Kalman Filtering

Dhaneshwaran Jotheeswaran*

School of Electrical and Computer Engineering

E-mail: dhaneshwaran@gatech.edu

Abstract

Tracking objects, or estimating the state of a system is fundamental to many Robotics applications. Kalman Filter is one technique that is widely used for this purpose. This (the default Kalman Filter) can be viewed as a state estimator that assumes Linear Dynamics, and Gaussian Noise on both motion model as well as the observation model. With these two assumptions, it estimates the underlying state of the system.

Introduction

The aim of this lab is to implement a Kalman Filter, which is then used to make a helicopter hover at some stationary position. The helicopter's dynamics were linearized around this operating point and is controlled by a Linear Quadratic Regulator **(LQR)**. Since, the linearization works only around this operating point, it is crucial to get an accurate state estimator. After all, an LQR used to control a Non-linear system is only as good as it's state estimator. So, if the helicopter veers off of the operating point larger than a threshold, or if the control, which is related to the state-feedback is wrong, then the operation won't be as desired. Now, let us look at the few tasks given to us for this lab.

Task 1

For the first task I zeroed down all the noise terms, and ran the Kalman Filter, to see that the LQR achieved the necessary hover. From the North-East-Down plot we can be certain that it stays very close to the operating point. Though there are oscillations along the East and the Down axes, it is very small (order of 10^{-16}). And since the system doesn't blow up (meaning, doesn't go to infinity), we can say that the system is stable. The following three plots show the NED position, Quaternion position and the Control Commands. The reason that this works well is because our Filter tracks the state really well, as the noises on the motion and observation models are zero.

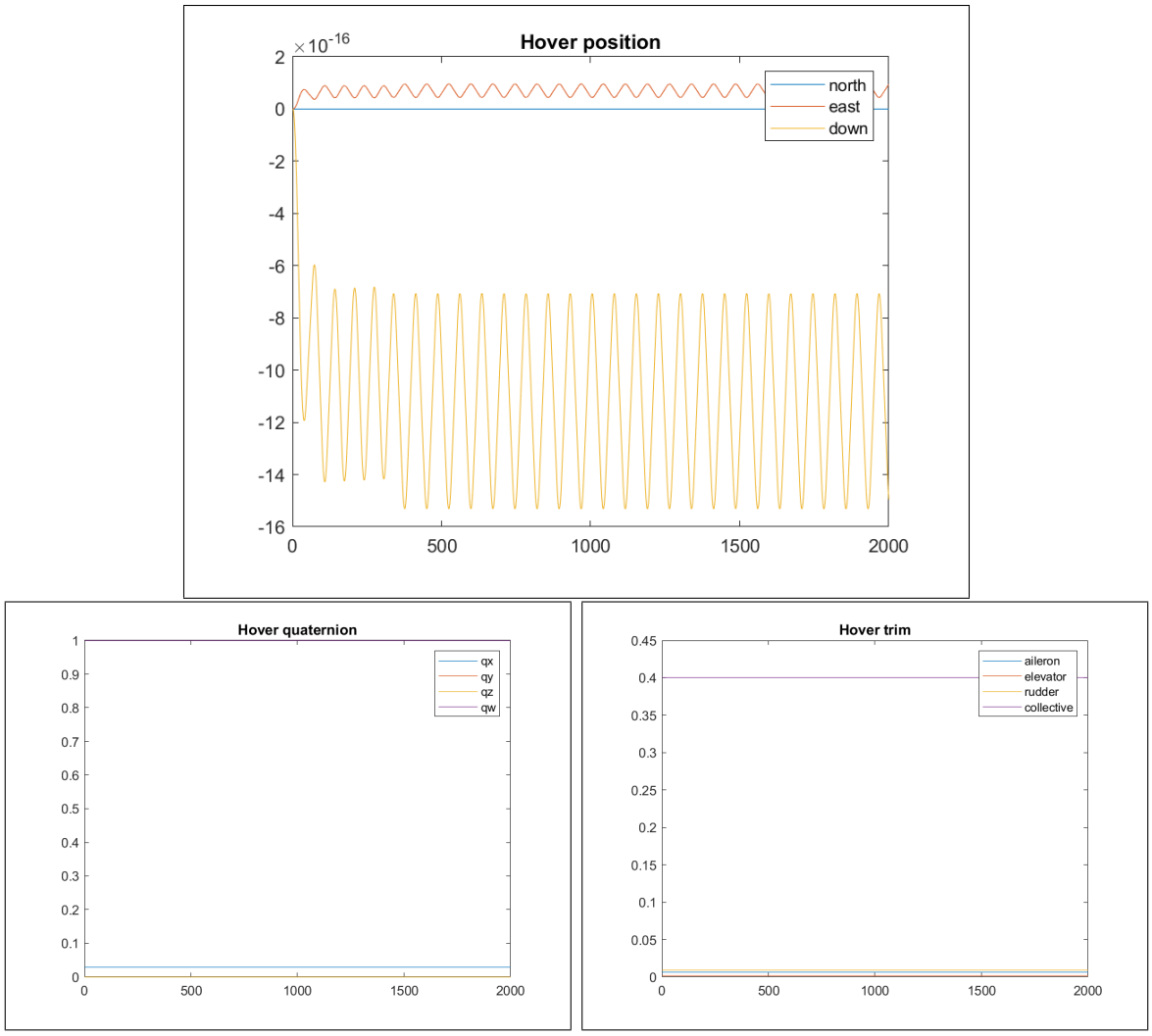

Figure 1: The NED, Quaternion, and Control plots

Task 2

For the second task there were two sub-tasks. In one, I kept the noise on the observations 0, and increased the noise on the dynamics to see when the default state estimator and LQR controller breaks, and in the second, I kept the noise on the dynamics 0, and increased the noise on the observation to see when the default state estimator and the LQR controller breaks.

Changing sigmaX

So, when I changed the noise on the dynamics and kept noise on observation as 0, the default estimator was very robust and worked fine for a lot of noise. Though it started to become unstable for a few runs with $sigmaX = 2.5$, it wasn't until $sigmaX = 3.3$ that the helicopter almost always became unstable. I am defining stability like we define it in Control

Theory, that is, a system is stable if it doesn't blow up. So, even if the system is oscillating wildly, but doesn't blow up, then the system is still stable. The reason I am saying that the system is stable for a larger value of noise is because for the other variable $sigmaY$, it became unstable for a much smaller noise value. I attribute it to the fact that the LQR and the default estimator, which just assumes the output is equal to the true state, will have a feedback law that is catered towards the true state. And it will take a lot of noise on the state dynamics to kick the system out of its operating point around which it was linearized by a large amount. This is the **NED** plot when the helicopter becomes unstable during this experiment.

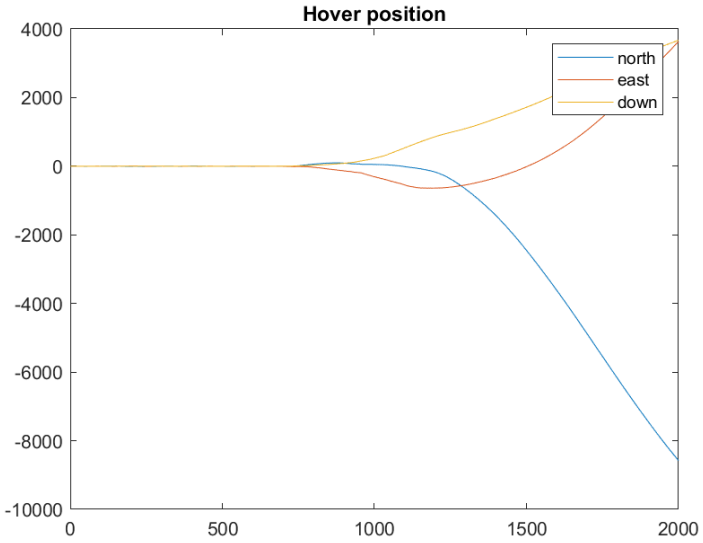

Figure 2: NED plot for changing sigmaX

Changing sigmaY

So, when I changed the noise on the observation and kept noise on dynamics as 0, the default estimator was not robust and was sensitive for even small noise. Though it started to become unstable for a few runs with $sigmaY = 0.35$, it wasn't until $sigmaY = 0.43$ that the helicopter almost always became unstable. I am defining stability the same way as before, like we define it in Control Theory, that is, a system is stable if it doesn't blow up. So, even if the system is oscillating wildly, but doesn't blow up, then the system is still stable. The reason I am saying that the system is stable only for a smaller value of noise is because for the other variable $sigmaX$, as we noted above, it became unstable for a much larger noise value. I attribute it to the fact that the LQR and the default estimator, which just assumes the output is equal to the true state, will have a feedback law that is really bad. Because LQR is basically "Feed-back" control, the system is only as good as the value we feed into it. And in this case, we are feeding in really noisy values. And it takes only a small noise on the observation to kick the system out of its operating point around which it was linearized by a large amount. The noisy sensor throws off the LQR, as it is not giving control commands based on the actual state. And as mentioned above, since LQR is

a feedback controller, it is only as good as its feedback, which in this case is really bad and noisy. This is the **NED** plot when the helicopter becomes unstable during this experiment.

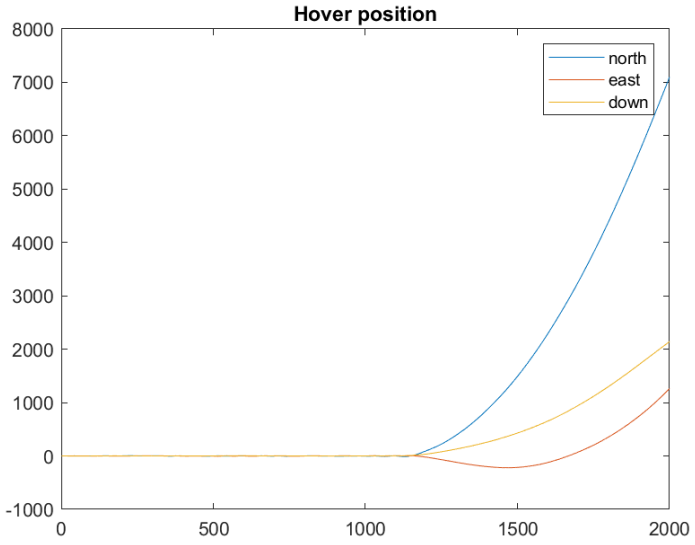

Figure 3: NED plot for changing sigmaY

Task 3

For this task, we were asked to implement a Kalman Filter and tweak the hyperparameters to make the system stable for the default noise values, which are $sigmaX = 0.1$ and $sigmaY = 0.5$. I used the four update equations for the Prediction and the Correction Step. In addition to that, I also took care of the edge case when $sigmaY = 0$. Because, the Matrix inside the inverse is Singular, the inverse doesn't exist. Since C is the Identity matrix in our case, the Kalman Gain, K, just becomes the identity when $sigmaY$ becomes 0. Other than that, there is nothing fancy happening in the code. For Q, I just used $sigmaX^2I$ and for R, I used $sigmaY^2I$. Since, I don't know how the cross coupled terms interact, doing this seems to be the most reasonable hing for me. In addition to that, the only other parameter I had to tune was the Covariance matrix in the Kalman Filter algorithm itself. And going with the same argument that I don't know how the cross terms interact, I initialized the Covariance Matrix here to a constant times the Identity Matrix. And now, to stabilize the system for the default noise settings, I just had to change this constant in the Covariance Matrix. After fiddling around with that term, I finally ended up with using **5**, as the constant. Using this, the system was stable for most of the runs. I ran the simulator for 50 runs, and the system stabilized for 49 runs. So, with a 98% success rate, this system is stable. Even though our control commands and position are wildly oscillating, it has not blown up, and stays close to the operating point. The following three plots show the NED position, Quaternion position and the Control Commands.

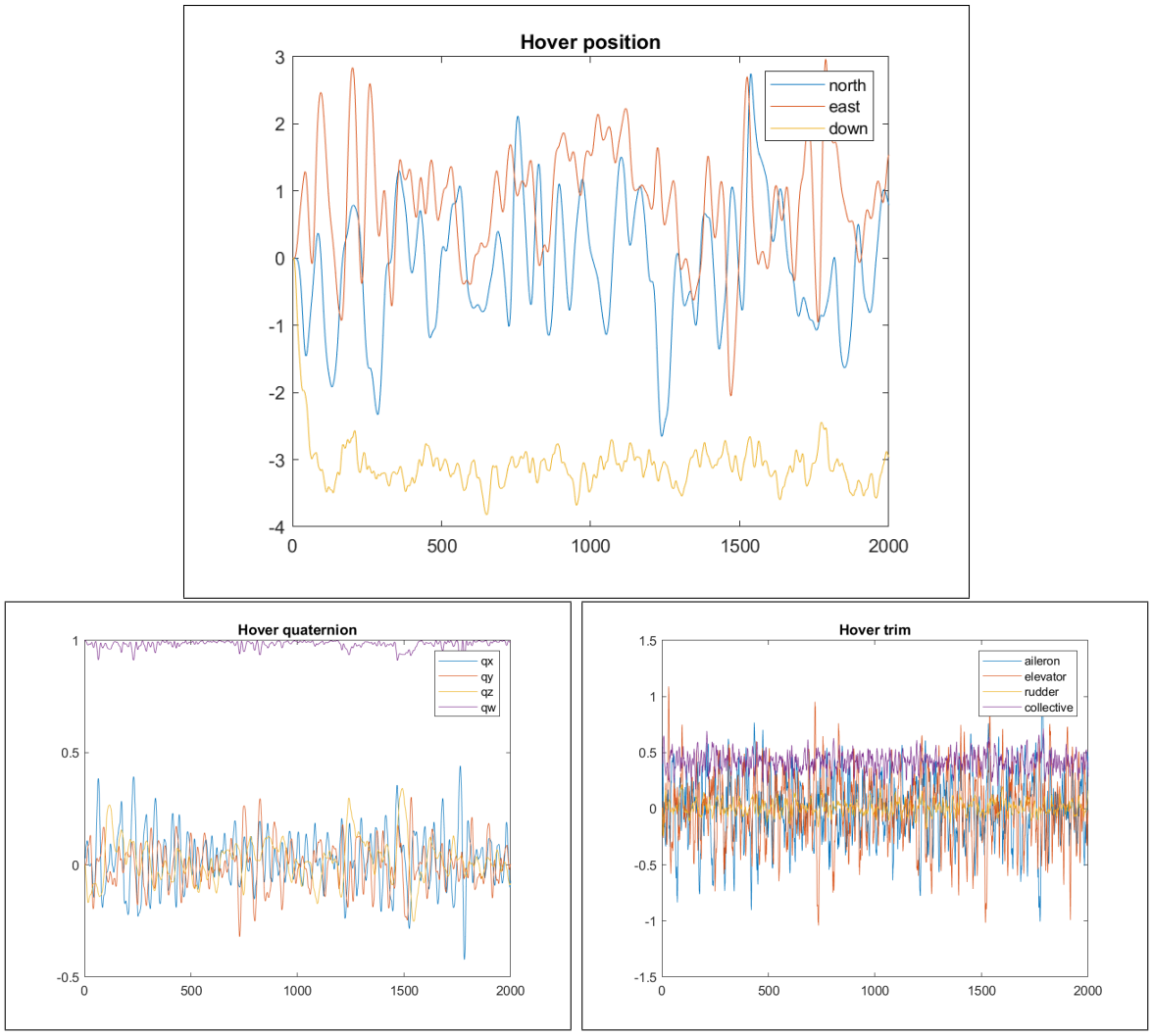

Figure 4: The NED, Quaternion, and Control plots

Task 4

For this task, we were asked to increase each noise term, just like in task 2 and see when the system is going unstable. For this task there were two sub-tasks. In one, I kept the noise on the observations 0, and increased the noise on the dynamics to see when the Kalman Filter and LQR controller breaks, and in the second, I kept the noise on the dynamics 0, and increased the noise on the observation to see when the Kalman Filter and the LQR controller breaks.

Changing sigmaX

So, when I changed the noise on the dynamics and kept noise on observation as 0, the Kalman Filter was very robust and worked fine for a lot of noise. Though it started to become unstable for a few runs with $sigmaX = 2.5$, it wasn't until $sigmaX = 3.5$ that the

helicopter almost always became unstable. The reason I am saying that the system is stable for a larger value of noise is because for the other variable $sigmaY$, it became unstable for a much smaller noise value. I attribute it to the fact that the LQR and the Kalman Filter, which has the Kalman Gain equal to Identity and thus the output is equal to the true state, will have a feedback law that is catered towards the true state. And it will take a lot of noise on the state dynamics to kick the system out of its operating point around which it was linearized by a large amount. As it can be observed, the behavior of the Kalman Filter and the default state estimator is the same in this experiment, as we are basically doing the same thing, since $sigmaY$ is 0. And so, the same shortcomings also affects the Kalman Filter. The following three plots show the NED position, Quaternion position and the Control Commands when the helicopter becomes unstable during this experiment.

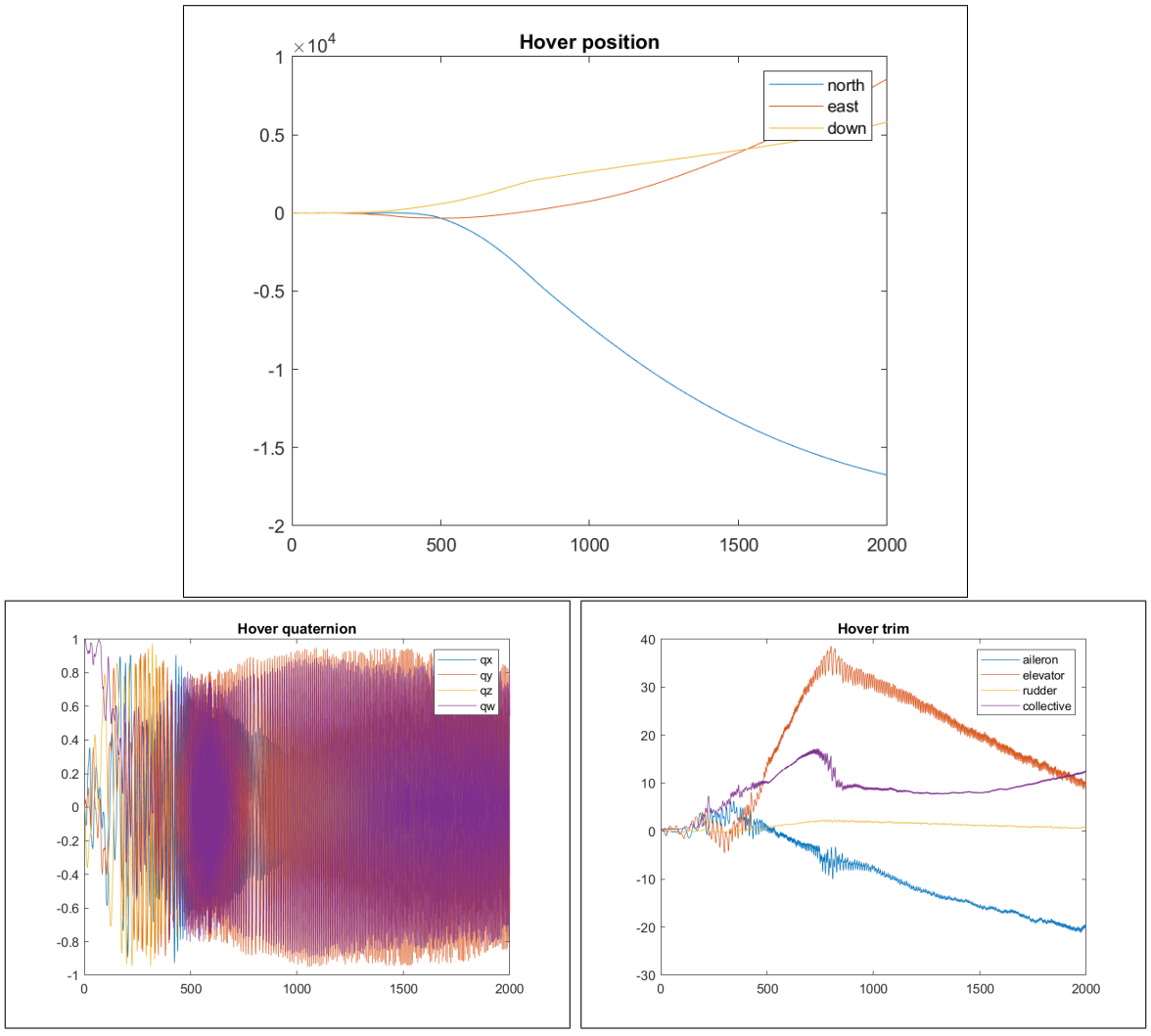

Figure 5: The NED, Quaternion, and Control plots

Changing sigmaY

So, when I changed the noise on the observation and kept noise on dynamics as 0, the Kalman Filter was not as robust as the previous experiment and was sensitive for smaller noise. But it was much more robust than the default state estimator we had in Task 1 and 2. Though this started to become unstable for a few runs with $sigmaY = 1.0$, it wasn't until $sigmaY = 1.4$ that the helicopter almost always became unstable. I attribute it to the fact that the LQR and the Kalman Filter, which weighs the output according to the noise on the observation, has the estimated state closer to the true state than what the default state estimator did. This will have a feedback law that is not as bad as the default state estimator. Because LQR is basically "Feed-back" control, the system is only as good as the value we feed into it. And in this case, we are feeding in noisy values, but corrected based on the noise on the observations. And so, it takes a relatively larger noise on the observation to kick the system out of its operating point around which it was linearized by a large amount, than it did the default state observer. Again, the noisy sensor throws off the LQR, as it is not giving control commands based on the actual state. And as mentioned above, since LQR is a feedback controller, it is only as good as its feedback, which in this case is bad, but not as bad as in Task 2. The following three plots show the NED position, Quaternion position and the Control Commands when the helicopter becomes unstable during this experiment.

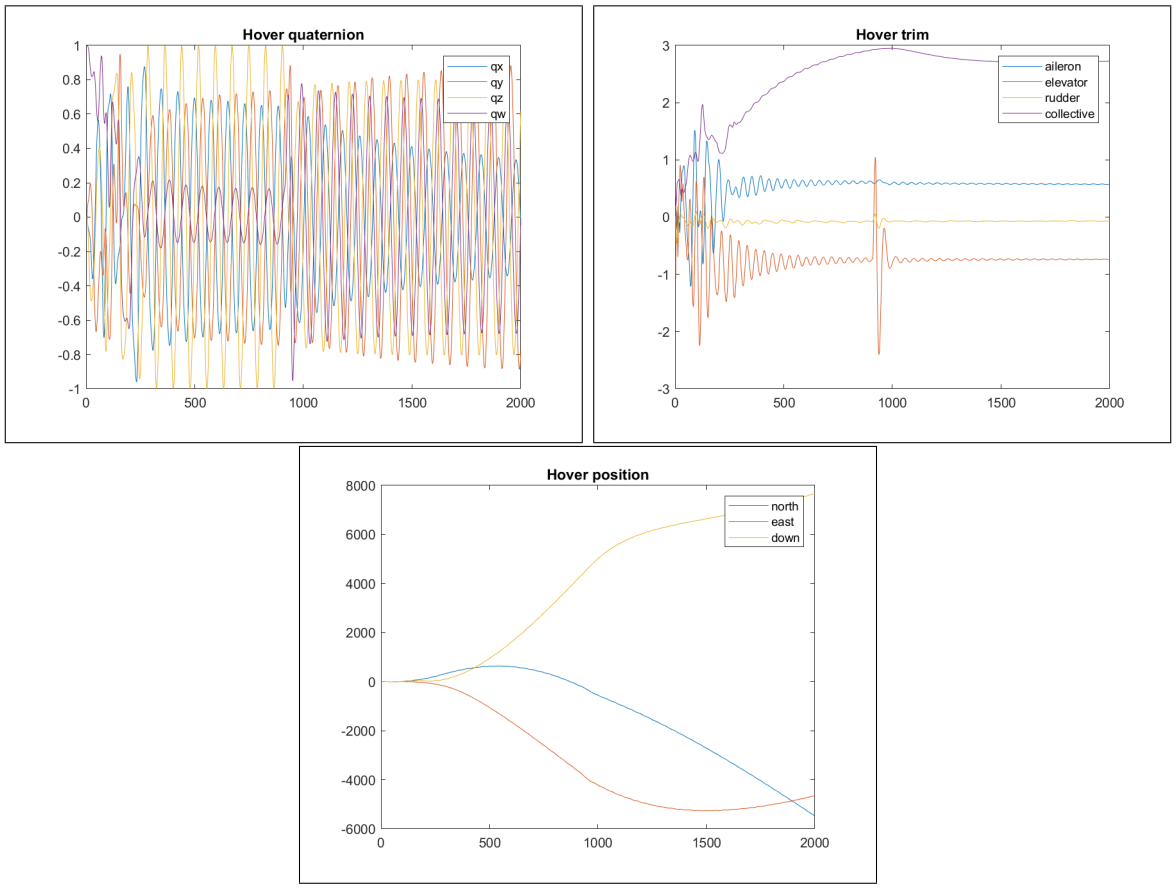

Figure 6: The Quaternion, Control, and NED plots