Task 4C: Open Loop Control

Objective

Implement an open-loop controller for moving the bots in three different shapes: Square, Triangle, and Circle. The goal is also to establish connection between the computer and the robot. The computer shall be running a node in ROS2 that shall be publishing velocities to the robot. The ESP32 on the bots shall be running microROS and shall be subscribing to the velocities and sending commands to the servo motors.

Overview :

1. ROS2 Nodes on Computer:

- Develop and run ROS2 nodes on the computer to interact with the robot.
- The node should use `/cmd_vel/bot1` , `/cmd_vel/bot2` and `/cmd_vel/bot3` topics for publishing the velocity commands to the bots.
- All of the shapes in this task can be achieved just using intuition and without even using the inverse kinematics matrix.
    - To create a **square**, move the bot up, right, down and left **without** turning the bot by 90 degrees. Two of these directions will be super simple either {up, down} or {left, right} (depending what orientation is theta = 0 for your robot) - simply turn two motors on with same speed in the appropriate direction and the reverse of those directions. The other two directions is not so straight forward. The matrix can help in calculating it systematically: it is simply one of the columns of the matrix.
      OR you could also use intuition and vector drawings to realise that two motors say Motor2 and Motor3 will have same magnitude and Motor1 will have a double the magnitude and directions must be such that the sum of velocity vectors of Motor2 and and Motor3 will be same of the Velocity Vector of Motor1.
    - To create the **triangle** (without turning), is easier than square since it is just a matter turning different combinations of two motors on at a time.
    - Creating a **circle**,
        - while also rotating on its own axis is much simpler to figure out wheel velocities manually with a little thinking and imagination of vectors. Can you figure this puzzle out?
        - without rotating on its own axis, will mostly need to use the inverse kinematics matrix.
          For Task4C you are free to draw any kind of circle of your choice.

2. Micro-ros on Esp32 :

- Now, we are moving to set up the **micro_ros_arduino** package in your Arduino IDE.
- Download the zip file from the **micro_ros_arduino** repository.
- Open Arduino IDE and include the downloaded zip file as library.
- You have to create three separate nodes: `bot1_node` , `bot2_node` , and `bot3_node` for three ESP32 controllers. Program the nodes to subscribe to the command velocity topics (mentioned above) and control the servo motors according to the commanded velocity. After completing your code, upload it to the ESP32.

3. Network Setup:

- Ensure that the computer and the ESP32 are connected to the same network.

4. Micro-ROS Agent on the computer :

- We will use a micro-ROS Agent to act as the bridge between DDS-XRCE and normal DDS. This means that ROS and micro-ROS nodes can communicate with the agent.
- For running the agent there are two methods :
    1. With micro-ros setup.
    2. With Docker.

1. With micro-ros setup :

```
# Create a workspace and download the micro-ROS tools
mkdir microros_ws
cd microros_ws
git clone -b $ROS_DISTRO https://github.com/micro-ROS/micro_ros_setup.git src/micro_

# Update dependencies using rosdep
sudo apt update && rosdep update
rosdep install --from-paths src --ignore-src -y

# Install pip
sudo apt-get install python3-pip

# Build micro-ROS tools and source them
colcon build
source install/local_setup.bash
```

After this source your workspace in bashrc.

Creating the micro-ROS agent

The micro-ROS app is now ready to be connected to a micro-ROS agent to start talking with the rest of the ROS 2 world. To do that, let's first of all create a micro-ROS agent:

```
# Download micro-ROS-Agent packages
ros2 run micro_ros_setup create_agent_ws.sh

Now, let's build the agent packages and, when this is done, source the installation:

# Build step
```

```
ros2 run micro_ros_setup build_agent.sh
source install/local_setup.bash
```

Running the micro-ROS app

At this point, you have both the client and the agent correctly installed in your host machine.

To give micro-ROS access to the ROS 2 dataspace, run the agent:

```
# Run a micro-ROS agent
ros2 run micro_ros_agent micro_ros_agent udp4 --port 8888
```

2. With Docker :

- For using docker you first need to install it. To install Docker in your system you can refer to this link.
- After installing Docker in your system run the below commands to run the micro-ros agent

```
docker run -it --rm -v /dev:/dev --privileged --net=host microros/micro-ros-agent:hu
```

Note : We have shared here two approaches. You are free to choose either approaches. We have followed the first approach on our systems. The second approach is also given by the microROS tutorials.

After running the agent ,Press EN button of Esp32 controller and you will be able to see some messages on the terminal. This messages ensures that the connection has been established

5. Run your ROS2 nodes :

- To publish velocity commands to the bots, you have to run your nodes using `ros2 run` (for a single node) or use `ros2 launch` (if you have multiple nodes to run).
- To check whether the node is publishing or a bot node is subscribing to the topic you can check rqt_graph by running `rqt_graph` on a new terminal.

A simple example

- Before writing the publishers and subscriber described in Step 1 and 2 let's take a simpler example of publishing and subscribing an integer.
- Let's get started by writing a publisher node in ROS 2 and a subscriber node in micro-ROS.

Publisher node:

By this time, you will be able to write the publisher node. Fill in the blanks in the code below.

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import Int32

class Publisher(Node):
    def __init__(self):
        super().__init__('Publisher_node')
```

```
        # Initialze Publisher with the "/Integer" topic
         self.timer = self.create_timer(0.5,self.timer_callback)
         self.i = 0
    def timer_callback(self):
         msg = Int32()
        # Assign the msg variable to i
        # Publish the msg
        # Increment the i

def main(args = None):
    rclpy.init(args=args)
    Publisher_node = Publisher()
    rclpy.spin(Publisher_node)
    Publisher_node.destroy_node()
    rclpy.shutdown()


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

Subscriber node :

- For subscriber node ,you can download from this **link**.
- When you look at the example of **micro_ros_subscriber_wifi.ino** we will see the instances of structures used from rclc.

```
rcl_subscription_t subscriber;
std_msgs__msg__Int32 msg;
rclc_executor_t executor;
rclc_support_t support;
rcl_allocator_t allocator;
rcl_node_t node;
rcl_timer_t timer;
```

Further more, memory allocation is organized by an allocator and the ROS context is initialized by using a support object.

`set_microros_wifi_transports` function is used to configure micro-ROS to use specific WiFi transports for communication. You'll of course need to specify YOUR network details here.

```
set_microros_wifi_transports("WIFI SSID", "WIFI PASS", "laptop_ip_address", 8888);
```

A node is created by below functions.

```
allocator = rcl_get_default_allocator();

  //create init_options
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

  // create node
  RCCHECK(rclc_node_init_default(&node, "subscriber_node", "", &support));
```

A subscriber is created with three parameters: node object, message type, and topic name. In this example, a Int32 subscriber with topic name `/Integer` is created.

```
/ create subscriber
  RCCHECK(rclc_subscription_init_default(
    &subscriber,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
    "/Integer"));

  // create executor
  RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
  RCCHECK(rclc_executor_add_subscription(&executor, &subscriber, &msg, &subscription_call
```

- Now, Upload the **micro_ros_subscriber_wifi.ino** to ESP32. Ensure that the computer and the ESP32 are connected to the same network.
- Run the micro-ROS agent. For running the micro-ROS agent, you can refer to the section above.
- After running the micro-ROS agent, press the EN button on the ESP32 to establish communication.
- You can check if all is well with `rqt_graph` or run the `ros2 topic list` or `ros2 node list`
- To get a visual confirmation, you could modify the code to change the esp32 LED status based on the integer msg received.

Submission instruction

- Record a video of the bots making the shapes, along with a screen recording of the terminal during the initial run.
- Run rqt_graph in a new terminal while recording the video, take a screenshot, and merge the image with the recorded video.
- Upload the video with the title `HB23_<Teamid>_Task4C` (For example: If your team ID is 1234 then, save it as HB23_1234_Task4C).
- Please note that while uploading the video on YouTube select the privacy setting option as **Unlisted**.
- Submit the unlisted youtube link on eYRC Portal
