Northeastern University April 24, 2019
Department of Electrical and Computer Engineering GNSS Signal Processing (Spring 2019)
Mark Higger

Kalman Filter Based Sensor Integration for Navigation

Abstract - Precision navigation is an important part of many systems. Recently, many autonomous systems such as AAVs, AUVs and Autonoumous cars rely on precise navigation for the functionality. In tandem with this, many navigational sensors have become cheaper and more accessible to use. Because of this, sensor fusion is often a useful tool to get the most out of multiple sensors, and can often replace a single high-cost sensor. With this in mind it is important to discuss how to fuse these sensors and what trade-offs occur during sensor fusion

Specifically this paper addresses the use of Kalman filtering for sensor integration, using the total-state and error-state estimation techniques. By using a total-state Kalman filter, it allows for flexible integration of multiple sensors in a single Kalman filter. In contrast, error-state estimation uses aiding sensor measurements for a Kalman filter to estimate the error inside a single reference navigation solution.

If implemented properly, Kalman filter-based solution can offer a high accuracy solution with a relatively low computational cost. However, this comes at a cost of needing an accurate system model, as the Kalman filter is extremely sensitive to inaccuracies in the model

1 Introduction

With a recent increase in autonomous technology, such as AAVs, AUVs and Autonoumous cars, precise navigation has become more and more important Throughout history navigation has been split into two types; dead reckoning and postion fixing [1]. Dead reckoning is measuring change in a body's position and applying that change to update the body's current position. Historically this was done my measuring distance traveled by a float over long periods of time to get velocity, however with more modern technology IMUs are often used to get linear acceleration and angular velocity using accelerometers and gyroscopes respectively. It is also important to note that other modern forms of dead reckoning are also available, such as using motor/servo encoders or visual odometry. However, regardless of the method, dead reckoning techniques all have a large buildup of error over time due to the cascading effect of being based off of previous position. In contrast position fixing finds the position of a body by using reference points with known position and distance from the body. Historically these reference points were stars but more recently satellite references or even computer vision-based locations have been used. This type of navigation is very useful since, unlike dead reckoning, the error in position does not build up over time, however it often still has high amounts of error in each measurement.

With this in mind I wanted to explore sensor integration techniques and how to combine sensors in order to reduce overall error in navigational systems. However, due to range of solutions available for sensor integration I wanted to focus specifically on using Kalman filtering for integration of GPS, IMU and tertiary sensor data. Furthermore, rather than just go into the theory behind how this integration works, data was collected using Northeastern University's Center of Robotic's Robotic car and processing was attempted using the described techniques as part of a group project for a robotic sensing and navigation class [2]. The P. Groves textbook [1] was used as the main source of information for sensor integration and Kalman filtering techniques, with various other sources being used for supplementary information.

2 Navigation Solutions

2.1 GNSS

GNSS (Global Navigation Satellite System) systems, such as GPS or GLONASS, use satellites as reference points for position fixing. Specifically, GNSS satellites broadcast their position (ephemeris) and a time stamp of the current time at the satellite. Since the propagation speed of a radio wave is known, a receiver can then use this time stamp and position to get a psuedorange from each satellite for a multilateration-based position (shown in Fig. 1). Unfortunately these psudoranges are not perfect, but can be modeled as having approximately gaussian error, with a mean approximately equal to the real value of the psudorange. Furthermore, there are also errors in the clock on the satellites (known) and the receiver (unknown). We can Model these psudoranges (Eq.1) between the i^{th} satellite and the receiver(ρ_i) as a function of the speed of light (c), the time delay between the receiver and the satellite ($t_{Rx} - t_{Tx}$), the difference between clock offsets ($\delta t - \delta t_i$) and some gaussian noise (ϵ_i) [3].

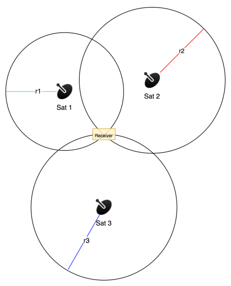

Figure 1: GNSS positioning

$$\rho_i = c((t_{Rx} - t_{Tx}) + (\delta t - \delta t_i)) + \epsilon_i \tag{1}$$

Without compensating for these errors in psudoranges and clock offsets, GNSS usually has a very low precision and accuracy. We can, however, use error estimation techniques such as Kalman filtering to more accurately estimate the position, velocity and clock offset of the receiver.

2.2 IMU

In contrast to GNSS, IMUs use relative dead reckoning methods. Specifically IMUs use a combination of 3-axis accelerometers and gyroscopes to find x,y,z linear acceleration and angular velocity. Using the measured acceleration and angular velocity (combined with an initial position, heading and velocity) position and heading of a body can be calculated. The body's velocity at some sample n is the integral of the measured acceleration plus the initial velocity (i.e. $v_b[n] = v_0 + \Sigma_0^n a_b[n]\tau_s$, with a sample rate of $\frac{1}{\tau_s}$). The attitude can similarly be obtained as Euler angles by integrating the angular velocity as $\Psi[n] = \Psi_0 + \Sigma_0^n \omega_b[n]\tau_s$. However, to obtain position of the body it is necessary to obtain a new frame of reference, since the position of the body in relation to the body is always **0**. While other frames exist (i.e. ECEF or ECI) the remainder of the calculations will be done in NED (North, East, Down) navigation frame, which can either be represented in UTM (x (m), y (m), height(m), zone) or Lat/Lon (ϕ (rad), λ (rad), height (m)). In order to transform between a body frame and NED frame, we need to employ a rotation matrix, which is derived from ω_b the x,y,z angular velocity of the gyroscope and the initial attitude by Eq. 2 modified from the Groves textbook, where C_b^n is the rotation matrix between the body and navigation frame [1, Chapter 5].

$$C_b^n[n] = C_b^n[n-1](I_3 + \Omega_b \tau_s) \tag{2}$$

$$I_3 \triangleq \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\Omega_b = [\omega_b \wedge] \triangleq \begin{bmatrix} 0 & -\omega_{bz} & \omega_{by} \\ \omega_{bz} & 0 & -\omega_{bx} \\ -\omega_{by} & \omega_{bx} & 0 \end{bmatrix}$$

Once the rotation matrix is calculated, it can be applied to the body frame velocity (v_b) to obtain the navigation frame velocity (v_b^n) using Eq. 3 which can be further integrated to obtain the position of the body in the navigation frame using Eq. 4 where p is the position in UTM coordinates. However, it is important to note that using UTM coordinates simplifies the math greatly, but causes error when going over larger scale distances since it ignores the curvature of the earth. I implemented these equations semi-successful in matlab (Appendix A) as a proof of concept.

$$v_b^n = \frac{1}{2}(C_b^n[n-1] + C_b^n[n])v_b \tag{3}$$

$$p_b^n[n] = p_b^n[n-1] + v_b^n \tau_s \tag{4}$$

It is important to note, that the measured values of the accelerometers and gyroscopes have inherent errors (ϵ_a, ϵ_g) that get propagated through the position updates. For raw IMU measurments we can usually assume that these errors are gaussian with some bias μ and variance σ^2. Since the position of the body is calculated from integrated measurements the error gets propagated quadratically with time, causing significant buildup of error over time. Because of this buildup, inertial navigation often needs to be paired with external sensors such as GNSS to reduce the error buildup. While IMUs or INSs can be built to minimize this error, the cost of the sensors increases dramatically, and there is often still need for sensor integration and error reduction techniques.

2.3 Tertiary Sensors

While GNSS and INS are the most common forms of navigation, there are many more methods for both current and future systems. Many of these tertiary sources of navigation can provide much more accurate sensor reading when available, but may not provide a comprehensive position on it's own, and thus it is important to discuss how to integrate them in a more generalized manner. For this paper the Ackermann steering model will be used as an example tertiary sensor where data has been collected and integrated [2].

The Ackermann steering model uses wheel and axle encoders on a car-like frame to find yaw rate and forward velocity . While measuring speed from the encoders is much more accurate than the IMU, it only gives velocity in the forward direction of the body frame. Similarly the axle encoder gives a much more accurate measurement of yaw rate, but the Ackermann model does not compensate for any changes in pitch or roll. Given this, the Ackermann model needs to be combined with IMU data in order to give a complete dead reckoning position of a body. For all of the calculations for forward velocity and yaw rate on our real data the ROS ackerman_msg package was used [4].

3 Kalman Filters

One of the problems with real sensors is that there is always some noise introduced into the measurements. Sometimes (as in the case of GNSS or IMU) this noise can be modeled as having approximately gaussian distribution, which means it can be characterized as having some mean μ and some variance σ^2. In the case of GNSS psudoranges we can assume that the actual value of the psudorange is μ, and in an IMU we can assume that the actual values of the acceleration and angular velocity are μ minus some bias (b). It is then important to represent our sensor measurements not as numbers but as gaussian random variables. We can assume that the actual value of each state of our measurements (x_t) is then dependent on both our previous state(x_{k-1}), our transitions between states(F_t), our measurements (y_t) and the transformation between our measurements and our states (H_t). We can represent this as in Eq. 5 [3] where ϵ_t and η_t are the zero-mean gaussuian noise in the states and measurements respectively. Using this idea we can estimate the mean of these random variables over time using a Kalman filter. A Kalman filter essentially represents each new state as a function of the previous state and weighted measurements. This weight is called the Kalman gain and is the optimal (in least squares sense) gain for the system.

$$\begin{cases} x_t &= F_t x_{k-1} + \epsilon_t \\ y_t &= H_t x_t + \eta_t \end{cases} \tag{5}$$

3.1 Basic Kalman Filter

A standard kalman filter has two major steps, a prediction and an update. The Prediction step (Eq. 6 and Eq. 7) takes in the previous state ($\hat{x}_{t-1|t-1}$) and the state transition (F_t) to predict the next state ($\hat{x}_{t|t-1}$). Similarly the predicted covariance of the state ($P_{t|t-1}$) is also predicted using the state transition and the system variance (Q_t).

$$\hat{x}_{t|t-1} = F_t \hat{x}_{t-1|t-1} \tag{6}$$

$$P_{t|t-1} = F_t P_{t-1|t-1} F_t^T + Q_t \tag{7}$$

The update step then finds the Kalman gain and residuals and applies them to find the state and state covariance estimate. The first part of the update step is to find the residual z_t (Eq. 8) as a function of the measurements (y_t), the predicted state ($\hat{x}_{t|t-1}$) and the transform between the measurements and the states (H_t). Similarly The residual covariance (S_t) is calculated using the measurement covariance (R_t), the measurement tansformation and the predicted state covariance (Eq. 9)

$$z_t = y_t - H_t\hat{x}_{t|t-1} \tag{8}$$

$$S_t = H_t P_{t|t-1} H_t^T + R_t \tag{9}$$

Next the optimal Kalman gain (K) is calculated (Eq. 10) and the new states ($\hat{x}_{t|t}$) and state covariances ($P_{t|t}$) are estimated (Eq. 11, Eq. 12)

$$K_t = P_{t|t-1} H_t^T S_t^{-1} \tag{10}$$

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t z_t \tag{11}$$

$$P_{t|t} = (I - K_t H_t) P_{t|t-1} \tag{12}$$

These steps can be iterated over time for each set of measurements in order to estimate states and their covariances. It is also important to note, that kalman filtering is only optimal when when the system model (i.e. F_t, H_t, Q_t and R_t) is fully known, and is very sensitive to inaccuracies in the model.

4 Navigation Integration

There are many ways to integrate different navigational sensors such as position fixing, RTK (use of multiple GNSS receivers) or complementary filtering (using different frequency components from redundant measurements). For the remainder of this paper however, Kalman filter-based solutions will be discussed. Specifically I will be addressing total state estimation, which directly estimates states of all the sensors, and error state estimation, which uses uses sensor data to estimate the error in a reference sensor.

4.1 Total State Estimation

One of the methods of sensor fusion is using a total state Kalman filter Estimation. This method uses a single Kalman filter who's states contain measurements from all of the sensors (shown in Fig. 2).

This is the method used for the data processing in the ros packages [5] used for our experiments [2]. In terms of the Kalman our states and covariences that we estimated are shown in Eq. 13.

$$\hat{x} = \begin{bmatrix} \hat{x}_{GNSS} \\ \hat{x}_{IMU} \\ \hat{x}_{Ack} \end{bmatrix} \tag{13}$$
$$P = \begin{bmatrix} P_{GNSS} & P_{IMU,GNSS} & P_{Ack,GNSS} \\ P_{GNSS,IMU} & P_{IMU} & P_{Ack,IMU} \\ P_{GNSS,Ack} & P_{IMU,Ack} & P_{Ack} \end{bmatrix}$$

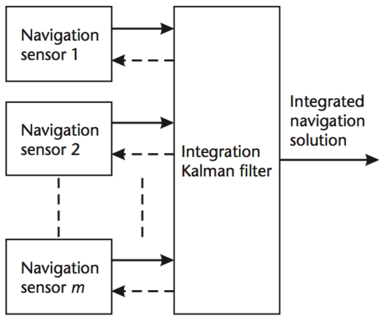

Figure 2: Total-State sensor integration [1, Chapter 14]

However we can also pick which states each sensor contributes to Kalman filter. For the states in our experiments we evaluated the strengths of our different sensors and selected the appropriate states (Table 1) to use. We picked the position fixing from the GNSS receiver since it has the most direct measure of position. Similarly we chose to discard measurements in the IMU that are redundant with the Ackermann model, since the Ackermann model much more accurately represents the actual values from measurements. With this state selection we used a the ROS ekf_localization node to implement the total state estimation using an extended Kalman filter. Using this method we were

Table 1: State selections for sensors in total state estimation

	$\hat{p}_x$	$\hat{p}_y$	$\hat{p}_z$	$\hat{\Psi}_x$	$\hat{\Psi}_y$	$\hat{\Psi}_z$	$\hat{v}_x$	$\hat{v}_y$	$\hat{v}_z$	$\hat{\omega}_x$	$\hat{\omega}_y$	$\hat{\omega}_z$	$\hat{a}_x$	$\hat{a}_y$	$\hat{a}_z$
GNSS	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0
IMU	0	0	0	0	0	0	0	0	0	x	x	0	0	x	0
Ack	0	0	0	0	0	0	x	x	0	0	0	x	0	0	0

able to get a position fix. Shown in Fig. 3 is an excerpt from the car driving through downtown Boston through tall buildings, leading to low quality GNSS data.

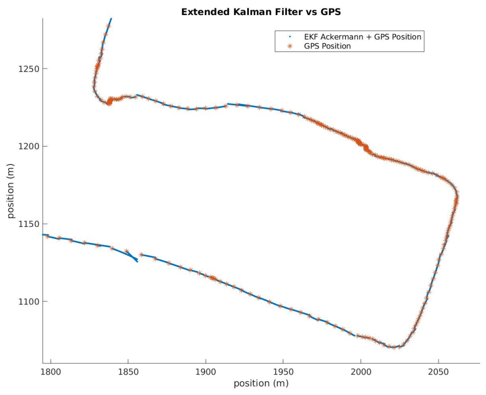

Figure 3: Total-state Sensor Integration from Car Data Collection [2]

While the data integrated nicely when GNSS reception was good, the impementation started to break down in areas of poor GNSS reception. One of the major issues with our implementation is that the GNSS measurements we collected from the sensor were already fixed, presumably by an aggressive EKF. This led to non-gaussian error with extremely low variance (on an order of 10^{-20} for still data.). This erroneous low variance caused the Kalman filter to weight the GNSS position much higher than it should have been, causing the body's position to fix to the GNSS position, even when GNSS data had poor accuracy. This demonstrates one of the major flaws with Kalman filter integration, which is its sensitivity to model accuracy. Since our actual GNSS measurements did not fit our model well, the integration had major failure points. While not as obvious in our data the total state integration has some other flaws as well. One of the major drawbacks is that it is computationally (in processing and memory) very expensive to model a very large kalman filter due to large matrix algebra, often making it unsuitable for real time needs of navigation. Another major drawback is that the covariance of all of the states builds up over time when using a dead-reckoning method, causing it to be unsuitable for long-term navigation.

4.2 Error State Estimation

Rather than model the all of the sensors with their own kalman filter states we can use a reference sensor and use measurements from the other sensors' measurements to determain the error in the reference sensor. This method is called error state estimation (Fig. 4). Error state estimation can further be split into either open or closed loop. In an open loop system, the errors are first calculated and then applied to the reference navigation solution after all errors are calculated over time. In a closed loop system, the errors estimated by the Kalman filter are directly applied to the navigation solution during calculation. [1, Chapter 12]

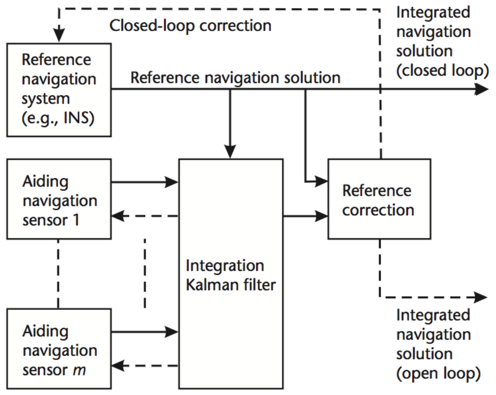
Closed-loop correction
Integrated navigation solution (closed loop)
Reference navigation system (e.g., INS)
Reference navigation solution
Aiding navigation sensor 1
Integration Kalman filter
Reference correction
Aiding navigation sensor *m*
Integrated navigation solution (open loop)

Figure 4: Error-State sensor integration [1, Chapter 14]

In most situations it is practical to use the IMU as the reference sensor, due to its high sample rate and comprehensive navigation solution. Furthermore, when using GNSS as an aiding sensor it can either be either loosely-coupled (Fig. 5) or tightly-coupled (Fig. 6). Loosely-coupled GNSS integration simply uses the output of the GNSS receiver's position and velocity solutions as measurements for the Kalman filter residuals.

The error-state Kalman filter for loosely-coupled integration included the error in attitude ($\delta\hat{\Psi}$),

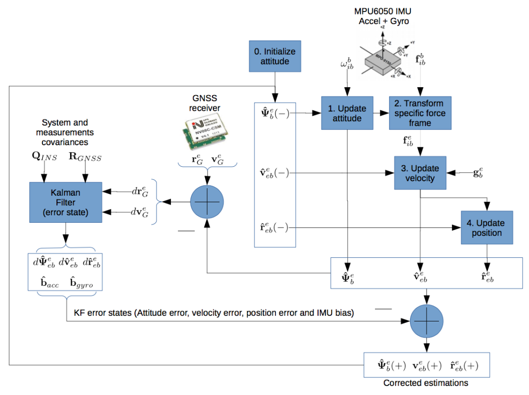
MPU6050 IMU
Accel + Gyro
0. Initialize attitude
1. Update attitude
2. Transform specific force frame
3. Update velocity
4. Update position
GNSS receiver
System and measurements covariances
Kalman Filter (error state)
KF error states (Attitude error, velocity error, position error and IMU bias)
Corrected estimations

Figure 5: Loosely-coupled GNSS-IMU integration using error-state estimations [6]

the error in velocity ($\delta\hat{v}$), the error in position ($\delta\hat{r}$) and the accelerometer and gyroscope biases ($\hat{b}_a, \hat{b}_g$) shown in Eq. 14, noting that these errors are represented in the same coordinate frame as your IMU solution.

$$\hat{x} = \begin{bmatrix} \delta\hat{\Psi} \\ \delta\hat{v} \\ \delta\hat{r} \\ \hat{b}_a \\ \hat{b}_g \end{bmatrix} \tag{14}$$

In contrast tight GNSS integration directly uses the psudorange meas uerments from the GNSS receiver, and allows for error in psudorange measurements to be input into the GNSS psudorange calculations [6]. In order to do this we add error in psudoranges ($\delta\hat{\rho}$) and psudorange rates($\delta\hat{\dot{\rho}}$) into the state estimation of the Kalman filter changing Eq. 14 to Eq. 15.

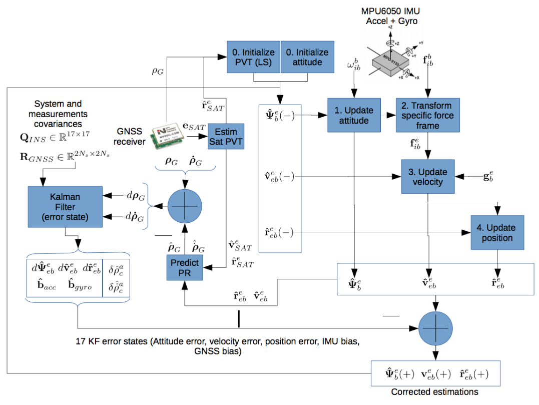
MPU6050 IMU
Accel + Gyro
0. Initialize PVT (LS)
0. Initialize attitude
1. Update attitude
2. Transform specific force frame
3. Update velocity
4. Update position
System and measurements covariances
GNSS receiver
Estim Sat PVT
Kalman Filter (error state)
Predict PR
17 KF error states (Attitude error, velocity error, position error, IMU bias, GNSS bias)
Corrected estimations

Figure 6: Tightly-coupled GNSS-IMU integration using error-state estimations [6]

$$\hat{x} = \begin{bmatrix} \delta\hat{\Psi} \\ \delta\hat{v} \\ \delta\hat{r} \\ \hat{b}_a \\ \hat{b}_g \\ \delta\hat{\rho} \\ \delta\hat{\dot{\rho}} \end{bmatrix} \tag{15}$$

While important to implementing integration, I will not be discussing error-state transformations and psudorange measurement transformations, as it could constitute its own full paper, however in-depth information for both loosely and tightly coupled integration can be found in chapter 12 of the Groves textbook [1]. Furthermore, appendix B contains matlab code that I wrote for loosely-coupled integration in a NED navigation frame using the actual data collected [2] that I was not able to get working, but should be mostly correct.

5 Conclusion

Overall Kalman filter integration of GNSS, IMU and tertiary navigation solutions has the ability to work very well, however due to the sensitivity of the Kalman filter it can often be very tricky to implement. Furthermore, when collecting real data from an off-the-shelf GNSS receiver it is often difficult or impossible to get either raw position or psuedorange measurements. This greatly limits the usability of these integration techniques, specifically in error state estimation, since the Kalman filter requires as raw of data as possible. It is possible to implement total-state estimation in this case, however, it is still very difficult to get proper system and measurement variances. There are also further improvements that can be used for these navigation solutions. One of the most common is to use an EKF or UKF instead of a standard Kalman filter, since it allows for nonlinear transition matrices. Another popular improvement is the use of zero-velocity updates. Zero-velocity updates rely on the fact that an IMU (or other sensors) are often very good at detecting when a body is still, and fixes the states and their covariances to $\mathbf{0}$, reducing cascading error in the states.

References

[1] P. D. Groves, *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems.* ARTECH HOUSE, 2008.

[2] M. Higger, J. Lynch, and C. Xu, *Integration of the Ackerman Steering Model to Improve State Estimation for Autonomous Vehicles*, 2019.

[3] P. Closas, "EECE5698: GNSS signal processing Topic 2.2: Fundamentals of satellite-based navigation," 2019.

[4] J. O'Quin. ackermann_msgs package. [Online]. Available: http://wiki.ros.org/ackermann_msgs

[5] T. Moore and D. Stouch, *A Generalized Extended Kalman Filter Implementation for the Robot Operating System.*

[6] J. Arribas, A. Moragrega, C. Fernandez-Prades, and P. Closas, “Low-cost GNSS/INS/Odometric Sensor Fusion Platform for Ground Intelligent Transportation Systems,” *Proceedings of the 30th International Technical Meeting of the Satellite Division of The Institute of Navigation(ION GNSS+ 2017)*, pp. 436–455, 2017.

A IMU NED Navigation Solution Code

```
function [C_new, f_new, v_new, h_new, Lat_new, Lon_new] = ...
    ImuNavSol_bn(C_old, w, f_body, v_old, h_old, Lat_old, Lon_old, Ti)
%Navigation solution for NED frame of IMU
%INPUTS:
%   C_old [3x3] - Previos Transform matrix between body and navigation
%       frame
%   w [3x1] - Angular Velocity of body frame (rads)
%   f_body [3x1] - Prev Specific Force (a + g) in Body Frame
%   v_old [3x1]- Prev Velocity in NED frame (m)
%   h_old - Prev Height from Geiod (m)
%   Lat_old - Prev Latitude (deg dec)
%   Lon_old - Prev Longitude (deg dec)
%   Ti - Time between last measurement
%OUTPUTS:
%   C_new [3x3] - New Transform Matrix between body and NED frame
%   f_new [3x1] - New Specific Force (a + g) in NED frame
%   v_new [3x1]- New Velocity in NED frame (m)
%   h_new - New Height from Geiod (m)
%   Lat_new - New Latitude (deg dec)
%   Lon_new - New Longitude (deg dec)

    %Gravity is fixed in NED
    g_n = [0 0 -9.8]';

    %% calculate attitude transform matrix
    C_new = C_old * (eye(3) + (skew(w)*Ti));

    %% calculate specific force as C*fb

    f_new = C_new * (f_body);

    %% calculate new velocity as V- + Ts(f + g)
    v_new = v_old + ((f_new - g_n) * Ti)';

    %% calculate new position in lat,lon and heaight
    %find height
    h_new = h_old - ((Ti/2) * (v_old(3) + v_new(3)));

    %Get North and east radii of earth
    [Rn, Re] = Radii_of_curvature(Lat_old);

    %Find Latitude
    Lat_new = Lat_old + (Ti/2) * ...
```

```
        ((v_old(1) / (Rn + h_old)) + (v_new(1) / (Rn + h_new)));

    LonVel_old = v_old(2) / (Re + h_old*cosd(Lat_old));
    LonVel_new = v_new(2) / (Re + h_new*cosd(Lat_new));
    Lon_new = Lon_old + (Ti/2) * (LonVel_old + LonVel_new);
end
```

B IMU Error Estimates Kalman Filter code

```
function [Er_new, P_new] = ...
    IMUErrorEst_NED(C, w, v, f, lat, lon, h, Er_old, P_old, ...
                    sig2_gy, sig2_a, sig2_ab, sig2_gyb, ...
                    p_G, v_G, Ti, Ts)
%Find Error estimates in IMU with Kalman Filter
%Takes in inputs from IMU navigation solution and fixed GPS coordinates and
%Inputs:
%   C [3 x 3] - Attitude transformation matrix between body and NED fzrame
%   w [1 x 3] - angular Velocity from IMU Gryos (rad/s)
%   v [1 x 3] - Velocity of IMU
%   f [1 x 3] - Specific force of IMU
%   lat - IMU lattitude (deg)
%   lon - IMU longitude (deg)
%   h - IMU height from Geiod (m)
%   Er_old [15 x 15] - Measurement errors of
%                          [att, vel, pos, acc bias, g bias]
%   P_old [15 x 15] - Measurement co-varience of
%                          [att, vel, pos, acc bias, g bias]
%   sig2_gy - sig^2 varience of Gyro
%   sig2_a - sig^2 varience of Accelerometer
%   sig2_ab - sig^2 varience of Accelerometer bias
%   sig2_gyb - sig^2 varirnce of Gyro Bias
%   p_G [1 x 3] - position from GPS in Lat, Lon, height
%   v_G [1 x 3] - Velocity from  GPS in NED;
%   Ti - Time interval of Kalman Filter
%   Ts - Time interval of IMU
%
%Outputs:
%   d_psi [1 x 3] - error in attitude in NED
%   d_v [1 x 3] - error in velocity (m/s)
%   d_lat - error in Lattitude (deg)
%   d_lon - error in Longitude (deg)
%   d_h - error in height (meters)
%   b_a - bias in accelerometer (m/s^2)
%   b_g - bias in Gyroscope (rad/s)

    %% DEFINE CONSTANTS
```

```
%Radius of Earth
[Rn, Re] = Radii_of_curvature(lat);
RadN = Rn + h;
RadE = Re + h;
R_0 = 6378137; %WGS84 Equatorial radius in meters
e = 0.0818191908425; %WGS84 eccentricity
R_geo = R_0 / sqrt(1 - (e * sind(lat))^2) *...
    sqrt(cosd(lat)^2 + (1 - e^2)^2 * sind(lat)^2); % from (2.137)
z3 = zeros(3);

%% Find state relationships between vars
%Attitude State transistion
F_11 = -skew(w);

%Attitude-Velocity State transisiton
F_12 = [0, -1/(Re + h), 0; ...
         1/(Rn + h), 0, 0; ...
         0, tand(lat) / (Re + h), 0];
%Attitude-position state Transisiton
%NOTE: Ommited factors dependent on w_ie as IMU accounts for this
F_13 = [0, 0, v(2)/((Re+h)^2); ...
         0, 0, v(1)/((Re+h)^2); ...
         v(2)/ ((Re + h)*(cosd(lat)^2)), 0, (-v(2)*tand(lat))/(Re + h)^2];
%Velocity-attitude state transition
F_21 = -skew(C*f');

%Velocity update 12.63 Groves
F_22_11 = v(3)/RadN;
F_22_12 = -(2*tand(lat))/RadE;
F_22_13 = v(1)/RadN;
F_22_21 = v(2)*tand(lat)/RadE;
F_22_22 = (v(1)*tand(lat) + v(3)) / RadE;
F_22_23 = v(2) / RadE;
F_22_31 = -2*v(1) / RadN;
F_22_32 = -2*v(2) / RadE;
F_22 = [F_22_11, F_22_12, F_22_13; ...
        F_22_21, F_22_22, F_22_23; ...
        F_22_31, F_22_32, 0];

%Velocity-Positon Update 12.64 groves
F_23_11 = -((v(2)^2*(secd(lat)^2)) / RadE);
F_23_13 = (v(2)^2*tand(lat)/RadE^2) - (v(1)*v(3)/RadN^2);
F_23_21 = (v(1)*v(2)*(secd(lat)^2)/RadE);
F_23_23 = (v(1)*v(2)*tand(lat) + v(2)*v(3)) / RadE^2;
F_23_33 = (v(2)^2/RadE^2) + v(1)^2/RadN^2 - (2*min(Gravity_NED(lat,h)) / R_geo
F_23 = [F_23_11, 0, F_23_13; ...
        F_23_21, 0, F_23_23; ...
        0, 0, F_23_33];
%Postion-Velocity Update 12.65 Groves
F_32 = [1/RadN, 0, 0; 0, 1/(RadE*cosd(lat)), 0; 0, 0, -1];

%Position-Position Update 12.66 Groves
F_33 = [0, 0, -v(1)/RadN^2; ...
        v(2)*sind(lat)/(RadE*cosd(lat)^2), 0, v(2)/(RadE^2*cosd(lat)); ...
        0, 0, 0];


F = [F_11, F_12, F_13, z3, C;...
    F_21, F_22, F_23, C, z3;...
    z3, F_32, F_33, z3, z3; ...
    z3, z3, z3, z3, z3; ...
    z3, z3, z3, z3, z3];
Phi = F.*Ti;
Phi = Phi + eye(size(F));

%% get IMU varience
n2g = sig2_gy*Ti;
n2a = sig2_a*Ti;
n2ab = sig2_ab*Ti;
n2gb = sig2_gyb*Ti;

Q = (eye(5*3) .* [n2g, n2g, n2g, ...
                  n2a, n2a, n2a, ...
                   0, 0, 0, ...
                    n2ab, n2ab, n2ab, ...
                      n2gb, n2gb, n2gb]) * Ts;

%% measurement Models

% measurement innovation is diff between gnss and IMU
d_z = [p_G - [lat, lon, h], v_G - v]';

% measurement matrix
H = [z3, z3, -eye(3), z3, z3; ...
    z3, -eye(3), z3, z3, z3];

%% Kalman Filter

Er_pred = Phi*Er_old;
P_pred = Phi*P_old*Phi' + Q;
```

```
    S = H*P_pred*H' + (eye(6)*0.1);
    K = P_pred*H'*inv(S);
    Er_new = Er_pred + K*d_z;
    P_new = (eye(15) - K*H)*P_pred;


end
```