Robotic Navigation and Exploration

Lab 4 & Homework 2: ORB-SLAM on Jetbot

Min-Chun Hu anitahu@cs.nthu.edu.tw
CS, NTHU

Lab 4

ORB-SLAM and Python Binding

GitHub Repository

- ORB-SLAM2:
  - https://github.com/raulmur/ORB_SLAM2
- ORB-SLAM2 Python Binding:
  - https://github.com/jskinn/ORB_SLAM2-PythonBindings
- Camera Calibration and Jetbot Example on ORB-SLAM:
  - https://github.com/jerrywiston/ORB_SLAM2-Python-Jetbot
- Clone this three repositories to same directory.

Pangolin

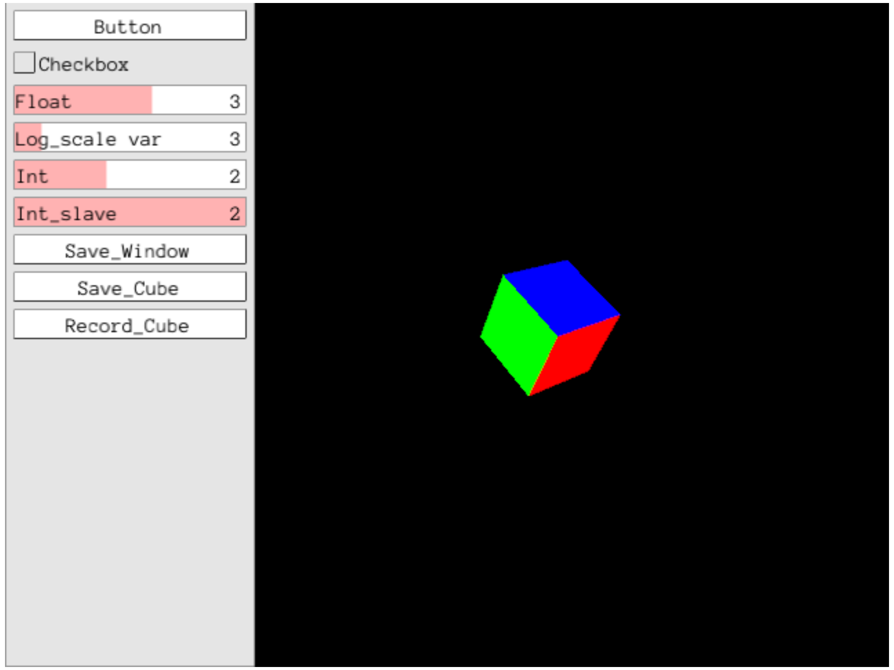
- Pangolin is a light weight 3d visualize tool for camera trajectory and point cloud. Most open source SLAM project utilize Pangolin such as ORB-SLAM, LSD-SLAM, DSO.

- Dependency

```
sudo apt install libgl1-mesa-dev
sudo apt install libglew-dev
sudo apt install cmake
sudo apt-get install libxkbcommon-x11-dev
```

- Build Pangolin

```
mkdir build
cd build
cmake ..
cmake --build .
```



ORB-SLAM2

- ORB-SLAM2 is the second version of ORB-SLAM algorithm, which provides monocular, stereo and RGBD mode.

1. Before building ORB-SLAM2, we first apply the git patch in python binding project to modify the CMake setting.

```
(In ORB-SLAM2 folder)
git apply [PATH_TO_PYTHON_BINDING]/orbslam-changes.diff
```

2. Add the include library `#include <unistd.h>` to `ORB_SLAM2/include/System.h` .

ORB-SLAM2

3. Modify the file authority and build ORB-SLAM:

```
chmod +x build.sh
./build.sh
```

4. Install the library:

```
sudo make install
```

ORB-SLAM Python Binding

- This repository utilize boost the bind the C++ library of ORB-SLAM with python.

1. The python version in jetbot is 3.6, and this repository is for python 3.5. We first modify the python setting in `ORB-SLAM-PythonBinding/CMakeList.txt` .

```
31 find_package(PythonLibs 3.5 REQUIRED) -> 3.6
33 find_package(Boost 1.45.0 REQUIRED COMPONENTS python-py35) -> 36
72 install(TARGETS ${TARGET_MODULE_NAME} DESTINATION lib/python3.5/dist-packages) -> 3.6
```

2. Fix the bug of get_tracked_mappoints()

```
[ In ORB_SLAM2-PythonBinding/src/ORBSlamPython.h Line:34]
boost::python::dict getTrackedMappoints() const;
```

ORB-SLAM Python Binding

```
[ In ORB_SLAM2-PythonBinding/src/ORBSlamPython.cpp Line:311]
boost::python::dict ORBSlamPython::getTrackedMappoints() const
{
    if (!system)
        return boost::python::dict();

    vector<ORB_SLAM2::MapPoint*> Mps = system->GetTrackedMapPoints();
    boost::python::dict map_points;
    for(size_t i=0; i<Mps.size(); i++)    {
        if(Mps[i] != NULL){    // There were several NULL map points.
            cv::Mat wp = Mps[i]->GetWorldPos();
            long unsigned int mid = Mps[i]-> mnId; // Record ids of map points.
            map_points[int(mid)] = boost::python::make_tuple(
                wp.at<float>(0,0),
                wp.at<float>(1,0),
                wp.at<float>(2,0));
        }
    }

    return map_points;
}
```

ORB-SLAM Python Binding

3. Build dependency and python binding :

```
sudo apt-get install libboost-all-dev
mkdir build
cd build
cmake ..
make
make install
```

4. Verify the installation.

```
python3
>> import orbslam2
```

3. If the error of missing .so file occurs, copy the library file:

```
sudo cp /usr/local/lib/libORB_SLAM2.so /usr/lib/libORB_SLAM2.so
sudo cp /usr/local/lib/libg2o.so /usr/lib/libg2o.so
sudo cp /usr/local/lib/libDBoW2.so /usr/lib/libDBoW2.so
```

ORB-SLAM Program Flow

- The usage of ORB-SLAM in python is simple:

```
import orbslam2
# Create System (Need to load BoW file, take about 50 seconds.)
slam = orbslam2.System(vocab_path, calibration_path, orbslam2.Sensor.MONOCULAR)
# Set Pangolin Visibility (Affect the Running Speed)
slam.set_use_viewer(False)
# Initialize System
slam.initialize()

while(True):
    # Feed the Image
    slam.process_image_mono(image, timestemp)
    # Get System State (NOT_INITIALIZE / OK / LOST)
    track_state = slam.get_tracking_state()
    # Get Trajectory (list of 3x4 transform matrix)
    trajectory = slam.get_trajectory_points()
    # Get Current Tracked Map Points (Dictionary {id:(x,y,z),…})
    tracked_points = slam.get_tracked_mappoints()
```

Run ORB-SLAM

- Python Example Code: `ORB_SLAM2-Python-Jetbot/orbslam_mono_tum.py`

```
python3 orbslam_mono_tum.py vocab_path calibration_path dataset_path
```

vocab_path: BoW Vocabulary file, can be found in ORB_SLAM2/Vocabulary

calibration_path: Camera Parameter file, Can be found in ORB-SLAM2/Examples/Monocular/*.yaml

- Datasets: https://vision.in.tum.de/data/datasets/rgbd-dataset/download
  - The number X in dataset of "fr**X**_YYY" or "freiburg**X**_YYY" is corresponding to the camera calibration file "TUM**X**.yaml".

Run ORB-SLAM Remotely

- The jupyter code `ORB_SLAM2-Python-Jetbot/orbslam_dataset_demo.ipynb` can operate remotely, but you have to install the extension to visualize the 3d point cloud and pose.
- Install Plotly Visualization Library.
  - Python Install `pip3 install plotly`
  - Install jupyter extension and rebuild jupyter.

    ```
    sudo jupyter labextension install @jupyterlab/plotly-extension
    jupyter lab build
    ```

  - Reboot your jetbot.
- The program will generate the trajectory file (dataset_poses.npz) and map points file (dataset_map_points.json).

Homework 2

Camera Calibration and JetBot Live Camera Testing

Camera Calibration

- To apply the ORB-SLAM on our kawaii jetbot, we have to calibrate the camera parameters including focal length, center position and distortion.

$$\text{Distortion}_{\text{coefficients}} = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3)$$

$$x_{\text{corrected}} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{\text{corrected}} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$x_{\text{corrected}} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{\text{corrected}} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

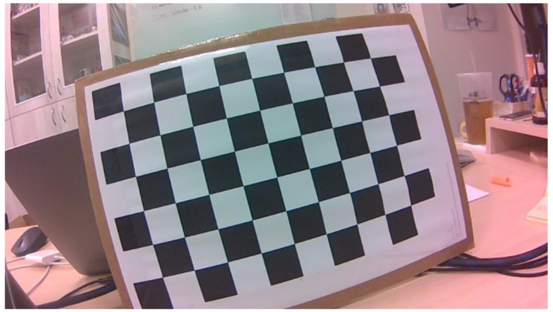
$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

- In ORB_SLAM-Python-Jetbot/Calibration/, I provide both the python code and jupyter code step-by-steps.

Camera Calibration – Data Collection

- Print the chessboard pattern downloaded from https://github.com/opencv/opencv/blob/master/doc/pattern.png
- Run "01_collect_data.py" or "01_collect_data.ipynb" to collect the images of chessboard in different perspective. The program will create the "Image" folder to store the images.

[5]:
```
camera = Camera.instance(width=960, height=540, capture_width=1280, capture_height=720)
image = widgets.Image(format='jpeg', width=480, height=270)  # this width and height doesn't necessarily have to match the camera
camera_link = traitlets.dlink((camera, 'value'), (image, 'value'), transform=bgr8_to_jpeg)
display(image)
```



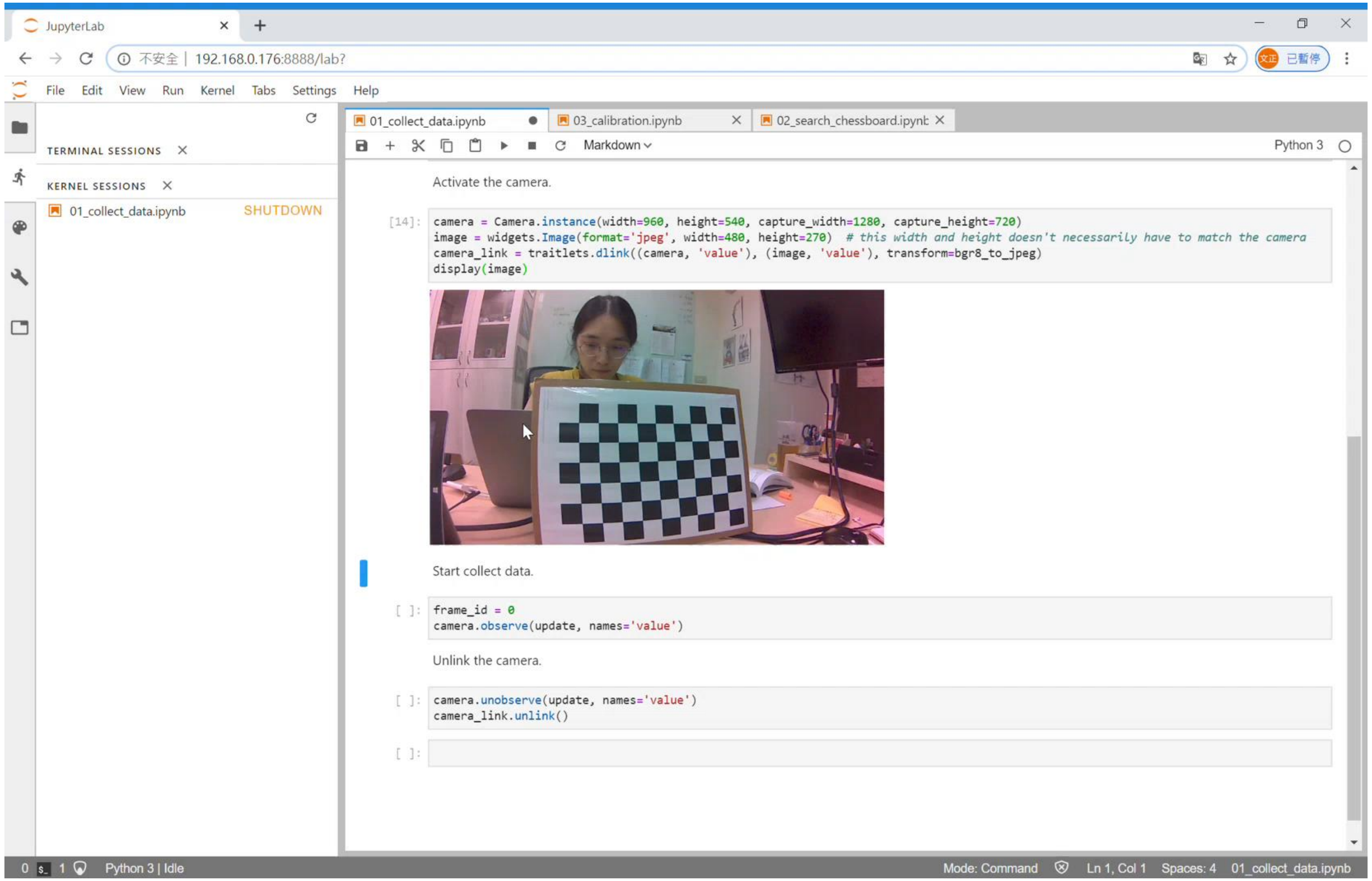
Start collect data.

[6]:
```
frame_id = 0
camera.observe(update, names='value')
```

```
save data 0056.jpg
```

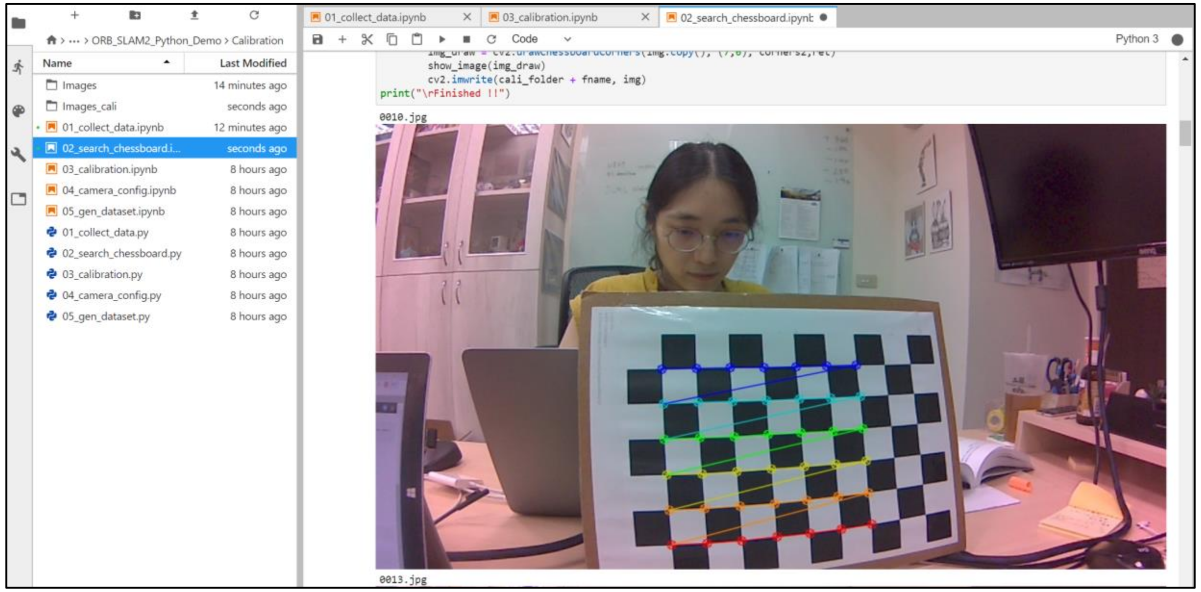
Hint: Make sure the whole chessboard is in the view. Avoiding too much out of plane rotation.

Camera Calibration – Data Collection



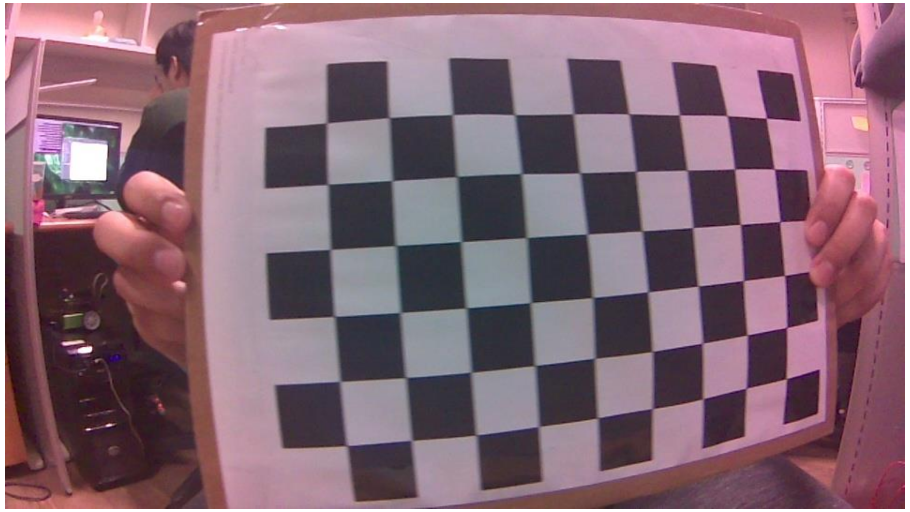
Camera Calibration – Search Chessboard

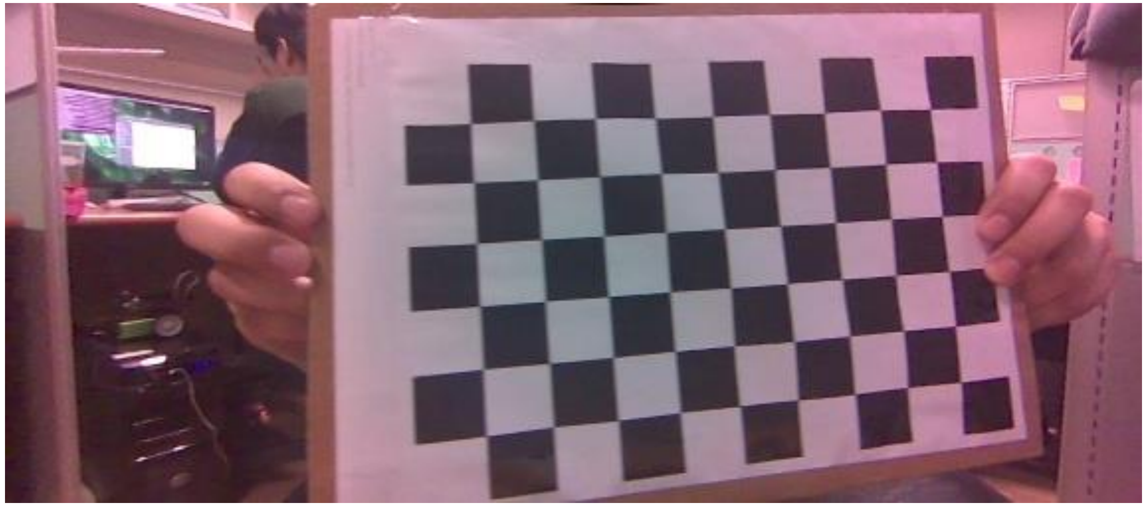
- Run "02_search_chessboard.py" or "02_search_chessboard.ipynb".
- We use the OpenCV function to automatically capture the chessboard. The images having detected the chessboard will be saved to "Images_cali/".
- Delete the blurry images, images with wrongly captured chessboard points, and similar images. Remain about 30 images.



Camera Calibration - Calibration

- Run "03_calibration.py" or "03_calibration.ipynb".
- Compute the optimizer camera parameter and generate the undistort image into the folder "Images_undist/". The parameters will be saved to "camera.npz".
- Check the ROI and undistort images to verify the correctness of camera parameters.





Camera Calibration – Camera Config

- Run “04_camnera_config.py” or “04_camera_config.ipynb”.
- Read ”camera.npz” and print the information with yaml format. Copy a yaml file from ORB-SLAM and overwrite the camera parameters of jetbot.

```
[[565.60419994   0.         488.81363132]
 [  0.         565.38709743 277.31877004]
 [  0.           0.           1.        ]] [[-3.45642791e-01  2.01114552e-01  4.24464846e-05  3.23563219e-03
  -8.42496324e-02]]
==============================
Camera.fx: 565.60419994
Camera.fy: 565.38709743
Camera.cx: 488.81363132
Camera.cy: 277.31877004

Camera.k1: -0.34564279
Camera.k2: 0.20111455
Camera.p1: 0.00004245
Camera.p2: 0.00323563
Camera.k3: -0.08424963
```

Copy the above information to ORB-SLAM's yaml file.

Generate TUM-like Dataset

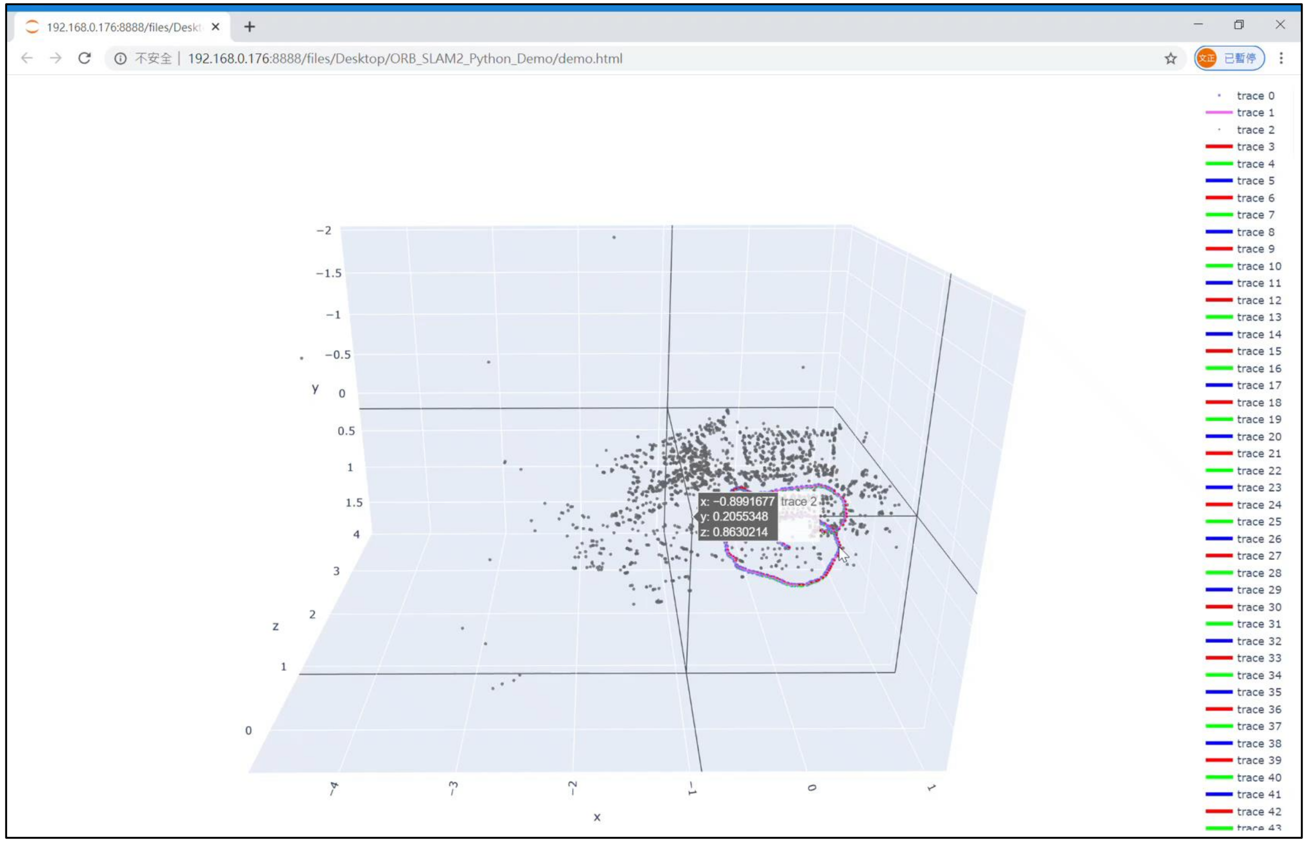
- Run "05_gen_dataset.py" or "05_gen_dataset.ipynb".
- Similar to collect data, the difference is this program will generate the a "rgb.txt" file such as TUM dataset.

```
jetbot_dataset_960/
    - rgb.txt
    - rgb/
        - 0000.jpg
        - 0001.jpg
```

```
1 rgb/0001.jpg
2 rgb/0002.jpg
3 rgb/0003.jpg
4 rgb/0004.jpg
5 rgb/0005.jpg
6 rgb/0006.jpg
7 rgb/0007.jpg
8 rgb/0008.jpg
9 rgb/0009.jpg
10 rgb/0010.jpg
11 rgb/0011.jpg
12 rgb/0012.jpg
13 rgb/0013.jpg
```

ORB-SLAM with Jetbot Live Camera

- Run "orbslam_jetbot_live.py" or "orbslam_jetbot_camera.ipynb".



Homework 2

- **Due: 5/21 10:00 pm** (Late submission is not allowed)
- Requirement
  - Images for calibration (About 30 images in "Images_cali/") (20%)
  - The camera calibration file (XXX.yaml) (30%)
  - The reconstruct map and camera trajectory (XXX.html) (50%)

Build the map of your scene and the trajectory of your camera.
Add `plot_figure.write_html("demo.html")` at the bottom of
"orbslam_jetbot_camera.ipynb" to generate the interactive HTML.