Project 6 - INS/GNSS Integration

Keith Chester

April 29, 2024

In this assignment we are looking at integrating our previous UKF implementation with an INS/GNSS model. Specifically, we are looking at two possible approaches - a Feedback (FB) model, and a Feed Forward (FF) model. The primary difference is within the state, wherein our FB model state is:

$$x = \begin{bmatrix} L \\ \lambda \\ h \\ \phi \\ \theta \\ \psi \\ V_N \\ V_E \\ V_D \\ e_L \\ e_\lambda \\ e_h \end{bmatrix} \tag{1}$$

...wherein our FF model state is:

$$x = \begin{bmatrix} L \\ \lambda \\ h \\ \phi \\ \theta \\ \psi \\ V_N \\ V_E \\ V_D \\ b_{a_x} \\ b_{a_y} \\ b_{a_z} \\ b_{g_x} \\ b_{g_y} \\ b_{g_z} \end{bmatrix} \tag{2}$$

...wherein our biases are calculated as the linear difference between latitude and longitude and the estimated state.

Propagation Model

In this UKF model, we follow the propagation model of applying an attitude update, a velocity update, and finally a position update.

Attitude Update

Our attitude update starts with ω_E, which is defined as the rotation of the the earth in radians a second - $7.292 \times 10^{-5} \frac{rad}{s}$. We form a screw rotation matrix Ω_e^i as:

$$\Omega_e^i = \begin{bmatrix} 0 & -\omega_E & 0 \\ \omega_E & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3}$$

We then calculate ω_e^n as:

$$\omega_e^n = \begin{bmatrix} \frac{v_e}{R_e(L)+h} \\ -\frac{v_n}{R_n(L)+h} \\ -\frac{v_E tan(L)}{R_e(L)+h} \end{bmatrix} \tag{4}$$

where:

$$v_e = R_i^e(v_i - \Omega_i r_i) \tag{5}$$

$$R_i^e = \begin{bmatrix} cos\omega_t & sin\omega_t & 0 \\ -sin\omega_t & cos\omega_t & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6}$$

$$R_E(L) = \frac{R_0}{\sqrt{1 - e^2 sin^2(L)}} \tag{7}$$

...and our state vars are essentially following the equation from our model of our aircraft for a curvilinear craft:

$$L = atan(\frac{z_e(R_e(L)+h)}{(1-e^2)(R_e(L)+h) - \sqrt{x_e^2 + y_e^2}}) \tag{8}$$

$$\lambda = atan(\frac{y_e}{x_e}) \tag{9}$$

$$h = \frac{1}{cos(L)}\sqrt{x_e^2 + y_e^2} - R_e(L) \tag{10}$$

Now that we have ω_e^n, we create Ω_e^n and Ω_i^b following the screw form, but also find $R_{b,t}^n$ as:

$$R_{b,t}^n \approx R_{b,t-1}^n(I_3 + \Omega_i^b \delta t) - (\Omega_i^e + \Omega_e^n)R_{b,t-1}^n \delta t \tag{11}$$

Velocity Update

We can now begin our velocity update. We first calculate our resulting rotational velocity:

$$f_{n,t} \approx \frac{1}{2}(R_{b,t-1}^n + R_{b,t}^n) f_{b,t} \tag{12}$$

$$v_{n,t} = v_{n,t-1} + \delta t(f_{n,t} + g(L_{t-1}, h_{t-1}) - (\Omega_{e,t-1}^n + 2\Omega_{i,t-1}^e)v_{n,t-1}) \tag{13}$$

wherein $g(L,h)$ is the Somigliana gravity model.

Position Update

Finally, we can calculate our position update utilizing the resulting velocities we calculated prior.

$$h_t = h_{t-1} + \frac{\delta t}{2}(v_{D,t-1} + v_{D,t}) \tag{14}$$

$$L_t = L_{t-1} + \frac{\delta t}{2}(\frac{v_{N,t-1}}{R_e(L_{t-1}) + h_{t-1}} + \frac{v_{N,t}}{R_e(L_{t-1}) + h_t}) \tag{15}$$

$$\lambda_t = \lambda_{t-1} + \frac{\delta t}{2}(\frac{v_{E,t-1}}{(R_E(L_{t-1}) + h_{t-1})cosL_{t-1}} + \frac{v_{E,t}}{(R_E(L_{t-1}) + h_t)cosL_t}) \tag{16}$$

Code

The following python code implements the above code, wherein

```python
def propagation_model(
self, state: np.ndarray, delta_t: float, fb: np.ndarray, wb=np.ndarray
) -> np.ndarray:
"""
Given a state, delta_t, and existing accelerations, return
the predicted state after delta_t time has passed.
"""
# Pull these values from the state
L = state[0]
lambda_ = state[1]
h = state[2]
phi = state[3]
theta = state[4]
psi = state[5]
Vn = state[6]
Ve = state[7]
Vd = state[8]

v_n = np.array([Vn, Ve, Vd]).reshape(3, 1)

if self.model_type == "FB":
fb -= state[9:12]
wb -= state[12:15]

R_nb_prev = Rotation.from_euler(
"xyz", np.array([phi, theta, psi]).reshape((3,)), degrees=True
).as_matrix()

#######################
# Attitude Update
#######################
omega_e = RATE
screw_omega_ei = np.array([[0, -omega_e, 0], [omega_e, 0, 0], [0, 0, 0]])

Rn_Lh, Re_Lh, Re_LhcosL = principal_radii(L, h)

omega_ne = np.zeros((3,))
omega_ne[0] = Ve / Re_Lh
omega_ne[1] = -Vn / Rn_Lh
omega_ne[2] = -(Ve * np.tan(np.deg2rad(L))) / Re_Lh

screw_omega_ne = np.array(
[
[0, -omega_ne[2], omega_ne[1]],
[omega_ne[2], 0, -omega_ne[0]],
[-omega_ne[1], omega_ne[0], 0],
]
)

omega_ne = omega_ne.reshape((3, 1))

wb = wb.reshape((3,))
screw_omega_bi = np.array(
[[0, -wb[2], wb[1]], [wb[2], 0, -wb[0]], [-wb[1], wb[0], 0]]
)
wb = wb.reshape((3, 1))

R_nb = R_nb_prev * (np.eye(3) + screw_omega_bi * delta_t) - (
(screw_omega_ei + screw_omega_ne) * delta_t * R_nb_prev
)

#######################
# Velocity Update
#######################

f_nt = 1 / 2 * np.dot(R_nb_prev + R_nb, fb)
```

```
    v_nt = v_n + delta_t * (
    f_nt
    + gravity_n(L, h).reshape((3, 1))
    - np.dot(screw_omega_ne + 2 * screw_omega_ei , v_n)
    )

    #######################
    # Position Update
    #######################
    h_new = h - (delta_t / 2) * (Vd + v_nt[2])
    Rn_Lhnew, _, _ = principal_radii(L, h_new)
    L_new = L
    L_new += (delta_t / 2) * (Vn / Rn_Lh + v_nt[0] / Rn_Lh)
    L_new += (delta_t / 2) * (Vn / Rn_Lh + v_nt[0] / Rn_Lhnew)

    _, _, Re_LhcosLnew = principal_radii(L_new, h_new)

    lambda_new = lambda_
    lambda_new += (delta_t / 2) * (Ve / Re_LhcosL)
    lambda_new += (delta_t / 2) * (Ve / Re_LhcosLnew)

    phi, theta, psi = Rotation.as_euler(
    Rotation.from_matrix(R_nb),
    "xyz",
    degrees=True,
    )

    new_state = np.zeros((self.n, 1))
    new_state[0] = L_new
    new_state[1] = lambda_new
    new_state[2] = h_new
    new_state[3] = phi
    new_state[4] = theta
    new_state[5] = psi
    new_state[6:9] = v_nt.reshape((3, 1))
    # We aren't modifying the biases, so keep
    # them as they were passed in
    new_state[9:] = state[9:]

    return new_state
```

Task 1

In this task we're asked to derive the observation model for our UKF for the FB and FF models.

To this end, we calculate our resulting z as:

$$z = I_{n \times n}\mu + N(0, R) \tag{17}$$

...wherein $I_{n \times n}$ is the identity matrix at a size of n, n being the dimensional size of our state (12 for FF, and 15 for our FB), μ is our state vector, and $N(0, R)$ is a noise term.

Thus, for our FF model, we have:

$$z = I_{12 \times 12} \begin{bmatrix} L \\ \lambda \\ h \\ \phi \\ \theta \\ \psi \\ V_N \\ V_E \\ V_D \\ e_L \\ e_\lambda \\ e_h \end{bmatrix} + N(0, R) = \begin{bmatrix} L \\ \lambda \\ h \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + N(0, R) \tag{18}$$

The errors are calculated here by taking the linear difference between the measured GNSS values and the resulting state between the *predict* and *update* steps. Thus we perform:

$$\bar{\mu}_{9:12} = \mu_{0:3} - gnss_{0:3} \tag{19}$$

...and for our FB model, we have:

$$z = I_{15\times 15} \begin{bmatrix} L \\ \lambda \\ h \\ \phi \\ \theta \\ \psi \\ V_N \\ V_E \\ V_D \\ b_{a_x} \\ b_{a_y} \\ b_{a_z} \\ b_{g_x} \\ b_{g_y} \\ b_{g_z} \end{bmatrix} + N(0,R) = \begin{bmatrix} L \\ \lambda \\ h \\ 0 \\ 0 \\ 0 \\ V_N \\ V_E \\ V_D \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + N(0,R) \tag{20}$$

Tasks 2 through 3

For tasks 1 through 3, we were charged with implementing the UKF filter, and then the feedback and feed forward (FB/FF, respectively). The implementation for all of this can be found in the attached *ukf.py* file.

We seek to get the performance of the UKF filter to an acceptable level - and, at times, even possible to complete the run through data without crashing due to absurd values causing latitude/longitude to escape its expected $(-90, 90)$ bounds, or creating a Σ covariance matrix that is positive non-definite with imaginary values. To achieve this, we needed to narrow down the expected hyperparameters. The largest range of hyperparameters we saw effects across was the scale of the noise added to the Q and R, our measurement and process covariance matricies.

To narrow down the range of noises, the provided *noise_search.py* performs a hyperparameter search. Over time we increased or lowered the ranges of search to narrow down on the decided noises for each model:

Model	Q Noise	R Noise
FB	2e-3	5e-5
FF	1e-9	0.18

Even with these discovered hyperparameters, we still introduced a jitter fix methodology to our UKF filter. Our *fix_covariance* method adds increasing amounts of jitter noise to the covariance matrix to knock it back to positive definite.

Task 4

Here we will look at the performance of the filter for both FB and FF models.

Feedback Model

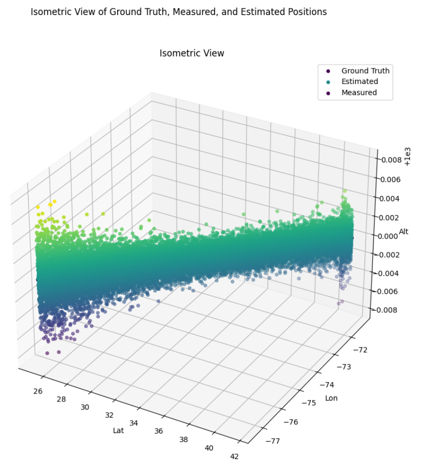

Figure 1: Isometric View of the Feedback Model

Here we see the filter path overlayed on the ground truth path of that the aircraft took.

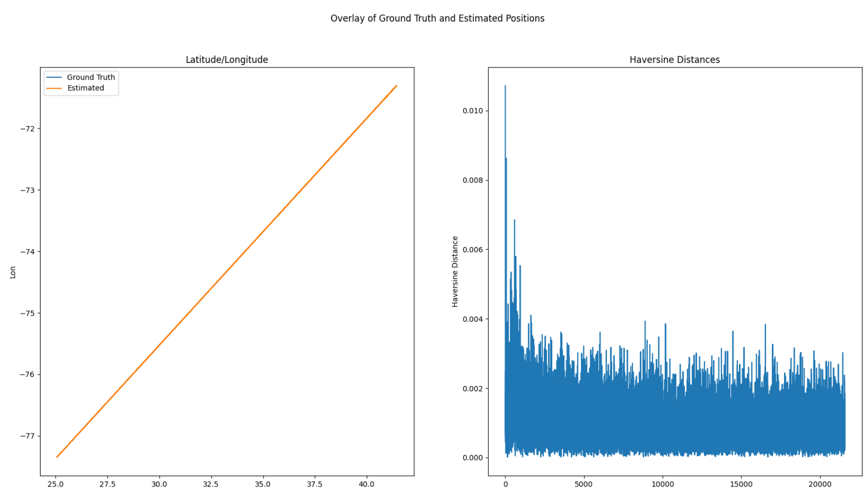

Figure 2: Latitude/Longitude + Haversine Distance for the Feedback Model

Here we see the "overhead" view of the latitude and longitude of the aircraft during its path, with the faint sign of the ground truth path behind it. We also see a charting of the haversine distance of our expected position versus our estimated over time.

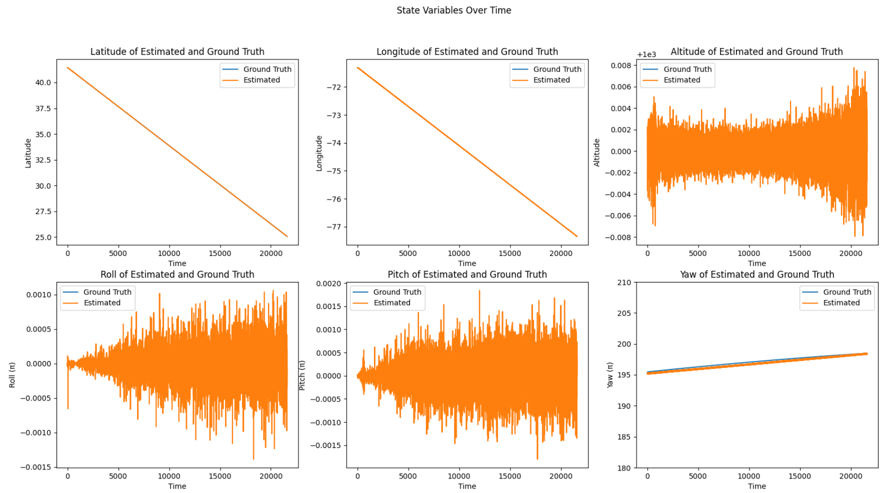

Figure 3: State Variables for the Feedback Model

Here we see the individual state variables over time for the feedback model, with our ground truth projected underneath. This covers the latitude, longitude, altitude, roll, pitch, and yaw of our system over time.

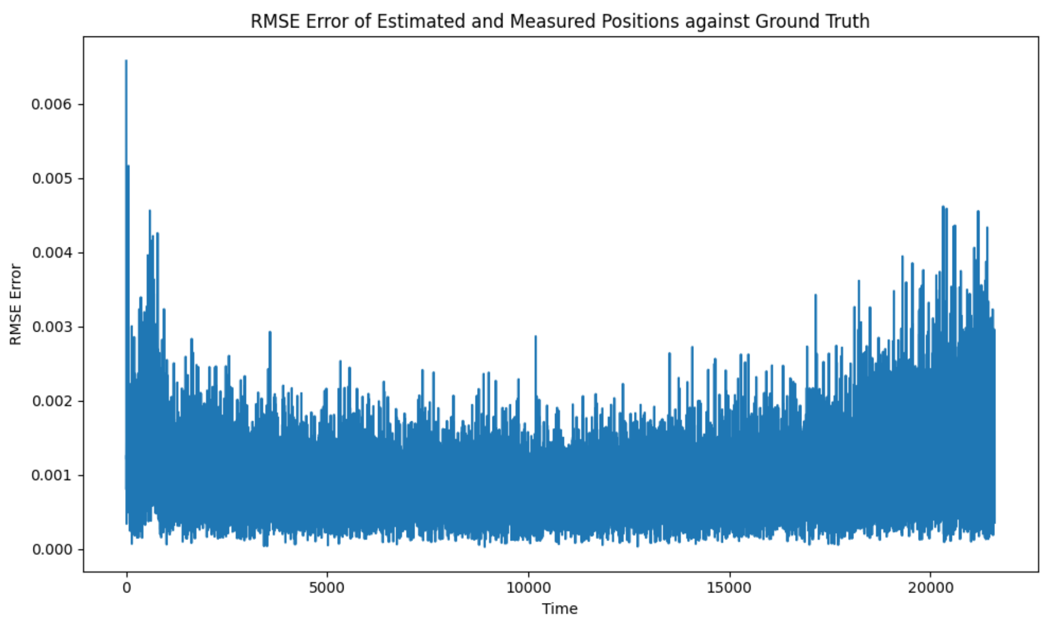

Figure 4: RMSE for the Feedback Model

Here we see the calculated RMSE error of our positional estimation from the feedback filter over time. We see that while we have some spikes, our error is generally low at around 2.5×10^{-3}.

Feed Forward Model

We now cover the feed forward model, which this author believes has an implementation issue, but still works:

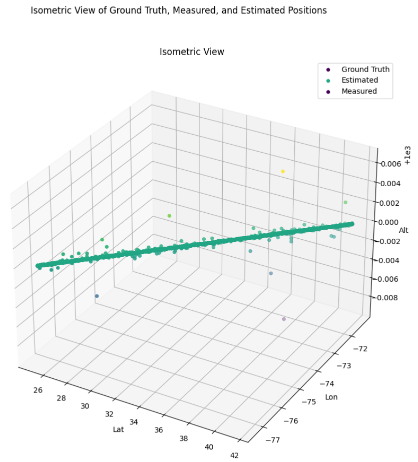

Figure 5: Isometric View of the Feed Forward Model

Here we see the filter path overlayed on the ground truth path of that the aircraft took.

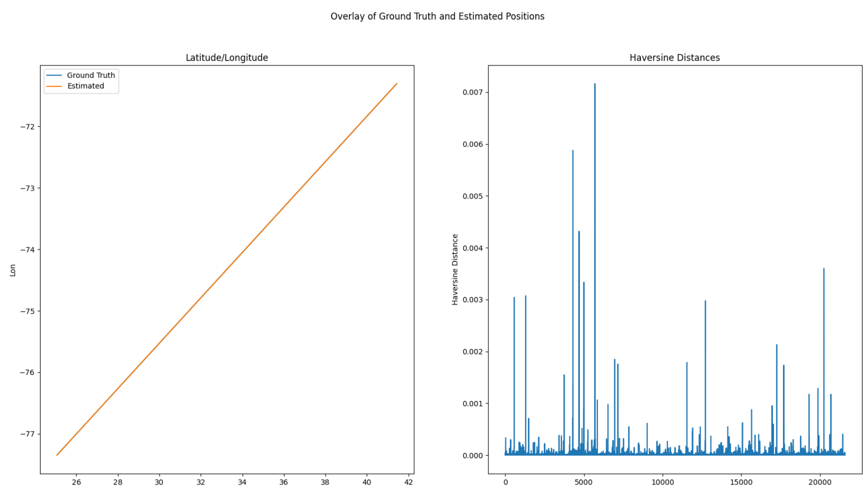

Figure 6: Latitude/Longitude + Haversine Distance for the Feed Forward Model

Here we see the "overhead" view of the latitude and longitude of the aircraft during its path, with the faint sign of the ground truth path behind it. We also see a charting of the haversine distance of our expected position versus our estimated over time. We can see the haversine distances are noticeably higher than the FB model.

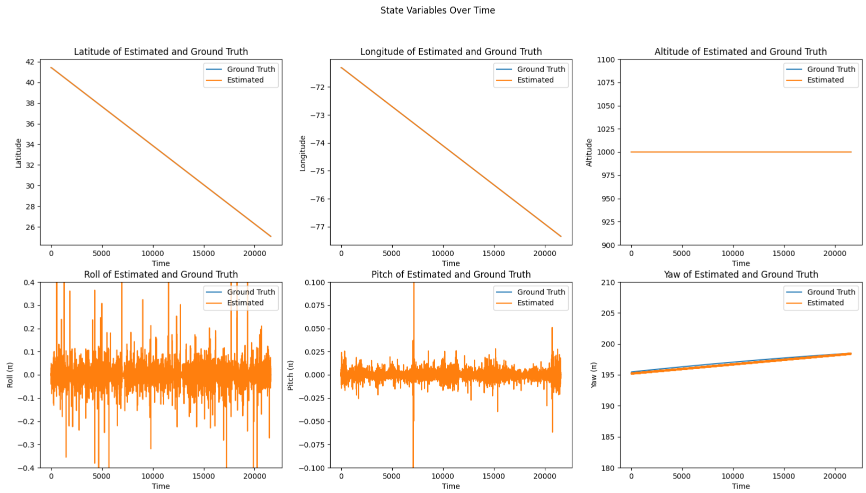

Figure 7: State Variables for the Feed Forward Model

Here we see the individual state variables over time for the feed forward model. Again we see some spikes of instability that cause results to seem skewed that we did not see in the FB model.

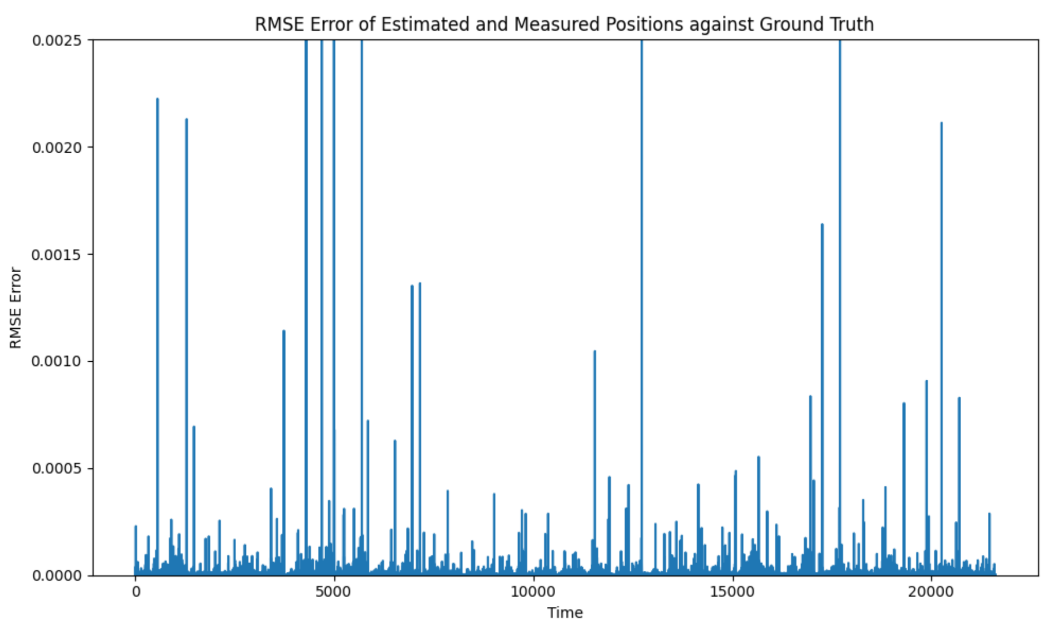

Figure 8: RMSE for the Feed Forward Model

Finally, we see the calculated RMSE error of our positional estimation from the feed forward filter over time. We see an average error close to before, at around 3.5×10^{-3}, but we see huge spikes of error due to those aforementioned instability spikes.

Discussion

The feedback model outperformed the feedforward method. This author expected the feedforward method to be more accurate, since the system considering the error and incorporating it into its estimations should create a more accurate model overall. However, this author believes they have an issue in the implementation somewhere, wherein they are introducing an inaccurate calculation of the covariance matrix to be a non positive definite matrix, causing issues down stream. To adjust this, we introduced a jitter fix method found within the *fix_covariance* recursive method, wherein we add increasing amounts of jitter noise to knock the generated covariance matrix back to positive definite. This works, but likely introduces downstreams effects to accuracy.

If we were tasked with improving this system, we would likely want to introduce some additional method to provide a loop closure in our localization of the plane, from which we can introduce ”resets” of a known position and extrapolate from there, preventing error from building over the time of the flight and creating untenable situations in navigation guidance.