Kalman Filter Equations

Girish Chowdhary

Assistant Professor,
Director Distributed Autonomous Systems Lab
University of Illinois at Urbana Champaign,
www.daslab.illinois.edu

December 5, 2017

Outline

Class outline and outcomes

In this class we will learn about Kalman Filters, the learning outcomes are:

- Origin, need, and applications of Kalman Filters (KF)
- Applications in field robotics
- Get familiar with the terminology, including process noise, measurement noise, predictive covariance, Gaussian white noise etc.
- Mathematically formulate the Kalman filtering problem
- Understand the KF algorithm and how to tune the filter
- Be familiar with advanced filtering techniques when the KF assumptions are violated

What is Kalman filter

- **The filtering problem**: Find the best estimate of the true value of a system's state given noisy measurements of some values of that system's states
- What should a good filter do?
  - Provide an accurate and un-biased estimate
  - Provide confidence in its estimate

What is Kalman filter

- **The filtering problem**: Find the best estimate of the true value of a system's state given noisy measurements of some values of that system's states
- What should a good filter do?
  - Provide an accurate and un-biased estimate
  - Provide confidence in its estimate

The Kalman Filter

The Kalman filter is an optimal estimator for estimating the states of a linear dynamical system from sensor measurements corrupted with Gaussian white noise of some of that system's states.

What is Kalman filter

- **The filtering problem**: Find the best estimate of the true value of a system's state given noisy measurements of some values of that system's states
- What should a good filter do?
  - Provide an accurate and un-biased estimate
  - Provide confidence in its estimate

The Kalman Filter

The Kalman filter is an optimal estimator for estimating the states of a linear dynamical system from sensor measurements corrupted with Gaussian white noise of some of that system's states.

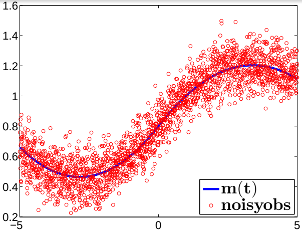


Figure: Noisy data and mean

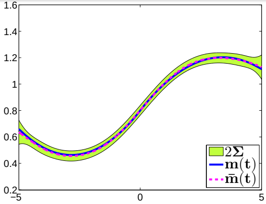


Figure: Estimate of the mean with predictive covariance

Why is the Kalman Filter so popular?

- It Works!

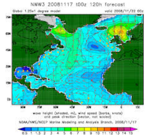
Why is the Kalman Filter so popular?

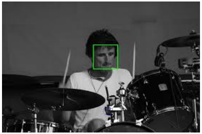
- It Works!
- Well... It works good-enough for many real-world applications
- Why?
  - Sensor noise does tend to be Gaussian in the limit of data received (central limit theorem)
  - Most systems behave linearly in local regions
  - Kalman filter utilizes feedback, which makes it robust to uncertainties
  - Its convenient to implement in an on-line manner to process streaming data
  - Amenable to real-time implementation for many problems

What is a Kalman Filter used for?

- The Kalman filter finds many many applications across pretty much all important scientific disciplines
- Its early application was on trajectory estimation on the Apollo space-craft
- Since then, it has been applied to many dynamical system state estimation problems, including: Cellphone, GPS, weather monitoring, precision agriculture, digital camera, ag-sensors.








Kalman filtering problem setup

- The Kalman filter will return a mean (average) of the quantity being estimated and provide a predictive variance on its estimate given:
  - The process and measurement noise variance is known
  - The dynamical system model is known
- The Kalman filter is guaranteed to be the optimal un-biased estimator for the following case:
  - The noise in the sensors in Gaussian
  - The dynamical system is linear
  - Sufficient number of states of the dynamical system are measured (the system is *observable*)
- If these assumptions are violated, the Kalman filter will be sub-optimal

Outline

- If the system dynamics is Linear and Time-Invariant (LTI), then the state space model is of the form

Noisy continuous-time LTI systems

$$\dot{x} = Ax + B\omega_t \tag{1}$$

$$y = Hx + v_t \tag{2}$$

$x \in \mathbb{R}^{n\times 1}$ state vector
$\omega \in \mathbb{R}^{n\times 1}$ (additive) process noise
$y \in \mathbb{R}^{l\times 1}$ sensor measurements
$v \in \mathbb{R}^{l\times 1}$ (additive) measurement noise

$A \in \mathbb{R}^{n\times n}$ state matrix
$B \in \mathbb{R}^{n\times m}$ the input matrix (here we are using it for inputting noise)
$H \in \mathbb{R}^{l\times n}$ output matrix

Q and R matrices

The assumptions on the process and measurement noise are:

- Zero mean, uncorrelated, i.e. $\omega_k \sim \mathbb{N}(0, \sigma_\omega^2)$, $v_k \sim \mathbb{N}(0, \sigma_v^2)$
- $E[(\omega_t, \omega_s)] = Q\delta(t-s)$, $E[(v_t, v_s)] = R\delta(t-s)$, where δ is the dirac delta function, s.t. $\delta(t-s) = 1$ when $t = s$.
- no cross correlation between ω_k and v_k, i.e. $cov(\omega_k, v_k) = 0$ for all k

Herein lies the "official" definition of Process and Measurement noise. What it is saying is that you can set the diagonal term as σ_ω^2 and σ_v^2

- Practically, Q matrix represents the confidence in the process model, larger the Q matrix, the less confident we are
- Practically, R matrix represents the confidence in the measurements from the correcting sensors, higher the R matrix, the less confident in the measurements we are

- If the system dynamics is Linear and Time-Invariant (LTI), then the state space model is of the form

Noisy discrete-time systems

$$x_{k+1} = \Phi_k x_k + \Gamma_k \omega_k \quad (3)$$

$$y_k = H_k x_k + v_k \quad (4)$$

$x \in \mathbb{R}^{n\times 1}$ state vector
$\omega \in \mathbb{R}^{n\times 1}$ (additive) process noise
$y \in \mathbb{R}^{l\times 1}$ sensor measurements
$v \in \mathbb{R}^{l\times 1}$ (additive) measurement noise

$\Phi_k \in \mathbb{R}^{n\times n}$ discretized state transition matrix
$\Gamma_k \in \mathbb{R}^{n\times m}$ Discretized input matrix
$H_k \in \mathbb{R}^{l\times n}$ output matrix

- The matrices Φ, H are called the system matrices and they depend on the physical parameters of the system such as mass, growth rate...
- note that these are the discrete versions of the equations: $\dot{x} = A(t)x + B(t)u; y = C(t)x$, in MATLAB the command is *c2d*
- In particular, $\Phi_k = e^{A\Delta t}$, $\Gamma_k = B(t)\Delta t$, $H_k = C(t)$, where Δt is the sampling time (dt)
- The process noise ω encodes our uncertainty in the knowledge of the dynamical evolution
- The measurement noise v encodes sensor measurement uncertainty

Relationship between Q and Q_d

Q_d is the discrete time version of Q, remember, $E[(\omega_t, \omega_s)] = Q_k\delta(t-s)$
TO get Q_d we integrate the dynamics in the continuous time case:

$$x(k+1) = \Phi_k x(k) + \int_{t_k}^{t_k+\Delta t} e^{A(t_{k+1}-\lambda)} B(\lambda)\omega(\lambda)d\lambda \tag{5}$$

So we have

$$\omega_k = \int_{t_k}^{t_k+\Delta t} e^{A(t_{k+1}-\lambda)} B(\lambda)\omega(\lambda)d\lambda \tag{6}$$

The exact solution is (assuming white noise process, and computing $Q_{d_k} = cov(\omega_k)$

$$Q_{d_k} = \int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, s)B(s)Q(s)B^T(s)\Phi(t_{k+1}, s)^T ds \tag{7}$$

A good approximation is: $Q_d = BQB^T\Delta T$ this is only good as long as the eigenvalue norm satisfies $\|A\Delta t\|_F << 1$

Covariance matrices

- Let $x = [x(1), x(2), ..., x(n)] \in \mathbb{R}^n$, with $x(i)$ the i^{th} component of x, then the covariance matrix $P \in \mathbb{R}^{n \times n}$ is defined as:

$$\begin{aligned} \mathrm{COV}(x) &\triangleq \mathbf{E}[(x - \bar{x})(x - \bar{x})^T] \\ &= \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} (x - \bar{x})(x - \bar{x})^T dx(1)dx(2) \cdots dx(n) \qquad \triangleq P \end{aligned}$$

- The i^{th} diagonal elements of P are the variance of $x(i)$ given by σ_i^2
- The off-diagonals are the cross-correlations, given by $\sigma_i \sigma_j$
- The covariance matrix is symmetric and positive definite, it can always be diagonalized

Noise properties

Additive process and measurement noise

The *zero-mean additive Gaussian white* *noise assumption*

$\omega_k \sim \mathbb{N}(0, Q_k)$: measurement noise, encodes the uncertainty in the sensors

$v_k \sim \mathbb{N}(0, R_k)$: The process noise, encodes our uncertainty in modeling the process

- Here $Q_k \in \mathbb{R}^{n\times n}$ and $R_k \in \mathbb{R}^{l\times l}$ are positive definite matrices, encoding the process and measurement noise covariances
- Typically sufficient to pick diagonal matrices with positive entries
- The measurement noise R_k (typically stationary: R) is typically provided in the sensor specification sheets, its the variance of the sensor
- Q_k (or when stationary: Q) is a little more difficult to find, typically this is the variable that needs to be *tuned*

$$E[x_{k+1}] = E[\Phi_k x_k + \omega_k] \quad (8)$$
$$E[x_{k+1}] = \Phi_k E[x_k] + 0 \quad (9)$$

Let $\mu_k = E[x_k]$ Covariance propagation

$$P_k = E[(x - \mu_k)(x - \mu_k)^T] \quad (10)$$

Hence

$$\begin{aligned}
P_{k+1} &= E[(x_{k+1} - \mu_{k+1})(x_{k+1} - \mu_{k+1})^T] && (11)\\
&= E[\Phi_k(x_k - \mu_k + \omega_k)(\Phi_k(x_k - \mu_k + \omega_k))^T] && (12)\\
&= E[\Phi_k(x_k - \mu_k)(x_k - \mu_k)^T\Phi_k^T + \omega_k\omega_k^T + \Phi_k(x_k - \mu_k)\omega_k^T && (13)\\
&+ \omega_k(x_k - \mu_k)^T\Phi_k^T]\\
P_{k+1} &= \Phi_k P_k \Phi_k^T + Q_k && (14)
\end{aligned}$$

where $E[w_k w_k^T] = Q_k$ and noting that $E[\omega_k] = 0$

$$\hat{x}_{k+1} = \Phi_k x_k + L_k(y_k - H_k \hat{x}_k^-) \tag{15}$$

Let $e_k^+ = x_k - \hat{x}_k^+$

$$e_k^+ = (I - L_k H_k) e_k^- - L_k v_k \tag{16}$$

From the above and utilizing the predictive error covariance matrix, we get

$$P_k^+ = (I - L_k H_k) P_k^- (I - L_k H_k)^T + L_k R_k L_k^T \tag{17}$$

To compute the optimal gain L_k we minimize $trace(P_k^+)$ wrt L_k. To do this, solve $\frac{\partial trace(P_k^+)}{\partial L_k} = 0$ for L_k Derivation in class

The Kalman Filtering Algorithm

- The Kalman filter has two steps:
- Prediction step:
  - In this step we predict forward the state of the system using our model and Q
  - This is our best guess of what the system state would look like
  - But it will deviate from the true state if the system evolves in a different manner than we expecte
- Correction step: To ensure that our predictions do not drift for too long, the KF utilizes the idea of feedback corrections
  - In this step we correct our predicted state using feedback between predicted measurement and the actual sensor measurement
  - The correction brings our prediction back on track, without having to have information about all the states, or doing it all the time
- Together the predict-correct framework leads to a robust state estimation technique

Derived in class

$$P_k^+ = (I - L_k)P_k^-(I - L_k)^T + L_k R_k L_k^T$$

The optimal Kalman Gain is found by solving: $\min_{L_K} trace(P_k^+)$

$$L_k = P_k^- H_k^T (H_k P_k^- H_k^T + R)^{-1}$$

Kalman filtering algorithm mathematical specifics

Initialize $x_0 \sim \mathbb{N}(0, P_0)$

Prediction step

$$
\begin{aligned}
x_k^- &= \Phi_k x_{k-1} \\
P_k^- &= \Phi_k P_{k-1} \Phi_k^T + Q_k
\end{aligned}
$$

Correction step

$$
\begin{aligned}
e_k &= y_k - H_k x_k^- \\
S_k &= H_k P_k^- H_k^T + R_k \\
L_k &= P_k^- H_k^T S_k^{-1} \\
x_k^+ &= x_k^- + L_k e_k \\
P_k^+ &= P_k^- - L_k S_k L_k^T
\end{aligned}
$$

Kalman filter algorithm

If we assume that Φ_k, H_k, Q_k, R_k do not change, we can re-write the algorithm more simply:

KF algorithm

Prediction step

$$x_k^- = \Phi x_{k-1}$$
$$P_k^- = \Phi P_{k-1} \Phi^T + Q$$

Correction step

$$L_k = P_k^- H^T (H P_k^- H^T + R)^{-1} \tag{18}$$
$$S_k = H P_k^- H^T + R \tag{19}$$
$$x_k = x_k^- + L_k(y_k - H x_k^-) \tag{20}$$
$$P_k = P_k^- - L_k S_k L_k^T \tag{21}$$

- Note that P_k does not depend on x_k, this is a direct consequence of the linearity and Gaussian noise assumption
- This means we can pre-compute K_k off line by iteratively solving (1) and (2) until they converge

Continuous time Kalman Filter

If we assume that A, C, Q, R are continuous-time counterparts:

KF algorithm

Prediction step Initialize $\hat{x}^-(t) = \hat{x}(t=k), P(t) = P_k(t=k)$

$$\dot{\hat{x}}^- = A\hat{x}$$

$$\dot{P}^- = AP + PA^T + Q$$

Correction step

$$P_k^- = P(t=k) \tag{22}$$

$$L_k = P_k^- C^T (CP_k^- C^T + R)^{-1} \tag{23}$$

$$x_k = x_k^- + L(y_k - Cx_k^-) \tag{24}$$

$$P_k = [I - L_k C]P^-(k) \tag{25}$$

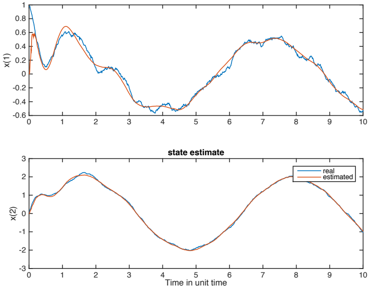
- Note that P_k does not depend on x_k, this is a direct consequence of the linearity and Gaussian noise assumption
- This means we can pre-compute K_k off line by iteratively solving (1) and (2) until they converge

Outline

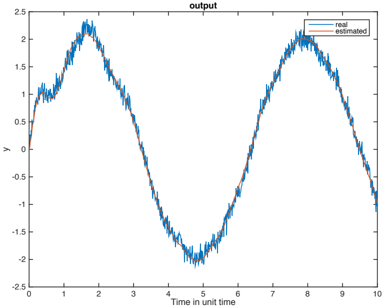
Software example

$$A = \begin{bmatrix} -1 & -5 \\ 6 & -1 \end{bmatrix}$$
$$C = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

- Matlab code `KF_simple.m`
- The measurement noise variance is 0.1 for both states
- The process noise variance is 0.01 for both states



(a) States



(b) Output

More details on the software implementation

- The software implementation use a slightly different notation for ease in variable naming
- x_k^- is termed $\tilde{x}$, P_k^- is termed $\tilde{P}$
- The true state of the system (required for comparison) is simulated, and called x_k, whereas the estimated state is called $\hat{x}_k$ consistent with estimation theory notation
- The system also has a known input $u(t) \in \mathbb{R}$ and is assumed to be continuous, that is:

$$\dot{x} = Ax + Bu + \zeta$$

- With $B = [10]^T$
- The input is sinusoidal, and the system is discretized to work with the framework outlined in class

What if the noise is non-Gaussian or the system is non-linear?

- If the system dynamics are non-linear but sufficiently smooth, then we can try local linearization
- This leads to the Extended Kalman Filter (more about this next week)
- If the dynamics are not sufficiently smooth, or the noise is non-Gaussian, we can utilize Particle Filters (more about this next week)
- The idea here is to create a cloud of particles, transform them through the dynamics and noise, and then re-compute the mean and variance at the other side
- A smart way of doing this is known as the Unscented Kalman Filter (Julier and Uhlmann, 1997)

Extended Kalman Filter

What if the dynamics are nonlinear?

$$\dot{x} = f(x, u) \quad (26)$$
$$y = h(x) \quad (27)$$

The Bayesian inference problem will in general be intractable (remember Champan Komlogrov equation and the challenges with Bayesian inference) Our options:

- Full-blown sample-based Bayesian inference (Markov Chain Monte Carlo (see ABE 598))
- Particle filters: Same as above, but with a specifically selected set of sample "particles"
- Extended Kalman Filter: Linearized filter in a nonlinear setting (as opposed to a fully linearized filter)

EKF algorithm

Initialize $\hat{x}_0 = x(0)$, $P_0 = diag([\sigma^2_{x_1}, \sigma^2_{x_2}, \ldots, \sigma^2_{x_n}])$

Prediction step

$$\hat{x}_k^- = \int_t^{t+\Delta t} f(\hat{x}_{k-1}, u_{k-1})dt$$

$$A_k = \frac{\partial f(x,u)}{\partial x}|_{x=\hat{x}_{k-1}}; \quad \Phi_k = e^{(A_k \Delta t)}$$

$$P_k^- = \Phi_k P_{k-1} \Phi_k^T + Q$$

Correction step

$$y(k) = h(x) \tag{28}$$

$$H_k = \frac{\partial h(x)}{\partial x}|_{x=\hat{x}k-1} \tag{29}$$

$$L_k = P_k^- H^T (H P_k^- H^T + R)^{-1} \tag{30}$$

$$S_k = H P_k^- H^T + R \tag{31}$$

$$x_k = x_k^- + L(y_k - H x_k^-) \tag{32}$$

$$P_k = P_k^- - L S_k L^T \tag{33}$$

Example: GPS-INS integration

States to estimate:

- p: x, y, z position in NED frame (which we are going to assume is inertial)
- v: U, V, W velocities in NED frame
- q: Attitude quaternion (note the discussion on error quaternion in a latter slide)
- b_ω: Bias of the rate gyro
- b_a: Bias of the accelerometer

Using the following measurements:

- $\tilde{a}$: 3 axis accelerometers
- $\tilde{\omega}$: 3 axis gyro
- $\tilde{m}$: 3 axis magnetometer (we won't really use this)
- $\tilde{p}$: position from GPS
- $\tilde{v}$: velocity from GPS

Now we note the kinematic equations of a rigid body moving in 3-dimensional space
They are:

$$^I\dot{\hat{p}} = {}^I\hat{v} \tag{34}$$

$$^I\dot{\hat{v}} = \hat{R}_{b\to I}({}^b a - \hat{b}_a) \tag{35}$$

$$\dot{\hat{q}} = -\frac{1}{2}\Omega(\omega)q \tag{36}$$

$$\dot{\hat{b}}_\omega = 0 \tag{37}$$

$$\dot{\hat{b}}_a = 0 \tag{38}$$

$$\tag{39}$$

$$\Omega(\omega) = \begin{bmatrix} 0 & P & Q & R \\ -P & 0 & -R & Q \\ -Q & R & 0 & -P \\ -R & -Q & P & 0 \end{bmatrix} \tag{40}$$

Inertial measurement models

• Acceleration

$$\tilde{a} = a - b_a \tag{41}$$

• Angular rates, $\omega = [P, Q, R]^T$

$$\tilde{\omega} = \omega - b_\omega \tag{42}$$

So, $P = p - b_{\omega_p}; Q = q - b_{\omega_q}; R = r - b_{\omega_r}$

Let the quaternion denoting the rotation from body to inertial frame $q = [q_1, q_2, q_3, q_4]$, where q_1 is the scalar part denoting the rotation and $[q_2, q_3, q_4]$ is the axis of rotation

$$R_{I\rightarrow b} = \left[\begin{array}{ccc} q_1^2+q_2^2-q_3^2-q_4^2 & 2(q_2q_3+q_1q_4) & 2(q_2q_4-q_1q_3) \\ 2(q_2q_3-q_1q_4) & q_1^2-q_2^2+q_3^2-q_4^2 & 2(q_3q_4+q_1q_2) \\ 2(q_2q_4+q_1q_3) & 2(q_3q_4-q_1q_2) & q_1^2-q_2^2-q_3^2+q_4^2 \end{array} \right] \tag{43}$$

And,

$$R_{b\rightarrow I} = R_{I\rightarrow b}^T \tag{44}$$

•Don't forget to normalize the quaternion in your code every time-step, you do this by setting $q \leftarrow \frac{q}{norm(q)}$

Linearization

$$F_{pv} = \frac{\partial^I \dot{\hat{p}}}{\partial^I \hat{v}} = I_{3\times 3} \tag{45}$$

$$F_{vq} = \frac{\partial^I \dot{\hat{v}}}{\partial^I \hat{q}} = \frac{\partial R_{b\to I}}{\partial q} {}^b a \tag{46}$$

$$F_{vb_a} = \frac{\partial^I \dot{\hat{v}}}{\partial^I \hat{b}_a} = -R_{b\to I} \tag{47}$$

$$F_{qq} = \frac{\partial^I \dot{\hat{q}}}{\partial^I \hat{q}} = -\frac{1}{2}\Omega(\omega) \tag{48}$$

$$F_{qb_\omega} = \frac{\partial^I \dot{\hat{q}}}{\partial \hat{b}_\omega} \tag{49}$$

Let us look at some of these equations in more detail

$$\frac{\partial^I \dot{\hat{v}}}{\partial^I \hat{q}} = \frac{\partial R_{b \to I}}{\partial q} {}^I a \tag{50}$$

$$\frac{\partial^I \dot{\hat{v}}}{\partial^I \hat{q}} = \begin{bmatrix} 2(q_1 a_x - q_4 a_y + q_3 a_z) & 2(q_2 a_x + q_3 a_y + q_4 a_z) & 2(-q_3 a_x + q_2 a_y + q_1 a_z) & 2(-q_4 a_x - q_1 a_y + q_2 a_z) \\ 2(q_4 a_x + q_1 a_y - q_2 a_z) & 2(q_3 a_x - q_2 a_y - q_1 a_z) & 2(q_2 a_x + q_3 a_y + q_4 a_z) & 2(q_1 a_x - q_4 a_y + q_3 a_z) \\ 2(-q_3 a_x + q_2 a_y + q_1 a_z) & 2(q_4 a_x + q_1 a_y - q_2 a_z) & 2(-q_1 a_x + q_4 a_y - q_3 a_z) & 2(q_2 a_x + q_3 a_y + q_4 a_z) \end{bmatrix} \tag{51}$$

$$\frac{\partial^I \dot{\hat{q}}}{\partial \hat{b}_\omega} = -0.5 \frac{\partial \Omega(\omega) q}{\partial b_\omega} \tag{52}$$

$$= \frac{1}{2} \begin{bmatrix} q_2 & q_3 & q_4 \\ -q_1 & q_4 & -q_3 \\ -q_4 & -q_1 & q_2 \\ q_3 & -q_2 & -q_1 \end{bmatrix} \tag{53}$$

Putting together the linearized transition matrix

Here Z is a zero matrix, and I is the identity matrix

$$A = \begin{bmatrix} Z(3\times 3) & I(3\times 3) & Z(3\times 4) & Z(3\times 3) & Z(3\times 3) \\ Z(3\times 3) & Z(3\times 3) & F_{vq} & Z(3\times 3) & F_{vb_a} \\ Z(4\times 3) & Z(4\times 3) & F_{qq} & F_{qb_\omega} & Z(4\times 3) \\ Z(3\times 3) & Z(3\times 3) & Z(3\times 4) & Z(3\times 3) & Z(3\times 3) \\ Z(3\times 3) & Z(3\times 3) & Z(3\times 4) & Z(3\times 3) & Z(3\times 3) \end{bmatrix} \tag{54}$$

•Don't forget to discretize A. If you want to use continuous A, i.e. 54 then use $\dot{P} = AP + PA^T + Q$

Our measurements are GPS position and velocity, so $z = [x, y, z, v_x, v_y, v_z]^T$ Where z contains positions and velocities at the CG. However, the GPS is mounted offset from the CG (this is very important) at the location r_{GPS}, where r is in the body fixed frame, so

$$p_{CG} = p_{GPS} - R_{b\to I} r_{GPS} \tag{55}$$

$$v_{CG} = v_{GPS} - R_{b\to I} \omega \times r_{GPS} \tag{56}$$

Remember $R_{b\to I}$ is a function of the quaternions, so we need to linearize this to get our H matrix. Now $r_{GPS} = [1.5, 0, 0]^T$ for the dataset we are using. So we can ignore the second and third column of $R_{b\to I}$

Linearizing the measurement model

$$H_{xq} = \begin{bmatrix} -r_{GPS}(1)2q_1 & -r_{GPS}(1)2q_2 & r_{GPS}(1)2q_3 & r_{GPS}(1)2q_4 \\ -r_{GPS}(1)2q_4 & -r_{GPS}(1)2q_3 & -r_{GPS}(1)2q_2 & -r_{GPS}(1)2q_1 \\ r_{GPS}(1)2q_3 & -r_{GPS}(1)2q_4 & r_{GPS}(1)2q_1 & -r_{GPS}(1)2q_2 \end{bmatrix} \tag{57}$$

$$H_{vq} = \begin{bmatrix} r_{GPS}(1)2q_3Q + r_{GPS}(1)2q_4R & r_{GPS}(1)2q_4Q - r_{GPS}(1)2q_3R & r_{GPS}(1)2q_1Q - r_{GPS}(1)2q_2R & r_{GPS}(1)2q_2Q + r_{GPS}(1)2q_1R \\ -r_{GPS}(1)2q_2Q - r_{GPS}(1)2q_1R & r_{GPS}(1)2q_2R - r_{GPS}(1)2q_1Q & r_{GPS}(1)2q_4Q - r_{GPS}(1)2q_3R & r_{GPS}(1)2q_3Q + r_{GPS}(1)2q_4R \\ r_{GPS}(1)2q_1Q - r_{GPS}(1)2q_2R & -r_{GPS}(1)2q_2Q - r_{GPS}(1)2q_1R & -r_{GPS}(1)2q_3Q - r_{GPS}(1)2q_4R & r_{GPS}(1)2q_4Q - r_{GPS}(1)2q_3R \end{bmatrix} \tag{58}$$

$$H = \begin{bmatrix} I(3\times 3) & Z(3\times 3) & H_{xq} & Z(3\times 6) \\ Z(3\times 3) & I(3\times 3) & H_{vq} & Z(3\times 6) \end{bmatrix} \tag{59}$$

Error Quaternion

Quaternion: $q = [q_0, q_1, q_2, q_3]$ Define an error quaternion: $\delta q = [1, s]^T$, such that

$$\delta q \circ \hat{q} = q \tag{60}$$

In practice, this error quaternion takes on very small values, so its ok to say $\hat{s} = 0$ If we do this, we can simplify some calculations. See JFR paper by Chowdhary et al.

HW 3 Q2

- To solve Q2, you need to simulate 2 different systems
- The first without noise, with the real a
- The second, with noise, with $\tilde{u}$

$$\begin{bmatrix} \dot{p} \\ \dot{v} \\ \dot{b} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} p \\ v \\ b \end{bmatrix} \tag{61}$$

HW3 Q3

Now you have a measurement, and it is of the form

$$y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} p \\ v \\ b \end{bmatrix} \tag{62}$$

When you do the correction step, make sure you do it every second.
The prediction should happen as fast as the inertial sensors give you data, assume 100Hz.

HW3 Q4

The states are now $[x, y, \theta, v, b]$ The linearized matrix is (before discretization)

$$A = \begin{bmatrix} 0 & 0 & -v\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & v\cos(\theta) & \sin(\theta) & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{63}$$

For the prediction step, integrate the continuous nonlinear dynamics first
For the correction step use the measurement matrix:

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \\ v \\ b \end{bmatrix} \tag{64}$$

The third measurement is the encoder

Observability

Observability tells us whether the measurements available (i.e. y) are sufficient to reconstruct the state (x).
For linear time invariant systems, this boils down to a simple condition on the matrix pair (C, A):

Observability condition

The pair (C, A) is observable if and only if the matrix $\mathscr{O}$ has full column rank

$$\mathscr{O} = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{65}$$

Example of observability

Consider the HW system from Q3, the linearized matrices are given in 63 and the measurement matrix: 64
Let us use MATLAB to check the observability condition, try it for the following cases:

- $\theta = 45^0$, x, y measured
- $\theta = 45^0$, x, y, v measured
- $\theta = 0^0$, x, y measured
- $\theta = 90^0$, x, y measured

Example of observability

Consider the HW system from Q3, the linearized matrices are given in 63 and the measurement matrix: 64
Let us use MATLAB to check the observability condition, try it for the following cases:

- $\theta = 45^0$, x, y measured
- $\theta = 45^0$, x, y, v measured
- $\theta = 0^0$, x, y measured
- $\theta = 90^0$, x, y measured

Further Reading

- Gelb Arthur, **Applied Optimal Estimation**, MIT Press, Chapter 4: Optimal Linear Filtering
- Slides on the Bayesian derivation of the Kalman Filtering equations by Simo Särkkä: http://becs.aalto.fi/~ssarkka/course_k2012/handout3.pdf
- Simo's book: **Bayesian Filtering and Smoothing**, Chapte 4, available online: http://becs.aalto.fi/~ssarkka/pub/cup_book_online_20131111.pdf
- Christophersen, Henrik B., et al. "**A compact guidance, navigation, and control system for unmanned aerial vehicles.**" Journal of aerospace computing, information, and communication 3.5 (2006): 187-213.
- Chowdhary, Girish, et al. "**GPS denied Indoor and Outdoor Monocular Vision Aided Navigation and Control of Unmanned Aircraft.**" Journal of Field Robotics 30.3 (2013): 415-438.

Other notes

- Useful to readup the Bayesian derivation of the Kalman filter from Simmi Sarakka's notes