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

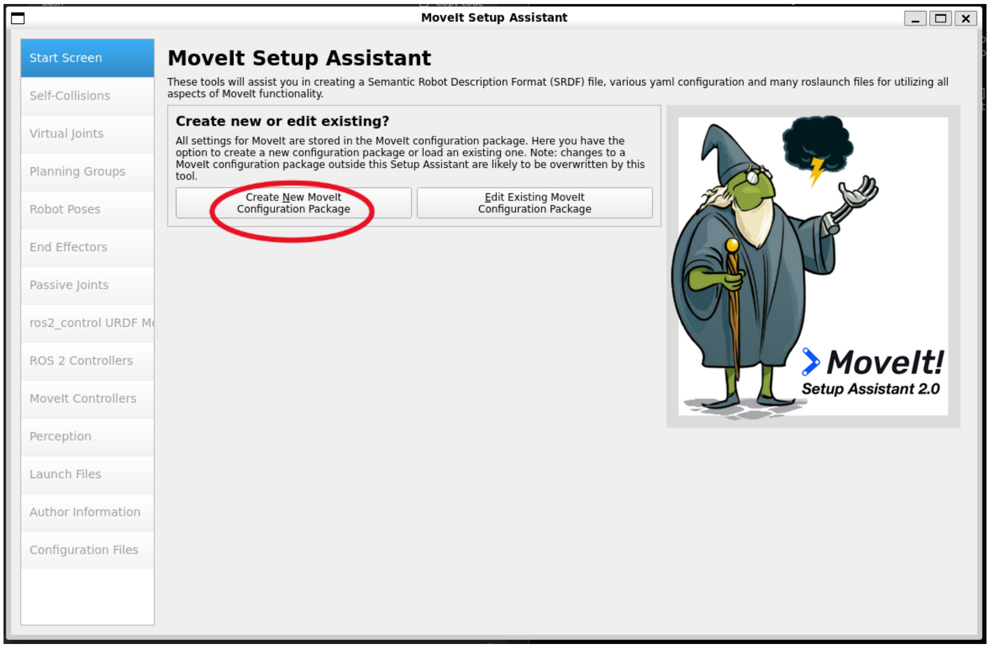
- Implement a robotic manipulator and mobile robot together
- Achieve autonomous navigation
    - Apply machine vision and machine learning to an integrated robotic system
    - Train the model with a predefined dataset

- Pick and place system
    - Deploy the system in a physical environment

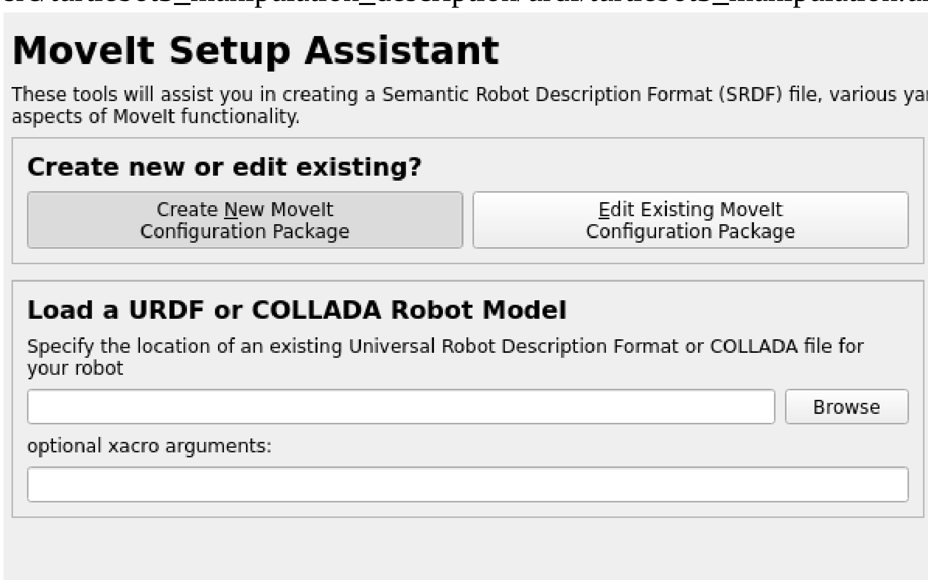
Week 10/27/24

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

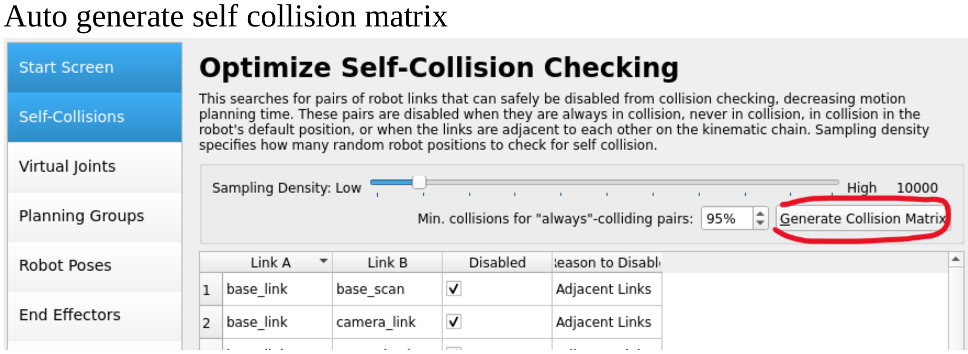
- Preface:
    - There are 3 packages in my workspace's src directory:
        - Turtlebot3_manipulation_bringup: spawns the model in Gazebo and contains the node scripts
        - Turtlebot3_manipulation_description: reference for the urdf and xacro files
        - Manual_moveit_config: move it configuration 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

- 

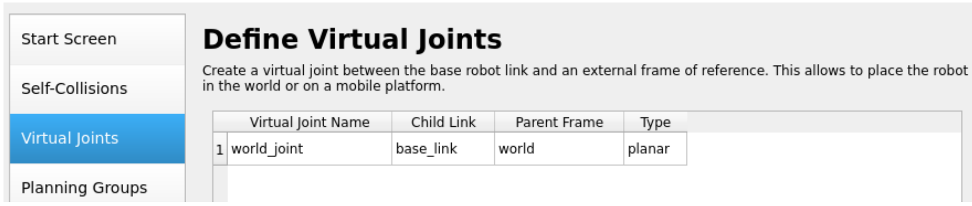
- Add the path to the urdf or xacro file of your model (mine was in src/turtlebot3_manipulation_description/urdf/turtlebot3_manipulation.urdf.xacro)



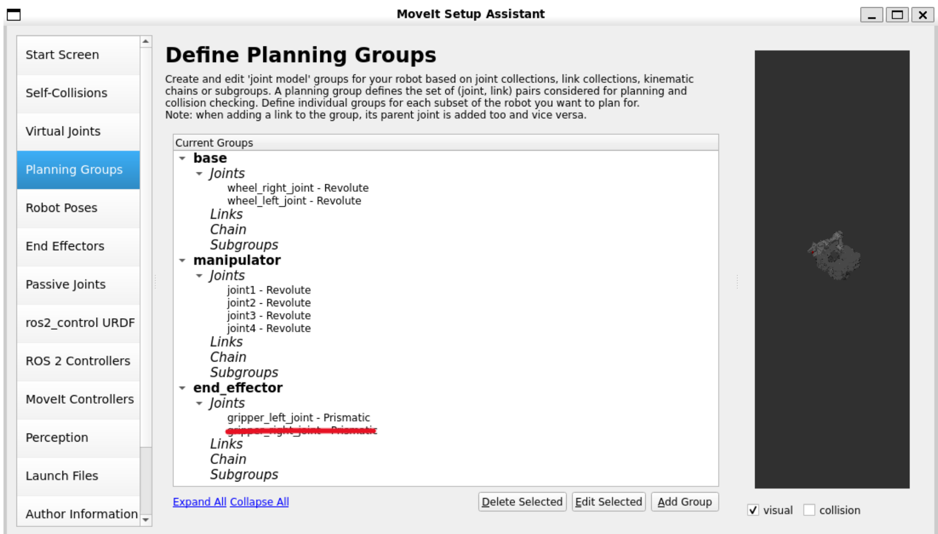
- Auto generate self collision matrix



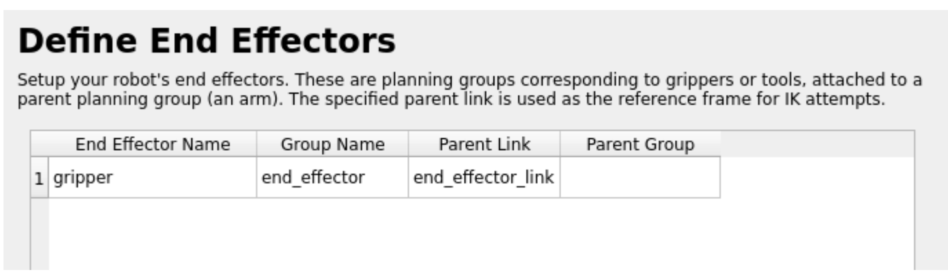
- Define virtual joints to connect robot to world



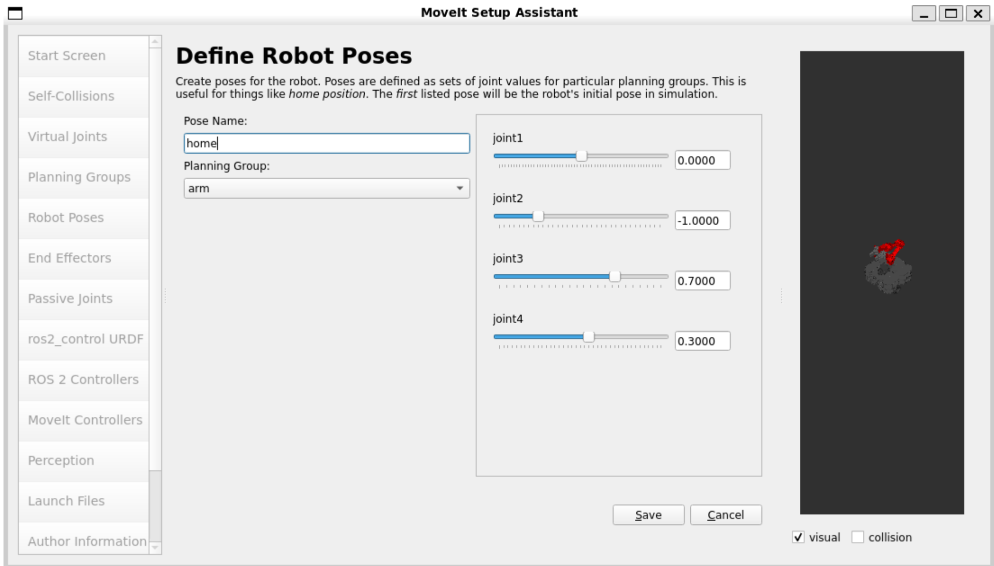
- Define planning groups – in this case, our joints work independently (differential drive) so specify them one by one

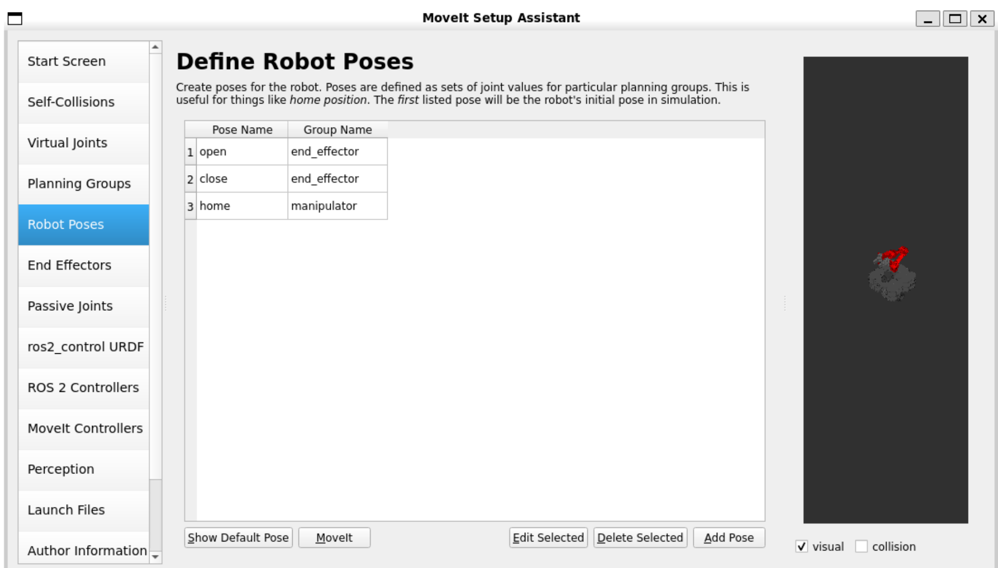


- Define the end effector

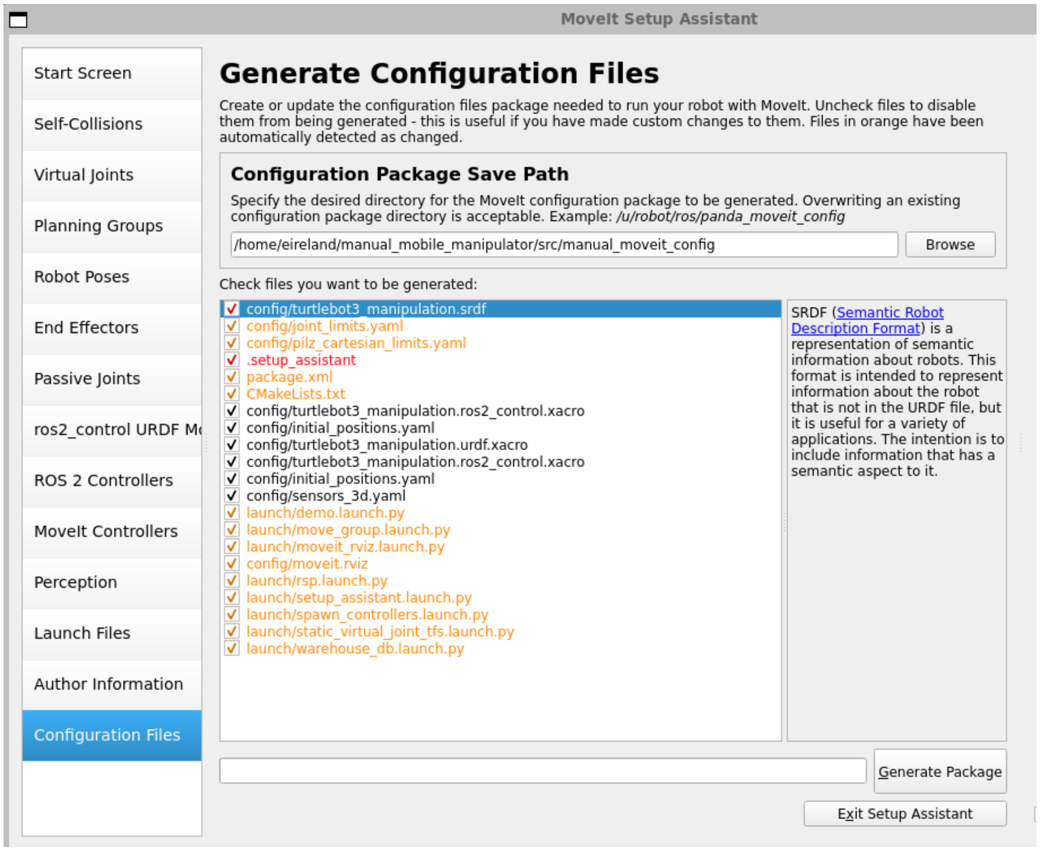


- Define poses for open and closed hand





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



- 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

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:

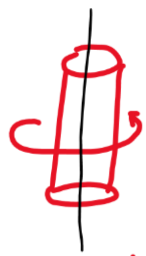
09/29/24

wsl in Windows cmd: launch windows subsystem for Linux

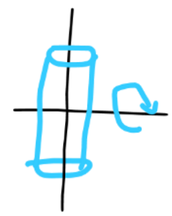
Create workspace:
- mkdir -p ~/*ws*/src
- cd ~/*ws*/src
    - from scratch:
        - ros2 pkg create --build-type ament_cmake *packageName*
        - cd ~/*ws*/src/*packageName*

        - mkdir launch urdf config
    - from git:
        - git clone <*github link*>
- cs ~/*ws*
- colcon build
- source install/setup.bash

Joints:



Axial



Transversal

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)

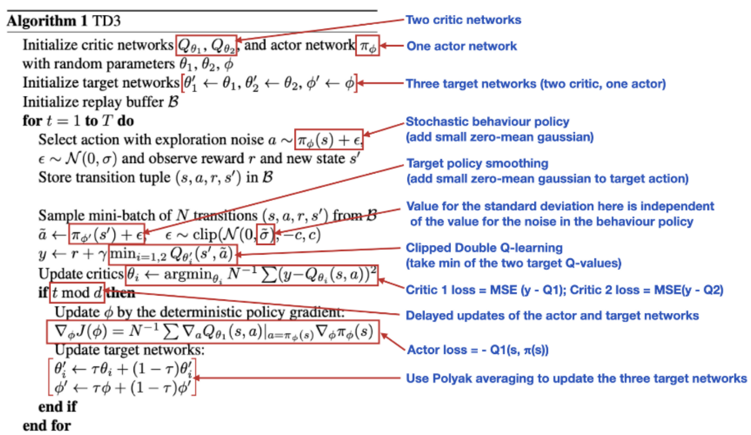
11/03/24

Vocab

- **Q-value function** – function that estimates the expected future rewards an agent can obtain by taking specific action α 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

<u>Code:</u>
[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_))
```

```python
        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
<?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 -->
```

```xml
    <!-- 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"/>
```

```xml
    </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>
```

```xml
                <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 -->
```

```xml
    <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"/>
```

```xml
            </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

```python
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

```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

```yaml
planner_configs:
  RRTConnectkConfigDefault:
    type: geometric::RRTConnect
    range: 0.0

group_name_configurations:
  manipulator:
    planner_configs:
```

```yaml
      - RRTConnectkConfigDefault
  base:
    planner_configs:
      - RRTConnectkConfigDefault
```

[7] move_group.launch.py

```python
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_manipulat
ion_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()
```