**AIG 240 PROJECT-3 ASSESSMENT Q&A (Apoorva)**

**1. How did you establish communication with the robot?**

To communicate with the robot, we set up a ROS 2 node on our system. First, we ensured the robot was powered on and connected to the same network as our ROS 2 system. Then, we used the ros2 topic list command to check for active topics, allowing us to confirm the available channels for sending movement commands. To control the robot, we published velocity commands using geometry msgs/Twist messages to the appropriate topic, such as /cmd\_vel. We verified the connection by running ros2 node info, which helped confirm that our node was successfully interacting with the robot.

**2. How different is controlling the robot in Gazebo versus the real world?**

There are several key differences between operating the robot in Gazebo and in a real-world setting:

* **Physics Simulation:** Gazebo provides a simplified simulation of physics, but it does not perfectly capture real-world effects like friction, sensor noise, or hardware wear.
* **Latency & Precision:** In the physical world, factors like network latency, motor inconsistencies, and hardware limitations can affect movement accuracy, whereas Gazebo offers a more controlled and predictable response.
* **Environmental Variables:** The real world introduces additional challenges such as surface friction, uneven floors, unexpected obstacles, and variations in lighting conditions for sensors, none of which are present in a controlled simulation.

**3. How did you implement the navigation strategy in your node?**

Our navigation approach involved creating a ROS 2 Python node that controlled the robot’s movement step by step:

* First, we published an angular velocity command to make the robot turn **45 degrees counterclockwise**. This was achieved by setting a rotational speed and keeping it active for a calculated duration.
* Once the turn was complete, we sent a linear velocity command to move the robot **forward for 2 meters**. The duration of this movement was based on the robot’s speed.
* Finally, after covering the required distance, we sent a command to stop the robot by publishing a zero-velocity message.

This sequence ensured that the robot followed the expected navigation path.

**4. What challenges did you face when making the robot move in the real world?**

We encountered several challenges while working with the robot in a real environment:

* **Inconsistent Movement:** The robot did not always turn or move the exact expected distance due to motor inaccuracies and wheel slippage.
* **Network Delays:** Occasionally, communication delays caused the robot to respond slower than expected.
* **Calibration Difficulties:** The robot’s speed was affected by factors such as battery level and slight motor variations, which required us to fine-tune movement durations for better accuracy.
* **Environmental Factors:** Real-world conditions like surface friction, small obstacles, or even slight inclines impacted the robot’s movement, making it harder to achieve precise navigation compared to the simulation.

**AIG 240 PROJECT-3 ASSESSMENT Q&A (Asgari)**

1. **How did you establish communication with the robot?**

We established communication with the robot using ROS 2 by setting up a ROS 2 node on our system. This involved making sure the robot was powered on and connected to the same network as the ROS 2 system, then using ros2 topic list to verify available topics for sending commands. Also publishing velocity commands (geometry\_msgs/Twist) to the appropriate topic (e.g., /cmd\_vel) using ros2 topic pub or a custom ROS 2 Python node and then running ros2 node info to check if the node was successfully communicating with the robot.

1. **How different is controlling the robot in Gazebo vs the real world?**

The difference between controlling the robot in Gazebo vs. the real world is the

* Physics Simulation: Gazebo approximates physics but doesn’t fully replicate real-world friction, sensor noise, or hardware imperfections.
* Latency & Accuracy: In the real world, network delays, hardware wear, and motor inaccuracies can affect movement, whereas Gazebo provides idealized motion.
* Environmental Factors: In real-world scenarios, surface conditions (e.g., friction, incline) and external interferences (e.g., obstacles, lighting for sensors) can impact navigation, while Gazebo operates in a controlled environment.

1. **How did you implement the navigation strategy in your node?**

We implemented the navigation strategy using a ROS 2 Python node that:

* Published velocity commands (Twist messages) to /cmd\_vel.
* Executed a 45-degree turn counterclockwise by publishing an angular velocity for a fixed duration.
* Moved the robot forward by setting a linear velocity for a calculated time based on the robot’s speed.
* Stopped the robot by publishing a zero-velocity message after moving 2 meters.

1. **What challenges did you face making the robot move in the real physical world?**

The challenges we faced were:

* Inconsistent Motion: Due to motor inaccuracies and wheel slippage, the robot didn’t always turn or move the exact expected distance.
* Network Delays: Communication latency sometimes caused delayed execution of commands.
* Calibration Issues: The robot’s speed varied slightly due to battery levels and motor inconsistencies, requiring fine-tuning of movement durations.
* Environmental Factors: Floor friction and potential obstacles required adjustments compared to the simulated environment.

**AIG 240 PROJECT-3 ASSESSMENT Q&A (Vipin)**

1. **Establishing Communication with the Robot**
   * **ROS 2 Network Configuration**:
     + Set a unique ROS\_DOMAIN\_ID on both the robot and control computer to avoid interference with other ROS networks.
     + Connected devices to the same Wi-Fi network and verified connectivity using ping and ifconfig.
   * **Topic Validation**:
     + Used ros2 topic list to confirm the robot exposed /cmd\_vel for velocity commands.
     + Tested communication by publishing a Twist message manually (e.g., ros2 topic pub /cmd\_vel) and observing robot movement.
   * **Security and Latency**:
     + Disabled firewalls on both devices to allow DDS traffic on specific ports (default UDP 7400-7500).
     + Addressed network latency by minimizing background traffic and ensuring a strong signal.
2. **Simulation (Gazebo) vs. Real-World Differences**
   * **Ideal vs. Imperfect Conditions**:
     + In Gazebo, motor responses are instantaneous, and odometry is noise-free. In the real world, motors exhibit latency, and wheels may slip (e.g., on classroom carpets).
   * **Sensor Reliability**:
     + Simulated LIDAR/IMU data is perfectly accurate, while real sensors (e.g., wheel encoders) suffer from drift (e.g., uneven floors caused odometry errors).
   * **Environmental Factors**:
     + Real-world testing required physical space management (e.g., avoiding obstacles) and safety checks, unlike the controlled Gazebo environment.
3. **Navigation Strategy Implementation**
   * **State Machine Design**:
     + **State 1 (Turn)**: Published angular velocity (twist.angular.z = 0.5 rad/s) for a precalculated duration (π/4 / 0.5 = 1.57s) to achieve a 45° counterclockwise turn.
     + **State 2 (Move)**: Switched to linear velocity (twist.linear.x = 0.2 m/s) for 2.0m / 0.2 = 10s to move forward.
     + **State 3 (Stop)**: Published zero velocities to halt the robot.
   * **Open-Loop Control**:
     + Chose timed commands due to lack of reliable odometry feedback. Parameters like angular\_speed and linear\_speed were calibrated through iterative testing (e.g., reducing speed from 0.3 m/s to 0.2 m/s to minimize wheel slippage).
4. **Challenges in Real-World Execution**
   * **Calibration Issues**:
     + Initial 45° turns overshot due to motor inertia. Added a 0.2s buffer to the turn\_duration to account for deceleration.
   * **Environmental Variability**:
     + Classroom floor unevenness caused inconsistent linear movement (e.g., 2m travel varied by ±10cm). Mitigated by lowering speed.
   * **Hardware Limitations**:
     + Battery voltage drops during operation reduced motor torque, affecting repeatability. Addressed by testing at full charge.
   * **Network Stability**:
     + Intermittent Wi-Fi latency caused delayed command execution. Switched to a wired Ethernet connection for critical demos.

**AIG 240 PROJECT-3 ASSESSMENT Q&A (Yan)**

**1. How did you establish communication with the robot?**

We established communication with the robot using **ROS 2** running on both a laptop and the robot’s onboard computer:

* We connected both the robot and the development machine to the same Wi-Fi network. We ensured that each device could ping the other, confirming network connectivity.
* We configured the ROS\_DOMAIN\_ID so that both the robot and the controlling machine could discover each other’s topics. This allowed the command velocity messages and other topics/services to be shared.

Overall, communication was managed by ensuring that both systems ran compatible versions of ROS 2, were on the same network, and had ROS environment variables correctly set.

**2. How different is controlling the robot in Gazebo vs the real world?**

1. **Physics and Noise**:
   * **Gazebo**: Although Gazebo includes physics simulations, friction and sensor noise can be idealized or simplified. Sensor readings tend to be more predictable.
   * **Real World**: Actual hardware introduces noise, drift, wheel slippage, and more unpredictable behavior due to real-world friction, uneven floors, or battery voltage variations.
2. **Sensors and Calibration**:
   * **Gazebo**: Sensor models are idealized. If you specify a 45-degree turn in simulation, you can expect near-perfect rotation.
   * **Real World**: Even slight miscalibrations in the IMU or wheel encoders can cause cumulative errors. You often need to calibrate and compensate for drift.
3. **Environment**:
   * **Gazebo**: The environment is a controlled virtual space. Walls, obstacles, and landmarks remain consistent. Lighting and reflectivity do not affect sensors the same way.
   * **Real World**: Lighting conditions, unexpected obstacles, people walking around, or changes in floor texture can affect sensor data and navigation performance.
4. **Software-Hardware Interaction**:
   * **Gazebo**: Commands in cmd\_vel are executed by a simulated differential drive or other robot models with immediate feedback.
   * **Real World**: Commands are translated through hardware drivers. Network latency or motor lag can cause slight delays in execution.

Because of these differences, testing in Gazebo is a great start but always requires real-world tuning and validation to account for hardware imperfections and unpredictable environmental factors.

**3. How did you implement the navigation strategy in your node?**

1. **Initialization**:
   * Imported necessary ROS 2 Python libraries.
   * Created a node that handles publishing velocity commands to the /cmd\_vel topic.
2. **Turning 45 Degrees**:
   * Calculated the needed angular velocity and duration to achieve a 45-degree turn.
   * Published cmd\_vel messages with that angular velocity for time\_to\_rotate seconds to complete the turn.
   * Once the turn was complete, set angular velocity to zero.
3. **Moving Forward 2 Meters**:
   * Calculated the needed linear velocity and duration to move 2 meters.
   * Published cmd\_vel messages with that linear velocity for time\_to\_move seconds.
   * Stopped publishing by setting the linear velocity to zero once the distance was covered.

**4. What challenges did you face making the robot move in the real physical world?**

1. **Wheel Slippage and Calibration**:  
    The floor surface, wheel traction, and any slight miscalibration in wheel diameter or wheelbase distance cause the actual distance traveled or rotation angle to deviate from the commands.
2. **Timing vs Odometry Discrepancies**:  
    Relying solely on “rotate for X seconds” or “move for Y seconds” works in simulation but can be less accurate in the real world if the robot’s speed isn’t perfectly matched. Using odometry feedback helps, but it requires correct sensor calibration.
3. **Network Latency**:  
    If controlling the robot via Wi-Fi, occasional latency spikes can delay velocity commands or sensor feedback, causing the robot to overshoot turns or distances.