**Week: 08/25/24**

Started researching ROS, TurtleBot3, and OpenManipulatorX.

* <https://www.ros.org/>
* <https://emanual.robotis.com/docs/en/platform/turtlebot3/features/>
* <https://emanual.robotis.com/docs/en/platform/openmanipulator_x/overview/>

**Week: 09/01/24**

Installed ROBOTC and Curriculum Virtual World

* Worked in the simulation environment
* Modeled a manipulator in C
* ROBOTC dead ended because we do not have a build license to work with our own models

**Week: 09/08/24**

ROS2 Humble

* Installed ROS2 for Windows 10
  + <https://docs.ros.org/en/humble/Installation/Alternatives/Windows-Development-Setup.html>
* Windows subsystem for Linux
  + <https://docs.ros.org/en/humble/Installation/Alternatives/Ubuntu-Development-Setup.html>
* Cannot run NVIDIA’s Isaac Simulator on my laptop due to graphics limitations
  + <https://developer.nvidia.com/isaac/ros>

**Week: 09/15/24**

Worked in ROS2 to create and test Python nodes [1]

* <https://www.youtube.com/playlist?list=PLLSegLrePWgJudpPUof4-nVFHGkB62Izy>

**Week: 09/22/24**

Worked in Gazebo to create worlds, models, and combine them together

* <https://youtu.be/YV8hlpBOhtw?si=R5v1UjO6sRUrpDBo> (World)
* <https://www.youtube.com/watch?v=_qQAfTmB5wc&t=438s> (Model)

**Week: 09/29/24**

Modeled a robotic manipulator arm in URDF and imported packages allowing for control of the system (Inertia added later: see [2])

* <https://youtu.be/t67JaKiZY_U?si=-X0nV7aKxBJ8Jc9e>
* <https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Colcon-Tutorial.html> (build package documentation)

**Week: 10/06/24**

Implemented customized control for manipulator model designed with nodes and imported and modified the OpenManipulatorX URDF files [2 – urdf/xacro design] [3 – intertia]

* Browser URDF model viewer: <https://gkjohnson.github.io/urdf-loaders/javascript/example/bundle/index.html>
* Node control series
  + <https://www.youtube.com/watch?v=OWeLUSzxMsw>
  + <https://www.youtube.com/watch?v=BcjHyhV0kIs>
  + <https://www.youtube.com/watch?v=IjFcr5r0nMs>
* OpenManipulatorX URDF
  + <https://github.com/hylander2126/OpenManipulatorX_ROS2/tree/main>
  + <https://github.com/husarion/rosbot_xl_manipulation_ros> (rosbot with manipulator)
  + <https://github.com/husarion/open_manipulator_x/blob/main/README.md> (manipulator only, no controller)

**Week 10/13/24**

Implement the OpenManipulatorX and TurtleBot3 Waffle Pi model together

* <https://medium.com/@nilutpolkashyap/setting-up-turtlebot3-simulation-in-ros-2-humble-hawksbill-70a6fcdaf5de> (turtlebot3 waffle pi)
* <https://emanual.robotis.com/docs/en/platform/turtlebot3/manipulation/> (combination)

Decided on application for robot unit

* <https://ieeexplore.ieee.org/document/8612150> (sorting manipulator) (by color and contour)
* <https://ieeexplore.ieee.org/document/7847799> (voice command manipulator) (takes very rudimentary commands – requires microphone)
* <https://ieeexplore.ieee.org/document/9461187> (silicon wafer grabber) (very cool, but vacuum based manipulator)

**Week 10/20/24**

Find more papers and do an in depth dive on their findings

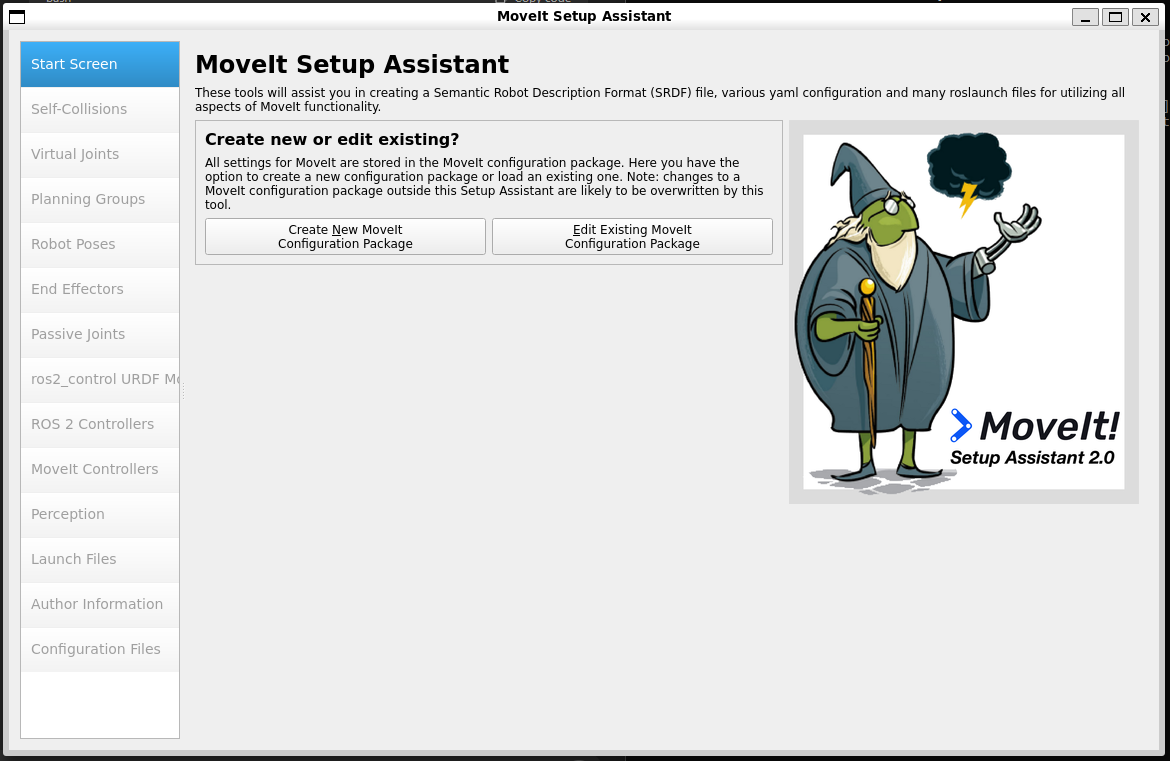
* <https://www.emerald.com/insight/content/doi/10.1108/IR-10-2020-0242/full/html#sec010>
* <https://openaccess.thecvf.com/content/WACV2023/papers/Griffin_Mobile_Robot_Manipulation_Using_Pure_Object_Detection_WACV_2023_paper.pdf>

Determine End Objective

* Implement a robotic manipulator and mobile robot together
* Achieve autonomous navigation
  + Apply machine vision and machine learning to an integrated robotic system
  + Train the model with a predefined dataset
* Pick and place system
  + Deploy the system in a physical environment

**Week 10/27/24**

Have the mobile manipulator travel from one point to pick up an object to either a new point or back to the start point

* Preface:
  + There are 3 packages in my workspace’s src directory:
    - Turtlebot3\_manipulation\_bringup: spawns the model in Gazebo and contains the node scripts
    - Turtlebot3\_manipulation\_description: reference for the urdf and xacro files
    - Manual\_moveit\_config: move it configuration specifies joint communications and how they should interact
  + Important to note:
    - For each terminal you open with wsl (in this project specifically), you may need to colcon build --allow-overriding turtlebot3\_manipulation\_description, since we are using package sharing. This will occur if you have other packages of the same name elsewhere
    - And don’t forget to source install/setup.bash for each terminal also to ensure the nodes running on each can communicate with each other
* **Step 1**: environment setup – get turtlebot3/manipulatorX model spawned in a Gazebo world
  + In Turtlebot3\_manipulation\_bringup package’s launch directory, *gazebo.launch.py* script spawns the manipulator in a specified Gazebo world [4]
    - Note it may not look like [4] at this step, since this is after I’ve performed future steps as well
  + ros2 launch turtlebot3\_manipulation\_bringup gazebo.launch.py
* **Step 2**: MoveIt configuration - call the MoveIt specification menu and create a new configuration package
  + ros2 launch moveit\_setup\_assistant setup\_assistant.launch.py
  + 



* + Add the path to the urdf or xacro file of your model (mine was in src/turtlebot3\_manipulation\_description/urdf/turtlebot3\_manipulation.urdf.xacro)

A screenshot of a computer program

Description automatically generated

* + Auto generate self collision matrix

A screenshot of a computer

Description automatically generated



* + Define virtual joints to connect robot to world

A screenshot of a computer

Description automatically generated

* + Define planning groups – in this case, our joints work independently (differential drive) so specify them one by one

A screenshot of a computer

Description automatically generated

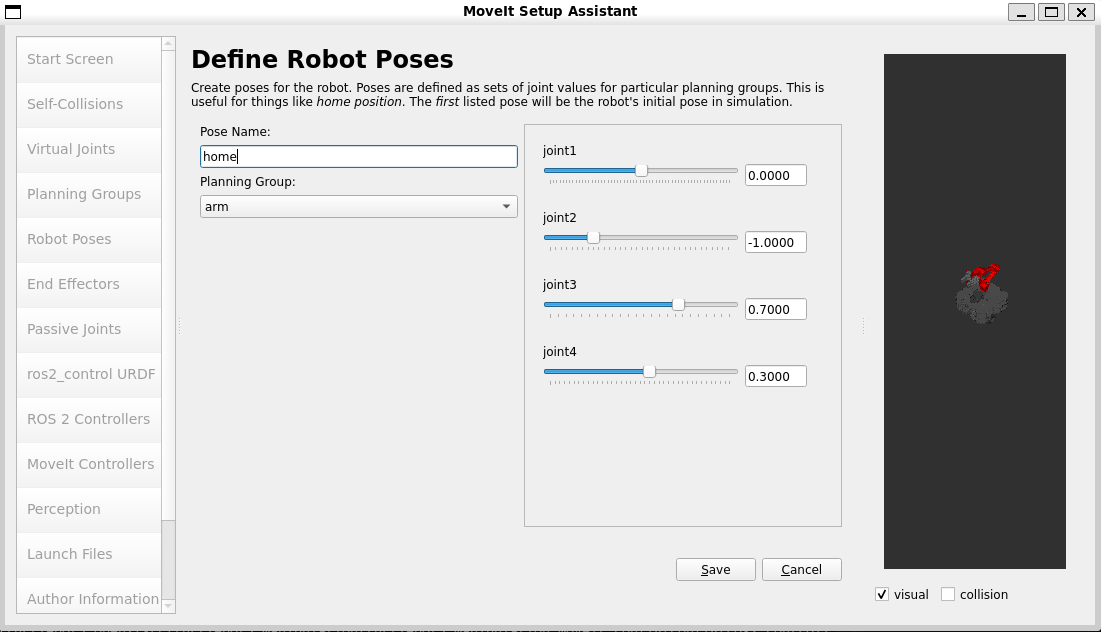


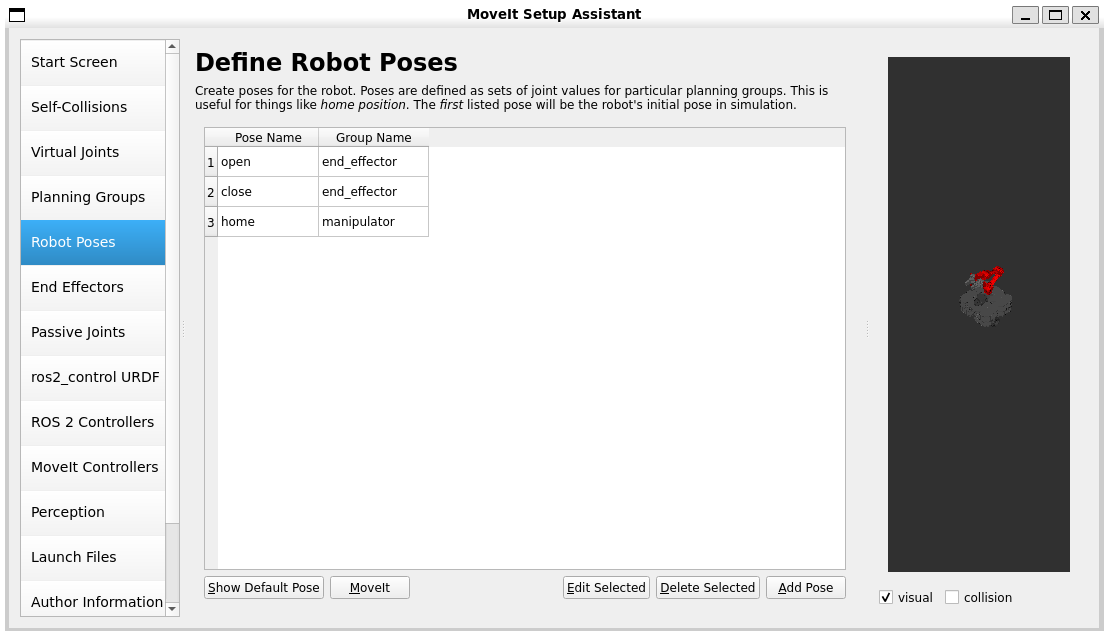
* + Define the end effector

A screenshot of a computer

Description automatically generated

* + Define poses for open and closed hand





* + Generate the configuration files (destination should be your workspace src folder, since it’s its own package)

A screenshot of a computer

Description automatically generated

* + In the package directory, you will need to alter the package.xml by entering maintenance and author emails and uncommenting the lines that will have MoveIt communicate with Gazebo (they are specified in the comments)
    - <maintainer email="elirel973@outlook.com">Ellie Ireland</maintainer>
  + You’ll want to add two more files to the config directory:
    - controllers.yaml [5]
    - ompl\_planning.yaml [6]
  + In the launch directory (in the manual\_moveit\_config library), you will need to update the move\_group.launch.py [7]
* **Step 3**: Create command script nodes in the Turtlebot3\_manipulation\_bringup package *scripts* directory
  + With gazebo.launch.py running, in another terminal, run the script node you want to execute (don’t forget to chmod +x <*path/script\_name.py*>)
  + My scripts
    - *move\_sequence.py*: moves the robot model indefinitely [8]
    - *move\_and\_grasp.py*: moves the base, moves the arm, moves the base again (having difficulty with grasping tasks) [9]

**Week 11/03/24**

Work on C-Day application poster/website

Research implementing machine vision for the mobile manipulator

* ROS 1: <https://emanual.robotis.com/docs/en/platform/turtlebot3/machine_learning/#software-setup>
* ROS 2: <https://www.youtube.com/watch?v=KEObIB7RbH0>

**Week 11/10/24**

Autonomous navigation should not be learning-based; should be based on a known map

* <https://www.youtube.com/watch?v=4OfCKg9vSVc> (referenced not used)
* <https://www.youtube.com/watch?v=jkoGkAd0GYk&t=8s> (referenced not used)
* Resources for Implementation
  + <https://docs.nav2.org/getting_started/index.html>
  + <https://emanual.robotis.com/docs/en/platform/turtlebot3/manipulation/>
  + <https://neobotix-docs.de/ros/ros2/autonomous_navigation.html#:~:text=In%20order%20to%20select%20the%20multiple%20goals.%20First%2C,is%20required%20to%20press%20the%20%E2%80%9CNav2%20goal%E2%80%9D%20button>.
  + Used the original model and packages

Object Recognition

* YOLOv8 model
  + <https://medium.com/discover-computer-vision/a-real-time-object-detection-model-with-yolov3-algorithm-for-non-gpu-computers-d999283542b2>
  + <https://www.youtube.com/watch?v=7n6gCqC075g&t=518s>
* Customizing the training model
  + Add your images (save some for testing in the data\_for\_test directory)
    - Once you clone the yolov8obb\_training, remove all the files in dataset/bolts\_dataset/images/train and in dataset/bolts\_dataset/images/val
    - Copy your own images into the /train directory (make sure to use png)
  + Specify the classes you want to identify
    - labelImg2 > data > predefined\_classes.txt > add different classes
    - yolov8obb\_training > train\_info.yaml > add different classes (and paths)
    - yolov8obb\_training > format\_converter.py > add different classes
  + classify your images
    - python3 labelImg2/labelImg.py
    - add bounding boxes (must be the rotating ones)
    - don’t forget to specify classes for each box
    - python3 yolov5-utils/voc2yolo5\_obb.py --path yolov8obb\_training/datasets/bolts\_dataset/images/train/ --class-file labelImg2/data/predefined\_classes.txt
  + organize the files
    - move a few images (just the png files) from yolov8obb\_training/dataset/bolts\_dataset/images/train to images/val directory
    - move the respective .txt and .xml files for those images to the yolov8obb\_training/dataset/bolts\_dataset/labels/val\_original
    - copy the .txt and .xml files for all images in the /images/train directory to the /labels/train\_original
  + python3 format\_converter.py
    - this will populate the images/labels/train and the images/labels/val
  + python3 yolov8\_obb\_train.py
    - this will train the model
    - move the best.pt file from the runs/obb/train directory to the yolov8obb\_training directory
  + python3 yolov8\_obb\_inference.py
    - make sure to alter this depending on which test image you want to be run – should ideally be pngs
    - outputs will save to the runs/obb/predict directory

Integration (1)

* move the object recognition directory (yolov8obb\_training) to the turtlebot3 autonomous navigation workspace that has been developed
  + change paths in train\_info.yaml
* create new script gazebo\_stream.py [10] in the object recognition directory (that’s inside the navigation workspace directory)
  + this model will grab images from the Waffle Pi camera and run them through the detection model
* note: if ROS is bugging, check camera stream with ros2 run rqt\_image\_view rqt\_image\_view
  + if the view is not displayed correctly, restart gazebo and gazebo\_stream.py
* Can’t stream directly to rqt\_image\_view because we’re on WSL
  + Add web\_server.py [11] that streams camera output via grabbing periodic image
  + Access via browser by <http://localhost:5000/>

Instructions for Running:

* cd ty/
* ros2 launch turtlebot3\_manipulation\_moveit\_config servo.launch.py
* ros2 launch turtlebot3\_manipulation\_bringup gazebo.launch.py
* ros2 run tf2\_ros static\_transform\_publisher 0 0 0 0 0 0 map odom
* ros2 launch turtlebot3\_manipulation\_navigation2 navigation2.launch.py map\_yaml\_file:=$HOME/map.yaml
* rm -r ty/runs
* python3 src/y/gazebo\_stream.py
* python3 src/y/web\_server.py
* http://localhost:5000/

Pose Estimation

* <https://github.com/atenpas/gpd?tab=readme-ov-file#pcd> (ROS1)
* <https://moveit.picknik.ai/humble/doc/tutorials/pick_and_place_with_moveit_task_constructor/pick_and_place_with_moveit_task_constructor.html> (Can’t build ??)
* <https://arxiv.org/html/2411.04386v1> (Paper)
* <https://link.springer.com/chapter/10.1007/978-3-319-23778-7_41> (Paper)
* <https://www.youtube.com/watch?v=mlXs5kIQ5p4> (Isaac Sim)

**Notes:**

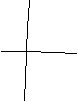
**09/29/24**

*wsl* in Windows cmd: launch windows subsystem for Linux

Create workspace:

* mkdir -p ~/*ws*/src
* cd ~/*ws*/src
  + from scratch:
    - ros2 pkg create --build-type ament\_cmake *packageName*
    - cd ~/*ws*/src/*packageName*
    - mkdir launch urdf config
  + from git:
    - git clone <*github link*>
* cs ~/*ws*
* colcon build
* source install/setup.bash

Joints:



**10/06/24**

Node – any program that has access to the ROS functionality and communications

Topics – used to send data streams

Services – used for interactions

* rviz2 -d src/my\_bot/config/view\_bot.rviz

**10/13/24**

Reachable spaces – areas within the manipulator’s reach

**10/20/24**

Differential Drive – motors are independent of each other

Mobile robot sensors

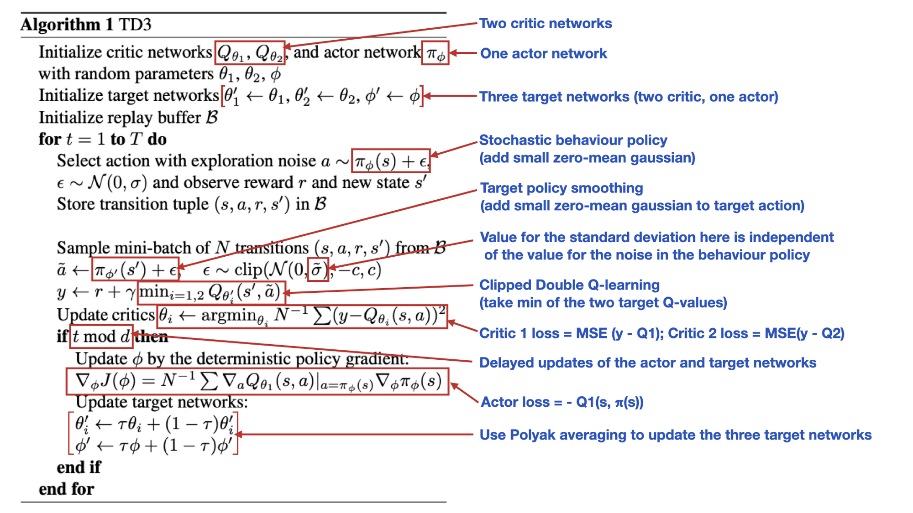
* Proprioceptive Sensors - tell how much the robot has moved (shaft encoders)
  + Dead Reckoning – tells how far the robot has gone and how much it has rotated (use IMU)
  + Odometry - use of data from motion sensors to estimate change in position over time (can be faulty)
* Exteroceptive Sensors – gives perception of world (camera)
* Exteroceptive Sensors – allow for operation in dynamic environments (lidar, radar, sonar)

**11/03/24**

Vocab

* **Q-value function** – function that estimates the expected future rewards an agent can obtain by taking specific action in a specific state *s*
* **Critic network** – neural network that approximates the Q-value function
* **Actor** – neural network that represents the policy of the agent
* **Policy** – strategy that the agent employs to determine its actions given a state
  + Can be deterministic or stochastic
  + **Deterministic** – actor directly outputs a specific action for each state
  + **Stochastic** – policy outputs a probability distribution over possible actions
* **Target networks** – copies of the main actor and critic networks used to stabilize training

**Twin Delayed Deep Deterministic Policy** – good for continuous control problems like autonomous robot driving

* Twin (two) critic networks (Q value functions)
* Delayed updates from the actor for the target and policy
* Action noise regularization so policy is less likely to exploit actions with high Q-value estimates
* 

<https://towardsdatascience.com/td3-learning-to-run-with-ai-40dfc512f93>

<https://www.mathworks.com/help/reinforcement-learning/ug/td3-agents.html>

<https://saashanair.com/blog/blog-posts/twin-delayed-ddpg-td3-how-does-the-algorithm-work#block-5c32a3cf6b624bfe848b44ceb795399d>

**11/17/24**

Ideally don’t tinker with the display settings too much without knowing how to fix them, but if you do, this solved my problems of Gazebo/Rviz2 not launching GUIs:

* echo 'export LIBGL\_ALWAYS\_INDIRECT=0' >> ~/.bashrc
* echo 'export LIBGL\_ALWAYS\_SOFTWARE=1' >> ~/.bashrc
* source ~/.bashrc

**Code:**

[1] my\_first\_node.py

#!/usr/bin/env python3

import rclpy

from rclpy.node import Node

class MyNode(Node):

    def \_\_init\_\_(self):

        #node name: first\_node

        super().\_\_init\_\_("first\_node")

        self.counter\_ = 0

        self.create\_timer(1.0, self.timer\_callback)

    def timer\_callback(self):

        self.get\_logger().info("Hello " + str(self.counter\_))

        self.counter\_ += 1

def main(args=None):

    rclpy.init(args=args)

    node = MyNode()

    rclpy.spin(node)

    rclpy.shutdown()

if \_\_name\_\_ == '\_\_main\_\_':

    main()

[2] robot\_core.xacro

<?xml version="1.0"?>

<robot xmlns:xacro="http://www.ros.org/wiki/xacro">

    <xacro:include filename="inertial\_macros.xacro"/>

    <!-- define colors -->

    <!-- rgba: red green blue transparency -->

    <material name="blue">

        <color rgba="0 0 0.8 1"/>

    </material>

    <material name="red">

        <color rgba="0.8 0 0 1"/>

    </material>

    <material name="green">

        <color rgba="0 0.8 0 1"/>

    </material>

    <material name="yellow">

        <color rgba="1 1 0 1"/>

    </material>

    <material name="cyan">

        <color rgba="0 1 1 1"/>

    </material>

    <!-- links: connected together with joints -->

    <!-- joints -->

    <!-- origin: defines position and orientation of joint relative to parent link -->

    <!-- axis: defines axis of rotation -->

    <!-- limit: defines limits for joint's motion -->

    <!-- world link to connect robot to the world -->

    <link name="world"/>

    <!-- fixed\_joint: fixed to connect base\_link to world -->

    <joint name="fixed\_joint" type="fixed">

        <!--rpy: roll pitch yaw -->

        <!-- xyz: coordinate displacement -->

        <origin rpy="0 0 0" xyz="0 0 0"/>

        <parent link="world"/>

        <child link="base\_link"/>

    </joint>

    <!-- base\_link of manipulator -->

    <link name="base\_link">

        <visual>

            <geometry>

                <cylinder length="0.2" radius="0.4"/>

            </geometry>

            <origin rpy="0 0 0" xyz="0 0 0.1"/>

            <material name="green"/>

        </visual>

        <collision>

            <geometry>

                <cylinder length="0.2" radius="0.4"/>

            </geometry>

            <origin rpy="0 0 0" xyz="0 0 0.1"/>

        </collision>

        <xacro:inertial\_cylinder mass="2.5" length="0.2" radius="0.4">

            <origin rpy="0 0 0" xyz="0 0 0.1"/>

        </xacro:inertial\_cylinder>

    </link>

    <gazebo reference="base\_link">

        <material>Gazebo/Green</material>

    </gazebo>

    <!-- joint\_1: axial revolute to connect base to link\_1 -->

    <joint name="joint\_1" type="revolute">

        <parent link="base\_link"/>

        <child link="link\_1"/>

        <origin rpy="0 0 0" xyz="0 0 0.15"/>

        <axis xyz="0 0 1"/>

        <limit effort="300" velocity="2.0" lower="-3.141593" upper="3.141593"/>

    </joint>

    <!-- link\_1 of manipulator -->

    <link name="link\_1">

        <visual>

            <geometry>

                <cylinder length="0.6" radius="0.15"/>

            </geometry>

            <origin rpy="0 0 0" xyz="0 0 0.3"/>

            <material name="yellow"/>

        </visual>

        <collision>

            <geometry>

                <cylinder length="0.6" radius="0.15"/>

            </geometry>

            <origin rpy="0 0 0" xyz="0 0 0.3"/>

        </collision>

        <xacro:inertial\_cylinder mass="0.3" length="0.6" radius="0.15">

            <origin rpy="0 0 0" xyz="0 0 0.3"/>

        </xacro:inertial\_cylinder>

    </link>

    <gazebo reference="link\_1">

        <material>Gazebo/Yellow</material>

    </gazebo>

    <!-- joint\_2: transverse revolute to connect link\_1 to link\_2 -->

    <joint name="joint\_2" type="revolute">

        <parent link="link\_1"/>

        <child link="link\_2"/>

        <origin rpy="0 -1.570796 0" xyz="-0.15 0 0.5"/>

        <axis xyz="0 0 1"/>

        <limit effort="300" velocity="2.0" lower="-3.141593" upper="3.141593"/>

    </joint>

    <!-- link\_2 of manipulator -->

    <link name="link\_2">

        <visual>

            <geometry>

                <cylinder length="1" radius="0.1"/>

            </geometry>

            <origin rpy="0 1.570796 0" xyz="0.3 0 0.1"/>

            <material name="red"/>

        </visual>

        <collision>

            <geometry>

                <cylinder length="1" radius="0.1"/>

            </geometry>

            <origin rpy="0 1.570796 0" xyz="0.3 0 0.1"/>

        </collision>

        <xacro:inertial\_cylinder mass="0.1" length="1" radius="0.1">

            <origin rpy="0 1.570796 0" xyz="0.3 0 0.1"/>

        </xacro:inertial\_cylinder>

    </link>

    <gazebo reference="link\_2">

        <material>Gazebo/Red</material>

    </gazebo>

    <!-- joint\_3: transverse revolute to connect link\_2 to link\_3 -->

    <joint name="joint\_3" type="revolute">

        <parent link="link\_2"/>

        <child link="link\_3"/>

        <origin rpy="0 0 0" xyz="0.7 0 0"/>

        <axis xyz="0 0 1"/>

        <limit effort="300" velocity="2.0" lower="-3.141593" upper="3.141593"/>

    </joint>

    <!-- link\_3 of manipulator -->

    <link name="link\_3">

        <visual>

            <geometry>

                <cylinder length="1" radius="0.1"/>

            </geometry>

            <origin rpy="0 1.570796 0" xyz="0.4 0 -0.1"/>

            <material name="blue"/>

        </visual>

        <collision>

            <geometry>

                <cylinder length="1" radius="0.1"/>

            </geometry>

            <origin rpy="0 1.570796 0" xyz="0.4 0 -0.1"/>

        </collision>

        <xacro:inertial\_cylinder mass="0.05" length="1" radius="0.1">

            <origin rpy="0 1.570796 0" xyz="0.4 0 -0.1"/>

        </xacro:inertial\_cylinder>

    </link>

    <gazebo reference="link\_3">

        <material>Gazebo/Blue</material>

    </gazebo>

    <!-- joint\_4: axial revolute to connect link\_3 to end effector -->

    <joint name="joint\_4" type="revolute">

        <parent link="link\_3"/>

        <child link="link\_4"/>

        <origin rpy="-1.570796 0 0" xyz="1 0 -0.1"/>

        <axis xyz="0 0 1"/>

        <limit effort="300" velocity="2.0" lower="-3.141593" upper="3.141593"/>

    </joint>

    <!-- link\_4 of manipulator (end effector) -->

    <link name="link\_4">

        <visual>

            <geometry>

                <cylinder length="0.2" radius="0.1"/>

            </geometry>

            <origin rpy="0 0 0" xyz="0 0 0.1"/>

            <material name="cyan"/>

        </visual>

        <collision>

            <geometry>

                <cylinder length="0.2" radius="0.1"/>

            </geometry>

            <origin rpy="0 0 0" xyz="0 0 0.1"/>

        </collision>

        <xacro:inertial\_cylinder mass="0.025" length="0.2" radius="0.1">

            <origin rpy="0 0 0" xyz="0 0 0.1"/>

        </xacro:inertial\_cylinder>

    </link>

    <gazebo reference="link\_4">

        <material>Gazebo/Turquoise</material>

    </gazebo>

    <!-- joint\_5: prismatic moves manipulator's finger (link\_5) -->

    <joint name="joint\_5" type="prismatic">

        <parent link="link\_4"/>

        <child link="link\_5"/>

        <origin rpy="0 0 0" xyz="0.08 0 0.2"/>

        <axis xyz="1 0 0"/>

        <limit effort="300" velocity="2.0" lower="-0.12" upper="0"/>

    </joint>

    <!-- link\_5 of manipulator (finger 1) -->

    <link name="link\_5">

        <visual>

            <geometry>

                <box size="0.03 0.06 0.15"/>

            </geometry>

            <origin rpy="0 0 0" xyz="0 0 0.075"/>

            <material name="green"/>

        </visual>

        <collision>

            <geometry>

                <box size="0.03 0.06 0.15"/>

            </geometry>

            <origin rpy="0 0 0" xyz="0 0 0.075"/>

        </collision>

        <xacro:inertial\_box mass="0.025" x="0.03" y="0.06" z="0.15">

            <origin rpy="0 0 0" xyz="0 0 0.05"/>

        </xacro:inertial\_box>

    </link>

    <gazebo reference="link\_5">

        <material>Gazebo/Green</material>

    </gazebo>

    <!-- joint\_6: fixed to connect link\_6 to link\_4 -->

    <joint name="joint\_6" type="fixed">

        <parent link="link\_4"/>

        <child link="link\_6"/>

        <origin rpy="0 0 0" xyz="-0.08 0 0.2"/>

    </joint>

    <!-- link\_6 of manipulator (finger 2) -->

    <link name="link\_6">

        <visual>

            <geometry>

                <box size="0.03 0.06 0.15"/>

            </geometry>

            <origin rpy="0 0 0" xyz="0 0 0.075"/>

            <material name="green"/>

        </visual>

        <collision>

            <geometry>

                <box size="0.03 0.06 0.15"/>

            </geometry>

            <origin rpy="0 0 0" xyz="0 0 0.075"/>

        </collision>

        <xacro:inertial\_box mass="0.02" x="0.03" y="0.06" z="0.15">

            <origin rpy="0 0 0" xyz="0 0 0.05"/>

        </xacro:inertial\_box>

    </link>

    <gazebo reference="link\_6">

        <material>Gazebo/Green</material>

    </gazebo>

</robot>

[3] inertial\_macros.xacro:

<?xml version="1.0"?>

<robot xmlns:xacro="http://www.ros.org/wiki/xacro" >

    <!-- Specify some standard inertial calculations https://en.wikipedia.org/wiki/List\_of\_moments\_of\_inertia -->

    <!-- These make use of xacro's mathematical functionality -->

    <xacro:macro name="inertial\_sphere" params="mass radius \*origin">

        <inertial>

            <xacro:insert\_block name="origin"/>

            <mass value="${mass}" />

            <inertia ixx="${(2/5) \* mass \* (radius\*radius)}" ixy="0.0" ixz="0.0"

                    iyy="${(2/5) \* mass \* (radius\*radius)}" iyz="0.0"

                    izz="${(2/5) \* mass \* (radius\*radius)}" />

        </inertial>

    </xacro:macro>

    <xacro:macro name="inertial\_box" params="mass x y z \*origin">

        <inertial>

            <xacro:insert\_block name="origin"/>

            <mass value="${mass}" />

            <inertia ixx="${(1/12) \* mass \* (y\*y+z\*z)}" ixy="0.0" ixz="0.0"

                    iyy="${(1/12) \* mass \* (x\*x+z\*z)}" iyz="0.0"

                    izz="${(1/12) \* mass \* (x\*x+y\*y)}" />

        </inertial>

    </xacro:macro>

    <xacro:macro name="inertial\_cylinder" params="mass length radius \*origin">

        <inertial>

            <xacro:insert\_block name="origin"/>

            <mass value="${mass}" />

            <inertia ixx="${(1/12) \* mass \* (3\*radius\*radius + length\*length)}" ixy="0.0" ixz="0.0"

                    iyy="${(1/12) \* mass \* (3\*radius\*radius + length\*length)}" iyz="0.0"

                    izz="${(1/2) \* mass \* (radius\*radius)}" />

        </inertial>

    </xacro:macro>

</robot>

[4] gazebo.launch.py

import os

from launch import LaunchDescription

from launch.actions import DeclareLaunchArgument

from launch.actions import IncludeLaunchDescription

from launch.launch\_description\_sources import PythonLaunchDescriptionSource

from launch.substitutions import LaunchConfiguration

from launch.substitutions import PathJoinSubstitution

from launch.substitutions import ThisLaunchFileDir

from launch\_ros.actions import Node

from launch\_ros.substitutions import FindPackageShare

def is\_valid\_to\_launch():

    # Path includes model name of Raspberry Pi series

    path = '/sys/firmware/devicetree/base/model'

    if os.path.exists(path):

        return False

    else:

        return True

def generate\_launch\_description():

    if not is\_valid\_to\_launch():

        print('Can not launch fake robot in Raspberry Pi')

        return LaunchDescription([])

    start\_rviz = LaunchConfiguration('start\_rviz')

    prefix = LaunchConfiguration('prefix')

    use\_sim = LaunchConfiguration('use\_sim')

    world = LaunchConfiguration(

        'world',

        default=PathJoinSubstitution(

            [

                FindPackageShare('turtlebot3\_manipulation\_bringup'),

                'worlds',

                'empty\_world.model' #model name change accordingly

            ]

        )

    )

    pose = {'x': LaunchConfiguration('x\_pose', default='-2.00'),

            'y': LaunchConfiguration('y\_pose', default='-0.50'),

            'z': LaunchConfiguration('z\_pose', default='0.01'),

            'R': LaunchConfiguration('roll', default='0.00'),

            'P': LaunchConfiguration('pitch', default='0.00'),

            'Y': LaunchConfiguration('yaw', default='0.00')}

    return LaunchDescription([

        DeclareLaunchArgument(

            'start\_rviz',

            default\_value='false',

            description='Whether execute rviz2'),

        DeclareLaunchArgument(

            'prefix',

            default\_value='""',

            description='Prefix of the joint and link names'),

        DeclareLaunchArgument(

            'use\_sim',

            default\_value='true',

            description='Start robot in Gazebo simulation.'),

        DeclareLaunchArgument(

            'world',

            default\_value=world,

            description='Directory of gazebo world file'),

        DeclareLaunchArgument(

            'x\_pose',

            default\_value=pose['x'],

            description='position of turtlebot3'),

        DeclareLaunchArgument(

            'y\_pose',

            default\_value=pose['y'],

            description='position of turtlebot3'),

        DeclareLaunchArgument(

            'z\_pose',

            default\_value=pose['z'],

            description='position of turtlebot3'),

        DeclareLaunchArgument(

            'roll',

            default\_value=pose['R'],

            description='orientation of turtlebot3'),

        DeclareLaunchArgument(

            'pitch',

            default\_value=pose['P'],

            description='orientation of turtlebot3'),

        DeclareLaunchArgument(

            'yaw',

            default\_value=pose['Y'],

            description='orientation of turtlebot3'),

        IncludeLaunchDescription(

            PythonLaunchDescriptionSource([ThisLaunchFileDir(), '/base.launch.py']),

            launch\_arguments={

                'start\_rviz': start\_rviz,

                'prefix': prefix,

                'use\_sim': use\_sim,

            }.items(),

        ),

        IncludeLaunchDescription(

            PythonLaunchDescriptionSource(

                [

                    PathJoinSubstitution(

                        [

                            FindPackageShare('gazebo\_ros'),

                            'launch',

                            'gazebo.launch.py'

                        ]

                    )

                ]

            ),

            launch\_arguments={

                'verbose': 'false',

                'world': world,

            }.items(),

        ),

        Node(

            package='gazebo\_ros',

            executable='spawn\_entity.py',

            arguments=[

                '-topic', 'robot\_description',

                '-entity', 'turtlebot3\_manipulation\_system',

                '-x', pose['x'], '-y', pose['y'], '-z', pose['z'],

                '-R', pose['R'], '-P', pose['P'], '-Y', pose['Y'],

                ],

            output='screen',

        ),

    ])

[5] controllers.yaml

controller\_names:

  - manipulator\_controller

  - base\_controller

manipulator\_controller:

  type: FollowJointTrajectory

  action\_ns: follow\_joint\_trajectory

  joints:

    - joint1

    - joint2

    - joint3

    - joint4

    - end\_effector\_joint

base\_controller:

  type: Twist

  action\_ns: cmd\_vel

  joints:

    - base\_joint

[6] ompl\_planning.yaml

planner\_configs:

  RRTConnectkConfigDefault:

    type: geometric::RRTConnect

    range: 0.0

group\_name\_configurations:

  manipulator:

    planner\_configs:

      - RRTConnectkConfigDefault

  base:

    planner\_configs:

      - RRTConnectkConfigDefault

[7] move\_group.launch.py

import os

import yaml

import xacro

from launch import LaunchDescription

from launch\_ros.actions import Node

from launch.actions import DeclareLaunchArgument

from launch.substitutions import LaunchConfiguration

from launch\_ros.substitutions import FindPackageShare

from moveit\_configs\_utils import MoveItConfigsBuilder

from moveit\_configs\_utils.launches import generate\_move\_group\_launch

from ament\_index\_python.packages import get\_package\_share\_directory

def generate\_launch\_description():

    robot\_description\_config = xacro.process\_file(os.path.join(get\_package\_share\_directory("turtlebot3\_manipulation\_description"), "urdf", "turtlebot3\_manipulation.urdf.xacro",))

    robot\_description = {"robot\_description": robot\_description\_config.toxml()}

    robot\_description\_semantic\_path = os.path.join(get\_package\_share\_directory("manual\_moveit\_config"), "config", "turtlebot3\_manipulation.srdf",)

    with open(robot\_description\_semantic\_path, "r") as file:

        robot\_description\_semantic\_config = file.read()

    robot\_description\_semantic = {

        "robot\_description\_semantic": robot\_description\_semantic\_config

    }

    #kinematics

    kinematics\_yaml\_path = os.path.join(get\_package\_share\_directory("manual\_moveit\_config"), "config", "kinematics.yaml",)

    with open(kinematics\_yaml\_path, "r") as file:

        kinematics\_yaml = yaml.safe\_load(file)

    #planning functinoality

    ompl\_planning\_pipeline\_config = {

        "move\_group": {

            "planning\_plugin": "ompl\_interface/OMPLPlanner",

            "request\_adapters": """default\_planner\_request\_adapters/AddTimeOptimalParameterization \

            default\_planner\_request\_adapters/FixWorkspaceBounds \

            default\_planner\_request\_adapters/FixStartStateBounds \

            default\_planner\_request\_adapters/FixStartStateCollision \

            default\_planner\_request\_adapters/FixStartStatePathConstraints""",

            "start\_state\_max\_bounds\_error": 0.1,

        }

    }

    ompl\_planning\_yaml\_path = os.path.join(get\_package\_share\_directory("manual\_moveit\_config"), "config", "ompl\_planning.yaml",)

    with open(ompl\_planning\_yaml\_path, "r") as file:

        ompl\_planning\_yaml = yaml.safe\_load(file)

    ompl\_planning\_pipeline\_config["move\_group"].update(ompl\_planning\_yaml)

    #trajectory execution

    trajectory\_execution = {

        "moveit\_manage\_controllers": True,

        "trajectory\_execution.allowed\_execution\_duration\_scaling": 1.2,

        "trajectory\_execution.allowed\_goal\_duration\_margin": 0.5,

        "trajectory\_execution.allowed\_start\_tolerance": 0.01,

    }

    #moveit controllers

    moveit\_simple\_controllers\_yaml\_path = os.path.join(get\_package\_share\_directory("manual\_moveit\_config"), "config", "controllers.yaml",)

    with open(moveit\_simple\_controllers\_yaml\_path, "r") as file:

        moveit\_simple\_controllers\_yaml = yaml.safe\_load(file)

    moveit\_controllers = {

        "moveit\_simple\_controller\_manager": moveit\_simple\_controllers\_yaml,

        "moveit\_controller\_manager":

            "moveit\_simple\_controller\_manager/MoveItSimpleControllerManager",

    }

    #planning scene monitor parameters

    planning\_scene\_monitor\_parameters = {

        "publish\_planning\_scene": True,

        "publish\_geometry\_updates": True,

        "publish\_state\_updates": True,

        "publish\_transforms\_updates": True,

    }

    ld = LaunchDescription()

    use\_sim = LaunchConfiguration('use\_sim')

    declare\_use\_sim = DeclareLaunchArgument(

        'use\_sim',

        default\_value='false',

        description='Start robot in Gazebo simuation.')

    ld.add\_action(declare\_use\_sim)

    move\_group\_node = Node(

        package="moveit\_ros\_move\_group",

        executable="move\_group",

        output="screen",

        parameters=[

            robot\_description,

            robot\_description\_semantic,

            kinematics\_yaml,

            ompl\_planning\_pipeline\_config,

            trajectory\_execution,

            moveit\_controllers,

            planning\_scene\_monitor\_parameters,

            {'use\_sim\_time': use\_sim},

        ],

    )

    ld.add\_action(move\_group\_node)

    return ld

    '''

    # Set up MoveIt configurations with your package

    moveit\_config = (

        MoveItConfigsBuilder("turtlebot3\_manipulation", package\_name="manual\_moveit\_config")

        .robot\_description(file\_path="config/robot\_description.urdf")

        .joint\_limits(file\_path="config/joint\_limits.yaml")

        .ompl\_planning(file\_path="config/ompl\_planning.yaml")

        .trajectory\_execution(file\_path="config/controllers.yaml")

        .to\_moveit\_configs()

    )

    # Launch MoveIt move\_group node with loaded configurations

    return LaunchDescription([

        generate\_move\_group\_launch(moveit\_config),

        # Optional RViz launch for visualization

        Node(

            package="rviz2",

            executable="rviz2",

            name="rviz2",

            output="screen",

            arguments=["-d", os.path.join(

                FindPackageShare("manual\_moveit\_config").find("config"),

                "moveit.rviz"

            )],

        ),

    ])

    '''

[8] move\_sequence.py

# move\_forward.py

import rclpy

from rclpy.node import Node

from geometry\_msgs.msg import Twist

class MoveForward(Node):

    def \_\_init\_\_(self):

        super().\_\_init\_\_('move\_forward')

        self.publisher = self.create\_publisher(Twist, '/cmd\_vel', 10)

        timer\_period = 0.5  # seconds

        self.timer = self.create\_timer(timer\_period, self.publish\_velocity)

    def publish\_velocity(self):

        move\_cmd = Twist()

        move\_cmd.linear.x = 0.5  # Set forward speed

        self.publisher.publish(move\_cmd)

def main(args=None):

    rclpy.init(args=args)

    node = MoveForward()

    rclpy.spin(node)

    node.destroy\_node()

    rclpy.shutdown()

if \_\_name\_\_ == '\_\_main\_\_':

    main()

[9] grasp\_and\_move.py

#!/usr/bin/env python3

import rclpy

from rclpy.node import Node

from trajectory\_msgs.msg import JointTrajectory, JointTrajectoryPoint

from geometry\_msgs.msg import Twist

from time import sleep  # Use sleep for timing control

class CombinedControl(Node):

    def \_\_init\_\_(self):

        super().\_\_init\_\_('combined\_control')

        # Publishers

        self.arm\_publisher = self.create\_publisher(JointTrajectory, '/arm\_controller/joint\_trajectory', 10)

        self.base\_publisher = self.create\_publisher(Twist, '/cmd\_vel', 10)

        # Timer for periodic execution

        self.timer = self.create\_timer(1.0, self.execute\_sequence)

        # Sequence control variables

        self.step = 0

    def execute\_sequence(self):

        if self.step == 0:

            # Step 1: Move the base forward

            self.move\_base(1.18)

            self.step += 1

            sleep(2)  # Move for 2 seconds

            self.stop\_base()

        elif self.step == 1:

            # Step 2: Move the arm down to pick something up

            p = [0.0, 0.75, -0.25, 0.35]

            self.move\_arm\_to\_position(p)

            self.step += 1

            sleep(2)  # Allow some time for the arm movement

        elif self.step == 2:

            # Step 3: Move the arm back up to the home position

            self.move\_arm\_to\_home\_position()

            self.step += 1

            sleep(2)  # Allow some time for the arm movement

        elif self.step == 3:

            # Step 4: Move the base forward again

            self.move\_base(0.5)

            self.step += 1

            sleep(2)  # Move for 2 seconds

            self.stop\_base()

            self.get\_logger().info("Sequence complete.")

            # Stop the timer to end the sequence

            self.timer.cancel()

    def move\_arm\_to\_position(self, position):

        # Define joint names

        joint\_names = ["joint1", "joint2", "joint3", "joint4"]

        # Initialize JointTrajectory message

        traj\_msg = JointTrajectory()

        traj\_msg.joint\_names = joint\_names

        # Set up positions for each joint

        point = JointTrajectoryPoint()

        point.positions = position  # Move to the position for picking

        point.time\_from\_start.sec = 1  # Duration to reach the position

        traj\_msg.points.append(point)

        # Publish the trajectory to move the arm

        self.arm\_publisher.publish(traj\_msg)

        self.get\_logger().info("Arm movement command sent to pick position.")

    def move\_arm\_to\_home\_position(self):

        # Define the home position

        home\_position = [0.0, -1.0, 0.7, 0.3]

        self.move\_arm\_to\_position(home\_position)

        self.get\_logger().info("Arm movement command sent to home position.")

    def move\_base(self, speed):

        move\_cmd = Twist()

        move\_cmd.linear.x = speed  # Set forward speed

        self.base\_publisher.publish(move\_cmd)

        self.get\_logger().info("Base moving forward.")

    def stop\_base(self):

        move\_cmd = Twist()

        move\_cmd.linear.x = 0.0  # Stop the base

        self.base\_publisher.publish(move\_cmd)

        self.get\_logger().info("Base stopped.")

def main(args=None):

    rclpy.init(args=args)

    combined\_control = CombinedControl()

    rclpy.spin(combined\_control)

    combined\_control.destroy\_node()

    rclpy.shutdown()

if \_\_name\_\_ == '\_\_main\_\_':

    main()

[10] gazebo\_stream.py

import rclpy

from rclpy.node import Node

from sensor\_msgs.msg import Image

from cv\_bridge import CvBridge

import cv2

import os

from ultralytics import YOLO

class ObjectRecognitionNode(Node):

    def \_\_init\_\_(self):

        super().\_\_init\_\_('object\_recognition\_node')

        self.subscription = self.create\_subscription(

            Image,

            '/pi\_camera/image\_raw',

            self.image\_callback,

            10)

        self.subscription  # Prevent unused variable warning

        self.bridge = CvBridge()

        # Directory to save images

        self.save\_directory = './gazebo\_img'

        os.makedirs(self.save\_directory, exist\_ok=True)

        # State variables

        self.image\_ready = True

        self.detected = False

        # Load YOLO model

        self.model = YOLO("best.pt")  # Load your custom model

    def image\_callback(self, msg):

        # Only process a new image if we haven't detected anything and an image is ready

        if not self.detected and self.image\_ready:

            self.image\_ready = False  # Set image\_ready to False until we process this image

            # Convert ROS Image message to OpenCV format

            cv\_image = self.bridge.imgmsg\_to\_cv2(msg, "bgr8")

            # Save the image to the specified directory, overwriting the old image

            image\_path = os.path.join(self.save\_directory, 'captured\_image.jpg')

            cv2.imwrite(image\_path, cv\_image)

            self.get\_logger().info(f"Image saved to {image\_path}")

            # Run object detection

            self.detected = self.run\_object\_detection(image\_path)

            # If nothing is detected, set image\_ready to True to process the next image

            if not self.detected:

                self.image\_ready = True

            else:

                self.get\_logger().info("Object detected, stopping image collection.")

    def run\_object\_detection(self, image\_path):

        # Use the YOLO model to predict objects in the image

        results = self.model(image\_path, save=True)  # Perform object detection without saving results

        # Check if results are valid and contain detections

        if results is None or len(results) == 0:

            return False

        # Iterate through results and check for detections

        for result in results:

            if result is not None and hasattr(result, 'boxes') and result.boxes is not None:

                if len(result.boxes) > 0:  # Check if there are any detected boxes

                    return True

        return False

def main(args=None):

    rclpy.init(args=args)

    node = ObjectRecognitionNode()

    rclpy.spin(node)

    node.destroy\_node()

    rclpy.shutdown()

if \_\_name\_\_ == '\_\_main\_\_':

    main()

[11] web\_server.py

from flask import Flask, send\_from\_directory

import os

app = Flask(\_\_name\_\_)

# Directory where images are saved

IMAGE\_DIR = '/home/eireland/ty/runs/obb/predict'

@app.route('/')

def index():

    # List all images in the directory

    images = [f for f in os.listdir(IMAGE\_DIR) if f.endswith('.jpg')]

    # Generate HTML to display the images

    html = '''

    <h1>Object Detection Results</h1>

    <meta http-equiv="refresh" content="5">  <!-- Refresh every 5 seconds -->

    '''

    for image in images:

        html += f'<div><img src="/images/{image}" width="640"></div>'

    return html

@app.route('/images/<path:filename>')

def serve\_image(filename):

    return send\_from\_directory(IMAGE\_DIR, filename)

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True, host='0.0.0.0', port=5000)