## What command did you use to launch the JetAuto robot in Gazebo?

### Launching Gazebo with the Robot

To launch the JetAuto robot in Gazebo, I used the following ROS launch command **from the root of my catkin workspace** (after sourcing the workspace’s setup.bash):

cd ~/jetauto\_ws

source devel/setup.bash

roslaunch jetauto\_gazebo worlds.launch

This command starts the full simulation environment, spawning the JetAuto robot in a Gazebo world with all required controllers, plugins, and visual tools.

### Launch the Custom Code to get the Required Movement for Project 2

To execute my Project 2 movement script (making the robot move in a square pattern using odometry), I ran this in a **separate terminal**:

cd ~/jetauto\_ws

source devel/setup.bash

rosrun lab4\_jetauto\_control jetauto\_square.py

This command launches my custom ROS node, which publishes movement commands to control the JetAuto robot’s motion within the Gazebo simulation.

## Which two other launch files were called when you launched worlds.launch?

When **worlds.launch** is executed, it *includes* (calls) two other launch files, as seen in the XML structure of **jetauto\_ws/src/jetauto\_simulations/jetauto\_gazebo/launch/worlds.launch**:

#### empty\_world.launch

* + Path: **jetauto\_ws/src/jetauto\_simulations/jetauto\_gazebo/launch/empty\_world.launch**
  + Purpose: Starts a blank Gazebo environment (optionally with GUI, sim time, etc.) so the robot can be spawned into it.

#### spwan\_model.launch

* + Path: **jetauto\_ws/src/jetauto\_simulations/jetauto\_gazebo/launch/spwan\_model.launch**
  + Purpose: Spawns the JetAuto robot model into the running Gazebo world and sets up sensor/actuator plugins as specified by launch arguments.

## Describe the process of setting up the ROS workspace and creating the package.

**Workspace Setup (as performed in this project):**

* **Workspace Provided:**  
  I did not create the workspace from scratch. Instead, the professor provided a complete jetauto\_ws workspace as a ZIP file, which I unzipped into my desired folder.
* **Symbolic Link:**  
  To make the workspace accessible at the default ROS location.
* **Python Shebang Issues:**  
  The professor’s code used #!/usr/bin/env python3, but ROS Melodic defaults to Python 2 on Ubuntu 18.04.
* **Make Scripts Executable:**  
  I ensured all Python scripts were executable:
* **catkin\_build vs. catkin\_make:**  
  The provided workspace had previously been built with catkin build (not catkin\_make). Running catkin\_make gave errors about the existing build/devel structure, so I switched to catkin build.  
  Although catkin build gave some errors (due to missing dependencies in non-essential packages), it still built the new/modified packages (like my Project 2 node) successfully.

|  |
| --- |
| catkin\_create\_pkg project2\_jetauto\_control rospy std\_msgs geometry\_msgs turtlesim |
| catkin\_create\_pkg lab4\_jetauto\_control rospy std\_msgs geometry\_msgs turtlesim |

* **Creating the Project 2 Package:**

Since the project refers to **lab4\_jetauto\_control**, I used it as the ROS interface. However, I developed the actual implementation in ***project2\_jetauto\_control***. To ensure both packages reflect the same code, I created symbolic links from ***project2\_jetauto\_control*** to point to the corresponding folders in **lab4\_jetauto\_control**. This setup allows me to run the node using either package name, while maintaining a single folder of code.

* **Building and Running:**  
  See [env-setup.md](https://github.com/jcp-tech/Seneca_Class_Notes/blob/master/Semester%202/AIG240%20-%20Robotics/Project%202/env-setup.md) for the full step-by-step note’s on what all I had to do to Complete the Setup & Project.

## How do you estimate that the robot moved roughly 1 meter in each of the steps?

To ensure the JetAuto robot moved approximately 1 meter per step of the square:

* **Odometry Feedback:**  
  The movement control script (*jetauto\_square.py*) subscribes to */odom* and continuously reads the robot’s x, y position.
* **Goal Logic:**  
  Each “move” command targets a specific (x, y) goal (e.g., from *(0, 0)* to *(1, 0)*), using a **Euclidean distance threshold** (*dist < 0.02*) to determine arrival.
* **Units:**  
  ROS odometry and Gazebo use SI units: *x=1.0* means 1 meter.
* **Code Verification:**  
  The code directly moves the robot until the odometry reports it has reached the next waypoint (goal\_x, goal\_y), so every straight segment in the pattern is as close as possible to 1 meter, regardless of drift/slippage.

Additionally, you can verify the scale by:

* Observing the robot’s path in Gazebo’s grid (each square = 1m),
* Or echoing */odom* data during/after movement for confirmation.

## What challenges did you face with making the robot move in a straight line while turning, and how did you overcome them?

**Challenges:**

* **Turning While Moving:**  
  It’s non-trivial to move the robot in a straight line while simultaneously rotating to a target heading (especially with omnidirectional/Mecanum wheels).
* **Odometry Drift:**  
  Simulated odometry can drift if movement and turning are not precisely controlled, causing the robot to “cut corners” or overshoot waypoints.
* **Shebang & Python Compatibility:**  
  Scripts defaulted to Python 3, but ROS Melodic uses Python 2 by default, causing execution errors.
* **Build Tool Conflicts:**  
  Attempting to use catkin\_make in a workspace previously built with catkin build led to errors.
* **Package Build Errors:**  
  Some non-critical packages failed to build, but this did not affect the main functionality.

**How I Solved Them:**

* **Code Structure:**  
  I implemented a move\_to() function that:
  + First moves straight to the (x, y) target, keeping orientation fixed (or gradually rotating if rotate\_while\_moving is True).
  + If rotate\_while\_moving is False, performs an explicit turn-in-place to the final angle after reaching the waypoint.
  + Uses odometry feedback with a tight threshold to ensure accurate stops.
* **Parameter Tuning:**  
  Used small distance and angle thresholds (0.02m for distance, 3 degrees for angle) and proportional control for velocity and rotation.
* **Batch Shebang Fix:**  
  Used find and sed to update all Python shebangs to Python 2.
* **catkin Build:**  
  Switched to catkin build for compatibility, ignoring unrelated errors.
* **Executable Permissions:**  
  Made sure all scripts were chmod +x before execution.

**Result:**  
With these changes, the robot moves accurately in the expected square pattern, turns at the corners as needed, and can even rotate while translating (for diagonal or curved movement as specified).

## Link’s:

* <https://drive.google.com/drive/folders/1xo8FyBYAjo8GBozoRDsZwsJWSkM6hG3W?usp=sharing>

NOTE: In this there are 2 MP4 Video’s – I’ve Done the Project on 2 Platform’s.

* + [AIG240\_Robotics\_Project2.mp4](https://drive.google.com/file/d/1JPTyoDoqwrWcOBU4PCbh6BB1MVfMqgr4/view?usp=drive_link)

This Done on Ubuntu20.04 with a [Docker Container](https://github.com/jcp-tech/Seneca_Class_Notes/tree/master/Semester%202/AIG240%20-%20Robotics/Docker) Configured with everything needed

Showing the Original Perspective.

* + [AIG240\_Robotics\_Project2\_Alternative - WSL (WindowsSubsystemForLinux).mp4](https://drive.google.com/file/d/1e17gNHV2rvS73NUXPDiliU4rVKEvmOCp/view?usp=drive_link)

This is Done on Ubuntu18.04 [WSL (Windows Subsystems for Linux)](https://github.com/jcp-tech/Seneca_Class_Notes/blob/master/Semester%202/AIG240%20-%20Robotics/SETUP.md) Configuration with everything needed.

Showing an Alternative Perspective.

* For More Information on all the Step’s I did to Complete the Project Refer to [env-setup.md](https://github.com/jcp-tech/Seneca_Class_Notes/blob/master/Semester%202/AIG240%20-%20Robotics/Project%202/env-setup.md)