Estimation for Quadrotors

Stefanie Tellex
Brown University

Andy Brown
Udacity, Inc.

Sergei Lupashin
Fotokite

July 7, 2018

This document describes standard approaches for filtering and estimation for quadrotors, created for the Udacity Flying Cars course. We assume previous knowledge of probability and some knowledge of linear algebra. We do not assume previous knowledge of Kalman filters or Bayes filters. This document derives an EKF for various models of drones in 1D, 2D, and 3D. We use the EKF and notation as defined in Thrun et al. [13]. We also give pseudocode for the Bayes filter, the EKF, and the Unscented Kalman filter [14]. The motivation behind this document is the lack of a step-by-step EKF tutorial that provides the derivations for a quadrotor helicopter. The goal of estimation is to infer the drone's state (pose, velocity, accelleration, and biases) from its sensor values and control inputs. This problem is challenging because sensors are noisy. Additionally, because of weight and cost issues, many drones have limited on-board computation so we want to estimate these values as quickly as possible. The standard method for performing this method is the Extended Kalman filter, a nonlinear extension of the Kalman filter which linearizes a nonlinear transition and measurement model around the current state. However the Unscented Kalman filter is better in almost every respect: simpler to implement, more accurate to estimate, and comparable runtimes.

1 Averaging

When performing estimation, the first thing one might think of is averaging the sensor measurements. We consider averaging for the 1D case, where the range sensor value, z_t is directly observing the state (the height of the drone), x_t, both of which are scalars. The notation $\hat{x_t}$ means our estimate of the true state x_t. Then the average is:

$$\hat{x_t} = \frac{1}{t}\Sigma_0^t z_t \tag{1}$$

However this form requires saving all sensor measurements from the beginning of time, requiring memory and computation that grows linearly in the number of observations. Instead we rewrite the update recursively in terms of our previous estimate, $\hat{x}_{t-1}$, allowing us to transform $\hat{x}_{t-1}$ and z_t to $\hat{x}$:

$$\hat{x_t} = \frac{1}{t}\left[\hat{x}_{t-1} \times (t-1) + z_t\right]. \tag{2}$$

The recursive form, where the next estimate is written in terms of the previous estimate, allows us to perform a constant time update, and requires us to store a constant amount of information about past sensor values (only the previous estimate, $\hat{x}_{t-1}$ - in other words, our estimate is Markov). However this form updates slowly when the drone's state changes. Imagine the drone is stationary for one minute, and then starts moving. Then $\frac{1}{t}$ will be very small, and the estimate will take a long time to move to the new value (longer and longer, the longer the drone is running). Instead, we want an average that moves more quickly. One way to achieve a faster updating average is to simply average the old estimate with the new sensor value:

$$\hat{x_t} = \frac{\hat{x}_{t-1} + z_t}{2} \tag{3}$$

A more general formulation is a weighted average of the old value and the new observation:

$$\hat{x_t} = (\alpha)\hat{x}_{t-1} + (1-\alpha)z_t \tag{4}$$

This formulation allows the designer to tune the parameter α, which weights how much to weight the old estimate compared to the new sensor value.

2 Bayes Filter

None of the above formulations update the estimate based on the drone's movements. Intuitively, if we have told the drone to go up, for instance, then our belief about the drone's position should also go up. The Bayes Filter gives us a way to incorporate motion prediction into our state estimate. First we *predict* the next state, given a control input, the current state, and a model of how the system evolves over time. We do not maintain a point estimate but rather a belief or distribution of the estimate. Then, we revise our prediction with an *update* from an observation. The update method takes the previous estimate from prediction, and an observed sensor value. It returns a new distribution that takes into account the sensor value, using a model of how the sensor works. Below you will see pseudocode for the prediction and update steps for the filter, following Thrun et al. [13]. In typical use one would call predict after determining the control input, and update after reading a sensor value. The BayesFilter function is illustrative only; in real life, one might call predict in the sensor callback for example.

Algorithm 1 General Bayes Filter algorithm. For specific filters such as the Kalman Filter or the particle filter, the representation for bel and $\bar{bel}$ changes, and the corresponding mathematical updates take specific computational forms.

function Predict($bel(x_{t-1}), u_t, \Delta t$)
  $\bar{bel}(x_t) = \int p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1}$
  return $\bar{bel}(x_t)$
function Update($\bar{bel}(x_t), z_t$)
  $bel(x_t) = \eta p(z_t|x_t) bel(x_t)$
  return $bel(x_t)$
function BayesFilter
  $u_t = \text{ComputeControl}(bel(x_{t-1}))$
  $\bar{bel}(x_t) = \text{Predict}(bel(x_{t-1}), u_t, \Delta t)$
  $z_t = \text{ReadSensor}()$
  $bel(x_t) = \text{Update}(\bar{bel}(x_t), z_t)$

3 E(KF)

The Kalman Filter and Extended Kalman Filter make the assumption that the distributions over belief state are Gaussian, represented as a mean and covariance matrix. Compare to the Bayes' filter, the distributions bel and $\bar{bel}$ are represented as a mean and covariance matrix. The KF assumes that the transition and observation models are linear, and can be defined by a matrix. The EKF is the extension to the nonlinear case, where we take use the Jacobian matrix of the transition and observation functions to compute a point-wise linear estimate, and then do the same updates as the Kalman Filter. We define the Extended Kalman Filter (EKF) algorithm following Thrun et al. [13]. We refactor it to include separate Predict and Update methods and to use our notation. We also unify the KF and EKF algorithm pseudocode. The transition or prediction covariance is Q_t; the measurement covariance is R_t. These matrices are often taken to be constant, but also sometimes people change them over time depending on the sensor model.

The EKF and KF are closely related. For the KF, the matrix G_t is constant every iteration of the function and does not need to be recomputed each time (except for Δt). Another way to say it is the implementation of g' ignores its state input. Similarly, for the KF, the matrix H_t is constant every iteration of the function and does not need to be recomputed. Another way to say it is the function h' ignores its state input. For the EKF, these matrices change each iteration, because it linearizes around the current state.

Algorithm 2 E(KF) algorithm.

function PREDICT($\mu_{t-1}, \Sigma_{t-1}, u_t, \Delta t$)
  $\bar{\mu}_t = g(u_t, \mu_{t-1})$
  $G_t = g'(u_t, x_t, \Delta t)$
  $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + Q_t$
  return $\bar{\mu}_t, \bar{\Sigma}_t$
function UPDATE($\bar{\mu}_t, \bar{\Sigma}_t, z_t$)
  $H_t = h'(\bar{\mu}_t)$
  $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + R_t)^{-1}$
  $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
  $\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$
  return μ_t, Σ_t
function EXTENDEDKALMANFILTER
  $u_t = \text{COMPUTECONTROL}(\mu_{t-1}, \Sigma_{t-1})$
  $\bar{\mu}_t, \bar{\Sigma}_t = \text{PREDICT}(\mu_{t-1}, \Sigma_{t-1}, u_t, \Delta t)$
  $z_t = \text{READSENSOR}()$
  $\mu_t, \Sigma_t = \text{UPDATE}(\bar{\mu}_t, \bar{\Sigma}_t, z_t)$

4 Unscented Kalman Filter

The Unscented Kalman Filter [14] is similar to the EKF in that it handles nonlinear transition models g and measurement models h. However there are no Jacobians! Instead of linearizing around the current estimate, the Unscented Kalman Filter picks magic "sigma points" which are sample points chosen according to the current state estimate and covariance. Then these sigma points are passed through the nonlinear transition or observation function, and the sample mean and sample covariance of the sigma points is used to construct a new Gaussian μ and σ. The pseudocode here follows Kandepu et al. [9] but removes the augmentation for tracking moving prediction and measurement covariance, and uses our notation.

Sigma points are computed using the matrix S which is defined from the covariance matrix, Σ_t. S_i denotes the ith colum of the matrix S.

$$S = \sqrt{\Sigma_t} \tag{5}$$

We define the sigma points $X_{i,t} \in X_t$ as follows:

$$X_{i,t} = \begin{cases} = \mu_t, & i = 0 \\ = \mu_t + \gamma S_i, & i = 1, \ldots, N \\ = \mu_t - \gamma S_{i-N}, & i = N+1, \ldots, 2N \end{cases} \tag{6}$$

Note that they are defined using the mean and covariance matrix of the distributions; by picking several representative points S_i away from the mean, we can use a relatively small set of points to represent the entire distribution. The weights when computing the sample mean, w_i^m are:

$$w_i^m = \begin{cases} = \frac{\lambda}{N+\lambda} & i = 0 \\ = \frac{1}{2(N+\lambda)}, & i = 1, \ldots, 2N \end{cases} \tag{7}$$

The weights when computing the sample covariance, w_i^c are:

$$w_i^c = \begin{cases} = \frac{\lambda}{N+\lambda} + (1 - \alpha^2 + \beta) & i = 0 \\ = \frac{1}{2(N+\lambda)}, & i = 1, \ldots, 2N \end{cases} \tag{8}$$

The parameters are defined as:

$$\gamma = \sqrt{N + \lambda} \tag{9}$$

$$\lambda = \alpha^2 (N + \kappa) - N \tag{10}$$

See Kandepu et al. [9] for tuning suggestions when implementing the filter.

The pseudocode for the Unscented Kalman Filter is given in Algorithm 3. Compared to the Bayes' filter, we use the computed sigma points to represent the distribution $\bar{bel}$. However we use a mean and covariance to represent the distribution *bel* at the beginning and end of each iteration of the filter.

Algorithm 3 Unscented Kalman Filter.

function ComputeSigmas(μ_t, Σ_t)
  return $X_{0,t}, \ldots, X_{2N,t}$ following Equation 6.
function Predict($\mu_{t-1}, \Sigma_{t-1}, u_t, \Delta t$)
  $X_{t-1} = \text{ComputeSigmas}(\mu_{t-1}, \Sigma_{t-1})$
  $\forall_{i=0}^{2N} \bar{X}_{i,t} = g(X_{i,t-1}, u_t, \Delta t)$
  return $\bar{X}_t$
function Update($\bar{X}_t, z_t$)
  $\bar{\mu}_t = \sum_{i=0}^{2N} (w_i^m \bar{X}_{i,t})$
  $\bar{\Sigma}_t = \sum_{i=0}^{2N} w_i^c (X_{i,t} - \bar{\mu}_t)(X_{i,t} - \bar{\mu}_t)^T + Q_t$
  $\forall_{i=1}^{2N} Z_{i,t} = h(\bar{X}_{i,t})$
  $\mu^z = \Sigma_{i=0}^{2N} w_i^m Z_{i,t}$
  $\Sigma_t^z = \Sigma_{i=0}^{2N} w_i^c (Z_{i,t} - \mu^z)(Z_{i,t} - \mu^z)^T + R_t$
  $\Sigma_t^{xz} = \Sigma_0^{2N} w_i^c (\bar{X}_{i,t} - \bar{\mu}_t)(Z_{i,t} - \mu^z)^T$
  $K_t = \Sigma_t^{xz} (\Sigma_t^z)^{-1}$
  $\mu_t = \bar{\mu}_t + K_t (z_t - \mu^z)$
  $\Sigma_t = \bar{\Sigma}_t - K_t \Sigma_t^z K_t^T$
  return μ_t, Σ_t
function UnscentedKalmanFilter
  $u_t = \text{ComputeControl}(\mu_{t-1}, \Sigma_{t-1})$
  $\bar{X}_t = \text{Predict}(\mu_{t-1}, \Sigma_{t-1}, u_t, \Delta t)$
  $z_t = \text{ReadSensor}()$
  $\mu_t, \Sigma_t = \text{Update}(\bar{X}_t, z_t)$

5 One Dimensional Quad

To implement the filters on specific vehicles, we need to define the state transition function, g and the measurement model, h. We will do this three times for increasingly more realistic models of a quadrotor. First, we define a 1D quad model, where the quadroter is moving in z, but not in x or y. It has one control input, the downward pointing thrust, and a noisy range sensor. The intention is that this is identical to the 1D quad used in the controls lesson. The state is then the position, z and velocity, $\dot{z}$:

$$x_t = \begin{bmatrix} \dot{z} \\ z \end{bmatrix} \tag{11}$$

We define the control input as directly setting the accelleration, $\ddot{z}$:

$$u_t = \begin{bmatrix} \ddot{z} \end{bmatrix} \tag{12}$$

5.1 Transition Model

Then we define a transition function $g(x_t, u_t, \Delta t)$ which returns a new x_{t+1}:

$$g(x_t, u_t, \Delta t) = \begin{bmatrix} x_{t,\dot{z}} + u_{t,\ddot{z}} \times \Delta t \\ x_{t,z} + x_{t,\dot{z}} \times \Delta t \end{bmatrix} \tag{13}$$

$$= \begin{bmatrix} 1 & 0 \\ \Delta t & 1 \end{bmatrix} \begin{bmatrix} \dot{z} \\ z \end{bmatrix} + \begin{bmatrix} \Delta t \\ 0 \end{bmatrix} \begin{bmatrix} \ddot{z} \end{bmatrix} \tag{14}$$

We can rewrite it in terms of the A_t and B_t matrix, as in a conventional Kalman filter. In this form we see that the control update is linear because it can be written in this form.

$$= A_t x_t + B_t u_t \tag{15}$$

Then $x_t \in \mathbb{R}^2$ and $g(x_t, u_t, \Delta t) \in \mathbb{R}^2$. So $g'(x_t, u_t)$ is a 2×2 matrix, defined as the partial derivative of g with respect to x_t for each component in x_t.

$$g'(x_t, u_t, \Delta t) = \begin{bmatrix} \frac{\partial}{\partial x_{t,\dot{z}}} g_{\dot{z}}(x_t, u_t, \Delta t) & \frac{\partial}{\partial x_{t,z}} g_{\dot{z}}(x_t, u_t, \Delta t) \\ \frac{\partial}{\partial x_{t,\dot{z}}} g_z(x_t, u_t, \Delta t) & \frac{\partial}{\partial x_{t,z}} g_z(x_t, u_t, \Delta t) \end{bmatrix} \tag{16}$$

$$= \begin{bmatrix} 1 & 0 \\ \Delta t & 1 \end{bmatrix} \tag{17}$$

$$\tag{18}$$

Since this function is linear, the Jacobian is a constant matrix except for Δt, and just reduces to the A_t matrix.

5.2 Measurement Model

Next we assume the drone has a range sensor pointed downwards at the ground, at $z = 0$. Then z_t is the range value, r:

$$z_t = \begin{bmatrix} r \end{bmatrix} \tag{19}$$

Then we define a measurement function $h(x_t)$ which returns a new z_t:

$$h(x_t) = \begin{bmatrix} x_{t,z} \end{bmatrix} \tag{20}$$

$$= \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{z} \\ z \end{bmatrix} \tag{21}$$

$$= C_t \begin{bmatrix} \dot{z} \\ z \end{bmatrix} \tag{22}$$

$$= C_t x_t \tag{23}$$

Finally we define $h'(x_t)$, the Jacobian of h with respect to x_t. The Jacobian is a 1×2 matrix.

$$h'(x_t) = \begin{bmatrix} \frac{\partial}{\partial x_{t,\dot{z}}} h_r(x_t) & \frac{\partial}{\partial x_{t,z}} h_r(x_t) \end{bmatrix} \tag{24}$$

$$= \begin{bmatrix} 0 & 1 \end{bmatrix} \tag{25}$$

Note that in this case the Jacobian does not depend at all on the input x_t. This is because this system is linear, so the EKF linearization will boil back down to a regular Kalman filter.

6 Two Dimensional Quad

We define a 2D quad model where the quadrotor is operating at a fixed height z. It has a range sensor pointing sideways, and it can move by rotating about its center, the angle ϕ. Additionally it can move right and left in y. Then we define the state x_t as the state transition function as follows:

$$x_t = \begin{bmatrix} \phi \\ \dot{y} \\ y \end{bmatrix} \tag{26}$$

We define the control input as directly setting the angle, ϕ.

$$u_t = \begin{bmatrix} \phi \end{bmatrix} \tag{27}$$

6.1 Transition Model

Then we define $g(x_t, u_t, \Delta t)$ and returns a new x_{t+1}:

$$g(x_t, u_t, \Delta t) = \begin{bmatrix} u_{t,\phi} \\ x_{t,\dot{y}} - \sin(x_{t,\phi}) \times \Delta t \\ x_{t,y} + x_{t,\dot{y}} \times \Delta t \end{bmatrix} \tag{28}$$

Then $x_t \in \mathbb{R}^3$ and $g(x_t, u_t, \Delta t) \in \mathbb{R}^3$. So $g'(x_t, u_t)$ is a 3×3 matrix.
Finally, we take the partial derivative of g with respect to x_t for each component in x_t.

$$g'(x_t, u_t, \Delta t) = \begin{bmatrix} \frac{\partial}{\partial x_{t,\phi}} g_\phi & \frac{\partial}{\partial x_{t,\dot{y}}} g_\phi & \frac{\partial}{\partial x_{t,y}} g_\phi \\ \frac{\partial}{\partial x_{t,\phi}} g_{\dot{y}} & \frac{\partial}{\partial x_{t,\dot{y}}} g_{\dot{y}} & \frac{\partial}{\partial x_{t,y}} g_{\dot{y}} \\ \frac{\partial}{\partial x_{t,\phi}} g_y & \frac{\partial}{\partial x_{t,\dot{y}}} g_y & \frac{\partial}{\partial x_{t,y}} g_y \end{bmatrix} \tag{29}$$

$$= \begin{bmatrix} \frac{\partial}{\partial x_{t,\phi}} u_{t,\phi} & \frac{\partial}{\partial x_{t,\dot{y}}} u_{t,\phi} & \frac{\partial}{\partial x_{t,y}} u_{t,\phi} \\ \frac{\partial}{\partial x_{t,\phi}} x_{t,\dot{y}} - \sin(x_{t,\phi}) \times \Delta t & \frac{\partial}{\partial x_{t,\dot{y}}} x_{t,\dot{y}} - \sin(x_{t,\phi}) \times \Delta t & \frac{\partial}{\partial x_{t,y}} x_{t,\dot{y}} - \sin(x_{t,\phi}) \times \Delta t \\ \frac{\partial}{\partial x_{t,\phi}} x_{t,y} + x_{t,\dot{y}} \times \Delta t & \frac{\partial}{\partial x_{t,\dot{y}}} x_{t,y} + x_{t,\dot{y}} \times \Delta t & \frac{\partial}{\partial x_{t,y}} x_{t,y} + x_{t,\dot{y}} \times \Delta t \end{bmatrix} \tag{30}$$

$$= \begin{bmatrix} 0 & 0 & 0 \\ -\cos(x_{t,\phi})\Delta t & 1 & 0 \\ 0 & \Delta t & 1 \end{bmatrix} \tag{31}$$

6.2 Measurement Model

Next we assume the drone has a range sensor pointed sideways at a $\text{wall}_y = 5$. Then z_t is the range value, r:

$$z_t = \begin{bmatrix} r \end{bmatrix} \tag{32}$$

Then we define $h(x_t)$ and returns a new z_t:

$$h(x_t) = \begin{bmatrix} \frac{\text{wall}_y - x_{t,y}}{\cos(x_{t,\phi})} \end{bmatrix} \tag{33}$$

Finally we define $h'(x_t)$, the Jacobian of h with respect to x_t.

$$h'(x_t) = \begin{bmatrix} \frac{\partial}{\partial x_{t,\phi}} h_r(x_t) & \frac{\partial}{\partial x_{t,\dot{y}}} h_r(x_t) & \frac{\partial}{\partial x_{t,y}} h_r(x_t) \end{bmatrix} \tag{34}$$

$$= \begin{bmatrix} \frac{[\text{wall}_y - x_{t,y}]\sin(x_{t,\phi})}{\cos^2(x_{t,\phi})} & 0 & \frac{-1}{\cos(x_{t,\phi})} \end{bmatrix} \tag{35}$$

Using the above math, we have implemented an EKF in Python for this model, and showed it is able to estimate position and velocity using the drone's simulated noisy range sensor.

7 Three Dimensional Quad

The state will be position and velocity which we will estimate with the GPS. We will use the magnetometer to estimate yaw. We will represent position and velocity in the global frame, and yaw that goes from body to local in the state. We would also then use a multirate Kalman Filter [4, 12] to perform updating, with separate updates for the GPS, the magnetometer, and the IMU (accelerometer plus rate gyro).

We track yaw as the heading from magnetic north. So it is the reading one would get if one reads from a compass.

Variable	Description
u_t	The control input at time t.
x_t	The state at time t.
z_t	The observation at time t.
$x_{t,\text{variable}}$	The scalar value of the state vector at the index corresponding to variable. Similar notation for u_t and z_t.
Δt	The elapsed time between updates in seconds.
Q_t	Transition model covariance
R_t	Measurement model covariance
ϕ	Roll
θ	Pitch
ψ	Yaw
R_{bg}	Rotation matrix from body to global
R_{gy}	Rotation matrix from global to yaw frame
x^y	the x coordinate in the yaw frame.
x	The x coordinate in the global frame.
x^b	The x coordinate in the body frame.
$g(x_t, u_t, \Delta t)$	The transition function.
$h(x_t)$	The observation function.

Table 1: Table of variables.

$$x_t = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \psi \end{bmatrix} \tag{36}$$

Then u_t is the acceleration in the body frame, where $\dot{\psi}$ is global frame yaw.

$$u_t = \begin{bmatrix} \ddot{x}^b \\ \ddot{y}^b \\ \ddot{z}^b \\ \dot{\psi} \end{bmatrix} \tag{37}$$

7.1 Attitude Filter

Markley [10] gives an EKF in quaternions for attitude estimation. Higgins [7] compares complementary filters to Kalman filters, in a way not specific to quads or attitude estimation. Quan [12] says to use either a complementary filter or Kalman filter for attitude estimation and gives very terse, hard to understand math. Johansen and Kristiansen [8] describes the MEKF model used in a quadrotor with adaptive fading. Crassidis et al. [3] gives a survey of nonlinear attitude estimation including the MEKF. Nowicki et al. [11] compares the complementary filter with an EKF for attitude estimation on mobile phones. They find that the complementary filter is simpler to implement, but the EKF is able to achieve in most cases better accuracy. Both have comparable processor loads.

We assume the state we are tracking is the vehicle's attitude, that is roll ϕ and pitch, θ. Then the obser-

vation, z_t consists of the gyro angular velocity and pitch and roll angles as estimated from the accelerometer.

$$z_t = \begin{bmatrix} \theta \\ \phi \\ p \\ q \end{bmatrix} \tag{38}$$

The accelerometer estimates for θ and ϕ are in the global frame, but the velocities from the gyro are in the body frame. Our approach follows the Linear Complementary Filter from Quan [12]. We assume that θ and ϕ are small, so that the turn rates measured by the gyro in the body frame approximate the global turn rates. In other words,

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \approx \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{39}$$

The state is the roll angle and pitch angle:

$$x_t = \begin{bmatrix} \theta \\ \phi \end{bmatrix} \tag{40}$$

7.1.1 Linear Complementary Filter

We define a linear complementary filter following Quan [12]. Here τ is a time constant and T_s is the filter sampling period:

$$\hat{\theta}_t = \frac{\tau}{\tau + T_s}\left(\hat{\theta}_{t-1} + T_s z_{t,\dot{\theta}}\right) + \frac{T_s}{\tau + T_s} z_{t,\theta} \tag{41}$$

Similarly for roll:

$$\hat{\phi}_t = \frac{\tau}{\tau + T_s}\left(\hat{\phi}_{t-1} + T_s z_{t,\dot{\phi}}\right) + \frac{T_s}{\tau + T_s} z_{t,\phi} \tag{42}$$

We do not estimate the yaw with a complementary filter because we will use the magnetometer and do it in the GPS.

The above math assumes that the angular velocity (which is in body frame) can be used directly as angular rotation.

7.1.2 Nonlinear Complementary Filter

For the nonlinear complementary filter, following Quan [12] Section 9.1.3, we use the state to define a quaternion, q_t, for the euler angles for ϕ, θ and ψ. Then we can define dq to be the quaternion that consists of the measurement of the angular rates from the IMU in the body frame, following Equation 84 in Diebel [5]. Using these two, we can define a predicted quaternion, $\bar{q}_t$ as follows:

$$\bar{q}_t = dq * q_t \tag{43}$$

Finally we can define $\bar{\theta}_t$ and $\bar{\phi}_t$ as follows:

$$\bar{\theta}_t = Pitch(\bar{q}_t) \tag{44}$$

$$\bar{\phi}_t = Roll(\bar{q}_t) \tag{45}$$

Using these predicated estimates, we can compute the non-linear complementary filter as above.

$$\hat{\theta}_t = \frac{\tau}{\tau + T_s}\left(\bar{\theta}_{t-1} + T_s z_{t,\dot{\theta}}\right) + \frac{T_s}{\tau + T_s} z_{t,\theta} \tag{46}$$

Similarly for roll:

$$\hat{\phi}_t = \frac{\tau}{\tau + T_s}\left(\bar{\phi}_{t-1} + T_s z_{t,\dot{\phi}}\right) + \frac{T_s}{\tau + T_s} z_{t,\phi} \tag{47}$$

7.2 Transition Model

We define the transition function in terms of the rotation matrix R_{bg} which rotates from the body frame to the global frame. As described in Diebel [5], there are 12 different orders one could perform the rotation; we follow the convention from aerospace of using the $1, 2, 3$ order for roll, pitch, and yaw.

This matrix is defined as follows, taken from the transpose (or inverse) of Diebel [5], equation 67.

$$R_{bg} = \begin{bmatrix} \cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix} \tag{48}$$

Then the transition function is:

$$g(x_t, u_t, \Delta t) = \begin{bmatrix} x_{t,x} + x_{t,\dot{x}}\Delta t \\ x_{t,y} + x_{t,\dot{y}}\Delta t \\ x_{t,z} + x_{t,\dot{z}}\Delta t \\ x_{t,\dot{x}} \\ x_{t,\dot{y}} \\ x_{t,\dot{z}} - g\Delta t \\ x_{t,\psi} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ R_{bg}[0:] & & & 0 \\ R_{bg}[1:] & & & 0 \\ R_{bg}[2:] & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} u_t \Delta t \tag{49}$$

Then we take the Jacobian:

$$g'(x_t, u_t, \Delta t) = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & \frac{\partial}{\partial x_{t,\psi}}\left(x_{t,\dot{x}} + R_{bg}[0:]u_t[0:3]\Delta t\right) \\ 0 & 0 & 0 & 0 & 1 & 0 & \frac{\partial}{\partial x_{t,\psi}}\left(x_{t,\dot{y}} + R_{bg}[1:]u_t[0:3]\Delta t\right) \\ 0 & 0 & 0 & 0 & 0 & 1 & \frac{\partial}{\partial x_{t,\psi}}\left(x_{t,\dot{z}} + R_{bg}[2:]u_t[0:3]\Delta t\right) \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{50}$$

$$= \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & R'_{bg}[0:]u_t[0:3]\Delta t \\ 0 & 0 & 0 & 0 & 1 & 0 & R'_{bg}[1:]u_t[0:3]\Delta t \\ 0 & 0 & 0 & 0 & 0 & 1 & R'_{bg}[2:]u_t[0:3]\Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{51}$$

We define R'_{bg} as $\frac{\partial}{\partial x_{t,\psi}}$, defined as Equation 71 from Diebel [5]:

$$R'_{bg} = \begin{bmatrix} -\cos\theta\sin\psi & -\sin\phi\sin\theta\sin\psi - \cos\phi\cos\psi & -cos\phi\sin\theta\sin\psi + \sin\phi\cos\psi \\ \cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ 0 & 0 & 0 \end{bmatrix} \tag{52}$$

7.3 Measurement Model

We provide measurement models for the GPS and Magnetometer. We use the IMU as a control input so do not provide a measurement model for it here.

7.3.1 GPS

We assume we get position and velocity from the GPS. We considered using heading from the GPS, but this does not take into account the drone's orientation, only the direction of travel. Hence we are removing it from the observation.

$$z_t = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \tag{53}$$

Then the measurement model is:

$$h(x_t) = \begin{bmatrix} x_{t,x} \\ x_{t,y} \\ x_{t,z} \\ x_{t,\dot{x}} \\ x_{t,\dot{y}} \\ x_{t,\dot{z}} \end{bmatrix} \tag{54}$$

Then the partial derivative is the identity matrix, augmented with a vector of zeros for $\frac{\partial}{\partial x_{t,\phi}} h(x_t)$:

$$h'(x_t) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{55}$$

7.3.2 Magnetometer

We assume we get a reading from the magnetometer reporting yaw in the global frame. (This measurement may need to be computed using roll and pitch from the attitude filter and the mag vector.)

$$z_t = \begin{bmatrix} \psi \end{bmatrix} \tag{56}$$

$$h(x_t) = \begin{bmatrix} x_{t,\psi} \end{bmatrix} \tag{57}$$

Again since this is linear, the derivative is a matrix of zeros and ones.

$$h'(x_t) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{58}$$

7.4 Further Information

Based on not tracking acceleration as part of the state, we will use the accelerometer and gyro inputs as control inputs. Note that Erdem and Ercan [6] describe using the accelerometer inputs in the measurement or control input phases for an EKF over camera and IMU, for all eight combinations. They show that it is always better to fuse both sensors in the measurement stage. We would also then use a multirate Kalman Filter [4, 12] to perform updating, with separate updates for the GPS, magnetometer, and IMU.

However this means that both acceleration and angular velocity appear in the state vector, which we decided not to do in order to simplify the math and implementation. As one example, Ardupilot does it in this way [1]. The advantage is that it does not need to connect as deeply to the control system. Additionally, one could even use different state transition models in the control compared to the EKF. One might make this decision for computational reasons, for example.

Erdem and Ercan [6] give the math for an EKF with an IMU and camera (on a mobile device), showing the IMU and gyro measurements treated as a prediction and measurement input (all eight combinations). They show that the best perforance is obtained using the IMU as a measurement input. However the difference in pose accuracy estimation is around 1 cm. Note that the Ardupilot open source code base [1] makes a different decision, using the IMU as the control/prediction update, and that Bry et al. [2] describes this decision as a "commonly-used technique."

We will use the North/East/Down frame where the positive x gives the distance along the surface of the earth in the direction of north; the y coordinate gives the distance in the direction of *east*, and z is altitude, which is negative for distances above the surface of the earth.

We define an intermediate frame, p, which is a quaternion initialized from the roll, θ, and the pitch, ϕ, from the complementary filter, and no yaw correction. We define the global frame, q, as a quaternion filled out with the roll, θ, the pitch, ϕ, and the yaw, ψ, from the EKF.

8 Conclusion

We have provided equations for filtering for a quadrotor helicopter, combining information and notation from Thrun et al. [13], Quan [12] and other sources.

References

[1] Extended kalman filter navigation overview and tuning, Accessed March 5, 2018. http://ardupilot.org/dev/docs/extended-kalman-filter.html#extended-kalman-filter.

[2] Adam Bry, Abraham Bachrach, and Nicholas Roy. State estimation for aggressive flight in gps-denied environments using onboard sensing. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1–8. IEEE, 2012.

[3] John L Crassidis, F Landis Markley, and Yang Cheng. Survey of nonlinear attitude estimation methods. *Journal of guidance, control, and dynamics*, 30(1):12–28, 2007.

[4] Roberto Cristi and Murali Tummala. Multirate, multiresolution, recursive kalman filter. *Signal Processing*, 80(9):1945–1958, 2000.

[5] James Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, 58 (15-16):1–35, 2006.

[6] Arif Tanju Erdem and Ali Özer Ercan. Fusing inertial sensor data in an extended kalman filter for 3d camera tracking. *IEEE Transactions on Image Processing*, 24(2):538–548, 2015.

[7] Walter T Higgins. A comparison of complementary and kalman filtering. *IEEE Transactions on Aerospace and Electronic Systems*, (3):321–325, 1975.

[8] Tor-Aleksander Johansen and Raymond Kristiansen. Quadrotor attitude estimation using adaptive fading multiplicative ekf. In *American Control Conference (ACC), 2017*, pages 1227–1232. IEEE, 2017.

[9] Rambabu Kandepu, Bjarne Foss, and Lars Imsland. Applying the unscented kalman filter for nonlinear state estimation. *Journal of process control*, 18(7-8):753–768, 2008.

[10] F Landis Markley. Attitude error representations for kalman filtering. *Journal of guidance, control, and dynamics*, 26(2):311–317, 2003.

[11] Michał Nowicki, Jan Wietrzykowski, and Piotr Skrzypczyński. Simplicity or flexibility? complementary filter vs. ekf for orientation estimation on mobile devices. In *Cybernetics (CYBCONF), 2015 IEEE 2nd International Conference on*, pages 166–171. IEEE, 2015.

[12] Quan Quan. *Introduction to multicopter design and control*. Springer, 2017.

[13] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[14] Eric A Wan and Rudolph Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, pages 153–158. Ieee, 2000.