BLUE SKY

IMU-Based Navigation System for Autonomous Vehicles Using Different Types of Kalman Filters

Anschel Almog

Supervised by:
Oshra Belapolsky

Faculty of Electrical and Computer Engineering
Control and Robotics Machine Learning Lab (CRML)
Technion - Israel Institute of Technology

This project is a collaboration between RAFAEL and CRML.

October 8, 2024

Contents

Definitions and Abbreviations

DCM	Direction Cosine Matrix
IMU	Inertial Measurement Unit
EKF	Extended Kalman Filter
IEKF	Iterated Extended Kalman Filter
IMU	Inertial Measurement Unit
KF	Kalman Filter
RMSE	Root Mean Square Error
UKF	Unscented Kalman Filter
UT	Unscented Transform
Euler ψ	
Euler θ	
Euler ϕ	

1 Introduction

talk about the project and non GPS navigation, IMU aided navigation, IMU VS image matching

2 Maps

Maps are the cornerstone of non-GPS navigation systems, providing essential spatial context to reconcile estimated trajectories with actual geographic locations. In IMU based navigation, the choice of map can significantly influence the accuracy and efficiency of trajectory planning and analysis. Factors such as the method of acquisition, scale, and date of capture play a crucial role in determining a map's utility for a specific navigation task.

In this project, we employ a (DEM) to represent the topographical features of the surrounding area. This particular map offers detailed elevation data, crucial for understanding the terrain's contours and variations, which is especially important for navigating in areas where altitude changes are significant.

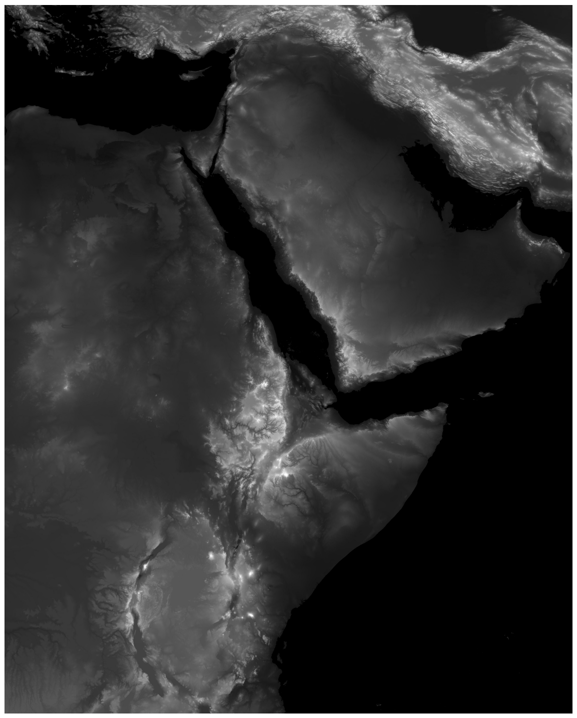

Figure 1: Geographical area covered by the entire map

The map, as illustrated in Figure 1, covers an extensive geographical area, delineated by the Northwest corner at (40°N, 20°E) and the Southeast corner at (10°S, 60°E), with the center point positioned at (15°N, 40°E). This broad coverage area, encompassing diverse landscapes such as seas and mountains, ensures that the navigation system can operate effectively across a wide range of environments.
The DEM maps used in this project were obtained from ?. These maps were acquired at a specific point in time and represent the ground surface elevation without including buildings or other structures. The maps are subject to various sources of errors, including measurement inaccuracies, data processing artifacts, and limitations in spatial resolution. The errors in DEM maps can impact the accuracy of terrain representation and, consequently, the performance of the navigation system.

To manage the map's extensive data efficiently, we have divided the area into smaller, more manageable blocks, each measuring 1200×1200 units, referred to as `dt1` blocks. Each element in a `dt1` block represents the average height over a $100m^2$ area. This division not only facilitates faster data processing and retrieval but also allows for localized map updates and error corrections, enhancing the system's overall performance and accuracy.

Furthermore, we can achieve a higher level of detail by dividing each `dt1` block into smaller 3600×3600 tiles, called `dt2` blocks. In a `dt2` block, each element represents the average height over a $30m^2$ area, providing a more detailed representation of the terrain. This increased resolution can be particularly beneficial for precise navigation in complex environments. Figure 2 illustrates the difference in resolution between the `dt1` and `dt2`

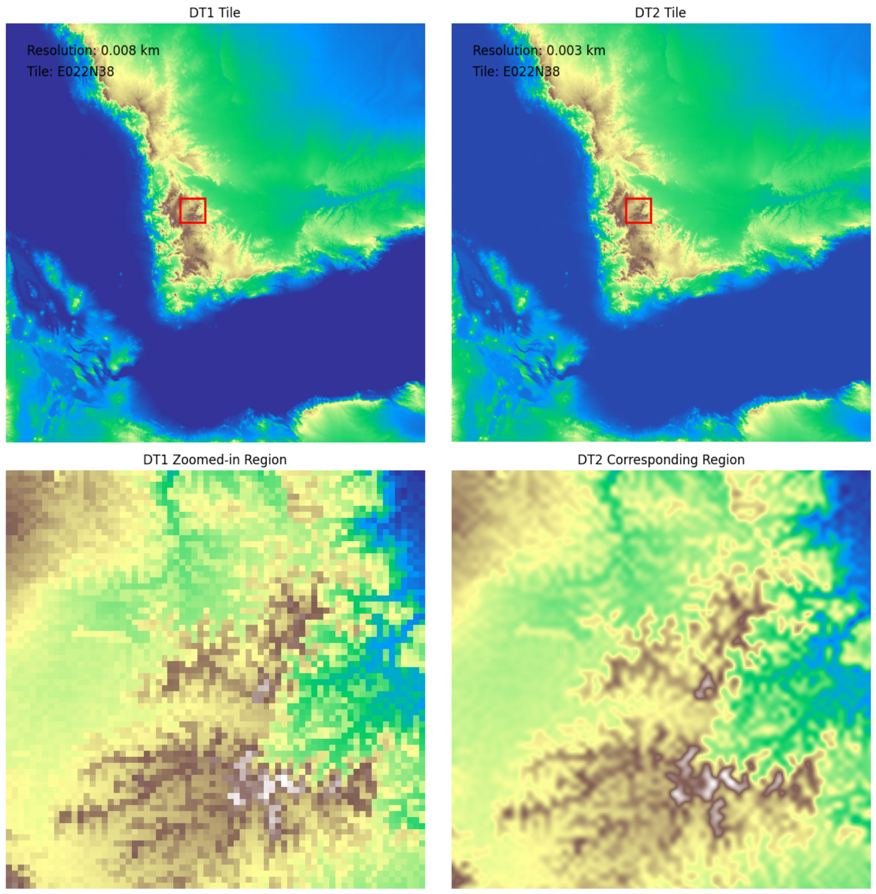

Figure 2: Resolution comparison between `dt1` and `dt2`

The choice between `dt1` and `dt2` blocks depends on the specific requirements of the navigation task. For large-scale navigation or in areas with relatively uniform terrain, the `dt1` blocks may suffice, offering a good balance between data size and representational accuracy. However, in scenarios that demand high precision or involve navigating through complex environments with intricate terrain features, the `dt2` blocks become essential. The increased resolution provided by `dt2` enables the navigation system to make more informed decisions and maintain a higher level of accuracy in its trajectory planning and localization processes.

Assumptions made about the maps in this project include linearity in the tracks over small distances. This means we assume that within each `dt1` or `dt2` block, the terrain changes are minimal enough to be approximated linearly. This simplifies the calculations and is reasonable due to the high resolution of the DEM maps used.

3 Trajectory Generation

Trajectory generation is a crucial component in simulating and analyzing vehicle motion. The process involves creating a simulated trajectory that closely resembles the vehicle's actual movement based on various initialization parameters and sensor data. These parameters include the number of data points, time vector, initial position (latitude, longitude, height), average speed, accelerations, and Euler angles (yaw, pitch, roll).

The trajectory generation process begins by determining the vehicle's orientation at each point along the trajectory using Euler angles. These angles are computed based on the initial values and angular rates, providing a representation of the vehicle's attitude throughout the motion.

To simulate the vehicle's movement, constant acceleration vectors (north, east, down) are generated for the entire trajectory. These accelerations account for any changes in velocity over time and play a role in determining the vehicle's position and velocity at each time step.

The velocity vectors are calculated by considering the initial average speed, Euler angles, and accelerations. These vectors represent the speed and direction of the vehicle at each point along the trajectory, enabling accurate simulation of the vehicle's motion.

`Figures/dummy_traj_gen_dynamic_equitions.png`

Figure 3: **?**

To incorporate the influence of terrain on the vehicle's motion, the trajectory generation process interpolates the map data at each trajectory point. This step determines the corresponding heights of the terrain at each position, ensuring that the generated trajectory accurately reflects the variations in elevation encountered by the vehicle.

Throughout the project, a key assumption is that the Euler angles and accelerations change linearly over time due to the sensor rates. This assumption simplifies the trajectory generation process and allows for efficient computation of the vehicle's orientation and motion.

The trajectory generation process is encapsulated within the `CreateTraj` class, which provides a convenient interface for generating trajectories based on the provided initial conditions and sensor data.

It is important to note that at this stage, the heights acquired during trajectory generation may not represent the actual heights below the vehicle due to its tilted orientation. The process of determining the true heights considering the vehicle's attitude will be addressed in the next section.

In addition to the standard trajectory generation methods, this project also utilizes a 6DOF drone simulation. This advanced simulation model considers the drone's movement in three-dimensional space, accounting for translational motion along the x, y, and z axes, as well as rotational motion around these axes (roll, pitch, and yaw). By incorporating 6DOF simulations, we achieve a more comprehensive and realistic representation of the drone's behavior, allowing for precise trajectory planning and analysis in complex environments.

4 Range Estimation

The pinpoint finding process, implemented in the PinPoint class, aims to determine the precise location of the vehicle at each point along the generated trajectory. This process is based on the method proposed in Oh et al. (2018) , which calculates the true range from the vehicle to the pinpoint and determines the vehicle's true coordinates .

Figure 4 illustrates the measurement model used in the pinpoint finding process. The vehicle's slant range (r_k) and look angle (θ_k) are used to determine the pinpoint location ($p_{k,alt}$) on the ground, considering the vehicle's altitude ($x_{k,alt}$). The DCM plays a crucial role in this process by converting the vehicle's local coordinate system to the global coordinate system. This conversion is essential because the vehicle's orientation affects its position relative to the Earth's surface. By applying the DCM, we can accurately calculate the delta north and delta east components of the vehicle's movement, which are necessary to determine the precise latitude and longitude of the pinpoint.

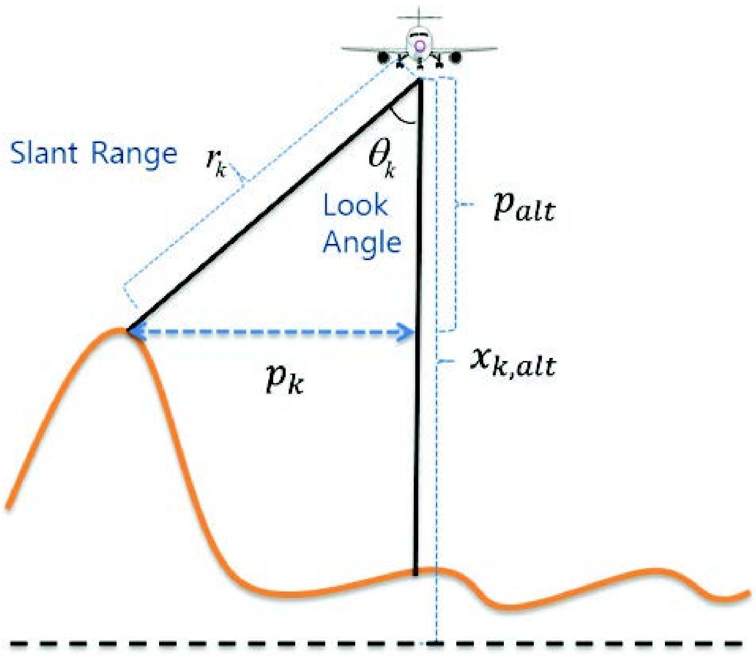

Figure 4: Measurement Model, figure taken from **?**

The pinpoint finding algorithm can be summarized in Algorithm 1:

Algorithm 1 Pinpoint Finding

```
1:  **Inputs:** map, trajectory
2:  **for** each point on the trajectory **do**
3:      $range \leftarrow$ altimeter measurement
4:      $(\text{lat}, \text{lon}, \text{Euler}) \leftarrow$ current latitude, longitude, Euler angles
5:      DCM $\leftarrow$ compute Direction Cosine Matrix
6:      $(\Delta\text{north}, \Delta\text{east}) \leftarrow \text{DCM} \cdot (\Delta x, \Delta y)$
7:      $(\Delta\text{lat}, \Delta\text{lon}) \leftarrow$ convert $(\Delta\text{north}, \Delta\text{east})$ to latitude and longitude
8:      $r_{\text{pin}} \leftarrow$ interpolate height difference along range using map data
9:      update $(\text{lat}_{\text{pin}}, \text{lon}_{\text{pin}})$ with $(\Delta\text{lat}, \Delta\text{lon})$
10:     $ground\ elevation \leftarrow$ interpolate map data at pinpoint
11: **end for**
```

By performing these steps for each point on the trajectory, the pinpoint finding process determines the true range and map height for each pinpoint. This information is crucial for accurate navigation and localization of the vehicle in an IMU-based navigation system.

The pinpoint finding process, in combination with the trajectory generation, enables precise tracking and positioning of the vehicle as it moves along the simulated path. By considering the vehicle's orientation, position, and the terrain elevation, the pinpoint finding algorithm provides a reliable estimate of the vehicle's true location at each point along the trajectory.

Figure 5 shows the results of the pinpoint finding algorithm. The plot compares the true trajectory, pinpoint results, and the measured trajectory.

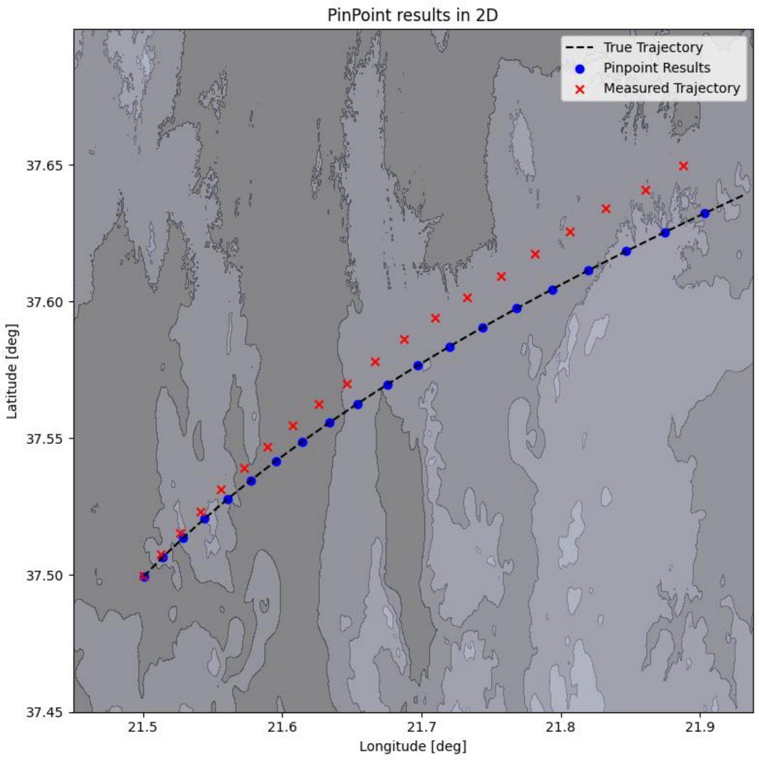

Figure 5: Results of Pinpoint Finding Algorithm

The black dashed line represents the true trajectory of the vehicle. The blue dots indicate the pinpoint results calculated by the algorithm. The red crosses show the measured trajectory. From the results, it is evident that the pinpoint algorithm closely follows the true trajectory, demonstrating its effectiveness in accurately determining the vehicle's position. The measured trajectory deviates more significantly.

5 Noising the IMU's

Introducing noise into trajectory simulations is essential for creating realistic scenarios that reflect the uncertainties and inaccuracies present in real-world sensor data. This process can be approached from two main perspectives: the "bottom-up" approach and the "top-down" approach. Both methods simulate the real-world inaccuracies and uncertainties that sensors and systems face, but they differ in their approach and the nature of the results they produce.

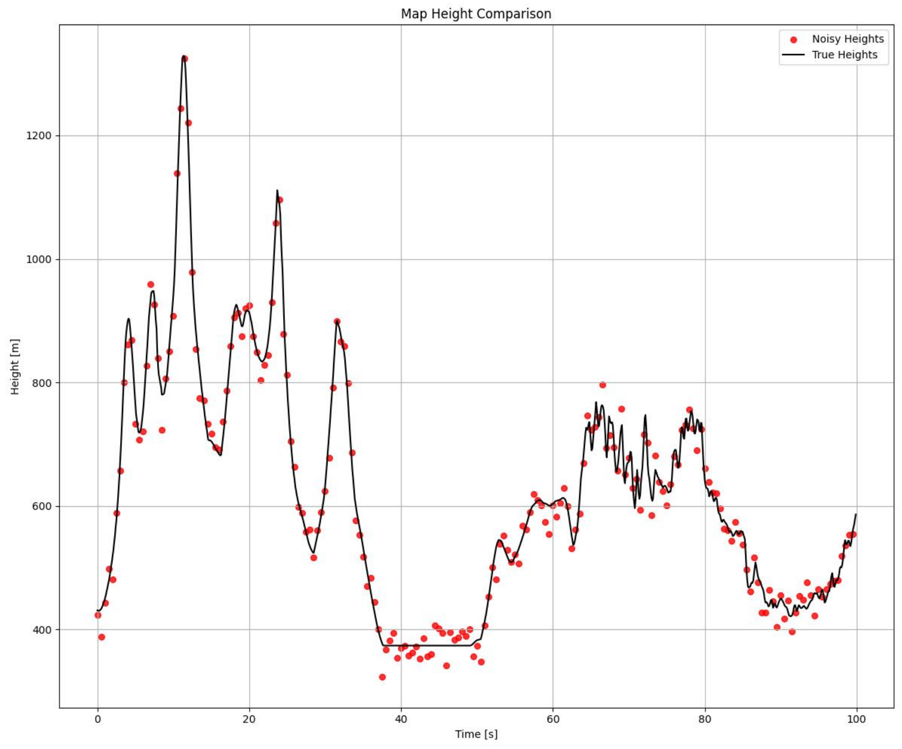

Figure 6: Example of noised acquired heights vector

5.1 Bottom-up Approach: Sensor-Level Noise Simulation

In the bottom-up approach, noise is introduced at the fundamental levels of data acquisition, mimicking the inaccuracies and uncertainties inherent in sensor measurements. This method simulates the process of collecting raw sensor data from various instruments like accelerometers, altimeters, velocity meters, gyroscopes, and barometers. Each of these sensors is subject to specific noise influences that can affect the data quality, including random noise fluctuations, systematic biases, and long-term drifts.

For instance, in a trajectory simulation, the bottom-up approach involves simulating noisy acceleration data from the accelerometer, altitude changes from the altimeter, velocity readings from the velocity meter, orientation data from the gyroscope, and atmospheric pressure measurements from the barometer. These noised measurements are then used to compute the vehicle's velocity and position over time, offering a high degree of realism by closely mirroring real-world data collection processes.

5.2 Top-Down Approach: Trajectory-Level Noise Simulation

In contrast, the top-down approach involves applying noise to the higher-level computed trajectory data, such as position, velocity, and acceleration vectors, after they have been computed from ideal or 'clean' sensor data. This method does not simulate the sensor errors directly but instead introduces variations and inaccuracies into the computed trajectory itself.

This approach is easier to implement as it does not require detailed models of sensor behavior and can be applied directly to trajectory data. It allows for straightforward control over the extent and nature of the noise introduced into the trajectory. However, the noise applied independently at each timestep or position may not fully capture the cumulative nature of error propagation in real-world systems.

5.3 Comparative Analysis and Integration of Approaches

This project explores both the bottom-up and top-down approaches to introduce noise into the IMU data for trajectory simulations. In both approaches, noise is modeled using either a uniform or normal distribution, with the normal distribution being more common:

$$\text{noise} \sim \mathcal{N}(0, \sigma) \quad \text{or} \quad \text{noise} \sim \mathcal{U}(-\sigma, \sigma)$$

Each sensor has a different value of σ in both approaches.
The bottom-up approach also incorporates sensor drift and bias. Drift represents the gradual change in sensor bias over time, calculated as the cumulative sum of drift values d_i at each timestep i: $\text{Drift}(t) = \sum_{i=1}^{t} d_i$.
Bias is a constant offset b added to the true measurement x, resulting in a biased measurement $\tilde{x}$: $\tilde{x} = x + b$.

The noisy measurement $\hat{x}$ at timestep t in comparsion is given by:

$$\begin{aligned} \text{Bottom - Up:} \quad & \hat{x}(t) = \text{noise}(x(t) + \text{Drift}(t) + b) \\ \text{Top - Down:} \quad & \hat{x}(t) = \text{noise}(x(t)) \end{aligned}$$

where $x(t)$ is the true measurement, noise(t) is the random noise, Drift(t) is the cumulative drift, and b is the bias.

Including noise, drift, and bias in the bottom-up approach provides a more realistic representation of sensor behavior and uncertainties in the trajectory simulation and analysis process. as can be seen in fig. 7

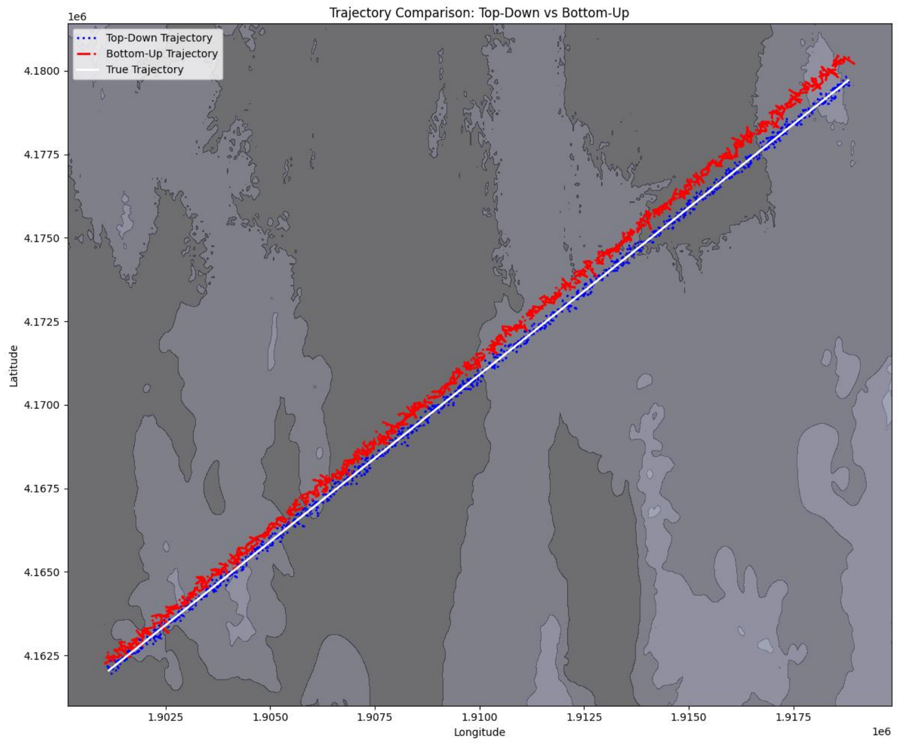

Figure 7: Noised Trajectory Comparison with both methods

5.4 Comparative Table

Aspect	**Bottom-up Approach**	**Top-down Approach**
Noise Source	Sensor-level (raw data)	Computed trajectory data
Implementation	Complex, requires detailed sensor models	Simple, no sensor models needed
Realism	High, closely mimics real-world sensor behavior	Moderate, cumulative error may not be fully captured
Flexibility	Lower, tied to sensor characteristics	Higher, easy to manipulate noise levels

Table 1: Comparison of Bottom-up and Top-down Approaches

By thoroughly analyzing these two approaches, we aim to develop a robust methodology for introducing and handling noise in IMU data, enhancing the fidelity and reliability of trajectory simulations.

6 Estimators

Estimators play a crucial role in IMU-based navigation systems by fusing sensor measurements and prior knowledge to estimate the vehicle's state, including position, velocity, and orientation. In this project, we explore two advanced estimators: the Iterated Extended Kalman Filter (`IEKF`) and the Unscented Kalman Filter (`UKF`). These estimators are designed to handle nonlinear systems and provide accurate state estimates in the presence of noise and uncertainties.

Kalman Filters are recursive algorithms that estimate the state of a dynamic system from a series of noisy measurements. They operate by propagating the state estimate and its uncertainty through a prediction step based on a system model, and then updating the estimate using sensor measurements in a correction step.

The `IEKF` and `UKF` are both extensions of the standard Kalman Filter that address the limitations of the filter when dealing with highly nonlinear systems.

6.1 Iterated Extended Kalman Filter (IEKF)

The Iterated Extended Kalman Filter (IEKF) is an extension of the Extended Kalman Filter (EKF), proposed in Julier et al. (1995), that aims to improve the accuracy of state estimation in nonlinear systems. The EKF linearizes the nonlinear system model around the current state estimate using a first-order Taylor series expansion. However, this linearization can introduce errors, especially when the system is highly nonlinear or the initial state estimate is far from the true state. The IEKF addresses this limitation by iteratively refining the state estimate within each measurement update step.

6.1.1 Overview of the IEKF Algorithm

The IEKF algorithm consists of two main steps: prediction and update. The prediction step is similar to the standard EKF, where the state estimate and covariance are propagated forward in time using the system model. The update step, however, is modified to include an iterative refinement process.

Algorithm 2 presents the pseudocode for the IEKF algorithm, highlighting the prediction and update steps, as well as the iterative refinement process within the update step.

Figure 8 illustrates the block diagram of the IEKF, highlighting the iterative update step.

6.1.2 Prediction Step

In the prediction step, the IEKF estimates the state at the current time step based on the previous state estimate and the system model. The predicted state estimate $\hat{x}_{k|k-1}$ and predicted covariance $P_{k|k-1}$ are computed as follows:

Algorithm 2 Iterated Extended Kalman Filter (IEKF)

```
1: **Initialization:**
2: Initialize state estimate $\hat{x}_0$ and covariance $P_0$
3: Set measurement vector $z$ and control input $u$
4: Define process noise covariance $Q$ and measurement noise covariance $R$
5: for $k = 1$ to $K$ do                                        ▷ Main IEKF loop over time steps
6:     **Prediction Step:**
7:     Predict state: $\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_k)$
8:     Linearize process model: $F_k = \frac{\partial f}{\partial x}\Big|_{\hat{x}_{k-1|k-1}, u_k}$
9:     Predict covariance: $P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k$
10:    **Update Step:**                                         ▷ Iterative measurement update
11:    for $iter = 1$ to $MaxIter$ or until convergence do
12:        Compute innovation: $y_k = z_k - h(\hat{x}k|k-1)$
13:        Linearize measurement model: $H_k = \frac{\partial h}{\partial x}\Big|\hat{x}_{k|k-1}$
14:        Compute Kalman gain: $K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1}$
15:        Update state estimate: $\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k y_k$
16:        Update covariance: $P_{k|k} = (I - K_k H_k) P_{k|k-1}$
17:        if convergence criteria met then
18:            break                                            ▷ Exit inner loop if converged
19:        end if
20:    end for
21:    Store estimates and proceed to next time step
22: end for
23: return Final state estimates and covariances
```

$$\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_k) \tag{1}$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k \tag{2}$$

where $f(\cdot)$ is the nonlinear system model, u_k is the control input, F_k is the Jacobian matrix of the system model evaluated at the previous state estimate, and Q_k is the process noise covariance.

6.1.3 Iterative Update Step

The key difference between the IEKF and the standard EKF lies in the update step. In the IEKF, the update step is performed iteratively to refine the state estimate. The iterations aim to minimize the linearization errors introduced by the first-order Taylor series approximation.

The iterative update step consists of the following equations:

$$y_k = z_k - h(\hat{x}_{k|k-1}) \tag{3}$$

$$H_k = \frac{\partial h}{\partial x}\Big|\hat{x}_{k|k-1} \tag{4}$$

$$K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1} \tag{5}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k y_k \tag{6}$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} \tag{7}$$

where y_k is the measurement innovation, z_k is the measurement vector, $h(\cdot)$ is the nonlinear measurement model, H_k is the Jacobian matrix of the measurement model evaluated at the predicted state estimate, K_k is the Kalman gain, and R_k is the measurement noise covariance.

The iterative update step is repeated for a fixed number of iterations or until a convergence criterion is met. During each iteration, the state estimate $\hat{x}_{k|k}$ is updated using the Kalman gain and the measurement innovation. The covariance matrix $P_{k|k}$ is also updated accordingly.

6.1.4 Why Iterative Updates Improve Accuracy?

The iterative update step in the IEKF is designed to improve the accuracy of the state estimate by reducing the linearization errors. By iteratively refining the state estimate, the IEKF allows the linearization point to move closer to the true state, providing a better approximation of the nonlinear system.

The first-order Taylor series approximation used in the standard EKF can introduce significant errors when the system is highly nonlinear or the initial state estimate is far from the true state. The IEKF addresses this issue by iteratively updating the state estimate, effectively ”re-linearizing” the system around the updated estimate. This process helps to capture the nonlinearities more accurately and reduces the impact of linearization errors.

The iterative updates in the IEKF can be seen as a form of Gauss-Newton optimization Bell and Cathey (1993), where the goal is to find the state estimate that minimizes the difference between the predicted measurement and the actual measurement. By iteratively refining the state estimate, the IEKF converges towards a more accurate solution, improving the overall estimation performance.

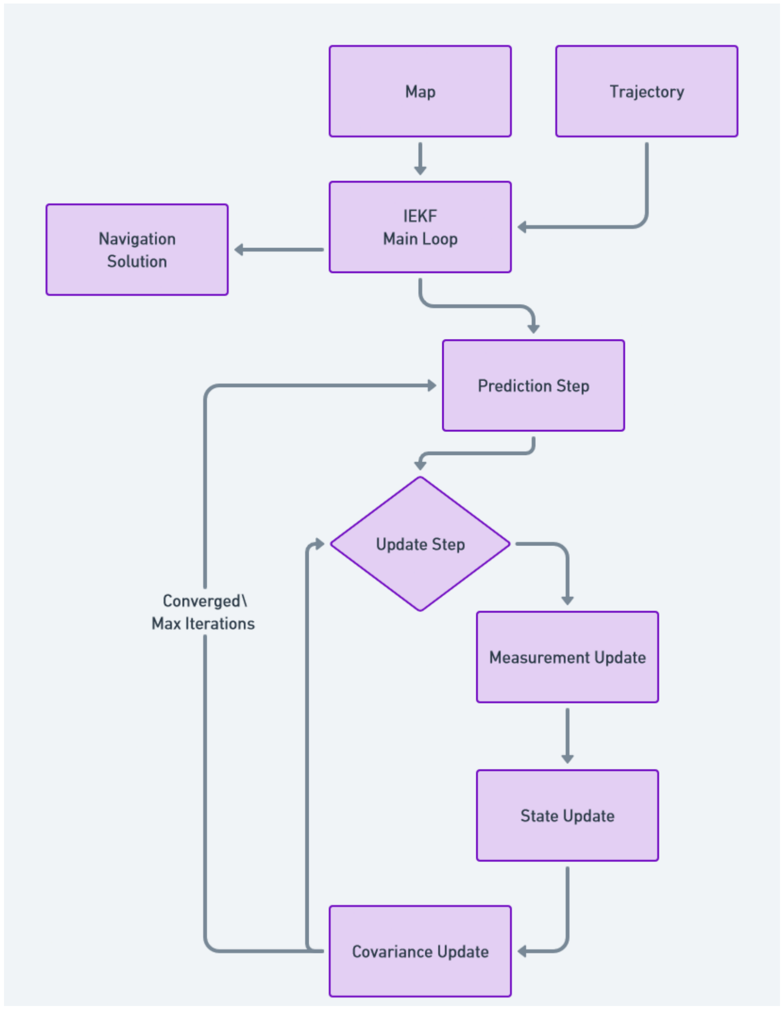

Figure 8: Iterated Extended Kalman Filter (IEKF) block diagram

6.2 Acquiring Measurement Noise with Terrain Gradients

In the implementation of the IEKF, accurate estimation of the measurement noise covariance R_k is crucial for optimal performance. The measurement noise is influenced by the terrain's gradients (slopes) since the vehicle's altitude measurements are affected by the underlying terrain features.

To calculate the terrain slope estimation at each time step k, we estimate the north-south (S_N) and east-west (S_E) slopes of the terrain at the vehicle's current position (ϕ_k, λ_k), where ϕ denotes latitude and λ denotes longitude. The slopes are estimated using a local window around the current position, based on the DEM.

We define a grid of terrain elevation points around the current position:

$$\phi_i = \phi_k + i \cdot \Delta\phi, \quad i = -N, \ldots, N \ \lambda_j \quad = \lambda_k + j \cdot \Delta\lambda, \quad j = -N, \ldots, N \tag{8}$$

where $\Delta\phi$ and $\Delta\lambda$ are increments in latitude and longitude corresponding to a fixed ground distance d_P, and N determines the size of the grid.

We interpolate the terrain elevations h_{ij} at each grid point (ϕ_i, λ_j) from the DEM, and compute the reference elevation h_0 at the current position (ϕ_k, λ_k).

The slopes S_N and S_E are computed using the least squares estimation:

$$S_N = \frac{\sum_{i=-N}^{N} \sum_{j=-N}^{N} (\phi_i - \phi_k)(h_{ij} - h_0)}{\sum_{i=-N}^{N} (\phi_i - \phi_k)^2}, \quad S_E = \frac{\sum_{i=-N}^{N} \sum_{j=-N}^{N} (\lambda_j - \lambda_k)(h_{ij} - h_0)}{\sum_{j=-N}^{N} (\lambda_j - \lambda_k)^2} \tag{9}$$

These slopes represent the rate of change of elevation with respect to latitude and longitude, respectively.

6.2.1 Estimation of Measurement Noise Covariance

The measurement noise covariance R_k at time step k is composed of two components:

$$R_k = R_c + R_{\text{fit},k} \tag{10}$$

where R_c is a constant term representing the measurement noise inherent to the sensor (e.g., altimeter noise), and $R_{\text{fit},k}$ is the error due to the terrain slope estimation.

To compute $R_{\text{fit},k}$, we calculate the residual error between the actual terrain elevations and the elevations predicted by the slope model:

$$R_{\text{fit},k} = \frac{1}{M-1} \sum_{i=-N}^{N} \sum_{j=-N}^{N} \left(h_{ij} - h_0 - S_N(\phi_i - \phi_k) - S_E(\lambda_j - \lambda_k)\right)^2 \tag{11}$$

where $M = (2N+1)^2$ is the total number of grid points.

This residual represents the variance of the elevation differences after removing the linear slope component, effectively quantifying how well the linear slope model fits the actual terrain in the vicinity of the current position.

6.3 Unscented Kalman Filter (UKF)

The Unscented Kalman Filter (UKF), proposed in Wan and Van Der Merwe (2000) and Julier and Uhlmann (1997)is another advanced state estimation technique that addresses the limitations of the Extended Kalman Filter (EKF) in dealing with highly nonlinear systems. While the EKF relies on linearization using first-order Taylor series approximation, the UKF uses a deterministic sampling approach called the Unscented Transform (`UT`) to capture the mean and covariance of the state distribution more accurately.

6.3.1 Overview of the UKF Algorithm

The UKF algorithm consists of three main steps: initialization, prediction, and update. The initialization step involves setting the initial state estimate and covariance. The prediction step propagates the state estimate and covariance through the nonlinear system model using the Unscented Transform. The update step incorporates the measurements to refine the state estimate and covariance.

Algorithm 3 presents the pseudocode for the UKF algorithm, highlighting the key steps involved.

Algorithm 3 Unscented Kalman Filter (UKF) for Trajectory Estimation

```
1: Initialization:
2: Initialize state estimate $\hat{x}_0$ and covariance $P_0$
3: Choose scaling parameters $\lambda$, $\alpha$, $\beta$, and $\kappa$ for sigma point generation
4: Set measurement vector $z$ and control input $u$
5: Define process noise covariance $Q$ and measurement noise covariance $R$
6: for $k = 1$ to $K$ do                                        ▷ Main UKF loop over time steps
7:     Sigma Point Generation:
8:     Compute square root of $(\lambda + n_x)P_{k-1}$ using Cholesky decomposition
9:     Generate sigma points around $\hat{x}_{k-1}$
10:    Prediction Step:
11:    Propagate each sigma point through state transition function $f(\cdot)$
12:    Compute predicted state $\hat{x}_{k|k-1}$ by weighted average of sigma points
13:    Compute predicted covariance $P_{k|k-1}$ using weighted covariance of sigma points and process noise $Q$
14:    Update Step:
15:    Propagate predicted sigma points through measurement function $h(\cdot)$
16:    Compute predicted measurement $\hat{z}_k$ by weighted average of sigma points in measurement space
17:    Compute innovation covariance $S_k$ and cross-covariance $C_k$
18:    Compute Kalman gain $K_k = C_k S_k^{-1}$
19:    Update state estimate $\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(y_k - \hat{z}_k)$
20:    Update covariance $P_{k|k} = P_{k|k-1} - K_k S_k K_k^T$
21: end for
22: return Final state estimates and covariances
```

6.3.2 Sigma Point Generation

One of the key concepts in the UKF is the generation of sigma points. Sigma points are a set of deterministically chosen sample points that capture the mean and covariance of the state distribution. The sigma points are generated using the Unscented Transform, which involves the following steps:

1. Choose scaling parameters λ, α, β, and κ based on the desired properties of the sigma points.
2. Compute the square root of the scaled covariance matrix using Cholesky decomposition.
3. Generate sigma points by adding and subtracting the scaled square root of the covariance matrix from the state estimate.

The sigma points are then propagated through the nonlinear system model and measurement model to compute the predicted state and measurement estimates.

6.3.3 Prediction Step

In the prediction step, the UKF propagates the sigma points through the nonlinear state transition function $f(\cdot)$. The predicted state estimate $\hat{x}_{k|k-1}$ is computed as the weighted average of the propagated sigma points. The predicted covariance $P_{k|k-1}$ is computed using the weighted covariance of the propagated sigma points and the process noise covariance Q.

6.3.4 Update Step

In the update step, the UKF incorporates the measurements to refine the state estimate and covariance. The predicted sigma points are propagated through the nonlinear measurement function $h(\cdot)$ to obtain the predicted measurements. The predicted measurement $\hat{z}_k$ is computed as the weighted average of the propagated sigma points in the measurement space.

The innovation covariance S_k and cross-covariance C_k are computed using the weighted covariance of the predicted measurements and the measurement noise covariance R. The Kalman gain K_k is then computed using the innovation covariance and cross-covariance.

Finally, the state estimate $\hat{x}_{k|k}$ is updated by adding the product of the Kalman gain and the measurement innovation $(y_k - \hat{z}_k)$ to the predicted state estimate. The covariance $P_{k|k}$ is updated using the Kalman gain and the innovation covariance.

7 Results

Figures/dummy_map_elevation.png

Figure 9: Map Elevation at Trajectory Points

7.1 Trajectory Estimation Results

Figure 9 shows the map elevation along the trajectory points. The impact of terrain variations on the estimation accuracy is discussed. The filter's ability to estimate the terrain elevation compared to the ground truth is analyzed, highlighting any challenges or successes in handling terrain information.

7.1.1 Map and Trajectory Visualization

Figure 10 shows the estimated trajectory compared to the ground truth and measured positions on the map. The plot provides a visual representation of the filter's performance in terms of trajectory tracking.

Figures/dummy_map_trajectory.png

Figure 10: Map and Estimated Trajectory

7.1.2 Position Estimation Performance

The position estimation errors in the North and East directions are shown in Figure 11. The plot compares the errors with the filter's covariance estimates, providing insights into the filter's uncertainty estimation. The RMSE, maximum absolute error, and error bound percentage for position are shown, considering the results available in the plots. The performance is compared across different noise types to evaluate the filter's robustness.

Figures/dummy_position_errors.png

Figure 11: Position Estimation Errors

7.1.3 Velocity Estimation Performance

Figure 12 presents the velocity estimation errors in the North, East, and Down directions. Similar to the position errors, the velocity errors are analyzed in comparison with the filter's covariance estimates. The RMSE, maximum absolute error, and error bound percentage for velocity are discussed based on the results available in the plots. The performance across different noise types is compared to assess the filter's effectiveness in estimating velocities.

`Figures/dummy_velocity_errors.png`

Figure 12: Velocity Estimation Errors

7.1.4 Altitude Estimation Performance

The altitude estimation errors are depicted in Figure 13. The plot shows the errors in comparison with the filter's covariance estimates. The RMSE, maximum absolute error, and error bound percentage for altitude are analyzed based on the results available in the plots. The performance under different noise conditions is compared to evaluate the filter's accuracy in estimating altitude.

`Figures/dummy_altitude_errors.png`

Figure 13: Altitude Estimation Errors

7.1.5 Attitude Estimation Performance

Figure 14 shows the attitude estimation errors in terms of Euler angles (yaw, pitch, roll). The errors are compared with the filter's covariance estimates to assess the filter's uncertainty estimation. The RMSE, maximum absolute error, and error bound percentage for attitude are discussed based on the results available in the plots. The performance across different noise types is analyzed to evaluate the filter's robustness in estimating attitude.

`Figures/dummy_attitude_errors.png`

Figure 14: Attitude Estimation Errors

7.2 Filter Analysis

7.2.1 Model Errors and Corrections

Figure 15 illustrates the model errors and corrections applied by the filter. The plot shows the measurement mismatch, error correction, process noise, and model fit. An analysis of how the filter adapts to model errors and uncertainties is provided, highlighting the filter's capability to handle model discrepancies.

`Figures/dummy_model_errors.png`

Figure 15: Model Errors and Corrections

7.2.2 Kalman Gains

The behavior of the Kalman gains over time for different state variables is shown in Figure 16. The analysis of the Kalman gains provides insights into how the filter balances the trust in the measurements versus the model predictions. The variations in the gains reflect the filter's adaptation to the changing conditions and uncertainties.

Figures/dummy_kalman_gains.png

Figure 16: Kalman Gains

8 Conclusions

8.1 Results Discussion, Comparison of IEKF and UKF

8.2 Future Works

References

B. Bell and F. Cathey. The iterated kalman filter update as a gauss-newton method. *IEEE Transactions on Automatic Control*, 38(2):294–297, 1993. doi: 10.1109/9.250476.

S. Julier, J. Uhlmann, and H. Durrant-Whyte. A new approach for filtering nonlinear systems. 3:1628–1632 vol.3, 1995. doi: 10.1109/ACC.1995.529783. Link.

S. J. Julier and J. K. Uhlmann. New extension of the kalman filter to nonlinear systems. 3068:182–193, 1997. Link.

J. Oh, C.-K. Sung, J.-S. Lee, and M.-J. Yu. A new method to calculate relative distance of closest terrain point using interferometric radar altimeter output in real flight environment. *2018 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, 2018. Link.

E. Wan and R. Van Der Merwe. The unscented kalman filter for nonlinear estimation. pages 153–158, 2000. doi: 10.1109/ASSPCC.2000.882463. Link.