William Tidwell
gtg004s
CS8803 - Adaptive Control and Reinforcement Learning
February 13, 2019

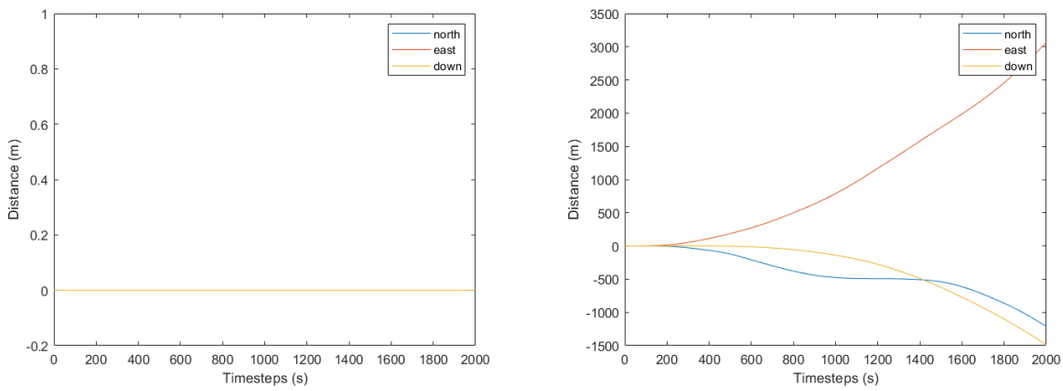
Assignment 1

a) Discussed with: Pol Llado
b) Sources: Textbooks and TA Office Hours

1 Initialization

The purpose of this assignment is to gain a better understanding of Linear Quadratic Regulators (LQR) by designing a controller for an unstable robot. A model that simulates an XCell Tempest helicopter in the flight regime close to hover is used for the exercise. We use (n, e, d) to represent North, East, Down, which is a common flight frame. To represent the state, we use position (n, e, d), orientation (quaternion q), velocity $(\dot{n}, \dot{e}, \dot{d})$, and angular rate (p, q, r) in the helicopter's coordinate frame. Standard controls for a helicopter are $(u1, u2, u3, u4)$. $u1$ and $u2$ are the latitudinal and longitudinal cycle pitch controls, also referred to as elevator and aileron. $u3$ is the tail rotor collective pitch control, also referred to as rudder. $u4$ is the main rotor pitch control.

First, we run the helicopter in open-loop, starting from the trim state. We do this once without noise and once with noise, to help get an idea of what is going on and how things look. An NED plot representing both of these is shown in Figure 1 below. We can see that the model without noise remains in its initial position for all dimensions, while the model with noise very quickly deviates from the hover state, both of which are to be expected.



(a) Without Noise

(b) With Noise

Figure 1: Open Loop NED Plots

2 Steady State LQR Controller

The first real task is to design a steady state LQR controller. Given states x, we can use the controller k to find a corresponding set of actions a with the following relationship

$$u_t = k_t x_t \tag{1}$$

The goal is to find a value of k once the system has stabilized and is running at steady state. We will call this value K_{ss}. The algorithm outlined below can be used to find K_{ss}.

1: **function** K-SS(A, B, Q, R, T)
2: $P \leftarrow Q$
3: **for** $i \leftarrow 1 : T$ **do**
4: $k[i] \leftarrow -(B^T P B + R)^{-1} B^T P A$
5: $P \leftarrow Q + k[i]^T R k(i) + (A + Bk[i])^T P (A + Bk[i])$
6: **return** $k(T)$

where A and B are used to represent the dynamics in the form

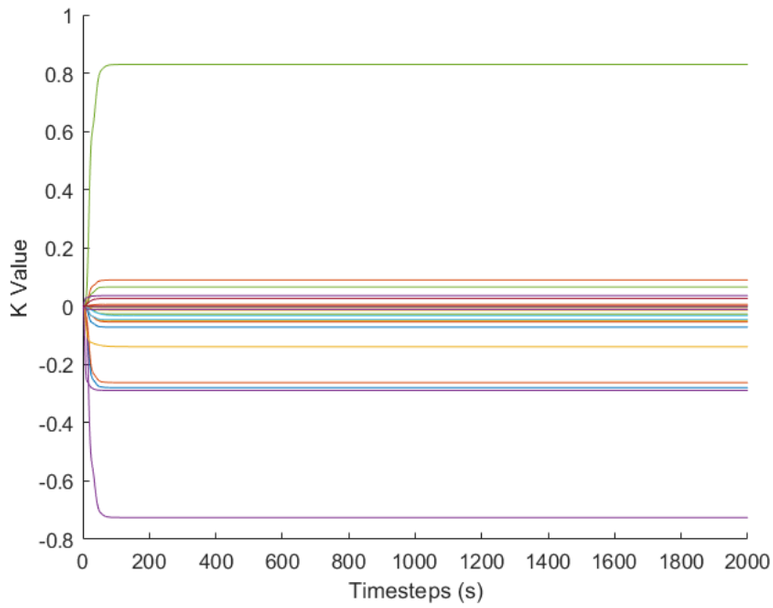
$$x_{t+1} = Ax_t + Bu_t \tag{2}$$

Q and R are used to calculate the quadratic cost as

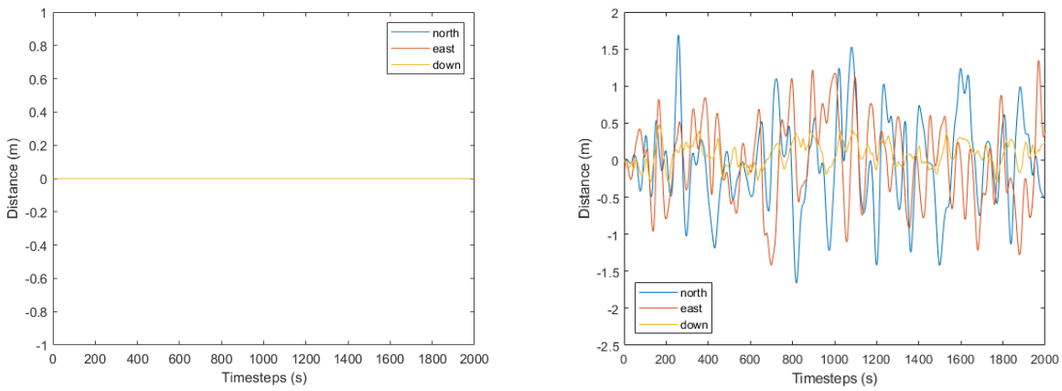
$$c(x, u) = x^T Q X + u^T R u \tag{3}$$

and T represents the horizon, or number of timesteps the algorithm runs. The plot below shows the convergence of all K-values, meaning the controller has reached steady-state. The values seem to have converged after around 100 iterations, and the process could have been terminated much earlier than it was.

Figure 2: Convergence of K Values



Next, an NED plot of two simulations using the new controller were created, one without noise and one with noise. The one without noise looks identical to the Open Loop NED plot without noise from Figure 1 above. This indicates that the controller is at least capable of keeping the helicopter hovering in the desired position under perfect conditions. Figure 3b shows the simulation with noise. While there is quite a bit of movement, the helicopter is still hovering in the area near the target hover state, so the controller seems to be working even after the introduction of noise.



(a) Without Noise

(b) With Noise

Figure 3: Using K_{ss} Controller

3 Testing Linearization

The controller was designed assuming linearization around hover trim condition. We now want to see what happens as we get further from linearization. Here, we attempt to find the "breaking" points for translation by increasing the starting (n, e, d) values and rotations by increasing the starting rotation angle about (x, y, z). Tables 1 and 2 below show the results. It should be noted that each of these coordinates was tested independently, meaning only one of them was adjusted at a time, while the others remained 0. The NED plots for each axis before and after divergence can be found in the Appendix.

n	e	d
30	20	60

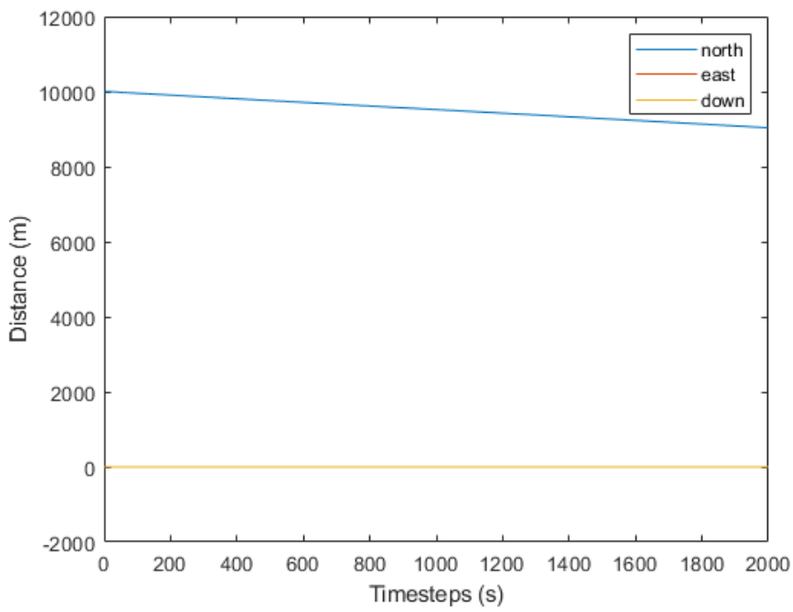
Table 1: Starting (n, e, d) Positions (m) for Poor Performance

x	y	z
$\frac{3}{4}\pi$	$\frac{5}{8}\pi$	$\frac{11}{8}\pi$

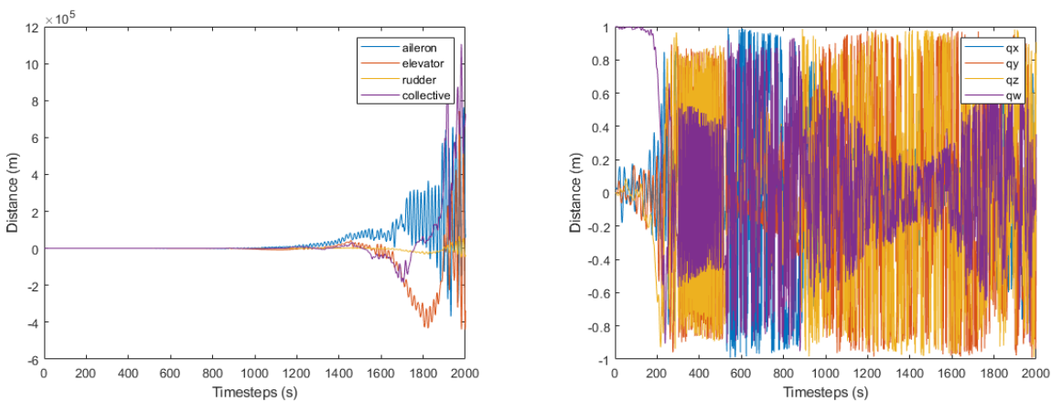
Table 2: Starting (x, y, z) Rotation Angles (radians) for Poor Performance

Next, we were asked to examine clipping by finding the maximum clipping distance that allows the controller to perform well. The helicopter was given a starting position of $north = 10000$ m, for which the linear controller performs poorly. Clipping is used to "clip" or cutoff all values of dx above the threshold. This helped to stabilize the helicopter and allow it to return to the target hover state, as demonstrated in Figure 4 below. A clipping distance of 20 m was found to be the maximum effective distance.

Figure 4: NED Plot with Clipping



The final test of the linear assumptions was latency. Latency is a delay between when a signal is sent, in this case the control, and when it is received. This means the controls are not being applied to the most recent set of actions, which can cause the helicopter to diverge from the target state if too large. Using the K_{ss} controller designed above, a latency of only 3 seconds caused the helicopter to beceome unstable, as evidenced in Figure 5 below. Figure 5a shows the previous controls, and Figure 5b shows the quaternion. Though the controls seem fairly stable, this is due largely to the scale of the y-axis. By around 250 times steps, the controls are already oscillating betwen -10m and 10m, which is already very unstable. The quaternion appears much more unstable than the controls, but this is again due to the scale of the y-axes. Looking closely, we can see that the quaternion really starts to diverge around 200 time steps, which lines up closely with what we saw from the controls.



(a) Previous Controls

(b) Quaternion

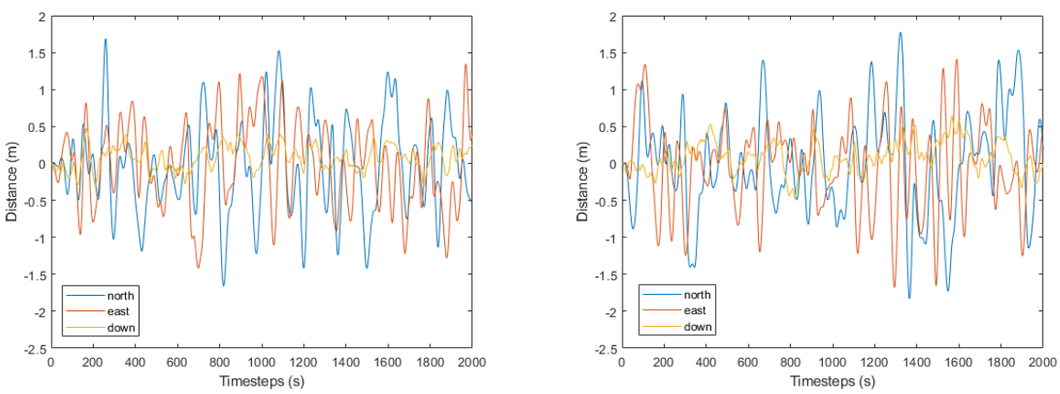
Figure 5: Latency = 3s with K_ss Controller

4 Custom Controls

For the last part of the assignment, we were tasked with building a controller that has better performance than the K_{ss} controller. I started by building a cost function that takes a controller K as a parameter,

performs n simulations with a specified horizon T, and returns the average cost of all simulations. Using MATLAB's built-in fminsearch() function I was able to search over a space of controllers and select the one that minimized the cost. I set $n = 25$ and $T = 250$. n was chosen such that it would be large enough to minimize the effect on any outliers on the average cost. T was chosen such that it would hopefully allow enough time for the system to stabilize while also reducing the amount of time needed for optimization. A comparison of the NED plots of the original K_{ss} controller and the optimized K_{opt} controller can be seen in Figure 6 below. At first glance, it does not appear as if there is any real improvement. However, closer inspection reveals that the optimized controller results in fewer peaks and valleys than the steady-state controller and does a better job of keeping the helicopter near the target hover state. Running the simulation with a horizon $T = 2000$ results in a cost of approximately 2040 using K_{ss} and cost of approximately 1938 using K_{opt}. Thus, the optimal controller leads to about a 5% reduction in cost over the duration of the simulation.

I also attempted to build a nonlinear controller using a simple 2-layer neural network. Using essentially the same approach as above, the neural network output a set of actions u based on a set of states x as input. Initially, I simply randomized the weights and used fminsearch() to optimize them. However, due to the complexity of the search, searching for over 10,000 iterations still left me with worse results than the linear controller. In attempt at a better initialization, I created a set of actions from states using the linear controller and attempted to initialize the neural networks weights to mimic those action. The hope was that, with a better set of initial weights, the neural network would converge more quickly when minimizing cost. Though I did see improvements over starting with random weights, the final cost was still orders of magnitude larger than with the simple linear controller. I am very interested in finding a nonlinear controller that outperforms the linear one, but due time constraints I was unable to pursue it further.



(a) K_{ss} Controller (b) K_{opt} Controller

Figure 6: Comparison of Controllers

Finally, we attempted to design a controller that is more robust to latencies than the K_{ss} controller found above. In a previous exercise, we found that the K_{ss} controller is only effective with latencies up to 2 seconds. The approach was very similar to the one outlined above with a few key differences. This time, a range of latencies was included in the cost function. For each simulation, a latency was randomly selected from the specified range, and the simulation was run with the latency, calculating the cost along the way. The costs were averaged over all iterations as before. The optimization was run twice: once starting with the K_{ss} controller, and once starting with a controller initialized with random weights. The idea being that, since fminsearch() is a type of random search, and the steady-state controller was built with no latency, it might be entirely different than a controller built to handle latency, causing the search to take longer than if starting from a random position. The first time I ran the optimization, I used a range of latencies from 1 second to 4 seconds. The results were very poor, and the controller did not perform very well at all. I ran the optimization once more with a range of latencies from 1 second to 3 seconds. Considering the K_{ss} controller was only effective at latencies up to 2 seconds, even getting to 3 seconds would be quite an improvement. Unfortunately, with the linear controller based on the initial K_{ss} controller, I could not make it any more robust to latency. It is worth mentioning that the controller optimized for latency did result in modest cost reductions at latencies of 2 and 3 seconds, as shown in Figure 7 below. However, due to the stochastic nature of the model, this could have just been randomness at play. NED plots of the controller optimized for various latencies can be found in Figure 8. We can see how the latency of 3 seconds caused divergence.

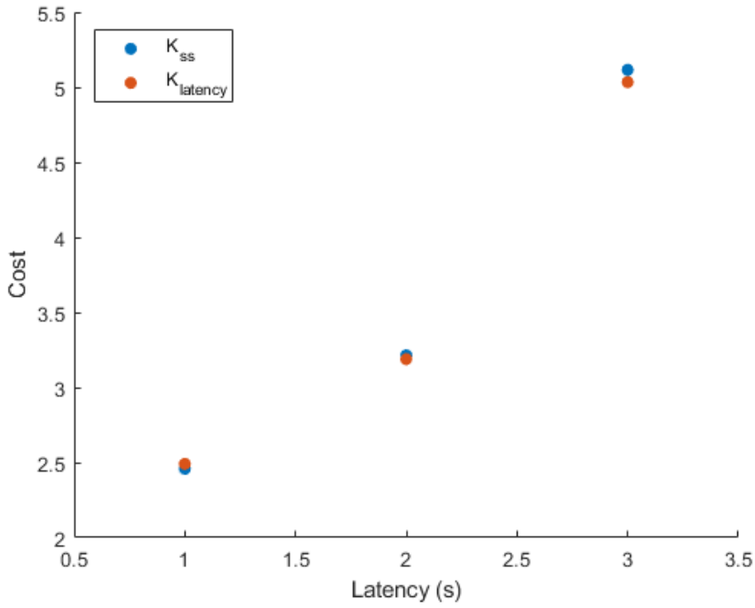
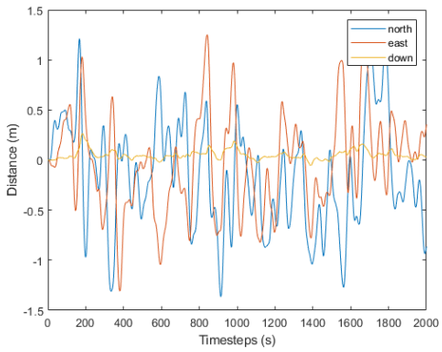
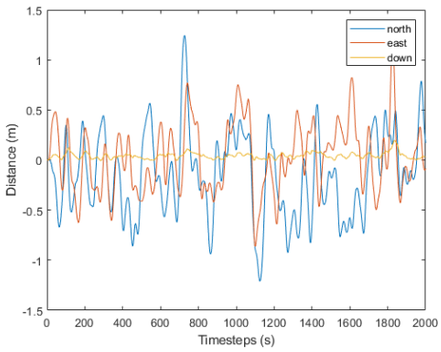


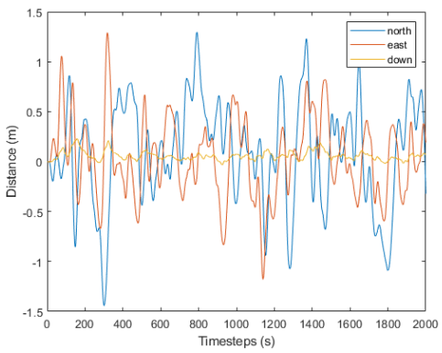
Figure 7: Latency vs Cost



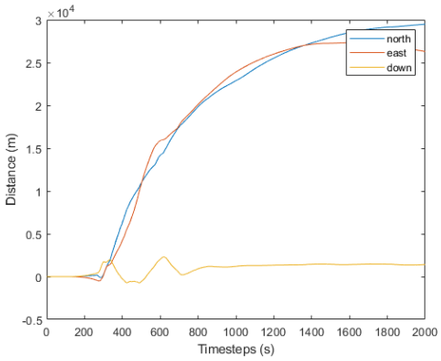
(a) $K_{latency}$ Controller, latency=0



(b) $K_{latency}$ Controller, latency=1



(c) $K_{latency}$ Controller, latency=2



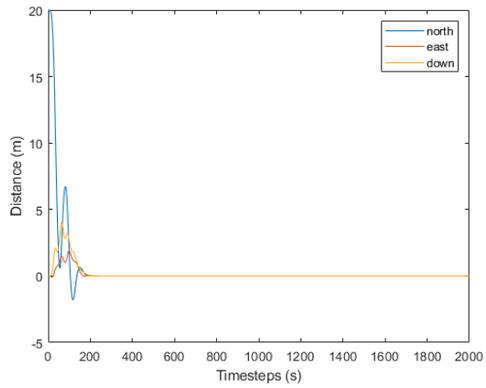
(d) $K_{latency}$ Controller, latency=3

Figure 8: NED Plots for Various Latencies

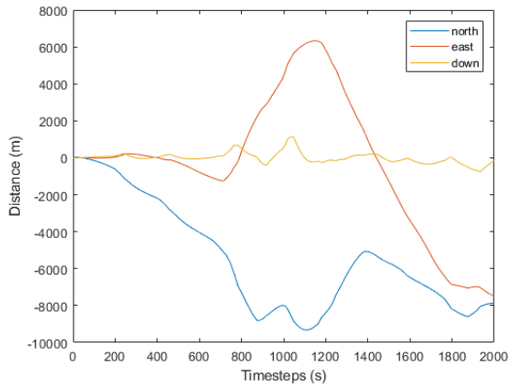
5 Conclusion

The biggest takeaway from this assignment for me is how well a linear controller performs using linearized dynamics with a nonlinear system. Considering how relatively "simple" the design is, it performs quite well. When testing how far from the linearization point we could go, the minimum distance of 10 m in the e direction is still quite large (and 50 m in the d direction seems incredible). The same holds for the rotation angles. By implementing clipping we were able to effectively return to the target hover state after starting 10,000 meters away. I am still very curious to see how much I could have potentially improved performance with a nonlinear controller, but the amount of effort to build one is huge compared to building the simple linear one, and that is probably a very valuable lesson.

6 Appendix

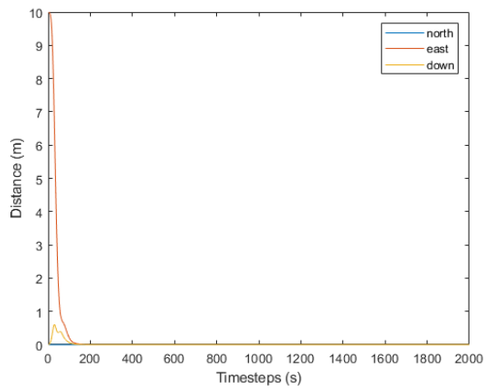


(a) Starting $n = 20$

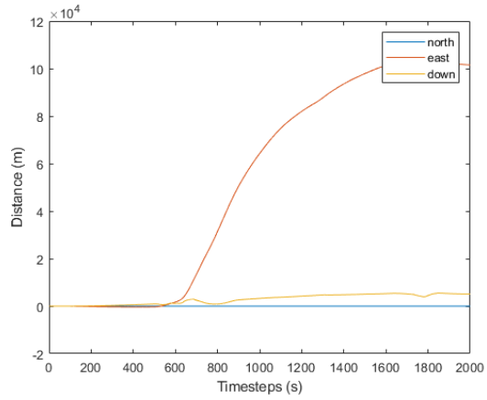


(b) Starting $n = 30$

Figure 9: Maximum Linearity about n

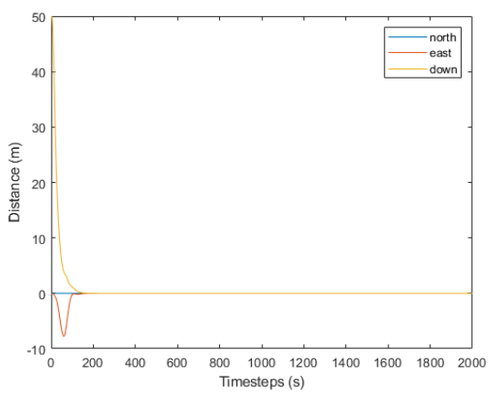


(a) Starting $e = 10$

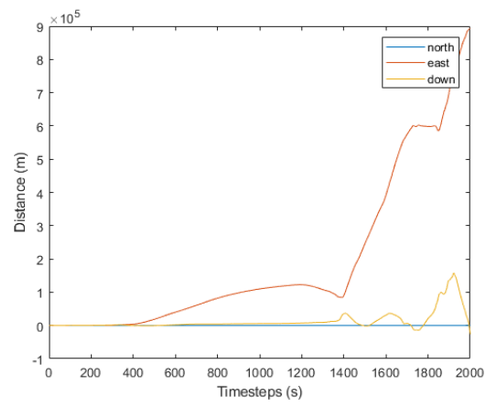


(b) Starting $e = 20$

Figure 10: Maximum Linearity about e

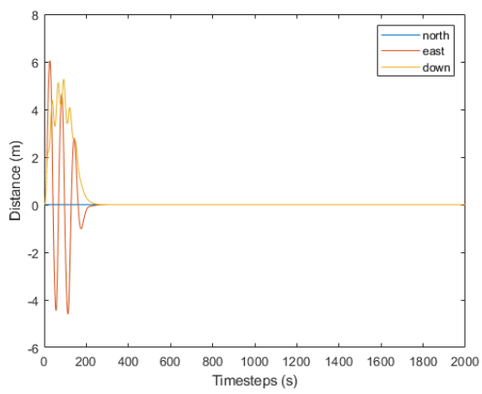


(a) Starting $d = 50$

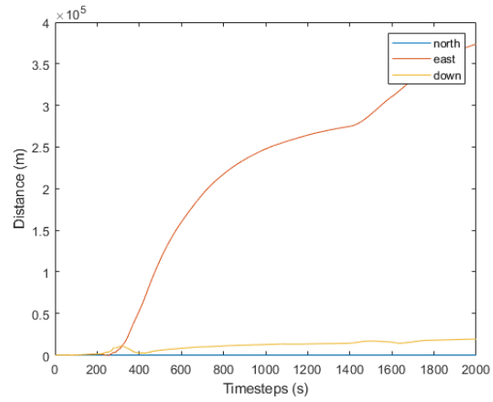


(b) Starting $d = 60$

Figure 11: Maximum Linearity about d

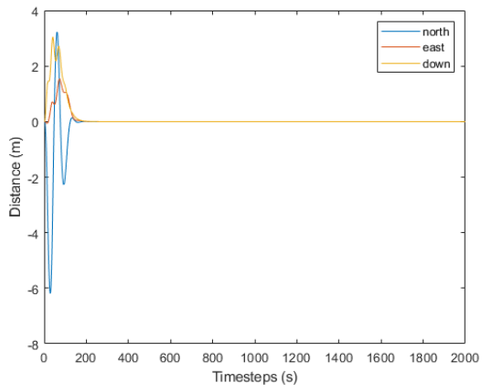


(a) Starting $x = \frac{5}{8}\pi$

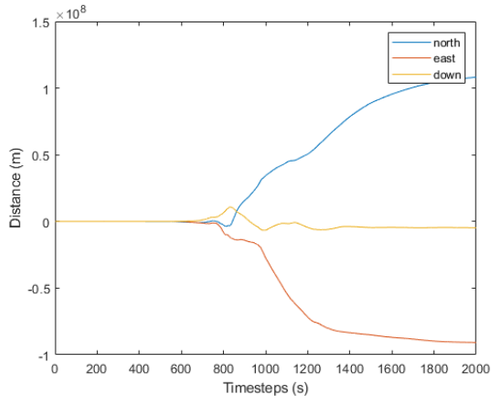


(b) Starting $x = \frac{3}{4}\pi$

Figure 12: Maximum Linearity about x

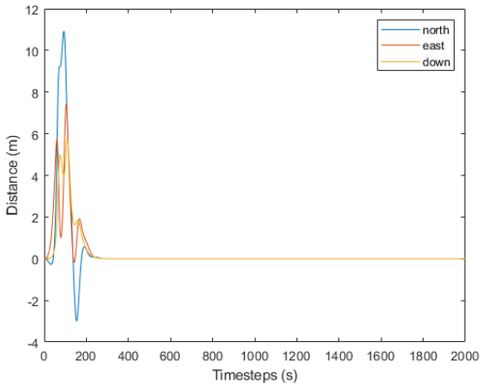


(a) Starting $y = \frac{1}{2}\pi$

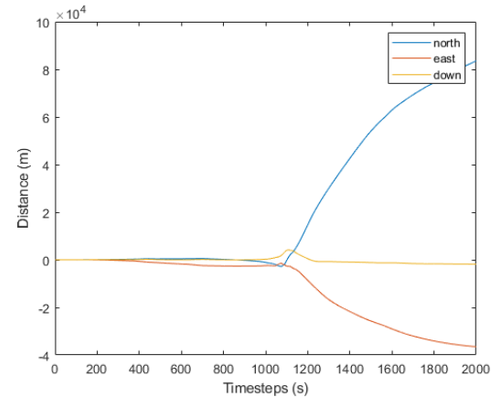


(b) Starting $y = \frac{5}{8}\pi$

Figure 13: Maximum Linearity about y



(a) Starting $z = \frac{5}{4}\pi$



(b) Starting $z = \frac{11}{8}\pi$

Figure 14: Maximum Linearity about z