Department of Mechanical and Mechatronics Engineering

ME 360 – Introduction to Control Systems

Cart Pole Position Tracking Mini Project

Prepared by:

Amaar Quadri

Corwin MacMillan

Carter Ward

Fahmid Kader

Sepehr Talebi

Prepared for:

Arash Arami

Submitted on:

Friday, November 27, 2020

Table of Contents

List of Figures 2

Summary 3

1 Deriving the Equations of Motion 4

1.1 Assumptions and Free Body Diagram 4

1.2 Finding the Lagrangian 4

1.3 Getting the Equations of Motion 5

1.4 Linearizing the Equations of Motion for $\theta = 0$ 6

1.5 Calculating the Transfer Functions for $\theta = 0$ 6

1.6 Linearizing the Equations of Motion for $\theta = \pi$ 6

1.7 Calculating the Transfer Functions for $\theta = \pi$ 7

2 Parameter Discovery 7

2.1 Finding M and L 7

2.2 Finding b 9

2.3 Finding M, L and b, using Initial and Final Value Theorems 10

2.4 Finding m 11

2.5 Verification of System Parameters using the Linearized System 13

3 Controller Design 14

3.1 Controller Strategy 14

3.2 Gain Tuning via Routh-Hurwitz Stability Criterion 16

3.3 Gain Tuning via Linear-Quadratic Regulator 16

3.4 Stability 18

3.5 Controller Behavior 20

3.5.1 MATLAB Graphs 20

3.5.2 Python Graphs 21

3.5.3 Force and Power Draw Requirements 23

3.6 Issues and Further Improvements 24

3.6.1 Steady State Error due to Ramp Input 24

3.6.2 Steady State Error due to Disturbance 25

3.6.3 Final Thoughts 25

References 26

Appendix 26

Routh-Hurwitz Table Entries 26

Python Code 27

List of Figures

Figure 1: Free Body Diagram 4
Figure 2: $x(t)$ for $f(t) = 1(t)$ 8
Figure 3: $\theta(t)$ for $f(t) = 1(t)$ 8
Figure 4: $x(t)$ for $f(t) = 1(t)$ 9
Figure 5: $\theta(t)$ for $f(t) = 1(t)$ 9
Figure 6: MATLAB Diagram for computing $\int_0^T \dot{x}^2\,dt$ 12
Figure 7: $x(t)$ for $f(t) = 1(t)$ 12
Figure 8: $\int_0^t \dot{x}^2\,\mathrm{dt}'$ for $f(t) = 1(t)$ 12
Figure 9: Block Diagram for Verifying System Parameters with Linearized System 13
Figure 10: Error due to Linearized System 13
Figure 11: $x(t)$ and $\theta(t)$ for $f(t) = 0.1sin(t)$ 14
Figure 12: PD Controller Block Diagram 15
Figure 13: Hand Tuning of Gains based on Routh-Hurwitz Stability Criteria 16
Figure 14: Routh-Hurwitz Criteria Applied to Linear-Quadratic Regulator Chosen Gains 18
Figure 15: Pole-Zero Plot for C 19
Figure 16: Pole-Zero Plot for D 19
Figure 17: Controller Response for $x_r(t) = 1(t)$ 20
Figure 18: Controller response for $x_r(t) = sin\left(\frac{\pi}{5}t\right)$ 20
Figure 19: Controller Response for $x_r(t) = t * 1(t)$ 21
Figure 20: Controller Response for $x_r(t) = 1(t)$ with Disturbance 21
Figure 21: Controller Recovery from $\theta_0 = 75°$ 22
Figure 22: Recovery from $x_0 = 15m$ 22
Figure 23: Recovery from $\dot{x}_0 = 15m/s$ 23
Figure 24: Recovery from $\dot{\theta}_0 = 5rev/s$ 23

Summary

This project aims to analyze a cart-pole system and design a controller to stabilize it with the pendulum in the inverted state and track a desired input trajectory, x_r. We derive the equations of motion using Lagrangian mechanics, linearize the equations of motion, and calculate transfer functions for $\theta = 0$ and $\theta = \pi$. We then reverse engineer the key system parameters using a combination of the initial value theorem, Newton's second law, and conservation of energy; and we verify the results by comparing it to the original system. We then design a PD controller by hand using the Routh-Hurwitz criteria and then optimize the parameters further using the linear-quadratic regulator algorithm. We prove that our controller is stable by creating a pole-zero plot. We demonstrate the position tracking capabilities of our controller and its ability to recover from poor initial conditions. We also discuss the required force and power requirements of the system and discuss the effect of saturation. Finally, we the calculate the steady state error when subject to both a ramp input and a constant disturbing force; and we suggest ways that these errors can be eliminated.

1 Deriving the Equations of Motion

1.1 Assumptions and Free Body Diagram

We modelled the system as a point mass at the tip of a massless pendulum which is connected to the cart. The cart is represented with a point mass at the base of the pendulum. A viscous damping force as well as an applied external force is applied to the cart. The cart is restricted to moving horizontally so gravity is only applied to the pole mass, m. We assume that torsional friction at the pole joint is negligeable.

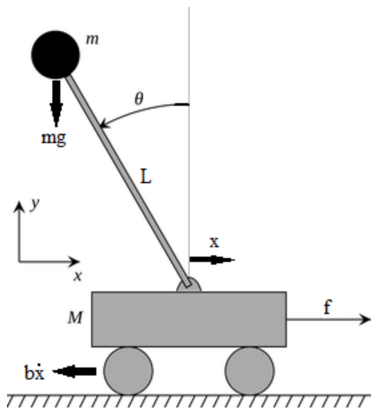


Figure 1: Free Body Diagram

1.2 Finding the Lagrangian

From geometry we have

$$x_{rel,pole} = -L\sin(\theta)$$

$$y_{rel,pole} = L\cos(\theta)$$

Where the subscript 'rel' refers to the relative displacement of the pole's tip from the cart's position. Taking the derivative of each of these with respect to time gives

$$\dot{x}_{rel,pole} = -L\dot{\theta}\cos(\theta)$$

$$\dot{y}_{rel,pole} = -L\dot{\theta}\sin(\theta)$$

Combining these relative velocities between the cart and the pole with the overall velocity of the cart gives us the absolute velocities of the center of mass of the pole to be

$$\dot{x}_{pole} = \dot{x} + \dot{x}_{rel,pole} = \dot{x} - L\dot{\theta}\cos(\theta)$$

$$\dot{y}_{pole} = \dot{y}_{rel,pole} = -L\dot{\theta}\sin(\theta)$$

We can now calculate the total kinetic energy of the pole to be

$$KE_{pole} = \frac{1}{2}m\dot{x}_{pole}^2 + \frac{1}{2}m\dot{y}_{pole}^2$$

$$KE_{pole} = \frac{1}{2}m\left(\dot{x} - L\dot{\theta}\cos(\theta)\right)^2 + \frac{1}{2}mL^2\dot{\theta}^2\sin^2(\theta)$$

$$KE_{pole} = \frac{1}{2}m\dot{x}^2 + \frac{1}{2}mL^2\dot{\theta}^2 - mL\dot{x}\dot{\theta}\cos(\theta)$$

The kinetic energy of the cart is simply

$$KE_{cart} = \frac{1}{2}M\dot{x}^2$$

So, the total kinetic energy is

$$KE = KE_{pole} + KE_{cart}$$

$$KE = \frac{1}{2}m\dot{x}^2 + \frac{1}{2}mL^2\dot{\theta}^2 - mL\dot{x}\dot{\theta}\cos(\theta) + \frac{1}{2}M\dot{x}^2$$

The potential energy of the system is simply

$$PE = PE_{cart} = mgy_{cart} = mgL\cos(\theta)$$

Finally, the Lagrangian can be calculated as

$$L = KE - PE$$

$$L = \frac{1}{2}m\dot{x}^2 + \frac{1}{2}mL^2\dot{\theta}^2 - mL\dot{x}\dot{\theta}\cos(\theta) + \frac{1}{2}M\dot{x}^2 - mgL\cos(\theta)$$

1.3 Getting the Equations of Motion

The equations of motion can be determined from the Lagrange equation of the second kind. In our case, these equations are the following

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) = \frac{\partial L}{\partial x} + F_{external}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) = \frac{\partial L}{\partial \theta}$$

In our case, the $F_{external}$ is due to the control input and the viscous damping

$$F_{external} = f - b\dot{x}$$

Where f is the control output force that is applied to the cart.

Plugging in the equation for the Lagrangian and taking the partial derivatives yields the following two equations

$$m\ddot{x} - mL\ddot{\theta}\cos(\theta) + mL\dot{\theta}^2\sin(\theta) + M\ddot{x} = f - b\dot{x}$$

$$mL^2\ddot{\theta} - mL\ddot{x}\cos(\theta) + mL\dot{x}\dot{\theta}\sin(\theta) = mL\dot{x}\dot{\theta}\sin(\theta) + mgL\sin(\theta)$$

This is now a system of equations which can be solved for $\ddot{x}$ and $\ddot{\theta}$. This yields the final equations of motion

$$\ddot{x} = \frac{-mL\dot{\theta}^2\sin(\theta) - b\dot{x} + \frac{mg}{2}\sin(2\theta) + f}{M + m\sin^2(\theta)}$$

$$\ddot{\theta} = \frac{(M+m)g\sin(\theta) - \frac{mL}{2}\dot{\theta}^2\sin(2\theta) + (f - b\dot{x})\cos(\theta)}{L[M + m\sin^2(\theta)]}$$

1.4 Linearizing the Equations of Motion for $\theta = 0$

For the controller design, we want to linearize the equations of motion about the operating point

$$x = \dot{x} = \theta = \dot{\theta} = f = 0$$

This can be done by applying the following first order Maclaurin series approximations

$$\ddot{x} \approx \ddot{x}|_{op} + \frac{\partial \ddot{x}}{\partial x}\bigg|_{op}(x-0) + \frac{\partial \ddot{x}}{\partial \dot{x}}\bigg|_{op}(\dot{x}-0) + \frac{\partial \ddot{x}}{\partial \theta}\bigg|_{op}(\theta-0) + \frac{\partial \ddot{x}}{\partial \dot{\theta}}\bigg|_{op}(\dot{\theta}-0) + \frac{\partial \ddot{x}}{\partial f}\bigg|_{op}(f-0)$$

$$\ddot{\theta} \approx \ddot{\theta}\Big|_{op} + \frac{\partial \ddot{\theta}}{\partial x}\bigg|_{op}(x-0) + \frac{\partial \ddot{\theta}}{\partial \dot{x}}\bigg|_{op}(\dot{x}-0) + \frac{\partial \ddot{\theta}}{\partial \theta}\bigg|_{op}(\theta-0) + \frac{\partial \ddot{\theta}}{\partial \dot{\theta}}\bigg|_{op}(\dot{\theta}-0) + \frac{\partial \ddot{\theta}}{\partial f}\bigg|_{op}(f-0)$$

Evaluating the partial derivatives yields the following simplified linear equations of motion

$$\ddot{x} = -\frac{b}{M}\dot{x} + \frac{mg}{M}\theta + \frac{1}{M}f$$

$$\ddot{\theta} = -\frac{b}{ML}\dot{x} + \frac{(M+m)g}{ML}\theta + \frac{1}{ML}f$$

1.5 Calculating the Transfer Functions for $\theta = 0$

Taking the Laplace transform of the linearized equations of motion yields the following

$$s^2X = -\frac{b}{M}sX + \frac{mg}{M}\Theta + \frac{1}{M}F$$

$$s^2\Theta = -\frac{b}{ML}sX + \frac{(M+m)g}{ML}\Theta + \frac{1}{ML}F$$

This is now a system of equations which can be solved for the two transfer functions $\frac{X}{F}$ and $\frac{\Theta}{F}$ resulting in the following transfer functions

$$G \equiv \frac{X}{F} = \frac{Ls^2 - g}{s[MLs^3 + Lbs^2 - (M+m)gs - gb]}$$

$$H \equiv \frac{\Theta}{F} = \frac{s}{MLs^3 + Lbs^2 - (M+m)gs - gb}$$

1.6 Linearizing the Equations of Motion for $\theta = \pi$

The linearized equations of motion can be found using the same first order Maclaurin series approximation as in the previous case except with the following operating point

$$x = \dot{x} = \dot{\theta} = f = 0, \theta = \pi$$

This yields the following linearized equations of motion

$$\ddot{x} = -\frac{b}{M}\dot{x} + \frac{mg}{M}(\theta - \pi) + \frac{1}{M}f$$

$$\ddot{\theta} = \frac{b}{ML}\dot{x} - \frac{(M+m)g}{ML}(\theta - \pi) - \frac{1}{ML}f$$

Since these equations have $(\theta - \pi)$ we must apply the substitution $\theta' = (\theta - \pi)$ before proceeding. This yields the following linearized equations of motion for the modified coordinate system

$$\ddot{x} = -\frac{b}{M}\dot{x} + \frac{mg}{M}\theta' + \frac{1}{M}f$$

$$\ddot{\theta}' = \frac{b}{ML}\dot{x} - \frac{(M+m)g}{ML}\theta' - \frac{1}{ML}f$$

Where we used the fact that

$$\ddot{\theta}' = \frac{\partial^2}{\partial t^2}(\theta - \pi) = \ddot{\theta}$$

1.7 Calculating the Transfer Functions for $\theta = \pi$

Taking the Laplace transform of the linearized equations of motion in the modified coordinate system yields the following

$$s^2 X = -\frac{b}{M}sX + \frac{mg}{M}\Theta' + \frac{1}{M}F$$

$$s^2\Theta' = \frac{b}{ML}sX - \frac{(M+m)g}{ML}\Theta' - \frac{1}{ML}F$$

This is now a system of equations which can be solved for the two transfer functions $\frac{X}{F}$ and $\frac{\Theta'}{F}$ resulting in the following transfer functions

$$G_\pi = \frac{X}{F} = \frac{Ls^2 + g}{s[MLs^3 + Lbs^2 + (M+m)gs + gb]}$$

$$H_\pi = \frac{\Theta'}{F} = \frac{-s}{MLs^3 + Lbs^2 + (M+m)gs + gb}$$

Where the subscript π indicates that these are linearized about $\theta = \pi$.

2 Parameter Discovery

2.1 Finding M and L

To find M and L we applied a step input to the plant in the configuration where the pole starts facing downwards.

$$f(t) = 1(t)$$

Since at $t = 0$, we have $x = \dot{x} = \dot{\theta} = 0, \theta = \pi, f = 1$ we can substitute these values into the original (nonlinearized) equations of motion to get

$$\ddot{x}(0) = \frac{f}{M}$$

$$\ddot{\theta}(0) = -\frac{f}{ML}$$

This can be rearranged to give

$$M = \frac{f}{\ddot{x}(0)}$$

$$L = -\frac{\ddot{x}(0)}{\ddot{\theta}(0)}$$

The following two plots show the resulting graphs for $\ddot{x}$ and $\ddot{\theta}$

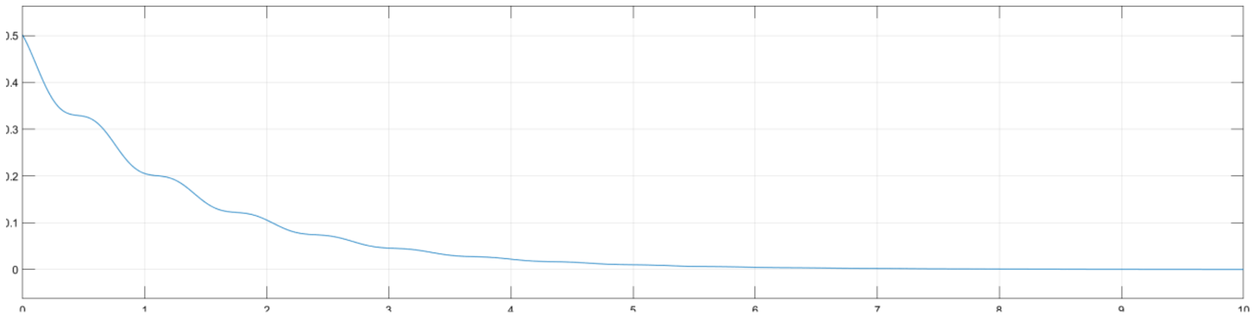


Figure 2: $\ddot{x}(t)$ *for* $f(t) = 1(t)$

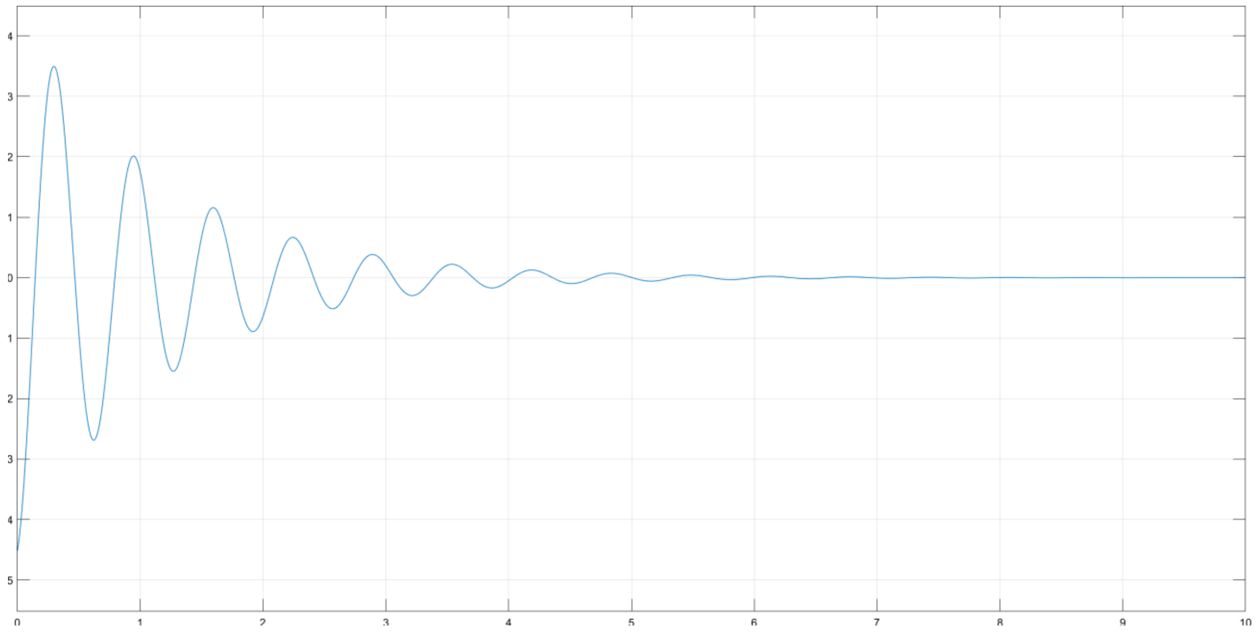


Figure 3: $\ddot{\theta}(t)$ *for* $f(t) = 1(t)$

The graphs show that

$$\ddot{x}(0) = 0.501410$$

$$\ddot{\theta}(0) = -4.517376$$

Thus,

$$M = \frac{1}{0.501410} = 1.994376$$

$$L = -\frac{0.501410}{-4.517376} = 0.110996$$

2.2 Finding b

When subject to the step function as the input force, the system reaches steady state after a few seconds. At that point, $\ddot{x}_{ss} = \dot{\theta}_{ss} = \ddot{\theta}_{ss} = 0$ and $\theta_{ss} = \pi$ based on physical intuition as well as the following two graphs.

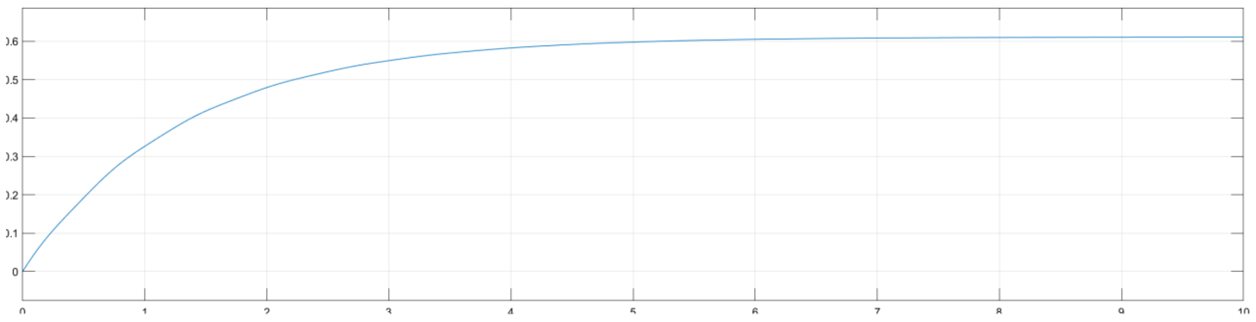


Figure 4: $\dot{x}(t)$ for $f(t) = 1(t)$

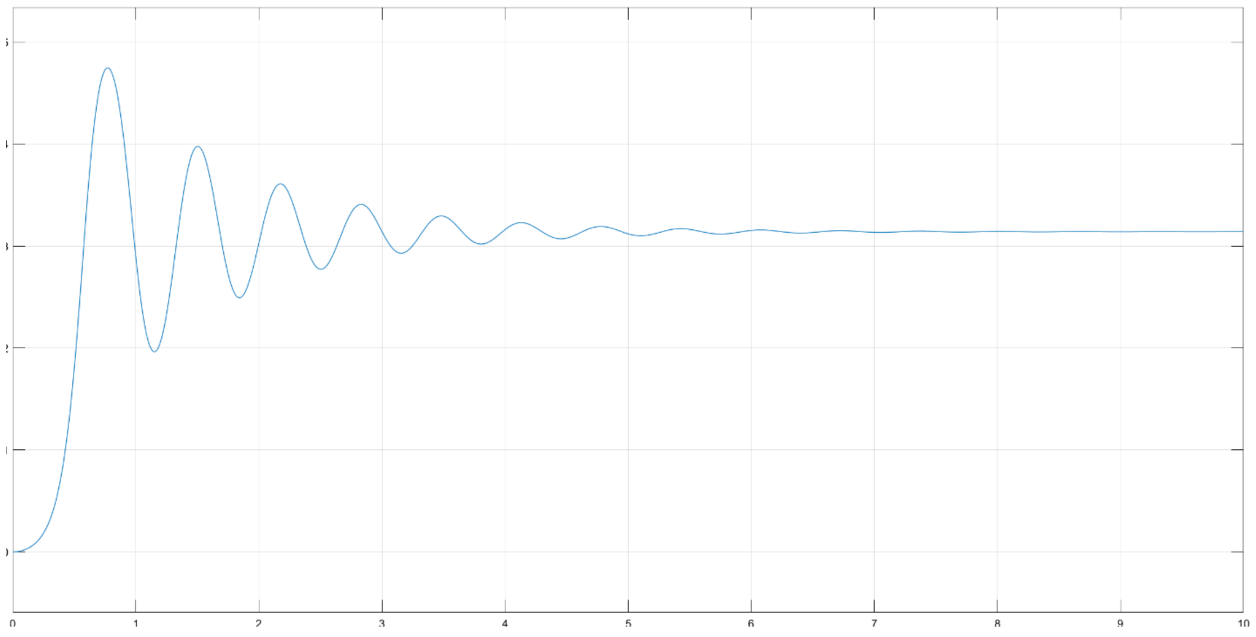


Figure 5: $\theta(t)$ for $f(t) = 1(t)$

Since θ stabilizes at the value of π, it is not changing and thus no longer relevant at this point. Thus, the system can be treated as consisting of a single mass of $(M + m)$. From Newton's second law we have

$$\Sigma F = f - b\dot{x}_{ss} = (M+m)\ddot{x}_{ss} = 0$$

$$b = \frac{f}{\dot{x}_{ss}}$$

From the velocity graph, we can see that $\dot{x}_{ss} = 0.6113$. Thus,

$$b = \frac{1}{0.6113} = 1.6359$$

2.3 Finding M, L and b, using Initial and Final Value Theorems

The equations above that were used for calculating M, L and b can also be derived using the initial and final value theorems. For finding M, we start with

$$X = G_\pi F = \frac{Ls^2 + g}{s^2(MLs^3 + Lbs^2 + (M+m)gs + gb)}$$

$$Xs^2 = \frac{Ls^2 + g}{MLs^3 + Lbs^2 + (M+m)gs + gb}$$

Applying the initial value theorem gives

$$\lim_{t\to 0} \ddot{x}(t) = \lim_{s\to\infty} \frac{Ls^3 + gs}{MLs^3 + Lbs^2 + (M+m)gs + gb}$$

$$\ddot{x}(0) = \frac{1}{M}$$

Which matches the equation used above to find M. For finding L we start with

$$\Theta' = H_\pi F = \frac{-1}{MLs^3 + Lbs^2 + (M+m)gs + gb}$$

$$\Theta' s^2 = \frac{-s^2}{MLs^3 + Lbs^2 + (M+m)gs + gb}$$

Applying the initial value theorem gives

$$\lim_{t\to 0} \ddot{\theta}(t) = \lim_{s\to\infty} \frac{-s^3}{MLs^3 + Lbs^2 + (M+m)gs + gb}$$

$$\ddot{\theta}'(0) = \frac{-1}{ML}$$

Which matches the equation used to calculate L. For finding b we start with

$$X = G_\pi F = \frac{Ls^2 + g}{s^2(MLs^3 + Lbs^2 + (M+m)gs + gb)}$$

$$Xs = \frac{Ls^2 + g}{s(MLs^3 + Lbs^2 + (M+m)gs + gb)}$$

Applying the final value theorem gives

$$\lim_{t\to\infty}\dot{x}(t)=\lim_{s\to 0}\frac{Ls^2+g}{MLs^3+Lbs^2+(M+m)gs+gb}$$

$$\dot{x}(\infty)=\frac{1}{b}$$

Which matches the equation that was used to find b.

2.4 Finding m

During the 10 seconds for which the above simulation was run. Energy was being added to the system by the applied force, f, and energy was being removed from the system by the viscous damping. Also, the system starts with no kinetic energy and has no change in potential energy between the start and the end. Thus, the final kinetic energy can be determined by energy balance.

$$W_f-W_b=KE$$

$$\int_{x_0}^{x_f} f\,dx-\int_{x_0}^{x_f} b\dot{x}\,dx=\frac{1}{2}(M+m)\dot{x}_{ss}^2$$

Since $f=1$

$$[x_f-x_0]-\int_{x_0}^{x_f} b\dot{x}\,dx=\frac{1}{2}(M+m)\dot{x}_{ss}^2$$

Since $x_0=0, x_f=x(T)$

$$x(T)-\int_{x_0}^{x_f} b\dot{x}\,dx=\frac{1}{2}(M+m)\dot{x}_{ss}^2$$

Since

$$\dot{x}=\frac{dx}{dt}\to dx=\dot{x}dt$$

$$when\ x=x_0\to t=0$$

$$when\ x=x_f\to t=T$$

$$x(T)-b\int_0^T \dot{x}^2dt=\frac{1}{2}(M+m)\dot{x}_{ss}^2$$

$$m=\frac{2}{\dot{x}_{ss}^2}\left[x(T)-b\int_0^T \dot{x}^2dt\right]-M$$

The remaining integral can be calculated by numerically integrating as the simulation runs. This can be done in MATLAB via the following

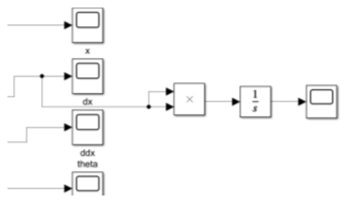


Figure 6: MATLAB Diagram for computing $\int_0^T \dot{x}^2 dt$

In our case, the simulation was run for $T = 10$ seconds. The following graph shows x over time

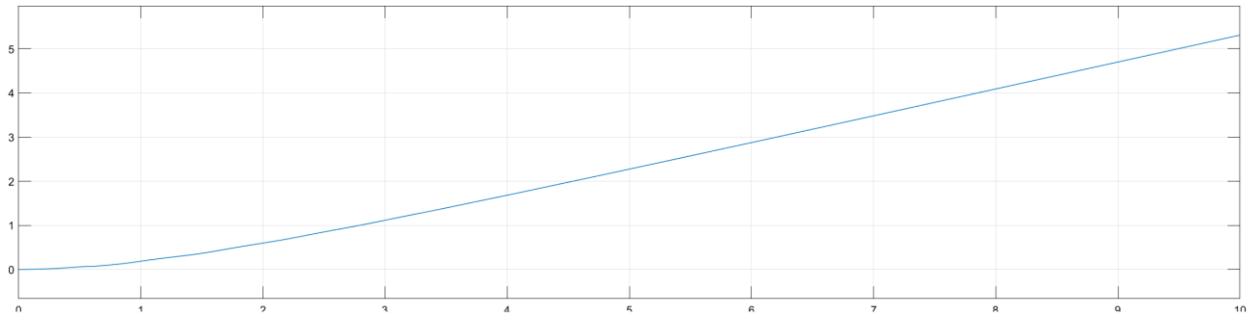


Figure 7: $x(t)$ for $f(t) = 1(t)$

From which it can be seen that $x(T) = x(10) = 5.315149$

The following graph shows the integral $\int_0^t \dot{x}^2 dt'$ over time

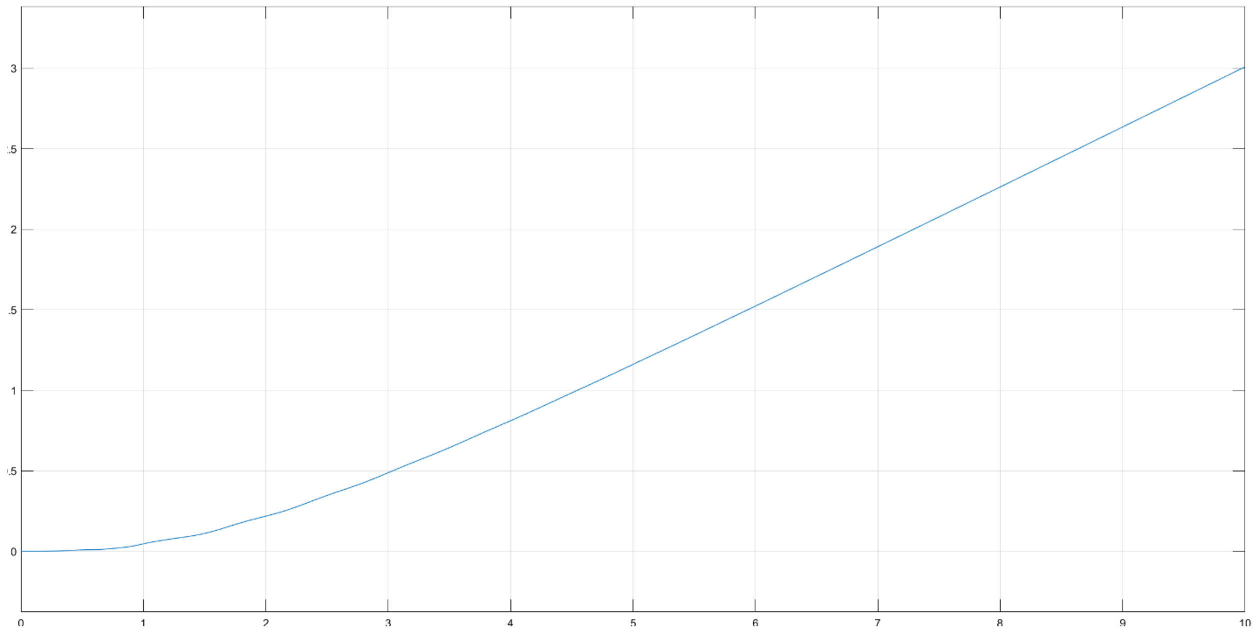


Figure 8: $\int_0^t \dot{x}^2 dt'$ for $f(t) = 1(t)$

From which it can be seen that $\int_0^T \dot{x}^2 dt = \int_0^{10} \dot{x}^2 dt = 3.009239$

Substituting all the relevant values into the equation gives a value for m of

$$m = \frac{2}{0.6113^2}[5.315149 - (1.6359)(3.009239)] - 1.994376 = 0.105425$$

Thus, in summary

$$M = 1.994376, L = 0.110996, b = 1.6359, m = 0.105425$$

2.5 Verification of System Parameters using the Linearized System

The following figures shows the MATLAB block diagram that was used to verify our system parameters, and the resulting graphs. This was done by inputting a force of $f(t) = 0.1\sin(t)$ and graphing the difference between the x and θ' output of the real system and the system linearized about $\theta = \pi$.

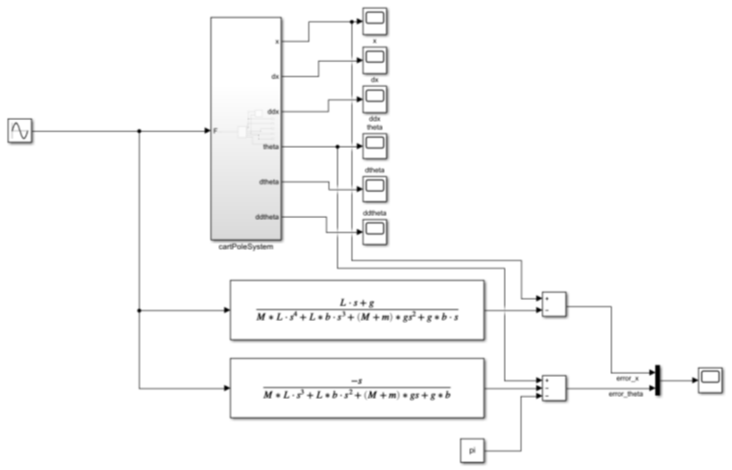

x
dx
ddx
theta
dtheta
ddtheta
F
cartPoleSystem
$\frac{L\cdot s+g}{M*L\cdot s^4+L*b\cdot s^3+(M+m)*gs^2+g*b\cdot s}$
$\frac{-s}{M*L\cdot s^3+L*b\cdot s^2+(M+m)*gs+g*b}$
pi
error_x
error_theta

Figure 9: Block Diagram for Verifying System Parameters with Linearized System

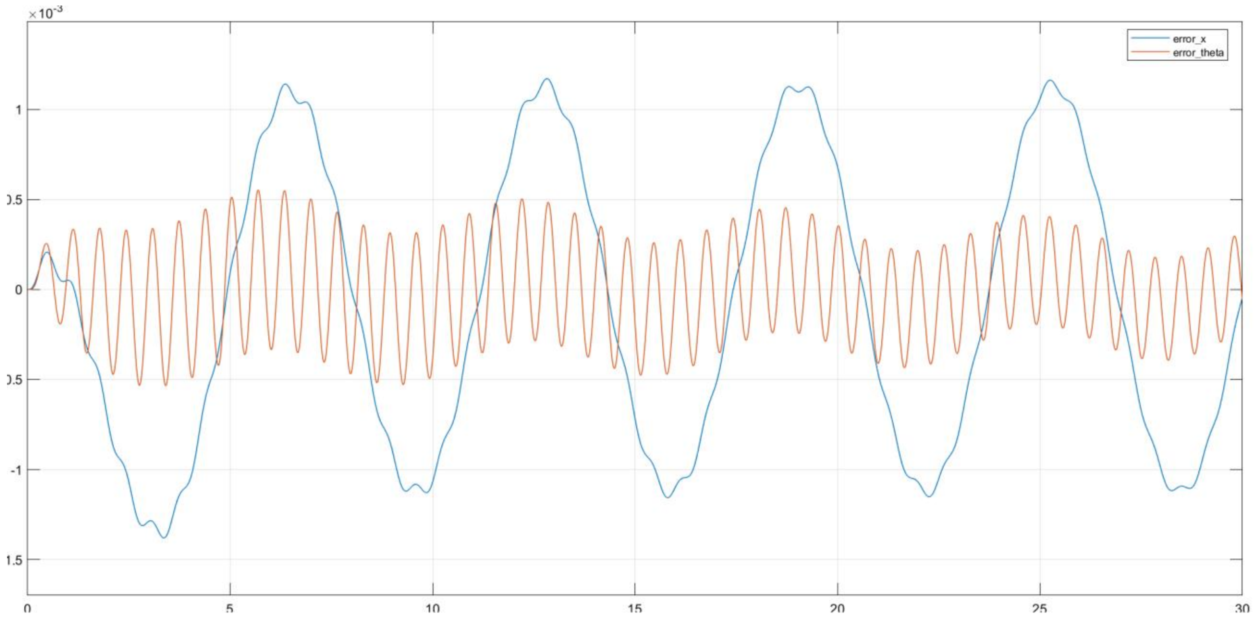


Figure 10: Error due to Linearized System

As shown in Figure 10, the error is less than $1.5 \cdot 10^{-3}$ which shows that our system parameters are very accurate. Moreover, running the simulation again with a 10 times smaller force of $f(t) = 0.01\sin(t)$

leads to a 10 times smaller error of less than $1.5 \cdot 10^{-4}$. This suggests that the remaining error is due to the approximation inherent in the linearization as opposed to any error in the system parameters. This can also be confirmed by the following graph which shows that $x \approx 0, \theta \approx \pi, \dot{x} \approx 0, \dot{\theta} \approx 0$. These values are relatively close to, but not exactly, the values we linearized about which explains the small but non-zero error.

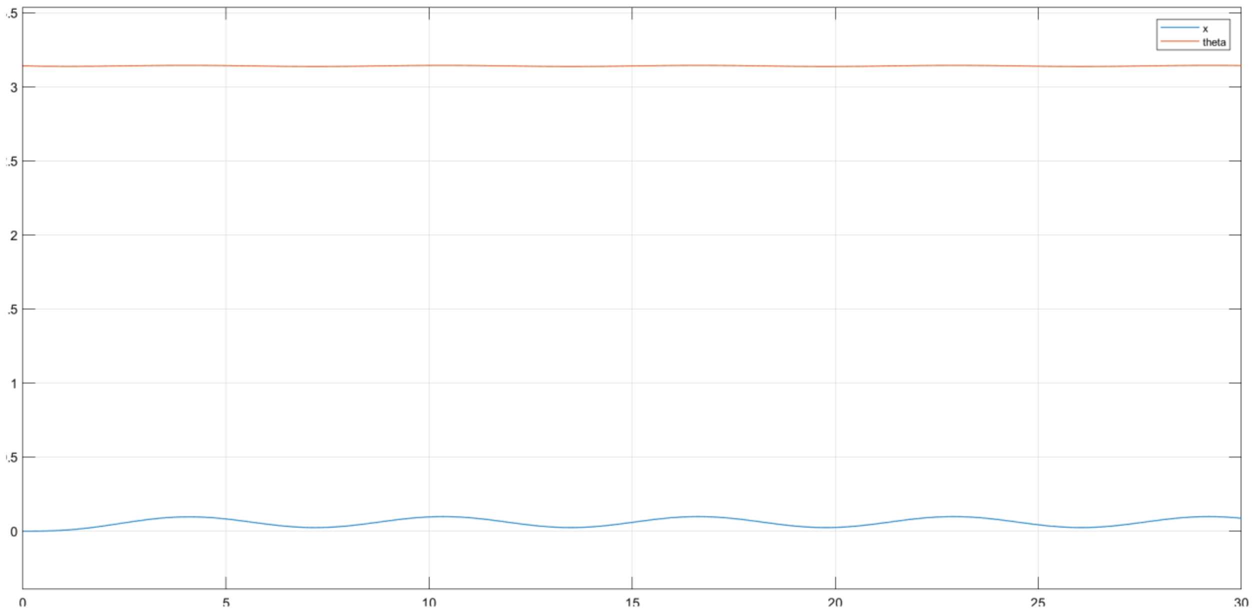


Figure 11: $x(t)$ and $\theta(t)$ for $f(t) = 0.1\, sin(t)$

3 Controller Design

3.1 Controller Strategy

Our controller strategy is to use a PD controller that takes in both the x and θ inputs as well as their derivatives, and combines them to produce a single output force, f, which serves as the input to both the X and Θ transfer functions. The goal is then to ensure suitable output behavior for both x and θ via this single controller. The block diagram for this controller is shown below.

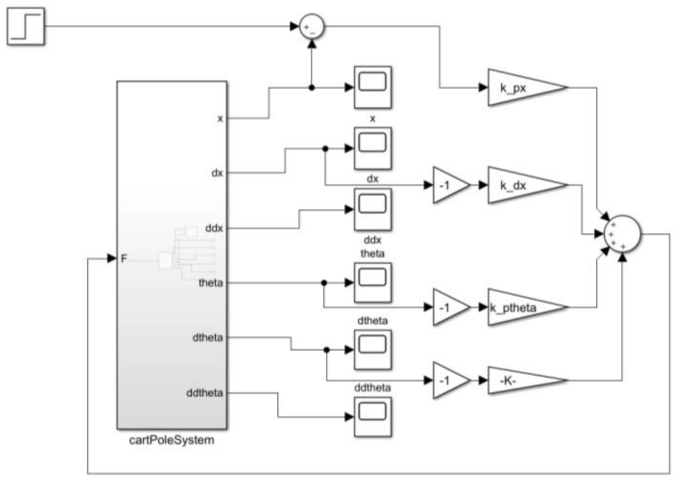


Figure 12: PD Controller Block Diagram

From this block diagram, it can be seen that the input force is given by

$$f = k_{p,x}(x_r - x) + k_{d,x}(\dot{x}_r - \dot{x}) + k_{p,\theta}(\theta_r - \theta) + k_{d,\theta}(\dot{\theta}_r - \dot{\theta})$$

Where x_r is the desired tracking for the x value of the system. Since the desired values for the other variables are zero, we have $\dot{x}_r = \theta_r = \dot{\theta}_r = 0$. So, the force can be re-written as

$$f = k_{p,x}(x_r - x) - k_{d,x}\dot{x} - k_{p,\theta}\theta - k_{d,\theta}\dot{\theta}$$

The reason behind this formulation and the negative gains in front of $\dot{x}, \theta$, and $\dot{\theta}$ are so that all four of the variables are treated on equal footing which simplifies the calculations. Taking the Laplace transform of both sides yields

$$F = k_{p,x}(X_r - X) - k_{d,x}sX - k_{p,\theta}\Theta - k_{d,\theta}s\Theta$$

$$F = k_{p,x}X_r - (k_{p,x} + k_{d,x}s)X - (k_{p,\theta} + k_{d,\theta}s)\Theta$$

This is effectively the transfer function of our controller, although it has 3 inputs instead of 1. Combining this with the linearized transfer functions of the plant, G and H, gives the following two equations

$$GF = G[k_{p,x}X_r - (k_{p,x} + k_{d,x}s)X - (k_{p,\theta} + k_{d,\theta}s)\Theta] = X$$

$$HF = H[k_{p,x}X_r - (k_{p,x} + k_{d,x}s)X - (k_{p,\theta} + k_{d,\theta}s)\Theta] = \Theta$$

This forms a system of equations that can be solved for the following transfer functions of the combined controller-plant system

$$C \equiv \frac{X}{X_r} = \frac{k_{p,x}(Ls^2 - g)}{MLs^4 + (Lk_{d,x} + k_{d,\theta} + Lb)s^3 + (Lk_{p,x} + k_{p,\theta} - (M+m)g)s^2 - (k_{d,x} + b)gs - gk_{p,x}}$$

$$D \equiv \frac{\Theta}{X_r} = \frac{k_{p,x}s^2}{MLs^4 + (Lk_{d,x} + k_{d,\theta} + Lb)s^3 + (Lk_{p,x} + k_{p,\theta} - (M+m)g)s^2 - (k_{d,x} + b)gs - gk_{p,x}}$$

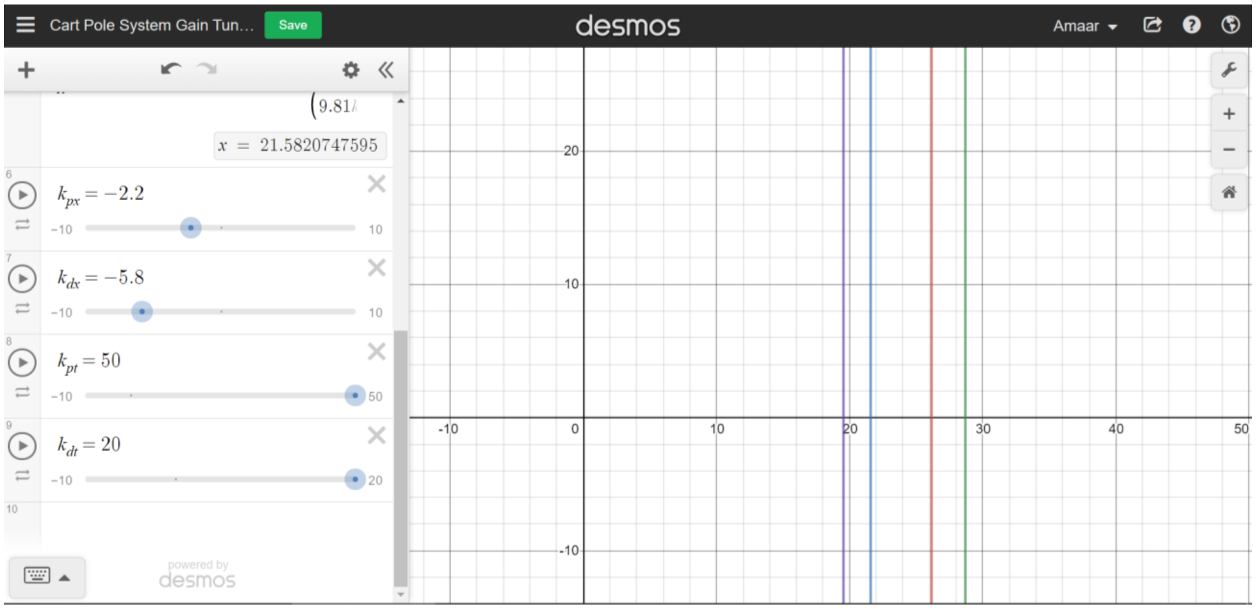
Now to get a functioning controller, we need to find values for the 4 gains such that all poles are located on the left half of the complex plane in a pole zero plot. Also, the fact that both of these transfer functions have the same denominator means that a set of gains that ensures that C has stable poles will automatically produce stable poles for D.

3.2 Gain Tuning via Routh-Hurwitz Stability Criterion

Given the denominator of the two transfer functions C and D, the Routh-Hurwitz stability criterion can be used to determine a set of inequalities that the gains must satisfy. The corresponding entries in the table become exceedingly complicated, so Python was used to generate the following table. The values for N_1, N_2, N_3, and N_4 can be found in the appendix.

s^4	0.221368	$k_{p,\theta} + 0.110996k_{p,x} - 20.599$	$-9.81 * k_{p,x}$
s^3	$k_{d,\theta} + 0.110996 * k_{d,x} + 0.181578$	$-9.81 * k_{d,x} - 16.0482$	0
s^2	N_1	N_2	0
s	N_3	0	0
1	N_4	0	0

The stability criterion is then that every entry in the first column must be positive. To satisfy these criteria by hand, the stability criteria were pasted into the online graphing calculator Desmos, and sliders were configured for each of the 4 gains. The gains were then tuned by hand until all the criteria were positive. The following screenshot shows the result. This graph can also be accessed by going to https://www.desmos.com/calculator/lzjifkd6vb.



$x = 21.5820747595$

$k_{px} = -2.2$ (slider −10 to 10)

$k_{dx} = -5.8$ (slider −10 to 10)

$k_{pt} = 50$ (slider −10 to 50)

$k_{dt} = 20$ (slider −10 to 20)

Figure 13: Hand Tuning of Gains based on Routh-Hurwitz Stability Criteria

3.3 Gain Tuning via Linear-Quadratic Regulator

Although the hand tuned gains would (and do) result in a stable controller, a more systematic approach to tuning the gains was desired. For this, the linear-quadratic regulator algorithm will be used as opposed to performing trial and error by hand. To set this up, first the linearized equations of motion for the system need to be put into the following matrix form

$$\frac{\partial}{\partial t}\begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = A \cdot \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} + B \cdot f$$

By comparing this form to the linearized equations of motion, it can be seen that the following values are necessary

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{b}{M} & \frac{mg}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{b}{ML} & \frac{(M+m)g}{ML} & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ \frac{1}{ML} \end{bmatrix}$$

Also, the algorithm requires 2 parameters Q and R which define what is being optimized. Q represents the relative importance of the four parameters $x, \dot{x}, \theta, \dot{\theta}$, and R is a regularization term that prevents the controller from outputting arbitrarily large forces. Formally, the resulting controller aims to minimize the following integral

$$\int_0^\infty \left(\begin{bmatrix} x & \dot{x} & \theta & \dot{\theta} \end{bmatrix} \cdot Q \cdot \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} + f \cdot R \cdot f \right) dt$$

We chose the following values for Q and R which emphasize the importance of the maintaining θ close to zero and have a relatively weak bound on the maximum allowed force.

$$Q = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 50 \end{bmatrix}$$

$$R = 1$$

The linear-quadratic regulator algorithm used the matrices A, B, Q, and R to determine the following controller gains

$$K = \begin{bmatrix} -3.162278 \\ -8.718722 \\ 63.239034 \\ 9.624838 \end{bmatrix}$$

The optimal controller is then given by

$$f = -K \cdot \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} = K \cdot \begin{bmatrix} -x \\ -\dot{x} \\ -\theta \\ -\dot{\theta} \end{bmatrix}$$

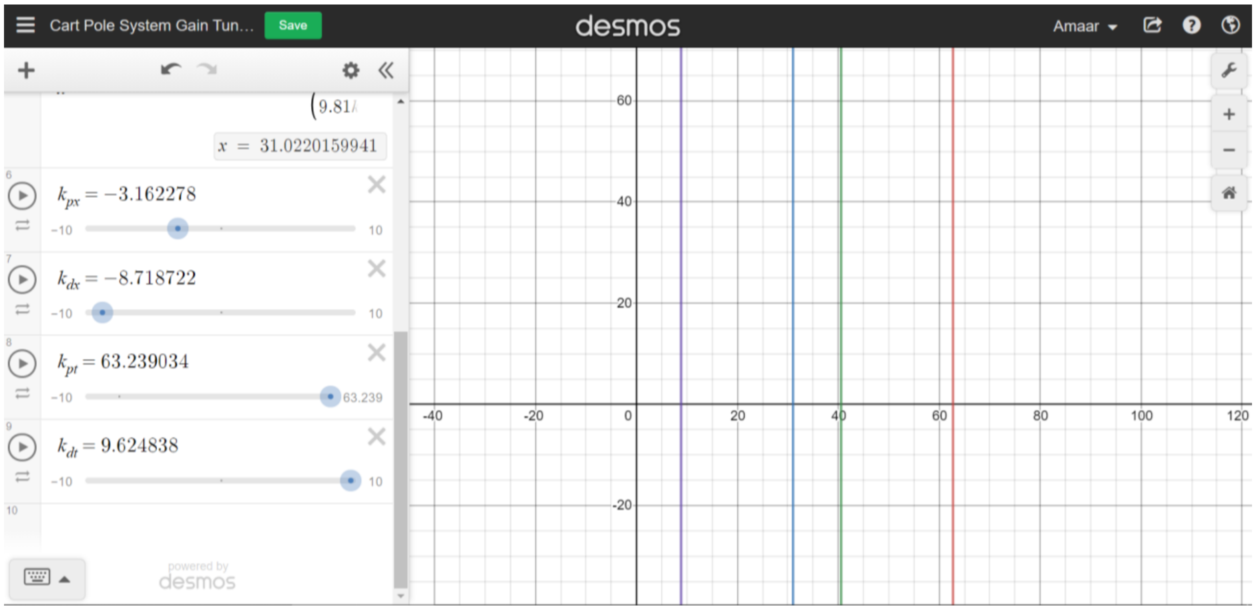
However, this does not account for the x position tracking aspect of the problem. Instead it attempts to maintain $x = 0$. Luckily, since the equations of motion do not depend on the x variable, the desired tracking functionality can be achieved by simply shifting the input to the controller by x_r. This effectively "tricks" the controller into thinking that x_r is actually 0.

$$f = K \cdot \begin{bmatrix} x_r - x \\ -\dot{x} \\ -\theta \\ -\dot{\theta} \end{bmatrix}$$

$$f = \begin{bmatrix} -3.162278 \\ -8.718722 \\ 63.239034 \\ 9.624838 \end{bmatrix} \cdot \begin{bmatrix} x_r - x \\ -\dot{x} \\ -\theta \\ -\dot{\theta} \end{bmatrix} \equiv \begin{bmatrix} k_{p,x} \\ k_{d,x} \\ k_{p,\theta} \\ k_{d,\theta} \end{bmatrix} \cdot \begin{bmatrix} x_r - x \\ -\dot{x} \\ -\theta \\ -\dot{\theta} \end{bmatrix}$$

$$f = -3.162278(x_r - x) + 8.718722\dot{x} - 63.239034\theta - 9.624838\dot{\theta}$$

These gains and the corresponding controller are what will be used going forward. Also, plugging in the corresponding gains into the same Desmos graph as before, results in the following figure. This can also be accessed by going to https://www.desmos.com/calculator/hyc8jwt9r5.



Cart Pole System Gain Tun...

$x = 31.0220159941$

$k_{px} = -3.162278$

$k_{dx} = -8.718722$

$k_{pt} = 63.239034$

$k_{dt} = 9.624838$

Figure 14: Routh-Hurwitz Criteria Applied to Linear-Quadratic Regulator Chosen Gains

This figure shows that the resulting controller is stable since it satisfies all the Routh-Hurwitz criteria (i.e. all the relevant inequalities are satisfied since the lines shown are at positive x values).

3.4 Stability

Substituting the gains and the system parameters into the overall control system transfer functions yield the following two rational functions

$$C = \frac{31.0219 - 0.351s^2}{0.221368s^4 + 8.83867s^3 + 42.289s^2 + 69.4825s + 31.0219}$$

$$D = \frac{-3.16228s^2}{0.221368s^4 + 8.83867s^3 + 42.289s^2 + 69.4825s + 31.0219}$$

Python was used to determine the gains, poles, and zeros of each of these transfer functions. This is summarized in the following table and pole-zero graphs.

	C	D
Gain	-1.58560	-14.2852
Poles	-34.6761 -0.705575 $-2.27294 \pm 0.749311i$	Same as for C
Zeros	± 9.40115	0 (multiplicity 2)

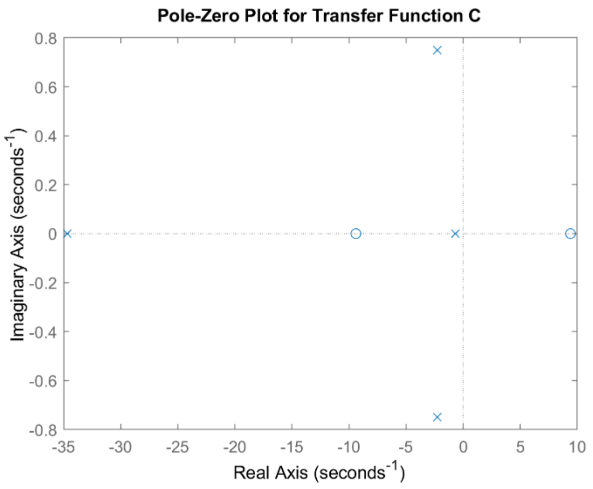


Figure 15: Pole-Zero Plot for C

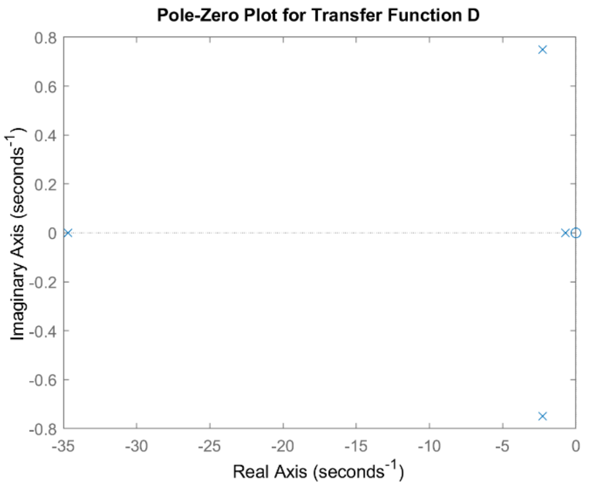


Figure 16: Pole-Zero Plot for D

Since the poles are all on the left half of the complex plane, this serves as further proof that the resulting controller will be stable. Moreover, the larger gain for D means that θ is expected to converge faster than x. This is to be expected since the Q matrix that we chose for the linear-quadratic regulator algorithm emphasized the importance of maintaining $\theta = 0$.

3.5 Controller Behavior

3.5.1 MATLAB Graphs

The following graphs were generated in MATLAB with the initial condition of the pole being upright. Various desired x_r trajectories were used as the input.

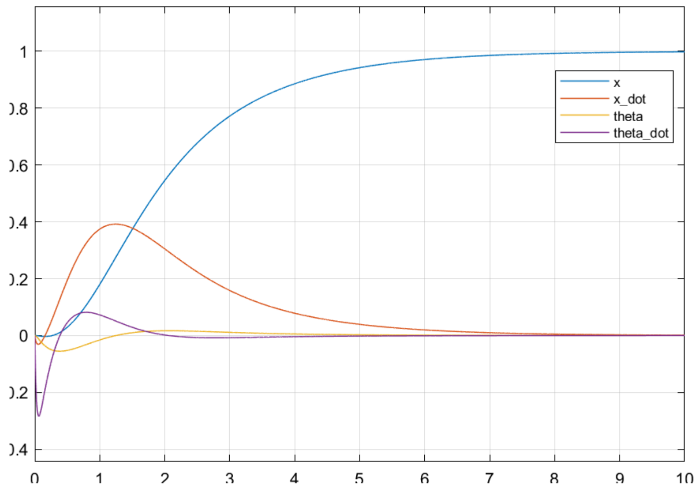


Figure 17: Controller Response for $x_r(t) = 1(t)$

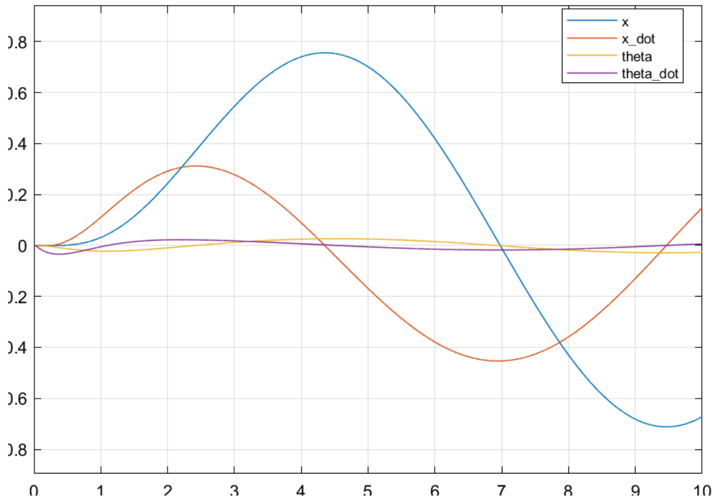


Figure 18: Controller response for $x_r(t) = sin\left(\frac{\pi}{5}t\right)$

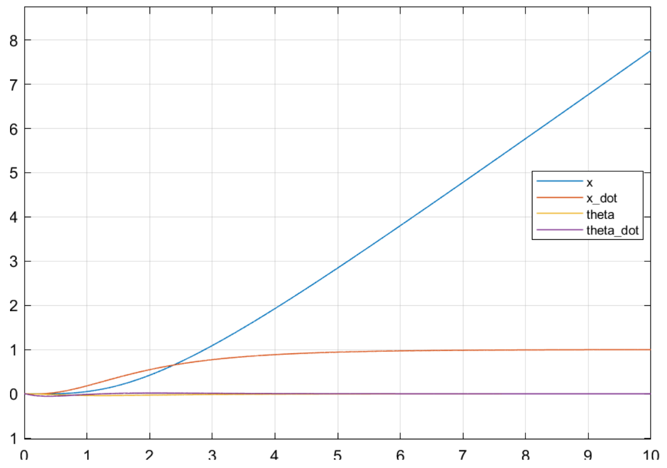


Figure 19: Controller Response for $x_r(t) = t * 1(t)$

In Figure 19, it can be seen that there is a fairly large steady state error (greater than 2 meters), which is to be expected since we are using a PD controller. This is explained further in section 3.6.

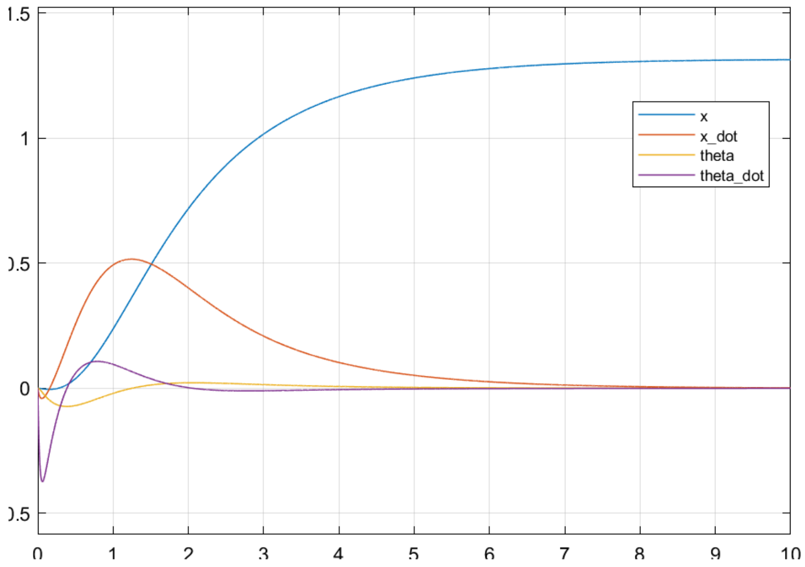


Figure 20: Controller Response for $x_r(t) = 1(t)$ with Disturbance

Figure 20 shows the response of the controller when a constant disturbance force of 1 Newton is subtracted from the controller's output before it is inputted into the plant. This results in a steady state error which causes the controller to stabilize at an x value greater than the target of 1. This is explained in more detail in section 3.6.

3.5.2 Python Graphs

The following graphs were generated in Python using numerical integration of the original non-linearized equations of motion, and the output of our controller. In all cases, the desired x_r trajectory was zero, but the initial conditions were varied to see if the controller could recover. Numerical integration was done using the explicit Runge-Kutta algorithm of order 5 (RK45).

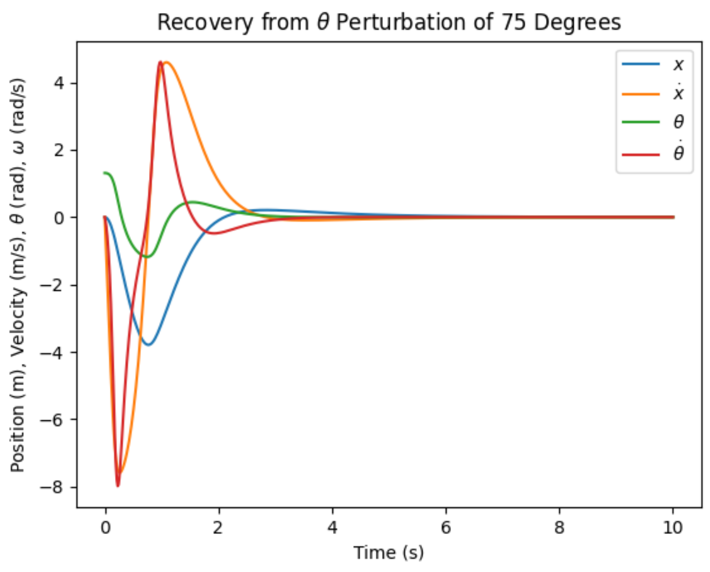


Figure 21: Controller Recovery from $\theta_0 = 75°$

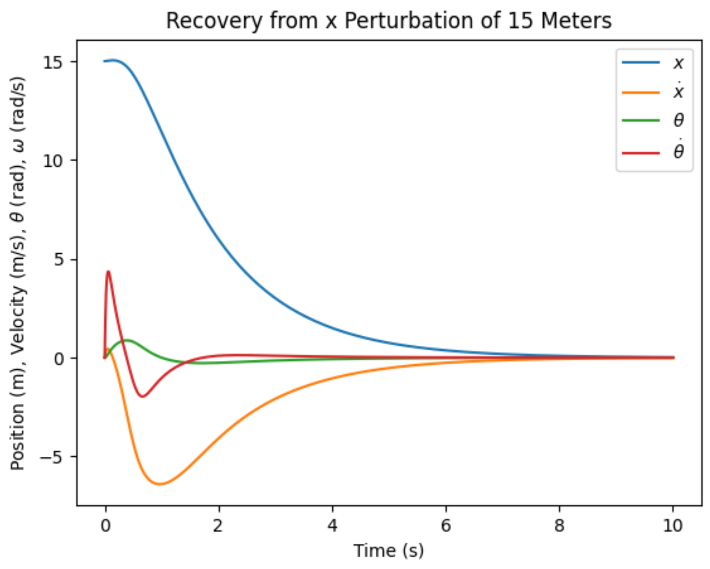


Figure 22: Recovery from $x_0 = 15m$

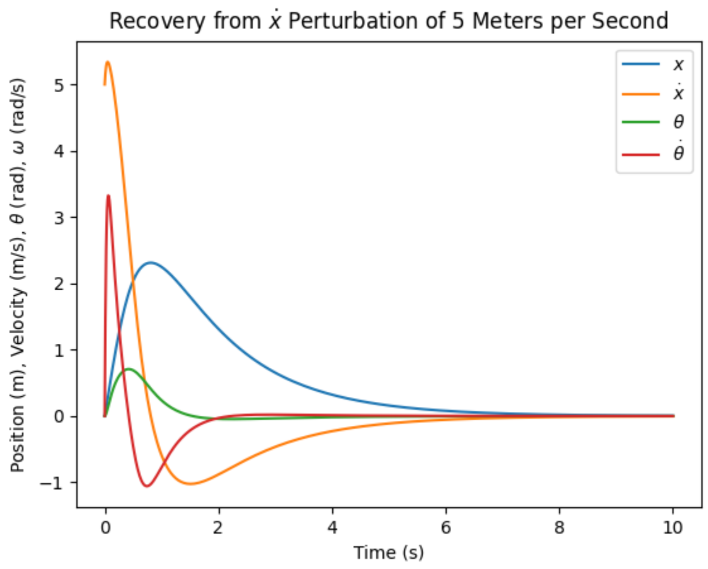


Figure 23: Recovery from $\dot{x}_0 = 15m/s$

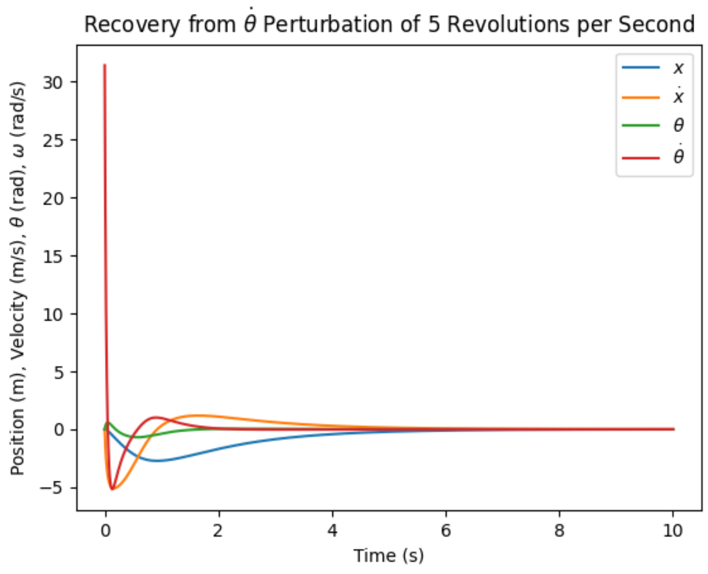


Figure 24: Recovery from $\dot{\theta} = 5rev/s$

Figure 23 is particularly impressive because it shows that despite the large initial positive velocity of the system, the controller first further increases its velocity in order to increase θ. It then is able to steadily decrease its velocity while balancing θ at this increased angle before returning to $x = 0$ at which point it stabilizes θ again.

3.5.3 Force and Power Draw Requirements

The peak force, peak power draw, and average power draw over the 10 seconds were also calculated in the Python code. The power was simply determined by multiplying the controller's output force with the velocity, $\dot{x}$.

Disturbance	Figure	Peak Force (N)	Peak Power Draw (W)	Average Power Draw (W)
$\theta_0 = 75°$	Figure 21	95.72	458.66	43.78
$x_0 = 15m$	Figure 22	47.43	137.54	21.72
$\dot{x}_0 = 5m/s$	Figure 23	43.59	217.97	11.97
$\dot{\theta}_0 = 5rev/s$	Figure 24	302.37	373.60	31.39

It is thought that these requirements should be well within the capabilities of the physical system, but it is hard to say for sure since specifications were not provided. Estimates can be taken from the controller from previous labs for the power it should be able to output. Using voltage power and armature resistance from ME360_Lab_Background, the max power draw is estimated to be

$$P = \frac{V^2}{R_a} = \frac{24^2}{2.6} = 221.54\,W$$

Signifying that the controller would likely not be able to get the power necessary to recover from 75°. However, increasing R (or equivalently decreasing Q) and recomputing the gains with the linear-quadratic regulator algorithm will result in a system that is more reluctant to apply excessive forces. This will however mean that the controller will not be able to recover from such drastic perturbations and will take longer to reach steady state.

Another option that we considered was to simply cap the maximum force output of the controller by the capabilities of the physical system. This was tested by applying a saturation block to the output of the controller which limited the maximum force it could output. For a desired trajectory of $x_r(t) = 1(t)$, the controller requested a maximum force of 3.16 Newtons. When the saturation threshold was 2 Newtons, the system did converge, but when the threshold was 1 Newton the system did not converge. However, when we re-ran the linear-quadratic regulator algorithm with $R = 10$ instead of $R = 1$, the system did converge despite the threshold of 1 Newton. This suggests that although applying a saturation can work to a point, the better option is to recompute the gains with a modified value for R so that the controller can be designed with the limited force output in mind.

3.6 Issues and Further Improvements

3.6.1 Steady State Error due to Ramp Input

In the case where the input is a ramp function as seen in Figure 19, we can see that there is a steady state error. For a ramp input we have

$$x_r = t \cdot 1(t)$$

$$X_r = \frac{1}{s^2}$$

The Laplace transform of the error is then defined to be

$$E = X_r - X = X_r - CX_r$$

$$E = \frac{1 - C}{s^2}$$

Substituting in the transfer function for C, performing partial fraction decomposition, and completing the square gives

$$E = \frac{0.490365s + 1.30142}{(s + 2.27294)^2 + 0.749309^2} + \frac{2.23979}{s} - \frac{2.73011}{s + 0.705573} - \frac{0.0000411651}{s + 34.676}$$

As expected, the denominators of the terms of E can be seen to correspond exactly with the poles of the transfer function D. Moreover, when taking the inverse Laplace transform and then taking the limit as $t \to \infty$, all of these terms except for the $\frac{1}{s}$ term will go to zero since they will all have an exponential decay term. Thus,

$$\lim_{t \to \infty} e(t) = 2.23979$$

Which matches what we saw empirically in Figure 19. This proves that our controller will always have a steady state error when the input x_r is not constant.

3.6.2 Steady State Error due to Disturbance

In the case of a constant disturbance being applied to the system as in Figure 20, we can see that there is a steady state error. This is because when the controller is at the target state where $x = x_r = 1, \dot{x} = \theta = \dot{\theta} = 0$, the controller will output a force of 0. Thus, the disturbance force will cause the system to move so this is not a steady state situation.

Instead, the real steady state occurs at $x_r - x_{ss} = e_{ss}, \dot{x}_{ss} = \theta_{ss} = \dot{\theta}_{ss} = 0$. The steady state error, e_{ss}, can be calculated via force balance. Since θ is no longer relevant at the steady state, the system can be treated as a single solid body (like in section 2.2).

$$\Sigma F = F_{controller} - 1 = (M + m)\ddot{x} = 0$$

$$F_{controller} = 1$$

$$-3.162278(x_r - x_{ss}) + 8.718722\dot{x}_{ss} - 63.239034\theta_{ss} - 9.624838\dot{\theta}_{ss} = 1$$

$$-3.162278(x_r - x_{ss}) = 1$$

$$e_{ss} = x_r - x_{ss} = -0.316228$$

$$x_{ss} = x_r + e_{ss} = 1.316228$$

Which matches perfectly with the steady state error seen in Figure 20.

3.6.3 Final Thoughts

The reason behind the steady state error in the case of the ramp input and the disturbance is that we are using a PD controller. One way to remove this error would be to use a PID controller with $k_{i,x}$ and $k_{i,\theta}$ terms that are proportional to the integral of x and θ respectively. This would serve to reduce the error to 0 in both the above cases once sufficient time has passed.

References

"Cart-Pole Control." Daniel Piedrahita, 28 Dec. 2017, https://danielpiedrahita.wordpress.com/portfolio/cart-pole-control/.

"6.832 Underactuated Robotics MIT 2019." YouTube, YouTube, https://www.youtube.com/playlist?list=PLYJuNKl9CZHANuN9YHZ07_3OeUYZR3YOX.

Cris, et al. "Lagrangian for a Forced System." Physics Stack Exchange, 1 Feb. 1969, https://physics.stackexchange.com/questions/518045/lagrangian-for-a-forced-system.

"Graphing Calculator." Desmos, https://www.desmos.com/calculator.

Markwmuller. "Markwmuller/Controlpy." GitHub, https://github.com/markwmuller/controlpy/blob/master/controlpy/synthesis.py.

Online LaTeX Equation Editor - Create, Integrate and Download, https://latex.codecogs.com/eqneditor/editor.php.

Appendix

Routh-Hurwitz Table Entries

$$N_1 = \frac{1.0k_{d,\theta}k_{p,\theta} + 0.111k_{d,\theta}k_{p,x} - 20.6k_{d,\theta} + 0.111k_{d,x}k_{p,\theta} + 0.0123k_{d,x}k_{p,x} - 0.115k_{d,x} + 0.182k_{p,\theta} + 0.0202k_{p,x} - 0.188}{1.0k_{d,\theta} + 0.111k_{d,x} + 0.182}$$

$$N_2 = -\frac{k_{p,x}\left(9.81k_{d,\theta} + 1.08887076k_{d,x} + 1.781283676284\right)}{1.0k_{d,\theta} + 0.110996k_{d,x} + 0.1815783564}$$

$$N_{3,num} = 9.81k_{d,\theta}^2k_{p,x} - 9.81k_{d,\theta}k_{d,x}k_{p,\theta} + 1.09k_{d,\theta}k_{d,x}k_{p,x} + 202.0k_{d,\theta}k_{d,x} - 16.0k_{d,\theta}k_{p,\theta} + 1.78k_{d,\theta}k_{p,x} + 331.0k_{d,\theta} - 1.09k_{d,x}^2k_{p,\theta} + 1.39 \cdot 10^{-17}k_{d,x}^2k_{p,x} + 1.13k_{d,x}^2 - 3.56k_{d,x}k_{p,\theta} + 5.55 \cdot 10^{-17}k_{d,x}k_{p,x} + 3.68k_{d,x} - 2.91k_{p,\theta} - 5.55 \cdot 10^{-17}k_{p,x} + 3.01$$

$$N_{3,den} = 1.0k_{d,\theta}k_{p,\theta} + 0.111k_{d,\theta}k_{p,x} - 20.6k_{d,\theta} + 0.111k_{d,x}k_{p,\theta} + 0.0123k_{d,x}k_{p,x} - 0.115k_{d,x} + 0.182k_{p,\theta} + 0.0202k_{p,x} - 0.188$$

$$\begin{aligned} N_{4,num} = -1.0k_{p,x}&\left(1.0k_{d,\theta}k_{p,\theta} + 0.111k_{d,\theta}k_{p,x} - 20.6k_{d,\theta} + 0.111k_{d,x}k_{p,\theta} + 0.0123k_{d,x}k_{p,x} - 0.115k_{d,x} + 0.182k_{p,\theta} + 0.0202k_{p,x} - 0.188\right)\left(96.2k_{d,\theta}^3k_{p,x}\right. \\ &- 96.2k_{d,\theta}^2k_{d,x}k_{p,\theta} + 21.4k_{d,\theta}^2k_{d,x}k_{p,x} + 1.98 \cdot 10^3k_{d,\theta}^2k_{d,x} - 157.0k_{d,\theta}^2k_{p,\theta} + 34.9k_{d,\theta}^2k_{p,x} + 3.24 \cdot 10^3k_{d,\theta}^2 - 21.4k_{d,\theta}k_{d,x}^2k_{p,\theta} \\ &+ 1.19k_{d,\theta}k_{d,x}^2k_{p,x} + 231.0k_{d,\theta}k_{d,x}^2 - 69.9k_{d,\theta}k_{d,x}k_{p,\theta} + 3.88k_{d,\theta}k_{d,x}k_{p,x} + 756.0k_{d,\theta}k_{d,x} - 57.2k_{d,\theta}k_{p,\theta} + 3.17k_{d,\theta}k_{p,x} + 618.0k_{d,\theta} \\ &- 1.19k_{d,x}^3k_{p,\theta} + 1.51 \cdot 10^{-17}k_{d,x}^3k_{p,x} + 1.23k_{d,x}^3 - 5.82k_{d,x}^2k_{p,\theta} + 8.52 \cdot 10^{-17}k_{d,x}^2k_{p,x} + 6.02k_{d,x}^2 - 9.52k_{d,x}k_{p,\theta} + 3.84 \cdot 10^{-17}k_{d,x}k_{p,x} \\ &\left.+ 9.84k_{d,x} - 5.19k_{p,\theta} - 9.89 \cdot 10^{-17}k_{p,x} + 5.37\right) \end{aligned}$$

$$N_{4,den} = \left(9.81k_{d,\theta}^2k_{p,x} - 9.81k_{d,\theta}k_{d,x}k_{p,\theta} + 1.09k_{d,\theta}k_{d,x}k_{p,x} + 202.0k_{d,\theta}k_{d,x} - 16.0k_{d,\theta}k_{p,\theta} + 1.78k_{d,\theta}k_{p,x} + 331.0k_{d,\theta} - 1.09k_{d,x}^2k_{p,\theta} + 1.39 \cdot 10^{-17}k_{d,x}^2k_{p,x} + 1.13k_{d,x}^2 - 3.56k_{d,x}k_{p,\theta} + 5.55 \cdot 10^{-17}k_{d,x}k_{p,x} + 3.68k_{d,x} - 2.91k_{p,\theta} - 5.55 \cdot 10^{-17}k_{p,x} + 3.01\right)\left(1.0k_{d,\theta}^2k_{p,\theta} + 0.111k_{d,\theta}^2k_{p,x} - 20.6k_{d,\theta}^2 + 0.222k_{d,\theta}k_{d,x}k_{p,\theta} + 0.0246k_{d,\theta}k_{d,x}k_{p,x} - 2.4k_{d,\theta}k_{d,x} + 0.363k_{d,\theta}k_{p,\theta} + 0.0403k_{d,\theta}k_{p,x} - 3.93k_{d,\theta} + 0.0123k_{d,x}^2k_{p,\theta} + 0.00137k_{d,x}^2k_{p,x} - 0.0127k_{d,x}^2 + 0.0403k_{d,x}k_{p,\theta} + 0.00447k_{d,x}k_{p,x} - 0.0417k_{d,x} + 0.033k_{p,\theta} + 0.00366k_{p,x} - 0.0341\right)$$

Python Code

The following code was used for symbolically performing the algebra and calculus throughout this project to verify our hand calculations. It was also used for performing the linear-quadratic regulator algorithm, for computing the Routh-Hurwitz stability criteria, and for solving for the poles and zeros of the final transfer functions. The code can also be found online at https://github.com/amaarquadri/ME360/blob/master/mini_project/cart_pole.py.

```python
import numpy as np
import sympy as sp
from tbcontrol.symbolic import routh
from scipy.linalg import solve_continuous_are
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
from matplotlib import use
from sympy.physics.vector.printing import vlatex
use('TkAgg')


def to_string(expression, to_word=True):
    text = vlatex(expression).replace(r'\operatorname{Theta}', r'\Theta')
    if to_word:
        text = text.replace(r'p f x', r'p,x') \
            .replace(r'd f x', r'd,x') \
            .replace(r'p f \theta', r'p,\theta') \
            .replace(r'd f \theta', r'd,\theta')
    else:
        text = text.replace(r'p f x', r'px') \
            .replace(r'd f x', r'dx') \
            .replace(r'p f \theta', r'pt') \
            .replace(r'd f \theta', r'dt')
    return text


def get_gain(s, ratio, zeros, poles, epsilon=0.1):
    # find a value sufficiently far from all poles and zeros
    i = 0
    while any([abs(i - zero) < epsilon for zero, _ in zeros.items()]) or \
            any([abs(i - pole) < epsilon for pole, _ in poles.items()]):
        i += 1
    gain = ratio.subs(s, i)
    gain *= np.product([(i - pole) ** multiplicity for pole, multiplicity in poles.items()])
    gain /= np.product([(i - zeros) ** multiplicity for zeros, multiplicity in zeros.items()])
    return sp.re(gain.evalf())


def get_cart_pole_physics(constants):
    M, m, L, b, g = constants

    t = sp.symbols('t')

    # define state variables
    x = sp.Function('x')(t)
    theta = sp.Function('theta')(t)

    # define control variables
    f = sp.Function('f')(t)

    x_rel = -L * sp.sin(theta)
    v_x_rel = sp.diff(x_rel, t)
    v_x_pole = sp.diff(x, t) + v_x_rel

    y_rel = L * sp.cos(theta)
    v_y_pole = sp.diff(y_rel, t)  # relative and absolute velocities are the same

    KE_cart = (M * sp.diff(x, t) ** 2) / 2
    KE_pole = (m * v_x_pole ** 2) / 2 + (m * v_y_pole ** 2) / 2
    KE = KE_cart + KE_pole
    PE = m * g * L * sp.cos(theta)
    L = sp.simplify(KE - PE)

    f_x_external = f - b * sp.diff(x, t)  # dissipative and external forces for the x direction
    f_theta_external = 0

    x_vec = np.array([x, sp.diff(x, t), theta, sp.diff(theta, t)])
    u_vec = np.array([f])

    return t, x_vec, u_vec, L, [f_x_external, f_theta_external]


def get_equations_of_motion(t, x_vec, L, F_external_vec):
    """
    The generalized coordinates should be in the even indices of x_vec
    and their corresponding derivatives should be in the odd indices.
    """
    lagrange_equations = []
    for x, F_external in zip(x_vec[::2], F_external_vec):
        lagrange_equations.append(sp.Eq(sp.diff(sp.diff(L, sp.diff(x, t)), t), sp.diff(L, x) + F_external))
    result = sp.solve(lagrange_equations, [sp.diff(x, (t, 2)) for x in x_vec[::2]])

    equations_of_motion = []
    for i, x in enumerate(x_vec[::2]):
        equations_of_motion.append(sp.diff(x, t))
        equations_of_motion.append(sp.simplify(result[sp.diff(x, (t, 2))]))

    return equations_of_motion


def linearize(t, x_vec, u_vec, expression, operating_point=None):
    if operating_point is None:
        operating_point = {}

    # set unspecified required values to 0
    for x in x_vec[::2]:
        if x not in operating_point:
            operating_point[x] = 0
        if sp.diff(x, t) not in operating_point:
            operating_point[sp.diff(x, t)] = 0
        if sp.diff(x, (t, 2)) not in operating_point:
            operating_point[sp.diff(x, (t, 2))] = 0
    for u in u_vec:
        if u not in operating_point:
            operating_point[u] = 0

    # The operating points are not included in the linearization which means the resulting linear system
    # (x_dot = A * x + B * u) is in reference to the primed coordinates.
    # Thus the K gains for the controller will also correspond to the primed coordinates.
    # However, K * (x_r' - x') = K * (x_r - x) so the controller works the same regardless.
    return sum([sp.diff(expression, var).subs(operating_point) * var for var in np.concatenate((x_vec, u_vec))])


def get_transfer_functions(t, x_vec, u_vec, equations_of_motion):
    s = sp.symbols('s')

    X_vec = np.array([sp.Function(x.name.capitalize())(s) for x in x_vec[::2]])
    U_vec = np.array([sp.Function(u.name.capitalize())(s) for u in u_vec])

    laplace_tf = {}
    for x, X in zip(x_vec[::2], X_vec):
        laplace_tf[x] = X
        laplace_tf[sp.diff(x)] = s * X
        laplace_tf[sp.diff(x, (t, 2))] = s ** 2 * X
    for u, U in zip(u_vec, U_vec):
        laplace_tf[u] = U

    laplace_equations = [sp.Eq(sp.diff(x, t), eq).subs(laplace_tf)
                         for x, eq in zip(x_vec[1::2], equations_of_motion[1::2])]
    result = sp.solve(laplace_equations, list(X_vec))  # need to convert to list because sympy doesn't support numpy

    laplace_transforms = [sp.simplify(result[X]) for X in X_vec]
    return s, X_vec, U_vec, laplace_transforms


def get_controller_transfer_functions(s, X_vec, U_vec, transfer_functions):
    k_p_mat = np.array([[sp.symbols(f'k_p_{U.name.lower()}_{X.name.lower()}') for X in X_vec] for U in U_vec])
    k_d_mat = np.array([[sp.symbols(f'k_d_{U.name.lower()}_{X.name.lower()}') for X in X_vec] for U in U_vec])

    X_r_vec = np.array([sp.Function(X.name + '_r')(s) for X in X_vec])

    U_control_vec = np.dot(k_p_mat, (X_r_vec - X_vec)) - np.dot(k_d_mat, s * X_vec)

    eqs = [sp.Eq(X, tf).subs(zip(U_vec, U_control_vec)) for X, tf in zip(X_vec, transfer_functions)]
    result = sp.solve(eqs, list(X_vec))  # need to convert to array because sympy doesn't support numpy
    control_tfs = [result[X] for X in X_vec]

    k_pd_mat = np.zeros((len(U_vec), 2 * len(X_vec))).astype(object)
    k_pd_mat[:, 0::2] = k_p_mat
    k_pd_mat[:, 1::2] = k_d_mat

    return k_pd_mat, X_r_vec, control_tfs


def get_matrix_equations_of_motion(x_vec, u_vec, equations_of_motion):
    A = np.array([[sp.diff(eq, x) for x in x_vec] for eq in equations_of_motion])
    B = np.array([[sp.diff(eq, u) for u in u_vec] for eq in equations_of_motion])
    return A, B


def linear_quadratic_regulator(A, B, Q, R):
    """
    https://youtu.be/bMiiC94FJ5E?t=3276
    https://github.com/markwmuller/controlpy/blob/master/controlpy/synthesis.py
    System is defined by dx/dt = Ax + Bu
    Minimizing integral (x.T*Q*x + u.T*R*u) dt from 0 to infinity
    Returns K such that optimal control is u = -Kx
    """
    # first, try to solve the Ricatti equation
    P = solve_continuous_are(A, B, Q, R)

    # compute the LQR gain
    K = np.linalg.multi_dot([np.linalg.inv(R), B.T, P])
    return K


def analyze_controller(s, X_vec, X_r_vec, k_pd_mat, controller_transfer_functions, K):
    for X_r in X_r_vec:
        # set all others in X_r_vec to zero
        tfs = [sp.simplify(tf.subs([(X_r_, 0) for X_r_ in X_r_vec if X_r_ is not X_r]) / X_r)
               for tf in controller_transfer_functions]
        for X, tf in zip(X_vec, tfs):
            numerator, denominator = tf.as_numer_denom()

            routh_table = np.array(routh(sp.Poly(denominator, s)))
            routh_conditions = routh_table[:, 0]

            zeros = sp.roots(numerator.subs(zip(k_pd_mat.flatten(), K.flatten())), s)
            poles = sp.roots(denominator.subs(zip(k_pd_mat.flatten(), K.flatten())), s)
            gain = get_gain(s, tf.subs(zip(k_pd_mat.flatten(), K.flatten())), zeros, poles)

            zeros_description = ', '.join(
                [str(zero.evalf(6)) if multiplicity == 1 else f'{zero.evalf(6)} (x{multiplicity})'
                 for zero, multiplicity in zeros.items()])
            poles_description = ', '.join(
                [str(pole.evalf(6)) if multiplicity == 1 else f'{pole.evalf(6)} (x{multiplicity})'
                 for pole, multiplicity in poles.items()])
            routh_conditions_description = '\n'.join([to_string(condition.evalf(3)) for condition in routh_conditions])
            print(f'{X}/{X_r}: Routh Conditions:\n{routh_conditions_description}\nGain: {gain.evalf(6)}\n'
                  f'Poles: {poles_description}\nZeros: {zeros_description}\n\n')


def test_controller(x_vec, u_vec, equations_of_motion, K, x_r_func=None, x_0=None, t_f=10):
    """
    A, B, K, and x_0 must be numpy arrays and x_r_func must be a function that returns a numpy array.
    """
    if x_0 is None:
        x_0 = np.zeros(len(equations_of_motion))
    if x_r_func is None:
        # use step input for first variable in x_r
        target = np.zeros(len(equations_of_motion))
        target[0] = 1

        def x_r_func(_):
            return target

    equations_of_motion = [sp.lambdify([x_vec, u_vec], eq) for eq in equations_of_motion]

    def state_derivative(t, x_vec_):
        u_vec_ = np.dot(K, x_r_func(t) - x_vec_)
        return np.array([eq(x_vec_, u_vec_) for eq in equations_of_motion])

    result = solve_ivp(state_derivative, (0, t_f), x_0, method='RK45', rtol=1e-6)
    # noinspection PyUnresolvedReferences
    return result.t, result.y, x_r_func


def build_and_test_controller(constants, physics_func=get_cart_pole_physics, operating_point=None, Q=None, R=None,
                              x_r_func=None, x_0=None, t_f=10):
    """
    Generates

    :param constants: A dictionary that maps sympy variables to floating point values.
                      The sympy variables must be in the order that physics_func expects.
    :param physics_func:
    :param operating_point:
    :param Q:
    :param R:
    :param x_r_func:
    :param x_0:
    :param t_f:
    """
    print('For displaying LaTeX:', 'https://latex.codecogs.com/eqneditor/editor.php')

    def substitute_constants(expression):
        return np.vectorize(lambda v: v.subs(constants))(expression)

    t, x_vec, u_vec, lagrangian, F_external_vec = physics_func(constants.keys())
    print('Lagrangian:', to_string(lagrangian))
    print('\nExternal Forces:', to_string(F_external_vec))

    equations_of_motion = get_equations_of_motion(t, x_vec, lagrangian, F_external_vec)
    print('\nEquations of Motion:')
    for var, eq in zip(x_vec[1::2], equations_of_motion[1::2]):
        print(to_string(sp.Eq(sp.diff(var, t), eq)))
```

```
# operating_point = {x_vec[2]: sp.pi}
linearized_equations_of_motion = [linearize(t, x_vec, u_vec, eq, operating_point) for eq in equations_of_motion]
print('\nLinearized Equations of Motion (in Primed Variables):')
for var, eq in zip(x_vec[1::2], linearized_equations_of_motion[1::2]):
    print(to_string(sp.Eq(sp.diff(var, t), eq)))

s, X_vec, U_vec, transfer_functions = get_transfer_functions(t, x_vec, u_vec, linearized_equations_of_motion)
transfer_functions = [sp.expand(tf) for tf in transfer_functions]
print('\nTransfer Functions (in Primed Variables):')
for var, tf in zip(X_vec, transfer_functions):
    print(to_string(sp.Eq(var, tf)))

k_pd_mat, X_r_vec, control_transfer_functions = \
    get_controller_transfer_functions(s, X_vec, U_vec, transfer_functions)
print('\nControl Transfer Functions (in Primed Variables):')
for var, tf in zip(X_vec, control_transfer_functions):
    print(to_string(sp.Eq(var, tf)))

A, B = get_matrix_equations_of_motion(x_vec, u_vec, linearized_equations_of_motion)
print('\nA (in Primed Variables):\n', to_string(A), '\nB (in Primed Variables):\n', to_string(B))

A = substitute_constants(A).astype(float)
B = substitute_constants(B).astype(float)
if Q is None:
    Q = np.identity(len(x_vec))
if R is None:
    R = np.identity(len(u_vec))
K = linear_quadratic_regulator(A, B, Q, R)
print('K:\n', K)

analyze_controller(s, X_vec, X_r_vec, k_pd_mat, substitute_constants(control_transfer_functions), K)

equations_of_motion = [substitute_constants(eq) for eq in equations_of_motion]
t_vals, state_vals, x_r_func = test_controller(x_vec, u_vec, equations_of_motion, K,
                                               x_r_func=x_r_func, x_0=x_0, t_f=t_f)

forces = [np.dot(K, x_r_func(t_) - state_vals[:, i])[0] for i, t_ in enumerate(t_vals)]
powers = [force * velocity for force, velocity in zip(forces, state_vals[1, :])]
print('\nMax Force:', np.max(np.abs(forces)))
print('Max Power Draw:', np.max(np.abs(powers)))
print('Mean Power Draw:', np.mean(np.abs(powers)))

for i, x in enumerate(x_vec):
    plt.plot(t_vals, state_vals[i, :], label=f'${to_string(x)}$')
```

```python
def main():
    M, m, L, b, g = sp.symbols('M, m, L, b, g')
    constants = {
        M: 1.994376,
        m: 0.105425,
        L: 0.110996,
        b: 1.6359,
        g: 9.81
    }
    build_and_test_controller(constants, Q=np.diag([10, 20, 100, 50]), R=np.array([[1]]),
                              x_r_func=lambda t: np.zeros(4),
                              x_0=np.array([0, 0, np.radians(75), 0]))

    # plt.plot(t_vals, x_r_func(t_vals)[0], label=r'$x_{r}$')
    plt.xlabel('Time (s)')
    plt.ylabel(r'Position (m), Velocity (m/s), $\theta$ (rad), $\omega$ (rad/s)')
    plt.title(r'Recovery from x Perturbation of 15 Meters')
    plt.legend()
    # plt.savefig('theta_perturbation.png')
    plt.show()


if __name__ == '__main__':
    main()
```