# Quaternion-Based EKF and UKF Attitude Estimation for the Arduino Teensy

Delia Stephens

***Abstract*—This semester, I implemented a quaternion-based Extended Kalman Filter and Unscented Kalman Filter for the Teensy Microcontroller + IMU. The Unscented Kalman Filter is still not fully functional; there appears to be some bugs either with the measurement model, the process model, the tuning of the covariance matrices, or all of the above. I welcome any feedback on how to improve the functionality of the Unscented Kalman Filter.**

## I. Introduction

THE Extended and Unscented Kalman Filters are two methods for estimating states with nonlinear dynamics and measurement models. The **Extended Kalman Filter** (EKF) handles this nonlinearity by forming a Gaussian approximation to the joint distribution of the state $\mathbf{x}$ and measurements $\mathbf{z}$ and linearizing the dynamics and measurements functions with Taylor-based approximations. The **Unscented Kalman Filter** (UKF), on the other hand, does not linearize the dynamics or measurements functions. Instead, it propagates a few *sigma points* from one time step to the next and compares the expected measurements generated by those sigma points to those actually generated by the system.

Here, I've implemented the EKF as outlined by [1], and attempted to implement the UKF as outlined by [2]. The EKF was substantially more successful than the UKF; this is likely because I made a mistake in the implementation of the process or measurement models. My Python files and notebooks with demonstrations are available at https://github.com/deliastephens/teensy_kf.

## II. Quaternion-Based Attitude

A quaternion, $\mathbf{q}$ is a way of measuring orientation. Quaternions are generally represented in the form $\mathbf{q} = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$; here, we use *unit quaternion*, or versors, which encode information about an axis-angle rotation about an arbitrary axis. Quaternions are simpler to compose than Euler angles and avoid the problem of gimbal lock, which is essential for any dynamics system attempting to orient itself in the world.

At the end of our quaternion estimation, it becomes necessary to convert the quaternion to a Euler angle. This is accomplished by the following transformation:

$$\begin{aligned}\text{roll} &= \arctan\left(\frac{2*(q_0*q_1+q_2*q_3}{1-2*(q_1^2+q_2^2)}\right)\\ \text{pitch} &= \arcsin\left(2*(q_0*q_2-q_3*q_1)\right)\\ \text{yaw} &= \arcsin\left(\frac{2*(q_0*q_3+q_1*q_2}{1-2*(q_2^2+q_3^2)}\right)\end{aligned}$$

## III. Extended Kalman Filter

Like a regular Kalman Filter, the EKF is split into two steps. They are described mathematically as follows:

*1) Prediction:*

$$\begin{aligned}\hat{\mathbf{x}}_t &= \mathbf{f}(\mathbf{x}_{t-1}, \mathbf{u}_t)\\ \hat{\mathbf{P}}_t &= \mathbf{F}(\mathbf{x}_{t-1}, \mathbf{u}_t)\mathbf{P}_{t-1}\mathbf{F}^T(\mathbf{x}_{t-1}, \mathbf{u}_t) + \mathbf{Q}_t\end{aligned} \tag{1}$$

*2) Update:*

$$\begin{aligned}\mathbf{v}_t &= \mathbf{z}_t - \mathbf{h}(\mathbf{x}_t)\\ \mathbf{S}_t &= \mathbf{H}(\mathbf{x}_t)\hat{\mathbf{P}}_t\mathbf{H}^T(\mathbf{x}_t) + \mathbf{R}_t\\ \mathbf{K}_t &= \hat{\mathbf{P}}_t\mathbf{H}^T(\mathbf{x}_t)\mathbf{S}_t^{-1}\\ \mathbf{x}_t &= \hat{\mathbf{x}}_t + \mathbf{K}_t\mathbf{v}_t\\ \mathbf{P}_t &= \big(\mathbf{I}_4 - \mathbf{K}_t\mathbf{H}(\mathbf{x}_t)\big)\hat{\mathbf{P}}_t\end{aligned} \tag{2}$$

where $\mathbf{x}_t$ is the state at time $t$, $\mathbf{f}$ is the dynamics function, $\mathbf{h}$ is the measurement function, $\mathbf{z}_t$, $\mathbf{P}$ is the estimation covariance matrix. Importantly, $\mathbf{F}$ and $\mathbf{H}$ are the Jacobians of the dynamics and measurement models, respectively.

Generally, the EKF is not optimal—it is only optimal if both the process and measurement functions are linear; in this case, it returns to being the simple Kalman Filter.

## IV. Unscented Kalman Filter

The unscented Kalman Filter is based on the concept of extractin predictions and knowledge about a system from the propagation of sigma points. The basic algorithm is outlined in the 12-15 from the lecture notes, and the method of approximating the sigma points is outlined in 12-19. I used the code from Homework 4 as the basis for my implementation.

## V. Implementation

I implemented the following in a few Python scrips and a Jupyter notebook. To confirm my results, please install the requisite files and run either the UKF or EKF notebooks. Below are two figures of my results; they are also present in the uploaded Jupyter notebook.

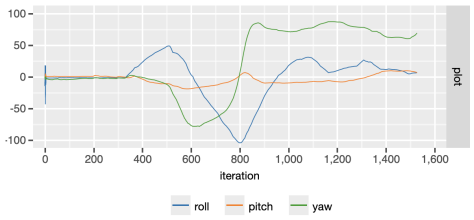


Fig. 1. EKF Simulation.

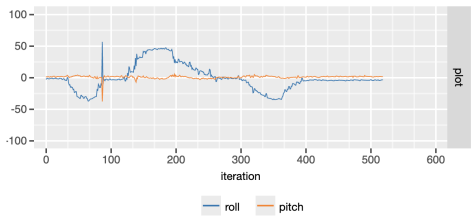


Fig. 2. UKF Simulation, roll only

After some experimentation, I determined that the Teensy is operates in the global NED frame. Therefore, the gravitational acceleration vector is defined as $\mathbf{g}_{\text{NED}} = \begin{bmatrix}0 & 0 & -9.81\end{bmatrix}^T$, and the representation of the Earth's magnetic field is expressed as $\mathbf{r}_{\text{NED}} = \begin{bmatrix}\cos\theta & 0 & \sin\theta\end{bmatrix}^T$

### A. EKF

In the EKF, we chose our state vector to be the quaternion, $\mathbf{q}$, and the angular velocity $\omega$ (rad/s) to be the control vector. Expressed mathematically, our state and control vectors are as follows:

$$\mathbf{x} = \mathbf{q} = \begin{bmatrix}q_w \mathbf{q}_v\end{bmatrix}^T = \begin{bmatrix}q_w & q_x & q_y & q_z\end{bmatrix}^T$$
$$\mathbf{u} = \boldsymbol{\omega} = \begin{bmatrix}\omega_x & \omega_y & \omega_z\end{bmatrix}^T$$

Our dynamics functions are given as:

$$\dot{\mathbf{q}}_t = \mathbf{A}\mathbf{q}_{t-1} + \mathbf{B}\boldsymbol{\omega}_t + \mathbf{w}_{\mathbf{q}}$$
$$\hat{\mathbf{q}}_t = \mathbf{F}\mathbf{q}_{t-1} = e^{\mathbf{A}\Delta t}\mathbf{q}_{t-1}$$

A clearer and more detailed explanation of these calculations is given in [1]. Here, I have focused on only deriving the relevant matrices used in my calculations.

*1) Prediction:* During the prediction step, it is necessary to linearize our dynamics. Rotating a quaternion by an arbitrary angular velocity, $\begin{bmatrix}\omega_x & \omega_y & \omega_z\end{bmatrix}$, is completed as follows:

$$\begin{aligned}\dot{\mathbf{q}} &= \lim_{\Delta t \to 0} \frac{\mathbf{q}(t+\Delta t) - \mathbf{q}(t)}{\Delta t} \\ &= \frac{1}{2}\boldsymbol{\Omega}_t \mathbf{q}_{t-1}\end{aligned}$$

where $\boldsymbol{\Omega}_t$ is given by:

$$\boldsymbol{\Omega}_t = \begin{bmatrix}0 & -\boldsymbol{\omega}^T \\ \boldsymbol{\omega} & \lfloor\boldsymbol{\omega}\rfloor_\times\end{bmatrix} = \begin{bmatrix}0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0\end{bmatrix}$$

To propagate this model over a short $\Delta t$ as given by the Teensy, our process model becomes the following:

$$\begin{aligned}\hat{\mathbf{q}}_t &= \mathbf{f}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t) \\ &= \left(\mathbf{I}_4 + \frac{\Delta t}{2}\boldsymbol{\Omega}_t\right)\mathbf{q}_{t-1} \\ \begin{bmatrix}\hat{q_w} \\ \hat{q_x} \\ \hat{q_y} \\ \hat{q_z}\end{bmatrix} &= \begin{bmatrix}q_w - \frac{\Delta t}{2}\omega_x q_x - \frac{\Delta t}{2}\omega_y q_y - \frac{\Delta t}{2}\omega_z q_z \\ q_x + \frac{\Delta t}{2}\omega_x q_w - \frac{\Delta t}{2}\omega_y q_z + \frac{\Delta t}{2}\omega_z q_y \\ q_y + \frac{\Delta t}{2}\omega_x q_z + \frac{\Delta t}{2}\omega_y q_w - \frac{\Delta t}{2}\omega_z q_x \\ q_z - \frac{\Delta t}{2}\omega_x q_y + \frac{\Delta t}{2}\omega_y q_x + \frac{\Delta t}{2}\omega_z q_w\end{bmatrix}\end{aligned}$$

We then take the Jacobian of $\mathbf{f}$ to get $\mathbf{F}$:

$$\begin{aligned}\mathbf{F}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t) &= \frac{\partial \mathbf{f}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)}{\partial \mathbf{q}} \\ &= \begin{bmatrix}1 & -\frac{\Delta t}{2}\omega_x & -\frac{\Delta t}{2}\omega_y & -\frac{\Delta t}{2}\omega_z \\ \frac{\Delta t}{2}\omega_x & 1 & \frac{\Delta t}{2}\omega_z & -\frac{\Delta t}{2}\omega_y \\ \frac{\Delta t}{2}\omega_y & -\frac{\Delta t}{2}\omega_z & 1 & \frac{\Delta t}{2}\omega_x \\ \frac{\Delta t}{2}\omega_z & \frac{\Delta t}{2}\omega_y & -\frac{\Delta t}{2}\omega_x & 1\end{bmatrix}\end{aligned}$$

.

Our process noise covariance, $\mathbf{Q}$, results mostly from gyroscope noise. We define a matrix $\mathbf{W}_t$

$$\begin{aligned}\mathbf{W}_t &= \frac{\partial \mathbf{f}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)}{\partial \boldsymbol{\omega}} \\ &= \begin{bmatrix}\frac{\partial \mathbf{f}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)}{\partial \omega_x} & \frac{\partial \mathbf{f}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)}{\partial \omega_y} & \frac{\partial \mathbf{f}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)}{\partial \omega_z}\end{bmatrix} \\ &= \frac{\Delta t}{2}\begin{bmatrix}-q_x & -q_y & -q_z \\ q_w & -q_z & q_y \\ q_z & q_w & -q_x \\ -q_y & q_x & q_w\end{bmatrix}\end{aligned}$$

and the spectral density noise matrix as:

$$\boldsymbol{\Sigma}_{\boldsymbol{\omega}} = \begin{bmatrix}\sigma^2_{\boldsymbol{\omega}x} & 0 & 0 \\ 0 & \sigma^2_{\boldsymbol{\omega}y} & 0 \\ 0 & 0 & \sigma^2_{\boldsymbol{\omega}z}\end{bmatrix}$$

For this implementation, I chose the following values, as recommended by the tutorial:

$$\sigma_{\boldsymbol{\omega}^2} = 0.3^2$$
$$\sigma_{\mathbf{a}^2} = .5^2$$
$$\sigma_{\mathbf{m}^2} = 0.8^2$$

For convenience, we assume assumed that the noises are equal on each axis, and don't influence each other, yielding a white, uncorrelated and isotropic noise. Therefore, our process noise covariance is given by $\mathbf{Q}_t = \sigma_{\boldsymbol{\omega}^2}\mathbf{W}_t\mathbf{W}_t^T$. At this point, we have all of the requisite information to complete the prediction step.

*2) Update:* During the update step, we compare our measurements from the accelerometer and gyroscope, $\mathbf{z}_t = \begin{bmatrix}\mathbf{a}_t \\ \mathbf{m}_t\end{bmatrix} = \begin{bmatrix}a_x & a_y & a_z & m_x & m_y & m_z\end{bmatrix}^T$, to our predicted measurements from the estimated quaternion $\hat{\mathbf{q}}$. Our measurement model involves calculating the expected measurements from a quaternion, and is aided by the definition of a matrix which rotates any vector through a quaternion.

$$\begin{aligned}\mathbf{x}' &= \mathbf{C}(\hat{\mathbf{q}})\mathbf{x} \\ &= \begin{bmatrix}1-2(\hat{q}_y^2+\hat{q}_z^2) & 2(\hat{q}_x\hat{q}_y - \hat{q}_w\hat{q}_z) & 2(\hat{q}_x\hat{q}_z + \hat{q}_w\hat{q}_y) \\ 2(\hat{q}_x\hat{q}_y + \hat{q}_w\hat{q}_z) & 1-2(\hat{q}_x^2+\hat{q}_z^2) & 2(\hat{q}_y\hat{q}_z - \hat{q}_w\hat{q}_x) \\ 2(\hat{q}_x\hat{q}_z - \hat{q}_w\hat{q}_y) & 2(\hat{q}_w\hat{q}_x + \hat{q}_y\hat{q}_z) & 1-2(\hat{q}_x^2+\hat{q}_y^2)\end{bmatrix}\begin{bmatrix}x_1 \\ x_2 \\ x_3\end{bmatrix}\end{aligned}$$

Therefore, our measurement model is given as:

$$\mathbf{h}(\hat{\mathbf{q}}_t) = \begin{bmatrix}\hat{\mathbf{a}} \\ \hat{\mathbf{m}}\end{bmatrix} = \begin{bmatrix}\mathbf{C}(\hat{\mathbf{q}})^T\mathbf{g} \\ \mathbf{C}(\hat{\mathbf{q}})^T\mathbf{r}\end{bmatrix}$$

This measurement is nonlinear, necessitating the computation of the Jacobian. The Jacobian of our measurement model can be refactored as:

$$\mathbf{H}(\hat{\mathbf{q}}_t) = \begin{bmatrix}\frac{\partial \hat{\mathbf{a}}}{\partial \mathbf{q}} \\ \frac{\partial \hat{\mathbf{m}}}{\partial \mathbf{q}}\end{bmatrix} = 2\begin{bmatrix}\mathbf{u}_g & \lfloor\mathbf{u}_g + \hat{q}_w\mathbf{g}\rfloor_\times + (\hat{\mathbf{q}}_v\cdot\mathbf{g})\mathbf{I}_3 - \mathbf{g}\hat{\mathbf{q}}_v^T \\ \mathbf{u}_r & \lfloor\mathbf{u}_r + \hat{q}_w\mathbf{r}\rfloor_\times + (\hat{\mathbf{q}}_v\cdot\mathbf{r})\mathbf{I}_3 - \mathbf{r}\hat{\mathbf{q}}_v^T\end{bmatrix}$$

Finally, we define our matrix $\mathbf{R}$ as the following:

$$\mathbf{R} = \begin{bmatrix} \boldsymbol{\sigma}_{\mathbf{a}}^2 \mathbf{I}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \boldsymbol{\sigma}_{\mathbf{m}}^2 \mathbf{I}_3 \end{bmatrix}$$

The EKF is not optimal, so it can diverge rapidly with incorrect models. To remedy this situation, I found it advantageous to renormalize the unit quaternion $\mathbf{q}$ after every predict and update steps.

### *B. UKF*

For the UKF, we defined a seven-dimensional state composed of the quaternions and angular velocities.

The measurement model and the dynamics models remain similar, but we do not use the Jacobians in our calculations. For a more detailed look at the implementation, please check out my code!

The sigma points are chosen as Merwe Sigma points using the procedure outlined in [3] and implemented in the Homework 4 solutions. One important distinction was that, after every sigma point selection, I had to renormalize them—we needed to deal explicitly with the unit quaternion, and normalizing the vector helped avoid unreasonable predicted measurements.After some tinkering, the parameters we chose were as follows:

$$\alpha = 0.5$$
$$\beta = 1$$
$$\kappa = 0$$

It's important to note that the UKF looked "noisier" than I expected it to. Unfortunately, I was unable to determine the reason for this before the project deadline. I think it might have something to do with the inclusion of the angular velocities in the states, which are never compared to anything from the measurement model; perhaps this is why the state estimates jump more rapidly than they should.

## VI. Conclusion

Ultimately, I wasn't able to accomplish my goal of perfectly implementing an Unscented Kalman Filter. I'm still not sure why this is. I also did not calibrate the sensors before capturing measurements; this would have likely improved my results drastically as well.

## References

[1] M. Garcia, "Attitude and Heading Reference System: Extended Kalman Filter." [Online]. Available: https://ahrs.readthedocs.io/en/latest/filters/ekf.html

[2] E. Kraft, "A quaternion-based unscented Kalman filter for orientation tracking," in *Sixth International Conference of Information Fusion, 2003. Proceedings of the*. Cairns, Queensland, Australia: IEEE, 2003, pp. 47–54. [Online]. Available: http://ieeexplore.ieee.org/document/1257247/

[3] E. Wan and R. Van Der Merwe, "The unscented Kalman filter for nonlinear estimation," in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*. Lake Louise, Alta., Canada: IEEE, 2000, pp. 153–158. [Online]. Available: http://ieeexplore.ieee.org/document/882463/