Jonathan Schwartz
Rohan Daran
MAE 6245/4245

A Survey of Common, Closed-Loop Feedback Control Methods

A control system implements an appropriate actuating signal to provide a desired output. These systems are used ubiquitously in both man-made applications and biology. From maintaining the speed of an automobile to maintaining homeostasis of a biological system, control systems methods allow for any system to reach a desired state. Modern closed-loop feedback control systems were introduced in the early 20th century. These control methods utilize a portion of the systems output to regulate the systems input and create a response. In feedback control, the output of the system is fed back into the system and compared to the desired output to identify the system's error. Linear feedback methods apply inputs to the system that are the result of multiplying one or several state variables by a constant. These constants are often referred to as gains. While the technicalities of each linear feedback control method differ, the principle of the methods are the same: to measure, monitor, and control a process to reach a desired output of a system.

The simplest linear feedback control method is a **proportional feedback controller**. These controllers work by applying an input to the system that is proportional to the difference between the current state and the desired state. In Figure 1, the desired state is R(s) and the current state is C(s). E(s) is the difference of these two values, and is multiplied by the proportionality constant K_p to become U(s). This is the input applied to the transfer function representing the system, G(s). The result is a new current state C(s) which is then used in conjunction with the desired state of R(s) to repeat this process.

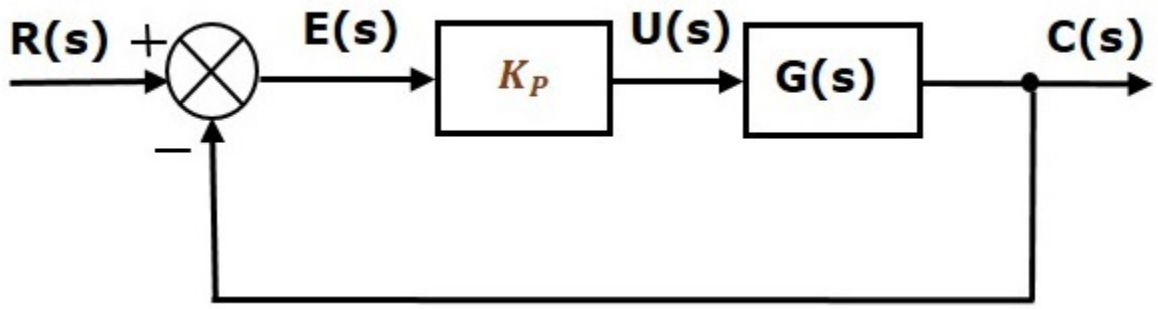

Figure 1: Block diagram of Proportional Controller

The main challenge when implementing a proportional controller is deciding what value to use for the gain K_P. There are ways to analyze a system in order to determine an appropriate gain value, such as the Routh-Hurwitz Stability Criterion. That particular method will provide a range of all the potential gains that will keep the system stable. There are also experimental methods for determining, such as the Ziegler-Nichols method. This method states that an optimal proportional gain can be determined by first setting the gain to 0, and then increasing the gain value until the system's oscillations are stable and consistent [1]. The largest gain that exhibits this type of oscillation is called the "ultimate gain" [1]. A depiction of a system with this type of

gain can be seen in Figure 2. The best stability comes from a proportionality constant that is half the size of the ultimate gain, as can be seen in Figure 3. Note that this rule stands only for controllers that just have a proportional term.

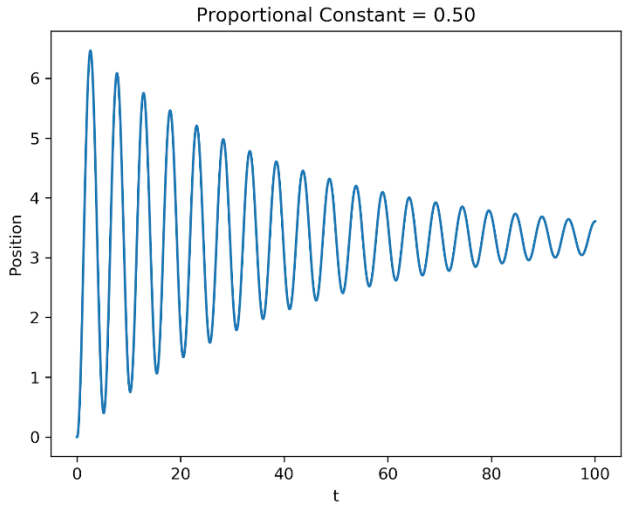

Figure 2: Plot of a system with the Ultimate Gain

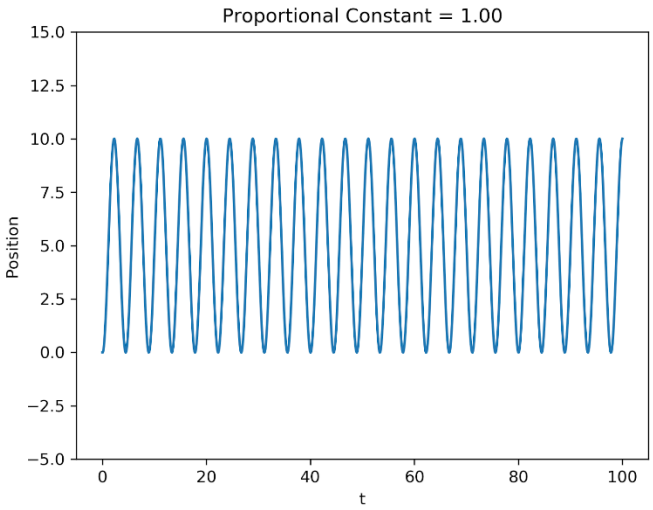

Figure 3: Plot of a system with

The ultimate gain value for a system depends on its dynamics, so it can be extremely difficult to guess for more complex systems. Figures 2 and 3 were generated from a basic spring-mass-damper system, which is why its ultimate gain was as simple as 1. If faster convergence is desired, the proportional gain can be decreased at the cost of greater steady state error. The inverse is also true; steady state error can be decreased by increasing the proportional gain, but the time to convergence will be greater. To improve both of these system properties, the advent of a derivative gain and an integral gain will be of much more assistance than simply adjusting the proportional gain's value.

Proportional Integral Derivative controllers (PID) are a more advanced version of a proportional controller, and are the most commonly used systems in feedback control. PID controllers are characterized by their three control gains: their proportional gain, integral gain, and derivative gain. By applying corrections based on proportional, integral, and derivative terms, a PID controller can adjust the output of a system to compensate for error and match a set point.

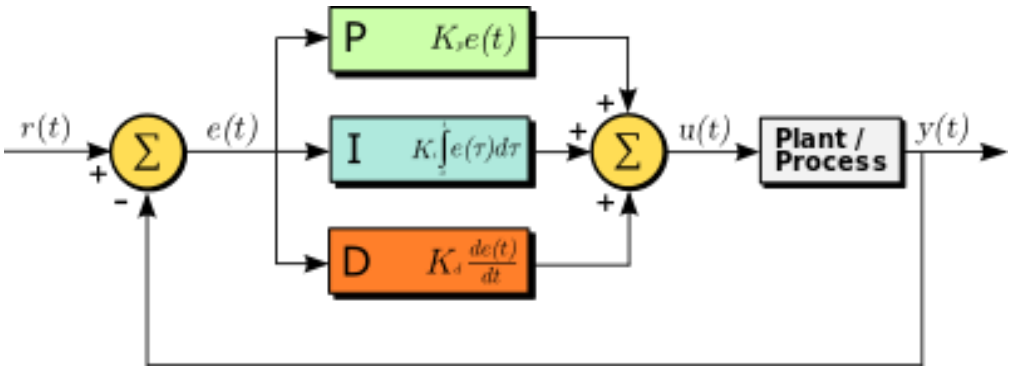

Figure 5: Block Diagram of a PID Controller

PID controllers have the ability to optimize time to convergence, minimize system overshoot, and eliminate steady state error of a system. PID controllers do this by utilizing integral gains to keep a running total of past inputs to eliminate steady state error. In addition to utilizing integral gains, the controllers also utilize derivative gains to predict future error and reduce system overshoot.

Tuning PID controllers revolves around modifying the gains of the state values until the desired behavior of the system is reached. This can be achieved by trial and error based on an intuitive knowledge of how the respective gains affect the system as a whole.

With the proportional loop of the controller, increasing the gain will increase the rise time of the system, but decrease the time to convergence. Proportional controllers may have steady state error, to eliminate this we can utilize the integrator loop, however we must be careful of integrator windup. PI controllers function well with a decent rise time and steady state value, however they fall behind in their time to convergence. To decrease this time to convergence, we can add the derivative loop and its respective gain value. Increasing the gain value of the derivative loop serves to improve transient response by predicting future error. This can greatly reduce the time to convergence of the system by damping the system and reducing overshoot. When using the derivative loop of PID control, we must be careful to understand the point of critical damping to avoid over damping the system. This can result in increasing the time to convergence by completely eliminating overshoot. The effects of adding the I and D terms to a proportional controller can be seen in Figure 6 below.

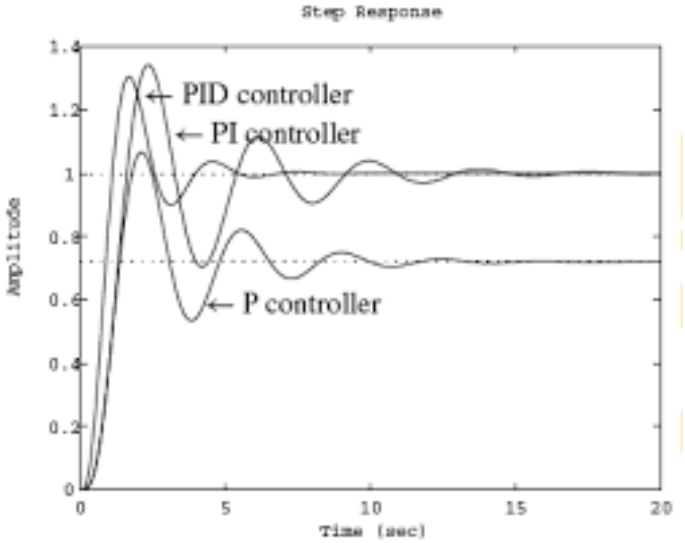

Figure 6: Effects of I and D terms on system output

As manually tweaking PID controls can be time consuming to its nature of trial and error, we can also utilize heuristic techniques to establish initial gains of a system and then manually tweak those gains from there. The Ziegler Nichols and Cohen-Coon methods are popular methods to establish these initial gains of a system.

The Ziegler Nichols method is an experimental method used to determine the optimal proportional gain of a system and then to extrapolate the integral, and derivative gains of the system. As was illustrated before, the first step of the Ziegler Nichols method is to determine the

ultimate gain of the system. Once found, this ultimate gain, along with the oscillation period, can be used to set the P, I, and D gains. The Ziegler Nichols method provides PID controllers with the optimal disturbance rejection, however it also yields an aggressive gain and overshoot. In addition, as the Ziegler Nichols method requires the system to be pushed almost to the point of instability which can be dangerous in certain systems.

The Cohen-Coon method is similar to the Ziegler Nichols method however the system provides a faster rise time. As opposed to the Ziegler Nichols method, the Cohen-Coon method works well on self regulating processes that require fast adjustments while being less prone to oscillations. To establish initial gains in the Cohen Coon method we must first wait for the system to reach steady state in manual mode. Next, we must introduce a step input in the system to analyze how long it takes for the system to reach steady state once more. To establish the initial gains we must then establish when the system reaches half of its steady state value (t2) and 63.2% of this steady state value (t3). Using these determined time values, we can approximate a first order process with a delay of t_{DEL} units from when the input step was introduced. This process is illustrated in Figure 7 and 8. A comparison of the results of this tuning method compared to those of the Ziegler Nichols method can be seen in Figure 9.

$$t_1 = (t_2 - \ln(2)\, t_3)/(1 - \ln(2))$$

$$t = t_3 - t_1$$

$$t_{DEL} = t_1 - t_0$$

$$K = B/A$$

Figure 7: Identification of process parameters with Cohen-Coon Method

	K_c	t_I	t_D
P	$\frac{1}{K \cdot r}\left(1 + \frac{r}{3}\right)$		
PI	$\frac{1}{K \cdot r}\left(0.9 + \frac{r}{12}\right)$	$\tau_{DEL} \cdot \frac{30 + 3 \cdot r}{9 + 20 \cdot r}$	
PID	$\frac{1}{K \cdot r}\left(\frac{4}{3} + \frac{r}{4}\right)$	$\tau_{DEL} \cdot \frac{32 + 6 \cdot r}{13 + 8 \cdot r}$	$\tau_{DEL} \cdot \frac{4}{11 + 2 \cdot r}$

where $r = \frac{\tau_{DEL}}{\tau}$

Figure 8: Identification of controller parameters with Cohen-Coon Method

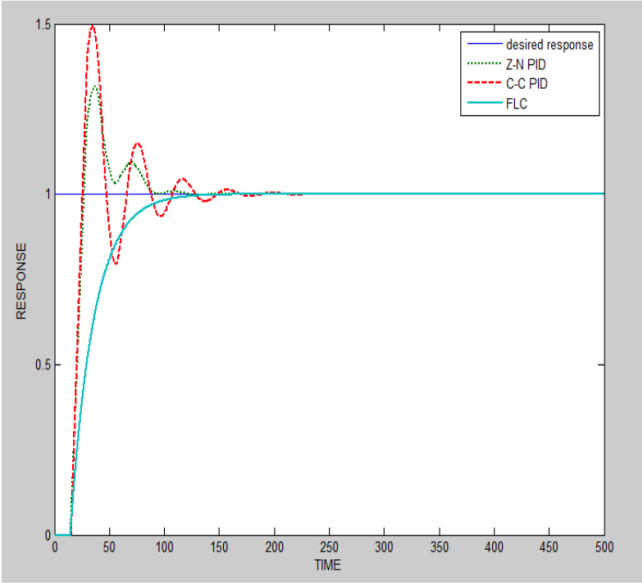

Figure 9: Comparison rise time and time to convergence of Ziegler-Nichols and Cohen Coon methods

When using PID control, we must be especially careful when utilizing our integral gain to reduce the steady state error of the system to avoid the phenomenon known as integral windup. Integral windup is caused when the actuator reaches a point of saturation, when it can no longer abide by the command of the controller. A motor's maximum rotational speed or a battery's maximum capacity are examples of saturation limits. When a controller is unaware of the saturation limit of an actuator, it will continue to increase its response to eliminate steady state error. This may not be an issue during the "windup" of the actuator, as it has already reached its limit, but it can become an issue when conditions change and the integral response must "unwind." During the "unwind" process the integral begins to decrease its response as the steady state error changes signs. As the integral has already exceeded the saturation point of the actuator, however, it must decrease to below that saturation point before implementing a change in the actuator. This can dramatically increase the overshoot in a system and lead to an increased time to convergence. This process can be seen in Figure 10.

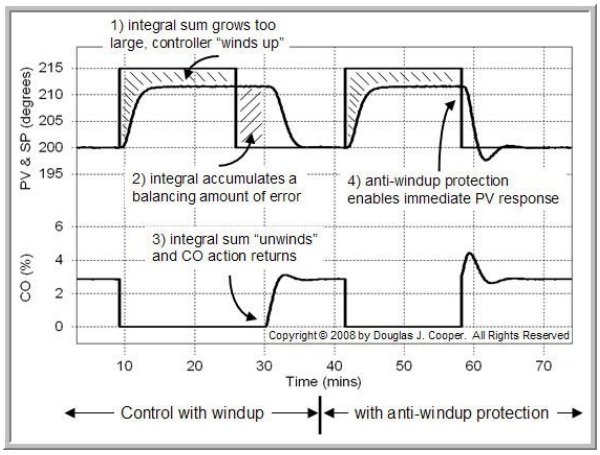

Figure 10: Depiction of Integral Windup and Unwinding

To avoid integral windup and minimize the time it takes to reverse a command, we can utilize a method known as clamping. Clamping limits the output of an actuator beyond its saturation point and disables the integral loop when integral windup is occurring. To do this we perform 2 checks, the first of which checks the value of the PID controller before and after a saturation check. If the values of the controller are equal to each other, the system is not saturating, if the values are not equal, the system is saturating. The second check compares the signs of the controller output and error. If both of these are positive the integrator is adding to the output to make it more positive. If both of the signs are negative, the integrator is subtracting from the output to make it more negative. When comparing these two checks, we can see when the system is saturating and when the integrator is contributing to windup, we can then disable the integral loop when this occurs.

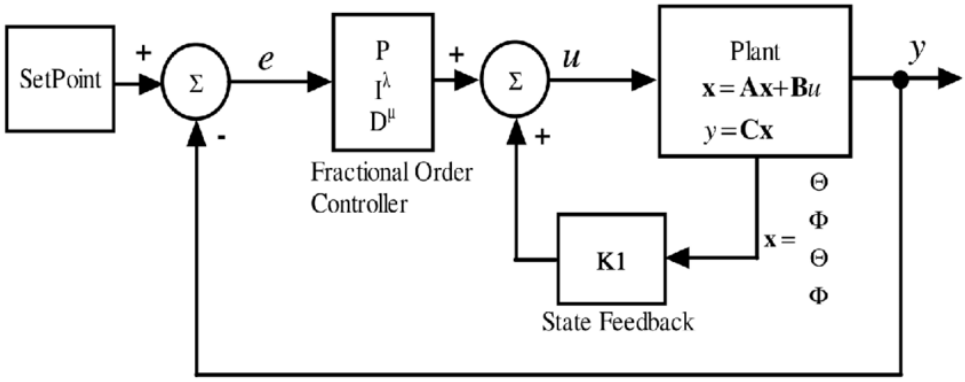

Figure 11: Block Diagram of a Fractional Order Controller

One method that provides improved accuracy over a typical PID controller while maintaining the same structure is a **Fractional Order PID controller (FOPID)**. A typical block diagram for this kind of controller can be seen in Figure 11. The only difference between it and a PID controller is the addition of the exponents λ and μ on the Integral and Derivative terms, respectively. This inevitably increases the controller's potential accuracy, as there are "more knobs to tune" than with a typical PID controller [2]. In a typical PID controller, λ and μ would both be 1. In an FOC, λ and μ can be any real, positive numbers. A popular method for determining the values of λ and μ is to set a threshold for a certain type of system error and to adjust the system parameters until that threshold is met. Commonly used types of error are Integral Squared Error (ISE, defined as: $J_0 = \int_0^\infty [e(t)]^2\, dt$), Integral Squared Time Error (ISTE, defined as: $J_1 = \int_0^\infty [t * e(t)]^2\, dt$), and Integral Squared Time Squared Error (IST^{2}E, defined as: $J_2 = \int_0^\infty [t^2 * e(t)]^2\, dt$). Some methods use absolute error instead of error squared, such as Integral of Absolute Error (IAE, defined as: $J_{AE} = \int_0^\infty |e(t)|^2\, dt$.

When the error function is a polynomial E(s) = c(s) / d(s), expressions can be derived for the various types of error using the constants in the c(s) and d(s) polynomials [3]. These constants can then be represented in terms of the system's state dynamics and the gains from the controller. By then differentiating the error expression in terms of one of the variable gains, the gain can be set to a value where this derivative is zero. Another method is to view setting the gain values as an optimization problem, where you want to minimize $J_n(x)$, where $x=[K_P, K_I, K_D]$ [3]. After these gain values are determined, another optimization problem is performed. This

time, $J_a(x)$ is minimized when $x=[K_P, K_I, K_D, \lambda, \mu]$. Initial values for the K gains are the values determined from the first optimization problem, and λ and μ are initially set to 1.

These are the two most common approaches to tuning a FOPID controller. Both are tuned using the entire history of the device, either analytically by integrating error over time or experimentally by adjusting the controller's variable gains based on performance. The fact that these methods take the system's entire history into account, combined with the addition of two more tuning parameters, makes the FOPID controller robust for a wide variety of applications.

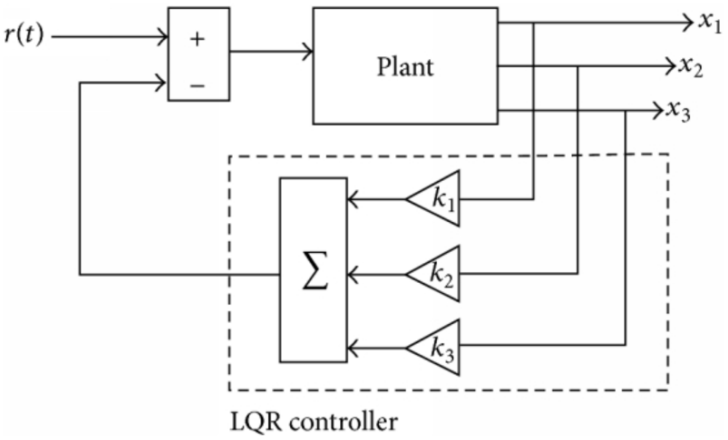

Figure 12: Block Diagram of an LQR Controller

A **linear quadratic regulator (LQR)** works similarly to a PID controller in that gains are determined for multiple state values, and the sum of these gains multiplied by their respective state values is the input to the system. This can be seen by comparing Figures 5 and 12. The gains in an LQR controller are generated by solving a Riccati Equation that describes the system's cost. The cost function is defined like so:

$$J(U) = \sum_{\tau=0}^{N-1} \left(x_\tau^T Q x_\tau + u_\tau^T R u_\tau\right) + x_N^T Q_f x_N$$

Where Q is a matrix that determines how much the system can vary from the desired value of each state variable, and R is a matrix that indicates how expensive it is for the system to actuate itself in order to adjust each state variable. Both of these take the general form of an identity matrix, with its dimensions being the same length as the state array. If there is no preference on how much state variables vary from their desired values, then an identity matrix can be adequate for the Q matrix. Realistically, nearly all systems have some preference. When that is the case, the values along the diagonal can be adjusted to reflect how much variation from nominal values is tolerated. For example, the Q matrix for a system with 5 state variables would look something like this:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 100000 & 0 \\ 0 & 0 & 0 & 0 & 10 \end{bmatrix},$$

There are a few approaches to finding ideal Q matrix values, with some as loose as setting Q = I and adjusting the diagonal values until system performance is acceptable. A more calculated approach is to decide what the largest acceptable deviation for each state variable, and to select q_i such that: $q_i * x_i^2 = 1$ where x_i is the largest acceptable deviation of the state variable and q_i is the value on the Q matrix diagonal that corresponds with it [4].

Something important to note is that while some state variables are operating about a nominal value, others are approaching a nominal value from relatively far away. The Q matrix value for these types of state variables correlates with how quickly the value converges. This can be seen in Figures 13 and 14, which are output from a simple simulation of a flying robot. In this situation, X is a state variable trying to converge to a point and theta and omega are trying to stay within a certain range about a nominal value of 0. This is because the robot is only stable when theta and omega are close to 0. Otherwise, its state will slowly diverge and the robot will crash without a controller correcting error. A picture depicting the state variables in relation to this robot is included in the appendix for reference.

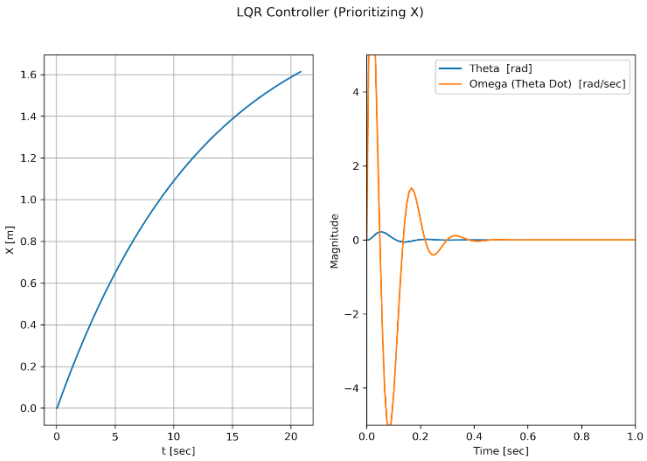

Figure 13: Q matrix prioritizing X value

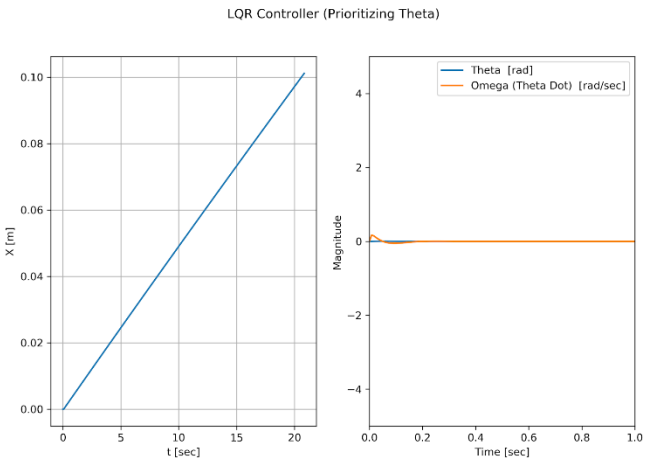

Figure 14: Q Matrix Prioritizing Theta Value

As can be seen in the plots, prioritizing one state variable too much in relation to others can lead to poor system performance. When X is prioritized, theta deviates more significantly past 0 radians, which means the robot is at a greater risk of becoming unstable and falling to the ground. On the other hand, prioritizing theta and omega too much leads to very slow convergence for the x state variable. Moving at such slow speeds is also not ideal because tasks

will take much longer thus much more energy will be expended keeping the robot in the air throughout the duration of the task.

The R matrix works similar to the Q matrix in that it takes the general form of an identity matrix, and the values along the diagonal can be adjusted. The R matrix values indicate how expensive it is to adjust each state variable in practice. Similar to the Q matrix, these are all relative to the other diagonal values so it is best to pick an arbitrary constraint to define the values about. For instance, Ric(xi)2=1 could work where $c(x_i)$ is the energy cost to adjust state variable x_i by 5%. Note that the percentage 5%, as well as the constant the expression is set equal to (in this case 1) are entirely arbitrary and there is a wide range of constants that can be used. It is worth noting that the system being controlled will affect what constants can be used. For instance, if one state variable cannot be adjusted past 10% of its nominal value, it would not make sense to determine R matrix values with how much it costs to adjust them by 50%.

The larger an R value is, the more expensive it is to actuate that state variable and the less the LQR controller will adjust it when possible. A simpler method for the R matrix that is adequate for simulations and low-cost systems is to set R = ρ*I. This means all state variables cost the same to adjust, and for systems where cost is not of great concern this is often adequate. The effect changing ρ has on the system can be seen in Figures 15 and 16. In those plots, R in the titles indicates the value of ρ. In Figure 15 it is very small, meaning the system is unconcerned with the cost of adjusting its variables. In Figure 16, ρ is very large, which means the system is careful to only adjust state variables when absolutely necessary. As you can see, a small ρ value allows the systems to approach desired values relatively quickly. A large ρ value means it will take much longer for state variables to converge to desired values, but there is much less unnecessary energy expended on state variables. This difference in expenditure can be best seen in the difference in omega values between Figures 15 and 16.

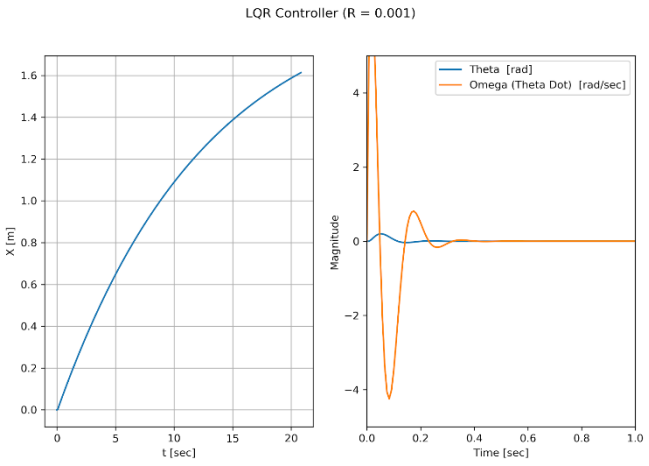

Figure 15: Small ρ value

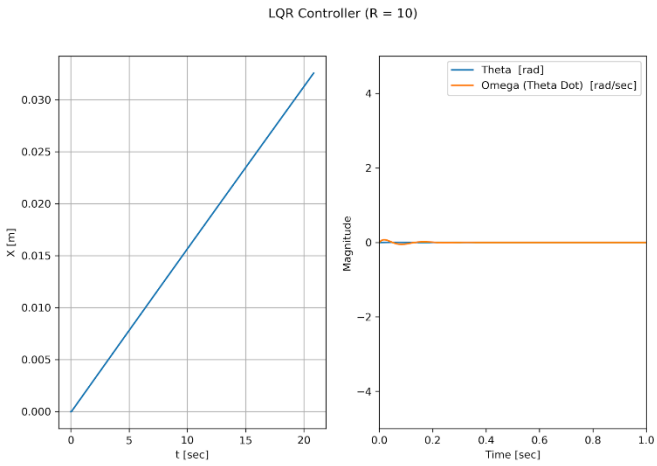

Figure 16: Large ρ value

One of the main drawbacks of an LQR controller is it assumes that all state readings are totally accurate. This is not the case in any real-world application, as noise will inevitably taint signals. In some cases, particularly those where high accuracy performance is not required, noise can be ignored. When that is not the case, a more robust control method like a **linear quadratic**

gaussian controller (LQG) should be used [5]. The difference between an LQR and an LQG controller is the addition of a **Kalman filter**.

A Kalman filter works by calculating a weighted average of the predicted state of the system and the measured state. The predicted state is determined using the system's plant dynamics, most recent input, and prior state. The weights for the weighted average are determined by how much uncertainty there is surrounding each measurement. This is done using a covariance matrix for all of the predicted and measured state values being used for the weighted average [6]. A covariance matrix of dimensions n x n provides the variation between each pair of variables in an n-length vector. For example, the covariance matrix of 3 variables would look something like this:

$$\sum = \begin{bmatrix} 1 & 0{\cdot}63 & 0{\cdot}4 \\ 0{\cdot}63 & 1 & 0{\cdot}35 \\ 0{\cdot}4 & 0{\cdot}35 & 1 \end{bmatrix}.$$

The predicted and measured state values with the largest covariance values are given larger weights, as they are likely less affected by noise.

The block diagram for an LQG controller can be seen in Figure 17. Upon juxtaposition with the block diagram for an LQR controller in Figure 12, it becomes clear that the only difference between the two is the addition of a Kalman Filter. The Filter takes in the input applied to the system **u** and the observed system output **y**, and uses these to generate an accurate estimate of the state vector **x**. This is passed to the LQR controller, which uses the estimated state to calculate gains to scale the system's inputs upon its next iteration.

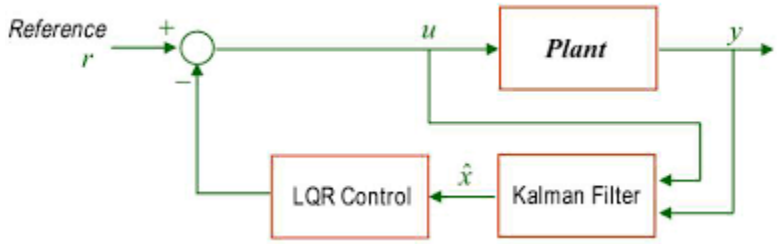

Figure 17: Block Diagram of an LQG Controller

The effect of a Kalman filter is clearing out most of the noise in a signal. This can be seen in Figure 10, where the estimated signal is nearly identical to the real-time quantity. The Kalman filter is passed the measurement, which is actual theta plus sensor noise. One issue with this filtering method is that it only uses the estimation of the last time step in order to generate its new estimate. This can cause the estimation to deviate if it is passed a value that is significantly skewed by noise. This is usually not a problem, as a Kalman filter is robust enough to handle all but the largest deviations in stride, as shown in Figure 10. Most sensor noise will not severely skew the sensor's readings, so an LQG controller is adequate for most applications.

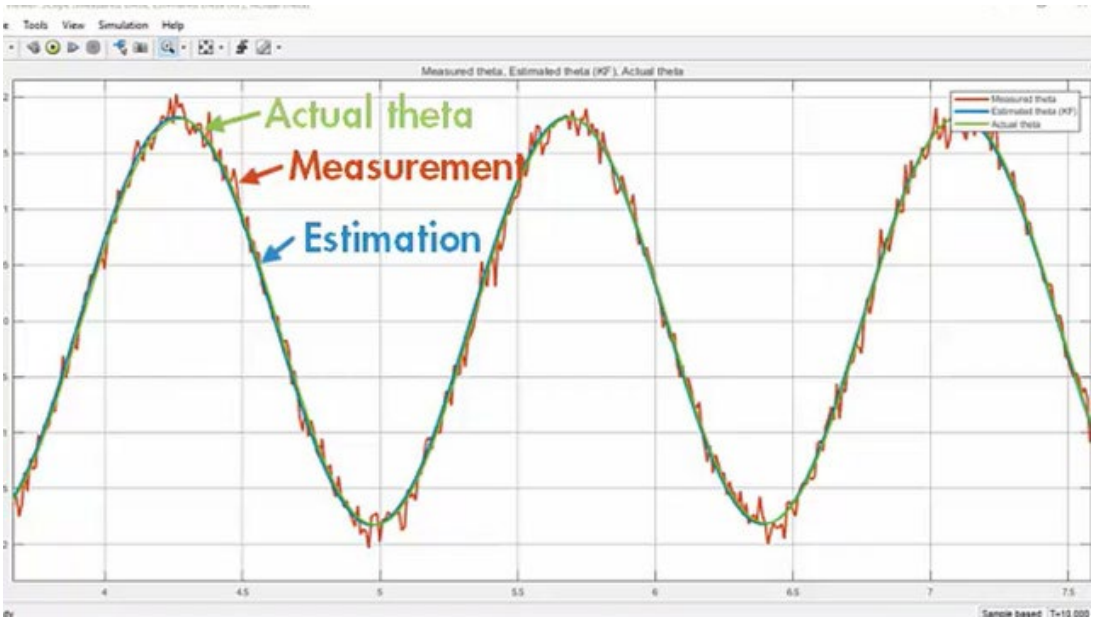

Figure 10: Effect of a Kalman Filter

A proportional controller is the simplest type of linear closed-loop feedback controller, and as such it is only suitable for simple or easy to control systems. It consists solely of a proportional gain for one of the system's state variables. The next step up from that is a PID controller, which adds gains for the integral and derivative of a specific state on top of a proportional gain. These controllers are fairly robust and are used in a wide array of applications. PID controllers are tuned manually, and there are several different approaches one can take to do so. Some popular tuning methods are the Ziegler-Nichols and Cohen-Coon methods. A more complex form of a PID controller is a FOPID controller, which allows the exponents on the I and D terms to be adjusted to any positive real number. This allows the controller to be more accurately tuned than a standard PID controller.

Complex systems can require more advanced control methods like an LQR. The main advantage of an LQR controller over a PID controller is the gains are determined using the plant physics, so no tuning is required. The Q and R matrices can be adjusted depending on desired performance regarding total energy expenditure and how quickly certain state variables converge to their desired values. Adjusting these matrices is much more intuitive than tuning PID gains, and thus provides a better user experience. In real systems, there is always some noise in the system that can disrupt a controller. One way of handling this is to use an LQG controller, which adds a Kalman filter in front of the gains generated by the LQR controller. A Kalman filter compares measured and predicted state values, and performs a weighted average of them to determine a confident estimate for the actual state values. The weights are determined using the covariance matrix of the state variables.

An even more robust controller is an FOPID. Its gains take the same form as a PID controller, with the addition of the I and D gains having variable exponents that can be set to any positive real number. This makes the controller more accurate because it has more variables that can be adjusted. Also, some of the variable tuning methods take into account the entire state history during the system's operation, which make it highly resistant to noise and other perturbations.

APPENDIX

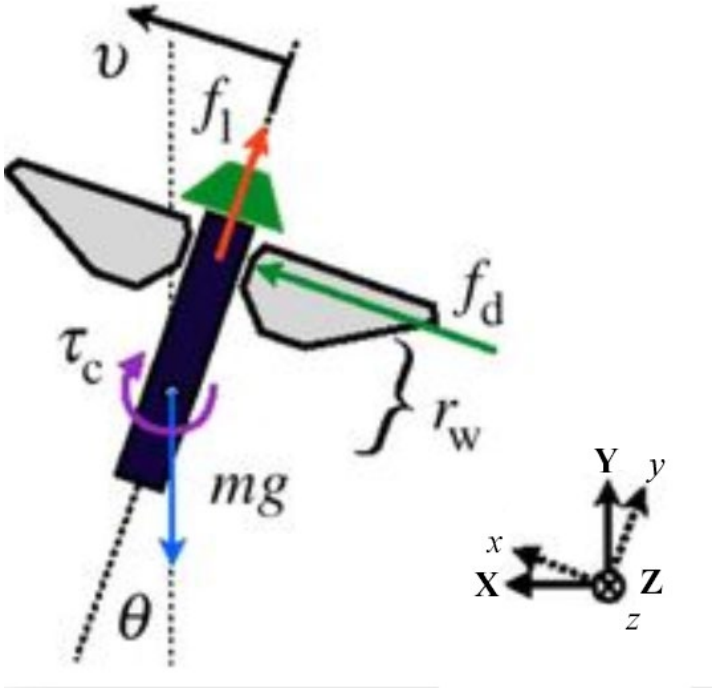

Figure A-1: Flying robot state variables depicted
Note: *Robots wings flap about y axis in inertial frame to generate an upward force*

REFERENCES

[1] Ziegler, J.G. and Nichols, N.B. *Optimum Settings for Automatic Controllers* (Transactions of the ASME vol. 64, 1942). Available Online: https://staff.guilan.ac.ir/staff/users/chaibakhsh/fckeditor_repo/file/documents/Optimum%20Settings%20for%20Automatic%20Controllers%20(Ziegler%20and%20Nichols,%201942).pdf

[2] Chen, Y., Petráš, I., and Xue, D. *Fractional Order Control - A Tutorial* (American Control Conference, 2009). Available Online: http://folk.ntnu.no/skoge/prost/proceedings/acc09/data/papers/1371.pdf

[3] Deniz, F.N., Yüce, A., Tan, N., and Atherton, D.P. *Tuning of Fractional Order PID Controllers Based on Integral Performance Criteria Using Fourier Series Method* (International Federation of Automatic Control, Volume 50, Issue 1, 2017). Available Online: https://www.sciencedirect.com/science/article/pii/S240589631731964X

[4] Murray, R.M. *Lecture 2 - LQR Control* (California Institute of Technology, January 2006). Available Online: https://www.cds.caltech.edu/~murray/courses/cds110/wi06/lqr.pdf

[5] Xue, D., Chen, Y., and Artherton, D.P. *Linear Feedback Control: Analysis and Design with MATLAB* (Society for Industrial and Applied Mathematics, 2007), p. 237.

[6] Tuzlukov, Vyacheslav. *Signal Processing Noise, Electrical Engineering and Applied Signal Processing Series* (CRC Press, 2010). ISBN 9781420041118.