

Integrating RGBD Camera to TB3 using ROS – A step-by-step Guide.

Objective:

This guide explains how to integrate an RGBD camera (OAK-D Pro) with the TurtleBot3 (Burger model) for static and dynamic map creation using sensor fusion. This system will allow the TurtleBot to perform autonomous navigation using LiDAR and RGB-D sensors.

System Components:

1. TurtleBot3 (Burger model)
2. RGBD Camera (OAK-D Pro)
3. LiDAR sensor (Mounted on TurtleBot3)
4. 2 Personal Computers (High-performance recommended)
5. 1 Portable PC

Software Requirements:

- ROS Noetic
- libopencv-dev
- python3-rosdep
- depthimage_to_laserscan
- ira_laser_tools

Installation Links:

- [ROS Noetic Installation Guide](#)
- [DepthImage to LaserScan](#)
- [Ira Laser Tools](#)

Part 1: Static Map (Using LiDAR sensor to create maps and navigation)

This section assumes that the TurtleBot3 has been fully configured. For setup, refer to the [TurtleBot3 Quick Start Guide](#).

1) Run roscore from the Master Laptop

Open a terminal on the master laptop and run the following command to start ROS:

```
jakelcj@jakelcj-Latitude-3420:~$ roscore
... logging to /home/jakelcj/.ros/log/866a71c4-4a2f-11ef-afba-0d80916fbd09/roslaunch-jakelcj-La
titude-3420-9169.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://192.168.0.142:32879/
ros_comm version 1.16.0
```

Diagram 1.0

2) Connect the Master laptop to the Turtlebot

Ensure both the TurtleBot and the master laptop are on the same network. Use SSH to connect:

```
jakelcj@jakelcj-Latitude-3420:~$ ssh ubuntu@192.168.0.141
ubuntu@192.168.0.141's password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-1104-raspi aarch64)
```

Diagram 2.0

Default username: `ubuntu`, password: `turtlebot`.

3) Configure Network Settings

Modify the `.bashrc` file on both the TurtleBot and the laptop. Append the following lines:

- **Master Laptop:**

```
echo 'export ROS_MASTER_URI=http://192.168.1.100:11311' >> ~/.bashrc
echo 'export ROS_HOSTNAME=192.168.1.200' >> ~/.bashrc
```

Replace `192.168.1.100` with the IP address of your ROS master and `192.168.1.200` with the IP address of your laptop.

- **Turtlebot:**

```
echo 'export ROS_MASTER_URI=http://192.168.1.100:11311' >> ~/.bashrc
echo 'export ROS_HOSTNAME=192.168.1.200' >> ~/.bashrc
```

Replace 192.168.1.100 with the IP address of your ROS master and 192.168.1.200 with the IP address of your Turtlebot3.

Once done, apply the changes immediately by using the command 'source ~/.bashrc'.

4) Start TurtleBot3 Core

```
ubuntu@ubuntu:~$ roslaunch turtlebot3_bringup turtlebot3_robot.launch
... logging to /home/ubuntu/.ros/log/b3fa95ba-4a39-11ef-afba-0d80916fbd09/roslaunch-ubuntu-1389
.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://192.168.0.141:42395/
```

Diagram 3.0

1) When successfully done, the terminal will print these messages.

```
SUMMARY
=====

PARAMETERS
* /roscpp: noetic
* /rosversion: 1.16.0
* /turtlebot3_core/serial_port: /dev/ttyACM0
* /turtlebot3_core/serial_baud: 115200
* /turtlebot3_core/tf_prefix:
* /turtlebot3_lds/frame_id: base_scan

NODES
/
  turtlebot3_core (roscpp/python/serial_node.py)
  turtlebot3_diagnostics (turtlebot3_bringup/turtlebot3_diagnostics)
  turtlebot3_lds (ld08_driver/ld08_driver)

ROS_MASTER_URI=http://192.168.0.142:11311

process[turtlebot3_core-1]: started with pid [1406]
process[turtlebot3_lds-2]: started with pid [1407]
process[turtlebot3_diagnostics-3]: started with pid [1408]
/dev/ttyUSB0 CP2102 USB to UART Bridge Controller
/dev/ttyACM0 OpenCR Virtual ComPort in FS Mode
FOUND LiDAR_LD08 @port :/dev/ttyUSB0
[INFO] [1721880218.579298]: ROS Serial Python Node
[INFO] [1721880218.639821]: Connecting to /dev/ttyACM0 at 115200 baud
[INFO] [1721880220.763215]: Requesting topics...
[INFO] [1721880220.824019]: Note: publish buffer size is 1024 bytes
[INFO] [1721880220.832980]: Setup publisher on sensor_state [turtlebot3_msgs/SensorState]
[INFO] [1721880220.972804]: Setup publisher on firmware_version [turtlebot3_msgs/VersionInfo]
[INFO] [1721880221.110848]: Setup publisher on imu [sensor_msgs/Imu]
[INFO] [1721880221.131893]: Setup publisher on cmd_vel_rc100 [geometry_msgs/Twist]
[INFO] [1721880221.176739]: Setup publisher on odom [nav_msgs/Odometry]
[INFO] [1721880221.203059]: Setup publisher on joint_states [sensor_msgs/JointState]
[INFO] [1721880221.247974]: Setup publisher on battery_state [sensor_msgs/BatteryState]
[INFO] [1721880221.269213]: Setup publisher on magnetic_field [sensor_msgs/MagneticField]
[INFO] [1721880221.336492]: Setup publisher on /tf [tf/tfMessage]
[INFO] [1721880221.370921]: Note: subscribe buffer size is 1024 bytes
[INFO] [1721880221.383085]: Setup subscriber on cmd_vel [geometry_msgs/Twist]
[INFO] [1721880221.431927]: Setup subscriber on sound [turtlebot3_msgs/Sound]
```

Diagram 3.1

5) **Save the TURTLEBOT3_MODEL parameter permanently.**

```
$ echo 'export TURTLEBOT3_MODEL=burger' >> ~/.bashrc
$ source ~/.bashrc
```

Diagram 4.0

6) **Launch SLAM Node:**

- 1) Launch a SLAM package (default SLAM method will be gmapping) to generate a 2D occupancy grid map.

```
jakelej@jakelej-Latitude-3420:~$ roslaunch turtlebot3_slam turtlebot3_slam.launch
... logging to /home/jakelej/.ros/log/b3fa95ba-4a39-11ef-afba-0d80916fbd09/roslaunch-jakelej-Latitude-3420-14554.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.

xacro: in-order processing became default in ROS Melodic. You can drop the option.
started roslaunch server http://192.168.0.142:44343/
```

Diagram 5.0

- 2) 3D visualization tool such as rviz will automatically open displaying the current surrounding around the TB3.

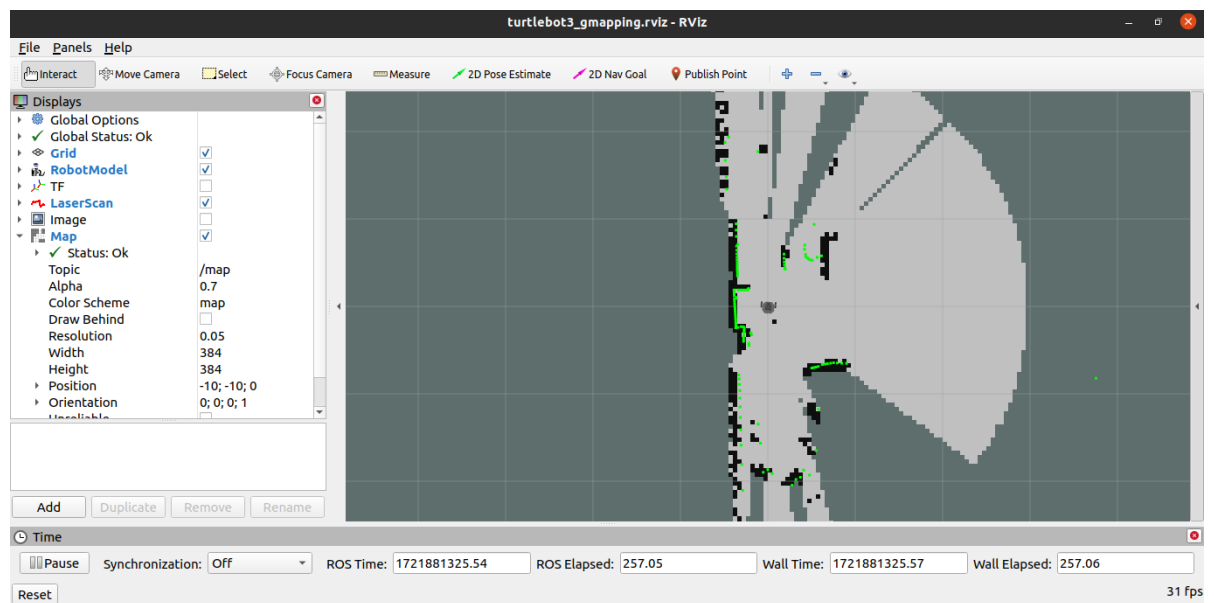


Diagram 5.1

7) **Run the teleoperation node from the Remote PC.**

```
jakelcj@jakelcj-Latitude-3420:~$ export TURTLEBOT3_MODEL=burger
jakelcj@jakelcj-Latitude-3420:~$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
... logging to /home/jakelcj/.ros/log/b3fa95ba-4a39-11ef-afba-0d80916fbd09/roslaunch-jakelcj-Latitude-3420-15007.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://192.168.0.142:41329/
```

Diagram 6.0

- 1) When successfully done, a virtual keyboard analog will appear to control the TurtleBot.

```
ROS_MASTER_URI=http://192.168.0.142:11311

process[turtlebot3_teleop_keyboard-1]: started with pid [15021]

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
```

Diagram 6.1

8) **Collect information by exploring using the TurtleBot3.**

9) **Save the Static Map:**

- 1) Once the mapping is complete, use the map_saver node from the map_server package to save the generated map as a .pgm and .yaml file for future use.

```
jakelcj@jakelcj-Latitude-3420:~$ roslaunch map_server map_saver -f ~/map
[ INFO] [1721887120.127019239]: Waiting for the map
[ INFO] [1721887120.386785739]: Received a 384 X 384 map @ 0.050 m/pix
[ INFO] [1721887120.386830820]: Writing map occupancy data to /home/jakelcj/map.pgm
[ INFO] [1721887120.393611323]: Writing map occupancy data to /home/jakelcj/map.yaml
[ INFO] [1721887120.393860736]: Done
```

Diagram 8.0

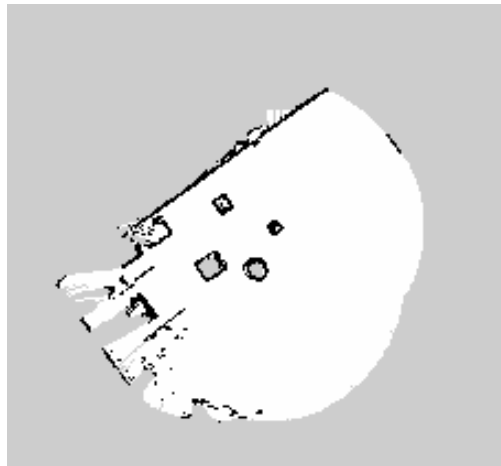


Diagram 8.1

```
Y map.yaml X
C: > Users > jacob > Downloads > Y map.yaml
1 image: /home/jakelcj/map.pgm
2 resolution: 0.050000
3 origin: [-10.000000, -10.000000, 0.000000]
4 negate: 0
5 occupied_thresh: 0.65
6 free_thresh: 0.196
7
8
```

Diagram 8.2

10) Autonomous Navigation of a Known Map with TurtleBot:

- 1) Test the navigation stack by loading the static map and observing how the TurtleBot3 navigates within the mapped area.
- 2) Use RViz to visualize the map and verify its accuracy.

Steps:

- 1) Launch the Navigation on RViz with the map saved using SLAM.

```
jakelcj@jakelcj-Latitude-3420:~$ roslaunch turtlebot3_navigation turtlebot3_navigation.launch m
ap_file:=$HOME/map.yaml
```

Diagram 9.0

- 2) Once the map is opened, check whether the robot is correctly placed and whether the surroundings detected matched the map (There should be red light around the robot representing the objects).
 - If in case the robot did not detect any object near it, check the time synchronization between the Remote PC and the robot.
- 3) To move the robot to the correct location, perform the Initial Pose Estimation by selecting the 2D Pose Estimation function and dragging the green arrow toward the direction where the robot is facing.
- 4) You can also launch keyboard teleoperation node to precisely locate the robot on the map. Don't forget to terminate the keyboard to prevent different cmd_vel values from being published from multiple nodes during Navigation.
- 5) Lastly, set the navigation goal by selecting the 2D Nav Goal function and dragging the red arrow toward the direction where you want the robot to go, and the robot will find the shortest path to reach the destination.

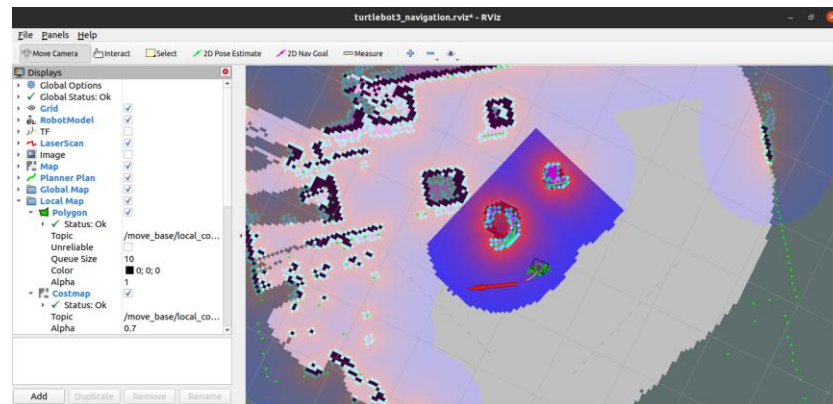


Diagram 9.1

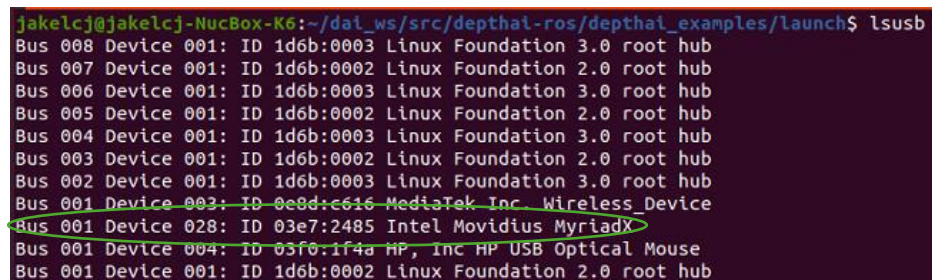
Part 2: Dynamic Map (Integrating Depth Camera for 3D Vision)

To enable dynamic mapping using the RGBD camera (OAK-D Pro), we will connect the camera to a portable PC and integrate its depth data into the TurtleBot's navigation system.

Setup and initial procedure

Connect the Camera and the Portable PC

- 1) Connect the camera to the Portable PC via USB cable.
- 2) Ensure that the camera is successfully detected by the device by using the command 'lsusb'.



```
jake1cj@jake1cj-NucBox-K6:~/dai_ws/src/depthai-ros/depthai_examples/launch$ lsusb
Bus 008 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 007 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 006 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 005 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 004 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 003: ID 0e8d:c616 MediaTek Inc. Wireless Device
Bus 001 Device 028: ID 03e7:2485 Intel Movidius MyriadX
Bus 001 Device 004: ID 03f0:1f4a HP, Inc HP USB Optical Mouse
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Install necessary dependencies and setting up the procedures.

- 1) Once verified, go to <https://github.com/luxonis/depthai-ros> and scroll until you see the link to the newest documentation.
- 2) It will bring you to luxonis page which specify the steps to setup DepthAI ROS.
- 3) You will need to install from source by just following the steps given.
 - If you are using Raspberry Pi, before you try to build your workspace by using the command 'catkin_make', use the command 'export MAKEFLAGS="-j4"' to specify the number of parallel jobs so that the Raspberry Pi won't crash.
- 4) Check if the environment and all the required files are properly built and all directory is available.

Configure Network Settings between Master and Client.

1) You will need to append the environment variable settings directly into the ~/.bashrc file of those devices.

- Set ROS Master URI and Hostname:

- For the Portable PC:

```
echo 'export ROS_MASTER_URI=http://192.168.1.100:11311' >> ~/.bashrc
echo 'export ROS_HOSTNAME=192.168.1.200' >> ~/.bashrc
```

Replace 192.168.1.100 with the IP address of your ROS master and 192.168.1.200 with the IP address of your Portable PC.

2) Once done, apply the changes immediately by using the command 'source ~/.bashrc'.

Camera Startup

Launch the DepthAI camera node

- 1) Run the roscore command from the 'Master' Laptop.
- 2) In the Portable PC, you will need to source your ROS environment first by using the command 'source ~/dai_ws_ws/devel/setup.bash'.
- 3) Next, find the node to activate the camera functionality.

```
jakelcj@jakelcj-Latitude-3420:~/dai_ws/src/depthai-ros$ cd depthai_examples/
jakelcj@jakelcj-Latitude-3420:~/dai_ws/src/depthai-ros/depthai_examples$ ls
CHANGELOG.rst  CMakeLists.txt  launch  nodelet_plugins.xml  package.xml  params  resources  rviz  scripts  src
jakelcj@jakelcj-Latitude-3420:~/dai_ws/src/depthai-ros/depthai_examples$ cd launch
jakelcj@jakelcj-Latitude-3420:~/dai_ws/src/depthai-ros/depthai_examples/launch$ ls
image_laser.launch  mobile_publisher.launch  stereo_inertial_node.launch  stereo_node.launch  yolov4_publisher.launch
image_laser.launch.save  rgb_stereo_node.launch  stereo_minicer.launch  stereo_nodelet.launch
```

- 4) Launch the 'stereo_inertial_node.launch' file to gain depth image output.

```
jakelcj@jakelcj-Latitude-3420:~/dai_ws/src/depthai-ros/depthai_examples/launch$ roslaunch stereo_inertial_node.launch
```

Camera Calibration Using an 8x6 Checkerboard

1) Install the Camera Calibration Package:

- Make sure you have the necessary calibration package installed in ROS:

```
sudo apt-get install ros-noetic-camera-calibration
```

2) Prepare the Checkerboard:

- Print an 8x6 checkerboard pattern (8 corners horizontally, 6 vertically) on A3 paper. Ensure that:

- 1) The checkerboard is flat and without any wrinkles.
- 2) It is printed to scale without any resizing.
- 3) Secure the checkerboard on a flat surface for calibration.

- You can find a printable checkerboard here:

<https://calib.io/pages/camera-calibration-pattern-generator>

3) Configure the Camera Calibration Node:

- 1) Connect your camera to the system.
- 2) Make sure the camera node is publishing the image topic (e.g.,
`/camera/rgb/image_raw`).
- 3) Launch the ROS camera driver to start publishing image data if not already running.
- 4) Identify the image topic and camera info topic for your camera
(typically `/camera/rgb/image_raw` and `/camera/rgb/camera_info`).

4) Launch the camera calibration node.

- Run the following command to start the camera calibration:

```
jake1cj@jake1cj-NucBox-K6:~/dat_ws/src/depthai-ros/depthai_ros_driver/launch$ ls
calibration.launch  example_marker_publish.launch  example_segmentation.launch  rgbd_pcl.launch  sr_rgbd_pcl.launch
camera.launch       example_multicam.launch        pointcloud.launch            rtabmap.launch
jake1cj@jake1cj-NucBox-K6:~/dat_ws/src/depthai-ros/depthai_ros_driver/launch$ roslaunch calibration.launch
```

- Ensure that:
 - --size 8x6 corresponds to the checkerboard pattern (8 corners horizontally, 6 vertically).
 - --square 0.025 indicates the size of each square in meters (adjust according to your checkerboard size).
- Move the checkerboard around the camera's field of view. Ensure you cover different angles and distances until the calibration node indicates success.

5) Save and Commit the Calibration:

- Once calibration is completed, save the calibration parameters. You'll be prompted to save the data to a .yaml file. This file will contain your camera's intrinsic parameters (camera matrix, distortion coefficients, etc.).

```
jake1cj@jake1cj-NucBox-K6:~$ cd /home/jake1cj/.ros/camera_info
jake1cj@jake1cj-NucBox-K6:~/ros/camera_info$ ls
rgb.yaml
```

```
GNU nano 4.8                                rgb.yaml
image_width: 1280
image_height: 720
camera_name: /rgb
camera_matrix:
  rows: 3
  cols: 3
  data: [1125.952159295945, 0, 670.5382640966848, 0, 1130.407409679329, 395.1580955440204, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [0.1813849244330842, -0.218675721099819, 0.003059744216814556, 0.01114603191941432, 0]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [1165.798461914062, 0, 681.8176628263682, 0, 0, 1182.637329101562, 396.2931986125295, 0, 0, 0, 1, 0]
```

Correct the Coordinate Transform for Related Sensors with the Robot

1) Open the Camera Launch File:

- Locate the launch file for your camera. This file usually defines the transformation between the camera frame and the robot's base frame.
- Example: `stereo_inertial_node.launch`

2) Measure the Real-World Position of the Camera:

- Position the Camera: Start with the camera mounted on your robot.
- Measure Offsets: Physically measure the camera's position relative to the robot's base.
 - Measure the:
 1. X (forward/backward distance),
 2. Y (left/right distance), and
 3. Z (height).
 - Also measure the orientation of the camera in terms of roll, pitch, and yaw.

3) Update the Transformation Parameters:

- In your camera's launch file or in a custom transform node, modify the x, y, z, roll, pitch, and yaw values according to your measurements.

4) Test and Verify the Transform:

- Launch your camera node and robot.
- Use rviz to visualize the camera frame and ensure it aligns correctly with the robot's frame.
- If necessary, adjust the x, y, z, roll, pitch, and yaw values until the camera is correctly positioned and oriented in the real world.

5) Save the Transform:

- Once you are satisfied with the transform, ensure it is permanently saved in the launch file for future use.

Create a Static Map using SLAM

Please refer to the previous example.

First Layer (Displaying the Laser Scan onto the map)

Build the Node from Source.

- 1) Now you will need to clone the 'depthimage_to_laserscan' repository from the ROS GitHub in your Portable PC.

```
jakelcj@jakelcj-Latitude-3420:~$ cd dai_ws/src
jakelcj@jakelcj-Latitude-3420:~/dai_ws/src$ git clone https://github.com/ros-perception/depthimage_to_laserscan.git
```

- 2) Then build the package again.

```
catkin_make
source devel/setup.bash
```

- 3) Check if the 'depthimage_to_laserscan' node is available in the launch file.

```
jake@jake-Latitude-3420:~$ cd dai_ws/src/depthimage_to_laserscan/launch/
jake@jake-Latitude-3420:~/dai_ws/src/depthimage_to_laserscan/launch$ ls
depthimage_to_laserscan.launch
jake@jake-Latitude-3420:~/dai_ws/src/depthimage_to_laserscan/launch$
```

4) Open depthimage_to_laserscan.launch file

```
<!-- Depth Image to Laser Scan Node -->
<node pkg="depthimage_to_laserscan" type="depthimage_to_laserscan" name="depthimage_to_las
  <param name="scan_height" value="1" />
  <param name="scan_time" value="3.0" />
  <param name="range_min" value="0.1" />
  <param name="range_max" value="2.5" />
  <param name="output_frame_id" value="map" />
  <remap from="image" to="/stereo_inertial_publisher/stereo/depth" />
  <remap from="camera_info" to="/stereo_inertial_publisher/color/camera_info" />
  <remap from="scan" to="/laser_scan" />
</node>
```

5) Change the configuration to synchronize with the actual placement.

```
<!-- Depth Image to Laser Scan Node -->
<node pkg="depthimage_to_laserscan" type="depthimage_to_laserscan" name="depthimage_to_laser
  <param name="scan_height" value="3" />
  <param name="scan_time" value="0.051" />
  <param name="range_min" value="0.1" />
  <param name="range_max" value="2.4" />
  <param name="transform_tolerance" value="1.0" />
  <param name="output_frame_id" value="oak-d_frame" />
  <remap from="image" to="/stereo_inertial_publisher/stereo/depth" />
  <remap from="camera_info" to="/stereo_inertial_publisher/color/camera_info" />
  <remap from="scan" to="/laser_scan" />
</node>
```

Merging the laser from the camera and LiDaR sensor.

- 1) Now, in your 'Master' Laptop you will need to clone the 'ira-laser-tools' repository from the ROS GitHub.

```
jake1c@jake1c-Latitude-3420:~/dai_ws/src$ git clone https://github.com/ira-laser-tools/ira_laser_tools.git
```

- 2) Build the package and source your workspace.

```
catkin_make
source devel/setup.bash
```




- 3) Open laserscan_multi_merger.launch file and change the configuration to synchronize with the actual placement.

```
<!-- Launch the existing multi-merger node if needed -->
<node pkg="ira_laser_tools" name="laserscan_multi_merger" type="laserscan_multi_merger" >
  <param name="destination_frame" value="map"/>
  <param name="laserscan_topics" value="/laser_scan /scan" />
  <param name="angle_min" value="-10.0"/>
  <param name="angle_max" value="50.0"/>
  <param name="angle_increment" value="0.005"/>
  <param name="scan_time" value="5"/>
  <param name="range_min" value="0.01"/>
  <param name="range_max" value="80.0"/>
</node>
</launch>
```

- 4) Launch the 'laserscan_multi_merger.launch' file.

```
jake1c@jake1c-Latitude-3420:~/dai_ws/src/ira_laser_tools/launch$ roslaunch laserscan_multi_merger.launch
```

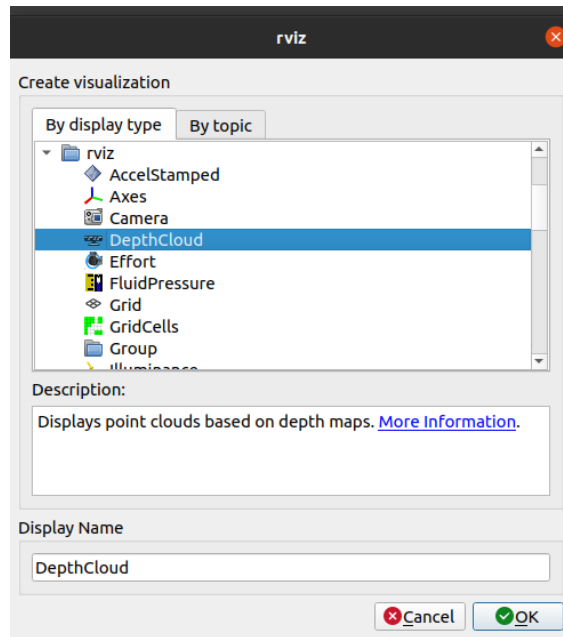
- 5) Subscribe the right laser scan topic to subscribe to.

 LaserScan	<input checked="" type="checkbox"/>
 Status: Ok	
Topic	/scan_multi
Unreliable	<input type="checkbox"/>
Queue Size	10
Selectable	<input checked="" type="checkbox"/>
Style	Flat Squares
Size (m)	0.03
Alpha	1
Decay Time	0
Position Transfor...	XYZ
Color Transformer	FlatColor
Color	 0; 255; 0

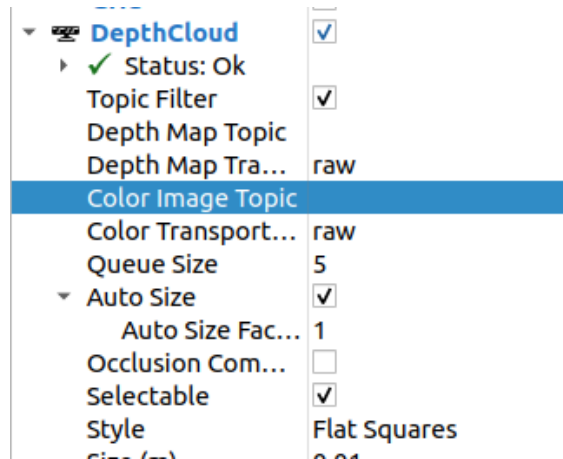
Second Layer (Displaying the 3D images onto the map)

Subscribe to the camera node.

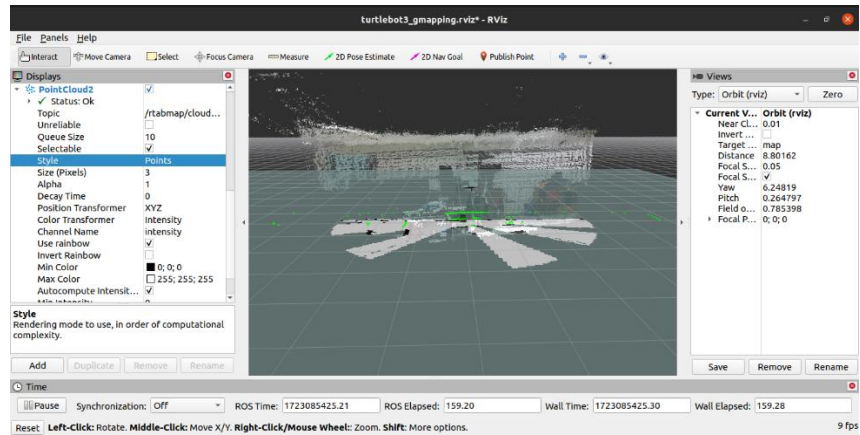
- 1) In the rviz click the button 'add' and add either DepthCloud or Pointcloud2 topic.



- 2) Click on the topic and choose the right image topic to subscribe to.



3) The image will be displayed on the map.



Autonomous Navigation of a Known Map with TurtleBot

Please refer to the previous example.

Configuring the costmap files

Creating inflation area around the objects detected by the LaserScan.

- 1) Locate the param directory for the turtlebot3_navigation and you will find 3 main cost map file that you will need to configure.

1) Local_costmap_params.yaml

```
local_costmap:
  global_frame: map
  robot_base_frame: base_footprint

  update_frequency: 4.0
  publish_frequency: 10.0
  transform_tolerance: 2.5

  static_map: true
  rolling_window: true
  width: 6
  height: 6
  resolution: 0.05

  obstacle_layer:
    obstacle_range: 2.0
    raytrace_range: 5.5
    obstacle_padding: 0.1
    observation_sources: scan
    scan:
      sensor_frame: map
      data_type: LaserScan
      topic: /scan_multi
      marking: true
      clearing: true

  inflation_layer:
    inflation_radius: 0.3
    cost_scaling_factor: 30.0
    enabled: true

  plugins:
    - { name: "obstacle_layer", type: "costmap_2d::ObstacleLayer" }
    - { name: "inflation_layer", type: "costmap_2d::InflationLayer" }

  footprint: [[-0.25, -0.25], [0.25, -0.25], [0.25, 0.25], [-0.25, 0.25]]
```

2) Global_costmap_params.yaml

```
global_costmap:
  global_frame: map
  robot_base_frame: base_footprint

  update_frequency: 4.0
  publish_frequency: 10.0
  transform_tolerance: 2.5

  static_map: false

  obstacle_layer:
    obstacle_range: 2.0
    raytrace_range: 5.5
    obstacle_padding: 0.1
    observation_sources: scan
    scan:
      sensor_frame: map
      data_type: LaserScan
      topic: /scan_multi
      marking: true
      clearing: true

  inflation_layer:
    inflation_radius: 0.5
    cost_scaling_factor: 30.0
    enabled: true

  plugins:
    - { name: "static_layer", type: "costmap_2d::StaticLayer" }
    - { name: "obstacle_layer", type: "costmap_2d::ObstacleLayer" }
    - { name: "inflation_layer", type: "costmap_2d::InflationLayer" }

  # Ensure the resolution is suitable for your application
  resolution: 0.05

  # Optional parameters
  footprint: [[-0.25, -0.25], [0.25, -0.25], [0.25, 0.25], [-0.25, 0.25]]
  rolling_window: false
```

3) costmap_common_params_burger.yaml

```
obstacle_range: 3.0
raytrace_range: 3.5

footprint: [[-0.105, -0.105], [-0.105, 0.105], [0.041, 0.105], [0.041, -0.105]]
#robot_radius: 0.105

map_type: costmap
observation_sources: laser_scan
laser_scan: {sensor_frame: map, data_type: LaserScan, topic: /laser_scan, marking: true, clear: true}

inflation_layer:
  inflation_radius: 0.5
  cost_scaling_factor: 10.0
  enabled: true
```

Results