5、ORB_SLAM2 basics

5、ORB_SLAM2 basics

5.1、Introduction

5.2、Official case

5.2.1、Dataset location

5.2.2、Entering Docker Container

5.2.3、Monocular testing

5.2.4、Binocular testing

5.2.5、RGBD testing

5.3、ORB_ SLAM2 ROS2 camera testing

5.3.1、Internal reference modification

5.3.2、Monocular testing

5.3.3、Binocular testing

5.3.4、RGBD testing

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 reference configurations for software and hardware are as follows:

- Reference model：ROSMASTER X3
- Robot hardware configuration：Arm series main controller, 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；

The ORB-SLAM2 version supports monocular, binocular and RGBD interfaces;

The ORB-SLAM3 version adds IMU coupling and supports fisheye cameras.

All steps of ORB-SLAM uniformly use the ORB features of the image. ORB feature is a very fast feature extraction method, which is rotation invariant and can use pyramids to build scale invariance. The use of unified ORB features helps the SLAM algorithm to be consistent in the steps of feature extraction and tracking, key frame selection, 3D reconstruction, and closed-loop detection. The system is also robust to vigorous motion, supporting wide-baseline closed-loop detection and relocalization, including fully automatic initialization. Since the ORB-SLAM system is a SLAM system based on feature points, it can calculate the trajectory of the camera 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) The results of RGBD show that more accuracy can be obtained by using BA than ICP or minimization based on photometric and depth errors.

3) By using the far and near points in the binocular, as well as the monocular observation, the binocular result is more accurate than the direct binocular SLAM algorithm.

4) The light positioning mode can effectively reuse the map.

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

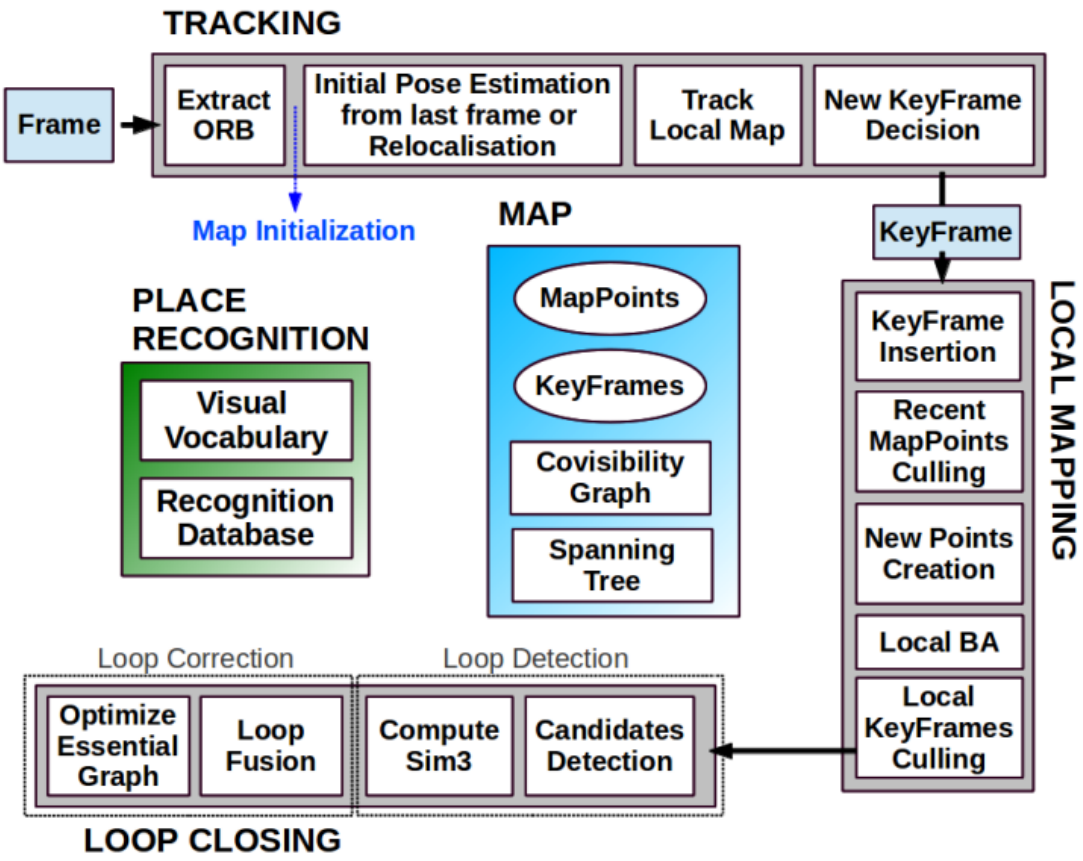


Fig. 1. ORB-SLAM system overview, showing all the steps performed by the tracking, local mapping and loop closing threads. The main components of the place recognition module and the map are also shown.

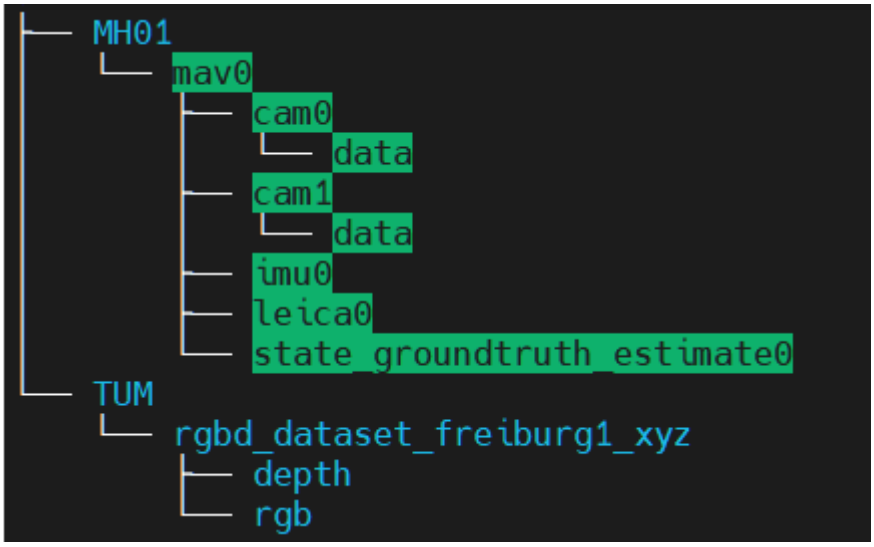
5.2、Official case

Note: The commands or positions mentioned below, unless otherwise specified, refer to the commands or positions in the Docker container.

5.2.1、Dataset location

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

It has downloaded EuRoC's MH01 dataset and TUM's rgbd_ Dataset_ Freiburg1_ Xyz dataset.



If you need other datasets, you can download them at the following address:

```
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 Docker Container

For the steps to enter the Docker container, please refer to 【the Docker Course Chapter -5. Entering the Docker Container for Robots】

```
#Raspberry Pi 5 master user
~/run_docker.sh
#Orin master user
~/ros2_orbslam2.sh
```

Here, take orin master as an example and modify the corresponding car model

```
                        [System Information]
IP_Address_1: 192.168.2.102
IP_Address_2: 172.17.0.1
------------------------------------------------------
ROS_DOMAIN_ID: 28 | ROS: humble
my_robot_type: x3 | my_lidar: s2 | my_camera: astraplus
------------------------------------------------------
jetson@yahboom:~$ ./ros2_orbslam2.sh
access control disabled, clients can connect from any host
WARNING: Published ports are discarded when using host network mode
------------------------------------------------------
ROS_DOMAIN_ID: 28
ros-foxy: OrbSlam2 | my_robot_type: x3 | my_camera: astraplus
------------------------------------------------------
root@yahboom:/#
```

5.2.3、Monocular testing

Taking 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:

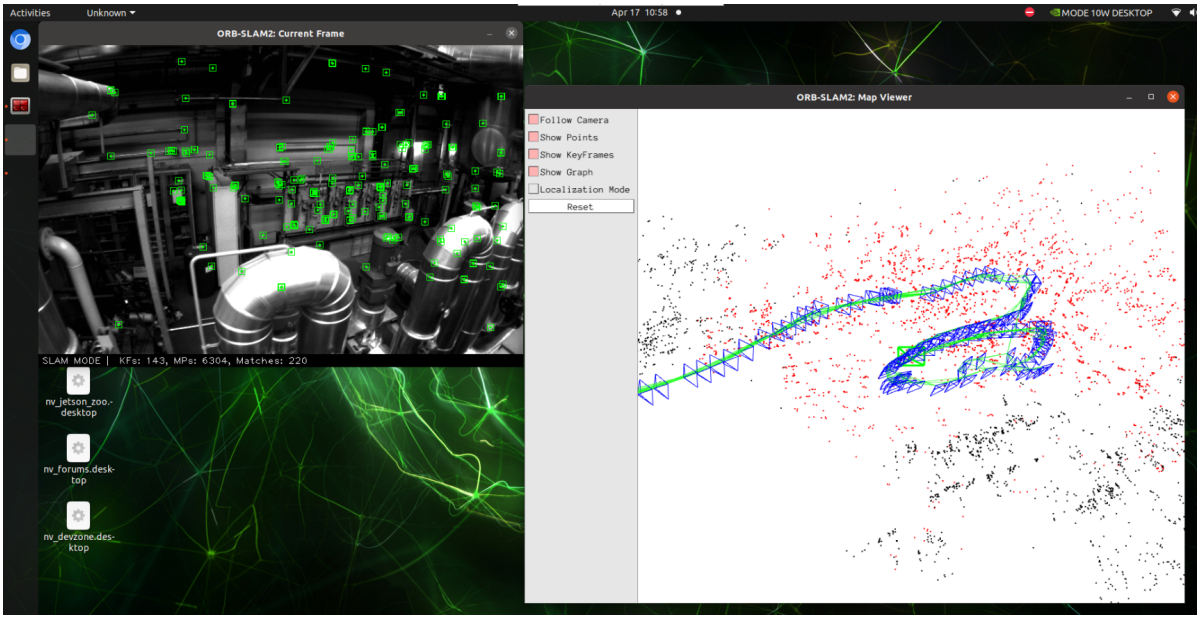
```
Examples/Monocular/mono_euroc Vocabulary/ORBvoc.txt Examples/Monocular/EuRoC.yaml
data/MH01/mav0/cam0/data Examples/Monocular/EuRoC_TimeStamps/MH01.txt
```

If encountering the following problems:

```
-------
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.)
```

Due to the issue with the Docker display, it can be successfully run multiple times. The successful run interface is shown in the following figure, which will be displayed on the VNC or car screen.



The blue box represents the keyframes, the green box represents the camera orientation, the black dots represent the saved points, and the red dots represent the points currently seen by the camera.

After the test is completed, save the keyframes to the KeyFrameTrajectory.txt file in the current directory:

```
# 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
```

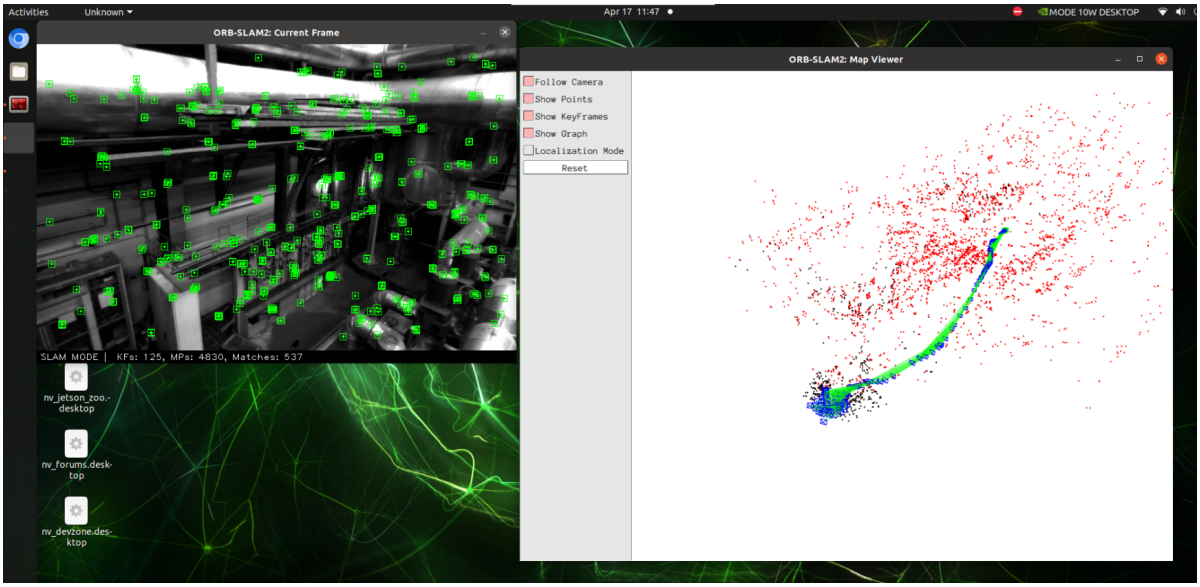
5.2.4、Binocular testing

Taking the EuRoC dataset as an example, enter the Docker container and then enter the ORB_SLAM2 directory, 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 operation interface is shown in the following figure:



The blue box represents the keyframes, the green box represents the camera orientation, the black dots represent the saved points, and the red dots represent the points currently seen by the camera.

After the test is completed, save the keyframes to the CameraTrajectory.txt file in the current directory

```
# 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 testing

Here, we use the TUM dataset and add depth information this time. Here, it is necessary to match the rgb and depth images, and merge the depth data and color image 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

Using Python to run associate.py, you can see that the associations.txt file has been generated in the specified path. 【This step is already configured before the product leaves the factory】

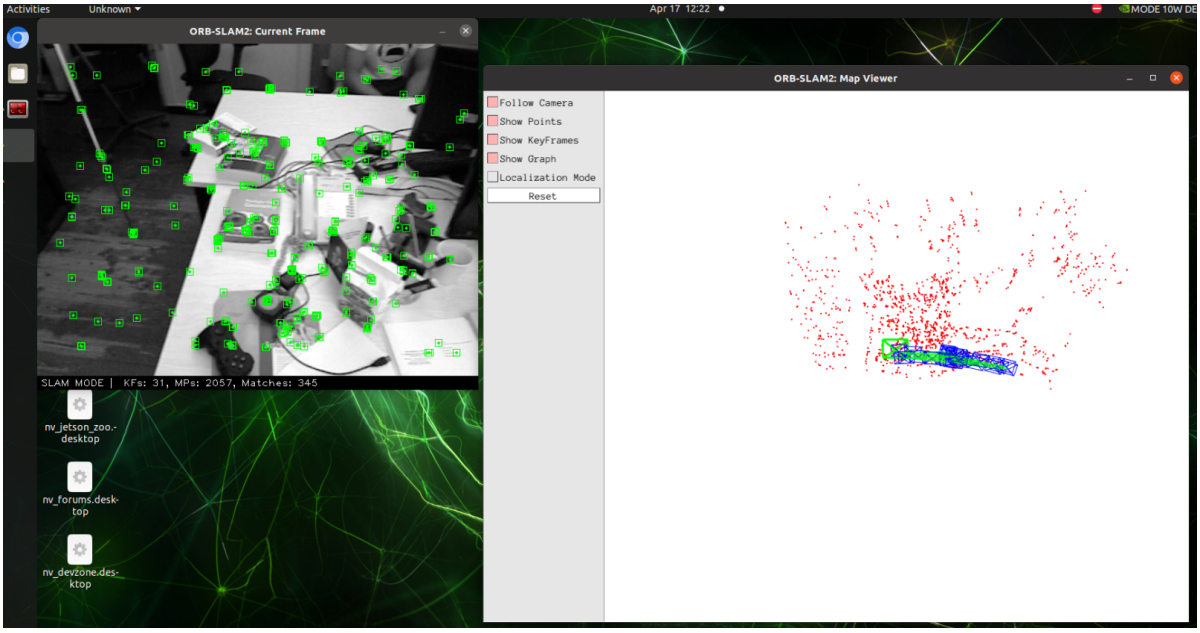
```
# Enter Dataset Directory
root@ubuntu:cd /root/yahboomcar_ros2_ws/software/orbslam2/ORB_SLAM2-
master/data/TUM/rgbd_dataset_freiburg1_xyz
# Execute associate. py to generate the associations. txt file
root@ubuntu:~/yahboomcar_ros2_ws/software/orbslam2/ORB_SLAM2-
master/data/TUM/rgbd_dataset_freiburg1_xyz# python3 associate.py rgb.txt
depth.txt > associations.txt
# 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 conduct testing: enter the Docker container, and then enter the ORB_ In the SLAM2 directory, enter the following command:

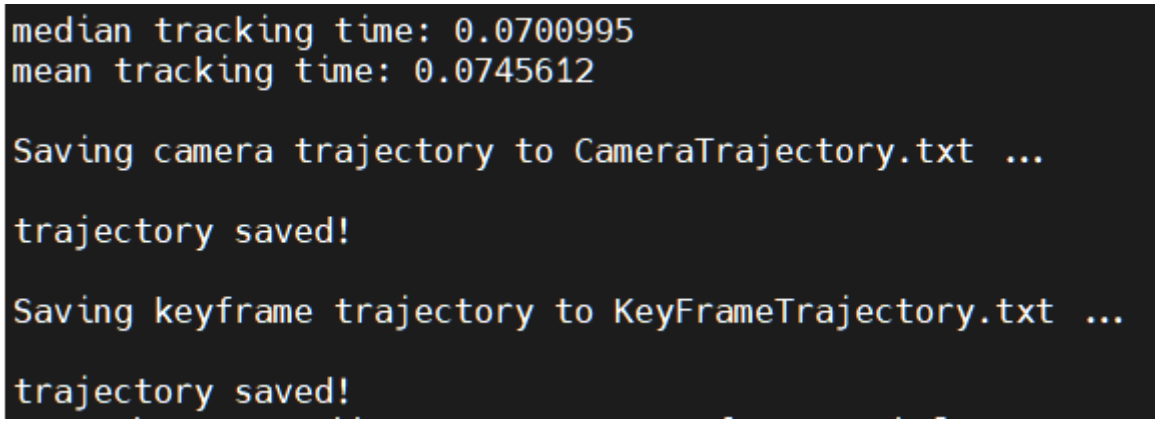
```
cd /root/yahboomcar_ros2_ws/software/orbslam2/ORB_SLAM2-master

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 operation interface is shown in the following figure:



After running, CameraTrajectory.txt and KeyFrameTrajectory.txt will also be saved



5.3、ORB_ SLAM2 ROS2 camera testing

The camera internal parameters have been modified before the product leaves the factory, and testing can start directly from 5.3.2. The required learning methods can be found in the section [5.3.1. Internal Parameter Modification].

5.3.1、Internal reference modification

Before running ORBSLAM, the camera requires internal parameters, so camera calibration must be performed first. The specific method can be found in the Astra Camera Calibration lesson.

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

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

After decompressing, open **[ost. yaml]** in the folder and find the camera's internal parameter matrix, such as:

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

Camera's internal parameter matrix:

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

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

Params folder path:

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

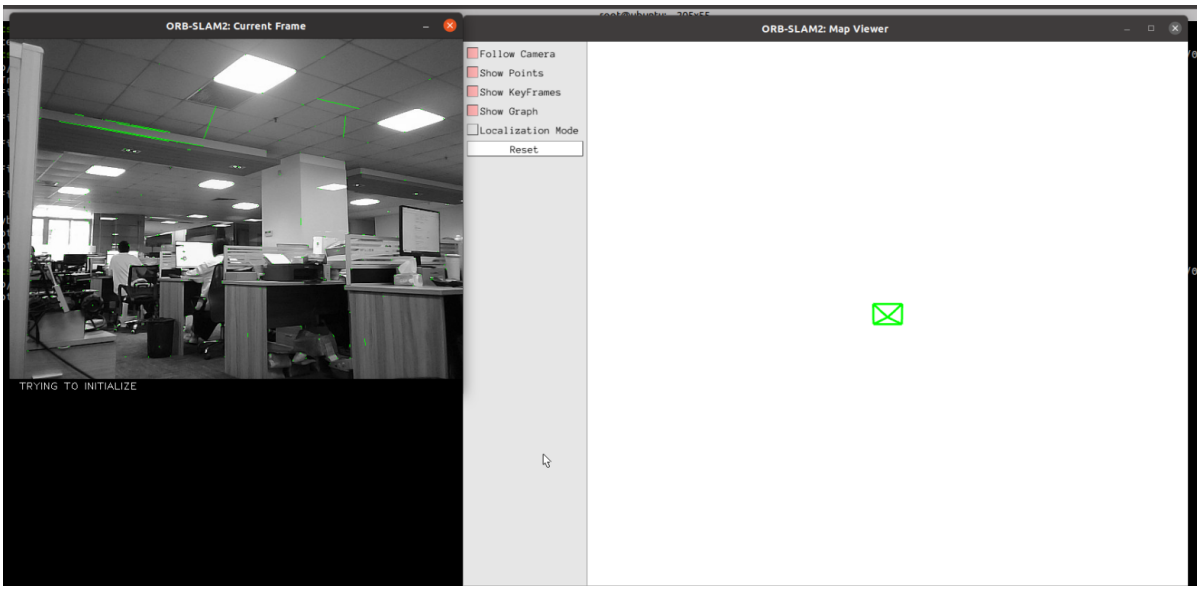
```
Camera.fx: 683.90304
Camera.fy: 679.88513
Camera.cx: 294.56102
Camera.cy: 228.05956
```

5.3.2、Monocular testing

Enter Docker container:

Start camera ORB_ SLAM2 testing

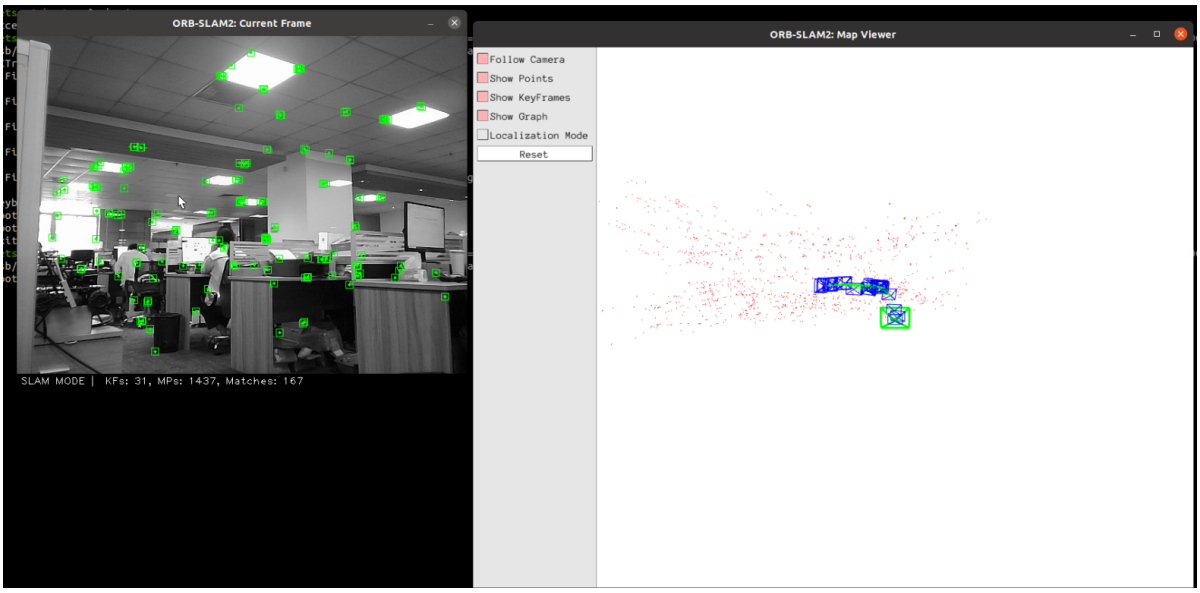
```
# Activate camera: only Astra is used here_ Pro_ RGB image of plus camera
(equivalent to monocular)
ros2 launch astra_camera astro_pro_plus.launch.xml
# Start the orbslam node
ros2 launch yahboomcar_slam orbslam_ros_launch.py orb_slam_type:=mono
```



When the command is executed, there is only a green box in the interface of the【 ORB_SLAM2: Map Viewer】,In the interface of 【ORB_SLAM2: Current Frame】, initialization is being attempted. At this time, slowly move the camera up, down, left, right, and search for feature points in the screen and initialize the SLAM.

After the test is completed, the keyframes 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 above figure, at this point, the camera enters the **[SLAM Mode]** mode and must continuously obtain each frame of image for camera positioning. If the **[Localization Mode]** pure positioning mode in the left image is selected, the camera will not be able to find its own position and must start from scratch to obtain key frames.

5.3.3、Binocular testing

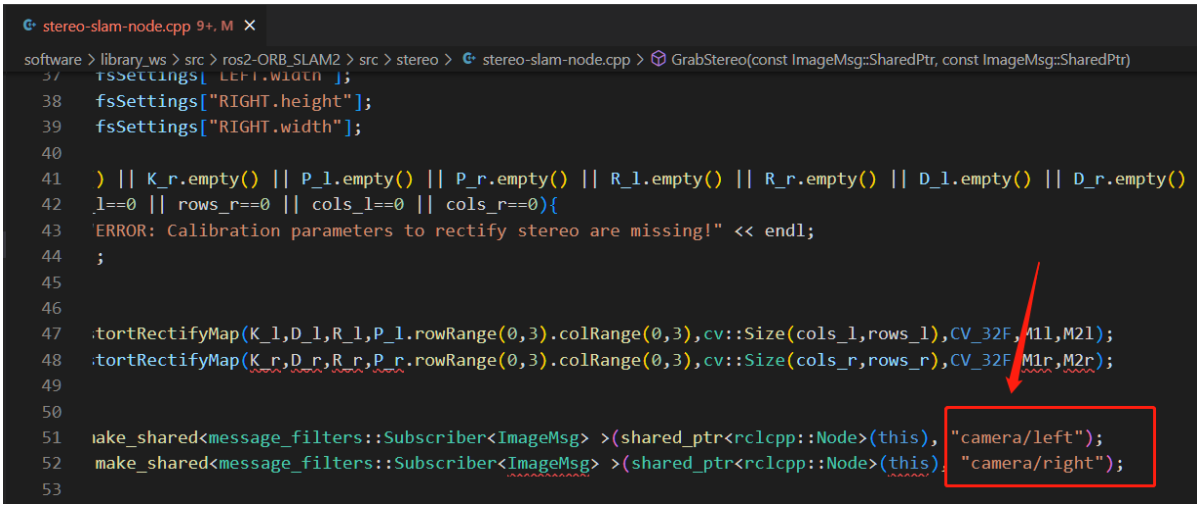
Since there is no binocular camera on the car, there will be no demonstration here. Users with binocular cameras can follow the following steps to test:

Enter Docker container:

1、Launch the binocular camera node to view the topic names posted by the camera

2、Modify the subscription topic for binocular cameras in orbslam to your own published topic for binocular cameras:

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



3、Recompile ROS2_ Orbslam feature pack:

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

4、Restart the binocular camera node and then run the orbslam node

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

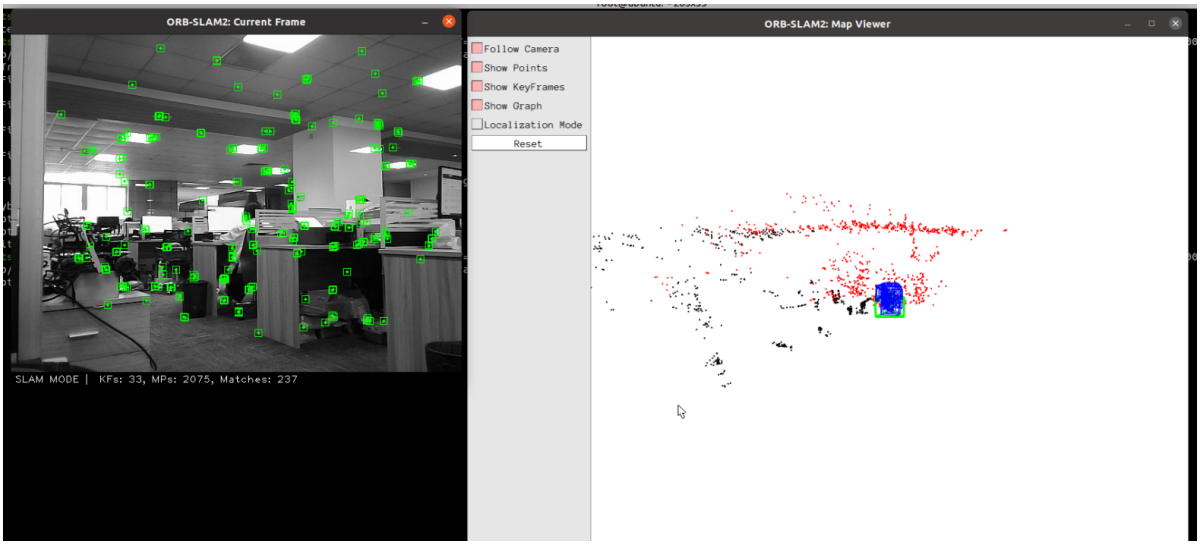
After the test is completed, the keyframes are saved in the KeyFrameTrajectory.txt file in the following directory:

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

5.3.4、RGBD testing

Enter Docker container:

```
# Start camera:
ros2 launch astra_camera astro_pro_plus.launch.xml
# Start the orbslam node
ros2 launch yahboomcar_slam orbslam_ros_launch.py orb_slam_type:=rgbd
```



RGBD does not need to continuously obtain each frame of image like running a single camera. If you choose the pure **【localization mode】** in the upper left image, you can locate the key frame just obtained.

After the test is completed, the keyframes 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
```