ROS 2 + Gazebo Simulation on an Aalto-Managed Laptop (*No sudo*)

Playbook for EKF/UKF Sensor Fusion Development

Prepared for: Aalto Internship – Drone Sensor Fusion

August 19, 2025

Abstract

This guide shows how to build a practical **ROS 2 + Gazebo** simulation environment on an Aalto-managed Ubuntu laptop **without** full admin (`sudo`) rights, so you can start fusing simulated `IMU` and `GPS` with your own EKF/UKF. It also includes a "Plan B" using a remote environment (VDI/HPC) for **PX4 SITL** with realistic quadrotor dynamics. All commands are copy–paste ready.

1 Scope / Constraints

- **Primary user model**: you can install via `pkcon` (PackageKit) but typically *cannot* use `sudo apt`.
- **Goal**: obtain Gazebo sensor streams (IMU, GPS) and ROS 2 topics for EKF development; add simple autonomous motion for regression tests.
- **State-estimation notation (Särkkä)**: we will refer to EKF variables as m_k, P_k, F_{k-1}, H_k, Q, R, and measurements y_k (consistent with your internship docs).

2 Decision Tree (Quick)

1. **Need full autopilot realism (PX4, missions, arming, failsafes)?** Use *Plan B (Remote PX4 SITL)*.
   Otherwise, for EKF dev with controllable motion and realistic sensors, *Plan A (Local, no-sudo)* is recommended.
2. **Later**: you can compare your EKF with PX4's estimator by switching to Plan B without changing your EKF topics.

3 Plan A (Local, No-`sudo`): RoboStack ROS 2 + Gazebo in User Space

A1. Install Micromamba (user-space Conda)

Open a terminal and run:

```
# Install micromamba into ~/.local
curl -L https://micro.mamba.pm/install.sh | bash
# Re-source your shell (choose your shell as prompted)
source ~/.bashrc   # or:  source ~/.zshrc
```

Tip: If your shell init isn't updated automatically, add:

```
export MAMBA_EXE=~/.local/bin/micromamba
export MAMBA_ROOT_PREFIX=~/.local/micromamba
eval "$($MAMBA_EXE shell hook -s bash)"  # or -s zsh
```

A2. Create a ROS 2 + Gazebo environment (RoboStack)

We use RoboStack (ROS on conda) and Gazebo (Ignition/GZ) packages **without root**. Create and activate an env:

```
micromamba create -n ros2 -y -c robostack-staging -c conda-forge \
  ros-humble-desktop \
  ros-humble-ros-gz-bridge ros-humble-ros-gz-sim \
  gz-sim gz-tools gz-msgs gz-transport \
  python=3.10
micromamba activate ros2
```

Notes:

- We pick `ros-humble` for maximal package availability in RoboStack. Your EKF node (topics, messages) will be the same when you later run on Jazzy/Ubuntu 24.04.
- The `ros_gz_*` packages provide the ROS ↔ Gazebo bridge and tools.

A3. Verify toolchain

```
# ROS 2
ros2 --help
ros2 pkg list | grep ros_gz
# Gazebo (GZ/Ignition)
gz --help
```

A4. Minimal quadrotor world with IMU + GPS

Create a working directory (no root needed):

```
mkdir -p ~/drone_sim/worlds && cd ~/drone_sim
```

Save the following as `worlds/drone_world.sdf` (simplified GZ SDF with a kinematic quadrotor, IMU and GPS sensors, and a basic trajectory plugin to generate gentle motion). You can tune noise later:

File: `worlds/drone_world.sdf`

```
<?xml version='1.0'?>
<sdf version='1.8'>
  <world name='drone_world'>
    <gravity>0 0 -9.81</gravity>
    <scene><ambient>0.6 0.6 0.6 1</ambient></scene>

    <model name='drone'>
      <pose>0 0 1.0 0 0 0</pose>
      <link name='base_link'>
        <inertial><mass>1.5</mass></inertial>
        <!-- IMU sensor -->
        <sensor name='imu' type='imu'>
          <always_on>1</always_on>
          <update_rate>200</update_rate>
```

```
        <imu>
          <angular_velocity>
            <x><noise type='gaussian'><mean>0</mean><stddev>0.002</
                stddev></noise></x>
            <y><noise type='gaussian'><mean>0</mean><stddev>0.002</
                stddev></noise></y>
            <z><noise type='gaussian'><mean>0</mean><stddev>0.002</
                stddev></noise></z>
          </angular_velocity>
          <linear_acceleration>
            <x><noise type='gaussian'><mean>0</mean><stddev>0.1</
                stddev></noise></x>
            <y><noise type='gaussian'><mean>0</mean><stddev>0.1</
                stddev></noise></y>
            <z><noise type='gaussian'><mean>0</mean><stddev>0.1</
                stddev></noise></z>
          </linear_acceleration>
        </imu>
      </sensor>
      <!-- GPS-like NavSat sensor -->
      <sensor name='gps' type='navsat'>
        <always_on>1</always_on>
        <update_rate>5</update_rate>
        <navsat>
          <position_sensing>
            <horizontal>
              <noise type='gaussian'><mean>0</mean><stddev>1.5</
                  stddev></noise>
            </horizontal>
            <vertical>
              <noise type='gaussian'><mean>0</mean><stddev>3.0</
                  stddev></noise>
            </vertical>
          </position_sensing>
        </navsat>
      </sensor>
    </link>

    <!-- Simple scripted motion (circle in XY) -->
    <plugin name='traj' filename='libgz-sim-movable-system.so'>
      <linear_velocity>0 0 0</linear_velocity>
      <angular_velocity>0 0 0.2</angular_velocity>
    </plugin>
  </model>

  <plugin name='scene_broadcaster' filename='libgz-sim-scene-
      broadcaster-system.so'/>
 </world>
</sdf>
```

Notes: The IMU/GPS noise values are conservative and can be adjusted to match your QAV250/S500 specs later. The simple motion plugin gently rotates yaw so you get nontrivial dynamics to test your EKF.

A5. Launch Gazebo

```
# From ~/drone_sim
gz sim -r worlds/drone_world.sdf
```

You should see the world with the `drone` model at $z = 1\,\mathrm{m}$. Leave it running.

A6. Bridge Gazebo sensors to ROS 2 topics

Create a bridge config that maps GZ sensor topics to ROS message types. Save this as `bridge.yaml` in `/drone_sim`:

File: `bridge.yaml`

```
- ros_topic_name: /sim/imu
  gz_topic_name: /world/drone_world/model/drone/link/base_link/sensor/
     imu/imu
  ros_type_name: sensor_msgs/msg/Imu
  gz_type_name: gz.msgs.IMU
  direction: GZ_TO_ROS

- ros_topic_name: /sim/gps/fix
  gz_topic_name: /world/drone_world/model/drone/link/base_link/sensor/
     gps/navsat
  ros_type_name: sensor_msgs/msg/NavSatFix
  gz_type_name: gz.msgs/NavSat
  direction: GZ_TO_ROS
```

Then run the bridge from another terminal (same conda env activated):

```
ros2 run ros_gz_bridge parameter_bridge --ros-args -p config_file:=
   bridge.yaml
```

Verify topics:

```
ros2 topic list | grep -E "/sim/imu|/sim/gps"
ros2 topic echo /sim/imu   # orientation, ang.vel, lin.acc
ros2 topic echo /sim/gps/fix
```

Now your EKF can subscribe to `/sim/imu` (`sensor_msgs/Imu`) and `/sim/gps/fix` (`sensor_msgs/NavSatFix`).

A7. EKF node I/O contract (consistent notation)

- **Subscribe**: `/sim/imu` (`sensor_msgs/Imu`), `/sim/gps/fix` (`sensor_msgs/NavSatFix`).
- **Publish**: `/ekf/odom` (`nav_msgs/Odometry`), optionally `/ekf/state` (custom) and `/ekf/P` (diag).
- **Filter loop**: at IMU rate, compute m_k^-, P_k^- via process model; on GPS events, compute innovation $v_k = y_k - h(m_k^-)$, S_k, gain K_k, and update m_k, P_k.

This follows your internship EKF equations and Jacobians F_{k-1}, H_k.

A8. Noise tuning (match QAV250/S500)

Adjust IMU/GPS noise in the SDF (`stddev`) to approximate your sensor model; then reflect this in Q and R for proper consistency. Typical steps:

1. Increase IMU `stddev` to observe faster covariance growth (P_k^-);
2. Increase GPS `stddev` to reduce update aggressiveness (smaller K_k).

4 Plan B (Remote, Full Realism): PX4 SITL + Gazebo + MAVROS

If you need realistic quadrotor dynamics with autonomous missions and a Pixhawk pipeline, run PX4 SITL in a **remote** environment (VDI/HPC node) where `sudo` is allowed. Summary (copy–paste once on the remote host):

```
# Install ROS 2 Jazzy + Gazebo + MAVROS (remote host with sudo)
sudo apt update
sudo apt install -y ros-jazzy-desktop ros-jazzy-gazebo-ros-pkgs \
  ros-jazzy-mavros ros-jazzy-mavros-extras
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

# PX4 SITL (Gazebo Ignition; Holybro X500 is close to S500)
cd ~
git clone https://github.com/PX4/PX4-Autopilot.git --recursive
cd PX4-Autopilot && bash Tools/setup/ubuntu.sh
make px4_sitl gz_x500
```

In a second terminal on the remote host:

```
# Bridge PX4 <-> ROS 2 via MAVROS (PX4 sends on UDP 14540)
ros2 launch mavros px4.launch fcu_url:=udp://:14540@localhost:14557

# Inspect topics
a) ros2 topic echo /mavros/imu/data_raw
b) ros2 topic echo /mavros/global_position/raw/fix
```

Your EKF can subscribe to MAVROS topics (replace `/sim/...` with `/mavros/...`).

5 Recording and Regression

```
# Rosbag (local or remote)
ros2 bag record /sim/imu /sim/gps/fix  # Plan A
# or
ros2 bag record /mavros/imu/data_raw /mavros/global_position/raw/fix  #
    Plan B
```

Re-run your EKF offline on the same bag for deterministic comparisons across noise settings and filter parameters.

6 Troubleshooting (Quick)

- **No gz command?** Ensure `micromamba activate ros2`. Check `which gz`.
- **Bridge shows unknown topics**: open `gz topic -l` to locate the exact sensor topic paths; update `bridge.yaml` accordingly.
- **ROS messages mismatch**: keep message types exact (`sensor_msgs/Imu`, `sensor_msgs/NavSatFix`).
- **Simulation too fast/slow**: add `-r` to run real-time, or limit GUI rendering.

7 Appendix: Exact Files (Copy–Paste)

A. Shell init for micromamba

```
export MAMBA_EXE=~/.local/bin/micromamba
export MAMBA_ROOT_PREFIX=~/.local/micromamba
eval "$($MAMBA_EXE shell hook -s bash)"
```

B. Bridge YAML (`bridge.yaml`)

(As above; adjust `gz_topic_name` with `gz topic -l` if your model name or world differs.)

C. EKF topic map (Plan A vs Plan B)

Plan A (Local)	Plan B (PX4/MAVROS)
`/sim/imu` (`sensor_msgs/Imu`)	`/mavros/imu/data_raw` (`sensor_msgs/Imu`)
`/sim/gps/fix` (`sensor_msgs/NavSatFix`)	`/mavros/global_position/raw/fix` (`sensor_msgs/NavSatFix`)
`/ekf/odom` (output)	`/ekf/odom` (output)

You are ready to implement and test your EKF/UKF. Start with Plan A to validate your filter against configurable sensor noise; then switch to Plan B to compare against PX4's onboard estimator under realistic flight dynamics.