ChatGPT

Setting Up ROS 2 + Gazebo for PX4 Drone Simulation (Ubuntu 24.04 LTS)

1. Install ROS 2 (Jazzy Jalisco) and Gazebo

Begin by installing ROS 2 **Jazzy** (the ROS 2 LTS for Ubuntu 24.04) and the Gazebo simulator. On Ubuntu 24.04 ("Noble"), ROS 2 Jazzy deb packages are available via the official ROS apt repository [1] [2]. Use the following commands to add the ROS 2 apt source and install the ROS desktop (which includes common GUI tools) and Gazebo packages:

```
# Add Ubuntu universe repository and ROS 2 apt source
sudo apt update && sudo apt install -y software-properties-common curl
sudo add-apt-repository universe

# Add ROS 2 GPG key and repository for Ubuntu 24.04 (Noble)
sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key \
     -o /usr/share/keyrings/ros-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] \
     http://packages.ros.org/ros2/ubuntu $(lsb_release -cs) main" | \
     sudo tee /etc/apt/sources.list.d/ros2.list >/dev/null

# Install ROS 2 Jazzy desktop and Gazebo simulator
sudo apt update
sudo apt install -y ros-jazzy-desktop ros-jazzy-gazebo-ros-pkgs
```

After installation, source the ROS 2 setup script in your shell (or add it to your `~/.bashrc` for future sessions):

```
source /opt/ros/jazzy/setup.bash
```

This ensures ROS 2 environment variables are set. The package `ros-jazzy-gazebo-ros-pkgs` provides ROS <-> Gazebo integration. By default on Ubuntu 24.04, the new **Gazebo (Ignition)** simulator is used instead of Gazebo Classic [2], so we will use the Ignition-based Gazebo for PX4 simulation (which is fully supported on Ubuntu 22.04+ and later [3]).

2. Install MAVROS and Dependencies

We will use **MAVROS** (MAVLink ROS interface) to bridge PX4 with ROS 2 topics. Install MAVROS packages via apt:

```
sudo apt install -y ros-jazzy-mavros ros-jazzy-mavros-extras
```

After installing, run the GeographicLib datasets setup script that MAVROS requires for accurate geodesy calculations (this installs geographic reference data for GPS):

```
# Download and run the GeographicLib dataset installer
wget https://raw.githubusercontent.com/mavlink/mavros/ros2/mavros/scripts/
install_geographiclib_datasets.sh
chmod +x install_geographiclib_datasets.sh
sudo ./install_geographiclib_datasets.sh
```

This will install gravity, magnetic field, and geoid models used by MAVROS for GPS and magnetometer data (ensuring correct conversion of coordinates) [4] .

3. Set Up PX4 Autopilot (SITL)

Next, set up the **PX4 Autopilot** Software-In-The-Loop (SITL) simulation environment. Clone the PX4-Autopilot repository and install its dependencies:

```
# Clone PX4-Autopilot (including submodules for models)
cd ~
git clone https://github.com/PX4/PX4-Autopilot.git --recursive
cd PX4-Autopilot

# Run PX4 dependency installation script for Ubuntu 24.04
bash Tools/setup/ubuntu.sh
```

The `ubuntu.sh` script will install necessary tools, libraries, and the Gazebo simulator (Gazebo “Harmonic”/ Ignition) for PX4 on Ubuntu 24.04 [5] . It also pulls the PX4 Gazebo model assets (via submodules) including the drone models and sensor plugins.

Once dependencies are installed, build the PX4 SITL code:

```
cd ~/PX4-Autopilot
make px4_sitl
```

This compiles the PX4 firmware for SITL. You should see a successful build of the PX4 SITL binary.

4. Launch the Drone Simulation in Gazebo

PX4 provides make targets to launch the simulation with a specific vehicle model. We will use the **X500 quadcopter** model (a Pixhawk 6C-based Holybro X500, similar in size to an S500 frame). The X500 model includes a Pixhawk controller and is a close match to a Holybro S500 kit. Launch the simulation with:

```
# Launch PX4 SITL with Gazebo (Ignition) using the X500 quadrotor model
cd ~/PX4-Autopilot
make px4_sitl gz_x500
```

This single command will start the PX4 SITL firmware and the Gazebo simulator with the X500 quadcopter in a default world [6] [7]. You should see a Gazebo window open with the quadcopter model and a PX4 SITL console running in your terminal. The PX4 SITL will automatically connect to Gazebo and start receiving sensor data and sending actuator commands (the system runs in lockstep simulation time).

Tip: The PX4 make command outputs the SITL console (`pxh>` shell). You can enter PX4 shell commands there if needed, or just leave it running. By default, QGroundControl (if installed) can also connect on UDP port 14550 to monitor the vehicle, though this is optional.

If you prefer to run PX4 and Gazebo separately (e.g., for debugging or if you want to launch Gazebo with ROS integration), you can run PX4 in *standalone mode*. For example, in one terminal:

```
PX4_GZ_STANDALONE=1 make px4_sitl gz_x500    # Start PX4 SITL and wait for
Gazebo
```

And in another terminal, launch Gazebo manually:

```
gz sim ~/PX4-Autopilot/Tools/simulation/gz/worlds/default.sdf    # Start Gazebo
Ignition
```

PX4 will detect the Gazebo server and connect (via the `gz_bridge` interface). However, for most cases the single `make px4_sitl gz_x500` is simplest.

5. Launch MAVROS and Confirm ROS 2 Sensor Topics

When the simulation is running, PX4 listens on localhost UDP port **14540** for offboard connections. We will now launch the MAVROS node to bridge PX4 to ROS 2. In a new terminal (with ROS 2 environment sourced), run:

```
# Launch MAVROS with a UDP bridge to PX4 SITL
ros2 launch mavros px4.launch fcu_url:=udp://:14540@localhost:14557
```

This uses the MAVROS launch file with `fcu_url` configured for PX4 SITL: it opens a MAVLink UDP connection on port 14540 (where PX4 is sending data) and directs outgoing messages to PX4's port 14557 [8] [9]. You should see MAVROS connect to the vehicle in the console logs. For example, MAVROS will log messages like "Got HEARTBEAT, connected. FCU: PX4 Autopilot" [10] indicating a successful connection.

At this point, the simulation's sensor data is being published on ROS 2 topics via MAVROS. You can list the ROS topics to see available sensor streams:

```
ros2 topic list | grep mavros
```

Key topics of interest include:

- **IMU data:** `/mavros/imu/data` (orientation and angular velocity, fused by PX4's estimator) and `/mavros/imu/data_raw` (raw IMU readings) – published as `sensor_msgs/Imu`. There is also `/mavros/imu/mag` for magnetometer if enabled [11].
- **GPS fix:** `/mavros/global_position/raw/fix` (GPS sensor data as `sensor_msgs/NavSatFix`) and `/mavros/global_position/raw/gps_vel` (GPS velocity).
- **Local pose:** `/mavros/local_position/pose` (vehicle pose in ENU frame) and velocity (`/mavros/local_position/velocity_local`), which come from PX4's EKF.
- **State and status:** `/mavros/state` (connected/armed state) etc.

For example, to echo the IMU readings and GPS fix in the terminal:

```
ros2 topic echo /mavros/imu/data_raw        # Raw accelerometer & gyro
ros2 topic echo /mavros/global_position/raw/fix   # GPS latitude/longitude/
altitude
```

These topics allow your custom EKF to subscribe and receive realistic sensor data from the simulation in real time.

6. Configuring Sensor Noise Levels

The default PX4 Gazebo simulation already injects **realistic sensor noise** for the IMU, GPS, etc., comparable to real hardware. For instance, the Gazebo model SDF for the X500 includes Gaussian noise for sensors. The barometric pressure sensor in the X500 model is defined with a noise standard deviation of 0.01 (in pressure units) [12]. The IMU plugin similarly applies noise and bias to accelerometer and gyroscope readings based on typical sensor characteristics (e.g., configured via parameters like noise density and random walk in the plugin).

If you need to **tune the noise levels** to more closely mimic a specific platform (e.g. Pixhawk IMU on a QAV250 vs. an S500), you have a couple of options:

- **Edit the model's SDF sensor parameters:** You can open the model SDF file (e.g. `PX4-Autopilot/Tools/simulation/gz/models/x500/model.sdf`) and adjust the `<noise>` tags for each

sensor. For example, you could modify the IMU's `<linear_acceleration>` and `<angular_velocity>` noise mean or standard deviation to reflect the noise density of your target IMU. The SDF format allows specifying Gaussian noise per axis. For instance, an IMU sensor section might look like:

```
<sensor name="imu_sensor" type="imu">
  <update_rate>250</update_rate>
  <imu>
    <angular_velocity>
      <x>
        <noise type="gaussian">
          <mean>0.0</mean>
          <stddev>0.002</stddev>
        </noise>
      </x>
      ...
    </angular_velocity>
    <linear_acceleration>
      ... <!-- similar noise settings -->
    </linear_acceleration>
  </imu>
</sensor>
```

Here you can tune the `<stddev>` values to simulate more or less IMU noise. After editing, restart the simulation to apply changes.

- **GPS noise:** In Gazebo Classic, a `<gpsNoise>` flag in the SDF toggles realistic GPS drift/jitter [13]. In the Ignition Gazebo setup, PX4's **GPS simulator** is used by default (PX4 internally converts the true vehicle pose to a GPS signal). PX4's GPS module already adds some noise, but you can further tune it via PX4 parameters. For example, check parameters like `SENS_GPS_NOISE` or `EKF2_GPS_P_NOISE` (if available) to adjust how the EKF handles GPS noise. If you want full control, you could disable the PX4 GPSSIM (set `SENS_EN_GPSSIM = 0` in the PX4 airframe config) and instead add a `sensor[type="gps"]` in the SDF with specified noise characteristics, though this is advanced. By default, for models like Iris/X500, GPS noise is enabled and handled either by Gazebo or PX4's simulation module [13] [14] – ensuring the reported GPS data in ROS has realistic variance.

- **Magnetometer:** Similarly, PX4 can simulate magnetometer data (via `SENS_EN_MAGSIM`). The simulation uses the world's magnetic field (defined in the world SDF) or a default field and adds noise. You typically don't need to tweak this unless you have specific requirements; just be aware noise and bias are present by default.

In summary, the **default noise settings are usually sufficient** to approximate real sensors on platforms like the Holybro X500/S500 or QAV250. You can fine-tune the standard deviations in the SDF if needed. Remember that if you remove *all* noise, PX4's EKF might consider the sensor readings "stuck" (e.g., zero variance) and produce errors [15] – so it's actually beneficial to have realistic noise in the simulation.

7. Logging and Data Recording for EKF Analysis

To compare your custom EKF against the PX4 EKF or analyze sensor data offline, you will likely want to **record the ROS 2 topics** during flight simulations:

- Use **Rosbag 2** to record topics. For example, in a new terminal, you can record IMU and GPS data (and any other relevant topics) with:

```
# Create a ROS 2 bag recording specified topics
ros2 bag record /mavros/imu/data_raw /mavros/imu/mag /mavros/global_position/
raw/fix /mavros/global_position/raw/gps_vel
```

Let the simulation run (e.g., fly the drone or execute an autonomous mission) while recording. Once done, stop the `ros2 bag` process (Ctrl+C). The recorded bag can be played back or converted to CSV for analysis. You can also record **all topics** with `ros2 bag record -a`, but specifying only the needed topics will keep the bag files smaller.

- PX4 also logs high-rate data in its own **ULog** format (located in the `PX4-Autopilot/log` directory for each run). These contain IMU, baro, mag, GPS, and EKF states from the PX4 perspective. You can use the PX4 **Flight Review** or **Pyulog** tools to extract data from ULog files [16] [17]. However, since you have ROS 2 topics via MAVROS, using rosbag is straightforward for feeding data to a custom EKF.

- Ensure that the simulation is running in **real time** (the PX4-Gazebo integration uses a synchronized simulation loop). If you intend to log long runs, consider running the simulation in real-time (not faster-than-real-time) so sensor data is timestamped realistically. This is usually the default when using the PX4 lockstep SITL.

- For offline analysis, you might also timestamp the data or log ground-truth. PX4 publishes ground truth via `/mavros/local_position/odom` (an Odometry message) which contains the simulated "true" pose. This can be useful to compare against your EKF's estimated pose.

Finally, always verify that your log contains the expected data (e.g., use `ros2 bag info <bagfile>` and `ros2 bag play` to inspect it). With the recorded IMU and GPS data, you can run your custom EKF offline and compare its state estimates to the ground truth or to PX4's onboard EKF (available via `/mavros/local_position/pose` or PX4 logs).

Sources:

- PX4 Documentation – *ROS with Gazebo Simulation* [18] [8], *Gazebo Simulation (Ignition)* [6] [7]
- PX4 User Guide (v1.13+) – *Simulating GPS Noise* [13], PX4 X500 Model SDF (sensor noise example) [12]
- ROS 2 MAVROS Setup – Satrya B. Pratama's guide (ROS2 Jazzy, MAVROS, Gazebo) [19] [9]

---

1 ROS2 Iron Irwini availability on UBUNTU 24.04 Noble Numbat - Robotics Stack Exchange
https://robotics.stackexchange.com/questions/114255/ros2-iron-irwini-availability-on-ubuntu-24-04-noble-numbat

2 3 5 6 7 Gazebo Simulation | PX4 Guide (main)
https://docs.px4.io/main/en/sim_gazebo_gz/

4 9 10 19 How to Setup Development Environment using ROS 2 (Jazzy), Mavros, and Gazebo | Satrya Budi Pratama
https://satrya.zeroinside.id/blog/how-to-setup-drone-development-environment-using-ros2-mavros-and-gazebo

8 18 ROS with Gazebo Simulation | PX4 User Guide (v1.13)
https://docs.px4.io/v1.13/en/simulation/ros_interface.html

11 Data from AP to ROS — Dev documentation - ArduPilot
https://ardupilot.org/dev/docs/ros-data-from-ap.html

12 14 Missing sensors for a Gazebo simulation - PX4 Autopilot - Discussion Forum for PX4, Pixhawk, QGroundControl, MAVSDK, MAVLink
https://discuss.px4.io/t/missing-sensors-for-a-gazebo-simulation/32787

13 16 17 Gazebo Classic Simulation | PX4 Guide (main)
https://docs.px4.io/main/en/sim_gazebo_classic/

15 Eliminating all sensor noise and bias - PX4 Autopilot - Discussion Forum for PX4, Pixhawk, QGroundControl, MAVSDK, MAVLink
https://discuss.px4.io/t/eliminating-all-sensor-noise-and-bias/13717