AE 483 – Lab 1

9/13/2015

Zach Fester, Steven Macenski, Jacob Dray

Vehicle Kinematics and Data Collection Lab Report

In this lab, we learned to program, communicate and analyze data with the Hummingbird quadcopters. As per the lab objectives, we used all of these skills to replay the flight using Matlab's simulation features.

Problem 1.1

Part (a)

Using Euler's Method of numerical integration, we integrated the angular velocities as obtained from sensors on-board the Hummingbird to receive the Euler angles as a function of time. We also have the angle data as a function of time from the motion capture system. Plotting these points, we get Figure 1, Figure 2, and Figure 3 shown below in Pitch, Yaw, and Roll, respectively.

Figure 1

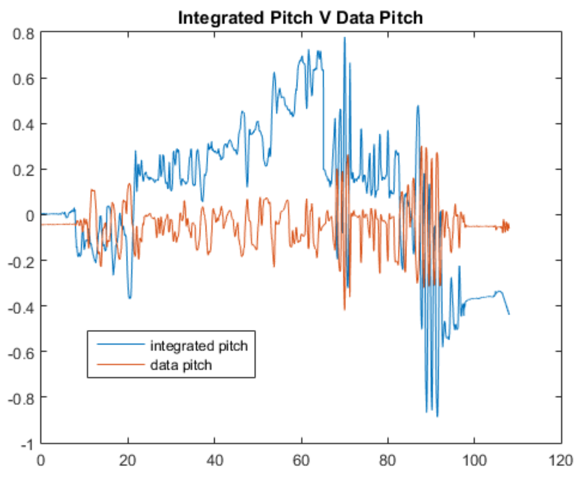


Figure 2

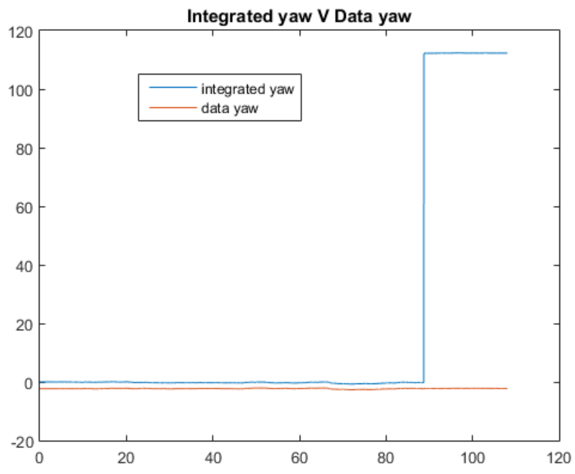
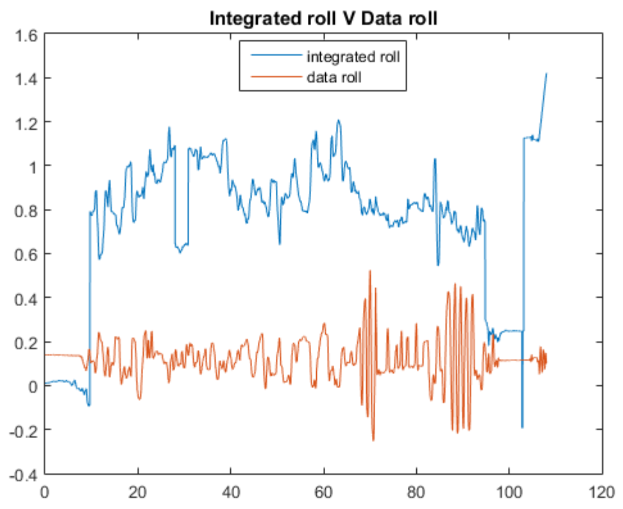


Figure 3



In Figure 1, the integrated on-board data and motion capture data pitch are fairly similar in a +/- 0.2 accuracy at the beginning of the test. Around 20 seconds the data begins to diverge more with the integrated data increasing regularly. This is due to the sensor pitch measurement spiking around 20 seconds and the integrated pitch responded to the fast increase. From 20 to 40 seconds the error between measured and integrated data is consistent. At 45 seconds, the increasing positive pitch data yields higher integrated errors. When time is at 70 seconds, it can be seen that extreme changes in pitch results in high uncertainty in integrated pitch resulting in higher error.

In Figure 2, the integrated on-board data and motion capture system data yaw are extremely similar until around 90 seconds. Before then, the data is closely related with a 2.3 degree offset between the data sets. Around 90 seconds into the run, the integrated yaw measurements largely spike 112.1 degrees. Root cause of the measurement spike would be purely speculative.

In Figure 3, the integrated on-board data and motion capture data roll quickly attains error. After 8 seconds into flight, the roll's value is largely positive propagating the integrated data further positive. Until a large amount of change in between 80 and 100 seconds, the integrated error is large. After such, it hits a trough rebalancing with the negative roll angles, then spikes quickly until the end of the run.

Part (b)

Using Euler's Method of numerical integration, we integrated the acceleration as obtained from sensors on-board the Hummingbird twice to receive the position as a function of time. Plotting these points, we get Figure 4, Figure 5, and Figure 6 shown below in X, Y, and Z, respectively.

Figure 4

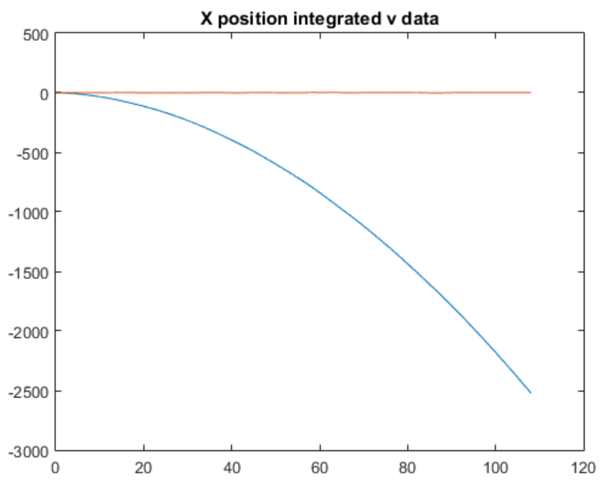


Figure 5

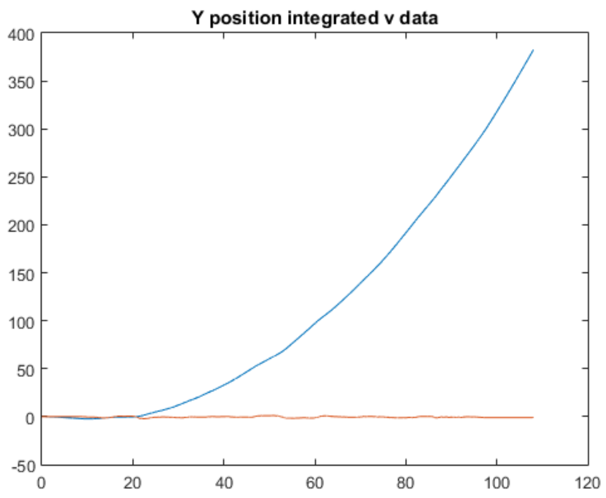
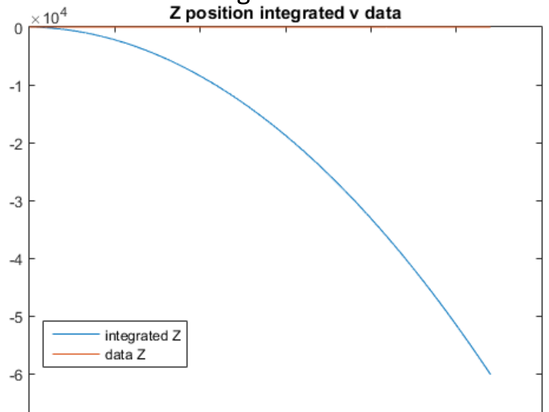


Figure 6



In all Figure 4, 5, and 6, we see divergence of the integrated data as compared to the motion capture system's data. In all cases, the positional data is well within the range of the room and operating range of the quadcopter. The integrated data, however, quickly explodes to magnitudes far outside the range of the lab and size of the Transportation Building. This is from poor quality accelerometers with unreliable measurements. The X and Z accelerations are always negative, with few exceptions, resulting in a large negative explosion. The Y acceleration was instead largely positive, resulting in an explosion in the positive direction. This data is unreliable.

Problem 1.2

Using the data from the motion capture system, we created a simulation of the quadcopter's trajectory through space in the frame of the Matlab display, relating to a 'universal' frame. In doing so, we ran into some issues with our data. The data received from the motion capture system is skewed ~45 degrees. This results in a shifting in the Matlab's view of the quadcopter. Using other teams' data, it can be found that our algorithms are indeed correct and the data is corrupt. Figure 7 displays a screen shot from our flight using our program and Figure 8 displays a screen shot of another teams' flight using our program. Note the difference in quad shifting. With this in mind, we have conformation that our data is indeed correct while our data has some to be desired.

Figure 7

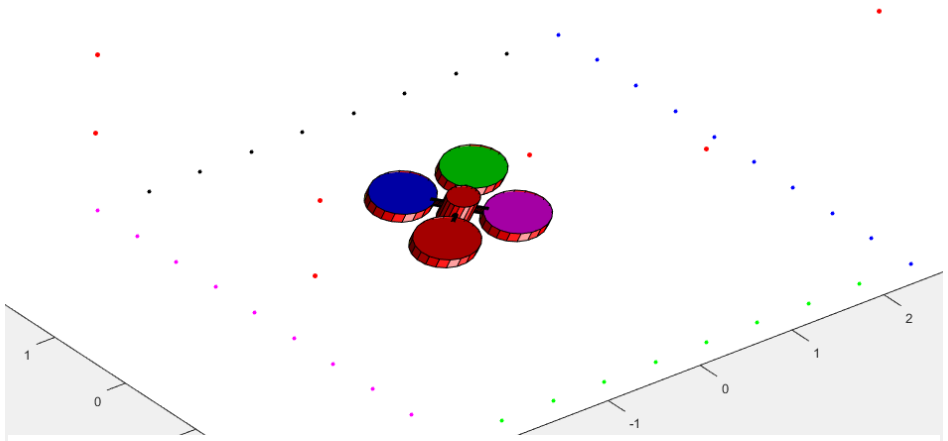
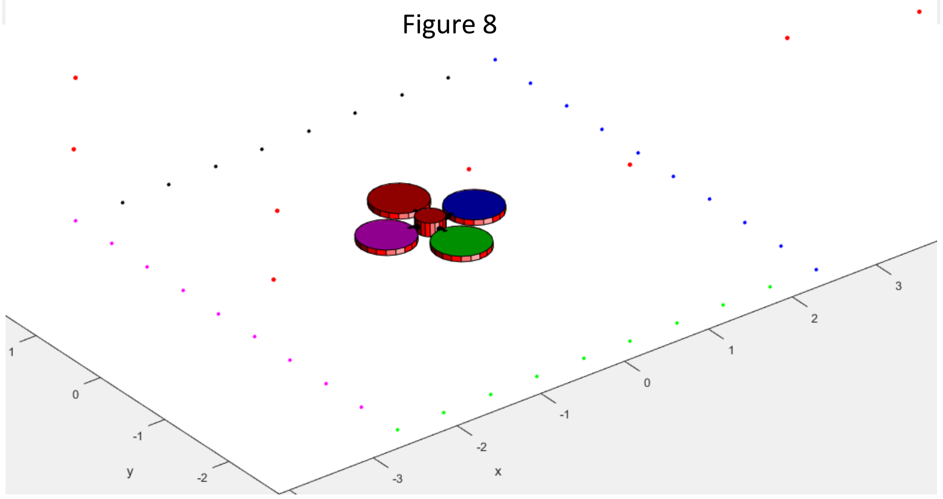


Figure 8



Besides the issues with data shifting, the quadcopter flight was representative of that observed in the lab. The motion capture data was otherwise resulted in a good simulation.

Lines of code of particular interest are as followed:

```
p2 = [1 0 0;0 -1 0;0 0 -1]*rotationMatrixFromEulerAngles(phi(1),theta(1),psi(1))*p1
w2 = [1 0 0;0 -1 0;0 0 -1]*w0;
p0 = rotationMatrixFromEulerAngles(phi(i),theta(i),psi(i))*p1 + repmat(q10(:,i),1,294);
p2 = [1 0 0;0 -1 0;0 0 -1]*p0;
output = [cos(psi)*cos(theta) cos(psi)*sin(theta)*sin(phi)-sin(psi)*cos(phi)
cos(psi)*sin(theta)*cos(phi)+sin(psi)*cos(phi);...
sin(psi)*cos(theta) sin(psi)*sin(theta)*sin(phi)+cos(psi)*cos(phi)
sin(psi)*sin(theta)*cos(phi)-cos(phi)*sin(phi);...
-sin(theta) cos(theta)*sin(phi) cos(theta)*cos(phi)];
```

The full code will be turned into the TAs via email.

Problem 1.3

Using the data from the motion capture system, we created a simulation of the quadcopter's trajectory in the frame of the quadcopter, relating to a 'universal' Matlab frame. In doing so, we ran into some issues with our data. The data received from the motion capture system is skewed ~45 degrees. This results in a shifting in the quadcopter's view of the Matlab environment. Using other teams' data, it can be found that our algorithms are indeed correct and the data is corrupt. Figure 8 displays a screen shot from our flight using our program and Figure 9 displays a screen shot of another teams' flight using our program. Note the difference in Matlab plane shifting. With this in mind, we have conformation that our data is indeed correct while our data has some to be desired.

Figure 8

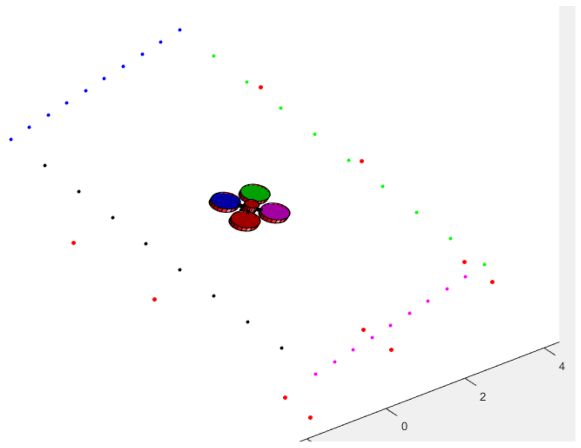
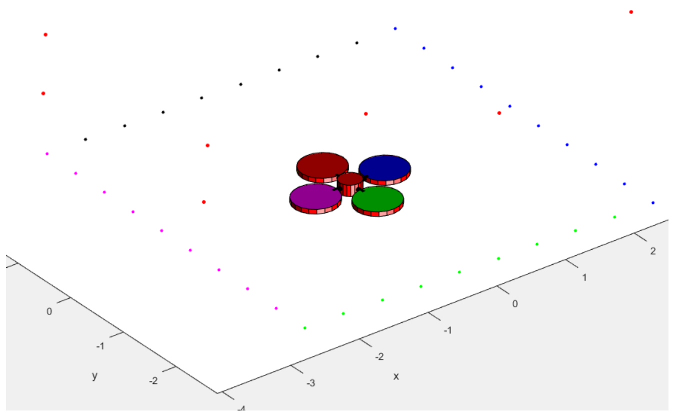


Figure 9



Lines of code of particular interest are as follows

```
p2 = [1 0 0;0 -1 0;0 0 -1]*rotationMatrixFromEulerAngles(phi(1),theta(1),psi(1))*p1;
w2 = [1 0 0;0 -1 0;0 0 -1]*w0;
R02 = [1 0 0;...
       0 -1 0;...
       0 0 -1];
R20 = transpose(R02);
R12 = rotationMatrixFromEulerAngles(phi(i), theta(i), psi(i))*R02;
R21 = transpose(R12);
w1 = transpose(rotationMatrixFromEulerAngles(phi(i), theta(i), psi(i)))*w0 +
repmat(q10(:,i),1,58);
w2 = R12*w1;
R10 = [cos(psi)*cos(theta) cos(psi)*sin(theta)*sin(phi)-sin(psi)*cos(phi)
cos(psi)*sin(theta)*cos(phi)+sin(psi)*cos(phi);...
sin(psi)*cos(theta) sin(psi)*sin(theta)*sin(phi)+cos(psi)*cos(phi)
sin(psi)*sin(theta)*cos(phi)-cos(phi)*sin(phi);...
-sin(theta) cos(theta)*sin(phi) cos(theta)*cos(phi)];
```

The full code will be turned into the TAs via email.