ASE 479W Laboratory 3 Report
Measurement Simulation and State Estimation

Harrison Qiu Jin

March 11, 2022

1 Introduction

Unmanned aerial vehicles, or UAVs, have many different applications in a variety of industries. UAVs pose many challenges that are still being widely researched. As researchers develop different decision-making policies and algorithms, it is important for them to be able to easily test and iterate those models. While testing can certainly be done on physical hardware UAVs, any mistakes in the software or faulty decision-making can lead to costly crashes. Thus, it is important for researchers to be able to test their work on a simulated UAV with a robust dynamic model. This paper explores the development of a simulator for a quadrotor drone.

When simulating a quadrotor drone, the trajectory and attitude can be easily controlled by accessing the true state of the drone, such as its position, velocity, and attitude. Using the true state of the drone allows for near-perfect reference tracking with extremely negligible amounts of error. However, physical drones in the real world rarely have the privilege of knowing their true state. Instead, they must estimate their state using measurements from a variety of sensors, which vary from vehicle to vehicle. This paper will explore the implementation of sensor models for a Global Navigation Satellite System (GNSS) sensor, a camera sensor, and an Inertial Measurement Unit (IMU). These sensor models will incorporate the noise and uncertainty that comes with the physical versions of these sensors. State estimation using an unscented Kalman filter (UKF) will be done using the measurements from the sensor models as input, allowing for a more realistic approach to simulating the control of a quadrotor drone.

2 Theoretical Analysis

2.1 GNSS Sensor Model

The quadrotor modeled in this simulator has two GNSS antennas, a primary and a secondary. Using a reference antenna with a fixed, known origin, the pose of the quad can be estimated by measuring the position of each antenna. Because there is no measurement error in the length of the vector pointing from the primary antenna to the secondary antenna r_{bG}, the covariance matrix R_{bG} for the measurement of this vector takes the following form:

$$R_{bG} = \|r_{bG}\|^2 \sigma_b^2 [I_{3\times 3} - r_{bG}^u (r_{bG}^u)^\intercal] \tag{1}$$

where r_{bG}^u is the unit norm vector in the direction of r_{bG} and σ_b is the standard deviation of the angular error (in radians). Using (1), it can be shown that the covariance matrix R_{bG} has rank 2.

Let

$$r_{bG}^{u} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\frac{R_{bG}}{\|r_{bG}\|^2\sigma_b^2} = [I_{3x3} - r_{bG}^{u}(r_{bG}^{u})^{\intercal}]$$

$$\frac{R_{bG}}{\|r_{bG}\|^2\sigma_b^2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} x^2 & xy & xz \\ xy & y^2 & yz \\ xz & yz & z^2 \end{bmatrix}$$

$$\frac{R_{bG}}{\|r_{bG}\|^2\sigma_b^2} = \begin{bmatrix} 1-x^2 & -xy & -xz \\ -xy & 1-y^2 & -yz \\ -xz & -yz & 1-z^2 \end{bmatrix} \tag{2}$$

Equation 2 shows that R_{bG} is symmetric, meaning that its rank is equal to the number of nonzero eigenvalues. The eigenvalues λ for the matrix are calculated to be

$$\lambda = \begin{bmatrix} 1 \\ 1 \\ 1-(x^2+y^2+z^2) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1-\|r_{bG}^{u}\|^2 \end{bmatrix}$$

Because r_{bG}^{u} is a unit vector, the last eigenvalue is zero regardless of the values of the components of r_{bG}. Therefore, R_{bG} has two nonzero eigenvalues, which necessarily means that it has rank 2 and is a positive semidefinite matrix.

2.2 Camera Sensor Model

The camera sensor model used in this simulator projects the three-dimensional position of a visible feature onto a two-dimensional image plane. The two dimensional position $[x, y]^{\intercal}$ can be represented in homogeneous coordinates as $[kx, ky, k]^{\intercal}$, for any $k \neq 0$. A line defined by

$$ax + by + c = 0$$

can be represented by $l = [a, b, c]^{\intercal}$. It can then be shown that the point defined by the intersection of two lines l_1 and l_2 can be found as $l_1 \times l_2$. Let

$$l_1 = \begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix}, l_2 = \begin{bmatrix} a_2 \\ b_2 \\ c_2 \end{bmatrix}$$

represent the lines $a_1x + b_1y + c_1 = 0$ and $a_2x + b_2y + c_2 = 0$, respectively.

$$l_1 \times l_2 = \begin{vmatrix} i & j & k \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{vmatrix} = \begin{bmatrix} b_1c_2 - b_2c_1 \\ -(a_1c_2 - a_2c_1) \\ a_1b_2 - a_2b_1 \end{bmatrix} \tag{3}$$

The final result in Equation 3 represents the intersection point of $l1$ and $l2$ in homogeneous coordinates such that $x = \frac{b_1c_2 - b_2c_1}{a_1b_2 - a_2b_1}$ and $y = \frac{a_2c_1 - a_1c_2}{a_1b_2 - a_2b_1}$. Both equations are satisfied, as shown below.

$$a_1x + b_1y + c_1$$

$$= a_1(\frac{b_1c_2 - b_2c_1}{a_1b_2 - a_2b_1}) + b_1(\frac{a_2c_1 - a_1c_2}{a_1b_2 - a_2b_1}) + c_1$$

$$= \frac{1}{a_1b_2 - a_2b_1}(a_1b_1c_2 - a_1b_2c_1 + a_2b_1c_1 - a_1b_1c_2) + c_1$$

$$= \frac{c_1}{a_1b_2 - a_2b_1}(-a_1b_2 + a_2b_1) + c_1$$

$$= (\frac{-(a_1b_2 - a_2b_1)}{a_1b_2 - a_2b_1} + 1)c_1 = 0$$

$$a_2x + b_2y + c_2$$

$$= a_2(\frac{b_1c_2 - b_2c_1}{a_1b_2 - a_2b_1}) + b_2(\frac{a_2c_1 - a_1c_2}{a_1b_2 - a_2b_1}) + c_2$$

$$= \frac{1}{a_1b_2 - a_2b_1}(a_2b_1c_2 - a_2b_2c_1 + a_2b_2c_1 - a_1b_2c_2) + c_2$$

$$= \frac{c_2}{a_1b_2 - a_2b_1}(a_2b_1 - a_1b_2) + c_2$$

$$= (\frac{-(a_1b_2 - a_2b_1)}{a_1b_2 - a_2b_1} + 1)c_2 = 0$$

2.3 IMU Sensor Model

An Inertial Meaurement Unit (IMU) measures specific force and angular rate. For specific force f_B, the following measurement model is used:

$$\tilde{f}_B = R_{BI}(a_I + \begin{bmatrix}0\\0\\g\end{bmatrix}) + b_a + v_a \tag{4}$$

where a_I is the acceleration with respect to I of a point fixed in the B frame, b_a is measurement bias noise, and v_a is a white noise term. Let the acceleration a_I of with respect to the inertial frame of a point fixed in the body frame and located at l_B in the B frame be

$$a_I = \ddot{R}_I + R_{BI}^{\intercal}a_B$$

where a_B is the acceleration of the point in the body frame. $a_B = R_{BI}\ddot{l}_I$ can be found by taking the double time derivative of l_B and applying the product rule, as shown below.

$$l_B = R_{BI}l_I$$
$$\dot{l}_B = \dot{R}_{BI}l_I + R_{BI}\dot{l}_I$$

Applying the kinematic equation for a direction cosine matrix $\dot{R}_{BI} = -[\omega_B\times] R_{BI}$ yields

$$\dot{l}_B = -[\omega_B\times] R_{BI} l_I + \dot{l}_I R_{BI} = -\omega_B \times l_B + R_{BI}\dot{l}_I \tag{5}$$

$$\ddot{l}_B = -\dot{\omega}_B \times l_B - \omega_B \times \dot{l}_B + \dot{R}_{BI}\dot{l}_I + R_{BI}\ddot{l}_I$$

To find a_B, $\dot{R}_{BI} = -[\omega_B\times] R_{BI}$ and (5) can be substituted into the above. Rearranging to solve for $a_B = R_{BI}\ddot{l}_I$ yields the following expression for a_B

$$a_B = R_{BI}\ddot{l}_I = \ddot{l}_B + (\dot{\omega}_B \times l_B) + 2(\omega_B \times \dot{l}_B) + \omega_B \times (\omega_B \times l_B)$$

For a point that is fixed in the body frame, $\dot{l}_B = \ddot{l}_B = 0$.

$$a_B = (\dot{\omega}_B \times l_B) + \omega_B \times (\omega_B \times l_B)$$

Finally,

$$a_I = \ddot{R}_I + R_{BI}^{\intercal}[(\dot{\omega}_B \times l_B) + \omega_B \times (\omega_B \times l_B)]$$

However, for the purposes of this simulator, it is assumed that $l_B = 0$. Thus, $a_I = \ddot{R}_I$ and (4) becomes

$$\tilde{f}_B = R_{BI}(\ddot{R}_I + \begin{bmatrix}0\\0\\g\end{bmatrix}) + b_a + v_a$$

The IMU also contains a rate gyro that measures the angular rate ω_B using the following measurement model.

$$\tilde{\omega}_B = \omega_B + b_g + v_g$$

where b_g is the gyro bias and v_g is a white noise term. Note that the lever arm l_B does not need to be accounted for in the measurement of ω_B as it does in a_I because the angular rate measured by the rate gyro will have the same value regardless of the position of the IMU with respect to the B frame.

3 Implementation

3.1 GNSS Sensor Model

The GNSS sensor was implemented using the following measurement models:

$$\tilde{r}_{pG}(t_k) = r_{pG}(t_k) + \omega_{pG}(t_k)$$

$$\tilde{r}_{bG}(t_k) = r_{bG}(t_k) + \omega_{bG}(t_k)$$

where $\tilde{r_{pG}}$ is the measured value of the position of the primary antenna at time t_k, and $\tilde{r_{bG}}$ is the vector pointing from the primary antenna to the secondary antenna at time t_k. Both of these values are in the Earth-centered, Earth-fixed (ECEF) frame. Both ω terms are white discrete-time Gaussian noise processes. These terms were modeled in this simulator using MATLAB's `mvnrnd` function, which generates the noise term using a covariance matrix. One important implementation detail is that the covariance matrix for ω_{bG} is only positive semidefinite (see Section 2.1) rather than positive definite. To prevent any issues from this, a very small value of 10^{-8} was added to the covariance matrix for ω_{bG}, as implemented below:

```
omega_bG_cov = ...
    norm(rbG)^2*P.sensorParams.sigmab^2*(eye(3,3)-(rbG_u*rbG_u'))+epsilon*eye(3,3);
omega_bG = mvnrnd(zeros(3,1), omega_bG_cov)';
```

After adding the simulated noise to the measurement, $\tilde{r}_{bG}$ was normalized to the known length of the distance between the primary and secondary antennas.

```
rbGtilde = rbGtilde/norm(rbGtilde)*norm(rbG);
```

3.2 Camera Sensor Model

The camera sensor was implemented by projecting the three-dimensional coordinates of a visible feature onto the camera's image plane. This was done using homogeneous coordinates that were rotated and translated into the camera's reference frame. One important condition to check at this point is to make sure that the feature itself is in front of the camera and not behind, as this information is lost when the coordinates are projected onto the two-dimensional image plane. After ensuring that the feature is in fact in front of the camera, the coordinates can safely be projected into two-dimensions. The other condition to check is that the feature is within the field of view of the camera, which can be verified by comparing the two dimensional coordinates to the limits of the image plane. Finally, the coordinates are transformed from meters into pixels, and a white noise term was added as in the GNSS sensor model (see Section 3.1).

3.3 IMU Sensor Model

The IMU sensor model was implemented as described in Section 2.3. The white noise processes were modeled in the same way as in the GNSS sensor model (see Section 3.1). The bias terms b_a and b_g were modeled as first order Gauss-Markov processes:

$$b_a(t_{k+1}) = \alpha_a b_a(t_k) + v_{a2}(t_k) \tag{6}$$

$$b_g(t_{k+1}) = \alpha_g b_g(t_k) + v_{g2}(t_k) \tag{7}$$

Here $\alpha_a = e^{-\Delta t/\tau_a}$ and $\alpha_g = e^{-\Delta t/\tau_g}$, where $\Delta t = t_{k+1} - t_k$ and τ is the correlation time of the Gauss-Markov process. The white noise terms v_{a2} and v_{g2} were modeled using MATLAB's `mvnrnd` function. Note that these terms are independent of v_a and v_g. Because the previous values of b_a and b_g must be known to calculate the current values, MATLAB `persistent` variables were used in this implementation.

3.4 State Estimation

An unscented Kalman filter (UKF) is used to estimate the state of the quadrotor from the simulated sensor models. A model replacement approach is taken where the dynamics model of the drone is driven primarily by the IMU measurements rather than Euler's equations. This approach requires two mathematical models to implement: a sensor measurement model and a dynamics model.

3.4.1 Measurement Model

A nonlinear measurement model of the form

$$z(k) = h[x(k)] + \omega(k)$$

is used for the simulator, where $z(K)$ is a vector containing stacked GNSS and camera measurements, $x(k)$ is the state vector, and $\omega(k)$ is a vector modeling measurement noise. This measurement model is used in the UKF to compare to the actual measurements taken from the sensors. The state vector $x(k)$ contains the following terms

$$x(k) = \begin{bmatrix} r_I(k) \\ v_I(k) \\ e(k) \\ b_a(k) \\ b_g(k) \end{bmatrix} \tag{8}$$

where $r_I(k)$ is the position of the center of mass of the quad, $v_I(k)$ is the velocity of the quad, $e(k)$ holds the Euler angles that represent the current attitude, and b_a and b_g are the measurement biases for the IMU (see Section 3.3).

3.4.2 Dynamics Model

The dynamics model used for this simulator is a nonlinear model of the form

$$x(k+1) = f[x(k), u(k), v(k)]$$

where $x(k)$ is the state vector (8), $u(k)$ is the input (IMU measurements in this case), $v(k)$ is a vector modeling process noise, and f is a propagation function. An explicit Euler method is used for velocity and attitude, while acceleration is also accounted for in the position propagation function. For the bias terms b_a and b_g, the Gauss-Markov models as described in (6) and (7) are used.

4 Results and Analysis

4.1 State Estimation

To initialize the UKF, Wahba's problem was solved with singular value decomposition (SVD) to find an initial attitude estimate. This implementation was verified for accuracy using the test script below.

```
rng('shuffle');
errors = [];
for x = -1:0.1:1
    for y = -1:0.1:1
        for z = -1:0.1:1
            % Set RBI
            RBI_actual = euler2dcm([x*pi y*pi z*pi]);
            % Generate synthetic data
            vIMat = zeros(10,3);
            vBMat = zeros(10,3);
            for k = 1:10
                vIMat(k,:) = rand(1,3);
                vBMat(k,:) = (RBI_actual*vIMat(k,:)')';
            end
            % Find RBI using wahbaSolver and synthetic data
            RBI_wahba = wahbaSolver(ones(10,1), vIMat, vBMat);
            errors = [errors;norm(RBI_wahba-RBI_actual)];
        end
    end
end
max(errors)
```

The test script loops over the entire domain of Euler angles and applies those rotations to ten randomly generated vectors. Each vector and its corresponding mapping is passed into the Wahba solver (with equal weights of 1). The output of the Wahba solver is then verified to match the expected rotation matrix. The result of the test script, which prints the maximum error norm of all the test cases, is the following:

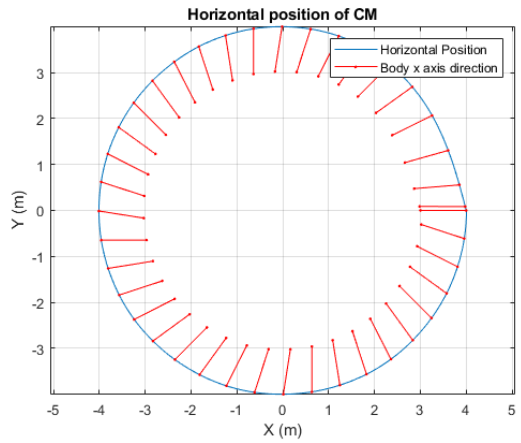
```
ans =

    1.1027e-14
```

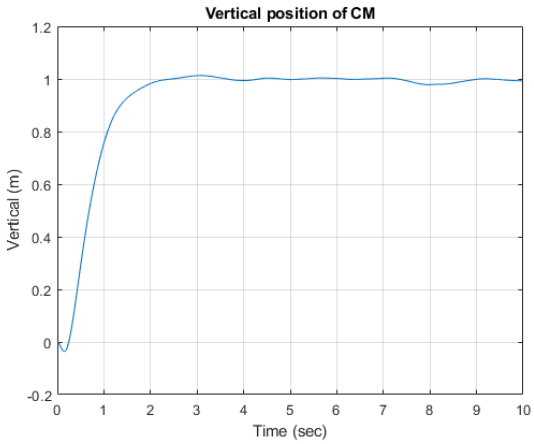
This indicates that the error for every case tested is less than or equal to the above, which can simply be attributed to floating point error accumulated during arithmetic operations.

4.2 Experimentation

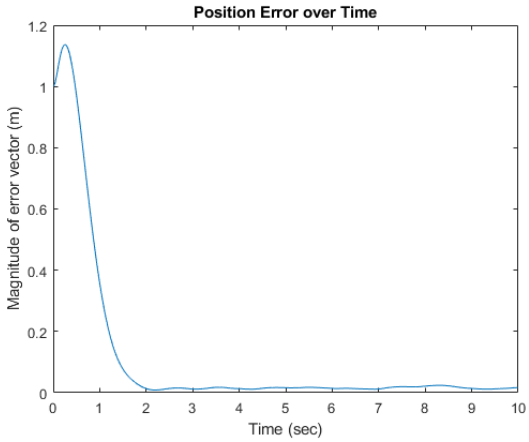
The sensor models and UKF were tested by feeding a circular trajectory of radius $R = 4m$, period $T = 10s$, and altitude $h = 1m$ into the quadrotor. The quadrotor was set to an initial state of zero velocity (linear and angular), zero angular rotor rate, and an initial position of $\begin{bmatrix}4 & 0 & 0\end{bmatrix}^\intercal$. The body x axis initially points towards the center of the circle. The quad was tested using both the estimated state from the sensor models as well as the actual state of the quad. The magnitude of the position error with respect to the desired trajectory was measured in each case. The results of the experiment are shown below in Figure 1.



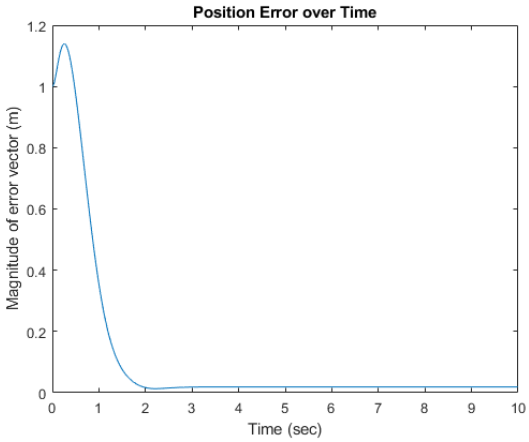
(a) The true horizontal position of the center of mass over one period, with the direction of the body x indicated.



(b) The vertical position of the center of mass over one period.



(c) The magnitude of the error with respect to the desired trajectory when flown using the state estimator.

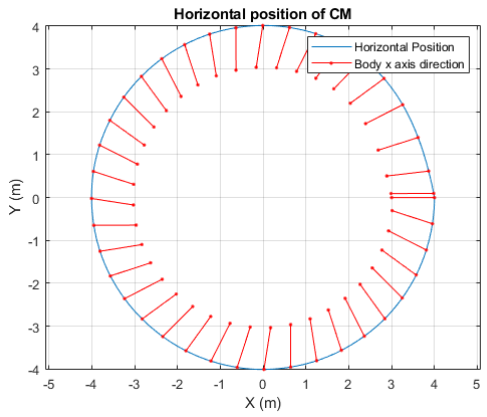


(d) The magnitude of the error with respect to the desired trajectory when flown using the actual state.

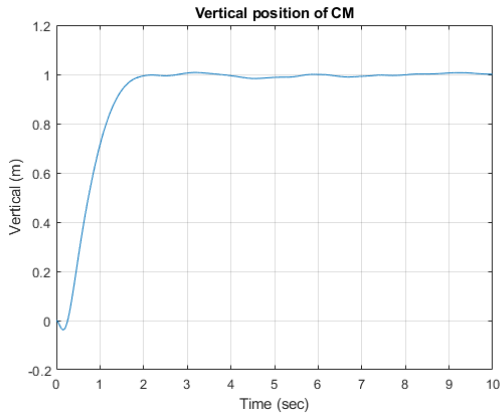
Figure 1: Results of circular flight trajectory experiment when flown using state estimator and true state.

Figures 1a and 1b show that using the state estimator allows the quad to fly very closely to the desired trajectory. The horizontal position is relatively circular, with the most noticeable deviation being the beginning of the quad's flight where it is "catching up" from zero initial velocity. The altitude of the quad stays within a very small margin of error once it has reached the steady state value of 1 meter. When comparing Figures 1c and 1d, it is clear that there is only a small amount of variation in the error when using the state estimator, but it stays very close to the amount of error seen when using the true state. This conclusion was verified by running multiple trials with `rng('shuffle')`, and the error did not change significantly.

The experiment was run again without any input from the camera sensor model. The results are shown below in Figure 2.



(a) The true horizontal position of the center of mass over one period, with the direction of the body x indicated.



(b) The vertical position of the center of mass over one period.

Figure 2: Results of circular flight trajectory experiment when flown without the camera sensor model.

As shown in Figure 2, removing the camera sensor did not significantly change the results of the experiment. This suggests that the state estimator is able to generate a reasonably accurate state estimate using only the GNSS sensor model and the IMU sensor model, and that these sensors alone are sufficiently accurate for quadrotor control. This makes sense considering the components of the state vector (8). Position, velocity, attitude, and measurement bias can all be estimated using just the GNSS and IMU sensors. The GNSS provides positional data, the IMU's accelerometer provides specific force which can be used to find acceleration and thus velocity, and the IMU's rate gyro can be used to estimate attitude given an initial attitude estimate (as provided by the solver of Wahba's problem). However, since some of these measurements need to be integrated in order to provide state estimates, more error is added when there aren't other measurements for estimation, such as a camera.

5 Conclusion

The UAV simulator was extended to more realistically simulate a physical quadrotor by adding models for a camera sensor, an IMU sensor, and a GNSS sensor. The measurements made by these sensor models allowed the state of the quadrotor to be estimated with an unscented Kalman filter. The state estimate was then used for PD control of the quadrotor's trajectory and attitude. This provided a much more realistic simulation environment as compared to using the true state of the quadrotor for PD control, as physical quadrotors are not able to have full knowledge of their state vectors. Controlling the quadrotor with the estimated state was shown to be quite accurate, with only small deviations as compared to controlling the quadrotor with the true state. Finally, it was shown that the camera sensor is not strictly necessary for accurate quadrotor control.