Nonlinear Control of Ball and Beam System

***Abstract*—The report outlines the design and implementation of a PD and LQR controller as well as an Extended Kalman Filter on a non-linear ball and beam system. The simulation and hardware results are reported and compared, and the effectiveness of the controllers and observer are analyzed.**

I. Introduction

For this project, we will be trying to design controllers for a non-linear ball and beam system. The task will involve having the ball follow a desired trajectory while minimizing the position error and input energy. First, we will simulate this platform in Matlab/Simulink to test and perform coarse-grain tuning of our controllers. Then, we will test these controllers on a physical ball and beam system, performing sim-to-real transfer and fine-grain tuning.

II. System Modeling

The system consists of a rolling ball on a lever arm with one end connected to a linkage controlled by a DC motor. A diagram of the system is shown in fig. 13.

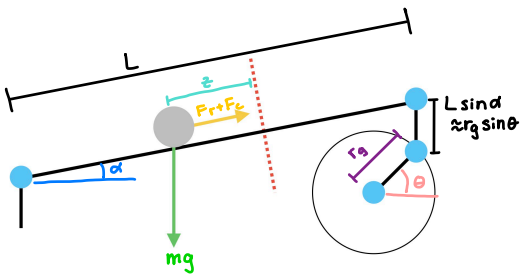


Fig. 1: Motor Controller Reactor Block Diagram

We represent the system state as follows:

$$x = [z, \dot{z}, \theta, \dot{\theta}]^T \tag{1}$$

Where z represents the position of the ball relative to the desired location and θ represents the angle of the motor. We are also given known constants: L is the length of the beam, r_b is the radius of the metal ball, and m is the mass of the ball.

The ball's moment of inertia is therefore:

$$J_b = \frac{5}{2} m {r_b}^2 \tag{2}$$

The two forces on the ball shown in the diagram F_c and F_r can be expressed as:

$$F_c = m\left(\frac{L}{2} - z\right)\dot{\alpha}^2 \tag{3}$$

$$F_r = \frac{J_r}{{r_b}^2}\ddot{z} \tag{4}$$

Using this, we can represent the ball's motion as:

$$m\ddot{z} = mg\sin(\alpha) - m\left(\frac{L}{2} - z\right)\dot{\alpha}^2 - \frac{J_r}{{r_b}^2}\ddot{z} \tag{5}$$

And we can calculate the motor dynamics using:

$$\tau\ddot{\theta} = -\dot{\theta} + KV \tag{6}$$

where V represents the motor voltage system input and K represents the motor's constant. This brings our system to the form:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} x_2 \\ \frac{5g}{7}\frac{r_g}{L}\sin x_3 - \frac{5}{7}\left(\frac{L}{2} - x_1\right)\left(\frac{r_g}{L}\right)^2 {x_4}^2 \cos^2 x_3 \\ x_4 \\ -\frac{x_4}{\tau} + \frac{K}{\tau}u \end{bmatrix} \tag{7}$$

The system constants are as given [1] :

Name	Value	Units
r_g	0.0254	m
L	0.4255	m
g	0.981	m/s^2
K	1.5	rad/sV
τ	0.025	s

TABLE I: Model constants

In the hardware setup, we are able to measure the current ball position $x_2 = z$ and the current motor angle $x_3 = \theta$. Since we are unable to observe the entire state, we must also design an observer to determine the missing state values.

III. Observer and Controller Design

For this experiment, our cost function is defined as:

$$J_{score} = \omega_t J_t + \omega_e J_e + \omega_s J_s \tag{8}$$

with our goal being to minimize the controller's overall cost (J_{score}) as much as possible.

The tracking score is represented by:

$$J_t = \frac{1}{T}\int_0^T (z(t) - z_r(t))^2\, dx \approx \int_0^T \sum_{n=1}^{\infty} (z(t_k) - z_r(t_k))^2 \Delta t \tag{9}$$

The energy score is represented by:

$$J_e = \frac{1}{T}\int_0^T (u(t))^2\, dx \approx \int_0^T \sum_{n=1}^{\infty} (u(t_k))^2 \Delta t \tag{10}$$

The safety score is represented by:

$$J_s = \begin{cases} 1 & \exists t, z(t) unsafe \\ 0 & otherwise \end{cases} \tag{11}$$

With the safety conditions being:

$$\begin{cases} g(t) \leq -0.19cm & \text{Collision on right} \\ g(t) \geq -0.19cm & \text{Collision on left} \\ \theta(t) \geq 60^\circ & \text{Servo out of range} \\ \theta(t) \leq -60^\circ & \text{Servo out of range} \end{cases} \tag{12}$$

The values $\omega_t, \omega_t, \omega_t$ represent the different cost weights.

IV. Observer and Controller Design

A. Controller: *PD*

The proportional-derivative (PD) controller works by structuring the input in two parts in terms of the system error. For a control problem with a given reference signal $r(t)$ and an output $y(t)$, we can define the tracking error as follows:

$$e(t) = r(t) - y(t)$$

Using this error signal, we can now build the input to the system as follows:

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

The control input is a sum of two terms, the proportional term and the derivative term, hence the name PD. The proportional term serves to reduce steady-state error and increases the responsiveness of the system. The derivative term reduces overshoot by improving the damping of the response. This, in turn, allows us to speed up the response of the system. Together, these two terms can be tuned based on the experimental results until the system is able to stably track and reach its target.

When implementing our PD loop, we implemented two nested PD loops, one that output the arm angle θ which fed into the second that output the input servo voltage, $u(t)$. By having two cascading loops, we are therefore able to have a higher level of control over the response of the system by being able to tune separate aspects of it.

For our θ PD controller, we ended up with gains of $K_p = 5$ and $K_d = 5.5$ while for our $u(t) = V_{servo}$ controller, we had gains of $K_p = 5$ and $K_d = 3$.

B. Controller: *LQR*

The LQR controller is a cost-minimizing linear control scheme. It works by defining a cost function based on the control input $u(t)$ and state $u(t)$ of the form:

$$J = \int_0^\infty \{x^T(t) Q x(t) + u^T(t) R u(t)\} \tag{13}$$

We can then work backwards from our desired final location to find the optimal control sequence $u^o(t)$ which minimizes the cost J.

Since we are trying to control a non-linear system, we must first linearize it. This is done by taking the Jacobian of the system, generating an A matrix of the form:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & a_{23} & a_{24} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -40 \end{bmatrix} \tag{14}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 60 \end{bmatrix} \tag{15}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{16}$$

$$a_{23} = 0.0051 \cdot \cos(x_3) \cdot \sin(x_3) \cdot x_4^4 + 0.4183 \cdot \cos(x_3) \tag{17}$$

$$a_{24} = -0.0102 \cdot x_4^3 \cdot \cos(x_3)^2 \tag{18}$$

With x_1, x_2, x_3, x_4 coming from the current state x. This lets us construct a linear system of the form $\dot{x} = Ax + Bu$, $y = Cx$.

Initially, our code solution was to use Matlab's built-in **lqr()** function to obtain the optimal control matrix K. However, we encountered compatibility issues between Matlab's predefined functions and Simulink's code-gen build flow. Thus, we relied on a manual implementation of the **lqr()** function. This was done by using an iterative method [2] to determine a matrix sign function, which was then used to solve the Riccati equation and obtain the control constant K.

This allows us to set the velocity input to $u = -K * x'$ which is used during this timestep. This process is repeated every time our step function is called

For our cost constants, we ended up with the values of

$$Q = \begin{bmatrix} 125 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{19}$$

$$R = 1 \tag{20}$$

Since the only value we care about is the ball's current position, all but the top left corner of Q can be set to zero. Moreover, when tuning parameters, the absolute values of Q and R are not that important, only their relative values. Thus, we kept R at 1 and tuned Q to compensate.

C. Observer: ***Extended Kalman Filter***

We deployed a Kalman Filter to estimate the system's current state while simultaneously filtering out sensor and input noise. Since we are using continuous time system equations, we will accordingly need to use a continuous time Kalman Filter [3]. Also, since the Kalman Filter is only built for linear systems, we will need to linearize the system equations (turning it into an Extended Kalman Filter).

The Extended Kalman Filter algorithm uses the same linearization method as before, taking the jacobian and generating a linear system of the form $\dot{x} = Ax + Bu$, $y = Cx$ as seen in 14, 15, 16. We will also need to save our current estimated state $\hat{x}$ and estimated noise covariance P

To operate the Kalman Filter, first, we determine an estimate of the next state and covariance using the equations:

$$\hat{x}_p = \hat{x} + (A\hat{x} + Bu) * \Delta t \tag{21}$$

$$P_p = P + (APA^T + W) * \Delta t \tag{22}$$

$$y_p = C\hat{x}_p \tag{23}$$

This gives us a predicted output value y_p. Then, we can iterate our values $\hat{x}$ and P, given the difference in our expected output y_p and our measured output y

$$\hat{x} = \hat{x}_p + K(y - y_p) \tag{24}$$

$$K = PC^T(CP_pC^T + V)^{-1} \tag{25}$$

$$P = (I = KC)P_p \tag{26}$$

This produces a continuous estimate of the system's current state in the form of our $\hat{x}$ which is then passed along to the controllers. The algorithm continually iterates on its predictions based on new measurement values, allowing it to learn the system's noise.

For our implementation, the input/output noise estimate values were:

$$W = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{27}$$

$$V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{28}$$

With an initial estimated covariance matrix of:

$$P = \begin{bmatrix} 0.0046 & 0.0044 & 0.0000 & 0.0000 \\ 0.0044 & 0.1043 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0044 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0012 \end{bmatrix} \tag{29}$$

D. Safety Measure: ***Input Filter***

For a safety measure for our project, we designed an input filter for the servo voltage. This served to both create a ceiling for the magnitude of the input as well as to de-noise the signal to create a smoother system response.

Our input filter worked in two stages: the first controlled the magnitude of the change in voltage at each time step while the second was a saturation of the input signal.

The first stage ensured two things: that the change in the input signal would not exceed some maximum amount, meaning that any jarring large swings were controlled and that any changes in the voltage below a certain magnitude were omitted. The maximum difference in voltage was tuned to be 0.04V while the minimum change allowed to pass through was 0.005V.

The second stage essentially saturated the input by setting a cap on the magnitude of the voltage. We set ours to a magnitude of 5V and this was able to prevent any large voltage spikes that would cause the arm to swing too abruptly.

V. Simulation Results

During simulation, we assume a starting position of $[-0.05, 0.0, 0.0, 0.0]$. This is because for the hardware experiments, we are expected to manually hold the ball close to the center of the beam before starting the test. Thus, for the simulation, we model this as a slightly off-center starting position - enough so that we have some amount of starting error, but not too much as to interfere with the controller's performance.

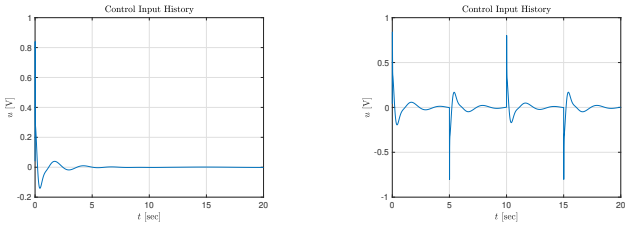
TABLE II: Simulation Performance

Reference	Controller	Tracking Cost	Energy Cost	Total Score
Sine Wave	PD	0.18	0.01	0.19
	LQR	0.15	0.03	0.18
Square Wave	PD	0.18	0.01	1.36
	LQR	1.33	0.16	1.49

A. PD Results

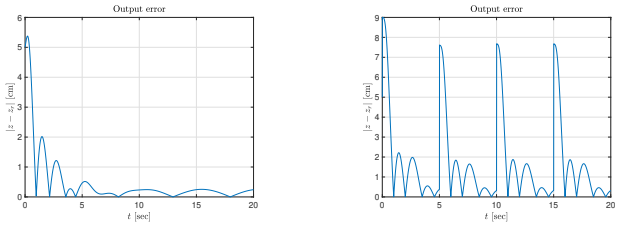
TABLE III: PD Controller Parameters - Simulation Experiments

PD Loop	Proportional Gain K_p	Derivative Gain
Lever arm angle, $\theta(t)$	6	10
Input, $u(t)$	7	8



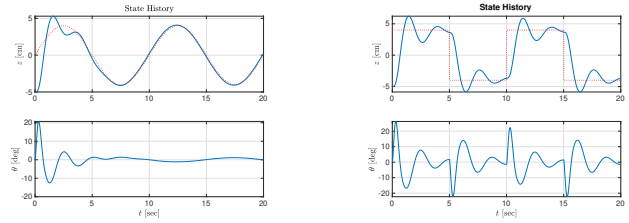
(a) Sinusoidal (b) Square wave

Fig. 2: PD Simulation Control Input Graphs



(a) Sinusoidal (b) Square wave

Fig. 3: PD Simulation Output Error Graphs



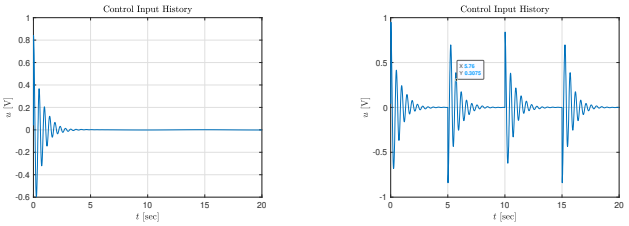
(a) Sinusoidal (b) Square wave

Fig. 4: PD Simulation State History

B. LQR Results

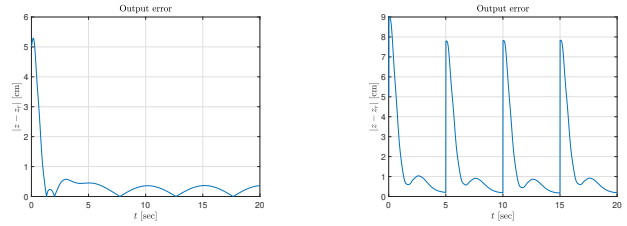
TABLE IV: LQR Controller Parameters - Simulation Experiments

Q Matrix	R Matrix
$\begin{bmatrix} 75 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$	1



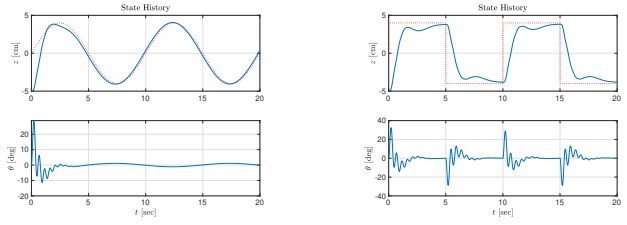
(a) Sinusoidal (b) Square wave

Fig. 5: LQR Simulation Control Input Graphs



(a) Sinusoidal (b) Square wave

Fig. 6: LQR Simulation Output Error Graphs



(a) Sinusoidal (b) Square wave

Fig. 7: LQR Simulation State History

VI. Hardware Experiment Results

Include the parameters used as a table, include **experiment plots** for the evaluation trajectory (using the command given in the instructions), and results for the final reference trajectory in a single table **comparing the two methods**

TABLE V: Hardware Performance

Reference	Controller	Tracking Cost	Energy Cost	Total Score
Evaluation Trajectory	PID	2.20	1.91	4.107
	LQR	2.98	1.06	4.04

TABLE VI: PD Controller Parameters - Hardware Experiments

PD Loop	Proportional Gain K_p	Derivative Gain
Lever arm angle, $\theta(t)$	5	5.5
Input, $u(t)$	5	3

A. PD Results

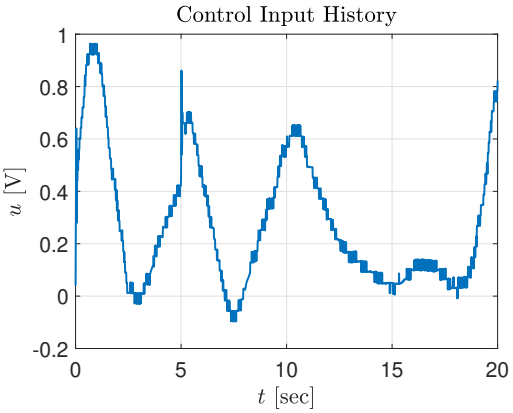


Fig. 8: PD Hardware Control Input Graphs

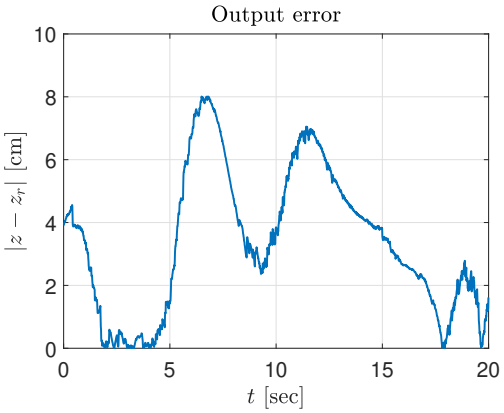


Fig. 9: PD Hardware Output Error Graphs

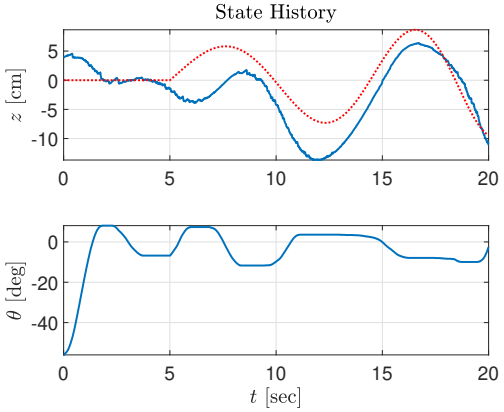


Fig. 10: PD Hardware State History

B. LQR Results

TABLE VII: LQR Controller Parameters - Hardware Experiments

Q Matrix	R Matrix
$\begin{bmatrix} 125 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$	1

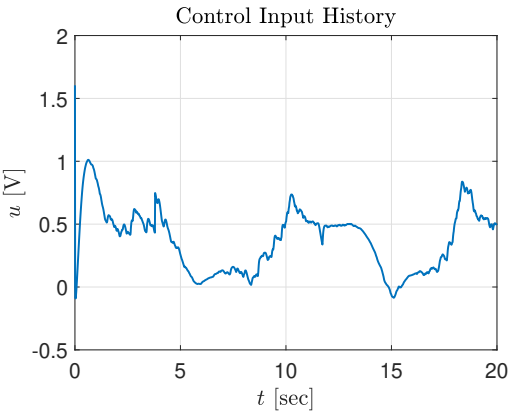


Fig. 11: LQR Hardware Control Input Graphs

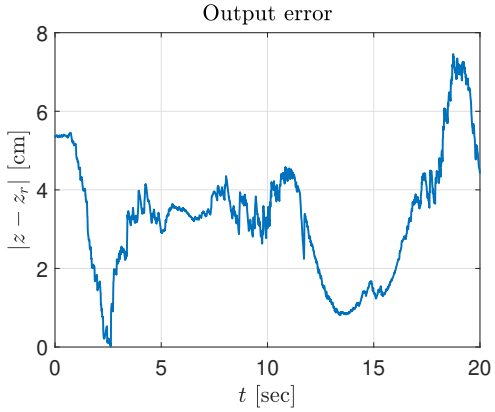


Fig. 12: LQR Hardware Output Error Graphs

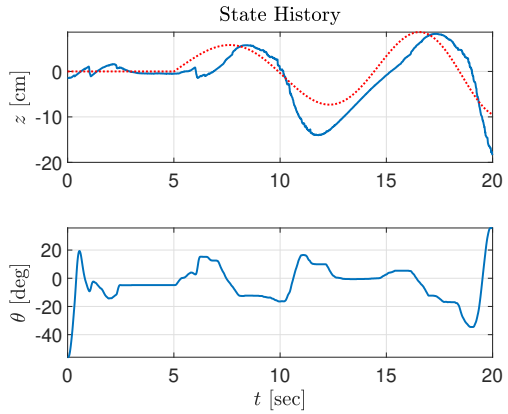


Fig. 13: LQR Hardware State History

VII. Conclusion

A. Controller Comparison

From our simulation results, it is clear that both the PD and LQR controllers performed very well, obtaining minimum scores of 0.18 and 0.15 respectively when run on a sine wave reference trajectory and 1.36 and 1.49 respectively when run with a square wave reference. In both reference trajectory cases, the PD controller performed better. This was likely due to the fact that LQR controllers use a linearized version of the the system while PD controllers do not.

To derive the gains for an LQR controller, a non-linear system must first be linearized around its target point. In doing so, the system is approximated and some aspects of the true dynamics are lost in this process. For a PD controller, the dynamics are not altered, the input is simply adjusted in terms of the error. This means that a PD controller theoretically models the system more accurately than an LQR controller in instances where the linearization is very different from the true dynamics at that moment in time. The ability to tune the PD controller gains also allows the controller to be tuned more intuitively than the Q and R matrices used in LQR while solving the ARE.

B. Sim-to-Real Gap

Bridging the sim-to-real gap is often one of the biggest challenges when implementing a control system. In our case specifically, this was clearly an issue as the scores for both of our controllers significantly increased when implemented on hardware rather than in sim. There are many factors that likely compounded to cause this.

The main contributor to the sim-to-real gap is inaccuracies in the system model. The equations we used to model the lever arm beam and ball and the servo motor powering it were all approximations of the hardware. Differences in the parts as well as physical factors such as friction between the ball and the beam as well as the internal inertia of the servo motor all accumulate error between the simulated and real models.

The primary issue we faced specifically was figuring out the best process to tune our controllers once they were deployed on the hardware. This proved to be difficult as the impact of each change on the controllers was less obvious on the hardware than in sim and the system often did not change as we expected it to as we tuned the controllers. To get to our final scores, a lot of tuning was necessary, and the final parameters were fairly different from the ones in sim.

C. Lessons Learned

Running this project on hardware was a great lesson in the challenges of deploying a controller that may seem effective in sim. Figuring out how to rework controllers to be effective on the physical hardware and bridging the sim-to-real gap was really challenging and we were definitely able to gain some experience in doing so from this lab.

If we were to repeat the lab, I think we could try to develop a more systematic way of tuning the controllers. In sim, one option could be figuring out a way to automate iteratively tuning the gains and matrices used as parameters for the controllers. This would allow us to essentially automate optimizing the gains which we could then use as an optimized baseline for the real hardware testing.

This lab was also a great lesson in seeing real world applications of these controllers in a setting where we could clearly see the differences in their effectiveness as well as how the different parameters truly effect the system response. The real world is full of similar hardware applications that are modeled non-linearly, such as robotics and autonomous cars, so this experience was really valuable in exposing us to the control systems underlying them.

Appendix

GitHub Link
PD simulation Sinusoidal Input Video
PD simulation Square Input Video
LQR simulation Sinusoidal Input Video
LQR simulation Square Input Video

PD Hardware Video
LQR Hardware Video

References

[1] Quanser, "Ball and beam - introduce unstable closed loop system control concepts," in *https://www.quanser.com/products/ball-and-beam/*.
[2] N. M. A. Ilka and J. Sjöberg, "An iterative newton's method for output-feedback lqr design for large-scale systems with guaranteed convergence," in *2019 18th European Control Conference (ECC)*, 2019.
[3] M. T. Xu Chen, "Introduction to modern controls – with illustrations in matlab and python." 2023.