TTK4250 Sensor Fusion

Graded Assignment 2 (15% of total grade)

Code and results hand in (5%): *Friday 8. November 16.00* on Blackboard as a single .zip file.
Report hand in (10%): *Wednesday 27. November 16.00 (preliminary deadline)* on Blackboard as a single PDF file.

Pairs: This assignement is to be solved in pairs. These pairs will be the same for all the graded assignments.

Code hand in: Functioning code that produces reasonable, but not necessarily perfect, results for task 3, along with one paragraph describing why you ended up tuning as you did in task 3, should be handed in by 8. November 16.00 as a zip file on Blackboard. This will account for one third of the possible points for this assignment (5% of total grade).

Report hand in: At the end of the semester you will hand in a report for all the graded assignements combined. The whole report should be maximum 10 pages long, including abstract, introduction and conclusion. This leaves approximately 3 pages per assignment, although this is *up to you*. You should, therefore consider carefully what to have in it. We recomend starting to write this now, when you actually are given some time to do it.

A full answer and result analysis is expected for task 3 and 4. For task 3 you should include a plot of the different states over time as well as the error states for your choice of parameters, in addition to NEES and NIS over time for the same set of parameters. In task 4 the same plots are expected, of course excluding anything that needs the ground truth. For both tasks it should be made clear why the parameters were chosen in terms of error metrics, consistency and overall result. Answers and analysis should connect theory and results to the real world, and show your understanding for the problem and solution. Try to connect the results on the simulated data to the results of the real data where it is possible.

Some more background on IMU measurements

An IMU is never perfect. Even if it is well calibrated there is always some residual errors. The main errors are scale and orthogonality errors in addition to the measurement noise. In addition we are never able to mount the IMU perfectly aligned with the body coordinates. This latter problem could be neglected if we simply define the body coordinate axes to be aligned with the IMU axes, but this will then later give a problem if we want to use the estimates to control heading, for example.

The mounting error can be specified by a rotation matrix R_M. If we denote the body coordinate frame by superscript b and IMU coordinate frame by superscript imu we get that the accleration in the different coordinate systems are related through $a^{\text{imu}} = R_M a^b$ and the rotation rate likewise $\omega^{\text{imu}} = R_M \omega^b$. The orthogonality error can be described by a 3×3 matrix, with rows corresponding to the direction of the axes it really measures. Calling this matrix O_a for the accelerometers and O_g for the gyros we have that the measured direction of the accelerations are given by $\tilde{a}^{\text{imu}} = O_a R_M a^b$ and $\tilde{\omega}^{\text{imu}} = O_g R_M \omega^b$ for the rotation rate. Notice that the rows of $O_a R_M$ and $O_g R_M$ specifies the direction of measurement in body coordinates. The last error we accomodate is the scale which can be represented by a diagonal matrix with positive entries.

Letting D_a and D_g denote this matrix for accelleration and rotation rate, respectively, we get the relations

$$a_m = D_a O_a R_M a_t^b + a_{bt} + a_n \tag{1}$$
$$a^b = R_M^T O_a^{-1} D_a^{-1}(a_m - a_{bt} - a_n) = S_a(a_m - a_{bt} - a_n) \tag{2}$$
$$\omega_m = D_g O_g R_M \omega_t^b + \omega_{bt} + \omega_n \tag{3}$$
$$\omega^b = R_M^T O_\omega^{-1} D_\omega^{-1}(\omega_m - \omega_{bt} - \omega_n) = S_g(\omega_m - \omega_{bt} - \omega_n), \tag{4}$$

where we have also included the biases and measurment noises. Since we do not care about the intermediate values, we let the matrices $S_a = R_M^T O_a^{-1} D_a^{-1}$ and $S_g = R_M^T O_\omega^{-1} D_\omega^{-1}$ specify all the systematic linear IMU measurement errors. In the following you can assume these matrices to be known, and they will be given to you in the data sets. In code this is compensated for by preprocessing a_m, a_b, ω_m and ω_b by multilpying with these matrices, and by multiplying the entries of the error state system matrices from the right. This will be taken care of for you in the code that is handed out.

Some background on GNSS measurements

Global navigation satelite systems include GPS, GLONASS, Galileo and BeiDou. These consists of several sattelites in a specific constellation that sends out a signal. A receiver can pick up these signals and can calculate what is known as pseudoranges for each received signal. From these pseudoranges and knowing the satelite position one can estimate the position and time of the receiver. The uncertainty of this estimate is higly dependent on how many satelites that is received and their geometric configuration. As such it is also calculated an uncertainty estimate for each measurement based on the geometry and pseudoranges. In the given real dataset this is represented by a single number and you can use this in the measurement covariance matrix if you want.

Task 1: *Quaternion functions*

(a) Implement the function `S = crossProdMat(n)` that should implement equation (10.5). That is, it takes a vector $n \in \mathbb{R}^3$ and creates the skew symetric matrix that corresponds to the matrix multiplication implementation of the cross product.

(b) Implement the quaternion product in the function `q = quatProd(ql, qr)`. Let the function operate on both 3 and 4 dimensional vectors representing a pure and general quaternion, respectively. Either equation (10.21) or (10.34) gives a formula you can use.

(c) Implement the function `R = quat2rotmat(q)` that returns a rotation matrix corresponding to a given quaternion.

(d) Implement the function `eul = quat2euler(q)` that takes a $4 \times n$ array with quaternions as columns and outputs the corresponding euler angles in a $3 \times n$ array.

Task 2: *ESKF implementation*

This task is worth up to 3% for code hand in.

(a) Implement the discrete time prediction of equation (10.58) in the function `xnompred = ESKF.predictNominal(obj, xnom, acc, omega, Ts)`. Assume that $a \approx R(q)(a_m - a_b)$ and $\omega \approx \omega_m - \omega_b$ is constant over the sampling time period, and assume that `acc` and `omega` are debiased. Remember to keep the predicted quaternion normalized.

Hint: The assumptions give that $v(k+1) = v(k) + T_s R(q)a$, $p(k+1) = p(k) + T_s v(k) + \frac{T_s^2}{2} R(q)a$ and local rotation vector increment $k = T_s\omega$ in body frame. The local rotation vector increment k in body coordinates gives the predicted quaternion $q(k+1) = q(k) \otimes e^{\frac{k}{2}} = q(k) \otimes \left[\cos\left(\frac{|k|}{2}\right) \quad \sin\left(\frac{|k|}{2}\right) \frac{k^T}{|k|}\right]^T$.

(b) Implement the error state system matrix $A(x)$ in equation (10.68) in the function `A = ESKF.Aerr(obj, xnom, acc, omega)`. Assume that acc and omega are debiased. The matrices S_a and S_g are handled for you already at the end of the function.

(c) Implement the noise input matrix $G(x)$ in equation (10.68) in the function `G = ESKF.Gerr(-, xnom)`

Hint: The MATLAB function blkdiag might simplify things.

(d) Discretize the error state matrices using Van Loan's method in the function `[Ad, GQGd] = ESKF.discreteErrMats(obj, xnom, acc, omega, Ts)`. Assume that acc and omega are debiased.

Hint: Van Loan gives that

$$\exp\left(\begin{bmatrix} -A & GQG^T \\ 0 & A^T \end{bmatrix} T_s\right) = \begin{bmatrix} \exp(-AT_s) & \exp(-AT_s)Q_d \\ 0 & \exp(A^T T_s) \end{bmatrix} = \begin{bmatrix} \exp(-AT_s) & \exp(-AT_s)Q_d \\ 0 & \exp(AT_s)^T \end{bmatrix},$$

and therefore gives both the matrices you need in the latter culumns.

Hint: If you profile your code you might find that `expm` makes your code slow, and that you want to use a simpler taylor or Padé approximation.

(e) Implement the function `Ppred = ESKF.predictCovariance(obj, xnom, P, acc, omega, Ts)` that predicts the error state covariance. Assume that acc and omega are debiased.

Hint: This now amounts to a discrete time KF covariance prediction using the previous functions to get the discrete system matrices A_d and Q_d.

(f) Combine the above into a single function that predicts both the nominal state and the error state covariance in the function `[xnompred, Ppred] = ESKF.predict(obj, xnom, P, zAcc, zGyro, Ts)`.

Hint: zAcc and zGyro are the real measurements and should be corrected and debiased before further use. The correction using S_a and S_g is already handled for you, and debiasing is done through $\omega = \omega_m - \omega_b$, where ω_b is the estimated bias.

(g) Implement the error state injection in the function `[xreset, Preset] = ESKF.inject(-, xnom, deltaX, P)`. This should impement the equations (10.85) and (10.86). Again, remember to keep the quaternion normalized.

(h) Implement the function `[v, S] = innovationGNSS( , xnom, P, zGNSSpos, RGNSS, leverarm)` that give the innovation and innovation covariance for a GNSS position measurement. The compensation of a potential lever arm is implemented for you.

(i) Implement the GPS update function in `[xupd, Preset] = updateGPS(obj, xnom, P, zGPS, RGPS, leverarm)`. Note that the output should be the values after injection. The compensation of a potential lever arm is implemented for you.

(j) Implement the function `NIS = NISGNSS(obj, xnom, P, zGNSSpos, RGNSSpos, leverarm)` that calculates the NIS of a GNSS position measurement.

(k) Implement the function `deltaX = deltaX( , xnom, xtrue)` that calculates the true error state.

Hint: $q_t = q \otimes \delta q \Rightarrow \delta q = q^* \otimes q_t$ and $\mathrm{Im}(\delta q) \approx \frac{1}{2}\delta\theta$.

(l) Implement the function `[NEES, NEESpos, NEESvel, NEESatt, NEESaccbias, NEESgyrobias] = NEES(obj, xnom, P, xtrue)` that give out the NEES of the total state and the NEES of the "individual" substates. You might also want to expand this to include total NEES for position, velocity and attitude as well.

Task 3: *Run ESKF on simulated data x*

This task is worth up to 2% for code hand in and 4% for report hand in.

A fixed wing UAV has been simulated with 100Hz IMU measurements, and 1Hz GNSS measurements for 900 seconds. You are given a data set consisting of

- zAcc (3 x 90000): the accelerometer measurements
- zGyro (3 x 90000): the gyro measurements
- timeIMU (1 x 90000): the timing of the IMU measurements starting at 0.
- S_a (3 x 3): the accelerometer correction matrix
- S_g (3 x 3): the gyro correction matrix
- zGNSS (3 x 900): the GNSS position measurements
- timeGNSS (1 x 900): the timing of the GNSS measurements with the same clock as the IMU measurements

Note that an IMU is causal and that it measures the acceleration over the last time step in some manner, and not the acceleration/rotation into the future. This means that it makes sense to use the measurements at a time step k to predict the state from $k-1$ to k and not from k to $k+1$.

A MATLAB script has been handed out to you to get you started on this task. Note that this script is only there for your convenience and you can make your own from scratch if you like.

(a) Tune the parameters of the filter. These includes the IMU measurement noises, the IMU bias time constants (or maybe rather the reciprocal of the time constant) and its driving disturbance and the GNSS position noise. You can do this by looking at both NIS for the GNSS measurements and NEES, as well as the errors.

Hint: The GNSS has been simulated with a constant noise covariance. The planar uncertainties can safely be taken to be the same, while the altitude uncertainty tends to be higher than the planar for GNSS. We have no indication that anything other than diagonal measurement noise covariance should be used. The bias time constants are probably better specified in hours rather than in seconds (ie. they should be relatively large, perhaps even infinite giving a random walk).

(b) Take a look at the attitude estimation error. If you have done it right, it should be quite small. Roll and pitch can be estimated since gravity acts on the accelerometer, so that is fine. But what is it that makes the heading observable for us? Would you trust this system to give the correct heading if it was standing still as well?

Hint: Think of what happens when a GNSS measurment arrives if you have integrated up the IMU measurements in the wrong direction.

(c) Try to set the IMU misalignment matrices (`eskf.Sa` and `eskf.Sg` for an ESKF object `eskf`) to the identity matrix, effectively neglecting any mounting errors, scale errors and orthogonality errors. What does it do to your estimates, and why do you believe it does so? Could you disregard this matrix in real life?

Task 4: *Run ESKF on real data*

This task is worth up to 6% for report hand in.

You are given a data set from a real fixed wing UAV flight where a STIM300 IMU gives 250Hz measurements, and two ublox-8 GNSS receiver gives 1 Hz measurements. The UAV can be seen in Figure **??**, where the two black bits sticking up are the two GNSS antennas that are on board. You are only given the position data from the front one. A plot of the GNSS trajectory can be seen in Figure **??**. The dataset you are given consists of

- zAcc (3 x K): The accelerometer measurements

Figure 1: The UAV used in the experiment

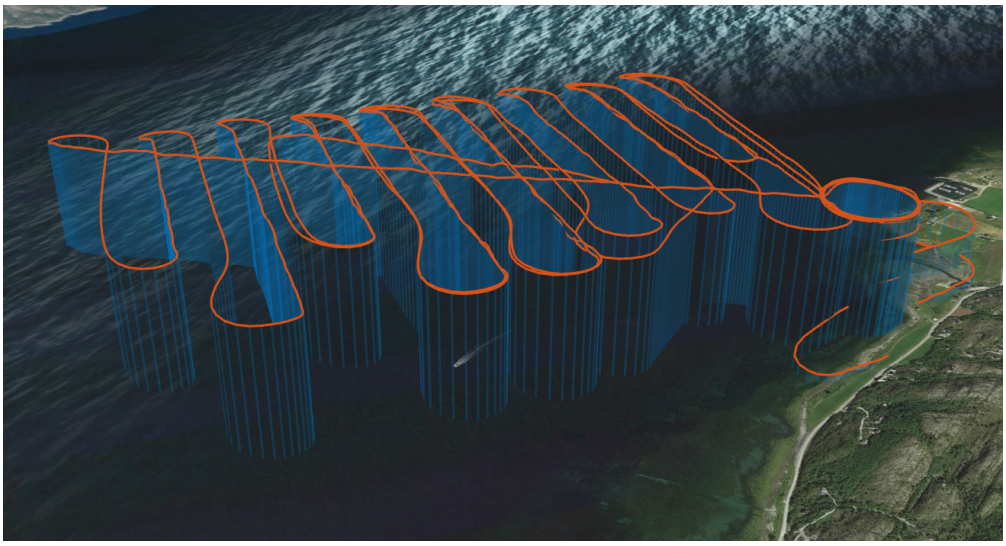

Figure 2: The path taken by the UAV

- zGyro (3 x K): The gyro measurements
- timeIMU (1 x K): The timing of the IMU measurements in seconds into the GPS week.
- S_a (3 x 3): Acceleration correction matrix.
- S_g (3 x 3): Gyro correction matrix.
- zGNSS (3 x KGNSS): GNSS position measurements
- timeGNSS (1 x KGNSS): The timing of the GNSS measurements in the same "time variable" as the IMU measurements.
- GNSSaccuracy (1 x KGNSS): The position accuracy claimed by the receiver in meters.
- leverarm (3 x 1): The lever arm in meters. Specified as the vector from the IMU center to the GNSS antenna center.

(a) Tune the parameters of the filter. It is the same parameters to tune as for the last task. You do not have the ground truth anymore, but you can still use NIS.

The datasheet for the STIM300 (available online) can give you some hints on where to start tuning. Also, a good starting point would be the same values as for the last task. If discrete noises were specified they will need to be changed to the new sampling time.

You can use the GNSSaccuracy in some manner (ie, scale it etc.) to specify the position covariance or use a constant one. This is up to you.

Hint: Dividing the NIS into planar and altitude can possibly be helpful.

(b) Try to set the IMU misalignment to the identity matrix, effectively neglecting any mounting errors, scale errors and orthogonality errors. How does this change your estimates? If you did not know, could you tell that you are now using the "wrong" IMU measurements? What does this tell you about setting up a ESKF to be used in a real world application?