Università degli studi di Bergamo

DIPARTIMENTO DI INGEGNERIA
Corso di Laurea magistrale in Ingegneria Informatica

OpenFabMap2

Documentazione progetto

Studenti:
Giovanni Bergamelli
Matricola 1024891

Angelo Iapichino
Matricola 1025210

Luca Maurelli
Matricola 1025208

Docente:
Prof. Davide Brugali

Contents

1 Introduction

FABMAP (Fast Appearance-Based MAPping) is a system for appearance-based navigation geared toward autonomous mobile robots. It doesn't provide a precise localization of the robot, instead permitting the recognition of already visited places, as well as recognizing the case of being in a not previously visited location. This is mainly useful in a SLAM computational problem as loop-closure detection. As FABMAP is available only as a binary file, the OpenFABMAP implementation has been developed as an open-source replacement, also providing some enhancements in terms of flexibility of use. The aim of this project is to develop a software architecture for place recognition and robot localization using the OpenFABMAP2 algorithm. As OpenFABMAP is now integrated into OpenCV, which already is a dependency of the OpenFABMAP ROS package.

2 OpenFABMAP

A Fast Appearance-Based MAPping algorithm uses a visual bag-of-words (or bag-of-visual words) to describe the content of images.
A visual bag-of-words is a vector of occurrence counts of a vocabulary of local image features. Such vocabulary was provided by FABMAP, whereas OpenFABMAP can be run in learn mode to create such a vocabulary.
During *learn mode*, a great amount of pictures of the environment is captured; than all images are processed to infer how every relevant aspect of the environment can be described by a finite amount of visual details. These details become the images features that form the vocabulary. After the learn step, the run mode can be undertake.
In *run mode*, pictures of the environment are continuously captured. For every taken picture, a bag-of-words is computed using words from the previously created vocabulary. Then these bag-of-words are compared to give a probability of match between every pair of images, that is the probability that the two compared images depict the same place. Every time a picture is taken, it is compared against all previously taken pictures.
From this description we can easily tell which are the inputs and the outputs of the algorithm:

- camera images form the sole input to the algorithm;
- the output is a vector of probabilities about compared images depicting the same place.

3 Project

During the development of the new software architecture, the hardest challenge faced was in the available implementation of the openFABMAP2 ROS packages, that weren't suitable for the localization purpose: every time the run mode is launched, the robot starts to create an empty codebook, thus not permitting the use of a previous state of the codebook.
The idea to overcome this problem is the re factoring of the available implementation of the run mode.

In the run mode it's possible to identify two distinct on-line phases: the first one creates the codebook by detecting and collecting environment's BOWs, the second one checks if the actual captured image matches with the previously images.
We developed two different mode that coincide with the phases previously described: Place Exploration mode and Place Recognition mode.
This implementation allow the Place Exploration mode to work off-line and this allows the robot to perform the exploration phase on the same place without having to run each time the phase of recognition.

4 ROS

ROS is an open-source, meta-operating system for robots. It provides services expected from an operating system, such as:

- hardware abstraction;
- low-level device control;
- implementation of commonly-used functionality;
- message-passing between processes;
- package management;
- tools and libraries for obtaining, building, writing, and running code across multiple machines.

The main goal of ROS is to ease code re-use in the robotic field of software development. We will now explain some fundamental concepts and components of ROS necessary to understand how it achieves his goal and how we can proceed to reach our aim.

4.1 Concepts

Node A node is a process that perform computation. A robot control system usually comprises many nodes.

Nodelet A nodelet compares to a node as a thread compares to a process. Nodelets allow multiple algorithm to run in the same process, easing the passage of data between them. They can also be dynamically loaded, breaking build time dependencies. Writing a nodelet is minimally different from writing a node.
Package A package might contain ROS nodes, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or anything else that logically constitutes a useful module. The goal of these packages it to provide this useful functionality in an easy-to-consume manner so that software can be easily reused.
Message A message is a simple data structure composed of typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays. Packages can define messages as *.msg files under the msg sub folder within the package folder.
Topic Topics are named buses over which nodes exchange messages. Nodes that are interested in data subscribe to the relevant topic; nodes that generate data publish to the relevant topic. There can be multiple publishers and subscribers to a topic. Topics are the main way in which ROS nodes exchange data; every time a node publish a message in a topic, subscribers will have their callback functions called (callback function is specified when subscribing to a topic). This mechanism allows non blocking many-to-many data exchange. Topics are organized like a Unix file system, with a hierarchical structure and with / as root.
Service Services are a RPC-like way of exchanging data. A providing ROS node offers a service under a string name, and a client calls the service by sending the request message and awaiting the reply (the reply is a message too). Packages can define services as *.srv files under the srv sub folder within the package folder.

For more information about ROS commands, please read the on-line documentation at `http://wiki.ros.org`

5 OpenCV nonfree module

Other than the base OpenCV installation, also the nonfree module is required. This is required by the openfabmap2 package to compile. This module contains algorithms that may be patented in some countries or have some other limitations on the use, so it's not included in the default package sources; therefore, in order to install it, we must add an additional repository to our distribution package sources.

```
# add the new repository
$ sudo add -apt - repository ppa: xqms /opencv - nonfree
# update packages index
$ sudo apt -get update
# install the module
$ sudo apt -get install libopencv - nonfree -dev
```

6 Code Repository

These packages can be downloaded as zip file from `https://github.com/jLuca/ros-openfabmap2`; alternatively, while having git installed, they can be cloned with the command:

```
$ git clone https://github.com/jLuca/ros-openfabmap2
```

Once obtained the packages, follow these steps:

- move the openfabmap2_msgs folder in the catkin workspace under the src folder;
- in a terminal, navigate to the catkin workspace then run

```
$ catkin_make install
$ source devel/setup.bash
```

- move the openfabmap2 folder in the catkin workspace under the src folder;
- in a terminal, navigate to the catkin workspace then re-run

```
$ catkin_make install
$ source devel/setup.bash
```

We have to build two times because of openfabmap2 dependency on openfabmap2_msgs.

6.1 ROS packages

We now describe the ROS packages implementing the previously described behaviour.

6.1.1 The openfabmap2_msgs ROS package

This package only define the Match message, which gives the informations about matches of a just taken image against all the previous ones. This is separated from the openfabmap2 package to let other nodes depend only on this package, that is only the message definition.
An openfabmap2_msgs/Match message is composed by the following fields (the first word is the type, the second one is the name):

- **Header header** typical ROS header for a message, which has itself the following fields:
  - **uint32 seq** sequence number of the message in the topic.
  - **time stamp** composed by sec and nsec (as offset from sec).
  - **string frame_id** "0" if the massage has no associated frame, "1" if the message is associated with a global frame.
- **int32 fromImgSeq** seq of the header of the image for which the Match message is generated
- **int32[] toImgSeq** seq of the header of the images that fromImgSeq is compared against
- **float64[] toImgMatch** toImgMatch[i] is the normalized probability of match between toImgSeq[i] and fromImgSeq

6.1.2 The openfabmap2 ROS package

The openfabmap2 ROS package implements all the features described in section 6.1, with a binary for learn mode and another for run mode.
Below we list some useful parameters that can be changed to tune the behaviour of the algorithm.

6.1.3 Learn mode parameters

- **sampleRate** the rate at which pictures will be captured.
- **maxImages** the amount of pictures to capture; note that taking a lot of pictures results in a long processing time, so we advise not to take more than 200 pictures.

- **visualise** whether to show captured images with overlaying features detected points.
- **vocab** path and name of the file where the generated vocabulary will be saved; all directories in the path must exist.
- **clTree** path and name of the file where the generated Chow-Liu Tree will be saved; all directories in the path must exist.
- **trainbows** path and name of the file where the generated trainbows will be saved; all directories in the path must exist

6.1.4 Place Exploration mode parameters

- **sampleRate** the rate at which pictures will be captured.
- **visualise** whether to show the confusion matrix depicting the computed matching probabilities over time.
- **vocab** path and name of the vocabulary file.
- **clTree** path and name of the Chow-Liu Tree file.
- **trainbows** path and name of the trainbows file.
- **maxBOW** maximum number of BOW to end the Place Exploration mode
- **MinDescriptorCount** minimum number of descriptor that an image should have
- **storagePath** path where ID images are saved
- **imagePath** path where images's descriptors are saved (matrix)

6.1.5 Place Recognition parameters

- **sampleRate** the rate at which pictures will be captured.
- **visualise** whether to show the confusion matrix depicting the computed matching probabilities over time.
- **vocab** path and name of the vocabulary file;
- **clTree** path and name of the Chow-Liu Tree file;
- **trainbows** path and name of the trainbows file;
- **maxBOW** maximum number of BOW to end the Place Exploration mode;

- **MinDescriptorCount** minimum number of descriptor that an image should have;
- **only_new_places_** if true saves only new descriptors;

6.2 Codebooks files

The vocabolary, Chow-Liu Tree and Trainbows files are placed under the openfabmap2 package folder, in the *new* subfolder. There are also two additional files, *storage.yml* and *image.yml*. The first one contains the descriptors of the captured images during the Exploration mode while the second one the informative images IDs. These last files are created by our implementation as an input for the Recognition mode to recognize the already visited places. The IDs are used to compare the images.

6.3 Launch files

We provided eight launch files, three for learn mode, two for place exploration mode, two for place recognition mode and one for the uEye camera:

- **learn.launch** run the learn mode; requires the image parameter to be set to the name of a topic publishing messages of type sensor_msgs/Image (namely a camera published topic);
- **learn_ueye.launch** run the learn mode and the uEye camera node, assuming that the uEye camera is connected and "piping" the camera topic to the running learn mode;
- **learn_usb.launch** same as previous one, but with a usb camera;
- **place_exploration.launch** run the place exploration mode, same assumption of the learn.launch;
- **place_exploration_ueye.launch** run the place exploration mode and the uEye camera node, same assumption of the learn_ueye.launch;
- **place_recognition.launch** run the place recognition mode, same as previous;
- **place_recognition_ueye.launch** run the place recognition mode and the uEye camera node, same as previous;
- **rgb8.launch** run the uEye camera;

7 Cameras

OpenFABMAP does not impose constraints on the camera to be used to take pictures. Pictures only have to be descriptive enough; the threshold is setted by the feature detection algorithm, so e.g. pictures without details or blurred because of robot's motion are more likely to be discarted than pictures taken with a low quality camera. The Robotics Laboratory has two cameras:

- a High-Definition Trust web-cam, USB pluggable;
- an uEye LE industrial camera.

For webcams, ROS provide the usb_cam package; it can be used like this:

- plug the web-cam in a USB port;
- start roscore:

```
$ roscore
```

- start the usb_cam node with the command:

```
$ roslaunch usb_cam usb_cam-test.launch
```

7.1 uEye Camera

uEye cameras are industrial quality cameras used for intrusion detection systems. They can be used via USB or Ethernet, so they require special drivers to work, and can't be used like normal web-cams
ROS provides three packages for this cameras to be used with ROS itself:

- **ueye** this package contains a ROS driver node for IDS Imaging uEye cameras
- **iri_ueye_camera** same as previously, with low-level driver;
- **ueye_cam** this package provides a ROS interface for the uEye digital cameras; this ROS interface exposes many of the features of the underlying Linux camera API from IDS, and is compatible with image_pipeline, camera_calibration, and image_geometry.

After some tests, we choose the **ueye_cam** package. This one was already cloned from the repository `https://github.com/anqixu/ueye_cam`. It can be used like this:

- plug the uEye camera in a USB port;

- make sure that the uEye USB daemon is running (see 4.3);
- start roscore:

```
$ roscore
```

- start the ueye_cam node with the command:

```
$ roslaunch ueye_cam rgb8.launch
```

7.1.1 uEye drivers

uEye drivers can be downloaded from this url:

- 32 bit Linux uEye drivers: `https://www.1stvision.com/cameras/IDS/sw/uEye_Linux_4.61_32_Bit.tgz`
- 64 bit Linux uEye drivers: `https://www.1stvision.com/cameras/IDS/sw/uEye_Linux_4.61_64_Bit.tgz`

Newest drivers can also be downloaded from the following link, note that these weren't tested to work with this project: `https://www.1stvision.com/ids_downloads.htm#linux`

Start Drivers From the first restart of the machine, uEye drivers daemons should already be running. If it is not the case, these are the control commands:

```
# to run the USB daemon
sudo /etc/init.d/ueyeusbdrc start
# to stop the USB daemon
sudo /etc/init.d/ueyeusbdrc stop
# to force stop the USB daemon
sudo /etc/init.d/ueyeusbdrc force-stop

# to run the ethernet daemon
sudo /etc/init.d/ueyeethdrc start
# to stop the ethernet daemon
sudo /etc/init.d/ueyeethdrc stop
# to force stop the ethernet daemon
sudo /etc/init.d/ueyeethdrc force-stop
```

If something goes wrong the downloaded archive contains a read-me file that explains in detail how to install and configure the drivers. Refer to this file if something goes wrong while following the previously explained steps.

7.2 Camera Calibration

Geometric camera calibration, also referred to as camera resectioning, estimates the parameters of a lens and image sensor of an image or video camera.
We can use these parameters to correct for lens distortion, measure the size of an object in world units, or determine the location of the camera in the scene.

To calibrate a camera, two ROS packages must be installed:

```
$ sudo apt-get install \
ros-indigo-camera-calibration \
ros-indigo-camera-calibration-parsers
```

The first gives an easy interface for the OpenCV camera calibration procedure, while the latter allow to translate calibration data from the OpenCV format to the ROS format. Before proceeding, make sure that you have the following:

- a large checker-board with known dimensions. We used a 8x6 checker-board with 24mm squares. Calibration uses the interior vertex points of the checker-board, so an "8x6" board is nine squares wide and seven high;
- a monocular camera (the uEye camera) publishing images over ROS.

The calibration can be started with the following command:

```
$ rosrun camera_calibration cameracalibrator.py \
--size <n>x<m> -- square <l> \
image:=<image_topic> camera :=<camera_topic>
```

where:

- **n** and **m** represent the size (interior vertex points) of the board;
- **l** is the side of the squares, in meters;
- image_topic is the ROS topic over which raw images are published by the camera;
- **camera_topic** is the ROS topic over which camera informations are published by the camera itself.

The following is a concrete example:

```
rosrun camera_calibration cameracalibrator.py \
--size 8x6 -- square 0.024 \
image:=/ueye_cam/image_raw camera:=/ueye_cam
```

After launching the command, a window will pop-up, allowing to monitor the progress of the calibration process; simply move and tilt the camera until the SAVE button gets enabled; then click it and wait for the procedure. After this step, check under the tmp directory for a file called calibrationdata.tar.gz; it should contain several image files, that are the images from which calibration data is computed, and an ost.txt file, that contains the calibration data in OpenCV format. To convert this data to the standard ROS format, do the following:

- extract the ost.txt file to a desiderd location;
- rename it to ost.ini;
- run

```
$ rosrun camera_calibration_parsers \
convert ost.ini camera.yaml
```

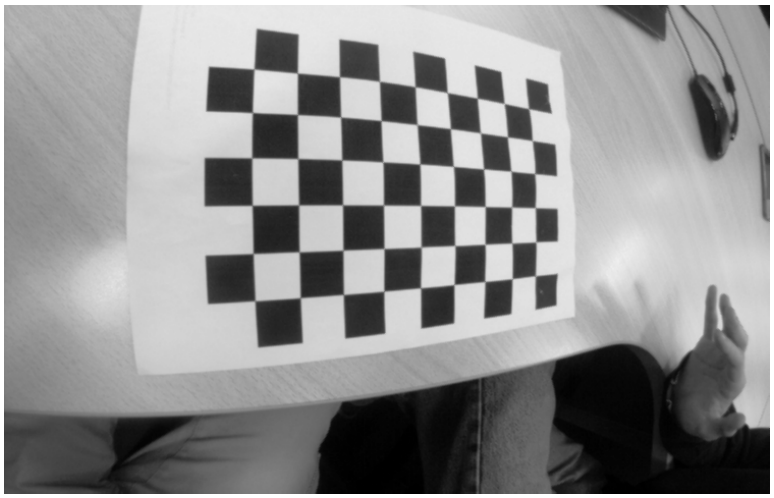

Figure 1: Camera frame during the calibration

Now the camera.yaml file should be moved where the (uEye) camera launch file expects it to be: place it in the openfabmap2 package under the camera_conf sub folder.

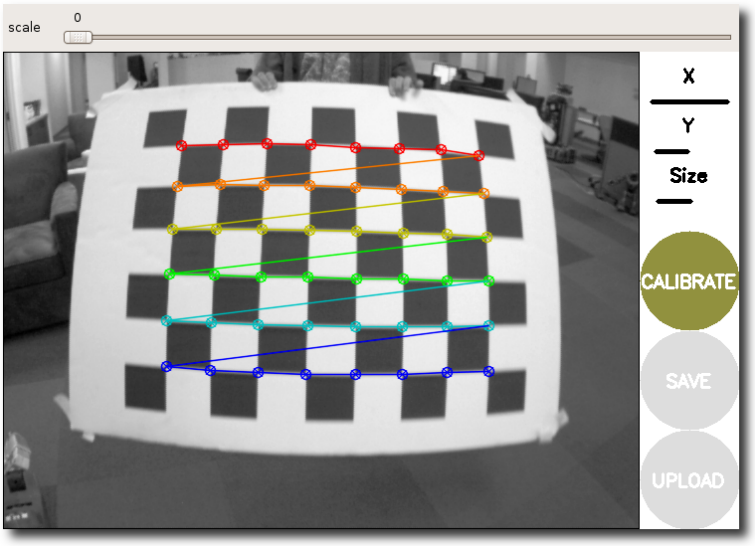

Figure 2: Example of window during camera calibration

7.3 Camera Configuration

There is a file called camera_ueye.ini in the openfabmap2 package under the camera_conf sub folder that contains the camera parameters. As an alternative you can configurated the camera directly from the camera launch file.

8 Usage

8.1 Configuration

8.1.1 Features

Making sure 'good' detection of features is occurring is important to achieving positive results. The result of the FAB-MAP calculation is highly dependent on the statistics collected from training data. If the feature detection is noisy and inconsistent it will be reflecting in the results. It is often poor feature detection that hinders performance.
Good detection should:

- Have similar amounts of features in each frame. Depending on the video resolution and richness of the environment this could be 50 - 500.
- Not detect features on bland/uninformative areas on an image. This is just added noise to the system.
- Consistently detect features in the same location or from the same objects. In continuous videos you should be able to 'track' features along with the movement of the object yourself.

Feature detection can be adjusted:

- Change feature detection type. Many features exist in openCV (SIFT, SURF, STAR, MSER)
- Change feature detection thresholds. The default thresholds for detection may not work for every dataset and environment type.

8.1.2 Vocabulary

The visual vocabulary is another common source of reduced performance. The vocabulary quantises the high dimensional feature descriptor by assigning it part of a single class.

Factors affecting the vocabulary:

- The number of words. More words in the vocabulary makes openFABMAP more specific. The number of words that should be chosen will be dependent on the size and variance in appearance of the physical environment, as well as the number of features extracted from the training data. As a reference point:
  - Cummins & Newman 2008 [**?**] - high definition outdoor dataset 10,000 words

  - Glover et al. 2010 [10] - standard definition outdoor dataset 6,000 words.
  - Cummins & Newman 2010 [7] - 1,000km outdoor dataset 100,000 words.

- The clustering method. Cummins & Newman 2010 discusses the use of the popular k-means algorithm, concluding in a reduced performance for FAB-MAP applications. The modified sequential clustering algorithm provided is an alternative that avoids the over-generalising that affects k-means.

- The more data you can throw at the codebook training stage the better.

8.1.3 Training Dataset

- The training dataset should have no repeated paths. Viewing the same scene multiple times biases the algorithm and will not produce an accurate statistical model of the features observed in the environment.

- Use separate training and test datasets. Desirably from completely separate environments which are somewhat similar in appearance. Lacking a suitable training environment do not use exactly the same images for training and testing as this will decrease performance, especially when using the sample-based new-place method.

- Cummins & Newman 2008 suggests using disjoint (non-overlapping) for training and testing. OpenFABMAP can be trained on continuous data but it is possible it can result in over-training. In addition, varying your speed through the environment when using a continuous dataset can also lead to biased statistics as per the first point.

- Again the more data you can include in the training dataset the better, erring in the direction of continuous datasets over low amounts of data.

8.2 Let's start

Open a new terminal window and run the roscore:

```
$ roscore
```

Everytime we open a new terminal window we have to run the following commands:

```
$ cd path of the workspace
$ source devel/setup.bash
```

The aviable launch files are valid only for the uEye Camera. Starting a ROS-Node, thought a launch file, is done with the following commands:

```
$ cd pathname
$ roslaunch filename.launch
```

8.3 Camera Usage

The commands to run the camera are the following:

```
# to run the camera node
$ roscd app_camera/
$ roslaunch camera_viewer.launch
# to run the viewer node
$ roscd openfabmap2/
$ cd launch/ueye/
$ roslaunch rgb8.launch
```

Note that the camera_viewer.launch file is located il the app_camera package, provided by Professor. The above commands are only an example, the rgb8 node will be automatically runned when using the openfabmap2 nodes.

8.4 Example of usage

8.4.1 Learn Mode

This phase has already been performed in a part of the laboratory. The content is sufficiently informative for places of that type. All files in ros-openfabmap2/localization/openfabmap2/codebooks/default/. This step allows you to create a dictionary with the elements necessary to compare images

```
$ roscd openfabmap2/
$ cd launch/ueye/
$ roslauch learn_ueye.launch
```

Learn mode automatically shows images of the framed scene, with overlaying found key-points, as shown in figure 3 and 4.

8.4.2 Place Exploration Mode

This step allows you to store features characteristics of the area in which you are operating. The results will be stored in their respective file (described above). The default path is ros-openfabmap2/localization/openfabmap2/codebooks/new/ Possible output is shown in figure 5.

Figure 3: Picture captured during Learn mode

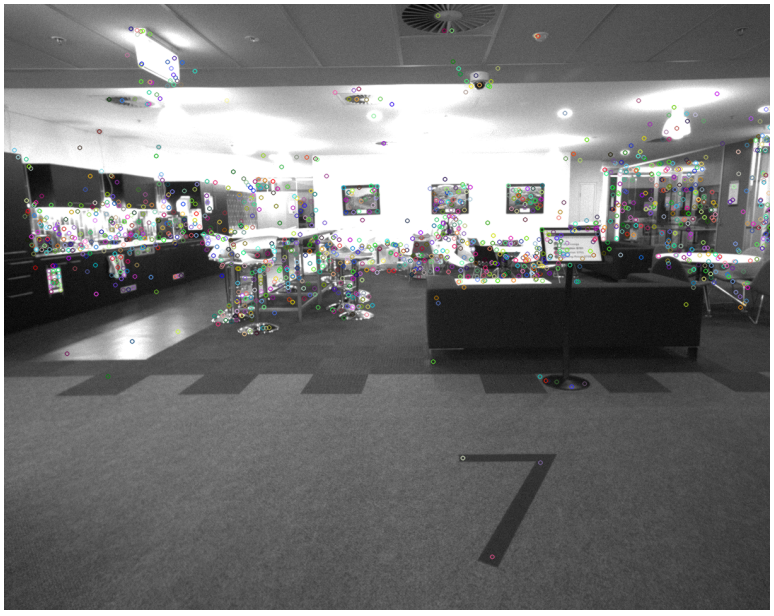

Figure 4: Picture captured during Learn mode

```
$ roscd openfabmap2/
$ cd launch/ueye/
$ roslauch place_exploration_ueye.launch
```

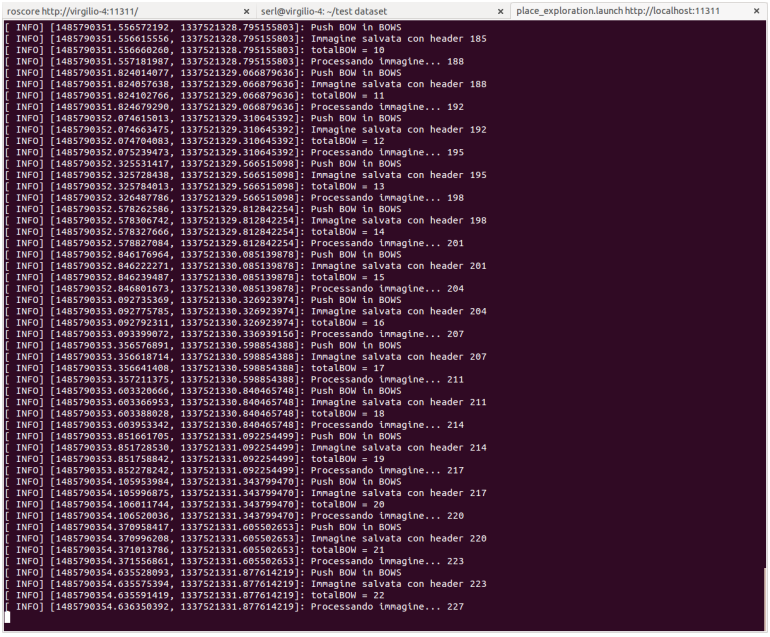

Figure 5: Example of the output during Exploration mode

8.4.3 Place Recognition Mode

Creates confusion matrix that compares the current images with those with features saved in the previous step. Possible output is shown in figure 6.

```
$ roscd openfabmap2/
$ cd launch/ueye/
$ roslauch place_recognition_ueye.launch
```

It was noticed the presence of a straight line less dense than the diagonal one. This is due to the imprecision of the repeatability of the experiment. The slope is different because the timing of the two routes is different.

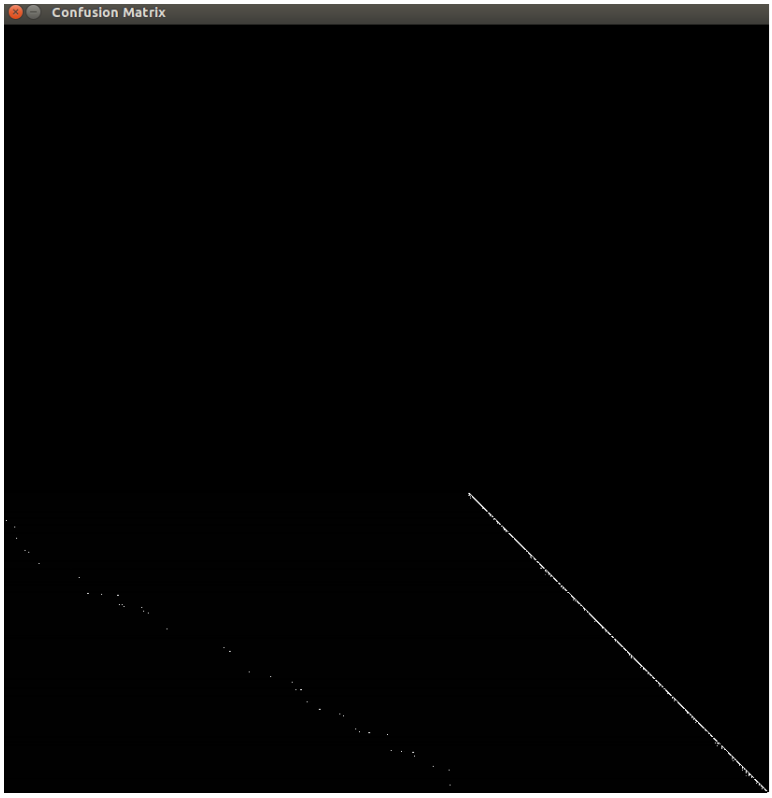

Figure 6: Picture captured during Recognition mode obtained by performing a path around the lab table

8.4.4 Rosbag Dataset

We also used an 3747 Frame Subset provided by the original authors of the OpenFABMAP2 ROS wrapper. The dataset is available for the download at: `https://wiki.qut.edu.au/display/cyphy/Indoor+Level+7+S-Block+Dataset`

To merge the downloaded archives run the following command:

```
$ cat file.tar.gz.part-* | tar xz
```

Rosbag Usage Running the rosbag is done by the following command:

```
$ rosparam set use_sim_time true
$ roscd openfabmap2/
$ cd launch/ueye/
$ roslaunch learn.launch image:=/stereo/left/image_raw
```

In a new terminal window:

```
$ cd test\ dataset/
$ rosbag play level7_truncated.bag --clock
```

You can also run the following command to get more information about the rosbag:

```
$ rosbag info level7_truncated.bag
```

level7_truncated.bag is the name of the rosbag file that we use in the our project.
It's possible to notice that using the rosbag file during all phases (Learn, Exploration and Recognition) the straight lines are almost the same. This is due to the fact that the frames and timing are the same.

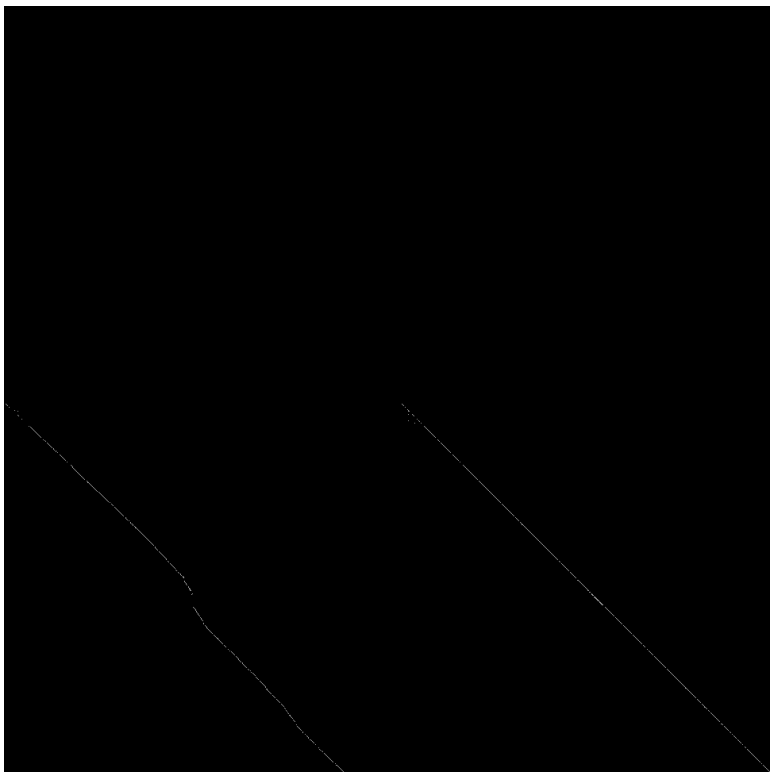

Figure 7: Picture captured during Recognition mode using a bag file

References

[1] GitHub repository for robotics project. https://github.com/jLuca/ros-openfabmap2

[2] GitHub repository of the original openfabmap2 project. https://github.com/anand-ajmera/cyphy-vis-slam/tree/master/openfabmap2

M. Cummins and P. Newman, *FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance.* International Journal of Robotics Research, 27(6):647–665, 2008. 1377517.

[3] A. Glover,W. Maddern, M.Warren, S. Reid, M. Milford, and Gordon-Wyeth, *OpenFABMAP: An open source toolbox for appearance-based loop closure detection.* In Robotics and Automation (ICRA), 2012 IEEE International Conference on, pages 4730–4735, May 2012.

[4] T. Morris and L. Murphy, *openfabmap2* - ROS Wiki. http://wiki.ros.org/ openfabmap2, 2012.

[5] OpenFABMAP OpenCV 2.4.13.2 documentation, http://docs.opencv.org/2.4/modules/contrib/doc/openfabmap.html, 2012.

[6] The Oxford Mobile Robotics Group. Software - FABMAP browse. http://www.robots.ox.ac.uk/~mobile/wikisite/pmwiki/pmwiki.php?n=Software.FABMAP, 2012.

[7] M. Cummins and P. Newman, *Appearance-only SLAM at large scale with FAB-MAP 2.0* - The International Journal of Robotics Research. 2010.

[8] A. Glover, W. Maddern, M. Warren, S. Reid, M. Milford, and G. Wyeth, *Open Source C++ Code for Bag-of-words models and the FABMAP algorithm.* https://code.google.com/p/openfabmap/, 2012.

[9] A. Glover, W. Maddern, M. Warren, S. Reid, M. Milford, and G. Wyeth, *Open Source C++ Code for Bag-of-words models and the FABMAP algorithm.* https://code.google.com/p/openfabmap/, 2012.

[10] Arren J. Glover ; William P. Maddern ; Michael J. Milford ; Gordon F. Wyeth *FAB-MAP + RatSLAM: Appearance-based SLAM for multiple times of day*, 2010.