What's My Robot's Yaw? Fusing IMU and LiDAR Data via an Extended Kalman Filter

Table of Contents

Introduction

Objective

Background and Context: The Problem of "Knowing" in Robotics

At its core, a mobile robot's autonomy depends on its ability to answer a deceptively simple question: "Where am I?" This is the problem of state estimation. Our robot's state, in its simplest form, includes its 2D position (x, y) and its orientation (heading, θ).

Robots don’t have an innate sense of location. Instead, they rely on sensors, and all sensors are imperfect. They suffer from:

- Noise: Random fluctuations in readings (e.g., “jitter” in a gyroscope).
- Bias: A constant, underlying error (e.g., a compass reading that is always 5 degrees off).
- Drift: Errors that accumulate over time (e.g., a wheel encoder that becomes less accurate the farther you travel).

No single sensor provides a perfect, reliable estimate of the robot's state. This is where *Sensor Fusion* comes in.

The Core Concept: Sensor Fusion & The Kalman Filter
The Kalman Filter is useful mathematical algorithm that optimally combines noisy sensor data to produce a statistically best guess of the true state of a system. It also tracks the uncertainty (or covariance) of that estimate.

The Lab Exercise:

1. Implement the EKF to fuse IMU heading and LiDAR-based odometry to estimate the state of the robot.
2. Use a teleoperation keyboard node to drive the robot in a specific pattern (e.g., a square or spin in place).
3. Plot the EKF-Fused path using python matplotlib
4. Analyze the covariance ellipses drawn by the filter. These ellipses represent the filter's confidence. You should see them grow during fast motion or prediction steps and shrink when a good sensor update is received.

Background

The Kalman Filter is an elegant mathematical algorithm that optimally combines noisy sensor data to produce a statistically best guess of the true state of a system.

Think of it as a "trust manager" for your sensors:

- If your gyroscope is very stable but your magnetometer is noisy, the filter will learn to trust the gyro more.
- If you get a very confident GPS reading, the filter will correct its position and reduce its overall uncertainty.

The key innovation is its two-step, recursive process:

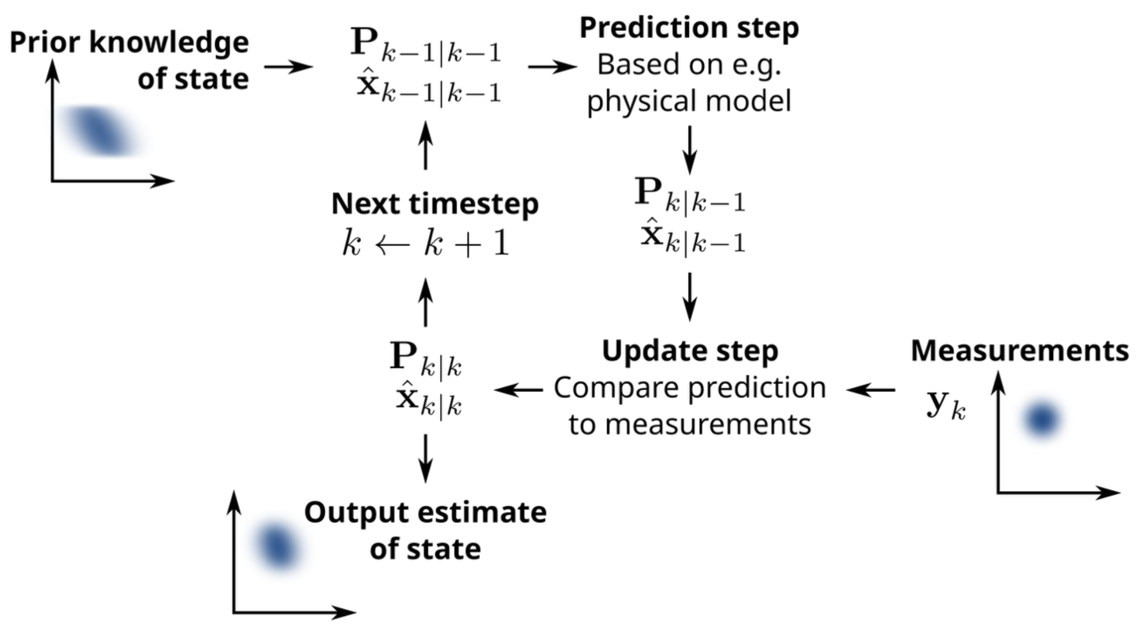
1. **Predict**: Use your robot's motion model to predict where the robot will be and how much more uncertain it has become since the last update.

2. **Update**: Use a sensor measurement to correct the prediction. The filter blends the prediction and the measurement, weighted by their respective uncertainties.

Extended Kalman Filter

The Extended Kalman Filter (EKF) emerged as the essential and most widely adopted extension of the Kalman Filter to handle the nonlinear systems prevalent in the real world, such as those in robotics, aerospace, and tracking. The primary challenge with nonlinearity is that propagating a Gaussian probability distribution through a nonlinear function does not result in another Gaussian distribution, breaking a core assumption of the original filter.

The EKF's ingenious solution to this problem is to linearize the nonlinear system around the current best state estimate. It does this by employing a first-order Taylor expansion, using the Jacobian matrices of the system's motion and observation models. In essence, the EKF approximates the nonlinear system at each timestep with a custom, locally linear filter that is valid only in the immediate vicinity of the current operating point, thus allowing the application of the classic Kalman Filter equations to a locally linearized version of the problem.


Prior knowledge of state
$\mathbf{P}_{k-1|k-1}$ $\hat{\mathbf{x}}_{k-1|k-1}$
Prediction step
Based on e.g. physical model
$\mathbf{P}_{k|k-1}$ $\hat{\mathbf{x}}_{k|k-1}$
Next timestep
$k \leftarrow k+1$
Update step
Compare prediction to measurements
Measurements
$\mathbf{y}_k$
$\mathbf{P}_{k|k}$ $\hat{\mathbf{x}}_{k|k}$
Output estimate of state

While the EKF is a powerful tool, this linearization introduces its own set of challenges and limitations. The most significant is that the filter is only an approximation, and its performance degrades rapidly as the system's nonlinearity increases or the uncertainty in the state estimate grows large.

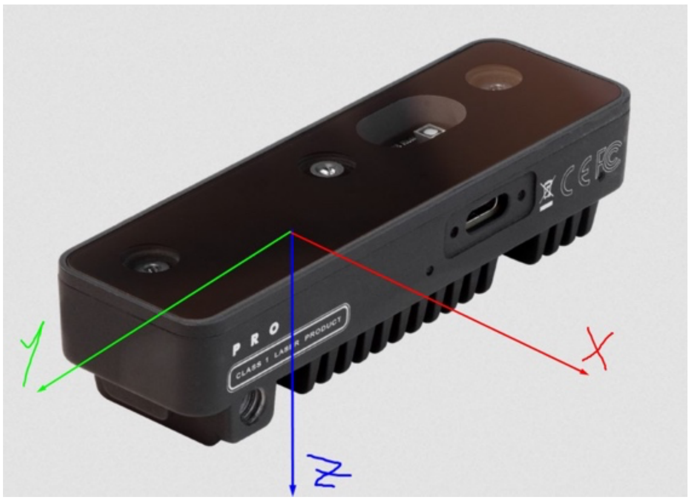
If the initial estimate is poor or the system undergoes aggressive maneuvers, the linearization around an incorrect point can lead to highly suboptimal performance or even catastrophic divergence of the filter.

Furthermore, the need to compute Jacobian matrices analytically can be a complex and error-prone process. Despite these drawbacks, the EKF established itself as the de facto standard for nonlinear state estimation for decades due to its conceptual simplicity and computational tractability, forming the backbone of early navigation systems and enabling countless applications in autonomous robotics where models for sensor fusion and motion planning are inherently nonlinear.

Inertial Measurement Unit (IMU) from the OAK-D Pro Camera

For this lab we will be working with IMU data to augment LiDAR odometry in ROS2. This is a useful step in creating a robust state estimation pipeline. The process begins by subscribing to the standard the IMU's topic, which provides raw or processed data from the inertial measurement unit, including 3-axis linear acceleration and angular velocity.

The primary value of the IMU in this context is its high-frequency, short-term accuracy for tracking rotational dynamics. LiDAR odometry, which estimates motion by aligning consecutive laser scans, can produce accurate displacement but often fails during rapid rotations or in feature-poor environments.



By fusing the high-bandwidth angular velocity from the IMU, the filter can precisely track the robot's orientation between LiDAR updates, effectively de-skewing the point cloud and providing a more accurate prediction for the scan-matching process.

Check out the ROS2 message definition for working with IMU data, *sensor_msgs/Imu* and determine which variables will be necessary for the task.

Setup

GoBilda Hardware Bring up

Follow the instructions provided in the *gobilda_driver* starting from the section: ***Compiling the Driver***

That will get you started with the software to command the robot via velocities and read scan data from a topic. These instructions are the same as the previous lab. The main differences are that the simulation this time won't be as useful since the sim does not include an IMU sensor.

Furthermore, we will be working with the camera this time since that is where the IMU is placed. Thus, make sure that it is plugged in to the Orin via USB and powered on. As you can read from the repo the same launch file as the previously used one will launch the camera node.

Finally check out the other nodes in the *gobilda_utilities* package. There you will find the *teleop_gobilda* nodes which you can run with the following command:

```
ros2 run gobilda_utilities teleop_gobilda
```

The above node will allow you to control your robot using your keyboard. Just as we did a while back using the turtlebot_sim nodes. You this node to drive the robot around and chart the quality of your EKF's estimate.

Lab Assignment

Robot's Heading via Extended Kalman Filter. No starter code provided this time.

Tasks

The Lab Exercise:

1. Implement the EKF to fuse IMU heading and LiDAR-based odometry.
2. Use a teleoperation keyboard node to drive the robot in a specific pattern (e.g., a square or spin in place).
3. Plot the EKF-Fused path
4. Include a write-up analyzing the covariance calculated by the filter. This represents the filter's confidence.

State Model Representation:

You should use the following state model to represent our 2D robot:

$$[x, y, \theta]$$

Your predict method should try to estimate the new orientation of the robot by *integration* of the angular velocity on the z-axis (yaw).

Finally, your *update* method should incorporate the LiDAR odometry message to be integrated into the overall EKF odometry estimate. For intial covariance values you can use small numbers such as 1.0e-4

Note that the non-linearity comes from the fact that the robot's heading is bounded between the values: $[-\pi, \pi]$. Meaning at some point you're prediction for the robot's yaw from the IMU integration needs to wrap around these values.

Conditions:

- Your callbacks for the should **only** copy the data from the topics. The callback shouldn't perform anymore calculations. You will need to use queues to push the IMU data and LiDAR data.
- Implement a **separate timer** function that runs at 20Hz that processes the data in the queues.
- Your node should **publish the transformation** computed after the LiDAR data is integrated (after the update step)

A note on plotting data. You can use the ros2bag CLI to record data coming over the topics of the robot. Then you can write a python script to load the data and plot it.

Important topics:

- /oak/camera/imu_data/ (IMU data)
- /kiss-icp/odom (LiDAR odom)

Submission

No demo for this lab simply turn in the plots and the source code. I will need you to submit your source code on Canvas. Compress your entire ROS2 workspace and upload the files to canvas.