Project 3 Assessment Questions

Robot #16: Aliyyah Jackhan, Jonathan Chacko, Aadil Shaikh

## How did you establish communication with the robot?

Communication with the robot was established using ROS (Robot Operating System) publishers and subscribers, together with custom ROS launch files, as implemented in this project:

* A ROS publisher was created for the /jetauto\_controller/cmd\_vel topic using geometry\_msgs/Twist messages to send linear and angular velocity commands for robot movement.
* A ROS subscriber was set up for the /odom topic using nav\_msgs/Odometry messages to receive real-time feedback about the robot’s position and orientation.
* The ROS node was initialized with rospy.init\_node('smart\_mover'), allowing the script (jetauto\_square.py) to register as an active node within the ROS communication network and interact with other nodes and topics in the system.
* All communication takes place over the ROS master, which manages message passing and ensures that nodes can exchange data reliably across the ROS network.
* In addition, **custom ROS launch files**—jetauto\_square.launch and custom\_controller.launch—were used to automate and streamline the process of starting the necessary nodes and setting parameters for the robot’s operation. These launch files make it easy to bring up the control environment by handling node initialization, topic remapping, and parameter configuration, ensuring that all components required for communication and control are launched consistently and correctly each time.

How to Launch:

cd ~/jetauto\_ws

sudo systemctl stop start\_app\_node.service

source devel/setup.bash

roslaunch lab5\_jetauto\_control jetauto\_square.launch

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

Controlling the robot in Gazebo is much more predictable and reliable because the simulation provides perfect actuation, instant feedback, and no physical disturbances. Movements and turns happen exactly as scripted. In contrast, running the same script on the real robot introduces real-world challenges—such as sensor noise, wheel slippage, actuator inaccuracies, and uneven surfaces. These factors can cause the robot to drift, overshoot, or deviate from its path, requiring more robust error checking and correction in the code. That’s why some scripts that work well in Gazebo need modifications to work reliably on the physical robot.

1. **How do you control that the robot moved roughly 1 meter in each of the steps?**

To control the robot’s movement for roughly 1 meter in each step, we relied on both the code’s use of odometry and a physical verification method:

* In the SmartMover class (jetauto\_square.py), the robot subscribes to the /odom topic to continuously track its current position using odometry data. For each movement, the code calculates the distance traveled from the starting point and commands the robot to stop once it has covered the specified distance (e.g., 1 meter) as measured by its onboard odometry.
* To verify the accuracy of the robot’s movement in the real world, we taped a square box with sides of exactly 1 meter on the floor. After running the robot, we observed whether it stayed within or along these lines, allowing us to visually confirm that the robot’s odometry-based movements corresponded closely to the intended 1-meter distances.

This approach helped us ensure both software-controlled precision and real-world accuracy for each side of the robot’s square path.

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

Several challenges arose during real-world implementation:

* **Odometry/Topic Availability:** One major challenge was that the required /odom topic was not always available or properly configured, depending on which launch file was used. Some launch files (like jetauto\_peripherals teleop\_key\_control.launch) did not provide the necessary odometry data, making it impossible for the robot to accurately track its position or move precisely by distance. This led to significant confusion and troubleshooting as I had to identify which launch configurations actually provided odometry (/odom) and the correct velocity topics (/jetauto\_controller/cmd\_vel).
* **Custom Launch File Creation:** To resolve the above, I created a custom launch file that combined the working robot bringup launch file with the necessary odometry and control nodes. This allowed my project script to receive the required topics and finally run as intended.
* **Overshooting and Momentum:** The robot often overshot target distances and angles due to its physical size, inertia, and momentum. This required compensations in the code, such as deducting an offset from the intended movement distance.
* **Rotation and Accuracy Issues:** Rotations were sometimes inaccurate, causing the robot to misalign or deviate from the intended path.
* **Odometry Drift and Accumulated Errors:** Over multiple movements, odometry drift and sensor errors would accumulate, making it harder to maintain accurate trajectories over longer sequences.

Additionally, through trial and error with the available launch files, I discovered that only specific configurations exposed the necessary topics which I needed, and this sometimes required combining multiple launch and node files together.