Optimizing a PID Controller for Simulated Single-Joint Arm Dynamics

By: Indrani Mikkilineni
Shyam Patel
Chia-Hung Tai

BENG 221
November 21st, 2014

Table of Contents

Introduction

IPMC Motivation and Potential for Innovative

Ionic polymer metal composites (IPMC) are good candidates to simulate artificial muscles due to their specific properties [1]. Low density, low voltage requirement, simple fabrication, wide range of electrically induced bending, and mechanical flexibility are a few of such characteristics for IPMC. IPMC bends due to changing voltage, which is within the range of -3V to 3V [3]. How the IPMC bends depends on the current that flows through it – when a voltage is applied, the material will bend [1]. The direction of the bending (toward the anode) relies on the direction of the cation migration towards the cathode, and the direction will reverse as the polarity changes [3]. This response is very similar to the response of human muscle.

IPMCs have already been implemented as actuators – serving as precision surgical equipment and modeling finger-like designs to perform delicate gripping (**Fig 1**) [2]. Yet applications with IPMC are limited and the control still needs to be fine-tuned. Our goal is to utilize the unique properties by modeling the IPMC strips to actuate the prosthetic arm and to provide precise control using a PID controller.

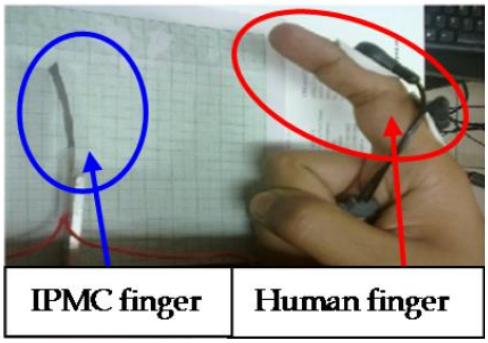


Figure 1. IPMC vs. Human Finger[6]

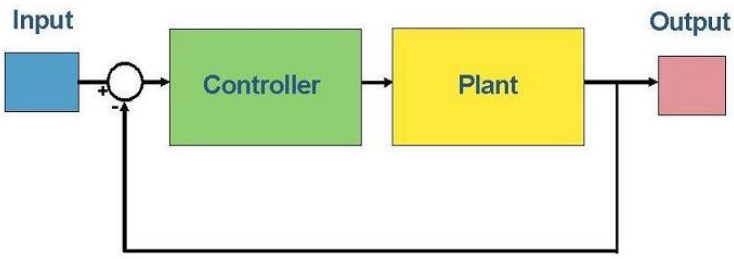


Figure 2. Closed Loop System[4]

Feedback Loop

In order to provide information to the controller about whether the plant has performed its task or not, a closed loop system is used so the controller knows what the plant is actually doing (**Fig 2**) [4]. The output from the plant is monitored and feedback is provided to the controller, which is then compared with the system input to determine deviations from the expected output, allowing the controller to make any necessary adjustments. This allows the system to counteract errors and decrease response time. The PID controller will be added to the feedback loop.

Response Properties

The main purpose of the feedback loop system is to correct for error. When optimizing the system, a few specific properties defined below and shown in **Fig 3** are focused on to improve the output response. [4]

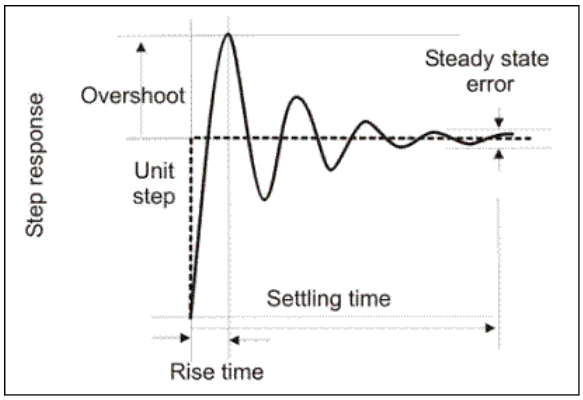


Figure 3. Properties of Response[4]

- Rise time: Time it takes to go from 10% to 90% value of the input response
- Overshoot: Maximum peak reached (in a step response, it can be referred to as a percentage overshoot – taking the difference of max value and steady-state value and divide by the steady-state value)
- Settling time: Time it takes for the output to reach a 2% tolerance a band of the steady-state value
- Steady-state error: The difference between the actual position and where the position should be (2% tolerance)
- Oscillation: The system can oscillate infinitely, ends when it reaches the steady state, or have no oscillations depending on the characteristics of the system.

Each property represents a behavior in the movement of a prosthetic arm controlled by an input voltage:

- Rise time: The time when the bulk of the motion is observed.
- Overshoot: The displacement of the arm position past its desired range due the initial response.
- Settling time: Time it takes for the arm to reach its final position.
- Steady-state error: The difference between actual location and desired location.
- Oscillation: The arm cycles back and forth between the desired location until it settles to its final position

The optimal system would have the shortest possible rise time and settling time; as well as the smallest steady-state error and overshoot. Depending on the objective for the system, some properties are valued more than others. The role of the controller is to tune the response to best meet the criteria.

PID Controller

PID is an acronym for each of the components of the system: proportional, integral, derivative. It is a common type of controller for linear systems, with each component bringing an improvement to specific properties of the response at the expense of other properties (see **Table 1** for clarification) [5]. The controller is tuned by manipulating the constants K_P, K_I, and K_D.

Parameter	Rise Time	Overshoot	Settling Time	S.S Error	Stability
Kp	Decrease	Increase	Small Change	Decrease	Worse
Ki	Decrease	Increase	Increase	Significant Decrease	Worse
Kd	Minor Dec.	Minor Dec.	Minor Dec.	No change	If Kd small, Better.

Table 1. Relationships of different control terms on properties[5]

P component – Proportional (K_pE)

This component is the bulk of the PID control system. The controller takes the instantaneous error value, multiplies by the constant K_P, and adds it to the input signal. Since the term depends only on the current error value, it is referred to as the "present" error. The proportional component by itself makes a respectable controller, reducing rise time and slightly lowering steady-state error [5]. The tradeoff is the loss in stability and increase in overshoot.

I component – Integral ($K_i \int E\ dt$)

The integral component helps to improve on the proportional control by further reducing the steady-state error. This control, referred to as the "past" error, compiles all the past error (by integrating the error value for a moving time window) and multiplies by the constant K_I. Although the steady-state error is reduced, this causes the system's oscillation to increase and the speed of response to decrease [5].

D component – Derivative ($K_D \frac{dE}{Dt}$)

The last component focuses on the "future" error, which is done by taking the derivative of the error signal and multiplying by the K_D constant. The derivative portion helps to improve overshoot, rise time, and settling time [5]. These improvements are theoretical since the values are based on an approximation from the current errors. This component may not be reliable in real-world scenarios, since it does not respond stably to noisy signals.

Fig 4 demonstrates the closed loop PID system. The final equation of the PID controller is demonstrated by *Equation 1.1*. [5]

$$U = K_p E + K_i \int E dt + K_D \frac{dE}{Dt} \tag{1.1}$$

$$U = Controller\ Output \qquad E = System\ Error$$

$$K_p = Proportional\ Gain \qquad K_i = \frac{K_p}{T_i} = Integral\ Gain \qquad K_D = K_p * T_D = Derivative\ Gain$$

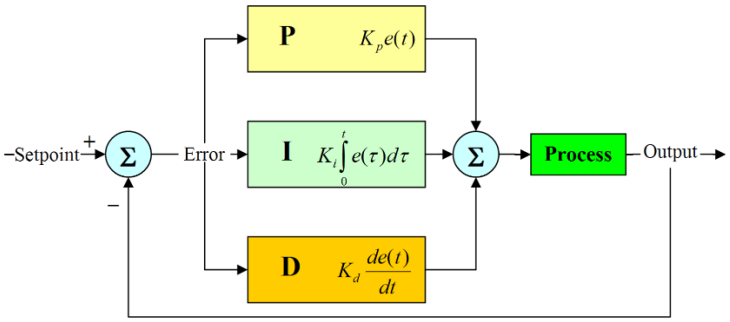


Figure 4. PID Controller Block Diagram[5]

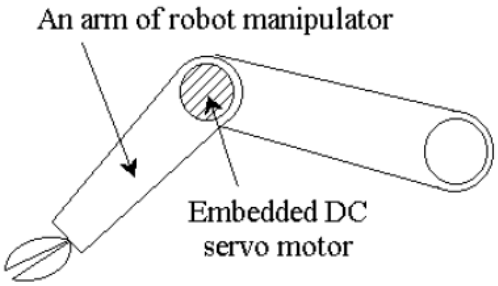


Figure 5. Diagram of our elbow[7]

Problem Statement

The main goal of this study is to closely mimic human arm movement by incorporating IPMC strips to a prosthetic arm to model the elbow movement, controlled by a neural input (**Fig 5**). We plan to improve the step response of a prosthetic arm using a PID control system in order to address past, present, and future error. Ultimately, we will attempt to optimize among the following parameters for the best outcome – the response time, overshoot, and steady-state error.

Arm Model

Based on the torque balance between inertia and friction, the torque for the elbow joint can be modeled by the ordinary differential Equation (ODE) 2.1. [7]

$$J * \theta'' + f * \theta' = \tau + Mgl * \cos(\theta) \quad (2.1)$$

Since this equation is non-linear, assume gravity will not affect the arm in horizontal motion. The range of motion and the gravitational field are perpendicular, thus excluding it from our equation making it easier to solve. [7]

$$J * \theta'' + f * \theta' = \tau \quad (2.2)$$

$J = intertia\ of\ arm$ $\theta = angle\ of\ rotation$

$f = friction\ coefficient\ of\ joint$ $\tau = actuator\ torque$

Assumptions

- Arm
  - Has a horizontal range of motion (to eliminate the gravity/weight term)
  - Geometrically an ideal cylinder
- Elbow joint
  - Modeled to have only one degree of freedom
  - Coefficient of friction is constant

- Controller
  - Motor torque output is completely linear relative to the input voltage
- Signal input
  - Neural input is processed to behave like an EMG (generating a unit step instead of pulses)

Our ODE is modeling the equivalence of torques, with friction and inertia resisting an applied torque from our actuator. Since we are modeling the output position to a step input voltage, the system without control will never stop at a particular position. Although when a feedback loop is added, the actual position and desired position can be compared, and voltage (effectively, motor torque) can be increased or reduced to get to the target position.

In order to demonstrate a steady response without feedback, another situation is hypothesized:

Arm Model with Resistance Band

We need an extra torque dependent on the position itself, so a "spring" term was added to get a stable open loop response seen in *Equation 2.3.*

$$J * \theta'' + f * \theta' + R_b * \theta = \tau \tag{2.3}$$

$$R_b = Resistance\ Band \text{ (Spring Constant)}$$

To linearize the "spring" term, we assumed a straight trajectory for the resistive band instead of the actual circular path. All assumptions remain the same as the original arm model.

Characteristics of System Response

We are using the Arm Model with the Resistance band to observe the characteristics, as the normal Arm Model doesn't converge at a finite value as time approaches infinity. By observing *Equation 2.3*, we can calculate the following two properties – damping ratio (ζ) and undamped natural frequency(ω_n) [8]. These properties of a second order system will explain a relationship among the parameters we will be optimizing later on such as rise time, overshoot, settling time.

Solve for $\zeta\ and\ \omega_n$. First divide *Equation 2.3* by J.

$$\theta'' + \frac{f}{J} * \theta' + \frac{R_b}{J} * \theta = \tau \tag{3.1}$$

$$\omega_n^2 = \frac{R_b}{J} \qquad 2\omega_n\zeta = \frac{f}{J} \tag{3.2, 3.3}$$

Constant Values	
Mass (m)	$1kg$
Radius (r)	$8.89 * 10^{-2}m$
Friction (f)	0.2
Torque (τ)	$1.39\ kgm$
Input Voltage (E_a)	$6V$
Resistance Band (R_b)	$1kg/rad$
Intertia $(J) = m * r^2$	$7.90 * 10^{-3}m^2kg$

Table 2. Calculated and given constants describing the system[3, 10, 11]

Above is the table of constants that we will be using throughout the paper. The mass, measured torque, and distance from the center of mass to the joint are obtained from Boston Digital Arm since we were not able to find such values pertaining to IPMC arms [10]. The friction of coefficient is obtained for that of Ultra-high molecular weight polyethylene instead of the IPMC, which is a common material is used for the prosthetic joints [11]. The input voltage used is based on a voltage range common for IPMC [3].

$$\omega_n = \sqrt{\frac{R_b}{J}} = 11.2\ \frac{rad}{s} \qquad \zeta = \frac{f}{2\sqrt{JR_b}} = 0.225\frac{kgs}{rad}$$

The damping ratio (ζ) will show how much the system oscillates before reaching steady-state. As the value of the damping ratio is less than 1, we can determine that the system will be underdamped. The natural frequency (ω_n) is the frequency the system would oscillate without damping, in our case friction.

Overshoot is an important parameter to optimize as having the mechanical arm fling past the point which would be detrimental to having control. Overshoot is inversely proportional to the damping ratio and completely unrelated to the natural frequency. The equation below shows the relationship between overshoot and the damping ratio. [9]

$$Percent\ Overshoot = 100 * e^{-\frac{\zeta\pi}{\sqrt{1-\zeta^2}}} \tag{3.4}$$

Settling time (T_s) is the time until the system reaches steady-state. For estimation purposes, we will find the time when the system is within 2% of the steady-state value. The following equation shows the relationship between the damping ratio/natural frequency and settling time. [9]

$$T_s = -\ln\left(-\frac{0.02\sqrt{1-\zeta^2}}{\zeta\omega_n}\right) \approx 4/\zeta\omega_n \tag{3.5}$$

Rise Time (T_r) is the time that the system takes to reach from 10% to 90% of the steady-state value. There is not a direct relationship to compare rise time and the damping ratio, but in general they are directly proportional. [9]

Keep in mind that these properties/parameters are specifically true for **second-order systems**.

Derive Transfer Functions

Transfer function of Arm

Start with the Arm Model *Equation 2.2*. Take the Laplace and assume initial conditions are 0, as the arm is not moving initially.

$$\theta(s) * (Js^2 + fs) = T(s) \tag{4.1}$$

Solve for the Transfer Function and multiply by $\left(\frac{1}{f}\right)/(\frac{1}{f})$.

$$G_{arm}(s) = \frac{\theta(s)}{T(s)} = \frac{\frac{1}{f}}{s(T_m s + 1)} \; where \; T_m = \frac{J}{f} \tag{4.2}$$

Assume Input Voltage and Torque are linearly proportional: $\tau = A * E_a$

$$T(s) = A * E_a(s) \tag{4.3}$$

$$G_{arm}(s) = \frac{\theta(s)}{E_a(s)} = \frac{C}{T_m s^2 + s} \; where \; C = \frac{A}{f} \tag{4.4}$$

Solve for constants A and C.

$$T_m = \frac{J}{f} = 3.95 * 10^{-2} m^2 kg$$

$$A = \frac{\tau}{E_a} = 2.31 * 10^{-1} mkg/V$$

$$C = \frac{A}{f} = \frac{1.15 mkg}{V}$$

Transfer function of Arm with Resistance Band

Start with the Arm Model with Resistance Band *Equation 2.3.* Take the Laplace and assume initial conditions are 0, as the arm is not moving initially.

$$\theta(s)(Js^2 + fs + R_b) = T(s) \tag{4.5}$$

Solve in a similar manner as the above Transfer function of the arm.

$$G_{arm\_modified} = \frac{\theta(s)}{E_a(s)} = \frac{C}{T_m s^2 + s + \frac{R_b}{f}} \tag{4.6}$$

Transfer function of PID Controller

Start with the PID Controller *Equation 1.1*. Take Laplace, assume initial conditions are 0, as there is no initial controller output or error.

$$U(s) = E(s)\left(K_p + K_i * \frac{1}{s} + K_D * s\right) \tag{4.7}$$

Solve for the transfer function of PID

$$G_{PID}(s) = \frac{U(s)}{E(s)} = K_p + K_i \frac{1}{s} + K_D s \tag{4.8}$$

Transfer function of Closed Loop System

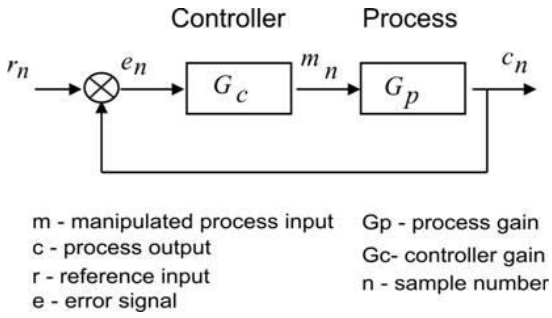


Figure 6. Feedback/Controller Block Diagram [4]

First, let's solve for a general Transfer function of the above closed loop system in Figure 6. For our system assume $G_p(s) = G_{arm}\ and\ G_c(s) = G_{PID}$ [4]. Also, the fact that we know gain = output/input gives us the following equations.

$$G_p(s) = \frac{c_n}{m_n} \qquad G_c(s) = \frac{m_n}{e_n} \qquad G_{sys}(s) = \frac{c_n}{r_n} \tag{4.9, 4.10, 4.11}$$

$$e_n = r_n - c_n \tag{4.12}$$

$$c_n = G_p(s)m_n \qquad m_n = G_c(s)e_n \tag{4.13, 4.14}$$

$$c_n = G_p(s)G_c(s)e_n \tag{4.15}$$

$$c_n = G_p(s)G_c(s)(r_n - c_n) \tag{4.16}$$

$$G_{sys}(s) = \frac{c_n}{r_n} = \frac{G_p(s)G_c(s)}{1 + G_p(s)G_c(s)} \tag{4.17}$$

Now let's solve for our system by convoluting the $G_{arm}\ and\ G_{PID}$ in *Equations 4.4 and 4.8*.

$$G_{system}(s) = \frac{G_{arm}(s)G_{PID}(s)}{1 + G_{arm}(s)G_{PID}(s)} \tag{4.18}$$

$$G_{system}(s) = \frac{\frac{C}{(T_m s + 1)}(K_p + K_i \frac{1}{s} + K_D s)}{1 + \frac{C}{(T_m s + 1)}(K_p + K_i \frac{1}{s} + K_D s)} \quad (4.19)$$

$$G_{system}(s) = \frac{s^2(CK_D) + s(CK_p) + CK_i}{s^3(T_m) + s^2(1 + CK_D) + s(CK_p) + CK_i} \quad (4.20)$$

Analytical solution

Starting with the transfer functions of both the normal arm and modified arm (model with resistant band), let's simplify by using partial fractions and then take the inverse Laplace. The open loop system below in Figure 6 is the system in which our analytical solutions will be solved, as the analytical solutions for the closed loop system with the PID Controller is too complex.

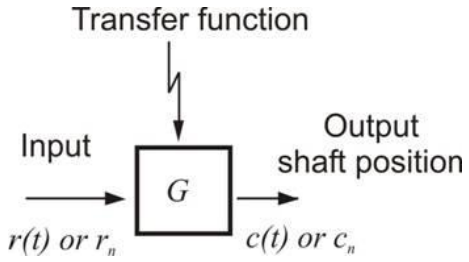


Figure 7. Open Loop System Block Diagram[4]

For our system, assume the input - $E_a(s)$ and output - $\theta(s)$.

Arm

Given *Equation 2.2,* we were able to obtain *Equation 4.4*. Then, separate using partial fractions.

$$G_{arm}(s) = \frac{\theta(s)}{E_a(s)} = \frac{C}{T_m s^2 + s} = \frac{A}{s} + \frac{B}{T_m s + 1} = \frac{A(T_m s + 1)}{(T_m s + 1)(s)} + \frac{B(s)}{(T_m s + 1)(s)} \quad (4.21)$$

$$A(T_m s + 1) + B(s) = C$$

$$A = C \; and \; B = -T_m C$$

$$G_{arm}(s) = \frac{C}{s} + \frac{-T_m C}{T_m s + 1} = \frac{\theta(s)}{E_a(s)} \quad (4.22)$$

Take the inverse Laplace.

$$\theta(t) = \left(C - Ce^{-\frac{t}{T_m}}\right) E_a(t) \quad (4.23)$$

Modified Arm

Given the modified arm *Equation 2.3*, we were able to obtain the transfer function in *Equation 4.6.* To simplify the equation let's set $R = R_b/f$. To find the partial fractions, use the following quadratic equation.

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{4.24}$$

$$G_{arm\_modified} = \frac{\theta(s)}{E_a(s)} = \frac{C}{T_m s^2 + s + R} = \frac{A}{s + \frac{1 + \sqrt{1 - 4(T_m)(R)}}{2T_m}} + \frac{B}{s + \frac{1 - \sqrt{1 - 4(T_m)(R)}}{2T_m}} \tag{4.25}$$

$$A = -\frac{C}{\sqrt{1 - 4RT_m}} \; and \; B = \frac{C}{\sqrt{1 - 4RT_m}}$$

Take the inverse Laplace.

$$\theta(t) = \left(\frac{-Ce^{t*\left(-\frac{\sqrt{1-4RT_m}}{2T_m} - \frac{1}{2T_m}\right)} + Ce^{t*\left(\frac{\sqrt{1-4RT_m}}{2T_m} - \frac{1}{2T_m}\right)}}{\sqrt{1 - 4RT_m}} \right) * E_a(t) \tag{4.26}$$

These analytical solutions show the solution of how the angle of rotation is related to the input voltage in the time domain compared to the transfer functions which were in the evaluated in the s domain.

Optimization Methods

When evaluating optimization methods, it is important to remember that this process is very subjective, and that there is no "right" answer when selecting control parameters, thus there are many different methods to select parameters. A method is chosen based on the response specific to the system its constraints.

Manual

The simplest optimization method is manual tweaking of PID parameters. A simulation can be set up in Matlab (or other appropriate software) to evaluate the numeric value of the response for a certain time range. Using what is known about each of the different control terms in PID, parameters are changed and the updated response is evaluated for quality. This method is essentially guess-and-check.

Ziegler-Nichols

We were originally going to use the Ziegler-Nichols optimization method, but this would not work for our situation since it requires constant oscillations in a P controller. The reason for this is because of an input torque being converted to a position output (the reason why this system is not stationary if there is no controller or spring force). Since we can't do this, we decided to use Cohen-Coon instead.

Cohen-Coon

The Cohen-Coon optimization method (see Table 3) is done by analyzing the open-loop response of a system and getting time values for when the response is 50% of the steady state value and 63.2% of the steady state value [12]. These time values are used with a predetermined table specific to this method to compute the values of the control constants. This method is most appropriate for systems that have a relatively long rise time.

Cohen – Coon Time Values
t_0 = time when input initiated
t_2 = time when 50% of steady state value is reached
t_3 = time when 63.2% of steady state value is reached

Cohen – Coon Calculated Values	
t_1	$\frac{t_2 - (ln(2) * t_3)}{1 - ln(2)}$
τ	$t_3 - t_1$
τ_{del}	$t_1 - t_0$
K	$\frac{Steady\ state}{Input\ step\ magnitude}$
r	$\frac{\tau_{del}}{\tau}$
K_p	$\left(\frac{1}{K} * r\right) * \left(\frac{4}{3} + \frac{r}{4}\right)$
T_i	$\frac{\tau_{del} * (32 + 6r)}{13 + 8r}$
T_D	$\frac{4 * \tau_{del}}{11 + 2r}$

Table 3. Cohen-Coon Algorithm[12]

Simulink

The Simulink module in MATLAB is a very user-friendly method, provided that the system model can be created. To start optimization, simply "tune" the PID controller block in the model. The program will show you the current and "tuned" response, the latter of which will change as the properties sliders are adjusted. Simply raising the sliders to max will usually not yield a perfect result, as there are tradeoffs between desirable system properties. For tuning of real systems, this seems to be the most appropriate choice, as it asks the user to input the properties they desire, not specific constants. It also takes into account certain design rules for real systems (like minimizing the derivative gain when possible in case of noisy signals).

Results/Discussion

Responses/Manual Optimization

As described earlier, the uncontrolled system with no resistive band (**Fig 8**) forms an infinite ramp. This corresponds to the arm spinning at a constant angular velocity. This is because a step torque is being input into the system, which eventually forms an equilibrium with the friction and inertia forces, and maintains a constant angular velocity. The arm is not actually going to an infinite position value, rather the arm keeps rotating, and the output can be rewritten as an angular position, along with a certain number of complete rotations.

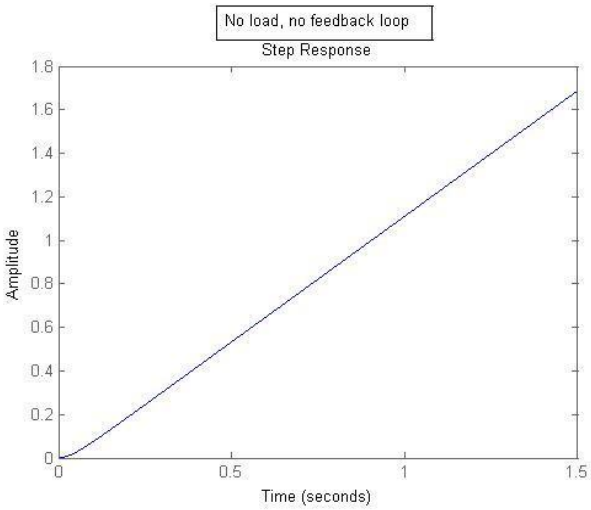


Figure 8. No load, no feedback loop

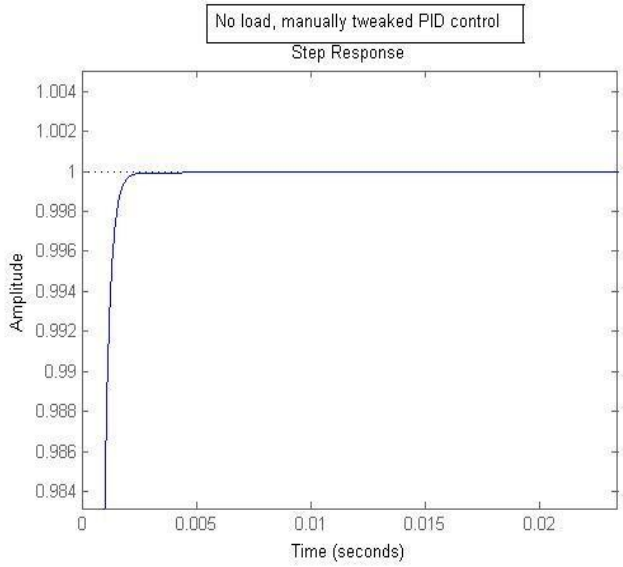


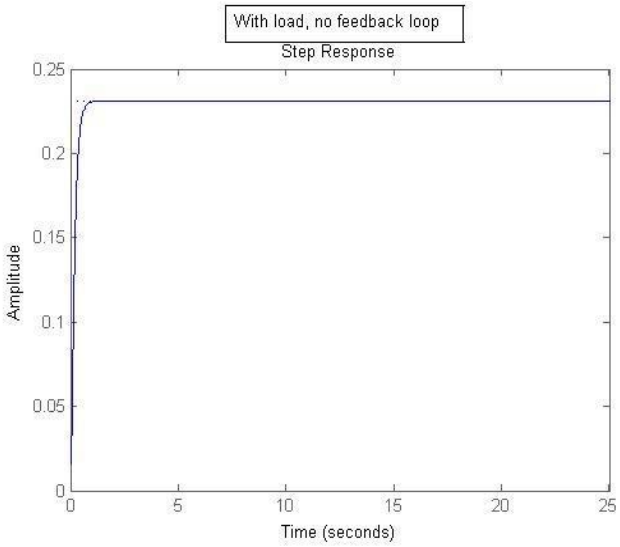
Figure 9. No load, manually optimized PID Control

Percent Overshoot: 6.2172e-13%
Steady State Error: 1.9984e-15
Rise Time: 5.4000e-04 seconds

Control Constants

K_p = 3500
K_i = 0
K_D = 140

With the addition of feedback control, the steady state value converges onto the target value. This system is simple and has no steady state error even with a simple proportional controller (**Fig 9**). A derivative term was added as well to minimize overshoot and allow for a smooth response at higher proportional gains. The method here was to raise the proportional gain to reduce rise time, then raise the derivative gain to reduce overshoot. This iteration was done several times to come to the final PID controller shown above. The response shown above has no apparent issues, with minimal overshoot and steady state error, and a fast response time. This would appear as an arm that rapidly arrives at its target position, with no oscillations.

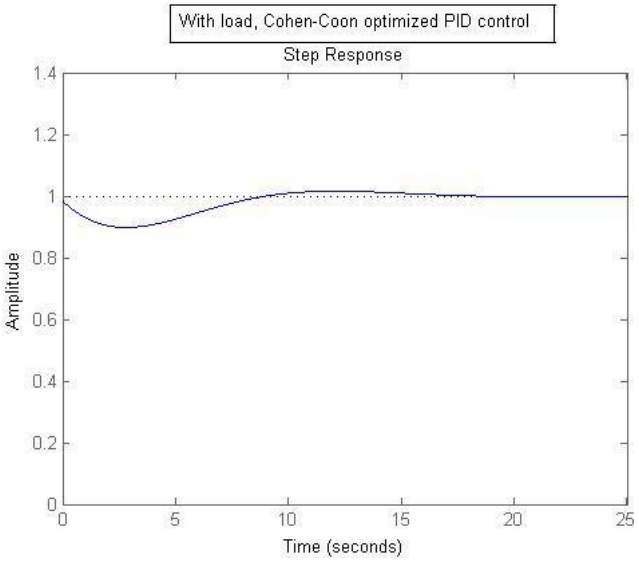


Percent Overshoot: N/A
Steady State Error: -.7691
Rise Time: .3546 seconds

Figure 10. Resistive band without feedback loop

With the addition of a resistive band, the system no longer needs a controller to have a finite steady state value (**Fig 10**). Response properties are still in much need of improvement at this stage, with the most noticeable problem being the extremely large steady state error. This response would appear as an arm that reaches its final position fairly quickly, but stops much earlier than it should. Thus its actual position is very far from its desired position.

Cohen-Coon



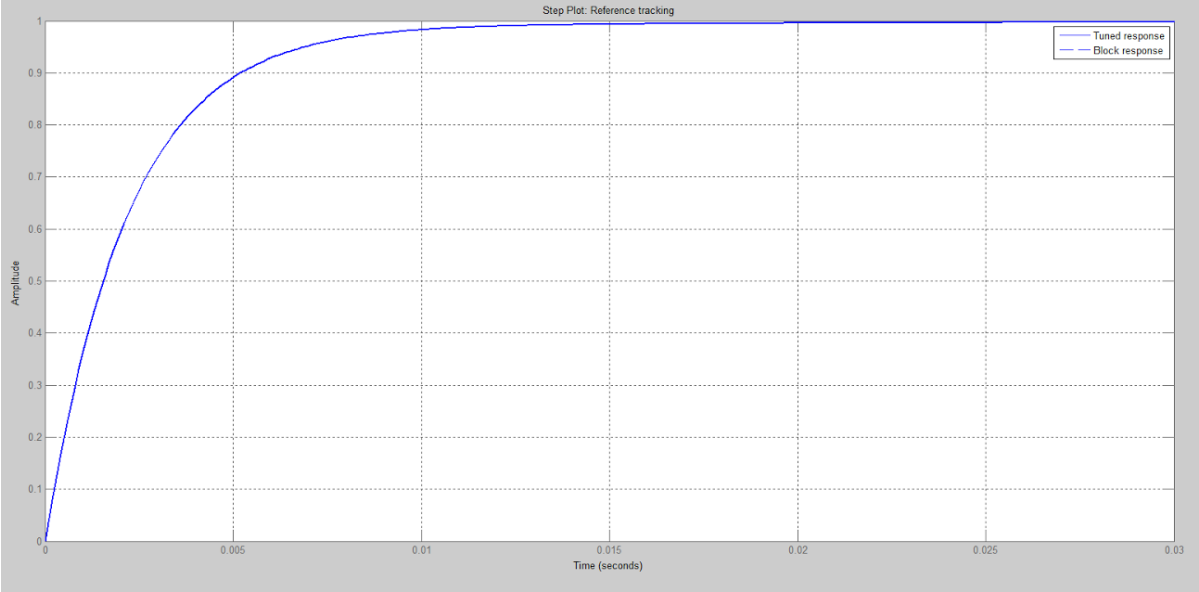
Percent Overshoot: 1.59%
Steady State Error: 1.9984e-15
Rise Time: 5.4000e-04 seconds

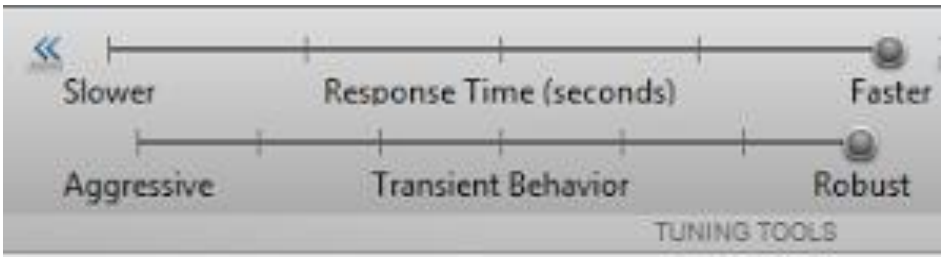
Control Constants
$\mathbf{K_p}$ = 19.1485
$\mathbf{K_i}$ = 9.0770
$\mathbf{K_D}$ = 57.5846

Figure 11. Cohen-Coon optimized PID on resistive band system

Cohen - Coon optimization gives a PID controller an extremely fast response time (see **Fig 11**). This comes at the expense of oscillations, and what is effectively a “negative” overshoot. This response would appear as an arm that initially moves extremely fast towards its final position, but then experiences a slow but significant oscillation for the next 10-15 seconds.

Simulink



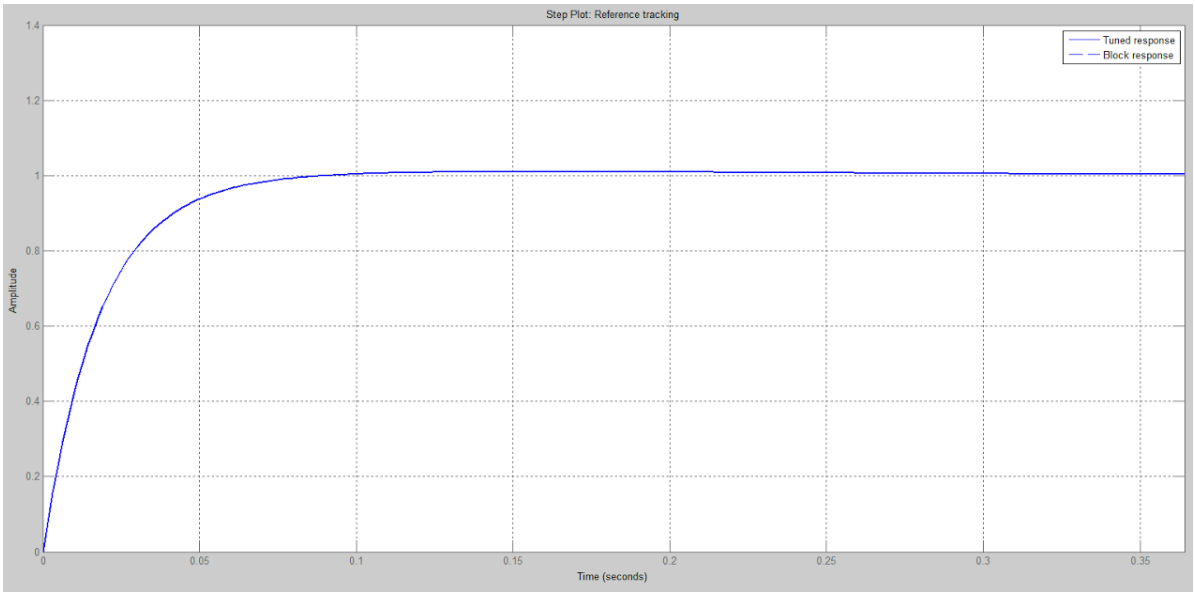


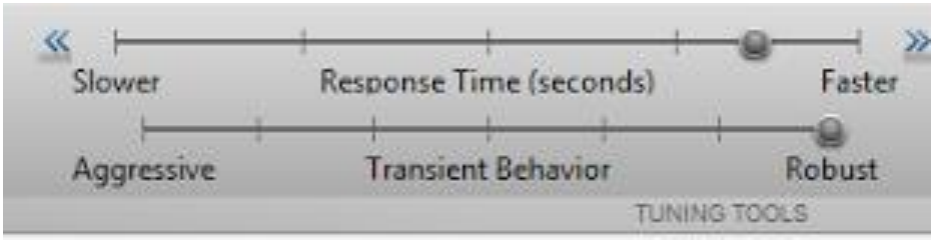
Percent Overshoot: 0%
Steady State Error: -.001
Rise Time: .00495 seconds

Control Constants
K_p = 329.5
K_i = 3.215
K_D = .04683

Figure 12. Simulink optimized without Resistance Band – included tuning settings from module (Both sliders set to max)

The tuning settings for an arm with no resistance are pretty simple. Both sliders are just put to the maximum value and the “cleanest” and fastest response is achieved (**Fig 12**). The physical behavior of this system is similar to the manually tuned system, with a rapid convergence on the target position, and no overshoot. There is a key difference in the controller itself, in that the derivative gain is much lower than that of the manually tuned system, which makes the system more resistant to noisy signals.





Percent Overshoot: 1.05%
Steady State Error: .005
Rise Time: .0396 seconds

Control Constants
K_p = 46.6601
K_i = 5.949
K_D = .04057

Figure 13. Simulink optimized with Resistance Band – included tuning settings from module

The tuning settings for an arm with a resistance illustrate some of the tradeoffs involved in control system optimization (**Fig 13**). A faster response time can be specified, though this will come at the expense of either a significant steady state error (approximately 10%), or some overshoot in the response. Since this response is still very fast given the physical requirements, with a rise time of only 40 ms, stability of the response has been prioritized over response speed.

Optimization Method Selection

It was found that the best type of optimization for both scenarios was the Simulink-tuned PID controller. The Simulink-tuned controller was preferable to the manually tuned controller in the system without the resistance band, even though the system response was slower. This was because the derivative term had a much lower gain, and the system would be much more stable in the field, where there can be noisy signals. The Simulink-tuned controller

was preferable to the Cohen-Coon controller in the system with the resistance band, despite having the same issue of slower response time. In this case, Cohen-Coon was deemed inferior because of the long lasting oscillations that were present in the step response.

Model Limitations

There are several limitations in our mathematical model. The main limitation in our model is our inability to include gravitational force, and thus an inability to model movement in the vertical plane. Another limitation in our model is the lack of multiple degrees of freedom. Theoretically, each joint would be independent of the surrounding joints when considering its own angular displacement, but the presence of inertia and momentum from other members would make each joint dependent on the other joints. This is made even more significant if we decide to model a multi-joint arm that is moving in the vertical plane with the influence of gravity.

Conclusion

We attempted to model a single joint arm that has a range of motion perpendicular to gravity, and with no external load. A torque was applied to the arm, and the forces that resist this torque are the inertia of the arm and the friction of the joint. It was found that this system alone would not produce a stable step response without a feedback loop, so a resistive band was added to stabilize the position in an open loop system. Different optimization methods were applied for the PID controllers of each of these systems. The methods utilized were manual tuning, Cohen-Coon optimization, and tuning in the Simulink module in MATLAB. It was determined that tuning with Simulink would give the best overall PID solution in both scenarios.

We explored and verified the use of PID controllers for IPMC prosthetics, using Simulink to select the controller characteristics. Simulink assisted us in finding the optimum balance of response characteristics that would be appropriate for replicating a human arm. This advantage was observed most in the arm with the resistance band, where there was no clear “best” solution and some tradeoffs had to be made. The Simulink UI was very useful in quickly observing a range of solutions, as opposed to using the guess and check method of manual tuning.

To improve this model, we can attempt to introduce gravity into this mode and add multiple joints. Introduction of multiple joints would create dependencies on the momentums of other joints. This effect would be magnified if gravity was also introduced into the model. An issue with the introduction of gravity in the model is the nonlinear equation that results from an accurate representation. A specific challenge in introducing gravity is linearizing the equation for a range of angles greater than 0-90 degrees. Despite the lack of these factors, our model still demonstrates a method of optimization for PID-controlled IPMC arm.

References

[1] Prajan, A., Krishnan, N., Srinivas, P., & Vigneswaran, N. (n.d.). Design, Development and Implementation of Neurologically Controlled Prosthetic Limb Capable of Performing Rotational Movement. *IEEE,* 241-244.
[2] Aw, K., Fu, L., & Mcdaid, A. (2013). An IPMC actuated robotic surgery end effector with force sensing. *International Journal of Smart and Nano Materials,* 1-11.
[3] Jain, R., Datta, S., Mukherjee, S., Sadhu, D., Samanta, S., & Banerjee, K. (n.d.). Bio-mimetic Behaviour of IPMC Artificial Muscle Using EMG Signal. *IEEE,* 186-190.
[4] Transfer functions. (n.d.). Retrieved November 13, 2014, from http://eent3.lsbu.ac.uk/units/b3embsys/Week 8 TransferFunction.htm
[5] Ari, K., Asal, F., & Cosgun, M. (n.d.). PI, PD, PID Controllers. Retrieved November 13, 2014, from http://www.eee.metu.edu.tr/~ee402/2012/EE402RecitationReport_4.pdf
[6] Jain, R., Datta, S., & Majumder, S. (2012). *Design and Control of an EMG Driven IPMC Based Artificial Muscle Finger*. INTECH Open Access Publisher.
[7] Takaya, K. (Director) (2003, January). Review of Analog Controller Design. Lecture conducted from University of Saskatchewan.
[8] An Introduction To System Dynamics - Second Order Systems. (n.d.). Retrieved November 20, 2014, from http://www.facstaff.bucknell.edu/mastascu/econtrolhtml/sysdyn/sysdyn2.html
[9] Time Response. (n.d.). Retrieved November 21, 2014, from http://aerostudents.com/files/aerospaceSystemsAndControlTheory/Book/ControlSystemsEngineeringCH4.pdf
[10] Boston Digital Arm System. (n.d.). Retrieved November 13, 2014, from http://www.liberatingtech.com/products/documents/Catalog_-_Boston_Digital_Arm_System_2013.pdf
[11] Kahyaoglu, O., & Unal, H. (2012). Friction and wear behaviours of medical grade UHMWPE at dry and lubricated conditions. *International Journal of Physical Sciences, 7*(16), 2478-2485.
[12] Cohen Coon Tuning Method. (n.d.). Retrieved November 13, 2014, from http://www.chem.mtu.edu/~tbco/cm416/cctune.html

Appendix: Matlab Code

```
clear all
y=0
%MANUAL TUNING OF BASIC ARM MODEL
%Manual tuning of arm (without band)
s = tf('s')
P = (1.1545)/(((.039516)*(s^2))+s);

%Establish controller parameters, and convolute with arm transfer function.
Kp = 3500;
Ki = 0;
Kd = 140;
C = pid(Kp, Ki, Kd)
T = feedback(C*P,1)

%Setup time vector and plot step response of entire system for time vector
%t.
figure
t = 0:0.00001:1.5;
step(T,t);
y=step(T,t);


%Overshoot
over=max(y);
if over>1
   percent_overshoot_noloadcontrolled=(over-1)/y(length(t))
end

%Steady state error
final=y(length(t));
steady_state_error_noloadcontrolled = (final-1)/1

%Rise time
x=false;
z=false;
for i=1:length(t);
   if y(i)>=.1*final;
      if x==false&z==false;
         x=true;
         t10=t(i);
      end
   end
   if y(i)>=.9*final;
      if x==true&z==false;
         z=true;
         t90=t(i);
```

```
      end
   end
end
rise_time_noloadcontrolled=t90-t10


%COHEN-COON OPTIMIZATION OF ARM WITH RESISTIVE BAND

%Setup arm transfer function (with band).
%s = tf('s');
P = (1.1545)/(((.039516)*(s^2)) + s + 5);

%Setup time vector and plot step response of entire system for time vector
%t.
t = 0:0.00001:25;
figure
step(P,t)

%Cohen Coon optimization of PID controller. Get open loop response of the
%arm, and measure time values where the output is 50% and 63.2% of the
%settled value is reached.
y=step(P,t);
x=false;
z=false;
t_0=t(1);
settle_value=y(length(t))
for i=1:length(t);
   if x==false;
      if y(i)>= settle_value/2;
         x=true;
         t_2=t(i);
      end
   end
   if z==false;
      if y(i)>= settle_value*(.632);
         z=true;
         t_3=t(i);
      end
   end
end

%Overshoot
over=max(y);
if over>1
   percent_overshoot_loaduncontrolled=(over-1)/y(length(t))
end

%Steady state error
```

```
final=y(length(t));
steady_state_error_loaduncontrolled = (final-1)/1

%Rise time
x=false;
z=false;
for i=1:length(t);
  if y(i)>=.1*final;
     if x==false&z==false;
        x=true;
        t10=t(i);
     end
  end
  if y(i)>=.9*final;
     if x==true&z==false;
        z=true;
        t90=t(i);
     end
  end
end
rise_time_loaduncontrolled=t90-t10

%Calculate K as output settle value divided by input,
%which is 1 (unit step). Other constants specified by Cohen-Coon method.
K=settle_value/1;
t_1=(t_2-(log(2)*t_3))/(1-(log(2)));
tor=t_3-t_1;
t_del=t_1-t_0;
r=t_del/tor;

Kp=(1/(K*r))*((4/3)+(r/4))
Ki=(13+8*r)/(t_del*(32+6*r))
Kd=(11+2*r)/(4*t_del)

%Convolute calculated PID controller with system and graph response.
C = pid(Kp, Ki, Kd);
T = feedback(C*P,1);
figure
step(T, t);
y=step(T,t);

%Overshoot
over=max(y);
if over>1
  percent_overshoot_loadcohen=(over-1)/y(length(t))
end

%Steady state error
```

```
final=y(length(t));
steady_state_error_loadcohen = (final-1)/1

%Rise time
x=false;
z=false;
for i=1:length(t);
  if y(i)>=.1*final;
     if x==false&z==false;
        x=true;
        t10=t(i);
     end
  end
  if y(i)>=.9*final;
     if x==true&z==false;
        z=true;
        t90=t(i);
     end
  end
end
rise_time_loadcohen=t90-t10
```