DP2: Control of a fixed-wing glider

Ficalora Luke * and Veranga, Joshua †
AE 353 - Design Project 2

We were tasked with designing and implementing a control system for a fixed-wing glider. In this report we will give a detailed description of the theory behind our methods, our implementation and our results. Using an iterative process and new control theory concepts, we were able to create a robust system that guarantees the safety of our pilots. Our control system executed safe landings in 10 out of 10 tests.

I. Nomenclature

p_x, p_y, p_z	=	World frame components of position, m
ϕ, θ, ψ	=	Body frame roll, pitch, and yaw angles, rad
v_x, v_y, v_z	=	Body frame velocity components, $\frac{\text{m}}{\text{s}}$
w_x, w_y, w_z	=	Body frame angular velocity components, $\frac{\text{rad}}{\text{s}}$
δ_r, δ_l	=	Elevator deflection angle for right and left elevators respectively, rad
R_B^W	=	Defined rotation matrix describing body frame components relative to world frame
F, M	=	Respective matrices describing body frame relative aerodynamic forces and aerodynamic moments
J^B	=	Body frame glider moment of inertia, $\text{kg} \cdot \text{m}^2$
Y, g, A, B, K, Q, R	=	Coefficient Matrices
x, u	=	Controller state-space vector and input vector
I	=	Identity matrix
β	=	Controller weight alteration constant

II. Introduction

Cat Plane Co. has been tasked with designing, simulating, and implementing a control system for a fixed-wing aircraft. Our goal is to create a robust and safe system for our skilled cat pilots. We will execute a thorough iteration and testing process to ensure that our system is the safest possible option.

The system we are controlling is composed of two parts: a fixed wing glider and a runway. The glider has no tail and is very similar to the 'Zagi' remote control plane. The lack of a tail reduces drag, but it does make controlling the glider more difficult. The only control surfaces on the glider are two 'evelons.' When they move in coordination, they act as elevators; when they move in opposition, they act as ailerons.

The runway is a 25 meter long floating platform. It is located 15 meters down and 147.5 meters horizontally from the gliders starting position. The glider starts with a randomized orientation and velocity within certain ranges: $\left[-\frac{\pi}{6}, \frac{\pi}{6}\right]$rad and $[4.5, 5.5]\frac{\text{m}}{\text{s}}$ respectively. The end goal is to control the glider as it approaches the runway, making it successfully touch down 2.5 meters from the edge and come to a stop. We will consider our system safe when three out of four landings are successful.

III. Theory

A. Equations of Motion

The Zagi UAV is modeled as a rigid body with six degrees of freedom with its motion governed by Newton-Euler equations. Where the state variables consist of [1]:

- Position $\boldsymbol{p}$: $\left[p_x, p_y, p_z\right]^T$

*lukeaf3
†veranga2

- Orientation $\boldsymbol{o}$: $\left[\psi, \theta, \phi\right]^T$
- Linear velocity $\boldsymbol{v}$: $\left[v_x, v_y, v_z\right]^T$
- Angular velocity $\boldsymbol{w}$: $\left[w_x, w_y, w_z\right]^T$
- Control inputs: Elevon deflections $\left[\delta_r, \delta_l\right]^T$.

And the governing equations of the system are given [1] by:

where $\boldsymbol{R}$ is a given rotational matrix [1], F is the aerodynamic and gravitational forces, and M represents the aerodynamic moments.

B. Equilibrium Computation

To analyze the aircraft around a steady-state condition, an equilibrium point —or trim condition— is found by solving the nonlinear optimization problem with the goal of adjusting the necessary* states and control inputs:

$$g = [p_y, \boldsymbol{o}, \boldsymbol{v}, \boldsymbol{w}, \delta_r, \delta_l]^T \tag{1}$$

to minimize the difference between the forces and moments acting on the aircraft and the desired equilibrium, i.e., zero net force and moment. This is accomplished by numerically minimizing the squared norm of the force and moment equations. Given an initial guess of g_0, the optimization problem is defined as:

$$\min_g \|Y(g)\|^2, \tag{2}$$

where $Y(g)$ represents the residual forces and moments, and the optimization is solved using a nonlinear optimizer [2]

C. Linearization

Once the equilibrium state is determined from solving Eq.(2), the system is linearized around this respective point. Where conventionally to state-space form equations, the linearized system is expressed as:

$$\dot{x} = Ax + Bu \tag{3}$$

where x represents the state vector and u represents the control input vector. For this system the aforementioned are expressed as:

$$x = \left[p_y, \psi, \theta, \phi, \boldsymbol{v}, \boldsymbol{w}\right]^T, \quad u = \left[\delta_r, \delta_l\right]^T. \tag{4}$$

Additionally the equilibrium state x_e and equilibrium input u_e are defined as the solution from Eq.(2):

$$x = \left[p_{y_e}, \psi_e, \theta_e, \phi_e, \boldsymbol{v}_e, \boldsymbol{w}_e\right]^T, \quad u = \left[\delta_{r_e}, \delta_{l_e}\right]^T. \tag{5}$$

Subsequently, the state and input matrices, A and B, are computed as Jacobians of the force matrix:

$$A = \left.\frac{\partial Y}{\partial x}\right|_{x_e, u_e}, \quad B = \left.\frac{\partial Y}{\partial u}\right|_{x_e, u_e} \tag{6}$$

and numerically evaluated in the equilibrium state.

D. LQR Control Design

To stabilize the system, an LQR controller is designed. Where the optimal feedback control law is defined as:

$$u = -Kx, \tag{7}$$

where K is a computed gain matrix obtained by solving the Continuous Algebraic Riccati Equation (CARE) after defining weight matrices Q and R.

*p_y is chosen as the only relevant term in $\boldsymbol{p}$ for the controller as it is used to ensure the flight path of the glider lines up with the runway and both p_x and p_z naturally change due to flight.

E. Obtaining Control Values

To fully devise a controller for the system, the process outlined above is implemented in Python code [3][†] utilizing parameters and code templates provided [1]. First, the optimization starts with an initial guess for the equilibrium state and control input, denoted respectively as g_0. These values are chosen based on potential initial conditions of the flight simulation [1]:

$$g_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 5 & 0 & -0.5 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T. \tag{8}$$

Where next numerical optimization minimizes the norm of the system's governing equations, solving $F(g) = 0$ from Eq.(2) and yielding the following equilibrium state x_e and control input u_e:

$$x_e = \begin{bmatrix} 0 & 3.80 \times 10^{-6} & \dots & 3.79 \times 10^{-7} & -6.99 \times 10^{-7} \end{bmatrix}, \quad u_e = \begin{bmatrix} -0.766 & -0.766 \end{bmatrix}. \tag{9}$$

Now utilizing the obtained equilibrium state, calculating the Jacobian of F with respect to state-space x and control input u (6) and subsequently evaluating numerically with the equilibrium x_e and u_e (9), produce the following:

$$A = \begin{bmatrix} 0 & 5.07 & \dots & 0 & 0 \\ 0 & 0 & \dots & 3.96 \times 10^{-6} & 1.04 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & -2.59 & -8.59 \times 10^{-8} \\ 0 & 0 & \dots & -4.66 \times 10^{-9} & -0.0774 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ -7.79 & -7.79 \\ 0.0531 & -0.0531 \end{bmatrix}. \tag{10}$$

While Python contains library commands that are capable of solving for P in the CARE, the matrices Q and R, which define penalized control over weighing the elements of the respective matrices x and u, must be defined by user input. Initial chosen values of the aforementioned are as followed:

$$Q = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T \circ I, \quad R = \begin{bmatrix} 1 & 1 \end{bmatrix}^T \circ I. \tag{11}$$

Meaning that each value is weighted equally to derive the respective control gain matrix. However, for reasons described in **Section IV**, the matrix Q was defined as:

$$Q = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 3 & 3 & 3 \end{bmatrix}^T \circ I. \tag{12}$$

Which solving for P then allows for the calculation of the gain matrix K. The final K obtained and utilized was the following:

$$K = \begin{bmatrix} -0.707 & -9.17 & -3.73 & -5.98 & 0.266 & -1.43 & 0.339 & -1.40 & -1.37 & -7.24 \\ 0.707 & 9.17 & -3.73 & 5.98 & 0.266 & 1.43 & 0.339 & 1.40 & -1.37 & 7.24 \end{bmatrix}. \tag{13}$$

IV. Experimental Methods

To develop the controller for the system, the gain matrix is implemented in the respective controller code cell provided in the Zagi code template [1]. This is done by denoting the matrix multiplication defined in Eq.(7) which produces the control matrix u, which again defines the necessary inputs of δ_r and δ_l to maintain the trim condition.

As expressed in **Section III.E**, the Q matrix was altered during testing as initial flight tests failed. This complication was overcome by increasing the weight assigned to the respective angular velocity, $\boldsymbol{w}$, terms in the state-space vector (4). The respective alteration in weight was incorporated as a multiplicative modifier for the $\boldsymbol{w}$ terms defined as β. Making the Q matrix take the form

$$Q = \text{daig}\left(\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & \beta & \beta & \beta \end{bmatrix}^T\right) \tag{14}$$

The aforementioned process was implemented for trials where the value of β was changed before enacting a simulated flight —with randomized initial conditions for $\boldsymbol{o}$ and $\boldsymbol{v}$ seen in Fig.1 and for 30 seconds. Where if a successful landing occurred for a respective β value, another subsequent trial with the same β value was run. Testing[‡] to determine the finalized β value was terminated when a successful landing was obtained five times in a row for a defined β value.

[†]For further explanation, refer to the 'Isolated_Ficalora_Veranga_DP2.ipnyb' notebook [3] where the respective "THEORY" cell outlines the process in **Subsection III.E** with annotations describing each relevant line of code.

[‡]Total flight data for individual trials can be found in [3]

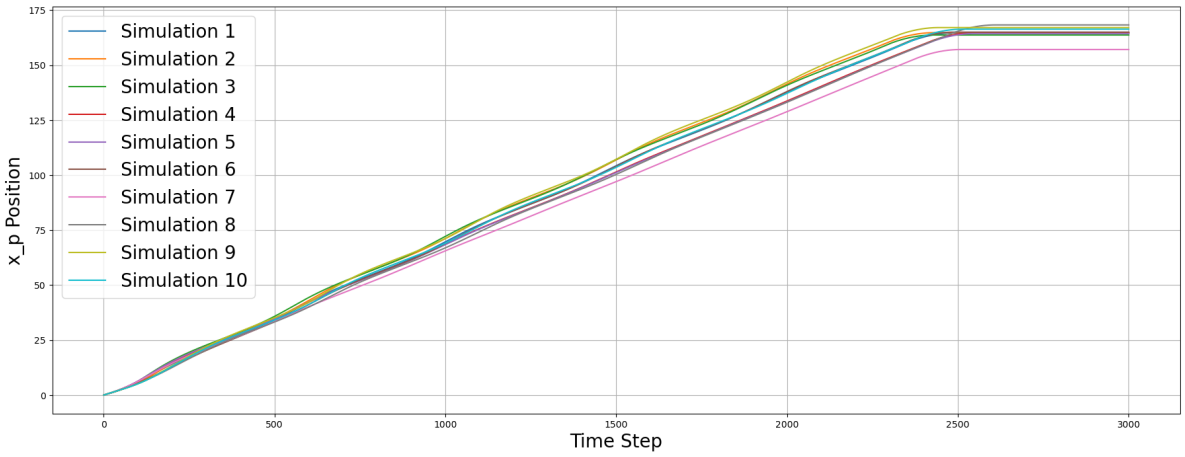
Fig. 1 Initial conditions and landing outcomes for respective flights determining the final weight matrix Q

Trial	β	$\boldsymbol{o}_e$	$\boldsymbol{v}_e$	Landing Judgment
1	1	(-0.12, 0.12, 0.13)	(4.63, 0.00, 0.00)	Success
2	1	(-3.03, 0.26, 1.81)	(4.74, 0.00, 0.00)	Fail
3	2	(0.03, 0.01, 0.00)	(5.33, 0.00, 0.00)	Success
4	2	(2.51, 0.68, 0.03)	(4.79, 0.00, 0.00)	Fail
5	3	(0.02, 0.01, 0.00)	(4.86, 0.00, 0.00)	Success
6	3	(0.07, 0.01, 0.00)	(5.21, 0.00, 0.00)	Success
7	3	(0.02, 0.01, 0.00)	(4.93, 0.00, 0.00)	Success
8	3	(-0.05, 0.01, 0.00)	(5.03, 0.00, 0.00)	Success
9	3	(0.00, 0.01, 0.00)	(4.68, 0.00, 0.00)	Success

V. Results and Discussion

Once we settled on a β value of 3, we continued our testing process. We conducted ten tests, of which all ten landed safely. Fig.2 provides "Flight-logs" of the ten successful flights. The figure shows glider x_positions with respect to time. We see that x_position increases steadily until the time step equals 2000. After this point, x_position levels out around 170. When we compare these results with the videos of our flights, we see that this leveling is when the gliders touch down on the runway and come to a stop.

Fig. 2 The glider's x positions with respect to time for ten flights



The initial conditions for each simulation are recorded in Fig.3. Even when the initial conditions were in the maximum range of possible values, the controller guided the glider back to the intended path and executed a safe landing.

The ability to control not only standard flights, but also gliders with undesirable initial conditions, demonstrates the resilience of our controller. Our goal at the beginning of this project was to land safely 75% of the time. With our controller, we were able to land every test flight safely.

VI. Conclusion

The glider control system met our design requirements and prioritized the pilot's safety. This success doesn't mean there isn't room for improvement. The next steps for this project, whether conducted by us or future developers, would include further Q matrix optimization and greater edge case handling. There may be edge cases outside of our current range that the controller cannot handle. We could identify the safe range of starting conditions that our controller can accurately react to. We could also continue to mitigate the cost factor of our controller. This would minimize the amount of correction required by the elevators.

Fig. 3 Initial conditions for ten successful flights

Simulation	o_e	v_e
1	(-0.45, -0.18, 0.18)	(5.27, 0.00, 0.00)
2	(0.41, -0.10, 0.11)	(5.03, 0.00, 0.00)
3	(0.49, -0.50, 0.00)	(5.47, 0.00, 0.00)
4	(-0.18, -0.30, -0.23)	(5.16, 0.00, 0.00)
5	(-0.16, -0.35, -0.26)	(5.23, 0.00, 0.00)
6	(0.51, 0.09, -0.30)	(5.24, 0.00, 0.00)
7	(0.17, -0.40, -0.29)	(4.90, 0.00, 0.00)
8	(0.13, 0.24, 0.21)	(5.27, 0.00, 0.00)
9	(-0.36, 0.19, -0.16)	(5.21, 0.00, 0.00)
10	(-0.20, 0.20, -0.25)	(4.97, 0.00, 0.00)

This project allowed us to implement new control theory techniques such as LQR controller design. By applying techniques learned in class to real world situations, the concepts are effectively reinforced.

Appendix

As the resident pilot for Cat Plane Co., I am always glad to work on their projects. They set strict design requirements and prioritize safety. This control system guided me to a safe landing in seven of our nine tests. I would consider the two unsuccessful landing to just be a little bumpy but I may just be a positive cat.

Acknowledgments

Authors Joshua Veranga and Luke Ficalora would like to thank Professor Bretl, the team of teaching assistant and course assistants for their continued guidance throughout this project.

References

[1] Bretl, T., "Design Project 2 (flying wing)," March 13, 2025. URL `https://tbretl.github.io/ae353-sp25/projects/02-zagi.html`.

[2] "Nelder Mead method," *wikipedia*, March 6, 2025. URL `https://en.wikipedia.org/wiki/Nelder%E2%80%93Mead_method`.

[3] Veranga, J., "SP25_AE353_Code/Design_Project_2," March 14, 2025. URL `https://github.com/muraSHUki/SP25_AE353_Code.git`.