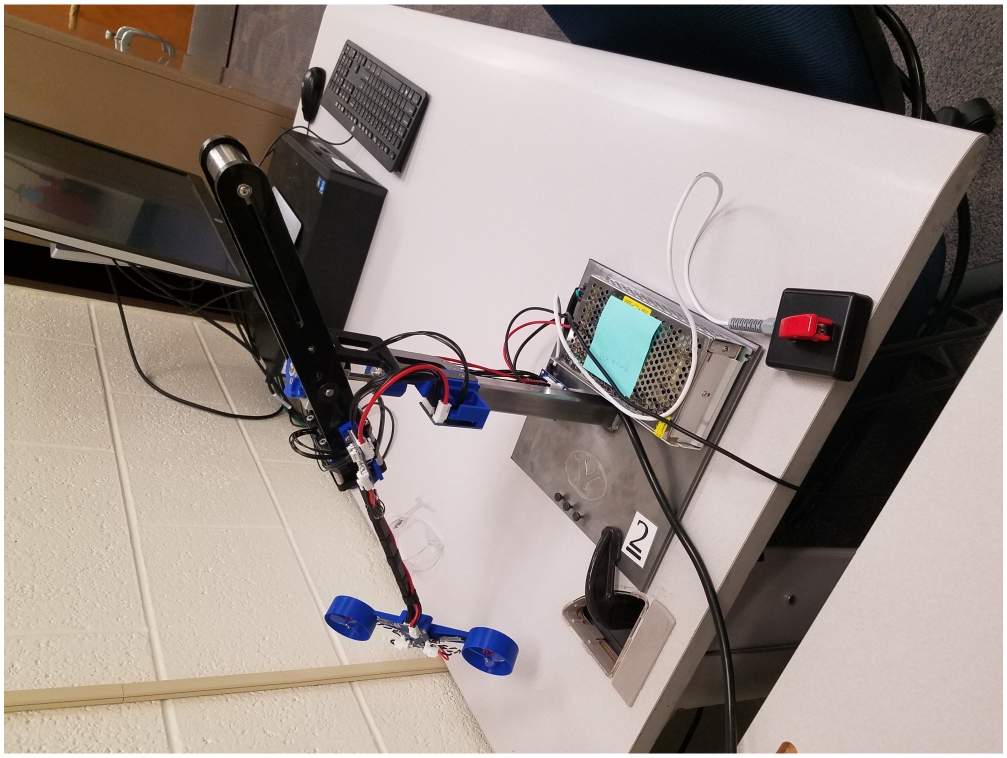
The Fastest, Most Realistic, Hummingbird

Tristan Russell

April 23, 2020



1 Abstract

This paper focuses on minimizing the settling time of the Hummingbird, seen in the title page, under PID control methods while keeping the design variables within feasible bounds. This was done first in simulation and the results were evaluated with the feasibility of several physical systems in mind. Genetic Algorithm was chosen as the optimization methods in order to take advantage of its ability to produce several local optimal values and thus provide several choices of acceptable design variables.

2 Introduction

The BYU ME En 431 Design of Control Systems course uses hardware named Hummingbirds to practice implementing their control algorithms. The Hummingbird hardware is essentially two rotors attached to a counterweight with the ability to pitch yaw and roll. The hardware is controlled by changing the output values of the two motors attached to the rotors. By manipulating these values, a controls engineer can manipulate the Hummingbird to their desire. In the class, it becomes apparent very quickly that each Hummingbird is unique and has different dynamics even though they are all built from the same design. This is due to imperfection in production and construction as well as wear and tear over time. The secondary goal of this project is to produce several sets of design variables that all produce an optimal settling time while giving the user several options. This gives the best chance that one set of design variables works with each Hummingbird.

In this case, the control algorithm being implemented is PID. The hummingbird has two independent values to solve for in order to achieve a controllable system. The first is Force which is solved for directly by analyzing the Theta (pitch) state. The second is Torque, which relies on both the Phi and Psi states (roll and yaw). A cascade structure has been chosen to represent the Torque relationship and simplify the equations. In this algorithm, the engineer solves for a proportional gain, an integrator gain and a derivative gain for each state. The exception is Phi, which doesn't need an integrator because it is the inner loop in the Torque equations. These gains are solved from the 8 design variables: Theta rise time, Phi rise time, Psi rise time, Theta dampening coefficient (zeta), Phi zeta, Psi zeta, Ki Theta, and Ki Psi. The gains are then put through equations 1 and 2 to produce an output Force and Torque which can then be assimilated into motor PWM values. PID is one of the easiest and most effective control algorithms. Not only is it simple to code, but it possesses the capability of eliminating steady state error even while a disturbance is present.

3 Methodology

3.1 Settling Time Function

As the nature of the Genetic Algorithm optimization method is gradient free, the first thing required to solve this minimization problem is a function using the design variables as inputs and outputting a settling time. This function can be found in Appendix A, however a simplified step by step process will be explained below.

First one needs to set initial conditions as well as what determines that the device has settled. An initial state and reference state for pitch and yaw need to be chosen. In this paper the initial states of zero zero will be used as well as reference states of 40 40. This means the settling time will measure how long it takes the device to move from 0 degrees pitch to 40 degrees pitch and from 0 degrees yaw to 40 degrees yaw. The device will be considered settled once the difference between the reference state and actual state is less than .0001 for both Psi and Theta.

The equations for the linearized Force and Torque output to the motors are below in equations 1 and 2. Ki is a design variable and Kp and Kd are solved below using other design variables in equation 5, 6, 7, 8, 9, and 10 . The derivatives are calculated using equation 3 and the integrators are calculated using equation 4. The notion of prime is used in the equations to signify the derivative of that specific state.

$$F = (Kp_\theta * (\theta_r - \theta)) + (Ki_\theta * Integrator_\theta) - (Kd_\theta * \theta') \tag{1}$$

$$T = (Kp_{\phi} * (\phi_r - \phi)) - (Kd_{\phi} * \phi') \tag{2}$$

These equations require solving for the derivatives of each of the states. While the simulation of PID can measure velocity states, the hardware on the hummingbird can't. The derivatives must then be calculated using the Dirty Derivative. The general equation for the Dirty Derivative is as follows:

$$Derivative(n) = (2\sigma - dt)/(2\sigma + dt) * (Derivative(n-1)) + (2/(2\sigma + dt) * (State(n) - State(n-1)) \tag{3}$$

In this equation, Sigma is small and generally .05. The State represents whatever state variable we are looking at and derivative refers to the derivative of that state variable. dt is whatever time step is chosen beforehand by the user or program being used. This equation is used to solve for the derivatives of the three states.

With the derivatives found, one must solve for the value of the two integrators. The integrator is the I in PID control. The integrator gains are design variables, but the actual value of the integrator itself is calculated by using equation 2. In equation 4, dt is again the time step and error represents the difference between the reference state and the actual state. Integrators will often present problems in PID control if the error is too large, so it is necessary to add a form of integrator anti-windup. According to Vandoren, "Even after the error has been eliminated, the controller will continue to generate an output based on the history of errors that have been accumulating in the controller's integrator. The process variable may then overshoot the setpoint, causing an error in the opposite direction." (4). In these calculations, an "if" statement based on the error of the two states being integrated is being used. Specifically, only integrate if the absolute value of Theta prime is less than .03 and if then absolute value of Psi prime is less than 1. This protects against integrator windup and system instability.

$$Integrator(n) = Integrator(n-1) + ((dt/2) * (error(n) - error(n-1))) \tag{4}$$

Kp represents the proportional gain value and the P of PID control. It is solved by using values from the transfer function of each state. The equations for the Kp of each state are listed below in equations 5, 6 and 7. The unrecognizable variables are constants that are part of each transfer function and have been previously solved. These constants are defined in Appendix A. For additional information on how the Kp values are derived from a transfer function please see the textbook Introduction to Feedback Control (2).

$$Kp_{\theta} = ((2.2/tr_{\theta})^2)/b_{\theta} \tag{5}$$

$$Kp_{\psi} = ((2.2/tr_{\psi})^2)/b_{\psi} \tag{6}$$

$$Kp_{\phi} = ((2.2/tr_{\phi})^2) * J1_x \tag{7}$$

Kd represents the derivative gain value and the D of PID control. It is also solved by using values from the transfer function of each state. The equations for the Kd of each state are listed below in equations 8, 9, and 10. The unrecognizable variables are constants that are part of each transfer function and have been previously solved. These constants are defined in Appendix A. For additional information on how the Kd values are derived from a transfer function please see the textbook Introduction to Feedback Control (2).

$$Kd_{\theta} = (2 * (2.2/tr_{\theta}) * \zeta_{\theta})/b_{\theta} \tag{8}$$

$$Kd_{\psi} = (2 * (2.2/tr_{\psi}) * \zeta_{\psi})/b_{\psi} \tag{9}$$

$$Kd_{\phi} = 2 * (2.2/tr_{\phi}) * \zeta_{\phi} * J1_x \tag{10}$$

The last unsolved variable in equations 1 and 2 is the reference state of Phi. Yaw is solved by looking at the transfer functions and controllers of Psi and Phi in a cascade block format. In this format, the Phi part of the system comprises the inner loop and is run about 10x the speed of the outer Psi loop. Since we know the Psi reference state and the Psi current state, as well as its transfer function and Kp, Kd, and Ki values, we can work the loop from the inside out. Solving for the Phi reference state this way, results in equation 11 below.

$$\phi_r = Kp_\psi * (\psi_r - \psi) + (Integrator_\psi * Ki_\psi) - (Kd_\psi * \psi') \tag{11}$$

Everything necessary to solve for equations 1 and 2 has now been shown. Force and Torque are not the end goal however, in the end a total settling time is desired. To solve for total settling time, the Force and Torque need to be applied to the states over a time step of in order to move them to a new state. This process is done continuously until the requirements needed to satisfy settlement of the device are met. The equations for the new states achieved by advancing through the time step are show below in equations 12, 13, and 14.

$$\theta(n) = (\theta'' * (dt^2)/2) + (\theta' * dt) + \theta(n-1) \tag{12}$$

$$\psi(n) = (\psi'' * (dt^2)/2) + (\psi' * dt) + \psi(n-1) \tag{13}$$

$$\phi(n) = (\phi'' * (dt^2)/2) + (\phi' * dt) + \phi(n-1) \tag{14}$$

Each of these equations requires one additional unknown, the second derivative of each state. The equations for each second derivative are listed below in equations 15, 16, and 17. In these equations, the J variables represent moments of inertia and their values can be found in Appendix A. Fe represents the equilibrium force and its equation and value can also be found in Appendix A. The full derivation of each of the following equations can be found in the Hummingbird Lab Manual (1).

$$\theta'' = b_\theta * F \tag{15}$$

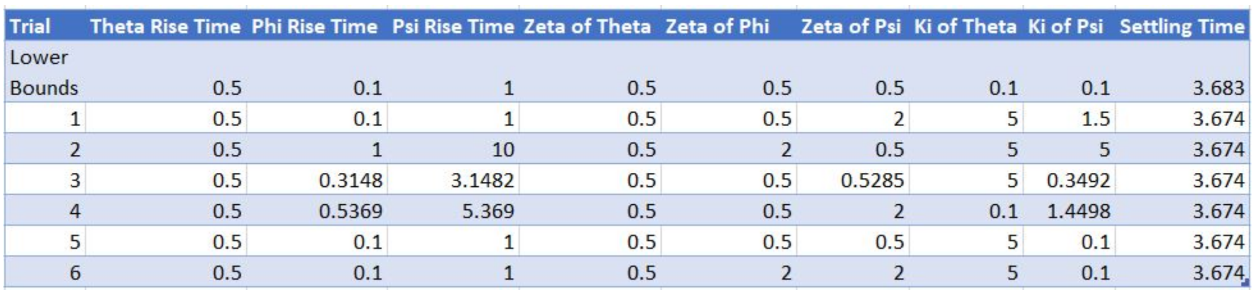
$$\psi'' = (.355 * F_e * \phi)/(Jt + J1_z) \tag{16}$$

$$\phi'' = T/J1_x \tag{17}$$

The culmination of all these equations allows for the calculation of the new pitch and yaw states after a defined time step. In order to calculate the total settling time, the number of time steps required to reach a settling time needs to be counted and summed. This summation results in the settling time of the Hummingbird from 0 degrees pitch to 40 degrees pitch and from 0 degrees yaw to 40 degrees yaw under PID control.

3.2 Genetic Algorithm

For the Genetic Algorithm method, an initial population of 10 design variable sets was created randomly. This was done by using equation 6.13 in the Multidisciplinary Design Optimization book (3). This entails using real numbers instead of converting to binary. Once the initial population was created, the "genes" were put through a tournament selection. Population 1 was compared to 2 and 3 to 4 and so on. The populations that resulted in the least total time won their "match" and continued to the next phase of crossover. In the crossover phase, the remaining populations were bred together to produce 10 populations once again. This was done using equation 6.14 in the same book. Each population was bred once with a few populations double breeding based on random assignment. Once there were 10 populations, they went through a stage of mutation based on equation 6.16 in the book. a delta i of .05 was used. Finally, the values were checked against constraints and if they were violated, they were rounded up or down to the appropriate constraint. This process was performed until a new minimum time wasn't determined for 10 generations in a row.



Trial	Theta Rise Time	Phi Rise Time	Psi Rise Time	Zeta of Theta	Zeta of Phi	Zeta of Psi	Ki of Theta	Ki of Psi	Settling Time
Lower Bounds	0.5	0.1	1	0.5	0.5	0.5	0.1	0.1	3.683
1	0.5	0.1	1	0.5	0.5	2	5	1.5	3.674
2	0.5	1	10	0.5	2	0.5	5	5	3.674
3	0.5	0.3148	3.1482	0.5	0.5	0.5285	5	0.3492	3.674
4	0.5	0.5369	5.369	0.5	0.5	2	0.1	1.4498	3.674
5	0.5	0.1	1	0.5	0.5	0.5	5	0.1	3.674
6	0.5	0.1	1	0.5	2	2	5	0.1	3.674

Table 1: In this table, the Lower Bounds refer to the design variables at their lowest numerical value and each trial after that was the Genetic Algorithm run until a new set of optimal design variables was produced.

4 Results

The Genetic Algorithm method was able to find several different sets of design variables satisfying the constraints and bounds limitations while producing a minimum settling time value. Several of the sets involve all the variables residing on either the upper or lower bound. This is to be expected, however these sets will also probably be the first to fail if a systems dynamics are different than in theory. For that reason it is important that several options are available. Unfortunately, due to the Covid-19 Pandemic and measures taken to prevent against its spread, testing of the calculated design variables didnt take place. However, the equations and formulas used to calculate them are accurate and the multiple options suggest that the likelihood of at least one of the design sets working is very high.

5 Discussion

It is clear that several options came to light upon the completion of these experiments. This is contrary to Xu's warning when he said, "The design of the PID parameters is rather ad-hoc, so it is difficult to achieve optimal performance." (5). The Genetic Algorithm manged to find several variations of design variables that produced an optimal settling time. This is very fortunate as it allows the user several options when writing their PID control algorithm. As hardware is always different, and each set reacts differently due to error in production and construction, having several options available is ideal. In the case that one option doesn't work for one set of hardware, there are several other choices to ensure that the user has the highest possible chance of writing a time minimizing PID controller.

6 Appendix A

```
function [Time]=Calculate(x)
sigma=.05;
m1=.108862;
m2=.4717;
m3=.1905;
l1=.247;
l2=-.039;
J1x=.000189;
J1z=.001894;
J2z=.003416;
l3x=-.039;
l3y=-.007;
dt=.001;
Fe=(m1*l1 + m2*l2)*9.81/.355;
Jt=m1*l1^2+ m2*l2^2 + J2z + m3*(l3x^2 + l3y^2);
```

```
trtheta=x(1);
trphi=x(2);
trpsi=x(3);
ztheta=x(4);
zphi=x(5);
zpsi=x(6);
kitheta=x(7);
kipsi=x(8);
Time=0;
wnt=2.2/trtheta;
wnphi=2.2/trphi;
wnpsi=2.2/trpsi;
btheta=28.21;
bpsi=.355*Fe/(Jt+J1z);

kpt=wnt^2/btheta;
kdt=2*wnt*ztheta/btheta;

kppsi=wnpsi^2/bpsi;
kdpsi=2*zpsi*wnpsi/bpsi;

kpphi=wnphi^2*J1x;
kdphi=2*zphi*wnphi*J1x;
%Reference Values
thetar=40;
psir=40;
%initial Values
theta=0;
psi=0;
phi=0;
thetaprev=0;
thetadprev=0;
psiprev=0;
psidprev=0;
phiprev=0;
phidprev=0;
thetaerror=0;
psierror=0;
thetaerrorprev=0;
psierrorprev=0;
Int_theta=0;
Int_psi=0;
while (psir-psi) >= .0001 && (thetar-theta) >=.0001
%derivative Values
thetaerror=thetar-theta;
psierror=psir-psi;
thetad=((2*sigma-dt)/(2*sigma+dt))*thetadprev+(2/(2*sigma+dt))*(theta-thetaprev);
psid=((2*sigma-dt)/(2*sigma+dt))*psidprev+(2/(2*sigma+dt))*(psi-psiprev);
phid=((2*sigma-dt)/(2*sigma+dt))*phidprev+(2/(2*sigma+dt))*(phi-phiprev);
thetaprev=theta;
thetadprev=thetad;
psiprev=psi;
psidprev=psid;
```

```
phiprev=phi;
phidprev=phid;

if abs(thetad)<.03
    Int_theta=Int_theta+((dt/2)*(thetaerror-thetaerrorprev));
    thetaerrorprev=thetaerror;
end
if abs(psid)<1
    Int_psi=Int_psi+((dt/2)*(psierror-psierrorprev));
end


%Feedback Linearized Force
%Ffl=(m1*l1 + m2*l2)*9.81*cos(theta)/.355;
%Linearized Force
Ftilda=kpt*(thetar-theta)+(kitheta*Int_theta)-(kdt*thetad);
%Calculating phir
phir=kppsi*(psir-psi)+(Int_psi*kipsi)-(kdpsi*psid);
%Calculating Torque
T=kpphi*(phir-phi)-(kdphi*phid);
%Total Force
%F=Ffl+Ftilda;


thetaddot=btheta*Ftilda;
theta=thetaddot*dt^2/2+ thetad*dt +theta;

psiddot=.355*Fe*phi/(Jt+J1z);
psi=psiddot*dt^2/2 +psid*dt +psi;

phiddot=T/J1x;
phi=phiddot*dt^2/2 + phid*dt + phi;

Time=Time+dt;
end
Time;
end
```

7 References

1. Beard, R., McLain, T., Morris, D., and Peterson, M., 2020. *Hummingbird Manual*

2. Beard, R., McLain, T., and Peterson, C., 2016. *Introduction to Feedback Control using Design Studies*

3. Martins, J. R. R. A., Ning, A., and Hicken, J., 2017. *Multidiciplinary Design Optimization.*

4. VanDoren, Vance J. (2003), PID: Still the One. Control Engineering; Barrington Vol. 50, Iss. 10, (Oct 2003): 32-37.

5. Xu, Jingwei. (2006), Fuzzy PID control through optimization: A new method for PID control; Marquette University, ProQuest Dissertations Publishing, 2006. 3231315.