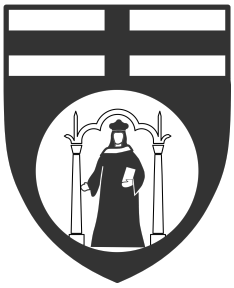

Universitá degli Studi di Genova

Software Architecture Project

Semantical Monocular SLAM

Student(s): Andrea Arlotta, Giovanni Battista Borré, Tommaso Gruppi
Tutor(s): Fulvio Mastrogiovanni
Supervisor(s): Syed Yusha Kareem, Antony Thomas
Year: 2018 - 2019

Contents

1 Objective of the Project

The objective of this project is to develop an architecture that allows a mobile robot to perform autonomous exploration and navigation in the world, merging both environment and the information in order to develop a semantic map of the surroundings.
For this project the mobile robot chosen is MiRo designed by Consequential Robotics [1] and which will be described in detail in the next paragraph. More specifically MiRo must be able to perform the SLAM (Simultaneous Localization And Mapping) task taking into account also the semantic information about the external environment.
To reach this goal the project has been divided into five modules, each one of which has to deal with different types of information extracted by means of various sensors from the external environment. Furthermore, while the mobile robot navigates through the environment, these acquired data, provided by multiple perception systems, have to be merged among modules in order to be able to extract the desired information and command to the robot and consequently to reach the goal desired.
The system will find the correlations among the objects recognized by all the perception modules, will merge all the information and then create a topological map. Finally, the system enables the user to give some commands (i.e., using keyboard) that correct labels of recognized objects (i.e., deleting or changing) or perform other tasks, like reaching a position of a specific object or a bluetooth beacon previously detected.

1.1 MiRo

MiRo (shown in Figure 1) is a fully programmable autonomous robot for researchers, educators, developers and health care professionals.
It is the first robot in a development program that aims to accelerate progress in social and companion robotics research and to foster public engagement with robots. MiRo also makes a great show-piece, demonstrating as it does the engaging nature of robots built on biological principles of morphology and behaviour [1]. MiRo has on board six exteroceptive senses, an innovative brain-inspired control system, a simulation software package and it has eight degrees of freedom. MiRo is a flexible platform suited for developing companion robots as in the future the social robots will interact with each other and will be capable of providing emotional engagement and entertainment.
MiRo's built-in "biomimetic core", an implementation of a set of components from the Biomimetic Brain-Based Control System (3B-CS), generates

life-like behaviour out of the box.
MiRo has two stereo cameras, one in each eye. The cameras perform automatic image adjustment (e.g. exposure) and focus is fixed with a large depth of field. MiRo's eyes are fixed in the head. Each camera has an horizontal/vertical field of view of 120/62 degrees and an aspect ratio of 16:9 [1] (pixel aspect ratio is 1:1). The stereo overlap region is a little more than 60 degrees and the two cameras together provide a wide horizontal field of view of nearly 180 degrees.

Figure 1: MiRo by Consequential Robotics.

2 System Architecture

2.1 Overall Architecture

This project is divided as aforementioned into five different modules, each one of them dedicated to different tasks and described more in details in the following:

- **Module 1 - Navigation:** This module allows MiRo to navigate through the external environment with two different modalities: autonomous or manual. This module also includes the obstacle avoidance task. It takes as input: the sonar data from MiRo platform, the pose of

the mobile robot with respect to the world reference frame from Module 2 and, the movement modality and the position of the goal that has to be reached, from Module 5. Its output consists of the motion commands that must be sent to the platform and a message addressed to Module 5 to warn once the goal has been reached;

- **Module 2 - Monocular SLAM based depth perception:** This module recognizes distances to objects/obstacles, acquiring depth information from monocular cameras while the mobile robot is moving in the external environment. Moreover, as soon as it receives a message from one of the other modules, it also determines the position associated to the semantic data provided: in this way we are able to work out a semantic topological map. This module takes as input the images acquired from the camera and a message composed by a label from Module 3 or 4 every time they detect respectively an object or a beacon. In principle any kind of sensor providing semantic information can be used to construct the semantic map. This is possible thanks to our adapters based approach: each message is converted to a semantic information starting from a label. Then, once the label message is received, the pose of the robot is extracted and the label received is associated to the corresponding set of coordinates on the horizontal plane (*semantic point*). The outputs of this module are the topological semantic map which can be stored and sent to Module 5 and the pose of the robot that will be provided to Module 1;

- **Module 3 - RGB based object recognition:** This module uses a neural network to recognize objects (e.g., table, chair, fridge, garbage bin, etc.) determining their positions in terms of bounding boxes and names (i.e. labels). This module takes as input only the images acquired from the camera and it executes the object detection task whose output is sent to Module 2;

- **Module 4 - Bluetooth beacons based distance estimation:** This module uses RSSI signal strength with respect to estimated beacons (which are pre-tagged with labels such as "Kitchen", "LivingRoom", etc.) in order to compute the distance between the robot and the bluetooth beacon detected. The input of this module is the RSSI signal strenght, depending on which this module detects the bluetooth beacon and once detected, sends a message composed by the corresponding label to Module 2.

- **Module 5 - Graphic User Interface:** This module provides a GUI

in order to allow the user to interact with the robot through specific tools (i.e. when the robot is in manual mode the user can control its movements by means of the joystick, he/she can also edit labels, visualize map and other functionalities). This module receives as input the topological semantic map from Module 2 and a message from Module 1 once the goal has been reached; its output is sent to Module 1 and is composed by the modality according to which the robot will move and, if needed, also by the position of the goal that the robot has to reach.

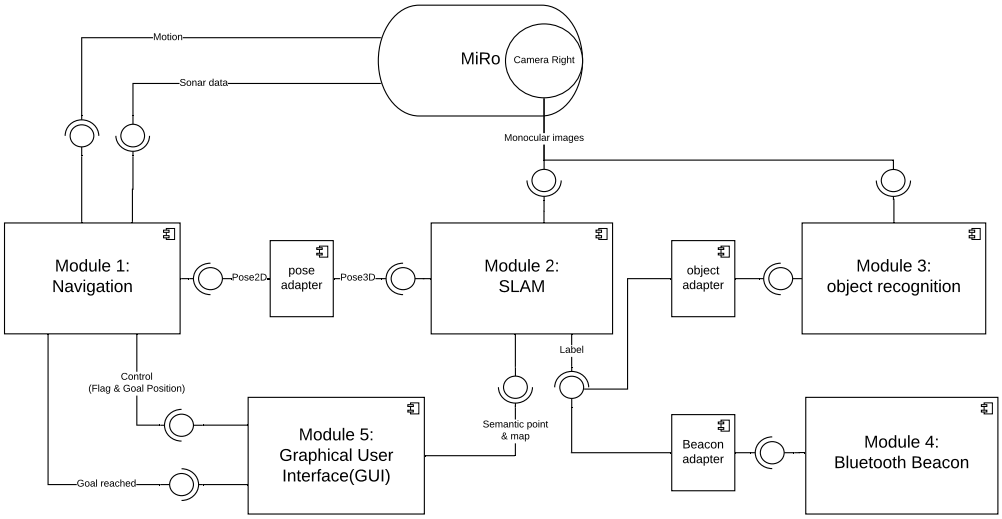

Figure 2: Overall Architecture.

Module 2 is described more in detail in the next paragraph.

2.2 Description of Module 2: Monocular SLAM based depth perception

The main task of this module is to merge topological data from the monocular view, from the SLAM algorithm and from the other modules, extrapolating semantic information. More specifically the semantic data are generated by modules 3 and 4: the messages they publish are strings which label the objects or the bluetooth beacons recognized (i.e bedroom/kitchen' s beacon or chair, table, fridge, etc.). The fact that the *labeling* procedure only needs a string message makes this components reliable and opened to every *labelling*

modules; in this way it is possible to build a semantic map with a significant amount of various information. In this use case we enabled the component to accept data from from the object recognition module (module 3) and from another one executing the bluetooth beacons detection (module 4). Data coming from both modules are made coherent by a proper adapter node: only the string containing the label is extracted from the message received and then given to our module.
Once a message from these modules is sent to our module, the actual 2D position of the robot in the environment is extracted, more specifically in the callback of the subscriber named *adapted_message*: the x and y coordinates of the mobile robot extracted are the ones related to in the instant in which a specific object has been recognized by the corresponding module.
After that, the label information and the 2D position information are merged and stored, as a new customized message, in a rosbag created for this purpose.
Moreover the ORB_SLAM module makes possible, once the node is launched, to choose between the *creation* and the $localization - only$ modality; the former allows the user to create a new map while the latter can be used to load a map previously created by using the *creation* modality or another suitable means.

2.2.1 Architecture

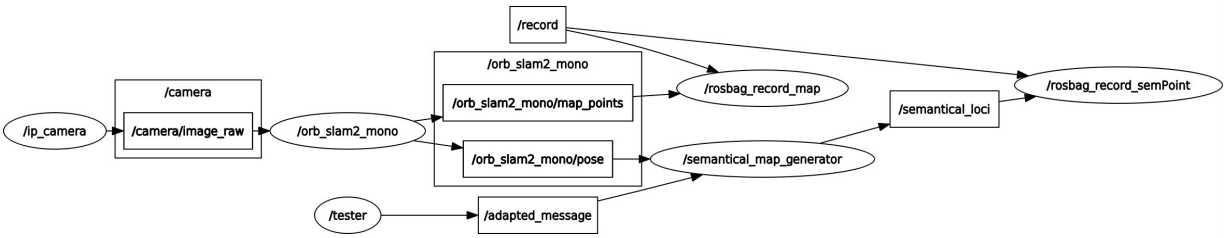

Figure 3: rqt_graph describing our architecture

The architecture of our module is shown in Figure 3. From this Figure it can be noted that the camera is connected to the */camera/image_raw* topic while if the MiRo platform is used it should connect to many other different topics (each topic corresponds to a "square" box inside which its name is written while each "circular" box represents a node) as shown in the following figure.

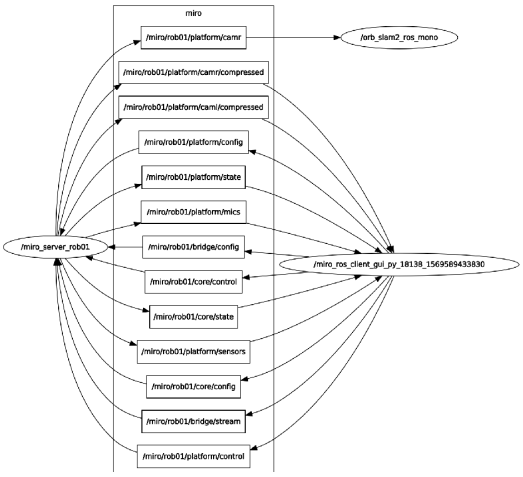

Figure 4: Topics if MiRo platform is used

The *MiRo* macro topic was implemented by the MiRo producers and it can be found inside the SDK folder. The other part of the structure represents what has been implemented. The so called */orb_slam2_ros_mono* node, implemented by us, is subscribed to the camera topic in which one monocular camera is publishing images. Please note that in principle any kind of camera is suitable for our implementation: we succeed to run the SLAM task collecting images also from the smartphone, using common ip Android camera application.
In order to perform the SLAM task itself we decided to use a state of the art monocular slam algorithm, ORB SLAM 2 [2], [3]. This algorithm takes as input the images collected from the camera and outputs both the map of the environment, as a *PointCloud2* datum, and the pose (*Pose3D*) of the camera with respect to the world reference frame. After that a node called */pose_adapter* reads the pose published and extracts the x and y coordinates: this is done in order to send to Module 1 a *Pose2D* message with the actual position of the robot since is necessary for being able to communicate with the architecture implemented in the other module.
Once one of the nodes acquiring semantic data sends a message, this message is adapted to a string (as aforementioned) and published to the */adapted_message* topic. As soon as the */semantical_map_generator* node reads a message from that topic, the custom messages are constructed and then published in the topic named */semantical_loci*. This callback takes the actual pose of the robot and the label of the semantic entity recognized and assembles the *SemanticalPoint* ros message. This ros message has been designed for this purpose and it contains a *Pose2D* and a *String*. The map and the seman-

tic point, published on the */semantical_loci* topic, are then recorded in a rosbag. These can be then used to operate the system in *localization-only* mode, loading a previously constructed map and also the related semantic map.

3 Implementation

3.1 Prerequisites

3.1.1 ROS

MiRo exposes its interface to the network as a ROS node, so ROS must be installed on the workstation in order to communicate with the robot; more specifically the following specifications are needed:

- rosdistro: kinetic
- rosversion: 1.12.13

If the user has installed the base version of ROS, in order to install the extra items needed, he can run the command *sudo apt install ros-kinetic-tf2-ros*.

3.1.2 Gazebo

The user can also decide to use the robot simulator Gazebo if he/she plans to work also with a simulated MiRo robot. For doing that the user can run the command *sudo apt-get install ros-kinetic-gazebo-dev*.

3.1.3 MiRo Workstation

In order to be able to operate MiRo first of all the user needs to install the MiRo developer kit (MDK) provided by Consequential Robotics©. In the "Documentation" section of the official website [1] an introduction to MiRo robot with specification, requirements, and legal information can be also found.
The so called MDK can be found and downloaded in the "Developer" section and firstly it requires that some additional packages are installed in the user's PC. For this purpose the following command can be run in the terminal by the user:
sudo apt install python-matplotlib python-tk python-gi python-gi-cairo
Then the MDK folder needs to be unzipped in a directory of the PC that will be used as workspace for the project and then from the terminal, once the

corresponding directory has been accessed, the following commands must be run:

Listing 1: bash version

```
#Use deb32 instead of deb64 if the machine
#works at 32bit
$ cd path/to/mdk/mdk-190211/bin/deb64
$ ./install_mdk.sh
```

After typing these commands the user needs to do the configuration of the IP address of the local network (i.e. the local machine used by the user) that can be set in:

- static mode: if the network address is fixed, the user can set the variable MIRO_NETWORK_MODE=static and the MIRO_STATIC_IP with its own ID;
- dynamic mode: in this way the network address is determined dynamically through the function *MiRo_get_dynamic_address()* once *setup.bash* is run.

3.2 How to download and run the project

This section describes step by step how to download and run the project on a computer.

3.2.1 Install the ORB_SLAM Module

The user needs to access the workspace in which the MDK kit developer has been installed. Then he/she has to clone the Module2 github repository by running in the terminal the following command:
git clone https://github.com/tommi95/catkin_workspace_SOFAR_semantic_slam/tree/module2.
If the elements of the directory cloned do not correspond to the ones inside the github repository of the Module2 the user can run the command *git checkout track origin/master2* in order to obtain them. After that, the user has to access on the terminal the folder cloned and to run the following commands in order to build, compile and link correctly all the executables inside the repository:

Listing 2: bash version

```
$ catkin_make
$ source /opt/ros/kinetic/setup.bash
$ source devel/setup.bash
```

Moreover the user has to be sure that all the files have the permission for the execution, this can be done by running the command *chmode +x file_name.py* and replacing *file_name* with the name of the executable.

After that the user needs to download the official app of MiRo to be able to connect the robot with the local machine.
The download of the app can be done at the following URL:
http://labs.consequentialrobotics.com/MiRo-b/software/.

3.2.2 How to run the ORB_SLAM Module

To run the ORB_SLAM Module the user has to open a new terminal and run the command *roscore* in order to permit the communication among ROS nodes.
Then he has to open another terminal in order to connect the robot with the local machine.
For doing that, the user has to:

1. switch on the bluetooth of the device in which he has downloaded and installed the app;

2. switch on the robot;

3. open the app and to connect the smartphone to MiRo by selecting it among the bluetooth devices detected.

Once connected, the user can set between many behaviour modalities through the GUI of the app (e.g. normal, demo or others) and other tools (e.g. immobile, silent, ...). Among many details provided by the app, MiRo' s IP can be easily found by the user there. Then he/she can run in the new terminal the command *ssh root@MiRo_IP*, replacing MiRo_IP with the IP read in the GUI of the app, and finally enter the password provided with the robot. Finally he/she has to run the command *sudo nano .profile* and to replace the variable called ROS_MASTER_IP with the IP of his local machine.

The next step consists of sourcing a new terminal with the following command:

Listing 3: bash version

```
$ source PATH/TO/WORKSPACE/
/catkinworkspaceSOFARsemanticslam/mdk/setup.bash
```

replacing PATH/TO/WORKSPACE with the path of the workspace' s directory. Then the user can access the correct directory and run the GUI provided for the robot running the function *MiRo_ros_client_gui.py* through the following commands:

Listing 4: bash version

```
$ cd PATH/TO/WORKSPACE/
/catkin_workspace_SOFAR_semantic_slam/mdk/bin/
/shared
$ ./MiRo_ros_client_gui.py robot=rob01
```

Once this is done, the GUI is running as a node called */MiRo_ros_client_gui_py_* followed by some numbers, as shown in Figure 4; this node is subscribed to many topics on which MiRo is publishing and that provide the information needed by the GUI. Note that if the robot is simulated, for example using *Gazebo*, some further arrangements like topic remapping are needed.

The last step consists of accessing the directory cloned in another new terminal and run the command *./launch.sh* .
After that the node called */orb_slam2_ros_mono* is:

- subscribed to the topic called */MiRo/rob01/platform/camr* on which the robot (so the node called */MiRo_server_rob01*) is publishing images coming from the rightmost robot's videocamera;
- publishing the pose 3D with respect to the map frame to a topic called */orb_slam2_ros_mono/pose*

.

Moreover the nodes */pose_adapter* and */semantical_map_generator* are launched and subscribed to the topic */orb_slam2_ros_mono/pose* in which the node */orb_slam2_ros_mono* is publishing the *Pose3D*.
The first node receives as input a Pose3D and modifies this data in order to output a *Pose2D* into a topic called */adapted_pose2D*. *Module1*, which deals with the navigation task, will read from the latter, in order to know the position of the robot. The second node receives the same input from the same topic but it also reads the labels published by other modules in the topic called */adapted_message*. Then a message composed by the current position of the robot and the label read each time, is published on a topic called */semantical_loci* .

Moreover the user can visualize the *PointCloud* detected and published by the node */orb_slam2_ros_mono* in the topic called */orb_slam2_ros_mono/tf* using RViz, so opening a new terminal and simply inserting the command *rviz*. The GUI provided by ROS for RViz will be visualized and then the user can select the data that he/she wants to display (in this case the PointCloud data and the 3D camera pose) in the interface as it can be seen from the following figure.

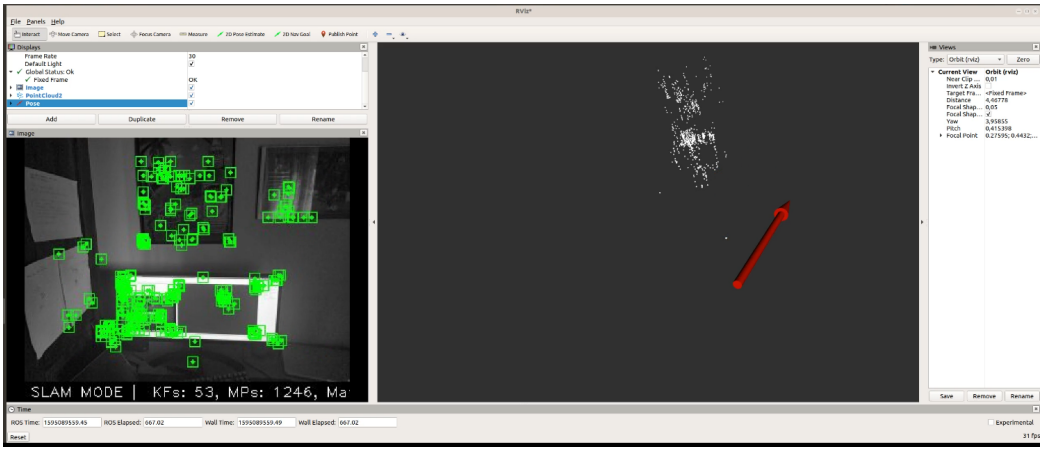

Figure 5: RViz interface visualizing the PointCloud messages

In the leftmost picture the images acquired from the camera are shown with some green squares representing the points detected by the algorithm.

3.3 Calibration

Before the generation of the previous results MiRo or the camera node, is activated and the calibration of the camera must be done. For this purpose first of all a grid is necessary, then the user can run the file named *calibrate.sh*. Once it is launched, a GUI will appear on the screen and the user has to show the grid to the camera in different positions and orientations. The node records some measurements during this operation; as soon as it stores enough points, the calibration can be run: the code will compute the intrinsic and extrinsic parameters related to the camera used. The retrieved values have to be inserted in the configuration file, whose format is *.yaml*, in the following path:
PATH/TO/WORKSPACE/src/orb_slam_2_ros/orb_slam2/config/
In the following we propose a picture collected during this operation.

Figure 6: Calibration of the camera

4 Results

We recorded a video by using the camera of a smartphone while the ORB-SLAM 2 node was running. The video is available at the following link. In this video the red arrow represents the 3D pose of the camera in terms of position and orientation in the world frame. In the left window the camera stream is shown: the features extracted by the ORB SLAM algorithm are represented and denoted with green boxes. In the right one the pointcolud data obtained from the ORB-SLAM calculations are represented along with the world pose. From a qualitative point of view the results are appreciable but they can be improved. The initialization of the algorithm takes some time but, once initialized, it detects many points in the image and stores them on a map. The algorithm is quite stable but many improvements which lead to more robustness could be done.

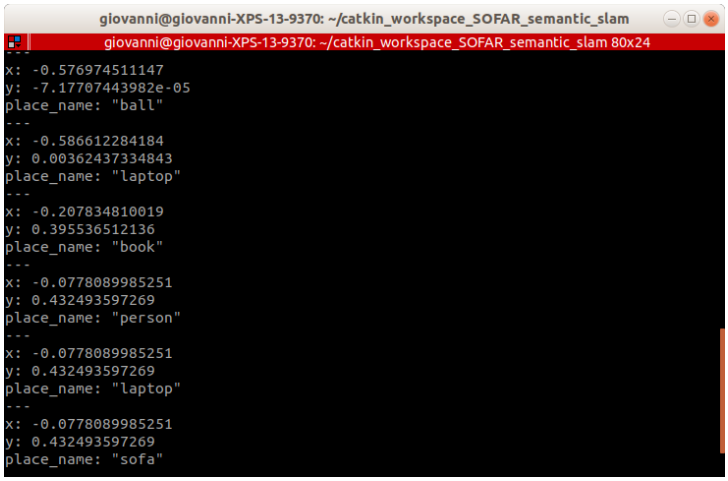

Figure 7: Costumized message created

From Figure 7 it can be noted that once a label is received from other modules, simulated through instances of some publishers in this example, the related callback is executed, makes the computations and then stores the customized message with the *2DPose*, or x and y coordinates, of the camera associated with the label received by the other modules.

5 Recommendations

The major issue we had to deal with consisted in some specifications of the monocular camera of MiRo that were not efficient enough to obtain an appreciable map of the external environment in particular from the point of view of the Frame Per Second (fps) specification.
All the specifications of MiRo can be found at the following link: http://labs.consequentialrobotics.com/MiRo-e/docs/index.php?page=Technical. From this consideration we have planned to use another device, like a smartphone, in order to build a more detailed map of the external environment that could be then loaded and used by MiRo, since also this option is available. Moreover the Github repository that can be found at the following link: https://github.com/EmaroLab/catkin_workspace_SOFAR_semantic_slam/tree/module2 has to be cloned inside the workspace and finally the source file has to be modified changing some parameters inside itself.

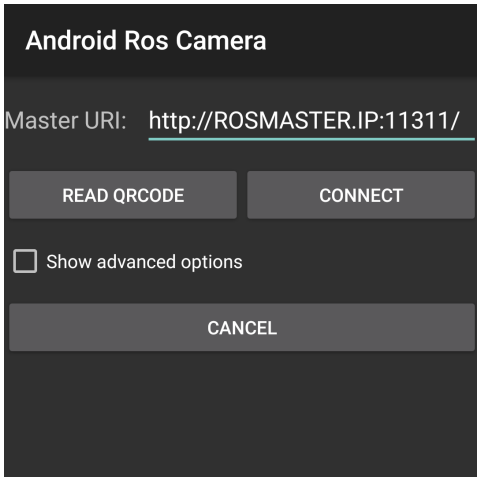

Figure 8: Android ROS camera application

Considering Figure 8 the variable called $ROSMASTER.IP$ has to be substituted with the IP address of the local machine on which the ROS master is running, then click on *CONNECT*. The IP can be found typing on the shell the command $ifconfig$.
This application allows the user to use the camera of the smartphone as a node running on ROS and consequently more performing systems can be used to construct an orb slam map.

The *calibration* is another fundamental step; it is not described in detail in the report but it has to be considered in most of the cases in which a videocamera is used.
In ROS for this purpose the *camera_calibration* node can be used and run; its tutorial is available in the following link:
http://wiki.ros.org/camera_calibration.
In order to do the calibration also a grid is needed: in the command that starts node running, the user has to set the dimension of the grid used (number of rows-1 x number of columns-1) and the dimension of the side of each square inside it.
If the above parameters are set correctly the node starts the calibration of the videocamera's node and once it is finished the GUI leads to store the values computed in a *.yaml* file.
This file has to be placed in the following path:
$PATH/TO/WORKSPACE/src/orb\_slam\_2\_ros/orb\_slam2/config$.
Then, once the orb slam node is launched and has automatically loaded this file, if the calibration has been performed in a correct way, it can start to build the semantic map.

6 Documentation

The node orb_slam2_ros_mono is based on the previous project ORB-SLAM by Raul Mur-Artal [2], while the android camera node is based on the rosjava project.

7 Team

- Andrea Arlotta: rltt.ndr@gmail.com
- Giovanni Battista Borré: giovaborr@gmail.com
- Tommaso Gruppi: tommygruppi@gmail.com

References

[1] http://consequentialrobotics.com/

[2] Mur-Artal, Raúl, Montiel, J. M. M. and Tardós, Juan D., ORB-SLAM: a Versatile and Accurate Monocular SLAM System, IEEE Transactions on Robotics, 31-5, 2015

[3] https://github.com/raulmur/ORB_SLAM2