Lab 2: ROS Perception System Basics — Image Subscription and Processing

1. Experiment Objectives

- Learn how to subscribe to and process image data (RGB and Depth) in ROS.
- Master basic image processing techniques using OpenCV: Color Space Conversion (HSV), Object Detection, and Contour Detection.
- Understand how to calculate 3D coordinates of objects using Depth maps and Camera Intrinsics.
- Learn to transform coordinates from the Camera Frame to the Robot Base Frame using TF.
- Implement a basic Visual Servoing control loop to follow a target.
- Visualize 3D Point Clouds using Open3D.

2. Setup and Compilation

First, ensure your environment is set up and the package is compiled.

```
# 1. Navigate to your workspace
cd ~/catkin_ws

# 2. Build the package
# We whitelist only this package to speed up compilation
catkin_make -DCATKIN_WHITELIST_PACKAGES="lab2_perception"

# 3. Source the setup script to register the new package
source devel/setup.bash

# 4. Grant execution permissions
chmod +x ~/catkin_ws/src/lab2_perception/tools/*.py
chmod +x ~/catkin_ws/src/lab2_perception/scripts/*.py
```

3. Step-by-Step Experiment

Part 1: Start simulation

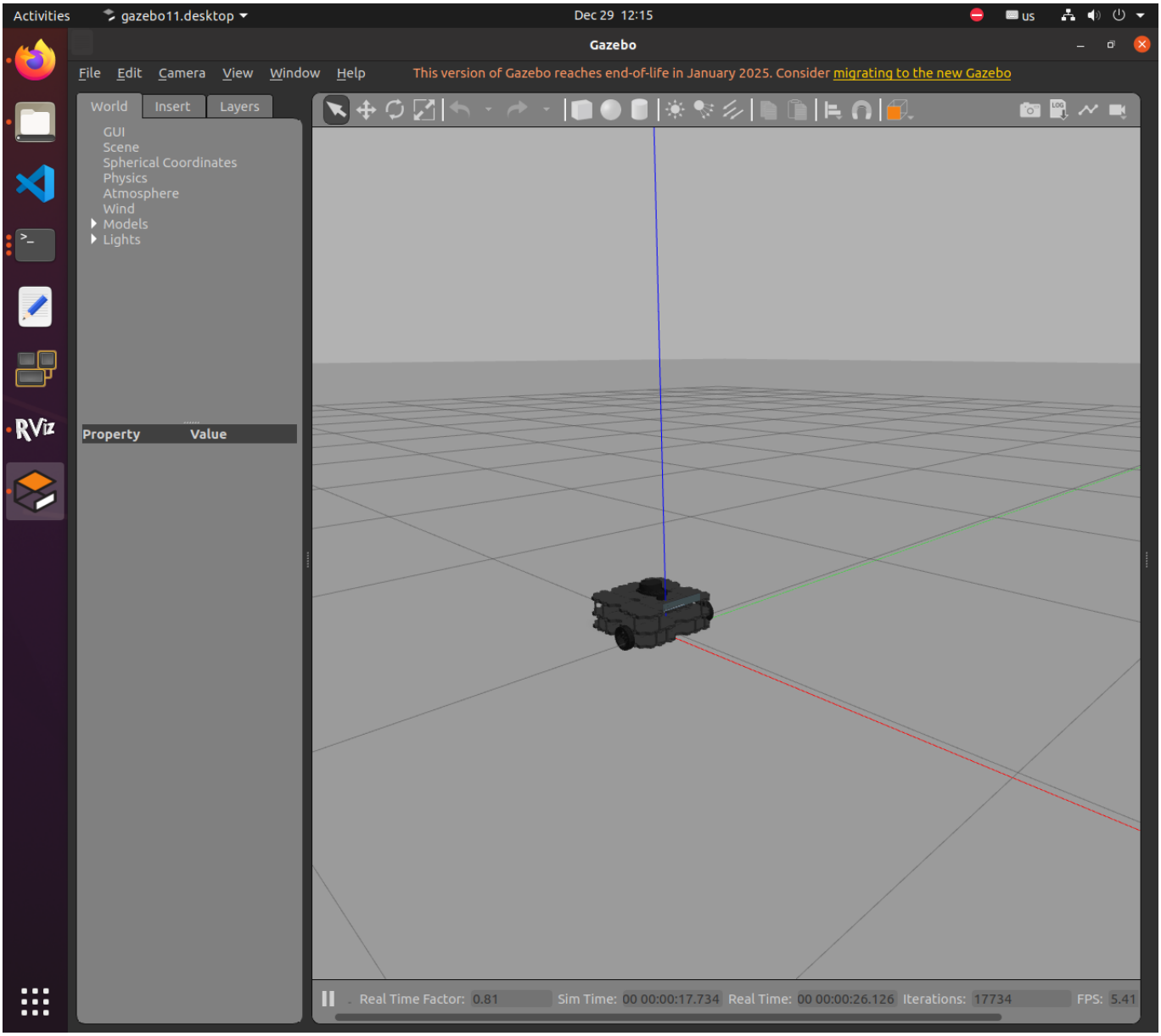
Before diving into complex processing, let's verify we can receive images from the camera.

1.1 Run Simulation

```
# Terminal 1: Start ROS Master (if not running)
roscore
```

```
# Terminal 2: Start a robot simulation
source ~/catkin_ws/devel/setup.bash
```

```
export TURTLEBOT3_MODEL=waffle
roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
```

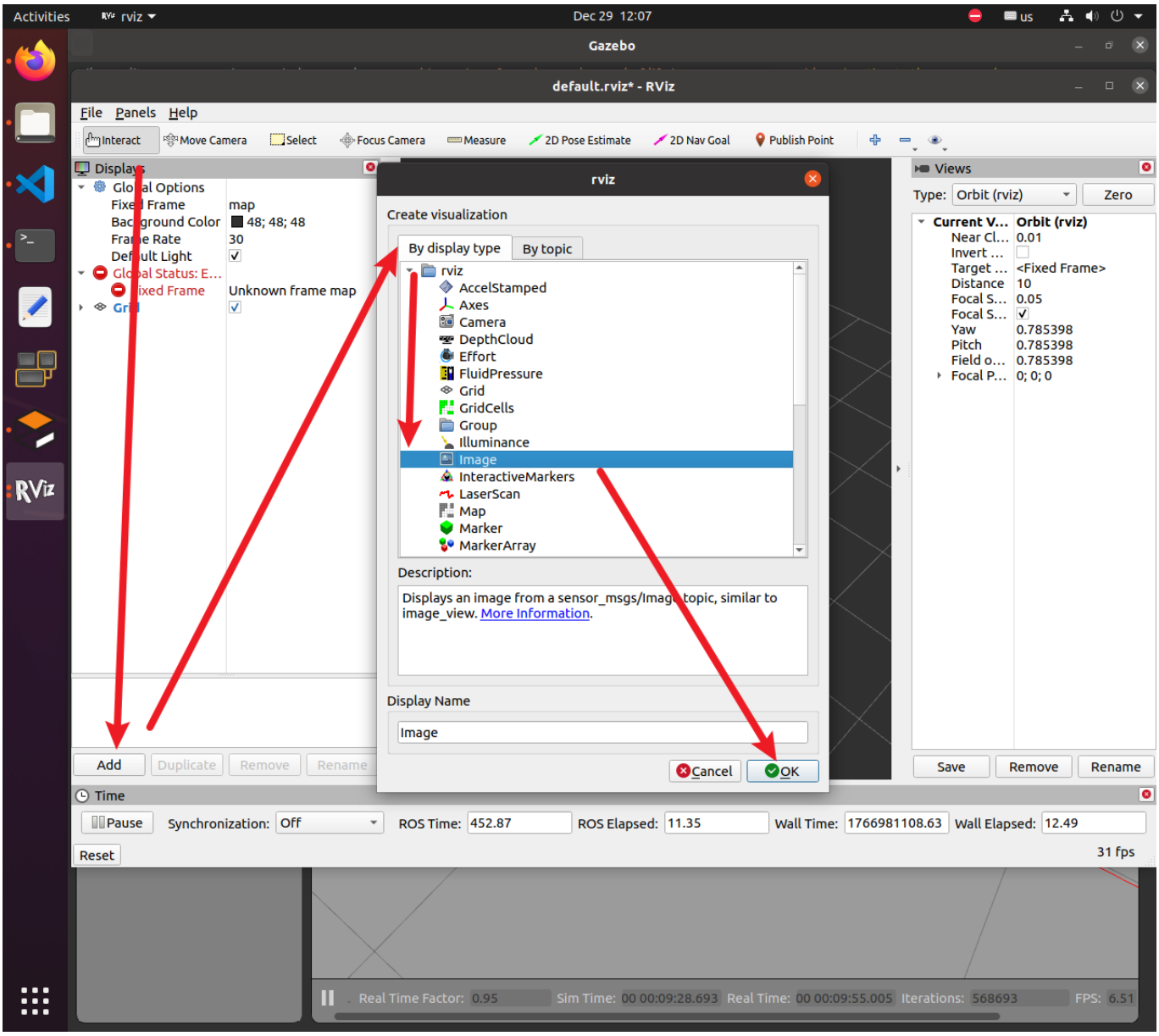


1.2 Run rviz to visualize rgb and depth image

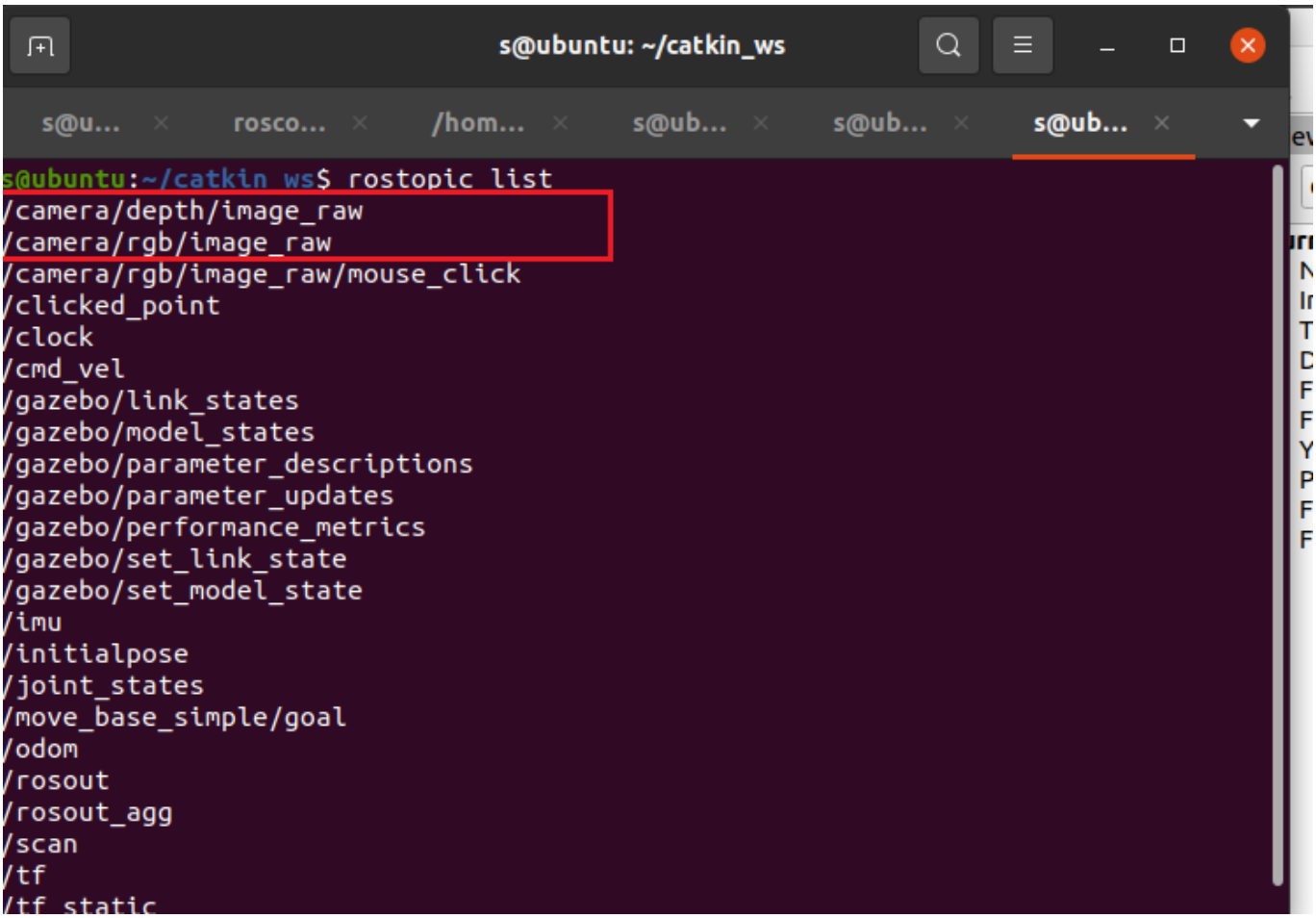
Open RViz and add image displays for both RGB and depth topics.

```
# Terminal 3: Run RViz
rviz
```

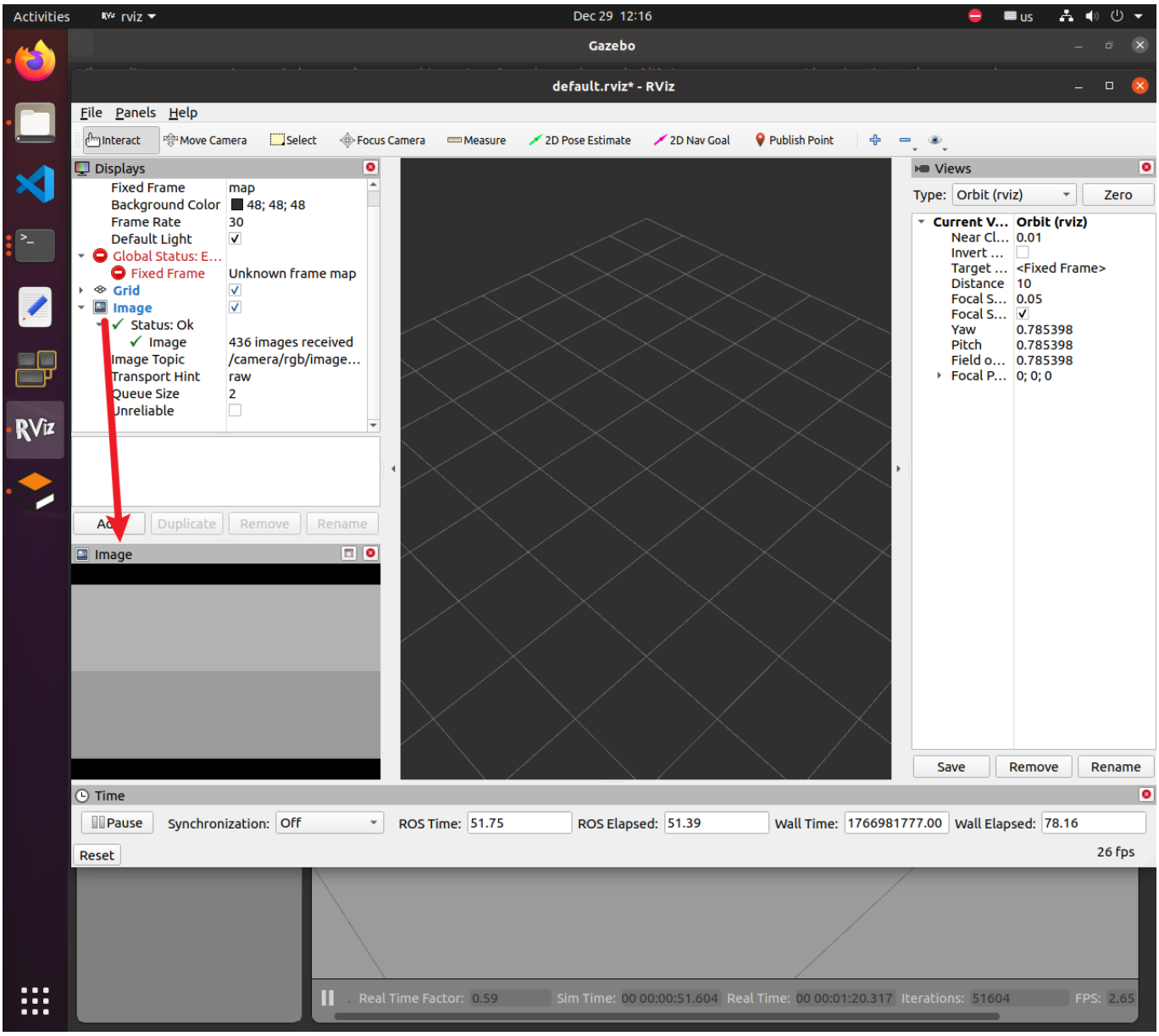
In RViz: Click Add → By display type → Image



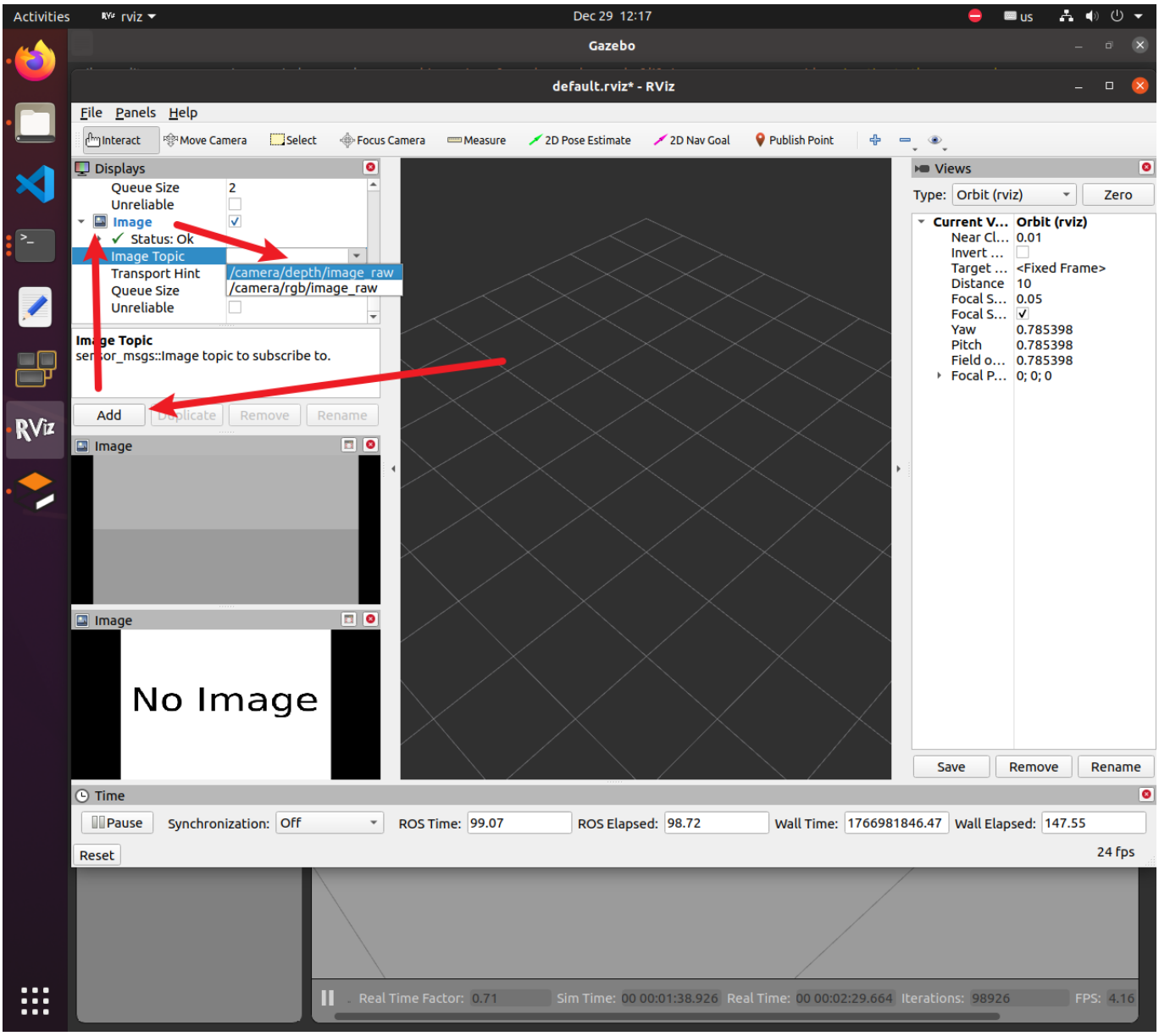
Use `rostopic list` to check the camera topic name.

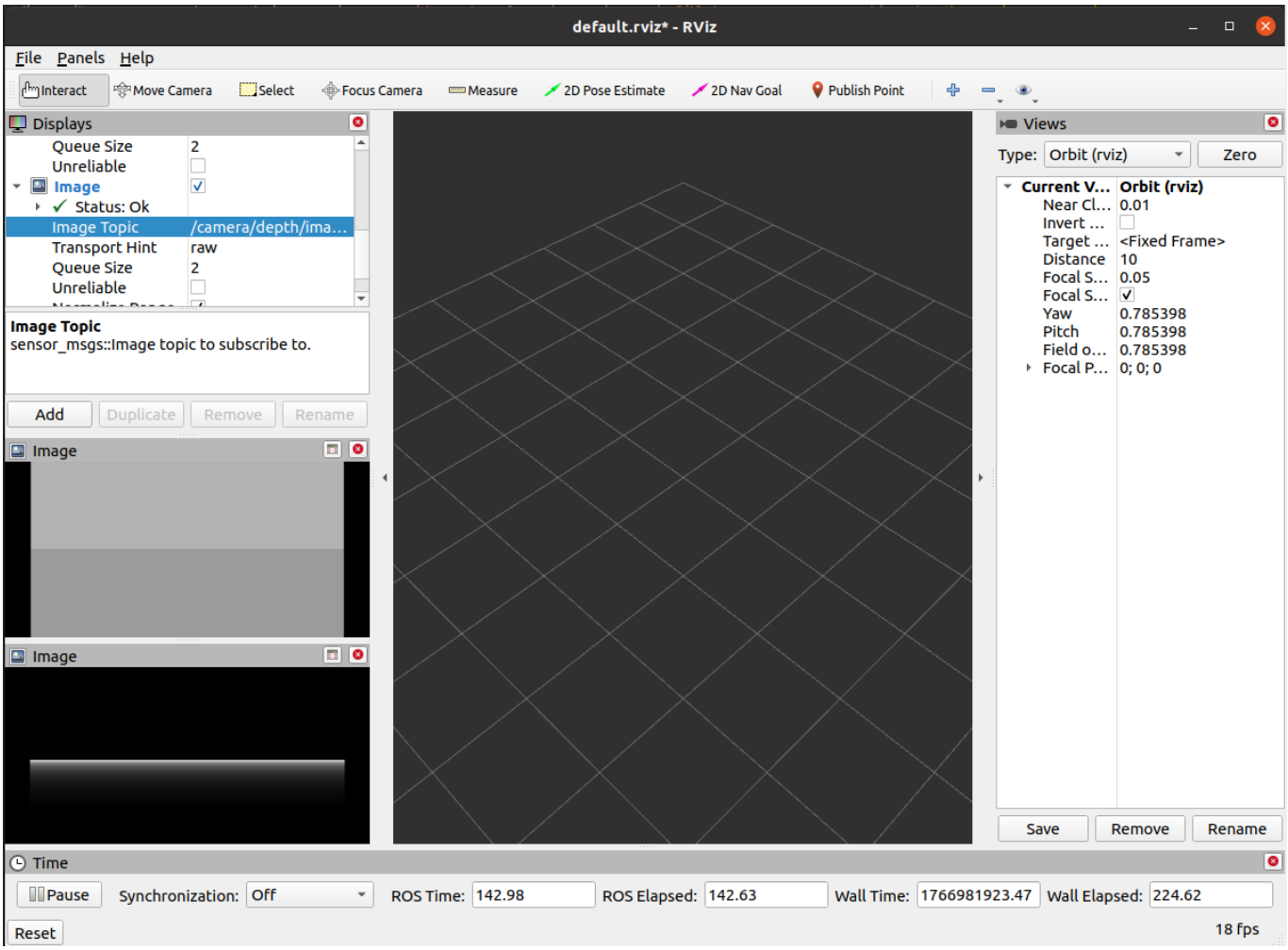


Set the topic to /camera/rgb/image_raw



Add another Image display and set the topic to /camera/depth/image_raw Adjust the Fixed Frame (e.g., camera_link or base_link) if needed This allows you to visually inspect both the RGB stream and the depth map in real time.

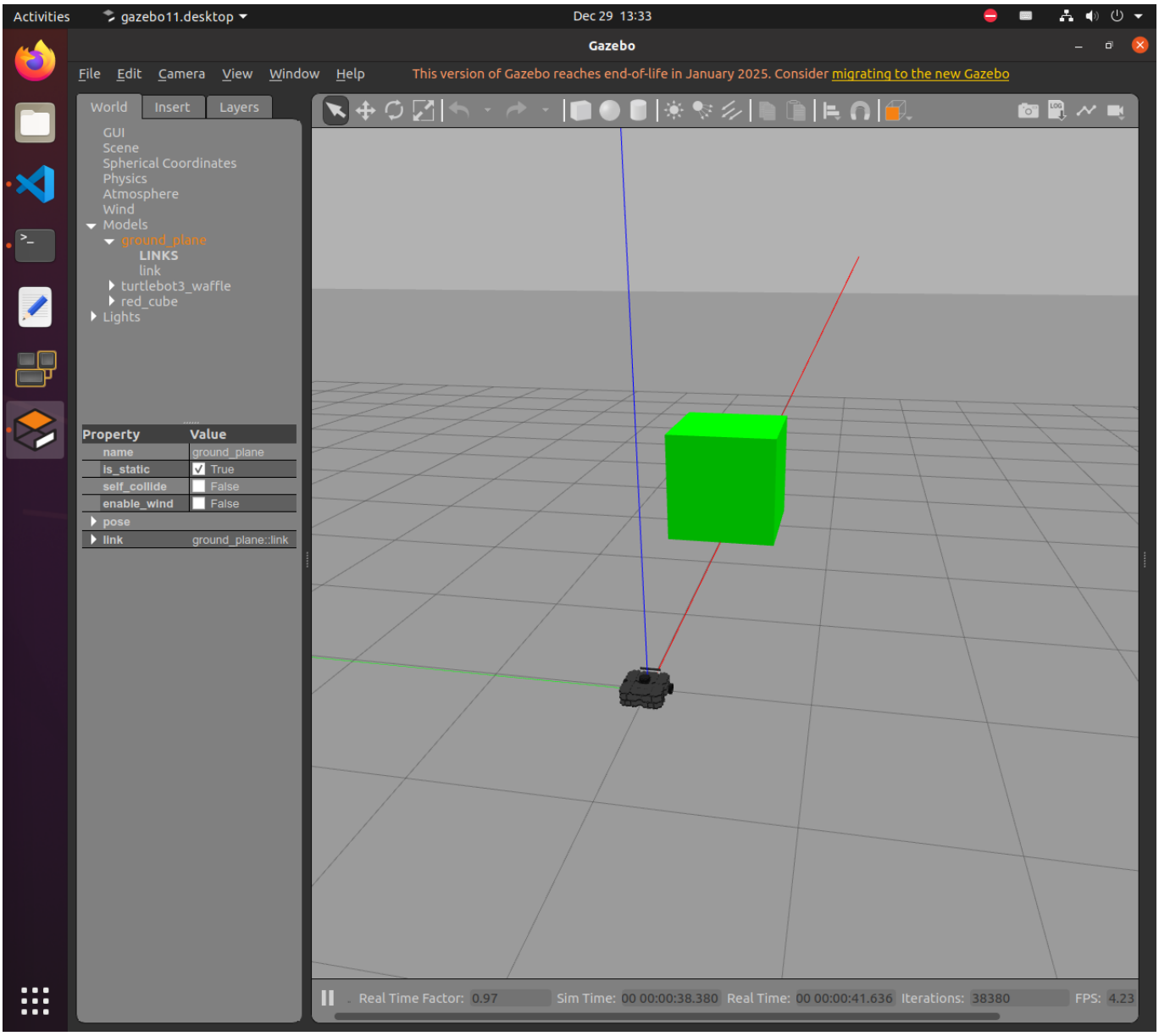

Activities
rviz
Dec 29 12:17
Gazebo
default.rviz* - RViz
File Panels Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Displays
Queue Size 2
Unreliable
Image
Status: Ok
Image Topic
Transport Hint
Queue Size
Unreliable
/camera/depth/image_raw
/camera/rgb/image_raw
Image Topic
sensor_msgs::Image topic to subscribe to.
Add
Duplicate
Remove
Rename
Image
Image
No Image
Views
Type: Orbit (rviz)
Zero
Current V... Orbit (rviz)
Near Cl... 0.01
Invert ...
Target ... <Fixed Frame>
Distance 10
Focal S... 0.05
Focal S...
Yaw 0.785398
Pitch 0.785398
Field o... 0.785398
Focal P... 0; 0; 0
Save
Remove
Rename
Time
Pause
Synchronization: Off
ROS Time: 99.07
ROS Elapsed: 98.72
Wall Time: 1766981846.47
Wall Elapsed: 147.55
Reset
24 fps
Real Time Factor: 0.71
Sim Time: 00 00:01:38.926
Real Time: 00 00:02:29.664
Iterations: 98926
FPS: 4.16



1.3 Spawn an obstacle/object for detection

Set a green cube for object detection:

```
source ~/catkin_ws/devel/setup.bash
rosrun gazebo_ros spawn_model \
  -file ~/catkin_ws/src/lab2_perception/demo_images/green_cube.sdf \
  -sdf \
  -model green_cube
```



Part 2: Image Processing Tools (HSV Tuning)

To detect a specific object (like a green block), we need to find the correct HSV (Hue, Saturation, Value) thresholds. We have provided tools to help you with this.

2.1 Generate Test Images First, let's generate some sample images to test our algorithm.

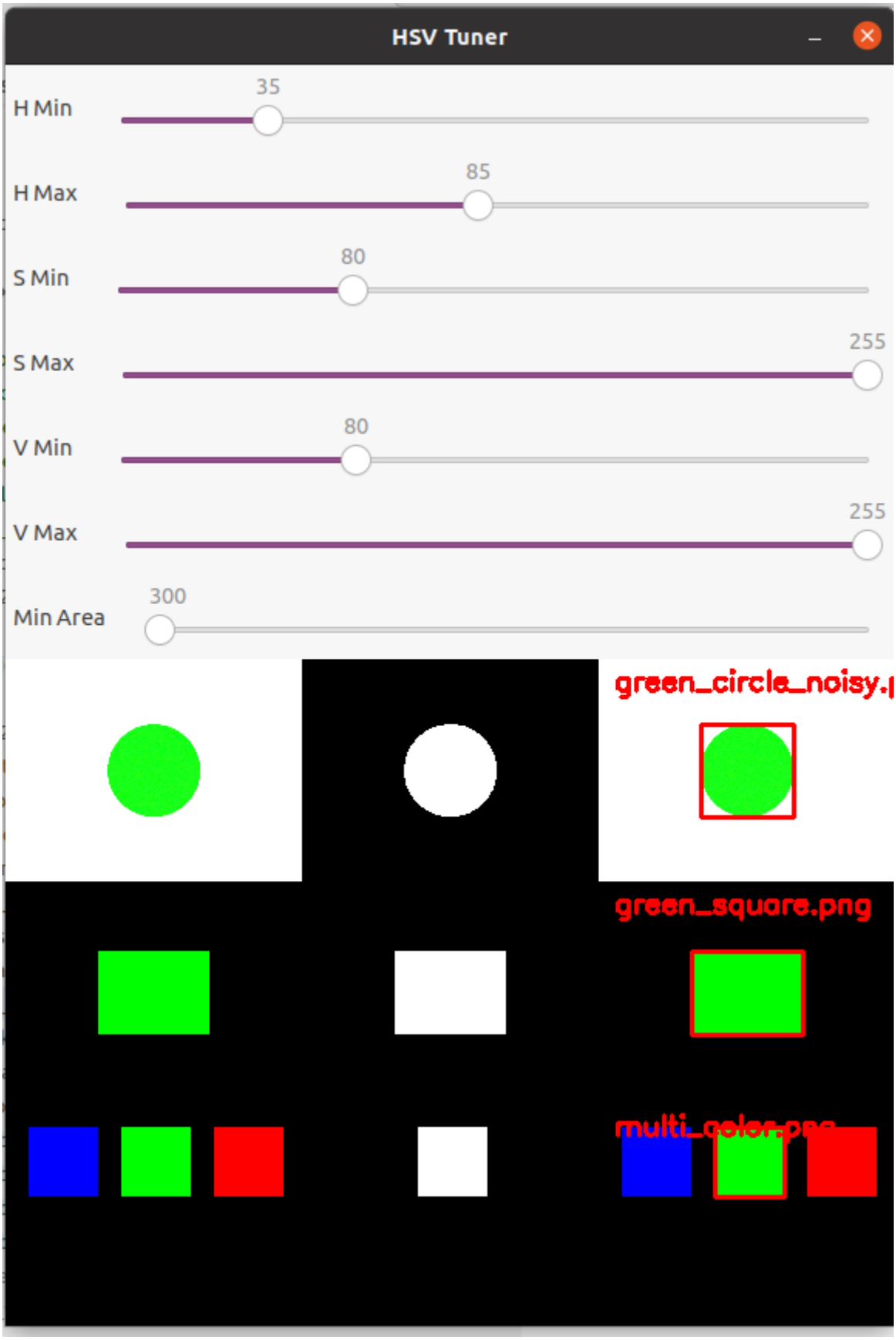
```
# Generate dummy images in src/lab2_perception/demo_images/
rosrun lab2_perception generate_images.py
```

2.2 Tune HSV Thresholds Use the tuner tool to find the best values to isolate the green color.

```
# Run the tuner tool
rosrun lab2_perception hsv_tuner.py
```

Instructions:

- Adjust `H Min`, `S Min`, `V Min`, etc., until only the desired object is white in the middle mask window.
- `Esc` will exit the program.
- **Record these values**. You will need them for the main perception node. (Default values for green are already set in the code)



2.3 Get camera param

```
rostopic echo /camera/rgb/camera_info
```

```
s@ubuntu:~$ rostopic echo /camera/rgb/camera_info
header:
  seq: 11
  stamp:
    secs: 160
    nsecs: 752000000
  frame_id: "camera_rgb_optical_frame"
height: 1080
width: 1920
distortion_model: "plumb_bob"
D: [0.0, 0.0, 0.0, 0.0, 0.0]
K: [1206.8897719532354, 0.0, 960.5, 0.0, 1206.8897719532354, 540.5, 0.0,
 0.0, 1.0]
R: [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0]
P: [1206.8897719532354, 0.0, 960.5, -84.48228403672648, 0.0, 1206.889771
9532354, 540.5, 0.0, 0.0, 0.0, 1.0, 0.0]
binning_x: 0
binning_y: 0
roi:
  x_offset: 0
  y_offset: 0
  height: 0
  width: 0
  do_rectify: False
---
```

This parameter is critical for calculating the correct object coordinates.

This parameter will be loaded autonomously in the perception node.

Part 3: Integrated Perception & Visual Servoing

This is the core of the experiment. We will run the TurtleBot3 simulation in Gazebo. The robot will use its camera to detect a green object, calculate its 3D position, and drive towards it.

3.1 Launch Simulation and Perception Node

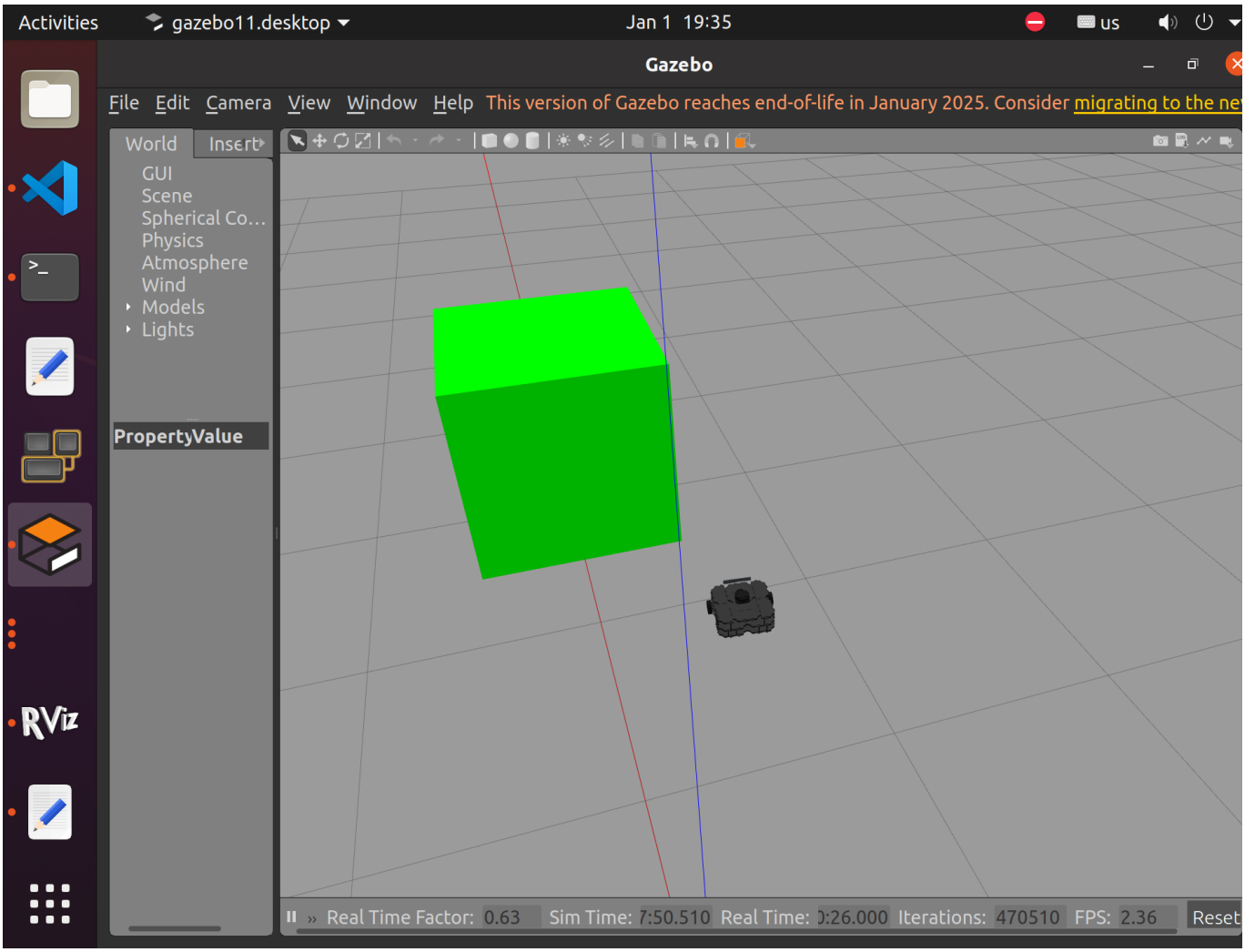
We have prepared a launch file that starts:

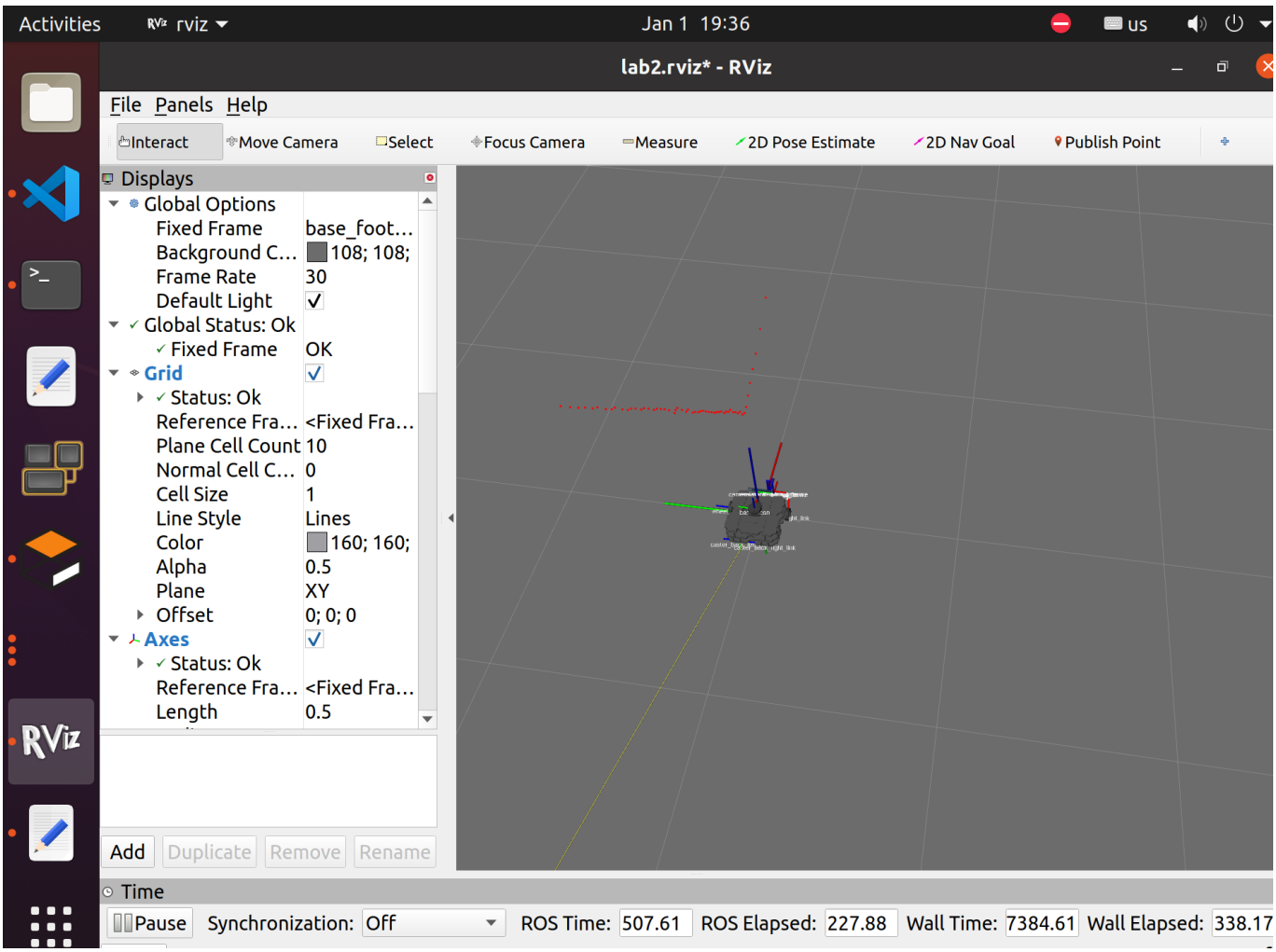
- Gazebo with TurtleBot3.
- The `perception_node` (performs detection and control).
- RViz for visualization.

```
# Close previous terminals if needed, then run:
# Terminal 1 start simulation
source ~/catkin_ws/devel/setup.bash
```

```
export TURTLEBOT3_MODEL=waffle
roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
```

```
# Terminal 2 start perception node
source ~/catkin_ws/devel/setup.bash
roslaunch lab2_perception lab2.launch
```


Activities
gazebo11.desktop
Jan 1 19:35
us
Gazebo
File Edit Camera View Window Help This version of Gazebo reaches end-of-life in January 2025. Consider migrating to the ne
World
Insert
GUI
Scene
Spherical Co...
Physics
Atmosphere
Wind
Models
Lights
PropertyValue
Real Time Factor: 0.63
Sim Time: 7:50.510
Real Time: 0:26.000
Iterations: 470510
FPS: 2.36
Reset


Activities
rviz
Jan 1 19:36
us
lab2.rviz* - RViz
File Panels Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Displays
Global Options
Fixed Frame base_foot...
Background C... 108; 108;
Frame Rate 30
Default Light
Global Status: Ok
Fixed Frame OK
Grid
Status: Ok
Reference Fra... <Fixed Fra...
Plane Cell Count 10
Normal Cell C... 0
Cell Size 1
Line Style Lines
Color 160; 160;
Alpha 0.5
Plane XY
Offset 0; 0; 0
Axes
Status: Ok
Reference Fra... <Fixed Fra...
Length 0.5
Add
Duplicate
Remove
Rename
Time
Pause
Synchronization: Off
ROS Time: 507.61
ROS Elapsed: 227.88
Wall Time: 7384.61
Wall Elapsed: 338.17

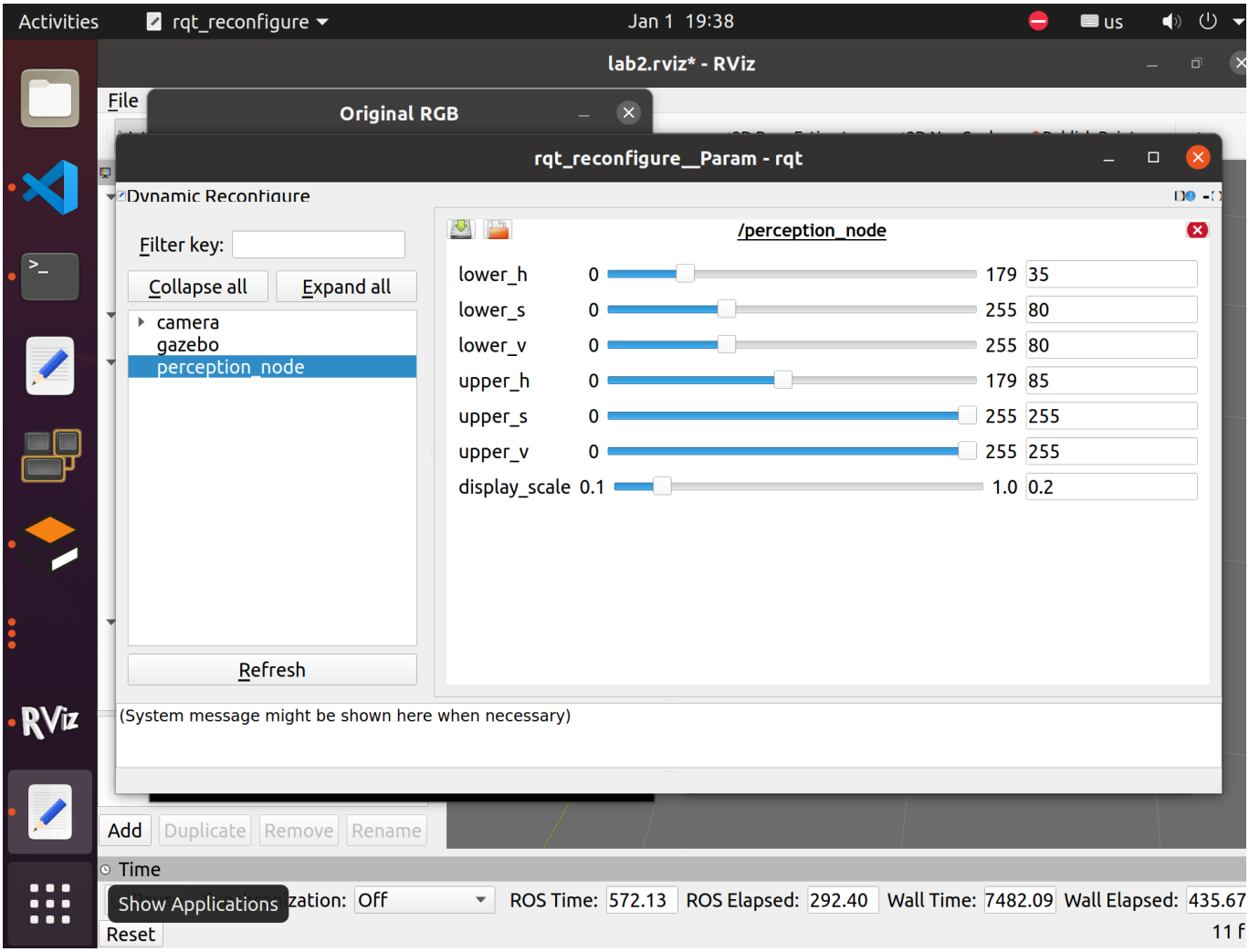
Reset **Left-Click:** Rotate. **Middle-Click:** Move X/Y. **Right-Click:**: Move Z. **Shift**: More options. 11 f

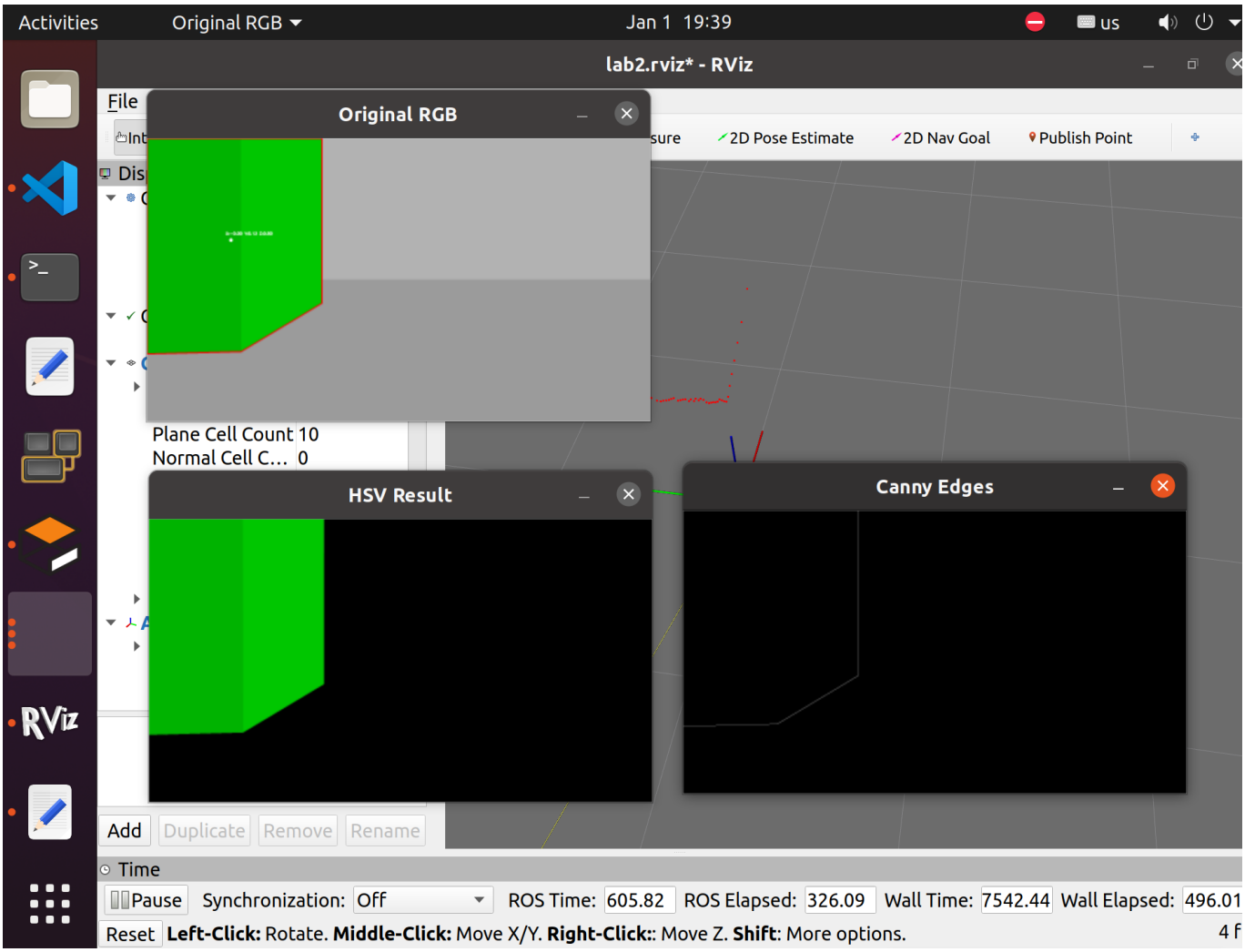
To stop the robot immediately, press Ctrl+C in the terminal running the perception node, or lift the robot (if real) / reset simulation (if Gazebo).

Use the HSV values (H_min, S_min, V_min, etc.) you recorded in Part 2 to update the parameters in the rqt_reconfigure window.

```
rosrun rqt_reconfigure rqt_reconfigure
```

Set the threshold like this:





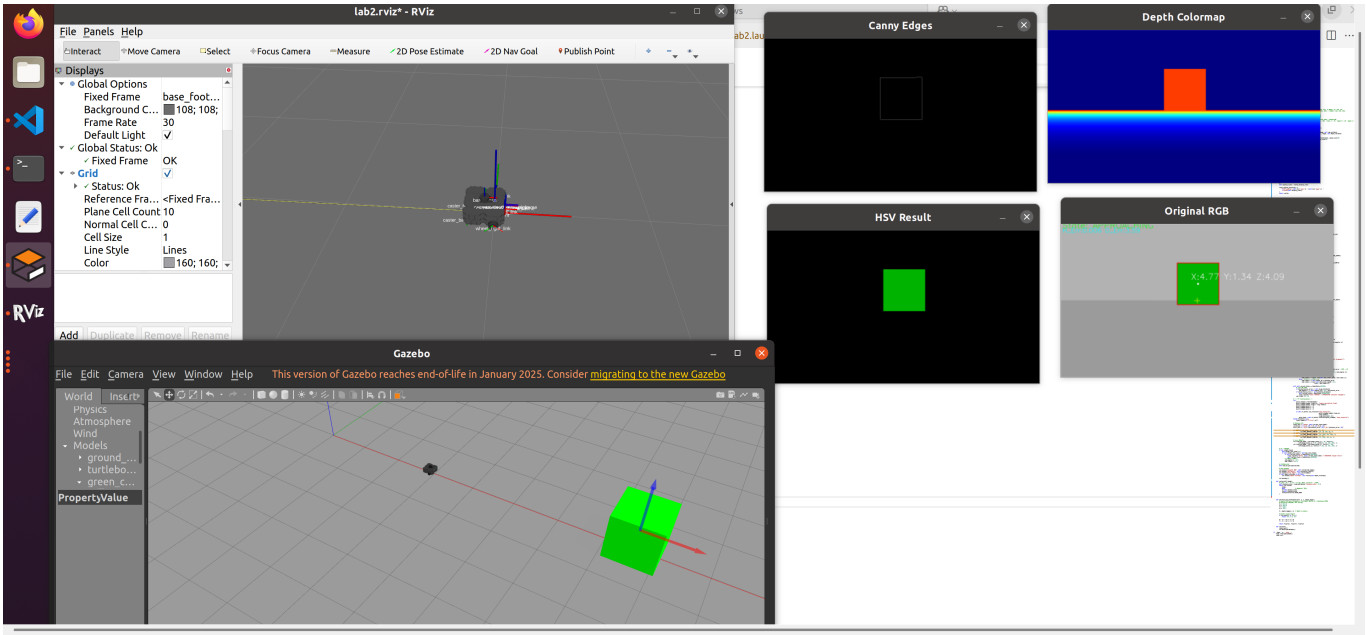
3.2 Data Visualization in Terminal

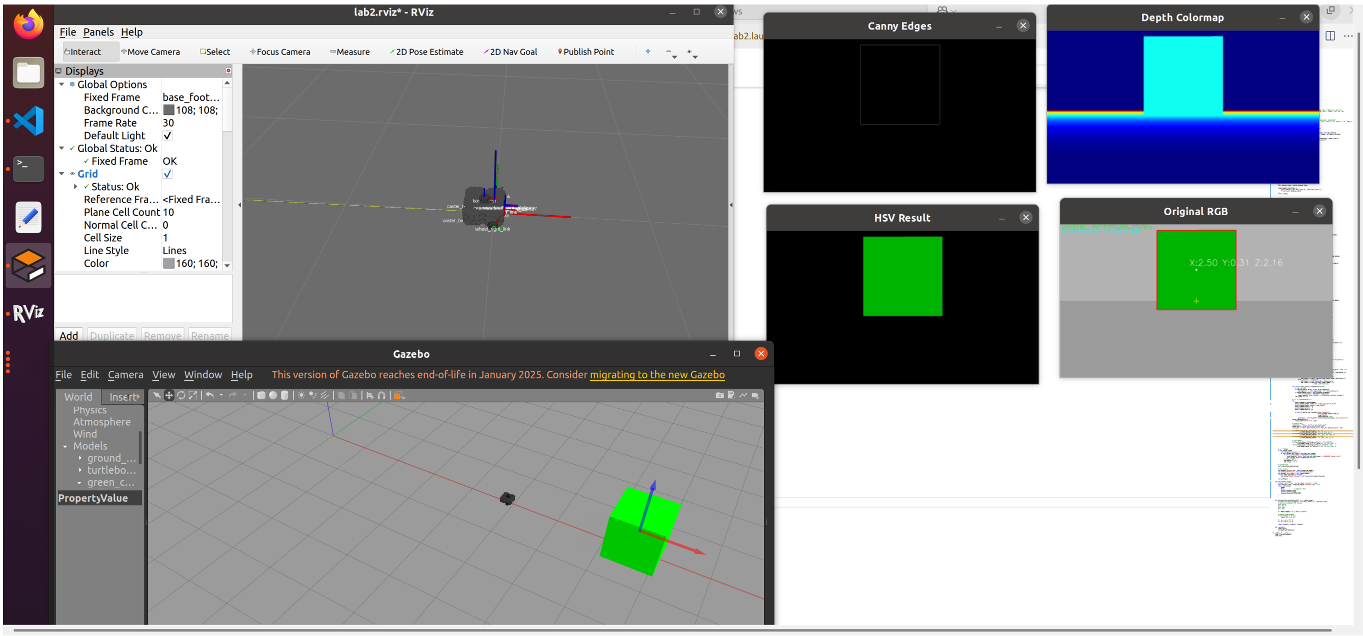
To see the custom messages being published:

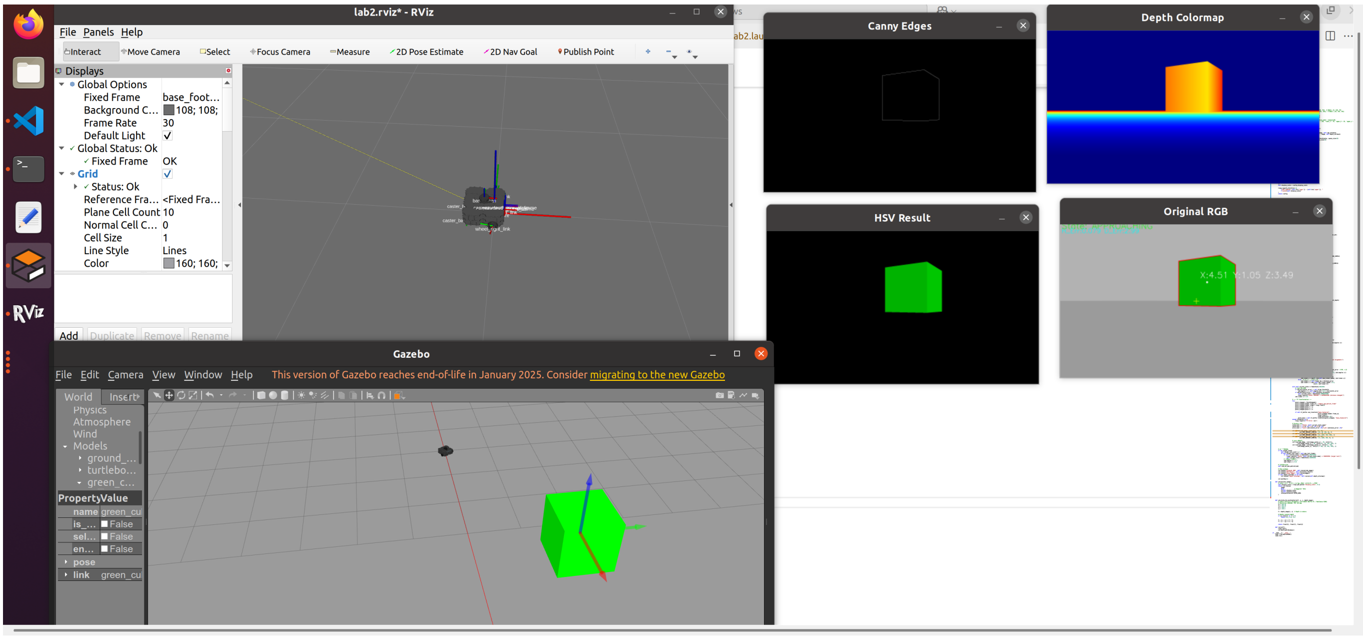
```
# Terminal 3
source ~/catkin_ws/devel/setup.bash

# Check the detected object coordinates
rostopic echo /detected_object
```

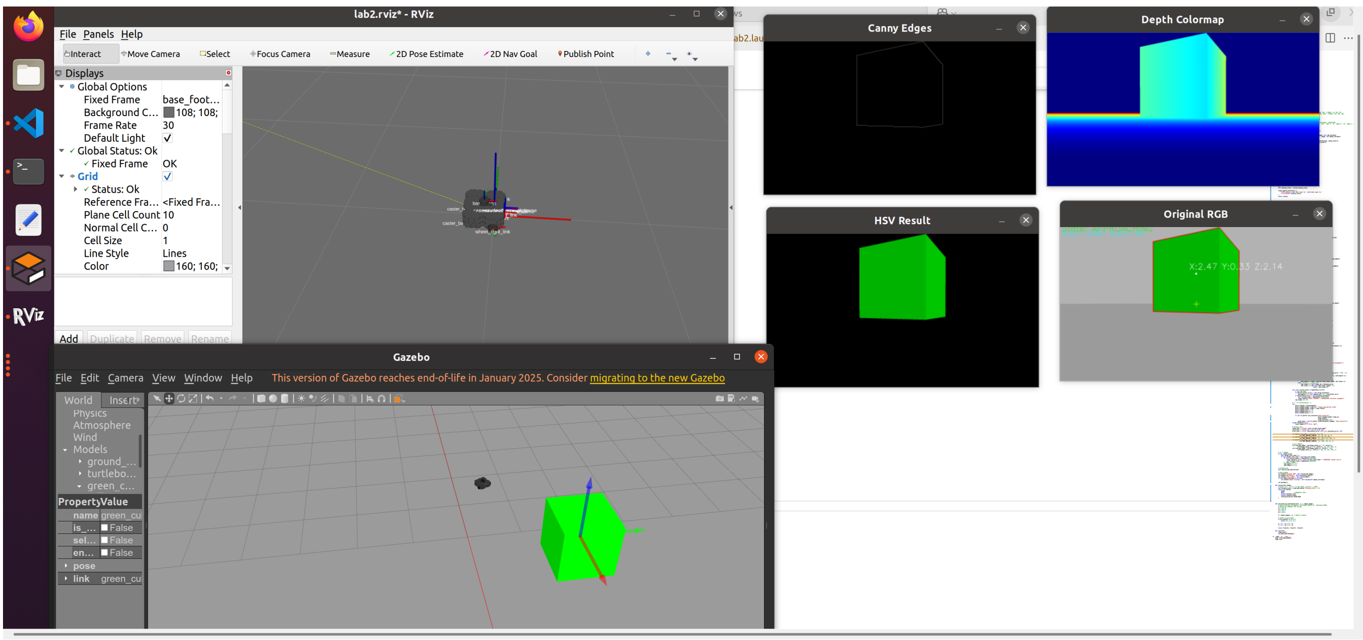
```
s@ubuntu:~$ source ~/catkin_ws/devel/setup.bash
s@ubuntu:~$ rostopic echo /detected_object
x: -0.0004510203143581748
y: 0.13305099308490753
z: 0.49995601177215576
---
x: -0.0004510196740739047 5
y: 0.133050799369812
z: 0.4999552965164184 6
---
x: -0.0004510199360083788 6
y: 0.13305087387561798
z: 0.4999555945396423 3
---
x: -0.0004510192957241088
y: 0.13305069506168365
z: 0.49995487928390503
---
x: -0.0004510189755819738
y: 0.133050590753555 3
z: 0.499954521656036 4
```

Move the cube, and you will observe the robot moving towards it.

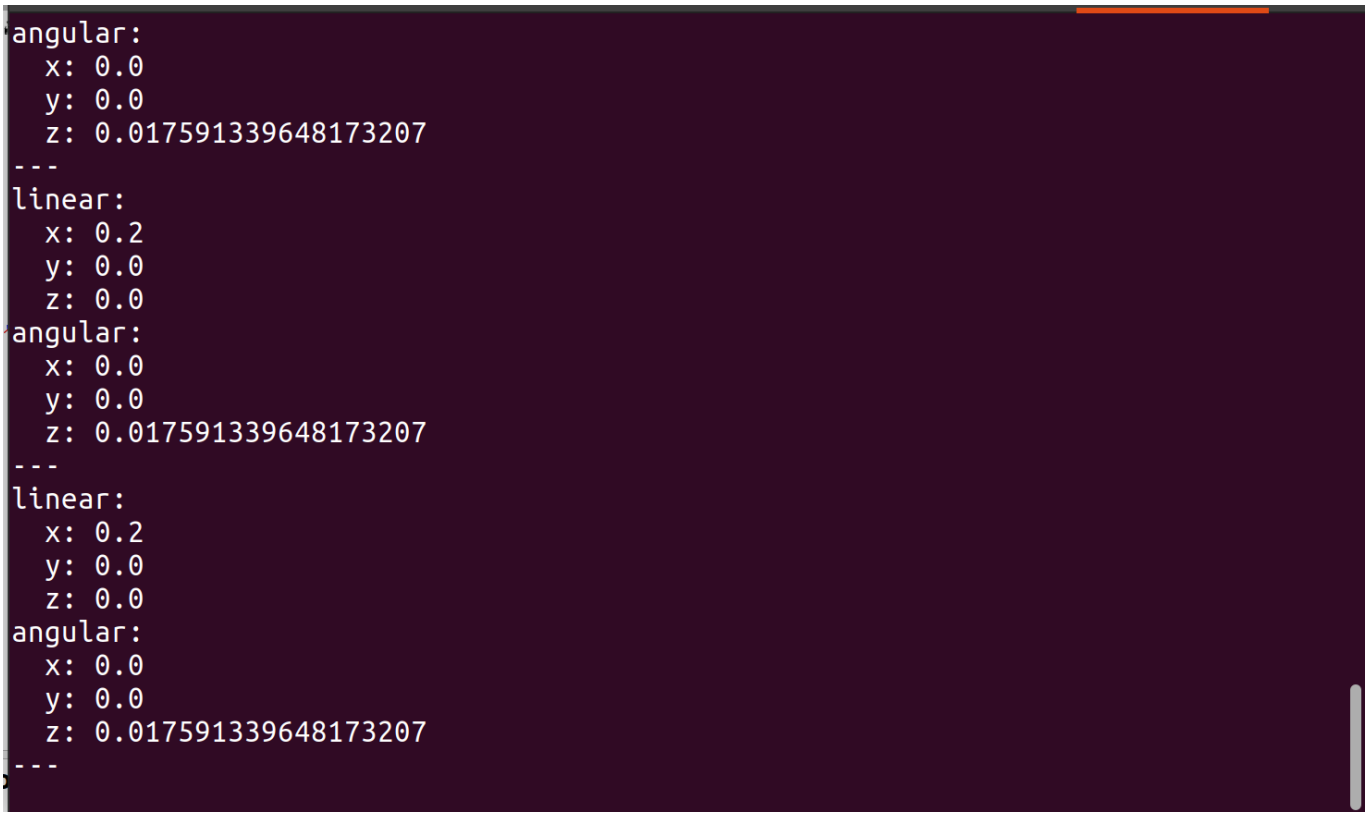



lab2.rviz* - RViz
Gazebo
This version of Gazebo reaches end-of-life in January 2025. Consider migrating to the new Gazebo
Canny Edges
Depth Colormap
HSV Result
Original RGB
X:2.50 Y:0.31 Z:2.16


lab2.rviz* - RViz
Gazebo
This version of Gazebo reaches end-of-life in January 2025. Consider migrating to the new Gazebo
Canny Edges
Depth Colormap
HSV Result
Original RGB
X:4.51 Y:1.05 Z:3.49


lab2.rviz* - RViz
Gazebo
This version of Gazebo reaches end-of-life in January 2025. Consider migrating to the new Gazebo
Canny Edges
Depth Colormap
HSV Result
Original RGB
X:2.47 Y:0.33 Z:2.14

```
# Check the velocity commands being sent to the robot
rostopic echo /cmd_vel
```



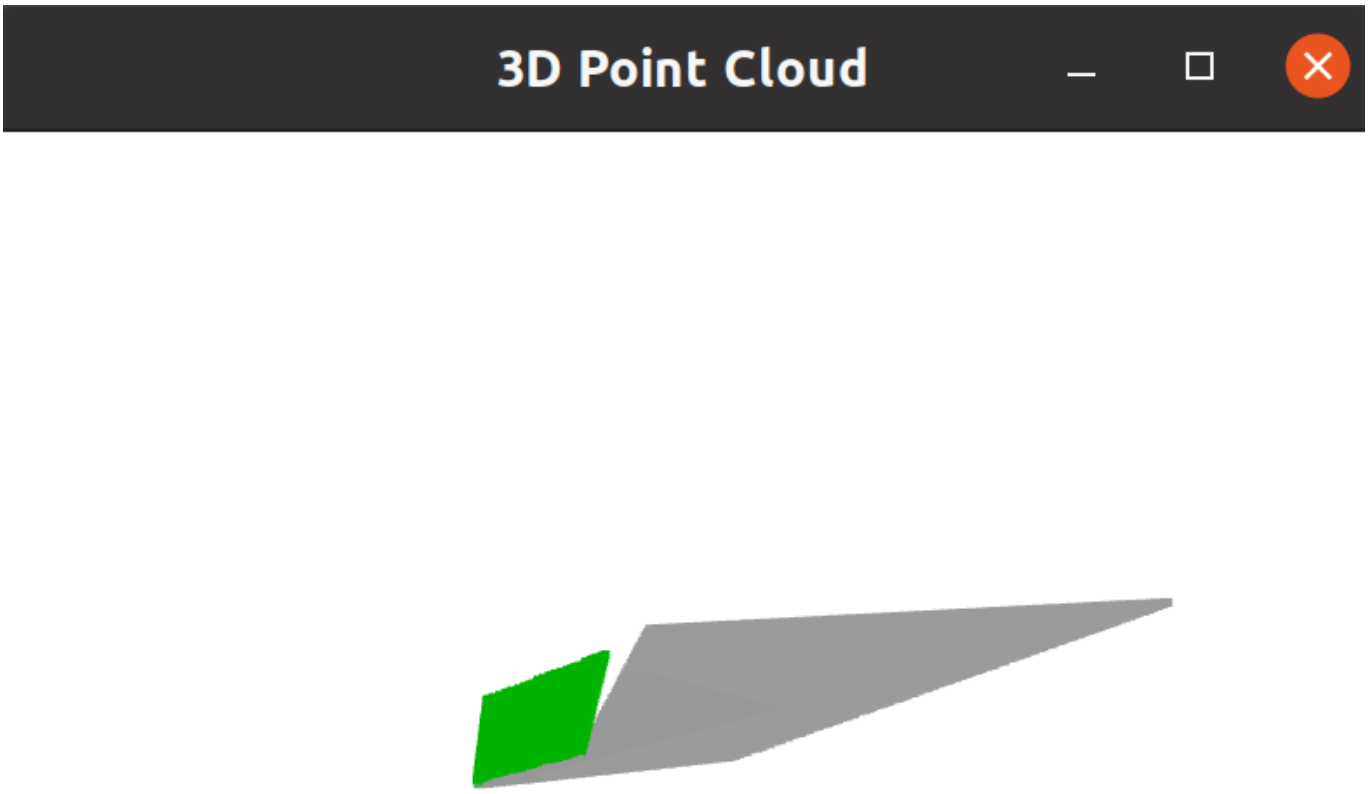
Part 4: 3D Point Cloud Visualization

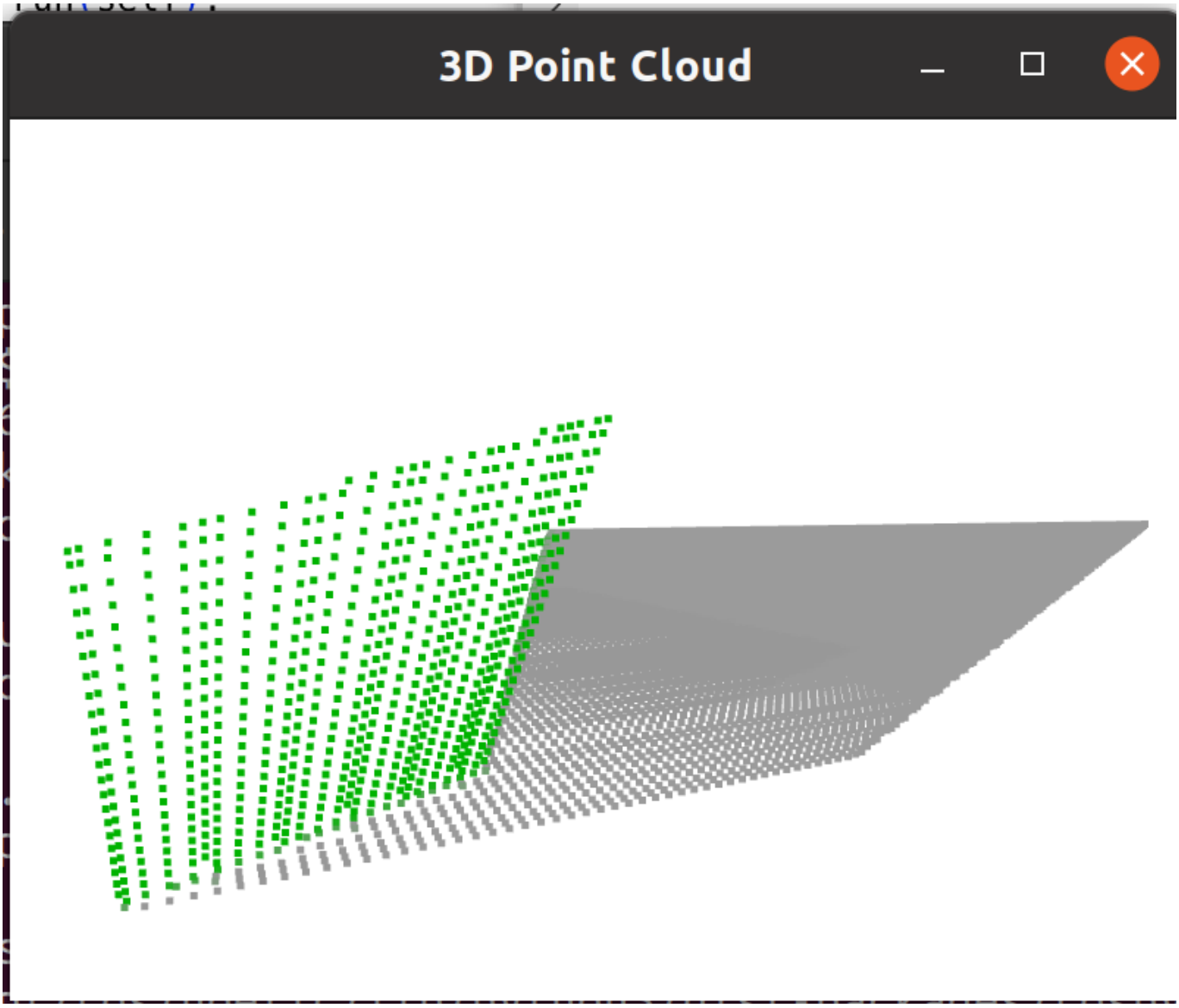
While RViz is great for ROS integration, Open3D provides powerful Python APIs for programmatic point cloud processing and visualization.

4.1 Run the Visualizer While the simulation (from Part 3) is running:

```
# Open a new terminal
source ~/catkin_ws/devel/setup.bash

# Run the Open3D visualizer
rosrun lab2_perception pointcloud_visualizer.py
```

- A new window "3D Point Cloud" will appear.
- **Left Click + Drag**: Rotate the view.
- **Scroll Wheel**: Zoom in/out.
- You should see the 3D reconstruction of the scene in front of the robot.


3D Point Cloud



Conclusion:

In this experiment, we successfully implemented a complete ROS perception pipeline for a mobile robot.

Data Acquisition: We learned how to subscribe to and visualize RGB and Depth data streams from the TurtleBot3 camera using RViz.

Image Processing: By utilizing the HSV color space and OpenCV tools, we effectively isolated specific objects (green cube) from the background and tuned thresholds for robust detection.

3D Localization & Control: We bridged the gap between 2D image pixels and 3D world coordinates using camera intrinsics and depth maps. Through TF transformations, we converted these coordinates into the robot's base frame to implement a closed-loop visual servoing controller, allowing the robot to autonomously track the target.

Visualization: Finally, we explored 3D scene reconstruction using Open3D, verifying the depth data fidelity.

This lab demonstrates the fundamental workflow of robot perception: Sense (Camera) → Perceive (OpenCV/HSV) → Plan/Act (Visual Servoing). These skills form the foundation for more advanced tasks such as SLAM, object manipulation, and semantic scene understanding.