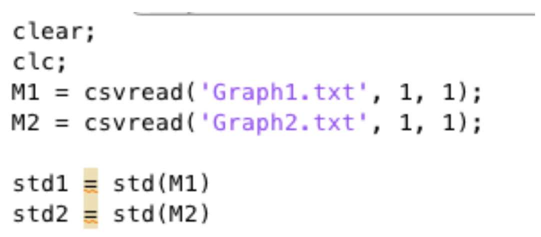
Jackson Burrus
1/5/19

Building an Estimator

1.

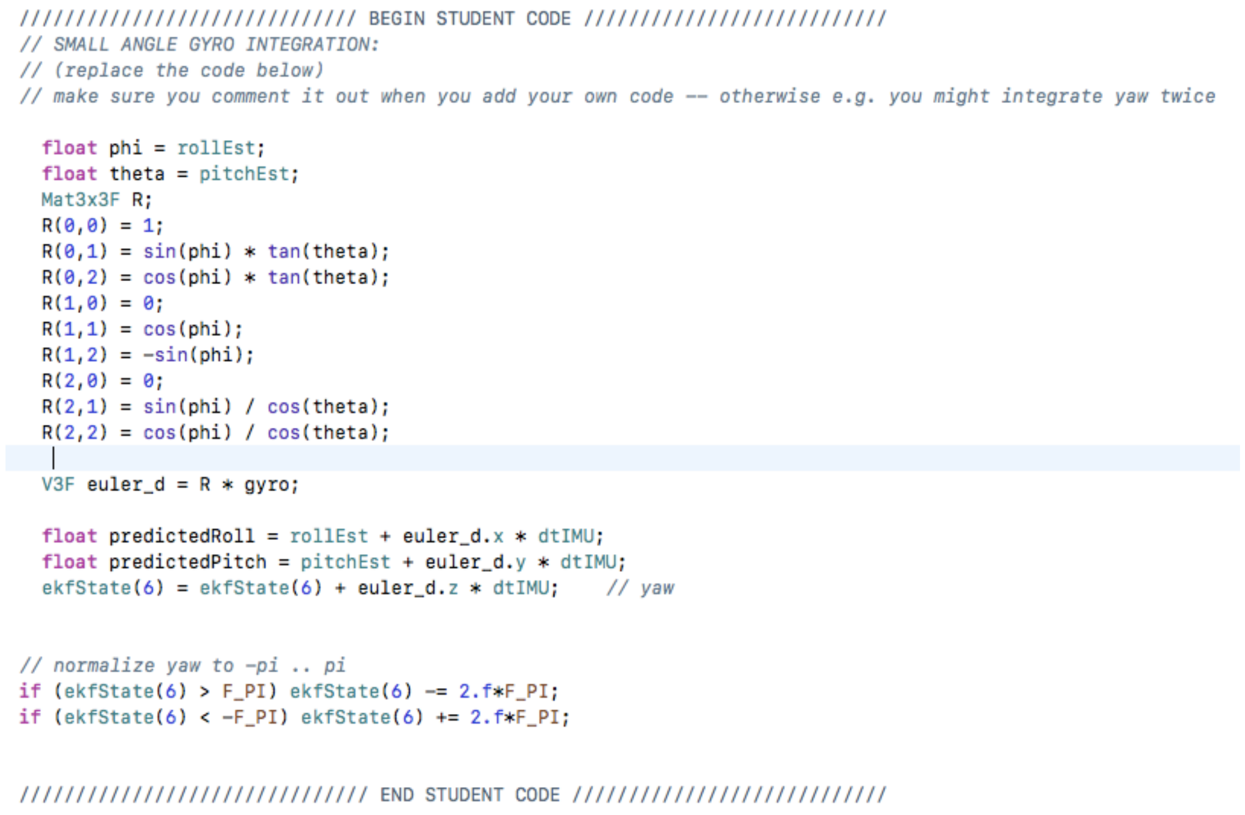
The standard deviation was calculated by first running the code to collect data, and then using a MATLAB script to calculate the corresponding standard deviation for both graphs. This script can be seen below



```
clear;
clc;
M1 = csvread('Graph1.txt', 1, 1);
M2 = csvread('Graph2.txt', 1, 1);

std1 = std(M1)
std2 = std(M2)
```

2.

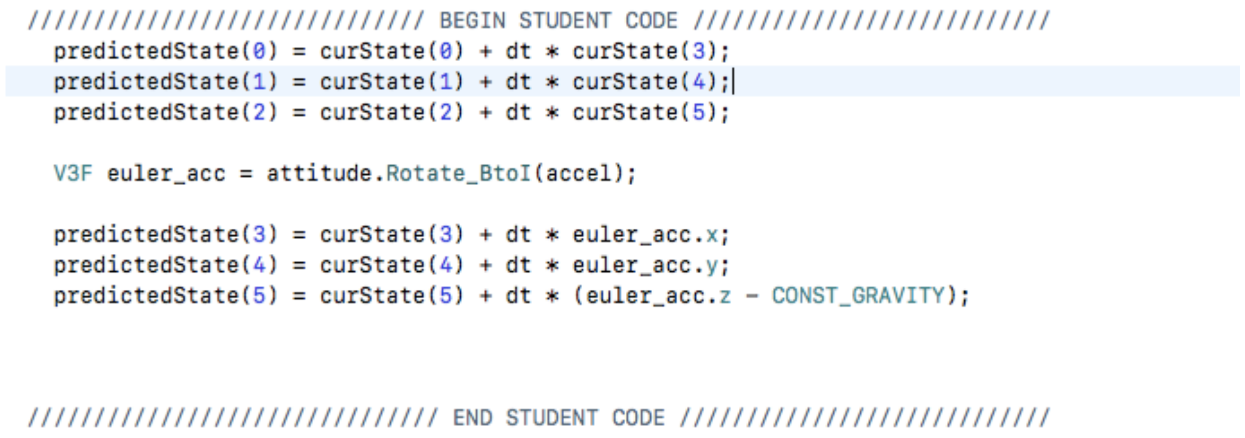
The IMU function is updated by first calculating the proper rotation matrix to convert from the body frame to the inertial frame. This rotation matrix is then multiplied by the gyro data to convert it to Euler coordinates. Then the original equation used to predict the roll, pitch, and yaw of the vehicle is valid. This code can be seen below



```
////////////////////////////// BEGIN STUDENT CODE ///////////////////////////
// SMALL ANGLE GYRO INTEGRATION:
// (replace the code below)
// make sure you comment it out when you add your own code -- otherwise e.g. you might integrate yaw twice

  float phi = rollEst;
  float theta = pitchEst;
  Mat3x3F R;
  R(0,0) = 1;
  R(0,1) = sin(phi) * tan(theta);
  R(0,2) = cos(phi) * tan(theta);
  R(1,0) = 0;
  R(1,1) = cos(phi);
  R(1,2) = -sin(phi);
  R(2,0) = 0;
  R(2,1) = sin(phi) / cos(theta);
  R(2,2) = cos(phi) / cos(theta);

  V3F euler_d = R * gyro;

  float predictedRoll = rollEst + euler_d.x * dtIMU;
  float predictedPitch = pitchEst + euler_d.y * dtIMU;
  ekfState(6) = ekfState(6) + euler_d.z * dtIMU;    // yaw


// normalize yaw to -pi .. pi
if (ekfState(6) > F_PI) ekfState(6) -= 2.f*F_PI;
if (ekfState(6) < -F_PI) ekfState(6) += 2.f*F_PI;


/////////////////////////////// END STUDENT CODE ////////////////////////////
```

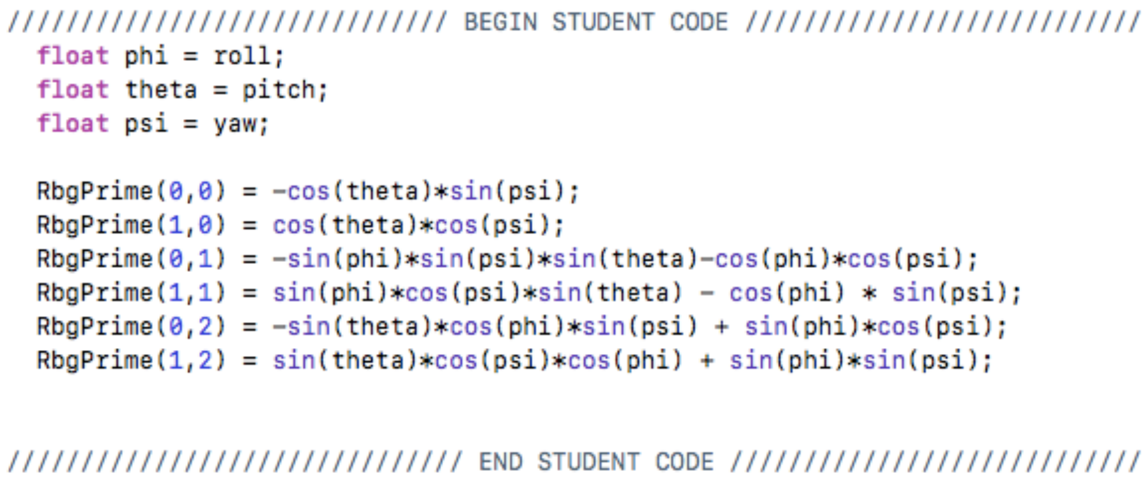
3.

For the prediction step, the *PredictState()* function is first updated so that the predicted positions are integrated from current velocity and predicted velocities are integrated from current accelerations. This portion of the code is below:



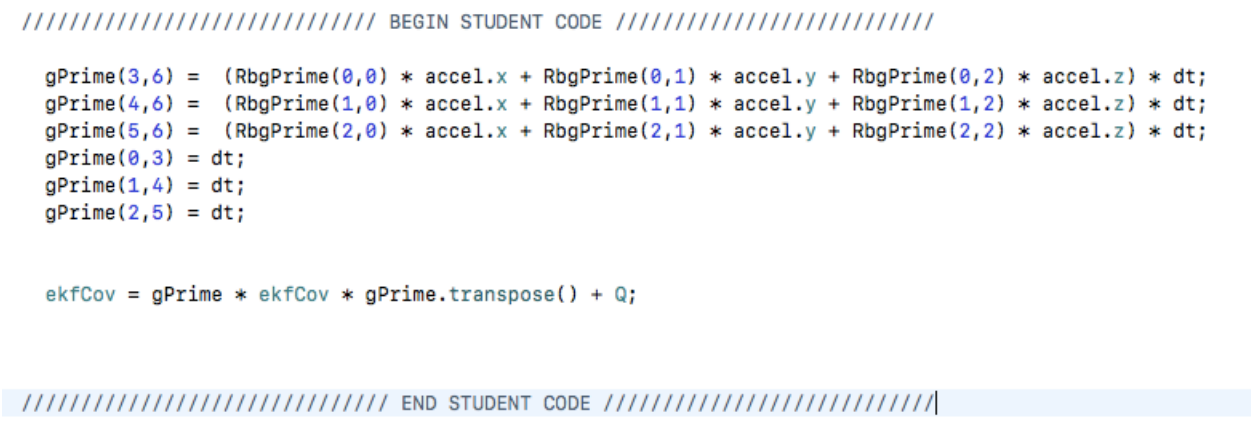
```
//////////////////////////////// BEGIN STUDENT CODE ///////////////////////////
  predictedState(0) = curState(0) + dt * curState(3);
  predictedState(1) = curState(1) + dt * curState(4);
  predictedState(2) = curState(2) + dt * curState(5);

  V3F euler_acc = attitude.Rotate_BtoI(accel);

  predictedState(3) = curState(3) + dt * euler_acc.x;
  predictedState(4) = curState(4) + dt * euler_acc.y;
  predictedState(5) = curState(5) + dt * (euler_acc.z - CONST_GRAVITY);



/////////////////////////////// END STUDENT CODE ////////////////////////////
```

Next, the *GetRbgPrime()* function is updated by plugging in all of the correct elements of the matrix, according to the lessons. This is below:



```
//////////////////////////////// BEGIN STUDENT CODE ////////////////////////////
  float phi = roll;
  float theta = pitch;
  float psi = yaw;

  RbgPrime(0,0) = -cos(theta)*sin(psi);
  RbgPrime(1,0) = cos(theta)*cos(psi);
  RbgPrime(0,1) = -sin(phi)*sin(psi)*sin(theta)-cos(phi)*cos(psi);
  RbgPrime(1,1) = sin(phi)*cos(psi)*sin(theta) - cos(phi) * sin(psi);
  RbgPrime(0,2) = -sin(theta)*cos(phi)*sin(psi) + sin(phi)*cos(psi);
  RbgPrime(1,2) = sin(theta)*cos(psi)*cos(phi) + sin(phi)*sin(psi);


/////////////////////////////// END STUDENT CODE /////////////////////////////
```

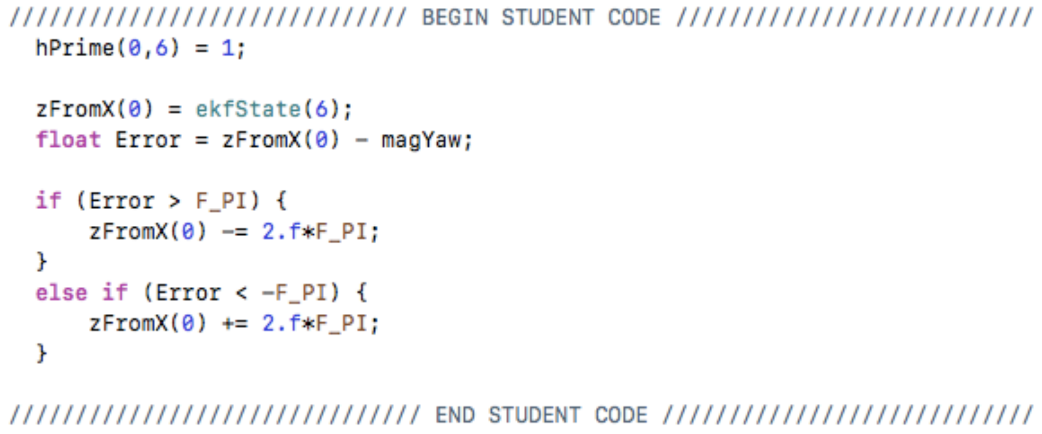
Finally the *Predict()* function is updated by plugging in the proper g prime matrix elements and then calculating the covariance matrix. This code is below:



```
////////////////////////////// BEGIN STUDENT CODE //////////////////////////

  gPrime(3,6) =  (RbgPrime(0,0) * accel.x + RbgPrime(0,1) * accel.y + RbgPrime(0,2) * accel.z) * dt;
  gPrime(4,6) =  (RbgPrime(1,0) * accel.x + RbgPrime(1,1) * accel.y + RbgPrime(1,2) * accel.z) * dt;
  gPrime(5,6) =  (RbgPrime(2,0) * accel.x + RbgPrime(2,1) * accel.y + RbgPrime(2,2) * accel.z) * dt;
  gPrime(0,3) = dt;
  gPrime(1,4) = dt;
  gPrime(2,5) = dt;


  ekfCov = gPrime * ekfCov * gPrime.transpose() + Q;


/////////////////////////////// END STUDENT CODE ///////////////////////////
```

4.

The magnetometer information is updated in the *UpdateFromMag()* function. First the h prime vector is updated, and then the error is calculated between the current estimated yaw and the measured yaw. This error is normalized so that the error is between [-π, π] and the proper variables are updated accordingly. This code is below



```
////////////////////////////// BEGIN STUDENT CODE ///////////////////////////
  hPrime(0,6) = 1;

  zFromX(0) = ekfState(6);
  float Error = zFromX(0) - magYaw;

  if (Error > F_PI) {
      zFromX(0) -= 2.f*F_PI;
  }
  else if (Error < -F_PI) {
      zFromX(0) += 2.f*F_PI;
  }

/////////////////////////////// END STUDENT CODE ////////////////////////////
```

5.

The GPS information is updated in the *UpdateFromGPS()* function. The current state and h prime matrix are calculated in this function to update the state estimate. This code is below

```
/////////////////////////////// BEGIN STUDENT CODE ///////////////////////////

  zFromX(0) = ekfState(0);
  zFromX(1) = ekfState(1);
  zFromX(2) = ekfState(2);
  zFromX(3) = ekfState(3);
  zFromX(4) = ekfState(4);
  zFromX(5) = ekfState(5);

  hPrime(0,0) = 1;
  hPrime(1,1) = 1;
  hPrime(2,2) = 1;
  hPrime(3,3) = 1;
  hPrime(4,4) = 1;
  hPrime(5,5) = 1;


//////////////////////////////// END STUDENT CODE /////////////////////////////
```