Development of a Three Degree-of-Freedom Simulation for Thrust Vectoring Vehicles

Govind Chari

In this paper I will discuss the dynamics, control, and state estimation for a three degree-of-freedom thrust vectoring vehicle. This is a final project report for MAE 4780: Feedback Controls

Nomenclature

$\boldsymbol{x}$	System State
$\boldsymbol{y}$	Observed State
$\boldsymbol{u}$	Control Input
$\boldsymbol{r}$	Reference State
A	Linearized Continuous-Time State Transition Matrix
B	Control Matrix
C	Observation Matrix
F	Linearized Discrete-Time State Transition Matrix
$P_{a\|b}$	State Covariance Matrix at time a given information at time b
$\hat{\boldsymbol{x}}_{a\|b}$	Estimated state at time a given information at time b
Q	Process Noise Covariance
R	Sensor Noise Covariance
K_f	Kalman Gain
K_c	Full-State Feedback Gain

1 Introduction

Over the summer, I worked with a friend to develop a vertical takeoff and landing vehicle powered by two axially aligned, counter-rotating (for roll control), racing drone motors with servo-driven thrust vectoring fins for vertical stabilization. When designing the controller for this vehicle, I utilized a PID controller with manually tuned gains. I did not consider vehicle dynamics. As a result, the vehicle crashed many, many times before successfully stabilizing. After taking Feedback Controls, I realized I now have the ability to systematically design a controller, so I decided to revisit the problem as my final project. However, due to time restrictions, I decided to model the problem in three degrees of freedom rather than the full six.

2 Theory

2.1 Dynamics

The two forces considered in the dynamics of the vehicle are gravity and the thrust from the propulsion system. A free body diagram can be drawn and analyzed to derive equations of motion for the system.

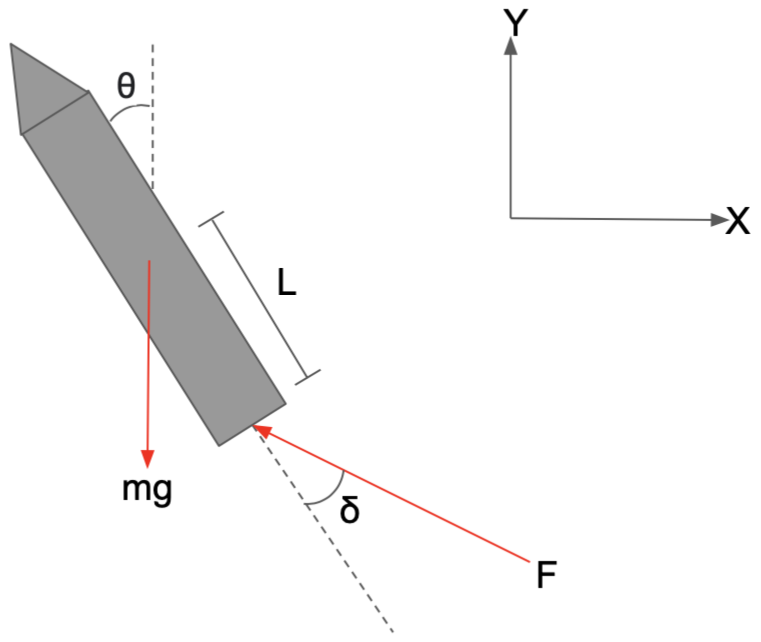

Figure 1: Free Body Diagram

m	vehicle mass
I	rotational inertia
g	gravitational acceleration
x	horizonatal displacement
y	vertical displacement
θ	angular displacement
F	magnitude of thrust
f	magnitude of thrust in excess of vehicle weight ($f = F - mg$)
δ	angle of thrust vector
L	moment arm

By summing forces in the x and y directions and summing moments, the equations of motion can be derived.

$$\ddot{x} = -\frac{F sin(\theta + \delta)}{m} \tag{1}$$

$$\ddot{y} = \frac{F cos(\theta + \delta)}{m} - g \tag{2}$$

$$\ddot{\theta} = -\frac{F L sin(\delta)}{I} \tag{3}$$

By introducing f, the equations can be rewritten

$$\ddot{x} = -gsin(\theta+\delta) - \frac{f}{m}sin(\theta+\delta) \tag{4}$$

$$\ddot{y} = \frac{f}{m}cos(\theta+\delta) \tag{5}$$

$$\ddot{\theta} = -\frac{mgL}{I}sin(\delta) - \frac{fL}{I}sin(\delta) \tag{6}$$

These equations will now be linearized assuming θ,δ, and f are small.

$$\ddot{x} = -g\theta - g\delta \tag{7}$$

$$\ddot{y} = \frac{f}{m} \tag{8}$$

$$\ddot{\theta} = -\frac{mgL}{I}\delta \tag{9}$$

The dynamics can now be written in canonical state space form.

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u} \tag{10}$$

$$\frac{d}{dt}\begin{pmatrix} x \\ y \\ \theta \\ \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -g & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ \theta \\ \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & -g \\ \frac{1}{m} & 0 \\ 0 & -\frac{mgL}{I} \end{pmatrix} \begin{pmatrix} f \\ \delta \end{pmatrix}$$

2.2 Controller Design

As a result of the separation principle, the controller and estimator can be designed separately and still ensure stability, given the controller and observer only have stable eigenvalues.

Firstly, I checked controllability of the system in Matlab. Intuitively it can be seen that with variable thrust and thrust angle, the system can reach any state in the state space. Sure enough, the controllability matrix is full rank, and thus the eigenvalues of the closed-loop system can be placed anywhere with a judicious choice of K_c. The control law I utilized takes the following form.

$$\boldsymbol{u} = K_c(\boldsymbol{r} - \boldsymbol{x}) \tag{11}$$

I made no attempt to design any sort of optimal controller. Frankly, I do not have enough knowledge of optimal controls yet, and as a result I chose the closed-loop eigenvalues rather arbitrarily. The only restriction I kept in mind was not placing my eigenvalues too aggressively. Aggressive eigenvalues result in sharp corrections which could quickly result in the system leaving the linear regime of the dynamics model, which can result in instabilities.

After some experimenting, I decided on eigenvalues of -1, -2, -3, -4, -5, and -6, as they resulted in good stability while still being conservative enough to keep the system in the linear regime.

2.3 Estimator Design

A typical avionics suite would be a GPS, IMU, and barometer. Thus, the measurements taken would by x, y, and $\dot{\theta}$. The observation equation is linear and can be written in canonical form below.

$$\boldsymbol{y}=\begin{pmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\0&0&0&0&0&1\end{pmatrix}\boldsymbol{x}$$

Firstly, observability needs to be proven. Intuitively, if we differentiate the x and y terms, we can get $\dot{x}$ and $\dot{y}$, and if we integrate $\dot{\theta}$ we get θ. Sure enough, the observability matrix is full rank, so the system is observable with the given measurements and dynamics.

Since the dynamics are nonlinear, I decided to use a discrete-time Extended Kalman Filter as the state observer. The EKF uses a similar predict-update algorithm that is used by the linear Kalman Filter, but firstly the dynamics are linearized about the previous state estimation and current control input via computation of the Jacobian. This is a necessary step to ensure that the state's probability distribution remains Gaussian after state propagation. The algorithm is mathamatically layed out below.

Linearization
$F = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}|_{\boldsymbol{x}_{k-1},\boldsymbol{u}_k}$

Predict
$\hat{\boldsymbol{x}}_{k|k-1} = \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)$

$\mathrm{P}_{k|k-1} = FP_{k-1}F^T + Q$

Update
$\mathrm{S}=\mathrm{CP}_{k|k-1}C^T + R$

$\mathrm{K}_f = P_{k|k-1}C^TS^{-1}$

$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_{k|k-1} + K_f(\boldsymbol{y} - C\hat{\boldsymbol{x}}_{k|k-1})$

$\mathrm{P}_k = (I - K_fC)P_{k|k-1}$

3 Simulation

3.1 Overview

I made a Simulink model for the system's dynamics, control, and state estimation. In this simulation, I attempted to make the vehicle follow a simple ascent, translation, descent trajectory similar to the SpaceX SN5 and SN6 tests. I used the following vehicle parameters.

m	1 kg
I	0.002 kg m^2
L	10 cm

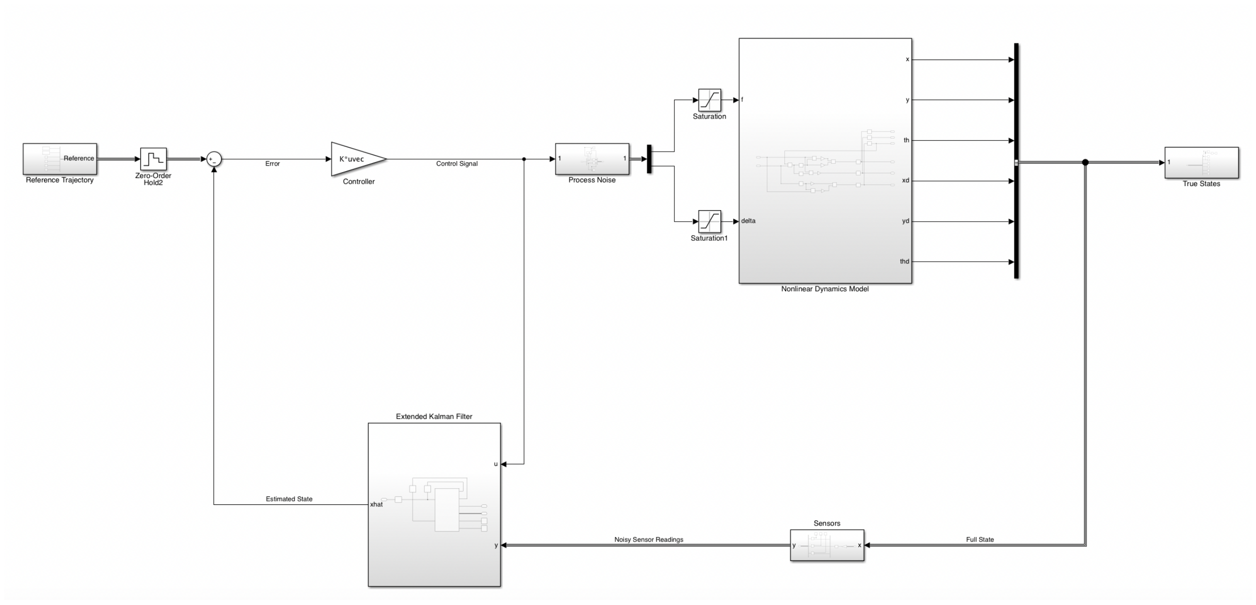

Figure 2: Full Block Diagram

The zero-order holds are meant to simulate flight software with a 100Hz control cycle, which was the quickest that data could be pulled from the IMU and Barometer that I used over the summer. Furthermore, I utilized two saturation blocks for the actuators. I limited the thrust vector angle to within 10 degrees from equilibrium. Similarly I allowed the thrust to range from 0 to $2mg$. This assumption means that the vehicle has a maximum thrust to weight ratio of 2. Most real engines have a lower throttle bound, in fact the SpaceX Falcon 9 booster lands with a minimum thrust to weight ratio greater than one. However, I will ignore this aspect for this model. It would be a fun problem to try to revisit at some point.

3.2 Reference Trajectory

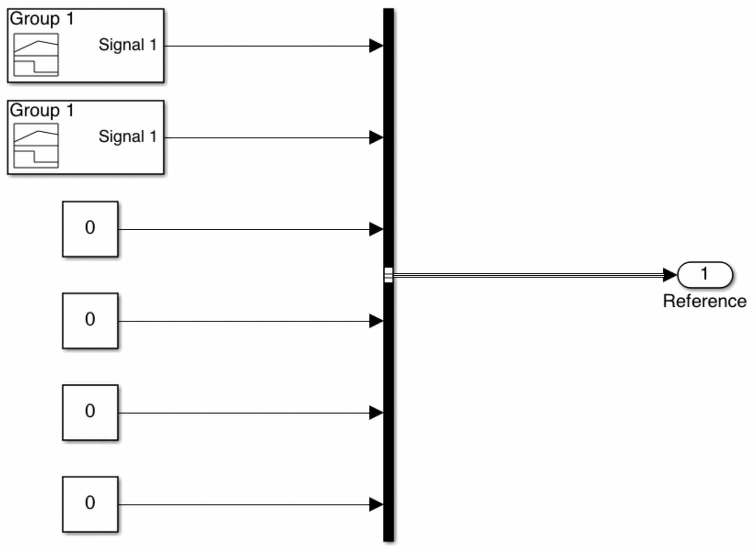

Figure 3: Reference Trajectory Block

The first two signals describe the reference x and y positions. I designed these signals to generate the ascent, translation, descent trajectory that I want. The rest of the reference signals are set to zero. The exact shape of the x and y reference trajectories can be found in the appendix.

3.3 Controller

The feedback controller inputs the state error and outputs the desired control commands. To design the controller, I used the place command in Matlab to place the poles at -1, -2, -3, -4, -5, and -6 for reasons mentioned in the Controller Design section. The controller gain matrix is shown below.

$$K_c = \begin{pmatrix} -10.66 & 13.12 & 45.94 & -13.68 & 7.59 & 4.81 \\ 0.0127 & -0.0015 & -0.1245 & 0.0226 & -3.65e-4 & -0.0278 \end{pmatrix}$$

3.4 Dynamics Model

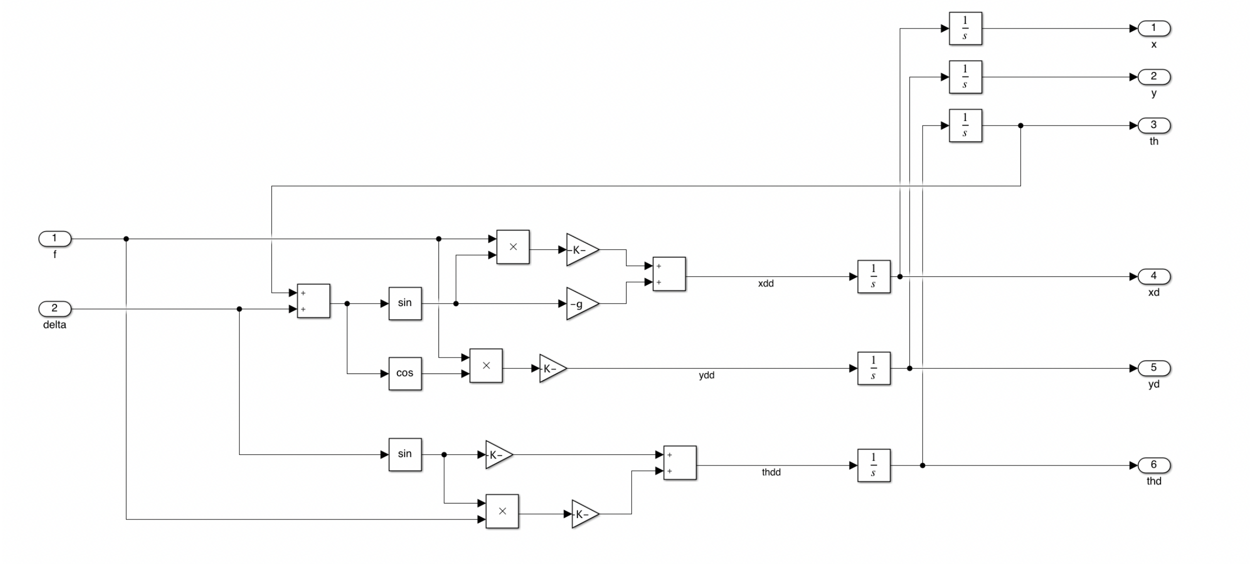

Figure 4: Nonlinear Dynamics Block

When I designed this block I used the full nonlinear equations of motion (Equations 4, 5, and 6) in order to see how effective the controller based on the linearized model would work to control the full nonlinear plant. The dynamics block does all computations in continuous time to mimic real dynamics.

3.5 Sensors/Noise

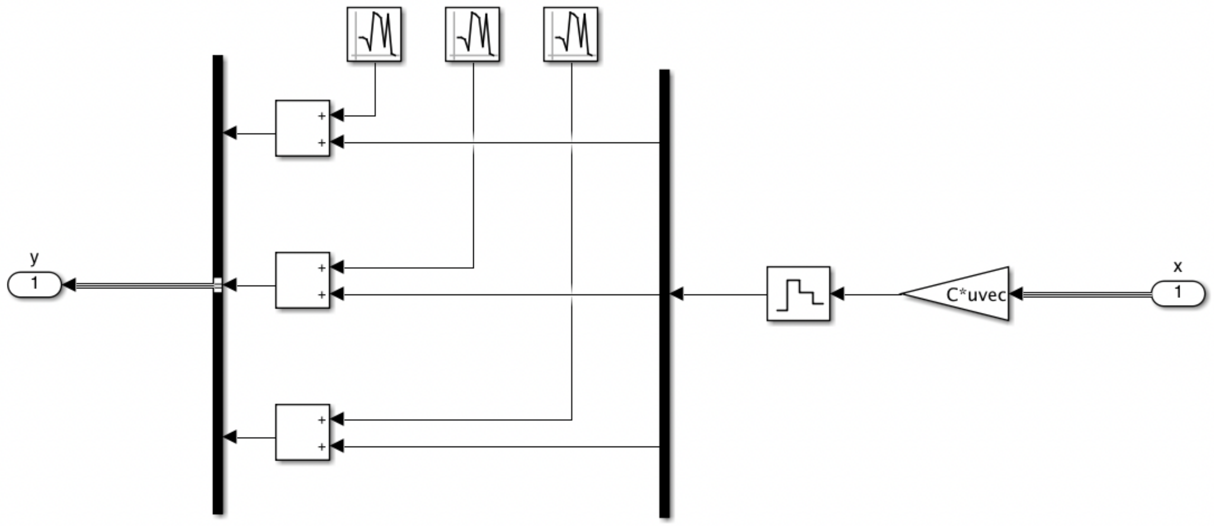

Figure 5: Sensor Block

First, the full state is multiplied by C to extract x, y, and $\dot{\theta}$, which are the measured states. Next, the measurements pass through a zero order hold to mimic sensors that sample at 100Hz. Finally, Gaussian noise is added to all three measurements to mimic sensor noise. For the x reading, I added noise with a standard deviation of 1 meter, which is what I observed with the GPS I used. For the y reading, I added noise with a standard deviation of 0.1 meters, which is what I observed with the barometer I used. For the $\dot{\theta}$ reading, I added noise with a standard deviation of 0.01 rad/s. I estimated this since I did not do any experiments on my gyro over the summer.

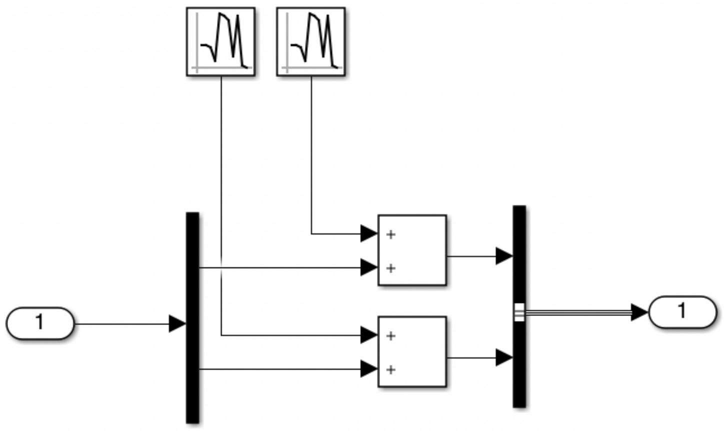

Figure 6: Process Noise

I added Gaussian process noise to both controller commands. I added noise with a standard deviation of 1 newton to the force and noise with a standard deviation of 0.5 degrees to the thrust vector angle command. I chose these numbers somewhat arbitrarily. I wanted them to be significant enough to disturb the system, but not too large to be infeasible.

3.6 Extended Kalman Filter

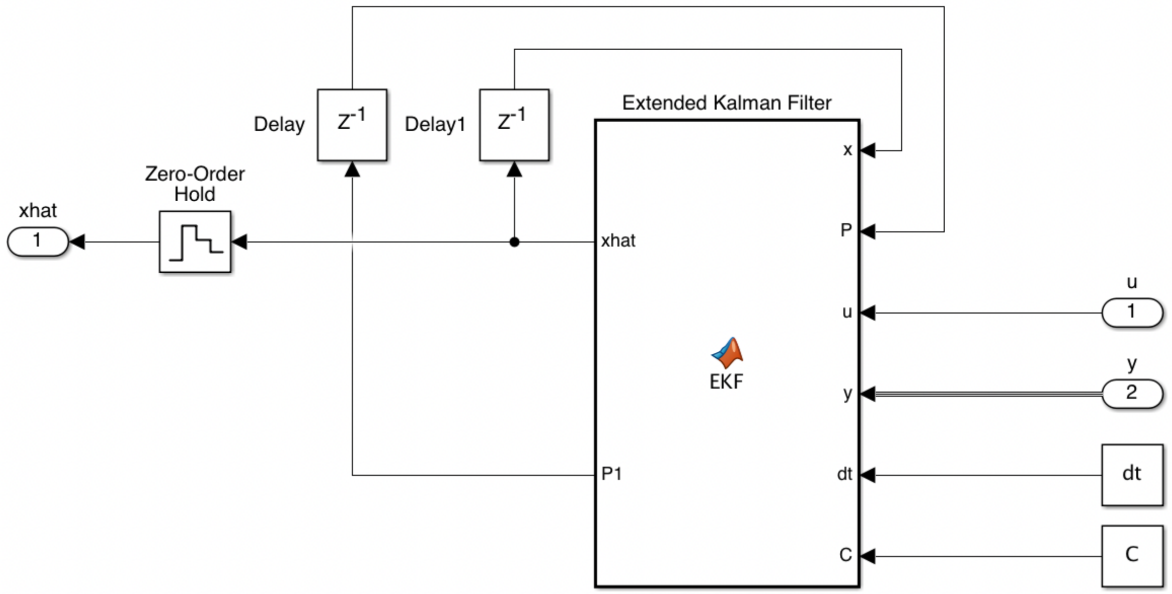

Figure 7: Extended Kalman Filter

The EKF block takes in the noisy sensor measurements $\boldsymbol{y}$ and the control inputs $\boldsymbol{u}$, and outputs the estimated state. I wrote an EKF function in Matlab, for simplicity (the code is included in the Appendix). The two delays are necessary for inputting the previous estimated state and the state covariance matrix into the function to calculate the next estimated state and state covariance matrix.

Firstly, a nonlinear state transition function must be written.

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_{k+1}) = \begin{pmatrix} \dot{x}_k \Delta t + x_k \\ \dot{y}_k \Delta t + y_k \\ \dot{\theta}_k \Delta t + \theta_k \\ -\left(g + \frac{f}{m}\right) sin(\theta + \delta)\Delta t + \dot{x}_k \\ \frac{f}{m} cos(\theta + \delta)\Delta t + \dot{y}_k \\ -\frac{L}{I}(mg + f) sin(\delta)\Delta t + \dot{\theta}_k \end{pmatrix} \tag{12}$$

The first step in the EKF is linearization via computation of the Jacobian. I computed this analytically beforehand for speed rather than numerically computing it with each control cycle.

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \begin{pmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & -\Delta t \left[cos(\theta + \delta) \left(g + \frac{f}{m}\right)\right] & 1 & 0 & 0 \\ 0 & 0 & -\Delta t \left(\frac{f}{m} sin(\delta + \theta)\right) & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{13}$$

To initialize the EKF I used a vector of all zeros for the initial state and the identity matrix for my initial state covariance. For the sensor noise matrix R, I used variances very similar to the actual sensor variances, since these numbers will be well known from sensor testing. For the process noise covariance, I used the actual injected variance transformed by the control matrix. It was difficult to make the filter converge unless the process noise was more or less well understood.

4 Results

In this section I will present plots that show the performance of the Extended Kalman Filter as well as the trajectory tracking.

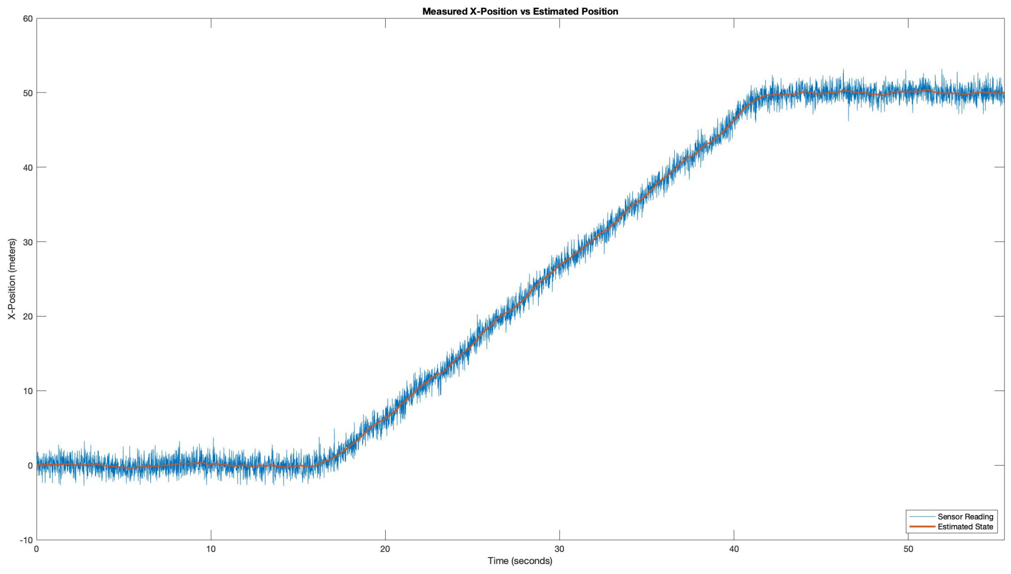

Figure 8: Measured vs Estimated X-Position

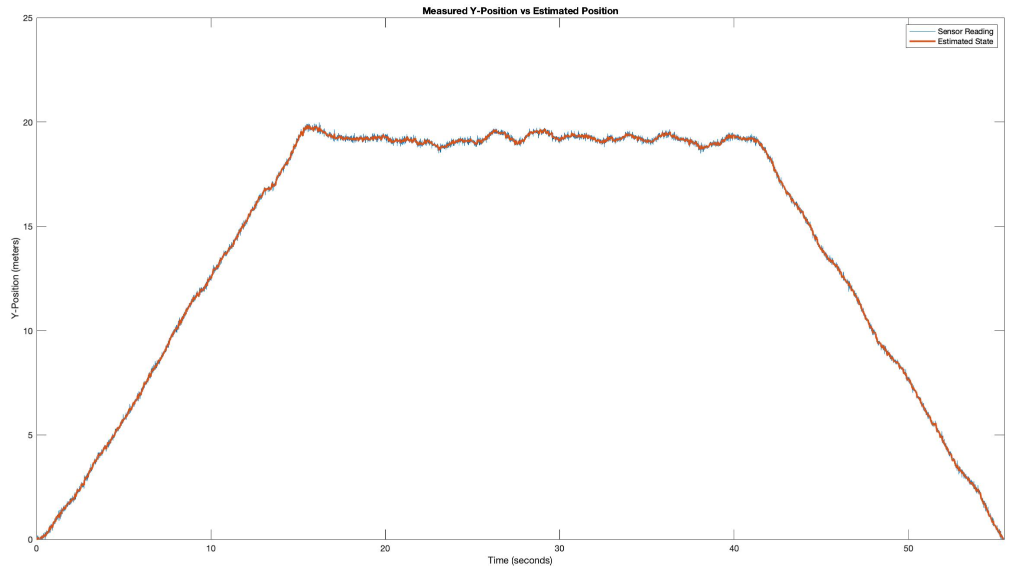

Figure 9: Measured vs Estimated Y-Position

As can be seen in the two plots, the EKF is doing a good job of filtering noise from the sensor readings, especially for x-position.

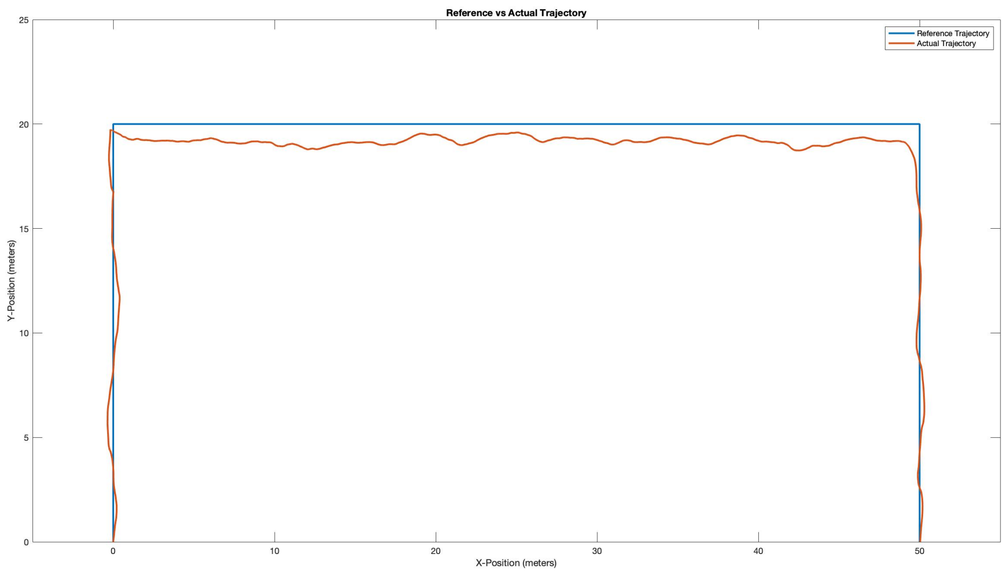

Figure 10: Reference vs Actual Trajectory

As can be seem from the above plot, the vehicle does a great job of tracking to within 35 centimeter on ascent and descent. This is great especially considering that the standard deviation of GPS readings is 1 meter. The main problem that can be seen is that the vehicle has about 5% steady-state error in the y-direction. This is somewhat expected, given that this controller did not include an integrator or integrated states.

5 Further Work

This project was a great learning experience for me: it was the first time I attempted trajectory tracking, first time I wrote an Extended Kalman Filter, and first time I used Simulink. I think that some natural next steps for me would be to learn dynamic programming, convex optimization, optimal controls, trajectory optimization and trajectory tracking. I would also like to experiment with some more exotic observers such as Unscented Kalman Filters and Particle Filters.

I would like to this general class of powered descent problems some time in the future. Perhaps I'll try to rebuild my VTOL vehicle and upgrade it with the capability to perform a bellyflop-propulsive landing similar to SpaceX's Starship, while conducting online trajectory optimization and utilizing some form of optimal controls to minimize battery power.

6 Appendix

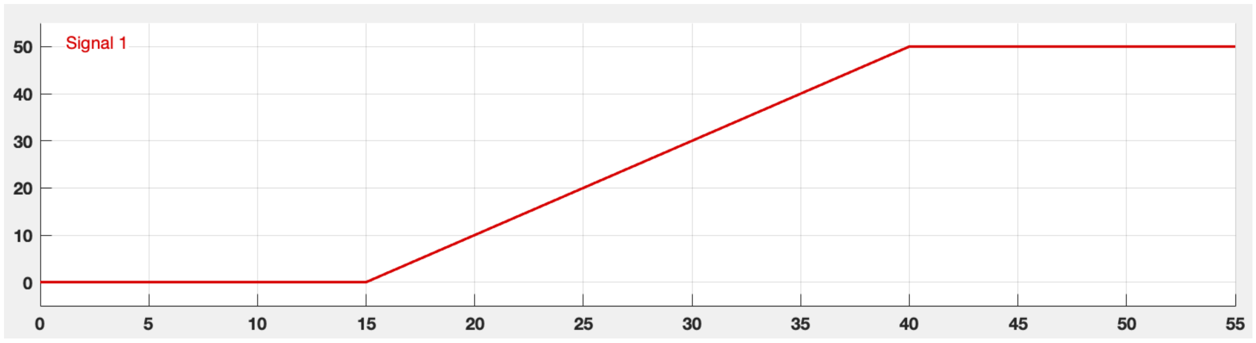

Figure 11: Reference X-Position

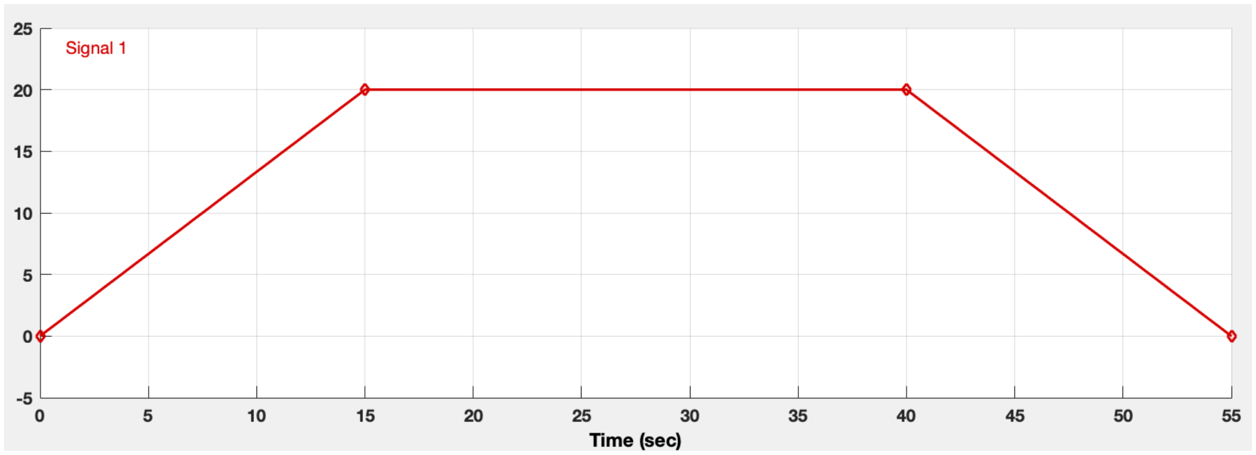

Figure 12: Reference Y-Position

EKF.m

```
function [xhat,P1] = EKF(x,P,u,y,dt,C)
     f=u(1);
     delta=u(2);
    m=1;
    g=9.8;
    L=0.2;
    I=0.002;

    %Process Noise Covariance
    Q=[0 0 0 0 0 0;
       0 0 0 0 0 0;
       0 0 0 0 0 0;
       0 0 0 0.0096 0 0.4802;
       0 0 0 0 1 0;
       0 0 0 0.4802 0 24.0100];

    %Sensor Noise Covariance
    R=diag([0.9 0.01 0.005]);

     if size(x)==1
        x0=zeros(6,1);
        P0=eye(6);
     else
        x0=x;
        P0=P;
     end

    %State Transition Matrix (Jacobian of Nonlinear State Transition)
    F=[1 0 0 dt 0 0;
       0 1 0 0 dt 0;
       0 0 1 0 0 dt;
       0 0 -dt*cos(delta+x0(3))*(g+f/m) 1 0 0;
       0 0 -dt*sin(delta+x0(3))*(f/m) 0 1 0;
       0 0 0 0 0 1];

    %Nonlinear State Propagation to get predicted xhat
    x1_p=[x0(4)*dt+x0(1);
          x0(5)*dt+x0(2);
          x0(6)*dt+x0(3);
          -dt*sin(delta+x0(3))*(g+f/m)+x0(4);
          dt*cos(delta+x0(3))*(f/m)+x0(5);
          -dt*sin(delta)*(m*g*L/I+f*L/I)+x0(6)];

    %Linearized Covariance Propagation to get predicted P
    P1_p=F*P0*F.'+Q;

    %Total Error
    S=C*P1_p*C.'+R;

    %Kalman Gain
    Kk=P1_p*C.'*S^-1;

    %Outputs
    xhat=x1_p+Kk*(y-C*x1_p);
    P1=(eye(6)-Kk*C)*P1_p;
```