University of South Florida
Control of Mobile Robots

Prof. Alfredo Weitzenfeld
Running ORB_SLAM2

Running the Programs

Step 1. Running camera node

An initial set up of ***ROS*** to allow communication between your computer and the robot as detailed in the documentation for installing the ***robot_client*** and ***ros_controller*** is necessary as well. In addition to running the primary launch file on the robot, you will want to run an instance of ***raspicam***(included in ***ros_kinetic***) in a separate terminal. In my testing, I had run into issues where the camera had to be restarted in order for ***ORB_SLAM2*** to receive the images for processing. The settings I had most success with were:

rosrun raspicam_node raspicam_node _width:=640 _height:=480 _framerate:=30 _quality:=100 _ISO:=200 _shutter_speed:=100000 _saturation:=50 _awb_mode:=horizon __name:=cam _enable_raw:=true _hFlip:=true _vFlip:=true

These are the parameters that I had found and modified from Pablo Scleidorovich's ***main_controller.py*** program in the ***ros_controller*** branch on the ***biorobaw*** github[1].

The camera appears to be mounted upside-down on my robot, so I had to flip the image. However, if yours is properly oriented you may set the ***hFlip*** and ***vFlip*** parameters to false.

Additionally, the **640x480** resolution worked best for me, however, you may wish to experiment with **320x240** or **1280x960**. Higher resolution images seem to work better with

[1] https://github.com/biorobaw/pi3_robot_2019/blob/ros_controller/scripts/main_controller.py

ORB_SLAM2 for generating the points in a map, but may slow down the rate of camera frames being received through the network, hindering the robot's ability to maintain localization while moving, as the time between two image frames are too significant. Whereas, lower resolutions have smoother frame rate, but seems to take more time and generate less accurate maps.

Step 2. Launching ORB_SLAM2

In the ***robot_client***/***launch*** folder, there exists two launch files:

pi3_orb_slam2.launch

pi3_orb_slam2_localization_only.launch

These launch files will launch ***ORB_SLAM2*** with the settings configured to work with the robot. You would run it with the command:

roslaunch robot_client pi3_orb_slam2.launch

Recommended: The camera should be calibrated as detailed here.[2]

After running the calibration program you can update the camera calibration parameters in the launch files:

```
<!-- Camera calibration parameters -->

    <!--If the node should wait for a camera_info topic to take the camera calibration data-->
    <param name="load_calibration_from_cam" type="bool" value="false" />
    <!-- Camera calibration and distortion parameters (OpenCV) -->
   <param name="camera_fx" type="double" value="583.01740710559432" />
   <param name="camera_fy" type="double" value="583.01740710559432" />
   <param name="camera_cx" type="double" value="320" />
   <param name="camera_cy" type="double" value="240" />
    <!-- Camera calibration and distortion parameters (OpenCV) -->
   <param name="camera_k1" type="double" value="0.20483552665926258" />
   <param name="camera_k2" type="double" value="-0.42414204428032326" />
   <param name="camera_p1" type="double" value="0.0" />
   <param name="camera_p2" type="double" value="0.0" />
   <param name="camera_k3" type="double" value=".0095177708663656737" />
```

[2] https://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html

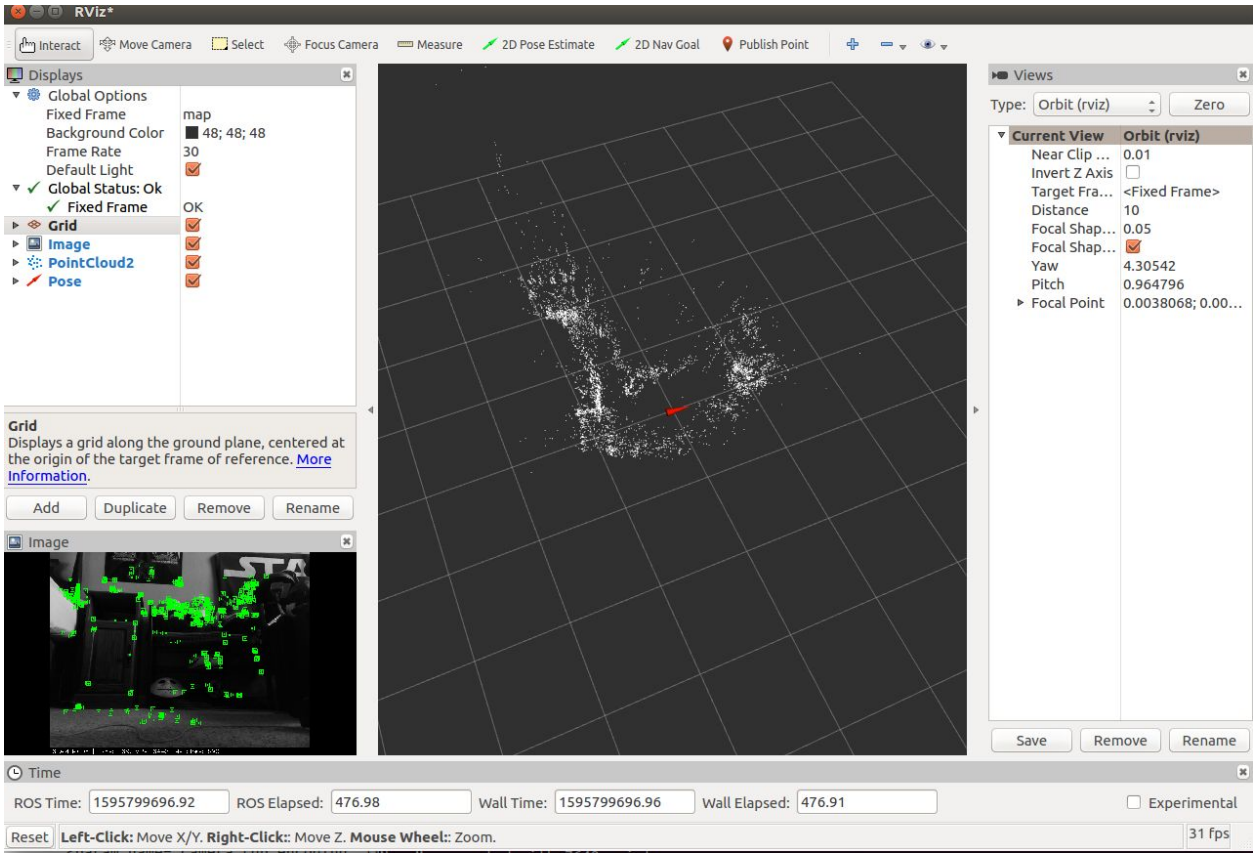
If the RoboBulls robot you have is equipped with the same camera as used in this paper, you may not need to update the parameters. However, if you are having problems with ***ORB_SLAM2*** you may need to update these parameters.

Step 3. Running Rviz

In order to view the map generated by ***ORB_SLAM2*** and the robot's position in the map, you will need to run rviz:

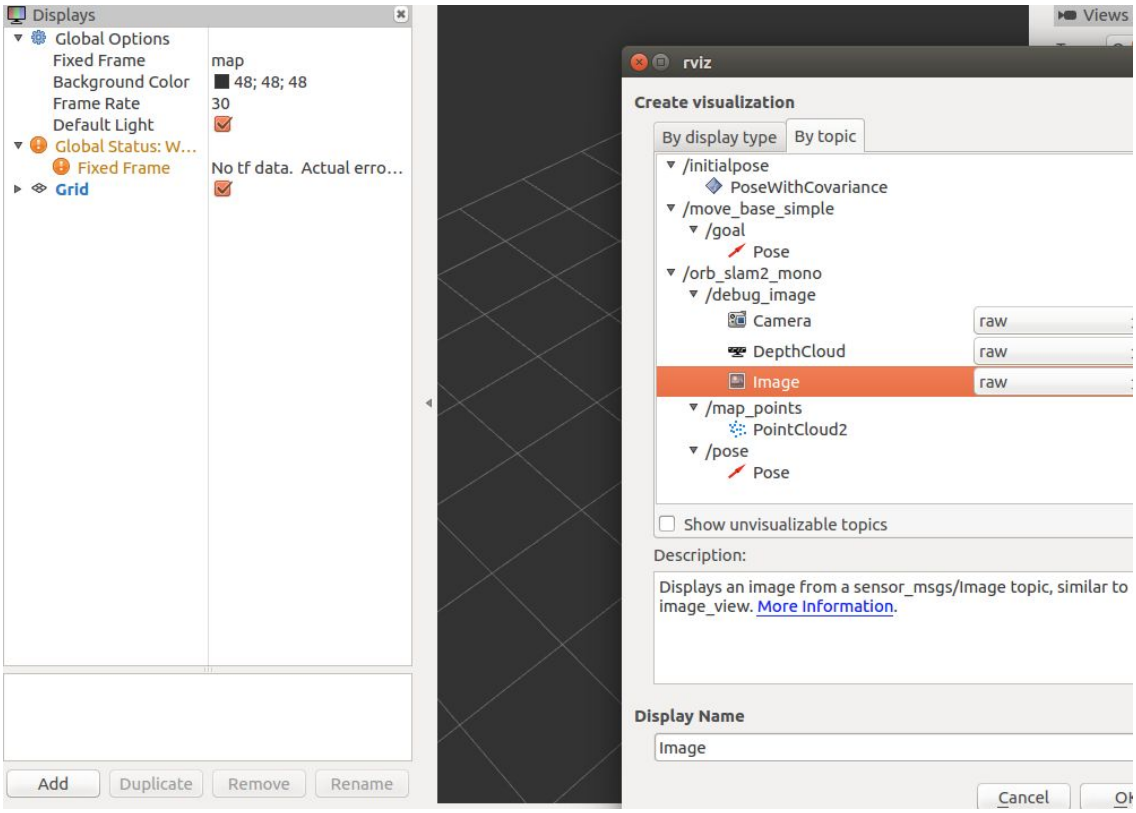
rosrun rviz rviz

Rviz fully configured and showing a map of a bedroom



You will want to add the ***debug_image***(Image), ***map_points***(PointCloud2), and ***pose***(Pose) topics that are located in /orb_slam2_mono.

Setting Rviz parameters

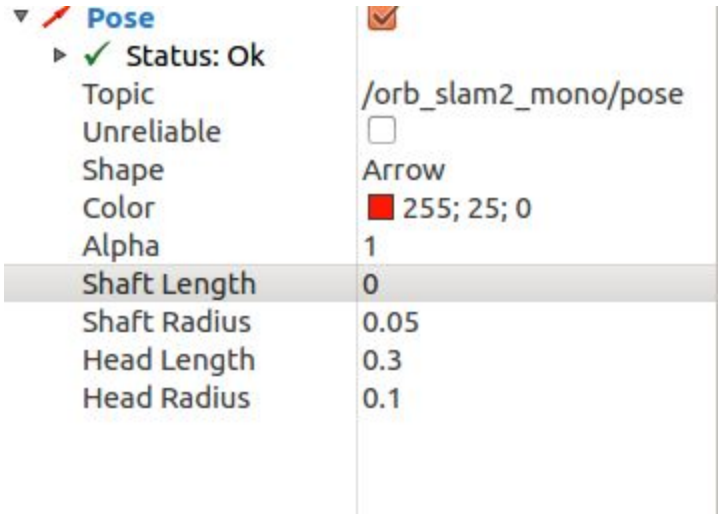


The ***debug_image*** topic allows you to view ***ORB_SLAM2*** generating points from the image frames as well as whether the robot is currently localized or lost.

The ***map_points*** topic shows you the map points in a three dimensional space, with ***pose*** showing the location of the robot. ***Rviz*** generates a model to represent the robot using the robot's

pose in the environment. I recommend that you set the ***Shaft Length*** of the pose display to 0 for a more accurate visual representation of the robot in the environment, as the robot does not have a significant body length that needs to be represented by the model.

Rviz pose parameters



You can move the camera around by clicking and scrolling on the 3D environment, while holding shift will allow you to change the point the camera is centered around. You may save these settings in Rviz so that you do not need to reapply the changes upon start up.

Step 4. Map generation using telop

Now that you have ***ORB_SLAM2*** generating a map from the video, the next step is generating the map. In order to do this, you will want to manually control the robot using ***telop***.

rosrun teleop_twist_keyboard teleop_twist_keyboard.py cmd_vel:=/pi3_robot_2019/r1/speed_vw

Depending on the complexity of the environment the robot is in, generating a map will take a varied amount of time and may involve different methods of navigating within it. For instance, generating points when the robot is facing a white wall will not work very well, and

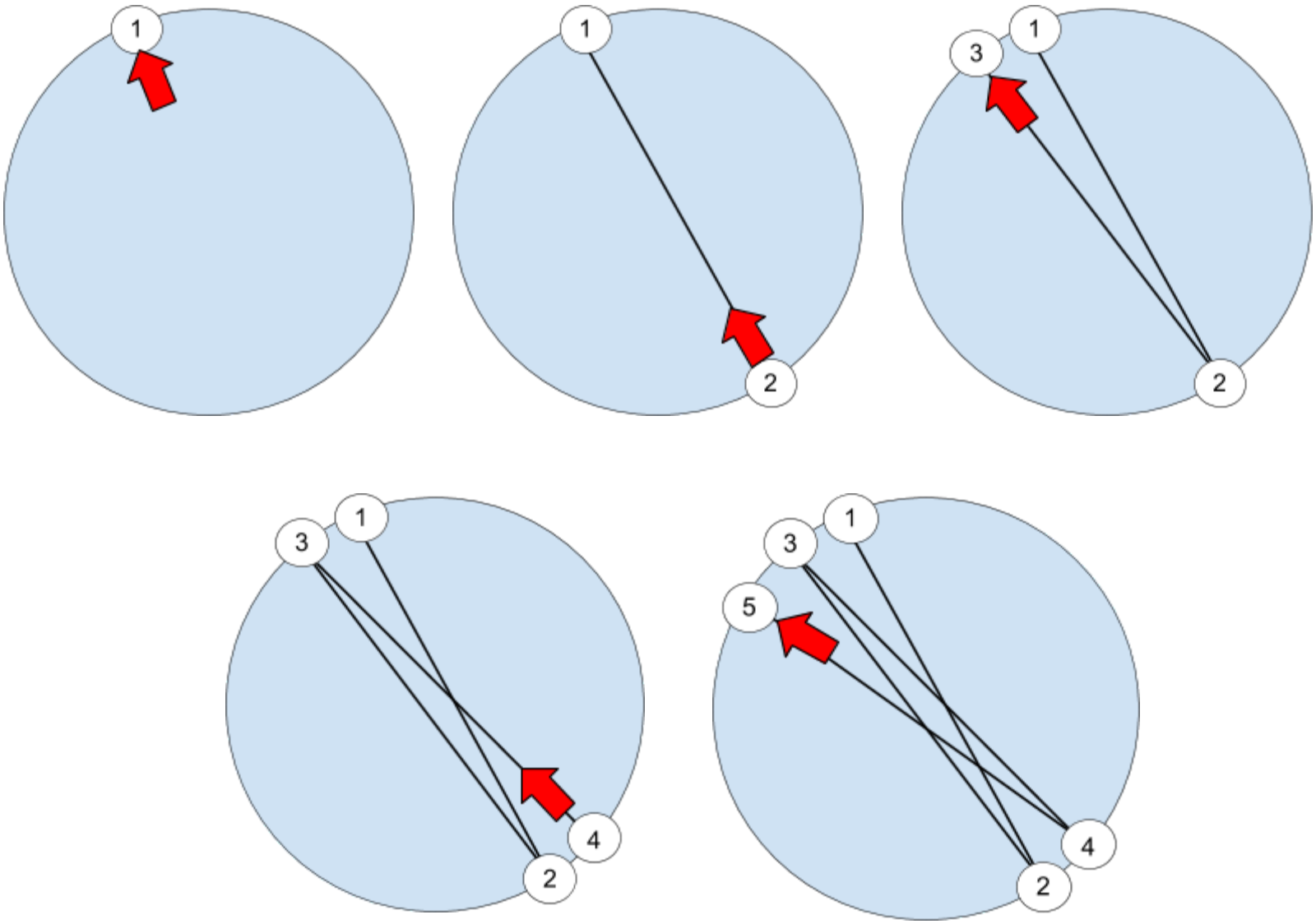
will require you to position the robot in a way that there are some distinguishable features in the captured image so that points can be generated. On the other hand, a cluttered room with colorful tapestry may generate a lot of points that are not accurately located. Keep in mind that this will play a part in localization as well, as if the robot is not able to recognize points, it will not know where it is located.

In order to test the system, I conducted tests in four different environments; a kitchen, two living areas, and a bedroom. I used methods of wall-following during map generation, having the robot rotate in the middle of the room during map generation, and manually navigating the robot throughout the environment.

The methodology I used for manually navigating the robot in the environment while creating maps is as followed; positioning the robot in one corner or side of the room. You will want to place the robot (camera) facing the corner and navigate the robot in reverse, away from the corner, until ***ORB_SLAM2*** picks up enough unique points to start initializing the map. Then you will want to start a process of moving the robot forwards and backwards the full length of the room, while slowly adjusting the angle of the robot, until the robot fully turns and captures 360 degrees. During this process, the robot will likely lose its location, so you will need to reposition the robot towards objects that it has already seen. After traveling a full circle, if the generated map appears to be mostly accurate, you will want to then manually navigate the robot throughout the environment, mapping all of the objects individually, to ensure that the finer details of the map are captured. You should test that the map is complete by navigating throughout the entirety of the map and ensuring that the robot stays localized. If the robot loses

localization, you should repeat the motion as detailed above until the robot can stay localized at that point.

Diagram showing the motion of the mapping methodology



Additionally, maps can be saved using:

```
rosservice call /orb_slam2_mono/save_map <name>.bin
```

Maps can be opened for localization only(where the map is no longer being built and only used for localization) or a map can continue being built. Modify ***pi3_orb_slam2_localization_only.launch*** to contain the name of the map you wish to save.

<!-- static parameters -->

<param name="load_map" type="bool" value="true" />

<param name="map_file" type="string" value="<name>.bin" /> ***//enter the name of the map file***

Additionally if you want to continue building a map, modify the same parameters in the ***pi3_orb_slam2.launch*** file

<!-- static parameters -->

<param name="load_map" type="bool" value="false" /> ***//set this to true***

<param name="map_file" type="string" value="<name>.bin" />

Step 5. Running bug algorithm

Once you have created a map that is detailed enough that the robot can maintain localization throughout the entirety of the environment, you can now run the bug algorithm

rosrun robot_client pi3_slam_nav.py

The GUI will display the robot's position (x, y, theta) in the map, and allow you to enter the (x, y) coordinates of the goal position. You will want to use Rviz to view the map and determine your goal point because, as aforementioned, the coordinates of the Pose from ***ORB_SLAM2*** are not to true scale, and are not represented by inches or meters. ***ORB_SLAM2*** appears to approximate the depth of the image differently each time the map is initialized, so there does not seem to be a way to convert these coordinates to a unit of measurement.

pi3_slam_nav.py GUI