Development of Classical and Modern Control Systems for Small Unmanned Fixed Wing Aircraft System

Brian Caskey*[2], Chris Chewning†[2], Christian Manahl‡[2], Gage Lochner§[1], Alex Wyatt¶[2], Youngro Lee‖[1], Victor Perez**[1], and HyeokJae Lee††[2]

[1]*Graduate Student, Department of Aerospace Engineering, Iowa State University*

[2]*Undergraduate Student, Department of Aerospace Engineering, Iowa State University*

This paper is written as a final report for the semester-long project in Aerospace Engineering 531 at Iowa State University in the Spring 2020 semester. The project involves understanding flight dynamics of aircraft to design a PID and LQR based controller, and then comparing their performance during in-flight testing. However, due to the COVID-19 outbreak, in flight testing was postponed and more comprehensive simulator work was done in order to ascertain the effectiveness of the controllers that were designed. This paper intends to address the complete process of controller design, verification, and a measure of effectiveness. The group has shown, via their work on these controllers, a fundamental understanding of how these controllers work, and how to design effective controllers.

*brcaskey@iastate.edu
†chewning@iastate.edu
‡cmanahl@iastate.edu
§glochner@iastate.edu
¶awyatt@iastate.edu
‖youngro@iastate.edu
**veperez@iastate.edu
††hyeokjae@iastate.edu

Contents

I. Nomenclature

C_{Lw}	=	Total Coefficient of Lift due to Wing
$C_{Lo}w$	=	Coefficient of Lift due to Wing at 0 angle of attack
C_{Law}	=	Coefficient of Lift due to angle of attack of wing
α	=	Angle of Attack
C_{Dw}	=	Total Coefficient of Drag due to Wing
C_{Dminw}	=	Minimum Coefficient of Drag due to Wing
K_w	=	Intermediate coefficient
E	=	Error
cg_t	=	Distance from center of gravity of aircraft to horizontal tail center of lift
cg_w	=	Distance from center of gravity of aircraft to center of lift of wing
V	=	Total velocity
AR_w	=	Aspect Ratio of wing
α_t	=	Angle of attack at horizontal tail
i_t	=	Horizontal tail incidence angle
T_e	=	Elevator effectiveness
d_e	=	Elevator deflection
q	=	pitch rate
C_{Lt}	=	Coefficient of lift due to horizontal tail
C_{Lat}	=	Coefficient of Lift due to horizontal tail angle of attack
C_{Dt}	=	Coefficient of Drag due to horizontal tail
C_{Dmint}	=	Minimum coefficient of drag due to horizontal tail
K_t	=	Intermediate coefficient for horizontal tail drag equation
C_{mf}	=	Coefficient of moment due to fuselage
C_{maf}	=	Coefficient of moment due to fuselage upon increasing angle of attack
C_{Dvt}	=	Coefficient of drag due to vertical tail
C_{Dminvt}	=	Minimum coefficient of drag due to vertical tail
C_{lp}	=	Coefficient of rolling moment
λ	=	Wing taper ratio
$\bar{q}$	=	Dynamic pressure
ρ	=	Air density
L_w	=	Total lift due to wing
S	=	Wing reference area
D_w	=	Total drag due to wing
M_w	=	Total pitching moment due to wing
c	=	Average chord length
C_{mw}	=	Coefficient of moment due to wing
L_t	=	Total lift due to horizontal tail
n_t	=	Tail area to wing area ratio
S_t	=	Horizontal tail area
D_t	=	Total drag due to horizontal tail
D_f	=	Total drag due to fuselage
C_{Df}	=	Coefficient of drag due to fuselage
M_f	=	Pitching moment due to fuselage
D_{vt}	=	Drag due to vertical tail
n_{vt}	=	Vertical tail area to wing area ratio
S_{vt}	=	Area of vertical tail
C_{Dvt}	=	Coefficient of drag due to vertical tail
L	=	Total Lift
D	=	Total Drag
Y	=	Total Side Force
q_b	=	Dynamic pressure
n_{vt}	=	Vertical tail area to wing area ratio

S_{vt}	=	Vertical tail area
C_{Lavt}	=	Vertical tail coefficient of lift
β	=	Sideslip angle
T_r	=	Rudder effectiveness
d_r	=	Rudder deflection
r	=	Yaw rate
m	=	Pitching Moment
n	=	Total Yawing moment
cgv_t	=	Distance from center of lift of vertical tail to center of gravity of aircraft
C_{nda}	=	Yawing coefficient due to aileron
d_a	=	Aileron deflection
l	=	Rolling Moment
b	=	Wingspan
C_{lp}	=	Coefficient of roll due to roll rate
p	=	Roll rate
C_{lb}	=	Coefficient of roll due to sideslip
C_{lr}	=	Coefficient of roll due to yaw rate
d_r	=	Rudder deflection
C_{lda}	=	Coefficient of roll due to aileron
	=	

II. Introduction

A. Problem Statement

Unmanned Aerospace Systems (UAS) have received attention from various fields including military and commercial applications such as attack[1], crop surveys, aerial photography, and search and rescue[2]. Though such systems present low cost with more safety and higher quality[3], applying such systems in the field present challenges with stability and control. Due to the absence of communication between UAS and pilot, automatic control is needed for the system to succeed [4], and the control of the UAS requires dynamics involving far more forces than their terrestrial counterparts including gravity and aerodynamic forces.[5] These difficulties pose issues in finding optimal control schemes, requiring research to find optimal control systems for the UAS. Until now, a 'one-size-fits-all' optimal control for every UAS does not exist and optimal control for the target system will not be identical to any other system, regardless of any shared similarities.

B. Vehicle description

The Rascal 110 EG ARF was selected as the subject model for this project because it is inexpensive, readily available, and a time-proven flight platform. Necessary physical property for developing the simulation environment will be described in section III.

Fig. 1 Rascal 110 EG ARF

C. Research Objectives

The goal of this project is to create an optimal control system for the Rascal 110 by comparing tuned PID and LQR controllers. The two PID controllers will be designed and tuned using classical control methods, while the LQR controllers will be designed using modern control methods. Due to the academic nature of the project, this is to be done through a "project-based-learning" framework offered by the Iowa State University Aerospace Department. To complete this task, the problem statement is broken into three parts: creating an accurate 6 DoF simulator in Matlab/Simulink, designing PID and LQR controllers to meet desired flight characteristics, and performing simulation testing to verify and compare the designs.

D. Literature Review

Unmanned Aerospace Systems (UAS) have proven their capabilities in several applications in the military and civil sectors. However, replacing the pilot with an automatic flight control system is not a simple task. To design a controller a dynamic mathematical model of the UAS is necessary, and the reliability of the controller is dependent on the level of detail applied to the numerical model. A nonlinear differential equation-based dynamic model leads to a stable simulation environment. Nevertheless, there will still be uncertainties present in the controller design; as a result, many different control methods have been studied, each with their advantages and disadvantages. In this report, a PID controller will be compared to an LQR controller. The classical PID controller is the most used controlled strategy in the industry nowadays; over 90 percent of industry controllers are based around its algorithm. Its ease of use, simplicity, and precise functionality is unmatched by any other controllers.[6] Yet, it's not an optimal control method, even though after tuning, it can achieve nearly optimal behavior. For such reason, the PID controller is compared to the modern LQR controller, which consists of an optimal control algorithm. The LQR control algorithm minimizes a specified cost function, which can vary depending on the controller design philosophy and system requirements. However, to achieve the optimal solution, the dynamic system must be linearized alongside other changes in the controller setup addressed in the following sections.

Pattern recognition control design: aircraft control design, including PID control, traditionally rely on the estimated mass property and effector performance of the vehicle to determine the desired force and moment for efficient control. However, knowing these properties is difficult during flying. [7] proposed Monte Carlo set through pattern recognition method to identify the relationship between the control effector commands and aircraft responses. Pattern recognition method utilize the database information of thruster firing commands and desired vehicle response instead of the mass and effector performance so that it does not rely on the estimation.

A Dynamic Performance Evaluation Technique for Unmanned Aerial Vehicles: The performance of the UAS depends on the dynamic modeling of the UAV and its control law. Transient response and steady-state response is mainly related to the response performance of the UAS. [8] proposed a technique based on the performance index and control energy of the closed-loop control system to analyze the response of a control system. This technique provides a robust criterion to judge the superiority and inferiority of control systems such as PID, LQR, robust linear, and robust nonlinear controllers.

E. Methodology

To complete the research objectives, the team broke the project into multiple parts. The first step in controller design is viewing the aircraft in person in a lab environment to gain a full understanding of the starting point for the project. To do this, the team travelled to the M2I Laboratory located in the basement of Howe Hall and inspected aircraft similar to the Rascal 110. After gaining valuable visual insight, the team spent the next portion of the semester creating an accurate 6 DoF simulator in Matlab/Simulink. This allowed the group to learn more about the mathematics and theory behind aircraft control, and further to design and test the controllers. Once this simulator was built and class instruction on control theory completed, full effort went into designing controllers to optimize the response of the UAS. Further details of these controllers and the effort put forth in designing them is included below. Once these controllers were completed, they were analysed and compared, the results of which are the conclusion of this paper.

III. Development of Simulation

A simulation environment was created in MATLAB/Simulink to test control systems created during the class project. This simulation was created by referencing a paper from the Air Force Institute of Technology* and takes advantage of Simulink's block diagram environment and MATLAB's iterative ability. The Simulink simulation model can be seen below in figure 2, and the code within the EOM block can be viewed at appendix B

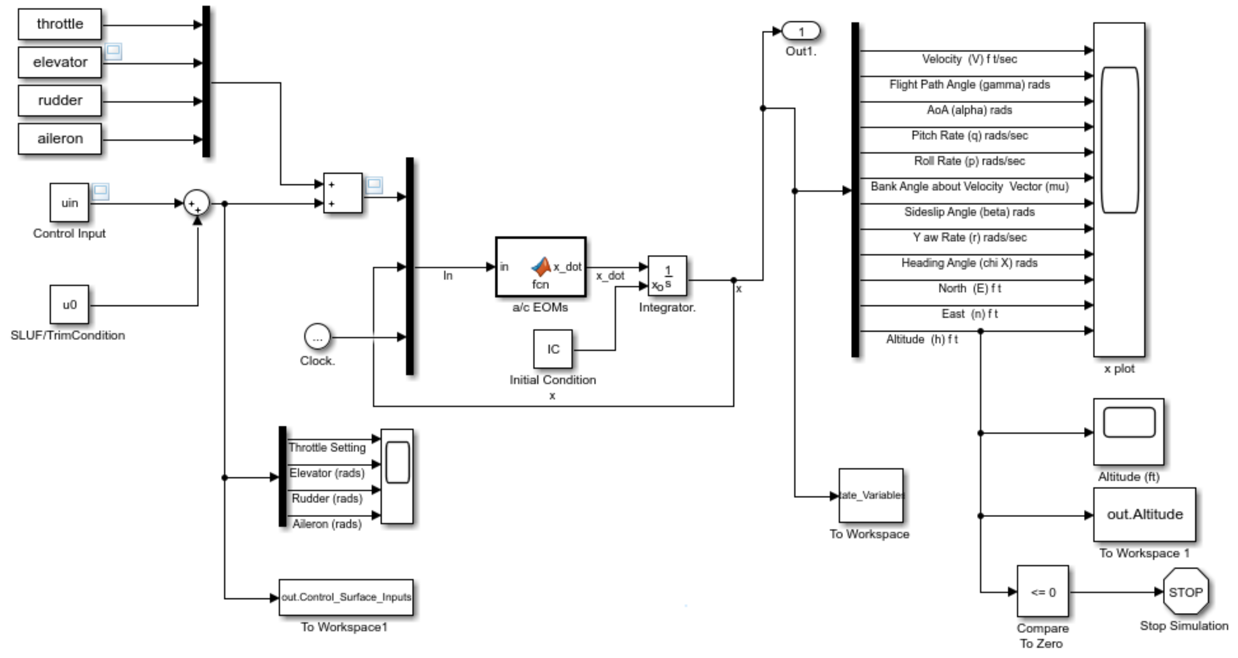

Fig. 2 Simulink Simulation Block

A. Equations of Motions Block: Forces and Moments Build up

To begin with lift, drag, and moment coefficients are built up from the important aerodynamic surfaces of the plane (wing, horizontal tail, vertical tail) and combined with common aerodynamic equations for force. These individual coefficients are summed to produce values for the overall aircraft in Eqns. (1) through (18).

$$C_{Lw} = C_{Lo}w + C_{Law}\alpha \tag{1}$$

$$C_{Dw} = C_{Dminw} + K_w {C_{Lw}}^2 \tag{2}$$

$$E = 2(C_{Low} + C_{Law}(\alpha - \dot{\alpha}\frac{(cg_t + cg_w)}{V})/(\pi AR_w) \tag{3}$$

$$\alpha_t = \alpha + i_t + T_e d_e + q\frac{cg_t}{V} - E \tag{4}$$

$$C_{Lt} = C_{Lat}\alpha_t \tag{5}$$

$$C_{Dt} = C_{Dmint} + K_t {C_{Lt}}^2 \tag{6}$$

$$C_{mf} = C_{maf}\alpha \tag{7}$$

$$C_{Dvt} = C_{Dminvt} \tag{8}$$

$$C_{lp} = -\frac{1}{12}C_{Law}\frac{(1+3\lambda)}{(1+\lambda)} \tag{9}$$

*https://doi.org/10.2514/6.2006-6261

$$\bar{q} = 0.5\rho V^2 \tag{10}$$
$$L_w = \bar{q}SC_{Lw} \tag{11}$$
$$D_w = \bar{q}SC_{Dw} \tag{12}$$
$$M_w = \bar{q}ScC_{mw} \tag{13}$$
$$L_t = n_t\bar{q}S_tC_{Lt} \tag{14}$$
$$D_t = n_t\bar{q}S_tC_{Dt} \tag{15}$$
$$D_f = \bar{q}SC_{Df} \tag{16}$$
$$M_f = \bar{q}ScC_{mf} \tag{17}$$
$$D_{vt} = n_{vt}\bar{q}S_{vt}C_{Dvt} \tag{18}$$

These yield the force and moment equations which are pertinent to the simulation in Eqns. (19) through (24).

$$L = L_w + L_t\cos(E - \frac{qcg_t}{V}) - (D_t + D_{vt})\sin(E - \frac{qcg_t}{V}) \tag{19}$$
$$D = D_w + (D_t + D_{vt})\cos(E - \frac{qcg_t}{V}) + L_t\sin(E - \frac{qcg_t}{V}) + D_f \tag{20}$$
$$Y = n_{vt}q_bS_{vt}C_{Lavt}(-\beta + T_rd_{rt} + \frac{rcg_t}{V}) \tag{21}$$
$$m = L_wcg_w\cos(\alpha) + D_wcg_w\sin(\alpha) + M_w - L_tcg_t\cos(\alpha - E + \frac{qcg_t}{V}) - (D_t + D_{vt})cg_tsin(\alpha - E + \frac{qcg_t}{V}) + M_f \tag{22}$$
$$n = -q_bn_{vt}S_{vt}C_{Lavt}(-\beta + T_rd_{rt} + \frac{rcgv_t}{V})cgv_t + (-q_bSbC_{nda}d_a) \tag{23}$$
$$l = \frac{q_bSb^2}{2V}(C_{lp}p + \frac{2V}{b}C_{lb}\beta + C_{lr}d_{rt} + \frac{C_{lda}d_a2V}{b}) \tag{24}$$

These force and moment equations are incomplete and rectilinear - the eleven, six DoF, coupled ordinary differential equations follow with Eqns (25) to (36).

$$\dot{V} = \frac{1}{m}(-D\cos(\beta) + Y\sin(\beta) + T\cos(\beta)\cos(\alpha)) - g\sin(\gamma) \tag{25}$$
$$\dot{\gamma} = \frac{1}{mV}(-D\sin(\beta)\sin(\mu) - Y\sin(\mu)\cos(\beta) + L\cos(\mu) + T(\cos(\mu)\sin(\alpha) + \sin(\mu)\sin(\beta)\cos(\alpha))) - \frac{g}{V}\cos(\gamma) \tag{26}$$
$$\dot{\alpha} = q - \tan(\beta)(p\cos(\alpha) + r\sin(\alpha)) - \frac{1}{mV\cos(\beta)}(L + T\sin(\alpha)) + \frac{g\cos(\gamma)\cos(\mu)}{(V\cos(\beta))} \tag{27}$$
$$\dot{p} = \frac{l}{I_{xx}} + \frac{m}{I_{xx}}(I_{yy}rq - I_{zz}qr) \tag{28}$$
$$\dot{q} = \frac{m}{I_{yy}} + \frac{m}{I_{yy}}(I_{zz}pr - I_{xx}rp) \tag{29}$$
$$\dot{r} = \frac{n}{I_{zz}} + \frac{1}{I_{zz}}(I_{xx}pq - I_{yy}pq) \tag{30}$$

$$\begin{aligned}\dot{\mu} = &\frac{p\cos(\alpha) + r\sin(\alpha)}{\cos(\beta)} - \frac{g\cos(\gamma)\cos(\mu)\tan(\beta)}{V} \\ &+\frac{1}{mV}\Big[D\sin(\beta)\cos(\mu)\tan(\gamma) + Y\tan(\gamma)\cos(\mu)\cos(\beta) + L(\tan(\beta) + \tan(\gamma)\sin(\mu)) \\ &+T(\sin(\alpha)\tan(\gamma)\sin(\mu) + \sin(\alpha)\tan(\beta) - \cos(\alpha)\tan(\gamma)\cos(\mu)\sin(\beta))\Big]\end{aligned} \tag{31}$$

$$\dot{\beta} = -r\cos(\alpha) + p\sin(\alpha) + \frac{1}{mV}(D\sin(\beta) + Y\cos(\beta) - T\sin(\beta)\cos(\alpha)) + \frac{g\cos\gamma\sin\mu}{V} \tag{32}$$

$$\dot{\chi} = \frac{1}{mV\cos\gamma}(D\sin(\beta)\cos(\mu) + Y\cos(\mu)\cos(\beta) + L\sin(\mu) + T(\sin(\mu)\sin(\alpha) - \cos(\mu)\sin(\beta)\cos(\alpha))) \tag{33}$$

$$\dot{\xi} = V\cos(\gamma)\cos(\chi) \tag{34}$$

$$\dot{\eta} = V\cos(\gamma)\sin(\chi) \tag{35}$$

$$\dot{h} = V\sin(\gamma) \tag{36}$$

Useful data of the Rascal 110 for developing the simulation environment, referenced by the Air Force Institute of Techonogy's paper, are noted in the table 1,2, and 3.

Coefficient	Symbol	Value
Wing Coefficient of Lift at 0°AoA	C_{L0w}	0.421
Wing Coefficient of Lift per AoA	$C_{L\alpha w}$	4.59
Wing Minimum coefficient of Drag	C_{Dminw}	0.011
Wing Moment Coefficient	C_{Mw}	-0.005
Vertical Tail Coefficient of Lift per AoA	$C_{L\alpha t}$	0.0969
Vertical Tail Minimum Coefficient of Drag	C_{Dmint}	0.001
Horizontal Tail Coefficient of Lift per AoA	C_{L_h}	0.76
Horizontal Tail Minimum Coefficient of Drag	C_{Dminh}	0.0002
Fuselage Moment Coefficient per AoA	$C_{M\alpha f}$	0.114
Fuselage Coefficient of Drag	C_{Df}	0.005

Table 1 Component Lift, Darg and Moment Coefficients

Longitudinal	Value/°	Lateral	Value/°	Lateral	Value/°
$C_{L\alpha}$	0.11	$C_{y\beta}$	-0.0056	C_{lr}	0.01
$C_{m\alpha}$	-0.006	$C_{l\beta}$	-0.0018	C_{nr}	-0.0006
C_{mq}	-0.233	$C_{n\beta}$	-0.00023	$C_{\delta a}$	0.244
$C_{m\delta}e$	0.011	C_{lp}	0.013	$C_{n\delta}a$	-0.0128

Table 2 Rascal 110 Stability Derivatives at Trimmed Steady Level Flight (Mach = 0.061)

B. Simulation Results

Those output values (Eqns 25 - 36) are then integrated in the Simulink model to obtain the aircraft's state variables, which can be plotted against time. The simulator was then tested against the reference paper using the dynamics of a SIG Rascall 110, the same aircraft used for their testing. The MATLAB code providing initial conditions, input values, and aircraft dynamics can be found in appendix C. The longitudinal dynamics were then tested by giving a doublet input to the elevator, and the response can be seen below in figure 3.

Variable (Symbol)	Value
Velocity (V_T)	64.8280 ft/sec
Roll Rate (P)	0 rad/sec
Pitch Rate (Q)	0 rad/sec
Yaw Rate (R)	0 rad/sec
AoA (α)	-0.0153 rad
Side slip angle (β)	0 rad
Bank angle (μ)	0 rad
Flight path angle (γ)	0 rad
Heading angle (χ)	0 rad
North Position (ξ)	0 ft
East Position (η)	0 ft
Altitude (h)	1119 ft
Throttle Trim (T)	1.29 ft/sec^2
Elevator Trim (δ_e)	-0.1174 rad
Rudder Trim (δ_r)	0 rad
Aileron Trim (δ_a)	0 rad

Table 3 Trim Condition

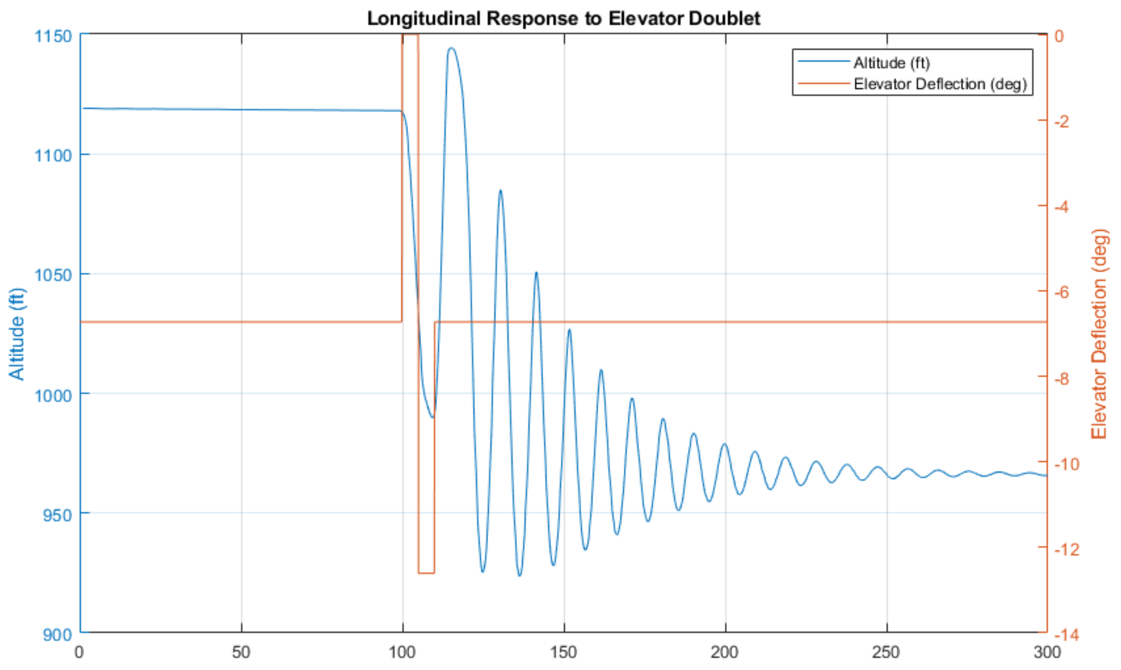

Fig. 3 Longitudinal Response to Elevator-doublet Input

The lateral response is tested next with a quick rudder deflection of 10 degrees, and can be seen in figure 4 with an induced bank angle and heading angle change. These responses match both the simulated and experimental tests run by the Air Force Institute of Technology, deeming this simulation to be in working order.

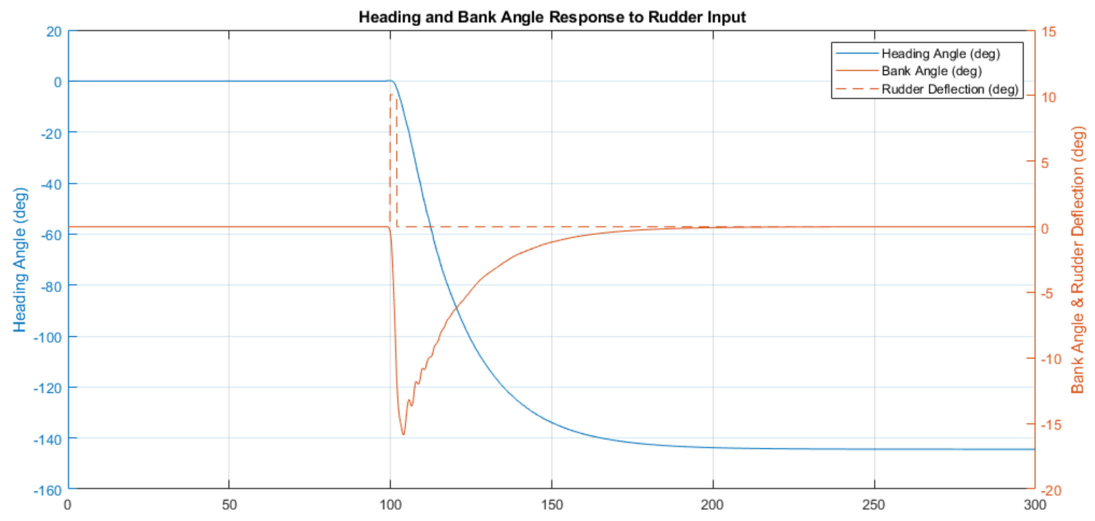

Fig. 4 Lateral Response to Rudder Input

IV. Classical Control Design

A. Proportional–Integral–Derivative controller

The proportional-integral-derivative feedback controller, or PID controller, is one of the most common controller designs in industry and aviation today because of its simplicity and adaptability. However, it is to be unfavorably compared to modern LQR control because it does not use state feedback and provide optimal mode control. PID control is generally tuned with the pole-placement method, the Ziegler-Nichols method, Cohen-Coon method, or heuristically by an experienced controls engineer.

In the PID control method, the control input $u(t)$ is is computed by taking the difference error between desired output $y(t)$ and reference input $y_r(t)$ as

$$u(t) = K_p(y(t) - y_r(t)) + K_i \int_0^t y(\tau) - y_r(\tau)d\tau + K_d \frac{d}{dt} \tag{37}$$

where K_p, K_i and K_d are the proportional, integral, and derivative gains, respectively

With PID control implementation, it is also important to watch the speed at which the controller turns the aircraft's actuators. A list of common aircraft servos was compiled [†] to get an idea for the max rotation speed an RC aircraft's servo could attain, and the target speed determined should be no more than 10 rad/s.

B. Longitudinal PID Design

The first step in the longitudinal PID controller design process was to close the loop for the dynamic model simulator. To control pitch of the aircraft commanding the pitch angle is desired, and to achieve that it must be fed back to the controller. However, the dynamic model did not output the Euler angles. To feedback the commands the pitch rate was integrated, and the PID block added to the model. The longitudinal dynamics model developed from [9] has a non-zero elevator trim which was must be accounted for and added to the control input. Additionally, servo dynamics were also simulated with a transfer function 10/(s+10) and added to the forward loop. For a 2 degree pitch angle command the following response was obtained, shown in Figure 7.

[†]`https://www.servocity.com/servos/hitec-servos`

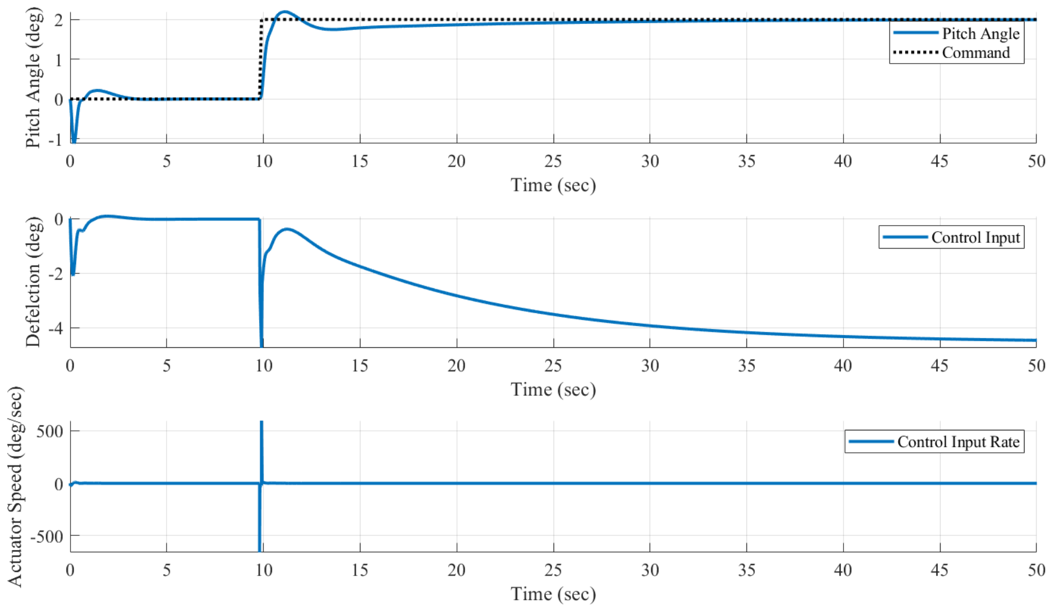

Fig. 5 Initial Longitudinal PID Response

From figure 5, the actuator speed was over 500 deg/sec when the pitch was commanded to 2 degrees, which is over the physical limit of servo specifications. Additionally for a zero command or no command at all, due to a faulty trim condition of the pitch angle (figure 6) the controller had to track back to 0. To account for both of these the a rate limiter was added to the controller output, reducing the actuator speed to a maximum of 5 rad/sec. The final Simulink model is shown in figure 7. The final tuned controller with the above mentioned modification, and altitude change can be seen in figure 8.

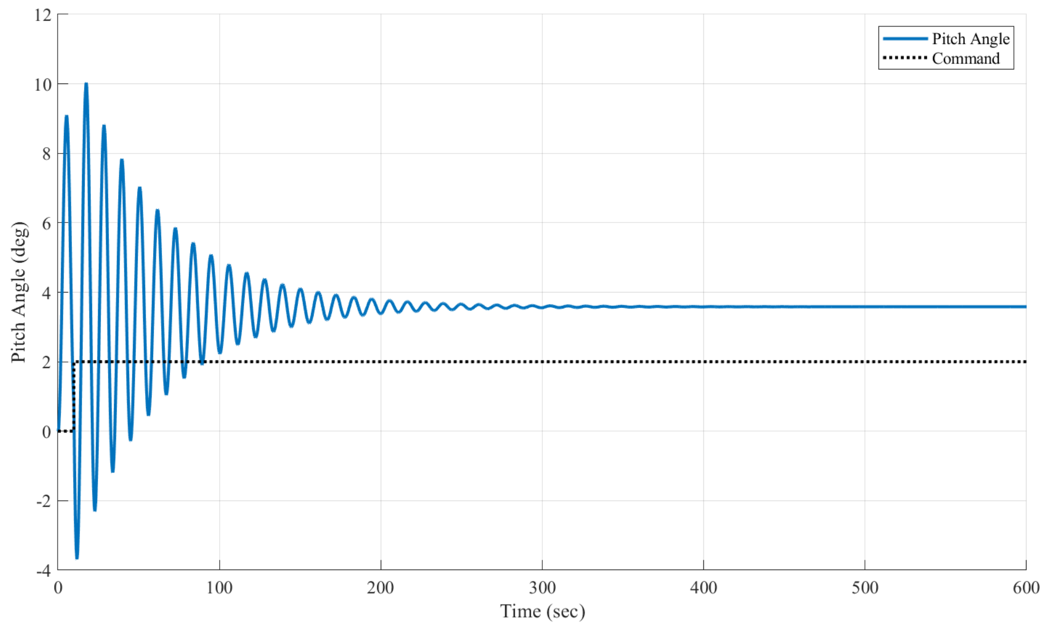

Fig. 6 Longitudinal Open Loop Response

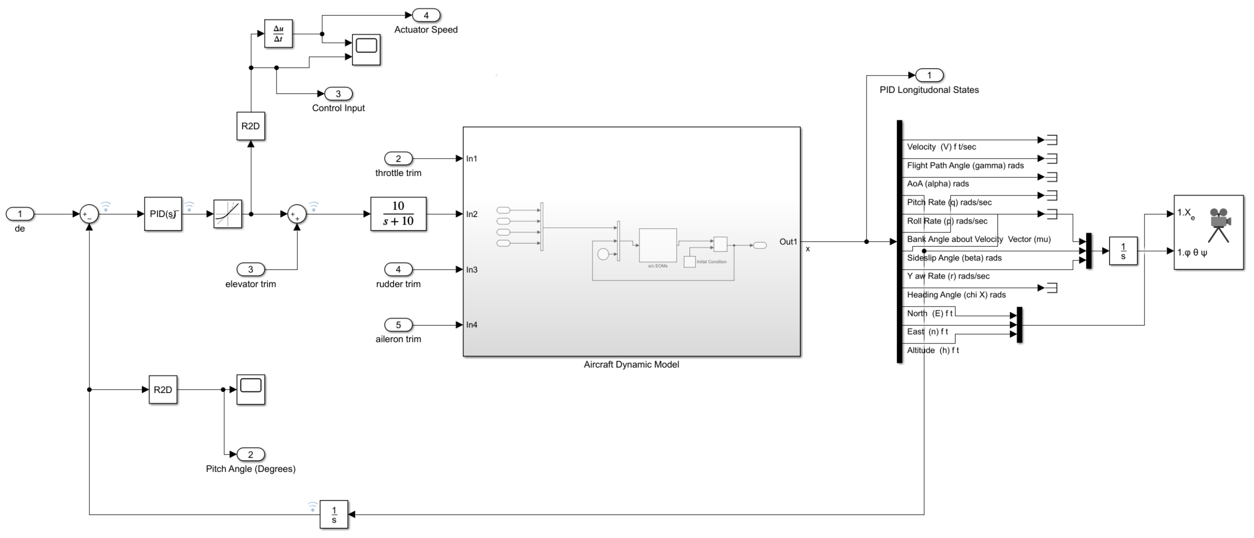

Fig. 7 Longitudinal PID Setup

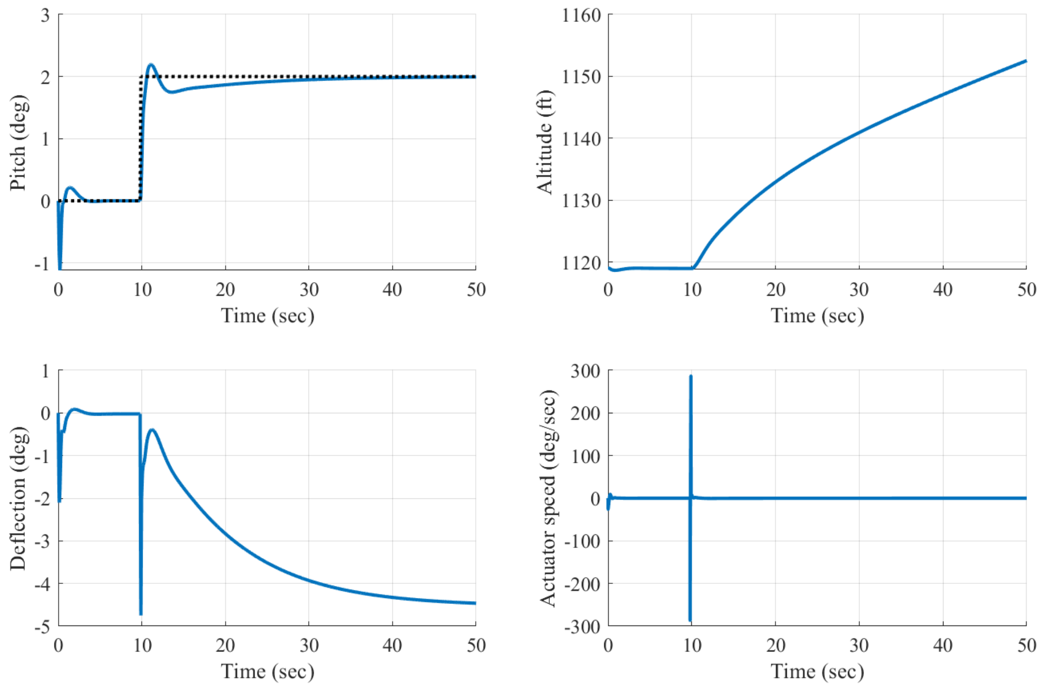

Fig. 8 Longitudinal PID Final Design

C. Lateral PID Design

The lateral PID design process began by implementing a PID block in Simulink around the simulated dynamic model already in place, as shown below in figure 9. The model feeds back an integrated roll rate value in order to control roll angle.

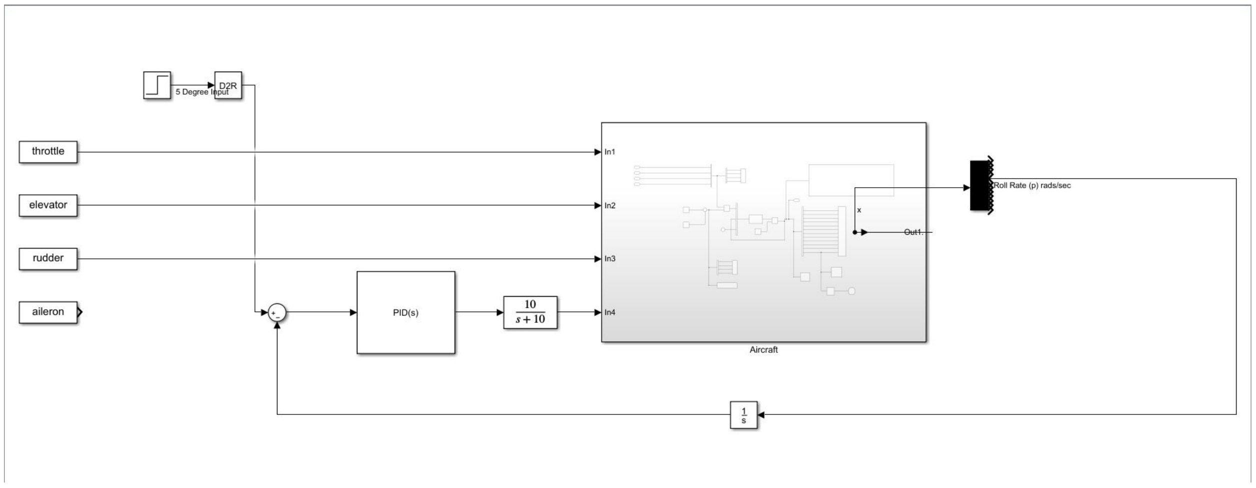

Fig. 9 Lateral PID Setup

The desired lateral response is a 5 degree roll angle. Before designing a PID controller for the lateral modes, a steady state response to the desired command is needed. Viewing the steady state response to a 5 degree roll angle command in figure 10, the Rascal is stable, but has highly oscillatory motion as well as an unachievable actuator speed. In addition to the highly oscillatory response, the roll angle output shows a 25% overshoot and a settling time of 4.4 seconds.

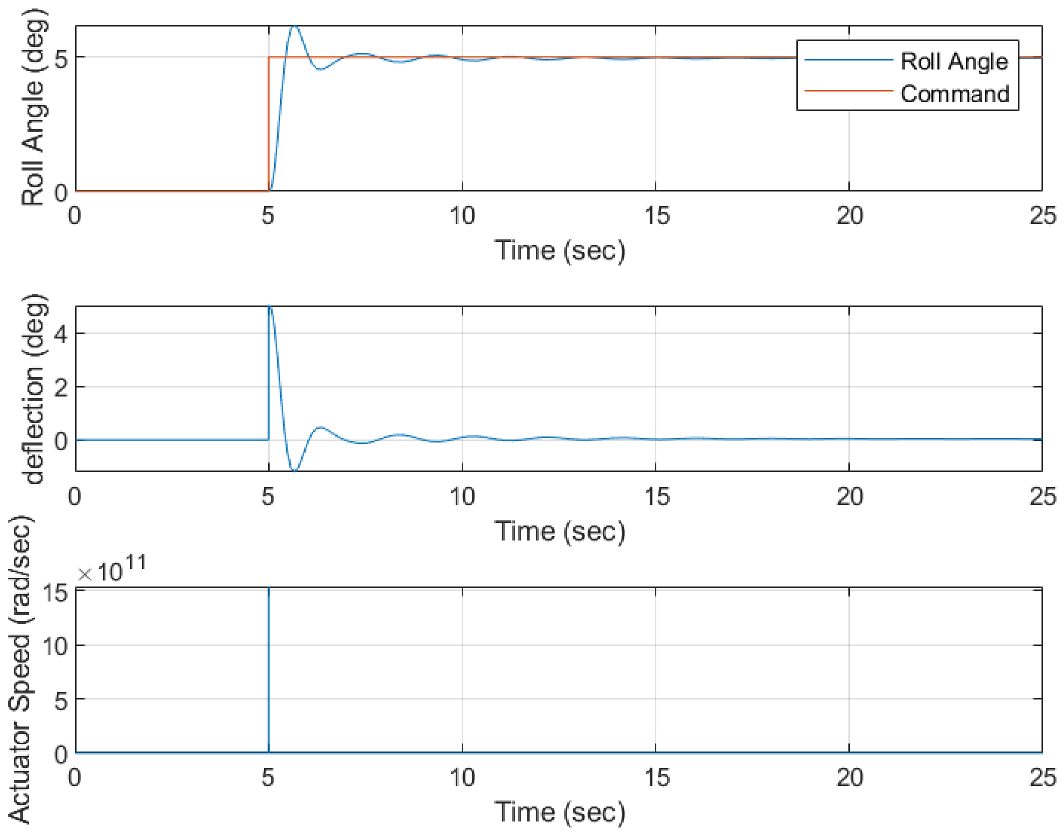

Fig. 10 Lateral Steady State Response

To improve upon the lateral response characteristics, a PID block was added to the Simulink model to control aileron input. The goal, in terms of quantitative improvement, was to reduce settling time while also reducing overshoot as much as possible (near zero overshoot). Another PID design goal was to reduce the oscillatory motion and force a critically damped response. Using the PID tuner in the Simulink PID block, the initial gains were $K_p = .695$, $K_i = .407$, and $K_d = .191$. The filter coefficient for this attempt was observed to be 457.75. As seen above in figure 11, these PID characteristics led to a better state response in terms of the goals defined above. Specifically, the roll angle overshoot value is reduced to 8.4 percent and the response is near fist order. However, the settling time for this controller response is 4.6 seconds which is slightly longer than the steady state response.

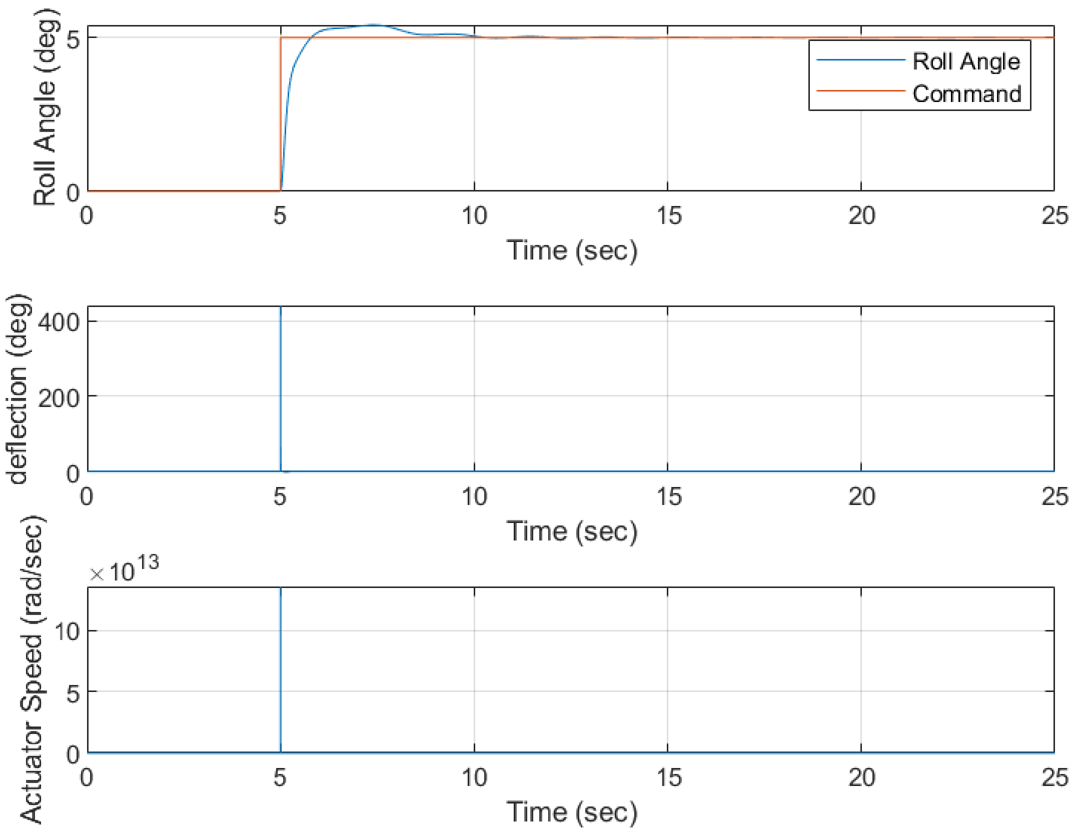

Fig. 11 State Response with PID

Although this controller led to a desirable state response, the PID tuning process did not consider the control speed of the input to the system, which needed to be 10 rad/sec. This value was approximated based on max angular rates for RC servos in the Ranger's application range. The control speed for the initial response proved to be in the 10^{13} order of magnitude, which is impossible for any actuator or servo to obtain. To fix this control speed limitation, a rate limiter block is needed after the PID block in the Simulink model. Using an upper limit of 10 rad/s (see section IV.A) in the rate limiter block, and subsequently re-tuning the PID gains to consider the control speed limit, the final gains were chosen to be 1.616, 1.221, and .357 for proportional, integral, and derivative gains respectively. The filter coefficient for this controller was viewed to be 1149.8. As viewed in figure 13, these PID gains led to a desirable roll angle response by reducing settling time by thirteen percent and forcing a near first order response. The overshoot is also reduced from 25 percent to 21 percent when compared to the steady state response. Overall, this PID controller met the design goals of reducing overshoot, settling time, and oscillatory motion. Overall, the PID controller allowed for faster rise times, better settling times, and is actually feasible for physical implementation. Compared to the uncontrolled response, the PID controller also would result in a less smooth initial response, and could result in a slight jitter of the aircraft when responding to large control changes due to the sharp rise time and damping ratio. As such, in the real world sharp control inputs are not likely to be avoided to ensure as stable of a UAS platform as possible, so this of less concern than it may appear from these simulated results.

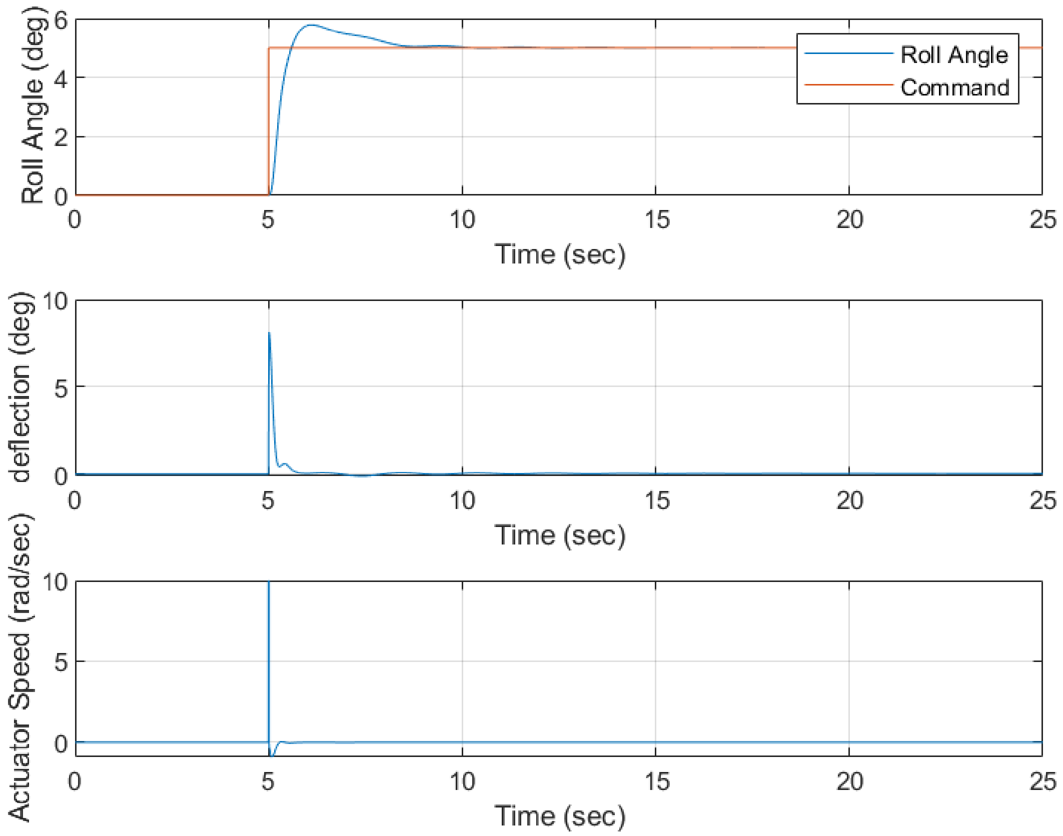

Fig. 12 State Response with PID Considering Control Speed

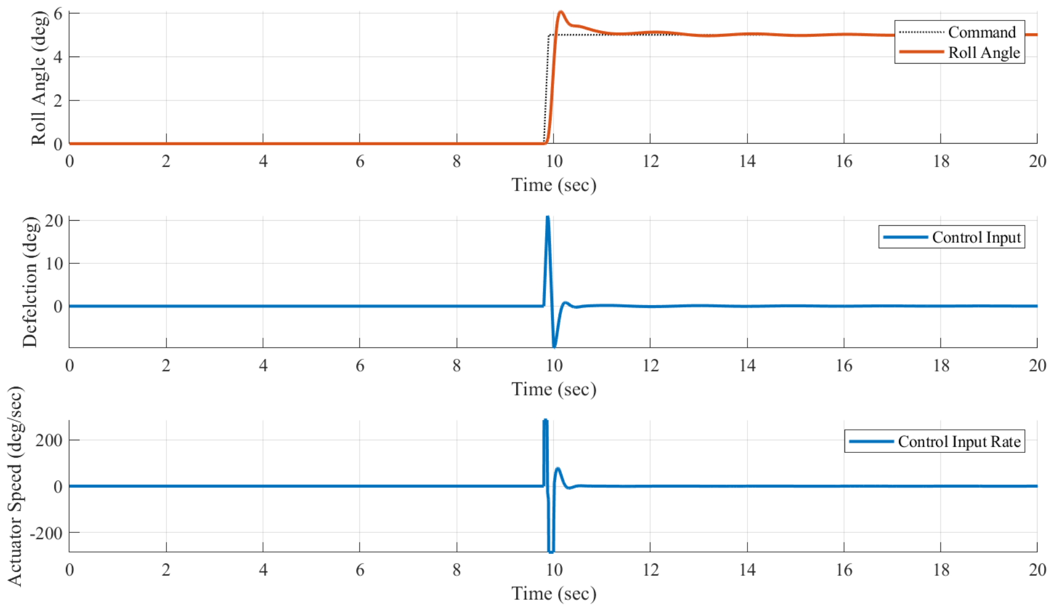

Fig. 13 State Response with PID Considering Control Speed

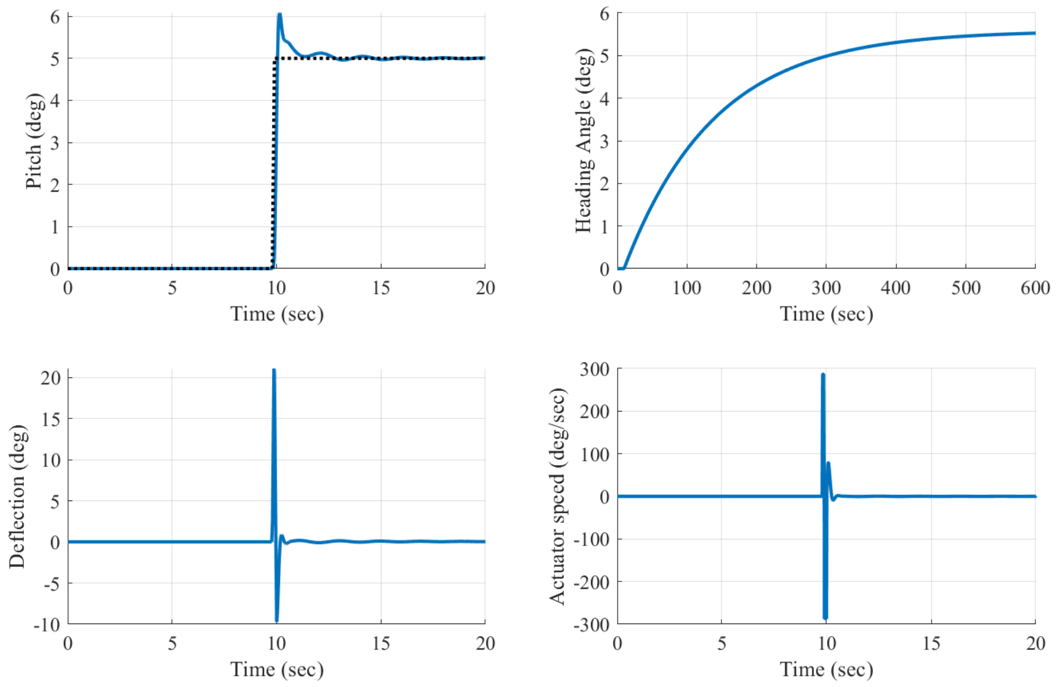

Fig. 14 Final State Response

V. Modern Control Design

A. Linear Quadratic Regulator

The objective of this section is to develop a controller based on one of the greatest works of modern control technique, which is known as the Linear Quadratic Regulator (LQR). Unlike the classical control techniques, such as lead/lag compensators and PID controllers, LQR is able to handle a Multi Input Multi Ouput (MIMO) system. This is because it is utilizing the state-space form of the system to represent the dynamic model. In addition, since LQR is based on optimal theory and a predefined performance index (or cost function), the controller knows what physical values need to be minimized. In the case of an UAS system, the goal of the controller is to reduce the final system states values to zero.

The derivation of LQR can be progressed briefly from now on. The general form of cost function can be defined as follows:

$$\begin{aligned} J =&\Psi(x(t_f)) + \int_{t_0}^{t_f} L(x(t), u(t))dt \\ &\textit{Subject to} \\ &\dot{x} = f(x(t), u(t), t) \\ &x(t_0) = x_0 \\ &\Psi(x(t_f)) : \textit{Terminal cost} \end{aligned} \tag{38}$$

Now, let's define co-state vector $\lambda(t)$, and then, build augmented function that involves co-state vector. Because $f(x(t), u(t), t) = \dot{x}$, the below equation still holds.

$$\bar{J} = \Psi(x(t_f)) + \int_{t_0}^{t_f} (L + \lambda(f - \dot{x}))dt \tag{39}$$

By the calculus of variation, if functional has maxima or mimina (collectively called extrema) at a certain point, then a small variation, or a derivative of the functional, should be zero. Note that this is not a sufficient condition, but a necessary one.

$$\delta\bar{J} = \Psi_x \delta x(t_f) + \int_{t_0}^{t_f} \left[L_x \delta_x + L_u \delta_u + \lambda^T f_x dx + \lambda^T f_u \delta_u - \lambda^T \delta\dot{x} \right] dt \tag{40}$$

Using integration by parts, the last term in the bracket can be solved as follows:

$$-\int_{t_0}^{t_f} \lambda^T \delta\dot{x} dt = -\lambda^T(t_f)\delta x(t_f) + \delta^T(t_0)\delta x(t_0) + \int_{t_0}^{t_f} \dot{\lambda}^T \delta x dt \tag{41}$$

Substitute the above equation into the original cost function.

$$\delta\bar{J} = \Psi_x \delta x(t_f) + \int_{t_0}^{t_f} \left[(L_u \delta_u + \lambda^T f_u)\delta u + (L_x + \lambda^T f_x \dot{\lambda}^T)\delta x \right] dt - \lambda^T(t_f)\delta x(t_f) + \delta^T(t_0)\delta x(t_0) \tag{42}$$

As stated above, δJ should be zero at a certain point that makes cost function extrema. Since control variation, δu, state variation, δx, and initial/final conditions are independent, all the terms bound by independent variables in the cost function variation should be zero independently. Therefore, the optimality condition is derived as follows:

$$\begin{aligned} &L_u + \lambda^T f_u = 0 \\ &L_x + \lambda^T f_x + \dot{\lambda}^T = 0 \\ &\Psi_x(x(t_f)) - \lambda^T(t_f) = 0 \end{aligned} \tag{43}$$

If a linear dynamical system is being controlled, and an appropriate cost function that represents the physical values of the UAS states exits, the optimal control can be easily calculated through the above derivation process. The general form of linear dynamics can be represented as follows:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) \end{aligned} \tag{44}$$

And then, a performance index that composed of quadratic of states and control can be adopted as the performance index. Q and R are weighting matrices that determines how much the corresponding physical value meed to be weighted. So Q and R weight on state error and control magnitude respectively. Mathematically, those matrices are positive semi-definite. The reason for this is that the quadratic form guarantees the simplicity of the solution after solving the optimality conditions.

$$J = \frac{1}{2}\int_{t_0}^{t_f} (x^T Qx + u^T Ru)dt \tag{45}$$

So that,

$$L = \frac{1}{2}x^T Qx + \frac{1}{2}u^T Qu \tag{46}$$

Now, the given dynamics equation and cost function can be used to apply into the derived optimality condition. And then, the following equations can be calculated.

$$\begin{aligned} L_x &= X^T Q \\ L_u &= u^T R \\ f_x &= A \\ f_u &= B \end{aligned} \tag{47}$$

Substitute the cost function (L) into the above equation:

$$\begin{aligned} &u^T R + \lambda^T B = 0 \\ &x^T Q + \lambda^T A + \dot{\lambda}^T \\ &\Psi_x(x(t_f) - \lambda^T(x) = 0 \end{aligned} \tag{48}$$

In the case of regulation problem, the terminal cost,$\Psi_x(x(t_f))$ will be zero. Rearrange the above equations:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ u &= -R^{-1}B^T\lambda \\ \dot{\lambda} &= -Qx - A^T\lambda \end{aligned} \tag{49}$$

Since the system is linear, a linear combination of state,$\lambda = Px$, can be a candidate of a solution of co-state without loss of generality. Inserting this term and linear dynamics into the $\dot{}$ equation, and then, the final form is:

$$\begin{aligned} &PA + A^T P + Q - PBR^{-1}B^T P = 0 \\ &u(t) = -R^{-1}B^T Px(t) \end{aligned} \tag{50}$$

The above equation is known as the matrix Riccati equation. By solving the Riccati equation, the optimal control solution can be obtained that satisfies minimizing the cost function.

B. Tracking Problem

As stated in the above, the purpose of regulation problem is to remove an initial condition error to zero. However, if the aircraft needs to be driven according to a certain objective, such as increasing altitude or holding a heading angle, the problem should be converted into a tracking problem. In order to track a reference signal, additional state is introduced in the state vector. By simply adding a cumulative error term in the state vector, the tracking problem can be resolved with LQR without any variation.

$$\begin{aligned} &\text{Heading control: } \int (\beta_{cmd} - \beta)dt \\ &\text{Pitch control: } \int (\theta_{cmd} - \theta)dt \end{aligned} \tag{51}$$

In the case of the given problem, the aircraft is supposed to track the desired angle or path throughout the simulation. Therefore, an infinite horizon regulator problem will be addressed. If final time is specified, the state variables will reach into the terminal state at a given final time.

$$J = \frac{1}{2}\int_0^{\infty} (x^T Qx + u^T Ru)dt \tag{52}$$

C. LQR Implementation

Based on what has been developed and derived, the LQR controller for controlling the Rascal 110 is constructed under MATLAB/Simulink environment. To take advantage of LQR control, a lateral and longitudinal state-space model of the Rascal 110 must be generated. Using physical measurement and DATCOM data from the paper by the Air Force Institute of Technology and a series of MATLAB computations found in appendix D, matrices were generated to design the LQR controllers around.

D. Longitudinal LQR Design Process

The goal of longitudinal LQR design is to manipulate pitch angle from 0 to 2 degrees, resulting in increasing altitude of the aircraft. First, since LQR is based on linear dynamic system, corresponding matrices should be defined as follows:

$$A_{long} = \begin{array}{c} \begin{array}{ccccc} \theta & V_T & \alpha & q & \theta_{error} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1.0000 & 0 \\ -32.2000 & -48.0000 & 13.1983 & 3.0000 & 0 \\ 0.0001 & -0.0253 & -0.2370 & 1.0000 & 0 \\ 0 & -0.0549 & -5.5837 & -3.7947 & 0 \\ -1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$B_{long} = \begin{array}{c} \delta_e \\ \begin{pmatrix} 0 \\ 0 \\ 0.9000 \\ 2.9035 \\ 0 \end{pmatrix} \end{array}$$

Note that because the linear dynamics used in LQR is described with respect to trim condition, the A and B matrices represents the perturbed dynamics. Therefore, the trim control surfaces values should be applied to the 6DoF nonlinear dynamics as well as the generated elevator, and state vector inserting into control calculating block should consider the trim conditions. As can be seen in the figure 15, trim condition, x_0, was subtracted from the output state vector. "Aircraft Dynamics" block contained the 6 degrees of freedom nonlinear dynamics. If control surface inputs insert into the dynamics block, then corresponding physical values are coming from the dynamics block. To track the reference input, which would be the pitch angle command, cumulative pitch angle error was being calculated at each step.

To facilitate the developing the realistic simulation environment, the servo motor dynamics was reflected in the Simulink setup. However, adding the actuator block caused a chattering phenomenon, so the "Low Pass Filter" block was included to resolve the chattering problem.

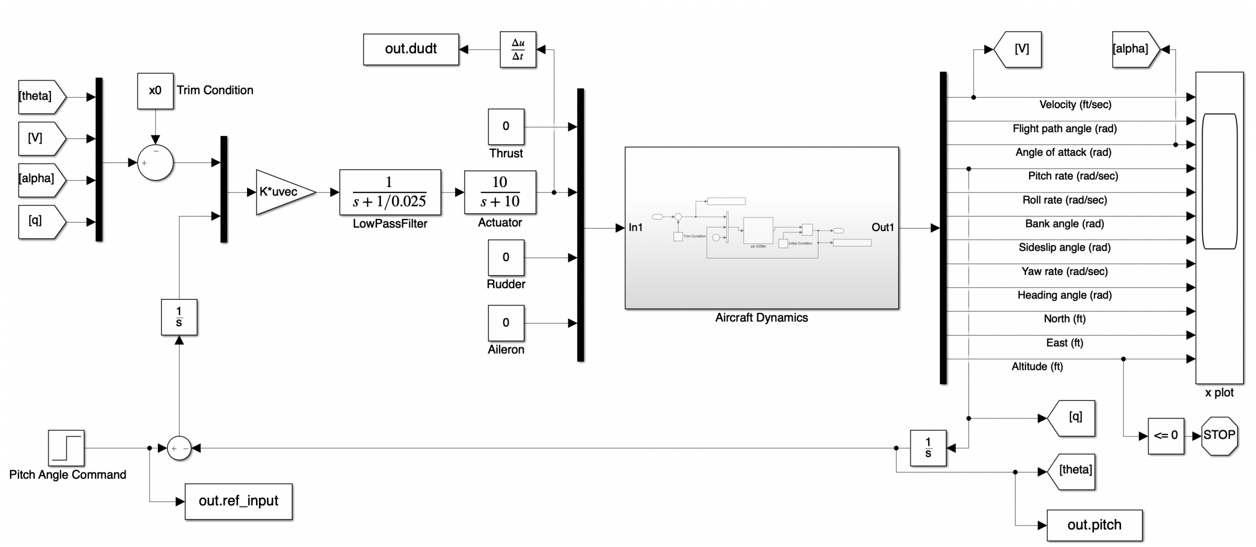

Fig. 15 Longitudinal LQR Setup

As indicated in the precious subsection, Q and R are weighting matrices that weight on the state error and control sum, respectively. The overall performance of the controller is dependent on the weighting matrices, and consequently, the determination of those matrices values can be determined based on the design criteria. The suitable values were determined through repeated simulations.

$$Q = \begin{array}{c} \\ \\ \\ \\ \\ \end{array}\begin{array}{ccccc} \theta & V_T & \alpha & q & \theta_{error} \end{array}$$

$$Q = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0 & 1000 \end{pmatrix}, \quad R = 0.1$$

The pitch angle tracking with longitudinal LQR design process worked well (figure 16). The settling time, which is required time for an output to reach and remain within a certain error bound following reference input, was about 10 seconds, and it resulted in about 30 feet altitude increasing. Since LQR is based on the optimal control theory, no overshoot was detected, which is positive consequence of using LQR. Usually the elevator is only control surface that needs to be operated for the pitch angle control. Therefore, the left-bottom plot shows the deflection angle of the elevator. In terms of servo motor control speed limit, the deflection angle change rate should be monitored. Right-bottom plot shows the actuator speed in the unit of degree per second. The range values shown are within the normal operation range.

As mentioned in the previous subsection, R matrix contributes to the control magnitude. Therefore, it can be deduced that a higher R value will impose more weight on the control magnitude, and this will result in a smaller amount of total control. In other words, smaller amount of control leads to slower response, and the settling time will cost more. Figure 17 verified this reasonable deduction. The yellow line has lowest value of R, resulting in longest settling time (35 seconds). In addition, cumulative angle deflection values for each case are 31.0965, 27.6916, and 26.5844 in the sequence of increasing R value. This agrees on the fact as well.

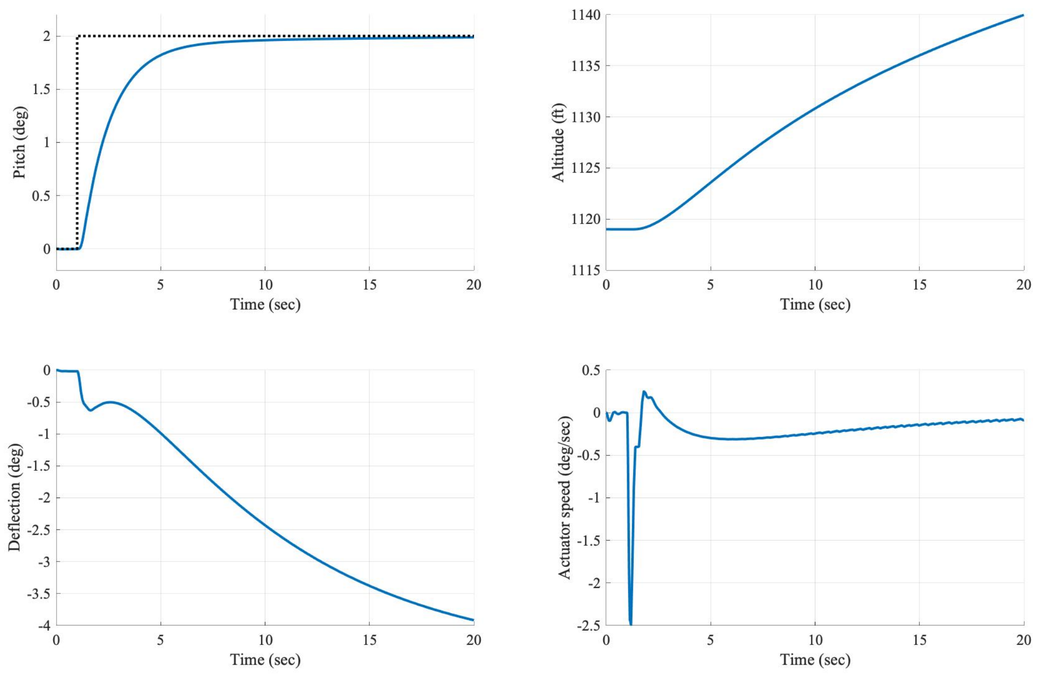

Fig. 16 Longitudinal LQR Control Result

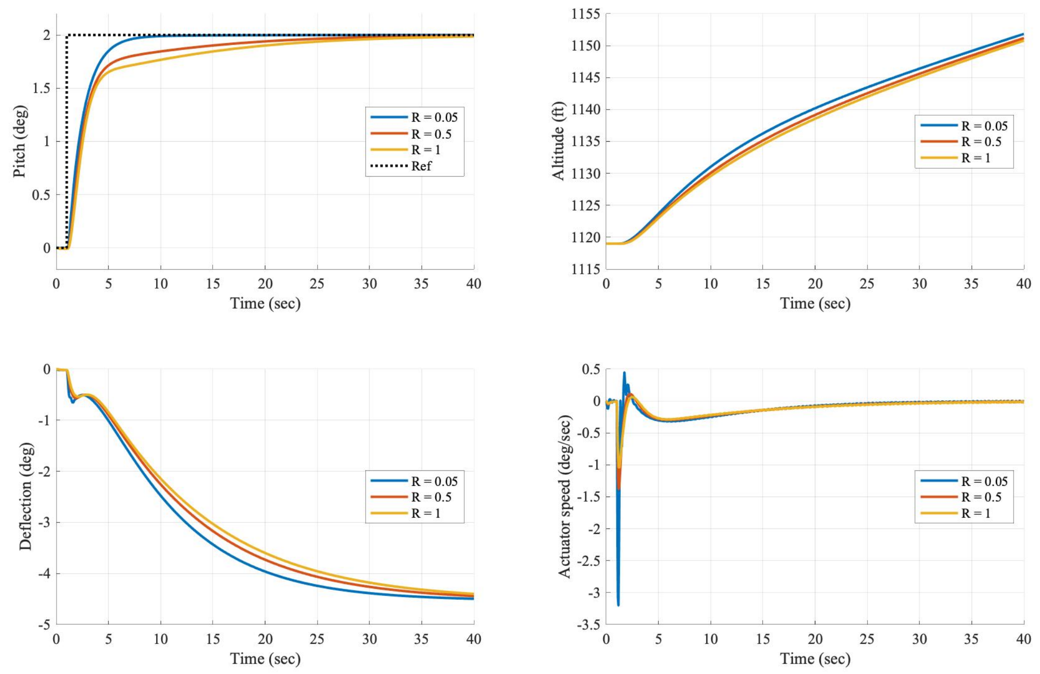

Fig. 17 Longitudinal LQR Control Results depending on R gain

E. Lateral LQR Design Process

The project objective statement for a lateral LQR design was to hold a steady roll angle of 5 degrees. First, an LQR controller was wrapped around the Simulink block, similar to the PID control, as seen below in figure 18.

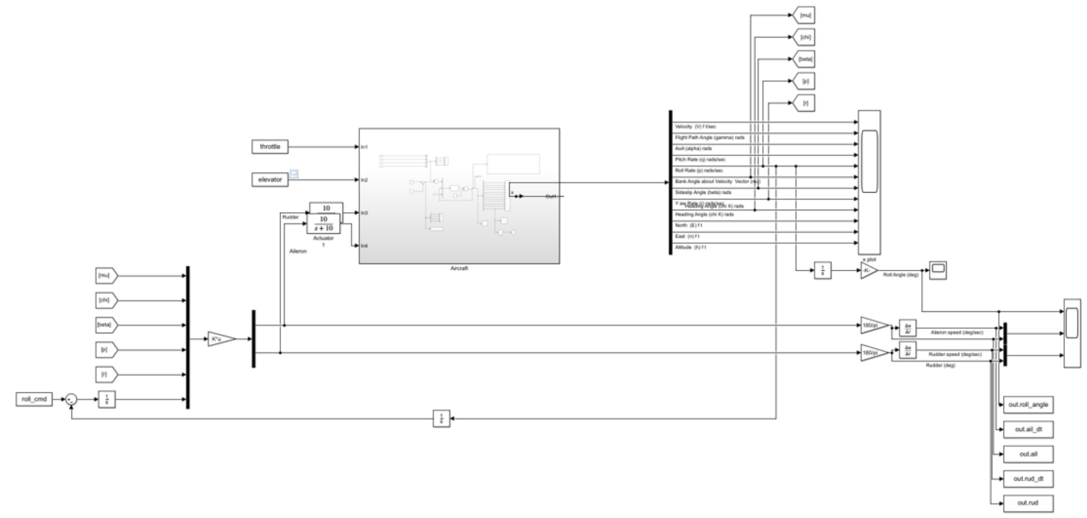

Fig. 18 Lateral LQR Setup

The state-space model was then generated using the attached MATLAB code and found as seen below. Note the extra state, Φ_{error} , used to feed the roll angle into the state matrix.

$$A_{lat} = \begin{array}{c} \begin{array}{cccccc} \Phi & \Psi & B & p & r & \Phi_{error} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1.0000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.0000 & 0 \\ 0.4967 & 0 & -0.0002 & -0.0130 & -0.9593 & 0 \\ 0 & 0 & -0.0076 & 0.0035 & 0.0030 & 0 \\ 0 & 0 & 0.0010 & -0.6571 & -0.0002 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$B_{lat} = \begin{array}{c} \begin{array}{cc} \delta_a & \delta_r \end{array} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0.3726 \\ 1.0256 & 4.6083 \\ -0.0541 & -28.1956 \\ 0 & 0 \end{pmatrix} \end{array}$$

With the state-space model generated, the gain then needed to be solved to transform the state to a command in the LQR loop. This was done using MATLAB's `lqr` command [‡]. This command outputs the 2x5 matrix used as the gain which is matrix-multiplied against the lateral state being fed-back. This multiplication of matrices then yields the aileron and rudder commands to be fed into the simulation.

[‡]`https://www.mathworks.com/help/control/ref/lqr.html`

Next, the Q and R matrices must be set for the K matrix to be found. For an initial test, the Q matrix is set to a 5x5 identity matrix and the R matrix is set to a values relating to the max deflection of the control surfaces.

$$R = \begin{bmatrix} \frac{1}{\delta_{r,max}{}^2} & 0 \\ 0 & \frac{1}{\delta_{a,max}{}^2} \end{bmatrix}$$

The K matrix was then generated by the command `k = lqr(A, B, Q, R)`, and the Simulink block was run for 60 seconds, with a roll command being give at 10 seconds. An error occurred, saying the calculations stopped converging after 26 seconds. To troubleshoot, the model was run for 26 seconds up until the calculations stopped converging and the result is plotted below:

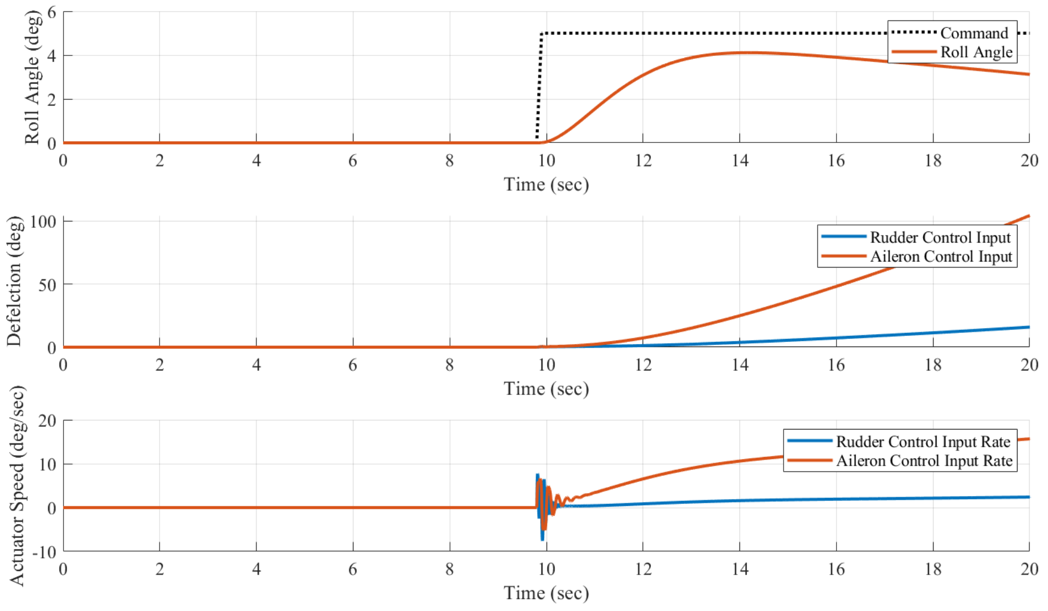

Fig. 19 Initial LQR Test

Quick reasoning found that because throttle and elevator are not being controlled, the aircraft could not physically hold a 5 degree roll and hold the heading angle steady. With an objective of only roll angle, the heading angle control could be effectively eliminated by giving the state a near-zero value in the Q-matrix.

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0.001 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The LQR controller is then run again with a much more desirable output, as seen below in figure 20.

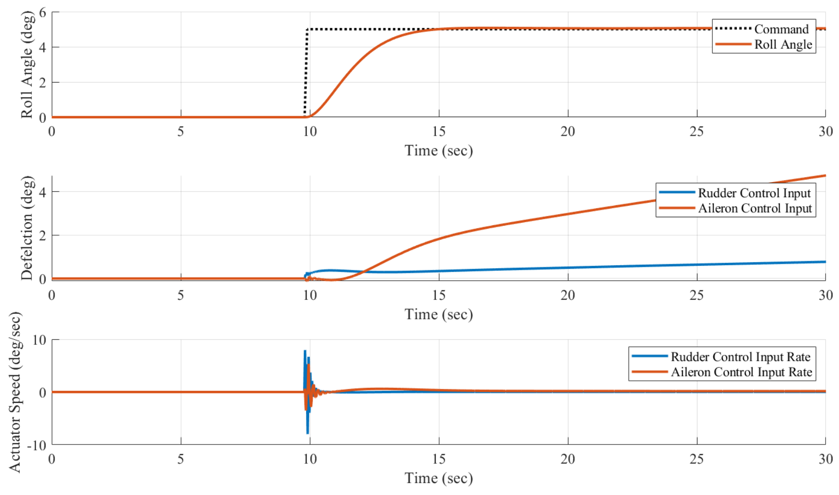

Fig. 20 Lateral LQR with no heading control

Noticing there is still quite a bit of chatter in the aileron def election after the state is commanded, a low-pass filter is implemented to remove much of this noise. The low-pass filter is then tuned to attenuate the small high-frequency noise from the controller output, which is most likely from inaccuracies in the state-space model. A preferred approach to removing the chatter would be to better model the aircraft's A and B matrices, however with limited time and information this was not accomplished in this project, and a low-pass filter sufficiently smoothed out the noise. The effects of the low-pass filter can be seen in figure 21, where the same Q and R matrices are used as in figure 20 but with the filter on both the aileron and rudder output.

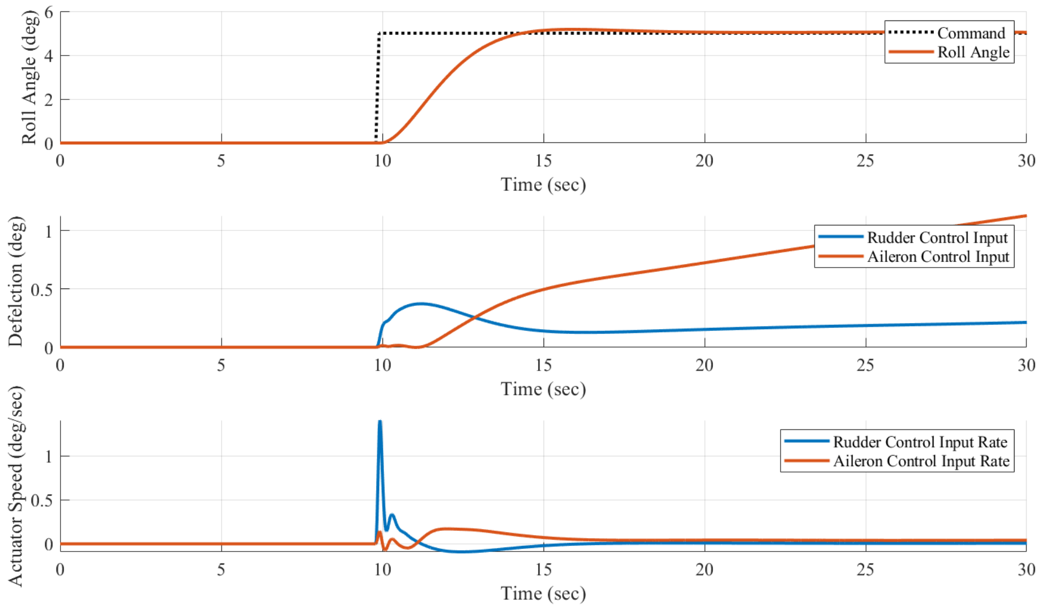

Fig. 21 Lateral LQR with low-pass filter

Using general knowledge of aircraft dynamics, the LQR controller can then be intuitively tuned for a much faster response. Minimal manipulation of the Q and R matrices yield a faster rise time by two seconds and acceptable actuator deflection speed, as seen in figure 23. The R matrix did not change, and the final Q matrix used is shown below

$$Q = \begin{bmatrix} 5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0001 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.01 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 500 \end{bmatrix}$$

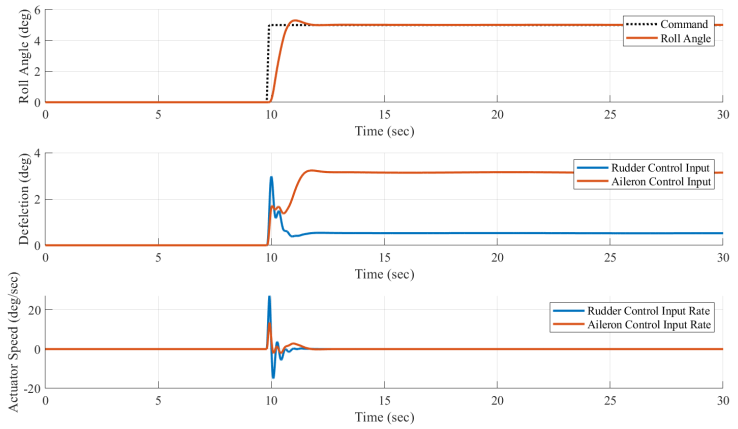

Fig. 22 Tuned Lateral LQR Controller

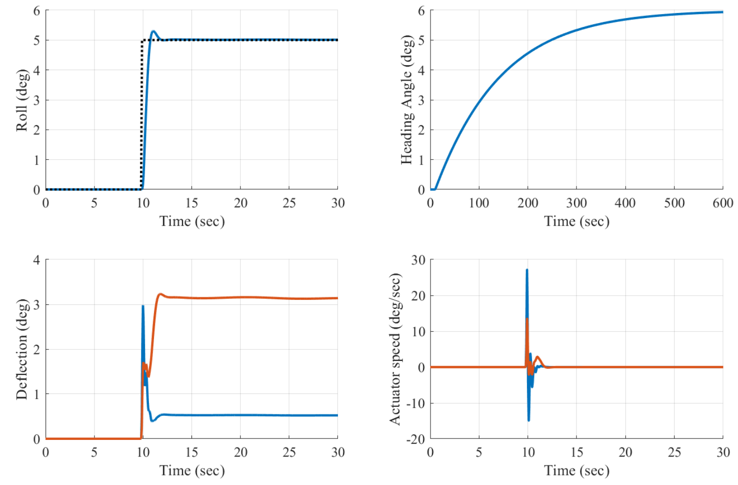

Fig. 23 Tuned Lateral LQR Controller

VI. Controller Comparisons

In general, the PID controller is to be favorably compared to the LQR controller in terms of its simplicity but the opposite applies to the fact that PID does not optimize any defined cost function and is usually tuned heuristically.

	RiseTime (sec)	Settling Time (sec)	Overshoot (%)	Undershoot (%)	Peak (amplitude)	Peak Time(sec)
PID$_{long}$	1.0535	7.9044	9.7773	55.9472	2.1955	1.11
PID$_{lat}$	0.12835	2.3763	21.5908	0	6.0796	0.15298
LQR$_{long}$	3.4422	8.7421	0	0.18549	2	219.93
LQR$_{lat}$	0.54445	1.5429	6.0542	0	5.3031	1.06

Table 4 Transient Response Comparison

A. PID vs LQR Longitudinal Response

Comparing the longitudinal response between PID and LQR control, figure 24 shows a considerably more desirable response from LQR. This, backed up with data in table 4, shows a more desirable settling time and elimination of any overshoot.

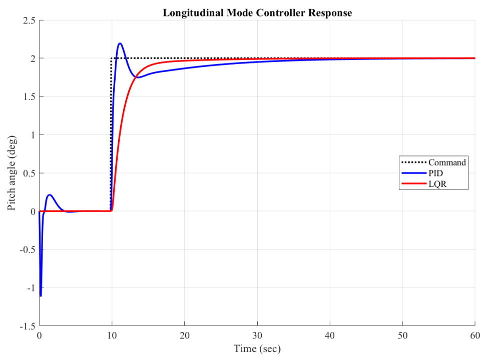

Fig. 24 Longitudinal Response Comparison

This more desirable response is due to the LQR controller using state feedback while also allowing the control engineer to tune specific parameters of the response. This allows for the system to be tuned quickly and intuitively while providing an optimal response with no overshoot or oscillations.

B. PID vs LQR Lateral Response

Comparing the lateral response between PID and LQR, figure 25 shows that the PID controller has more than triple the overshoot of the LQR. Due to this overshoot and lower damping ratio, there is a fair bit of oscillation as a trade off with the rise time. The PID controller does shoot up to the desired value far more quickly, but the transient response values found in table 4 show that settling time is actually worse than LQR. As was the case with the longitudinal comparison, PID initially responds much faster but is less stable and more error prone on longer time spans.

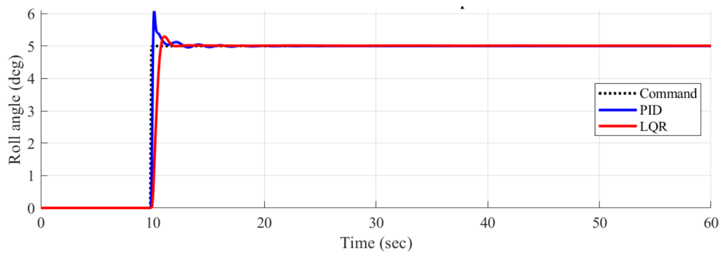

Fig. 25 Lateral Response Comparison

VII. Conclusion

Throughout the semester, this group has studied in order to understand the two different flight controllers, PID and LQR, and worked to implement them as best as possible in a simulated environment. As the group worked and learned from lectures, including "in class activities" that were assigned during the COVID-19 outbreak, the group found different methods of improvement that worked best for each controller, and were able to integrate these improvements into the Simulink code. While not every feature possible has been added to either of the controllers, it is clear that what the group learned from the class has been applied in order to make proper judgement as to if and how certain systems, such as CAS, should be added. The primary objectives of this class were all obtained by the group members by utilizing the dynamic model of the aircraft in order to design a classical controller (PID), and a more modern controller (LQR) for the given objectives. As such, all group members should now be better equipped to apply control theory to any dynamic system and design the right controller for any given situation.

As for the overall success of the controller designs, it should be easy to see that this group of students optimized them to the greatest extent possible given that hardware testing is currently not available. Considering the testing limitations imposed by the current COVID-19 pandemic, it is worth noting that the graphs generated by the simulation have a number of unknown factors that the physical hardware and testing conditions would reveal. However, based on the accuracy of the simulation results alone, the group is confident in the PID and LQR designs considering the objectives mentioned earlier. If there are to be future plans, by implementing the controllers produced in this paper on hardware and conducting test flights producing good results quickly should not be an issue. Aside from any changes that would be brought on as a result of those physical testing results, the controllers discussed above are adequate controllers for the aircraft modeled.

Appendix

A. Github

Here is the URL of the Github page that contains all of the code used in the project.

`https://github.com/brianrcaskey/Aer531`

B. Simulink EOM Block Code

```
function x_dot = EOMs(in)
    %% Aircraft Equations of Motion
    % Calculates the desired responses in "Development of Small Unmanned Aerial Vehicle Res
    % Modeling and Simulating with Flight Test Validation " Given the
    % following:

    % INPUT (in): vector containing control inputs and trim conditions
    %   in = [T, de, drt, da, V , gamma, alpha, q p, mu, beta, r, chi, north,
    %   east, h, t]

    % OUTPUT (x_dot): vector containg response states
    %   x_dot = [V , gamma, alpha, q p, mu, beta, r, chi, north,
    %   east, h]


    % Initializing variables
    %=====================================================================
    x_dot = zeros(12,1);
    alpha_dot = 0;

    %temporary input vector
    %u0 = [1.29, -0.1174, 0,  0, 64.828,     0,  -0.0153, 0,  0,  0,     0, 0,
0,     0,    0, 1119, 100];
        % [T,     de,    dr, da, V,      gamma,  AoA,     q,  p, mu,  beta, r, chi, north,
h, clock-input]


    % PARSING INPUTS
    %=====================================================================
    T   = in(1); %Throttle
    de  = in(2); %Elevator deflection (down is +) {deg}
    drt = in(3); %Rudder deflection {deg}
    da  = in(4); %Aileron deflection {deg}

    V        = in(5);  %Velocity {ft/s}
    gamma    = in(6);  %Flight path angle {rad}
    alpha    = in(7);  %Angle of attack {rad}
    q        = in(8);  %Pitch Rate {rad/s}
    p        = in(9);  %Roll Rate {rad/s}
    mu       = in(10); %Bank Angle (About Velocity Vector) {rad}
    beta     = in(11); %Sideslip Angle {rad}
    r        = in(12); %Yaw Rate {rad/s}
    chi      = in(13); %Heading angle {rads}
    north    = in(14); %North Position {ft}
    east     = in(15); %East Position {ft}
    h        = in(16); %Altitude {ft}

    tm = in(17);

    % ATMOSPHERIC PROPERTIES
    %==========================================================================
    rho = 0.0023081;    %Air Density (slugs per ft^3) -> do we want to make this a functio
    g = 32.17;          %ft/s^2


    % AIRCRAFT PROPERTIES
    %==========================================================================
    m = 0.487669;       %Slugs - empty mass w/o fuel
    Iyy = 1.5523;       %Inertia found with empty mass
    Ixx = 1.948;        %Inertia
    Izz = 1.9166;       %Inertia
    Ixz = 0;            %Assumed zero due to symmetric aircraft

    S = 10.56;          %Wing area - square feet
    b = 9.16;           % Wing span - feet

    CLow = 0.421;       %Wing coefficient of lift at 0deg AoA
    CLaw = 4.59;        %Wing coefficient of lift per AoA
    CDminw = 0.011;     %wing minimum coefficient of drag
    ARw = (b^2)/S;      %Wing aspect ratio
    e = 0.75;           %Span efficiency - ESTIMATION
    Kw = 1/(pi*ARw*e);  %
    Cmw = -0.005;       %Wing moment coefficient
    cgw = -(5/12);      %Distance aero center is back from cg - 5 inches
    c = (16/12);        %Root chord of wing (16") - feet
    lambda = 0.72955;   %Taper ratio from S = (Cr*(1+lambda)*b)/2
    CLat = 0.76;        %Vert tail coefficient of lift per AoA
    CDmint = 0.002;     %vert tail minimum coeffcient of drag
    Kt = 0.446;         %TO DO - MORE CALC?
    it = deg2rad(2);    %Tail incidence 2 degrees to radians
    Te = 0.422;         %Tail control surface effectiveness (??)
    nt = 1;             %??
    St = S;             %Horizontal tail area square feet = ref area (??)
    cgt = 3.5;          %Distance tail aero center back from a/c cg

    CLavt = 0.0969;     %Vertical tail coefficient of lift per AoA(??)
    CDminvt = 0.001;    %vert tail min coefficient of drag (??)
    Svt = S;            %Ref area of vertical tail = ref area (??)
    Tr = 0.434;         %Name??
    nvt = nt;           %same as hori tail (??)
    cgvt = cgt;         %same as hori tail (??)

    Cmaf = 0.114;       %fuselage moment coefficient
    CDf = 0.005;        %Fuselage coefficient of drag

    Cnda = -0.0128;     %per rad (??)
    Clda = 0.244;       %per rad (??)
```

```matlab


% EOM Parameters Computations
%==========================================================================
%Eqns 17-25
CLw = CLow + CLaw*alpha;
CDw = CDminw + Kw*CLw^2;
E = 2*(CLow + CLaw*(alpha - alpha_dot*(cgt + cgw)/V))/(pi*ARw);
alphat = alpha +it + Te*de + q*cgt/V - E;
CLt = CLat*alphat;
CDt = CDmint + Kt*CLt^2;
Cmf = Cmaf*alpha;
CDvt = CDminvt;
Clp = -1/12*CLaw*(1 + 3*lambda)/(1 + lambda);

Clb=-.1;
Clr=.01;

%Eqns 26 - 33
qb = .5*rho*V^2;
Lw = qb*S*CLw;
Dw = qb*S*CDw;
Mw = qb*S*c*Cmw;
Lt = nt*qb*St*CLt;
Dt = nt*qb*St*CDt;
Df = qb*S*CDf;
Mf = qb*S*c*Cmf;
Dvt = nt*qb*Svt*CDvt;

% Calculate Forces and Moments
% Lift
L = Lw+Lt*cos(E - q*cgt/V)-(Dt + Dvt)*sin(E - q*cgt/V);
% Drag
D = Dw +(Dt+Dvt)*cos(E - q*cgt/V)+Lt*sin(E - q*cgt/V) + Df;
% Side Force
Y = nvt*qb*Svt*CLavt*(-beta + Tr*drt + r*cgvt/V);
% Pitch Moment
Mc = Lw*cgw*cos(alpha) + Dw*cgw*sin(alpha) + Mw-Lt*cgt*cos(alpha - E+q*cgt/V)...
        -(Dt+Dvt)*cgt*sin(alpha - E+q*cgt/V) + Mf;
% Yaw Moment
Nc = -qb*nvt*Svt*CLavt*(-beta + Tr*drt + r*cgvt/V)*cgvt + (-qb*S*b*Cnda*da);
% Roll Moment
Lc = qb*S*b^2/(2*V)*(Clp*p + 2*V/b*Clb*beta+Clr*drt+Clda*da*2*V/b);




%% NONLINEAR 6-DOF EQUATION OF MOTION (EOMs)
%
% These are the state derivative equations; the comment names the state,
% but the equation is for its derivative (rate)
%
% The equations are arranged by aircraft mode
%
```

```
% NOTE: These assume that Ixz=0, if not, then eq's need to be modified
% Longitudinal (phugoid and short period): V, gamma, q, alpha
% Phugoid: V, gamma

% Velocity
V_dot = 1/m*(-D*cos(beta)+Y*sin(beta)+T*cos(beta)*cos(alpha))-...
        g*sin(gamma);
% Flight Path Angle
gamma_dot = 1/(m*V)*(-D*sin(beta)*sin(mu)-Y*sin(mu)*cos(beta)...
            +L*cos(mu)+T*(cos(mu)*sin(alpha)+sin(mu)*sin(beta)*cos(alpha)))...
            -g/V*cos(gamma);
% Short Period: alpha, q
% Angle of Attack
alpha_dot = q-tan(beta)*(p*cos(alpha)+r*sin(alpha))-1/(m*V*cos(beta))...
            *(L+T*sin(alpha))+g*cos(gamma)*cos(mu)/(V*cos(beta));
% Pitch Rate
q_dot = Mc/Iyy+1/Iyy*(Izz*p*r-Ixx*r*p);

% Lateral (roll) - Directional (yaw): p, mu, beta, r
%==========================================================================
% Roll: p, mu
% Roll Rate
p_dot = Lc/Ixx+1/Ixx*(Iyy*r*q-Izz*q*r);

% Bank Angle (about velocity vector)
mu_dot = 1/cos(beta)*(p*cos(alpha) + r*sin(alpha)) + 1/(m*V)*(D*sin(beta)...
         *cos(mu)*tan(gamma)+Y*tan(gamma)*cos(mu)*cos(beta)+L*(tan(beta)+...
         tan(gamma)*sin(mu))+T*(sin(alpha)*tan(gamma)*sin(mu)+sin(alpha)*...
         tan(beta)-cos(alpha)*tan(gamma)*cos(mu)*sin(beta)))-...
         g/V*cos(gamma)*cos(mu)*tan(beta);

% Dutch Roll: beta, r
% Side Slip Angle
beta_dot = -r*cos(alpha)+p*sin(alpha)+1/(m*V)*(D*sin(beta)+Y*cos(beta)-...
            T*sin(beta)*cos(alpha))+g/V*cos(gamma)*sin(mu);

% Yaw Rate
r_dot = Nc/Izz+1/Izz*(Ixx*p*q-Iyy*p*q);

% Heading Angle (from North)
chi_dot = 1/(m*V*cos(gamma))*(D*sin(beta)*cos(mu)+Y*cos(mu)*cos(beta) +...
           L*sin(mu)+T*(sin(mu)*sin(alpha)-cos(mu)*sin(beta)*cos(alpha)));

% Kinematic Equations
%==========================================================================
%  North Position
 n_dot=V*cos(gamma)*cos(chi);

% East Position
e_dot = V*cos(gamma)*sin(chi);

% Altitude
h_dot = V*sin(gamma);

```

```
    % Pack derivatives into output vector x_dot
    x_dot(1) = V_dot;
    x_dot(2) = gamma_dot;
    x_dot(3) = alpha_dot;
    x_dot(4) = q_dot;
    x_dot(5) = p_dot;
    x_dot(6) = mu_dot;
    x_dot(7) = beta_dot;
    x_dot(8) = r_dot;
    x_dot(9) = chi_dot;
    x_dot(10) = n_dot;
    x_dot(11) = e_dot;
    x_dot(12) = h_dot;

end
```

C. Simulink Simulation Verification Script

```
%% AerE 531 Final Project Initialization Script
% AF_Sim_Init.m defines the trim and initial conditions for the

clear,clc,
close all

%% Trim Conditions
u0 = [ 1.29  -0.1174  0  0];%[dt de dr da]
% u0 = [1.29, 0, 0,  0];

% setup initial conditions for state
IC = [64.828,     0,  -0.0153, 0,  0,  0,     0, 0,   0,     0,    0, 1119];

uin = [0, 0, 0, 0];

% Time array
T1 = 0.1*[0:5999]';

% Trim conditions arrays
SS = zeros(6000,1);
thr = 1.29*ones(6000,1);
throttle = [T1, thr];
aileron = [T1, SS];
rudder = [T1, SS];
E1 = -0.1174*ones(6000,1);
elevator = [T1, E1];

%% State Space

% Initial conditions for state
V0 = IC(1);
alpha0 = IC(3);
q0 = IC(4);
theta0 = 0;

uin = [0, 0, 0, 0];
```

```


% create elevator doublet input
% elevator(1000:1050, 2) = 0;
% elevator(1051:1100, 2) = -0.22;

roll = zeros(6000,1);
roll(100:6000) = 5;
pitch = zeros(5000,1);
pitch(100:6000) = 2;

roll_cmd = [T1, roll];%degrees
pitch_cmd = [T1, pitch];

%% Open Loop
% % create elevator doublet
% elevator(1000:1050, 2) = 0;
% elevator(1051:1100, 2) = -0.22;
%
% % run Simulink model for elevator doublet response
% sim('AF_Sim2',300);
% figure
% yyaxis left
% time = linspace(1,300,length(ans.Altitude))
% plot(time,ans.Altitude)
% ylabel('Altitude (ft)')
%
% yyaxis right
% plot(T1,(180/pi).*elevator(:,2))
% ylabel('Elevator Deflection (deg)')
% grid on
% title('Longitudinal Response to Elevator Doublet')
% legend on
% legend ('Altitude (ft)', 'Elevator Deflection (deg)')
% hold off
%
% % run Simulink model for rudder input response
% elevator = [T1, E1]; % reset elevator input matrix
% rudder(1000:1020, 2) = 10*pi/179;
% %rudder(1051:1100, 2) = -0.05;
%
%
% sim('AF_Sim2',300);
%
% figure
% yyaxis left
% time = linspace(1,300,length(ans.HeadingAngle))
% plot(time,(180/pi)*ans.HeadingAngle)
% ylabel('Heading Angle (deg)')
%
% yyaxis right
% plot(time,(180/pi)*ans.BankAngle)
% hold on
% plot(T1,(180/pi)*rudder(:,2))
% ylabel('Bank Angle & Rudder Deflection (deg)')
% grid on
% title('Heading and Bank Angle Response to Rudder Input')
% legend on
% legend ('Heading Angle (deg)', 'Bank Angle (deg)', 'Rudder Deflection (deg)')
% hold off

%% LQR Longitudonal
x0 = [ theta0 V0  alpha0  q0]';

A_long = [...
   0          0          0     1.0000       0 % theta
 -32.2000    -48    13.1983          3      0 % V
  0.0001    -0.0253    -0.2370     1.0000    0 % alpha
    0    -0.0549    -5.5837    -3.7947      0 % q
   -1         0          0          0       0 ]; % Int(theta_cmd - theta)dt

B_long = [...
     0
     0
     0.9
   2.9035
     0 ];

% LQR setup
Q = diag([1 0.1 0.1 0.1 1000]);

R = 1e-1;
% Q = diag([0.1 0.1 1 10 100]);
%
% R = 1e-2;

Klqr = lqr(A_long, B_long, Q, R);
%% LQR Lateral
max_dr = 30;
max_da = 15;

A_lat = [ 0         0          0     1.0000          0   0
         0          0          0          0     1.0000   0
    0.4967          0    -0.0002    -0.0130    -0.9593   0
         0          0    -0.0076     0.0040     0.0030   0
         0          0     0.0010    -0.6571    -0.0002   0
        -1          0          0          0          0   0];


B_lat = [0          0
         0          0
         0     0.3726
    1.0256     4.6310
   -0.0549   -28.1998
         0          0];


% First plot: STOP at 26 sec
% Q = diag([1,1,1,1,1,1]);

% % Q tune
% Q = diag([1,0.001,1,1,1,1]);

% % Third
% Q = diag([1,0.001,1,1,1,1]);
% wc = 0.025;
%
% % Final plot (tuned):
Q = diag([5,0.0001,0.01,2,1.5,500]);
wc = 0.025;

R = diag([1/(max_da^2), 1/(max_dr^2)]);
k_lqr = lqr(A_lat, B_lat, Q, R);

%% Simulation
tsim = 600;
out = sim('ControllersComparison.slx',tsim);
t = out.tout;


%% Parsing
% States
PID_Long_States = out.yout{1}.Values.Data;
PID_Lat_States = out.yout{5}.Values.Data;
LQR_Long_States = out.yout{9}.Values.Data;
LQR_Lat_States = out.yout{13}.Values.Data;

% Commands
% pitch_cmd = out.yout{17}.Values.Data;
% roll_cmd = out.yout{18}.Values.Data;

% Responses
pitch_PID = out.yout{2}.Values.Data;
pitch_LQR = out.yout{10}.Values.Data;
roll_PID = out.yout{6}.Values.Data;
roll_LQR = out.yout{14}.Values.Data;

% Controller
% Time History Input
u_PIDlong = out.yout{3}.Values.Data;
u_PIDlat = out.yout{7}.Values.Data;
u_LQRlong = out.yout{11}.Values.Data;
u_LQRdr = out.yout{16}.Values.Data(:,2);
u_LQRda = out.yout{15}.Values.Data(:,2);

% Time History Rate
du_PIDlong = out.yout{4}.Values.Data;
du_PIDlat = out.yout{8}.Values.Data;
du_LQRlong = out.yout{12}.Values.Data;
du_LQRdr = out.yout{16}.Values.Data(:,1);
du_LQRda = out.yout{15}.Values.Data(:,1);

```

```
% Open Loop Response
Long_response = out.yout{17}.Values.Data;
Lat_response = out.yout{19}.Values.Data;
oploop_pitch = out.yout{18}.Values.Data;
oploop_roll = out.yout{20}.Values.Data;

%% Comparisons
ctrl_PIDlong = stepinfo(pitch_PID,t);
ctrl_PIDlat = stepinfo(roll_PID,t);
ctrl_LQRlong = stepinfo(pitch_LQR,t);
ctrl_LQRlat = stepinfo(roll_LQR,t);

comb = [ctrl_PIDlong ctrl_PIDlat ctrl_LQRlong ctrl_LQRlat];
compTable = struct2table(comb);
%Fix for command at 10 sec
compTable.SettlingTime = compTable.SettlingTime-10;
compTable.PeakTime = compTable.PeakTime-10;

ctable = table(compTable,'RowNames',{'Longitudinal PID','Lateral PID','Longitudinal LQR','

table2latex(compTable,'comparison_table')

%% Plotting
linethic = 2.0;
r2d = 180/pi;

%% Longitudinal PID Results
[V,gamma,alpha,q,p,mu,beta,r,chi,N,E,alt_PID] = stateparser(PID_Long_States);
tf = 50; % Change this for plotting time, max is 100

figure('Name','LONG_OPLoop','Position', [100 100 1100 600]); hold on ; grid on
plot(t,oploop_pitch,'LineWidth',linethic,'DisplayName','Pitch Angle')
plot(T1,pitch_cmd(:,2),':K','LineWidth',linethic,'DisplayName','Command')
legend
ylabel('Pitch Angle (deg)')
xlabel('Time (sec)')
set(gca,'fontsize',12,'fontname','times')

% figure
figure('Name','LONG_PID_1','Position', [100 100 1100 600])

subplot(3,1,1); hold on ; grid on
plot(t,pitch_PID,'LineWidth',linethic,'DisplayName','Pitch Angle')
plot(T1,pitch_cmd(:,2),':K','LineWidth',linethic,'DisplayName','Command')
xlim([0 tf])
legend
ylabel('Pitch Angle (deg)')
xlabel('Time (sec)')
set(gca,'fontsize',12,'fontname','times')

subplot(3,1,2); hold on ; grid on
plot(t,u_PIDlong,'LineWidth',linethic,'DisplayName','Control Input')
xlim([0 tf])
```

```
legend
ylabel('Defelction (deg)')
xlabel('Time (sec)')
set(gca,'fontsize',12,'fontname','times')


subplot(3,1,3); hold on ; grid on
plot(t,du_PIDlong,'LineWidth',linethic,'DisplayName','Control Input Rate')
xlim([0 tf])
legend
ylabel('Actuator Speed (deg/sec)')
xlabel('Time (sec)')
set(gca,'fontsize',12,'fontname','times')


% figure
figure('Name','LONG_PID_2','position',[50 50 1000 600])
subplot 221; hold on ; grid on
plot(t, pitch_PID,'linewidth',2)
plot(T1, pitch_cmd(:,2),':k','linewidth',linethic)
xlim([0 tf])
% ylim([-0.2 2.2])
xlabel('Time (sec)'); ylabel('Pitch (deg)');
set(gca,'fontsize',12,'fontname','times')

subplot 222; hold on ; grid on
plot(t, alt_PID,'linewidth',linethic)
xlim([0 tf])
xlabel('Time (sec)'); ylabel('Altitude (ft)');
set(gca,'fontsize',12,'fontname','times')

subplot 223; hold on ; grid on
plot(t, u_PIDlong,'linewidth',linethic)
xlim([0 tf])
xlabel('Time (sec)'); ylabel('Deflection (deg)')
set(gca,'fontsize',12,'fontname','times')

subplot 224; hold on ; grid on
plot(t, du_PIDlong,'linewidth',linethic)
xlim([0 tf])
xlabel('Time (sec)')
ylabel('Actuator speed (deg/sec)')
set(gca,'fontsize',12,'fontname','times')


%% Lateral PID Results
[V,gamma,alpha,q,p,mu,beta,r,chi_PID,N,E,alt] = stateparser(PID_Lat_States);
tf = 20; % Change this for plotting time, max is 100

figure('Name','LAT_PID_1', 'Position', [100 100 1100 600])

subplot(3,1,1); hold on ; grid on
plot(T1,roll_cmd(:,2),':K','LineWidth',1.0,'DisplayName','Command')
plot(t,roll_PID,'LineWidth',linethic,'DisplayName','Roll Angle')
```

```
legend
ylabel('Roll Angle (deg)')
xlabel('Time (sec)')
xlim([0 tf])
set(gca,'fontsize',12,'fontname','times')

subplot(3,1,2); hold on ; grid on
plot(t,u_PIDlat,'LineWidth',linethic,'DisplayName','Control Input')
legend
ylabel('Defelction (deg)')
xlabel('Time (sec)')
xlim([0 tf])
set(gca,'fontsize',12,'fontname','times')

subplot(3,1,3); hold on ; grid on
plot(t,du_PIDlat,'LineWidth',linethic,'DisplayName','Control Input Rate')
legend
ylabel('Actuator Speed (deg/sec)')
xlabel('Time (sec)')
xlim([0 tf])
set(gca,'fontsize',12,'fontname','times')

% figure
figure('Name','LAT_PID_2','position',[50 50 1000 600])
subplot 221; hold on ; grid on
plot(t, roll_PID,'linewidth',2)
plot(T1, roll_cmd(:,2),':k','linewidth',linethic)
xlim([0 tf])
% ylim([-0.2 2.2])
xlabel('Time (sec)'); ylabel('Pitch (deg)');
set(gca,'fontsize',12,'fontname','times')

subplot 222; hold on ; grid on
plot(t, chi_PID,'linewidth',linethic)
xlim([0 t(end)])
xlabel('Time (sec)'); ylabel('Heading Angle (deg)');
set(gca,'fontsize',12,'fontname','times')

subplot 223; hold on ; grid on
plot(t, u_PIDlat,'linewidth',linethic)
xlim([0 tf])
xlabel('Time (sec)'); ylabel('Deflection (deg)')
set(gca,'fontsize',12,'fontname','times')

subplot 224; hold on ; grid on
plot(t, du_PIDlat,'linewidth',linethic)
xlim([0 tf])
xlabel('Time (sec)')
ylabel('Actuator speed (deg/sec)')
set(gca,'fontsize',12,'fontname','times')

%% Longitudinal LQR Results
[V,gamma,alpha,q,p,mu,beta,r,chi,N,E,alt_LQR] = stateparser(LQR_Long_States);
%
```

```
figure('Name','LONG_LQR_1','Position', [100 100 1100 600])

subplot(3,1,1); hold on ; grid on
plot(T1,pitch_cmd(:,2),'LineWidth',1.0,'DisplayName','Command')
title('Longitudinal LQR Control')
plot(t,pitch_LQR,'LineWidth',1.0,'DisplayName','Pitch Angle')
legend
ylabel('Pitch Angle (deg)')
xlabel('Time (sec)')
set(gca,'fontsize',12,'fontname','times')

subplot(3,1,2); hold on ; grid on
plot(t,u_LQRlong-u0(2),'LineWidth',1.0,'DisplayName','Control Input')
xlim([0 tf])
legend
ylabel('Defelction (deg)')
xlabel('Time (sec)')
set(gca,'fontsize',12,'fontname','times')

subplot(3,1,3); hold on ; grid on
plot(t,du_LQRlong,'LineWidth',1.0,'DisplayName','Control Input Rate')
legend
xlim([0 tf])
ylabel('Actuator Speed (deg/sec)')
xlabel('Time (sec)')
set(gca,'fontsize',12,'fontname','times')

%% Lateral LQR Results
[V,gamma,alpha,q,p,mu,beta,r,chi_LQR,N,E,alt] = stateparser(LQR_Lat_States);
tf = 30; % Change this for plotting time, max is 100

figure('Name','LAT_LQR_1','Position', [100 100 1100 600]);

subplot(3,1,1); hold on ; grid on
plot(T1,roll_cmd(:,2),':K','LineWidth',linethic,'DisplayName','Command')
plot(t,roll_LQR,'LineWidth',linethic,'DisplayName','Roll Angle')
legend
ylabel('Roll Angle (deg)')
xlabel('Time (sec)')
xlim([0 tf])
set(gca,'fontsize',12,'fontname','times')


subplot(3,1,2); hold on ; grid on
plot(t,u_LQRdr,'LineWidth',linethic,'DisplayName','Rudder Control Input')
plot(t,u_LQRda,'LineWidth',linethic,'DisplayName','Aileron Control Input')
legend
ylabel('Defelction (deg)')
xlabel('Time (sec)')
xlim([0 tf])
set(gca,'fontsize',12,'fontname','times')


subplot(3,1,3); hold on ; grid on
```

```
plot(t,du_LQRdr,'LineWidth',linethic,'DisplayName','Rudder Control Input Rate')
plot(t,du_LQRda,'LineWidth',linethic,'DisplayName','Aileron Control Input Rate')
legend
ylabel('Actuator Speed (deg/sec)')
xlabel('Time (sec)')
xlim([0 tf])
set(gca,'fontsize',12,'fontname','times')


% figure
figure('Name','LAT_LQR_2','position',[50 50 1000 600])
subplot 221; hold on ; grid on
plot(t, roll_LQR,'linewidth',linethic)
plot(T1, roll_cmd(:,2),':k','linewidth',linethic)
xlim([0 tf])
% ylim([-0.2 2.2])
xlabel('Time (sec)'); ylabel('Roll (deg)');
set(gca,'fontsize',12,'fontname','times')

subplot 222; hold on ; grid on
plot(t, chi_LQR,'linewidth',linethic)
xlim([0 t(end)])
xlabel('Time (sec)'); ylabel('Heading Angle (deg)');
set(gca,'fontsize',12,'fontname','times')

subplot 223; hold on ; grid on
plot(t, u_LQRdr,'linewidth',linethic,'DisplayName','Rudder')
plot(t, u_LQRda,'linewidth',linethic,'DisplayName','Aileron')
xlim([0 tf])
xlabel('Time (sec)'); ylabel('Deflection (deg)')
set(gca,'fontsize',12,'fontname','times')

subplot 224; hold on ; grid on
plot(t, du_LQRdr,'linewidth',linethic,'DisplayName','Rudder')
plot(t, du_LQRda,'linewidth',linethic,'DisplayName','Aileron')
xlim([0 tf])
xlabel('Time (sec)')
ylabel('Actuator speed (deg/sec)')
set(gca,'fontsize',12,'fontname','times')
%% Controller Results
figure('Name','Controller_Comp','position',[50 50 900 600])

subplot(2,1,1); hold on ; grid on
title('Longitudinal Mode Controller Response')
plot(T1,pitch_cmd(:,2),':K','LineWidth',2)
% plot(t,oploop_pitch,'LineWidth',2)
plot(t,pitch_PID,'b','LineWidth',2)
plot(t,pitch_LQR,'r','LineWidth',2)
ylabel('Pitch angle (deg)'); xlabel('Time (sec)'); xlim([0 60])
legend('Command', 'PID','LQR','location','east')
set(gca,'fontsize',12,'fontname','times')

subplot(2,1,2);hold on ;grid on
title('Lateral Mode Controller Response')
```

```
plot(T1,roll_cmd(:,2),':K','LineWidth',2)
% plot(t,oploop_roll,'LineWidth',2)
plot(t,roll_PID,'b','LineWidth',2)
plot(t,roll_LQR,'r','LineWidth',2)
ylabel('Roll angle (deg)'); xlabel('Time (sec)'); xlim([0 60])
legend('Command', 'PID','LQR','location','east')
set(gca,'fontsize',12,'fontname','times')

figure('Name','State_Comp','position',[50 50 900 600])

subplot(2,1,1); hold on ; grid on
plot(t,alt_PID,'b','LineWidth',2)
plot(t,alt_LQR,'r','LineWidth',2)
ylabel('Altitude (ft)'); xlabel('Time (sec)'); xlim([0 40])
legend('PID','LQR','location','east')
set(gca,'fontsize',12,'fontname','times')

subplot(2,1,2);hold on ;grid on
% title('Heading Angle')
plot(t,chi_PID,'b','LineWidth',2)
plot(t,chi_LQR,'r','LineWidth',2)
ylabel('Heading Angle (deg)'); xlabel('Time (sec)');
legend('PID','LQR','location','east')
set(gca,'fontsize',12,'fontname','times')


%% Final Gains Table

%% Save Figures
FolderName = 'Plots';   % Your destination folder
FigList = findobj(allchild(0), 'flat', 'Type', 'figure');
for iFig = 1:length(FigList)
  FigHandle = FigList(iFig);
  FigName   = get(FigHandle, 'Name');
  saveas(FigHandle, [FolderName '/' FigName '.png']);
end
```

D. Rascall State Space Model creation

```
clear,clc
I_xx_B = 1.948; %[slug*ft^2]
I_yy_B = 1.5523;
I_zz_B = 1.9166;
I_xz_B = 0;

S =10.56;
m = 0.487669;
cbar=16/2;
b=9.16;

u1=64.8280; %ft/sec
V = u1;
alpha = 2*pi/180;
alpha_dot = 0;
```

```

de = -0.1174; %rads

% ATMOSPHERIC PROPERTIES
%==============================================================
rho = 0.0023081;    %Air Density (slugs per ft^3) -> do we want to make this a function of
g = 32.17;          %ft/s^2


% AIRCRAFT PROPERTIES
%==============================================================


m = 0.487669;       %Slugs - empty mass w/o fuel
Iyy = 1.5523;       %Inertia found with empty mass
Ixx = 1.948;        %Inertia
Izz = 1.9166;       %Inertia
Ixz = 0;            %Assumed zero due to symmetric aircraft

S = 10.56;          %Wing area - square feet
b = 9.16;           % Wing span - feet

q = 0.5*rho*(V^2);


CLow = 0.421;       %Wing coefficient of lift at 0deg AoA
CLaw = 4.59;        %Wing coefficient of lift per AoA
CDminw = 0.011;     %wing minimum coefficient of drag
ARw = (b^2)/S;      %Wing aspect ratio
e = 0.75;           %Span efficiency - ESTIMATION
Kw = 1/(pi*ARw*e);  %
Cmw = -0.005;       %Wing moment coefficient
cgw = -(5/12);      %Distance aero center is back from cg - 5 inches
c = (16/12);        %Root chord of wing (16") - feet
lambda = 0.72955;   %Taper ratio from S = (Cr*(1+lambda)*b)/2
CLat = 0.76;        %Vert tail coefficient of lift per AoA
CDmint = 0.002;     %vert tail minimum coeffcient of drag
Kt = 0.446;         %TO DO - MORE CALC?
it = deg2rad(2);    %Tail incidence 2 degrees to radians
Te = 0.422;         %Tail control surface effectiveness (??)
nt = 1;             %??
St = S;             %Horizontal tail area square feet = ref area (??)
cgt = 3.5;          %Distance tail aero center back from a/c cg

CLavt = 0.0969;     %Vertical tail coefficient of lift per AoA(??)
CDminvt = 0.001;    %vert tail min coefficient of drag (??)
Svt = S;            %Ref area of vertical tail = ref area (??)
Tr = 0.434;         %Name??
nvt = nt;           %same as hori tail (??)
cgvt = cgt;         %same as hori tail (??)

Cmaf = 0.114;       %fuselage moment coefficient
CDf = 0.005;        %Fuselage coefficient of drag

```

```
Cnda = -0.0128;      %per rad (??)
Clda = 0.244;        %per rad (??)


%Eqns 17-25
%Calculating necessary coefficients for use in Lift, Drag, and Moment
%equations
CLw=CLow+CLaw*alpha; %Coefficient of lift due to wing
CDw=CDminw+Kw*CLw^2; %Coefficient of drag due to wing
E=2*(CLow+CLaw*(alpha-alpha_dot*(cgt+cgw)/V))/(pi*ARw); %Not sure what to call this variabl
alphat=alpha+it+Te*de+q*cgt/V-E; %angle of attack at horizontal tail
CLt=CLat*alphat; %Coefficient of lift due to horizonatl tail
CDt=CDmint+Kt*CLt^2; %Coefficient of drag due to horizontal tail
Cmf=Cmaf*alpha; %Moment coefficient due to fueselage
CDvt=CDminvt; %Coefficient of drag due to vertical tail
Clp=-1/12*CLaw*(1+3*lambda)/(1+lambda);

Clb=-.1; %rolling moment coefficient due to sideslip beta
Clr=.01;

%Eqns 26 - 33
%Forces and moments due to wing, horizontal and vertical tail, and
%fuselage:
qbar=.5*rho*V^2; %q bar (dynamic pressure)
Lw=qbar*S*CLw;   %lift due to wing
Dw=qbar*S*CDw;   %drag due to wing
Mw=qbar*S*c*Cmw; %pitching moment due to wing
Lt=nt*qbar*St*CLt; %lift due to horizontal tail
Dt=nt*qbar*St*CDt; %drag due to horizontal tai
Df=qbar*S*CDf;    %drag due to fuselage
Mf=qbar*S*c*Cmf;  %moment due to fuselage
Dvt=nt*qbar*Svt*CDvt; %drag due to vertical tail


%==================================================================


% NOTE EVERYTHING IS CURRENTLY PER RADIAN
C_D_u = 0; %??
C_D_1 = CDw+CDt+CDvt; %Total drag coefficient

C_T_x_1 = C_D_1;
C_D_alpha=0.362; %??
C_D_delta_e =0;

C_L_1 = CLw+CLt; %Total lift coefficient
C_T_x_u = -0.162; %??
C_L_u =0;
C_L_alpha = deg2rad(0.11); %Rascal pg72
C_L_alpha_dot=4.5; %??
C_L_q = 4.885;  %??
C_L_delta_e = 0.9;  %??

C_m_u =0;
```

```
C_m_1 =0;
C_m_T_u =0;
C_m_T_1 =0;
C_m_alpha = deg2rad(-0.006); %Rascal pg72
C_m_T_alpha =0;
C_m_alpha_dot = -14.8; %??
C_m_q = deg2rad(-0.233); %Rascal pg72
C_m_delta_e= deg2rad(0.011); %Rascal pg72

C_y_beta = deg2rad(-0.0056); %Rascal pg72
C_y_p = -0.1138; %Extra paper pg24
C_y_r =0.356;
C_y_delta_a =0;
C_y_delta_r = 0.23;
C_l_beta = deg2rad(-0.0018); %Rascal pg72
C_l_p = deg2rad(0.013); %Rascal pg72
C_l_r = deg2rad(0.01); %Rascal pg72
C_l_delta_a = deg2rad(0.244); %Rascal pg72
C_l_delta_r = 0.0192;
C_n_beta = deg2rad(0.00023); %Rascal pg72
C_n_T_beta=0;
C_n_p =-0.0380; %% Find in 'useful paper'
C_n_r = deg2rad(-0.0006); %Rascal pg72
C_n_delta_a= deg2rad(-0.0128); %Rascal pg72
C_n_delta_r =-0.1152;


save('Aircraft_531.mat')
```

```
clear,clc

% Loading the data file.
load('Aircraft_531.mat') % You have to modify this line for other aircraft and mode.

% Transform the moment of inertia from body coordinates to stability
% coordinates
[I_xx_s, I_yy_s, I_zz_s, I_xz_s]...
    = MOI_trasnform(...
    I_xx_B, I_yy_B, I_zz_B, I_xz_B,...
    alpha);

% Calculate the longitudinal dimensional derivative
[X_u, X_T_u, X_alpha, X_delta_e, Z_u, Z_alpha, Z_alpha_dot,...
    Z_q, Z_delta_e, M_u, M_T_u, M_alpha, M_T_alpha, M_alpha_dot,...
    M_q, M_delta_e] ...
    = Longitudinal_derivatives(qbar, S, m, u1, I_yy_s, cbar,...
    C_D_u, C_D_1, C_T_x_u, C_T_x_1, C_D_alpha, C_L_1,...
    C_D_delta_e, C_L_u, C_L_alpha, C_L_alpha_dot,...
    C_L_q, C_L_delta_e, C_m_u, C_m_1, C_m_T_u, C_m_T_1,...
    C_m_alpha, C_m_T_alpha, C_m_alpha_dot, C_m_q, C_m_delta_e);

% Calculate the lateral-directional dimensional derivative
[Y_beta, Y_p, Y_r, Y_delta_a, Y_delta_r,...
    L_beta, L_p, L_r, L_delta_a, L_delta_r,...
    N_beta, N_T_beta, N_p, N_r, N_delta_a, N_delta_r]...
    = Lateral_derivatives(qbar, S, u1, m, b, I_xx_s, I_zz_s,...
    C_y_beta, C_y_p, C_y_r, C_y_delta_a, C_y_delta_r,...
    C_l_beta, C_l_p, C_l_r, C_l_delta_a, C_l_delta_r,...
    C_n_beta, C_n_p, C_n_r, C_n_delta_a, C_n_delta_r, C_n_T_beta);

% Calculate the state space matrices
[A_long, B_long, A_lat, B_lat]...
    = State_space_model(...
    alpha, u1, I_xz_s, I_xx_s, I_yy_s, I_zz_s,...
    X_u, X_T_u, X_alpha, X_delta_e, Z_u, Z_alpha, Z_alpha_dot,...
    Z_q, Z_delta_e, M_u, M_T_u, M_alpha, M_T_alpha, M_alpha_dot,...
    M_q, M_delta_e,...
    Y_beta, Y_p, Y_r, Y_delta_a, Y_delta_r,...
    L_beta, L_p, L_r, L_delta_a, L_delta_r,...
    N_beta, N_T_beta, N_p, N_r, N_delta_a, N_delta_r);

A_lat
A_long
B_lat
B_long
```

References

[1] Sauer, F., and Schörnig, N., "Killer drones: The 'silver bullet' of democratic warfare?" *Security Dialogue*, Vol. 43, No. 4, 2012, pp. 363–380. https://doi.org/10.1177/0967010612450207, URL https://doi.org/10.1177/0967010612450207.

[2] Fung, B., "Why drone makers have declared war on the word 'drone'," *The Washington Post*, Vol. 17, 2013.

[3] Saadatseresht, M., Hashempour, A., and Hasanlou, M., "UAV photogrammetry: a practical solution for challenging mapping projects," *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, Vol. 40, No. 1, 2015, p. 619.

[4] Noshahri, H., and Kharrati, H., "PID controller design for unmanned aerial vehicle using genetic algorithm," *2014 IEEE 23rd International Symposium on Industrial Electronics (ISIE)*, 2014, pp. 213–217.

[5] Salih, A., Moghavvemi, M., Haf, M., and Gaeid, K., "Flight PID Controller Design for a UAV Quadrotor," *Scientific research and essays*, Vol. 5, 2010, pp. 3660–3667.

[6] Ang, K. H., Chong, G., and Li, Y., "PID control system analysis, design, and technology," *IEEE transactions on control systems technology*, Vol. 13, No. 4, 2005, pp. 559–576.

[7] Gambone, E. A., *Pattern Recognition Control Design*, chapter and pages. https://doi.org/10.2514/6.2018-0855, URL https://arc.aiaa.org/doi/abs/10.2514/6.2018-0855.

[8] Sadraey, M., and Colgren, R., *A Dynamic Performance Evaluation Technique for Unmanned Aerial Vehicles*, chapter and pages. https://doi.org/10.2514/6.2006-6372, URL https://arc.aiaa.org/doi/abs/10.2514/6.2006-6372.

[9] Jodeh, N. M., "DEVELOPMENT OF AUTONOMOUS UNMANNED AERIAL VEHICLE RESEARCH PLATFORM: MODELING, SIMULATING, AND FLIGHT TESTING," 2006, pp. 75–91.

[10] Knospe, C., "PID Control," *IEEE Control Systems Magazine*, Vol. 26, No. 1, 2006, pp. 30–31.

[11] Prasad, L. B., Tyagi, B., and Gupta, H. O., "Optimal control of nonlinear inverted pendulum dynamical system with disturbance input using PID controller & LQR," *2011 IEEE International Conference on Control System, Computing and Engineering*, IEEE, 2011, pp. 540–545.