Balancing Robot

Izzy Mones and Heidi Dixon

May 7, 2025

1 Helpful Papers

Rani and Kamlu (2025) Feriyonika and Hidayat (2020) Brunton and Kutx (2019) Rao (2020)

Robot Design

Should have list of all the components. Maybe a picture. Do we need to show circuit stuff?

Measuring the Robot State

The position x and velocity $\dot{x}$ along the x axis are computed from the motor encoders. The encoders return a number of counts which is positive in the clockwise direction and negative in the counter clockwise direction. Using the number of counts per wheel rotation and the circumference of the wheel we can compute the distance in the x direction. The instantaneous velocity can be estimated by computing the change in position over a very small time window times the length of the window. The angle θ is computed from the accelerometer and angular velocity $\dot{\theta}$ is given by the gyroscope.

Model

Need a drawing of model.

We modeled our balancing robot as a pendulum cart system.The wheels and motors are considered to be the cart. All other parts of the robot pivot around the axis created by the wheels and are considered part of the pendulum. A thorough discussion of this model is found in Brunton and Kutx (2019). Our system is described by the system of differential equations

$$\dot{x} = \dot{x} \tag{1}$$

$$\ddot{x} = \frac{-mg\cos(\theta)\sin(\theta) + mL\dot{\theta}^2\sin(\theta) - \delta\dot{x} + u}{M + m\sin^2\theta} \tag{2}$$

$$\dot{\theta} = \dot{\theta} \tag{3}$$

$$\ddot{\theta} = \frac{(m+M)g\sin(\theta) - \cos(\theta)(mL\dot{\theta}^2\sin(\theta) - \delta\dot{x}) - \cos(\theta)u}{L(M + m\sin^2\theta)} \tag{4}$$

where x is the cart position, $\dot{x}$ is the cart velocity, θ is the pendulum angle, $\dot{\theta}$ is the angular velocity, m is the pendulum mass, M is the cart mass, L is the distance from the pivot point to the center of mass of the pendulum, g is the gravitational acceleration, δ is a friction damping on the cart, and u is the control force applied to the cart. This forms a set of differential equations

$$\frac{d}{dt}\boldsymbol{x} = f(\boldsymbol{x}, \boldsymbol{u}) \tag{5}$$

where $\boldsymbol{x}$ is the state vector $\boldsymbol{x} = [x, \dot{x}, \theta, \dot{\theta}]$ and $\boldsymbol{u}$ is the input vector $\boldsymbol{u} = [u]$.

Modeling Motors

Talk about how we didn't model motor circuit. We assumed that torque is a linear function of duty cycle. We tuned the coefficient experimentally. This model is probably pretty good as long as are not near their max duty cycle.

Linearization

To build a control system for our model we will linearize the non-linear system of equations (1), (2), (3), and (4) around a fixed point $(\boldsymbol{x}_r, \boldsymbol{u}_r)$ where $\boldsymbol{x}_r$ is the position where the robot is vertical, unmoving and positioned at the origin and $\boldsymbol{u}_r$ is the input with motor torque at zero.

The nonlinear system of differential equations (5) can be represented as a Taylor series expansion around the point $(\boldsymbol{x}_r, \boldsymbol{u}_r)$.

$$f(\boldsymbol{x}, \boldsymbol{u}) = f(\boldsymbol{x}_r, \boldsymbol{u}_r) + \left.\frac{\boldsymbol{df}}{\boldsymbol{dx}}\right|_{x_r} (\boldsymbol{x} - \boldsymbol{x}_r) + \left.\frac{\boldsymbol{df}}{\boldsymbol{du}}\right|_{u_r} (\boldsymbol{u} - \boldsymbol{u}_r) + \left.\frac{\boldsymbol{d^2 f}}{\boldsymbol{dx^2}}\right|_{x_r} (\boldsymbol{x} - \boldsymbol{x}_r)^2 + \left.\frac{\boldsymbol{d^2 f}}{\boldsymbol{du^2}}\right|_{u_r} (\boldsymbol{u} - \boldsymbol{u}_r)^2 + \cdots \quad (6)$$

Because $(\boldsymbol{x}_r, \boldsymbol{u}_r)$ is a fixed point, we know that $f(\boldsymbol{x}_r, \boldsymbol{u}_r) = 0$. Additionally, this approximation is only accurate in a small neighborhood around $(\boldsymbol{x}_r, \boldsymbol{u}_r)$. In this neighborhood, we can assume that the values of both $(\boldsymbol{x} - \boldsymbol{x}_r)$ and $(\boldsymbol{u} - \boldsymbol{u}_r)$ are small, so higher order terms of this series will go to zero. So a fair estimate of our system is

$$\frac{d}{dt}\boldsymbol{x} \simeq \left.\frac{\boldsymbol{df}}{\boldsymbol{dx}}\right|_{x_r} (\boldsymbol{x} - \boldsymbol{x}_r) + \left.\frac{\boldsymbol{df}}{\boldsymbol{du}}\right|_{u_r} (\boldsymbol{u} - \boldsymbol{u}_r) \quad (7)$$

where $\left.\frac{\boldsymbol{df}}{\boldsymbol{dx}}\right|_{x_r}$ is the Jacobian matrix for our system of equations $f(\boldsymbol{x}, \boldsymbol{u})$ with respect to $\boldsymbol{x}$ and $\left.\frac{\boldsymbol{df}}{\boldsymbol{du}}\right|_{u_r}$ is the Jacobian matrix with respect to $\boldsymbol{u}$. Both are then evaluated at the fixed point $(\boldsymbol{x}_r, \boldsymbol{u}_r)$. Our partial differentials with respect to $\boldsymbol{x}$ are

$$\frac{\partial f_1}{\partial x} = \frac{\partial f_1}{\partial \theta} = \frac{\partial f_1}{\partial \dot{\theta}} = \frac{\partial f_3}{\partial x} = \frac{\partial f_3}{\partial \dot{x}} = \frac{\partial f_3}{\partial \theta} = 0$$

$$\frac{\partial f_1}{\partial \dot{x}} = \frac{\partial f_3}{\partial \dot{\theta}} = 1$$

$$\frac{\partial f_2}{\partial x} = 0$$

$$\frac{\partial f_2}{\partial \dot{x}} = \frac{-\delta}{M + m\sin^2\theta}$$

$$\frac{\partial f_2}{\partial \theta} = \frac{(M + m\sin^2\theta)(-mg(\cos^2\theta - \sin^2\theta) + mL\dot{\theta}^2\cos\theta) - (-mg\cos(\theta)\sin(\theta) + mL\dot{\theta}^2\sin(\theta) - \delta\dot{x} + u)(2m\sin\theta\cos\theta)}{(M + m\sin^2\theta)^2}$$

$$\frac{\partial f_2}{\partial \dot{\theta}} = \frac{mL\sin\theta}{M + m\sin^2\theta} \cdot 2\dot{\theta}$$

$$
\begin{aligned}
\frac{\partial f_4}{\partial x} &= 0 \\
\frac{\partial f_4}{\partial \dot{x}} &= \frac{\delta \cos\theta}{L(M+m\sin^2\theta)} \\
\frac{\partial f_4}{\partial \theta} &= \frac{(L(M+m\sin^2\theta))((M+m)g\cos\theta - mL\dot{\theta}^2(\cos^2\theta - \sin^2\theta) - \delta\dot{x}\sin\theta + u\sin\theta)}{L^2(M+m\sin^2\theta)^2} \\
&\quad - \frac{((m+M)g\sin(\theta) - \cos(\theta)(mL\dot{\theta}^2\sin(\theta) - \delta\dot{x}) - \cos(\theta)u)(2mL\sin\theta\cos\theta)}{L^2(M+m\sin^2\theta)^2} \\
\frac{\partial f_4}{\partial \dot{\theta}} &= -\frac{2\dot{\theta}mL\cos\theta\sin\theta}{L(M+m\sin^2\theta)}
\end{aligned}
$$

The Jacobian matrix for our system of equations evaluated at $\boldsymbol{x}_r = [0\ 0\ \pi\ 0]$ and $\boldsymbol{u}_r = [0]$ gives the matrix

$$
A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{\delta}{M} & -\frac{mg}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{\delta}{ML} & -\frac{(m+M)g}{ML} & 0 \end{bmatrix} \tag{8}
$$

Our partial differentials with respect to $\boldsymbol{u}$ are

$$
\begin{aligned}
\frac{\partial f_1}{\partial u} &= 0 \\
\frac{\partial f_2}{\partial u} &= \frac{1}{M+m\sin^2\theta} \\
\frac{\partial f_3}{\partial u} &= 0 \\
\frac{\partial f_4}{\partial u} &= -\frac{\cos\theta}{L(M+m\sin^2\theta)}
\end{aligned}
$$

Evaluating these equations at $\boldsymbol{x}_r = [0\ 0\ \pi\ 0]$ and $\boldsymbol{u}_r = [0]$ gives the matrix

$$
B = \begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ \frac{1}{ML} \end{bmatrix} \tag{9}
$$

Now we can approximate our system of equations (1), (2), (3), and (4) with the linear system

$$
\frac{d}{dt}\boldsymbol{x} = A(\boldsymbol{x} - \boldsymbol{x}_r) + B(\boldsymbol{u} - \boldsymbol{u}_r). \tag{10}
$$

Two Variable Model

We also built a two variable model for the balancing robot that has a state consisting of only the robot angle and angular velocity. This model works if you don't need to control the position of the robot. Working with this model was useful for testing purposes since it is a simpler system. It has the equations of motion

$$
\begin{aligned}
\dot{\theta} &= \dot{\theta} \\
\ddot{\theta} &= \frac{(m+M)g\sin(\theta) - \cos(\theta)(mL\dot{\theta}^2\sin(\theta)) - \cos(\theta)u}{L(M+m\sin^2\theta)}
\end{aligned}
$$

and the linearized matrices

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

$$
B = \begin{bmatrix} 0 \\ \frac{1}{ML} \end{bmatrix}
$$

Algorithm

Linear Quadratic Regulator

A Linear Quadratic Regulator (LQR) is a closed loop feedback system that uses a linearized model of the system based on differential equations to compute the optimal control parameters to stabilize the system around a fixed point. Additionally, LQR control allows some level of optimization over performance requirements, allowing tradeoffs between reducing the power consumption of our motors and the stability of the system. Our LQR algorithm and accompanying simulations were coded using Python's Control Systems library `control` and the numerical and scientific computing library `numpy`.

We need our linear system (10) to be **controllable**. A system is controllable if the column space of the controllability matrix

$$\mathcal{C} = \begin{bmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{bmatrix} \tag{11}$$

has n linearly independent columns, where n is the number of variables in our state. At a high-level, controllability means that our control $B\boldsymbol{u}$ has the ability to affect all of our state variables in $\boldsymbol{x}$. This is easily tested in Python with the line

```
print(np.linalg.matrix_rank(ctrb(A, B)))
```

For our system, the controllability matrix has rank 4 confirming that our system is controllable.

LQR works by finding a matrix K that we can use to compute our control input $\boldsymbol{u}$ from the current state $\boldsymbol{x}$

$$\boldsymbol{u} = -K(\boldsymbol{x} - \boldsymbol{x}_r). \tag{12}$$

We know that the bahaviour of our system can be approximated with our linearized model (10). If we substitute (12) into (10) we get

$$A(\boldsymbol{x} - \boldsymbol{x}_r) + B(-K(\boldsymbol{x} - \boldsymbol{x}_r)) = (A - BK)(\boldsymbol{x} - \boldsymbol{x}_r)$$

In a controllable system we can select a matrix K to make the real parts of all eigenvalues of $A - BK$ have negative values. This creates a stable system.

To select our matrix K, we define two diagonal matrices Q and R. Q contains weights for the cost of deviating from the goal state and R weights the cost of our control inputs. We use the Python `control.matlab.lqr` command to pick a K that minimizes a cost function built from Q and R. If the weight in R is low relative to the weights in Q, our choice of K will favor sticking close to our goal state with minimal limits on the size of motor torque. This can cause the robot to respond agressively to changes in state and make it overly sensitive to noise. Typically, an LQR implementation will require some tuning of the weights in Q and R to meet the requirements of the system.

```
from control.matlab import lqr

# equations of motion linearized about vertical pendulum position
A = np.array([[0, 1, 0, 0],\
    [0, -d/M, -m*g/M, 0],\
    [0, 0, 0, 1],\
    [0, -d/(M*L), -(m+M)*g/(M*L), 0]])

# linearization of control matrix
B = np.array([0,1/M,0,1/(M*L)]).reshape((4,1))
```

```
Q = np.array([[1, 0, 0, 0],\
    [0, 0.01, 0, 0],\
    [0, 0, 10, 0],\
    [0, 0, 0, 0.1]])
R = 1
K = lqr(A,B,Q,R)[0][0]
```

A properly tuned K matrix can be used in the control loop to compute the motor torque required to respond to the current state. This value is then used to power the motors.

```
def loop_iteration():
    theta, theta_dot = imu_sensor.angle_data()
    x, x_dot = motors.position_data()
    current_state = np.array([x, x_dot, theta, theta_dot])
    u = -K@(current_state - x_r)
    motors.run(u * duty_coeff)
```

Linear Quadratic Gaussian

A linear quadratic gaussian algortihm (LQG) is an extension of the LQR method with two advantages. It doesn't require sensor readings for all of the state variables and it can filter out noisy sensor data. Instead of reading the current state directly from the sensors, the algorithm estimates the current state using predictions from the linear model, the control input and readings from the sensors. Raw sensor data is averaged with the model prediction and state variables without sensors are estimated entirely. Our implementation estimates the full state from sensor readings of the position x, angle θ and the angular velocity $\dot{\theta}$.

The matrix C is our measurement model.

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{13}$$

Observable. Similar to controllable.

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{14}$$

```
Kf = lqr(A.transpose(), C.transpose(), Vd, Vn)[0].transpose()
```

Derive the Kalman filter matrix K_f using the `lqr` function from python control library. optimization of Q and R matrices

```
# This is our state disturbance matrix
Vd = np.eye(4)
# This is our sensor noise matrix
Vn = np.array([[1, 0], \
               [0, 1]])
Kf = lqr(A.transpose(), C.transpose(), Vd, Vn)[0].transpose()
```

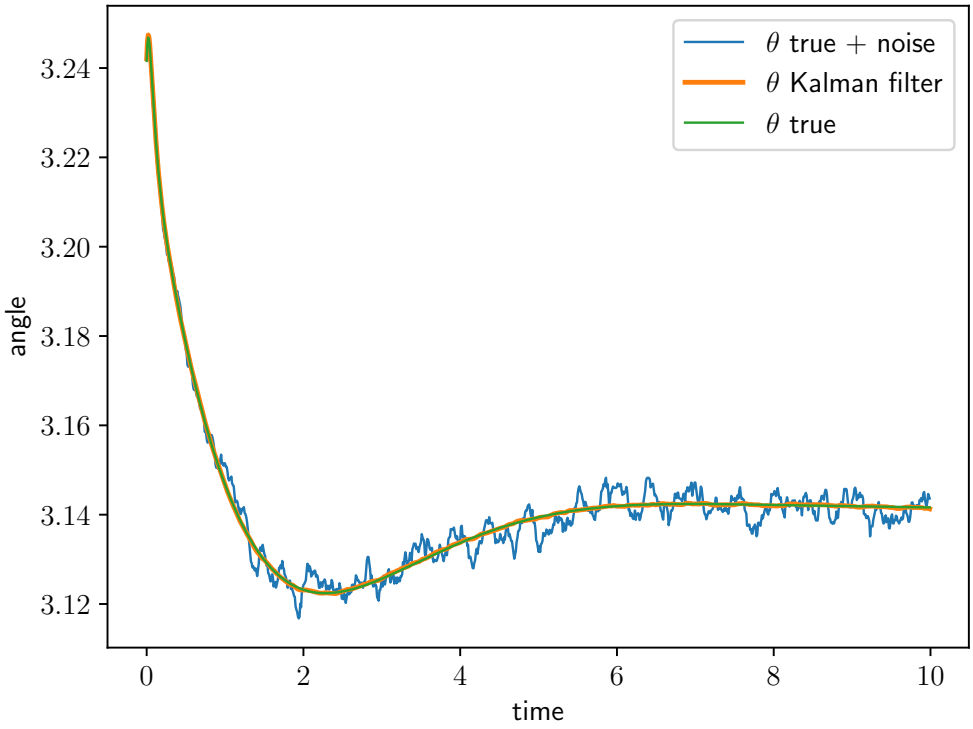

Figure 1: Simulation comparison of Kalman filter state estimation of angle with the true angle and noisy data.

To build the linear state space for our Kalman filter we build new matrices.

- $A_{kf} = A - K_f C$
- $B_{kf} = \begin{bmatrix} B & K_f \end{bmatrix}$
- $C = I_4$
- D is a 0 matrix with the same dimensions as K_f

These form a new linear system

$$\frac{d}{dt}\boldsymbol{x} = A_{kf}\boldsymbol{x} + B_{kf}\boldsymbol{u} \tag{15}$$

$$\boldsymbol{y} = C_{kf}\boldsymbol{x} + D\boldsymbol{u} \tag{16}$$

Our input vector $\boldsymbol{u} = [u, x_s, \theta_s, \dot{\theta}_s]$ is our motor torque u and our two sensor readings, position x_s and angular velocity $\dot{\theta}_s$

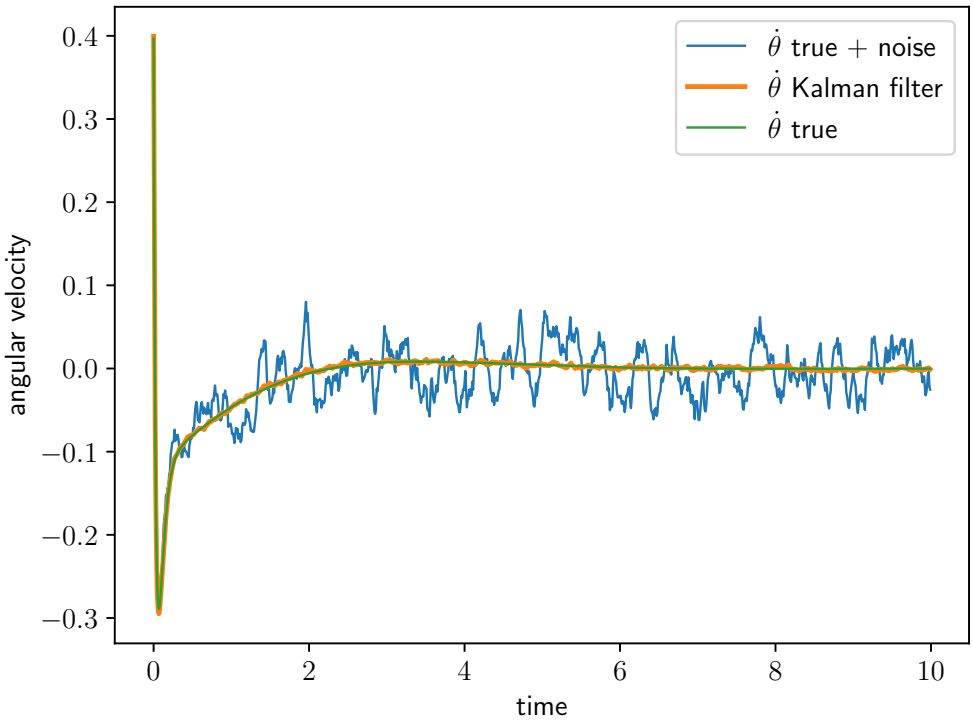

Figure 2: Simulation comparison of Kalman filter state estimation of angular velocity with the true angular velocity and noisy data.

1.1 Simulations with Python

Continuous vs. Discrete Models

Experiments

Conclusions

References

Monika Rani and Sushma S. Kamlu. Optimal lqg controller design for inverted pendulum systems using a comprehensive approach. *Scientific Reports*, 15(1):4692, Feb 2025. ISSN 2045-2322. doi: 10.1038/s41598-025-85581-3. URL `https://doi.org/10.1038/s41598-025-85581-3`.

Feriyonika Feriyonika and Asep Hidayat. Balancing control of two-wheeled robot by using linear quadratic gaussian (lqg). *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 12(3):55–59, Aug. 2020. URL `https://jtec.utem.edu.my/jtec/article/view/5619`.

Steven Brunton and J. Nathan Kutx. *Data Driven Science & Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019.

Akshay S Rao. Lqr-balancebot. `https://github.com/iamAkshayrao/LQR-BalanceBot/tree/master`, 2020.