

8 ORB_SLAM2 basics

8 ORB_SLAM2 basics

8.1 Introduction

8.2 Official case

8.2.1 Monocular test

8.2.2 binocular test

8.2.3 RGBD test

8.3 ORB_SLAM2_ROS camera test

8.3.1 Internal reference modification

8.3.2 Monocular

8.3.3 Monocular AR

8.3.4 RGBD

Official website: <http://webdiis.unizar.es/~raulmur/orbslam/>

ASL Dataset: <https://projects.asl.ethz.ch/datasets/doku.php?id=kmaavvisualinertialdatasets>

mono Dataset: <https://vision.in.tum.de/data/datasets/rgbd-dataset/download>

stereo Dataset: http://robotics.ethz.ch/~asl-datasets/ijrr_euroc_mav_dataset/machine_hall/MH_01_easy/

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

8.1 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.

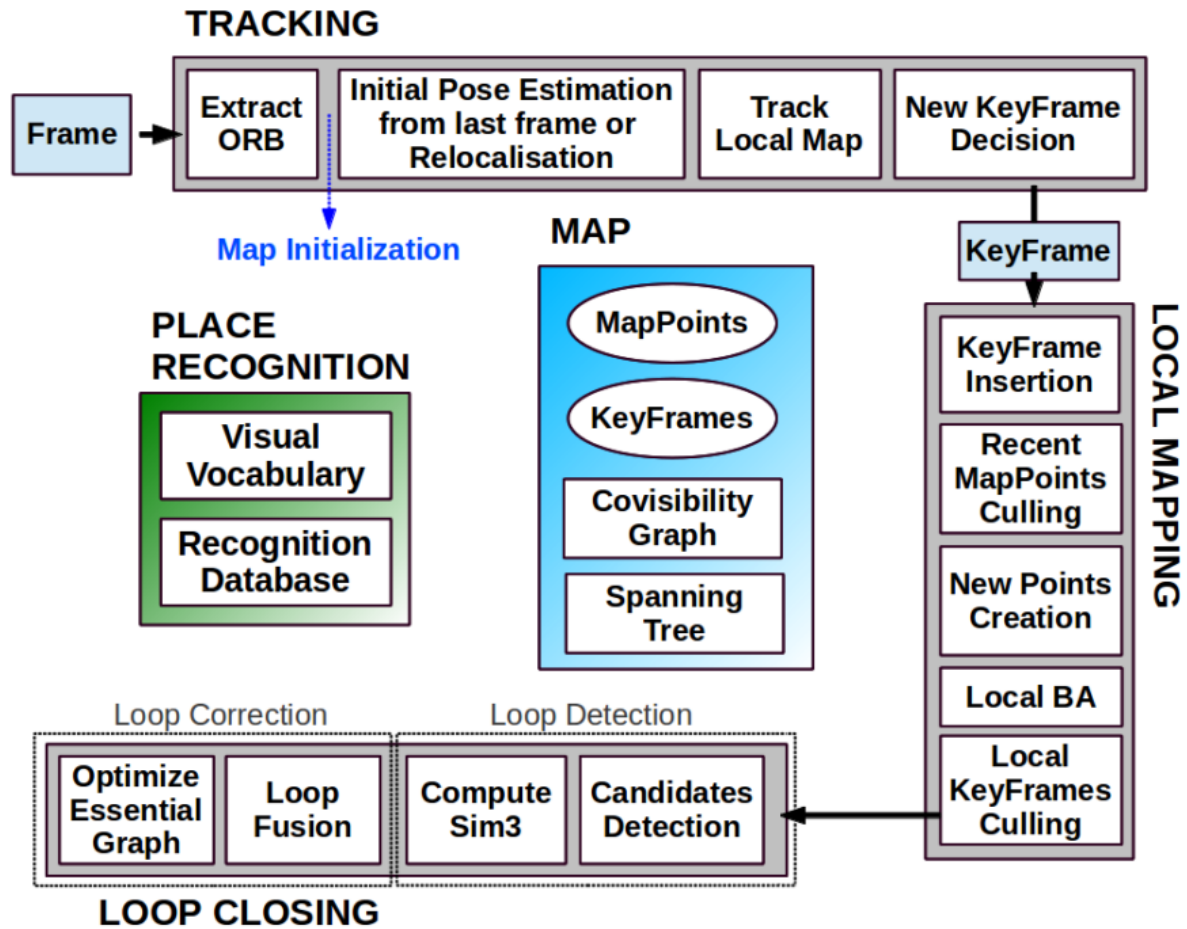
All steps of ORB-SLAM uniformly use the ORB features of the image. ORB feature is a very fast feature extraction method, which is rotation invariant and can use pyramids to build scale invariance. The use of unified ORB features helps the SLAM algorithm to be consistent in the steps of feature extraction 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.



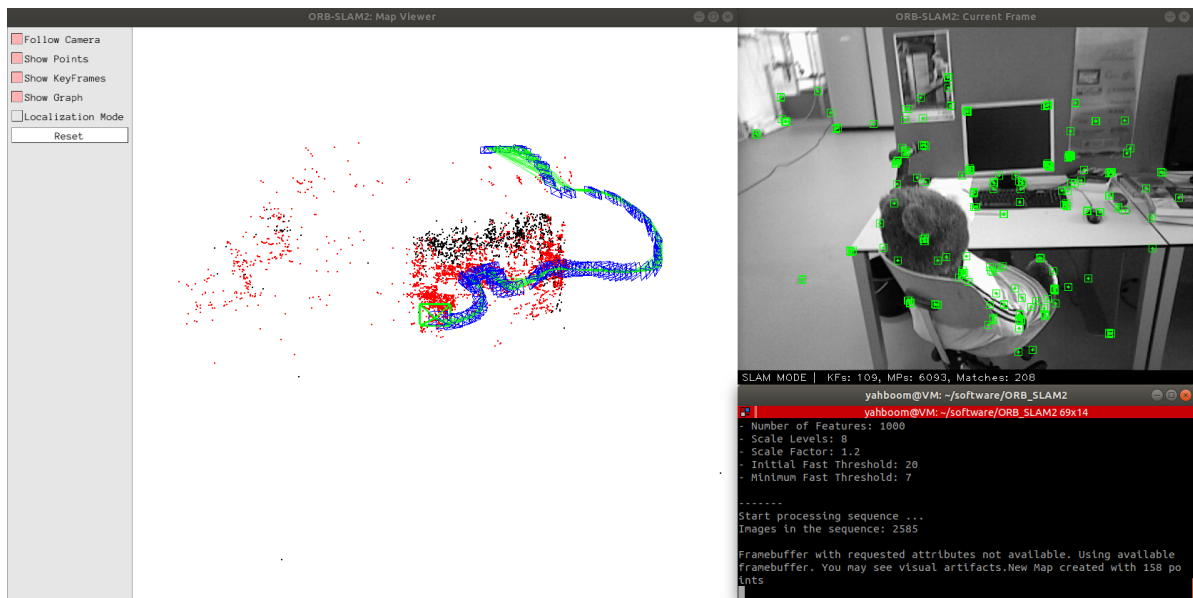
8.2 Official case

Open the terminal and enter ORB_SLAM2

```
cd ~/software/ORB_SLAM2
```

8.2.1 Monocular test

```
./Examples/Monocular/mono_tum vocabulary/ORBvoc.txt Examples/Monocular/TUM3.yaml  
~/software/ORB_SLAM_data/rgbd_dataset_freiburg3_long_office_household
```



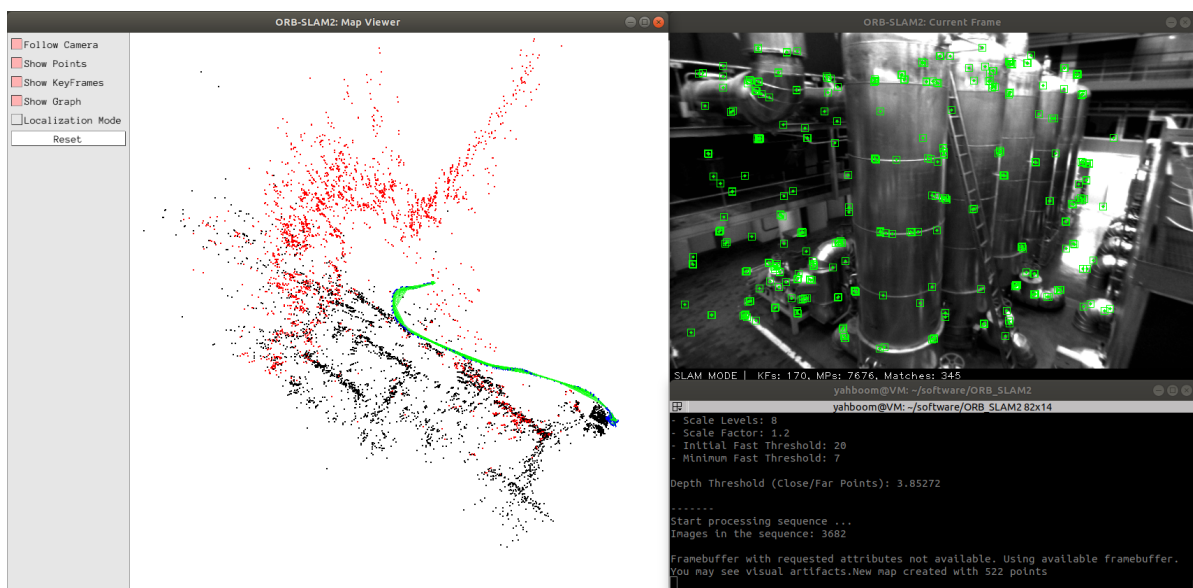
The blue frame is the key frame, the green frame is the camera orientation, the black point is the saved point, and the red point is the point currently seen by the camera.

After the test, the key frame is saved to the KeyFrameTrajectory.txt file in the current directory

```
# timestamp position(xyz) + pose(xyzw)
1341847980.722988 - 0.0000464 0.0001060 0.0000110 - 0.0000183 0.0001468 -
0.0000286 1.0000000
```

8.2.2 binocular test

```
./Examples/Stereo/stereo_euroc vocabulary/ORBvoc.txt Examples/Stereo/EuRoC.yaml1
~/software/ORB_SLAM_data/MH_01_easy/mav0/cam0/data
~/software/ORB_SLAM_data/MH_01_easy/mav0/cam1/data
Examples/Stereo/EuRoC_TimeStamps/MH01.txt
```



The blue frame is the key frame, the green frame is the camera orientation, the black point is the saved point, and the red point is the point currently seen by the camera.

After the test, the keyframes are saved to the CameraTrajectory.txt file in the current directory

```
# timestamp position(xyz) + pose(xyzw)
1403636597.963556 - 0.020445164 0.127641633 0.107868195 - 0.136788622 -
0.074876986 - 0.044620439 0.986757994
```

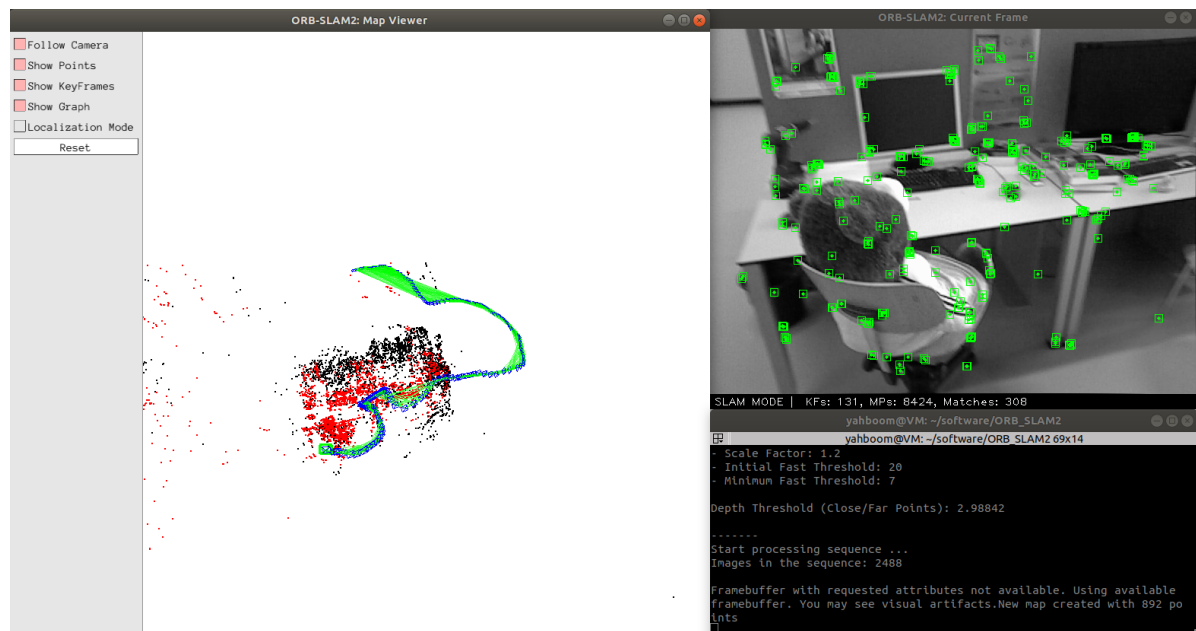
8.2.3 RGBD test

Combine depth data and color map data into rgbd data and save it to associations.txt file

```
cd ~/software/ORB_SLAM2_data/rgbd_dataset_freiburg3_long_office_household
python2 associate.py
```

test command

```
./Examples/RGB-D/rgbd_tum Vocabulary/ORBvoc.txt Examples/RGB-D/TUM3.yaml
~/software/ORB_SLAM2_data/rgbd_dataset_freiburg3_long_office_household
~/software/ORB_SLAM2_data/rgbd_dataset_freiburg3_long_office_household/associatio
ns.txt
```



8.3 ORB_SLAM2_ROS camera test

The internal parameters of the camera have been modified before the product leaves the factory. If you want to learn the method, see the section [8.3.1, Internal Parameters Modification]. It can be hand-held or a robot can be used as a mobile carrier for mobile testing.

If you hold it, you don't need to execute the next command, otherwise, execute it. (Robot side)

```
roslaunch yahboomcar_slam camera_driver.launch
```

Start the camera ORB_SLAM2 test(Robot or virtual machine)

```
roslaunch yahboomcar_slam camera_orb_slam.launch orb_slam_type:=mono
```

- [orb_slam_type] parameter: [mono, monoAR, rgbd], a total of three types are optional, monocular, monocular AR, rgbd.

8.3.1 Internal reference modification

The camera needs the internal parameters of the camera before running ORBSLAM, so the camera must be calibrated first. The specific method can be seen in the lesson [02, Astra Camera Calibration].

Start the monocular camera

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

Start the calibration node

```
roslaunch camera_calibration cameracalibrator.py image:=/usb_cam/image_raw  
camera:=/usb_cam --size 9x6 --square 0.02
```

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

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

After decompression, open [ost.yaml] in the folder and find the camera internal parameter matrix, for example: the following content.

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

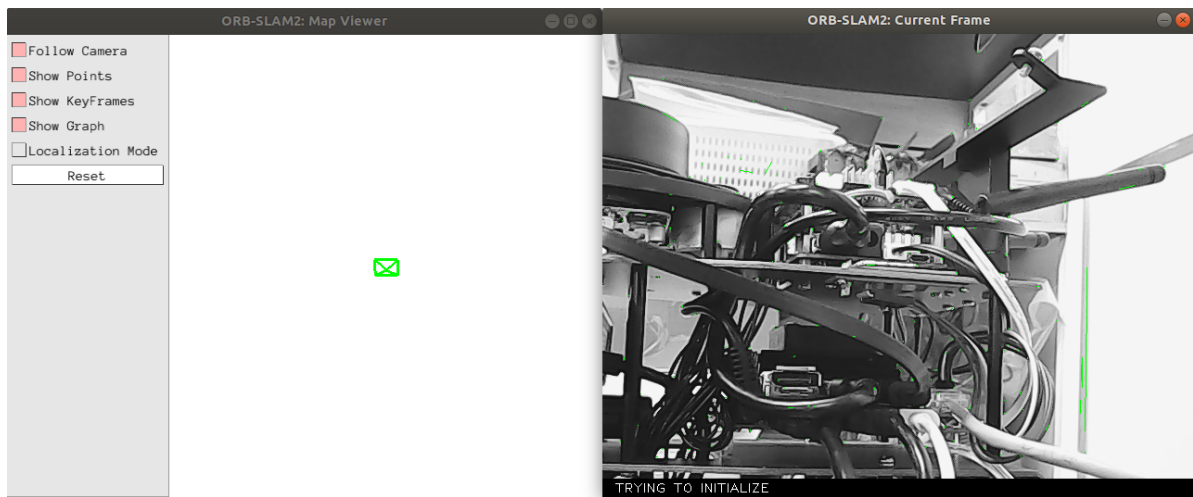
The camera's internal parameter matrix

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

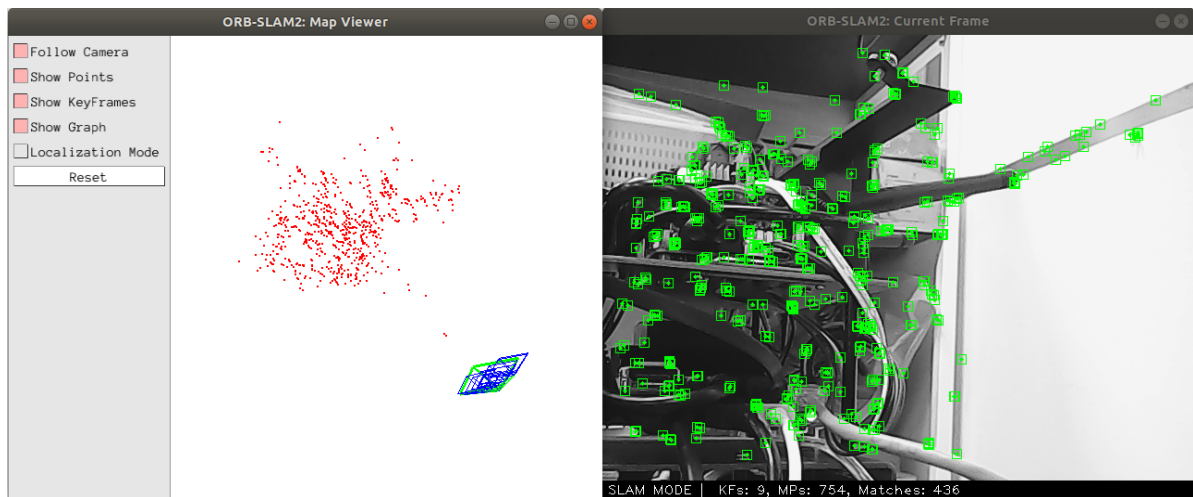
Modify the data in data to the values corresponding to [astra.yaml] and [astra1.0.yaml] in the [param] folder under the [yahboomcar_slam] function package.

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

8.3.2 Monocular

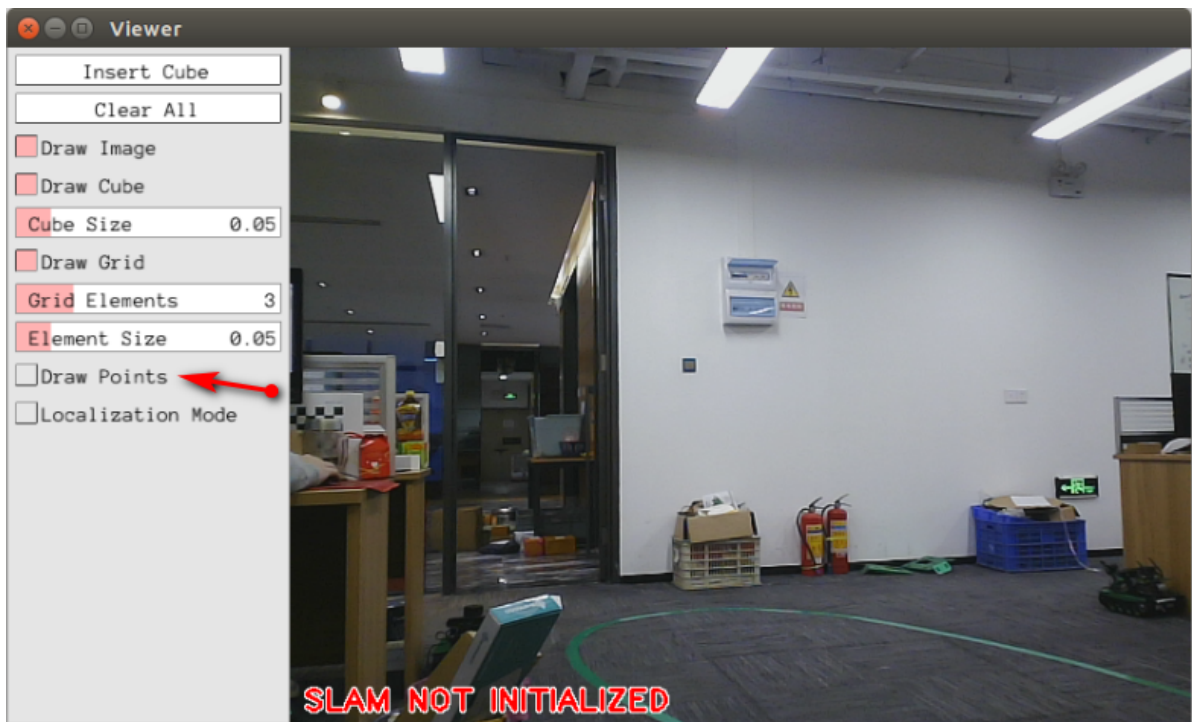


When the command is executed, there is only a green frame in the [ORB_SLAM2:Map Viewer] interface, and the [ORB_SLAM2:Current Frame] interface is trying to initialize. At this time, slowly move the camera up and down, left and right, find the feature points in the screen and initialize the slam.

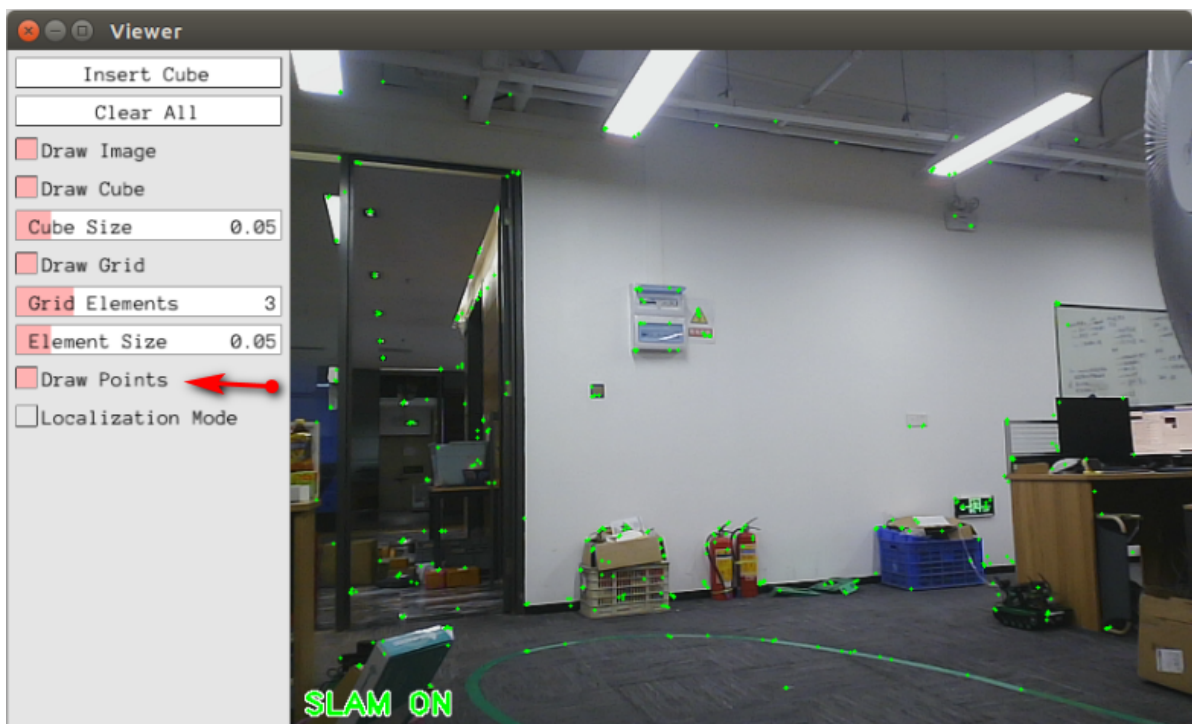


As shown in the figure above, enter the [SLAM MODE] mode at this time. When running the monocular, you must continuously obtain each frame of image to position the camera. If you select the pure positioning mode in the upper left figure [Localization Mode], the camera will not be able to find its own position. You have to start over to get keyframes.

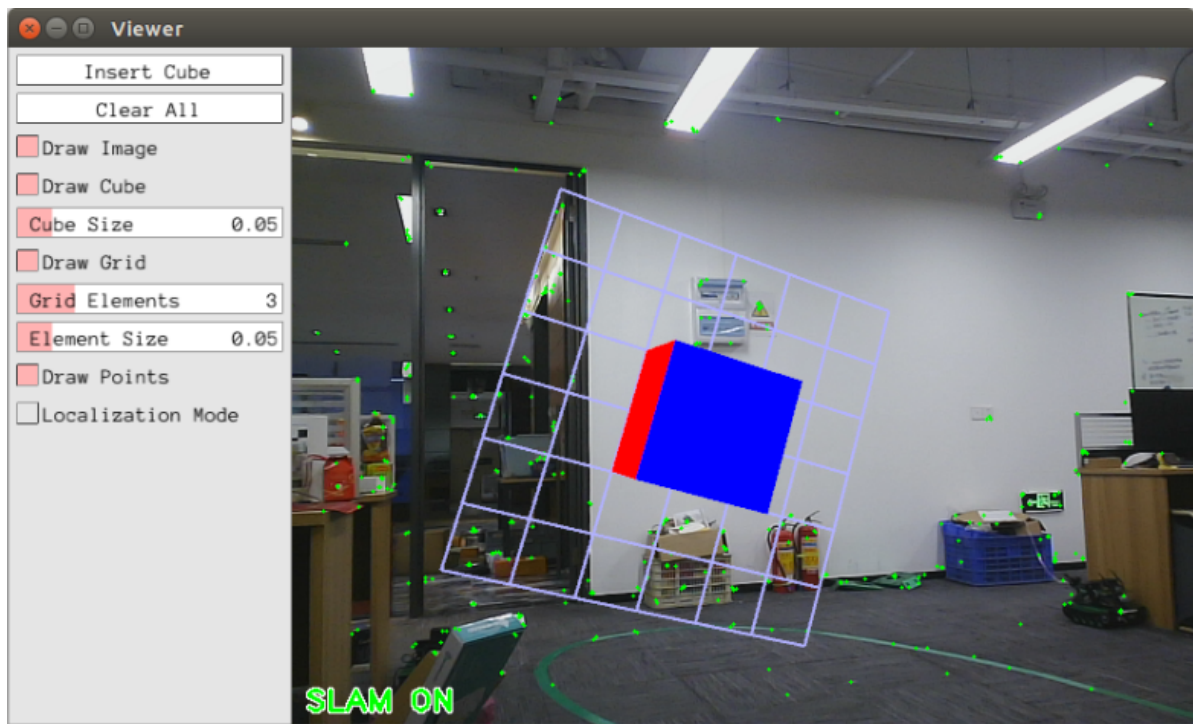
8.3.3 Monocular AR



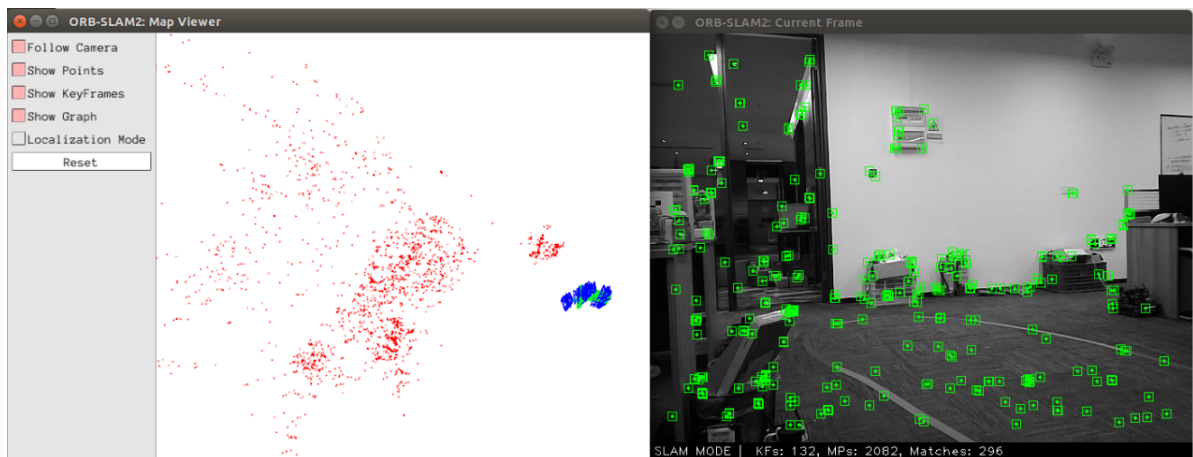
When the command is executed, there is only one interface and it displays [slam not initialized]. The slam is not initialized. Click the box on the left of [Draw Points] in the left column to display the feature points. At this time, slowly move the camera up, down, left and right, find the feature points in the picture and initialize the slam.



As shown in the figure above, enter the [SLAM ON] mode at this time, and click the [Insert Cube] screen to insert an AR cube in the place that is considered to be a plane. And the AR block will always be in a fixed position in the scene. not the fixed position of the camera screen. Click 【Clear All】 to clear.



8.3.4 RGBD



RGBD does not have to continuously acquire each frame of image like running a monocular. If you select the pure positioning mode in the upper left picture [Localization Mode], you can locate the key frame just acquired.