Flappy Junco: Autonomous flight for a 2-dimensional fixed-wing model

Matthew Wallace & Tony Smoragiewicz

***Abstract*—Recent research in autonomous flight has focused on the control of multirotor-drones. These systems are easy to use, low cost, and posses high control actuation making them good candidates traditional robotic autonomy techniques. However, their design limits flight endurance as compared to traditional fixed-wing configurations. In this report, we explore applying robotics techniques to traditional fixed-wing design using a longitudinal, or in-plane, flight model. We developed a flight simulator that incorporates nonlinear flight dynamics, an inner-loop flight controller to stabilize this plant, a lidar sensor simulation to generate an occupancy grid, and a path and trajectory planner to autonomous pilot the simulation around obstacles through a Model Predictive approach.**

I. Introduction and Motivation

The following project was originally completed for EECE5550: Mobile Robotics at Northeastern University. We developed a 2-dimensional prototype of the Flight Software (FSW) for the Junco Remote Controlled (RC) aircraft. A link to our Github repository and code can be found at Github.

A. Software overview

Figure 1 shows the architecture for the model and simulation. A World Model simulates the dynamics of the vehicle and track a set of obstacles the vehicle must maneuver around. The left, high frequency portion includes noise generation, which will corrupt state measurements, a navigation block with an Extended Kalman Filter (EKF), and an Linear Quadratic Regulator (LQR) flight controller. The right, low frequency portion begins with simulating image inputs for the perception algorithm. The perception block produces an updating occupancy grid, which is used in the path planner to generate an initial path. The CVX TrajOpt block employs a raytracing procedure to determine a feasible region around the path then optimizes a trajectory inside this region, which is handed to the flight controller to track.

Our software was developed in Python. The world model and noise blocks exists in the "Plant.py" class and is demonstrated in the "PlantDemo.ipynb" notebook. The navigation, flight controller, and perception occurs in the "Drone.py" class and is demoed in the "DroneDemo.ipynb" notebook. The path planning and CVX TrajOpt is implemented in the "ConvexMotionPlanning.py" library which is demoed in the "TrajectoryGenerationDemo.ipynb" and "MPCDemo.ipynb" notebooks.

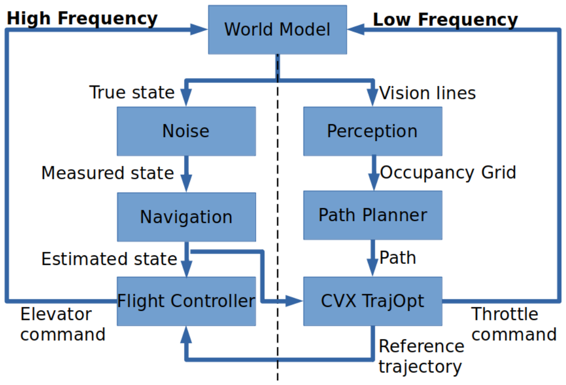

Fig. 1. Architecture for the simulation and modeling of the system.

B. World Model

1) Dynamics Model: We used a simple 6 state model of the flight dynamics, with states being the flight path angle, the pitch, the airspeed, the horizontal distance, and the vertical height $(x, z, V, \theta, \dot{\theta}, \gamma)$. For a complete treatment of flight dynamics, see [1].

The angle of attack is defined as:

$$\alpha = \theta - \gamma$$

The coefficients of lift, drag, and pitching moment are defined as:

$$\begin{aligned} C_L &= C_{L0} + C_{L\alpha}\alpha \\ C_D &= C_{D0} + K C_L^2 \\ C_{\mathcal{M}} &= C_{\mathcal{M}0} + C_{\mathcal{M}\alpha}\alpha + C_{\mathcal{M}\dot{\alpha}}\dot{\alpha} + C_{\mathcal{M}\delta_e}\delta_e \end{aligned}$$

The values for the above constants come from the 2-d wing profile used on Junco. The lift force, drag force, and pitching moment:

$$\begin{aligned} L &= \frac{1}{2}\rho v^2 S C_L \\ D &= \frac{1}{2}\rho v^2 S C_D \\ \mathcal{M} &= \frac{1}{2}\rho v^2 S c C_{\mathcal{M}} \end{aligned}$$

The underlying nonlinear time-invariant dynamical equations used for numerical integration:

$$\begin{aligned}
\dot{v} &= -\frac{D}{m} - g\sin\gamma + \frac{T}{m}\cos\alpha \\
\dot{\gamma} &= \frac{L}{mv} - \frac{1}{v}\cos\gamma - \frac{T}{mv}\sin\alpha \\
\dot{x} &= v\cos\gamma \\
\dot{z} &= -v\sin\gamma \\
\ddot{\theta} &= \frac{1}{I_{yy}}\mathcal{M}
\end{aligned}$$

The vehicle has two control inputs that can be applied at every time step. These are engine thrust (T) and elevator deflection (δ_e).

Our simulation environment is compatible with OpenAI's Gym environment. An example usage of the simulator can be seen in Algorithm 1. State returns a tuple of the state variables. Done returns True if the state limits have been exceeded. Finally, grid returns the occupancy grid where obstacles are represented by 1's and empty space by 0's. The systems uncontrolled step response is shown in Figure 2.

Algorithm 1 Using the simulation environment

```
0: env = Drone()
0: state = env.reset()
0: for i in iterations do
0:     action = [thrust[i], elevator[i]]
0:     state, done, grid = env.step(action) #High frequency included in step()
0:     if (i mod lf == 0) then #low frequency
0:         path = rrt(grid)
0:         [thrust[i:i+lf], elevator[i:i+lf]] = TrajOpt(grid, path, state)
0:     end if
0: end for=0
```

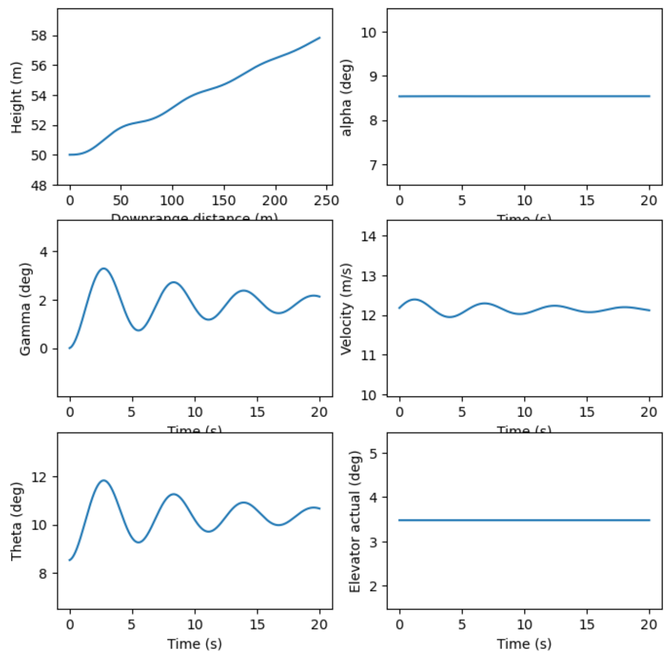
Fig. 2. Uncontrolled plant step response to 23% throttle increase.

II. High Frequency Loop

A. Noise

The noise is a Gaussian disturbance added to measurements of the flight angles and airspeed. The noise block generates measured airspeed, pitch angle, and position delta to be used in navigation. This functionality is captured in the plant model.

B. Navigation

Navigation is achieved through an Extended Kalman Filter (EKF) which estimates the true states based on pitch and airspeed indicator measurements. The estimated state is used both in the flight controller and trajectory generation. The performance of the EKF is shown if Figure 4.

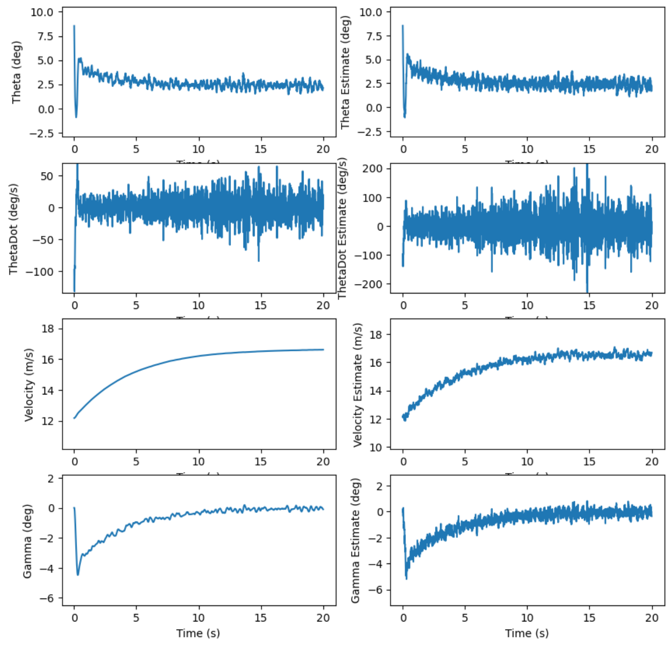
Fig. 3. EKF performance. Truth value is shown on left while the estimated value is given on the right.

C. Flight Controller

The systems flight controller is a Linear Quadratic Regulator (LQR). It's purpose is to regulate the flight path angle response of the system using elevator action. The throttle control is open-loop. We linearize around straight and level flight conditions. We used the continuous time LQR solver from the Python Control Systems Library to calculate the gains. The flight controller is responsible for tracking the desired pitch angle determined by the CVX TrajOpt. The Q matrix for state cost and R matrix for control cost are:

$$x_{ctrl} = \left[v, \theta, \dot{\theta}, \gamma\right]^T$$

$$Q = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10000 \end{bmatrix}$$

$$R = \left[1\right]$$

Note that the controller is mostly concerned with regulation flight path angle, which is the crucial state for obstacle avoidance. The controller significantly improves the step response of the system, which is shown in Figure 4.

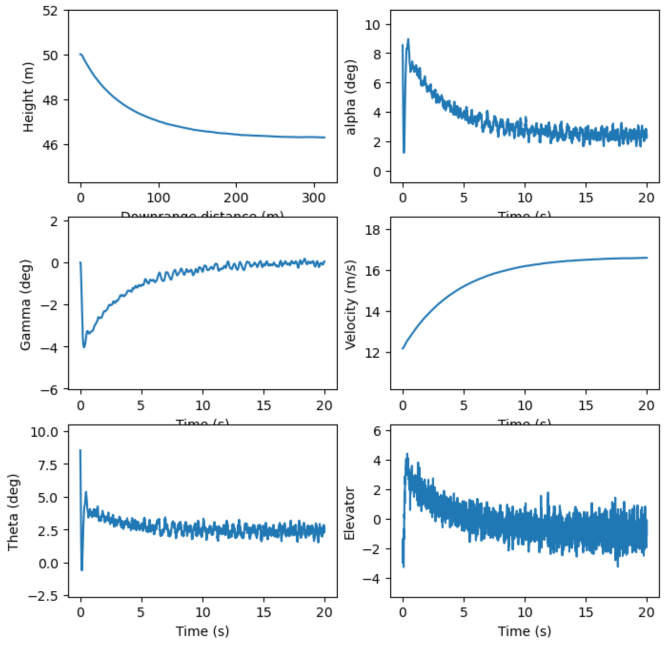

Fig. 4. Controlled plant step response to 23% throttle increase.

III. Low Frequency Loop

A. Perception

We modeled our perception system using an generated obstacle field and simulated lidar. The model identified objects at a particular angle and range with an error proportional to the range. These ray measurements are convert into an occupancy grid using Algorithm 2. This finds the (x, y) coordinates where the lidar rays intersect with the obstacles in the field of view. Figure 5 shows the performance of the system.

B. Path Planner

The Path Planner block employs an RRT to find a feasible path through the obstacle grid supplied by perception. The open-source library rrtplanner by Github user rland93 was used for this block. The RRT's output for the occupancy grid in Figure 5 is shown in Figure 6. The RRT was limited to searching in a narrow vertical corridor to increase the likelihood the resulting path was feasible dynamically.

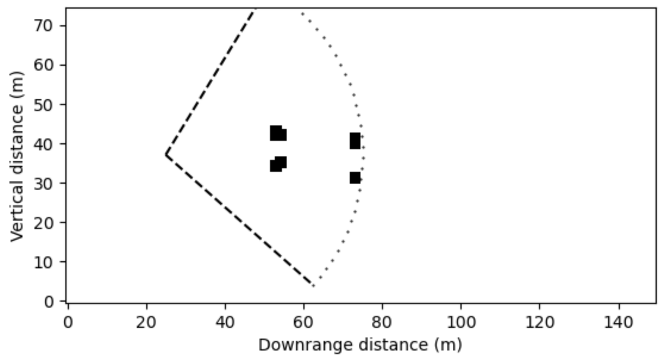

Fig. 5. The dashed lines encompass the visible area to the lidar sensor. The detected obstacles are shown in black.

Algorithm 2 Lidar-Obstacle intersection finding algorithm

```
0: $(x, y) \leftarrow state$
0: **for** i in lidar rays **do**
0:     **for** j in obstacles **do**
0:         $(obs_x, obs_y, radius) \leftarrow obstacle$
0:         convert coordinates to obstacle coordinate frame
0:         $dx \leftarrow obs_x - x$
0:         $dy \leftarrow obs_y - y$
0:         $dr \leftarrow \sqrt{dx^2 + dy^2}$
0:         $D = x * obs_y - y * obs_x$
0:         $\Delta \leftarrow radius^2 * dr^2 - D^2$
0:         **if** $\Delta \geq$ **then** {Intersection of line and circle}
0:             $intersect_x \leftarrow \frac{Dd_x \pm \text{sgn}(d_y) d_x \sqrt{\Delta}}{d_r^2}$
0:             $intersect_y \leftarrow \frac{-Dd_y \pm \text{sgn}(d_y) \sqrt{\Delta}}{d_r^2}$
0:             convert back to vehicle frame
0:             choose shortest distance of $\pm$ coordinates
0:         **else** {No ray intersection with obstacle}
0:         **end if**
0:     **end for**
0: **end for**=0
```

C. CVX TrajOpt

CVX TrajOpt block is a novel procedure to convert a path into a feasible trajectory for the vehicle to track. First ray tracing is performed around the path to determine a feasible region as outlined in [2]. The resulting QP is solves efficiently with the solver CVX [3], [4]. The resulting throttle commands are used open-loop while the desired pitch angle is tracked by the flight controller.

Beginning from, the path as shown in Figure 6, we find points along space along the feasible path and raytrace out from each point to find a feasible region. This region is encoded as a series of linear constraints on the optimization problem. Linear constraints from the system Jacobian captures the change in state over time. Figure 7 shows the procedure

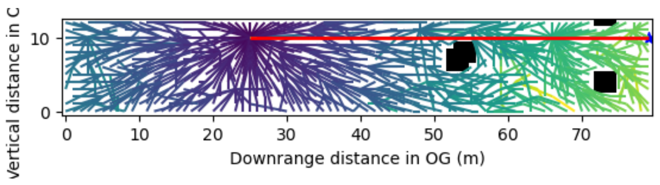

Fig. 6. RRT searching for a feasible path around an income obstacle.

in practice. The resulting matrices are:

$$\bar{G} = \begin{bmatrix} v_1^T & 0 & \dots & 0 \\ \vdots & & & \vdots \\ v_n^T & 0 & \dots & 0 \\ \vdots & & & \vdots \\ 0 & v_1^T & \dots & 0 \\ \vdots & & & \vdots \\ 0 & v_n^T & \dots & 0 \\ 0 & \dots & 0 & v_1^T \\ \vdots & & & \vdots \\ 0 & \dots & 0 & v_n^T \end{bmatrix}$$

$$\bar{h} = \begin{bmatrix} raytrace(x_{0,path}, v_1) \\ \vdots \\ raytrace(x_{0,path}, v_n) \\ \vdots \\ raytrace(x_{n,path}, v_1) \\ \vdots \\ raytrace(x_{n,path}, v_n) \end{bmatrix}$$

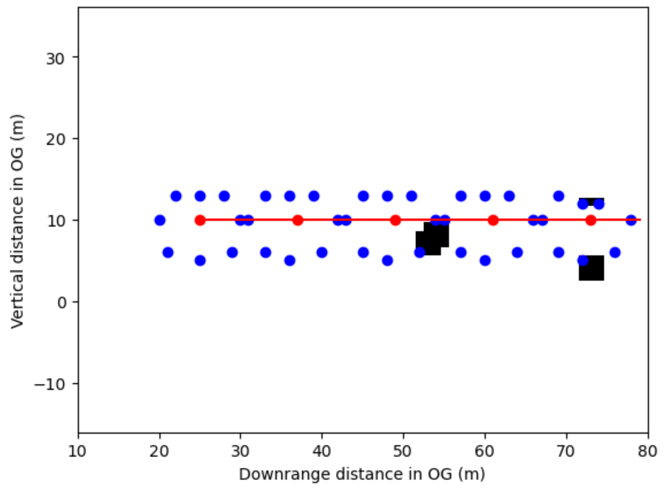

Fig. 7. Ray tracing around the path generated by the RRT.

Convex optimization requires linearized dynamics. The flight dynamics are linearized around straight and level flight and all obstacles are cast as constraints in this frame.

$$\bar{x} = \begin{bmatrix} x_0 \\ u_0 \\ \vdots \\ x_n \end{bmatrix}$$

$$\bar{A} = \begin{bmatrix} A & B & -I & 0 & \dots & 0 \\ \vdots & & & & & \vdots \\ 0 & \dots & 0 & A & B & -I \end{bmatrix}$$

$$\bar{b} = 0$$

Our cost function is a penalty on control effort along with a terminal state penalty for deviations from straight and level flight at a height of 50 m.

$$\bar{Q} = \begin{bmatrix} 0_{state} & 0 & 0 & \dots & & 0 \\ 0 & I_{ctrl} & 0 & \dots & & 0 \\ \vdots & & & & & \vdots \\ 0 & & \dots & 0 & I_{ctrl} & 0 \\ 0 & & \dots & 0 & 0 & I_{state} * W \end{bmatrix}$$

The full optimization problem is then:

$$min(\bar{x}^T Q \bar{x}) \quad S.T. \quad \bar{A}x = \bar{b} \,\&\, \bar{G}x <= \bar{h}$$

An additional constraint was added to capture the throttle lower limit and prevent the command from falling below zero. The final result for trajectory optimization is shown in Figure [**?**].

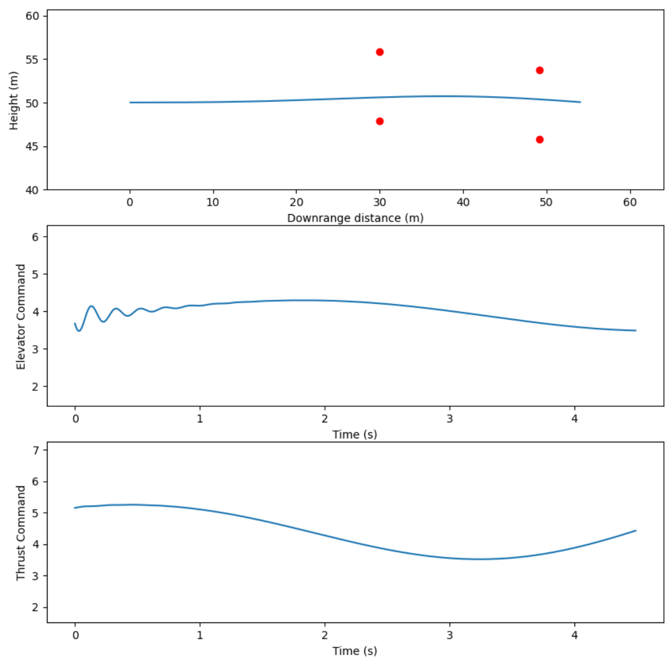

Fig. 8. Optimized tr.

IV. Results

Figure 9 shows an example of the scheme operating as Model Predictive Control. The vehicle has limited vision, so it can not see an obstacle for than 45 m ahead. The reference trajectory and open loop commands are calculate once per seconds, while the controller and phyics resolve at 100 Hz. This difference is notable in the throttle command, as it tends to change dramatically after this update.

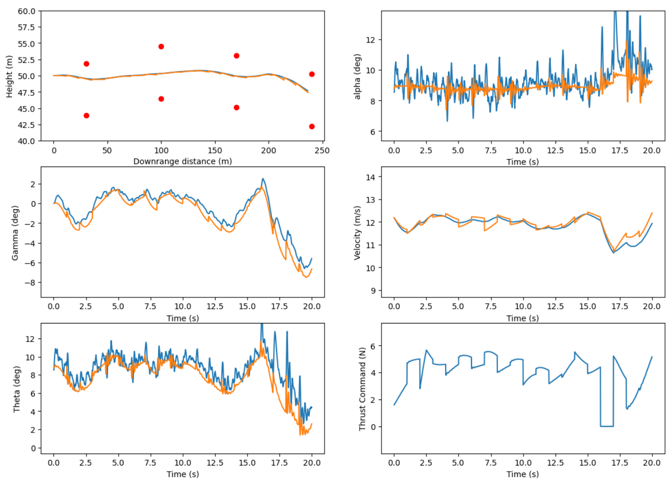

Fig. 9. Optimized tr.

The current limit on this scheme is the significant computational effort each iteration. This load is a split between RRT, ray tracing, and the convex optimization. In order to function in real-time, the computational load of these operations will need to be significantly reduced. Changing the RRT sampling strategy to remove unfeasible paths could greatly reduce the runtime.

References

[1] B. L. Stevens, F. L. Lewis, and E. N. Johnson, *Aircraft control and simulation: dynamics, controls design, and autonomous systems*. John Wiley & Sons, 2015.

[2] C. Meerpohl, M. Rick, and C. Büskens, “Free-space polygon creation based on occupancy grid maps for trajectory optimization methods,” *IFAC-PapersOnLine*, vol. 52, no. 8, pp. 368–374, 2019, 10th IFAC Symposium on Intelligent Autonomous Vehicles IAV 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896319304410

[3] M. Grant and S. Boyd, “CVX: Matlab software for disciplined convex programming, version 2.1,” http://cvxr.com/cvx, Mar. 2014.

[4] ——, “Graph implementations for nonsmooth convex programs,” in *Recent Advances in Learning and Control*, ser. Lecture Notes in Control and Information Sciences, V. Blondel, S. Boyd, and H. Kimura, Eds. Springer-Verlag Limited, 2008, pp. 95–110, http://stanford.edu/~boyd/graph_dcp.html.

V. Appendix

A. System Jacobians

The discrete nonlinear transformation is given by:

$$f(x_k,u_k) = \begin{bmatrix} v+\delta t(-\frac{D}{m} - gsin(\gamma) + \frac{T}{m}cos(\theta-\gamma)) \\ \theta + \delta t\dot{\theta} \\ \dot{\theta} + \frac{M}{I_{yy}} \\ \gamma + \delta t(\frac{L}{mv} - \frac{g}{v}cos(\gamma) + \frac{T}{mv}sin(\theta-\gamma)) \end{bmatrix}$$

Where the states are given by:

$$x_k = \begin{bmatrix} v \\ \theta \\ \dot{\theta} \\ \gamma \end{bmatrix} u_k = \begin{bmatrix} T \\ \delta_e \end{bmatrix}$$

And the aerodynamics in terms of states are:

$$\begin{aligned} L &= C_L qS = (C_{L0} + C_{L\alpha}(\theta-\gamma))qS \\ D &= C_D qS = (C_{D0} + KC_L^2)qS \\ &= (C_{D0} + K(C_{L0} + C_{L\alpha}(\theta-\gamma))^2)qS \\ M &= C_{\mathcal{M}} qSc = (C_{\mathcal{M}0} + C_{\mathcal{M}\alpha}\alpha + C_{\mathcal{M}\dot{\theta}}(\dot{\theta} - \dot{\gamma}) \\ &+ C_{\mathcal{M}\delta_e}\delta_e)qS \end{aligned}$$

The derivatives of drag are:

$$\begin{aligned} \frac{\partial D}{\partial v} &= \rho v S C_d \\ \frac{\partial D}{\partial \theta} &= 2KqS(C_{L\alpha}C_{L0} + C_{L\alpha}^2(\theta-\gamma)) \\ \frac{\partial D}{\partial \dot{\theta}} &= 0 \\ \frac{\partial D}{\partial \gamma} &= -2KqS(C_{L\alpha} + C_{L\alpha}^2(\theta-\gamma)) \end{aligned}$$

The derivatives of moment are

$$\begin{aligned} \frac{\partial M}{\partial v} &= \rho v S C_M \\ \frac{\partial M}{\partial \theta} &= qSC_{M\alpha} \\ \frac{\partial M}{\partial \dot{\theta}} &= 0 \\ \frac{\partial M}{\partial \gamma} &= -qSC_{M\alpha} \end{aligned}$$

The derivatives of lift are:

$$\begin{aligned} \frac{\partial L}{\partial v} &= \rho v S C_L \\ \frac{\partial L}{\partial \theta} &= qC_{L\alpha} \\ \frac{\partial L}{\partial \dot{\theta}} &= 0 \\ \frac{\partial L}{\partial \gamma} &= -qC_{L\alpha} \end{aligned}$$

The continuous system derivatives are:

$$\frac{\partial f_1}{\partial v} = -\frac{1}{m}\frac{\partial D}{\partial v}$$
$$\frac{\partial f_1}{\partial \theta} = (\frac{-1}{m}\frac{\partial D}{\partial \theta} - \frac{T}{m}sin(\theta - \gamma))$$
$$\frac{\partial f_1}{\partial \dot{\theta}} = 0$$
$$\frac{\partial f_1}{\partial \gamma} = (\frac{-1}{m}\frac{\partial D}{\partial \gamma} - gcos(\gamma) + \frac{T}{m}sin(\theta - \gamma))$$
$$\frac{\partial f_2}{\partial v} = 0$$
$$\frac{\partial f_2}{\partial \theta} = 0$$
$$\frac{\partial f_2}{\partial \dot{\theta}} = 1$$
$$\frac{\partial f_2}{\partial \gamma} = 0$$
$$\frac{\partial f_3}{\partial v} = \frac{1}{I_{yy}}\frac{\partial M}{\partial v}$$
$$\frac{\partial f_3}{\partial \theta} = \frac{1}{I_{yy}}\frac{\partial M}{\partial \theta}$$
$$\frac{\partial f_3}{\partial \dot{\theta}} = 0$$
$$\frac{\partial f_3}{\partial \gamma} = \frac{1}{I_{yy}}\frac{\partial M}{\partial \gamma}$$
$$\frac{\partial f_4}{\partial v} = \frac{1}{mv}\frac{\partial L}{\partial v}$$
$$\frac{\partial f_4}{\partial \theta} = \frac{1}{mv}\frac{\partial L}{\partial \theta} + \frac{T}{mv}cos(\theta - \gamma))$$
$$\frac{\partial f_4}{\partial \dot{\theta}} = 0$$
$$\frac{\partial f_4}{\partial \gamma} = (\frac{1}{mv}\frac{\partial L}{\partial \gamma} + \frac{g}{v}sin(\gamma) - \frac{T}{mv}cos(\theta - \gamma))$$

Our observability of the system is:

$$h(x_k) = \begin{bmatrix} v \\ \theta \end{bmatrix}$$

Therefor,

$$H_K = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$