Commanding MiRo with natural language [Project 12a]

**Student(s)**: Jacopo Favaro, Fabrizio Zavanone, Muhammad Talha Siddiqui, Muhammad Tahir, Muhammad Sayum Ahmed, Syed Hani Kazmi Hussain

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

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

**Year**: 2019 - 2020

# Table of Contents

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

# Abstract

MiRo is a companion robot aimed to foster public engagement with robots. In this project we try to build a system to control MiRo and make it perform simple actions using voice commands. The robot looks for known keywords in the command and tries to interpret them, understanding the action they convey and perform it. The code for this project can be found in [this repository](https://github.com/Thsuva/MiRo-Project12a).

# 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

The general structure of the system is shown in figure 1 below. The state machine is treated as a component which is subscribed to /speech\_to\_text and some MiRo topics. It also publishes to MiRo through the /miro/rob01/platform/control topic complex messages. The whole structure of the state machine is, instead, shown in figure 2. In figure 3, in the end, is shown our workplan through a Gantt chart.

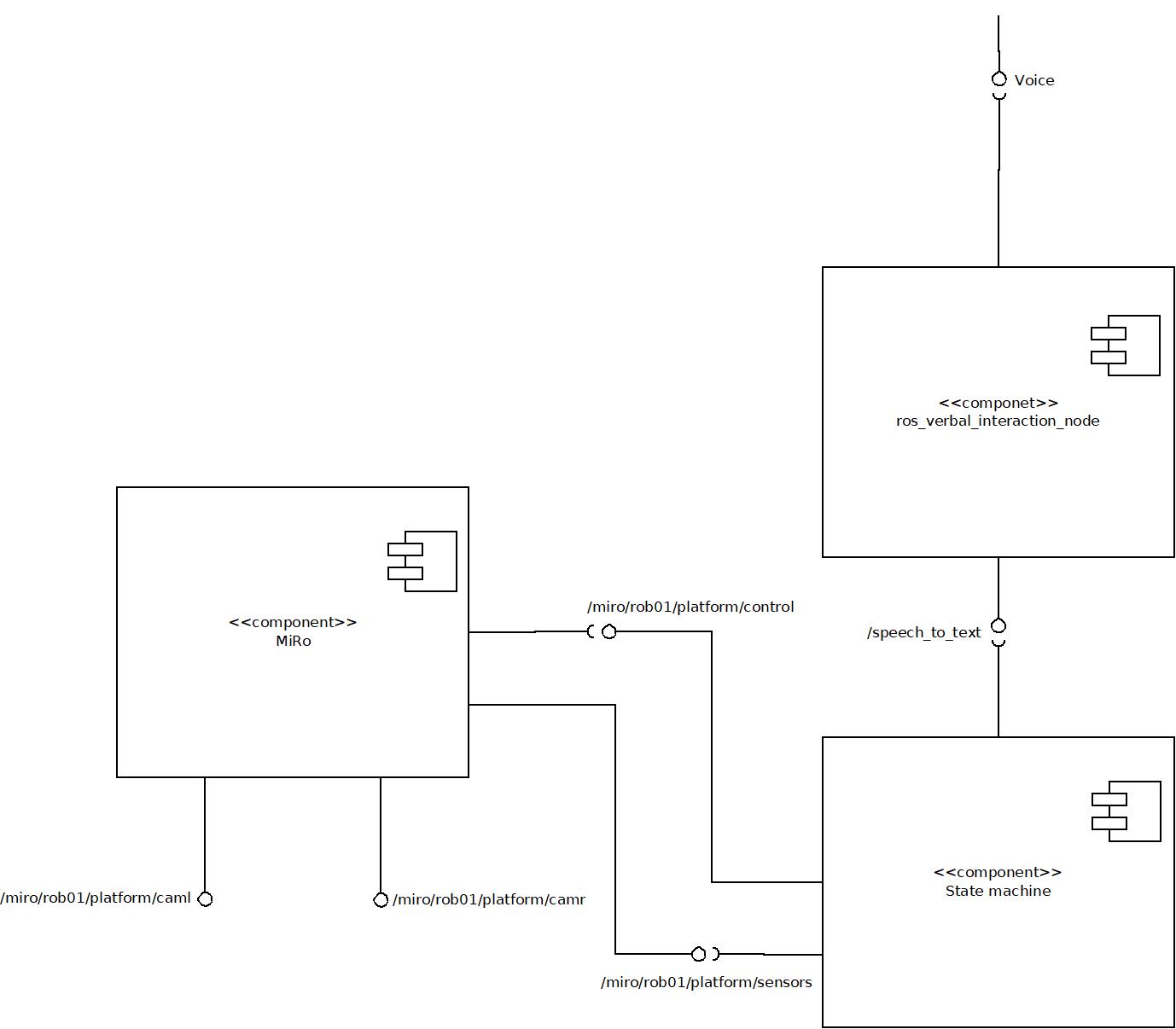


Figure : Component diagram

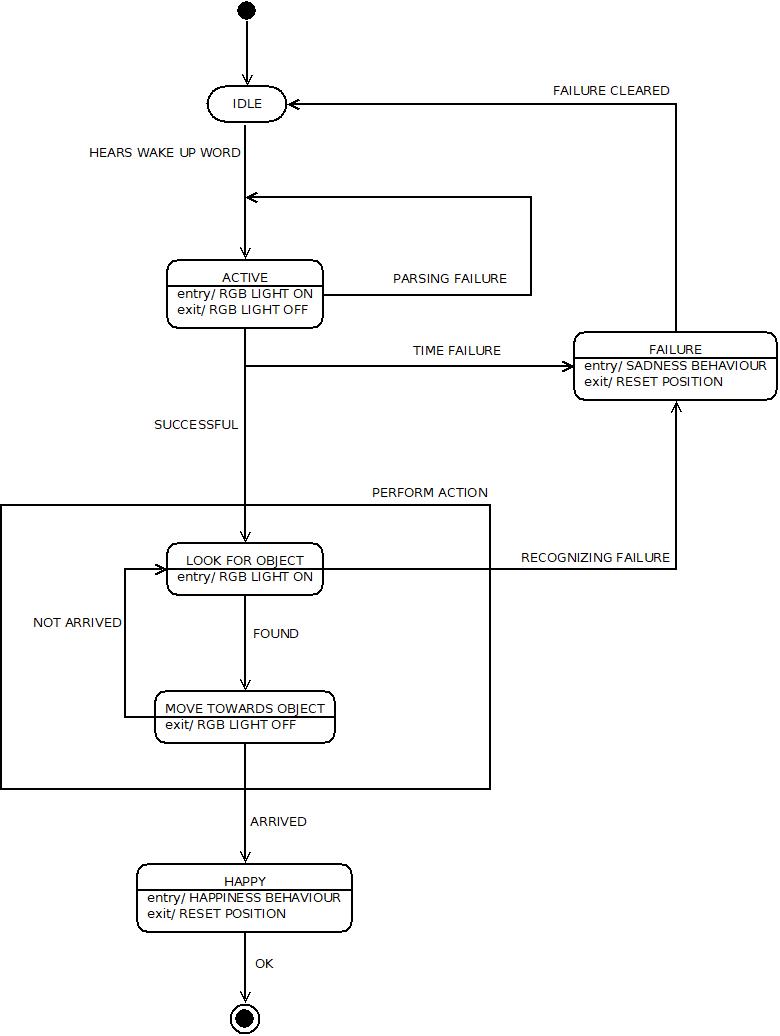


Figure : State Machine

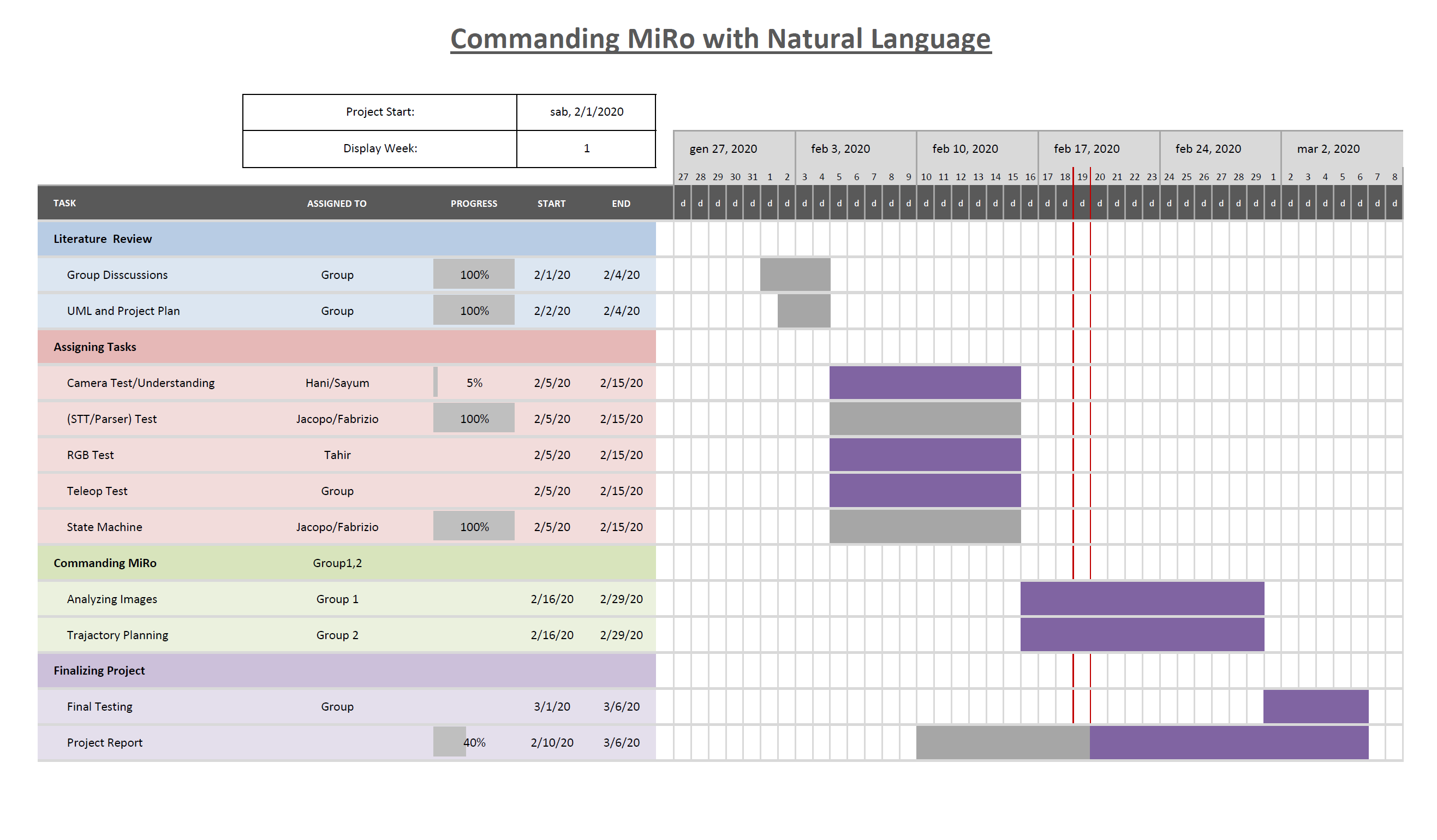


Figure : Workplan

# Description of the System’s Architecture

The project is developed using Ros Kinetic distro and also strongly depends on some common prerequisites:

* [MIROapp v1.0](http://labs.consequentialrobotics.com/download.php?file=miroapp-200107.apk): Android app needed to connect MiRo to your system.
* [MiRo Developer Kit](http://labs.consequentialrobotics.com/miro-b/software/) (release 180509): needed to set up your workstation and get ready to work with MiRo.

## Module <speech to text>

This module, that handles speech to text conversion, is taken from [this repository](https://github.com/EmaroLab/ros_verbal_interaction_node.git), which contains a web interface based on Google Speech Demo. Once runned, text converted from an audio input will be published on /speech\_to\_text alongside with its confidence and detected language. The interface also handles text to speech, but for our project we simply decided to discard this part by publishing on an unusubscribed topic. To do so, modify [speech\_web\_interface.html](https://github.com/EmaroLab/ros_verbal_interaction_node/blob/master/java-script/speech_web_interface.html), changing the topic name from /text\_to\_speech to something else.

## Module <state machine>

The state machine is the backbone of the project, developed by Jacopo Favaro and Fabrizio Zavanone. It uses smach and python 2.7 to build the state machine object and its states. The speech to text module is necessary to run this part and provides it with the desired command derived from vocal input to MiRo. The majority of the states are in fact subscribed to the */speech\_to\_text* topic. (TODO: opencv topic subscription)

The state machine object is built and starts its execution in the src/state\_machine\_main.py file. All the states are written as classes in different python files stored in the src/states folder. Starting from the “idle” state in which MiRo listens to messages published in */speech\_to\_text* topic, parses them through a parser function present in the *src*/*parser/parser.py* file, and waits for the activation word, switching then to the “active” state when it is heard. Here, using again the same structure previously mentioned to gather vocal command, it waits for a sentence containing all the needed information (action to complete, color of the target, shape of the target). If this happens, through the “successful” outcome, we send a list containing the three key elements previously parsed to a sub state machine called “perform action”. Here a loop between the “look for object” and the “move towards object” states allows MiRo to fulfill its goal. There is the possibility, both in the “active” and in the “look for object” states, that a timeout will lead MiRo to a state called “failure”. In this state it shows disappointment to the user for not been able to accomplish what was requested, before resetting itself back to “idle” state. The only remaining state is the “happy” state in which MiRo, once reached its goal, shows its happiness to the user that can also interact with it for a limited amount of time.

The module publishes to the */miro/rob01/platform/control* topic messages to get both simple and complex behaviors from MiRo as feedback from both “failure” and “happy” states.

## 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

In order to install the whole system, some common steps are necessary:

* Install ROS Kinetic following this [guide](http://wiki.ros.org/kinetic/Installation/Ubuntu).
* Install MiRo Developer Kit, instructions are provided [here](https://consequential.bitbucket.io/Developer_Preparation_Prepare_workstation.html).

## Module <speech to text>

To install this module, clone the repository in your catkin workspace’s src folder using:

* git clone <https://github.com/EmaroLab/ros_verbal_interaction_node.git>
* cd ..
* catkin\_make

## Module <OpenCV apps>

To install this module simply type in a terminal:

* sudo apt install ros-kinetic-opencv-apps

## Module <state machine>

To install the rest of the project, clone the repository in your catkin workspace’s src folder using:

* git clone <https://github.com/Thsuva/MiRo-Project12a>
* cd ..
* catkin\_make

# 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.

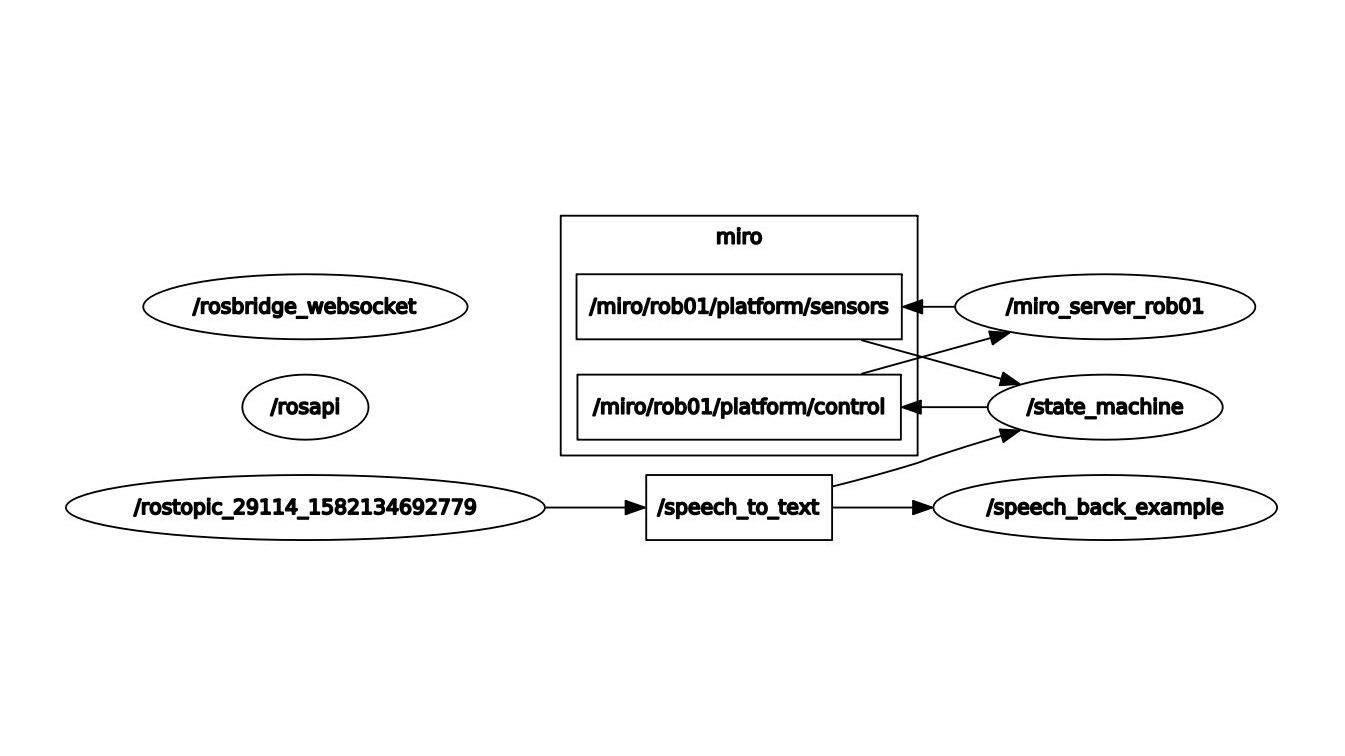
## Module <state machine>

These tests, carried on by Fabrizio Zavanone and Jacopo Favaro, consist of the simulation, using all the possible inputs, in order to obtain all the outcomes and behaviour of the state machine and seeing their result on the robot.

The first test was made with a complete and correct input to reach the happy state. [Here](https://unigeit-my.sharepoint.com/:v:/g/personal/s3947407_studenti_unige_it/ESkJzJxdhqROq0a3JLB8qZ8Bw03ZDNqTLj3ChaNlXdOLrw?e=xmkb9y) you can find a video that shows MiRo’s behaviour and [here](https://unigeit-my.sharepoint.com/:v:/g/personal/s3947407_studenti_unige_it/EQMeZvnLBx9OluR2AFkJLnYBAy0914OEhMV20R6UC7Ny_g?e=WnhXjn) you can see a screen recording of the terminal working on this task, where you can better appreciate the flow of the state machine.

In the second test, instead, MiRo is given a nonsensical order, which can’t be parsed thus resulting in a parsing error and then, since no other order is given, a timeout error occurs, leading the state machine to the failure state. After performing the “sad” behaviour of this state, MiRo returns idle and ready for new command inputs. A video that shows MiRo’s sad behaviour can be found [here](https://unigeit-my.sharepoint.com/:v:/g/personal/s3947407_studenti_unige_it/ET7sVmuhv5JAlO0pEUt8FDcBmhItLRQtnhiZq7-rXuLdyA?e=jLgAcA), while the terminal running on the task can be seen [here](https://unigeit-my.sharepoint.com/:v:/g/personal/s3947407_studenti_unige_it/ERFIsZhbV3RBkqEenLSmBmQBOJXQS6WQeMMWzCbNfcP9Sw?e=ETsVVp).

We show, below, the rqt graph of the architecture while these tests were running.



## 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

As a general recommendation, be sure to launch the app to connect to MiRo only after having launched the state\_machine.launch file, otherwise this will result in a failure of connection (MiRo topics won't be present in your rostopic list).

## Module <speech to text>

The module has been proven to be robust in recognizing most of the test commands. It is however necessary to avoid the combination “red ball”, which is often confused with “red bull” and we suggest using “red circle” instead. It is also necessary for the tool to work properly that the environmental noise is reduced as much as possible, for the tool waits for silence before putting the previous sound in to text.

## 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.