Software Architecture Project

Students: Alberto Grillo, Claudio Curti, Francesca Cantoni
Tutor: Alessandro Carfi
Supervisor: Alessandro Carfi
Year: 2018 - 2019

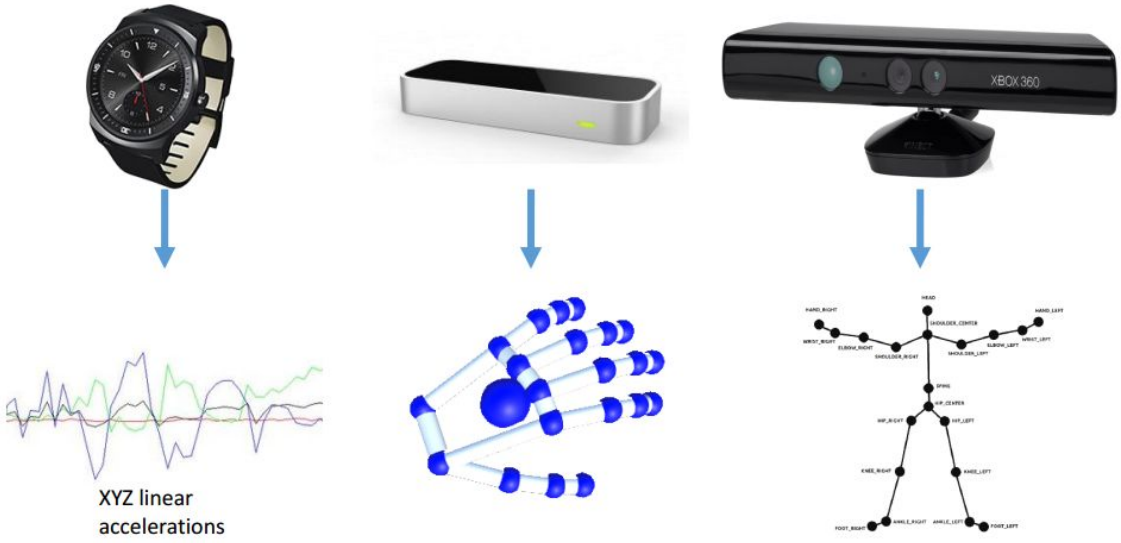
Mobile Robot Teleoperation using Kinect sensor

Objective of the Project **2**
HARDWARE 3
Kinect V1 3
SOFTWARE TOOL 5
Rviz 5
The System's Architecture **6**
Kinect Module Overall Architecture 6
Description of the Module 6
Implementation **8**
Prerequisites 8
OpenNI 8
NITE 9
openni_tracker 10
How to install the project 11
Download/install required libraries and packages 11
Build the ROS workspace 13
How to run the project 13
Rviz setup and visualization 13
Algorithms 15
Results **17**
Recommendations **19**

1. Objective of the Project

The aim of this project is to integrate different modules in order to obtain an architecture that allows the user to teleoperate the desired mobile robot via the motions of one arm.
The acquisition of movements can be provided by 3 different sensors:

- SmartWatch from which is possible to obtain linear and angular velocity
- Leap motion from which is possible to obtain only angular velocity[1]
- Kinect from which is possible to obtain only linear velocity[2]



The robot is equipped with another Kinect which enables to record the point cloud.
The point cloud data, processing with the SLAM[3] algorithm, allows to reconstruct a three-dimensional map and simultaneously localize the smart car within this map while it is moving.
With this technique the user can visualize in real time the 3D map, by wearing a VR headset, and control the robot navigation in unknown environments in absence of external reference system such as GPS.

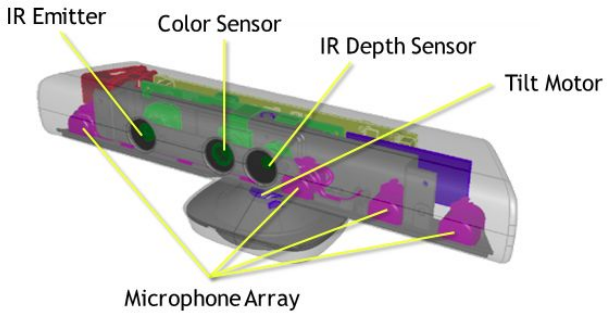
[1] For a better understanding see leap motion report.
[2] Due to some limitations provided by some libraries related to kinect sensor.
[3] Simultaneously Localization And Mapping.

During this report we will focus on Kinect sensor and the related module; the latter will allow us to send to the controller module the values of yaw, pitch, roll angles according to user's movements.

1.1. HARDWARE

1.1.1. Kinect V1

Kinect V1 is a motion sensing input device developed by Microsoft and originally made for its use with Xbox 360 and Windows Personal Computers. It allows the user to control and interact with the PC and it's capable of providing: full-body 3D motion capture, facial and voice recognition.



This sensor looks like a horizontal bar which is connected to a small base with a motorized pivot.

The cameras that are inside the Kinect are able to providing dense depth and color images at a good frame rate and this is the reason for which they are widely used for 3D reconstruction, SLAM, gesture and objection recognition.

The device consists of:

- Color Sensor: camera sensor to acquire RGB images
- IR Emitter + IR depth sensor: depth camera to acquire 3D images using infrared (IR) light
- Audio sensor: microphone array to capture sound signal and sound position
- Tilt motor: mechanical drive in the base of the Kinect sensor automatically tilts the sensor head up and down when needed

Feature	Microsoft Kinect v1
Hardware Compatibility	Stable work with various hardware models
USB Standard	USB 2.0
Size	33 x 16,5 x 12,7 cm
Weight	1 kg
Power Supply	USB + ACDC power supply
Power Consumption	12 watts
Vertical tilt range	±27°
Field of View	57° horizontal, 43° vertical
Frame rate	30 frame per second (FPS)
Color Camera	640 x 480 pixels FPS 30
Depth Camera	320 x 240 pixels FPS 30
Maximum Depth Distance	4 m
Minimum Depth Distance	80 cm
OS Platform Support	Xbox 360 Microsoft Windows Linux MacOS
Programming Language	C++/C# (Windows) C++ (Linux) Java

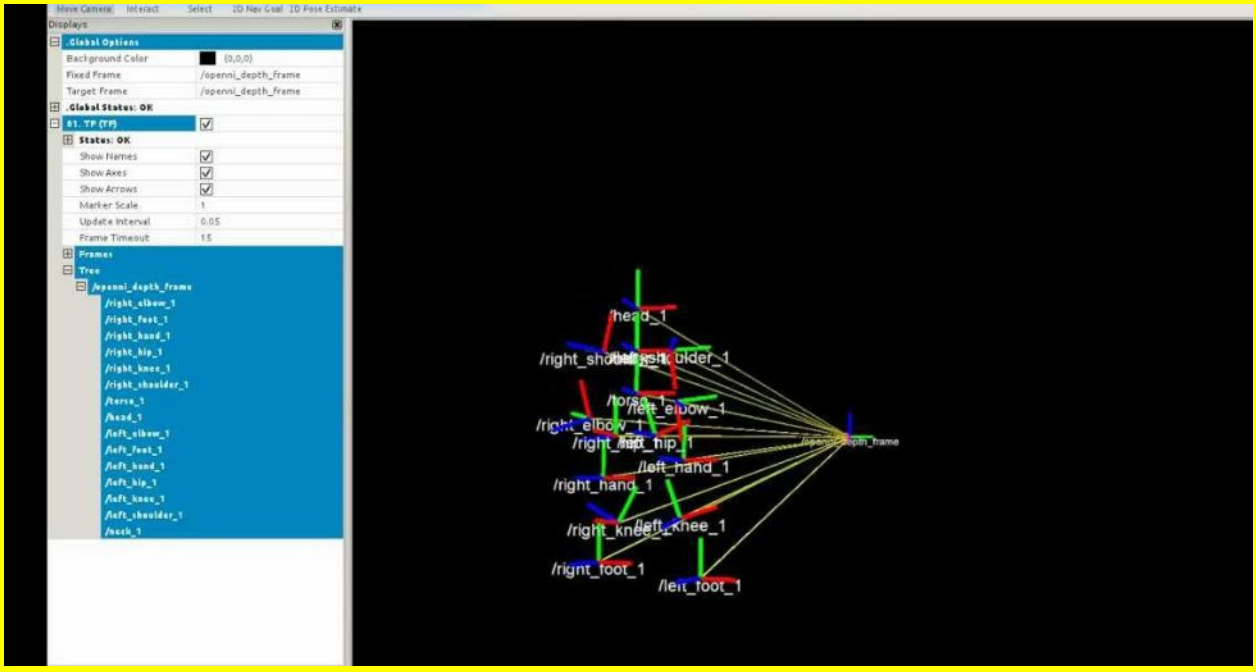
Table: Technical specifications Kinect V1

1.2. SOFTWARE TOOL

1.2.1. Rviz

Rviz, acronym for ROS visualization, is a powerful 3D visualization tool for ROS. It allows the user to view the simulated robot model, log sensor information from the robot's sensors, and replay the logged sensor information[4].

This software also permits to display in real time all the frames published on given topics. This technique is really useful in our project in order to check the skeleton tracking of the user's body provided by the Kinect sensor[5].

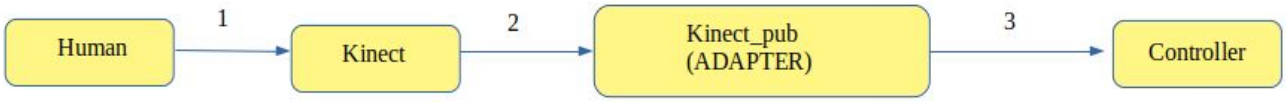


[4] For more information about rviz, see the original documentation at the ROS Wiki.
[5] For a better explanation of Rviz SetUp and utilization see chapter 3.3.1.

2. The System's Architecture

2.1. Kinect Module Overall Architecture

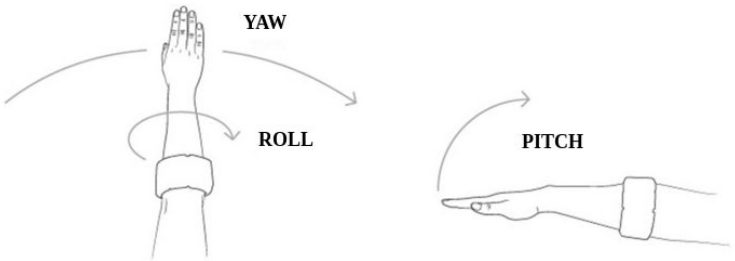
In this subsection are shown all blocks involved on the acquisition, processing and transmission of kinect data up to the controller.



1) User's pose
2) User's pose published as set of transforms
3) Customized ROS message with RPY information

2.2. Description of the Module

Kinect_pub module manages the Kinect sensor data flow and tracks user's arm by computing the correspondent pitch angle.



The value represented by the pitch angle is the main contributor for the controller computation of the robot linear velocity value[6].

[6] To calculate the overall linear velocity value the controller keeps into account also the pitch value provided by the smartwatch sensor.

- **Input:**
  As a input, the module takes the set of transforms representing the user's body pose.
  In our module we decide to consider the transformation matrices between shoulder and elbow each one with respect to the Kinect fixed frame.

- **Internal working:**
  Through the analysis of these two transformations, we can extract the vector joining elbow and shoulder; comparing this last value with a vertical reference vector we can compute the corresponding pitch angle.

- **Output:**
  The output of our module consists in customized RPY ROS message. However, for practical and implementation reasons, we set just the pitch angle field while the roll and yaw fields are set to a zero constant value[7].

It's important to specify that the pitch values sent to the controller have a $-\frac{pi}{2}$ offset in order to obtain an input accordant to the output range of the other sensors adapters. Applying this shift allows to have a smoother conversion between angles and velocity values.

[7] This decision derive by the fact that roll and yaw angles cannot be estimated with a good accuracy by Kinect v1 and we leave such estimation to the other sensors.

3. Implementation

3.1. Prerequisites

Minimum hardware requirements:

- 64 bit processor (x64)
- Dual-core 2.66-GHz or faster processor
- Dedicated USB 2.0 bus
- 2 GB RAM

Software requirements:

- OS: Ubuntu 16.04
- ROS DISTRO: kinetic[8]

In order to configure Kinect device on your PC, specifically to implement gesture recognition functionality, these libraries and frameworks are required: OpenNI, NITE and openni_tracker.

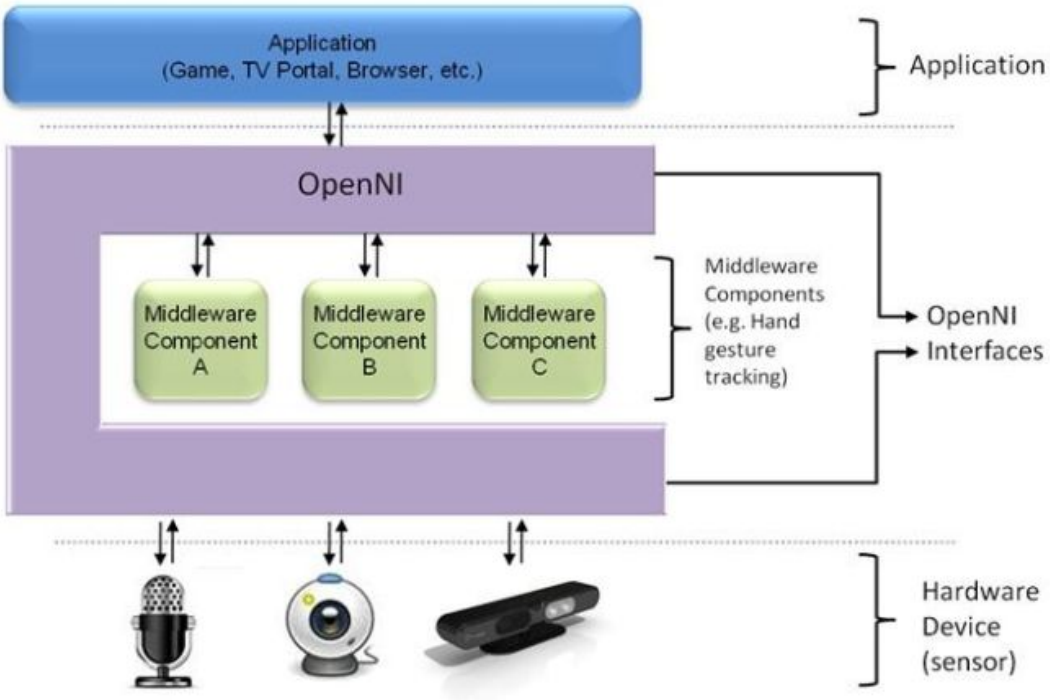
3.1.1. OpenNI

OpenNI[9] is a multi-language, cross-platform framework that defines APIs[10] for writing applications utilizing Natural Interaction (NI). OpenNI standard API enables NI application developers to track real-life (3D) scenes by utilizing data types that are calculated from the input of a given sensor (for example, representation of a full body, representation of a hand location, an array of the pixels in a depth map and so on).
The main purpose of this framework is to form a standard API that enables communication with both:

- Vision and audio sensors (the devices that 'see' and 'hear' the figures and their surroundings)
- Vision and audio perception middleware (the software components that analyze the audio and visual data that are recorded from the scene, and comprehend it)

[8] If you don't have ROS Kinect already install on your PC you can follow this tutorial ROS Kinetic that allow you to installing the platform and to configure your ROS Environment.
[9] Open Natural Interaction.
[10] Application Programming Interfaces.

For that reason OpenNI supplies a set of APIs to be implemented by the sensor devices, and a set of APIs to be implemented by the middleware components. By breaking the dependency between the sensor and the middleware[11], OpenNI's API enables applications to be written and ported with no additional effort to operate on top of different middleware modules ("write once, deploy everywhere").



Addition to OpenNI framework there is the openni_launch that is a package that contains launch files for using OpenNI-compliant devices such as the Microsoft Kinect in ROS. It creates a nodelet[12] graph to transform raw data from the device driver into point clouds, disparity images, and other products suitable for processing and visualization.

3.1.2. NITE

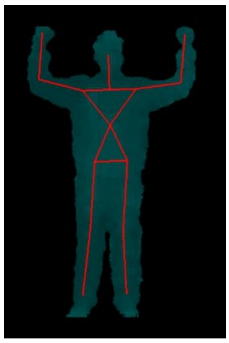
NITE is a binary distribution from PrimeSense offering skeleton tracking, hand point tracking, and gesture recognition and it is integrated with the OpenNI framework.

Nevertheless most of the time it is more convenient to use openni_tracker than NITE directly because using the first one allows you to have real time tracking of user's skeleton frames

[11] Each applications can be written regardless of the sensor or middleware providers.
[12] The nodelet package is designed to provide a way to run multiple algorithms on a single machine, in a single process, without incurring copy costs when passing messages intraprocess. For more information about nodelet see this ROS Wiki page.

published over tf[13]. Using openni_tracker package and with the auxiliary of Rviz[14] tool you can have a better understanding about skeleton tracking provided by the Kinect.

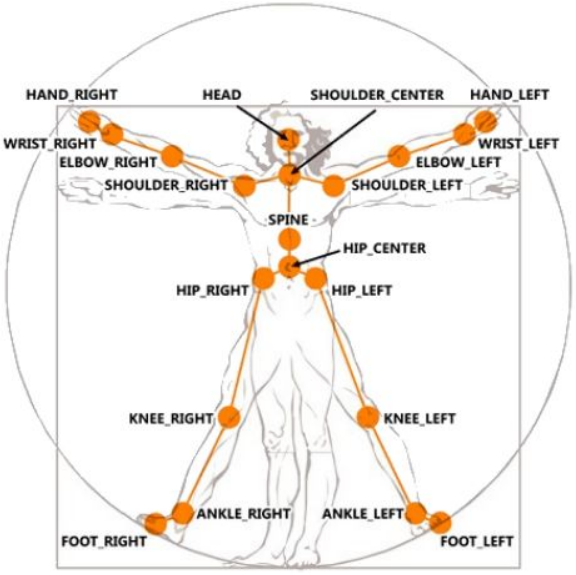
3.1.3. openni_tracker



The OpenNI tracker broadcasts the OpenNI skeleton frames using tf.
TF package maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time.
Once you have running the openni_tracker package[15], the user must has to stand in front of the Kinect and maintain the Psi Pose (see picture on the right) until successful calibration.

Once the calibration has been completed the user's pose will be published as a set of transforms (/tf) with the following frame names:

- /head_N
- /neck_N
- /torso_N
- /left_shoulder_N
- /left_elbow_N
- /left_hand_N
- /right_shoulder_N
- /right_elbow_N
- /right_hand_N
- /left_hip_N
- /left_knee_N
- /left_foot_N
- /right_hip_N
- /right_knee_N
- /right_foot_N



[13] Package that lets the user keep track of multiple coordinate frames over time.
[14] For more information see subsection 3.3.1.
[15] For more information see subsection 3.3.

REMARK: The frame names are mirrored considering the user's body (i.e. the right_foot_N[16] topic will be actually the left foot of the user) since they are named with respect to the Kinect point of view.

Since we assume that in our project there will be just one user for each run we decided to make the program listen only to the first user identified. This prevents any disturbances given by people accidentally walking in the camera vision at the price of re-initialization of the controller for each new user.

3.2. How to install the project

3.2.1. Download/install required libraries and packages

Install and update basic software components:

- apt-get update
- sudo apt-get install git build-essential python libusb-1.0-0-dev freeglut3-dev openjdk-8-jdk[17]
- sudo apt-get install doxygen graphviz mono-complete

It is preferable to place OpenNI, SensorKinect and NITE in a folder outside your workspace[18].

OpenNI:

- cd ~/
- mkdir kinect
- cd ~/kinect
- git clone https://github.com/OpenNI/OpenNI.git
- cd OpenNI
- git checkout Unstable-1.5.4.0
- cd Platform/Linux/CreateRedist
- chmod +x RedistMaker
- ./RedistMaker

[16] Kinect sensor will track any human inside its field of view as a new user with an increasing identification number (N) which will complete the topic name (i.e. User 1 right foot tf will be published as /left_foot_1 topic).
[17] For previous distros such as indigo you must replace *openjdk-8-jdk* with *openjdk-7-jdk*
[18] The following tutorial is done according to this suggestion.

- cd ../Redist/OpenNI-Bin-Dev-Linux-x64-v1.5.4.0
- sudo ./install.sh

SensorKinect[19]:

- cd ~/kinect
- git clone https://github.com/avin2/SensorKinect
- cd SensorKinect
- cd Platform/Linux/CreateRedist
- chmod +x RedistMaker
- ./RedistMaker
- cd ../Redist/Sensor-Bin-Linux-x64-v5.1.2.1
- chmod +x install.sh
- sudo ./install.sh [20]

Kinetic OpenNi:

- sudo apt-get install ros-kinetic-openni*

NITE 1.5.2.23[21]:

- cd ~/kinect
- git clone https://github.com/arnaud-ramey/NITE-Bin-Dev-Linux-v1.5.2.23.git
- cd x64
- sudo bash install.sh

openni_tracker:

- cd ~/<workspace_name>/src
- git clone https://github.com/ros-drivers/openni_tracker.git

Project repository:

- git clone https://github.com/EmaroLab/hrp_teleoperation/tree/MainControl/src/kinect_listener
- git clone https://github.com/EmaroLab/hrp_teleoperation/tree/MainControl/src/my_msgs[22]

[19] Module that allows the Kinect Controller to work with OpenNi.
[20] Be careful not to run this command more than once, otherwise you might run into problems.
[21] Also 1.5.2.21 version is compatible with openni_tracker package.
[22] The my_msgs folder is required for customized RPY ROS messages.

3.2.2. Build the ROS workspace

You have to do these steps in order to build and install the new libraries and packages.

- cd ~/<workspace_name>
- catkin_make
- catkin_make install

3.3. How to run the project

Open 3 different shells[23] [24]:

Shell_1. roslaunch openni_launch openni.launch camera:=openni
Shell_2. rosrun openni_tracker openni_tracker

If you have done all correctly, after the user's Psi Pose, you should see the following messages displayed on the second terminal:

```
New User 1
Pose Psi detected for user 1
Calibration started for user 1
Calibration complete, start tracking user 1
Lost User 1
```

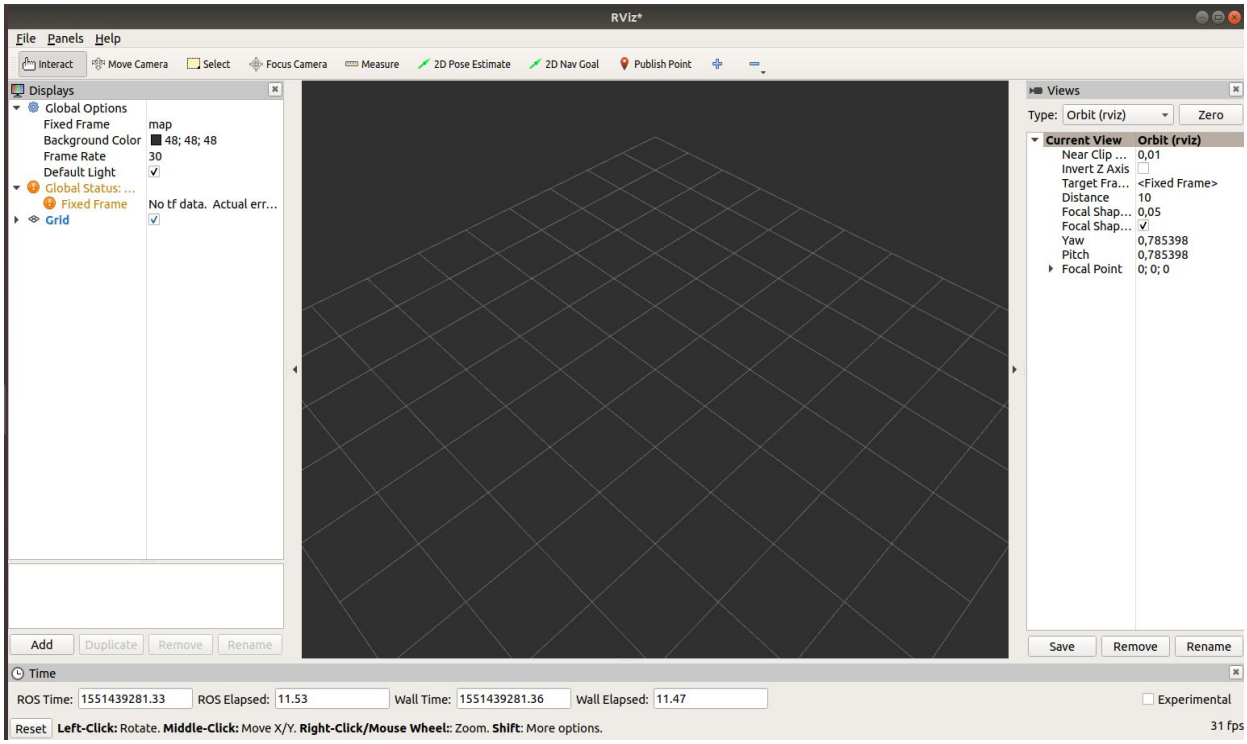
Shell_3. rosrun Kinect_listener Kinect_tf_listener[25]

3.3.1. Rviz setup and visualization

If you want to visualize the results on Rviz, open a new terminal and run the tool with this command:

- rosrun rviz rviz

[23] You don't need to explicitly run the master (using roscore command) because it will automatically calling by roslaunch.
[24] Every time you open a new terminal you need to source a setup.bash file so you have to write *source devel/setup.bash* on it otherwise you will obtain a rospack error. If you want every terminal to automatically source a particular setup.bash script, you must put that command in your ~/.bashrc file. For providing that you have to write: *echo source [PATH]/<workspace_name>/devel/setup.bash >> ~/.bashrc*
[25] To run the module explained in subsection 2.2.



Now you have to change some visualization options in such a way you can see in real time the tracking of the user's skeleton frames:

- Global Options > Fixed Frame > change from *map* to *openni_depth_optical_frame* [26]
- press the button *Add* (on the bottom-left of the window) and select *PointCloud2* [27] display type
- PointCloud2 > Topic > write */openni/depth_registered/points*
- press again button *Add* but this time select *TF* [28] display type

After these changing it is possible to see all the /tf sent by openni_tracker package to the computer and displayed by RVIZ in such a way you can use this tool as a debugger (i.e. to verify if the user's frames are coherent with the real user's pose).

[26] This will be the fixed frame with respect to which all the transformation matrix of the user will be provided.
[27] Displays a point cloud from a sensor_msgs::PointCloud2 message as points in the world, drawn as points, billboards, or cubes.
[28] Displays the TF transform hierarchy.

3.4. Algorithms

In this subsection is introduced our node implementation and also the logical steps which brought us to develop the algorithms described below.
Since the lack of a clear documentation, we experimented with the Kinect in order to obtained more confidence and knowledge of how it works and its behavior depending on various scenarios; visualizing the Kinect data from Rviz we have noticed that the frame positioned on the hand, by the skeleton tracking feature by NITE 1.5.2.23, wasn't changing its orientation.
This prevented us to use the quaternion of the hand to extract a RPY triplet, therefore we have implemented several algorithms looking for portability and the most accurate estimation of the arm pitch angle.

3.4.1. Fixed hand frame exploitation

Assumed the fixed hand frame, we decided to exploit the situation:

1. Listen to the transformation matrix of the hand with respect to the shoulder.
2. Use a tf library function to obtain the origin vector that connects the shoulder frame with the hand one.
3. Compute with a vector library function the angle between the previously found vector and a vertical reference vector. The result is the pitch of the arm.

This algorithm results are accurate unless a little offset given by the misreading of the position of the frames. The reliability of this algorithm is strictly connected to the fixed hand frame situation so we decided to look for a more general way.

3.4.2. Quaternions Algorithms

We then decided to work out an algorithm using quaternion of elbow and shoulder frames:

1. Listen to the transformation matrix of the elbow/shoulder with respect to the basis frame.
2. Extract the rotational matrix of the up-mentioned transform.
3. Use a matrix library function to get the RPY configuration and extract pitch data.

Analyzing the results we noticed that the data from the elbow quaternion isn't reliable since the matrix keeps rotating and translating due to Kinect accuracy issues.

The quaternion of the shoulder instead gives more reliable data (worse than the fixed hand algorithm), but loses accuracy when the arm is rotated in order to control the angular velocity and suffers from disturbances given by misreading of frame position.
This led us to consider again an algorithm based on distance vectors.

3.4.3. Distance vector between shoulder and elbow frames

In order to generalize the fixed hand algorithm, we used a geometrical computation to obtain a vector which doesn't rely on the assumption of the fixed frame:

1. Listen to the transformation matrix of the basis with respect to the shoulder and the transformation matrix of the basis with respect to the elbow[29].
2. Use a tf library function to obtain the origin vector that connects the shoulder and elbow frames with the basis. Compute the difference between those vectors to obtain a vector that connects the two frames.
3. Compute with a vector library function the angle between the previously found vector and a vertical reference vector. The result is the pitch of the arm.

The results were as accurate as the first algorithm but the doesn't rely on the assumption of the fixed frame.

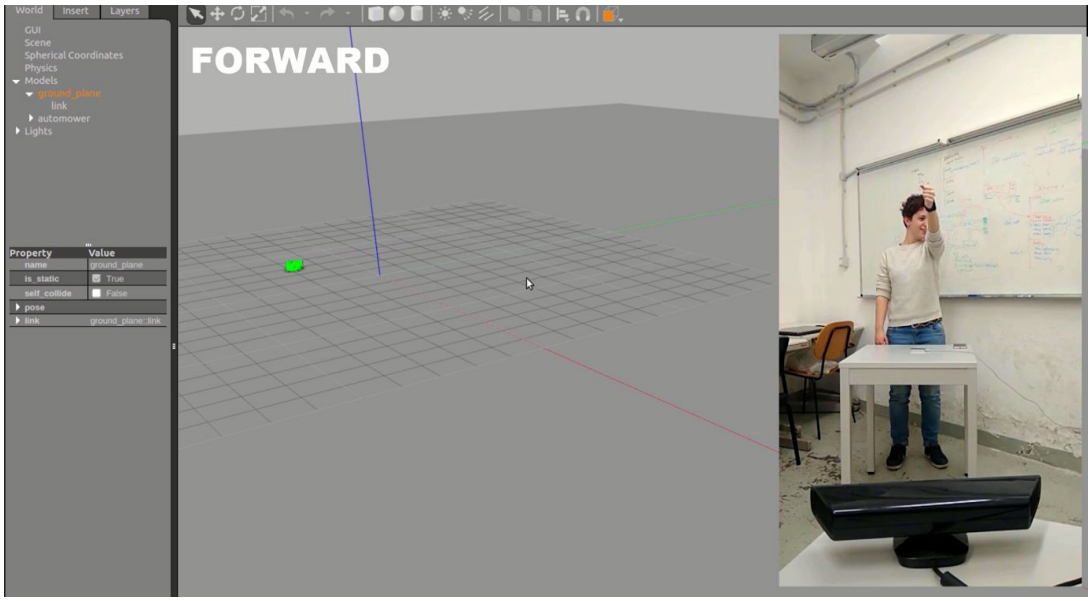
[29] We used the source as the target since the tf library function actually compute the vector with respect to the target.

4. Results

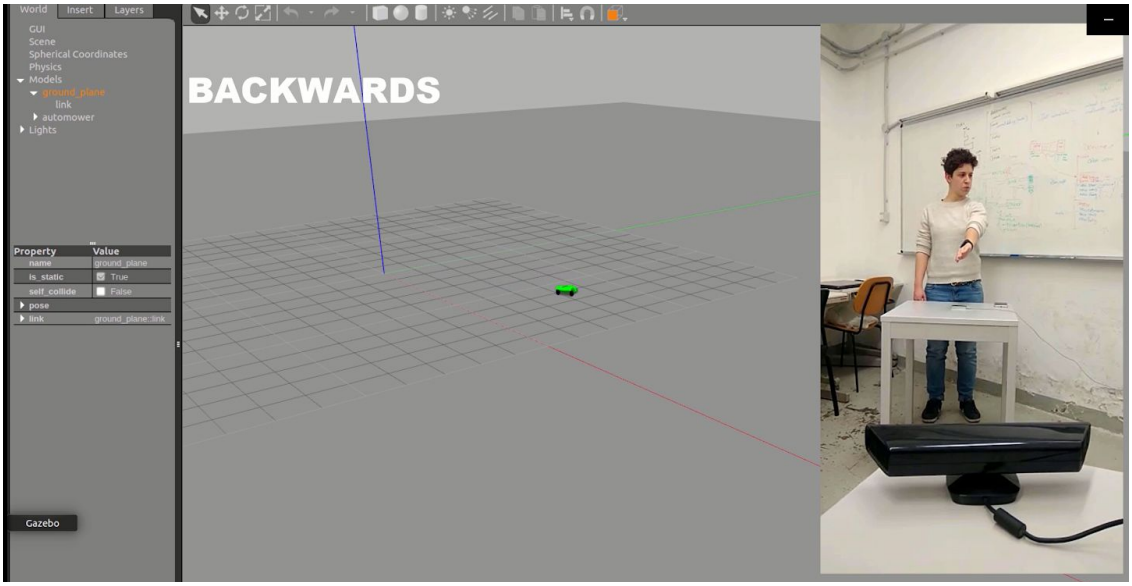
After the various attempts made on the different methods, listed in the previous section, we decided to implement last one algorithm, explained in subsection 3.4.3, in such a way to obtain a module that still have some some limitations[30] but with a good accuracy regarding the pitch angle.
In conclusion we have built a module that allow the user to control the linear velocity of the robot in the following way:

Degrees of left user's arm with respect the vertical axis	Robot movement
0° ÷ 45°	Backward with maximum speed
~90°	Stop
135° ÷ 180°	Forward with maximum speed

Due to the unavailability of the robot we simulated the overall project, considering also the leap motion and smartwatch sensors, with the auxiliary of Gazebo.



[30] Because the Kinect sensor can't provide an accurate estimation of the yaw and roll angles.



The video that shows the behaviour of the overall mobile robot teleoperation project can be seen at this link: https://youtu.be/Zv07ShMY1a4

5. Recommendations

In this section we present the key points that have limited our work and our results. We also put in evidence the assumptions on which our implementation leans.

Recommendations:

- Make sure that the user is in the vision field of the sensor.
- Limit the persons in the range of the Kinect to one to guarantee a good calibration during the configuration phase.
- For a better pose estimation it's better not move too fast.
- In order to control the linear velocity of the robot you have to move up and down the left arm even if in the code, and Rviz visualization, it is called as right arm. The reason for this mismatch is the fact that the frame names are named with respect to the Kinect point of view.

Limitations:

- Fixed frames of /left_hand and /right_hand so we could not work on quaternions of those frames. Due to this fact we could not provide roll angle information that prevented us from also controlling the angular velocity of the robot.
- Bad accuracy for the RPY angles considering the elbow frame.
- General lack of documentation for the gesture recognition using the Kinect.
- Apple, after the acquisition of OpenNI project[31], stopped public access to all middleware libraries and applications for 3D sensing, so also NITE libraries, and this prevented us to work with Kinect v2 because NITE2 was no more accessible.

[31] For more information see *https://www.theregister.co.uk/2014/03/02/openni_project_to_close/*