Team 2898 Control System

Team 2898 - Ryo Takei

March 2019

Intro to Control Theory

Classical Control Theory

The PID controller is known as a classical control, and it is one of the most commonly used FRC control systems. It uses 3 controllers: a proportional controller, integral controller, and derivative controller. The proportional controller is represented by $K_p e(t)$, where K_p is a constant gain and $e(t)$ is the error in the system. The integral controller, $K_i \int_t^0 e(t)dt$, integrates the error($e(t)$) over time(t) and adds the current total times K_i to the control input. This is often used to solve issues in which the system is close to the reference in steady-state, so the proportional term is too small to pull the output to the reference and the derivative term is zero. Lastly, the derivative controller is represented by $K_d \frac{de(t)}{dt}$. The derivative gain compensates for future error($e(t)$) by slowing the controller down if the error is changing over time. PID controller designers are focused on fiddling with controller parameters relating to the current, past, and future error, rather than the underlying system states. Below is the standard form for a PID controller:

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}$$

Where $e(t)$ is the error in the system, t is the current time, and K_p, K_i, K_d represents proportional gain, integral gain, and derivative gain for the system.

Modern Control Theory

While classical control theory only focuses on the error in the system, model-based control has a completely different mindset. Controls designers using model-based control care about developing an accurate model of the system, then driving the relevant states to zero (or to a reference, using the error in the system). State-space control is a well-known example of a control system that is model-based.

State-Space Notation

$$\hat{x} = Ax + Bu$$
$$y = Cx + Du$$

where

A	system matrix	x	state vector
B	input Matrix	u	input vector
C	output matrix	y	output vector
D	feedthrough mtrix		

In the continuous case, the change in state and the change in output are linear combinations of the state vector and the input vector. The A and B matrices are used to map the state vector x and the input vector u to a change in the state vector $\hat{x}$. The C and D matrices are used to map the state vector x and the input vector u to an output vector y

State Space Controller

Closed Loop Control

Let's say that we are controlling a DC brushed motor. With just a mathematical model and knowledge of all current states of the system (i.e., angular velocity, rpm, gear ratio, size of the wheel, etc.), we can predict all future states given the future voltage inputs. Why, then, do we need feedback control? Well, if the system is disturbed in any way that isn't modeled by our equations – like if a load is applied to the armature, or voltage sag in the rest of the circuit causes the commanded voltage to not match the actual applied voltage, motors wear out – the angular velocity of the motor will deviate from the model over time. To combat this, we can take measurements of the system and the environment to detect this deviation and account for it. For example, we could measure the current position and estimate an angular velocity from it. We can then give the motor corrective commands, as well as steer our model back to reality. This feedback allows us to account for uncertainty and be unaffected by it; it makes the system more robust.

State Space Closed Loop Equations

In State-Space Control, the input matrix u is defined as $u = K(r - x)$

State-Space closed loop consists of two equation. First is the state update

function:

$$\begin{aligned}\hat{x} &= Ax + Bu \\ \hat{x} &= Ax + BK(r - x) \\ \hat{x} &= Ax + BKr - BKx \\ \hat{x} &= (A - BK)x + BKr\end{aligned}$$

And second is the output equation:

$$\begin{aligned}y &= Cx + Du \\ y &= Cx + DK(r - x) \\ y &= Cx + DKr - DKx \\ y &= (C - DK)x + DKr\end{aligned}$$

where

A	system matrix	K	controller gain
B	input matrix	x	state vector
C	output matrix	r	reference vector
D	feedthrough mtrix	y	output vector

Linear Quadratic Regulator

Linear Quadratic Regulator (LQR) design places the poles for us based on acceptable error and controls effort constraints. This method of controller design uses a quadratic function for the cost-to-go defined as the sum of the error and control effort over time for the linear system $\hat{x} = Ax+Bu$. Below is the equation that finds J, where J represents a tradeoff between state excursion and control effort with the weighting factors Q and R.

$$J = \int_0^\infty (x^T Q x + u^T R u) dt$$

Q and R represent weighing factors, which the system relies on. The Q matrix decides the penalty of state where control state is none-zero, while The R matrix decides the penalty of state where input state is none-zero. Choosing a large value for R means trying to stabilize the system with less (weighted) energy. This is usually called expensive control strategy. On the other hand, choosing a small value for R means that you don't want to penalize the control signal - cheap control strategy. In FRC, it is common to use cheap control strategy, since our control effort(775 motors, CIM motors, etc) is extremely cheap. Rockets, satellites, etc., use expensive control strategy, since fuel is limited. By balancing

Q and R, LQR is able to find optimal gain for the system. By using Bryson's rule, we can find optimal Q, and R for inexpensive control.

$$J = \int_0^\infty (p[(\frac{x_1}{x_{1,max}})^2 + \cdots + (\frac{x_n}{x_{n,max}})^2] + [(\frac{u_1}{u_{1,max}})^2 + \cdots + (\frac{u_n}{u_{n,max}})^2])dt$$

$$Q = \begin{bmatrix} \frac{p}{x_{1,max}^2} & 0 & \dots & 0 \\ 0 & \frac{p}{x_{2,max}^2} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & \frac{p}{x_{n,max}^2} \end{bmatrix} R = \begin{bmatrix} \frac{1}{u_{1,max}^2} & 0 & \dots & 0 \\ 0 & \frac{1}{u_{2,max}^2} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{u_{n,max}^2} \end{bmatrix}$$

Team 2898 System Controllers

Virtual Four Bar

Continuous state-space model

The angle and angular rate derivative of the arm can be written as

$$\hat{\theta}_{arm} = \omega_{arm}$$
$$\hat{\omega}_{arm} = \hat{\omega}_{arm}$$

where

$$\hat{\omega} = -\frac{G^2 K_t}{K_v R J}\omega_{arm} + \frac{G K_t}{R J} V$$

State-space model for arm:

$$\hat{x} = Ax + Bu$$
$$\hat{y} = Cx + Du$$

$$x = \begin{bmatrix} \theta_{arm} \\ \omega_{arm} \end{bmatrix}$$

$$y = \theta_{arm} \quad u = V$$

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{G^2 K_t}{K_v R J} \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ \frac{G K_t}{R J} \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad D = 0$$

where

G	gear ratio	R	motor resistance
K_v	motor velocity constant	J	moment of inertia
K_t	motor torque constant	V	voltage

Discrete state-space model

After plugging in numbers based on the equation above, we can find the discrete state space model. it is worth mentioning that although FRC control loop is 50hz, Team 2898 is controlling our arm with 150hz for smoother, more accurate control.

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0.228 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 0.193 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad D = 0$$

LQR

Once we have the discrete state-space model, we can find optimal gain for the system. Once again, we follow Bryson's rule to find Q R matrices.

$$Q = \begin{bmatrix} 0.01745 & 0 \\ 0 & 0.01745 \end{bmatrix} \quad R = \begin{bmatrix} 12.0 & 0 \\ 0 & 12.0 \end{bmatrix}$$

And after that, we use the Matlab function, *lqr()* to find the optimal gain K for the discrete state-space model.

$$K = \begin{bmatrix} 25.29673028 & 1.27854674 \end{bmatrix}$$

Kalman Filter

A Kalman filter is a system that uses a series of measurements observed over time, which contain a lot of noise, inaccuracies, and time delay. Even the best encoder in the world has time delay and noise from the system. By using a Kalman filter, the system can make a "guess" as to where the system is in the space. This allows us to have more accurate and reliable data, based on the encoder, for the system to use. This can also remove all noise from the system, resulting in much smoother control. Once again, we use the Matlab function, *kalman()* to find the optimal Kalman filter gain, Km for our system.

$$Km = \begin{bmatrix} 0.79374646 \\ 0.0825426 \end{bmatrix}$$

Two State Feedforward

Unlike feedback systems, a feedforward system does not control variables based on error values in the system. Instead it uses a mathematical model of the process and knowledge about disturbances in the system. It is generally used to handle parts of control actions that we already know must be applied to make a system track a reference, and it allows the feedback system controller to correct for what we do not or cannot know about the system at runtime. two state feedforward gives greater control over the system.

Two-state feedforward gain(u_{ff}) can be found by using the equation:

$$u_{ff} = K_{ff}(r_{k+1} - Ar_k)$$

where

$$K_{ff} = (B^T Q R)^- 1 B^T Q$$

Note: Since we are using an inexpensive controller, $R \ll Q$. Once again, by using Matlab, we can find optimal two-state feedforward gain:

$$u_{ff} = \begin{bmatrix} 0.80116395 & 5.17683707 \end{bmatrix}$$

Gravity Feedfoward

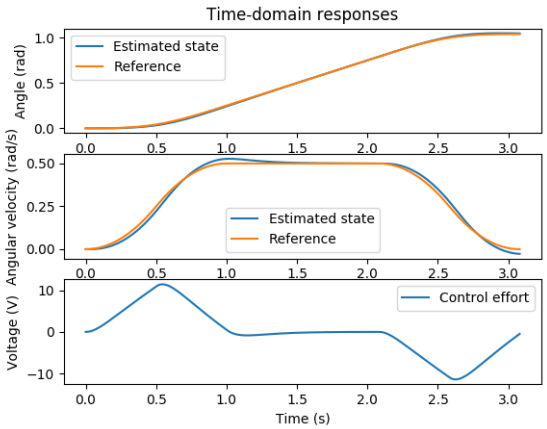
It's important to mention that state-space works best when the system is linear. For a system to be a linear system, a) and b) have to be true.

a) when $f(x) = y, f(Ax) = Ay$ where A is a constant
b) $f(A + B) = f(A) + f(B)$

With a system (i.e., arm) where gravity is always being applied, we have to linearize the system before we can feed it to the controller. Gravity feedforward is the most common feedfowarding system, and using this, we can mathematically predict the gravity force that is being applied to the system based on the position of the arm. Arm gravity feedforward can be written as:

$$K_g = K_f * cos(\theta_{arm})$$

Once we have linearized our system, we can mathematically predict the control that the system will need in order to make the system linear and let the feedback system handle the system disturbances that we are unable to predict.



Using everything mentioned above, Team 2898 are able to control our arm with 98.8% overall accuracy.