Austin Anderson
ASEN5519
Homework 3

2. Park-Deyst Algorithm

Matlab was used to implement the algorithm described in the Park Deyst paper. The algorithm generates lateral acceleration commands based on:

$$a = 2\frac{V^2}{L_1}\sin\eta$$

Where L_1 is a fixed distance to a point that intersects the trajectory in front of the aircraft, V is the aircraft's airspeed, and η is the angle that the velocity vector makes with the L_1 vector.

The control inputs to the aircraft dynamics model are turn rate, airspeed, and climb rate. The aircraft will be commanded to move at a constant airspeed, so lateral acceleration can be converted to command turn rate. The model of the aircraft is a unicycle model which decouples lateral motion and altitude. The airspeed input is then an input selected by the user, the Park-Deyst algorithm will generate turn rate commands, and as lateral motion is decoupled from altitude, climb rate will simply be calculated based on a proportional controller. The altitude control will be based on the error between the L_1 point altitude and the current aircraft altitude.

The most challenging aspect of this algorithm is determining the L_1 point. To simplify this calculation, an elliptical trajectory will be pre-computed as a series of discrete way-points. The L_1 point is then found as the way-point most nearly corresponding to the fixed L_1 length. To further simplify the calculation, once a way-point is initially selected, only the next n-points will be considered in the search for L_1. This algorithm is only stable if the aircraft is close enough to the desired trajectory to find an appropriate L_1. For this work, only initial conditions falling within a radius L_1 of the first way-point will be considered.

The system was simulated for a 10000-point trajectory, looking at 1000 points at a time, and only accepting way points if they fall within 1-meter of the L_1 length.

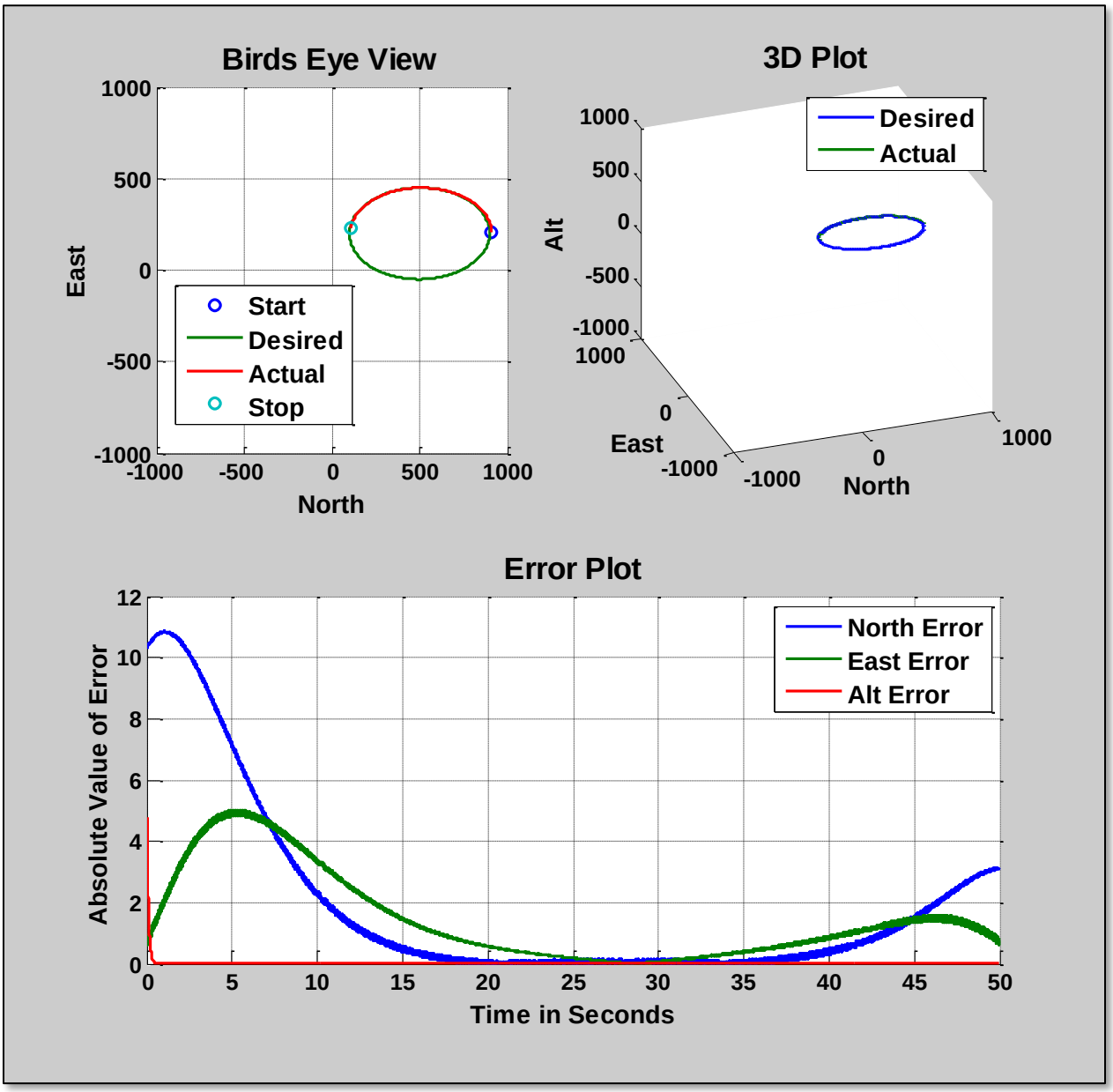

Figure 1. L1 = 100, p=[500,200,150]T.

The aircraft model was simulated by using small time steps in an iterative loop. The system was simple enough to be well approximated by numeric integration and differentiation. This was chosen instead of Simulink because this version of the algorithm uses a way-point look up table that was easier to manipulate in a software loop.

The algorithm was found to achieve good tracking with maximum tracking errors of around 4 m (absolute error, not squared). One subtlety of the current approach is that it does not handle index wrapping. This is a minor bug in the code that could be quickly fixed, but was ignored for the purposes

of this lab. This means that simulations can't complete a full period of the ellipse. Different initial pointing conditions were used:

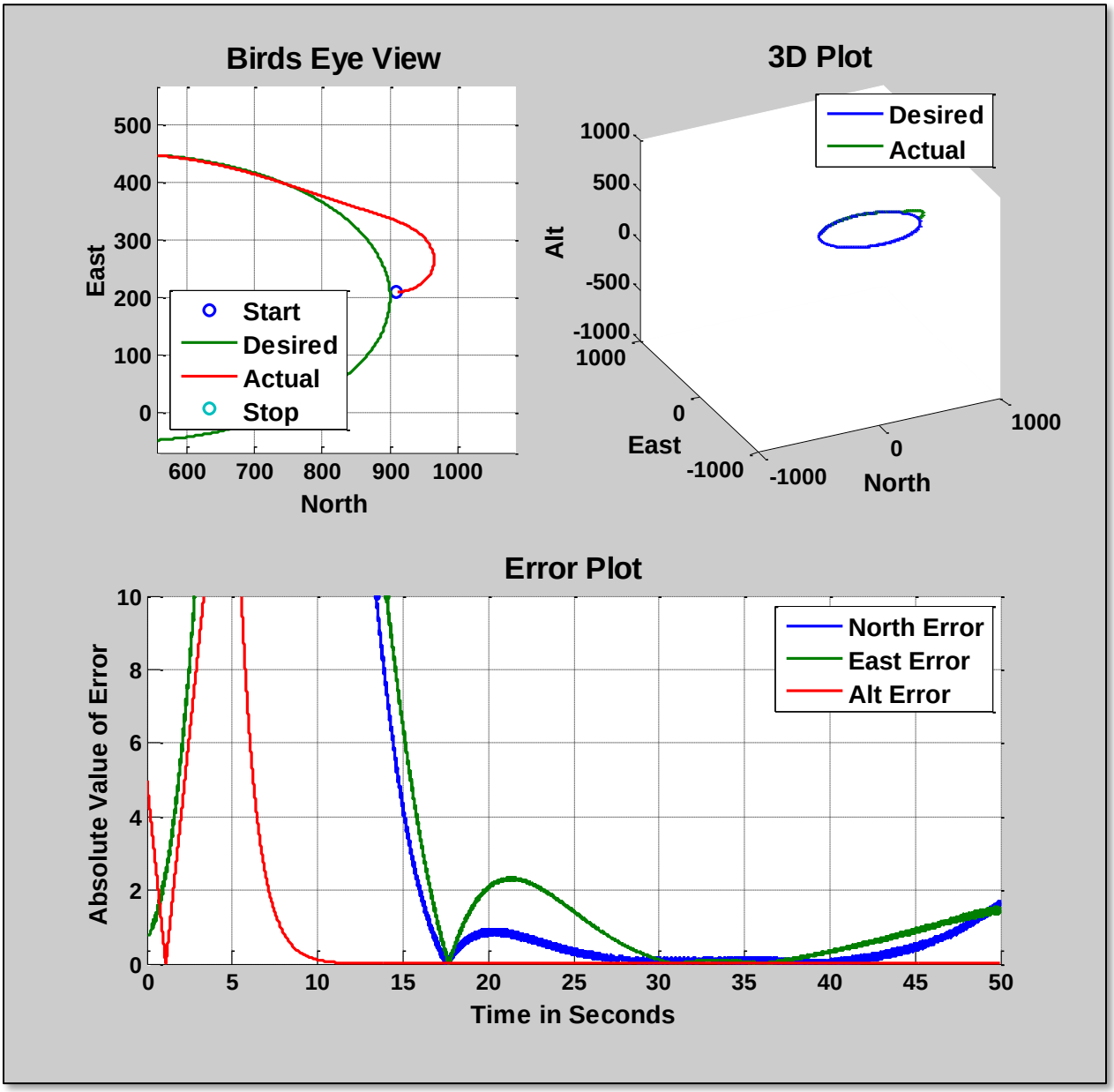

Figure 2. L1 = 100, initial course angle of 0 degrees.

At least with initial non-zero course angles, the guidance converges on the desired course. As expected initial error is high, until it settles back to the about 4 m errors.

Different values of L1 were used, and resulted in better error:

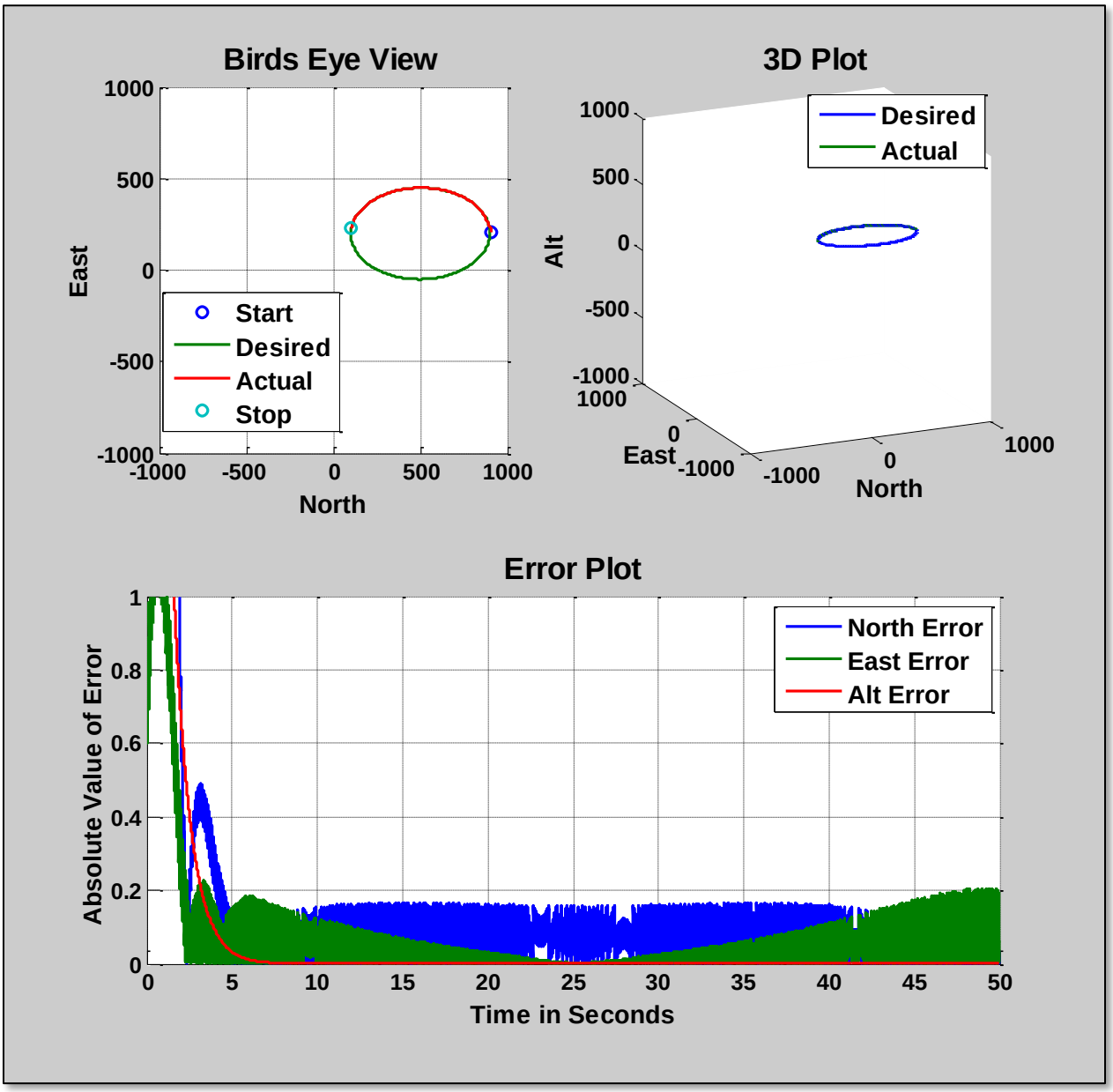

Figure 3. L1 = 20

It was found that smaller values of L1 resulted in better tracking accuracy. As an aside, that chatter in the error is a function of how the error is calculated in the code and is caused by quantization effect.

The algorithm was used to simulate a slanted ellipse trajectory next:

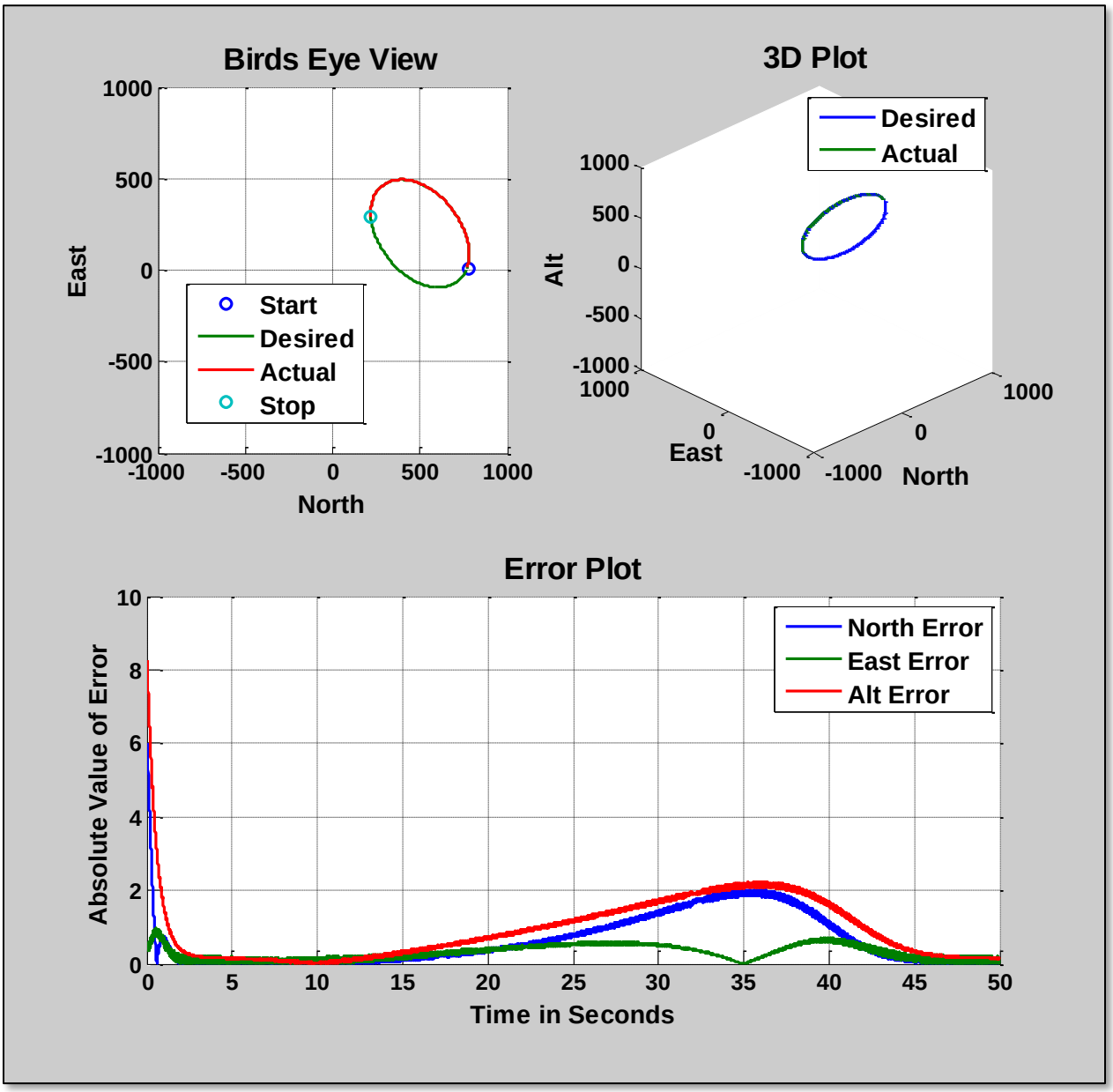

Figure 4. L1 = 10, Slanted trajectory.

The Park-Deyst algorithm was used to track lateral guidance commands and a simple proportional guidance controller was used to track the altitude. The proportional control did a good job tracking the change in altitude generated by the now slanted ellipse trajectory. Generally, both worked well, but struggled to track as the aircraft hit the trough of the trajectory and began to climb. This is somewhat expected as the highest accelerations should occur in the trough and peak. A closed form solution for ellipse trajectory intersection may improve performance.

Part 2. LGVF Algorithm

The algorithm described in the second paper was implemented in Matlab. Like the pervious algorithm, the dynamics of the system were approximated in a software loop using numeric integration and differentiation. Initially, a pre-computed trajectory was used to generate the vector field. In this method, the current angular position of the aircraft was used as a look-up index in the way-point memory. To achieve higher performance, a first-order linear interpolation was implemented to generate a fine estimate of desired position. The partial derivatives in the radius and altitude with respect to angular position were numerically calculated based on the interpolated trajectory positions. The controller ultimately generated desired velocities in North, East, and altitude axis. The altitude velocity could be directly passed to the aircraft model. The desired turn rate was calculated based on the error between the current velocity vector angle and the desired velocity vector angle (both in the North-East plane). This error was scaled by a proportional gain and a feed forward term was added to help in tracking a constantly curving elliptical trajectory. The change in error from previous time step to current is calculated and projected forward one simulation time step to determine the feed forward term.

This discrete trajectory method worked well for tracking circular trajectories:

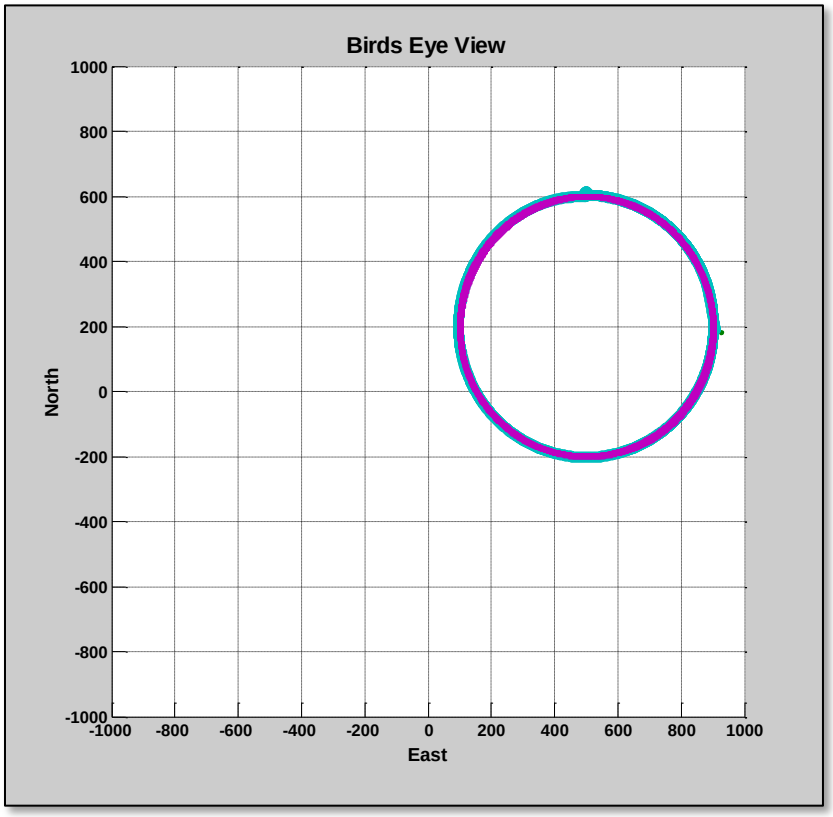

Figure 5. Circular trajectory r = 400. Cyan is aircraft path, purple is desired trajectory.

There was a subtle wrapping issue that occurred when the aircraft course angle crossed 180°. The shift between +/- 180° caused the measured angle error to explode. Coupled with a software saturation limit on turn rate commands caused the aircraft to enter a limit cycle that forced it to spiral away. A limit of the size of the angle error helped alleviate this problem, but a small circular orbit around that crossing point is still visible in Figure 4 at the Northern most point of the trajectory.

The discrete way-point method suffered from quantization effects, and struggled at tracking ellipses and slanted trajectories. The method was abandoned for a closed form approach. Using Matlab's symbolic toolbox, a closed form function for calculating the vector field based on elliptical shape and current aircraft position was prototyped and developed into an m-file. The initial cut of the algorithm worked for flat, elliptical trajectories centered at the global origin. To account for non-zero trajectory origins, the aircrafts position is initially adjusted in the control loop to account for the offset. The change in velocity commands don't have a dependence on absolute location, so this is valid. Initial simulation results are shown below:

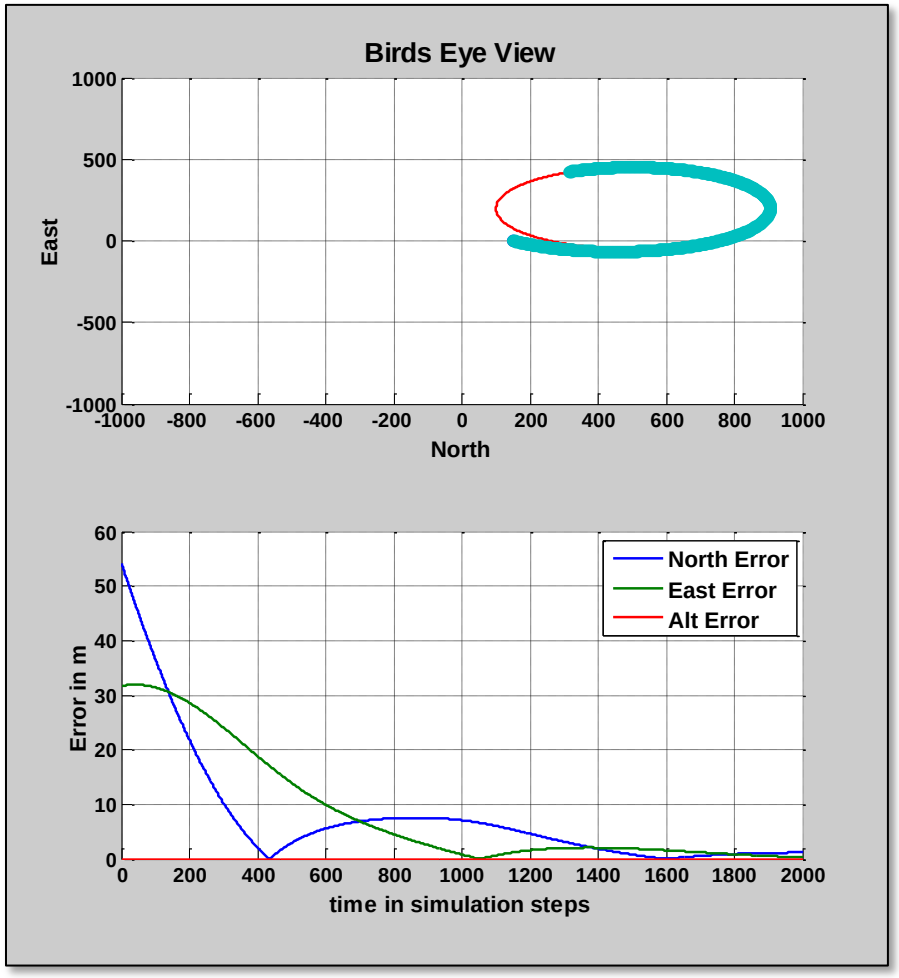

Figure 6. k1,k2=2, kp = 4, kf = 2

The vector field did a much better job at tracking the curve. The error starts high, but falls below 10m after the aircraft closes on the trajectory. Adjusting values for the vector field k's (k1,k2) and control ks (kp=proportional, kf = forward) should improve the error:

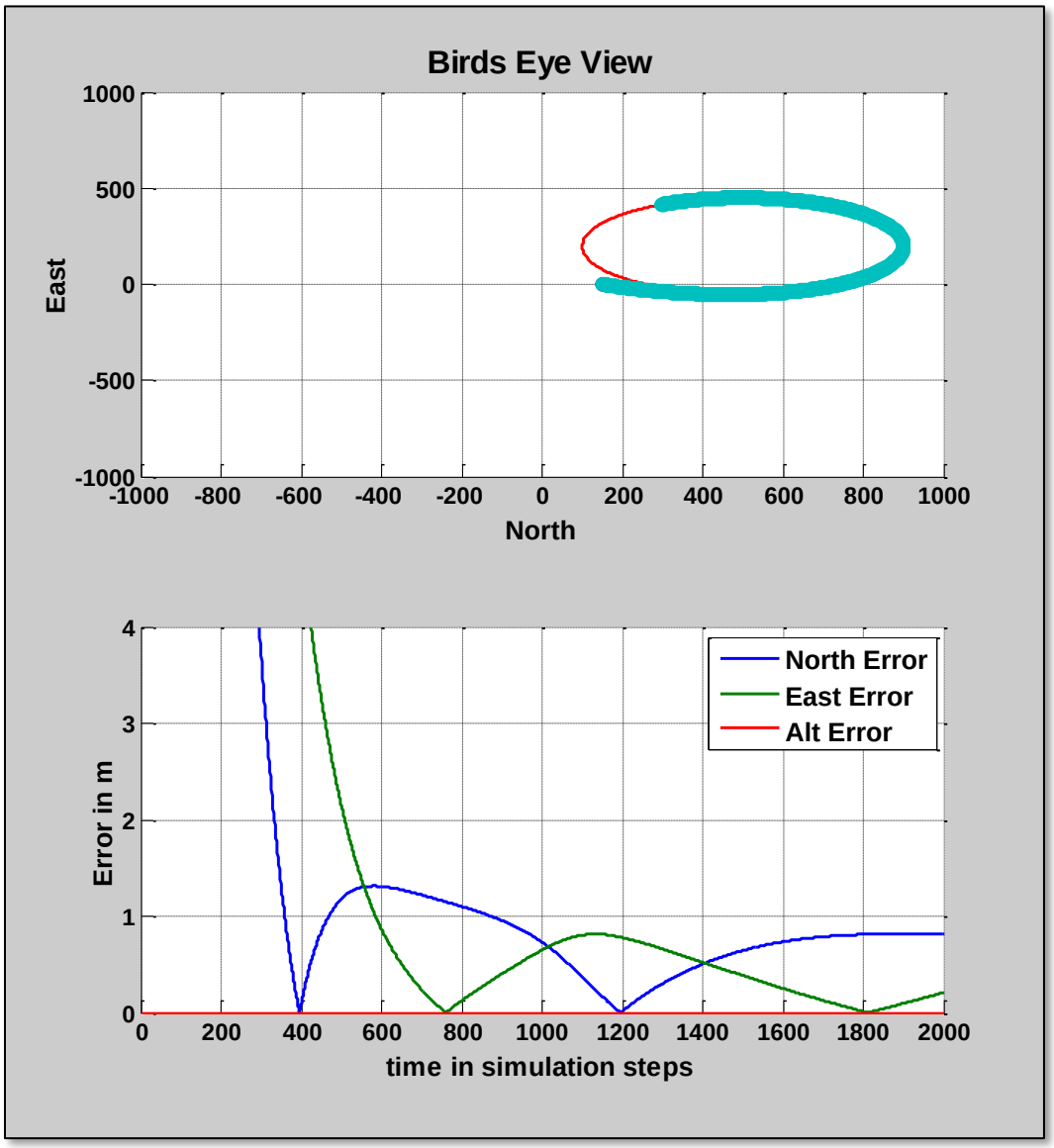

Figure 7. k1,k2=5, kp = 8, kf = 4

Now after the transient error caused by the initial conditions, the steady-state error falls to below 2m (absolute).

The algorithm was then updated to account for a tilted elliptical trajectory. In this scenario, the aircraft's position is translated to an ellipse-centric coordinate frame. This is done by first subtracting the aircraft's Cartesian global coordinates by the coordinates of the center of the ellipse. The now

shifted aircraft coordinates are then rotated by the inverse of the yaw-pitch-roll rotation matrix. The coordinates are now in the frame of the ellipse plane centered on 0,0,0. The newly rotate coordinates are transformed to cylindrical coordinates. The normal control law is used to generate the vector field, which is translated to Cartesian. The Cartesian velocity vectors are still relative to the rotated ellipse frame, so a normal yaw-pitch-roll rotation is applied to get it back to the aircraft global velocity frame. These commands are then fed into the normal controller and aircraft dynamics model. For a slanted ellipse with a yaw of 30 degrees and pitch of 40 degrees:

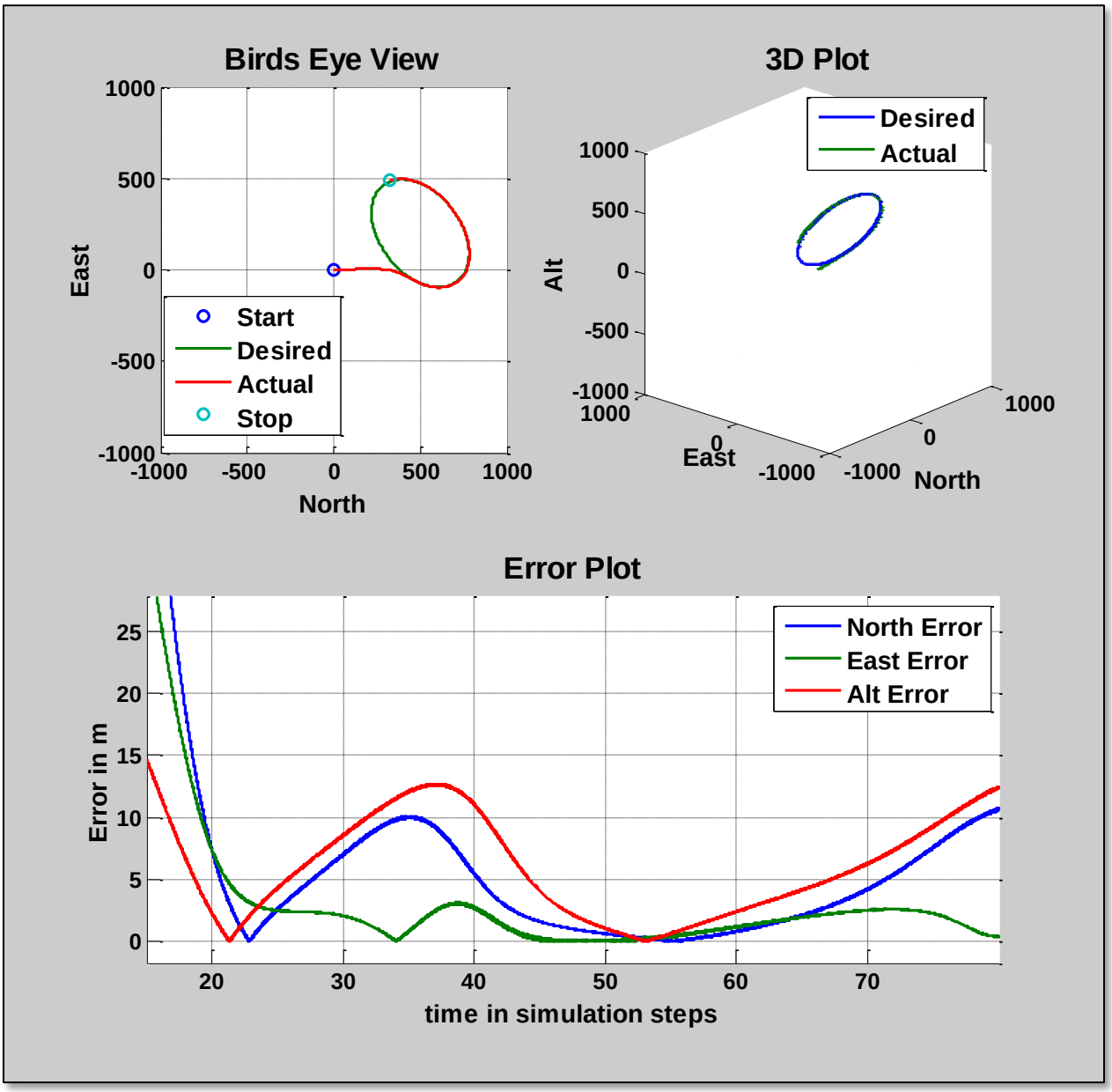

Figure 8. k1,k2 = 10, kp = 16, kf = 8

Like the Park-Deyst algorithm, there is more error in the tilted scenario. Again, the controller seems to struggle to keep up with the trough of the maneuver.

Part 3. Delay

In this simulation, the autopilot is assumed to perfectly track the aircraft. In reality, the autopilot will only update at a particular rate, while the aircraft's actual dynamics occur continuously. This can appear as a time delay between the autopilot and the aircraft dynamics. To approximate this in the software loop simulation, the guidance and controller will only be updated every m-simulation cycles while the aircraft dynamics will be updated every simulation cycle. The LGVF was simulated first. For a 250 Hz simulation and 12.5 Hz autopilot update (m=20), the behavior is largely the same:

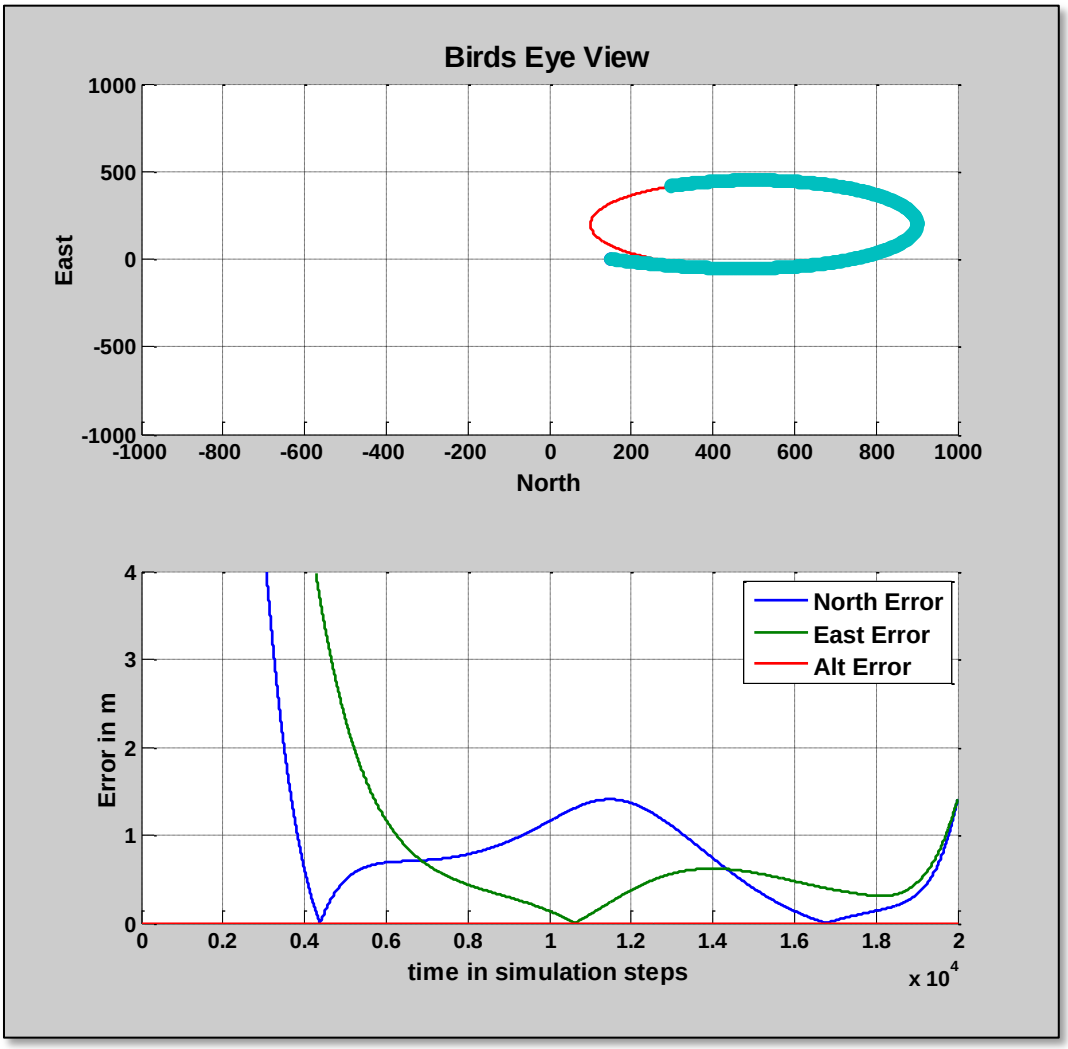

Figure 9. Same simulation as Figure 6 and m=20.

At a autopilot rate of 9.6 Hz (m=26), the control starts to chatter and the error increases significantly:

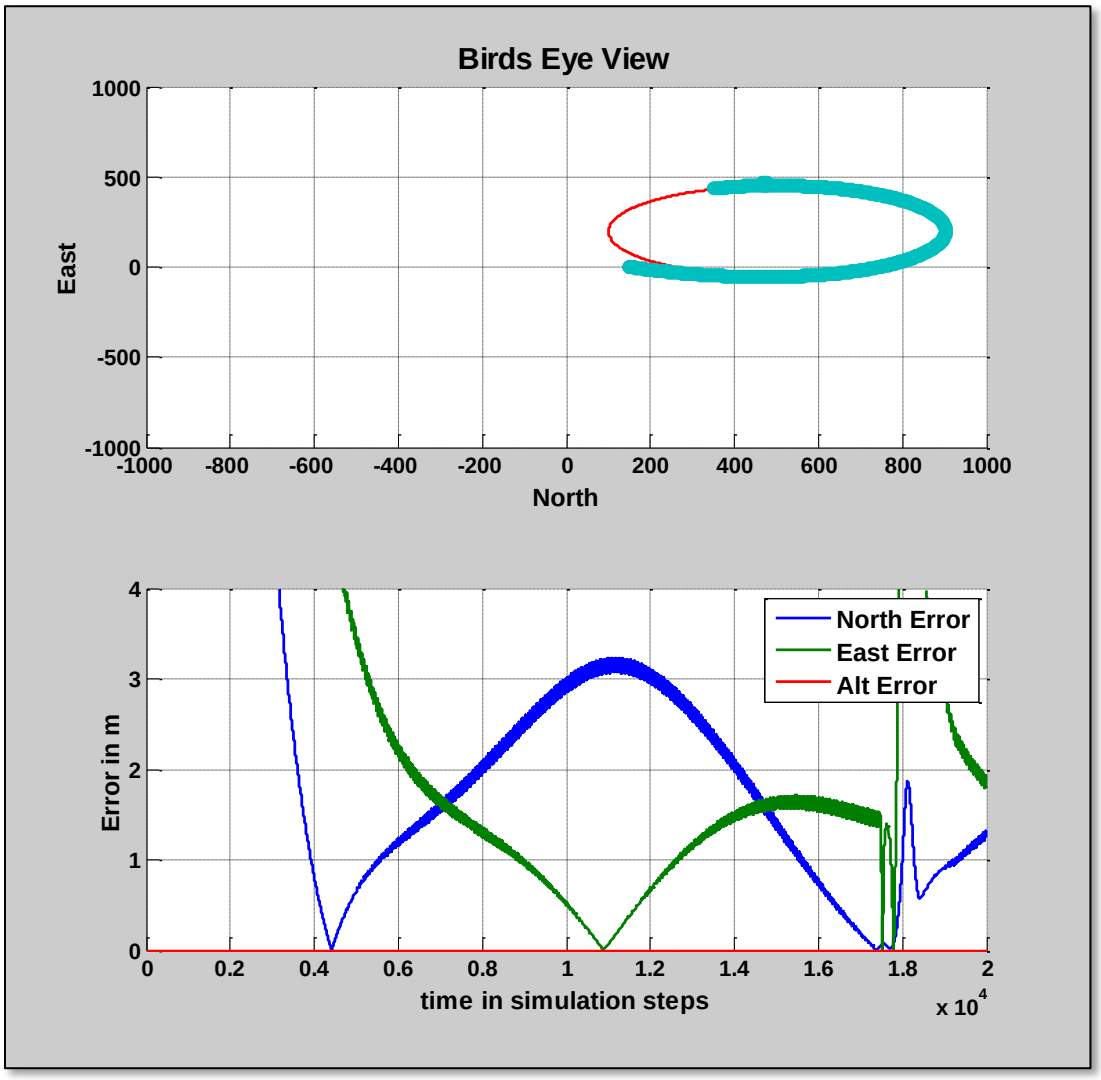

Figure 10. Same simulation as Figure 6 and m=26. The error in North and East shows chatter.

For the Park-Deyst algorithm, the control seems to chatter when m is set to be greater than 20 (12.5 Hz autopilot) for a 250 Hz simulation rate:

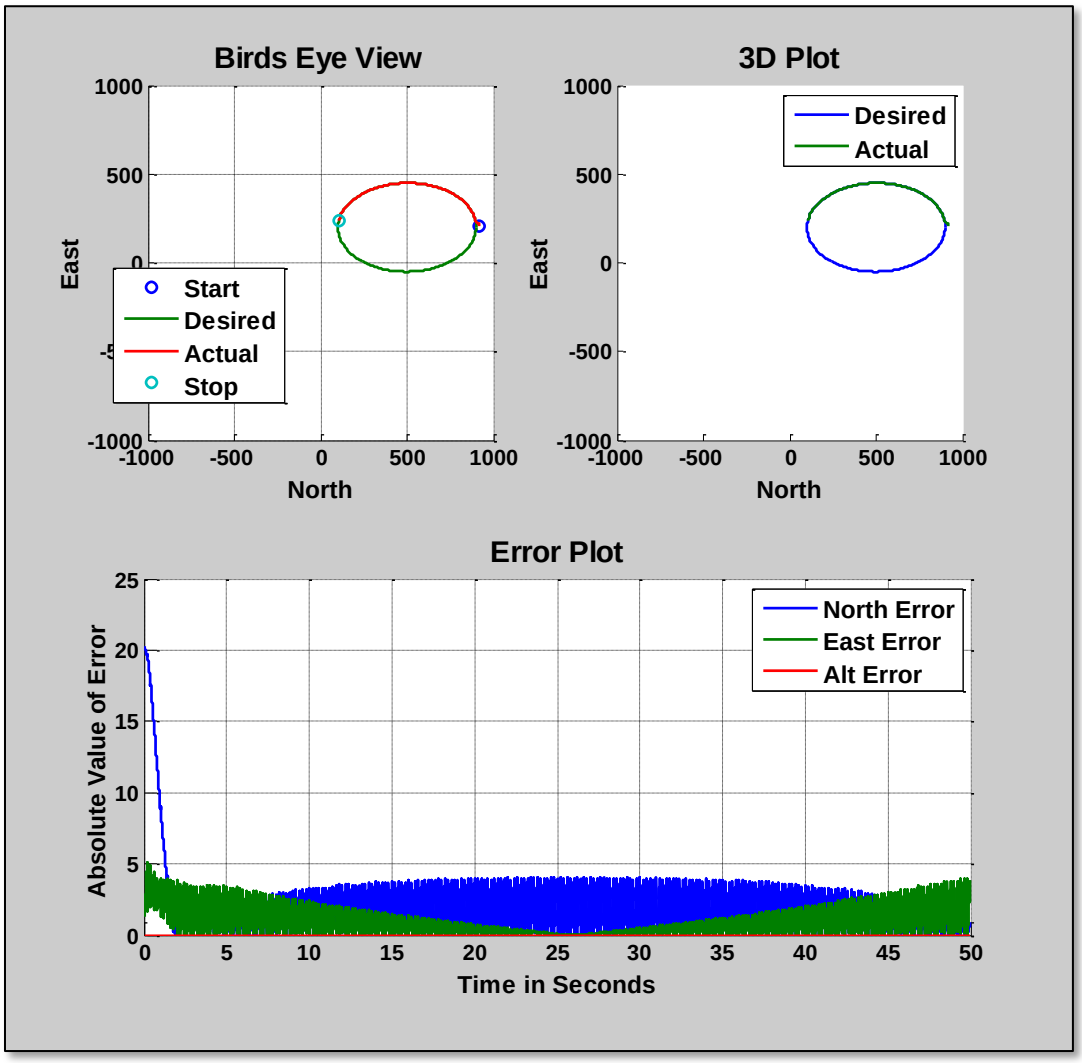

Figure 11. Same as simulation for figure 1. 5 Hz autopilot for 250 Hz simulation (m=50).

In conclusion, an autopilot rate greater than 10 Hz seems to produce good tracking for the LGVF algorithm, while rates less than 10 Hz seem to produce chatter. Rates faster than 8 Hz seem to produce good tracking for the Park-Deyst algorithm. Generally, both of these values are tuned for an aircraft moving at around 20 m/s. If the aircraft was moving faster, the autopilots would necessarily have to react faster.