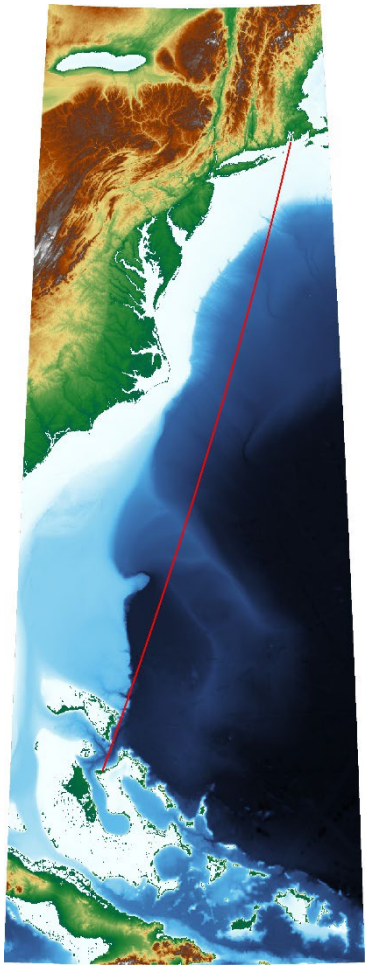
INS/GNSS Integration

Project #5



Introduction

Well, it's been a long semester and we've come a long way. We're all due for a vacation, so I've booked us all a flight to the Bahamas. Unfortunately, in order to afford tickets for the whole class I had to go with some budget airline, and the navigation system broke, and now they need us to make some repairs.

In this project, you will implement a INS/GNSS integration scheme using a nonlinear Kalman Filter of your choice (EKF or UKF). We will use the same state model as previously discussed in the previous Nonlinear Kalman Filter project. However, since we are now dealing with global scale navigation, the base units will be different. We will be using a Local Navigation Frame (either North-East-Down, or East-North-Up, your choice but both are functionally equivalent) and so our base position units will be (using North-East-Down):

$$\mathbf{p} = \begin{bmatrix} \text{decimal degrees latitude} \\ \text{decimal degrees longitude} \\ \text{meters below mean sea level} \end{bmatrix}$$

Similarly, the attitude is given by the relative orientation of the vehicle to the Local Navigation Frame in Euler angles with the rotation in that order (roll, pitch, yaw).

$$\mathbf{q} = \begin{bmatrix} \text{roll} \\ \text{pitch} \\ \text{yaw} \end{bmatrix} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}$$

Velocities are given in meters per second.

You will implement two architectures: a feed-forward and a feed-back INS. The feed-back architecture is essentially the same as the previous nonlinear Kalman filter in Project 3: the IMU biases will be included in the state vector, and you will report a position. The feed forward model will use an error-state vector: $\mathbf{x} = [\mathbf{p} \quad \mathbf{q} \quad \dot{\mathbf{p}} \quad \mathbf{e}]$ where $\mathbf{e}$ is the error correction to the base position states $\mathbf{p}$. The reported position should therefore be $\mathbf{p} - \mathbf{e}$.

For this project you will need to use the haversine formula to calculate the position error. There is an open source Python package that you can use, or you may implement this method yourself.

Input Data

The input data for this is simple. There is one file for this dataset: `trajectory_data.cvs`. This file contains the time-indexed (seconds) navigation information. We have the true latitude, longitude, and altitude (meters above sea level) and the true attitude/orientation in the form of the Euler roll-pitch-yaw angles. Additionally we have the IMU output (3-axis specific forces along the body frame x-, y-, and z-

axis, and the same 3-axis angular rates along roll, pitch, and yaw). Finally we have measurements of both position and velocity from the onboard GNSS receives (prefaced with `z_`). Additionally, WGS84 parameters (curvature, Somigliana gravity model, and rotation rate) are provided in `earth.py`.

Propagation model

The propagation (integrator) model for the IMU output is reiterated below. Remember, that it must be done in the three distinct steps in order (attitude, velocity, position) and that you should maintain both the prior $(t-1)$ and the posterior (t) state belief throughout the calculation and return only the posterior. It is also important to note that Groves maintains the attitude in the form of a complete rotation matrix. For our purposes (using a full state nonlinear Kalman filter), we must decompose the rotation matrix into its component angles. I suggest looking at the SciPy `Rotation` module in `spatial.transform`. I would suggest keeping it simple and using the functions `from_euler` and `as_euler` to go from Euler angles to rotation matrix and from rotation matrix to Euler angles.

Attitude Update

Given the previous state x_{t-1} and the gyroscope measurements $\omega_i^b = [\omega_{roll} \quad \omega_{pitch} \quad \omega_{yaw}]^T$, if you are using a feedback filter and modeling the IMU biases first subtract out the bias from the measured value ($\bar{\omega} = \omega_i^b - \mathbf{b}_g$, relace ω_i^b with $\bar{\omega}$ below), then proceed:

$$\omega_E = 7.2921157(10^{-5})\ \frac{\text{rad}}{\text{s}} \text{ (Earth's rate of rotation)}$$

$$\Omega_i^e = \begin{bmatrix} 0 & -\omega_E & 0 \\ \omega_E & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\omega_e^n = \begin{bmatrix} \dfrac{v_E}{R_E(L)+h} \\ -\dfrac{v_N}{R_N(L)+h} \\ -\dfrac{v_E \tan L}{R_E(L)+h} \end{bmatrix}$$

$$\Omega_e^n = [\omega_e^n \quad \wedge]$$

$$\Omega_i^b = [\omega_i^b \quad \wedge]$$

$$R_{b,\,t}^n \approx R_{b,\,t-1}^n \left(I_3 + \Omega_i^b \delta t\right) - (\Omega_i^e + \Omega_e^n) R_{b,t-1}^n \delta t$$

Velocity Update

Given the previous state, the prior and posterior rotation matrices, and the accelerometer input f_b, again if you are using a feedback filter and modeling the biases, first subtract out the bias from the measured value ($\bar{f} = f_b - \mathbf{b}_a$, replace f_b with $\bar{f}$ below):

$$f_{n,t} \approx \frac{1}{2}\left(R_{b,\,t-1}^n + R_{b,t}^n\right) f_{b,t}$$

$$\mathbf{v}_{n,t} = \mathbf{v}_{n,t-1} + \delta t \left(f_{n,t} + \mathrm{g}(L_{t-1}, h_{t-1}) - \left(\Omega_{e,t-1}^n + 2\Omega_{i,t-1}^e\right)\mathbf{v}_{n,t-1}\right)$$

Where $g(L_{t-1}, h_{t-1})$ is the Somigliana gravity model provided by the `gravity()` function in `earth.py`.

Position Update

Given the previous state and the posterior velocities $\mathbf{v}_{n,t} = [v_{N,t} \quad v_{E,t} \quad v_{D,t}]^T$ update the height, latitude, and longitude:

$$h_t = h_{t-1} - \frac{\delta t}{2}(v_{D,t-1} + v_{D,t})$$

$$L_t = L_{t-1} + \frac{\delta t}{2}\left(\frac{v_{N,t-1}}{R_N(L_{t-1}) + h_{t-1}} + \frac{v_{N,t}}{R_N(L_{t-1}) + h_t}\right)$$

$$\lambda_t = \lambda_{t-1} + \frac{\delta t}{2}\left(\frac{v_{E,t-1}}{(R_E(L_{t-1}) + h_{t-1})\cos L_{t-1}} + \frac{v_{E,t}}{(R_E(L_t) + h_t)\cos L_t}\right)$$

And un-skew the rotation matrix to get the Euler angles: $\mathbf{q}_t = [\phi \quad \theta \quad \psi]^T$ and return the updated belief:

$$\bar{x}_t = \begin{bmatrix} L_t \\ \lambda_t \\ h_t \\ \mathbf{q}_t \\ \mathbf{v}_{n,t} \end{bmatrix}$$

Be sure to include the appropriate error state or bias states as well.

Some suggestions

The above propagation model is referred to in some literature (namely Groves) as an "integrator" as it performs one-step integration of the navigation states. This integration will be the same across the nonlinear error state and full state implementations. You may want to compartmentalize it into its own function.

Historically, in error-state Kalman filter INSs, this integrator would be run in parallel to an error-state Kalman filter. The linear error-state Kalman filter would then estimate the error states directly (because the error states are linearly dependent, and they didn't have nonlinear variants at the time). The corrected navigation solution would then be the integrator output minus the error estimates from the Kalman Filter. The point of this project is to replicate that process using a nonlinear filter to estimate the error corrections needed for the nonlinear integration and to estimate the navigation solution (full navigation state with biases) directly.

I would suggest that you follow a somewhat similar architecture. Contain the propagation function in its own method/function and test and verify that it works. This is a relatively good IMU so the positions should track with lower error for some time. Then incorporate the integrator into each Kalman filter propagation step where the additional Kalman filter states (error correction or IMU biases) are added in and estimated. Do not feel obligated to follow this suggestion if it does not make sense to you. It is just a suggested approach, and you are free to implement the project however you see fit.

Task 1: Observation Model

For each trajectory you have measurements of both position and velocity. The measurements provided are position (latitude, longitude, altitude) and velocities (northward, eastward, and downward). Derive the observation model for use in your nonlinear Kalman Filter for both the feed-forward (error correction) and feed-back (bias modeling) versions.

Task 2: Nonlinear Error State implementation

Implement a nonlinear Kalman filter using the nonlinear error state model:

$$\mathbf{x} = [L, \lambda, h, \phi, \theta, \psi, v_N, v_E, v_D, e_L, e_\lambda, e_h]$$

Task 3: Nonlinear Full State implementation

Implement a nonlinear Kalman filter using the nonlinear bias state model:

$$\mathbf{x} = \left[L, \lambda, h, \phi, \theta, \psi, v_N, v_E, v_D, b_{a_x}, b_{a_y}, b_{a_z}, b_{g_x}, b_{g_y}, b_{g_z}\right]$$

Task 4: Discussion and performance analysis

Plot the overall position (latitude and longitude) as well as the errors for each state and the Haversine distance between the estimated position and the true (GNSS measured) position.

Please discuss and evaluate the accuracy and precision of each implementation for each dataset. Discuss how you developed and tuned the process noise for each implementation. Compare and contrast each implementation based on the results you found. Based on that analysis what implications does using such a GNSS based INS system have if any?

Finally, given that the GNSS measurements are themselves a noisy estimate of the true position state, how can we potentially improve upon this system? Is a basic haversine distance between INS estimate and GNSS estimate a fair error comparison?

Grading Rubric

Task 1: Observation Model	**Excellent – 3 pts** The student correctly implemented the observation models and explained their implementation.	**Acceptable – 2 pts** The student implemented both observation models.	**Marginal – 1 pts** The student implemented only one observation model.	**Unacceptable – 0 pts** The student did not attempt to implement the observation model and relied.
Task 2: Error state implementation	**Excellent – 5 pts** The student correctly implements the error state filter and explains their implementation.	**Acceptable – 2 pts** The student correctly implements the error state filter.	**Marginal – 1 pts** The student implemented the error state filter with some minor errors	**Unacceptable – 0 pts** The student did not attempt to implement the error state filter.

Task 3: Full state implementation	**Excellent – 5 pts** The student correctly implements the full state filter and explains their implementation.	**Acceptable – 2 pts** The student correctly implements the full state filter.	**Marginal – 1 pts** The student implemented the full state filter with some minor errors	**Unacceptable – 0 pts** The student did not attempt to implement the error state filter.
Task 4: Discussion	**Excellent – 6 pts** The student presents their results and offers a detail commentary on the suggested questions and provides some additional analysis or reflection or each method's performance, ease of implementation, or use case.	**Acceptable – 4 pts** The student presents their results and offers a simple commentary on the suggested questions.	**Marginal – 2 pts** The student presents their results and offers some sort of explanation or commentary on their performance	**Unacceptable – 0 pts** The student simply presents their results
Code and Report Quality	**Excellent – 6 pts** The student has intuitive, concise, well written code that follows a logical and organized structure. Comments are provided and document the functionality of the code. The student's report provides a very detailed description of their work. It addresses the questions and deliverables posed by the assignment and provides some discussion as to their results.	**Acceptable – 4 pts** The student's code makes sense given the context of the problem. Some comments or documentation is provided. The student's report provides a description of their work and addresses the questions and deliverables posed by the assignment.	**Marginal – 2 pts** The student's code is poorly structured and not intuitive. Little to no comments or documentation is provided. The student's report provides little to no description of their work or fails to address the questions and deliverables of the assignment beyond a basic answer.	**Unacceptable – 0 pts** The student's code fails to run, runs incorrectly, or otherwise fails to address the problem. The student's report fails to adequately describe their work or address the questions and deliverables of the assignment.