UKF-Based Bicycle State Estimator

Mauricio Vergara, Karthik Rajgopal, Jannik Heinen

June 1, 2025

Report

Bicycle Unscented Kalman Filter State Estimator Implementation

Introduction

In this project, a state estimator was developed for a bicycle using an Unscented Kalman Filter (UKF). The goal was to accurately track the bicycle's cartesian x-position, y-position, and heading angle over time, given steering inputs, pedaling inputs, and occasional position measurements meant to simulate a GPS. Because the bicycle's motion is nonlinear, a UKF was chosen as a derivative-free alternative to an EKF in order to better handle this nonlinearity. A UKF uses a set of sigma-point samples to propagate the state probability distribution through the nonlinear state and measurement models, yielding more accurate mean and covariance estimates than linearization. The UKF was integrated into the provided estInitialize and estRun functions, adhering to the project's constraints (only numpy and scipy libraries, etc.) while ensuring clear structure and thorough documentation.

System Model and Problem Setup

Bicycle Dynamics: The bicycle moves on a flat plane with x–y coordinates, where the y-axis points North and the x-axis points East. We model the rear wheel position (x_1, y_1) and the heading angle θ (orientation of the bicycle frame). The continuous-time kinematic model is:

$$\begin{aligned}\dot{x}_1(t) &= v(t)\cos\theta(t),\\ \dot{y}_1(t) &= v(t)\sin\theta(t),\\ \dot{\theta}(t) &= \frac{v(t)}{B}\tan(\gamma(t)),\end{aligned}$$

where $v(t)$ is the bicycle's linear velocity, $\gamma(t)$ is the steering angle (angle of the front wheel relative to the frame), and $\omega(t)$ is the pedal rotation speed. The bicycle is geared such that the rear wheel's angular speed is $5\times$ the pedal speed, so we compute the velocity as:

$$v(t) = 5r\omega(t),$$

where r is the rear wheel radius, and B is the wheelbase (distance between the front and rear wheels).

This model assumes no wheel slip and piecewise-constant inputs between sampling instants (a reasonable approximation given the data rate). Using Euler integration with a time step dt, we discretize the dynamics:

$$\begin{aligned} x_1^{(k+1)} &= x_1^{(k)} + v^{(k)} \cos\theta^{(k)}\, dt, \\ y_1^{(k+1)} &= y_1^{(k)} + v^{(k)} \sin\theta^{(k)}\, dt, \\ \theta^{(k+1)} &= \theta^{(k)} + \frac{v^{(k)}}{B^{(k)}} \tan(\gamma^{(k)})\, dt, \end{aligned}$$

with $v^{(k)} = 5r^{(k)}\omega^{(k)}$. The parameters B and r may be uncertain, as discussed next.

Measurement Model: We receive discrete position measurements $\mathbf{p}^{(k)} = [p_x, p_y]^\top$ of the bicycle's center at time t_k. The center point lies midway between the wheels and relates to the rear wheel position and heading angle as:

$$\mathbf{p}^{(k)} = \begin{bmatrix} x_1(t_k) + \frac{1}{2}B\cos\theta(t_k) \\ y_1(t_k) + \frac{1}{2}B\sin\theta(t_k) \end{bmatrix}.$$

Measurements arrive irregularly (about every 0.5 s) and may be missing; missing entries are denoted as NaN. The measurements are unbiased but affected by noise (e.g., GPS noise and timing jitter).

Parameter Uncertainty: The physical parameters of the bicycle are not perfectly known. The wheelbase B can vary by $\pm 10\%$, and the radius r by $\pm 5\%$ from their nominal values $B_{\text{nom}} = 0.8\,\text{m}$ and $r_{\text{nom}} = 0.425\,\text{m}$. Additionally, small errors may exist in the reported steering angle and pedal speed, and minor slip may occur. We incorporate B and r as part of the state vector—treating them as slowly varying—to allow the UKF to adapt them over time based on measurements.

Available Data: Each record contains time, steering angle γ, pedal speed ω, measured x and y position, and optionally ground truth. We use only the first five columns (time, inputs, and measurements) for estimation. The initial position is near $(0, 0)$, and the bike heads approximately North-East at startup—guiding our choice for initial state estimation.

Unscented Kalman Filter Approach

The Unscented Kalman Filter (UKF) iteratively predicts the bicycle's state forward using the true nonlinear motion model, and then updates the estimate using measurements (when available). Unlike the Extended Kalman Filter (EKF), the UKF does not rely on linearization. Instead, it approximates the state distribution using a set of carefully chosen sigma points that capture both mean and covariance. These points are propagated through the nonlinear dynamics, and from the transformed set, a new mean and covariance are computed that resemble the true distribution up to second-order approximation. This process—called the Unscented Transform—preserves the distribution's characteristics more accurately than linear approximations for strongly nonlinear systems.

At each time step, the UKF performs the following steps:

1. Sigma Point Generation: From the current state estimate (mean $\hat{\mathbf{x}}_{k|k}$ and covariance $\mathbf{P}_{k|k}$), a set of $2L+1$ sigma points is generated, where L is the state dimension. These are computed using the square root (Cholesky decomposition) of the scaled covariance. The sigma points include one at the mean and symmetric points along each principal axis of the covariance. The spread and weighting of the points are governed by parameters α, κ, and β.

2. State Prediction (Time Update): Each sigma point is propagated through the bicycle motion model $f(\cdot)$ using the current control inputs γ and ω over the time interval dt. This yields a predicted set of sigma points for time t_{k+1}. From these, the predicted mean $\hat{\mathbf{x}}_{k+1|k}$ and covariance $\mathbf{P}_{k+1|k}$ are computed as weighted averages. Process noise covariance $\mathbf{Q}$ is then added to capture uncertainty in the model.

3. Measurement Prediction: If a position measurement $\mathbf{z}_{k+1}$ is available (i.e., not NaN), the predicted sigma points are passed through the measurement model $h(\cdot)$ to produce expected measurements. From these, the predicted measurement mean $\hat{\mathbf{z}}_{k+1|k}$ and measurement covariance $\mathbf{S}$ are computed, including measurement noise covariance $\mathbf{R}$.

4. State Update (Measurement Update): The cross-covariance $\mathbf{P}_{xz}$ between the state and measurement is computed, and the Kalman gain is calculated as $\mathbf{K} = \mathbf{P}_{xz}\mathbf{S}^{-1}$. The actual measurement $\mathbf{z}_{k+1}$ is used to correct the prediction:

$$\hat{\mathbf{x}}_{k+1|k+1} = \hat{\mathbf{x}}_{k+1|k} + \mathbf{K}\left(\mathbf{z}_{k+1} - \hat{\mathbf{z}}_{k+1|k}\right),$$

$$\mathbf{P}_{k+1|k+1} = \mathbf{P}_{k+1|k} - \mathbf{K}\mathbf{S}\mathbf{K}^{\top}.$$

If no measurement is available, the predicted state is carried forward without update.

Sigma Point Parameters: We use a scaled unscented transform with typical values: $\alpha = 1.0$, $\beta = 2$ (optimal for Gaussian priors), and $\kappa = 0$. These define the scaling parameter:

$$\lambda = \alpha^2(L + \kappa) - L,$$

which is used to compute the weights for each sigma point. The weights are chosen to ensure the weighted mean is unbiased and the spread reflects the prior distribution.

Handling Angles: The heading angle θ is angular and must be wrapped within $[-\pi, \pi]$. We normalize angular values when computing the means or differences that involve θ. This ensures continuity across wrap-around boundaries and preserves consistency in the UKF's internal representation. Since θ only appears through $\sin\theta$ and $\cos\theta$ in the measurement model, wrap-around is implicitly handled during updates.

Initial Conditions and Covariances: We initialize the state as:

$$\hat{\mathbf{x}}_{0|0} = [0,\ 0,\ \pi/4,\ B_{\text{nom}},\ r_{\text{nom}}],$$

reflecting a starting point near the origin and heading approximately northeast. The initial covariance matrix is:

$$\mathbf{P}_{0|0} = \text{diag}\left([5^2,\ 5^2,\ (0.785)^2,\ (0.08)^2,\ (0.021)^2]\right),$$

which corresponds to 5 m position uncertainty, 45° heading uncertainty, and 10% and 5% uncertainty in B and r, respectively.

Process Noise: The process noise covariance is designed to capture unmodeled dynamics. We use:

$$\mathbf{Q} = \text{diag}([0.1^2,\ 0.1^2,\ 0.05^2,\ 0,\ 0]) \cdot dt,$$

introducing small uncertainty to (x, y, θ) per time step and assuming B and r are constant over the run.

Measurement Noise: From calibration data (e.g., when the bike is stationary), we assume the measurement noise standard deviation is approximately $\sigma_p = 1.0\,\text{m}$. Thus, the measurement noise covariance is:

$$\mathbf{R} = \text{diag}([\sigma_p^2,\ \sigma_p^2]).$$

Logging and Outputs: The estimator maintains internal state (mean and covariance) across time steps. It can log estimation errors for debugging or evaluation. The main script handles performance evaluation using ground truth data. The output includes (x, y, θ) and the internal state dictionary. Additional diagnostic tools, such as uncertainty ellipses or error plots, can be derived from the stored data or post-processed as needed.

Implementation Plan and Structure

The UKF design was then integrated into the codebase. The solution is structured around two key functions:

- `estInitialize()`: Initializes the filter by setting up the internal state, including the initial mean, covariance, noise matrices, and sigma point weights. It returns a tuple containing the internal state, student name(s), and the estimator type string.
- `estRun(time, dt, internalState, steeringAngle, pedalSpeed, measurement)`: Performs the UKF time and measurement updates for each timestep using the current inputs and available measurements. It returns the estimated state (x, y, θ) as well as the updated internal state.

Internal State Representation: We define `internalState` as a Python dictionary containing all necessary UKF components. This format is intuitive and extensible. The dictionary includes:

- `mean`: the current state mean vector $\hat{\mathbf{x}}_{k|k}$
- `cov`: the current state covariance matrix $\mathbf{P}_{k|k}$
- `Q_base`: the base process noise covariance matrix (scaled by dt at each step)
- `R`: the measurement noise covariance matrix
- `Wm`, `Wc`: sigma point weights for mean and covariance
- `state_dim`: dimensionality of the state vector L

Using a dictionary allows us to pass the internal state across function calls cleanly and update individual fields as needed. The function `estRun` modifies this structure in-place and returns it alongside the current estimated state, which is used for logging and evaluation by the main script.

Code Design and Constraints: The implementation adheres strictly to the project constraints: only standard libraries such as `numpy` and `scipy` are used—for instance, `numpy.linalg.cholesky` for matrix square root operations. The code is modular and well-commented, with each UKF step—sigma point generation, prediction, and measurement update—clearly delineated for readability and maintainability.

Implementation Notes:

- The sigma-point spread scaling factor $c = \sqrt{L + \lambda}$ is computed indirectly from the weight $W_m[1] = \frac{1}{2(L+\lambda)}$, allowing us to avoid explicitly storing λ. Cholesky decomposition is used to compute $\sqrt{P}$, with a small regularization term (1e$-$9) added to the diagonal if P is nearly singular. This ensures numerical stability in computing sigma points.

- The motion model propagates each sigma point using its own values of B and r, treating them as constant over each small time interval dt. The sigma points are integrated using Euler's method. To correctly compute the mean of the heading angle θ, we use a weighted average of $\sin(\theta)$ and $\cos(\theta)$, ensuring proper wrap-around behavior near $\pm\pi$. Angle differences are also normalized to $[-\pi, \pi]$ when computing the covariance, maintaining filter consistency.

- The process noise covariance Q is scaled by the timestep dt, reflecting that less uncertainty accumulates over shorter intervals. The base matrix Q_{base} represents the process noise over one second.

- The measurement update is skipped if either position component is missing (i.e., NaN). If a measurement is available, we propagate all predicted sigma points through the measurement model:
$$h(x) = \begin{bmatrix} x_{\text{rear}} + 0.5B\cos\theta \\ y_{\text{rear}} + 0.5B\sin\theta \end{bmatrix}$$
  We then compute the predicted measurement mean, innovation covariance S, and state-measurement cross-covariance P_{xz}. The Kalman gain is calculated as $K = P_{xz}S^{-1}$, and the innovation (difference between actual and predicted measurements) is applied to update the state and covariance. The heading angle θ is normalized again after the update to maintain it within $[-\pi, \pi]$.

- The final output consists of the filtered estimate (x, y, θ) and the updated internal state. Although the internal state's θ is already normalized, we explicitly normalize the output angle to ensure consistency.

Overall, the implementation matches the theoretical UKF design and satisfies all project constraints. The structure is clean, stable, and interpretable, with thoughtful handling of nonlinearities and angle periodicity.

Conclusion and Usage

Given the `estInitialize` and `estRun` functions outlined, integration into the project framework is trivial. The script calls `estInitialize()` to set up the filter, and iteratively calls `estRun()` at each timestep using whatever sensor inputs are available at that point in time. The Unscented Kalman Filter fuses control inputs (steering angle and pedal speed) with noisy position measurements and the nonlinear state model, producing a robust state estimate.

Model uncertainty, such as imperfect knowledge of the wheelbase B and tire radius r, is accounted for in the solution through adaptive parameter estimation. Noise covariances are also scaled appropriately. The main program assesses estimation performance using the ground truth data provided in the final time step, and calculates errors based on this. Errors should remain small throughout the run if the filter runs as intended.

This implementation results in a robust, UKF-based state estimator for the bicycle. It adheres to project specifications, is numerically stable, and is clearly organized for readability and testing. The design choices—especially the joint estimation of position, heading, and physical parameters—are transparent in the code and supported by sound theoretical reasoning.

Team Contribution

Karthik Rajgopal - UKF implementation, data processing, report introduction and problem statement

Jannik Heinen - UKF implementation, main code structure and logic, report approach and implementation

Mauricio Vergara - UKF implementation, code cleanup, report conclusion and format