Software Architecture Project

**Student(s)**: Laiba Zahid, Aliya Arystanbek, Daulet Babakhan, Amanzhol Raisov, Latif, Manoj Kunapalli, Vishruth XXX, Latif Xeka, Syed Muhammad Raza Rizvi

**Tutor(s)**: Fulvio Mastrogiovanni

**Supervisor(s)**: Yusha Kareem, Antony Thomas, Luca Buoncompagni

**Year**: 2019-2020

Commanding MiRo with natural language [Project 12b]

# Table of Contents

The Table of Contents helps the reader to find specific information and indicates how the report/article has been organised.

# Abstract

It presents briefly, in a single paragraph, the objective of this project. **Note that**: At the end of the paragraph, please add the link to your project’s GitHub repository.

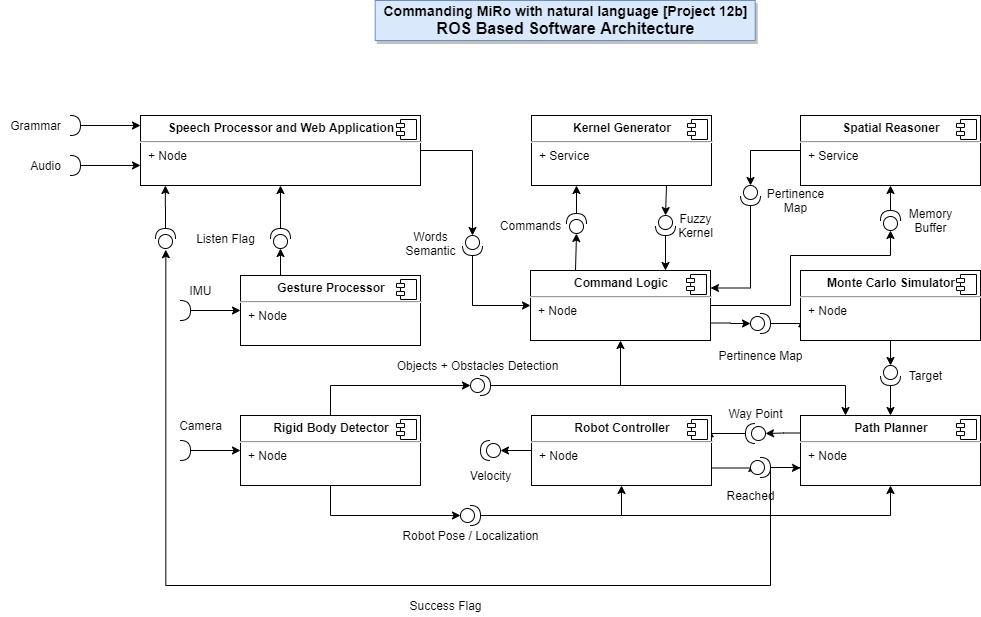
# Introduction

It describes what the project wants to achieve and defines all its underlying terminologies. It introduces the hardware and software tools used in the project. **Note that**: this section is a “common” part and is the responsibility of all students working on the project.

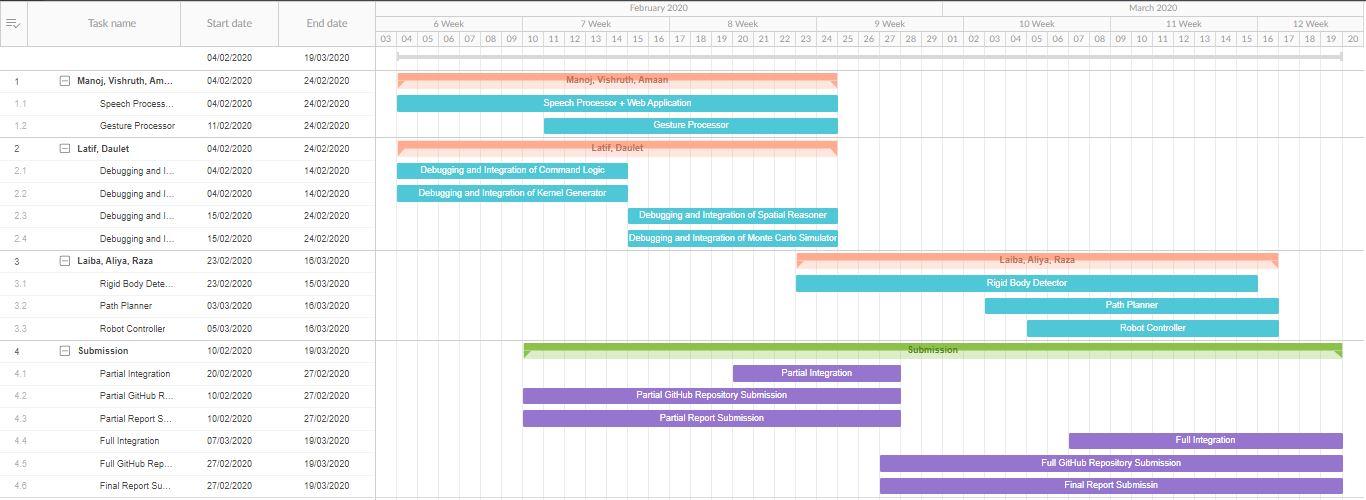
# Architecture of the System

It presents the overall architecture of the system with the help of a UML and it describes in simple words the overall architecture, which is designed to meet the objective of the project. **Note that**: this section is a “common” part and is the responsibility of all students working on the project.

2.1 UML



2.2 Work Plan



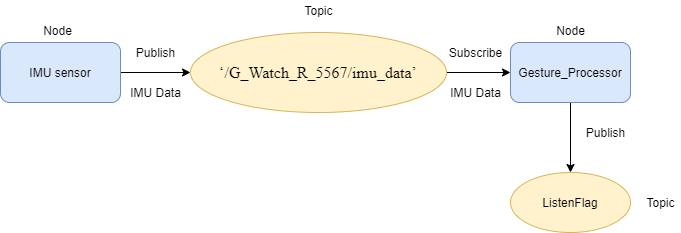
# Description of the System’s Architecture

This section presents in a detailed manner (in its sub-sections) each module within the architecture. **Note that**: each sub-section (which is a module) is the responsibility of the students working on that module.

## Module <Gesture\_Processor>

The Module <Gesture\_Processor> was developed by Amanzhol Raisov.

The node ‘Gesture processor’ subscribes to the information published by the node ‘IMU sensor’ and the sensor is inside the smartwatch which the user is wearing. This sensor is constantly publishing the IMU data on the topic. And the Gesture Processor node has subscribed to this topic. So, whatever information data has been published by the IMU sensor, it is received by the Gesture Processor. The Gesture processor then processes that information and tries to recognize, if a certain gesture has been performed or not. As soon as the gesture processor recognizes the required gesture, it publishes ‘1’ on the topic ‘ListenFlag’, else it publishes ‘0’. In our case, the required gesture is ‘up’.



## Module <name of the module>

This subsection describes the module in detail, i.e, **(i)** names of the key people who developed the module, **(ii)** the prerequisites (e.g., all the hardware and software) required for the module, **(iii)** the inputs to the module, **(iv)** the internal working of the module, and **(v)** the outputs of the module.

## Module <name of the module>

This subsection describes the module in detail, i.e, **(i)** names of the key people who developed the module, **(ii)** the prerequisites (e.g., all the hardware and software) required for the module, **(iii)** the inputs to the module, **(iv)** the internal working of the module, and **(v)** the outputs of the module.

# Installation

This section presents (in its sub-sections) how to install and run the module. **Note that**: if all the modules have successfully completed their work and integrated everything together, then this section can present the overall installation of the “whole” system, instead of having a sub-section dedicated to the installation of each module.

## Module < Gesture\_Processor >

Before starting using Ubuntu 16.04 , make sure, that the Base Memory of the system (see on Virtual Box) has enough space so that in the future the system works correctly without failures.

Smartwatch is connected to the ROS master through the Wi-Fi (EmaroLab-WiFi). Find the IP address of your ROS master that is on the computer using this command:

Ifconfig

Now, connect your Smartwatch with the ROS master by inserting the IP address of ROS master, inside the Smartwatch.

Note: In the case, when you use Oracle VM VirtualBox, open the Settings, open the Network, inside the “Adapter1”, change “Attached to:” to Bridged Adapter.

1. Install the Python development environment on your system

Check if your Python environment is already configured:

python --version  
pip --version  
virtualenv --version

Otherwise, install Python, the pip package manager, and Virtualenv.

Link to install Python: <https://askubuntu.com/questions/1050084/install-python-2-7-6-in-ubuntu-16-04>

Link to install Pip: <https://pip.pypa.io/en/stable/installing/>

On Ubuntu:

sudo apt update  
sudo apt install python-dev python-pip  
sudo pip install -U virtualenv  # system-wide install

Python virtual environments are used to isolate package installation from the system.

Create a new virtual environment by choosing a Python interpreter and making a ./venv directory to hold it:

virtualenv --system-site-packages -p python2.7 *./venv*

Activate the virtual environment using a shell-specific command:

source *./venv*/bin/activate  # sh, bash, ksh, or zsh

When virtualenv is active, your shell prompt is prefixed with (venv).

Install packages within a virtual environment without affecting the host system setup. Start by upgrading pip:

pip install --upgrade pip  
  
pip list  # show packages installed within the virtual environment

And to exit virtualenv later:

deactivate  # don't exit until you're done using TensorFlow

Each tensorflow version works only with specific version of Keras. Install tensorflow 1.10 and keras 2.1.6.

To install tensorflow 1.10 use following command:

pip install tensorflow==1.10

To check the version use:

pip show tensorflow

To install Keras 2.1.6. use following command:

pip install keras==2.1.6

To check the version use:

pip show keras

Clone the directory from GitHub by using the following command:

cd Downloads/

git status

git clone https://github.com/robotmiro1/Commanding-MiRo-with-natural-language.git

Jump to the Module Directory:

cd catkin\_ws/src/Gesture\_Processor/src/

Now, Run the ROS Master:

roscore

Now Run the Python Node:

python gesture\_recogniton.py

Visualize Published Messages:

Rostopic echo ListenFlag

Visualize the Nodes and Topics details using:

rostopic list

rostopic info /ListenFlag

rosnode list

rosnode info /G\_Watch\_R\_5567\_IMU

## Module <name of the module>

Please keep in mind, **do** **not** include in your repository the “entire” code of the external libraries that your module may use. Hence accordingly, **describe** to the new users how they can “install” the external libraries and then **describe** how they can “install” your module that uses those libraries. Afterwhich, **describe** how to run your module.

## Module <name of the module>

Please keep in mind, **do** **not** include in your repository the “entire” code of the external libraries that your module may use. Hence accordingly, **describe** to the new users how they can “install” the external libraries and then **describe** how they can “install” your module that uses those libraries. Afterwhich, **describe** how to run your module.

# System Testing and Results

This section presents (in its sub-sections) the testing-result of each module. **Note that**: if all the modules have successfully completed their work and integrated everything together, then this section can present the overall testing-result of the “whole” system, instead of having a sub-section dedicated to the testing-result of each module.

## Module <Gesture\_Processor>

The corresponding rqt graph of the working system is shown down below:

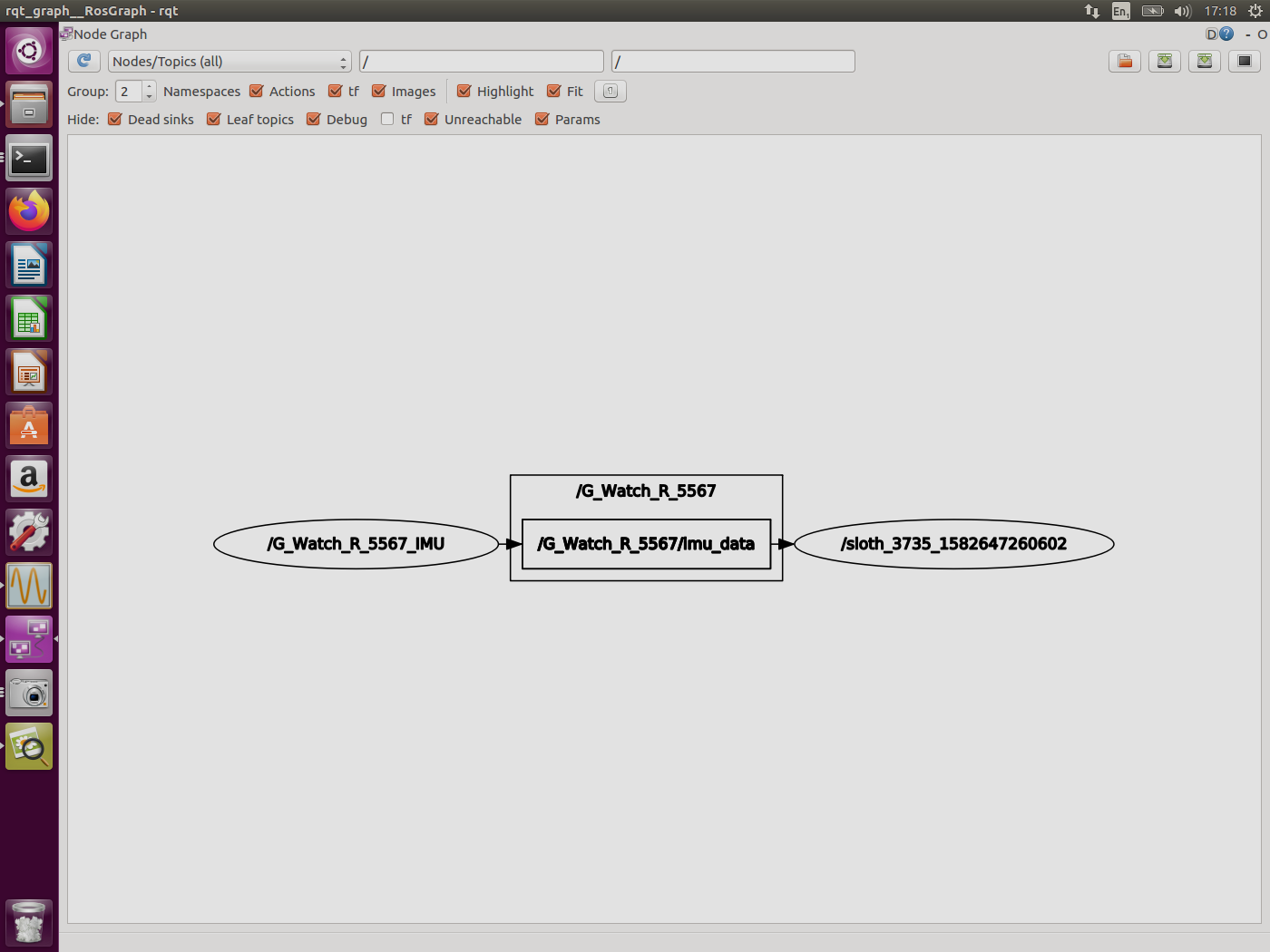
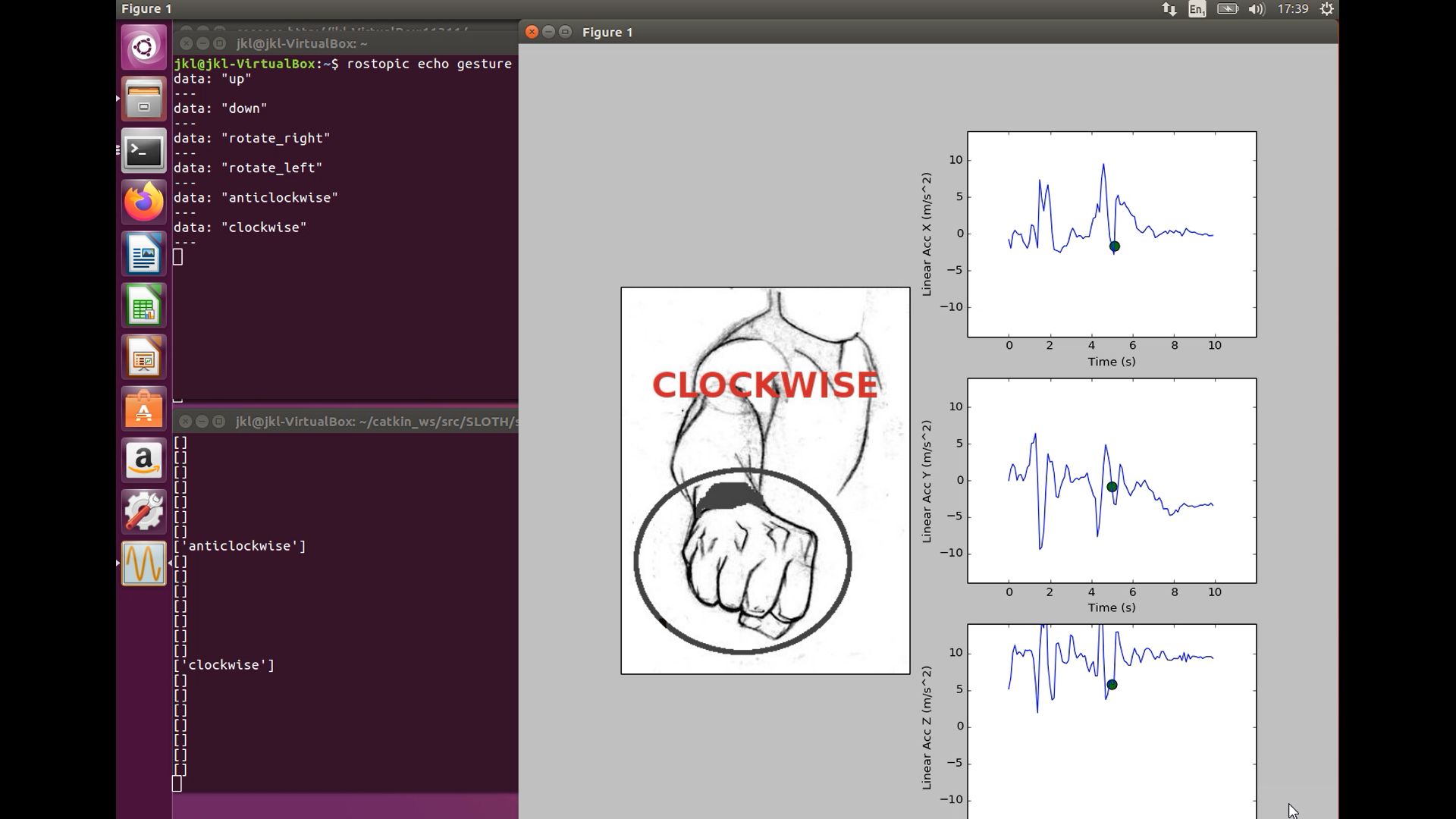


Figure 1: rqt graph for the Gesture Processor

After building the correspong package, It has been possible to test its results:



Especially, Our data package in Gesture\_Processor will only work with a specific Smart Watch model.

The link to the video is presented down below:

<https://www.youtube.com/watch?v=3E2BOBpMcWM&feature=youtu.be>

## Module <name of the module>

This subsection presents the testing-result of the module, i.e., **(i)** the rqt\_graph generated when the module is running, **(ii)** images or links to the videos showing the working of the module (in real or in simulation), and/or **(iii)** numeric results.

## Module <name of the module>

This subsection presents the testing-result of the module, i.e., **(i)** the rqt\_graph generated when the module is running, **(ii)** images or links to the videos showing the working of the module (in real or in simulation), and/or **(iii)** numeric results.

# Recommendations

The recommendations follow naturally from the system testing and results. **Note that**: if all the modules have successfully completed their work and integrated everything together, then this section can present the overall recommendations for the “whole” system, instead of having a sub-section dedicated to the recommendations for each module.

## Module <Gesture\_Processor>

During testing, it is recommended to change the frequency of data exchange from 50 Hz to 10 Hz in the IMU Wear for ease of use.

During the connection of Smartwatch with ROS, problems may occur on the Smartwatch this type of error as “master unreachable”. In this case, it is recommended to turn off the system completely, and also turn off the Smart Watch and try the procedure again.

## Module <name of the module>

This subsection presents the recommendations for the module, i.e., **(i)** the assumptions made while building the module (and/or) the limitations of the working module, **(ii)** presenting possible ideas that could overcome the limitations or assumptions.

## Module <name of the module>

This subsection presents the recommendations for the module, i.e., **(i)** the assumptions made while building the module (and/or) the limitations of the working module, **(ii)** presenting possible ideas that could overcome the limitations or assumptions.

References

[1] Alessandro Carfì. Gesture recognition method.

<https://github.com/EmaroLab/SLOTH/tree/25011180d90dd264207a32ee609a9df5d163d612> , 2018.