# Advanced Vehicle State Estimation with UKF in a SIL Environment

**Sanjeev Godbole**

August 12, 2025

---

## 1 Project Objective & Motivation

This project implements a foundational state estimation system for an autonomous vehicle using NVIDIA Isaac Sim and ROS 2. The core of the project is a sensor fusion node that subscribes to simulated Lidar and IMU data, processes it using a full **Unscented Kalman Filter (UKF)**, and publishes a real-time vehicle state estimate as a standard `nav_msgs/msg/Odometry` message.

The primary motivation is to tackle a fundamental challenge in robotics: creating a single, reliable understanding of a robot's state from multiple, imperfect sensors. By fusing data from different sensor modalities, we can create a system that is more robust and accurate than one relying on a single source of information. This project serves as a practical, hands-on introduction to the theory and implementation of modern probabilistic robotics.

## 2 Key Features

- **Software-in-the-Loop (SIL) Architecture**: The entire project is developed and validated in a SIL environment, allowing for rapid prototyping and testing without physical hardware.
- **High-Fidelity Simulation**: Utilizes NVIDIA Isaac Sim to create a physically accurate simulation environment with a sensor-equipped vehicle.
- **ROS 2 Integration**: Seamlessly integrates with ROS 2 (Jazzy) using the `omni.isaac.ros2_bridge` extension for real-time data transfer.
- **Advanced Sensor Fusion**: Fuses data from a simulated Lidar and IMU, leveraging the strengths of each sensor to overcome individual weaknesses.
- **Unscented Kalman Filter (UKF)**: Implements the full UKF algorithm in Python using NumPy. This includes sigma point generation, a non-linear motion model for the prediction step, and a measurement update step.
- **Standardized Messaging**: Publishes the final state estimate using the standard `nav_msgs/msg/Odometry` message type and broadcasts the corresponding TF2 transform (`odom` -¿ `base_link`).
- **Data Analysis**: Includes a workflow for recording performance data with `ros2 bag` and generating high-quality trajectory plots with `matplotlib`.

## 3 Technical Deep Dive: The Unscented Kalman Filter

This project uses an Unscented Kalman Filter (UKF) instead of a simpler Extended Kalman Filter (EKF). While an EKF works by linearizing non-linear models, a UKF offers a more robust solution by using the **unscented transform** to approximate the probability distribution of the state directly.

## 3.1 Sigma Point Generation

The first step is to generate a set of $2n+1$ sigma points (where $n$ is the dimension of the state vector) that capture the current state estimate ($x$) and its uncertainty ($P$).

### 3.1.1 Calculate Weights

The sigma points are weighted to recover the mean and covariance. The weights ($W^{(m)}$ for mean, $W^{(c)}$ for covariance) are calculated based on scaling parameters $\alpha, \beta, \kappa$.

$$\begin{aligned}
\lambda &= \alpha^2(n+\kappa) - n \\
W_0^{(m)} &= \frac{\lambda}{n+\lambda} \\
W_0^{(c)} &= \frac{\lambda}{n+\lambda} + (1-\alpha^2+\beta) \\
W_i^{(m)} = W_i^{(c)} &= \frac{1}{2(n+\lambda)} \quad \text{for } i = 1, \ldots, 2n
\end{aligned}$$

### 3.1.2 Generate Points

The sigma points $\mathcal{X}$ are generated as follows:

$$\begin{aligned}
\mathcal{X}_0 &= x \\
\mathcal{X}_i &= x + (\sqrt{(n+\lambda)P})_i \quad \text{for } i = 1, \ldots, n \\
\mathcal{X}_i &= x - (\sqrt{(n+\lambda)P})_{i-n} \quad \text{for } i = n+1, \ldots, 2n
\end{aligned}$$

## 3.2 Prediction Step

The generated sigma points are propagated through the non-linear motion model $g(\cdot)$ to get a predicted set of points.

$$\begin{aligned}
\mathcal{X}_i^* &= g(\mathcal{X}_i, u_k) \quad \text{for } i = 0, \ldots, 2n \\
x_k^- &= \sum_{i=0}^{2n} W_i^{(m)} \mathcal{X}_i^* \\
P_k^- &= \sum_{i=0}^{2n} W_i^{(c)} (\mathcal{X}_i^* - x_k^-)(\mathcal{X}_i^* - x_k^-)^T + Q
\end{aligned}$$

## 3.3 Update Step

The predicted state is corrected using a measurement $z_k$ from a sensor via the measurement model $h(\cdot)$.

$$\begin{aligned}
\mathcal{Z}_i &= h(\mathcal{X}_i^*) \quad \text{for } i = 0, \ldots, 2n \\
\hat{z}_k &= \sum_{i=0}^{2n} W_i^{(m)} \mathcal{Z}_i \\
S_k &= \sum_{i=0}^{2n} W_i^{(c)} (\mathcal{Z}_i - \hat{z}_k)(\mathcal{Z}_i - \hat{z}_k)^T + R \\
T_k &= \sum_{i=0}^{2n} W_i^{(c)} (\mathcal{X}_i^* - x_k^-)(\mathcal{Z}_i - \hat{z}_k)^T \\
K_k &= T_k S_k^{-1} \\
x_k &= x_k^- + K_k(z_k - \hat{z}_k) \\
P_k &= P_k^- - K_k S_k K_k^T
\end{aligned}$$

# 4 How to Run the Project

## 4.1 Step 1: Isaac Sim Scene Setup

Launch Isaac Sim and prepare the simulation scene.

- **Add Robot**: Drag a robot model (e.g., Carter v2) into the scene from the Content Browser.
- **Add Sensors**: Create and attach Lidar and IMU sensors to the robot model using the `Create -> Isaac -> Sensors` menu.
- **Configure Action Graph**: Create an Action Graph and add the necessary nodes to publish the sensor data to ROS 2 topics. This involves using an `On Playback Tick` node to trigger specialized publisher nodes (`ROS2 Publish Laser Scan`, `ROS2 Publish Imu`) for each sensor.

Once the scene is prepared, press the **Play** button to start the simulation and begin publishing data.

## 4.2 Step 2: Build the ROS 2 Workspace

In a terminal, build the package.

```
cd ~/ros2_ws
colcon build --symlink-install
```

## 4.3 Step 3: Run the Fusion Node

Open a new terminal, source the environment, and run the node.

```
source ~/ros2_ws/install/setup.bash
ros2 run sensor_fusion fusion_node
```

## 4.4 Step 4: Visualize in RViz2

Open a final terminal and launch RViz2.

```
source /opt/ros/jazzy/setup.bash
ros2 run rviz2 rviz2
```

In RViz2, set the **Fixed Frame** to `odom` and add an **Odometry** display with the topic set to `/fused_vehicle_state`.

# 5 Data Analysis: Recording and Plotting Trajectory

To create a high-quality plot of the robot's trajectory for analysis, follow these steps.

## 5.1 Step 1: Record the Trajectory Data

While the simulation and fusion node are running, open a new terminal and use `ros2 bag` to record the output data.

```
# Navigate to a directory to save the data
cd ~/isaacsim

# Record the topic into a folder named my_robot_data
ros2 bag record -o my_robot_data /fused_vehicle_state
```

Let the recording run for a sufficient amount of time, then stop it by pressing `Ctrl+C` in the terminal.

### 5.2 Step 2: Plot the Trajectory from the Bag File

This project includes a Python script, `live_plotter.py`, to generate a plot from the recorded data. This is done by playing back the bag file and having the script listen to the replayed data.

#### 5.2.1 Terminal 1: Play the Data

```bash
# Navigate to the directory containing the bag file
cd ~/isaacsim

# Play back the recorded data
ros2 bag play my_robot_data
```

#### 5.2.2 Terminal 2: Run the Plotter

While the bag file is playing, open a second terminal.

```bash
# Source the ROS 2 environment
source /opt/ros/jazzy/setup.bash

# Run the plotting script
python3 live_plotter.py
```

Once the bag playback is complete, stop the plotter script with `Ctrl+C`. It will then automatically generate, save, and display the final trajectory plot.