EE222 Project Report Part 1

Spring 2025

1 Controllers

1.1 LQR Controller

1.1.1 Implementation

The LQR controller is a cost-minimizing linear control scheme. Since we are trying to control a non-linear system, we must first linearize around the current position. This is done by taking the Jacobian of the system, generating an A matrix of the form:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0.0051 * cos(x_3) * sin(x_3) * x_4^4 + 0.4183 * cos(x_3) & -0.0102 * x_4^3 * cos(x_3)^2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -40 \end{bmatrix}$$

With x_1, x_2, x_3, x_4 being plugged in based on the current state x. This lets us construct a system of the form $\dot{x} = Ax + Bu$. After obtaining a linear system, we then use Matlab's built-in **lqr()** function to obtain the optimal control matrix K. This allows us to set the velocity input to $u = -K * x'$. This process is repeated at every timestep.

1.1.2 Comments

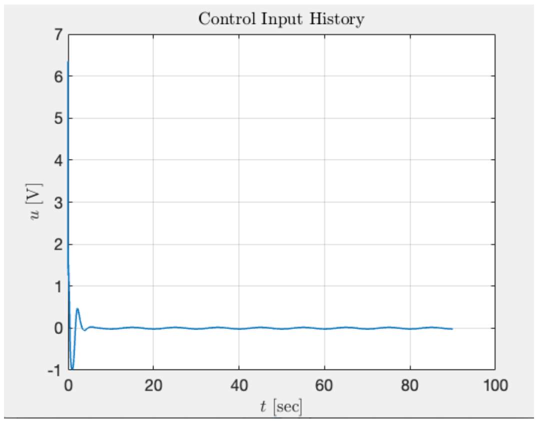
These are the LQ values we ended up choosing:

$$Q = \begin{bmatrix} 260 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad R = 1$$

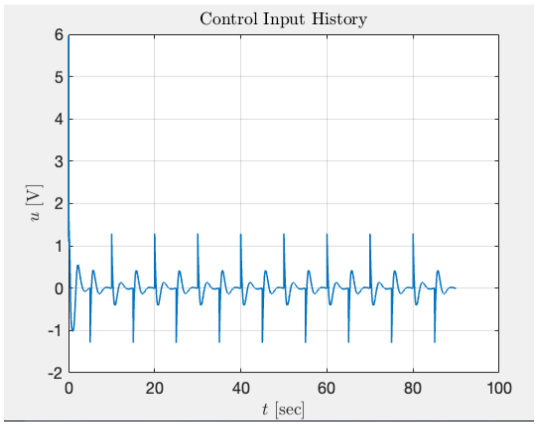
The LQR controller performed quite well in following the desired trajectory. The majority of the error seems to be due to the initial position and moving from the edge to the center of the beam. For the sinusoidal input, since the input is continuously differentiable, we see that the state does a very good job of following the desired trajectory with a small time delay. For the square input, we see a pretty sharp jump in error whenever the input changes, with the actual state then quickly converging on the desired position. Since we are only working with a reactionary control input, we are unable to predict when the input would change, needing some time for the controller to react.

1.1.3 Results

Metric	Sinusoidal	Square wave
Average Tracking Error	0.0004	0.0015
Average Energy Consumption	0.0216	0.0741
Tracking Cost	0.72	2.64
Energy Cost	0.11	0.37
Total Score	0.83	3.01

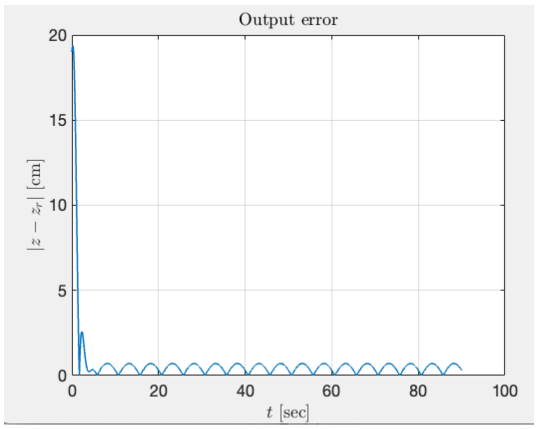


(a) Sinusoidal

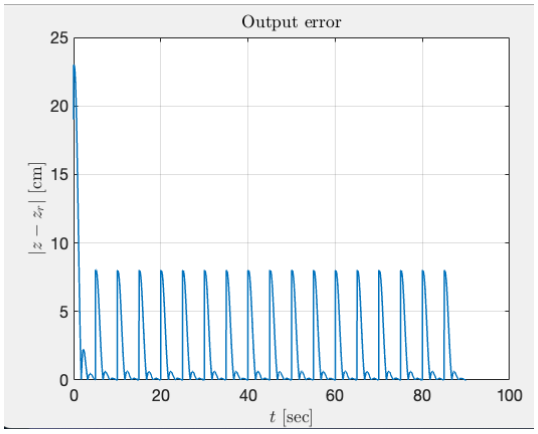


(b) Square wave

Figure 1: LQR Simulation Control Input Graphs

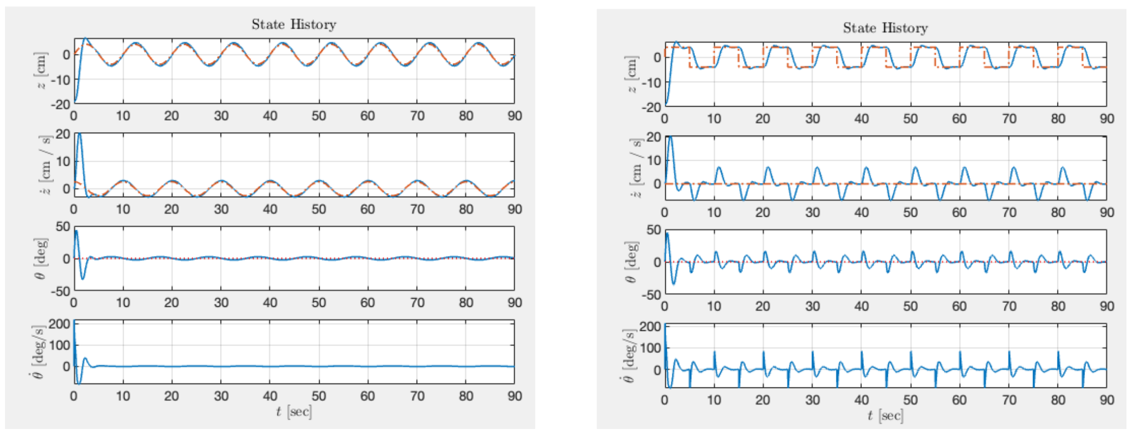


(a) Sinusoidal



(b) Square wave

Figure 2: LQR Simulation Output Error Graphs



(a) Sinusoidal (b) Square wave

Figure 3: LQR Simulation State History

1.2 LQG Controller

1.2.1 Implementation

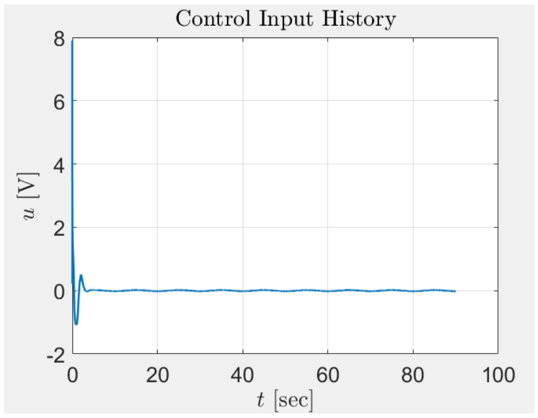
The linear quadratic Gaussian controller makes use of an extended Kalman filter and a linear quadratic regulator. An LQG controller uses the EKF controller for state measurement when noise is present and the LQR controller for optimal control using the estimated states. In this project, we implement an LQG-like controller by linearizing around the state at each timestep. The linearization applied is the same as that used for the LQR controller. This linearized system is then used to generate a gain matrix using MATLAB's **lqg** function which is then used to determine the input voltage. The controller is also split into two phases, the first for large corrections when the ball is very far from the desired target, and the second for smaller corrections when it is closer to the desired position. In the first phase, the gains are set to allow the ball to move as fast as possible towards the target without overshooting while the second phase is tuned for smaller corrections.
The matrices used to implement the LQG controller are as follows.

$$QXU = \begin{cases} \begin{bmatrix} 365 & 0 & 0 & 0 & 0 \\ 0 & 75 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} & \text{if abs}(x_1) > 0.005 \\ \begin{bmatrix} 550 & 0 & 0 & 0 & 0 \\ 0 & 150 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} & \text{else} \end{cases}, \; QWV = \begin{bmatrix} 0.025 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.05 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.025 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.05 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.025 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.025 \end{bmatrix}$$

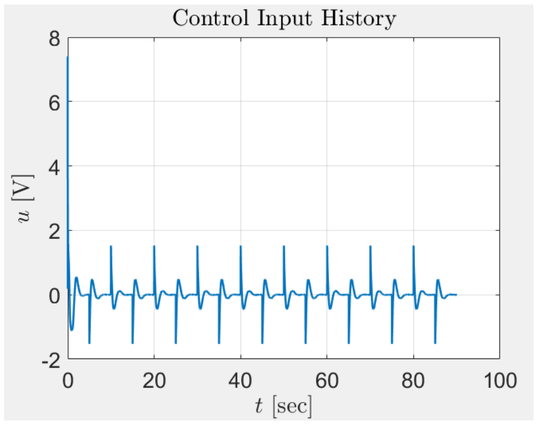
1.2.2 Comments

1.2.3 Results

Metric	Sinusoidal	Square wave
Average Tracking Error	0.0004	0.0014
Average Energy Consumption	0.0253	0.0860
Tracking Cost	0.70	2.60
Energy Cost	0.13	0.43
Total Score	0.83	3.03

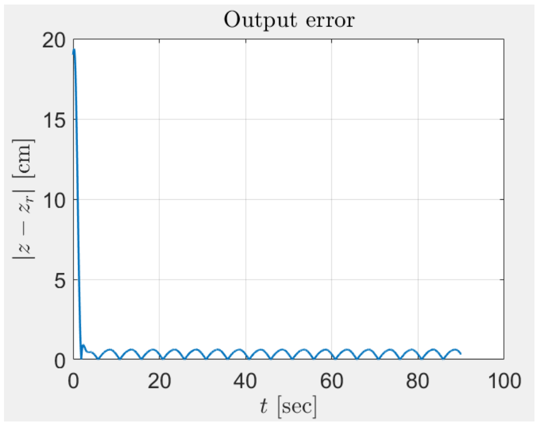


(a) Sinusoidal

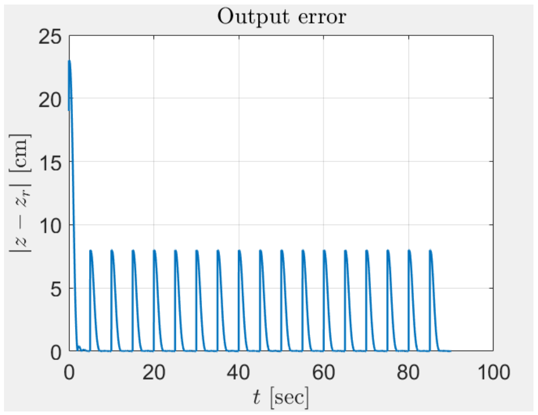


(b) Square wave

Figure 4: LQG Simulation Control Input Graphs

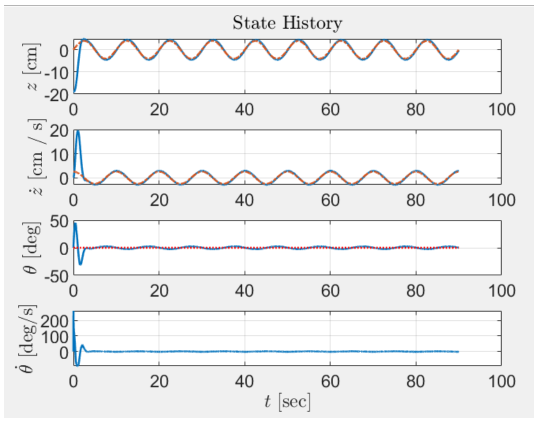


(a) Sinusoidal

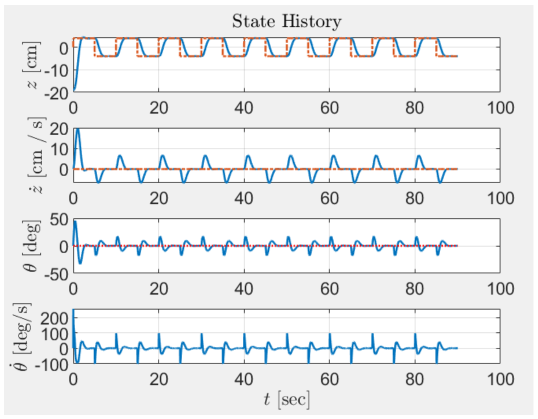


(b) Square wave

Figure 5: LQG Simulation Output Error Graphs



(a) Sinusoidal



(b) Square wave

Figure 6: LQR Simulation State History

2 Observers

2.1 Basic State Estimation

For basic testing, we created a basic estimator for the state. We have a desired state given by: $[x1, x2, x3, x4]$ Our observed output $y = [y_1, y_2]$ gives us two of the state values: $x_1 = y_1$ and $x_3 = y_2$ that represent the linear position z and angular position θ. we can then take the discrete-time derivative of these states to get the remaining two state variables $x_2 = \dot{z}$ and $x_4 = \dot{\theta}$ giving us: $\dot{z} = \frac{z - z_{prev}}{t - t_{prev}}$ and $\dot{\theta} = \frac{\theta - \theta_{prev}}{t - t_{prev}}$.

The basic estimator does a pretty good job of getting the state estimate. Single derivative based estimators are usually fine (albeit a bit noisy) which is convenient in this case.

2.2 Extended Kalman Filter

We also constructed an Extended Kalman-Filter based estimator. The base Kalman Filter only works on linear systems, so we must first linearize the system around each operating point. This is the same method as done in both the LQR and LQG controllers.
After linearizing the system, we obtain a system of the form $\dot{x} = Ax + Bu + W$ and $y = Cx + V$ with W and V representing zero-mean gaussian noise values.
Using the Kalman filter, we derive the state estimate $\hat{x}$. Since we have a continuous time system model, we need to use the continuous time Kalman filter. This takes the form:

$$\dot{\hat{x}} = A\hat{x}(t) + Bu(t) + F(t)[y(t) = C\hat{x}t]$$

$$F(t) = M(t)C^T V^{-1}$$

$$\dot{M}(t) = AM(t) + M(t)A^T + W - M(t)C^T V^{-1} CM(t)$$

With initial conditions:

$$\hat{x}(0) = x_0 = [-0.19; 0.00; 0; 0] \qquad M(0) = X_0$$

However, since we are only able to perform discrete-time inputs, we had to try to use use discrete time integration to get values of $\hat{x}(t)$ and $M(t)$ meaning our results were not ideal. Ultimately, the Kalman filter worked, though its results were not ad good compared to the simple state estimator. Although it was able to get a pretty accurate $\hat{x}(t)$ after some time, it was that initial ramp-up which increases the final cost.

Nevertheless, it will still be useful to test our implementation of the Extended Kalman Filter on the physical ball and beam. Unlike in simulation, real-world testing will likely have input and output noise play a more significant role, leading to the need for a better state estimator than the basic one.

3 Bibliography

References

[1] *Help Center*. (2024). MATLAB Help Center: LQR [Online] Available: `https://www.mathworks.com/help/control/ref/lti.lqr.html`

[2] *Help Center*. (2024). MATLAB Help Center: LQG [Online] Available: `https://www.mathworks.com/help/control/ref/ss.lqg.html`

[3] Xu Chen, Masayoshi Tomizuka (2023). Introduction to Modern Controls – with Illustrations in MATLAB and Python.

4 Appendix

GitHub Link
LQR Sin video
LQR Square video
LQG with Sinusoidal Input Video
LQG with Square Input Video