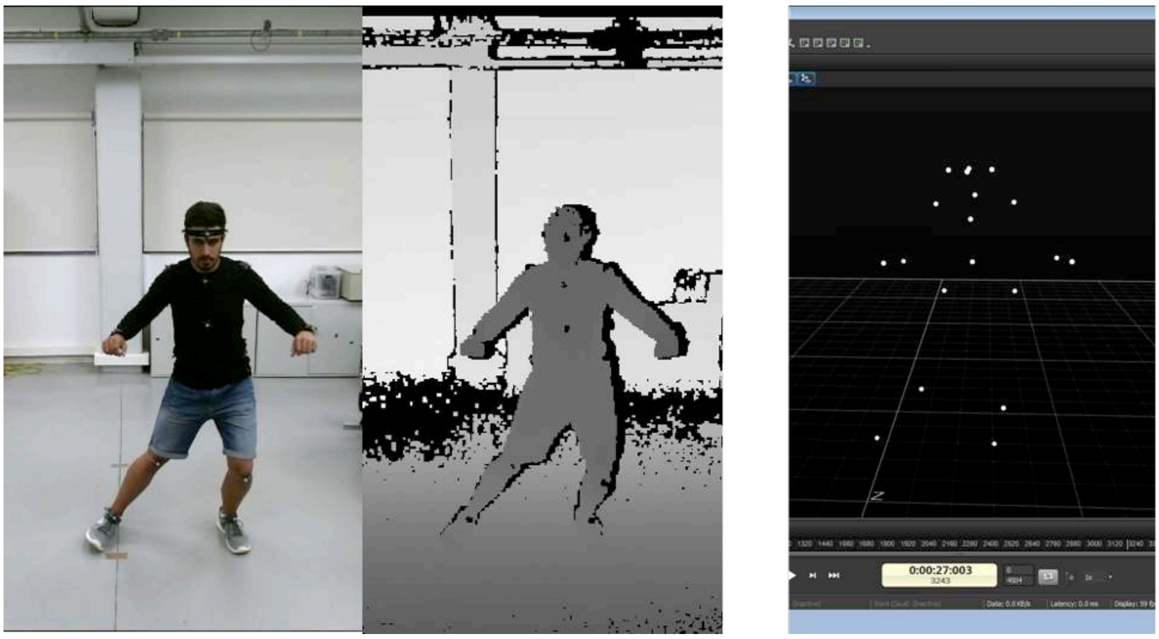


Fig.34: Associate images (RGB and Depth images of the Kinect and the "map" of OptiTrack)

To achieve this synchronization, two procedures have been considered: the first would be emiting some visible light signal that would associate the records in which said light signal appeared. However, since the recording frequency of both systems is not the same, this system would require some kind of interpolation performed in the intervals between the light signals. Also, the association of images would be fully manual and therefore quite toilsome.

The second method would consist in synchronizing the internal clocks of the computers controlling both recording equipments, and store the times when the Kinect and Optitrack's frames were taken. Since one computer uses the Windows operating system and the other is a Linux system, it is not possible to perform this synchronization by a function and it is necessary to connect both computers to an external server to synchronize the internal clocks of these computers with this external server.

The synchronization to external server is done in the case of Windows system using the system settings and in the case of Ubuntu system running in terminal the following commands:

```
sudo apt-get install ntp
sudo ntpdate -u server_name
```

We should do this before start recording the sequences to ensure the clocks of the computers controlling the recording equipment is synchronized. Once the computers are synchronized and the setup is ready we can start recording.

A record time for each of the recordings to be performed must be obtained. In ROS, these time records are obtained by the timestamp function that makes that each message has associated a recording time (with millisecond accuracy).

However, a problem arises because the timestamps of ROS are lost when exporting point clouds to format *.pcd*. Therefore we must implement a procedure to extract and store the timestamps associated with each of the point clouds recorded by the Kinect One camera (which, in any case, are recorded and ordered chronologically automatically). This procedure will be described in the following subsection.

OptiTrack system records the time (with millisecond accuracy) when the recording starts and for each data file, the elapsed time (also with millisecond accuracy) from this starting time to the time of recording. Therefore, in this case, each data file with the records of the instantaneous positions of the markers will be associated with its corresponding timestamp.

Another important aspect to consider in the image processing is related to the transformation of coordinates collected by the OptiTrack system (which are referred to their own coordinate system) to the system coordinates of the Kinect camera. If we want to match the Optitrack data with the depth data recorded by the Kinect we should be able to change the reference frame. To do this, we must place some reflectors on the Kinect camera, so that it can be tracked by the OptiTrack system as a rigid body. Thus, it controls the evolution of the central position of the reflectors and a centroid that can define freely. We can also set a reference system of the Kinect camera whose coordinate origin is at the centroid and whose orientation can be set conveniently.

That is, we must define the reference system (coordinate origin and orientation of the coordinate axes) that the Kinect camera uses, from the point of view of the reference system used by the Optitrack, and thus the Optitrack can track the position and orientation of the Kinect system and make the coordinate transformation that allows us to obtain the instantaneous positions of the reflectors in the reference system used by the Kinect camera.

The origin of coordinates is in the infrared sensor and the orientation of the axes is shown in Fig 35.

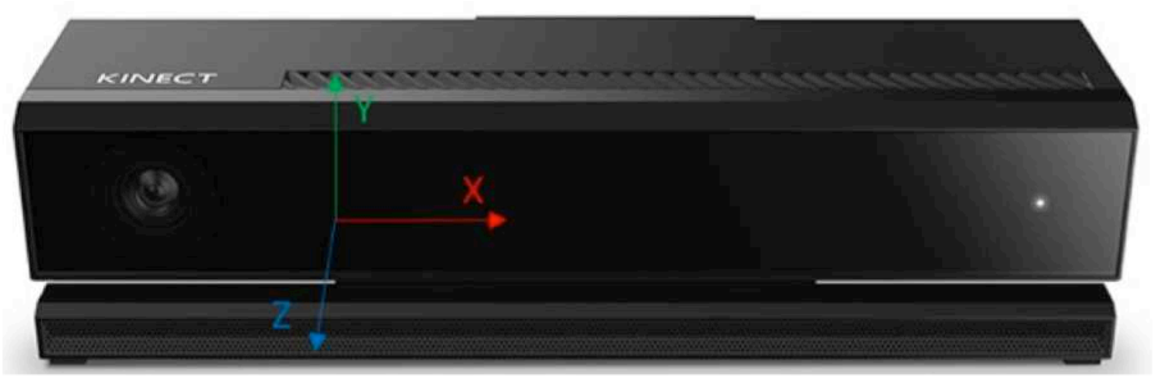


Fig.35: Reference system of the Kinect One camera

We place the centroid and the orientation of the reference system of the Kinect in the Optitrack as close as possible to the one shown in Fig. 35, as shows in the Fig. 36.

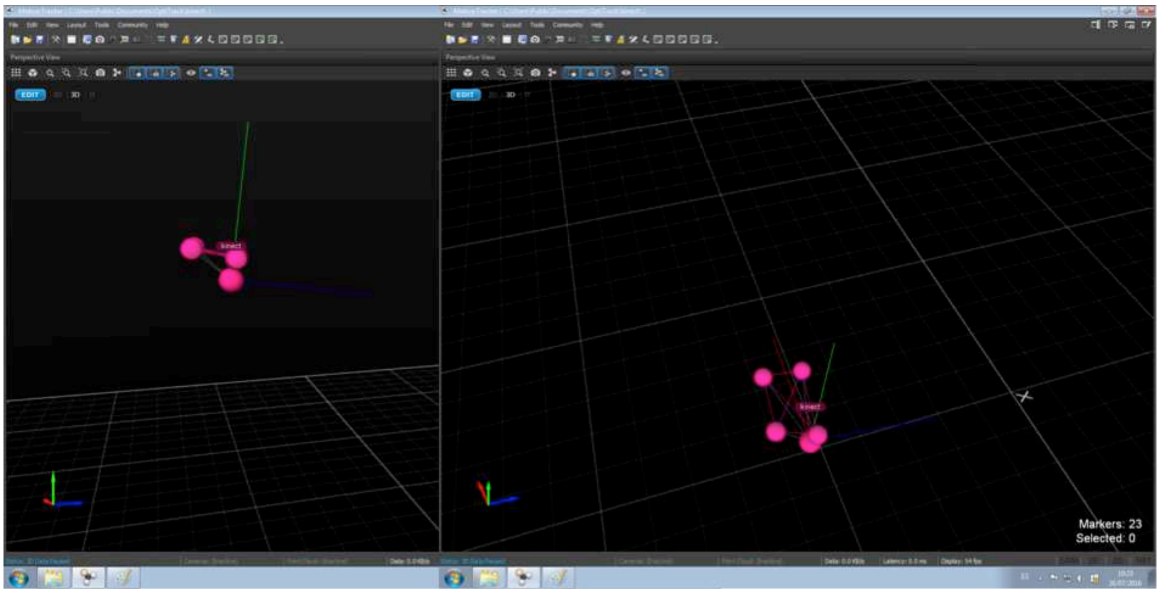


Fig.36: Screenshot of the OptiTrack

The orientation reference system OptiTrack system (Fig. 36) is shown in the bottom left corner, representing the cross on the right image, the origin. The body we see in the image corresponds to the Kinect camera, and the axes defined in the image on the left represent own reference system of the OptiTrack.

Once we have taken into account all this issues we can start recording the sequences. As a summary, the procedure that we followed to record is shown in Figure 37.

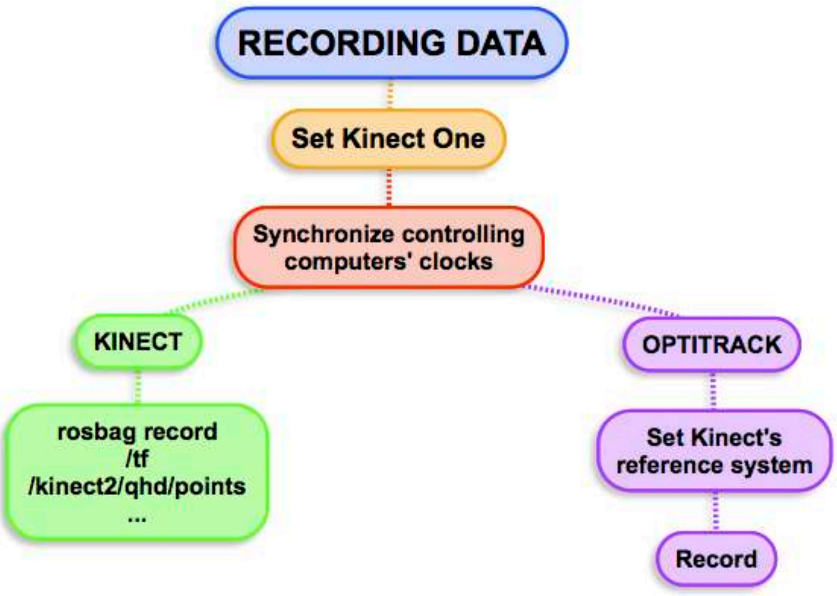

RECORDING DATA
Set Kinect One
Synchronize controlling computers' clocks
KINECT
rosbag record /tf /kinect2/qhd/points ...
OPTITRACK
Set Kinect's reference system
Record

Fig.37: Steps for recording data

We will need four people in the lab to perform the recondings, with two persons acting as models, one with markers placed on the body and the other assisting in the placement of the garments, and the two other people to start and log synchronously in the two control computers recording equipment. Four shots for each sequence are made to have more data available for the training algorithms. All the people in the laboratory will successively be acting as a model.

8.2.- Data Processing

Once the files of the data captured by both systems are stored in their corresponding control computers with a time stamp, it is necessary to establish the correspondence between timestamps. Since the OptiTrack system obtains captures with higher frequency than the Kinect camera, to create the database that associates each point cloud obtained with the Kinect camera, we will relate each Kinect One frame with the OptiTrack frame that is closest in time.

To carry out this image association, a Matlab script has been written ("sincroniza.m"). This program will be described later in this same subsection.

A possible process for extracting timestamps in ROS uses the following command:

```
rostopic echo topic_name/header -p > file_name.txt (o .csv)
```

The command rostopic echo topic_name reproduces the types of messages that are being published on the topic indicated; /header gives the timestamps, sequence number and id; -p reproduces the data in a format that can be used for Matlab reading; > file_name.txt (o .csv) indicates that all information will be stored in the file indicated.

Once the above command is ran, we execute:

```
rosbag play file_name.bag
```

to play the rosbag and start publishing messages whose time stamp and dump data we want to extract to a file.

However, this method has the disadvantage that the computer has to process too much information, and sometimes it gets stuck and skips messages, missing timestamps of some point clouds randomly. It is possible to get the rosbag to play slower with the following command:

```
rosbag play file_name.bag –r
```

being r the factor by which to multiply the playing speed.

In this way we do not lose information. However, the process is slow and it does not inspire much confidence that for high speeds it can lose information randomly. Therefore, we have decided that it is best to implement a program in Python that allows us to extract information on timestamps to a text file.

The code for this program (extract_time.py) is as follows:

```
#!/usr/bin/env python
import rosbag
import datetime
bag = rosbag.Bag('bag_name.bag')
print "file_name sequence timestamp date hour frame_id"
for topic, msg, t in
bag.read_messages(topics=['/kinect2/qhd/points']):
    date =
datetime.datetime.fromtimestamp(msg.header.stamp.to_sec(
)).strftime("%Y-%m-%d %H:%M:%S.%f")
    n=0
    a=''
    for i in str(msg.header.stamp):
    a=a+i
    if n==9:
        a=a+'.'
    n=n+1
    print a, msg.header.seq, msg.header.stamp, date,
msg.header.frame_id
bag.close()
```

With this software we get the timestamp from each message being published in the topic /kinect2/qhd/points, the topic from we extract the point clouds in *.pcd* format with pcl_ros ROS package. We also extract some other adittional information that we are not going to use, but that could be useful for further use of the data. Then we convert the timestamp to date format and also use it to get the point cloud file name associated to that frame (we put a point after the tenth number of the timestamp to get the file name, what is what the ROS package pcl_ros does to write the names of the point clouds that it extracts from the rosbag).

We execute the program and write the timestamps and the file names associated to each frame to a file by running the command /extract_time.py > file_name.txt (or .csv, depending on the desired format). If the file extract_time.py is not in the current directory we should type the path to the file instead of ./ The python program extract_time prints the output of the program on the terminal and by typing > file_name.txt we write it to a file.

And the file format of the file where the time stamps of the images are recorded by the camera Kinect is as follows:

```
      file_name        sequence     timestamp             date        hour                  frame_id
1469527676.245373344 1298 1469527676245373344 2016-07-26 12:07:56.245373 kinect2_rgb_optical_frame
1469527676.365348701 1299 1469527676365348701 2016-07-26 12:07:56.365349 kinect2_rgb_optical_frame
1469527676.608541970 1300 1469527676608541970 2016-07-26 12:07:56.608542 kinect2_rgb_optical_frame
```

Then we extract the point clouds to *.pcd* format with the library PCL. Using the program extract_time.py we extract the timestamp of each point cloud to the files *.csv* and *.txt*. As we mentioned in the previous subsection, the Optitrack system registers the time of each frame and it is also exported with the tracking data.

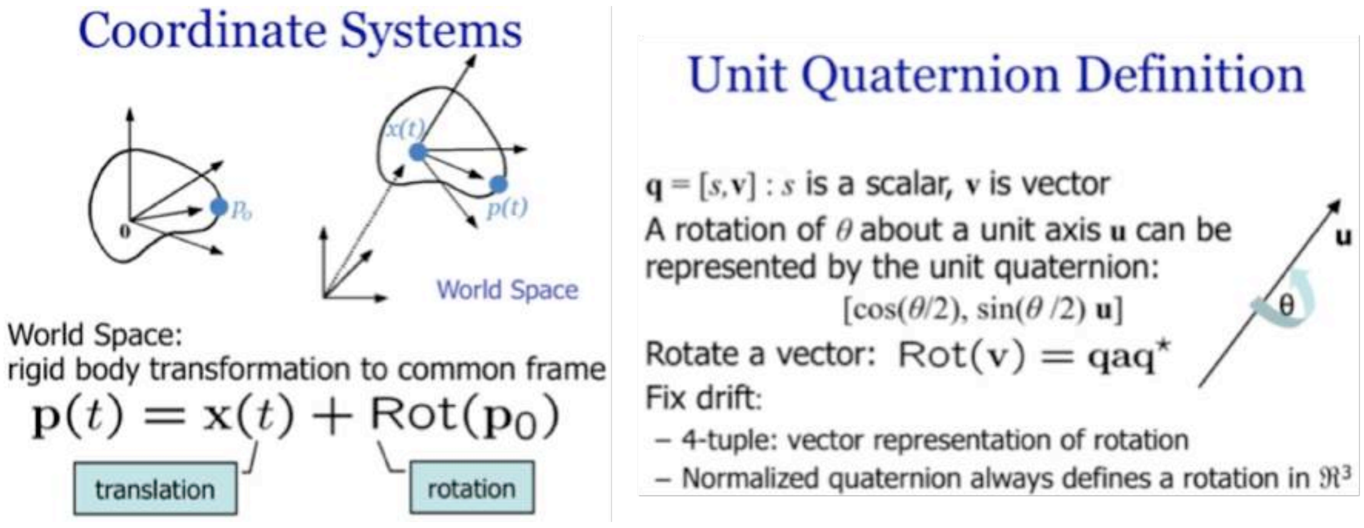
It remains to be done the association (correspondence) between the point clouds captured by the Kinect One camera and the data of the instant marker positions ("maps") recorded by the OptiTrack. As the OptiTrack registers frames with a 120Hz frequency, much higher that the Kinect camera, we will associate to each Kinect camera image the OptiTrack frame closest in time.

Additionally, the coordinates obtained by the OptiTrack system can be then transformed to the reference system of the Kinect. The reference system using Kinect itself is indicated at the web address https://msdn.microsoft.com/en-us/library/dn785530.aspx.

In the data file exported from OptiTrack system we will get the centroid position (origin) and the orientation of the axes of the reference system, expressed by the corresponding quaternion, of the Kinect camera. That is, at every moment, the OptiTrack system determines the quaternion and position of the centroid of the system we have attached to the Kinect camera (which is controlled by Motive:Tracker as a rigid solid). The software Motive:Tracker reports the rotation values as both quaternions, and as roll, pitch, and yaw angles (in degrees).

Quaternions are a four-dimensional rotation representation that provide greater mathematical robustness by avoiding "gimbal" points that may be encountered when using roll, pitch, and yaw (also known as Euler angles). However, quaternions are also more mathematically complex and are more difficult to visualize.

The coordinates obtained by the OptiTrack system, in its reference system for different markers at each moment, can be then transformed to the reference system of the Kinect. This transformation is performed for each of the "map" of instantaneous positions of the markers recorded by OptiTrack system using as data to perform this transformation the coordinates of the centroid position (which determines the translation between the two reference systems) and the quaternion (which determines the rotation between the two reference systems) linked to Kinect camera, as described in Fig. 38.



World Space:
rigid body transformation to common frame

$$\mathbf{p}(t) = \underbrace{\mathbf{x}(t)}_{\text{translation}} + \underbrace{\mathrm{Rot}(\mathbf{p}_0)}_{\text{rotation}}$$

Unit Quaternion Definition

$\mathbf{q} = [s, \mathbf{v}]$: s is a scalar, $\mathbf{v}$ is vector

A rotation of θ about a unit axis $\mathbf{u}$ can be represented by the unit quaternion:

$$[\cos(\theta/2), \sin(\theta/2)\ \mathbf{u}]$$

Rotate a vector: $\mathrm{Rot}(\mathbf{v}) = \mathbf{q}\mathbf{a}\mathbf{q}^*$

Fix drift:

- 4-tuple: vector representation of rotation
- Normalized quaternion always defines a rotation in $\Re^3$

Fig.38: **Transform World-Space (OptiTrack) Coordinates to Local Rigid Body Coordinates (Kinect)**

We have written a Matlab script, "sincroniza.m", that performs the following tasks:

1. It stablish as time zero the Optitrack timestamp and transforms the times of the Kinect file to this scale. After this, the frames of the Optitrack and the images of the Kinect camera are in the same time scale and can be paired.

2. We only consider the span of time where both systems (OptiTrack and Kinect) have been recording. We pair the frames (OptiTrack) and the images (Kinect) with the criteria that they are the closest in time. As there are many more frames than images, most of the frames will be discarded while, usually, all the images are retained. This way, we associate with each image (point cloud) of the Kinect a map of marker positions recorded by the OptiTrack.

3. Using the rotating quaternion and the centroid position, provided in the OptiTrack file, we will transform the coordinates, from the reference system of the OptiTrack to the reference system of the Kinect One camera.

4. We write four files as output:

   a. Two *.csv* files that contain, in the first column, the name of the Kinect One point cloud file and, in the other columns, the corresponding coordinates of the markers in:

      i. The OptiTrack reference system (name_out_Optitrack_ref.csv)
      ii. The Kinect One camera reference system (name_out_Kinect_ref.csv).

   b. Two *.mat* files that contain the same information than the previous ones and are named in the same way (but with *.mat* extension).

The workflow of data processing is shown in Figure 39.

After recording the data we will have: (a) on the side of the Kinect a rosbag file in *.bag* format for each take; and (b) on the side of the OptiTrack a *.tak* file readable with the Motive:Tracker software with the recorded sequence no edited (markers unlabelled, etc.).

From the rosbag file we should get: the point clouds; the color and depth images; and a data file with the timestamps of the point clouds. To extract the timestamps we use extract_time.py as explained previously and we will get the desired data file.

For extracting the point clouds we will use the ROS node bag_to_pcd from the pcl_ros package. We will extract them into a folder in *.pcd* format (we choose the name of the output folder when running the node).

After getting the point clouds, we can get the depth images and color images running the function pcl_pcd2png from the PCL library (for the usage information, run pcl_pcd2png–h).

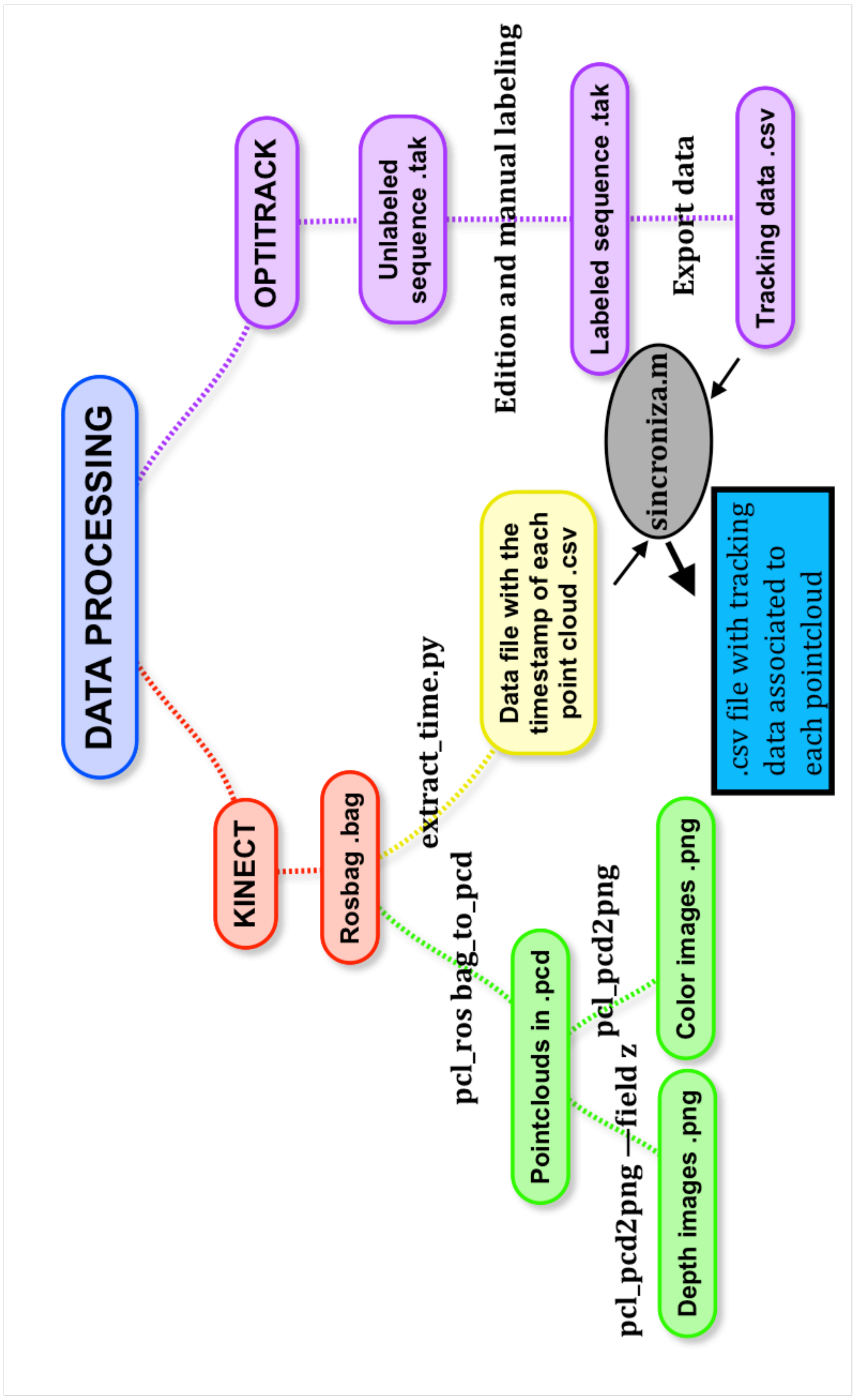


Fig.39: Workflow of data processing

We have to do this for each rosbag file and it can be tedious. For accelerating and making the process more efficient we have wrote some executable scripts *.sh*. Firstly we wrote a script which we called pcd2png.sh for extracting the color images and the depth images from the folder where we have the point clouds, to two other different folder in the same directory:

```
#!/bin/bash
mkdir depth_images
mkdir color_images
cd  s
for f in *; do
pcl_pcd2png --field z "$f"
"../depth_images/${f%.*}_depth.png"
pcl_pcd2png "$f" "../color_images/${f%.*}_color.png"
done
```

The script extracts the color and depth images, in *.png* format, from the folder pointclouds to two folders called color_images and depth_images respectively. The script calls the images with the same name that the point clouds from where they were extracted, adding the extension _color for the color images and the extension _depth for the depth images.

Then we made another script, which we called bag_to_pcd.sh, where we first move to the directory where the rosbag is, and then we run the ROS node bag_to_pcd to extract the point clouds from the rosbag. Then, we execute the program extract_time.py that we have pasted in the same directory of the rosbag and put the *.bag* name in the python script to get the timestamps data file. And, finally, we call the script that we have explained previously to get the images from the point clouds and return to the home directory again. We will do this for each rosbag. Here are some lines of the script to extract all the files from a single rosbag:

```
cd /home/marduengo/dataset/data/enric/bata/mov1/1/kinect
rosrun pcl_ros bag_to_pcd 2016-07-26-12-07-55.bag
/kinect2/qhd/points pointclouds
./extract_time.py > time.txt
./extract_time.py > time.csv
/home/marduengo/dataset/programs/pcd2png.sh
cd
```

In the OptiTrack side, we have the non-edited sequence. We have to edit it, as explained in section 5, and then export the tracking data.

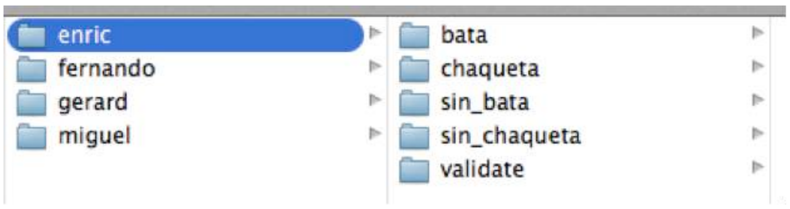
Once we have the file with the point cloud timestamps and the file with the tracking data, we use the Matlab script sincroniza.m, explained previously, to match the Kinect One data with the OptiTrack data.

9.- Results: Database

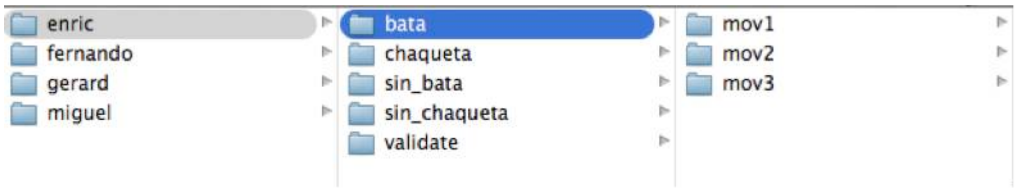
The database is organized as follows:

1. We created a folder for each model person. The folder has the name of the model whose data it contains.

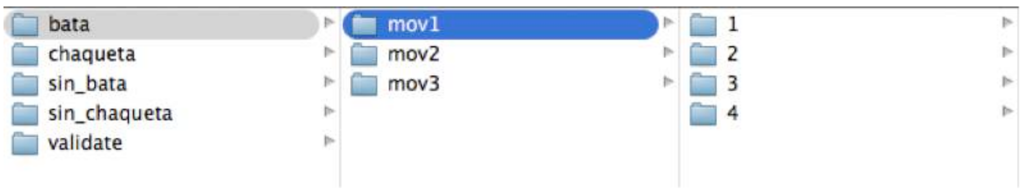
2. Inside the folder of each model we will have five different folders, one for each movement sequence. If we have a look to section 7 the folder validate corresponds to sequence 1, the folder bata (gown) corresponds with sequence 2, the folder sin_bata (without gown) corresponds to sequence 3, the folder chaqueta (jacket) corresponds to sequence 4 and the folder sin_chaqueta (without jacket) corresponds to sequence 5.



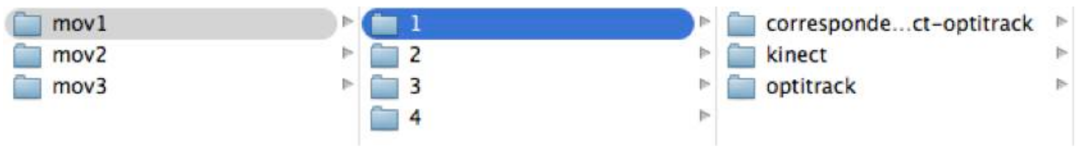
3. Inside of the folder of each movement sequence there will be three other folders. Each one of them corresponds to variations of the sequence. If we go back again to section 7, the folder called mov1 corresponds with "a sleeve is first introduced to the shoulder and then the other sleeve", the folder called mov2 corresponds with "the sleeve is introduced to the elbow and then the other, and finally the robe is placed in the correct position" and the folder mov3 corresponds with "both sleeves are placed at the same time".



4. Inside each mov folder we have usually four folders (sometimes five). Each folder corresponds to one take. In order to have more data we repeated the same movement four or five times. We called the first take 1 and the last take 4 (or 5).



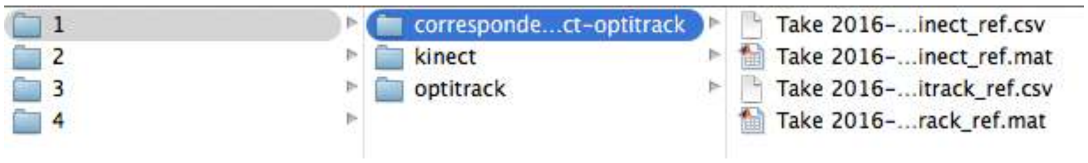
5. Inside each take we have three folders: (a) the one called kinect contains the data recorded and extracted from Kinect; (b) the folder called optitrack contains the data recorded and extracted from the OptiTrack; and (c) the last one, called correspondence_kinect-optitrack, contains the files that we got after executing the Matlab program sincroniza.m.


mov1
mov2
mov3
1
2
3
4
corresponde...ct-optitrack
kinect
optitrack

Now we will explain one by one the content of the last folders.

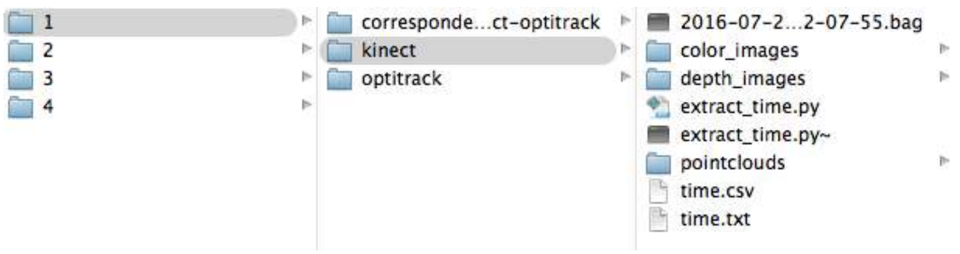
- correspondence_kinect-optitrack

  This folder contains four files, with the same name as the OptiTrack tracking data file that was used as input in sincroniza.m. It has: (a) a *.csv* file, where it is associated each point cloud with its correspondent OptiTrack frame (in the OptiTrack reference system); (b) another *.csv* file, where it is associated each point cloud of the take with its correspondent OptiTrack frame (in the Kinect reference system); (c) and (d) the above files in format *.mat*.


1
2
3
4
corresponde...ct-optitrack
kinect
optitrack
Take 2016-...inect_ref.csv
Take 2016-...inect_ref.mat
Take 2016-...itrack_ref.csv
Take 2016-...rack_ref.mat

- kinect

  This folder contains several files with the raw data captured by the Kinect after being processed. It contains:


1
2
3
4
corresponde...ct-optitrack
kinect
optitrack
2016-07-2...2-07-55.bag
color_images
depth_images
extract_time.py
extract_time.py~
pointclouds
time.csv
time.txt

a) rosbag: Here is all the information. This is the first file we get when recording with the Kinect One. From here we extract the information we want. The file is the one with the extension .bag and its name indicates when it was recorded.

b) extract_time.py: We have pasted the program here and substituted bag_name for the name of the rosbag in the folder for calling him and extract the timestamps with the script explained previously bag_to_pcd.sh.

c) Data files with the timestamps: We have two files called time with two different extensions, one in *.txt* and the other one in *.csv*. Here we have the correspondence between each point cloud and its associated timestamp. We use this files to match the Kinect data with the Optitrack data.

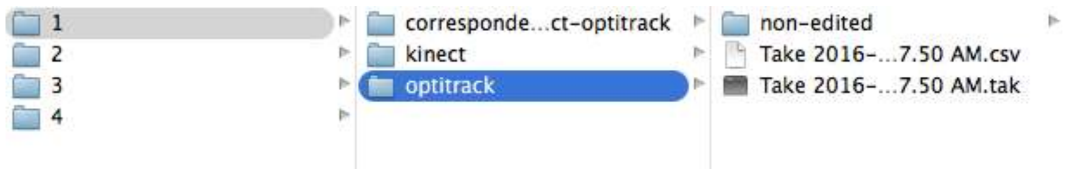
d) Point clouds: We have a folder called pointclouds where we have stored all the point clouds of the take after extracting it from the rosbag.

e) Color images: We have a folder called color_images where we have stored all the color images that we extracted from the point clouds.

f) Depth images: We have a folder called depth_images where we have stores all the depth images that we extracted from the point clouds.

- optitrack

This folder contains several files with the "raw" data extracted from the OptiTrack and after being processed. It contains:



a) Non-edited OptiTrack file: We have a folder called non-edited where we have stored the *.tak* file readable by the software Motive:Tracker before being edited.

b) Edited Optitrack file: We have a *.tak* file readable by the software Motive:Tracker that contains the recorded sequence after being edited. The name of the file corresponds to the date and time on which the sequence was recorded.

c) Tracking data: We have a *.csv* file that we exported from the edited *.tak* file. It contains the tracking data. The name of the file is the same as the OptiTrack file from where it was exported.

The database organization is described in Figure 40.

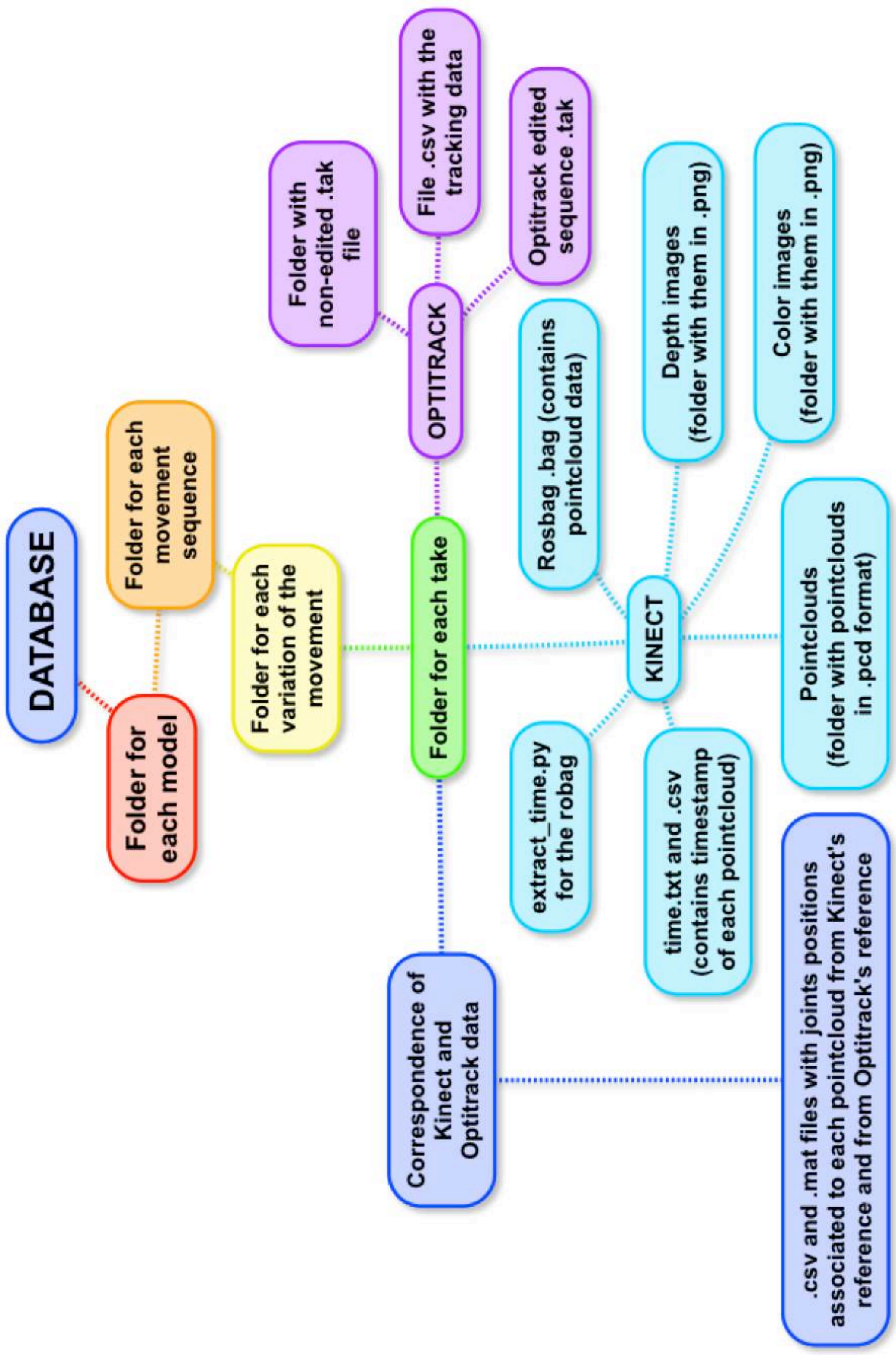

DATABASE
Folder for each model
Folder for each movement sequence
Folder for each variation of the movement
Folder for each take
OPTITRACK
Folder with non-edited .tak file
File .csv with the tracking data
Optitrack edited sequence .tak
KINECT
Rosbag .bag (contains pointcloud data)
Depth images (folder with them in .png)
Color images (folder with them in .png)
Pointclouds (folder with pointclouds in .pcd format)
extract_time.py for the robag
time.txt and .csv (contains timestamp of each pointcloud)
Correspondence of Kinect and Optitrack data
.csv and .mat files with joints positions associated to each pointcloud from Kinect's reference and from Optitrack's reference

Fig.40: Database organization

10.- Conclusions

This work is part of the project I-DRESS **[8]**, within the section (Work Packages) called WP2: "Perception and Tracking", that must be carried out by IRI**,** and it has developed under the specific target for the detection of human body postures and track their movements.

The aim of this work is to create a database (depth images and maps of snapshots positions of markers strategically placed on the human body), based on images obtained by a sensor Time-of-Flight (ToF) depth camera*s* type, such as the incorporated by the Kinect One (Kinect v2), and by the OptiTrack system.

For the recording of the images describing sequences of movements made by the models, two equipments have been used: the OptiTrack system and the Kinect One camera, each with its own control software: "Motive:Tracker" on Windows for the first and ROS (iai_kinect2) on Ubuntu 14.04 for the second.

In this report we explained:

- The experimental setup, the placing of the markers used to determine the position of the human body and the movements performed in the different sequences are described.

- The procedure for the collection and processing of data is discussed. We have paid special attention to two aspects: (a) The problem of synchronization between the recording systems, which is key for generating the database of images because the depth images of the human body captured by the Kinect and the instantaneous positions of reflectors positioned on the body, registered by the OptiTrack system, must be univocally associated; and (b) The need to refer the images captured by the two devices to a single reference system.

- We have organized data logging to meet these requirements, and implemented specific software for data processing.

- This way, we have created a database consisting of sequences of images, which have associated with each depth image of the Kinect a map of instantaneous positions of markers recorded by the OptiTrack, both referring to the same reference system (the reference system used by the Kinect).

- Finally, the image database organization is described. This database should be useful for the training of the algorithms of pose estimation for the artificial vision of the robotic system.

11.- References

1. Aguado, A. (2015), 'Reconocimiento de posturas mediante Kinect en ROS', Technical report, Facultad de Informática, Universidad del País Vasco, Bilbao, Proyecto Fin de Grado.

2. Alenyà, G. (2016), 'I-DRESS Assistive interactive robotic system for support in dressing''2016 CHISTERA meeting', Bern.

3. Cocchi, E. & Sibellas, M. (2015), 'Contrôle d'une caméra Kinect par le middleware ROS', Technical report, ISIMA – ISIBOT (Club de Robotique), Clermont Ferrand (France), Project d'etudes.

4. Craig, J. J. (2006), *Robótica (3ª Edicion)*, Prentice Hall Mexico.

5. Fankhauser, P.; Bloesch, M.; Rodríguez, D.; Kaestner, R.; Hutter, M. & Siegwart, R. (2015), Kinect v2 for mobile robot navigation: Evaluation and modeling, *in* 'Advanced Robotics (ICAR), 2015 International Conference on', pp. 388-394.

6. Foix, S.; Alenyà, G. & Torras, C. (2011), 'Lock-in Time-of-Flight (ToF) cameras: A Survey', *IEEE Sensors Journal* **11**(9), 1917-1926.

7. Haque, A.; Peng, B.; Luo, Z.; Alahi, A.; Yeung, S. & Li, F.-F. (2016), 'Viewpoint invariant 3D human pose estimation with recurrent error feedback', .

8. IRI, 'I-DRESS', https://www.i-dress-project.eu/.

9. Joseph, L. (2015), *Learning Robotics using Python*, Packt Publishing.

10. Joseph, L. (2015), *Mastering ROS for Robotics Programming*, Packt Publishing.

11. Laukkanenn, M. (2015), 'Performance Evaluation of Time-of-Flight Depth Cameras', Technical report, School of Electrical Engineering, Aalto Universiy, Otaniemi (Finland), Master's Thesis.

12. Osorio, O. P. & Peña, F. L. (2015), 'Captura de movimiento utilizando el Kinect para el control de una plataforma robótica controlada de forma remota por medio de seguimiento de los puntos de articulación del cuerpo', Technical report, Universidad Tecnológica de Pereira, Colombia, Trabajo de Fin de Grado, Programa de Ingeniería Electrónica.

13. Paniagua, F. S. I. (2011), 'Object Recognition using the Kinect', Technical report, ISIMA – ISIBOT (Club de Robotique), School of Computer Science and Communication, Stockholm (Sweden), Master's Thesis in Computer Science.

14. PCL (2016), 'Point Cloud Library', http://pointclouds.org/documentation/, Accesed July 2016.

15. Peng, B. & Luo, Z. (2016), 'Multi-view 3D pose estimation from single depth images', Technical report, Stanford University, USA, Report, Course CS231n: Convolutional Neural Networks for Visual Recognition.

16. Quigley, M.; Gerkey, B. & Smart, W. D. (2015), *Programming Robots with ROS*, O'Reilly Media.

17. ROS (2016), 'Robot Operating System Documentation', http://wiki.ros.org/, Accesed July 2016.

18. Rusu, R. B. & Cousins, S. (2011), 3D is here: Point Cloud Library (PCL), *in* 'IEEE International Conference on Robotics and Automation (ICRA)'.

19. Rviz (2016), 'Rviz Tutorial', http://wiki.ros.org/rviz/Tutorials, Accesed July 2016.

20. Seggers, R. (2015), 'People tracking in outdoor environments: evaluating the Kinect 2 performance in different lighting conditions', Technical report, Faculty of Science, University of Amsterdam, Amsterdam (Holland), Bachelor Thesis.

21. Sell, J. & O'Connor, P. (2014), 'The Xbox One system on a chip and Kinect sensor', *IEEE Micro* **34**(2), 44-53.

22. Shafaei, A. & Little, J. J. (2016), Real-time human motion capture with multiple depth cameras, *in* 'Proceedings of the 13th Conference on Computer and Robot Vision'.

23. Tf (2016), 'Tf Library', http://wiki.ros.org/tf/Tutorials, Accesed July 2016.

24. Wiedemeyer, T. (2014-2016), 'IAI Kinect2', Institute for Artificial Intelligence, https://github.com/code-iai/iai_kinect2, University Bremen, Accessed July, 2016.

25. Zennaro, S. (2014), 'Evaluation of Microsoft Kinect 360 and Microsoft Kinect One for robotics and computer vision applications', Technical report, Ingegneria Informatica. Università degli Studi di Padova, Italy, Magistrali Biennali.