5. ORB_SLAM2 Basics

official website：http://webdiis.unizar.es/~raulmur/orbslam/

TUM Dataset：http://vision.in.tum.de/data/datasets/rgbd-dataset/download

KITTI Dataset：http://www.cvlibs.net/datasets/kitti/eval_odometry.php

EuRoC Dataset：http://projects.asl.ethz.ch/datasets/doku.php?id=kmavvisualinertialdatasets

orb_slam2_ros：http://wiki.ros.org/orb_slam2_ros

ORB-SLAM：https://github.com/raulmur/ORB_SLAM

ORB-SLAM2：https://github.com/raulmur/ORB_SLAM2

ORB-SLAM3：https://github.com/UZ-SLAMLab/ORB_SLAM3

The operating environment and hardware and software reference configuration are as follows:

- Reference model: ROSMASTER X3
- Robot hardware configuration: Arm series main control, Silan A1 LiDAR, AstraPro Plus depth camera.
- Robot system: Ubuntu (version not required) + docker (version 20.10.21 and above)
- PC virtual machine: Ubuntu (20.04) + ROS2 (Foxy)
- Usage scenario: use on a relatively clean 2D plane

5.1, Introduction

ORB-SLAM is mainly used for monocular SLAM;

ORB-SLAM version 2 supports monocular, binocular and RGBD interfaces;

ORB-SLAM version 3 adds IMU coupling and supports fisheye cameras.

All steps of ORB-SLAM uniformly use ORB features of the image.ORB features are a very fast feature extraction method with rotational invariance and scale invariance can be constructed using pyramids. The use of uniform ORB features helps the SLAM algorithm to have consistency in the steps of feature extraction and tracking, keyframe selection, 3D reconstruction, and closed-loop detection. The system is also robust to strenuous motion and supports wide-baseline closed-loop detection and relocalization, including fully automatic initialization. Since the ORB-SLAM system is a feature point-based SLAM system, it is able to compute the camera trajectory in real time and generate sparse 3D reconstruction results of the scene.

On the basis of ORB-SLAM, ORB-SLAM2 contributes points:

1) The first open source SLAM system for monocular, binocular and RGBD cameras, including loopback and relocalization and map reuse.

2) RGBD results show that more accuracy than ICP or minimization based on photometric and depth errors can be obtained by using BA.

3) Binocular results are more accurate than direct binocular SLAM algorithms by utilizing far and near points in binoculars, as well as monocular observations.

4) The light localization mode allows for efficient reuse of maps.

ORB-SLAM2 contains modules common to all SLAM systems: Tracking, Mapping, Relocalization, and Loop closing. The following figure shows the flow of ORB-SLAM2.

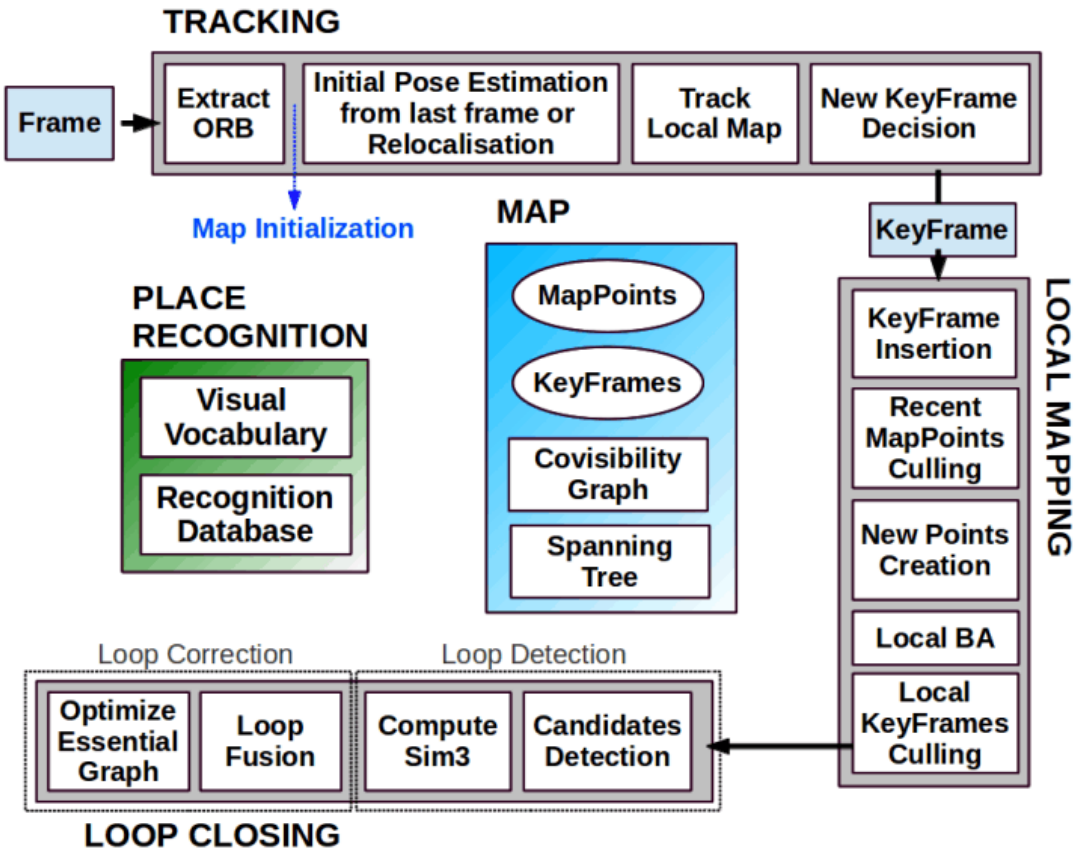


Fig. 1. ORB-SLAM system overview, showing all the steps performed by the tracking, local mapping and loop closing threads. The main components of the place recognition module and the map are also shown.

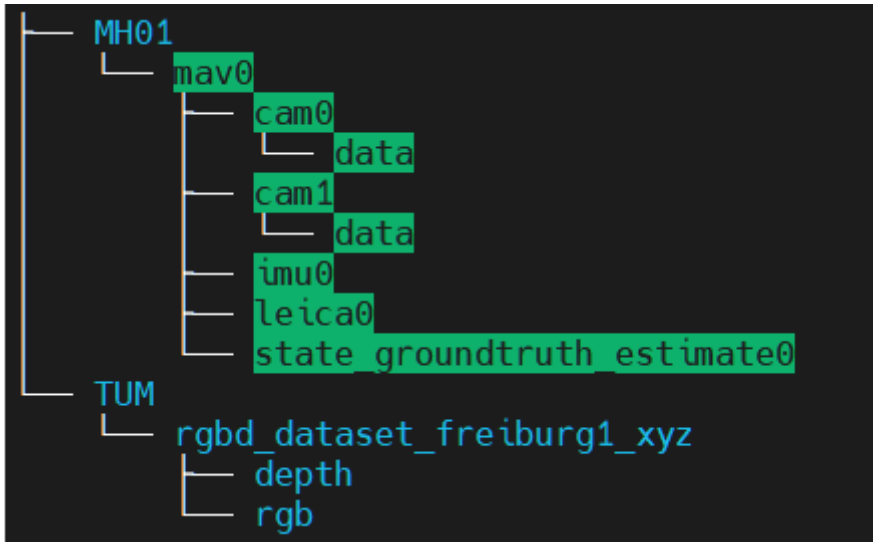
5.2. Official Cases

Note: The commands or locations mentioned below are all in the docker container unless otherwise stated.

5.2.1 Data set location

```
/root/yahboomcar_ros2_ws/software/orbslam2/ORB_SLAM2-master/data
```

Inside, the MH01 dataset from EuRoC and the rgbd_dataset_freiburg1_xyz dataset from TUM were downloaded.



If you need another dataset, you can go to the following address to download it:

```
TUM Dataset：http://vision.in.tum.de/data/datasets/rgbd-dataset/download
KITTI Dataset：http://www.cvlibs.net/datasets/kitti/eval_odometry.php
EuRoC Dataset：http://projects.asl.ethz.ch/datasets/doku.php?
id=kmavvisualinertialdatasets
```

5.2.2. Entering the docker container

To enter the docker container, please refer to [docker course chapter ----- 5, enter the docker container of the robot].

5.2.3 Testing with one eye

Here we take the EuRoC dataset as an example, first enter the docker container, and then enter the ORB_SLAM2 directory:

```
cd /root/yahboomcar_ros2_ws/software/orbslam2/ORB_SLAM2-master
```

Run the following command:

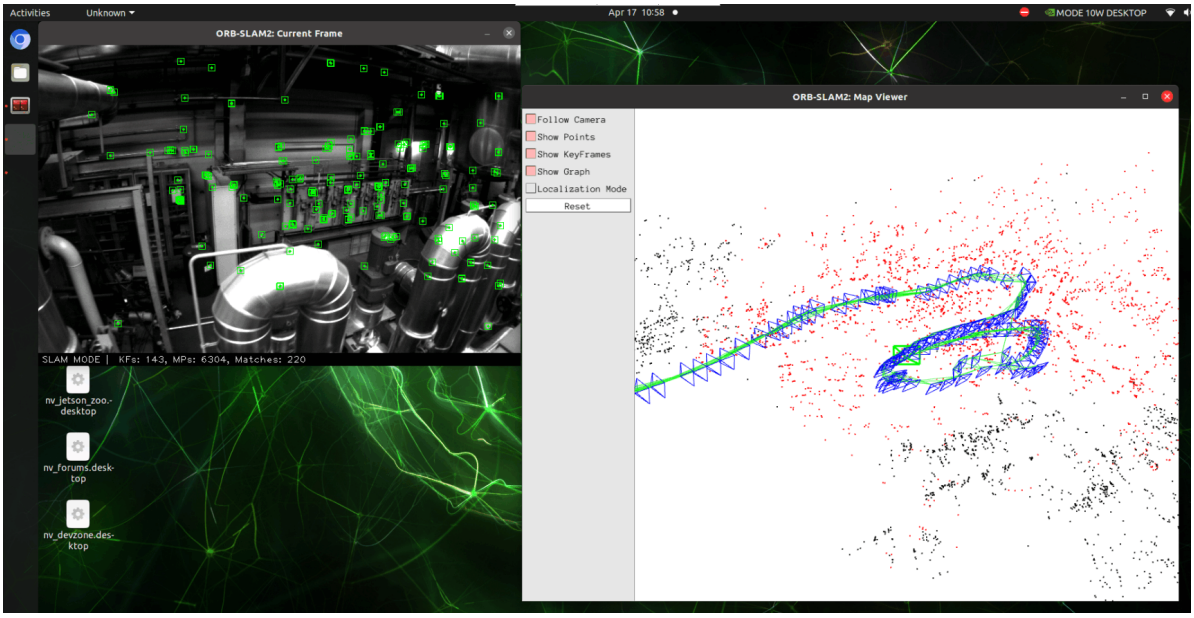
```
Examples/Monocular/mono_euroc Vocabulary/ORBvoc.txt Examples/Monocular/EuRoC.yaml
data/MH01/mav0/cam0/data Examples/Monocular/EuRoC_TimeStamps/MH01.txt
```

If the following problem is encountered:

```
-------
Start processing sequence ...
Images in the sequence: 3682

New Map created with 137 points
Track lost soon after initialisation, reseting...
System Reseting
The program 'ORB-SLAM2: Current Frame' received an X Window System error.
This probably reflects a bug in the program.
The error was 'BadValue (integer parameter out of range for operation)'.
  (Details: serial 52 error_code 2 request_code 130 minor_code 3)
  (Note to programmers: normally, X errors are reported asynchronously;
   that is, you will receive the error a while after causing it.
   To debug your program, run it with the --sync command line
   option to change this behavior. You can then get a meaningful
   backtrace from your debugger if you break on the gdk_x_error() function.)
```

is due to docker display problems, you can run a few more times will be successful, successful run interface is shown below, this interface will be displayed on the vnc or cart screen.



The blue boxes are the keyframes, the green boxes are the camera orientation, the black dots are the saved points, and the red dots are the points currently seen by the camera.

At the end of the test, the key frames are saved to the KeyFrameTrajectory.txt file in the current directory:

```
# 时间戳 位置（x y z）+ 姿态（x y z w）
# Time stamp Position (x y z) + Attitude (x y z w)
1341847980.722988 -0.0000464 0.0001060 0.0000110 -0.0000183 0.0001468 -0.0000286
1.0000000
```

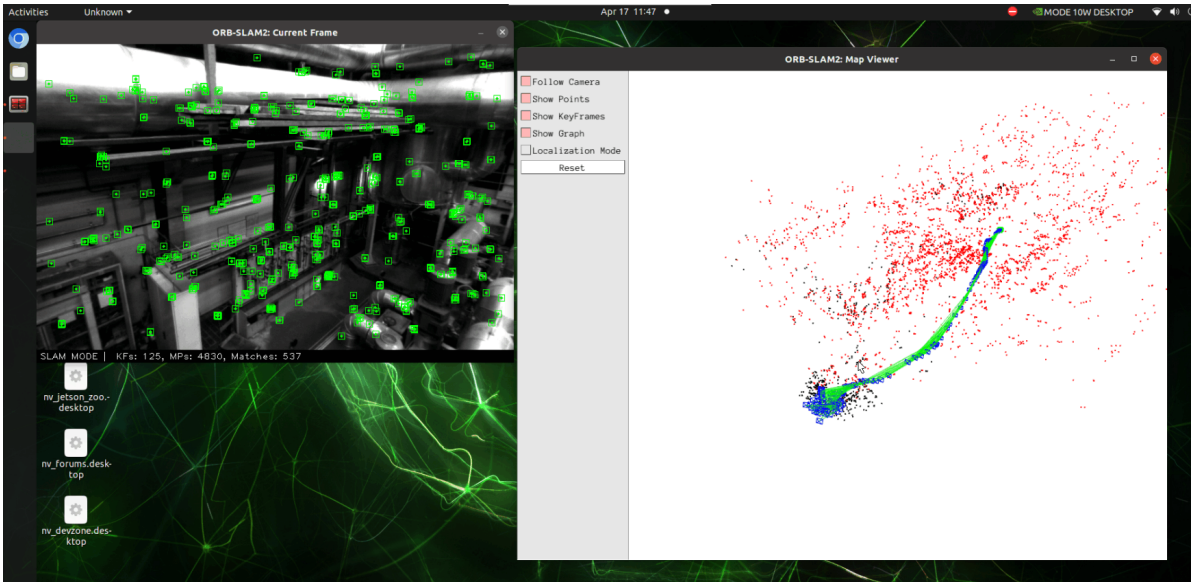
5.2.4, Binocular test

Here we take the EuRoC dataset as an example, enter the docker container, then go to the ORB_SLAM2 directory and run the following command:

```
cd /root/yahboomcar_ros2_ws/software/orbslam2/ORB_SLAM2-master

Examples/Stereo/stereo_euroc Vocabulary/ORBvoc.txt Examples/Stereo/EuRoC.yaml
data/MH01/mav0/cam0/data data/MH01/mav0/cam1/data
Examples/Stereo/EuRoC_TimeStamps/MH01.txt
```

The Successful Run screen is shown below:



The blue boxes are the keyframes, the green boxes are the camera orientation, the black dots are the saved points, and the red dots are the points currently seen by the camera.

At the end of the test, the keyframes are saved to the CameraTrajectory.txt file in the current directory.

```
# 时间戳 位置（x y z）+ 姿态（x y z w）
# Time stamp Position (x y z) + Attitude (x y z w)
1403636597.963556 -0.020445164 0.127641633 0.107868195 -0.136788622 -0.074876986
-0.044620439 0.986757994
```

5.2.5, RGBD test

Here we use the TUM dataset, this time adding the depth information. Here we need to match the rgb map and depth map, and merge the depth data and color map data into rgbd data.

The official script program associate.py is provided https://svncvpr.in.tum.de/cvpr-ros-pkg/trunk/rgbd_benchmark/rgbd_benchmark_tools/src/rgbd_benchmark_tools/associate.py

Download the associate.py file.

Use python to run associate.py, you can see the associations.txt file is generated in the specified path [this step has been configured before the product is shipped].

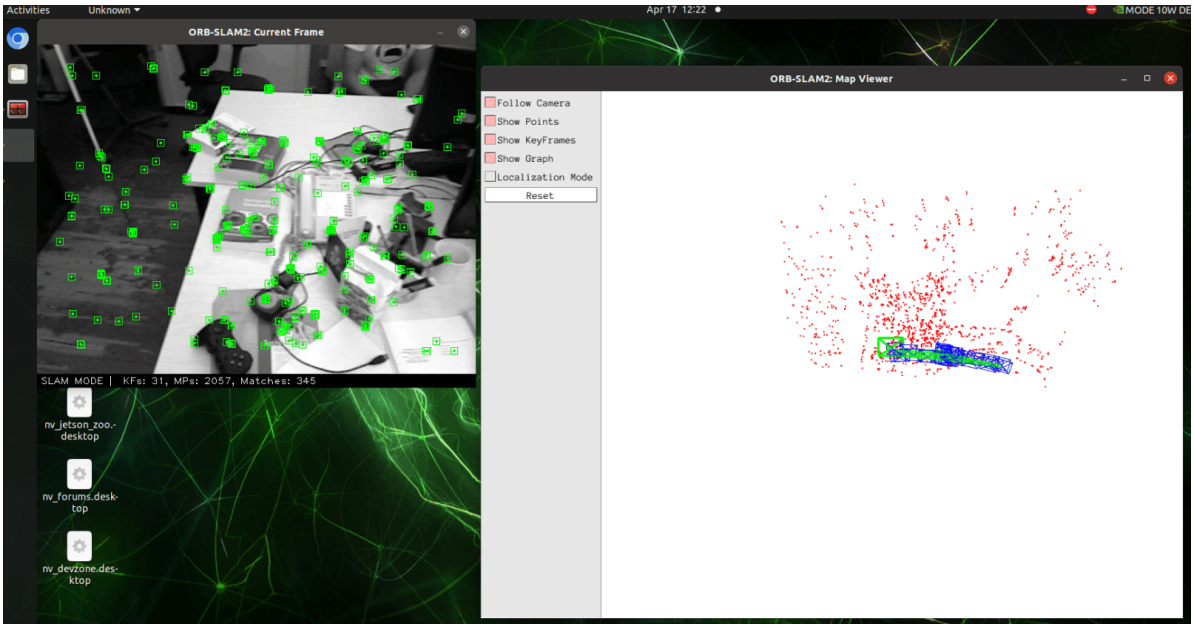
```
# 进入数据集目录 # Access to the dataset catalog
root@ubuntu:cd /root/yahboomcar_ros2_ws/software/orbslam2/ORB_SLAM2-
master/data/TUM/rgbd_dataset_freiburg1_xyz
# 执行associate.py，生成associations.txt文件
# Execute associate.py to generate associations.txt file
root@ubuntu:~/yahboomcar_ros2_ws/software/orbslam2/ORB_SLAM2-
master/data/TUM/rgbd_dataset_freiburg1_xyz# python3 associate.py rgb.txt
depth.txt > associations.txt
# 可以看到生成了associations.txt文件
# You can see that the associations.txt file has been generated.
root@ubuntu:~/yahboomcar_ros2_ws/software/orbslam2/ORB_SLAM2-
master/data/TUM/rgbd_dataset_freiburg1_xyz# ls
accelerometer.txt  associate.py  associations.txt  depth  depth.txt
groundtruth.txt  rgb  rgb.txt
```

Then test: go to the docker container and then to the ORB_SLAM2 directory and enter the following command:

```
cd /root/yahboomcar_ros2_ws/software/orbslam2/ORB_SLAM2-master

Examples/RGB-D/rgbd_tum Vocabulary/ORBvoc.txt Examples/RGB-D/TUM1.yaml
data/TUM/rgbd_dataset_freiburg1_xyz
data/TUM/rgbd_dataset_freiburg1_xyz/associations.txt
```

The Successful Run screen is shown below:



CameraTrajectory.txt and KeyFrameTrajectory.txt are also saved at the end of the run.

```
median tracking time: 0.0700995
mean tracking time: 0.0745612

Saving camera trajectory to CameraTrajectory.txt ...

trajectory saved!

Saving keyframe trajectory to KeyFrameTrajectory.txt ...

trajectory saved!
```

5.3, ORB_SLAM2 ROS2 camera test

The internal parameters of the camera have been modified before the product leaves the factory, **you can start testing directly from 5.3.2**. The learning method can be found in the subsection [5.3.1, Internal Reference Modification].

5.3.1, Internal Reference Modification

Camera running ORBSLAM before the need for internal parameters of the camera, so you must first calibrate the camera to be able to see the specific method of [Astra camera calibration] this lesson.

After calibration, move the [calibrationdata.tar.gz] file to the [home] directory.

```
sudo mv /tmp/calibrationdata.tar.gz ~
```

After unzipping, open [ost.yaml] inside that folder and find the camera internal reference matrix, e.g. the following:

```
camera_matrix:
    rows: 3
    cols: 3
    data: [ 683.90304, 0. , 294.56102,
            0. , 679.88513, 228.05956,
            0. , 0. , 1. ]
```

The internal reference matrix of the camera:

```
# fx 0 cx
# 0 fy cy
# 0 0 1
```

Modify the data in data to the corresponding values of [mono.yaml] and [rgbd.yaml] in the [params] folder under the [yahboomcar_slam] package.

Path to the params folder:

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_slam/params
```

```
Camera.fx: 683.90304
Camera.fy: 679.88513
Camera.cx: 294.56102
Camera.cy: 228.05956
```

5.3.2, Single-Eye Testing

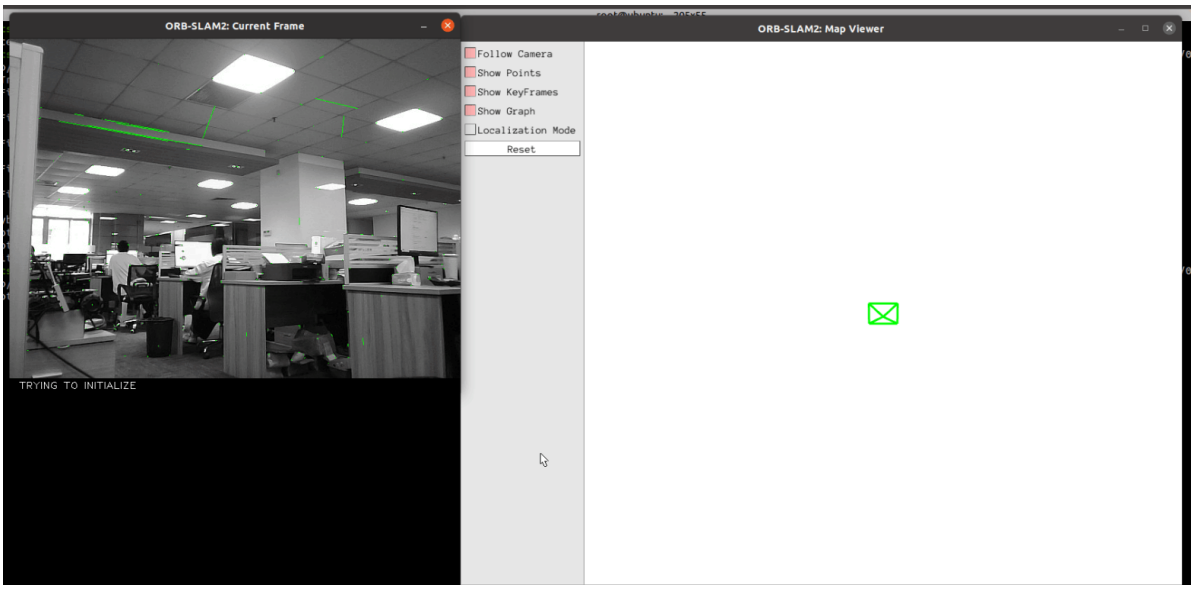
Enter the docker container:

Handheld robot can be used as a mobile carrier for mobile testing. If handheld there is no need to execute the next instruction, otherwise, execute it.

```
ros2 launch yahboomcar_slam move_driver_launch.py
```

Starting the camera ORB_SLAM2 test

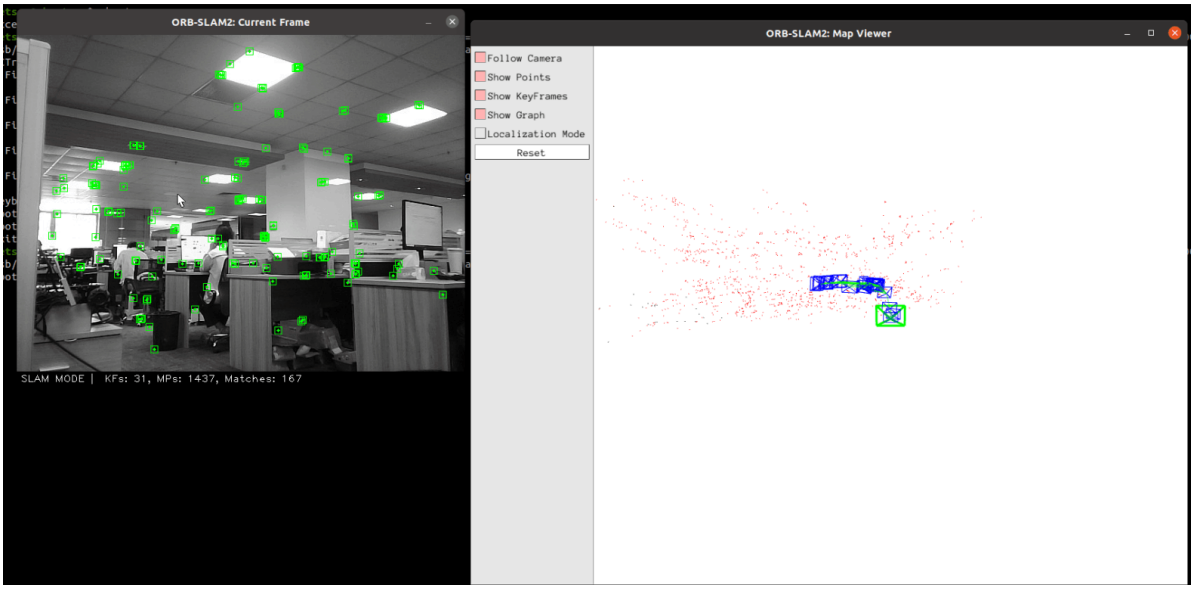
```
# 启动相机：这里只用了astra_pro_plus相机的rgb图像（相当于单目）
# Start camera: here only the rgb image of the astra_pro_plus camera is used
(equivalent to monocular)
ros2 launch astra_camera astro_pro_plus.launch.xml
# 启动orbslam节点 # Launch orbslam nodes
ros2 launch yahboomcar_slam orbslam_ros_launch.py orb_slam_type:=mono
```



When the command is executed, there is only a green box in the interface of [ORB_SLAM2:Map Viewer], and the initialization is being tried in the interface of [ORB_SLAM2:Current Frame], at this time the camera is moved slowly up and down, left and right, to find the feature points in the frame and initialize the slam.

After the test, the key frames are saved in the KeyFrameTrajectory.txt file in the following directory:

```
/root/yahboomcar_ros2_ws/software/library_ws/src/ros2-
ORB_SLAM2/src/monocular/KeyFrameTrajectory.txt
```



As shown in the figure above, at this time to enter the [SLAM MODE] mode, running monocular must be continuous access to each frame of the image in order to camera positioning, if you select the upper left [Localization Mode] pure positioning mode, the camera will not be able to find their own position, it is necessary to start from the beginning of a new access to the key frames.

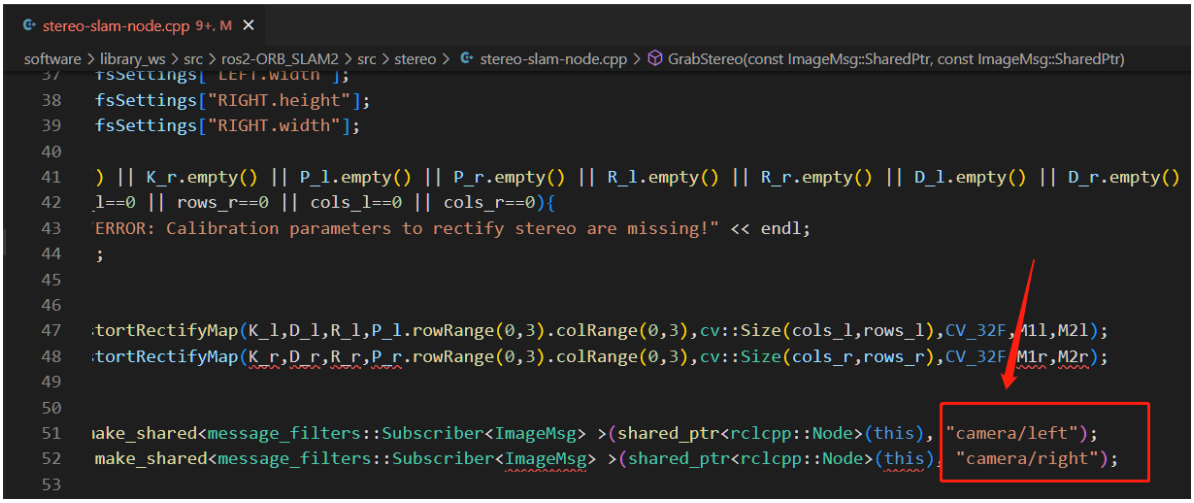
5.3.3, Binocular test

Since there is no binocular camera on the cart, we won't do a demo here. Those who have binocular camera can follow the following steps to test:

Enter the docker container:

1. Start the binocular camera node and check the name of the topic published by the camera
2. Modify the subscription topic of binocular camera in orbslam to the topic published by your own binocular camera:

```
/root/yahboomcar_ros2_ws/software/library_ws/src/ros2-
ORB_SLAM2/src/stereo/stereo-slam-node.cpp
```



3. Recompile the ros2_orbslam function package:

```
cd ~/yahboomcar_ros2_ws/software/library_ws/
colcon build --packages-select ros2_orbslam
```

4. Restart the binocular camera node, then run the orbslam node

```
ros2 launch yahboomcar_slam orbslam_ros_launch.py orb_slam_type:=stereo
```

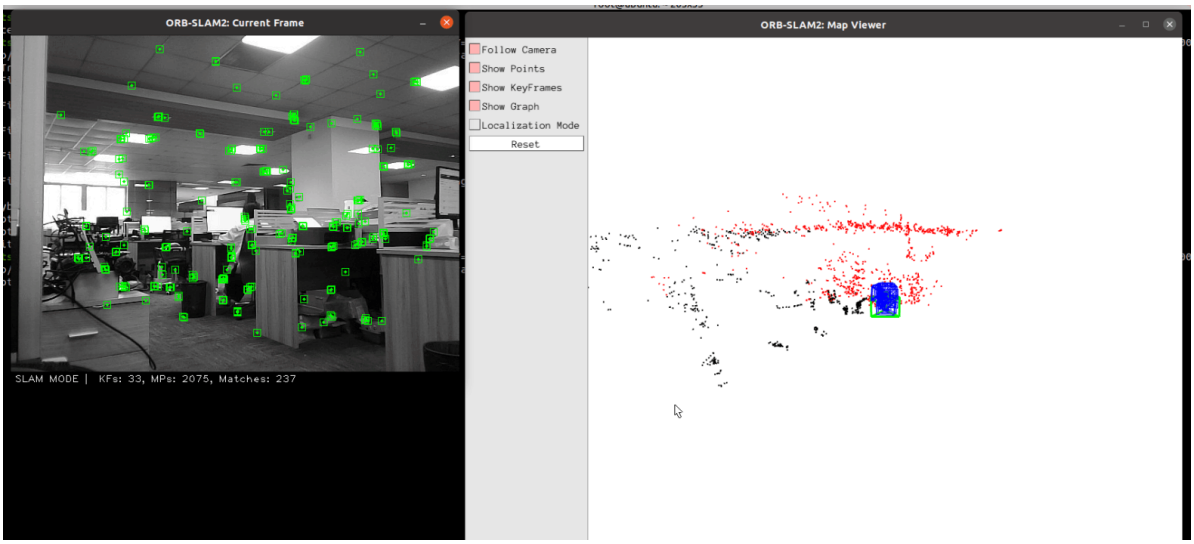
The key frames are saved in the KeyFrameTrajectory.txt file in the following directory at the end of the test:

```
/root/yahboomcar_ros2_ws/software/library_ws/src/ros2-
ORB_SLAM2/src/stereo/KeyFrameTrajectory.txt
```

5.3.4, RGBD testing

Enter the docker container:

```
# 启动相机： # Start the camera:
ros2 launch astra_camera astro_pro_plus.launch.xml
# 启动orbslam节点 # Launch orbslam nodes
ros2 launch yahboomcar_slam orbslam_ros_launch.py orb_slam_type:=rgbd
```



RGBD does not have to run monocular as it must acquire each frame consecutively, if you select the upper left figure [Localization Mode] pure localization mode, you can just get the key frames for localization.

After the test, the key frames are saved in the KeyFrameTrajectory.txt file in the following directory:

```
/root/yahboomcar_ros2_ws/software/library_ws/src/ros2-
ORB_SLAM2/src/rgbd/KeyFrameTrajectory.txt
```