Software Architecture Project

Student(s): Zaid, Davide Piccinini, Francesco Porta, Aurora Bertino, Chetan Chand Chilakapati, Muhammad Usman Ashraf, Polaka Surendra Kumar Reddy, Silvana Andrea Civiletto, Sara Romano
Tutor(s): Alessandro Carfì, Syed Yusha Kareem
Supervisor(s): Alessandro Carfì, Syed Yusha Kareem, Antony Thomas
Year: 2019 - 2020

[Project 13]
An architecture to build a multimodal dataset for gesture recognition

Table of Contents

Abstract

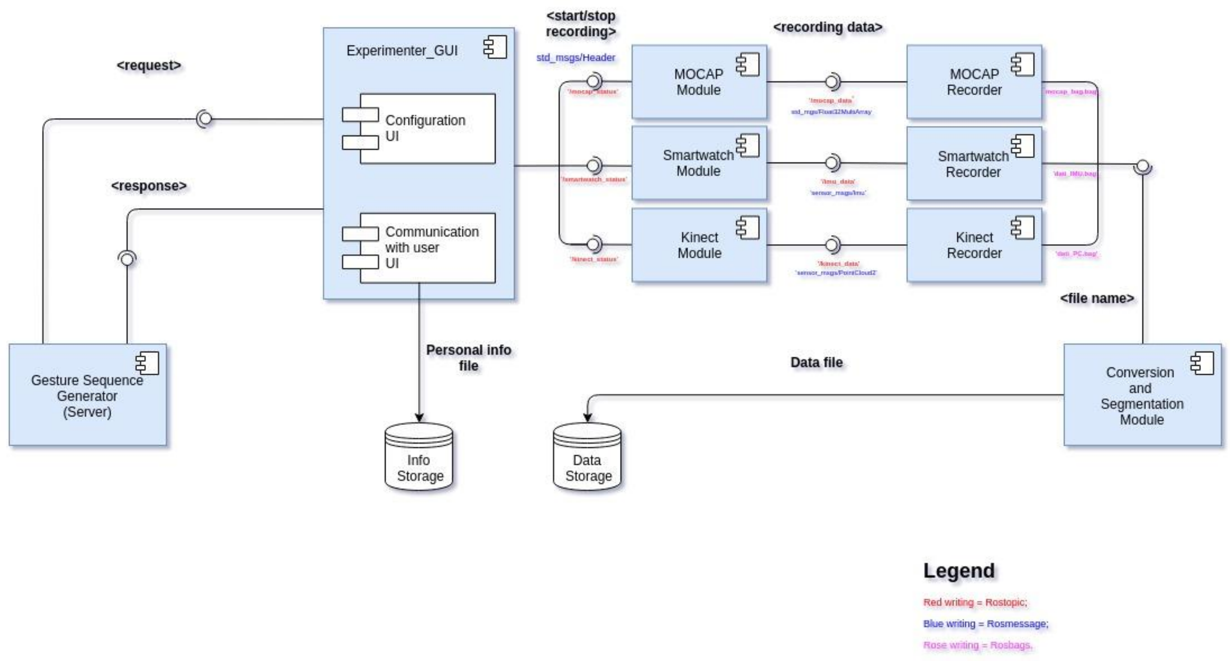
The objective of this project is to build a software architecture that allows to easily build a multimodal dataset that can be used for gesture recognition: the architecture is split into two different GUIs, one of which is used to acquire data from the sensors and the other one allows the user to visualize the stored data.
In this project we are interested in building the dataset for 5 types of gestures (i.e. pouring, drinking, sitting-down, standing-up, walking) using three different types of sensors (i.e. Kinect, Smartwatch and Motion Capture).

[GitHub repository link]

1. Introduction

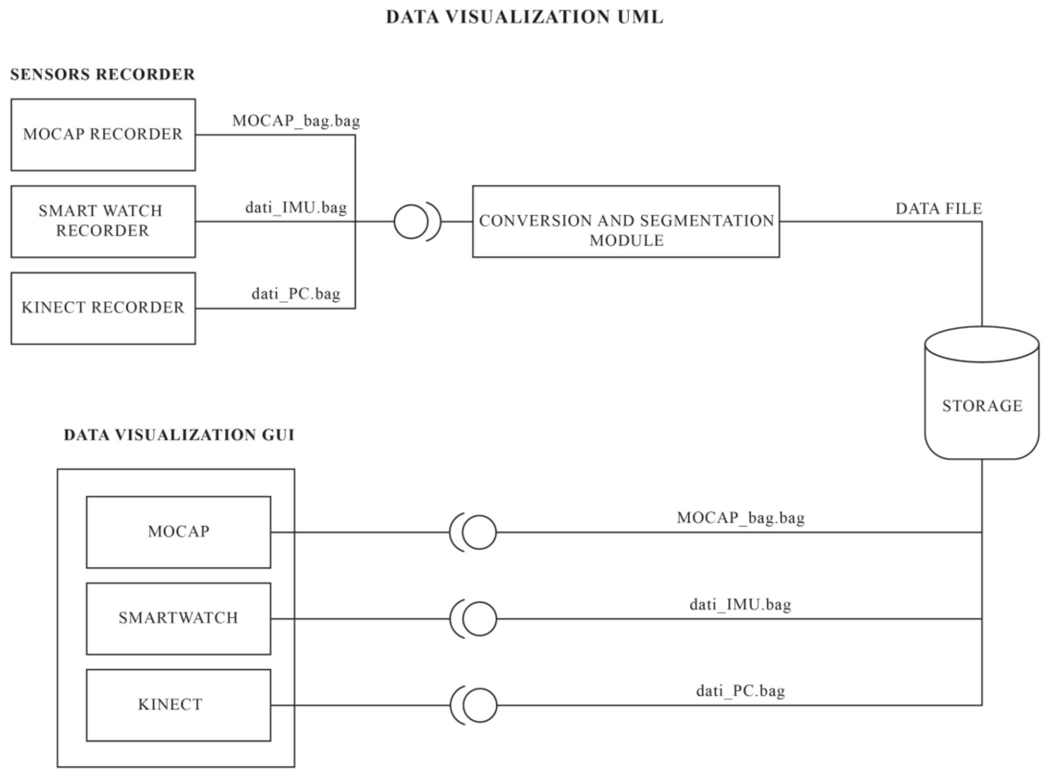
The main goal of this project is to create two software architectures that simplify building multimodal datasets and lets you visualize the data gathered.
In order to create an engaging and easy-to-pick-up application the most reasonable decision was to develop one graphical user interface (GUI) for each of the two architectures.
The two software architectures have been developed using ROS (Robot Operating System) for two main reasons: the first one is because with ROS we can achieve a distributed and highly modular design for the architectures, also allowing us to exploit its middleware features which are key to the project, the second one is because the sensors that we will be using are already properly interfaced with it.
A program called "QtDesigner5" was used to design the whole graphical part, which comprises all the windows, buttons, line edits, combo boxes, etc. which can be used to interact with the program.

2. Architecture of the System



Above depicted is the architecture of the dataset creation part of the project: it's made up of 7 different modules, one of which consists in two sub-modules.

The “Gesture Sequence Generator” module is the implementation of the server of a service-client pattern; the “Experimenter GUI” represents all the graphical part of the architecture (divided into “Configuration UI”, which contains the management of the recording’s configuration, and “Communication with User UI”, where the user can control the recording and where images and time elapsed are displayed); the three “Sensor Modules” collect data from the sensors (MOCAP, Smartwatch and Kinect) and when they receive the signals from the Experimenter_GUI, they publish datas in a topics (one for each type of message); the three “Record Modules” read data from the topics published by the Sensor Modules and record them into a Rosbags; the “Conversion and Segmentation” module is commissioned to convert the data from “rosbag” to “csv” format, to assign certain labels to it and to segment them.



The above illustrated picture is another integral part of the project that comes right after the recording and segmentation of the data. After recording and storing the data in rosbags, the contents of these rosbags needs to be visualized. For this purpose, a UI has been developed which helps to bridge the connection between user and visualization of the data stored in the rosbags. This UI comes along with the capability of portraying three types of data i-e /float32MultiArray, /IMU, /Pointcloud2.

3. Description of the System’s Architecture

Hereafter, every module will be explained in detail, describing how does it work, what it needs in order to function, what is its purpose and other important informations.

3.1. “Configuration UI” Sub-Module

Developed by Davide Piccinini and Francesco Porta.

This sub-module consists in the configuration window where the user can:

- enter his/her personal data (i.e. Name, Surname, Age, Gender, Height and Weight), which will be saved in a simple “txt” file if afterwards the recording will be initialized;
- choose whether requesting a casual gesture sequence to the “Gesture Sequence Generator” module by checking the respective box and entering how many gestures to request, or selecting himself/herself up to five gestures (from the ones available) which will be performed in the defined order;
- select which sensors will be used for the current recording by checking the desired boxes.

“Pathological” use cases have been also considered:

- if the user doesn’t write anything into the personal information frame, then what will be created will be a “txt” file containing only the default template, which can be filled in afterwards;
- if the user requests a floating-point number of casual gestures, the request will be carried out by rounding the number to the nearest integer: moreover, if the user writes something that isn’t a number, instead a request for only one casual gesture will be made;
- if the user decides to put the value “None” on all the combo boxes, the program will use a gesture sequence made up of only one gesture, namely pouring;
- if the user doesn’t check any of the sensor boxes, then the program will use as default sensor the Kinect.

This sub-module needs the “pyqt5” library in order to function; this is further mentioned in the “Installation” paragraph.

The sub-module works by exploiting the features of PyQt5, namely the GUI components (i.e. buttons, line edits, check boxes, combo boxes, etc.) which allow the user to interact actively with the program: what the program does is retrieving the inputs issued by the user, saving important informations that will be afterwards used in the “Communication with User UI” sub-module, eventually requesting a random gesture sequence to the “Gesture Sequence Generator” module and then initializing the next window, which is the recording/communication with user one.

Being this sub-module directly tied to the “Communication with User UI” one, with which it jointly forms the “Experimenter GUI” module, speaking about inputs or outputs for one or the other sub-module doesn’t make much sense. For this reason, this topic will be resumed after having explained also the second sub-module, so that a more general overview of the whole module can be established.

3.2. “Communication with User UI” Sub-Module

Developed by Davide Piccinini.

This sub-module naturally follows up from the “Configuration” one. In fact, by pressing the “Go to Recording” button in the first window, the recording/communication with user window is initialized, where the user can:

- go back to the configuration window if he/she made some mistakes;
- click the “Play” button to start a 10-second countdown (implemented to allow who is going to perform the gestures to get ready) after which the recording finally begins;
- see the image corresponding to the current gesture to be performed, observe how much time has passed since the beginning of the recording and what is its total duration;
- if the recording has been started, terminate it before the predefined time by clicking the “Stop” button.

No “pathological” use cases are present since only a few buttons are enabled at the time to avoid problems.

As with the previous sub-module, this one also needs the “pyqt5” library: other than that, default images that need to be displayed are in the “gui_content/pictures” folder.

As with the previous sub-module, the GUI components allow to interact with the program. The flow of this sub-module is the following: if the user doesn’t want to go back to the previous window then the only option is to click the “Play” button, which will execute a simple countdown function; when the countdown reaches 0, the personal information “txt” file is created together with the sensor data files and the recording is officially started by issuing a start command to the sensors; every second a function to update the time elapsed is called, which eventually also updates the image being displayed and, when the recording time has finished or the user has clicked the “Stop” button, sends a stop command to the sensors; the last thing the user can do is then click the “Go Back” button to roll back to the configuration window.

Let’s now resume the discussion about the inputs and outputs of the whole module.
The module is executed via the launch file but everything it needs in order to function is in the several folders in the repository so, strictly speaking, no input/parameter is required.
On the other hand, if we take into account what the user will write or select afterwards when dealing with the GUI, then his/her personal informations, how is the gesture sequence created and what sensors will be used can be considered as inputs, though not mandatory for the module’s functioning (the user may not write/select anything and it will still work).
What the module produces as outputs are, if the recording is started, the “txt” file containing the user’s personal informations and the data files corresponding to what sensors where selected in the configuration window.

3.3. “Gesture Sequence Generator” Module

Developed by Francesco Porta.

This module is the server part of the Gesture Sequence Ros service. When the “Casual Sequence” checkbox is ticked and the button “Go to Recording” is pressed in the Configuration Module, a request containing the number of desired gestures is sent to this module.
Each of the five gestures correspond to a number from 0 to 4:

- 0 ⇒ Drinking
- 1 ⇒ Pouring
- 2 ⇒ Sitting
- 3 ⇒ Standing Up
- 4 ⇒ Walking

The module creates an array with dimension equal to the requested number of gestures and fills it with a random sequence of these five gestures: if the number of required gestures is less than or equal to five, no repetitions of the same gestures are allowed.
Then the gesture sequence array is sent as a reply to the Configuration module.

This module has no specific prerequisites.

The inputs and outputs are only the request and response to the Ros Service, the request is a simple integer and the response is an integer array, as already mentioned.

3.4. “Fake Imu” Module

Developed by Sara Romano.

Due to the current situation (Covid-19) we are not able to have access to the lab, therefore the focus of this module is to generate a fake data with the same structure of the data that should be acquired from the real sensor ('sensor_msgs/imu').
Since looking for and understanding the structure of this kind of Ros message (http://docs.ros.org/api/sensor_msgs/html/msg/Imu.html), the creation of that was done, establishing any values to zero.
Hence the module publishes the fake ros message into a topic ('/fake_data_imu') in such a way they will be reached by the sensor module.
To activate the module, you have just to run it before launching all nodes of the package.

There are no prerequisites to define this module.

There is not input, while the output is just the fake data ('sensor_msgs/Imu').

3.5. "Smartwatch" Module

Developed by Sara Romano.

This module represents the node which collects a data from Smartwatch sensor when it is required.
When the user selects the Smartwatch sensor for the recording data, then the GUI module sends a signal, through the publishing a Ros message ('Header') into a topic ('/smartwatch_status'), so the node is a subscriber of this topic.
Likewise, the module is also subscribed to the topic that contains all the data recorded by the sensor (in this case the fake module publishes all this data setting them to zero).
When the Ros messages from every two topics are submitted, then the module publishes all the recorded data into another topic ('/imu_data').
The module will be active when the launch file ('Experimenter_GUI.launch') is launched on the Terminal.

This module does not require specific prerequisites.

The input are two: 'Header' message and 'sensor_msgs/Imu' message, while the output is just 'sensor_msgs/Imu'.

3.6. "Fake PointCloud2" Module

Developed by Sara Romano.

This module has been also built for the same causes as the "Fake Imu" Module was created: generating fake data equal to the interesting data that should be acquired from the Kinect sensor.
The structure of the Ros message such that contains this kind of data is 'sensor_msgs/PointCloud2'.
After browsing the structure of this kind of Ros message (http://docs.ros.org/api/sensor_msgs/html/msg/PointCloud2.html), the creation of that has been done.
Hence the module publishes the ros message into a topic ('/fake_data_PC'), which will be subscribed by the Kinect Module.
To activate this module, you have just to run it before launching all nodes of the package.

There are not a specific prerequisites for this module.

There are no inputs, rather the output is the ros message 'sensor_msgs/PointCloud2'.

3.7. "Kinect" Module

Developed by Sara Romano

This module, as well as the Smartwatch module, represents the node which collects a data from Kinect sensor when it is required.
When the user selects the Kinect sensor for the recording data, then the GUI module sends a signal, through the publishing a Ros message ('Header') into a topic ('/kinect_status'), so the node is a subscriber of this topic.
Likewise, the module is also subscribed to the topic that contains all the data recorded by the sensor (it should be published by the "Fake PC" module).
When the Ros messages from every two topics are submitted, then the module publishes all the recorded data into another topic ('/kinect_data').
The module will be active when the launch file ('Experimenter_GUI.launch') is launched on the Terminal.

Also in this module, the specific prerequisites are not required.

The inputs are the 'Header' Ros message, and 'sensor_msgs/PointCloud2' message; instead the output is just 'sensor_msgs/PointCloud2'.

3.8. "mocap_fake" Module

Developed by Aurora Bertino.

The module that collects data from Mocap Optitrack sensor system should be used.
This a node that publishes position for all the markers and not only for data coming from a rigid body.
Due to the current situation (Covid-19) we are not able to have access to the lab, therefore it was not possible to test properly the code in order to see the real work of this acquisition system.
Nevertheless, it has been computed a simulation considering fake mocap data, with the same structure of the data that should be acquired from the real sensor.
The repository containing the Mocap package in which there is the real node that publishes markers coordinates has been cloned by
https://github.com/ACarfi/SOFAR/tree/master/mocap_optitrac .
This package is included in "multimodal_dataset_creation" inside the repository "Dependence".
The 'mocap_fake' node publishes to the topic ('/fake_data_mocap') fake data with all values settled to zero as a structure std_msgs/Float32MultiArray'. These fake mocap data that represent markers coordinates are pushed in an array and also pushed on the topic ('/fake_data_mocap').

This node should be launched with the other sensors before that the experiment starts.

This module does not require specific prerequisites.

There are no inputs, rather the output is the ros message 'std_msgs/Float32MultiArray'.

3.9. "Mocap_node" Module

Developed by Aurora Bertino.

This module represents the node which collects data from Mocap sensor when it is required.
When the user selects the Mocap sensor for the recording data, the GUI module sends a signal, through the publishing a Ros message ('Header') into a topic ('/mocap_status'), so the node is a subscriber of this topic.
Likewise, the module is also subscribed to the topic that contains all the data recorded by the sensor (in this case the fake module publishes all this data setting them to zero).
Hence, the node subscribe to the topic ('/fake_data_mocap').
When the Ros messages from every two topics are submitted, then the module publishes all the recorded data into another topic ('/mocap_data').

The module will be active when the launch file ('Experimenter_GUI.launch') is launched on the Terminal.

This module does not require specific prerequisites.

The inputs are two: ‘Header’ message and the fake data with the structure std_msgs/Float32MultiArray which are published in the topic (‘/fake_data_mocap’).

The output is one: std_msgs/Float32MultiArray that are fake mocap data published in the topic (‘/mocap_data’).

3.10. “Conversion and Segmentation” Module

Developed by Silvana Andrea Civiletto.

This module converts files from Rosbag published by sensor modules into csv format. After verifying the correctness of the input arguments, you can access the Rosbag and create new csv files. Defined the ‘Header’, which is a timestamp of ROS, you make the recovery of the list of the topics from the Rosbags for every ‘FileName’. So, it’s possible to write the first line of the csv file from the first element of each pair of ‘Header’ values and to drag the value from each pair to the corresponding file.
This module is also responsible for labelling and segmenting data previously converted into csv format. In this module you have an explicit segmentation, because every time the GUI shows the person the gesture to make, the GUI saves the timestamp. Knowing the timestamp in which each gesture was requested, timestamps of consecutive gestures are used to segment the sequence. But this is not enough: the person doesn’t immediately simulate the gesture and the time left to the person to perform it is wider than what it takes to segmentation. So, it is necessary to isolate the instant of time in which the person is stationary, cut it and save the real timestamps of beginning and end of the gesture and use them as a model for segmenting other data.

This module does not have to be ‘online’ during the collection gestures data and during data storage in different rosbags. This means that it’s possible to run it anytime.

This module has no specific prerequisites.

The inputs for the module are the bag files coming from the sensors,which are protagonists of the conversion. The outputs are the same files converted to csv format and subsequently segmented and placed inside the data storage.

3.11. “Recorder IMU” Module

Developed by Sara Romano.

This module generates a rosbag (the name is dati_IMU.bag) that incorporates all data from the Smartwatch sensor when it is required. Practically, it is a subscriber of the topic ('/imu_data')

published by Smartwatch node, then it publishes them into a rosbag. Remarking that the ros message is 'sensor_msgs/Imu'.
The node does not require any prerequisites.
The input is the data from the sensor module (from '/imu_data' topic), and the output is a rosbag.

3.12. “Recorder PC” Module

Developed by Sara Romano.

This module generates a rosbag (the name is dati_PC.bag) that incorporates all data from the Kinect sensor whenever it is required. Practically, it is a subscriber of the topic ('/kinect_data') published by Kinect node, then it publishes them into a rosbag. Remarking that the ros message is 'sensor_msgs/PointCloud2'.
The node does not require any prerequisites.
The input is the data from the sensor module (from '/kinect_data' topic), and the output is a rosbag.

3.13. “ Recorder_mocap” Module

Developed by Aurora Bertino

Rosbag has both C++ and Python APIs for reading messages from and writing messages to bag files.
This module implements a creation of a rosbag each time the recording starts and writes into the rosbag the sensor mocap data published to the topic (‘/mocap_data’) by the Mocap module. Remarking that the ros message is 'std_msgs/Float32MultiArray'.

This node is launched with the command ‘experimenter_GUI.launch’ on the Terminal.

The node does not require any prerequisites.

The input is the data from the sensor module (from '/mocap_data' topic), and the output is a rosbag.

4. Installation

The first thing to do, after having cloned the repository in the Ros workspace, is to build the package and install in order to make the ‘msg’ and ‘srv’ files executable, using the following commands in the workspace:

```
catkin_make
catkin_make install
```

Then it is necessary to install a Ros related Python library (this passage may not be required if the pc on which the modules will be installed has already other Ros projects developed with Python 3).

```
sudo apt-update
sudo apt install python3-pip
sudo apt-get install python3-yaml
sudo pip3 install rospkg catkin-pkg
```

4.1. “Experimenter UI” Module

This is the graphical interface module, so it is required to install a library which is necessary to build the UI objects. The aforementioned library is PyQt5 and the commands needed to install and use it are the following:

```
pip3 install --user pyqt5
sudo apt-get install python3-pyqt5
```

The module will run automatically when the [script/roslaunch command] will be launched in the terminal

```
roslaunch multimodal_dataset_creation experimenter_GUI.launch
```

4.2. “Gesture Sequence Generator” Module

This module doesn’t need any particular library, it only uses ‘rospy’ which should already be included in the standard Ros installation.

4.3. “Fake Node” Module

We were going to simulate data generation as much similar to the real sensor as possible.
And thus, taking into account how the real sensors work, we decided to launch just the three fake nodes through a different launch file different from Experimenter_GUI_launch.
Considering this, the first thing to do, to allow that the system will work, is to launch "fake_nodes.launch".

```
roslaunch multimodal_dataset_creation fake_nodes.launch
```

4.4. Other Nodes

The other nodes which are not specified, will be launched with Experimenter_GUI.launch.

4.5 Data Visualization GUI

All the nodes of Data visualization GUI will be launched when all of the packages of Experimental GUI are installed

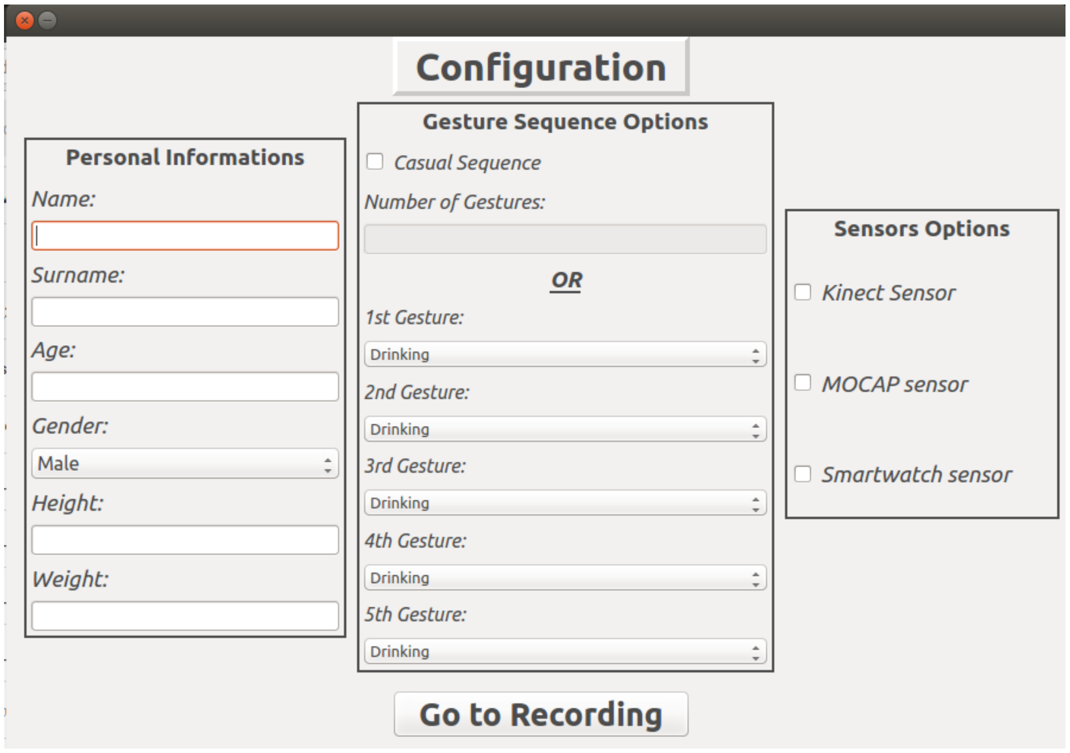
5. System Testing and Results

5.1. "Experimenter UI" and "Gesture Sequence Generator" Modules

The "Experimenter UI" is provided with a 4-page tutorial embedded into the application, which is useful to understand its functioning.

For completeness, we will hereafter explain using images how the two modules work.

The following image shows how the configuration window presents itself.



Name:

Surname:

Age:

Gender: Male

Height:

Weight:

1st Gesture: Drinking

2nd Gesture: Drinking

3rd Gesture: Drinking

4th Gesture: Drinking

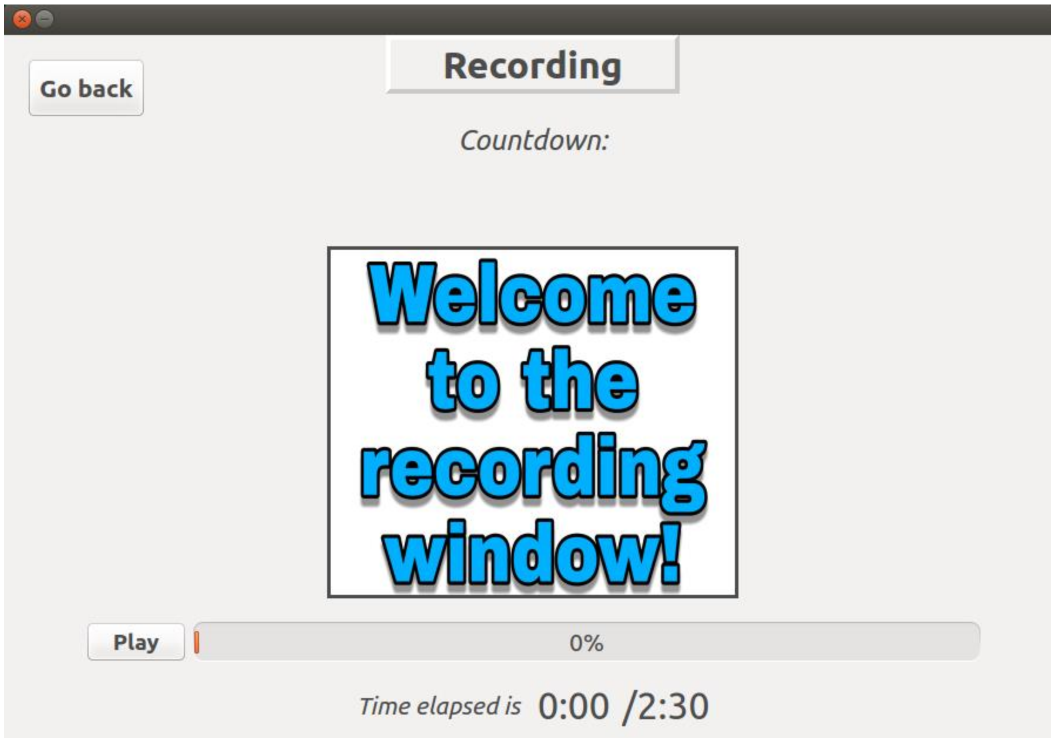
5th Gesture: Drinking

In the leftmost part you can see the frame dedicated to the personal informations: as previously explained, the user can write into the white boxes what he/she wants.

In the center part we have the choice of the gesture sequence. You can see how the “Number of Gestures” box is disabled because “Casual Sequence” isn’t checked: in this situation the user can select up to 5 gestures in the combo boxes to build up the sequence; if instead “Casual Sequence” is checked, the lower part of the frame will get disabled and you will be able to write in the “Number of Gestures” box. Doing so will trigger a request to the “Gesture Sequence Generator” module when the “Go to Recording” button is clicked.

In the rightmost part you will find the “Sensors Options” frame: check the boxes corresponding to the sensors you want to use.

The following image shows how the recording window presents itself.



In the upper-left corner there is the "Go Back" button which leads back to the configuration window when clicked.

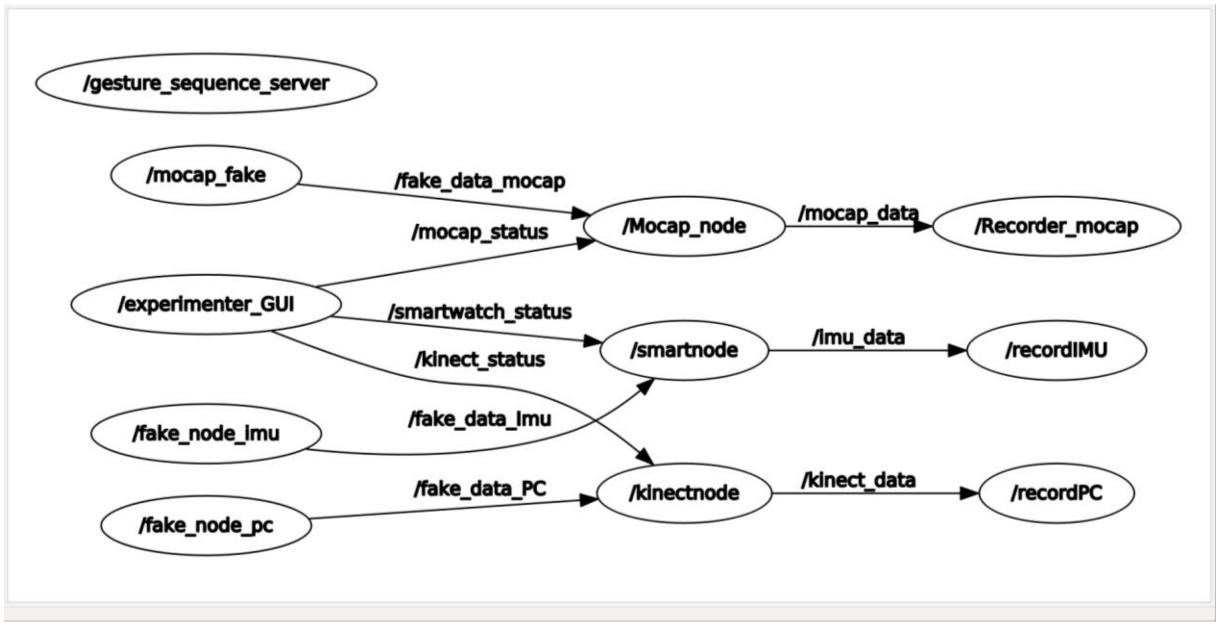
Right below the label containing "Countdown:" the countdown will be displayed.

In the center we can see the frame containing an image: when recording, the images will be updated every 30 seconds according to the gesture which needs to be performed.

Then we have the "Play" button, whose task is to start the countdown when clicked: when the countdown is concluded, the recording begins, the "Play" button becomes the "Stop" button and both the progress bar and the labels below it will show how much time has elapsed since the beginning of the recording. Moreover, clicking the "Stop" button will finish the recording instantly.

As you can see, the application is pretty straightforward to use.

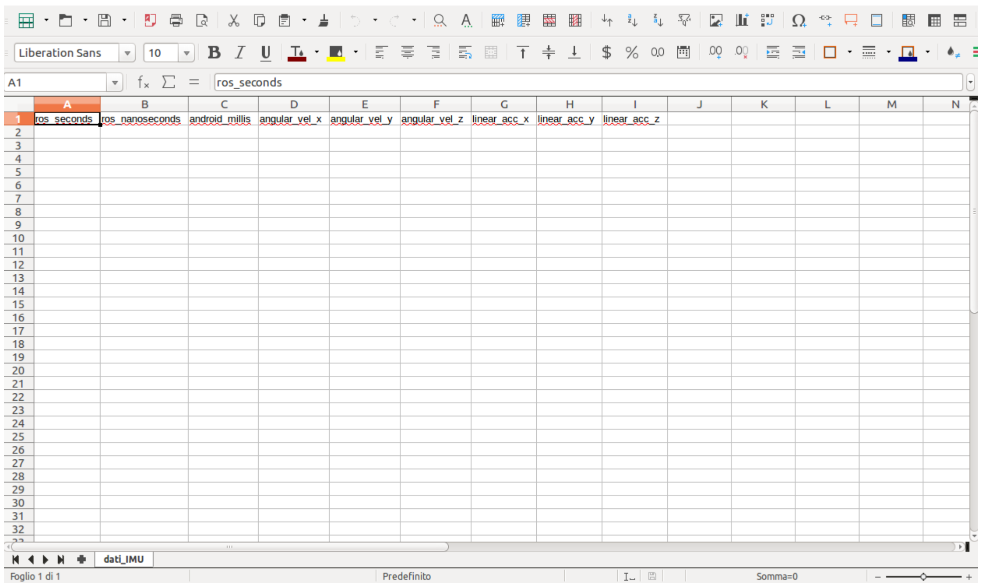
5.2. Complete Rqt_graph



This is the rqt_graph generated when all branches into GitKraken were merged with the master branch.
And thus, settling all nodes together and following the steps for the installation of the package, we should obtain the graph shown above from Ros.

5.3. "Conversion and Segmentation" Module

When the conversion code is executed, you get a csv file containing data from the bag files in input in a table form.

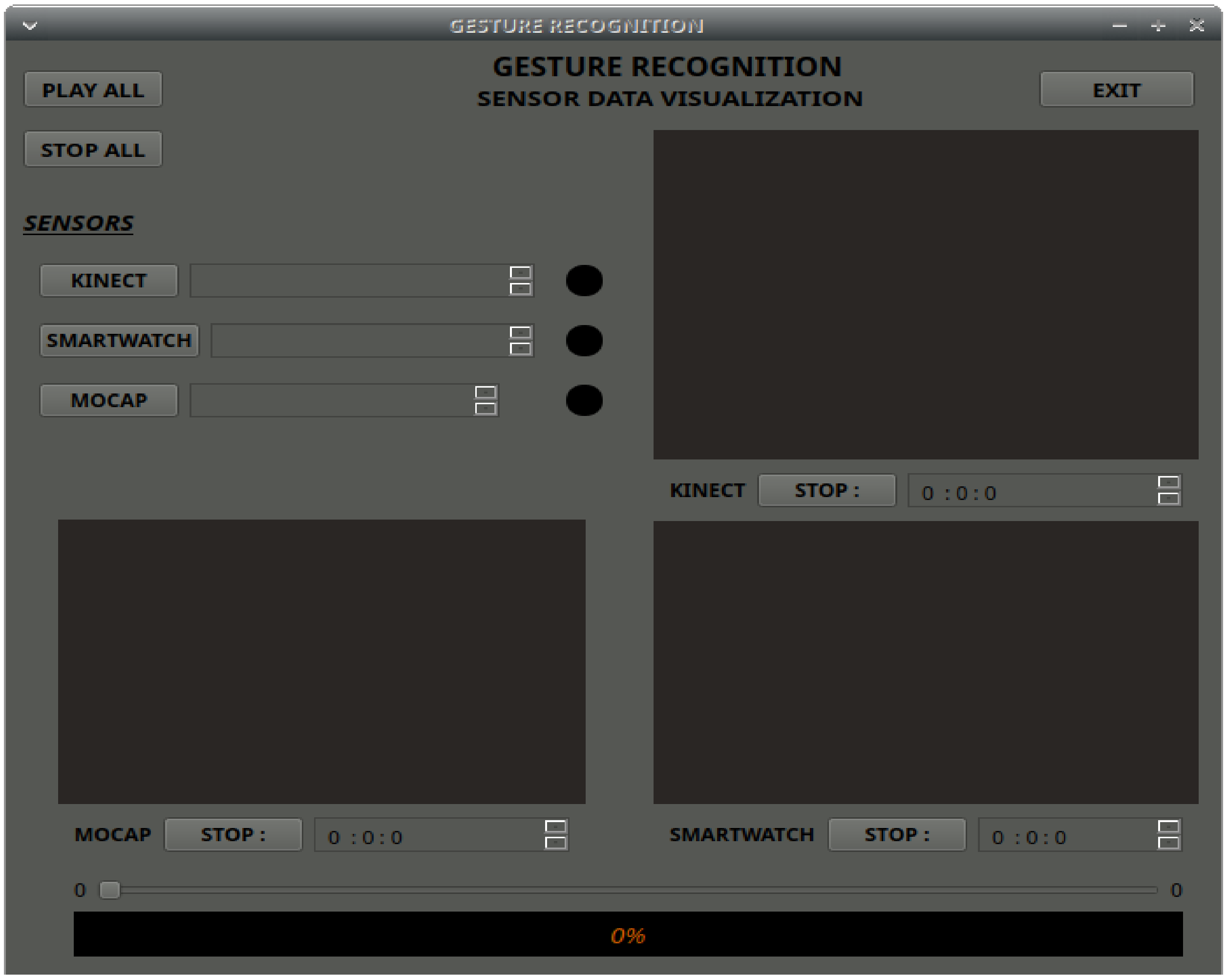


A1 ros_seconds

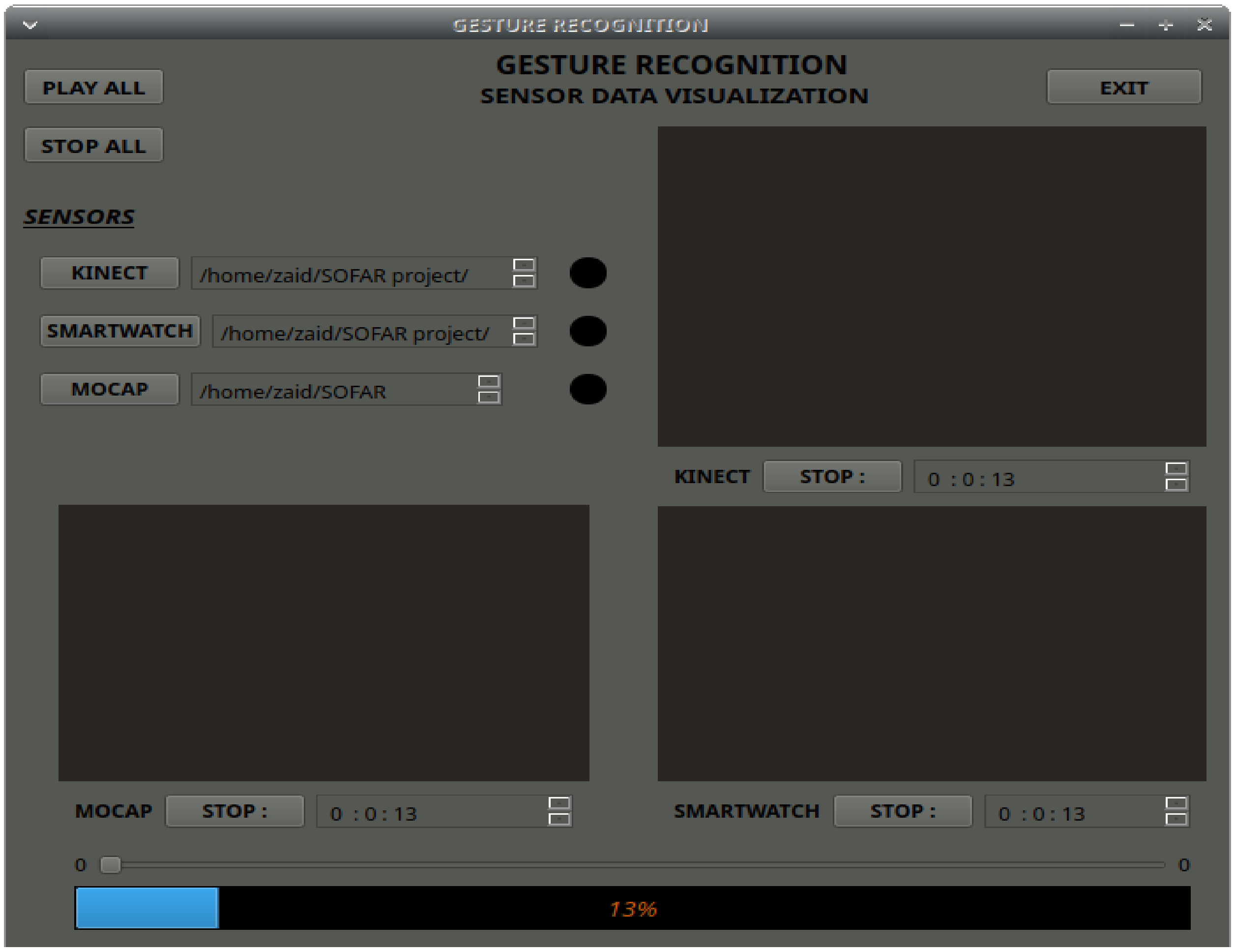
	A	B	C	D	E	F	G	H	I
1	ros_seconds	ros_nanoseconds	android_millis	angular_vel_x	angular_vel_y	angular_vel_z	linear_acc_x	linear_acc_y	linear_acc_z

5.3 Data Visualization GUI

Developed by Zaid



Datavisualization GUI is the overall graphical part of the visualization system where user can examine Rosbags i -e The already built dataset of three sensors (Kinect, SmartWatch, Motion Capture Sensor MOCAP) with their respective data types (PointCloud2, IMU, Float32). The GUI enables the user regarding the managing the configuration of Rosbags, letting the user to keep the track of time through a timer and the executed percentage of the bag files via Progressbar. GUI consists of three browsing options for the respective sensors, which allows the user to navigate through the system to a folder that contains the required bag files. Upon selecting, the path address will be displayed for an added certainty. After loading the bag files to the GUI, "PlayAll" button will initiate the Processing of the files by keeping track of them via Timer, ProgressBar and a slider. Below each of the display windows there is a STOP button which also works in individual capacity, while the StopAll button will bring the system to reset.



/home/zaid/SOFAR project/

/home/zaid/SOFAR project/

/home/zaid/SOFAR

0 : 0 : 13

0 : 0 : 13

0 : 0 : 13

0

0

13%

6. Recommendations

6.1. “Configuration UI” Sub-Module

When deciding what user’s informations should we include in the GUI, we picked the ones that seemed to be important or at least useful: obviously gathering these informations isn’t strictly a fundamental element of the architecture, but we thought that it would be a nice feature to add.

A limitation is that we don’t check for errors when the user writes his/her informations, so numbers, strings, symbols can be written in the fields: we chose not to implement a parsing of the values because it would be pretty time-consuming and, as previously mentioned, because this isn’t a very important part of the system.

Another limitation is that, if the user wants to define his/her gesture sequence, the maximum number of gestures that can be chosen is 5, whereas when asking for a casual sequence the number

is arbitrary. There is no easy solution for this, maybe the user can enter a specific string of numbers corresponding to the gesture sequence, but that would require additional parsing and it's not that user-friendly: this is why we decided to simplify this part by using only five combo boxes to choose a gesture from.
Moreover, this architecture has been thought to create multimodal datasets for five different gestures using three types of sensors: expanding the pool of gestures or implementing more sensors is by no means an easy task. One should change or add a lot of code but also modify the GUI accordingly, so this is a strong limitation.

6.2. "Communication with User UI" Sub-Module

In this sub-module we have two "soft" limitations, which can both be solved by adding a feature to the program.
The first one is the countdown duration: it's fixed to being 10 seconds, but maybe the user wants it longer, shorter or doesn't want it at all. The second one is that we assumed that every gesture in the sequence will be performed for exactly 30 seconds: again, the user might want to modify this value.
The solution to these two limitations would be adding an "Options" window where the user is free to modify these values and maybe even other ones which haven't been mentioned.

6.3. "Gesture Sequence Generator" Module

This is a quite simple module which has the only scope of generating the gesture sequence and sending it via service, so there is no particular limitation or improvement one could think of.
The only real limitation is the non-scalability in case of the addition of other gestures, but this is in accordance with the rest of the project.

6.4. "Kinect" Module

As already discussed in chapter 3, due to the current situation (Covid-19), we could not access to Lab and so we could not get real data from the sensors.
Nevertheless, whether the association with a real sensor is possible, getting real data is achievable too.
On Ros, ordinarily, recording data from Kinect, the library OpenNi is used.
The repository containing the package has been cloned by https://github.com/EmaroLab/OpenNI and the launch file should be launched before starting with the recorder (and so before launching the experimenter_GUI). In this way, immediately, a lot of topics (contained any informations concerning the received data) will be published, thus the node will subscribe to one of those which contains only interesting data.
In this case, we are interested in data regarding PointCloud2. Googling that, we discovered that this kind of message is reviewed on '/camera_depth_points' topic.

Inevitably, whether we are going to get real data, we have to modify the script of the Kinect module in such a way it will be subscribed to the topic published by OpenNi.
Therefore, doing this, the fake modules are unless.

Further notes on the installation

To use OpenNi to get real data from the Kinect some other installations are required to have the right environment.

```
sudo apt install -y freenect freeglut3* git-core cmake build-essential
libxmu-dev libxi-dev libudev* g++ python openjdk-11-jdk doxygen ros-
kinetic-rgbd-launch ros-kinetic-openni-* ros-kinetic-pcl-* ros-kinetic-
perception ros-kinetic-perception-pcl ros-kinetic-tf ros-kinetic-roslib
python-rosinstall python-rosinstall-generator python-wstool build-
essential tlp openjdk-11-jre openjdk-11-jdk gazebo9
```

```
sudo update-java-alternatives --set java-11-openjdk-amd64
```

Then, we have to install every needed libraries, cloning each repository from GitHub into a defined folder.

```
rm -rf /.dependence
mkdir /.dependence
cd /.dependence
git clone https://github.com/libusb/libusb
git clone https://github.com/OpenKinect/libfreenect
git clone https://github.com/EmaroLab/OpenNI
git clone -b unstable https://github.com/PrimeSense/Sensor
git clone https://github.com/arnaud-ramey/NITE-Bin-Dev-Linux-v1.5.2.23
```

1) Libusb Library that proved generic access to USB devices.

```
cd /.dependence/libusb
./autogen.sh
make
sudo make install
```

2) Libfreenect library for accessing the Microsoft Kinect USB camera.

```
cd /.dependence/libfreenect
mkdir build
```

```
cd build
cmake ..
make
sudo make install
sudo ldconif /usr/local/lib64/
```

3) OpenNi Library.

```
cd /.dependence/OpenNi/Platform/Linux/CreateRedist
./RedistMaker
cd ../Redist/OpenNi-Bin-Dev-Linux-*
sudo ./install.sh

cd /.dependence/Sensor/Platform/Linux/CreateRedist
./RedistMaker
cd — /Redist/Sensor-Bin-Linux-*
sudo ./install.sh
```

4) Nite Library.

```
cd /.dependence/NITE-Bin-Dev-Linux-v1.5.2.24/x64
sudo ./install.sh
```

When everything will work, the first thing to do is to launch OpenNi package to get data:

```
roslaunch openni_launch openni.launch device_id:=<device id>
```

The topics with all informations will be published and available.

6.5. “Mocap Optitrack” Module

The package "mocap_optitrack" that contains the launch file ‘mocap.launch’ should be launched before starting with the recorder. After the launch of ‘experimenter_GUI.launch’, if we could have carry out the experiment in the Lab, the node in the optitrack package that receives the real mocap data would replace the fake_mocap_node and in such a way as to allow the mocap_node to receives the signal from the GUI and the real mocap data in order to publish them into the topic /mocap_data. This topic will provide to the ‘Mocap_node’ the real markers positions in order to record them into a rosbag.
Currently, this node supports the NatNet streaming protocol v1.4 and it receives the binary packages that are streamed by the Arena software.

How Mocap Optitrack works:

Optitrack system needs software running on Microsoft windows machine to determine markers coordinates in order to calculate the right positions of the objects. The goal is to provide Optitrack measurements to programs running in ROS. Basically, there are two types of markers: active (LEDs) and passive (retroreflective). If passive markers are used in the experiment, Optitrack cameras send out pulsed infrared light which will be reflected by markers and detected by the Optitrack cameras. After some processing directly on the cameras, the live video will be streamed to a computer running the SW provided (Tracking Tools, Arena).
In the mocap_node (in the Optitrack package) is specified that the data streaming is computed by Arena software.
If we had the access to the Lab, the NanNet protocol should be used for data streaming on Windows machine combined with the working of multicast socket to receive the NaNet package. The NaNet server applications stream the following type of motion tracking data:

- MarkerSetData: a named of collection of identified markers and the markers positions.
- RigidBodyData: a named segment with a unique ID, position and orientation data and the collection of identified markers used to define it.
- SkeletonData: a named hierarchical collection of RigidBody.

6.6. “Conversion and Segmentation” Module

In this module you could have a little accurate segmentation. AsI have already said in Chapter 3, there may be a limit due to the fact that you might encounter a limit due to the fact that the person does not perform the gesture immediately.
The solution to this problem could be the use of a wider time interval not to be segmented that refers to the moment when the person is stationary both before and after performing the gesture.
In seeking better segmentation accuracy, however, there will be greater computational complexity. This module can be executed offline to make sure that you can access the rosbags at any time. Since it is not possible to use the laboratories, offline mode is the most logical choice, as it is not allowed to acquire data in real time.

7. Real Time Data Visualization

Developed by Chetan, Surendra

The already built dataset of three sensors (Kinect, Smart Watch, Motion Capture Sensor MOCAP) with their respective datatypes (PointCloud2, IMU, Float32). The GUI enables the user to manage the configuration of CSV files, lets the user keep the track of time through a timer and the executed percentage of the CSV files via Progress bar. The progress bar contains scatter, Browse Button which enables the User to connect GUI with the loaded files. GUI consists of three browsing options for the respective sensors, which allows the user to navigates through the system to a folder that contains the required bag files. Upon selecting, the path address will be displayed for an added certainty. To make the process simpler, we added two extra buttons, one for browsing and the other for running. So, in this case, the user need not access each file. He can simply click on the "Add" icon and opens the file and then by clicking the "play" button the data visualize. Also, Below each of the display windows, there is a STOP button, and an exit button to end the GUI.

The unavailability of lab due to the pandemic has made the work more complicated. In reality, all the data we are obtained by working on real sensors. So, as the work is carried away online, we cannot do this, we use the fake datasets to visualize the data.

7.1 GUI_Old

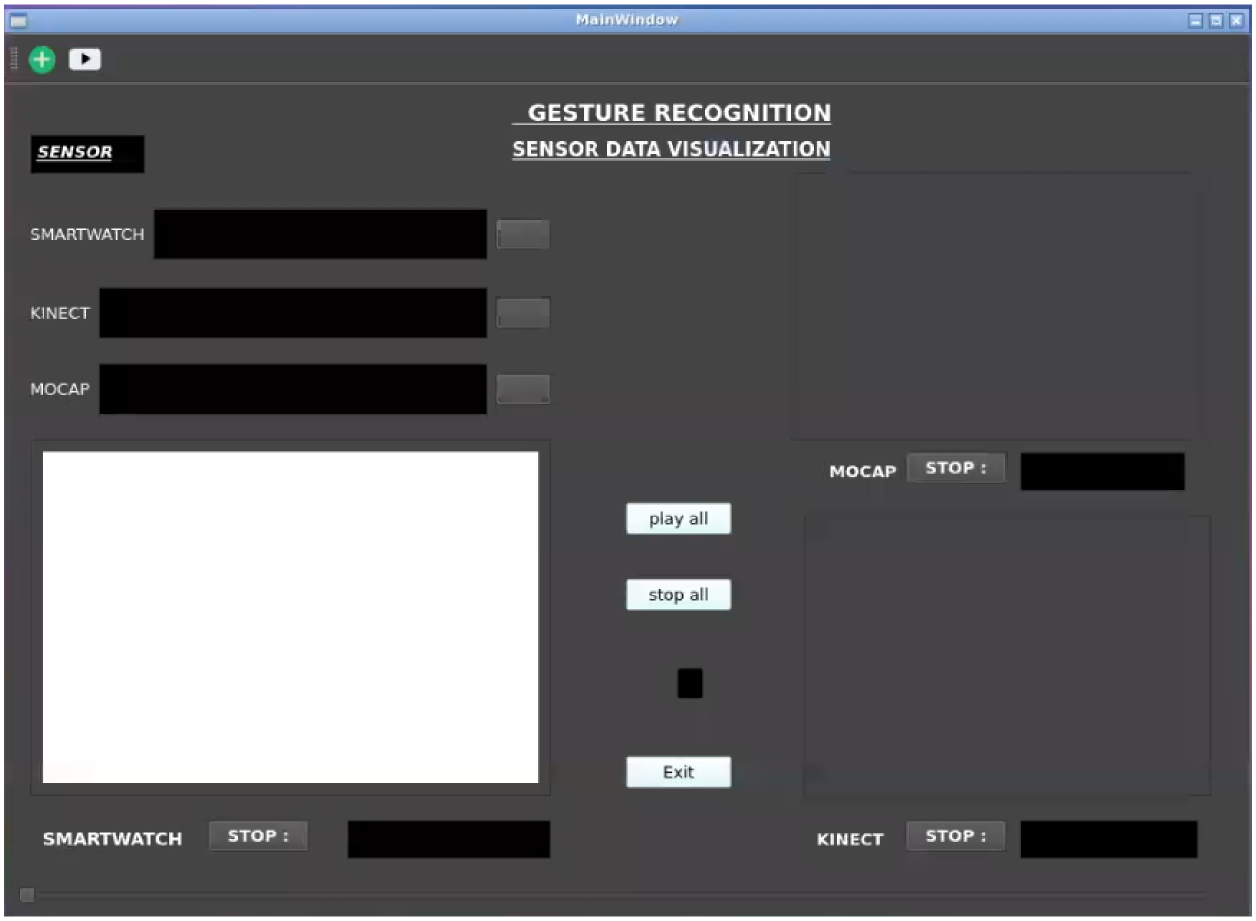
To begin with the data visualization module, the main aim is to visualize the data using multiple sensors data. For instance, we use IMU, Mocap, Point cloud sensors data to read the user gestures(movements) and visualize the raw output in the form of graphical visualization.

Our batchmate, Zaid has worked on designing the GUI and the data visualization part was done by Surendra and Chetan.

7.2 GUI_Updated

To visualize the IMU data, updating the GUI plays a major role. GUI has been modified by adding some extra push buttons. Originally, we have browsing buttons for each sensor. But as a part of the modification, we added a new toolbar at the top of GUI, to browse(Add icon) and visualize(play icon) the CSV files just by using a single button instead of clicking each button every time. So, in this case, if the user uses only a single CSV file, it is more feasible to browse.

Also, we added the line edit widget, if the user wants to visualize data in an extensive manner. As MOCAP and Kinect data are not used, We increased the size of the IMU visualizing screen to be visible.



GUI Main Window Updated Version

7.3 Installation and Running

The first thing to do, after having cloned the repository :

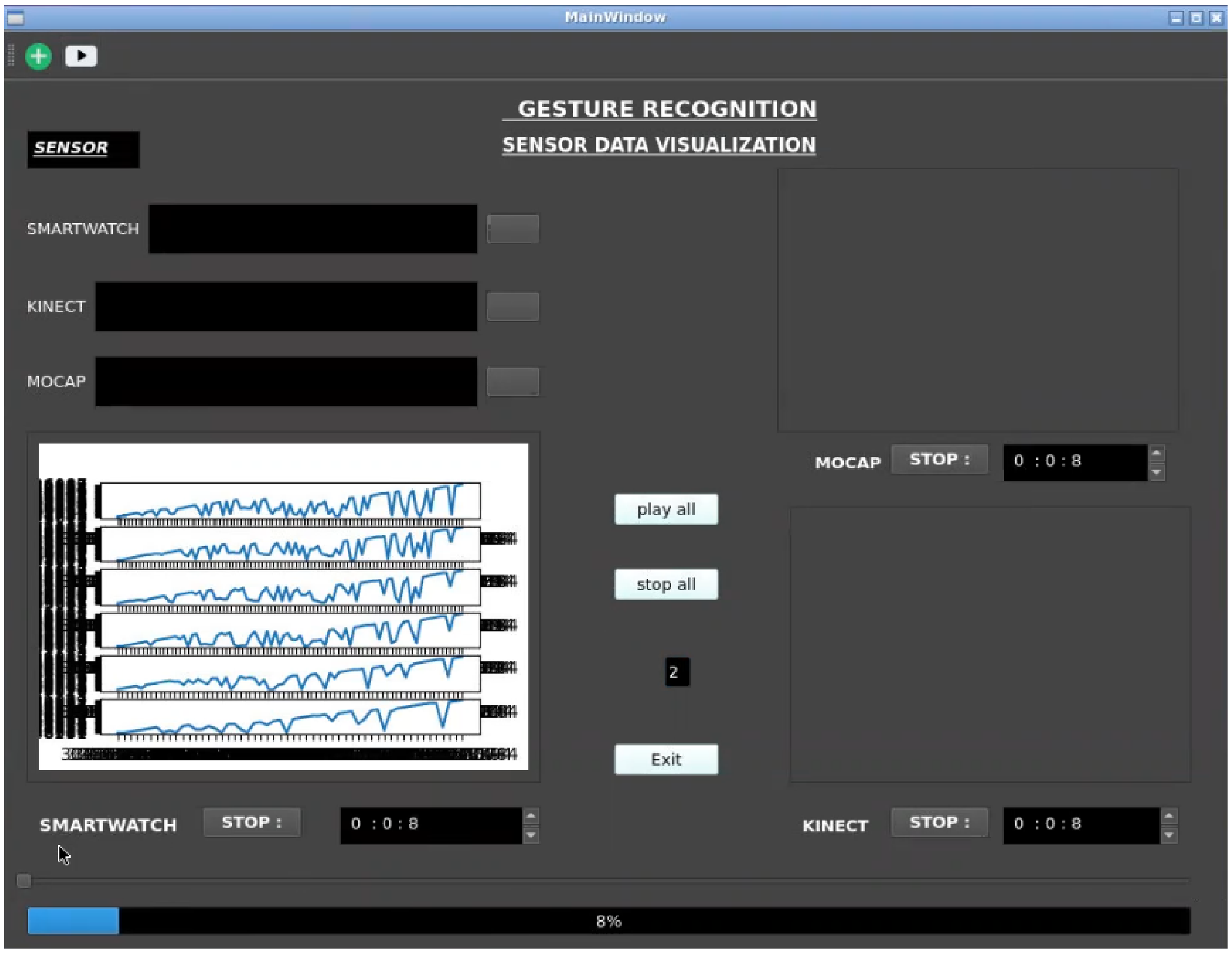
Execute the following commands for Python libraries

sudo apt-update
sudo apt install python3-pip
sudo apt-get install python3-yaml
pip3 install --user pyqt5
sudo apt-get install python3-pyqt5

To run the system:

python3 updated_gui.py

This individual python script can also be run from and python Integrated Development Environment (IDE) but make sure to have the project environment built accordingly with the required libraries installed i-e python and PyQt5 libraries



Visualizing IMU data using CSV file

7.4 Sensors Data

GUI has three sensors to visualize, Smart Watch(IMU), Motion Capture(MOCAP), and Kinect(Point Cloud). But, for instance, we have worked on IMU sensor data to visualize. We can either use ros bags or CSV files as the input in the GUI to visualize the output.