DRONE STATE ESTIMATION USING MULTIPLE INFRARED CAMERAS

Kadhir Umasankar*

A motion capture system is generally used to estimate drone states in laboratory environments. For this setup, some reflective markers are attached to the drone, and multiple infrared cameras are positioned around the experiment area. The reflective markers' positions in space, along with their relative positions with respect to each other, are captured by the cameras, processed by software on the command center computer and are used to return the 3D position, velocities, and roll, pitch, and yaw of the drone. In this study, the logic used by the software on the command center computer will be replicated. A system that uses multiple cameras (all with their own biases and noise) will be simulated to estimate the state of the drone.

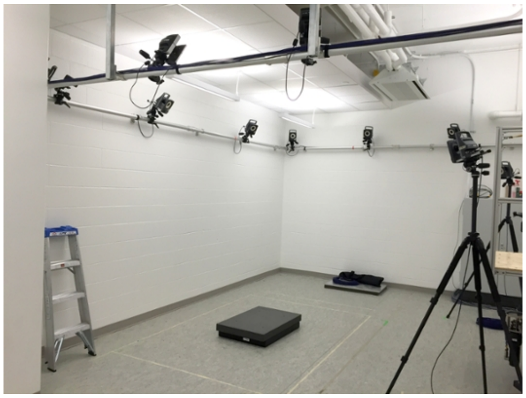
INTRODUCTION

Drones generally use GPS measurements in conjunction with measurements from an IMU to estimate their position, velocity, and their roll, pitch, and yaw. In laboratory environments, the drones cannot use GPS data to complement other sensory inputs, so a motion capture system is generally used.[1] In such a system, a local origin is set, and the drone uses that as the origin of its inertial frame of reference.

In optical-passive motion capture, retro-reflective markers are placed at various places on the drone. Multiple infrared cameras are positioned around the experiment area, and they capture the infrared light reflected back from the retroreflective markers at rates of around 120 Hz. Such high input frequency is important when testing out controllers, as a high number of samples would allow the controller to control better against sudden changes in the state of the system. Figure 1 shows examples of a motion capture space and a drone with retro-reflective markers attached.

Most previous studies in drone state estimation have assumed constant inputs. In this study, a linear-quadratic regulator (LQR) controller will be used to find the inputs to the system's dynamics at every timestep, thereby allowing more types of flight paths to be tracked. This study will use simulated motion capture system data to estimate the states of a drone in various flight patterns. The simulation will take place in a Gazebo 3D simulation environment.[4] A ROS (Robot Operating System)[5] package will be used to perform control and trajectory planning for the drone, and logs of the flight data will be will be run through an Extended Kalman filter (EKF) and an Unscented Kalman filter (UKF) in MATLAB to obtain an accurate estimate of the position of the drone. The performance of the filters on flight paths of varying levels of difficulty will be analyzed.

*Graduate Student, Daniel Guggenheim School of Aerospace Engineering, kadhir.umasankar@gatech.edu



(a) Example motion capture space setup[2]

(b) Example drone with retro-reflective markers[3]

Figure 1: Example setup that could be used if this study were to be repeated on real-life data

SIMULATION SETUP

Data for this study was obtained through simulation. The drone chosen for this experiment was the 3DR Iris,[6] a model of which is available for use in Gazebo.[4] The Spatial Data File (SDF) of the Iris' model was edited to include four reflective markers. The positions of these markers can be seen in Table 1 and Figure 2. This drone also contained an IMU, and the SDF of the Iris' model showed that it was located at the centroid of the drone.

Table 1: Positions of the reflective markers (in cm) with respect to the centroid of the drone

Marker Number	x	y	z
1	10.5	0	0
2	0	6.0	0
3	-11.5	0	0
4	0	-6.0	0

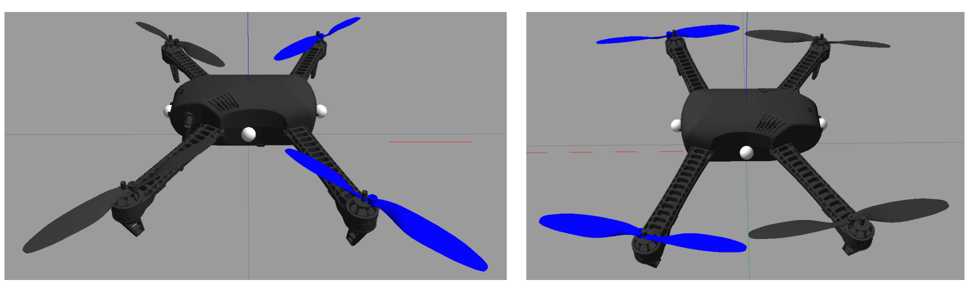


Figure 2: Right and left views of the drone that will be used for simulation

A world with four Vicon Vantage V16 infrared cameras[1] was then created in Gazebo, and their coordinates can be seen in Table 2. Figure 3 shows the setup of the environment. The distance from these cameras to reflective markers on the drone was measured at a rate of approximately 120 Hz, which is consistent with the spec sheet of the Vicon Vantage V16.[1]

Table 2: Coordinates of the cameras in m

Camera ID	x	y	z
1	10	10	10
2	-10	10	10
3	-10	-10	10
4	10	-10	10

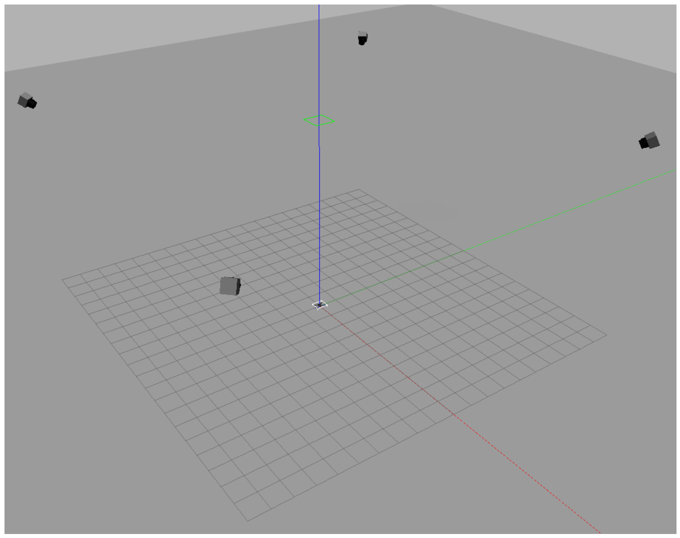


Figure 3: The Gazebo world that the simulation will take place in. Each square has a side length of 1m. Thus, the entire world is 10m $\times$ 10m $\times$ 10m

DYNAMICS MODELING

Eq. (1) shows the states that were to be observed for the filter, where x, y, and z are the x, y, and z positions of the drone, v_x, v_y, and v_z are the x, y, and z velocities of the drone, ϕ, θ, and ψ are the roll, pitch, yaw of the drone, and p, q, and r are the body angular velocities in terms of Euler angles and Euler rates.

$$X = \begin{bmatrix} x & y & z & v_x & v_y & v_z & \phi & \theta & \psi & p & q & r \end{bmatrix}^T \tag{1}$$

Taking the time derivative of X_1 through X_3 from Eq. (1) gives Eq. (2).

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \tag{2}$$

The velocities of the drone (i.e. v_x, v_y, v_z) will be perturbed by the thrust on the drone, which will be referred to as u_3. However, drone dynamics states that the thrust will only act in the $+z$ of the drone body-fixed frame, denoted by X_b, Y_b, and Z_b in Figure 4. Thus, the thrust input must be rotated with respected to the roll, pitch, and yaw of the drone to find its effect along the X_i, Y_i, and Z_i directions. An Euler 123 rotation, denoted by R in Eq. (3) is used to rotate from the body frame to the inertial frame*. Combining these steps, taking the time derivative of X_4 through X_6 gives Eq.

*Throughout this report, c, s, and t will be used in place of $\cos$, $\sin$, and $\tan$ for brevity

(5).

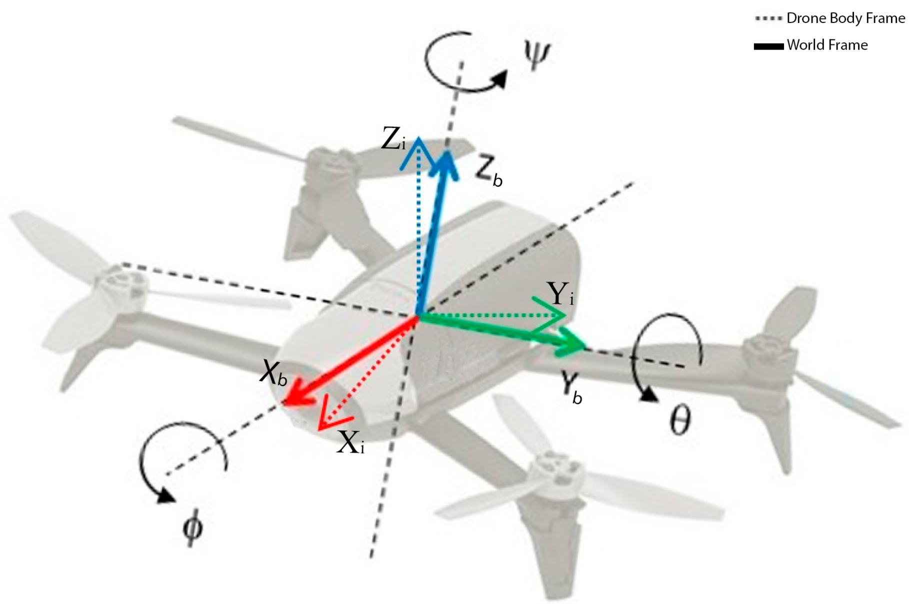


Figure 4: An illustration comparing the inertial world frame (denoted by dotted lines), and the body-fixed frame (denoted by solid lines)

$$R_{123} = R_z(\psi)R_y(\theta)R_x(\phi) = \begin{bmatrix} c(\psi) & s(\psi) & 0 \\ s(\psi) & c(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c(\theta) & 0 & s(\theta) \\ 0 & 1 & 0 \\ -s(\theta) & 0 & c(\theta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c(\phi) & -s(\phi) \\ 0 & s(\phi) & c(\phi) \end{bmatrix} \tag{3}$$

$$R_{123} = \begin{bmatrix} c(\theta)c(\psi) & c(\psi)s(\theta)s(\phi) - c(\phi)s(\psi) & s(\phi)s(\psi) + c(\phi)c(\psi)s(\theta) \\ c(\theta)s(\psi) & c(\phi)c(\psi) + s(\theta)s(\phi)s(\psi) & c(\phi)s(\theta)s(\psi) - c(\psi)s(\phi) \\ -s(\theta) & c(\theta)s(\phi) & c(\theta)c(\phi) \end{bmatrix} \tag{4}$$

$$\begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \frac{R_{123}\begin{bmatrix} 0 \\ 0 \\ u_3 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}}{m} = \begin{bmatrix} (u_3(s(\phi)s(\psi) + c(\phi)c(\psi)s(\theta)))/m \\ -(u_3(c(\psi)s(\phi) - c(\phi)s(\theta)s(\psi)))/m \\ -(mg - u_3c(\theta)c(\phi))/m \end{bmatrix} \tag{5}$$

Similarly, since p, q, and r (i.e. the roll, pitch, and yaw rates) are in the body frame, they must be rotated into the inertial frame. Taking the time derivative of X_7 through X_9 then gives Eq. (6).

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & s(\phi)t(\theta) & c(\phi)t(\theta) \\ 0 & c(\phi) & -s(\phi) \\ 0 & s(\phi)/c(\theta) & c(\phi)/c(\theta) \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} p + r\cos(\phi)\tan(\theta) + q\tan(\theta)\sin(\phi) \\ q\cos(\phi) - r\sin(\phi) \\ (r\cos(\phi))/\cos(\theta) + (q\sin(\phi))/\cos(\theta) \end{bmatrix} \tag{6}$$

The rotational equations of motion can be derived from Euler's equations for rigid body dynamics. Expressed in vector form, Euler's equations are written as

$$I\dot{\omega} + \omega \times (I\omega) = \tau$$

where ω is the angular velocity vector, I is the inertia matrix, and τ is a vector of external torques. This can be rewritten as

$$\dot{\omega} = \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = I^{-1}(\begin{bmatrix} u_4 \\ u_5 \\ u_6 \end{bmatrix} \omega \times (I\omega))$$

where u_4 through u_6 are the roll, pitch, yaw torques on the drone respectively. The quadcopter can be modeled as two thin uniform rods crossed at the origin with a point mass (the motor) at the end of each. This results in a diagonal inertia matrix of the form

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}$$

Combining these steps, the final result of the time derivative of states X_{10} through X_{12} can be seen in Eq. (7).[7]

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} (u_4 + I_{yy}qr - I_{zz}qr)/I_{xx} \\ (u_5 - I_{xx}pr + I_{zz}pr)/I_{yy} \\ (u_6 + I_{xx}qp - I_{yy}qp)/I_{zz} \end{bmatrix} \tag{7}$$

Combining Equations 2, 5, 6 and 7 gives Eq. (8).

$$\dot{X}(t) = F(X(t), t) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{v_z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ (u_3(\sin(\phi)\sin(\psi) + \cos(\phi)\cos(\psi)\sin(\theta))/m \\ (-u_3(\cos(\psi)\sin(\phi) + \cos(\phi)\sin(\theta)\sin(\psi))/m \\ (u_3\cos(\theta)\cos(\phi) - mg)/m \\ p + r\cos(\phi)\tan(\theta) + q\tan(\theta)\sin(\phi) \\ q\cos(\phi) - r\sin(\phi) \\ \frac{r\cos(\phi)}{\cos(\theta)} + \frac{q\sin(\phi)}{\cos(\theta)} \\ \frac{u_4 + qrI_{yy} - qrI_{zz}}{I_{xx}} \\ \frac{u_5 - prI_{xx} + prI_{zz}}{I_{yy}} \\ \frac{u_6 + qpI_{xx} - qpI_{yy}}{I_{zz}} \end{bmatrix} \tag{8}$$

Taking the partial of F with respect to X gives Eq. (9).

$$A = \frac{\partial F}{\partial X} = \begin{bmatrix} \frac{\partial F_1}{\partial x} & \cdots & \frac{\partial F_1}{\partial r} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_{12}}{\partial x} & \cdots & \frac{\partial F_{12}}{\partial r} \end{bmatrix} \tag{9}$$

$$
A = \begin{bmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{u_3\left(\mathrm{c}(\phi)\mathrm{s}(\psi) - \mathrm{c}(\psi)\mathrm{s}(\theta)\mathrm{s}(\phi)\right)}{m} & \frac{u_3\mathrm{c}(\theta)\mathrm{c}(\phi)\mathrm{c}(\psi)}{m} & \frac{u_3\left(\mathrm{c}(\psi)\mathrm{s}(\phi) - \mathrm{c}(\phi)\mathrm{s}(\theta)\mathrm{s}(\psi)\right)}{m} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -\frac{u_3\left(\mathrm{c}(\phi)\mathrm{c}(\psi) + \mathrm{s}(\theta)\mathrm{s}(\phi)\mathrm{s}(\psi)\right)}{m} & \frac{u_3\mathrm{c}(\theta)\mathrm{c}(\phi)\mathrm{s}(\psi)}{m} & \frac{u_3\left(\mathrm{s}(\phi)\mathrm{s}(\psi) + \mathrm{c}(\phi)\mathrm{c}(\psi)\mathrm{s}(\theta)\right)}{m} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -\frac{u_3\mathrm{c}(\theta)\mathrm{s}(\phi)}{m} & -\frac{u_3\mathrm{c}(\phi)\mathrm{s}(\theta)}{m} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & q\mathrm{c}(\phi)\mathrm{t}(\theta) - r\mathrm{t}(\theta)\mathrm{s}(\phi) & r\mathrm{c}(\phi)(\mathrm{t}(\theta)^2 + 1) + q\mathrm{s}(\phi)(\mathrm{t}(\theta)^2 + 1) & 0 & 1 & \mathrm{t}(\theta)\mathrm{s}(\phi) & \mathrm{c}(\phi)\mathrm{t}(\theta) \\
0 & 0 & 0 & 0 & 0 & 0 & -r\mathrm{c}(\phi) - q\mathrm{s}(\phi) & 0 & 0 & 0 & \mathrm{c}(\phi) & -\mathrm{s}(\phi) \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{q\mathrm{c}(\phi)}{\mathrm{c}(\theta)} - \frac{r\mathrm{s}(\phi)}{\mathrm{c}(\theta)} & \frac{r\mathrm{c}(\phi)\mathrm{s}(\theta)}{\mathrm{c}(\theta)^2} + \frac{q\mathrm{s}(\theta)\mathrm{s}(\phi)}{\mathrm{c}(\theta)^2} & 0 & 0 & \frac{\mathrm{s}(\phi)}{\mathrm{c}(\theta)} & \frac{\mathrm{c}(\phi)}{\mathrm{c}(\theta)} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{I_{yy}r - I_{zz}r}{I_{xx}} & \frac{I_{yy}q - I_{zz}q}{I_{xx}} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{I_{xx}r - I_{zz}r}{I_{yy}} & 0 & -\frac{I_{xx}p - I_{zz}p}{I_{yy}} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{I_{xx}q - I_{yy}q}{I_{zz}} & \frac{I_{xx}p - I_{yy}p}{I_{zz}} & 0
\end{bmatrix}
$$

Figure 5: The final A matrix, where c, s, and t, are $\cos$, $\sin$, and $\tan$. (This was included as a figure since LaTeX did not allow matrices this wide)

Since the SDF of the Iris model specifies its physical properties to be used in by the simulator, the drone's mass and moment of inertia values were taken from the file to be $m = 1.545$ kg, $I_{xx} = 0.029125 \text{ kg} \cdot \text{m}^2$, $I_{yy} = 0.029125 \text{ kg} \cdot \text{m}^2$, and $I_{zz} = 0.055225 \text{ kg} \cdot \text{m}^2$, and $g = 9.81 \text{ m/s}^2$ was used.

It can be noticed from Eq. 8 that to be able to propagate the drone's state forward, it is necessary to have some knowledge of u_3 through u_6, i.e. the thrust, and roll, pitch, yaw torques. A Linear–quadratic regulator (LQR) controller will be created for this purpose. The LQR controller will take in the drone's current state, compute A using Eq. 9 with $u_3 = mg$, compute B using Eq. 10, and uses Q and R from Eq. 11 to compute the optimal gain matrix for the LQR controller, K, according to Eq. 12, and return a vector of inputs u_3 through u_6 to control the drone*. The LQR logic will be done using MATLAB's `lqr` function.[8]

$$
B = \frac{\partial F}{\partial u} = \begin{bmatrix} \frac{\partial F_1}{\partial u_3} & \cdots & \frac{\partial F_1}{\partial u_6} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_{12}}{\partial u_3} & \cdots & \frac{\partial F_{12}}{\partial u_6} \end{bmatrix}
= \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
(\sin(\phi)\sin(\psi) + \cos(\phi)\cos(\psi)\sin(\theta))/m & 0 & 0 & 0 \\
-(\cos(\psi)\sin(\phi) - \cos(\phi)\sin(\theta)\sin(\psi))/m & 0 & 0 & 0 \\
(\cos(\phi)\cos(\theta))/m & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & I_{xx}^{-1} & 0 & 0 \\
0 & 0 & I_{yy}^{-1} & 0 \\
0 & 0 & 0 & I_{zz}^{-1}
\end{bmatrix} \tag{10}
$$

*Note that Q, R, and K in the LQR controller play a similar role as they do in the Kalman filter, but they are not the same value

$$
\begin{aligned}
Q &= \texttt{diag([100 100 100 100 100 100 0.1 0.1 0.1 10 10 10])} \\
R &= \texttt{diag([0.1 1000 1000 1000])}
\end{aligned}
\tag{11}
$$

$$
\begin{aligned}
&\text{Find } K \text{ such that } u = -Kx \text{ minimizes} J(u) = \int_0^\infty (x^T Q x + u^T R u) dt \\
&\text{subject to the system dynamics } \dot{x} = Ax + Bu
\end{aligned}
\tag{12}
$$

MEASUREMENT MODEL

The distance from the camera to the centroid of the drone was measured by each of the infrared cameras, and roll, pitch, yaw measurements of the drone would directly be measured using the IMU. To find the distance from the cameras to the centroid, the cameras would return the average of the distances of each of the reflective markers from the cameras. This math is done implicitly within the Vicon computer software, and just the ranges are returned. Thus, instead of maintaining a separate state for each of the reflective markers on the drone, it will be assumed that the measurement model uses the coordinates of the centroid to measure the range to the centroid*. The noise for the range measurements will be set as the claimed accuracy on the specsheet of the Vicon Vantage V16. The roll, pitch, and yaw measurements will be used directly from the IMU measurements. The noise for the roll, pitch, yaw measurements will be experimentally determined by find the variance of the IMU measurements when the drone is stationary. The measurement model was then formulated as in Eq. 13, where X_n, Y_n, and Z_n are the X, Y, and Z coordinates of the nth camera.

$$
G(X(t), t) = \begin{bmatrix}
\sqrt{(x - X_1)^2 + (y - Y_1)^2 + (z - Z_1)^2} \\
\sqrt{(x - X_2)^2 + (y - Y_2)^2 + (z - Z_2)^2} \\
\sqrt{(x - X_3)^2 + (y - Y_3)^2 + (z - Z_3)^2} \\
\sqrt{(x - X_4)^2 + (y - Y_4)^2 + (z - Z_4)^2} \\
\phi \\
\theta \\
\psi
\end{bmatrix}
\tag{13}
$$

The observation matrix, H, was found by taking partial of the measurement model, G, with respect to X.

$$
H = \frac{\partial G}{\partial X} = \begin{bmatrix}
\frac{\partial G_1}{\partial x} & \cdots & \frac{\partial G_1}{\partial r} \\
\vdots & \ddots & \vdots \\
\frac{\partial G_7}{\partial x} & \cdots & \frac{\partial G_7}{\partial r}
\end{bmatrix}
\tag{14}
$$

*The alternative to this is to maintain the state of each marker separately, and propagate each state forward individually. However, since the reflective markers are placed almost symmetrically across the axes of the drone-fixed frame, this will offer little to no improvement in the accuracy of the estimates, and that possible slight improvement in accuracy will not outweigh the increase in computational time

$$
H = \frac{\partial G}{\partial X} = \begin{bmatrix}
-\frac{X_1 - x}{\sqrt{(X_1 - x)^2 + (Y_1 - y)^2 + (Z_1 - z)^2}} & -\frac{Y_1 - y}{\sqrt{(X_1 - x)^2 + (Y_1 - y)^2 + (Z_1 - z)^2}} & -\frac{Z_1 - z}{\sqrt{(X_1 - x)^2 + (Y_1 - y)^2 + (Z_1 - z)^2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-\frac{X_2 - x}{\sqrt{(X_2 - x)^2 + (Y_2 - y)^2 + (Z_2 - z)^2}} & -\frac{Y_2 - y}{\sqrt{(X_2 - x)^2 + (Y_2 - y)^2 + (Z_2 - z)^2}} & -\frac{Z_2 - z}{\sqrt{(X_2 - x)^2 + (Y_2 - y)^2 + (Z_2 - z)^2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-\frac{X_3 - x}{\sqrt{(X_3 - x)^2 + (Y_3 - y)^2 + (Z_3 - z)^2}} & -\frac{Y_3 - y}{\sqrt{(X_3 - x)^2 + (Y_3 - y)^2 + (Z_3 - z)^2}} & -\frac{Z_3 - z}{\sqrt{(X_3 - x)^2 + (Y_3 - y)^2 + (Z_3 - z)^2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-\frac{X_4 - x}{\sqrt{(X_4 - x)^2 + (Y_4 - y)^2 + (Z_4 - z)^2}} & -\frac{Y_4 - y}{\sqrt{(X_4 - x)^2 + (Y_4 - y)^2 + (Z_4 - z)^2}} & -\frac{Z_4 - z}{\sqrt{(X_4 - x)^2 + (Y_4 - y)^2 + (Z_4 - z)^2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}
$$

Figure 6: The final H matrix where X_n, Y_n, and Z_n are the X, Y, and Z coordinates of the nth camera (This was included as a figure since LaTeX did not allow matrices this wide)

FILTER SETUP

Using the aforementioned dynamics and measurement model, a hybrid Extended Kalman filter and an Unscented Kalman filter were implemented. The hybrid EKF algorithm that was used for this study can be seen in Algorithm 1. The UKF algorithm used for this study can be seen in Algorithm 2. The MATLAB implementation of these algorithms can be seen in the Appendix*.

EXPERIMENTAL PROCEDURE

The range measurements from the cameras and the roll, pitch, yaw measurements from the IMU were stored as ROSbags, and were read into MATLAB. The a priori state was measured during every experiment. The a priori state covariance, P_0^+ was set to `diag([(0.001)^2 (0.001)^2 (0.001)^2 0 0 0 (3.8785e-5)^2 (3.8785e-5)^2 (3.8785e-5)^2 0 0 0])`, since it is possible that there were errors on the scale of millimeters when measuring the starting position of the drone, $3.8785 \cdot 10^{-5}$ radians of error from the IMU measurements (according to the standard deviation of the measurements when the drone was at rest), and 0 error in the velocities and angular rates since it was at rest. For the Unscented Kalman Filter, the velocity and p, q, r covariances were set as `(0.001)^2` and `(3.8785e-5)^2` respectively, since the matrix must be positive definite. The variance of the Vicon Vantage V16 motion capture cameras was specified to be 1.5 mm in their specsheet.[1] However, such small variances did not show how robust the filters' to non-ideal conditions. Hence, the variances of each of the four cameras' range measurements was set as 0.0015^2 m, 0.015^2 m, 0.002^2 m, and 0.1^2 m to showcase the filters' robustness, and the aforementioned $3.8785 \cdot 10^{-5}$ radians of error was used for the IMU measurements. Hence, the covariance matrix of the measurements was set as `R = diag([0.0015^2 0.015^2 0.002^2 0.1^2 (3.8785e-5)^2 (3.8785e-5)^2 (3.8785e-5)^2])`.

Four types of trajectories were used to analyze the performance of the filters for this study:

(a) Hover - The drone hovers to a height of 3 m and lands (as seen in Figure 7a)

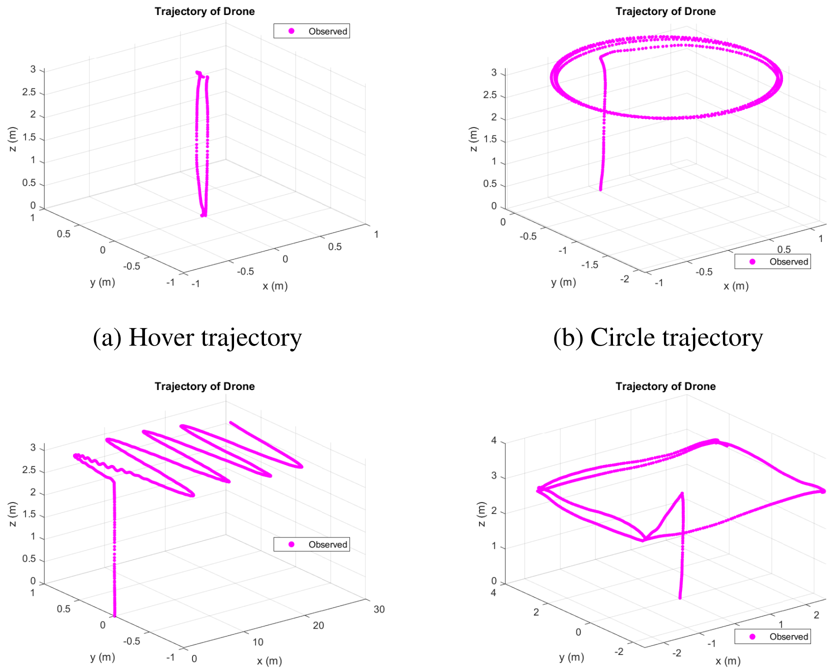
(b) Circle - The drone hovers to a height of 3 m and starts flying in a circle with radius 1 m (as seen in Figure 7b)

(c) Sine - The drone hovers to a height of 3 m and flies in a sine wave in the +x-direction (as seen in Figure 7c)

---

*The code for this study can also be found at

(d) Square - The drone hovers to a height of 3 m and flies in a square with a sidelength of 4 m (as seen in Figure 7d)



(a) Hover trajectory

(b) Circle trajectory

(c) Sine trajectory

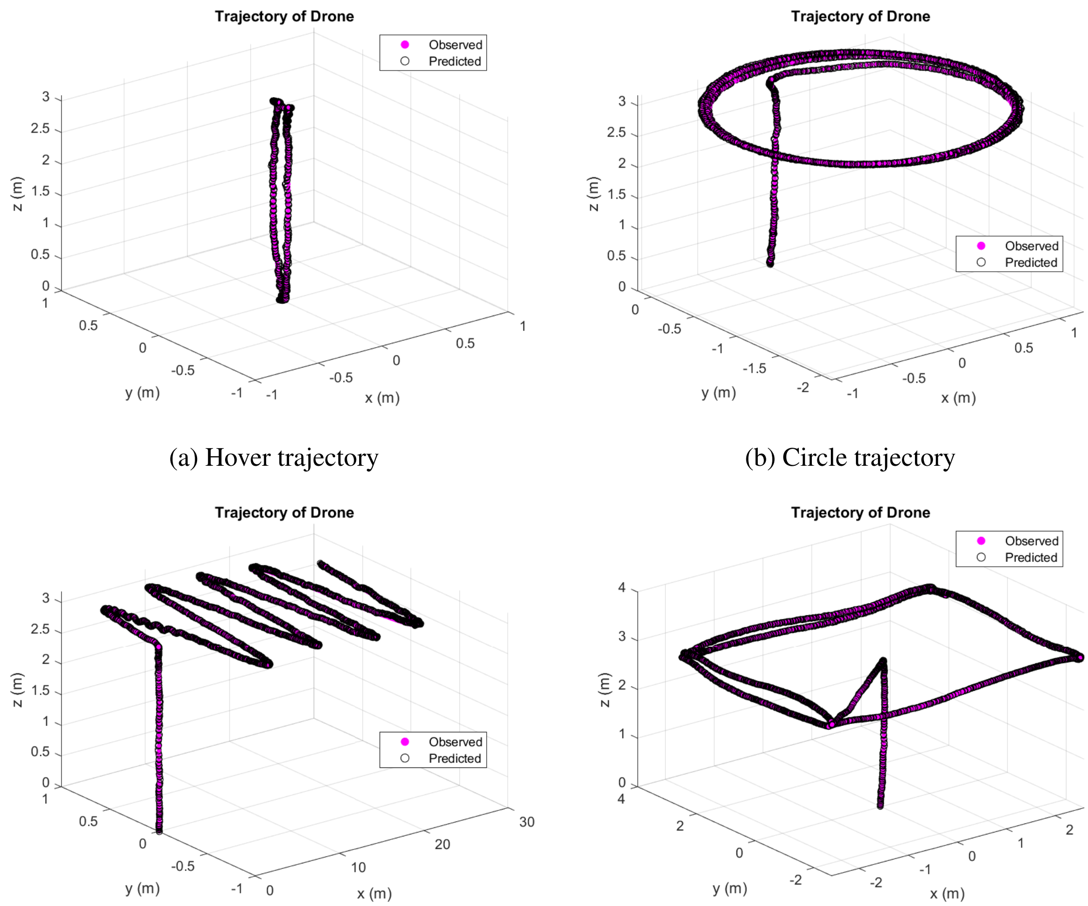
(d) Square trajectory

Figure 7: The trajectories that the drone followed

The default sampling rate of the cameras was at 120 Hz, but the stored data was modified to replicate a measurement rate of 60 Hz, 24 Hz, 12 Hz, 6 Hz, and an extreme of 1 Hz, and the performance of the filters over these measurement rates was comp1ared.

RESULTS AND DISCUSSION

The results of the hybrid Extended Kalman filter will be discussed first. The estimations from the EKF can be seen overlayed on top of the observations in Figure **??**.



(a) Hover trajectory

(b) Circle trajectory

(c) Sine trajectory

(d) Square trajectory

Figure 8: The observed trajectories (magenta) overlayed with the estimated drone position (black)

The standard deviations of errors in position estimates for the different trajectories at various sampling rates can be seen in Tables 3 through 5.

Table 3: Standard deviation of x-position error

Sample Rate \ Trajectory Type	σ_x (in m)			
	Hover	Circle	Square	Sine
120 Hz	0.0042149	0.0084198	0.0083749	0.0054409
60 Hz	0.0049716	0.010091	0.0099932	0.0055549
24 Hz	0.0059826	0.012189	0.011919	0.0070862
12 Hz	0.007542	0.014241	0.012638	0.0084996
6 Hz	0.007107	0.015207	0.012958	0.0091919
1 Hz	0.010384	0.027626	0.015892	0.01356

Table 4: Standard deviation of y-position error

Trajectory Type / Sample Rate	σ_y (in m)			
	Hover	Circle	Square	Sine
120 Hz	0.0042601	0.0083595	0.0083918	0.0052932
60 Hz	0.0050061	0.010083	0.0099853	0.0055125
24 Hz	0.0061169	0.012145	0.011965	0.0072212
12 Hz	0.0073317	0.014331	0.012841	0.0086543
6 Hz	0.0071077	0.014991	0.012912	0.0093219
1 Hz	0.011162	0.028407	0.01622	0.013620

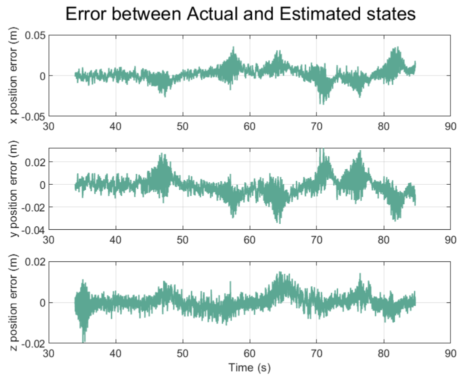
Table 5: Standard deviation of z-position error

Trajectory Type / Sample Rate	σ_z (in m)			
	Hover	Circle	Square	Sine
120 Hz	0.0038442	0.0030611	0.003646	0.011294
60 Hz	0.0032534	0.0032203	0.0041564	0.011154
24 Hz	0.0024082	0.0038679	0.004756	0.015028
12 Hz	0.0025436	0.0038703	0.0056868	0.01889
6 Hz	0.0023293	0.0039609	0.0053087	0.017223
1 Hz	0.018534	0.0077148	0.0068089	0.030209

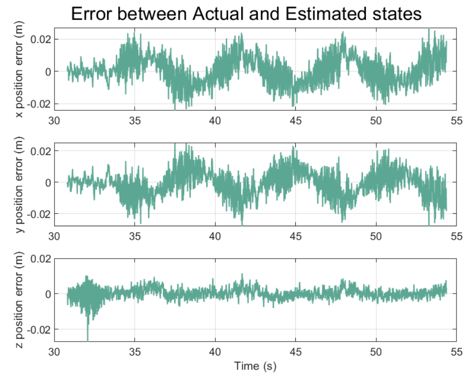
It can be seen from Tables 3 through 5 that the error in actual position and estimated position increases for all trajectories as the sampling rate is decreased. This is intuitive because as the resolution of data decreases, the system has to use its knowledge of the system's dynamics to propagate itself to the next state, and this may differ from the system's real-life behavior since many approximations were made to linearize the system's dynamics. This also explains why the magnitude of error in the z-direction is lower than the error in the x and y directions, since the drone does not move much in the z directions while performing the trajectory, therefore only causing small deviations from the actual. (WHAT) This can also be noticed in Figure 9.

That being said, it can be noticed from Figure 9 that the filter is still accurate to a hundredth of a meter, and having centimeter-level accuracy is incredible when considering that the errors in the measurements from the motion capture cameras were greatly exaggerated (used 1.5 mm, 1.5 cm, 2 mm, and 10 cm in each of the four cameras instead of the rated values of 1.5 mm[1]).

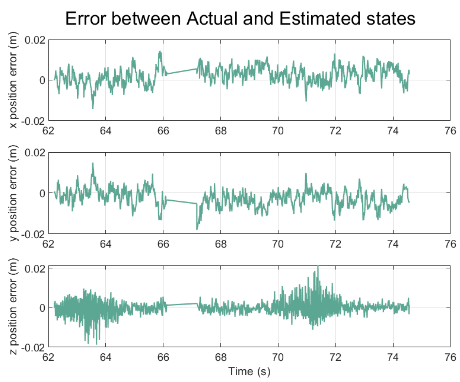
The results of the Unscented Kalman Filter will be discussed next. The first noticeable difference between the EKF and UKF was their runtime. Table 6 shows the runtimes of the EKF and UKF (measured using MATLAB's `tic` and `toc` functions) for different trajectories. The UKF takes 24 times longer per iteration than the EKF, mainly due to generating and propagating sigma points. With an average runtime of 2.9589 seconds, it is not a viable option to put the UKF on an embedded system to control a drone.



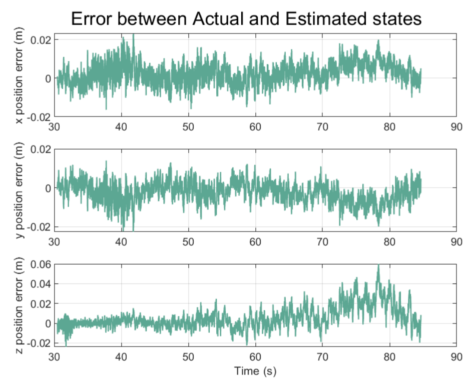
(a) Position error on the square trajectory



(b) Position error on the circle trajectory



(c) Position error on the hover trajectory



(d) Position error on the sine trajectory

Figure 9: Error between predicted and actual positions on different trajectories

Table 6: Runtimes (in seconds) of the EKF and UKF over all trajectories at a sample rate of 6 Hz

Trajectory \ Filter	EKF	UKF
Hover	0.125	2.8026
Circle	0.12666	3.0158
Square	0.11342	2.9119
Sine	0.12442	3.0153
Average	**0.12238**	**2.9589**

Table 7 shows the standard deviation of the position error of the UKF estimates. When compared with the "6 Hz" row of Tables 3 through 5, it can be seen that the UKF estimates are slightly better, with an improvement of approximately 2 mm. This slight improvement in accuracy does not outweigh the UKF's massive increase in runtime, especially considering that the filter will need to be deployed on an embedded system to control the drone.

Table 7: Standard deviation of the position error of UKF estimates at a sample rate of 6 Hz

Trajectory / Direction	Hover	Circle	Square	Sine
σ_x	0.013826	0.010209	0.0094935	0.0099276
σ_y	0.015173	0.010124	0.0094954	0.0095272
σ_z	0.0049896	0.0057478	0.0043911	0.01642
Average	**0.01133**	**0.008694**	**0.010209**	**0.011958**

Now that it has been decided that the EKF is a better option for the application that is being studied, the EKF will be rerun with measurement error values that are more consistent with the real-life specifications of the motion capture cameras. Recall that the variances of the cameras were greatly exaggerated for this study. According to the Vicon Vantage V16's specsheet,[1] the cameras have an error of 1.5 mm. After changing the R matrix to reflect this, the errors of the EKF change as can be seen graphically in Figure 10 or numerically in Table 8. Unsurprisingly, the standard deviations of the position errors are much better when compared to the errors with the perturbed R matrix.

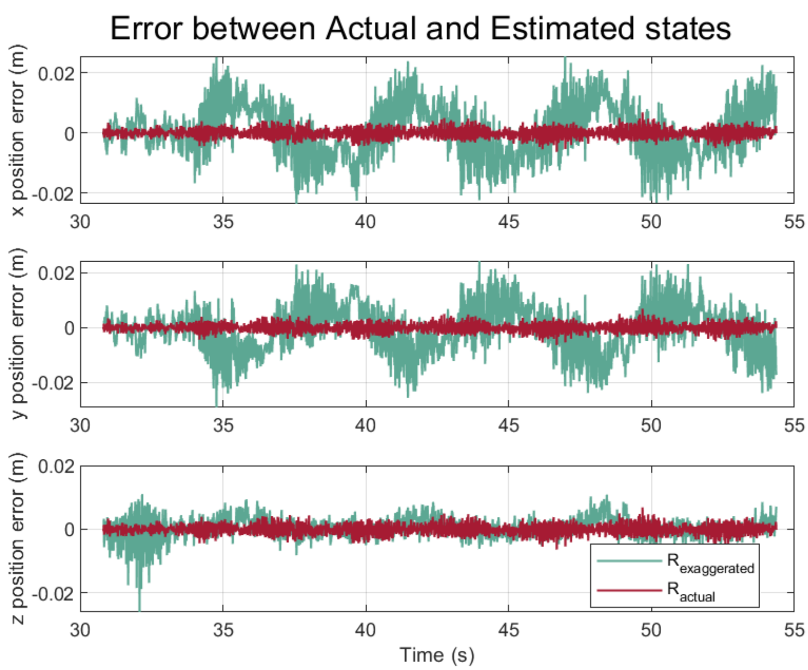


Figure 10: Comparison of the position error when using the actual errors ($R_a = R_{actual}$) v.s. the exaggerated errors ($R_e = R_{exaggerated}$)

Table 8: Standard deviation of the position error when using the actual errors ($R_a = R_{actual}$) v.s. the exaggerated errors ($R_e = R_{exaggerated}$)

	Hover		Circle	
	R_a	R_e	R_a	R_e
σ_x	0.0010586	0.0042149	0.0016466	0.0084198
σ_y	0.0012544	0.0042601	0.0016962	0.0083595
σ_z	0.0023459	0.0038442	0.0017194	0.0030611

	Square		Sine	
	R_a	R_e	R_a	R_e
σ_x	0.0015686	0.0083749	0.0014993	0.0054409
σ_y	0.0015528	0.0083918	0.0012982	0.0052932
σ_z	0.0015894	0.003646	0.0022801	0.011294

CONCLUSION

In this study, a hybrid Extended Kalman filter and an Unscented Kalman filter were implemented on the dynamics of a drone. In order to improve the quality of the study, an LQR controller was used to determine the inputs to the system at each time, thereby increasing the types of trajectories that could be analyzed, as opposed to past studies that just involved the drone hovering or with constant inputs. Simulated measurements from Gazebo were used to test the filter. Through this study, it was noted that even with greatly exaggerated measurement covariances, the EKF was able to predict the position of the drone with centimeter-level accuracy. When the measurement covariances were changed to more closely reflect their actual values, the filter performed exceptional well, showing millimeter-level accuracy. It was also noted that the UKF provided a slight improvement in accuracy, but took 24 times longer than the EKF. The slight improvement in accuracy does not outweigh the massive increase in times, and hence it can be concluded that implementing the UKF on an embedded system is not a viable option. This study proved that the EKF is accurate, quick, and robust to measurement perturbations.

ACKNOWLEDGMENTS

Thanks to Dr. Brian Gunter for a great semester. I'd heard of Kalman filters through my work in the past, but I hadn't known how they functioned. I learned a lot through this class, and the way you taught the subject was very interesting, so much so that I've asked my manager to try to put me on a project involving research Kalman filters this summer. I'm looking forward to working more intricately with Kalman filters and applying what I've learned from this class.

REFERENCES

[1] "Vicon in use: Case studies: Motion capture systems," Jan 2022.

[2] *10-camera Vicon Motion Capture System*. University of Saskatchewan.

[3] Tobias, W. H. says:, T. Says:, C. B. says:, A. Says:, M. says:, J. says:, S. Says:, and T. says:, "Mocap Deck,"

[4]

[5] "Robot operating system,"

[6] "3DR iris - the ready to fly UAV Quadcopter,"

[7] A. Gibiansky, "Andrew Gibiansky - Quadcopter Dynamics and Simulation,"

[8] "Linear-Quadratic Regulator (LQR) design,"

APPENDIX A: ALGORITHMS

Algorithm 1 Hybrid Extended Kalman filter

Recognize system equations $\hat{x} = f(x, u, w, t), y_k = h_k(x_k, v_k), w(t) \sim (0, Q), v_k \sim (0, R)$
Initialize $x_{initial}$, P, Q, R
Set $\hat{x}^+_{k-1} \leftarrow x_{initial}$, $P^+_{k-1} \leftarrow P$
for each measurement y **do**
    Assemble y_{obs} by reading in measurements at t_k
    $dt \leftarrow t_k - t_{k-1}$
    $\hat{x}^-_k \leftarrow$ propagate $\hat{x}^+_{k-1}$ using the dynamics $\dot{\hat{x}}_k$ over dt using ode45
    $A \leftarrow$ assemble according to Eq. 9
    $\dot{P} \leftarrow AP^+_{k-1} + P^+_{k-1}A^T + Q$
    Assemble y_{comp} according to Eq. 13 using $\hat{x}^-_k$
    Assemble H_k according to Eq. 14 using $\hat{x}^-_k$ and coordinates in Table 2
    $P^-_k \leftarrow P^+_{k-1} + \dot{P} \times dt$
    $K_k \leftarrow P^-_k H^T_k (H_k P^-_k H^T_k + R)^{-1}$
    $\hat{x}^+_k \leftarrow \hat{x}^-_k + K_k(y_{obs} - y_{comp}$
    $P^+_k \leftarrow (I_{12} - K_k H_k) P^-_k (I_{12} - K_k H_k)^T + R$
end for

Algorithm 2 Unscented Kalman filter

Recognize system equations $x_{k+1} = f(x_k, u_k, t_k) + w_k, y_k = h(x_k, t_k) + v_k, w(t) \sim (0, Q_k), v_k \sim (0, R_k)$
Initialize $x_{initial}$, P, Q, R
Set $\hat{x}^+_{k-1} \leftarrow x_{initial}$, $P^+_{k-1} \leftarrow P$, $n \leftarrow$ number of states
for each measurement y **do**
    Assemble y_{obs} by reading in measurements at t_k
    $dt \leftarrow t_k - t_{k-1}$
    for j=1:2n **do**
        $\hat{x}^{(i)}_{k-1} \leftarrow \hat{x}^+_{k-1} + \tilde{x}^{(i)}; \quad i = 1, \cdots, 2n$
        $\tilde{x}^{(i)} \leftarrow (\sqrt{nP^+_{k-1}})^T_i; \quad i = 1, \cdots, n$
        $\tilde{x}^{(n+i)} \leftarrow -(\sqrt{nP^+_{k-1}})^T_i; \quad i = 1, \cdots, n$
    end for
    $\hat{x}^{(i)}_k \leftarrow$ propagate $\hat{x}^{(i)}_{k-1}$ using the dynamics $\dot{\hat{x}}^{(i)}_k$ over dt using ode45
    $\hat{x}^-_k \leftarrow \frac{1}{2n} \sum_{i=1}^{2n} \hat{x}^{(i)}_k$
    $P^-_k \leftarrow \frac{1}{2n} \sum_{i=1}^{2n} (\hat{x}^{(i)}_k - \hat{x}^-_k)(\hat{x}^{(i)}_k - \hat{x}^-_k)^T + Q_{k-1}$
    for j=1:2n **do**
        $\hat{x}^{(i)}_k \leftarrow \hat{x}^+_k + \tilde{x}^{(i)}; \quad i = 1, \cdots, 2n$
        $\tilde{x}^{(i)} \leftarrow (\sqrt{nP^+_k})^T_i; \quad i = 1, \cdots, n$
        $\tilde{x}^{(n+i)} \leftarrow -(\sqrt{nP^+_k})^T_i; \quad i = 1, \cdots, n$
    end for
    Assemble $\hat{y}^{(i)}_k$ according to Eq. 13 using $\hat{x}^{(i)}_k$
    $\hat{y}_k \leftarrow \frac{1}{2n} \sum_{i=1}^{2n} \hat{y}^{(i)}_k$
    $P_y \leftarrow \frac{1}{2n} \sum_{i=1}^{2n} (\hat{y}^{(i)}_k - \hat{y}_k)(\hat{y}^{(i)}_k - \hat{y}_k)^T + R_k$
    $P_{xy} \leftarrow \frac{1}{2n} \sum_{i=1}^{2n} (\hat{x}^{(i)}_k - \hat{x}^-_k)(\hat{y}^{(i)}_k - \hat{y}_k)^T$
    $K_k \leftarrow P_{xy} P^{-1}_y$
    $\hat{x}^+_k \leftarrow \hat{x}^-_k + K_k(y_{obs} - \hat{y}_k)$
    $P^+_k \leftarrow P^-_k - K_k P_y K^T_k$
end for

APPENDIX B: ADDITIONAL PLOTS AND DATA

APPENDIX C: MATLAB CODE

```matlab
clc; clear all; close all;

%% Prepping state data

bag_list = ["circle", "hover", "sine", "square"];
% bag_list = ["hover"];
% freq_list = [1 2 5 10 20 60 120];
freq_list = [1];

for bag_idx = 1:length(bag_list)
for freq_idx = 1:length(freq_list)
clc; close all; clearvars -except bag_idx freq_idx bag_list freq_list
% Physical parameters and constants
m = 1.545;
Ixx = 0.029125;
Iyy = 0.029125;
Izz = 0.055225;
g = 9.81;
cam1loc = [10, 10, 10]';
cam2loc = [-10, 10, 10]';
cam3loc = [-10, -10, 10]';
cam4loc = [10, -10, 10]';

% Initial conditions
P = diag([(0.001)^2 (0.001)^2 (0.001)^2 (0.001)^2 (0.001)^2 (0.001)^2 (3.8785e-5)^2 (3.8785e-5)^2 (3.8785e-5)^2 (3.8785e-5)^2 (3.8785e-5)^2 (3.8785e-5)^2]);
Q = eye(12).*1e-3;
% R = diag([0.0015^2 0.015^2 0.002^2 0.1^2 (3.8785e-5)^2 (3.8785e-5)^2 (3.8785e-5)^2]);
R = diag([0.0015^2 0.0015^2 0.0015^2 0.0015^2 (3.8785e-5)^2 (3.8785e-5)^2 (3.8785e-5)^2]);

shape = bag_list(bag_idx);
sample_freq = freq_list(freq_idx);

data = importdata(strcat(shape, "_actual_states.csv"));
measurementdata = importdata(strcat(shape, "_measurements.csv"));

states = [];
target_states = [];
measurements = [];
time = [];
% Truncating data
for i = 1:size(data,2)
    if mod(i,sample_freq)==0
        states = [states data(1:12, i)];
        target_states = [target_states data(13:24, i)];
        measurements = [measurements measurementdata(:, i)+ sqrt(diag(R)).*randn(size(diag(R)))];
        time = [time data(end, i)];
    end
end
x_(:,1) = states(:,1); % Using the first column from the data
time = time./1e9;
disp("Everything in SI, angles in radians")

%%
% Visualizing data
figure(1)
scatter3(states(1,:), states(2,:), states(3,:), 10, 'm', 'filled')
xlabel('x (m)')
ylabel('y (m)')
zlabel('z (m)')
title('Trajectory of Drone')
```

```
legend('Observed', 'Predicted', 'Location', 'best')
if shape == "hover"
    xlim([-1 1])
    ylim([-1 1])
end
if sample_freq == 1
saveas(gcf, strcat(shape, num2str(sample_freq), '_just_traj.png'))
end

store_x = [x_(:,1)];
store_P = [norm(diag(P))];

timeElapsedArray = [];
for i=2:size(measurements,2)
    tic;
    dt = time(i) - time(i-1);
%   Observed
    y_obs(:,i) = measurements(:, i);

%   Propagation of state
    if dt ~= 0
        [t_out, y_out] = ode45(@(t,y) drone_dynamics(t, y, target_states(:,i-1), m, Ixx, Iyy, Izz, g), [0 dt], x_(:, i-1), odeset('RelTol',1e-2,'AbsTol',1e-4));
    end

    x_(:,i) = y_out(end,:)';

    % Propagation of state covariance
    A = find_A(x_(:,i-1), m, Ixx, Iyy, Izz, g);
    Pdot = A*P + P*A' + Q;
    P = P + Pdot*dt;

%   Assembling y_comp
    y_comp(:,i) = [vecnorm(x_(1:3,end)-cam1loc);
                   vecnorm(x_(1:3,end)-cam2loc);
                   vecnorm(x_(1:3,end)-cam3loc);
                   vecnorm(x_(1:3,end)-cam4loc);
                   x_(7,end);
                   x_(8,end);
                   x_(9,end)];

%   Computing H using y_comp
    X1 = cam1loc(1);
    Y1 = cam1loc(2);
    Z1 = cam1loc(3);
    X2 = cam2loc(1);
    Y2 = cam2loc(2);
    Z2 = cam2loc(3);
    X3 = cam3loc(1);
    Y3 = cam3loc(2);
    Z3 = cam3loc(3);
    X4 = cam4loc(1);
    Y4 = cam4loc(2);
    Z4 = cam4loc(3);
    drone_x = x_(1,end);
    drone_y = x_(2,end);
    drone_z = x_(3,end);
    H = [-(X1 - drone_x)/((X1 - drone_x)^2 + (Y1 - drone_y)^2 + (Z1 - drone_z)^2)^(1/2), -(Y1 - drone_y)/((X1 - drone_x)^2 + (Y1 - drone_y)^2 + (Z1 - drone_z)^2)^(1/2), -(Z1 - drone_z)/((X1 - drone_x)^2 + (Y1 - drone_y)^2 + (Z1 - drone_z)^2)^(1/2), 0, 0, 0, 0, 0, 0, 0, 0, 0;
         -(X2 - drone_x)/((X2 - drone_x)^2 + (Y2 - drone_y)^2 + (Z2 - drone_z)^2)^(1/2), -(Y2 - drone_y)/((X2 - drone_x)^2 + (Y2 - drone_y)^2 + (Z2 - drone_z)^2)^(1/2), -(Z2 - drone_z)/((X2 - drone_x)^2 + (Y2 - drone_y)^2 + (Z2 - drone_z)^2)^(1/2), 0, 0, 0, 0, 0, 0, 0, 0, 0;
```

```
            -(X3 - drone_x)/((X3 - drone_x)^2 + (Y3 - drone_y)^2 + (Z3 - drone_z)^2)^(1/2), -(Y3 - drone_y)/((X3 - drone_x)^2 + (Y3 - drone_y)^2 + (Z3 - drone_z)^2)^(1/2), -(Z3 - drone_z)/((X3 - drone_x)^2 + (Y3 - drone_y)^2 + (Z3 - drone_z)^2)^(1/2), 0, 0, 0, 0, 0, 0, 0, 0, 0;
            -(X4 - drone_x)/((X4 - drone_x)^2 + (Y4 - drone_y)^2 + (Z4 - drone_z)^2)^(1/2), -(Y4 - drone_y)/((X4 - drone_x)^2 + (Y4 - drone_y)^2 + (Z4 - drone_z)^2)^(1/2), -(Z4 - drone_z)/((X4 - drone_x)^2 + (Y4 - drone_y)^2 + (Z4 - drone_z)^2)^(1/2), 0, 0, 0, 0, 0, 0, 0, 0, 0;
            0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0;
            0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0;
            0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0];

%   Kalman gain
    K = P*H'*inv(H*P*H'+R);

    % Measurement update
    x_(:,i) = x_(:,i) + K * (y_obs(:,i) - y_comp(:,i));

    P = (eye(12)-K*H)*P*(eye(12)-K*H)' + K*R*K';

    store_x = [store_x x_(:, i)];
    store_P = [store_P norm(diag(P))];

    timeElapsed = toc;
    timeElapsedArray = [timeElapsedArray timeElapsed];

    fprintf("%g%% done\n", round(i/size(measurements,2)*100))
end
figure(1)
hold on
scatter3(x_(1,:), x_(2,:), x_(3,:), 20, 'black')
legend('Observed', 'Predicted', 'Location', 'best')
hold off;
if shape == "hover"
    xlim([-1 1])
    ylim([-1 1])
end
saveas(gcf, strcat(shape, num2str(sample_freq), '_BETTER_traj.png'))

err = store_x - states;

figure(2)
subplot(3,1,1)
hold on
plot(time, store_x(1,:), 'LineWidth', 1, 'Color', '#da7e30')
plot(time, states(1,:), '--', 'LineWidth', 1, 'Color', '#6b4c9a')
box on
grid on
ylabel("x position (m)")
subplot(3,1,2)
hold on
plot(time, store_x(2,:), 'LineWidth', 1, 'Color', '#da7e30')
plot(time, states(2,:), '--', 'LineWidth', 1, 'Color', '#6b4c9a')
box on
grid on
ylabel("y position (m)")
subplot(3,1,3)
hold on
plot(time, store_x(3,:), 'LineWidth', 1, 'Color', '#da7e30')
plot(time, states(3,:), '--', 'LineWidth', 1, 'Color', '#6b4c9a')
box on
grid on
ylabel("z position (m)")
xlabel("Time (s)")
legend("Estimated", "Actual", "Location", "best")
sgtitle("Comparison of Actual and Estimated states")
```

```
saveas(gcf, strcat(shape, num2str(sample_freq), '_BETTER_compare.png'))

figure(3)
subplot(3,1,1)
hold on
plot(time, err(1,:), 'LineWidth', 1, 'Color', '#5ca793')
box on
grid on
ylabel("x position error (m)")
subplot(3,1,2)
hold on
plot(time, err(2,:), 'LineWidth', 1, 'Color', '#5ca793')
box on
grid on
ylabel("y position error (m)")
subplot(3,1,3)
hold on
plot(time, err(3,:), 'LineWidth', 1, 'Color', '#5ca793')
box on
grid on
ylabel("z position error (m)")
xlabel("Time (s)")
sgtitle("Error between Actual and Estimated states")
saveas(gcf, strcat(shape, num2str(sample_freq), '_BETTER_err.png'))

writematrix([std(err(1,:));
             std(err(2,:));
             std(err(3,:));
             strcat(num2str(1/mean(time(2:end) - time(1:end-1))*1.2), "Hz");
             strcat(num2str(mean(timeElapsedArray)),"s per timestep")], strcat(shape, num2str(sample_freq), '_BETTER_std.csv'))
end
end

%% Functions
function [u3 u4 u5 u6] = get_u(states, target_states, m, Ixx, Iyy, Izz, g)
    q7 = states(7);
    q8 = states(8);
    q9 = states(9);
    q10 = states(10);
    q11 = states(11);
    q12 = states(12);
    u3 = m*g;
    A = find_A(states, m, Ixx, Iyy, Izz, g);
    B = [0, 0, 0, 0;
         0, 0, 0, 0;
         0, 0, 0, 0;
         (sin(q7) * sin(q9) + cos(q7) * cos(q9) * sin(q8)) / m, 0, 0, 0;
         -(cos(q9) * sin(q7) - cos(q7) * sin(q8) * sin(q9)) / m, 0, 0, 0;
         (cos(q7) * cos(q8)) / m, 0, 0, 0;
         0, 0, 0, 0;
         0, 0, 0, 0;
         0, 0, 0, 0;
         0, 1 / Ixx, 0, 0;
         0, 0, 1 / Iyy, 0;
         0, 0, 0, 1 / Izz];
    Q = diag([100 100 100 100 100 100 0.1 0.1 0.1 10 10 10]);
    R = diag([0.1 1000 1000 1000]);
    K = lqr(A, B, Q, R);
    u = -K*(states - target_states);
    u(1) = u(1) + m*g;
    u3 = u(1);
    u4 = u(2);
    u5 = u(3);
    u6 = u(4);
end
```

```
function A = find_A(states, m, Ixx, Iyy, Izz, g)
    q7 = states(7);
    q8 = states(8);
    q9 = states(9);
    q10 = states(10);
    q11 = states(11);
    q12 = states(12);
    u3 = m*g;
    A = [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0;
         0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0;
         0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0;
         0, 0, 0, 0, 0, 0, (u3 * (cos(q7) * sin(q9) - cos(q9) * sin(q7) * sin(q8))) / m, ...
                           (u3 * cos(q7) * cos(q8) * cos(q9)) / m, ...
                           (u3 * (cos(q9) * sin(q7) - cos(q7) * sin(q8) * sin(q9))) / m, ...
                           0, 0, 0;
         0, 0, 0, 0, 0, 0, -(u3 * (cos(q7) * cos(q9) + sin(q7) * sin(q8) * sin(q9))) / m, ...
                           (u3 * cos(q7) * cos(q8) * sin(q9)) / m, ...
                           (u3 * (sin(q7) * sin(q9) + cos(q7) * cos(q9) * sin(q8))) / m, ...
                           0, 0, 0;
         0, 0, 0, 0, 0, 0, -(u3 * cos(q8) * sin(q7)) / m, ...
                           -(u3 * cos(q7) * sin(q8)) / m, ...
                           0, 0, 0, 0;
         0, 0, 0, 0, 0, 0, q11 * cos(q7) * tan(q8) - q12 * sin(q7) * tan(q8), ...
                           q12 * cos(q7) * (tan(q8)^2 + 1) + q11 * sin(q7) * (tan(q8)^2 + 1), ...
                           0, 1, sin(q7) * tan(q8), cos(q7) * tan(q8);
         0, 0, 0, 0, 0, 0, -q12 * cos(q7) - q11 * sin(q7), 0, 0, 0, cos(q7), -sin(q7);
         0, 0, 0, 0, 0, 0, (q11 * cos(q7)) / cos(q8) - (q12 * sin(q7)) / cos(q8),  ...
                           (q12 * cos(q7) * sin(q8)) / cos(q8)^2 + (q11 * sin(q7) * sin(q8)) / cos(q8)^2, ...
                           0, 0, sin(q7) / cos(q8), cos(q7) / cos(q8);
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, (Iyy * q12 - Izz * q12) / Ixx, ...
                                       (Iyy * q11 - Izz * q11) / Ixx;
         0, 0, 0, 0, 0, 0, 0, 0, 0, -(Ixx * q12 - Izz * q12) / Iyy, ...
                                    0, -(Ixx * q10 - Izz * q10) / Iyy;
         0, 0, 0, 0, 0, 0, 0, 0, 0, (Ixx * q11 - Iyy * q11) / Izz, ...
                                    (Ixx * q10 - Iyy * q10) / Izz, 0];
end

function xhatdot = drone_dynamics(t, x, target_state, m, Ixx, Iyy, Izz, g)
    x_pos = x(1);
    y_pos = x(2);
    z_pos = x(3);
    x_vel = x(4);
    y_vel = x(5);
    z_vel = x(6);
    roll = x(7);
    pitch = x(8);
    yaw = x(9);
    roll_rate = x(10);
    pitch_rate = x(11);
    yaw_rate = x(12);
    [u3, u4, u5, u6] = get_u(x, target_state, m, Ixx, Iyy, Izz, g);

    xhatdot =[x_vel;
              y_vel;
              z_vel;
              (u3*(sin(roll)*sin(yaw) + cos(roll)*cos(yaw)*sin(pitch)))/m;
              -(u3*(cos(yaw)*sin(roll) - cos(roll)*sin(pitch)*sin(yaw)))/m;
              -(g*m - u3*cos(pitch)*cos(roll))/m;
              roll_rate + yaw_rate*cos(roll)*tan(pitch) + pitch_rate*tan(pitch)*sin(roll);
```

```
                pitch_rate*cos(roll) - yaw_rate*sin(roll);
                (yaw_rate*cos(roll))/cos(pitch) + (pitch_rate*sin(roll))/cos(pitch);
                (u4 + Iyy*pitch_rate*yaw_rate - Izz*pitch_rate*yaw_rate)/Ixx;
                (u5 - Ixx*roll_rate*yaw_rate + Izz*roll_rate*yaw_rate)/Iyy;
                (u6 + Ixx*pitch_rate*roll_rate - Iyy*pitch_rate*roll_rate)/Izz];

end
```

```
clc; clear all; close all;

%% Prepping state data
bag_list = ["circle", "hover", "sine", "square"];
% freq_list = [1 2 5 10 20 60 120];
freq_list = [20];

for bag_idx = 1:length(bag_list)
for freq_idx = 1:length(freq_list)
clc; close all; clearvars -except bag_idx freq_idx bag_list freq_list
% Physical parameters and constants
m = 1.545;
Ixx = 0.029125;
Iyy = 0.029125;
Izz = 0.055225;
g = 9.81;
cam1loc = [10, 10, 10]';
cam2loc = [-10, 10, 10]';
cam3loc = [-10, -10, 10]';
cam4loc = [10, -10, 10]';

R = diag([0.0015^2 0.015^2 0.002^2 0.1^2 (3.8785e-5)^2 (3.8785e-5)^2 (3.8785e-5)^2]);

shape = bag_list(bag_idx);
sample_freq = freq_list(freq_idx);


data = importdata(strcat(shape, "_actual_states.csv"));
measurementdata = importdata(strcat(shape, "_measurements.csv"));

states = [];
target_states = [];
measurements = [];
time = [];
% Truncating data
for i = 1:size(data,2)
    if mod(i,sample_freq)==0
        states = [states data(1:12, i)];
        target_states = [target_states data(13:24, i)];
        measurements = [measurements measurementdata(:, i)+
    ↪  sqrt(diag(R)).*randn(size(diag(R)))];
        time = [time data(end, i)];
    end
end
time = time./1e9;

% Initial conditions
P = diag([(0.001)^2 (0.001)^2 (0.001)^2 (0.001)^2 (0.001)^2 (0.001)^2 (3.8785e-5)^2
    ↪  (3.8785e-5)^2 (3.8785e-5)^2 (3.8785e-5)^2 (3.8785e-5)^2 (3.8785e-5)^2]);
Q = eye(12).*1e-3;
x_(:,1) = states(:,1); % Using the first column from the data
n = length(x_);
disp("Everything in SI, angles in radians")

%%
% Visualizing data
figure(1)
scatter3(states(1,:), states(2,:), states(3,:), 10, 'm', 'filled')
```

```
xlabel('x (m)')
ylabel('y (m)')
zlabel('z (m)')
title('Trajectory of Drone')
% legend('Observed', 'Predicted', 'Location', 'best')
if shape == "hover"
    xlim([-1 1])
    ylim([-1 1])
end
if sample_freq == 1
saveas(gcf, strcat(shape, num2str(sample_freq), '_just_traj.png'))
end

store_x = [x_(:,1)];
store_P = [norm(diag(P))];
timeElapsedArray = [];
for i=2:size(measurements,2)
    tic;
    dt = time(i) - time(i-1);
    % Observed
    y_obs(:,i) = measurements(:, i);

    xhat_k_1 = [];
    % Sigma points
    for j = 1:2*n
        if j<=n
            xtilda = chol(n*P);
            xtilda = xtilda(j,:)';
        else
            xtilda = -chol(n*P);
            xtilda = xtilda(j-n,:)';
        end
        xhat_k_1 = [xhat_k_1 x_(:,i-1)+xtilda];
    end

    % Propagation of state
    xhat_k = [];
    for j = 1:2*n
        current_x = xhat_k_1(:,j);
        if dt ~= 0
            [t_out, y_out] = ode45(@(t,y) drone_dynamics(t, y, target_states(:,i-1), m,
↪   Ixx, Iyy, Izz, g), [0 dt], current_x, odeset('RelTol',1e-2,'AbsTol',1e-4));
        end
        xhat_k = [xhat_k y_out(end,:)'];
    end
    xhatk_ = sum(xhat_k, 2)/(2*n);
    temp_Pk_ = 0;
    for j = 1:2*n
        temp_Pk_ = temp_Pk_ + (xhat_k(:,j) - xhatk_)*(xhat_k(:,j) - xhatk_)';
    end
    Pk_ = temp_Pk_/(2*n) + Q;

%     xhat_k = [];
%     % Sigma points
%     for j = 1:2*n
%         if j<=n
%             xtilda = chol(n*P);
%             xtilda = xtilda(j,:)';
%         else
%             xtilda = -chol(n*P);
%             xtilda = xtilda(j-n,:)';
%         end
%         xhat_k = [xhat_k xhatk_+xtilda];
%     end

    % Assembling y_computed
```

```
    y_comp = [];
    for j = 1:2*n
        current_x = xhat_k(:,j);
        y1_comp = vecnorm(current_x(1:3)-cam1loc);
        y2_comp = vecnorm(current_x(1:3)-cam2loc);
        y3_comp = vecnorm(current_x(1:3)-cam3loc);
        y4_comp = vecnorm(current_x(1:3)-cam4loc);

        y_comp = [y_comp [y1_comp; y2_comp; y3_comp; y4_comp; current_x(7); current_x(8);
    current_x(9)]];
    end
    y_comp_hat = sum(y_comp, 2)/(2*n);

    temp_Py = 0;
    temp_Pxy = 0;
    for j = 1:2*n
        temp_Py = temp_Py + (y_comp(:,j) - y_comp_hat)*(y_comp(:,j) - y_comp_hat)';
        temp_Pxy = temp_Pxy + (xhat_k(:,j)-xhatk_)*(y_comp(:,j)-y_comp_hat)';
    end
    Py = temp_Py/(2*n) + R;
    Pxy = temp_Pxy/(2*n);

    K = Pxy*inv(Py);
    x_(:,i) = xhatk_ + K*(y_obs(:,i) - y_comp_hat);
    P = Pk_ - K*Py*K';

    store_x = [store_x x_(:, i)];
    timeElapsed = toc;
    timeElapsedArray = [timeElapsedArray timeElapsed];
    figure(1)
    hold on
    scatter3(x_(1,i), x_(2,i), x_(3,i), 20, 'black')
    fprintf("%g%% done\n", round(i/size(measurements,2)*100))
end

figure(1)
hold on
scatter3(x_(1,:), x_(2,:), x_(3,:), 20, 'black')
legend('Observed', 'Predicted', 'Location', 'best')
hold off;
if shape == "hover"
    xlim([-1 1])
    ylim([-1 1])
end
saveas(gcf, strcat(shape, num2str(sample_freq), '_traj_UKF.png'))

err = store_x - states;

figure(2)
subplot(3,1,1)
hold on
plot(time, store_x(1,:), 'LineWidth', 1, 'Color', '#da7e30')
plot(time, states(1,:), '--', 'LineWidth', 1, 'Color', '#6b4c9a')
box on
grid on
ylabel("x position (m)")
subplot(3,1,2)
hold on
plot(time, store_x(2,:), 'LineWidth', 1, 'Color', '#da7e30')
plot(time, states(2,:), '--', 'LineWidth', 1, 'Color', '#6b4c9a')
box on
grid on
ylabel("y position (m)")
subplot(3,1,3)
hold on
plot(time, store_x(3,:), 'LineWidth', 1, 'Color', '#da7e30')
```

```
plot(time, states(3,:), '--', 'LineWidth', 1, 'Color', '#6b4c9a')
box on
grid on
ylabel("z position (m)")
xlabel("Time (s)")
legend("Estimated", "Actual", "Location", "best")
sgtitle("Comparison of Actual and Estimated states")
saveas(gcf, strcat(shape, num2str(sample_freq), '_compare_UKF.png'))

figure(3)
subplot(3,1,1)
hold on
plot(time, err(1,:), 'LineWidth', 1, 'Color', '#5ca793')
box on
grid on
ylabel("x position error (m)")
subplot(3,1,2)
hold on
plot(time, err(2,:), 'LineWidth', 1, 'Color', '#5ca793')
box on
grid on
ylabel("y position error (m)")
subplot(3,1,3)
hold on
plot(time, err(3,:), 'LineWidth', 1, 'Color', '#5ca793')
box on
grid on
ylabel("z position error (m)")
xlabel("Time (s)")
sgtitle("Error between Actual and Estimated states")
saveas(gcf, strcat(shape, num2str(sample_freq), '_err_UKF.png'))

writematrix([std(err(1,:));std(err(2,:));std(err(3,:));strcat(num2str(1/mean(time(2:end) - time(1:end-1))*1.2), "Hz"); strcat(num2str(mean(timeElapsedArray)),"s per timestep")], strcat(shape, num2str(sample_freq), '_std_UKF.csv'))
end
end

%% Functions

function xhatdot = drone_dynamics(t, x, target_state, m, Ixx, Iyy, Izz, g)
    x_pos = x(1);
    y_pos = x(2);
    z_pos = x(3);
    x_vel = x(4);
    y_vel = x(5);
    z_vel = x(6);
    roll = x(7);
    pitch = x(8);
    yaw = x(9);
    roll_rate = x(10);
    pitch_rate = x(11);
    yaw_rate = x(12);
    [u3, u4, u5, u6] = get_u(x, target_state, m, Ixx, Iyy, Izz, g);

    xhatdot =[x_vel;
              y_vel;
              z_vel;
              (u3*(sin(roll)*sin(yaw) + cos(roll)*cos(yaw)*sin(pitch)))/m;
              -(u3*(cos(yaw)*sin(roll) - cos(roll)*sin(pitch)*sin(yaw)))/m;
              -(g*m - u3*cos(pitch)*cos(roll))/m;
              roll_rate + yaw_rate*cos(roll)*tan(pitch) + pitch_rate*tan(pitch)*sin(roll);
              pitch_rate*cos(roll) - yaw_rate*sin(roll);
              (yaw_rate*cos(roll))/cos(pitch) + (pitch_rate*sin(roll))/cos(pitch);
              (u4 + Iyy*pitch_rate*yaw_rate - Izz*pitch_rate*yaw_rate)/Ixx;
              (u5 - Ixx*roll_rate*yaw_rate + Izz*roll_rate*yaw_rate)/Iyy;
```

```
               (u6 + Ixx*pitch_rate*roll_rate - Iyy*pitch_rate*roll_rate)/Izz];

end

function [u3 u4 u5 u6] = get_u(states, target_states, m, Ixx, Iyy, Izz, g)
    q7 = states(7);
    q8 = states(8);
    q9 = states(9);
    q10 = states(10);
    q11 = states(11);
    q12 = states(12);
    u3 = m*g;
    A = find_A(states, m, Ixx, Iyy, Izz, g);
    B = [0, 0, 0, 0;
         0, 0, 0, 0;
         0, 0, 0, 0;
         (sin(q7) * sin(q9) + cos(q7) * cos(q9) * sin(q8)) / m, 0, 0, 0;
         -(cos(q9) * sin(q7) - cos(q7) * sin(q8) * sin(q9)) / m, 0, 0, 0;
         (cos(q7) * cos(q8)) / m, 0, 0, 0;
         0, 0, 0, 0;
         0, 0, 0, 0;
         0, 0, 0, 0;
         0, 1 / Ixx, 0, 0;
         0, 0, 1 / Iyy, 0;
         0, 0, 0, 1 / Izz];
    Q = diag([100 100 100 100 100 100 0.1 0.1 0.1 10 10 10]);
    R = diag([0.1 1000 1000 1000]);
    K = lqr(A, B, Q, R);
    u = -K*(states - target_states);
    u(1) = u(1) + m*g;
    u3 = u(1);
    u4 = u(2);
    u5 = u(3);
    u6 = u(4);
end

function A = find_A(states, m, Ixx, Iyy, Izz, g)
    q7 = states(7);
    q8 = states(8);
    q9 = states(9);
    q10 = states(10);
    q11 = states(11);
    q12 = states(12);
    u3 = m*g;
    A = [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0;
         0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0;
         0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0;
         0, 0, 0, 0, 0, 0, (u3 * (cos(q7) * sin(q9) - cos(q9) * sin(q7) * sin(q8))) / m, ...
                           (u3 * cos(q7) * cos(q8) * cos(q9)) / m, ...
                           (u3 * (cos(q9) * sin(q7) - cos(q7) * sin(q8) * sin(q9))) / m, ...
                           0, 0, 0;
         0, 0, 0, 0, 0, 0, -(u3 * (cos(q7) * cos(q9) + sin(q7) * sin(q8) * sin(q9))) / m, ...
                           (u3 * cos(q7) * cos(q8) * sin(q9)) / m, ...
                           (u3 * (sin(q7) * sin(q9) + cos(q7) * cos(q9) * sin(q8))) / m, ...
                           0, 0, 0;
         0, 0, 0, 0, 0, 0, -(u3 * cos(q8) * sin(q7)) / m, ...
                           -(u3 * cos(q7) * sin(q8)) / m, ...
                           0, 0, 0, 0;
         0, 0, 0, 0, 0, 0, q11 * cos(q7) * tan(q8) - q12 * sin(q7) * tan(q8), ...
                           q12 * cos(q7) * (tan(q8)^2 + 1) + q11 * sin(q7) * (tan(q8)^2 + 1), ...
                           0, 1, sin(q7) * tan(q8), cos(q7) * tan(q8);
```

```
         0, 0, 0, 0, 0, 0, -q12 * cos(q7) - q11 * sin(q7), 0, 0, 0, cos(q7), -sin(q7);
         0, 0, 0, 0, 0, 0, (q11 * cos(q7)) / cos(q8) - (q12 * sin(q7)) / cos(q8),  ...
                           (q12 * cos(q7) * sin(q8)) / cos(q8)^2 + (q11 * sin(q7) *
    ↪  sin(q8)) / cos(q8)^2, ...
                           0, 0, sin(q7) / cos(q8), cos(q7) / cos(q8);
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, (Iyy * q12 - Izz * q12) / Ixx, ...
                                       (Iyy * q11 - Izz * q11) / Ixx;
         0, 0, 0, 0, 0, 0, 0, 0, 0, -(Ixx * q12 - Izz * q12) / Iyy, ...
                                    0, -(Ixx * q10 - Izz * q10) / Iyy;
         0, 0, 0, 0, 0, 0, 0, 0, 0, (Ixx * q11 - Iyy * q11) / Izz, ...
                                    (Ixx * q10 - Iyy * q10) / Izz, 0];
end
```