Overview

This project involved the creation and implementation of an Extended Kalman Filter (EKF) to predict a quadrotors state.

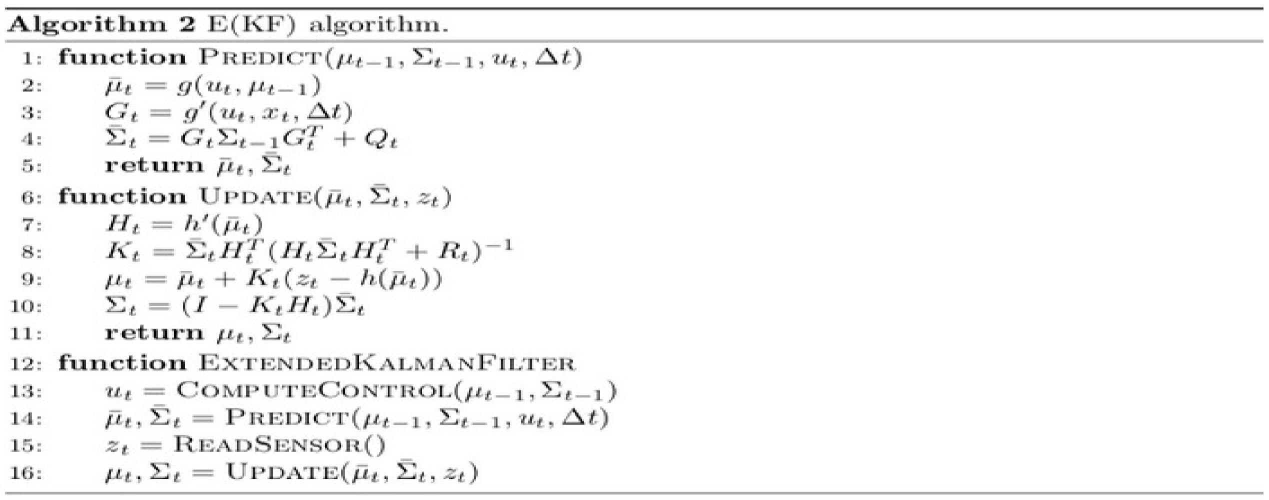
Instead of having a full 12 variable kalman filter to predict all 12 3D states of the vehicle, a 7 variable[x,y,z, x_dot, y_dot, z_dot, psi] EKF was used instead.

To simplify the system, first the body rates of the vehicle were used as prediction input instead of a state variable. These were used in a complementary filter that predicts the roll and pitch angles[phi, theta].

Another simplification is the use of the IMU in the prediction step instead of in the update step.

The results of this estimation was a quadrotor that could loosely follow the planned trajectory. More tuning will be needed to improve the effectiveness with which the vehicle follows the trajectory.

Below is the pseudocode for the EKF algorithm as it was implemented. The primary work done was to define the transition and measurement models g and h, and their derivatives g' and h'.



Algorithm 2 E(KF) algorithm.

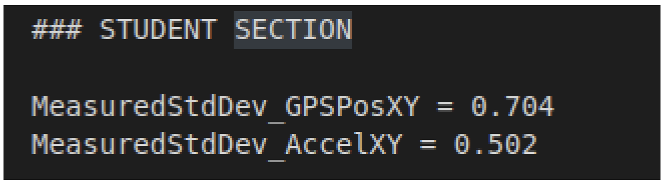
```
function PREDICT($\mu_{t-1}, \Sigma_{t-1}, u_t, \Delta t$)
    $\bar{\mu}_t = g(u_t, \mu_{t-1})$
    $G_t = g'(u_t, x_t, \Delta t)$
    $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + Q_t$
    return $\bar{\mu}_t, \bar{\Sigma}_t$
function UPDATE($\bar{\mu}_t, \bar{\Sigma}_t, z_t$)
    $H_t = h'(\bar{\mu}_t)$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

Rubic points:

Determine the standard deviation of the measurement noise of both GPS X data and Accelerometer X data.

To determine the standard deviation of the measurement noise of both GPS X data and Accelerometer X data data was collected and processed. This was done by importing the data into matlab and calculating the standard deviation of the measured data is the std(data) function.

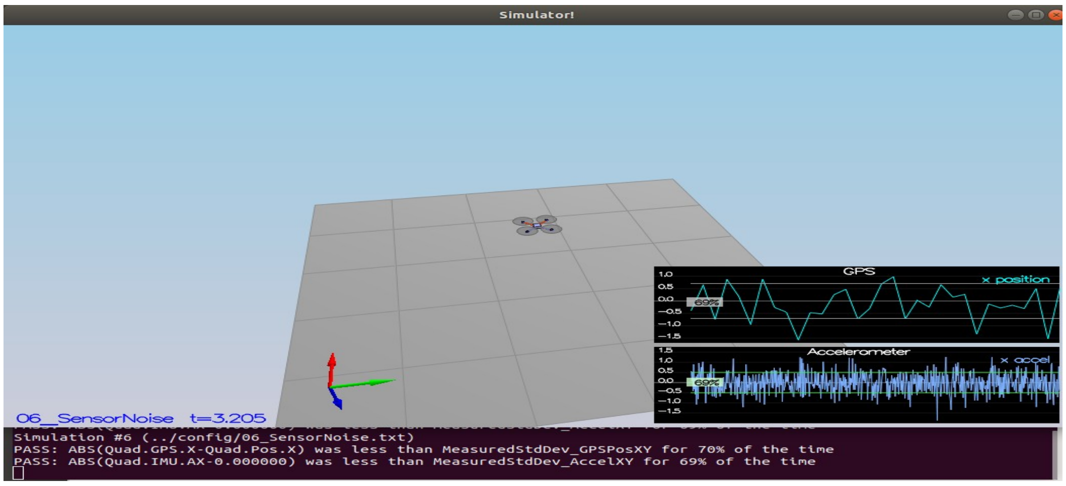
The GPS data sample had a standard deviation of 0.704 cm and the Accelerometer data sample had a standard deviation of 0.502 cm/s/s.



```
### STUDENT SECTION

MeasuredStdDev_GPSPosXY = 0.704
MeasuredStdDev_AccelXY = 0.502
```

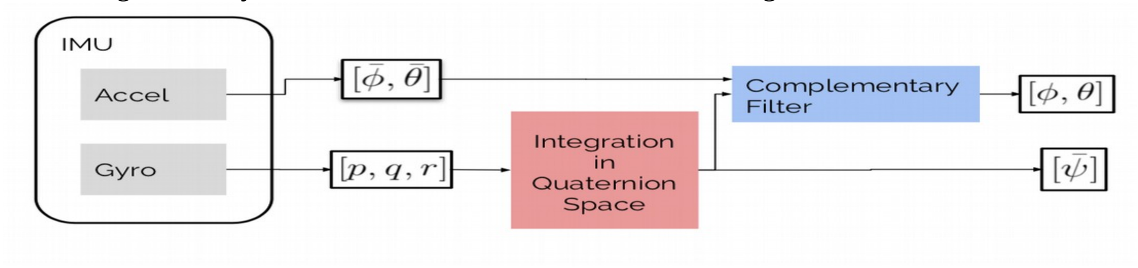
The GPS data sample had a standard deviation of 70% and the Accelerometer data sample had a standard deviation of 69%.



```
Simulation #6 (../config/06_SensorNoise.txt)
PASS: ABS(Quad.GPS.X-Quad.Pos.X) was less than MeasuredStdDev_GPSPosXY for 70% of the time
PASS: ABS(Quad.IMU.AX-0.000000) was less than MeasuredStdDev_AccelXY for 69% of the time
```

Implement a better rate gyro attitude integration scheme in the `UpdateFromIMU()` function.

The Complementary filter used in this project fuses Inertial Measurement Unit (IMU) data from the accelerometer and Gyro to determine pitch and roll angles. This is done using a weighted average GPS attitude and integrated body rates to achieve a new estimate. This is diagrammed below.

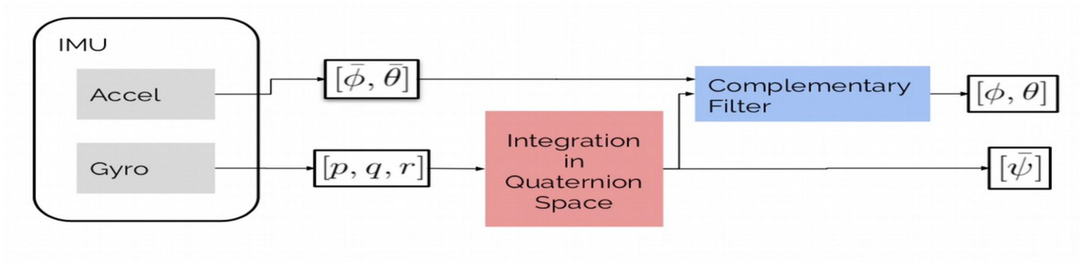


The implementation that was given assumed that the vehicle had small angular displacements from zero, which isn't always the case. In order to more accurately integrate the gyro sensor data, the current attitude of the vehicle should be taken into account. The integration scheme was improved by performing the integration in the nonlinear quaternion space.

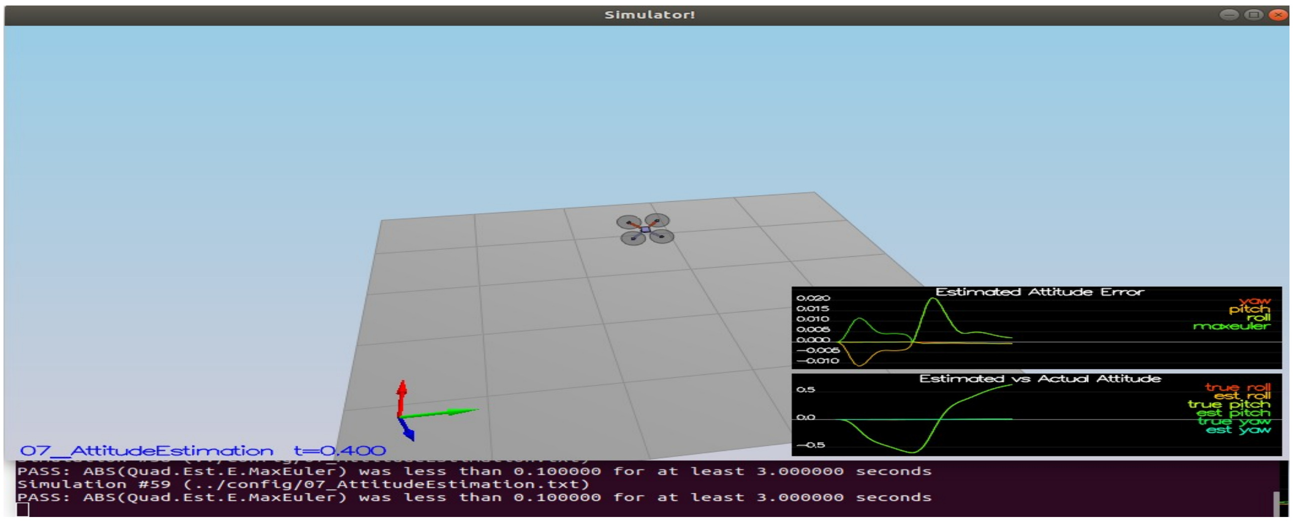
1. The current heading was converted to a quaternion q_t
2. The body rates were integrated to get a change in angle *dq* and then converted to quaternion space.
3. The new attitude estimate came from rotating q_t by *dq* by multiplying them together. dq x q_t

This results in much more accurate estimations of the attitude of the vehicle.

The modified flow chart looks like the following, where the integration step is done in the quaternion space before the estimates are filtered together.



The implemented attitude filter output looks like below.



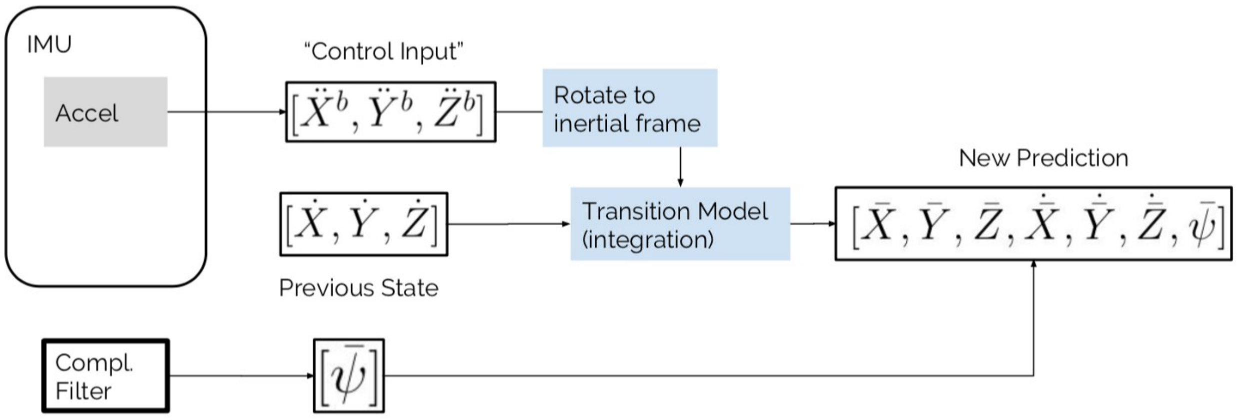
```
PASS: ABS(Quad.Est.E.MaxEuler) was less than 0.100000 for at least 3.000000 seconds
Simulation #59 (../config/07_AttitudeEstimation.txt)
PASS: ABS(Quad.Est.E.MaxEuler) was less than 0.100000 for at least 3.000000 seconds
```

Implement all of the elements of the prediction step for the estimator.

The state prediction is handled in `PredictState()` and the covariance prediction is handled in `Predict()`.

PredictState()

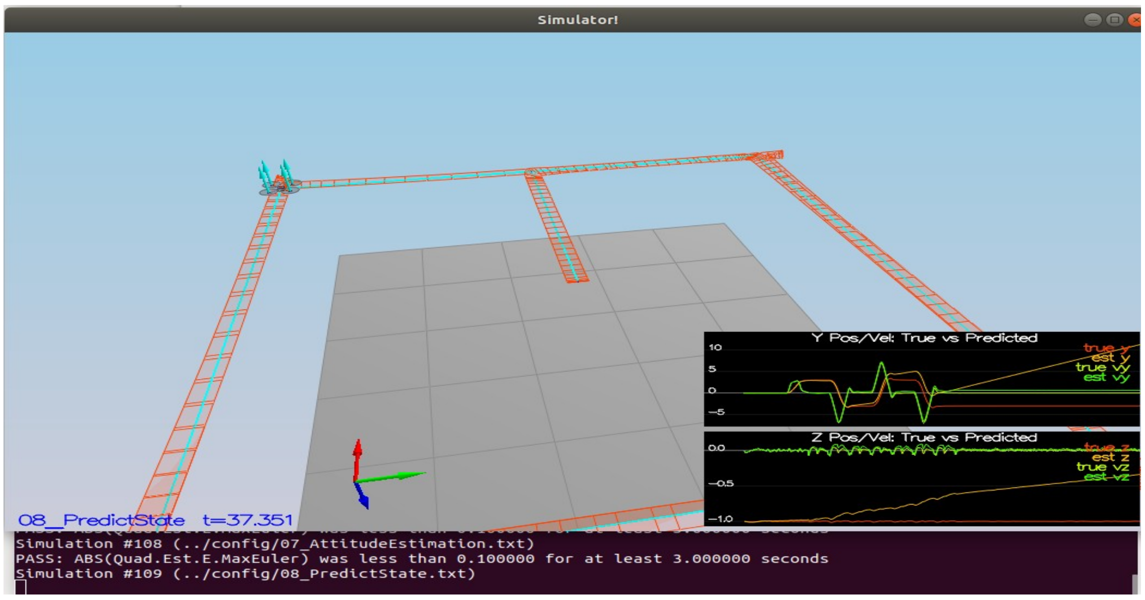
This function follows the pseudocode algorithm outlined below. The predicted state is essentially all of the state variables predicted with an integration step. This is defined by the transition function, *g*.


IMU
Accel
"Control Input"
$[\ddot{X}^b, \ddot{Y}^b, \ddot{Z}^b]$
Rotate to inertial frame
New Prediction
$[\dot{X}, \dot{Y}, \dot{Z}]$
Transition Model (integration)
$[\bar{X}, \bar{Y}, \bar{Z}, \dot{\bar{X}}, \dot{\bar{Y}}, \dot{\bar{Z}}, \bar{\psi}]$
Previous State
Compl. Filter
$[\bar{\psi}]$

Predict()

Changes to the covariance were calculated using the algorithm outlined in the EKF pseudocode. The previous covariance is premultiplied by the derivative of the transition function and post multiplied by the transpose of the derivative of the transition function. Added to this is the transition model covariance, Q_t.

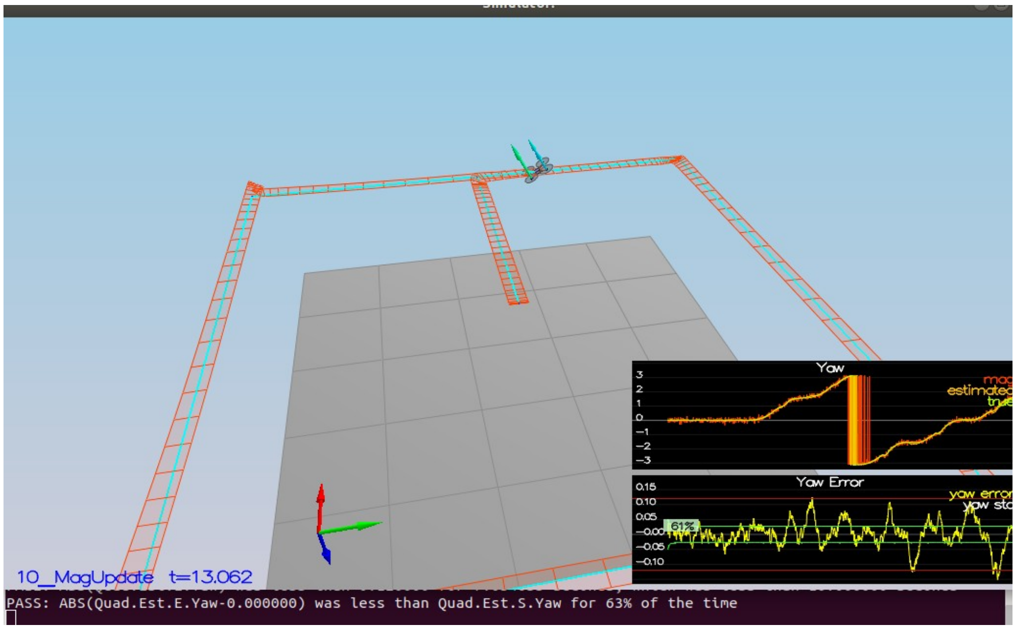
Below are two simulations that test the performance of the prediction step.



```
Simulation #108 (../config/07_AttitudeEstimation.txt)
PASS: ABS(Quad.Est.E.MaxEuler) was less than 0.100000 for at least 3.000000 seconds
Simulation #109 (../config/08_PredictState.txt)
```

Implement the magnetometer update.

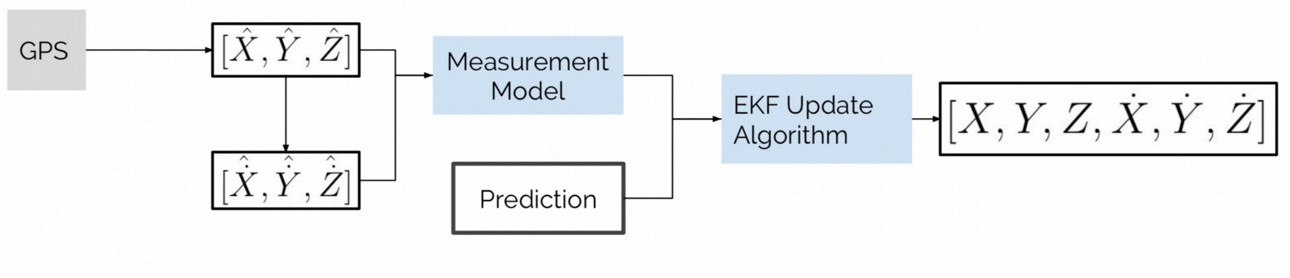
The magnetometer update is similar to GPS update except there is a different measurement model. This measurement model was implemented and the state is updated. Below is the simulation that tests the effectiveness of the yaw estimate with the magnetometer sensor data used to update the prediction.



```
PASS: ABS(Quad.Est.E.Yaw-0.000000) was less than Quad.Est.S.Yaw for 63% of the time
```

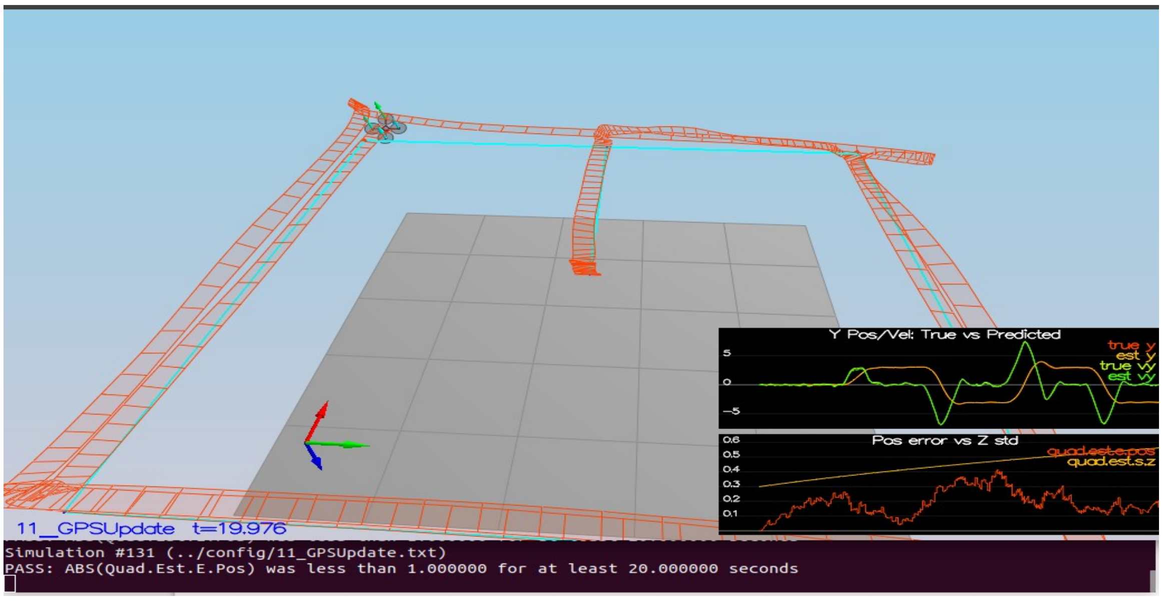
Implement the GPS update.

The main aspect of developing an EKF update step is to create an accurate measurement model h, and its derivative h'.


GPS
$[\hat{X}, \hat{Y}, \hat{Z}]$
$[\hat{\dot{X}}, \hat{\dot{Y}}, \hat{\dot{Z}}]$
Measurement Model
Prediction
EKF Update Algorithm
$[X, Y, Z, \dot{X}, \dot{Y}, \dot{Z}]$

GPS Update

The GPS update step follows the EKF algorithm and corrects the prediction on position and velocity.



```
Simulation #131 (../config/11_GPSUpdate.txt)
PASS: ABS(Quad.Est.E.Pos) was less than 1.000000 for at least 20.000000 seconds
```

Detuning controller

In order for the quadrotor to function with the imperfect estimation data, the controller gain parameters were "detuned." This allows realistic performance. All of the simulations shown in this document were recorded when the controller had been detuned and with the flight controller built in the previous project.