Slide

https://kasets.art/Ehfvji

SLAM workshop

Noppanut Thongton

SLAM with Turtlebot3

Using simulation of Turtlebot3 in Gazebo with SLAM.

Turtlebot3 - SLAM

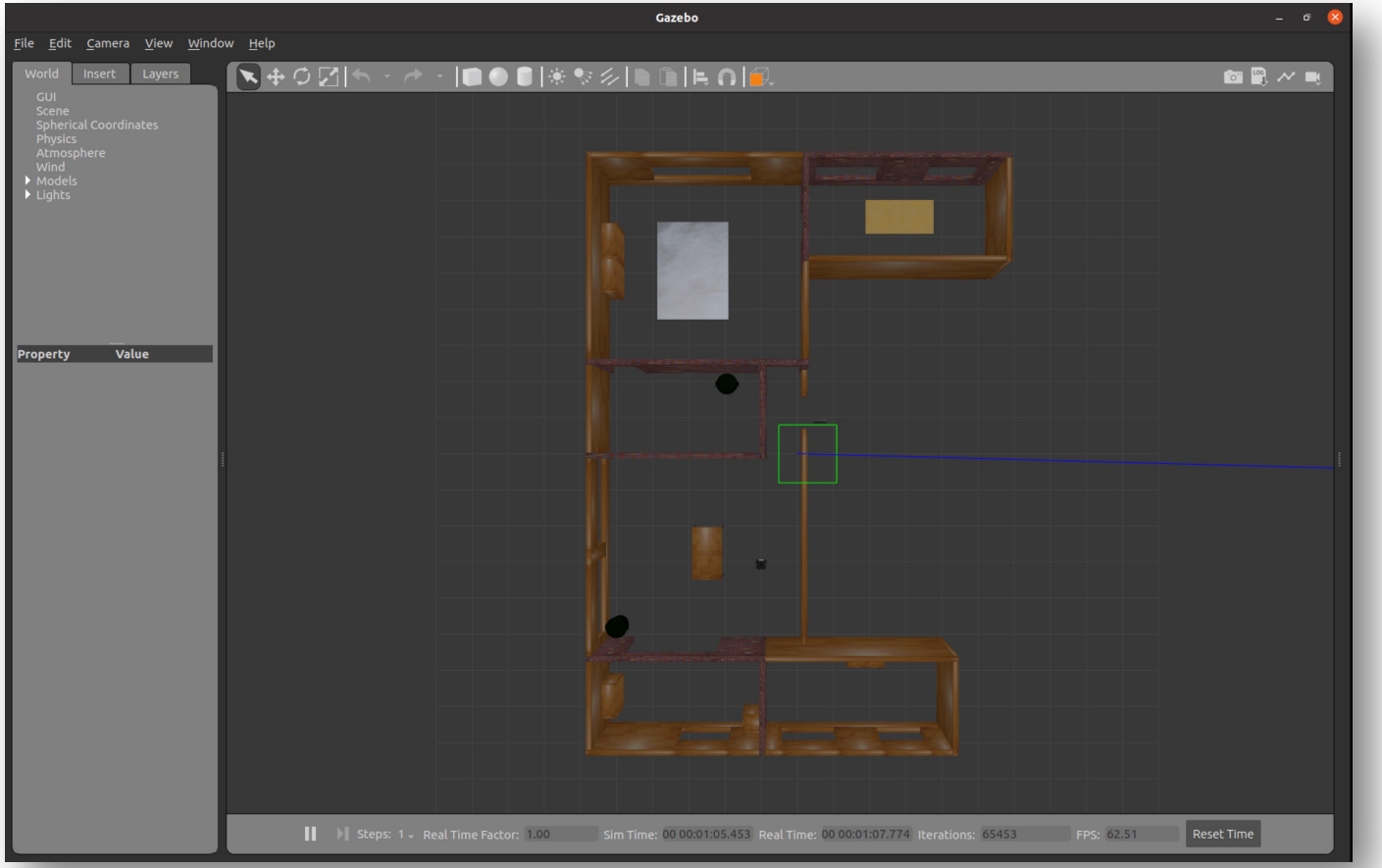
```
$ sudo apt install ros-noetic-turtlebot3*
```

Turtlebot3 - SLAM

```
$ export TURTLEBOT3_MODEL=burger
```

```
$ roslaunch turtlebot3_gazebo turtlebot3_house.launch
```

Turtlebot3 - SLAM



Turtlebot3 - SLAM

```
$ sudo apt install ros-noetic-gmapping
```

Turtlebot3 - SLAM

```
<launch>
  <!-- Arguments -->
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="slam_methods" default="gmapping" doc="slam type [gmapping, cartographer, hector, karto, frontier_exploration]"/>
  <arg name="configuration_basename" default="turtlebot3_lds_2d.lua"/>
  <arg name="open_rviz" default="true"/>

  <!-- TurtleBot3 -->
  <include file="$(find turtlebot3_bringup)/launch/turtlebot3_remote.launch">
    <arg name="model" value="$(arg model)" />
  </include>

  <!-- SLAM: Gmapping, Cartographer, Hector, Karto, Frontier_exploration, RTAB-Map -->
  <include file="$(find turtlebot3_slam)/launch/turtlebot3_$(arg slam_methods).launch">
    <arg name="model" value="$(arg model)"/>
    <arg name="configuration_basename" value="$(arg configuration_basename)"/>
  </include>

  <!-- rviz -->
  <group if="$(arg open_rviz)">
    <node pkg="rviz" type="rviz" name="rviz" required="true"
          args="-d $(find turtlebot3_slam)/rviz/turtlebot3_$(arg slam_methods).rviz"/>
  </group>
</launch>
```

Turtlebot3 - SLAM

```xml
<launch>
  <!-- Arguments -->
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="slam_methods" default="gmapping" doc="slam type [gmapping, cartographer, hector, karto, frontier_exploration]"/>
  <arg name="configuration_basename" default="turtlebot3_lds_2d.lua"/>
  <arg name="open_rviz" default="true"/>

  <!-- TurtleBot3 -->
  <include file="$(find turtlebot3_bringup)/launch/turtlebot3_remote.launch">
    <arg name="model" value="$(arg model)" />
  </include>

  <!-- SLAM: Gmapping, Cartographer, Hector, Karto, Frontier_exploration, RTAB-Map -->
  <include file="$(find turtlebot3_slam)/launch/turtlebot3_$(arg slam_methods).launch">
    <arg name="model" value="$(arg model)"/>
    <arg name="configuration_basename" value="$(arg configuration_basename)"/>
  </include>

  <!-- rviz -->
  <group if="$(arg open_rviz)">
    <node pkg="rviz" type="rviz" name="rviz" required="true"
          args="-d $(find turtlebot3_slam)/rviz/turtlebot3_$(arg slam_methods).rviz"/>
  </group>
</launch>
```

Turtlebot3 - SLAM

```
<launch>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="multi_robot_name" default=""/>

  <include file="$(find turtlebot3_bringup)/launch/includes/description.launch.xml">
    <arg name="model" value="$(arg model)" />
  </include>

  <node pkg="robot_state_publisher" type="robot_state_publisher" name="robot_state_publisher">
    <param name="publish_frequency" type="double" value="50.0" />
    <param name="tf_prefix" value="$(arg multi_robot_name)"/>
  </node>
</launch>
```

Turtlebot3 - SLAM

```
<launch>
  <arg name="model"/>
  <arg name="urdf_file" default="$(find xacro)/xacro --inorder '$(find turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro'" />
  <param name="robot_description" command="$(arg urdf_file)" />
</launch>
```

Turtlebot3 - SLAM

```
<launch>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="multi_robot_name" default=""/>

  <include file="$(find turtlebot3_bringup)/launch/includes/description.launch.xml">
    <arg name="model" value="$(arg model)" />
  </include>

  <node pkg="robot_state_publisher" type="robot_state_publisher" name="robot_state_publisher">
    <param name="publish_frequency" type="double" value="50.0" />
    <param name="tf_prefix" value="$(arg multi_robot_name)"/>
  </node>
</launch>
```

Turtlebot3 - SLAM

```
<launch>
  <!-- Arguments -->
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="slam_methods" default="gmapping" doc="slam type [gmapping, cartographer, hector, karto, frontier_exploration]"/>
  <arg name="configuration_basename" default="turtlebot3_lds_2d.lua"/>
  <arg name="open_rviz" default="true"/>

  <!-- TurtleBot3 -->
  <include file="$(find turtlebot3_bringup)/launch/turtlebot3_remote.launch">
    <arg name="model" value="$(arg model)" />
  </include>

  <!-- SLAM: Gmapping, Cartographer, Hector, Karto, Frontier_exploration, RTAB-Map -->
  <include file="$(find turtlebot3_slam)/launch/turtlebot3_$(arg slam_methods).launch">
    <arg name="model" value="$(arg model)"/>
    <arg name="configuration_basename" value="$(arg configuration_basename)"/>
  </include>

  <!-- rviz -->
  <group if="$(arg open_rviz)">
    <node pkg="rviz" type="rviz" name="rviz" required="true"
          args="-d $(find turtlebot3_slam)/rviz/turtlebot3_$(arg slam_methods).rviz"/>
  </group>
</launch>
```

Turtlebot3 - SLAM

```
<launch>
  <!-- Arguments -->
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="configuration_basename" default="turtlebot3_lds_2d.lua"/>
  <arg name="set_base_frame" default="base_footprint"/>
  <arg name="set_odom_frame" default="odom"/>
  <arg name="set_map_frame"  default="map"/>

  <!-- Gmapping -->
  <node pkg="gmapping" type="slam_gmapping" name="turtlebot3_slam_gmapping" output="screen">
    <param name="base_frame" value="$(arg set_base_frame)"/>
    <param name="odom_frame" value="$(arg set_odom_frame)"/>
    <param name="map_frame"  value="$(arg set_map_frame)"/>
    <rosparam command="load" file="$(find turtlebot3_slam)/config/gmapping_params.yaml" />
  </node>
</launch>
```

Turtlebot3 - SLAM

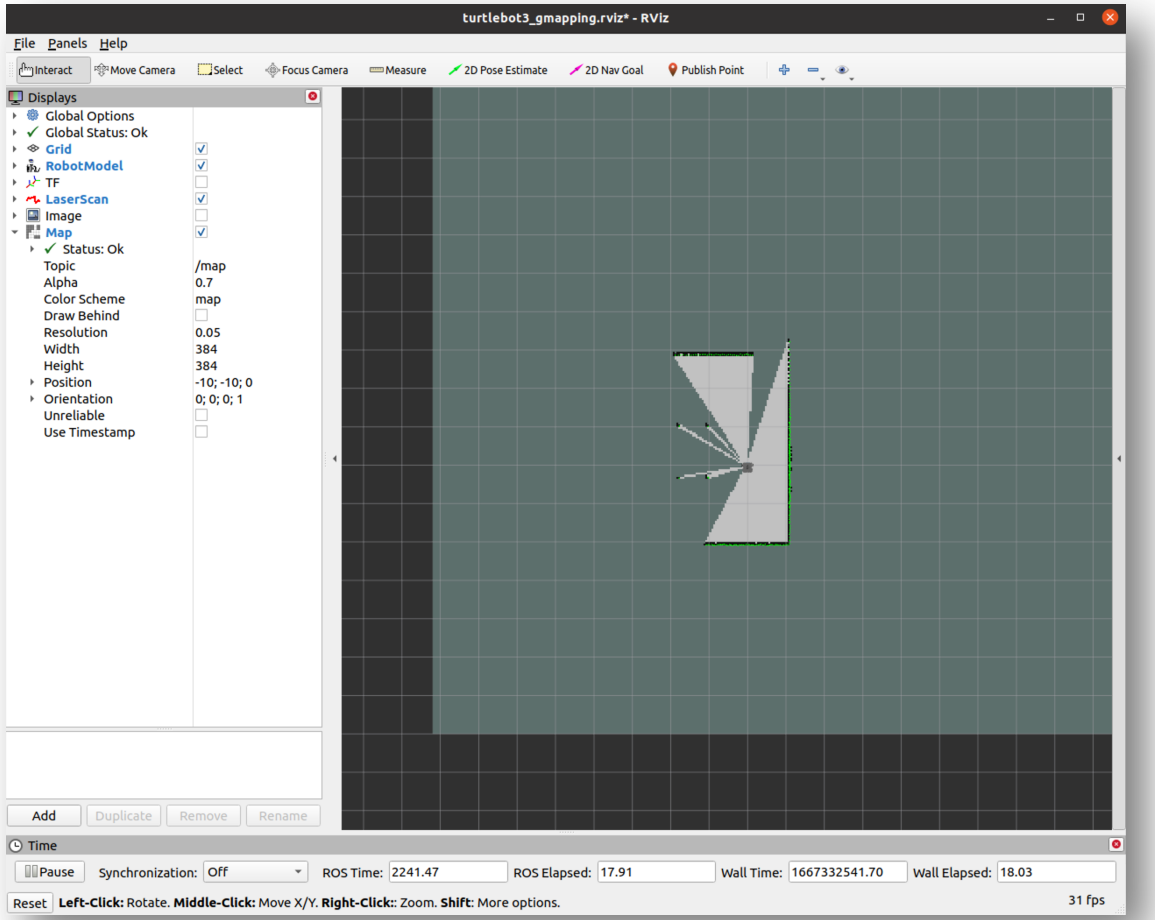
```
<launch>
  <!-- Arguments -->
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="slam_methods" default="gmapping" doc="slam type [gmapping, cartographer, hector, karto, frontier_exploration]"/>
  <arg name="configuration_basename" default="turtlebot3_lds_2d.lua"/>
  <arg name="open_rviz" default="true"/>

  <!-- TurtleBot3 -->
  <include file="$(find turtlebot3_bringup)/launch/turtlebot3_remote.launch">
    <arg name="model" value="$(arg model)" />
  </include>

  <!-- SLAM: Gmapping, Cartographer, Hector, Karto, Frontier_exploration, RTAB-Map -->
  <include file="$(find turtlebot3_slam)/launch/turtlebot3_$(arg slam_methods).launch">
    <arg name="model" value="$(arg model)"/>
    <arg name="configuration_basename" value="$(arg configuration_basename)"/>
  </include>

  <!-- rviz -->
  <group if="$(arg open_rviz)">
    <node pkg="rviz" type="rviz" name="rviz" required="true"
          args="-d $(find turtlebot3_slam)/rviz/turtlebot3_$(arg slam_methods).rviz"/>
  </group>
</launch>
```

Turtlebot3 - SLAM

```
$ export TURTLEBOT3_MODEL=burger
```

```
$ roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
```

Turtlebot3 - SLAM



Turtlebot3 - SLAM

```
$ export TURTLEBOT3_MODEL=burger
```

```
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Turtlebot3 - SLAM

```
pat@pat:~$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
... logging to /home/pat/.ros/log/04d8d5cc-5a19-11ed-8afc-53c03324dea8/roslaunch-pat-60445.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://pat:36361/

SUMMARY
========

PARAMETERS
 * /model: waffle
 * /rosdistro: noetic
 * /rosversion: 1.15.14

NODES
  /
    turtlebot3_teleop_keyboard (turtlebot3_teleop/turtlebot3_teleop_key)

ROS_MASTER_URI=http://localhost:11311

process[turtlebot3_teleop_keyboard-1]: started with pid [60483]

Control Your TurtleBot3!
---------------------------
Moving around:
        w
   a    s    d
        x

w/x : increase/decrease linear velocity (Burger : ~ 0.22, Waffle and Waffle Pi : ~ 0.26)
a/d : increase/decrease angular velocity (Burger : ~ 2.84, Waffle and Waffle Pi : ~ 1.82)

space key, s : force stop

CTRL-C to quit
```

Turtlebot3 - SLAM


```
$ cd ~/
```

```
$ rosrun map_server map_saver -f map
```

```
$ ls
```

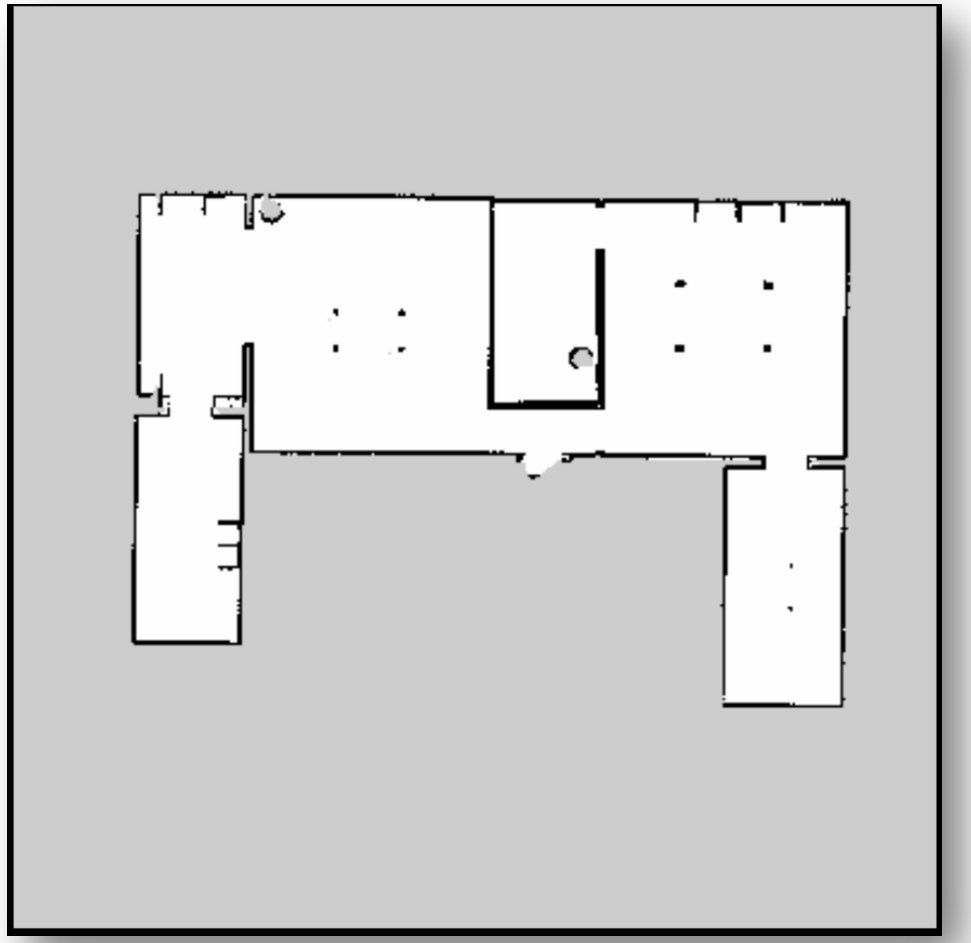
Turtlebot3 - SLAM

```
$ vim map.yaml
```

map.yaml

```
image: map.pgm
resolution: 0.050000
origin: [-10.000000, -10.000000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

Turtlebot3 - SLAM



map.pgm

Setup sensor

Setup the sensor to use with real robot.

Webcam

```
$ sudo apt install ros-noetic-usb-cam
```

Webcam

```
$ rospack profile
```

Webcam

```
$ roscore
```

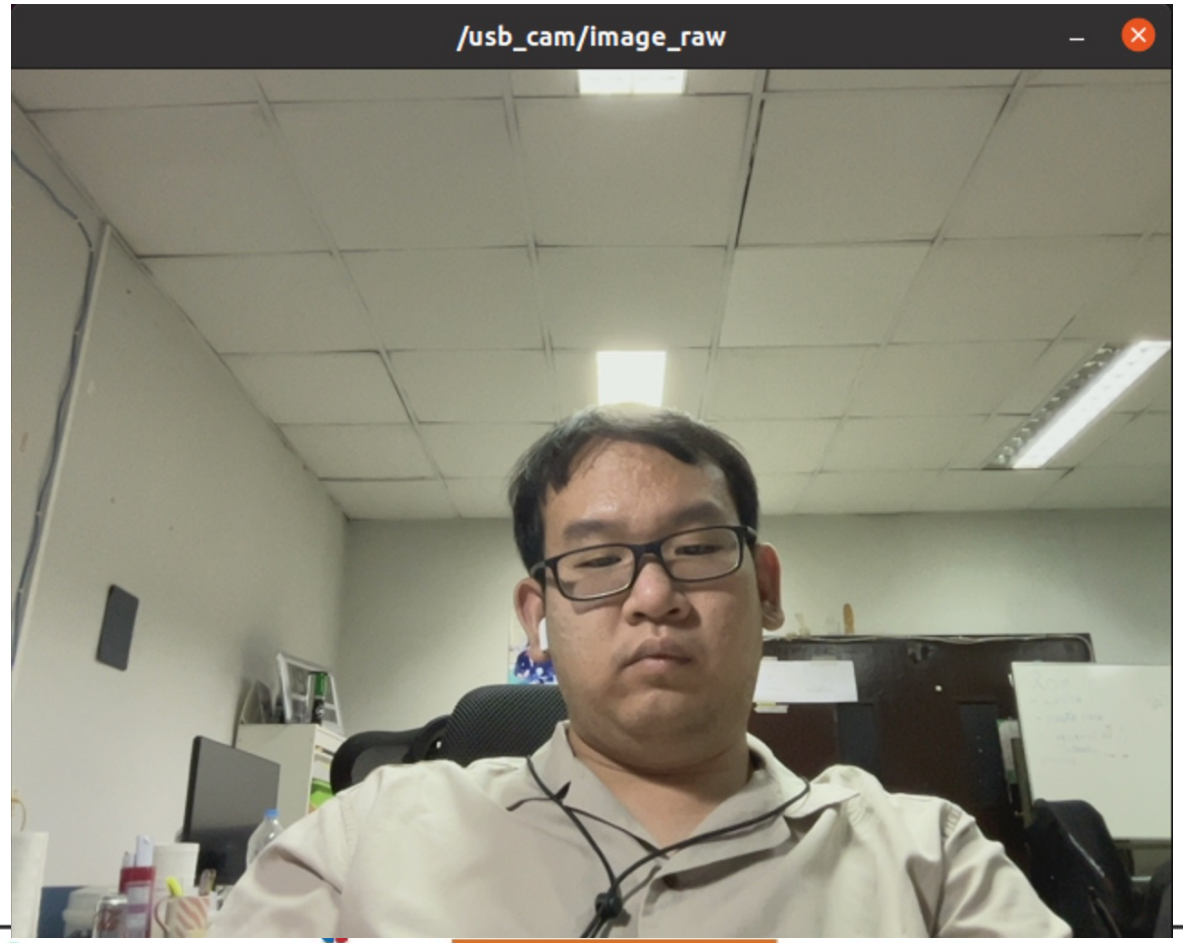
Webcam

Open new termimal

Webcam

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

Webcam



Webcam

Create launch file on your package

Make workspace

```
$ cd ~
```

```
$ mkdir -p your_ws/src
```

```
$ cd your_ws
```

```
$ catkin_make
```

Make workspace

```
$ echo "source
/home/$USER/your_ws/devel/
setup.bash" >> ~/.bashrc
```

Create package

```
$ cd ~/your_ws/src
```

```
$ catkin_create_pkg your_package roscpp rospy std_msgs
```

```
$ cd ~/your_ws
```

```
$ catkin_make
```

Test

```
$ roscd your_package
```

Webcam

```
$ roscd your_package
```

```
$ mkdir launch
```

```
$ cd launch
```

Webcam

```
$ gedit usb-cam.launch
```

Webcam

code

Webcam

```
<launch>
  <node name="usb_cam" pkg="usb_cam" type="usb_cam_node" output="screen" >
    <param name="video_device" value="/dev/video0" />
    <param name="image_width" value="1920" />
    <param name="image_height" value="1080" />
    <param name="pixel_format" value="yuyv" />
    <param name="camera_frame_id" value="usb_cam" />
    <param name="io_method" value="mmap"/>
  </node>
  <node name="image_view" pkg="image_view" type="image_view" respawn="false" output="screen">
    <remap from="image" to="/usb_cam/image_raw"/>
    <param name="autosize" value="true" />
  </node>
</launch>
```

Webcam

```
<launch>
  <node name="usb_cam" pkg="usb_cam" type="usb_cam_node" output="screen" >
    <param name="video_device" value="/dev/video0" />
    <param name="image_width" value="1920" />
    <param name="image_height" value="1080" />
    <param name="pixel_format" value="yuyv" />
    <param name="camera_frame_id" value="usb_cam" />
    <param name="io_method" value="mmap"/>
  </node>
  <node name="image_view" pkg="image_view" type="image_view" respawn="false" output="screen">
    <remap from="image" to="/usb_cam/image_raw"/>
    <param name="autosize" value="true" />
  </node>
</launch>
```

Webcam

```
$ ls /dev/ | grep video
```

```
pat@pat:~$ ls /dev/ | grep video
video0
video1
```

Webcam

```
<launch>
  <node name="usb_cam" pkg="usb_cam" type="usb_cam_node" output="screen" >
    <param name="video_device" value="/dev/video0" />
    <param name="image_width" value="1920" />
    <param name="image_height" value="1080" />
    <param name="pixel_format" value="yuyv" />
    <param name="camera_frame_id" value="usb_cam" />
    <param name="io_method" value="mmap"/>
  </node>
  <node name="image_view" pkg="image_view" type="image_view" respawn="false" output="screen">
    <remap from="image" to="/usb_cam/image_raw"/>
    <param name="autosize" value="true" />
  </node>
</launch>
```

Webcam

```
<launch>
  <node name="usb_cam" pkg="usb_cam" type="usb_cam_node" output="screen" >
    <param name="video_device" value="/dev/video0" />
    <param name="image_width" value="1920" />
    <param name="image_height" value="1080" />
    <param name="pixel_format" value="yuyv" />
    <param name="camera_frame_id" value="usb_cam" />
    <param name="io_method" value="mmap"/>
  </node>
  <node name="image_view" pkg="image_view" type="image_view" respawn="false" output="screen">
    <remap from="image" to="/usb_cam/image_raw"/>
    <param name="autosize" value="true" />
  </node>
</launch>
```

1080P

Webcam

```
<launch>
  <node name="usb_cam" pkg="usb_cam" type="usb_cam_node" output="screen" >
    <param name="video_device" value="/dev/video0" />
    <param name="image_width" value="1920" />
    <param name="image_height" value="1080" />
    <param name="pixel_format" value="yuyv" />
    <param name="camera_frame_id" value="usb_cam" />
    <param name="io_method" value="mmap"/>
  </node>
  <node name="image_view" pkg="image_view" type="image_view" respawn="false" output="screen">
    <remap from="image" to="/usb_cam/image_raw"/>
    <param name="autosize" value="true" />
  </node>
</launch>
```

Webcam

```
<launch>
  <node name="usb_cam" pkg="usb_cam" type="usb_cam_node" output="screen" >
    <param name="video_device" value="/dev/video0" />
    <param name="image_width" value="1920" />
    <param name="image_height" value="1080" />
    <param name="pixel_format" value="yuyv" />
    <param name="camera_frame_id" value="usb_cam" />
    <param name="io_method" value="mmap"/>
  </node>
  <node name="image_view" pkg="image_view" type="image_view" respawn="false" output="screen">
    <remap from="image" to="/usb_cam/image_raw"/>
    <param name="autosize" value="true" />
  </node>
</launch>
```

Webcam

```
1   <launch>
2     <node name="usb_cam" pkg="usb_cam" type="usb_cam_node" output="screen" >
3       <param name="video_device" value="/dev/video0" />
4       <param name="image_width" value="1920" />
5       <param name="image_height" value="1080" />
6       <param name="pixel_format" value="yuyv" />
7       <param name="camera_frame_id" value="usb_cam" />
8       <param name="io_method" value="mmap"/>
9     </node>
10    <!-- <node name="image_view" pkg="image_view" type="image_view" respawn="false" output="screen">
11      <remap from="image" to="/usb_cam/image_raw"/>
12      <param name="autosize" value="true" />
13    </node> -->
14  </launch>
```

Comment

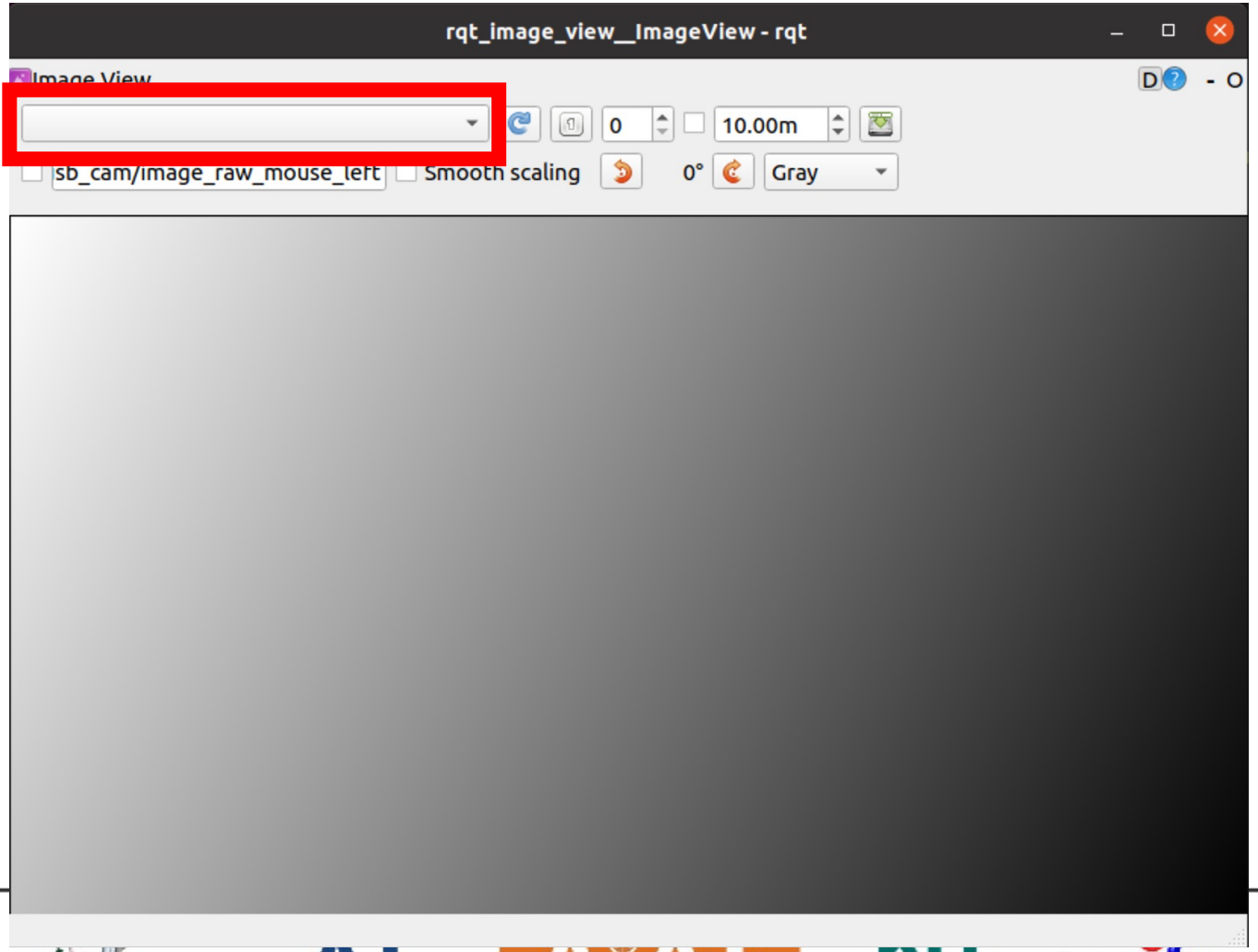
Webcam

```
$ roslaunch your_package usb-cam.launch
```

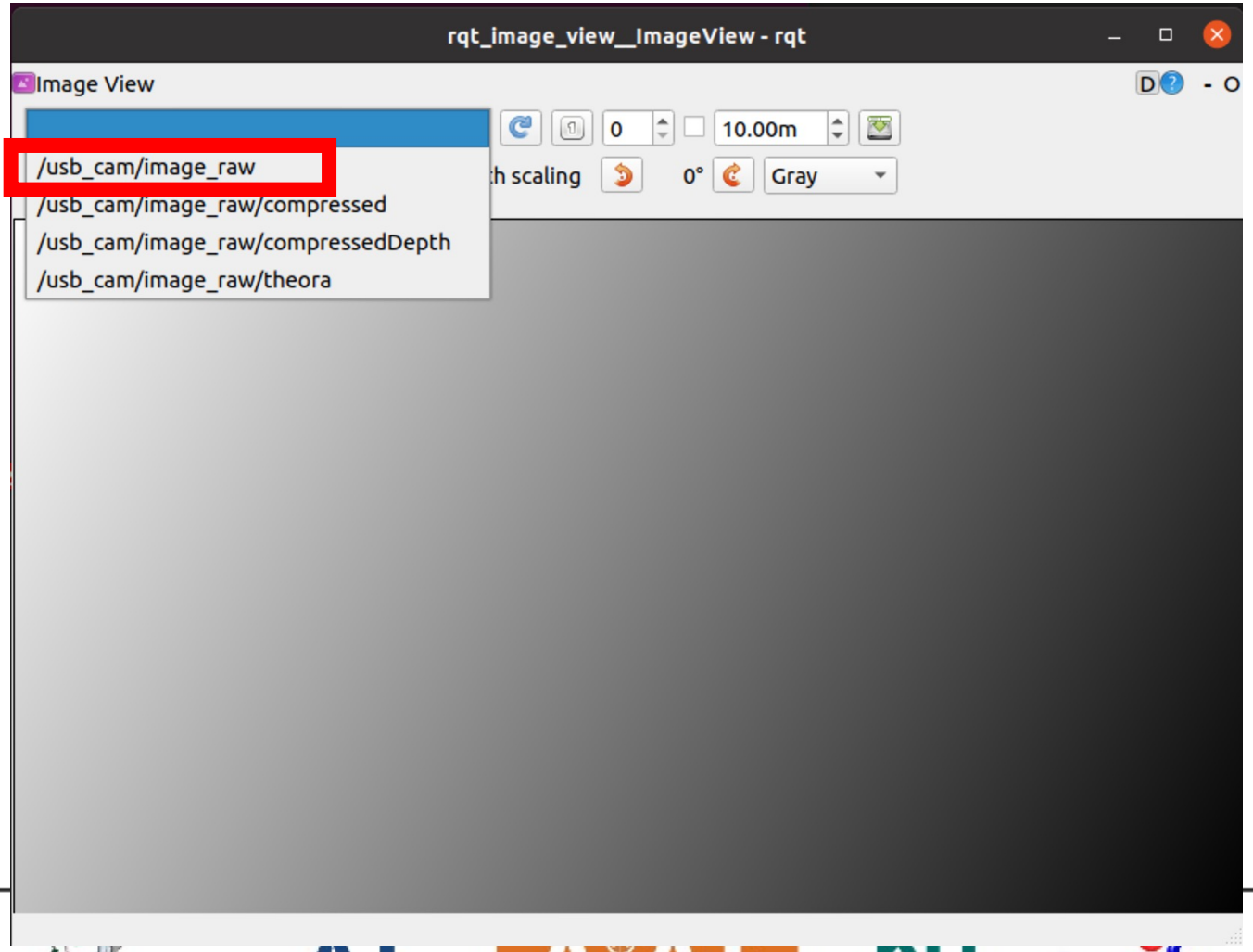
Webcam

```
$ rqt_image_view
```

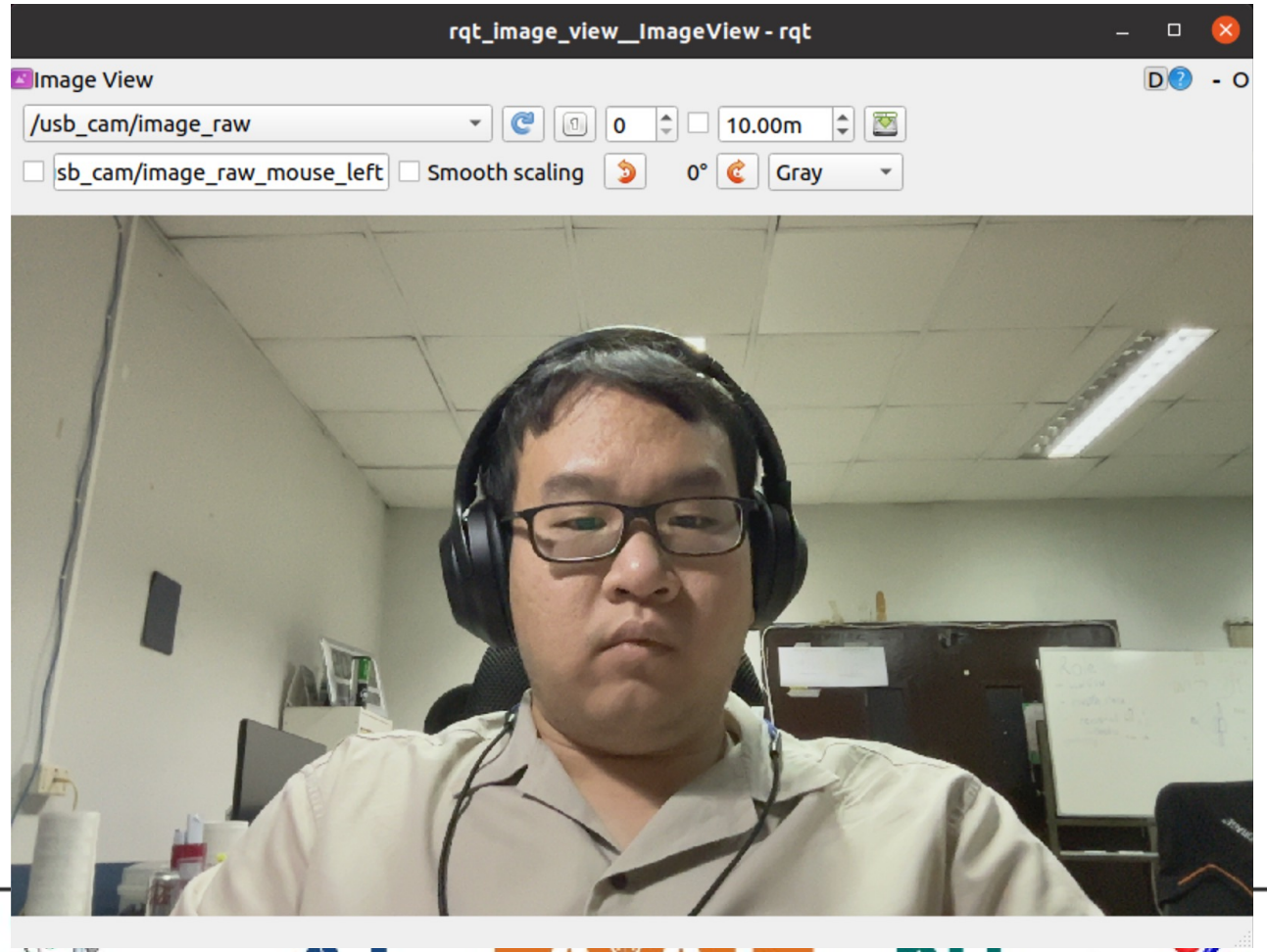
Webcam


rqt_image_view__ImageView - rqt
Image View
0
10.00m
/sb_cam/image_raw_mouse_left
Smooth scaling
0°
Gray

Webcam


rqt_image_view__ImageView - rqt
Image View
0
10.00m
/usb_cam/image_raw
/usb_cam/image_raw/compressed
/usb_cam/image_raw/compressedDepth
/usb_cam/image_raw/theora
0°
Gray

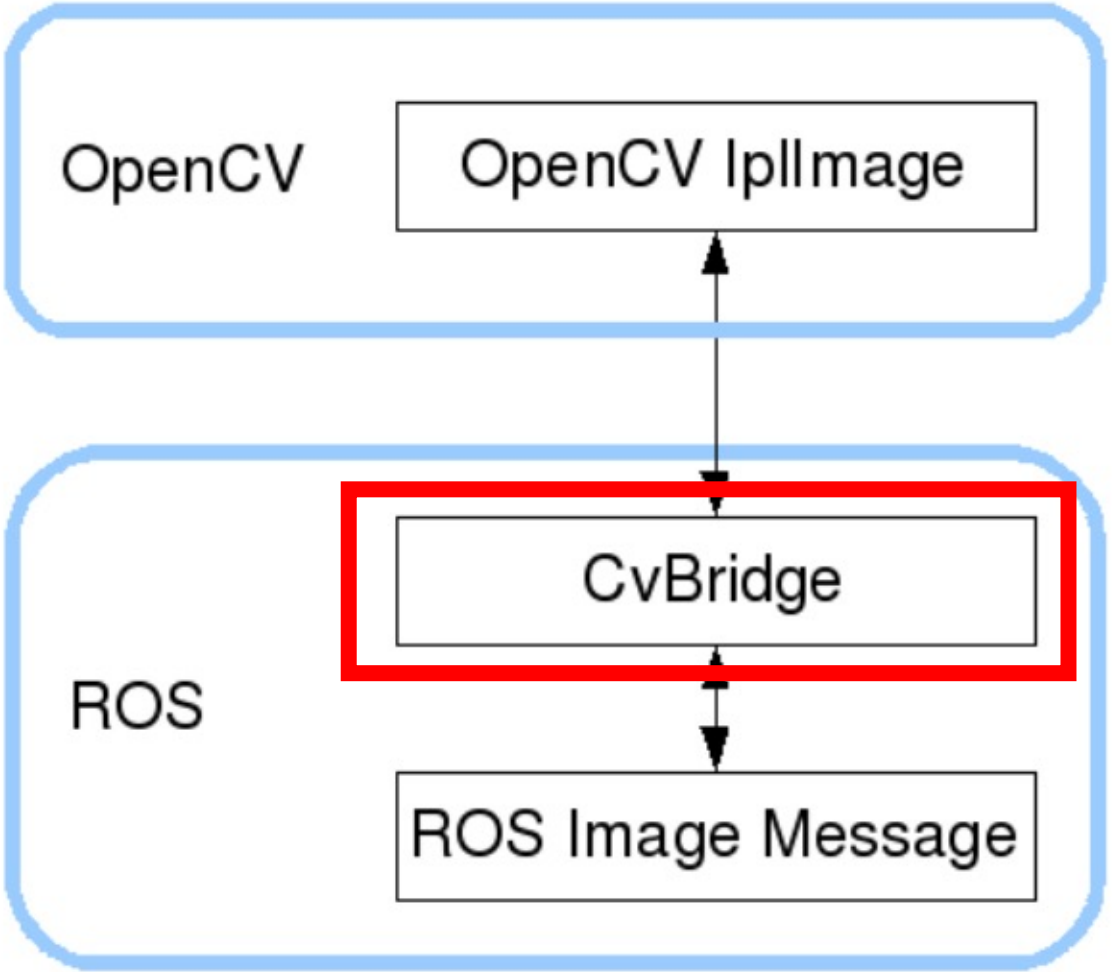
Webcam


rqt_image_view__ImageView - rqt
Image View
/usb_cam/image_raw
0
10.00m
sb_cam/image_raw_mouse_left
Smooth scaling
0°
Gray

Webcam

Using with OpenCV

Webcam



Webcam

```
$ sudo apt install ros-noetic-cv-bridge
```

Webcam

If you **don't** have directory ***script***.

```
$ roscd your_package
```

```
$ mkdir script
```

If you **already** have directory ***script***.

```
$ roscd your_package/script
```

Webcam

```
$ gedit opencv_ros.py
```

Webcam

code

Webcam

```python
#!/usr/bin/env python3
import cv2
import rospy
from cv_bridge import CvBridge
from sensor_msgs.msg import Image


class OpenCVROS(object):
    def __init__(self) -> None:
        rospy.init_node("opencv_ros", anonymous=True)
        self.img_sub = rospy.Subscriber("/usb_cam/image_raw", Image, callback = self.image_callback)
        self.bridge = CvBridge()
        self.image = None

    def image_callback(self, data):
        self.image = self.bridge.imgmsg_to_cv2(data, desired_encoding="bgr8")

    def show_image(self):
        if self.image is None:
            return 0
        cv2.imshow("image", self.image)
        if cv2.waitKey(1) == ord('q'):
            exit()

    def main(self):
        while not rospy.is_shutdown():
            self.show_image()

if __name__ == "__main__":
    opencv_ros = OpenCVROS()
    opencv_ros.main()
```

Webcam

```python
#!/usr/bin/env python3
import cv2
import rospy
from cv_bridge import CvBridge
from sensor_msgs.msg import Image


class OpenCVROS(object):
    def __init__(self):
        rospy.init_node("opencv_ros", anonymous=True)
        self.image_sub = rospy.Subscriber("/camera/image_raw", Image, callback = self.image_callback)
        self.bridge = CvBridge()
        self.image = None

    def image_callback(self, data):
        self.image = self.bridge.imgmsg_to_cv2(data, desired_encoding="bgr8")

    def show_image(self):
        if self.image is None:
            return 0
        cv2.imshow("image", self.image)
        if cv2.waitKey(1) == ord('q'):
            exit()

    def main(self):
        while not rospy.is_shutdown():
            self.show_image()

if __name__ == "__main__":
    opencv_ros = OpenCVROS()
    opencv_ros.main()
```

Webcam

```python
#!/usr/bin/env python3
import cv2
import rospy
from cv_bridge import CvBridge
from sensor_msgs.msg import Image


class OpenCVROS(object):
    def __init__(self) -> None:
        self.img_sub = rospy.Subscriber("/usb_cam/image_raw", Image, callback = self.image_callback)
        self.image = None

    def image_callback(self, data):
        self.image = self.bridge.imgmsg_to_cv2(data, desired_encoding="bgr8")

    def show_image(self):
        if self.image is None:
            return 0
        cv2.imshow("image", self.image)
        if cv2.waitKey(1) == ord('q'):
            exit()

    def main(self):
        while not rospy.is_shutdown():
            self.show_image()

if __name__ == "__main__":
    opencv_ros = OpenCVROS()
    opencv_ros.main()
```

Webcam

```python
#!/usr/bin/env python3
import cv2
import rospy
from cv_bridge import CvBridge
from sensor_msgs.msg import Image


class OpenCVROS(object):
    def __init__(self) -> None:
        rospy.init_node("opencv_ros", anonymous=True)
        self.bridge = CvBridge()

    def image_callback(self, data):
        self.image = self.bridge.imgmsg_to_cv2(data, desired_encoding="bgr8")

    def show_image(self):
        if self.image is None:
            return 0
        cv2.imshow("image", self.image)
        if cv2.waitKey(1) == ord('q'):
            exit()

    def main(self):
        while not rospy.is_shutdown():
            self.show_image()

if __name__ == "__main__":
    opencv_ros = OpenCVROS()
    opencv_ros.main()
```

Webcam

```python
#!/usr/bin/env python3
import cv2
import rospy
from cv_bridge import CvBridge
from sensor_msgs.msg import Image


class OpenCVROS(object):
    def __init__(self) -> None:
        rospy.init_node("opencv_ros", anonymous=True)
        self.img_sub = rospy.Subscriber("/usb_cam/image_raw", Image, callback = self.image_callback)
        self.image = None

    def image_callback(self, data):
        self.image = self.bridge.imgmsg_to_cv2(data, desired_encoding="bgr8")

    def show_image(self):
        if self.image is None:
            return 0
        cv2.imshow("image", self.image)
        if cv2.waitKey(1) == ord('q'):
            exit()

    def main(self):
        while not rospy.is_shutdown():
            self.show_image()

if __name__ == "__main__":
    opencv_ros = OpenCVROS()
    opencv_ros.main()
```

Webcam

```python
#!/usr/bin/env python3
import cv2
import rospy
from cv_bridge import CvBridge
from sensor_msgs.msg import Image


class OpenCVROS(object):
    def __init__(self) -> None:
        rospy.init_node("opencv_ros", anonymous=True)
        self.img_sub = rospy.Subscriber("/usb_cam/image_raw", Image, callback = self.image_callback)
        self.bridge = CvBridge()
        self.image = None

    def image_callback(self, data):
        self.image = self.bridge.imgmsg_to_cv2(data, desired_encoding="bgr8")

    def show_image(self):
        if self.image is None:
            return 0
        cv2.imshow("image", self.image)
        if cv2.waitKey(1) == ord('q'):
            exit()

    def main(self):
        while not rospy.is_shutdown():
            self.show_image()

if __name__ == "__main__":
    opencv_ros = OpenCVROS()
    opencv_ros.main()
```

Webcam

```python
#!/usr/bin/env python3
import cv2
import rospy
from cv_bridge import CvBridge
from sensor_msgs.msg import Image


class OpenCVROS(object):
    def __init__(self) -> None:
        rospy.init_node("opencv_ros", anonymous=True)
        self.img_sub = rospy.Subscriber("/usb_cam/image_raw", Image, callback = self.image_callback)
        self.bridge = CvBridge()
        self.image = None

    def image_callback(self, data):
        self.image = self.bridge.imgmsg_to_cv2(data, desired_encoding="bgr8")

    def show_image(self):
        if self.image is None:
            return 0
        cv2.imshow("image", self.image)
        if cv2.waitKey(1) == ord('q'):
            exit()

    def main(self):
        while not rospy.is_shutdown():
            self.show_image()

if __name__ == "__main__":
    opencv_ros = OpenCVROS()
    opencv_ros.main()
```

Webcam

```
$ roslaunch your_package usb_cam.launch
```

Webcam

```
$ python3 opencv_ros.py
```

Kinect (Depth camera)

```
$ git clone https://github.com/OpenKinect/libfreenect.git
```

```
$ cd libfreenect/
```

```
$ mkdir build && cd build
```

```
$ cmake -L ..
```

Kinect (Depth camera)

```
$ make
```

```
$ sudo make install
```

```
$ sudo ldconfig /usr/local/lib64/
```

Kinect (Depth camera)

```
$ cd ..
```

```
$ python3 src/fwfetcher.py
```

```
pat@pat:~/libfreenect$ python3 src/fwfetcher.py
Downloading SystemUpdate.zip from https://www.xbox.com/system-update-usb
Reading response...
done, saved to SystemUpdate.zip
Extracting $SystemUpdate/FFFE07DF00000001 from system update file...
done.
Handling LIVE/PIRS file.
Writing information file FFFE07DF00000001.txt
Creating and filling content directory FFFE07DF00000001.dir
Moving audios.bin to current folder
Cleaning up
Done!
```

Kinect (Depth camera)

```
$ sudo cp audios.bin /usr/local/share/libfreenect
```

Kinect (Depth camera)

```
$ sudo gedit /etc/udev/rules.d/51-kinect.rules
```

Copy all under this and paste to file

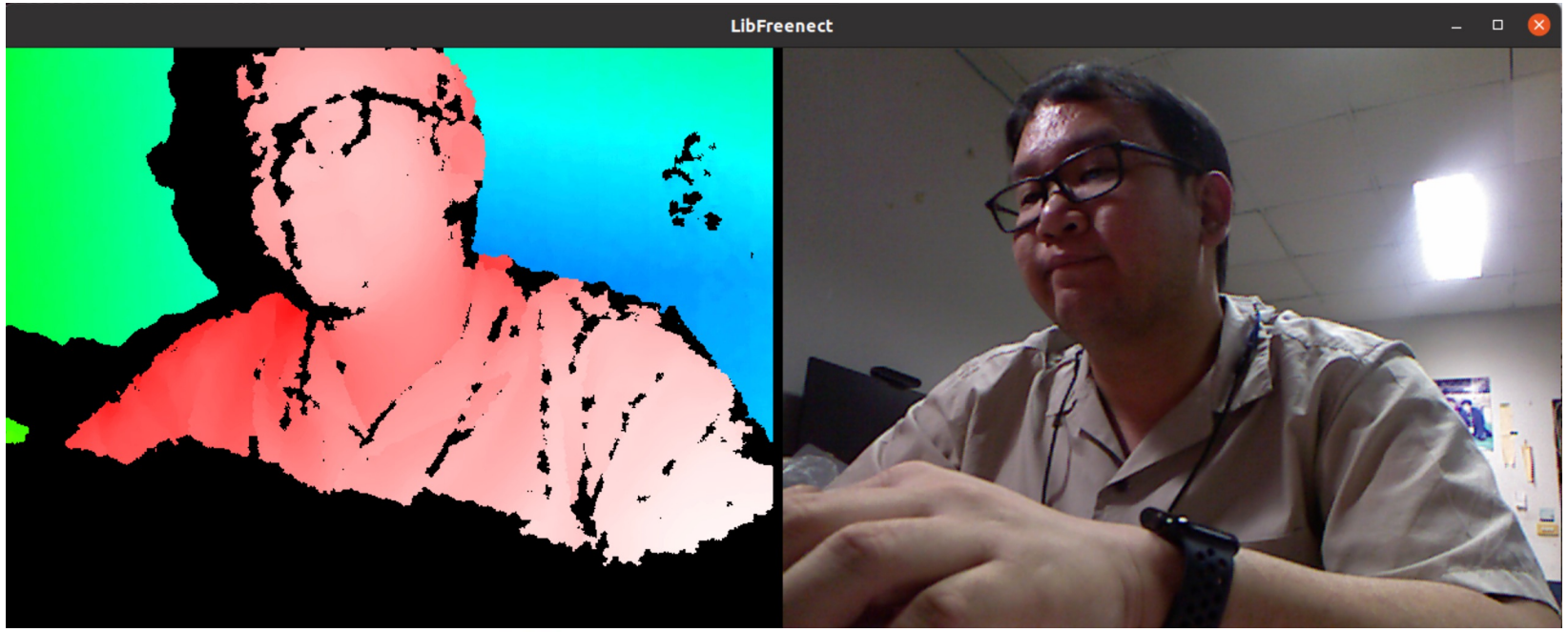
```
"Xbox NUI Motor"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02b0", MODE="0666"
# ATTR{product}=="Xbox NUI Audio"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02ad", MODE="0666"
# ATTR{product}=="Xbox NUI Camera"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02ae", MODE="0666"
# ATTR{product}=="Xbox NUI Motor"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02c2", MODE="0666"
# ATTR{product}=="Xbox NUI Motor"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02be", MODE="0666"
# ATTR{product}=="Xbox NUI Motor"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02bf", MODE="0666"
```

Kinect (Depth camera)

```
$ sudo reboot
```

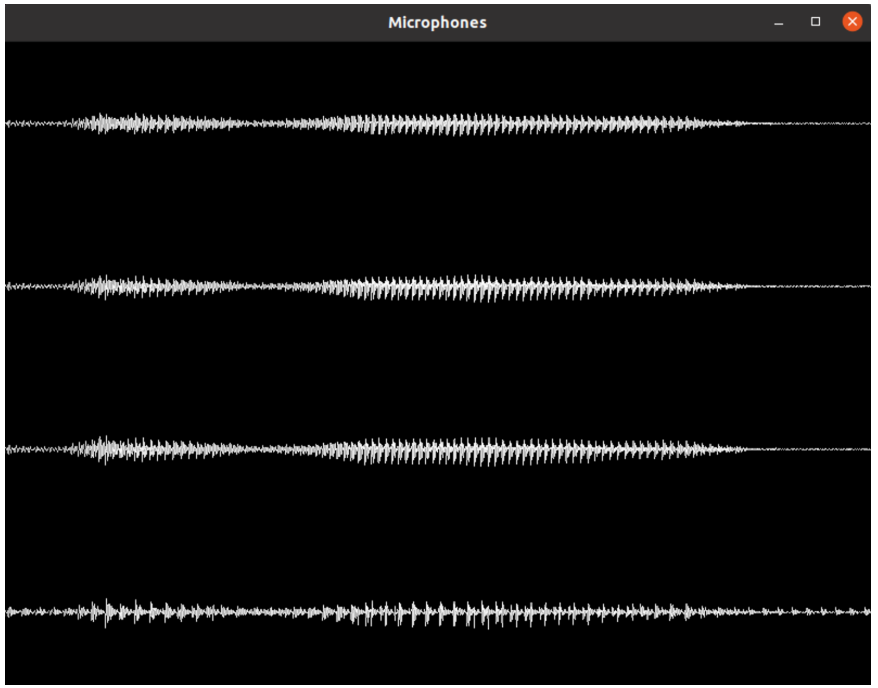
Kinect (Depth camera)

```
$ freenect-glview
```



Kinect (Depth camera)

```
$ freenect-micview
```



Kinect (Depth camera)

```
$ roscd
```

```
$ cd ../src
```

```
$ git clone https://github.com/ros-drivers/freenect_stack.git
```

Kinect (Depth camera)

```
$ cd ../
```

```
$ catkin_make
```

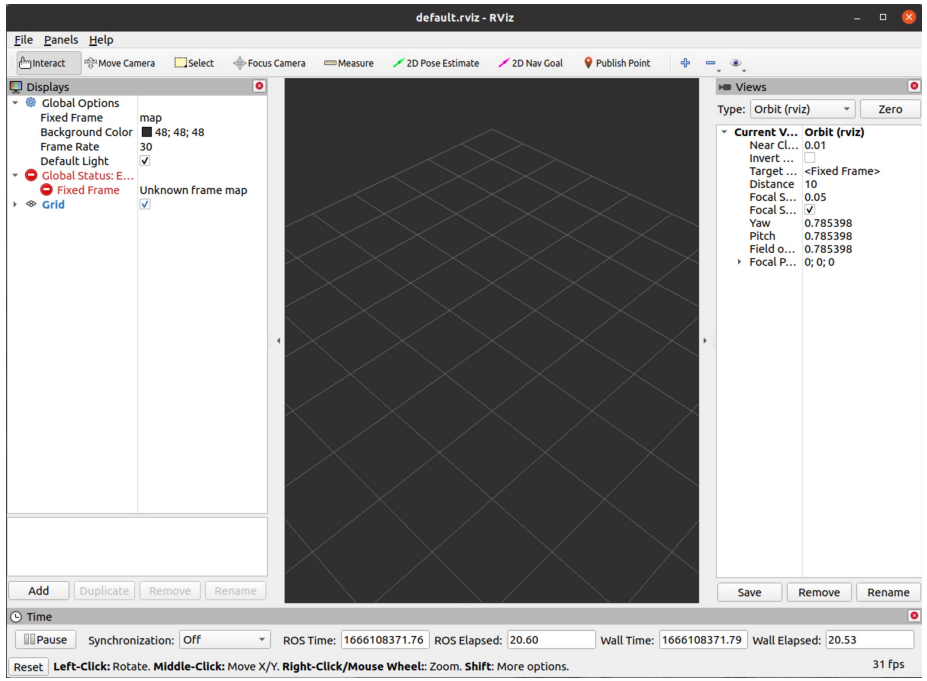
Kinect (Depth camera)

```
$ rospack profile
```

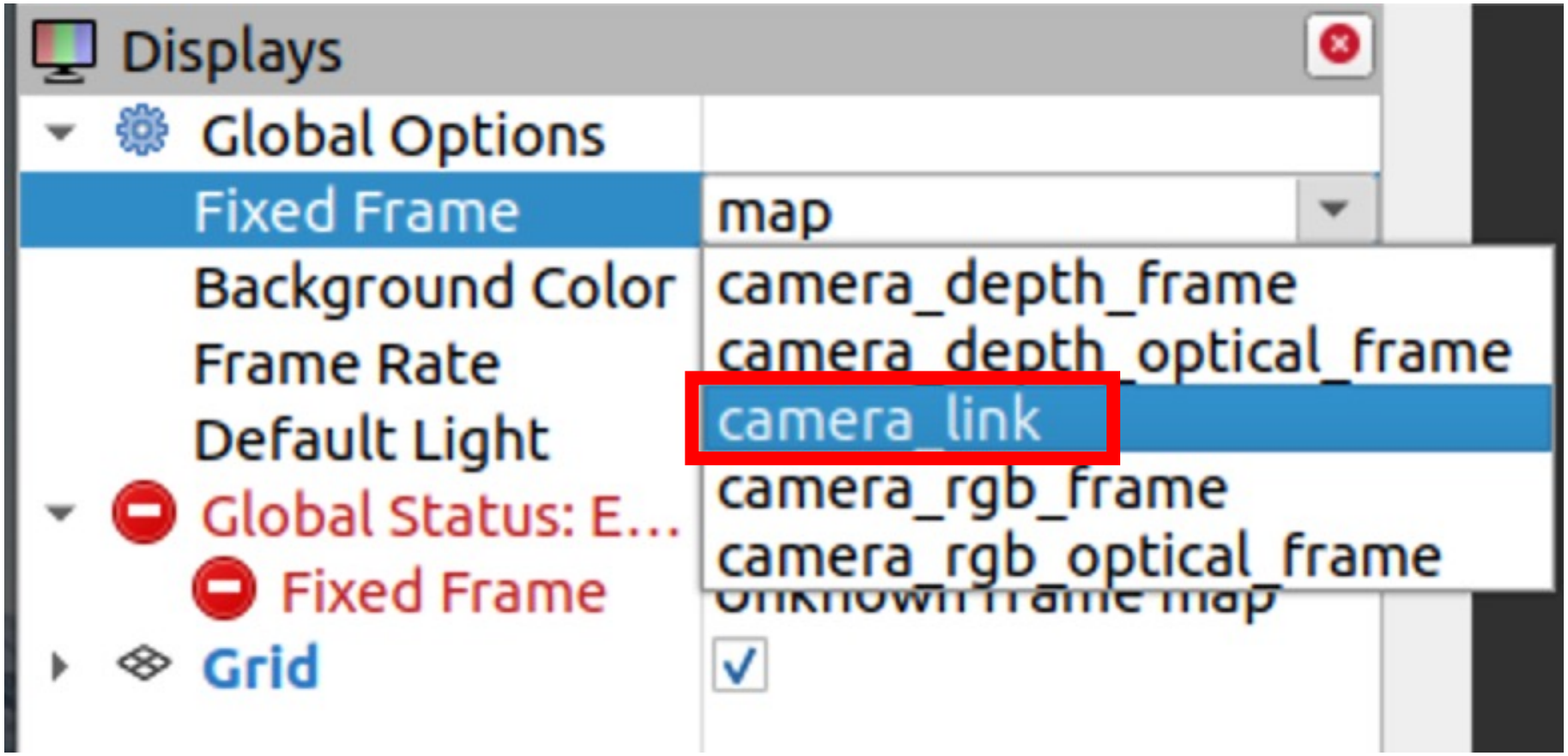
```
$ roslaunch freenect_launch freenect.launch depth_registration:=true
```

Kinect (Depth camera)

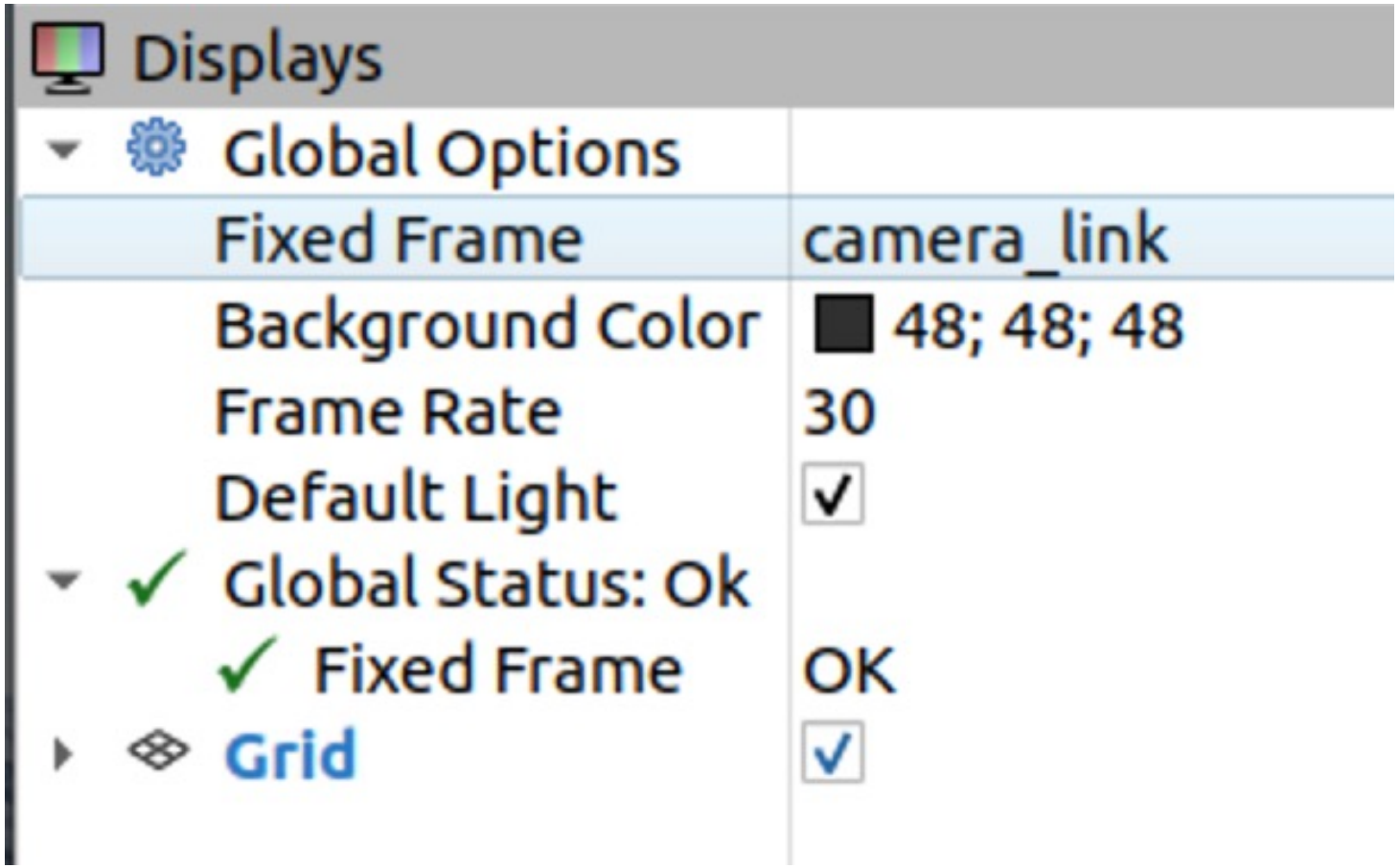
```
$ rviz
```



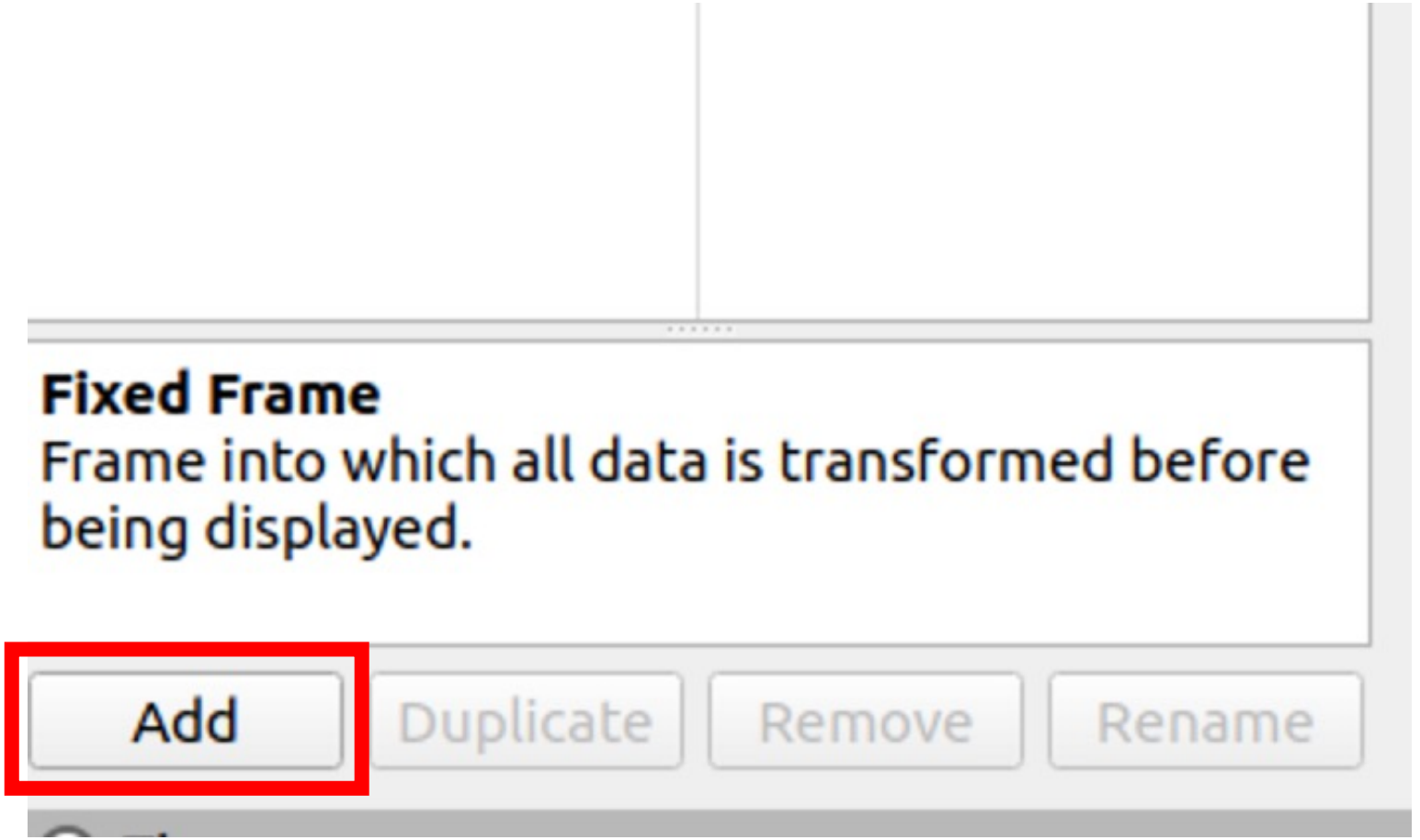
Kinect (Depth camera)



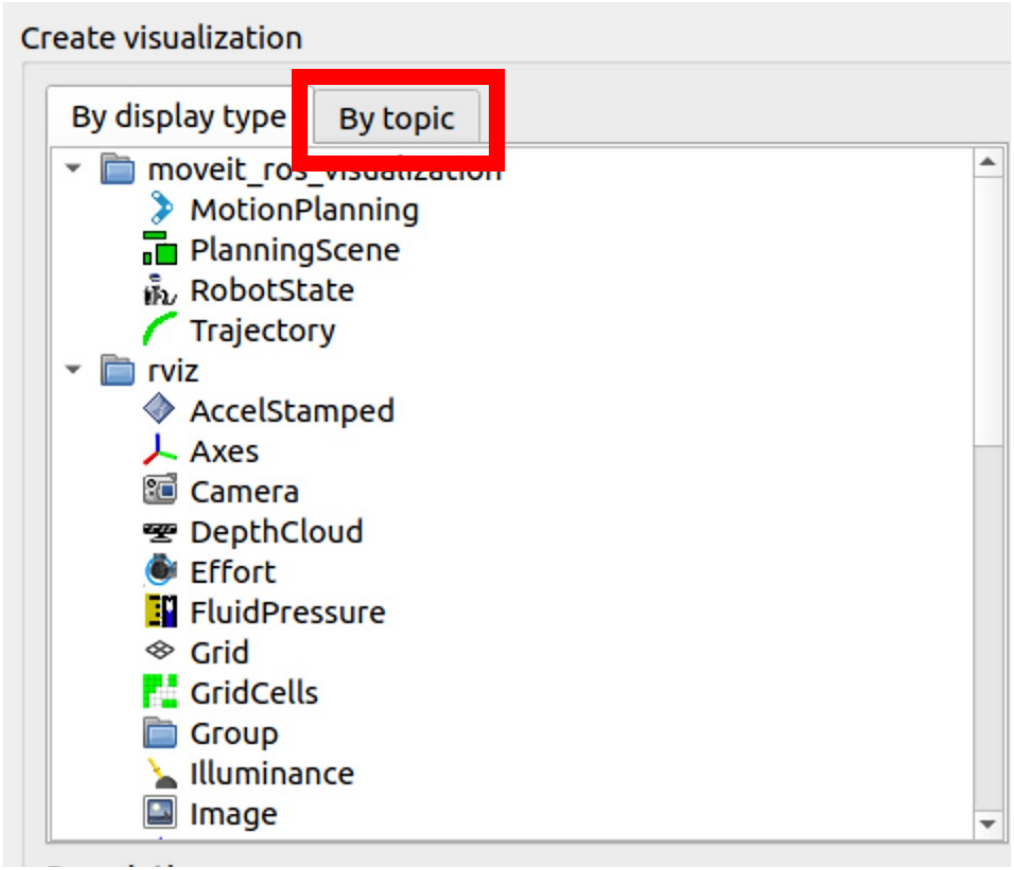
Kinect (Depth camera)



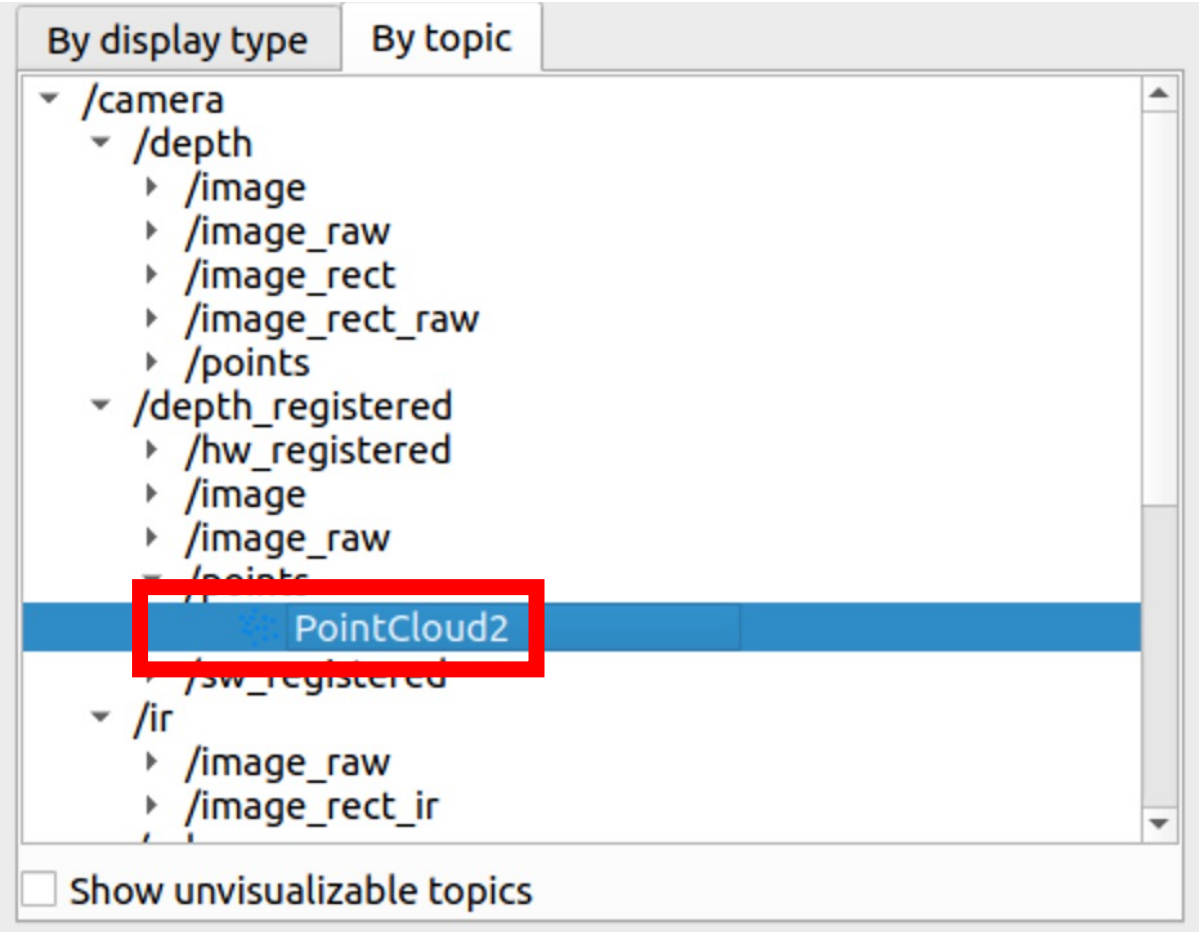
Kinect (Depth camera)



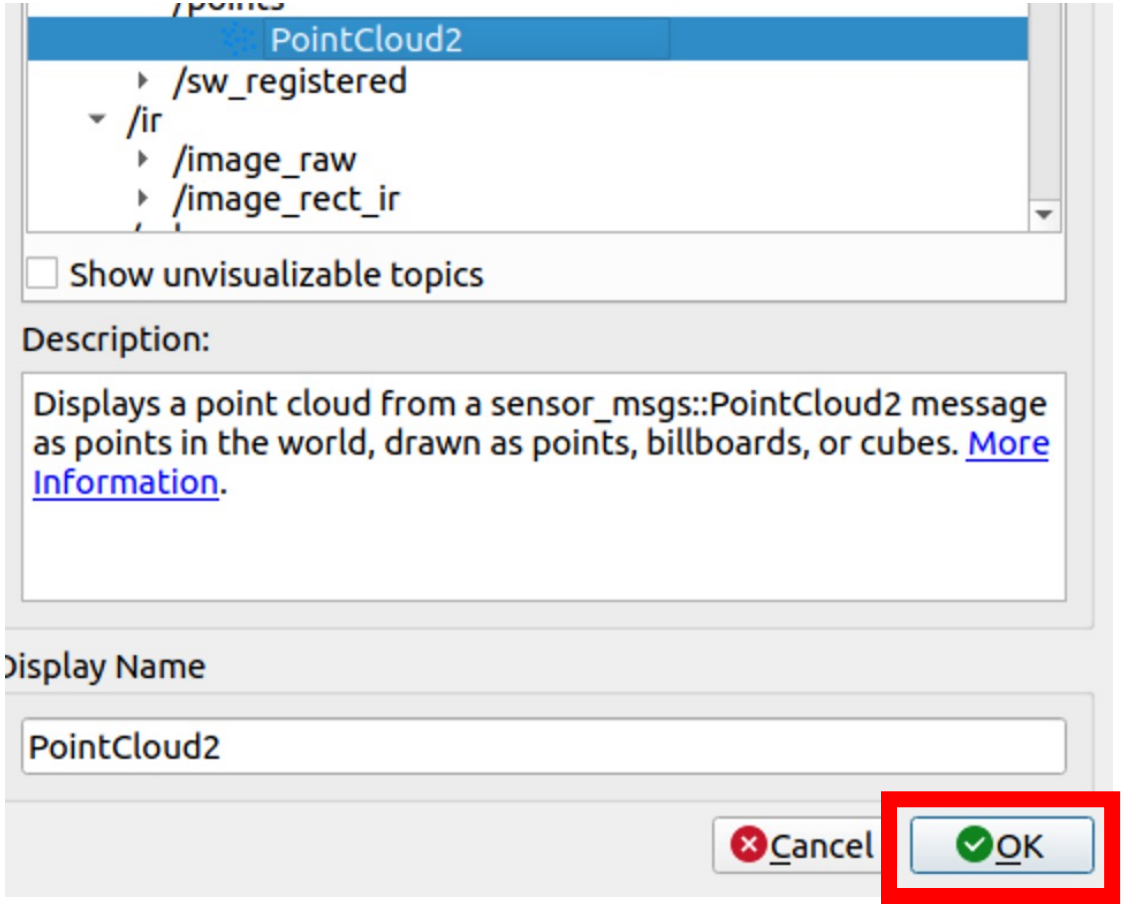
Kinect (Depth camera)



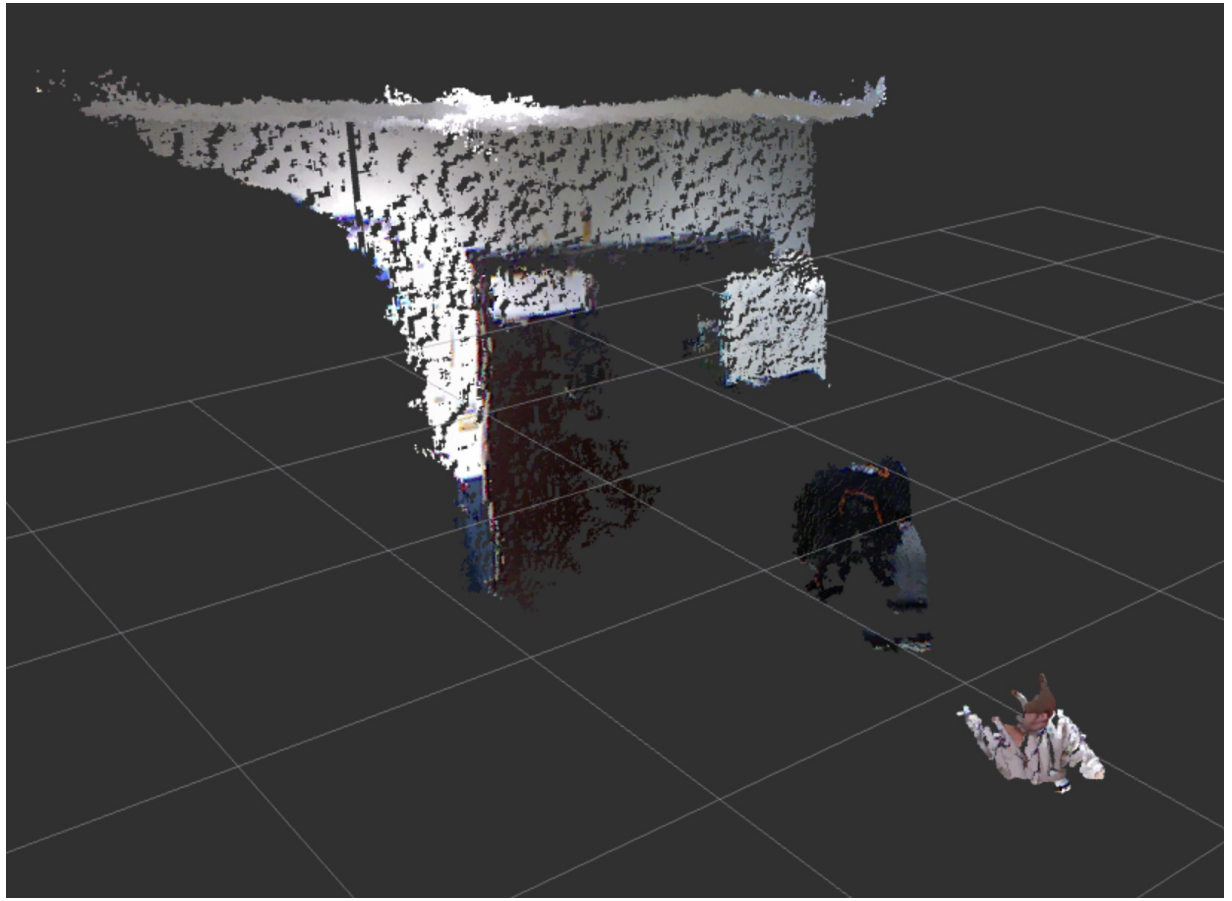
Kinect (Depth camera)



Kinect (Depth camera)



Kinect (Depth camera)

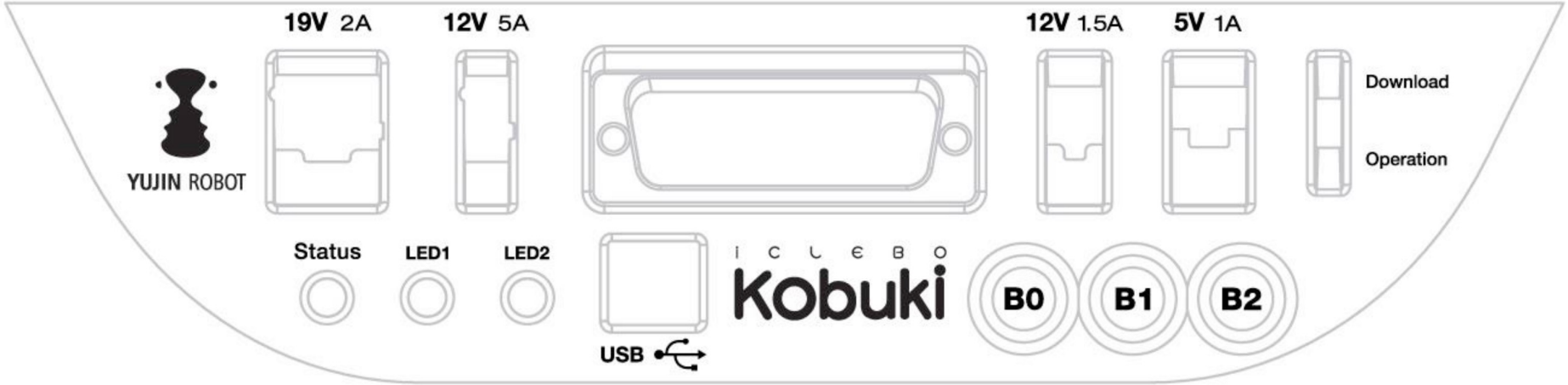


Turtlebot2

Using Turtlebot2

Turtlebot 2

Turtlebot2



19V/2A: Laptop power supply
12V/5A: Arm power supply
12v/1.5A: Microsoft Kinect power supply
5V/1A: General power supply
Status LED: Indicates Kobuki's status
Green: Kobuki is turned on and battery at high voltage level
Orange: On - Low battery voltage level (please charge soon)

Green blinking: On - Battery charging
Off: Kobuki is turned off.
LED1/2: Programmable LEDs
USB: Data connection
B0/1/2: Buttons
Firmware switch: Enable/disables the firmware update mode

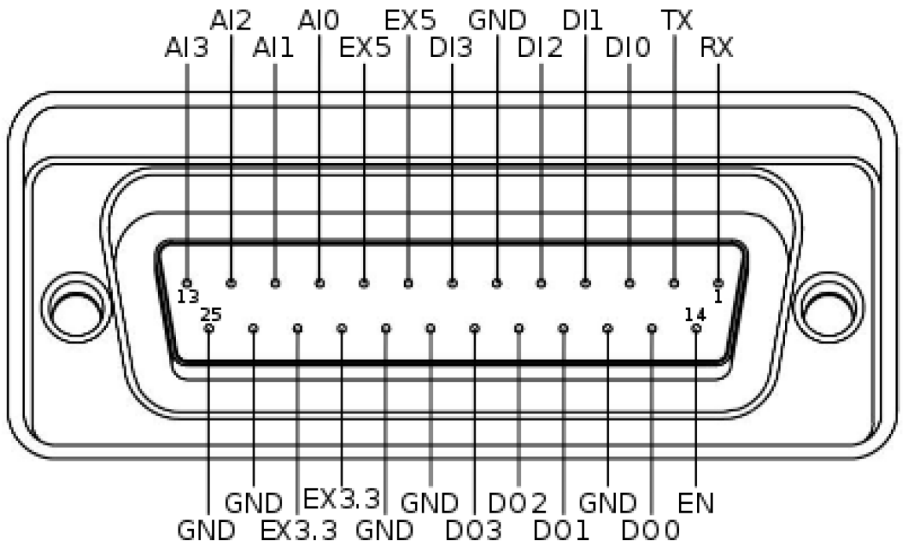
http://kobuki.yujinrobot.com/about2/

Turtlebot 2

Power

- 5V@1A Molex PN : 43650-0218 – for custom boards
- 12V@1.5A : Molex PN : 43045-0224 – specially supporting the Kinect
- 12V@5A : Molex PN : 3929-9023 – for high powered accessories (e.g. robotic arm)
- 19V@2A : Molex PN : 3928-9068 – for recharging netbooks

Turtlebot2

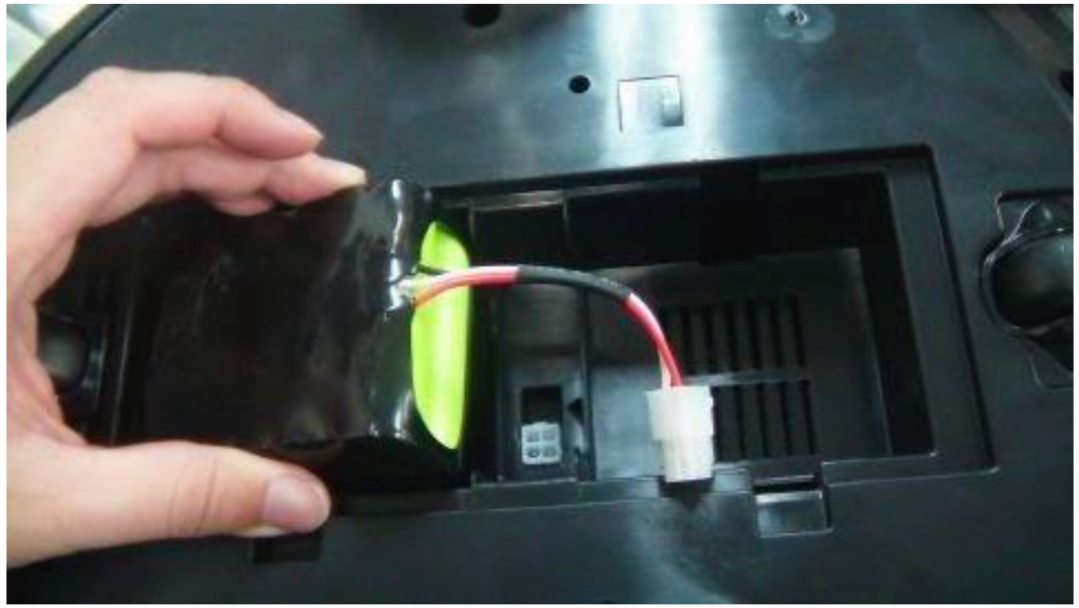


I/O Port

- DB25 pin D-SUB Female connector

RX / TX: Serial data connection (RS232; used voltage level is 3.3V!)
EX3.3 / EX5: 3.3V/1A and 5V/1A power supply
DI0 – 3: 4 x Digital input (high: 3.3 - 5V, low: 0V)
DO0 – 3: 4 x Digital output (open-drain, pull-up resistor required)
AI0–3: 4 x Analog input (12bit ADC: 0 - 4095, 0 - 3.3V)
GND: Ground
EN: Used for detecting an external board (connect to external ground)

Turtlebot2



Battery

- 4S1P/4S2P Battery Pack Connector

Turtlebot2

Cables

- **12V@1.5A :** Molex PN : 43025-0200 – specially supporting the Kinect

Turtlebot2 package

- Go into your workspace **src**

```
$ roscd
```

```
$ cd ../src
```

Turtlebot2 package

- Clone all these repository into your **src**

```
$ git clone <URL>
```

  - https://github.com/turtlebot/turtlebot.git
  - https://github.com/turtlebot/turtlebot_msgs.git
  - https://github.com/turtlebot/turtlebot_apps.git
  - https://github.com/yujinrobot/kobuki.git
  - https://github.com/yujinrobot/kobuki_msgs.git
  - https://github.com/yujinrobot/yujin_ocs.git

Turtlebot2 package

- Go back to your workspace

```
$ cd ..
```

- Build your workspace

```
$ catkin_make
```

- Update ROS package

```
$ rospack profile
```

Initialize USB

```
$ sudo usermod -a -G dialout $USER
$ sudo reboot
```

```
$ wget https://raw.githubusercontent.com/kobuki-
base/kobuki_ftdi/devel/60-kobuki.rules
```

```
$ sudo cp 60-kobuki.rules /etc/udev/rules.d
```

```
$ sudo service udev reload
```

```
$ sudo service udev restart
```

Turtlebot2 - SLAM

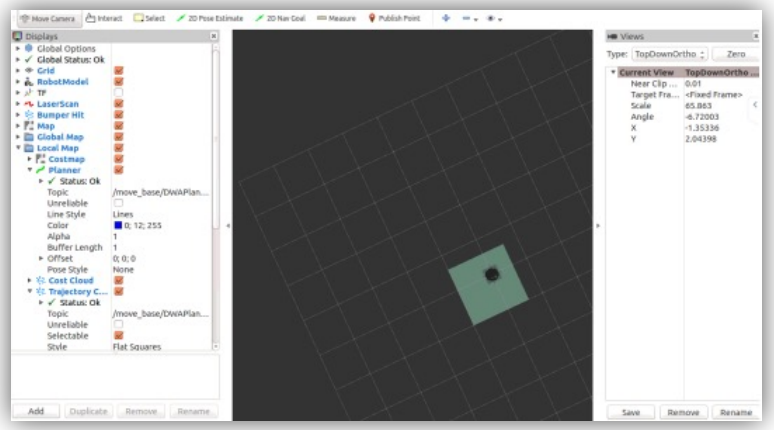
- Bring up your turtlebot2

```
$ roslaunch turtlebot_bringup minimal.launch
```

- Run package for SLAM

```
$ roslaunch turtlebot_navigation gmapping_demo.launch
```

Turtlebot2 - SLAM



- Run visualization system

```
$ roslaunch turtlebot_rviz_launchers view_navigation.launch
```

- Run remote controller

```
$ roslaunch turtlebot_teleop logitech.launch
```

- If don’t have joy

```
$ roslaunch turtlebot_teleop keyboard_teleop.launch
```

Turtlebot2 - SLAM

- Save your map

```
$ cd ~/
```

```
$ rosrun map_server map_saver -f map
```

```
$ ls
```