ROS2 INSTALLATION AND GAZEBO SIMULATION

1. Install ROS2 Humble on Ubuntu 22.04

ROS 2 Installation on Ubuntu 22.04 using WSL on Windows

Introduction:
how to install ROS 2 Humble on a Windows system using WSL (Windows Subsystem for Linux) and Ubuntu 22.04. It allows you to run Linux-based robotics software inside Windows without needing to dual boot.

System Requirements:

- Windows 10 or 11 (64-bit)
- Administrative privileges
- Stable internet connection

Section 1: Install WSL and Ubuntu

1. Open PowerShell as Administrator.

Run the following command to install WSL and Ubuntu 22.04:

```shell
wsl --install -d Ubuntu-22.04
```

Restart your computer when prompted.

2. After reboot, Ubuntu will complete its installation. You'll be asked to create a Linux username and password.

Section 2: Update Ubuntu and Configure Locale

1. Open Ubuntu from the Start menu.

Update the system:

```shell
sudo apt update && sudo apt upgrade -y
```

Set the correct locale:

```shell
sudo apt install locales -y
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8
```

Section 3: Add ROS 2 Package Repositories
Enable universe repository and install required tools:

```shell
sudo apt install software-properties-common -y
sudo add-apt-repository universe
sudo apt update
```

Add the ROS 2 GPG key:

```shell
sudo apt install curl -y
sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key
-o /usr/share/keyrings/ros-archive-keyring.gpg
```

Add the ROS 2 repository:

```shell
echo "deb [arch=$(dpkg --print-architecture)
signed-by=/usr/share/keyrings/ros-archive-keyring.gpg]
 http://packages.ros.org/ros2/ubuntu $(lsb_release -cs) main" |
 sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null
```

Section 4: Install ROS 2 Humble

Update and install ROS 2 desktop version:

```shell
sudo apt update
sudo apt install ros-humble-desktop -y
```

Source ROS setup script:

```shell
echo "source /opt/ros/humble/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Section 5: Test ROS 2 Installation

Open a terminal and run:

```shell
ros2 run demo_nodes_cpp talker
```

Open another terminal and run:

Shell
```
ros2 run demo_nodes_cpp listener
```

If both nodes start successfully and communicate, your ROS 2 installation is complete.

Section 6: (Optional) Install Development Tools

Install useful development tools:

Shell
```
sudo apt install python3-colcon-common-extensions python3-argcomplete -y
```

Getting Started with ROS 2 Humble on Ubuntu 22.04

This document provides a comprehensive guide to installing and working with ROS 2 Humble on Ubuntu 22.04, including setup, workspace creation, package development, and simulation.

1. Install ROS 2 Humble

Step 1: Update and Install Required Packages

Shell
```
sudo apt update && sudo apt upgrade

sudo apt install curl gnupg lsb-release
```

Step 2: Add ROS 2 Repository

Shell

```
curl -sSL https://repo.ros2.org/repos.yaml | sudo tee
/etc/apt/sources.list.d/ros2-latest.list

sudo apt update
```

Step 3: Install ROS 2 Humble Desktop

Shell

```
sudo apt install ros-humble-desktop
```

Step 4: Install Build Dependencies

Shell

```
sudo apt install python3-colcon-common-extensions python3-pip

python3 -m pip install -U setuptools
```

Step 5: Source ROS Environment

Add to ~/.bashrc:

Shell

```
source /opt/ros/humble/setup.bash
```

Apply the change:

Shell

```
source ~/.bashrc
```

2. Create a ROS 2 Workspace

Step 1: Create Workspace Directory

Shell

```
mkdir -p ~/ros2_ws/src

cd ~/ros2_ws
```

Step 2: Build the Workspace

Shell

```
colcon build
```

Step 3: Source the Workspace

Shell

```
source install/setup.bash
```

---

3. Create a ROS 2 Package

Step 1: Create the Package

Shell

```
cd ~/ros2_ws/src

ros2 pkg create --build-type ament_python my_robot_package
```

---

4. Write a Simple ROS 2 Node

Step 1: Create Node File

Navigate to:

Shell
```
cd ~/ros2_ws/src/my_robot_package/my_robot_package
```

Create `simple_node.py`:

Python
```python
import rclpy

from rclpy.node import Node


class SimpleNode(Node):

    def __init__(self):

        super().__init__('simple_node')

        self.get_logger().info('Hello from ROS 2 Node!')


def main():

    rclpy.init()

    node = SimpleNode()

    rclpy.spin(node)

    node.destroy_node()

    rclpy.shutdown()


if __name__ == '__main__':
```

```
    main()
```

Step 2: Make the File Executable

```Shell
chmod +x simple_node.py
```

5. Build the Package

```Shell
cd ~/ros2_ws

colcon build
```

6. Run Your Node

```Shell
source install/setup.bash

ros2 run my_robot_package simple_node
```

7. Communication Between Nodes

Example: Publisher Node

Python

```python
import rclpy

from rclpy.node import Node

from std_msgs.msg import String


class PublisherNode(Node):

    def __init__(self):

        super().__init__('publisher_node')

        self.publisher_ = self.create_publisher(String,
'topic_name', 10)

        self.timer = self.create_timer(1.0, self.timer_callback)


    def timer_callback(self):

        msg = String()

        msg.data = 'Hello ROS 2!'

        self.publisher_.publish(msg)

        self.get_logger().info('Publishing: "%s"' % msg.data)


def main():

    rclpy.init()

    node = PublisherNode()

    rclpy.spin(node)
```

```python
    node.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

8. Simulate with Gazebo (TurtleBot3)

Step 1: Install TurtleBot3 Packages

Shell

```shell
sudo apt install ros-humble-turtlebot3-*
```

Step 2: Launch TurtleBot3 in Gazebo

Shell

```shell
export TURTLEBOT3_MODEL=burger
ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
```

9. Debugging and Visualization Tools

- `ros2 topic list` – list all active topics
- `ros2 topic echo /topic_name` – view messages on a topic
- `ros2 service list` – list available services

- `rqt` – GUI for viewing ROS graph, topics, parameters, etc.

2. Test a ROS2 Node (Built-In)

Open 2 terminals:

Terminal 1: Start Talker

Shell
```shell
ros2 run demo_nodes_cpp talker
```

Terminal 2: Start Listener

Shell
```shell
ros2 run demo_nodes_cpp listener
```

This shows simple publish/subscribe using default packages.

3. Launch TurtleSim Example

Terminal 1:

Shell
```shell
ros2 run turtlesim turtlesim_node
```

Terminal 2:

Shell
```shell
ros2 run turtlesim turtle_teleop_key
```

Use keyboard arrows to move the turtle.

4. Create a ROS2 Workspace

Shell
```
mkdir -p ~/ros2_ws/src
cd ~/ros2_ws

# Install colcon build system
sudo apt install python3-colcon-common-extensions

# Build empty workspace
colcon build

# Source setup
echo "source ~/ros2_ws/install/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

5. Create Your First ROS2 Python Package

Shell
```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_python my_robot_controller
```

Inside `my_robot_controller/`:

- Edit `setup.py`, `package.xml`, `resource/`, and add a `my_robot_controller` directory for your Python code.

Create this file:

`my_robot_controller/my_robot_controller/turtle_controller.py`

Python
```python
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist

class TurtleController(Node):
```

```
    def __init__(self):
        super().__init__('turtle_controller')
        self.publisher_ = self.create_publisher(Twist,
'turtle1/cmd_vel', 10)
        timer_period = 0.5
        self.timer = self.create_timer(timer_period,
self.move_turtle)

    def move_turtle(self):
        msg = Twist()
        msg.linear.x = 2.0
        msg.angular.z = 1.0
        self.publisher_.publish(msg)
        self.get_logger().info(f'Publishing:
linear.x={msg.linear.x} angular.z={msg.angular.z}')

def main(args=None):
    rclpy.init(args=args)
    turtle_controller = TurtleController()
    rclpy.spin(turtle_controller)
    turtle_controller.destroy_node()
    rclpy.shutdown()
```

6. Update setup files

`setup.py`

Python

```python
from setuptools import setup

package_name = 'my_robot_controller'

setup(
    name=package_name,
```

```
    version='0.0.0',
    packages=[package_name],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='your_name',
    maintainer_email='your_email@example.com',
    description='Turtle controller in ROS2',
    license='MIT',
    entry_points={
        'console_scripts': [
            'turtle_controller =
my_robot_controller.turtle_controller:main',
        ],
    },
)
```

`package.xml` (minimal)

```xml
<package format="3">
  <name>my_robot_controller</name>
  <version>0.0.0</version>
  <description>Example ROS2 turtle controller</description>
  <maintainer email="you@example.com">You</maintainer>
  <license>MIT</license>

  <buildtool_depend>ament_python</buildtool_depend>
  <exec_depend>rclpy</exec_depend>
  <exec_depend>geometry_msgs</exec_depend>
</package>
```

7. Build Your Package

```shell
cd ~/ros2_ws
colcon build
source install/setup.bash
```

8. Run Your Node

Start TurtleSim in one terminal:

```shell
ros2 run turtlesim turtlesim_node
```

Run your controller in another:

```shell
ros2 run my_robot_controller turtle_controller
```

Setup ROS2 Workspace and Python Package

STEP 1: Install Colcon + Enable Autocompletion

```shell
# Install colcon build tool
sudo apt update
sudo apt install python3-colcon-common-extensions
```

Enable Colcon Autocompletion

Edit your `.bashrc` file:

```shell
gedit ~/.bashrc
```

Add this line **after ROS2 setup line**:

```shell
source /usr/share/colcon_argcomplete/hook/colcon-argcomplete.bash
```

Then apply changes:

```shell
source ~/.bashrc
```

STEP 2: Create ROS2 Workspace

```shell
# Go to home directory
cd ~

# Create a ROS2 workspace directory
mkdir -p ros2_ws/src
cd ros2_ws

# Build the empty workspace
colcon build
```

You'll now have these folders:

```shell
ros2_ws/
├── build/
├── install/
```

```
├── log/
└── src/
```

STEP 3: Source Your Workspace

To use custom nodes from your workspace:

Shell

```shell
source ~/ros2_ws/install/setup.bash
```

To make it permanent, add it to `.bashrc`:

Shell

```shell
echo "source ~/ros2_ws/install/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

STEP 4: Create a Python Package

Navigate to `src/` inside your workspace:

Shell

```shell
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_python my_robot_controller
--dependencies rclpy
```

This will generate the following structure:

None

```
my_robot_controller/
├── my_robot_controller/
```

```
│   └── __init__.py
├── resource/
│   └── my_robot_controller
├── test/
├── setup.py
├── package.xml
└── setup.cfg
```

STEP 5: Understand Package Purpose

In this step:

- `rclpy` is added as dependency: It is the Python client library for ROS2.
- The package name is `my_robot_controller`, following the common convention:
  - `<robot_name>_<functionality>`, e.g., `my_robot_controller`, `my_robot_camera`, etc.

ROS 2 Development Setup Summary (Terminal Commands + Notes)

Setup ROS 2 Auto-completion

```shell
sudo apt install python3-colcon-common-extensions
```

Add to `~/.bashrc`:

```shell
source /opt/ros/humble/setup.bash
source /usr/share/colcon_argcomplete/hook/colcon-argcomplete.bash
```

Then run:

```shell
source ~/.bashrc
```

Create a ROS 2 Workspace

```shell
cd ~
mkdir -p ros2_ws/src
cd ros2_ws
colcon build
```

Source the workspace (add to `~/.bashrc`):

```shell
source ~/ros2_ws/install/setup.bash
```

Then run:

```shell
source ~/.bashrc
```

Create a Python ROS 2 Package

```shell
cd ~/ros2_ws/src
ros2 pkg create my_robot_controller --build-type ament_python
--dependencies rclpy
```

Build the Package

```shell
cd ~/ros2_ws
```

```
colcon build
```

If build error due to `setuptools`, fix it:

```shell
pip3 install setuptools==58.2.0
```

Create First Python Node

Create file:

```shell
cd ~/ros2_ws/src/my_robot_controller/my_robot_controller
touch my_first_node.py
chmod +x my_first_node.py
```

`my_first_node.py` (Python Node)

```python
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node

class MyNode(Node):
    def __init__(self):
        super().__init__('first_node')
        self.get_logger().info("Hello from ROS2")

def main(args=None):
    rclpy.init(args=args)
    node = MyNode()
    rclpy.spin(node)
```

```
    rclpy.shutdown()
```

Register the Node in `setup.py`

In `~/ros2_ws/src/my_robot_controller/setup.py`:

```python
entry_points={
    'console_scripts': [
        'test_node = my_robot_controller.my_first_node:main',
    ],
},
```

Rebuild After Registering Node

```shell
cd ~/ros2_ws
colcon build
source install/setup.bash
```

Run Your Node (Installed Node)

```shell
ros2 run my_robot_controller test_node
```

ROS 2 Python Publisher Node (Draw a Circle in turtlesim)

1. Create New Python Node

Inside your ROS2 package (`my_robot_controller`):

```Shell
cd ~/ros2_ws/src/my_robot_controller/my_robot_controller
touch draw_circle.py
chmod +x draw_circle.py
```

2. Open the file and write the node code

```Shell
code draw_circle.py
```

Paste this code:

```Python
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist

class DrawCircleNode(Node):
    def __init__(self):
        super().__init__('draw_circle')
        self.get_logger().info("Draw Circle Node has been
started.")

        # Create publisher
        self.cmd_vel_pub = self.create_publisher(Twist,
'/turtle1/cmd_vel', 10)

        # Create timer: 0.5 seconds
        self.timer = self.create_timer(0.5,
self.send_velocity_command)

    def send_velocity_command(self):
        msg = Twist()
```

```
        msg.linear.x = 2.0     # Move forward
        msg.angular.z = 1.0    # Turn to make a circle
        self.cmd_vel_pub.publish(msg)

def main(args=None):
    rclpy.init(args=args)
    node = DrawCircleNode()
    rclpy.spin(node)
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

3. Update `package.xml` dependencies

In `package.xml`, ensure these dependencies are added:

XML

```xml
<exec_depend>rclpy</exec_depend>
<exec_depend>geometry_msgs</exec_depend>
<exec_depend>turtlesim</exec_depend>
```

4. Update `setup.py` to install the new executable

Inside `setup.py`, find the `entry_points` and add your new node:

Python

```python
entry_points={
    'console_scripts': [
        'draw_circle = my_robot_controller.draw_circle:main',
    ],
},
```

5. **Build the workspace with symlink for Python**

```shell
cd ~/ros2_ws
colcon build --symlink-install
```

6. **Source the workspace**

```shell
source install/setup.bash
```

(You can add this to `.bashrc` for automatic sourcing.)

7. **Run the turtlesim and your new node**

In Terminal 1:

```shell
ros2 run turtlesim turtlesim_node
```

In Terminal 2:

```shell
ros2 run my_robot_controller draw_circle
```

File: `turtle_controller.py`

1. Basic Setup

Python

```
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node
from turtlesim.msg import Pose
from geometry_msgs.msg import Twist

class TurtleControllerNode(Node):

    def __init__(self):
        super().__init__("turtle_controller")
        self.get_logger().info("Turtle Controller has been
started")

        # Create publisher for velocity commands
        self.cmd_vel_publisher = self.create_publisher(
            Twist,
            "/turtle1/cmd_vel",
            10
        )

        # Create subscriber for turtle pose
        self.pose_subscriber = self.create_subscription(
            Pose,
            "/turtle1/pose",
            self.pose_callback,
            10
        )

    def pose_callback(self, msg: Pose):
        # Create Twist message based on turtle's current position
        twist_msg = Twist()

        # Example logic: move forward and turn if near border
        if msg.x > 9.0 or msg.x < 2.0 or msg.y > 9.0 or msg.y <
2.0:
```

```
            twist_msg.angular.z = 2.0
            twist_msg.linear.x = 0.0
        else:
            twist_msg.linear.x = 2.0
            twist_msg.angular.z = 0.0

        self.cmd_vel_publisher.publish(twist_msg)

def main(args=None):
    rclpy.init(args=args)
    node = TurtleControllerNode()
    rclpy.spin(node)
    rclpy.shutdown()
```

`setup.py` Edit

Add this line under `entry_points > console_scripts`:

```python
'turtle_controller = my_robot_controller.turtle_controller:main',
```

Make sure you also have this dependency in your `package.xml`:

```xml
<exec_depend>turtlesim</exec_depend>
<exec_depend>geometry_msgs</exec_depend>
```

Terminal Commands Recap

1. Create the file and make executable:

```shell
cd ~/ros2_ws/src/my_robot_controller/my_robot_controller
touch turtle_controller.py
chmod +x turtle_controller.py
```

2. Open in VS Code:

```shell
cd ~/ros2_ws/src/my_robot_controller/my_robot_controller
code .
```

3. Build the workspace:

```shell
cd ~/ros2_ws
colcon build --symlink-install
source install/setup.bash
```

4. Run it:

```shell
ros2 run turtlesim turtlesim_node
ros2 run my_robot_controller turtle_controller
```

Test with `rqt_graph`

```shell
rqt_graph
```

`Python ROS2 Service Client for Set Pen`

Goal:

Change the turtle's pen color based on its X position:

- If x > 5.5 → Red
- If x <= 5.5 → Green

Step-by-Step Code Walkthrough:

In your TurtleController class, inside your Python node file (e.g., turtle_controller.py), add this method:

```python
def call_set_pen_service(self, r: int, g: int, b: int, width: int, off: int):

    from turtlesim.srv import SetPen


    client = self.create_client(SetPen, 'turtle1/set_pen')


    while not client.wait_for_service(timeout_sec=1.0):

        self.get_logger().info('Service not available, waiting...')


    request = SetPen.Request()

    request.r = r

    request.g = g
```

```
        request.b = b

        request.width = width

        request.off = off


        future = client.call_async(request)


        # Optional: You can add a callback or handle the
    response later
```

Modify Your pose_callback to Use This:

Now go to your pose_callback (where the turtle's current position is checked) and add:

```python
    if pose.x > 5.5:

        self.call_set_pen_service(255, 0, 0, 3, 0)  # Red

    else:

        self.call_set_pen_service(0, 255, 0, 3, 0)  # Green
```

To avoid calling the service every single pose update (which would be a lot), you should track the current color state and only call the service when the color needs to change.

Optimization with a State Variable:

At the top of your class (__init__):

```python
self.current_color = 'none'
```

And in the pose_callback:

```python
if pose.x > 5.5 and self.current_color != 'red':
    self.call_set_pen_service(255, 0, 0, 3, 0)
    self.current_color = 'red'
elif pose.x <= 5.5 and self.current_color != 'green':
    self.call_set_pen_service(0, 255, 0, 3, 0)
    self.current_color = 'green'
```

Dependencies and Imports:

At the top of your file (add if not present):

```python
from turtlesim.srv import SetPen
```

Make sure you have added the dependency in your package.xml:

```xml
<exec_depend>turtlesim</exec_depend>
```

And in CMakeLists.txt, if you use it:

```
None
    find_package(turtlesim REQUIRED)
```

Objective:

Change the Turtle's pen color based on its **X position** in the window (left = green, right = red) using a **ROS2 service client**.

ROS2 Service Client Setup:

- Used `SetPen` from `turtlesim.srv` to change pen color.
- Created service client using:

```python
self.create_client(SetPen, '/turtle1/set_pen')
```

2. **Request Handling:**

- Built and sent a request with:

```python
req = SetPen.Request()
req.r, req.g, req.b, req.width, req.off = 255, 0, 0, 3, 0  #
Example
```

Asynchronous Call:

- Used `call_async()` to avoid blocking the thread.
- Added callback with `future.add_done_callback()` using `functools.partial`.

Callback Logic:

- Processed the `future.result()` in a `try/except` block.
- Logged success or error messages.

Optimized Call Frequency:

- Avoided calling the service on every pose update (which happens at 60 Hz).
- Only called when the turtle **crosses** the midpoint (`x = 5.5`).

ROS Topic Debugging:

- Used:

Shell
```
ros2 topic hz /turtle1/pose
```

to confirm publishing frequency (~60 Hz).

Final Node Logic:

```python
if pose.x > 5.5 and self.previous_x <= 5.5:
    self.call_set_pen_service(255, 0, 0, 3, 0)  # Red
elif pose.x <= 5.5 and self.previous_x > 5.5:
    self.call_set_pen_service(0, 255, 0, 3, 0)  # Green

self.previous_x = pose.x
```

Commands to Run Everything:

```shell
# Start turtlesim
ros2 run turtlesim turtlesim_node

# Source workspace
source install/setup.bash

# Run your controller node
ros2 run my_robot_controller turtle_controller
```