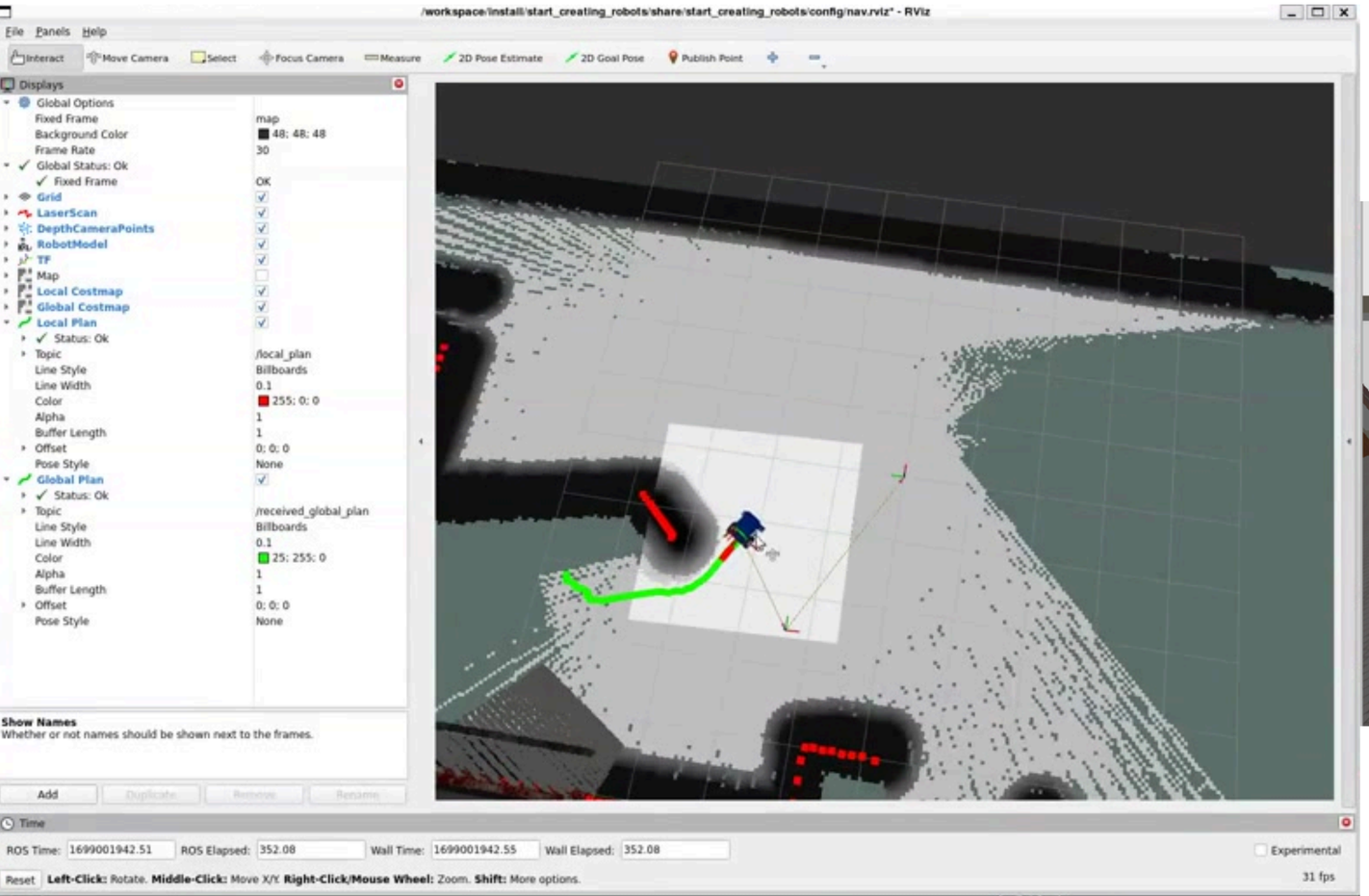
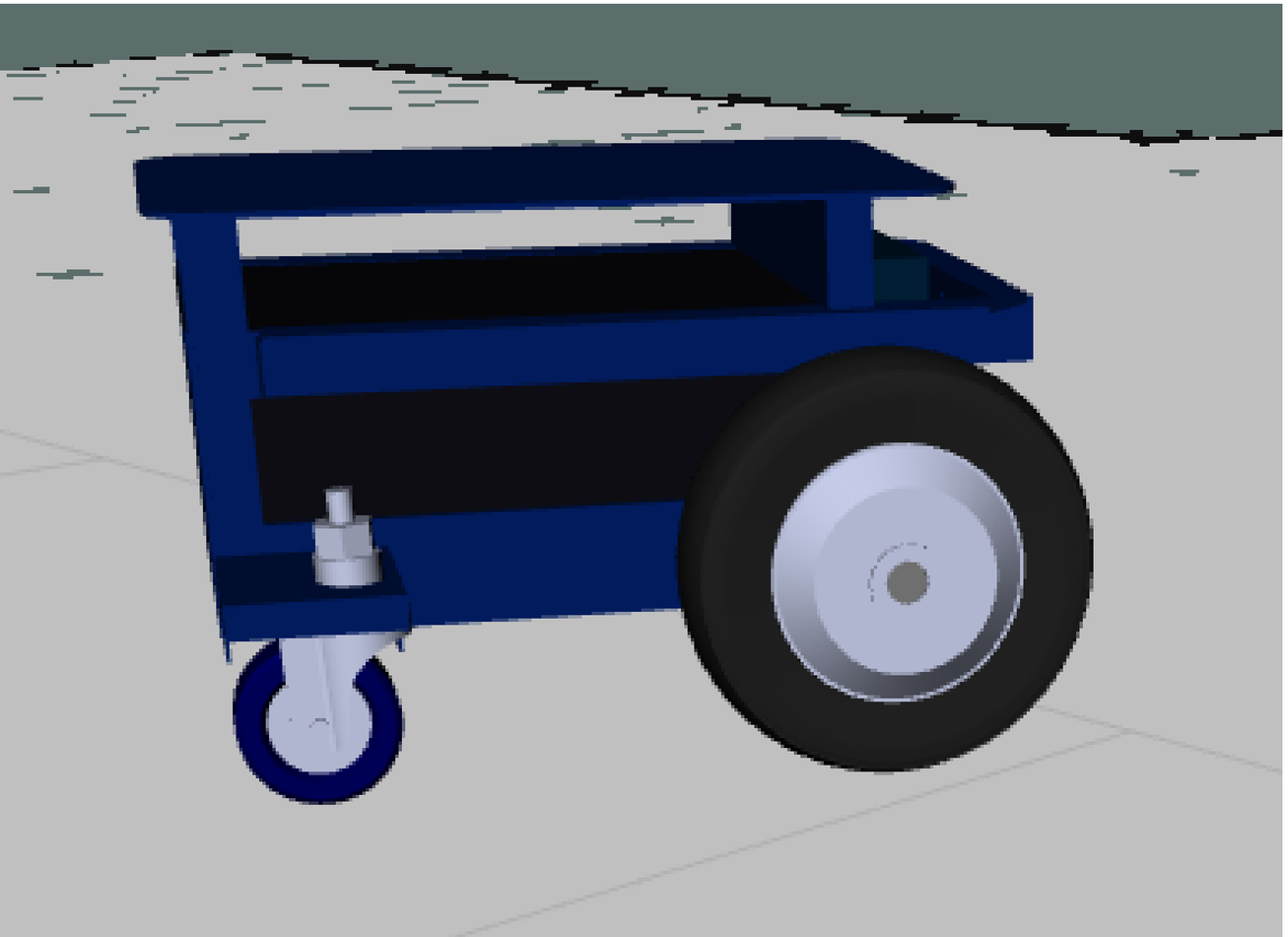

ROS Workshop

Todays Goal:
Autonomous Navigation



Krytn: Differential Drive Robot



2 Drive Wheels
2 Caster Wheels
2D Lidar

Robotics Operating System: ROS 2 “Humble”

Provides a framework for robotic systems integration.
Provides useful packages for:

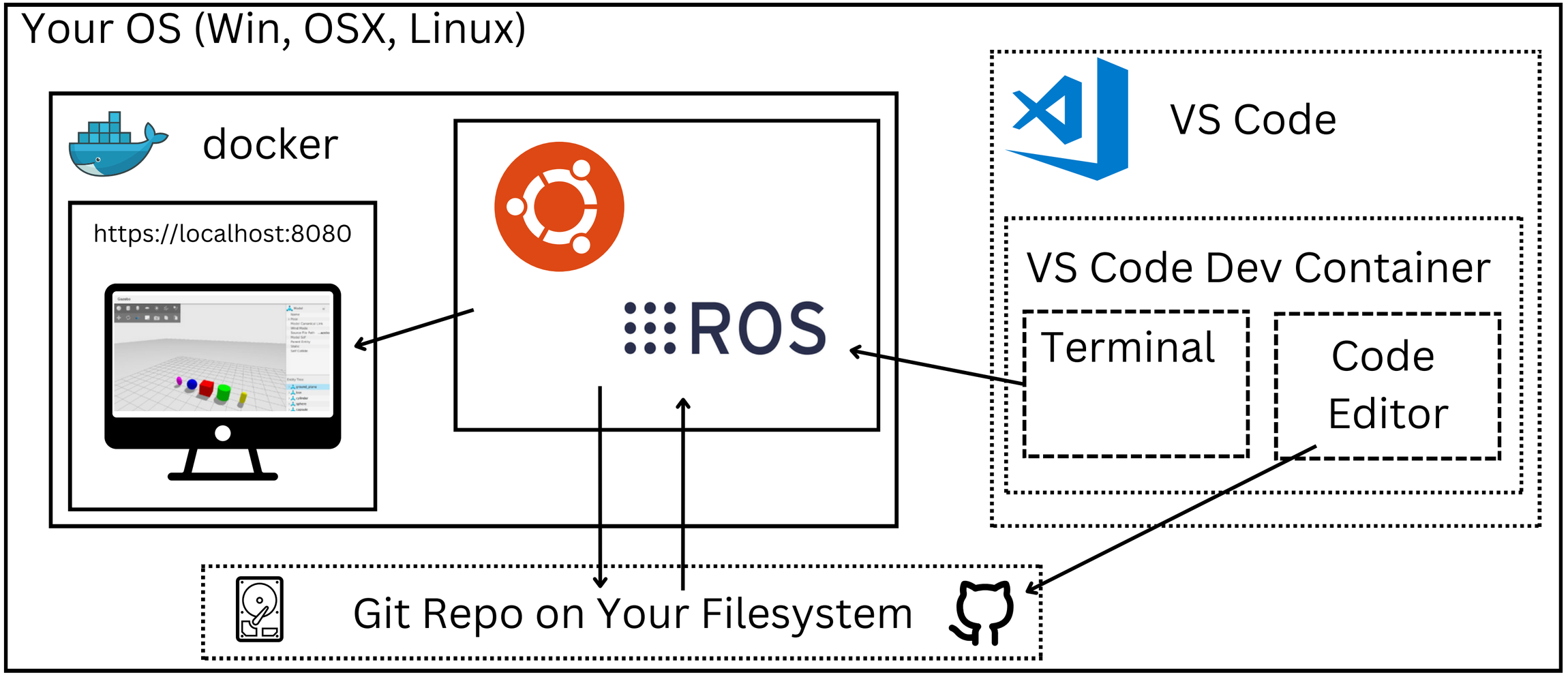
- teleoperation (rqt)
- motor control (ros2_control)
- mapping (slam_toolbox)
- navigation (nav2)

Simulation: Gazebo

Provides a physics simulation.
Plugins for sensors and other things

Todays System

Most robots just run ROS on Ubuntu, but this system runs on every computer!



Todays Plan

Step 1: Install Docker, VS Code

Step 2: Run Gazebo Simulator

Step 3: Run Teleoperation

Step 4: Mapping

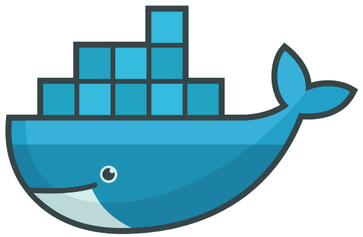
Step 5: Navigation


Download Tools
Step 1

Step 1 - Download

WSL 2 (WSL)

Git

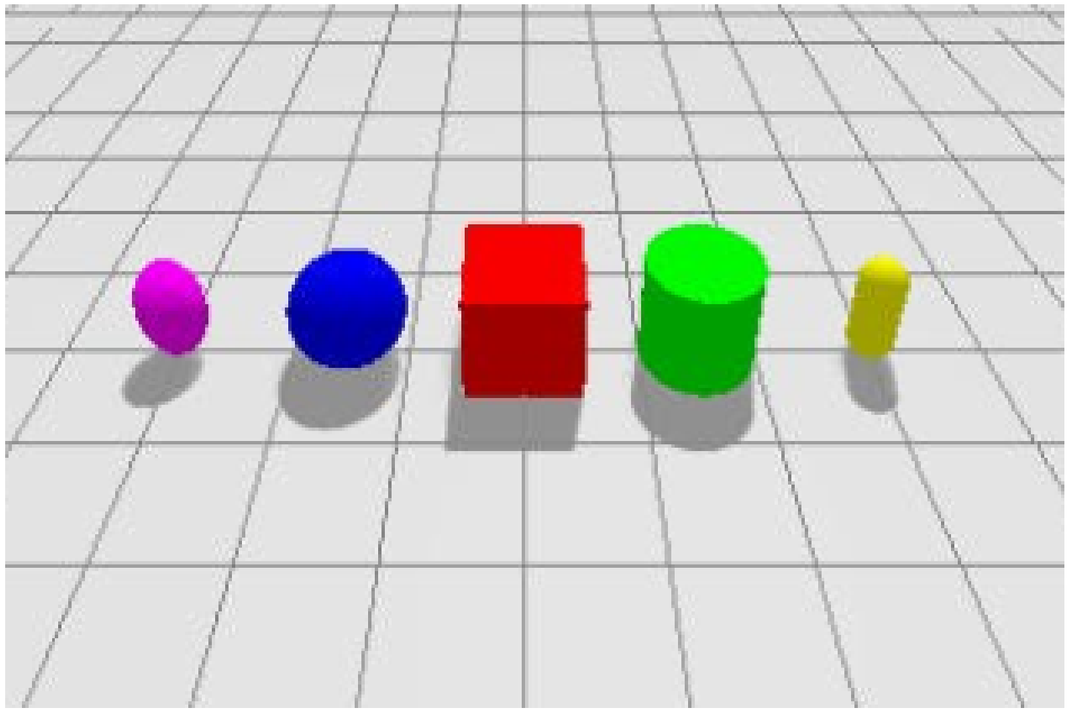


Docker

VS Code


Run Gazebo
Step 2

Goal:

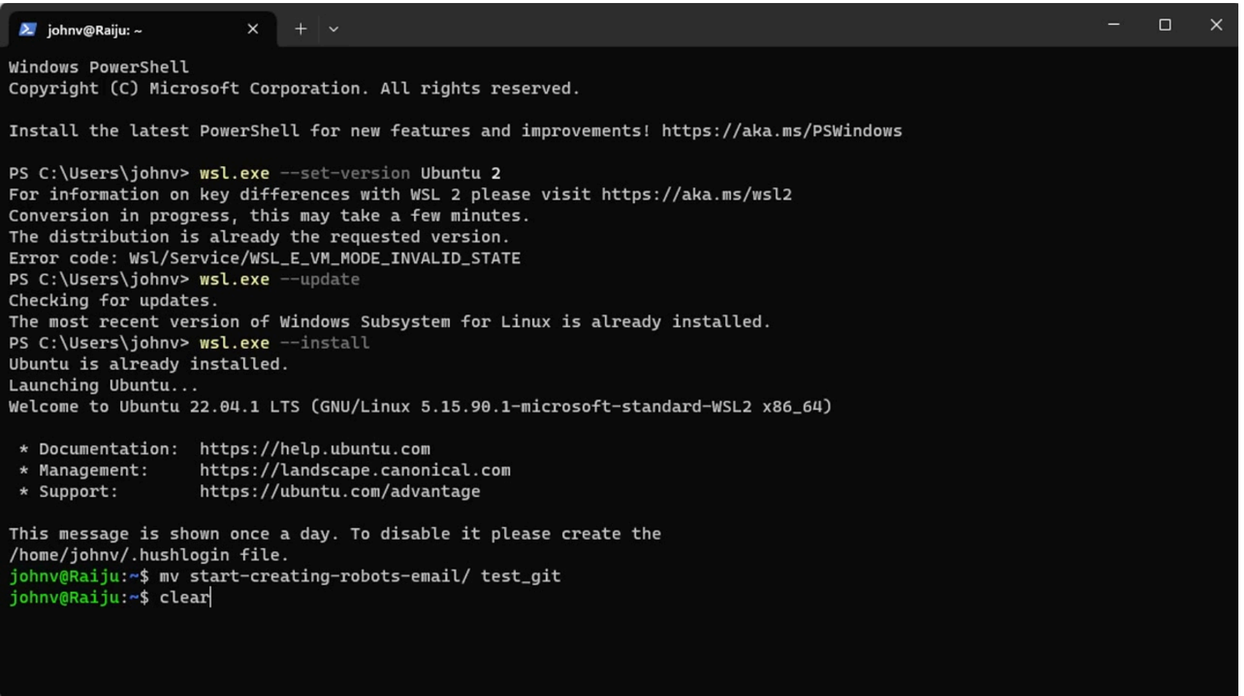


GAZEBO

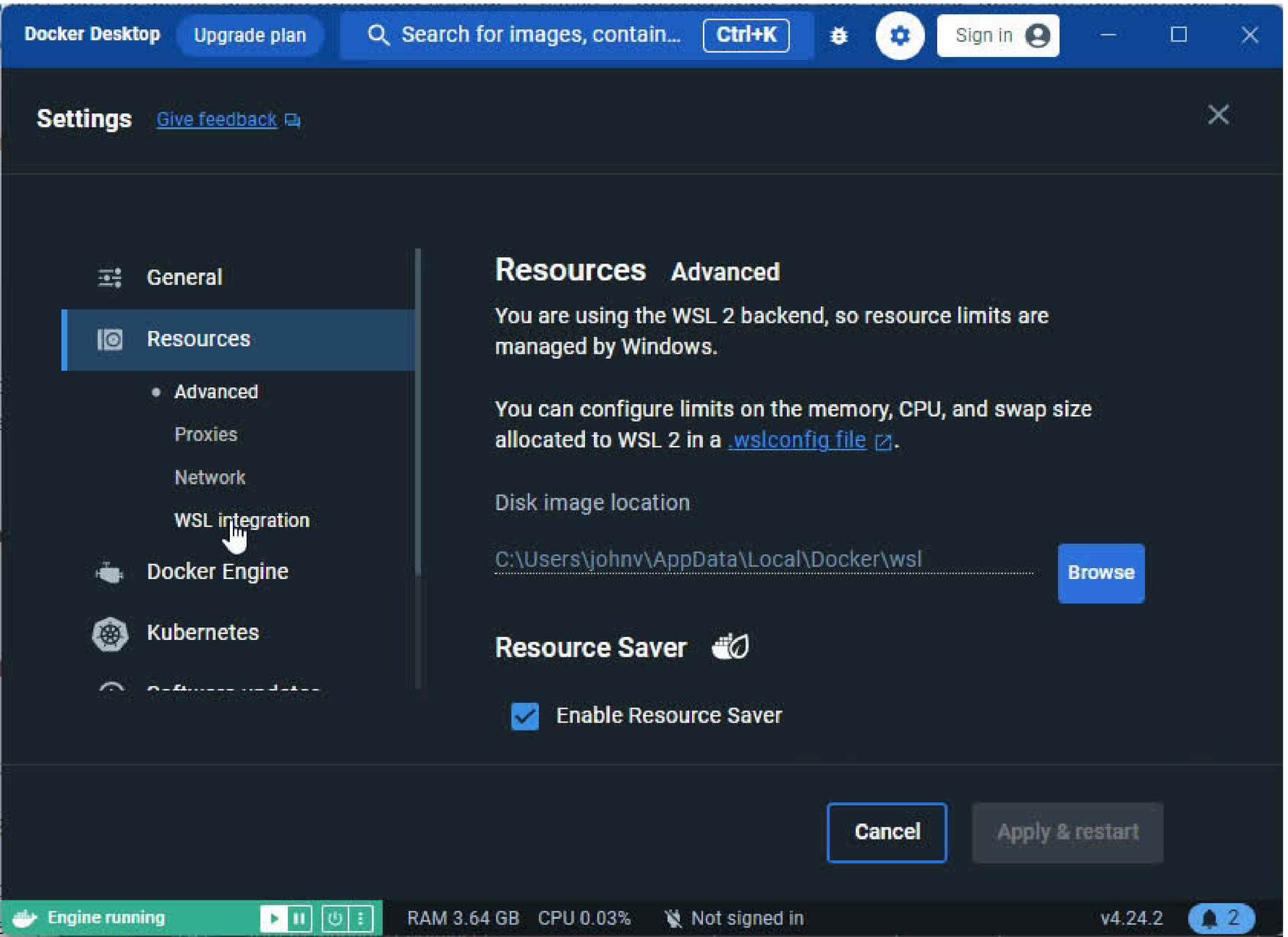
Most popular Robotics Operating System Simulation Tool

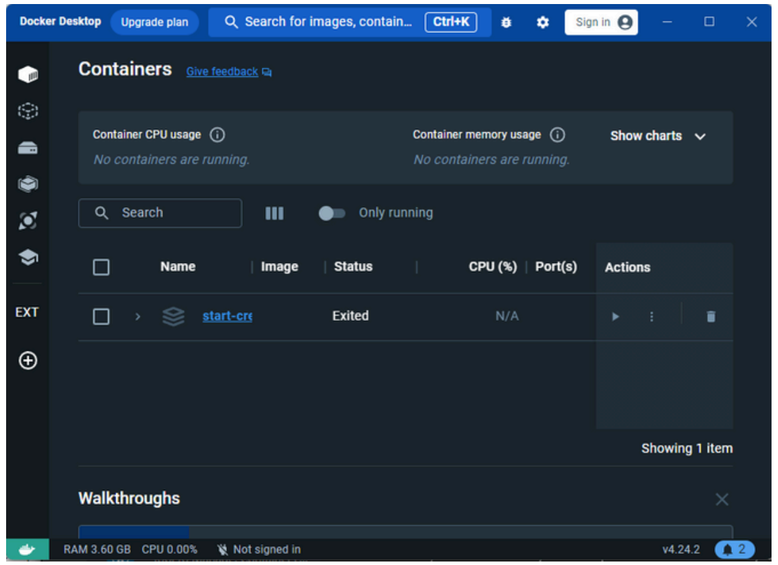
2a. Clone The Course

```
cd ~
git clone https://github.com/johnny555/start-creating-robots.git
cd start-creating-robots
code .
```



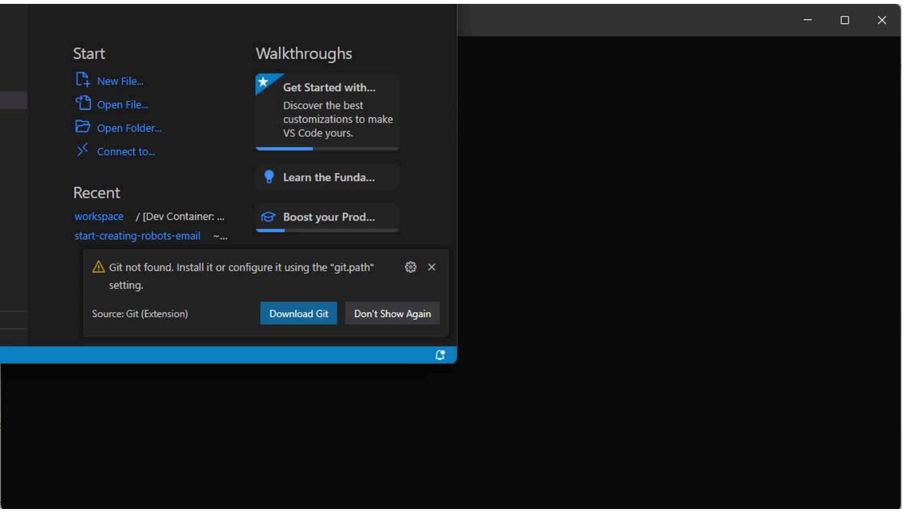
2b. Start Docker





Enable WSL Integration!

3. VS Code Dev Container



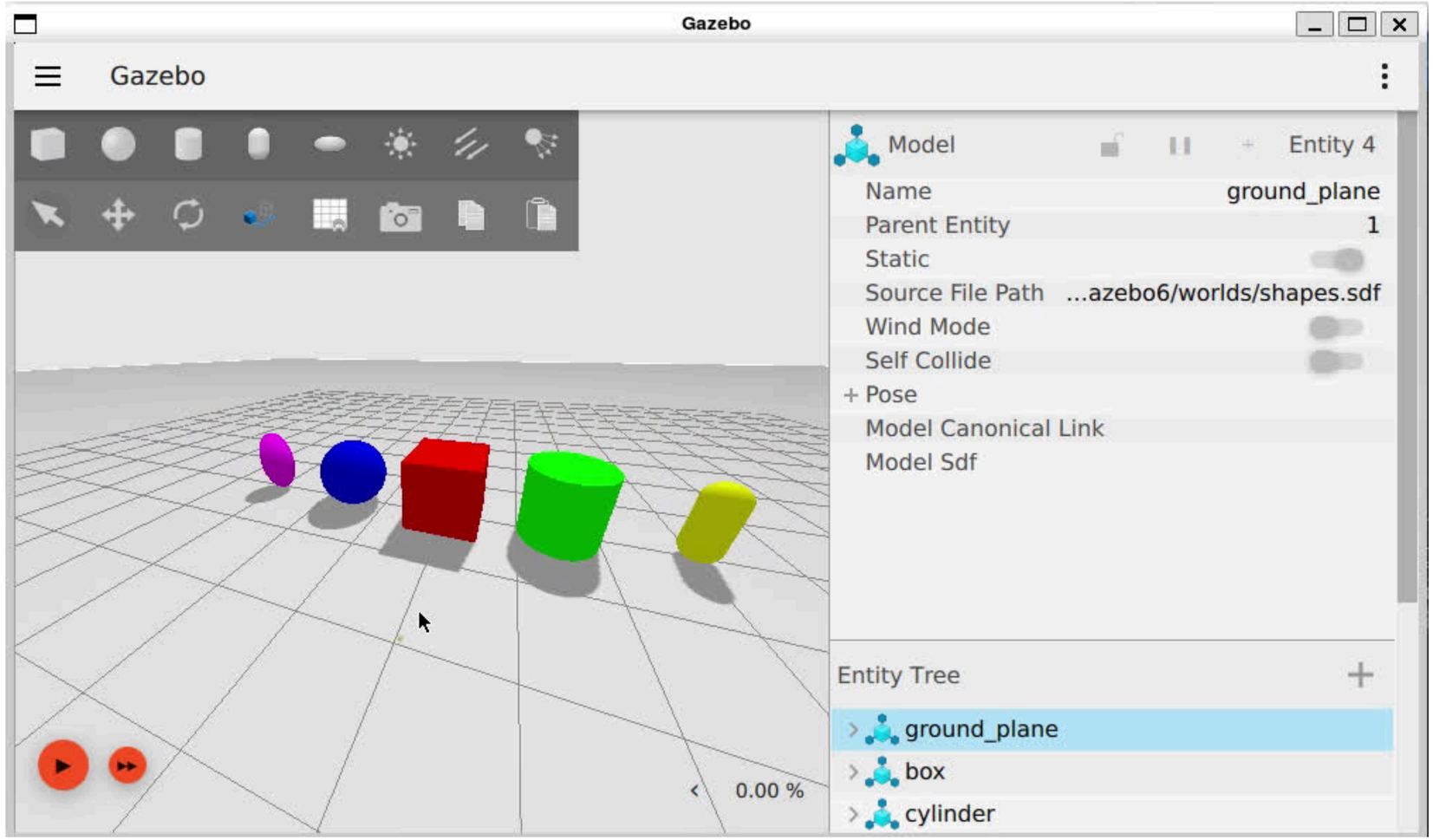
Install “Dev Containers” extension

CTRL+SHIFT+P

Open Folder In Dev Container

4. Run & Experiment With Gazebo

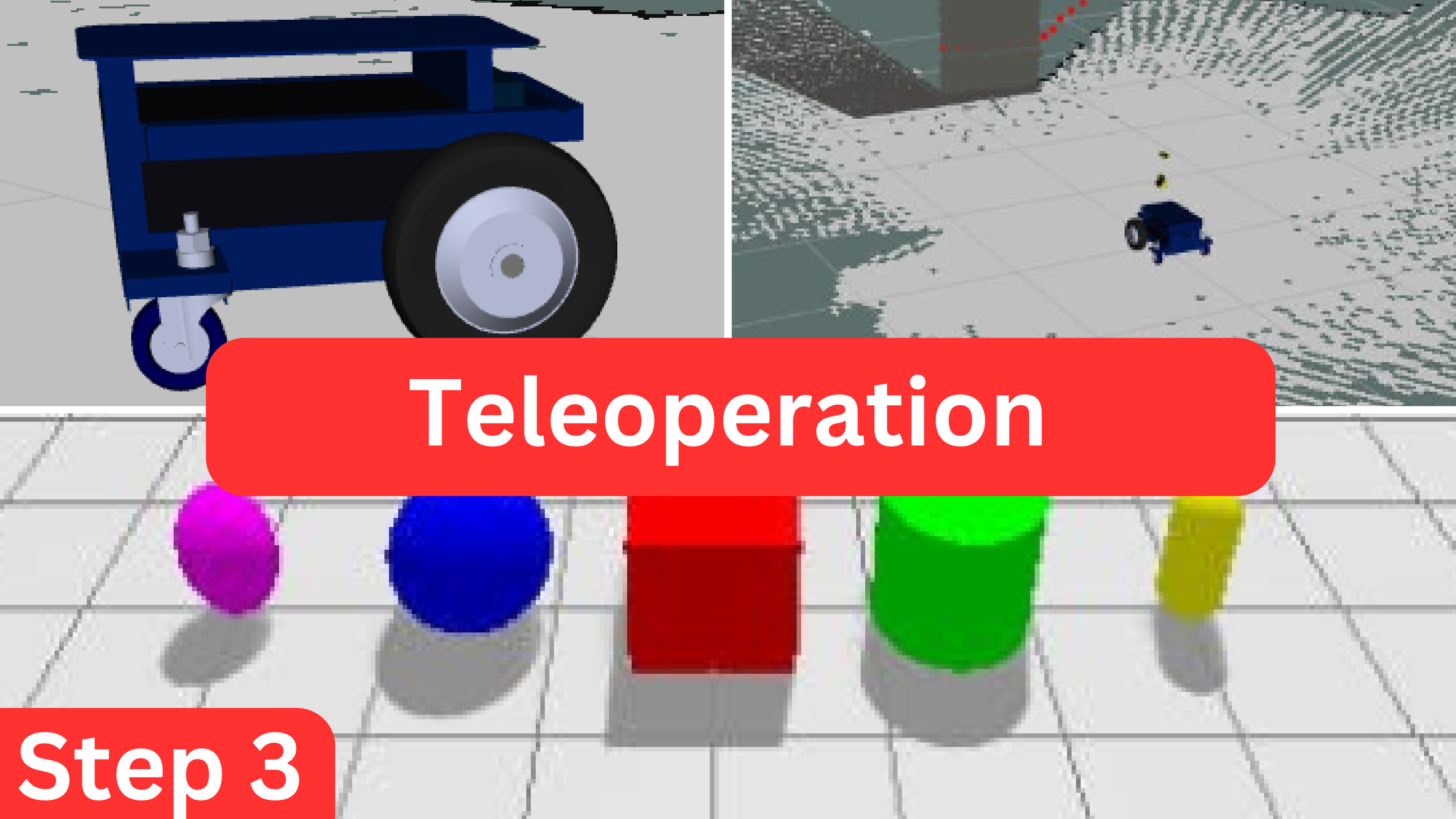
Ctrl+Shift+P
Run Task
Day 2



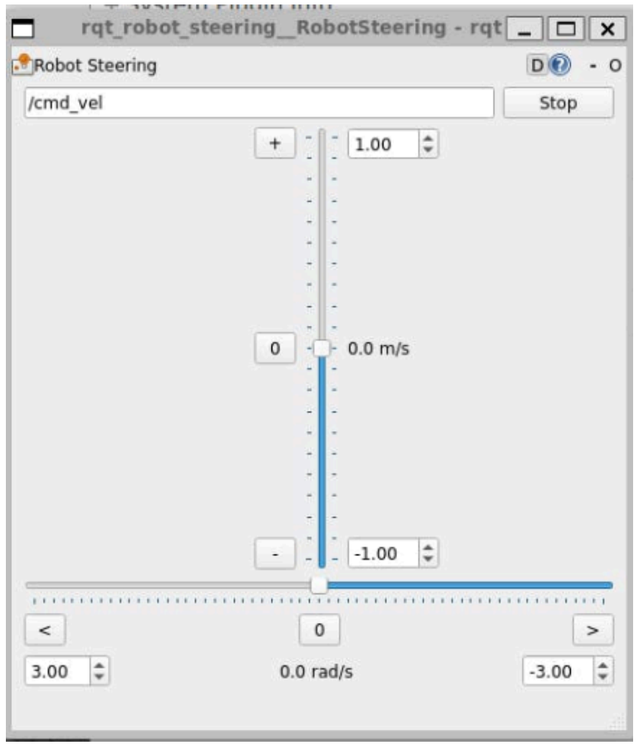
https://localhost:8080/vnc.html

Congratulations!

Now Lets Do Some ROBOTICS!

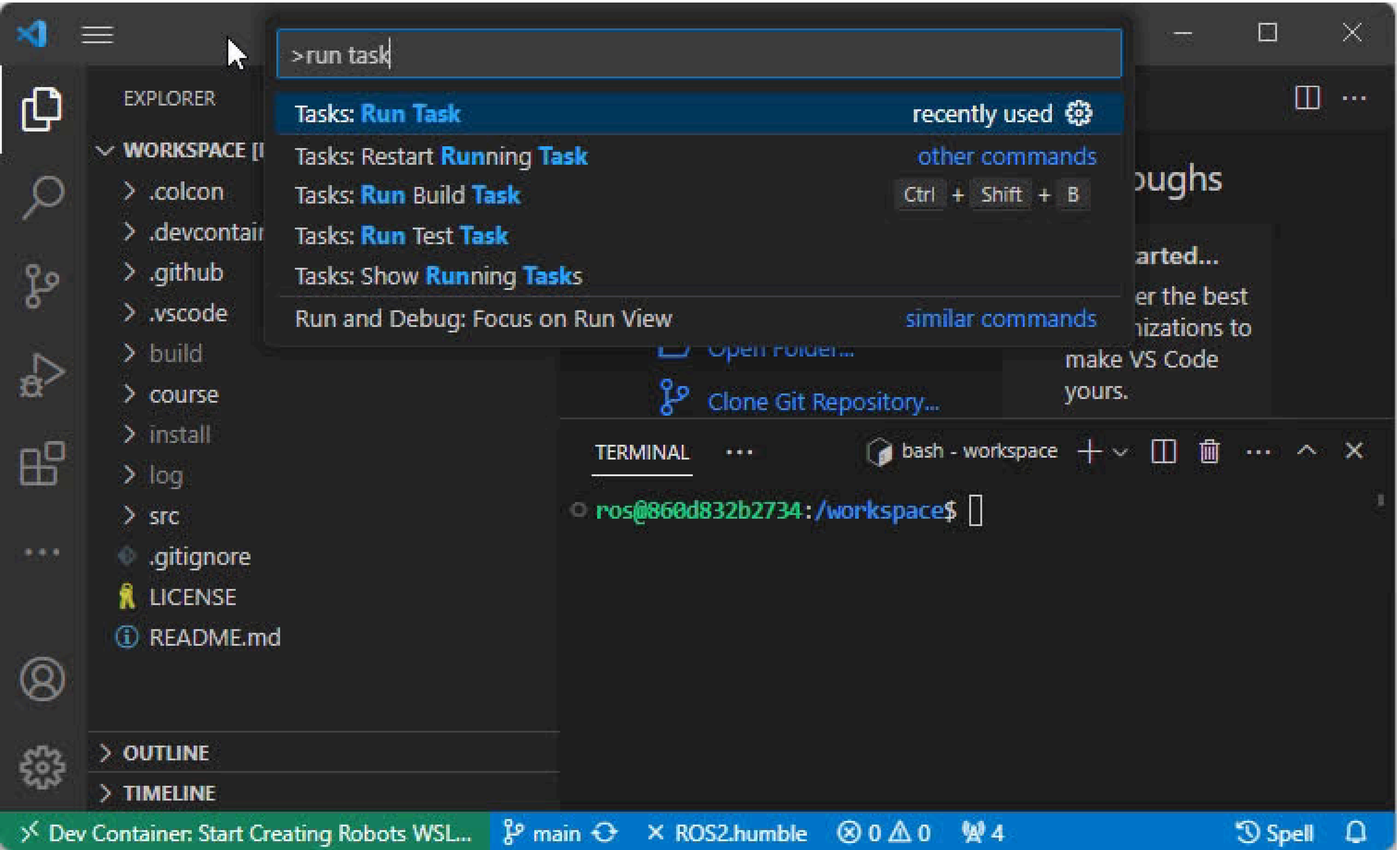

Teleoperation
Step 3

Goal: Teleoperation



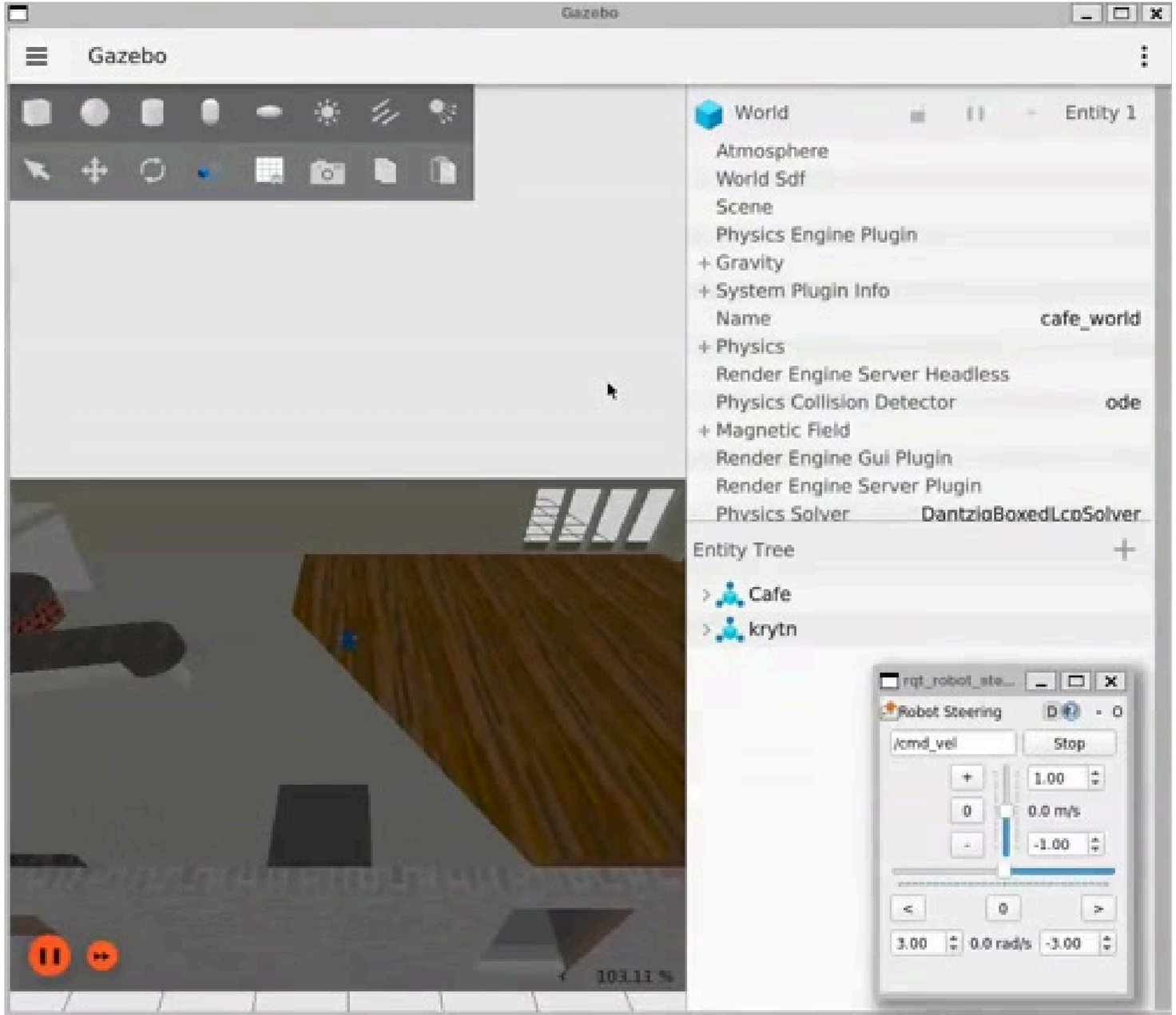
Using Robotics Operating System to drive simulated robot.

1. Start Simulation



Ctrl+Shift+P
Run Task
Day 3

2. Drive Robot In Gazebo



start_creating_robots_course/launch/gazebo.launch.py

3a. Code Intro

```python
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription, ExecuteProcess, DeclareLaunchArgument
from launch.launch_description_sources import get_launch_description_from_python_launch_file
from launch_ros.actions import Node
from ament_index_python.packages import get_package_share_directory
from os.path import join
from launch.substitutions import LaunchConfiguration, Command

import xacro


```

python imports

key launch function

```python
def generate_launch_description():

    # This allows us to have the with_sensors as an argument on the command line
    with_sensors_arg =  DeclareLaunchArgument(
        'with_sensors', default_value="false"
    )
    # This allows us to use the with_sensors variable in substitutions in this launch description.
    with_sensors = LaunchConfiguration('with_sensors', default="false")

    models_path = join(get_package_share_directory("start_creating_robots"), "worlds_and_models")

    # Start a simulation with the cafe world
    cafe_world_uri = join(models_path,"cafe.sdf")
    path = join(get_package_share_directory("ros_gz_sim"), "launch", "gz_sim.launch.py")

    gazebo_sim = IncludeLaunchDescription(path,
                                          launch_arguments=[("gz_args", '-r ' + cafe_world_uri)])

```

reuse other launch files

start_creating_robots_course/launch/gazebo.launch.py

3b. Code - Spawn Robot

```
# Create a robot in the world.
# Steps:
# 1. Process a file using the xacro tool to get an xml file containing the robot description.
# 2. Publish this robot description using a ros topic so all nodes can know about the joints of the robot.
# 3. Spawn a simulated robot in the gazebo simulation using the published robot description topic.

# Step 1. Process robot file.
robot_file = join(models_path, "krytn","krytn.urdf.xacro")
robot_xml = Command(["xacro ",robot_file," with_sensors:=",with_sensors])

#Step 2. Publish robot file to ros topic /robot_description
robot_state_publisher = Node(
    package='robot_state_publisher',
    executable='robot_state_publisher',
    name='robot_state_publisher',
    output='both',
    parameters=[{'robot_description':robot_xml,
                 'use_sim_time':True}],
)

# Step 3. Spawn a robot in gazebo by listening to the published topic.
robot = ExecuteProcess(
    cmd=["ros2", "run", "ros_gz_sim", "create", "-topic", "robot_description", "-z", "0.5"],
    name="spawn robot",
    output="both"
)
```

start_creating_robots_course/launch/gazebo.launch.py

3c. Code - Bridge & GUI

```
# Gazebo Bridge: This allows ROS to send messages to drive the robot in simulation.
bridge = Node(
    package='ros_gz_bridge',
    executable='parameter_bridge',
    arguments=['/model/krytn/cmd_vel@geometry_msgs/msg/Twist@gz.msgs.Twist'],
    output='screen',
    remappings=[('/model/krytn/cmd_vel','/cmd_vel')]
)

# A gui tool for easy tele-operation.
robot_steering = Node(
    package="rqt_robot_steering",
    executable="rqt_robot_steering",
)
```

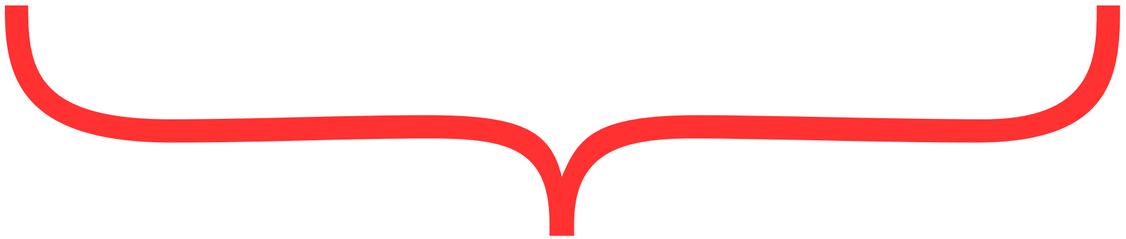
start_creating_robots_course/launch/gazebo.launch.py

3c. main function

```
12  def generate_launch_description():
13
```

-
-
-

```
71
72      return LaunchDescription([gazebo_sim, bridge, robot, robot_steering, robot_state_publisher, with_sensors_arg])
```

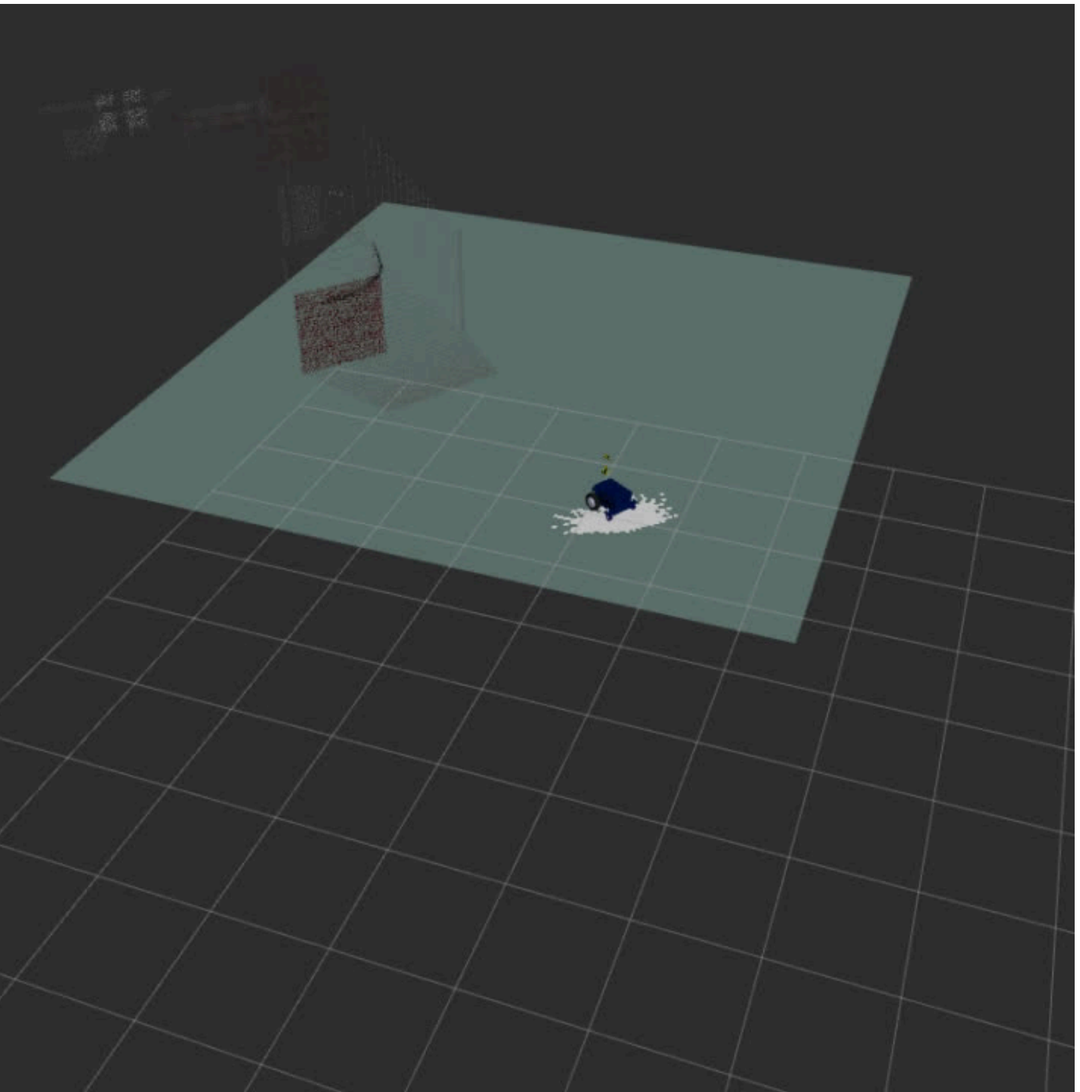


Actions To Run

Mapping

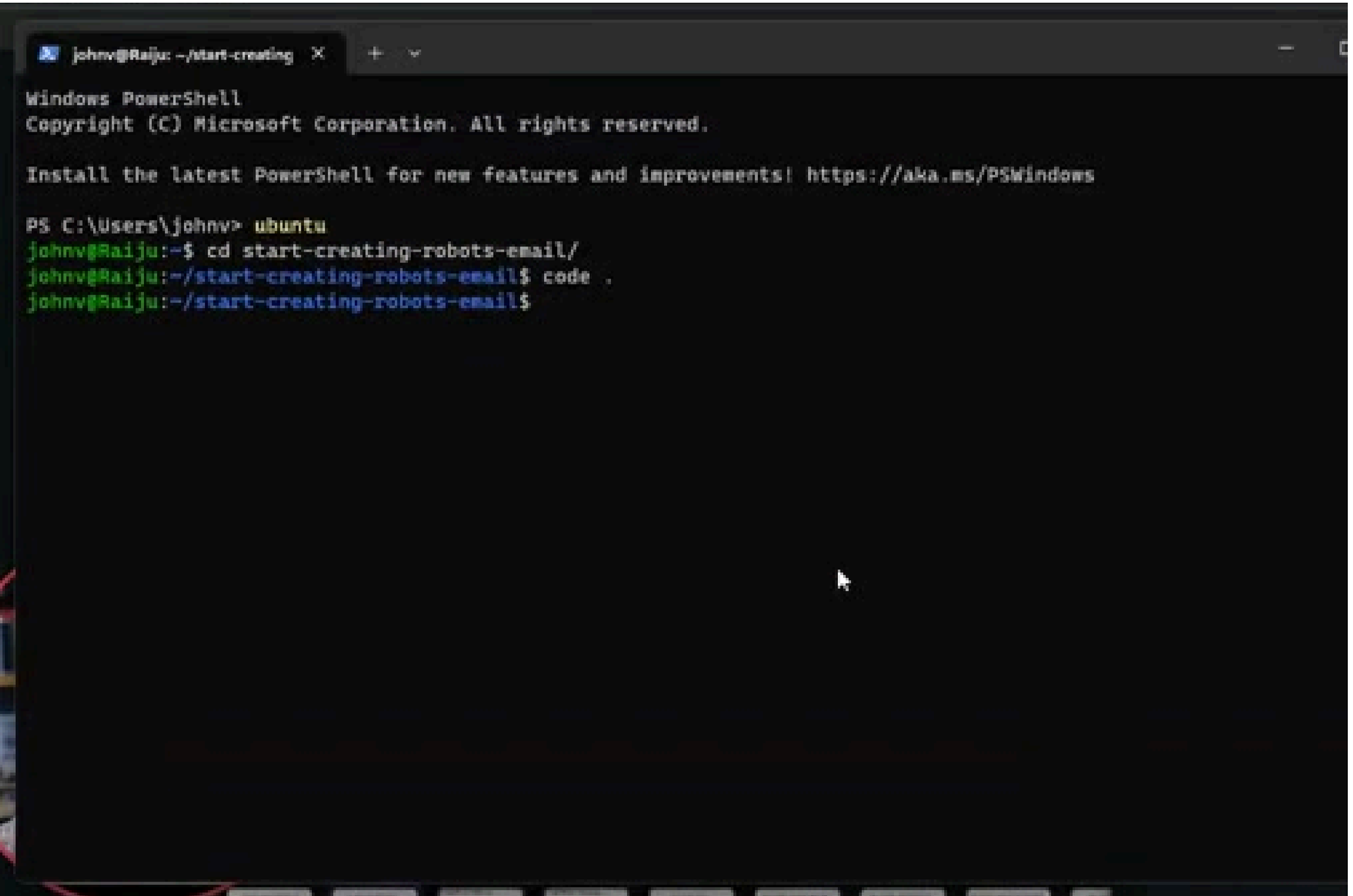
Step 4

Goal: Mapping



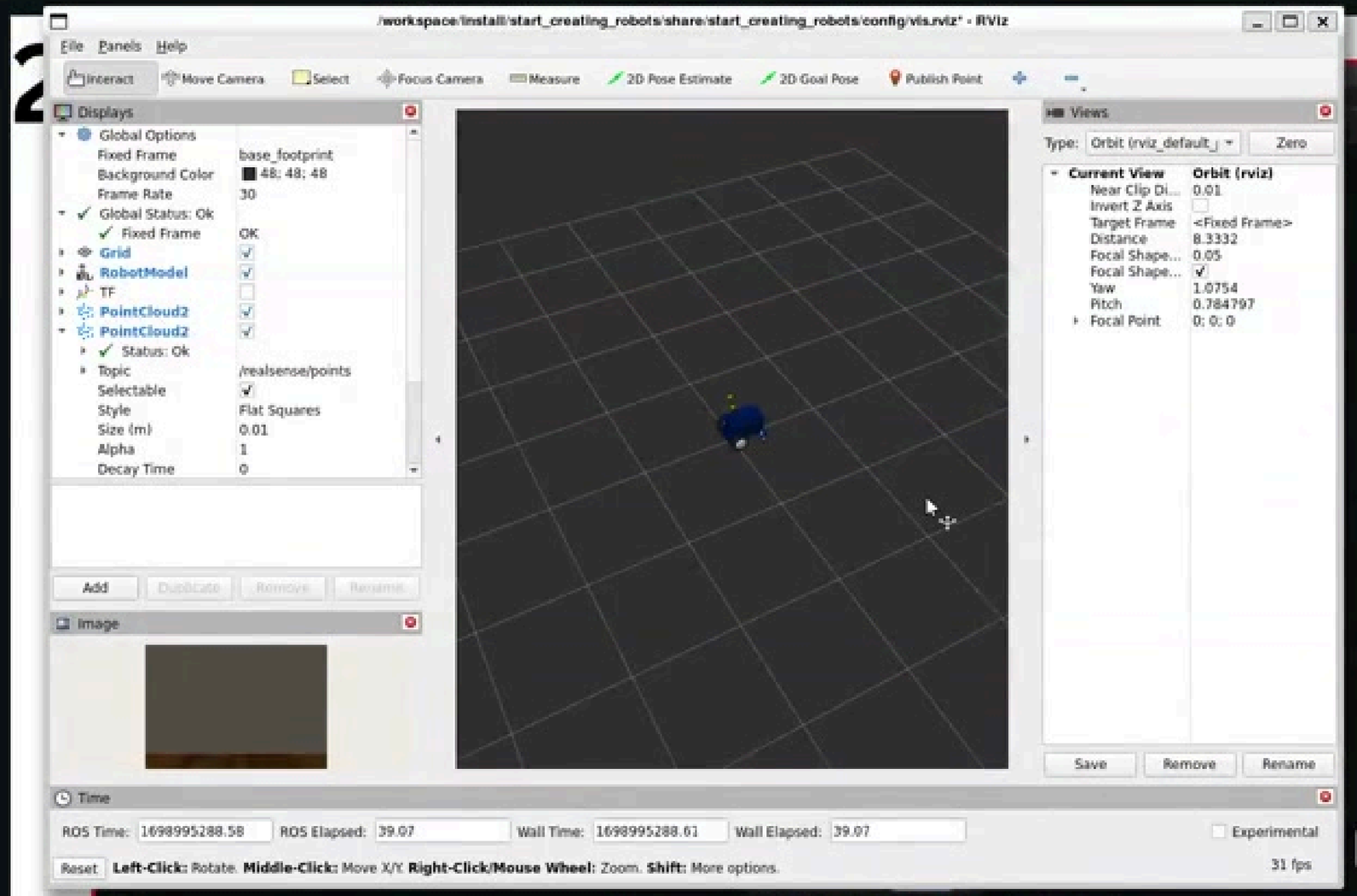
Using Robotics Operating System to create a map

1. Start Simulation



Ctrl+Shift+P
Run Task
Day 4

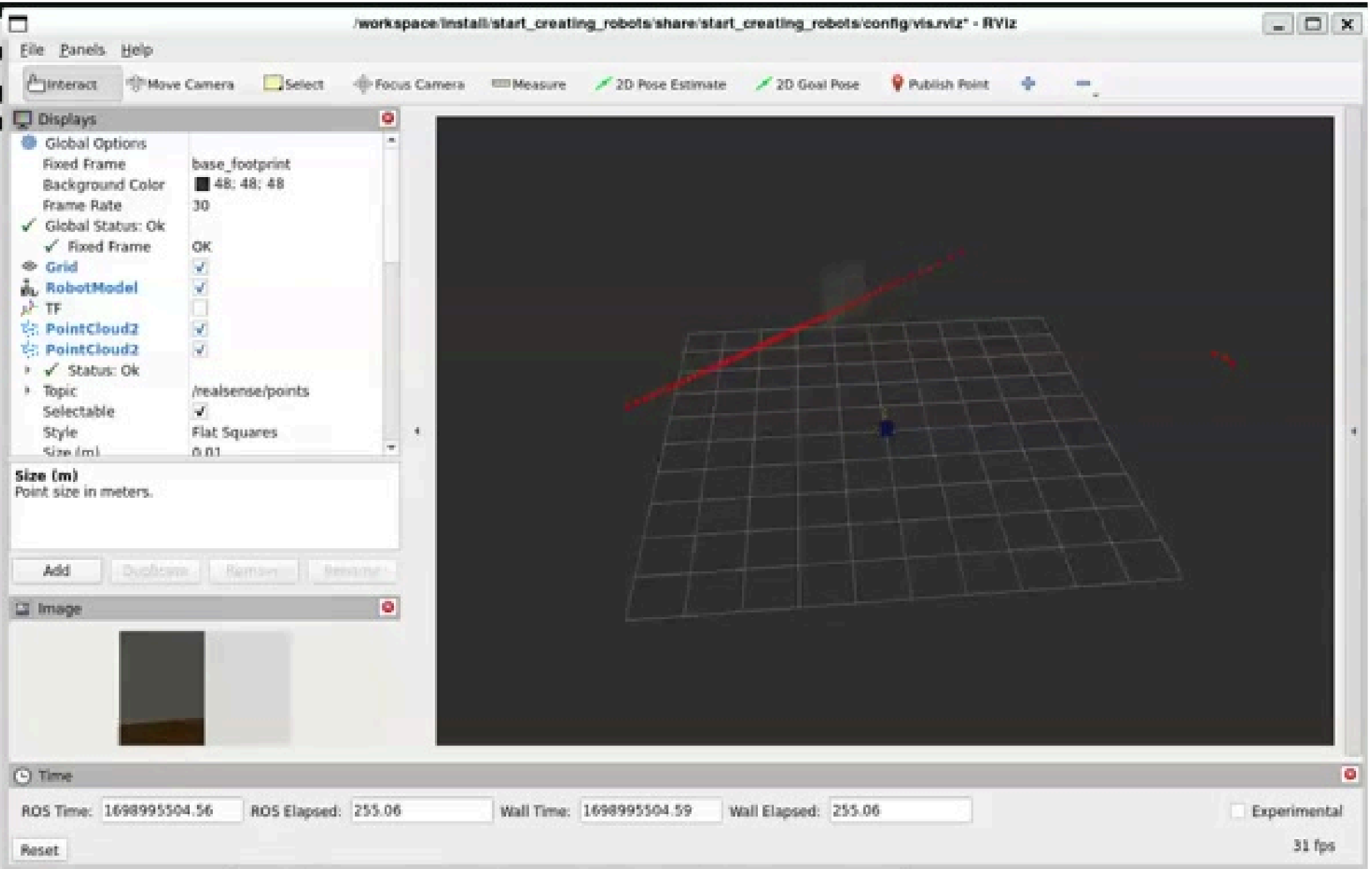
2a. Use RViz - Sensors



Drive your robot

Use RViz to look through your robots sensors.

2b. Use RViz - Mapping



Add Map To RViz

Click “Add”, then “By Topic” select `/map` and Map

then hit OK.

Create a map of the world

start_creating_robots/launch/mapping.launch.py

3a. Code Intro

```python
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription, DeclareLaunchArgument, SetLaunchConfiguration
from launch.substitutions import LaunchConfiguration, PathJoinSubstitution
from launch_ros.actions import Node
from launch_ros.substitutions import FindPackageShare
from ament_index_python.packages import get_package_share_directory
from launch.event_handlers import OnExecutionComplete

from os.path import join
```

python imports

key launch function

```python
def generate_launch_description():
    # This allows us to have the with_sensors as an argument on the command line
    rviz_config_arg =  DeclareLaunchArgument(
        'rviz_config', default_value="mapping.yaml"
    )
    # This allows us to use the with_sensors variable in substitutions in this launch description.
    rviz_config = LaunchConfiguration('rviz_config', default="vis.rviz")

    base_path = get_package_share_directory("start_creating_robots")

    # We will include everything from the gazebo launch file, making sure that sensors are now enabled

    with_sensors_true = SetLaunchConfiguration("with_sensors","true")
    gazebo = IncludeLaunchDescription(join(base_path, "launch","gazebo.launch.py"),
                                      launch_arguments=[('with_sensors','true')])
```

reuse other launch files

start_creating_robots/launch/mapping.launch.py

3b. Code - Gazebo Bridge

```
    # Extended Gazebo Bridge: To do mapping requires alot more info from the simulation. We need sensor data and e
    extended_bridge = Node( package='ros_gz_bridge', name="extended_gazebo_bridge", executable='parameter_bridge',
    arguments=['/model/krytn/odometry@nav_msgs/msg/Odometry[gz.msgs.Odometry',
                    '/model/krytn/tf@tf2_msgs/msg/TFMessage[gz.msgs.Pose_V',

                    '/lidar@sensor_msgs/msg/LaserScan@gz.msgs.LaserScan',
                    '/lidar/points@sensor_msgs/msg/PointCloud2[gz.msgs.PointCloudPacked',

                    '/realsense/image@sensor_msgs/msg/Image[gz.msgs.Image',
                    '/realsense/depth@sensor_msgs/msg/Image[gz.msgs.Image',
                    '/realsense/points@sensor_msgs/msg/PointCloud2[gz.msgs.PointCloudPacked',

                    '/clock@rosgraph_msgs/msg/Clock[gz.msgs.Clock',

                    '/joint_states@sensor_msgs/msg/JointState[gz.msgs.Model'],
                    output='screen', remappings=[('/model/krytn/odometry','/odom'),
                     ('/model/krytn/tf','/tf')]
    )
```

Connects Sensors To System

start_creating_robots/launch/mapping.launch.py

3c. Code - Workaround Transforms

```
46      # Gazebo fortress has a bug that won't respect our frame_id tags. So we have to publish a transform
47      depth_cam_link_tf = Node(package='tf2_ros',
48                               executable='static_transform_publisher',
49                               name='depthCamLinkTF',
50                               output='log',
51                               arguments=['0.0', '0.0', '0.0', '0.0', '0.0', '0.0',
52                               'realsense_d435', 'krytn/base_footprint/realsense_d435'])
53
54      krytn_base_fp_link_tf = Node(package='tf2_ros',
55                                   executable='static_transform_publisher',
56                                   name='base_fp_linkTF',
57                                   output='log',
58                                   arguments=['0.0', '0.0', '0.0', '0.0', '0.0', '0.0',  'krytn/base_footprint', 'base_footprint'])
59
```

start_creating_robots/launch/mapping.launch.py

3d. Code - Mapping and RViz

mapping config file

```
60      # SLAM Toolbox for mapping
61      slam_toolbox = Node( package='slam_toolbox',
62                           executable='async_slam_toolbox_node',
63                           parameters=[
64                                get_package_share_directory('start_creating_robots') + '/config/mapping.yaml'
65                           ], output='screen'
66      )
67
68      rviz = Node(
69          package='rviz2',
70          executable='rviz2',
71          arguments=[
72              '-d',
73              PathJoinSubstitution([base_path, 'config', rviz_config])
74          ]
75      )
76
```

start_creating_robots/launch/mapping.launch.py

3e. main function

```
12  def generate_launch_description():
13
```

●
●
●

```
78      return LaunchDescription([with_sensors_true,
79                                gazebo,
80                                extended_bridge,
81                                depth_cam_link_tf,
82                                krytn_base_fp_link_tf,
83                                slam_toolbox,
84                                rviz,
85                                rviz_config_arg])
```

Actions To Run

start_creating_robots/config/mapping.yaml

3f. mapping config

important parameters

```yaml
    # ROS Parameters
    odom_frame: odom
    map_frame: map
    base_frame: base_footprint
    scan_topic: /lidar
    use_map_saver: true
    mode: mapping #localization
    use_sim_time: true
```

```yaml
    # General Parameters
    use_scan_matching: true
    use_scan_barycenter: true
    minimum_travel_distance: 0.5
    minimum_travel_heading: 0.5
    scan_buffer_size: 10
    scan_buffer_maximum_scan_distance: 10.0
    link_match_minimum_response_fine: 0.1
    link_scan_maximum_distance: 1.5
    loop_search_maximum_distance: 3.0
```

tunable parameters

start_creating_robots/worlds_and_models/krytn/lidar_2d_v1.urdf.xacro

3g. Lidar Config

```
45          <scan>
46            <horizontal>
47              <samples>90</samples>
48              <resolution>1</resolution>
49              <min_angle>-1.41</min_angle>
50              <max_angle>1.41</max_angle>
51            </horizontal>
```

Change angle and number of lidar points

change max range

```
52          </scan>
53          <range>
54            <min>0.060</min>
55            <max>10</max>
56            <resolution>0.04</resolution>
57          </range>
58          <noise>
```

start_creating_robots/worlds_and_models/krytn/realsense_d435.urdf.xacro

3h. Camera Config

```
32      <topic>realsense</topic>
33      <ignition_frame_id>realsense_d435</ignition_frame_id>
34      <update_rate>5</update_rate>
35      <camera name="camera">
36        <horizontal_fov>0.5</horizontal_fov>
37        <lens>
38          <intrinsics>
```

change camera frames per second

change field of view

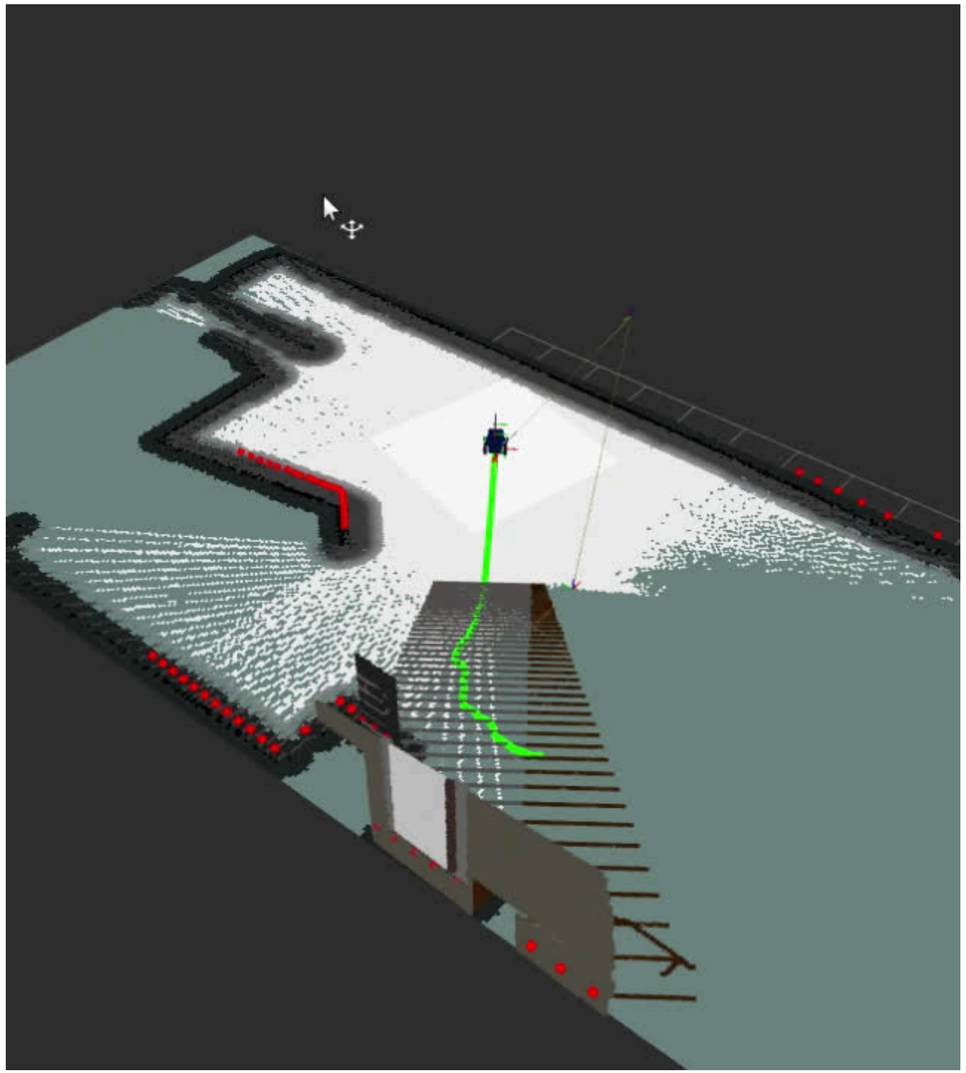
Change image resolution and min/max range

```
57        <image>
58          <width>160</width>
59          <height>120</height>
60          <format>R8G8B8</format>
61        </image>
62        <clip>
63          <near>0.01</near>
64          <far>10</far>
65        </clip>
66        <depth_camera>
67          <clip>
68            <near>0.1</near>
69            <far>10</far>
70          </clip>
71        </depth_camera>
```

Navigation

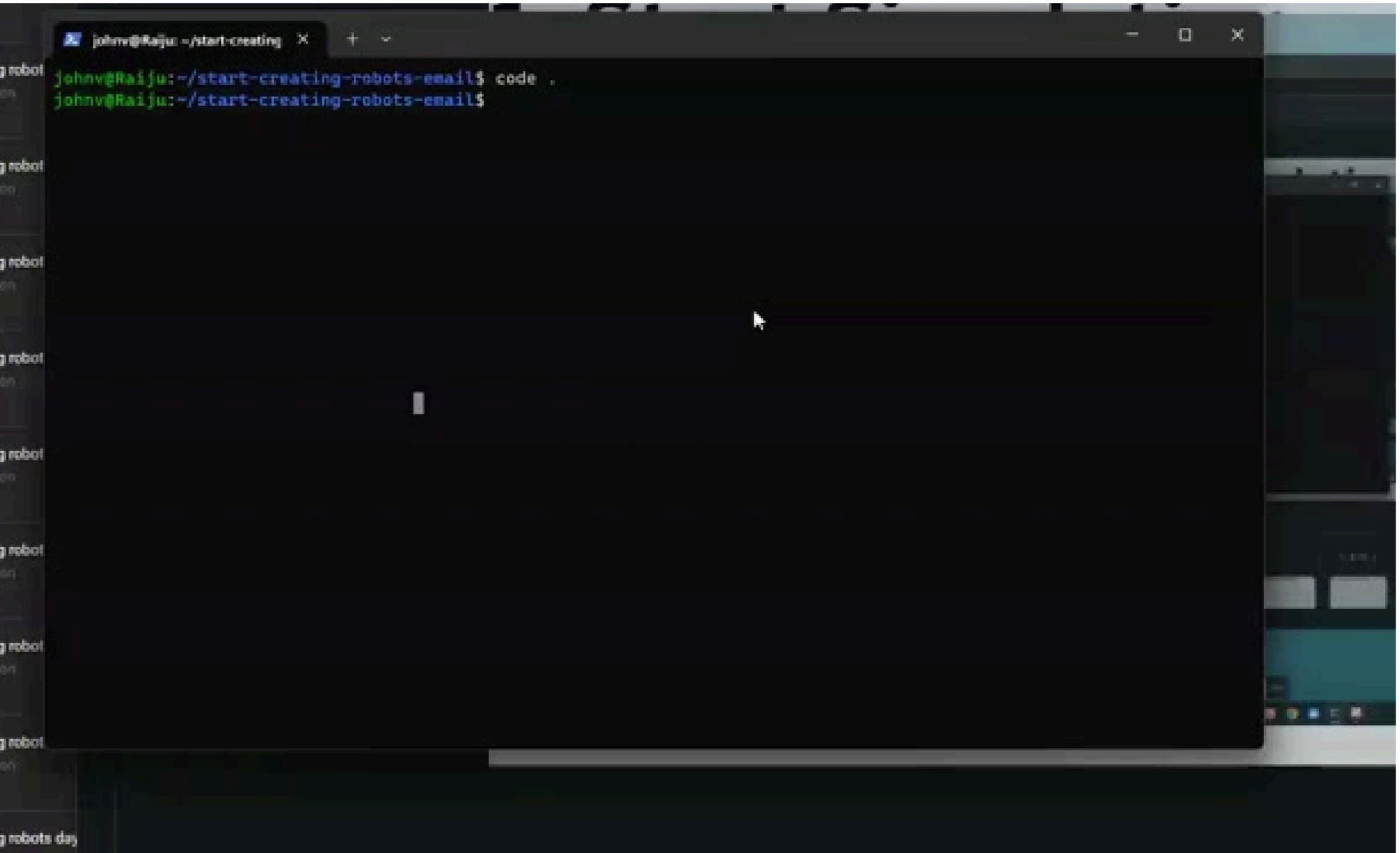
Step 5

Goal: Navigation



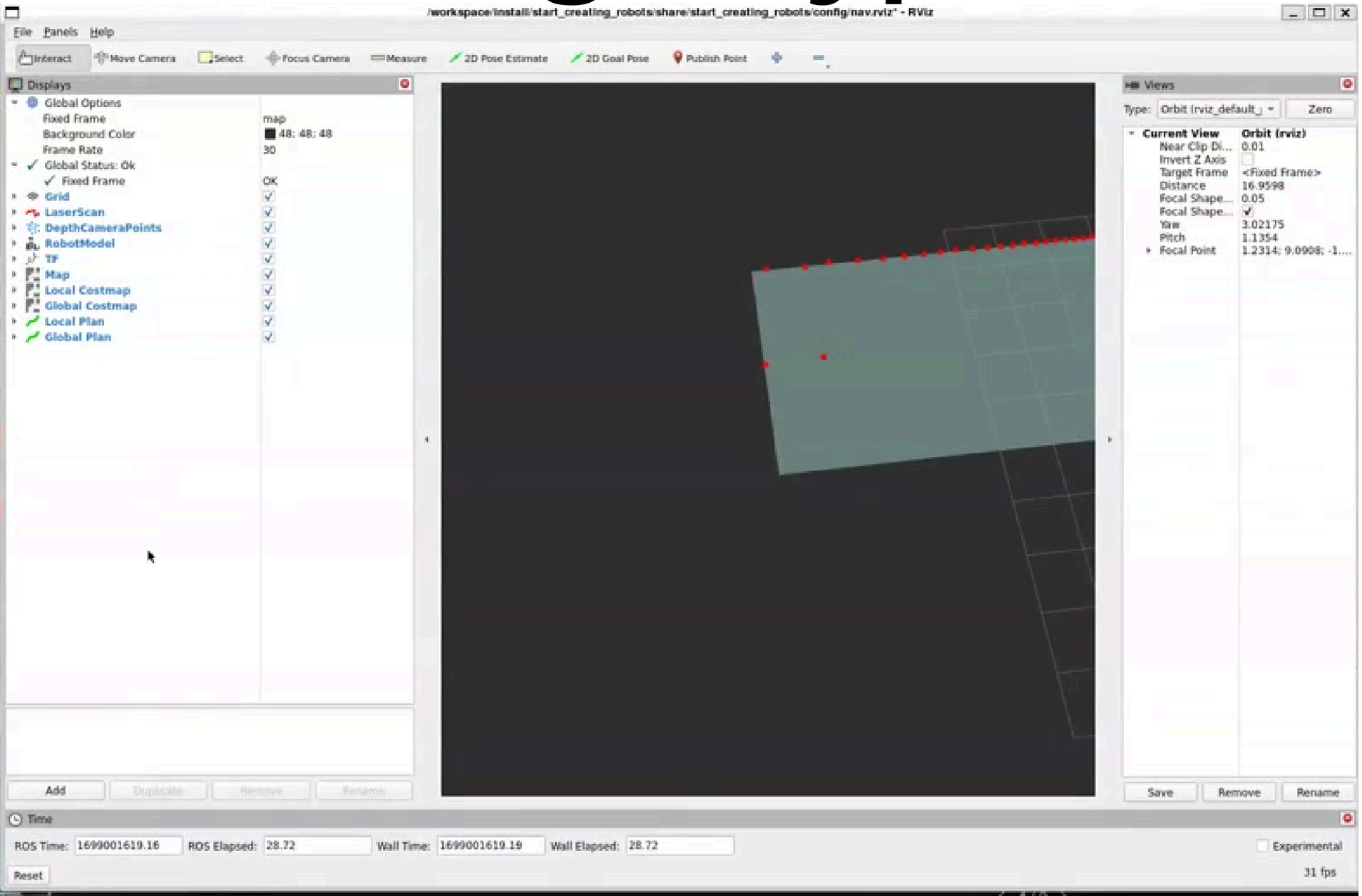
Using Robotics Operating System to autonomously drive

1. Start Simulation



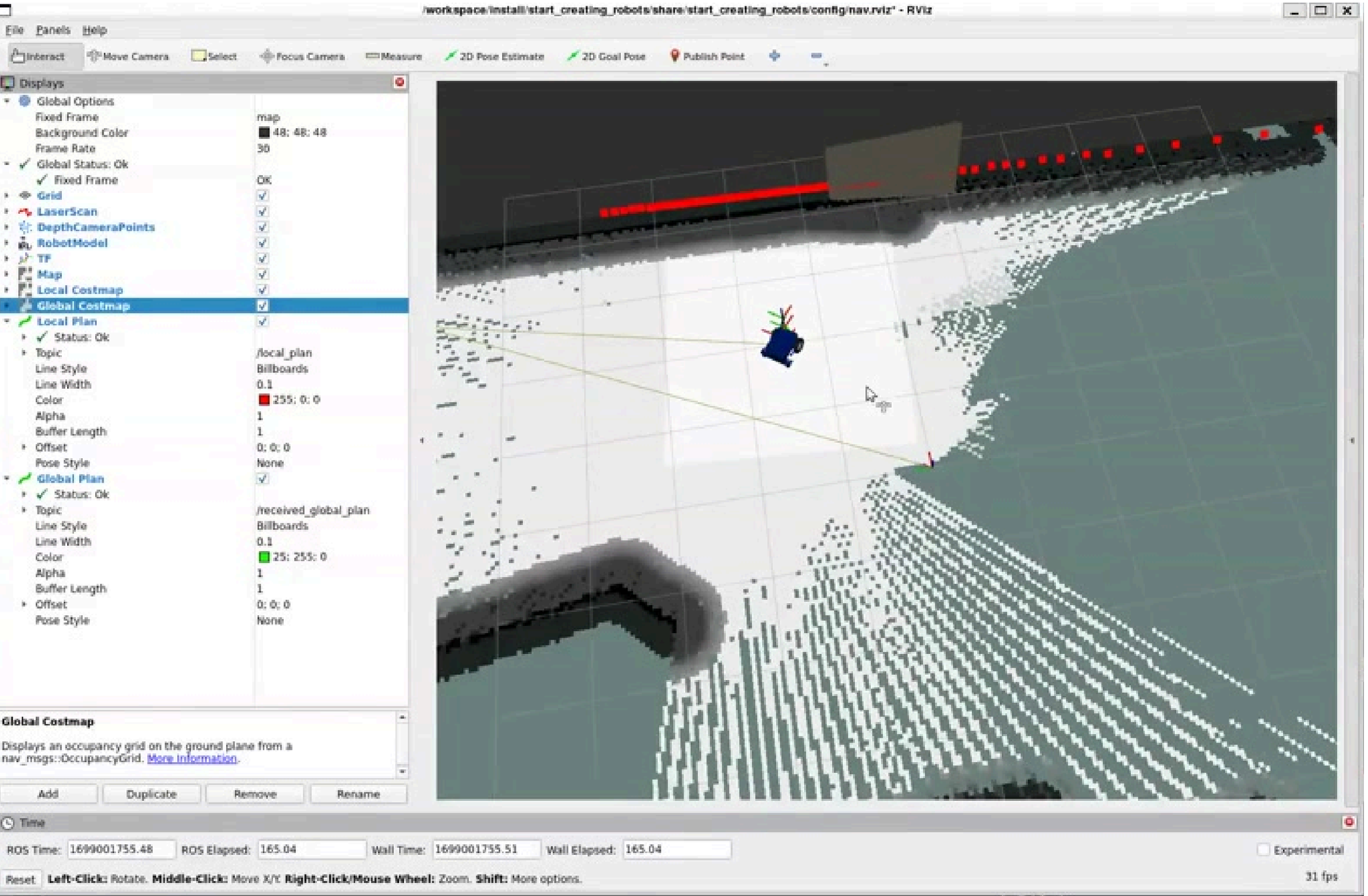
Ctrl+Shift+P
Run Task
Day 5

2a. Setting Waypoints



Click “2D Goal Pose” to send a waypoint.

2b. Explore Cost Maps



Try visualising the local and global costmaps.

What do you see? How are they different?

Look for the local and global path plans.

start_creating_robots/launch/navigation.launch.py

3a. Code

key launch function

reuse other launch files

navigation node

```python
def generate_launch_description():

    base_path = get_package_share_directory("start_creating_robots")

    # We will include everything from the mapping launch file, making sure
    # that sensors are now enabled and setting up RVIZ for navigation.
    gazebo_and_mapping = IncludeLaunchDescription(join(base_path, "launch","mapping.launch.py"),
                                launch_arguments=[("with_sensors","true"), ("rviz_config","nav.rviz")])

    # Nav2 bringup for navigation
    navigation = IncludeLaunchDescription(
        PythonLaunchDescriptionSource([get_package_share_directory('nav2_bringup'), '/launch/navigation_launch.py']),
        launch_arguments={
            'map_subscribe_transient_local': 'true',
            'use_sim_time': 'true',
            'params_file': get_package_share_directory('start_creating_robots') + '/config/navigation.yaml'
        }.items()
    )

    return LaunchDescription([
        gazebo_and_mapping, navigation
    ])
```

start_creating_robots/config/navigation.yaml

3f. navigation config

important parameters

behaviour plugins

```yaml
bt_navigator:
  ros__parameters:
    use_sim_time: True
    global_frame: map
    robot_base_frame: base_footprint
    odom_topic: /odom
    bt_loop_duration: 10
    default_server_timeout: 20
    # 'default_nav_through_poses_bt_xml' and 'default_nav_to_pose_bt_xml' are use defau
    # nav2_bt_navigator/navigate_to_pose_w_replanning_and_recovery.xml
    # nav2_bt_navigator/navigate_through_poses_w_replanning_and_recovery.xml
    # They can be set here or via a RewrittenYaml remap from a parent launch file to Na
    plugin_lib_names:
      - nav2_compute_path_to_pose_action_bt_node
      - nav2_compute_path_through_poses_action_bt_node
      - nav2_smooth_path_action_bt_node
      - nav2_follow_path_action_bt_node
      - nav2_spin_action_bt_node
```

Thank You!

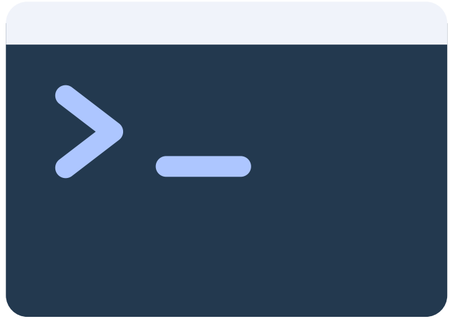
www.roborenegades.com

ROS Topics

Extra

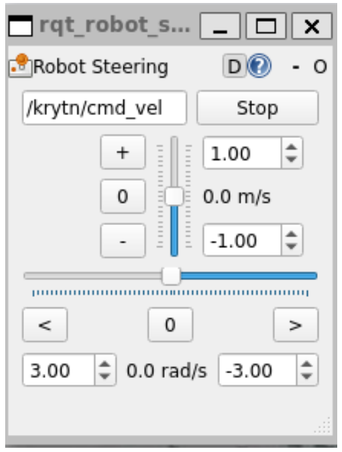
Listen To Your Robot

Launch
“Day 3”



Run Command

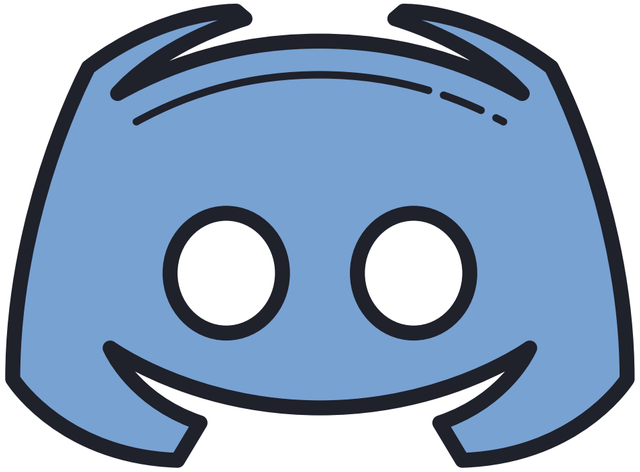
```
ros2 topic echo /cmd_vel
```

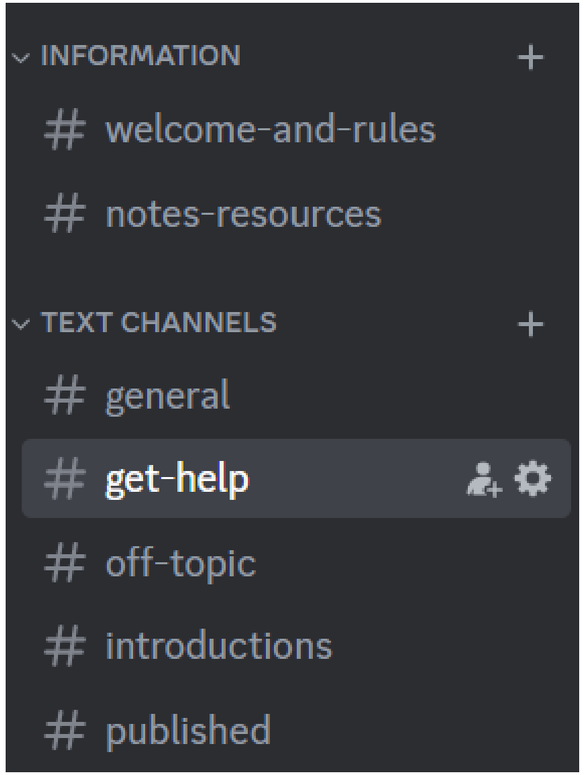


Drive Robot

You may need to run
`source install/setup.bash`

ROS Is Like Discord.
Topics are channels

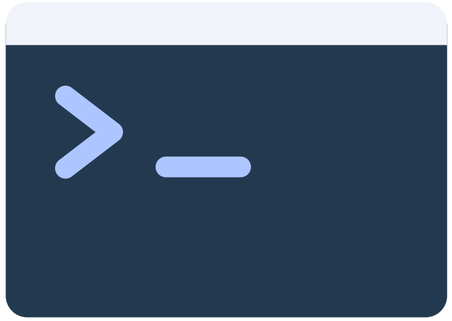





```
ros2 topic list
```

But Who Is Talking?

ROS Nodes are talking



Run Command

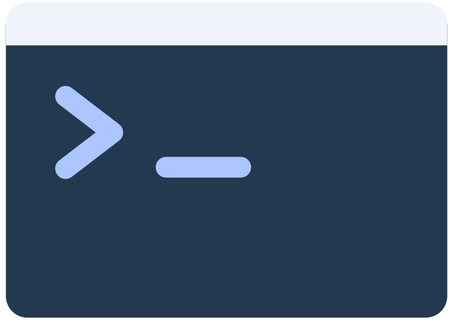
`ros2 node list`

```
ros@048e6f2f0453:/workspace$ ros2 node list
/robot_state_publisher
/ros_gz_bridge
/rqt_gui_py_node_2699
```

src/krytn/launch/gazebo.launch.py

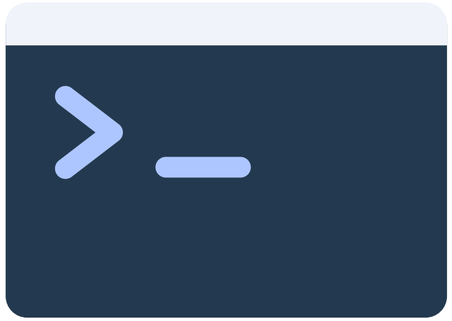
```
return LaunchDescription([gazebo_sim,
                          bridge,
                          robot,
                          robot_steering,
                          robot_state_publisher,
                          start_controllers])
```

The ros2 Command



Run Command

`ros2 --help`



Run Command

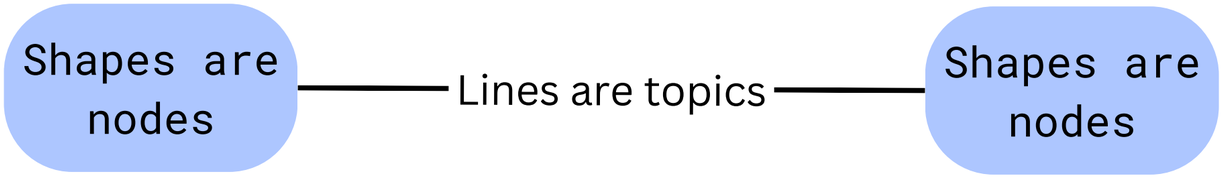
`ros2 topic --help`

Run Command

`ros2 topic echo --help`

It’s Time To Listen

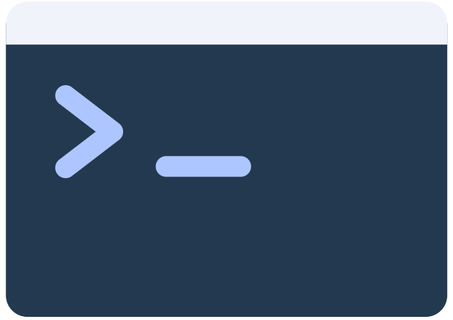
- [ ] Use ros2 command to map out how the 3 nodes talk to each other
- [ ] Draw a diagram using the standard below.




ROS Services & Actions
Extra

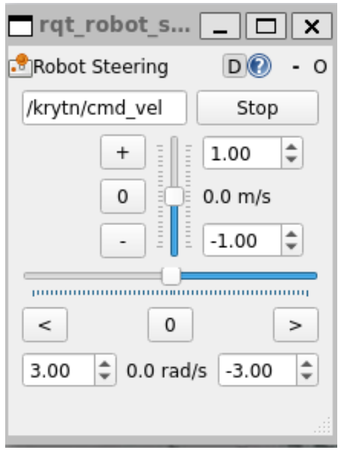
Listen To Your Robot

Launch
“Day 5”



Run Command

```
ros2 service list
ros2 action list
```

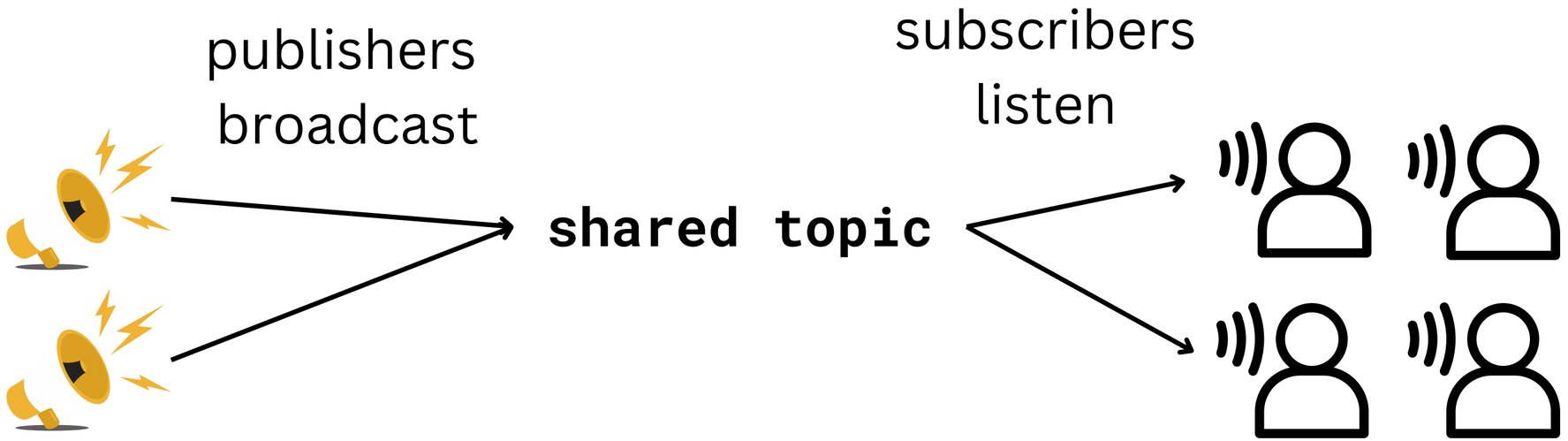


Drive Robot

You may need to run
`source install/setup.bash`

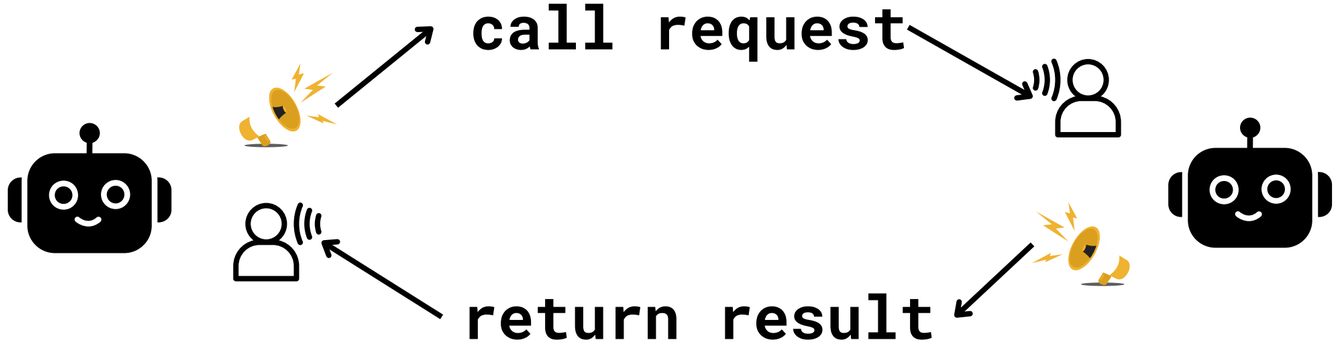
Services are Topics That Return

Topic



Service

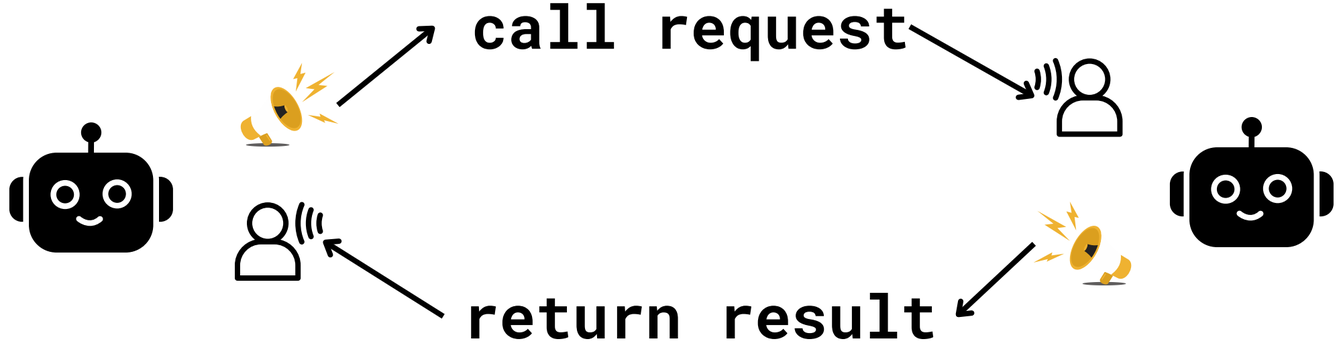
example:
map server



Actions are Services With Feedback

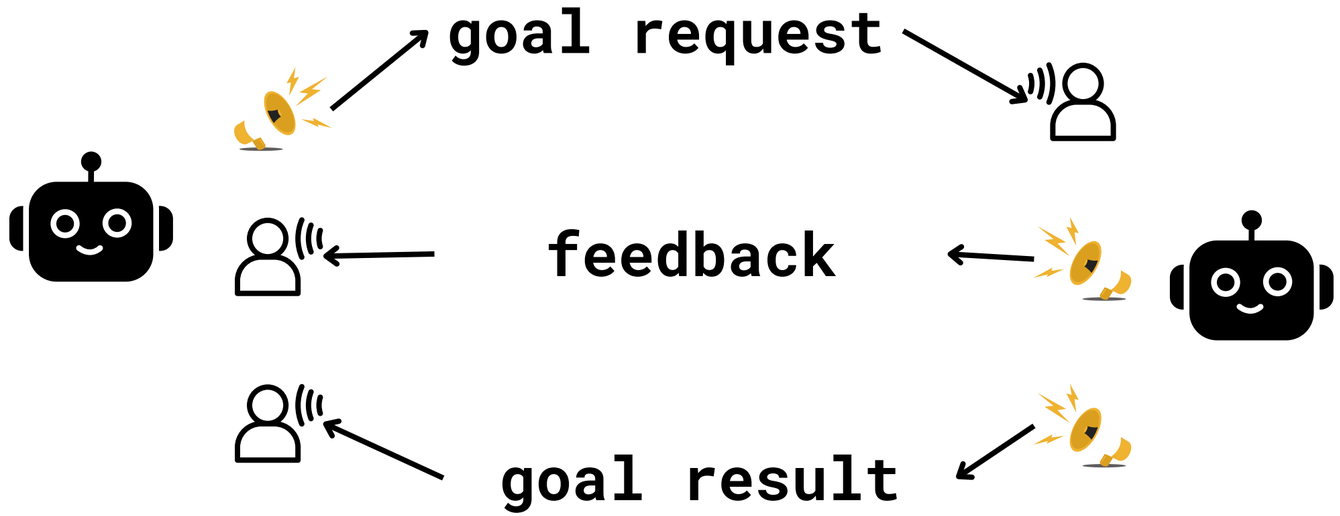
Service

example:
map server



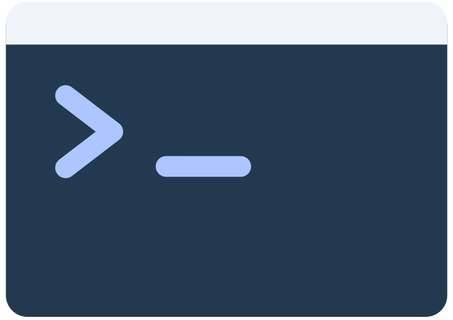
Action

example:
nav2
2d goal pose



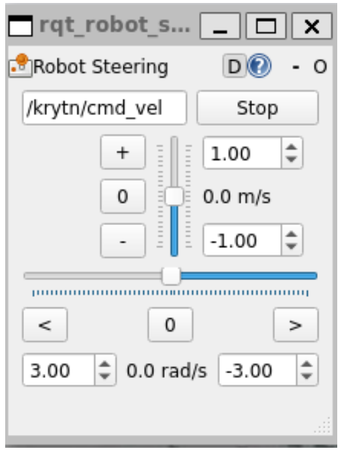
Explore Services and Actions With ros2

Launch
“Day 5”



Run Command

```
ros2 service list
ros2 action list
```



Drive Robot

You may need to run
`source install/setup.bash`

Thank You!

www.roborenegades.com