Software Architecture Project

Student(s): GALLO Thomas, BOUBAKRI Kenza, DEPALLE Theo, DOEHLE Lennart,NDJOMO Paul, VIJAYKUMAR INGLE Vivek
Supervisor(s): THOMAS Anthony, KAREEM Yusha, CARFI Alessandro
Year: 2019 - 2020

Hands on with SLAM (group 10)

Table of Contents

Contents

Abstract

The objective is to setup a scenario with interesting objects in a small room and achieve monocular SLAM using a phone camera. While building the map, it has to recognize (i.e., to label) these interesting objects using computer-vision object-recognition techniques. Then the recognized labels will be attached to their respective bounding boxes within the point cloud, so a map with readable description of localized objects is created.
Yet, it is possible to achieve image recognition and monocular slam using a phone camera that is stream thought WIFI.

1 Introduction

The objective of the project is to achieve a monocular SLAM using a phone camera and augment it by labelling the objects of the room using object recognition. To achieve the requirements of the project, 3 different main features have been used. A ROS package to get the phone camera through WIFI, a monocular SLAM package and a TensorFlow package. These 3 packages will be used to achieve the augmented monocular SLAM.

2 Organization of the project

This project has been carried out by six students from the "Universita degli studi di Genova" as part of the SOFtware ARchitecture (SOFAR) for Robotics course. The project has been divided in modules and distributed among the students. To organize the work, a timetable has been defined initially and updated at the date of the 3rd of February 2020. On this timetable the work has been divided in modules and those modules has been shared among the students to organize the work efficiently.

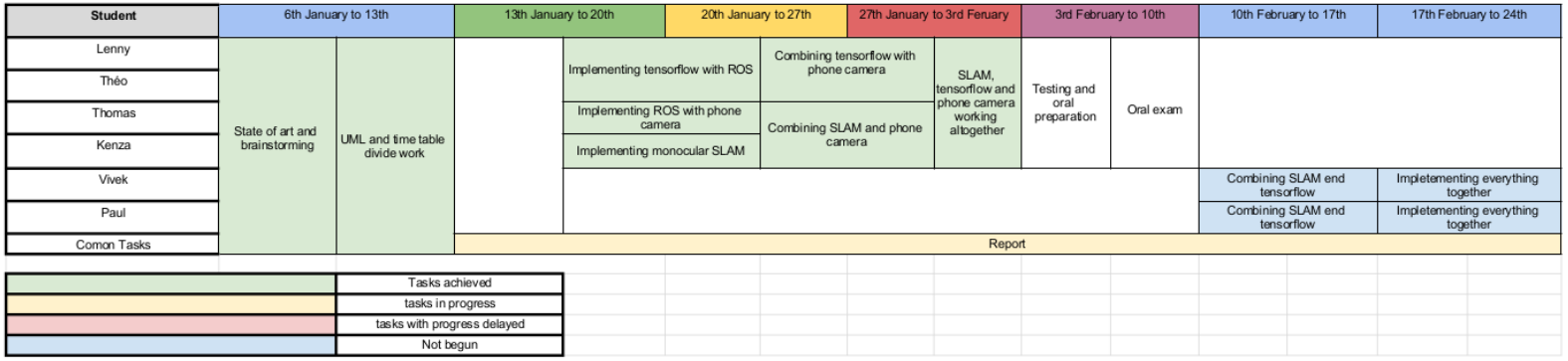

Figure 1: Timetable of the project

3 Architecture of the System

The system takes in entry the phone's camera stream, which is sent to the ROS master on the computer using the IPcamera Android PlayStore application. This is processed using a package that uses a streamed image and publishes it inside a ROS topic called **/camera/image_raw**.
Once the image has been extracted, it is used for the monocular SLAM and the tensorflow package, which subscribe both to the camera topic.
The tensorflow's bounding boxes surrounding the objects will be used to label the points cloud with the right label. Finally the label will be display on the ORB_SLAM2 viewer by linking the center point of the object with an existing point on the point cloud.

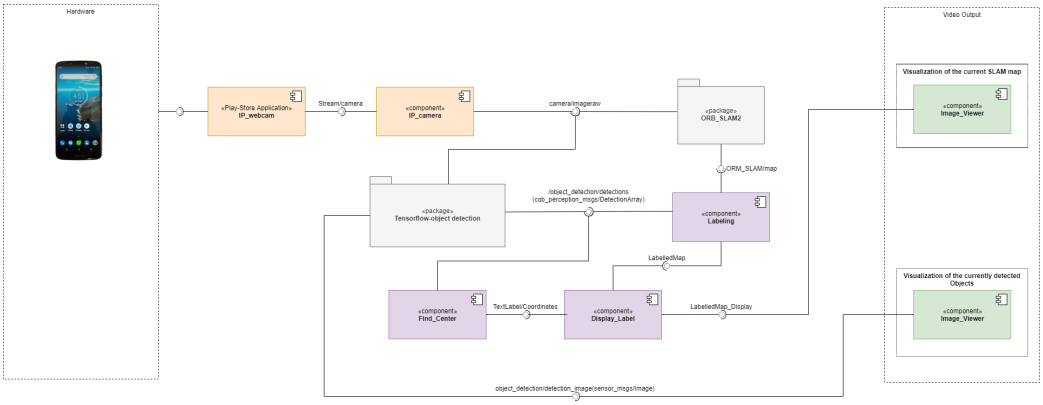

Figure 2: Architecture of the system

4 Description of the System's Architecture

4.1 Module $< IPcamera >$

This module use a ROS node to read frames from the phone camera stream through WiFi and publish it into a ROS topic called **/camera/image_raw**. It has been tested with the IP Webcam application of the google Play Store. The package take in input the stream camera frame from the phone using openCV and convert this image into ROS image message using Cv_Bridge to convert an openCV image to a ROS image message.

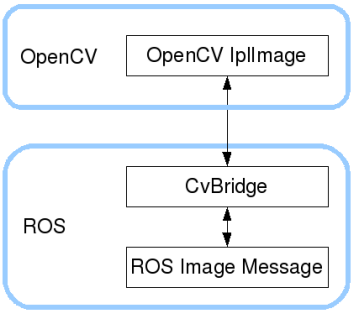

Figure 3: Conversion between OpenCV and ROS image message

Once the image is converted into ROS, it is publish inside a ROS topic. This ROS topic will be used for by the monocular slam module and the tensorflow module as explain in the following sections. This module[1] can been found on the following git repository ip_camera [1]. Its final version has been released the 21/12/17.

4.2 Module <ORB SLAM2>

This module is the second version of a SLAM library made by Raul Mur-Artal, Juan D. Tardos, J. M. M. Montiel and Dorian Galvez-Lopez (DBoW2). Its final version has been released the 13/01/17.[3]
Thanks to this library, it is possible to use real-time Monocular as well as Stereo SLAM and to test it with already-made examples directly in C++ or by using ROS. The organisation of the package is the following:

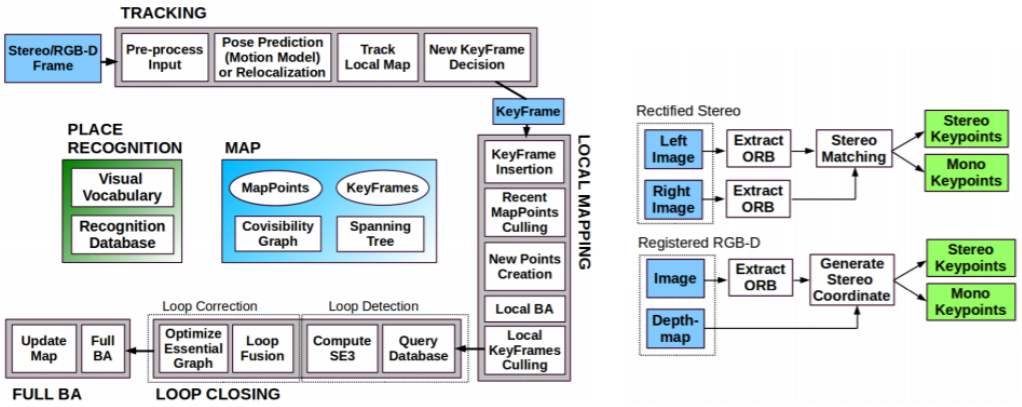

Figure 4: Architecture SLAM_ORB2 package

[1]link to the ip camera git repository

For more information about the architecture of the SLAM_ORB2, see the article about its functioning. In the case of this project, the input is not Stereo nor RGB-D Frame but Monocular instead. The functioning of the package is then the same but because information about the depth are missing (only one image source instead of two), the scale of the map and estimated trajectory is unknown.
In this report, the ROS version will be used. It requires no particular hardware prerequisites as the input can be any kind of raw image[5] but in this project, the purpose is to make it work with a smartphone and therefore the image will be acquired from the smartphone by subscribing to the **/camera/image_raw** topic. When it comes to softwares, C++11, Pangolin, OpenCv, Eigen3 and ROS should already be installed as explained in the repository's ReadMe.
On the principle, this library is close to a classic SLAM algorithm : images are harvested and interpreted in real time. With information about the depth of the different part of this image, it is possible to place points on a 3D maps corresponding of the points of interest of the image and their distance to the camera frame. For the Monocular version of the SLAM, the depth is evaluated which induce more error in the estimation.
From the position's differential of the points of interest, it is possible to deduct the movement of the camera compared to its last position. Once enough points are gathered, it is possible to have an overview of the environment (dimension, obstacle, structure,...) and even to close a "loop" if it is detected that the camera has came back to its initial position.
Finally, it is possible to get as an output this 3D map constituted from all the points of interest encountered in the past images.

4.3 Module <Tensorflow-Object-Detection>

This module is the object detection module out of the huge open source machine learning platform called Tensorflow[2]. Tensorflow was developed by Google and aspires to give everyone an easier access to Machine learning tasks. Its front end is programmed in python to be more convenient. The real computations inside are done in C++ but in this project we are just going to use them as they are. The Object detection part is not a part of the core Tensorflow, but still in the research part of the repository and still under constant development. It is based on a Convolutional Neural Network that extract features from the images given in the database.[2]

[2]link to the original page of this image

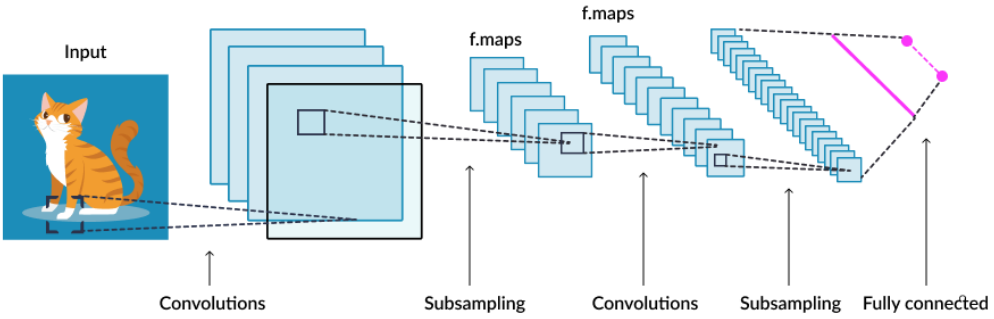

Figure 5: Convolutional Neural Network used in Tensorflow

The module has the image from the phone camera as an input, hence it is subscribing to the **camera/image_raw** topic. In those pictures the object detection module detects interesting objects. This works with a pre-trained neural network, therefore it can only detect those objects that is was previously trained for. On the internet many pre-trained object detection models made for Tensorflow-Object Detection can be found. We will further on use the Coco dataset with a ssd mobilenet model, which is able to detect a wide range of different objects from umbrellas over sandwiches to persons. To avoid false positive detections a smaller model could be used, when expected objects are known.

4.4 Module <Image_Viewer>

The GUI is represented on the UML diagram by two components with the same name, Image_Viewer. Indeed, to make sure that the system is correctly functioning, there will be two displays on the screen of the users.
The first display will correspond to the visualisation of the currently detected objects by tensorflow. It will ensure that the object detection is done properly. It corresponds to the GUI of the Tensorflow package and thus subscribe to the topic **/object_detection/detection_Image**.
The second will allow a more general view and correspond to the current points cloud with the embedded labels. To do so, this display will subscribe to the **/LabelledMap_Display** topic.

4.5 Module <Labelling>

At an instant t, an image coming from the camera is available as it is published in the topic **/camera/image_raw** by the component IP_Camera. By subscribing to this topic, the Tensorflow (Image recognition) and the ORB_SLAM2 packages will publish respectively an array with of detected object in the topic **/object_detection/detections** and the map generated by the SLAM in the topic **/ORB_SLAM/Map**.
These are the topics to which the Labelling component will subscribe. From this data, it can compute which point of the cloud correspond to which data and assign labels to the points in the map.
This modified map will then be published in a topic called **/LabelledMap**.

4.6 Module <Find_Center>

On the point cloud, several points can be linked to the same object as it represents the boundaries of the physical environment seen in the picture. Therefore, printing a label for each point might reduce the readability of the map and also inducing an error in its interpretation.
To address this issue, only one point should we labelled with the name of one object and ideally this point should be it center.
For this reason, the Find_Center component subscribe to the topic **/object_detection/detections** and get the coordinates of the center of each object detected. It then publish it in a topic called **/TextLabel/Coordinates**. With this information the Display Label node, can now find the closest Mappoint to this center coordinate and put the text label there.

4.7 Module <Display_Label>

Once the labelling node has identified all map-points, which belong to the detected object and the Find_Center node has published both the text-label and the ideal picture coordinate position, it is now the Display_Label node's task to select the map-point in the point cloud closest to the center coordinate and place a text box into the picture linked to this point.
To achieve this, the node has to subscribe to both topics, the map given in **/LabelledMap** and the center coordinate and object name given in **/TextLabel/Coordinates**.
The node is then going to publish the map with the text into the topic called **/LabelledMap_Display** which is then printed in the GUI by Image_Viewer. How to add text to the map, in a way that keeps it visualizable by the image viewer is a task not yet known how to achieve.

5 Installation

Here are the steps required to install the three packages used in this project. At the date of 6 of February, only the module ORB_SLAM2, Ip camera and Tensorflow has been tested simultaneously. The three other modules described in the previous section will be implemented later.

5.1 Module $< IPcamera >$

To run and test the ipcamera module:

1. Clone the repository in your catkin workspace [3]
2. Download a IP camera application on your phone [4]
3. Stream your camera using the same WiFi connection for your phone and your computer where the ROS master is running (for a more fluid stream, reduce the quality of the video in the setting of the application)
4. In the ip_camera.launch change the IP of the streamed video to your IP webcam
5. Launch the ROS node to have the streamed camera publish inside a ROS topic

For more details about the installation, consult the project github ReadMe [5]. The testing of this module will be done in cooperation with the ORB_SLAM2 module and the tensorflow module as explain in the following section.

5.2 Module <ORB SLAM2>

To run and test the ORB SLAM2 module:

1. Install all the required prerequisites mentioned in the part 2 of the repository.
2. Clone the repository as described in part 3 but do not build yet.
3. Add the lines:

[3] link to the IP camera git repository
[4] link to the IP webcam android application
[5] link to the IP camera git repository

```
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
```

At the beginning of the files Viewer.cc, Tracking.cc, LoopClosing.cc in src and in Examples, every file.cc in Monocular, Stereo and RGB-D (for instance: *Examples/Monocular/mono tum.cc*)

4. Build as mentioned in part 3 of the git repository
5. Modify the file

```
Examples/ROS/ORB8SLAM2/CMakeLists.txt
```

to have :

```
   set(LIBS
${OpenCV_LIBS}
${EIGEN3_LIBS}
${Pangolin_LIBRARIES}
${PROJECT_SOURCE_DIR}/../../../Thirdparty/DBoW2/lib/libDBoW2.so
${PROJECT_SOURCE_DIR}/../../../Thirdparty/g2o/lib/libg2o.so
${PROJECT_SOURCE_DIR}/../../../lib/libORB_SLAM2.so
-lboost_system
)
```

Also in this file, change *find package(Pangolin REQUIRED)* by *find package(Pangolin 0.2 REQUIRED)*

6. Build as explained in the part 7 of the git repository.
7. Source

5.3 Ros-Tensorflow

To use tensorflow together with ROS, not only the normal tensorflow object detection[2] has to be installed, but also the special module ros_people_object_detection_tensorflow [4]. Both can be installed following the following description:

1. Install all the prerequisites:

- will use tensorflow-gpu so you need to be sure you have all the drivers for you graphic card you can fin it in the section "Software requirements" on [this page](https://www.tensorflow.org/install/gpu)
- You need to install CUDA (if you have nvidia card): please follow the tutorial that can be found on https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html
- you need to have python2 on your computer :

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

- you need image_view if you do not have it

```
$ sudo apt-get install ros-kinetic-image-view
```

2. Clone the repository in you catkin workspace

```
$ cd ~/catkin_ws/src
$ git clone --recursive
↪  https://github.com/cagbal/ros_people_object_detection_tensorflow.git
$ git clone
↪  https://github.com/cagbal/cob_perception_common.git
$ cd ros_people_object_detection_tensorflow/src
$ protoc object_detection/protos/*.proto
↪  --python_out=.
$ cd ~/catkin_ws
$ rosdep install --from-path src/ -y -i
$ catkin_make
$ pip install face_recognition
```

3. Create your virtual environnement and install **Tensorflow**

```
$ virtualenv --system-site-packages -p python2.7
↪  ./venv
$ source ./venv/bin/activate
$ (venv) pip install --upgrade pip
$ (venv) pip install --upgrade tensorflow
```

4. try to see if it is working

```
$ (venv) python -c "import tensorflow as
↪  tf;print(tf.reduce_sum(tf.random.normal([1000,
↪  1000])))"
```

If the output of this is some errors, that's mean you probably don't have all the required libraries with your graphic card in order to run tensorflow (most of the time realated to CUDA), please see the error comments below.

5. Change the topic input: go on

   ```
   ~catkin_ws/src/ros_people_object_detection_tensorflow/launch
   ```

   and edit

   ```
   cob_people_object_detection_tensorflow_params.yaml
   ```

   you need to comment the line 14 (depth_namespace: "your_topic".
   For instance if you want to use the camera from ip_camera : camera_namesapce : "camera/image_raw".

6. The repository used depreciated function of tensorflow, so you need to change all this files by the ones in this repository (/!\TO DO /!\)

7. Run everything

   ```
   $ roscore
   $ roslaunch ip_camera ip_camera.launch #if you are
   ↪  using ip_camera
   $ source ~/venv/bin/activate
   $ (venv) roslaunch
   ↪  cob_people_object_detection_tensorflow
   ↪  cob_people_object_detection_tensorflow.launch#
   ```

   if you want to see the result :

   ```
   rosrun image_view image_view image:=/object_detection/detections_image
   ```

- Most common error :
- some error with libnvinfer.so.6 or other libraries : I advise you to use synaptic to install the missing libraries because it knows all the needed dependencies:

```
$ sudo apt-get install synaptic

$ sudo synaptic
```

- Then you can search the library by name and install it.

6 System Testing and Results

6.1 Module <ORB SLAM2> and <IPcamera>

To ensure the functionality of this module, we first try it with the computer's webcam using the package USB_cam. The SLAM works: it is possible to slowly move the computer around a room to detect points of interested and place them in a cloud map when we move along the room. Nevertheless the quality of the computer's webcam was not so good and the mapping was often losing track of his position.
To make sure the mapping was working and to begin the assembly of the project, the next step has been to use a phone's camera as an image source by using the IPCamera's module as described above. It was way easier to more around a room with it as a camera is more portable than a computer and guarantee a better stability. From the SLAM's viewer part, it was possible to get a map like below:

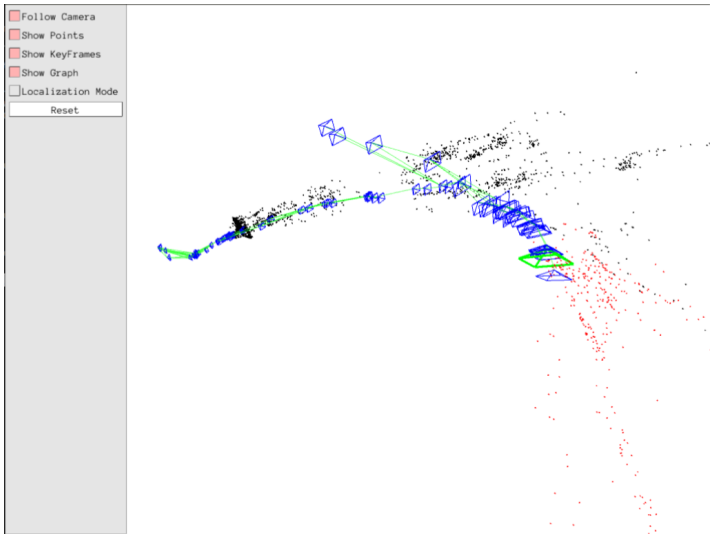

Figure 6: Resulting map of a room exploration

The blue part are the different frames at which images of the room have been processed. It is possible to see the computation made at each instant on the current image:

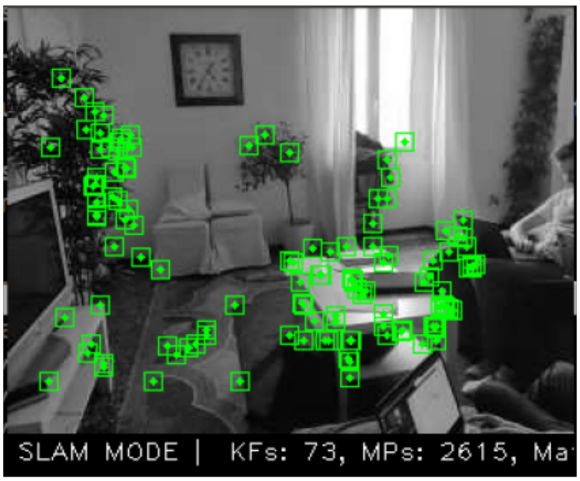

Figure 7: Current image being processed

On the current image, the points of interest are represented by green cross surrounded by a small rectangle. These points are also represented on the global map as points of different color.
The corresponding rqt graph of the working system is the following:

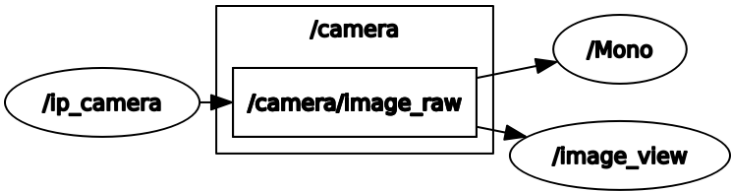

Figure 8: rqt graph for the monocular SLAM

The image harvested from the phone's camera is published in the IP camera topic called **/camera/image_raw**. The Monocular SLAM node subscribes to it as well as an image viewer that allows the user to verify that the image is correctly acquired by the phone.

6.2 Module <Tensorflow-Object-Detection>

After building the corresponding package, it has been possible to test its results in a real-life environment:

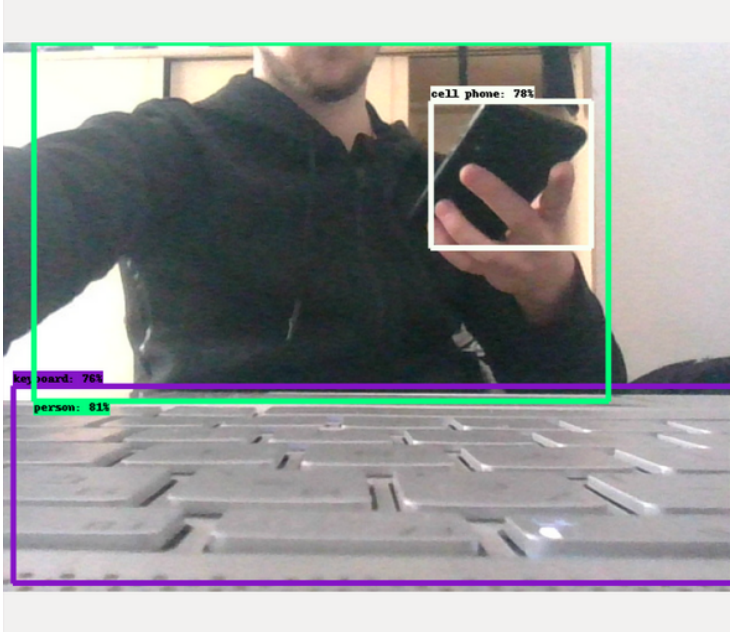

Figure 9: Test of Tensorflow on our machines

On this image, it is possible to see Tensorflow detectiong three superposed objects at the same time. In addition, the cell phone, the person and the keyboard have been detected with a confidence superior at 76% which is satisfying.
Especially, we used the same cell phone to send an image stream to two computers running separately the SLAM part and the Tensorflow part. While checking that the mapping was done correctly, it was possible to verify that most of the objects in the room were also well detected.

7 Recommendations

7.1 Module <IP_Camera>

The phone camera is streamed through the WiFi, therefor depending on the quality of the WiFi connection the stream could have some latency. This will have an impact on the monocular slam and tensorflow.
The first solution is to reduce the quality of the image send to reduce the amount of data send through the WiFi. If the WiFi is still not enough rapid to send the video, it is possible to achieve the project using directly the computer camera. This will delete all latency in the camera. To achieve this

the ROS package usb_cam [6] can be used to extract the video of the computer camera.

7.2 Module <ORB_SLAM2>

As mentioned in the article written by the creators of the ORB_SLAM2 package at the end of the testing phase, one should be careful when using the Monocular SLAM. Besides having no scale for the map, the system is not robust to pure rotations which means that it is possible to experiment a scale drift. During the testing phase of this module, such behaviour have been experimented: while using the phone, a significant variation in the height of the phone's position was considered as a movement along the z-axis which made the loop closing impossible.
To use the ORB_SLAM2 package as a Monocular SLAM, the conditions of execution should be as close as possible of the following ones:

1. The height of the phone's position should be as constant as possible
2. The amount of uncontrasted areas should be as small as possible (make sure that there as much points of interest as possible)
3. The precision in the mapping should not be a priority when using this project Nevert

7.3 Tensorflow-Object-Detection

The object detection software should definitely not be changed by us, as it is the best free version available, and we cannot improve it.
Although as our software is structured right now, tensorflow detects things in every single image frame. This occupies a lot of computation capacity. Since in SLAM only certain key-frame camera positions are saved, it might be useful to activate tensorflow only on these key-frame images because only bounding boxes in those images can be compared reliably with the point cloud.
This activation could be implemented with a ros service added to the SLAM Package, which activates tensorflow, while the system is currently in a key-frame position.

[6] link to the usb_cam git repository

7.4 Labelling

As this node does not exist yet, it can only be assumed which problems could occur with our software architecture in this part. But some issues will definitely be difficult to solve.
For example the map-points in SLAM are saved as 4x4 CV:Mat objects, i.e. a 4x4 matrix. Inside this matrix there is no free space to save labels for the points to define whether they are inside a bounding box of a detected object. So this Matrix would either have to be augmented which will lead to computation problems inside SLAM or create a different data structure for the labels which will spawn difficulties, because the map-points change position as new keyframes are generated and can even be removed later on. So this new data would have to change too in accord.

8 Discussions

As the project evolves, it becomes obvious that executing the image harvesting, the SLAM and the object detection at the same time might be a problem. In this part, the different architecture's problems will be addressed.

8.1 Computation capacities

First, a problem quiet simple has occurred during the installation of the problem. On most computer, even with a Graphical Processing Unit, running simultaneously two programs as demanding as Tensorflow and SLAM can be challenging. The context of the project is indeed to have a portable solution that will allows to map easily a new environment. If the hardware used to run the project is too solicited, this could affect the performance of the computation. A solution that we tried to exploit in order to fix this problem is to use a ROS MASTER on one computer having at least one of the package required for the project and use others computers with the complementary packages as slave to have it run on several machines at the same time.

8.2 Performance of the SLAM and of the object detection

As mentioned in the previous part of this report, the packaged used to do the SLAM and the object detection also have their limits.
By using the Monocular SLAM, one must be aware of its limits as it not as performing as a Stereo SLAM. This can be explained by the lack of depth

information for each image.
Let's have a quick reminder of the functioning of the object detection. A Learner is trained to recognize a defined list of objects by acquiring an understanding of each object's features by looking at the big number of image. Once it is done, this learner is used to recognize with a certain confidence these objects from new image sources. For this reason, the object detection is dependant on the predefined list of objects that should be recognize and on the data base on which the learner has been trained. Consequently, a new kind of object not present on the database will not be recognize by Tensorflow.

8.3 The Real-Time limit

Since all nodes and especially the two packages SLAM and object-detection have a certain computation time, a time difference between the outputs of the different nodes can occur. This would be a big problem for the whole software, since the bounding boxes of tensorflow need to be consistent with the camera position currently used by SLAM to identify the right map-points. This problem could be solved in multiple ways:

1. A time stamp is added to all the operations to check if the nodes work on the same image frame. This would have to be coded very robustly though, so that the program still works, when time-differences happen. It would need to be able to access older information.

2. Real time operation is discarded. A map is generated by Monocular SLAM, the keyframes and the corresponding images are saved. Then later on objects are detected in those images and labelled to the matching map-points in the point cloud.

References

[1] Engr.Ravi. Ip-camera. IIT Bombay. ravich3141@gmail.com.

[2] Sun C Zhu M Korattikara A Fathi A Fischer I Wojna Z Song Y Guadarrama S Murphy K Huang J, Rathod V. Speed/accuracy trade-offs for modern convolutional object detectors. *CVPR*, 2017.

[3] Montiel J. M. M. Mur-Artal, Raúl and Juan D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[4] Cagatay Odabasi. ros_people_object_detection_tensorflow. `https://github.com/cagbal/ros_people_object_detection_tensorflow`, 2017.

[5] Juan D. Tard Ìos Ra Ìul Mur-Artal, J. M. M. Montiel. Orb-slam: a versatile and accurate monocularslam system. IEEE publication.