Inverted Pendulum & PID Control

What Is An Inverted Pendulum?

Definition

An inverted pendulum is a pendulum in which its centre of mass is above the pivot point, making it unstable and prone to toppling without active control systems. It can be controlled by moving the base. A basic example of this is balancing a ruler or a pencil on the tip of your finger by moving your finger so that the pencil's centre of mass stays above the pivot.

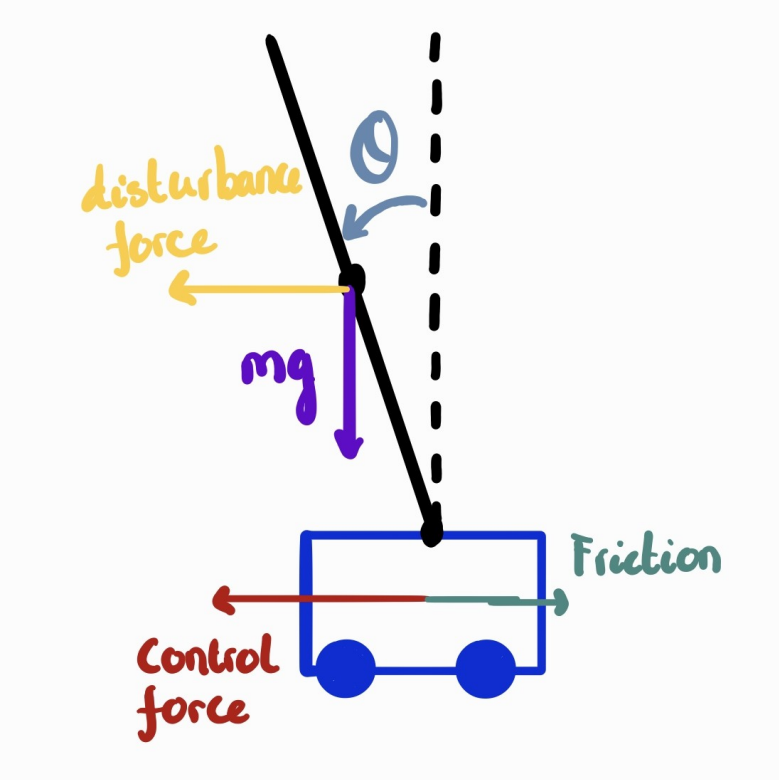

Figure 1: Diagram of an inverted pendulum (Made by Giorgi)

Measurements

The diagram shown above (Figure 1) displays the forces acting on the pendulum. To control the pendulum, we need a way to calculate the force and acceleration required to move the base, given the angle, angular velocity, and mass, to ensure that the centre of mass is always above the base. To get these measurements in robotics, you need to use an IMU which contains an Accelerometer (measures linear acceleration and gravity; it is used to determine the static tilt angle relative to the ground), a Gyroscope (measures the rate of rotation (angular velocity) around the axes, telling the controller how fast the pendulum is falling), and to estimate the tilt angle, you use an **Extended Kalman Filter**, providing a highly accurate, real-time estimation of the pendulum's tilt angle, velocity, and position by fusing noisy sensor data.

How to control an inverted pendulum

PID Control (Proportional-Integral-Derivative):

The error term is the difference between the pendulum being perfectly upright when it is directly above the pivot point (0 rads) and the pendulum's actual angle at that moment.

$$K_p \ e(t)$$

The first part of PID control is the **proportional** term, known as the P value, and its equation is shown above. This applies a correcting force based on how far the pendulum is leaning away from the centre position (the error). A larger angle produces a greater corrective force, while a smaller angle produces a smaller one. Although this helps the pendulum respond quickly, proportional control alone can make the system unstable. This is because the pendulum may swing past the centre point at too high a speed, causing repeated side-to-side oscillations and making the robot unstable.

$$K_i \int e(t)\,dt$$

The final part is the **integral** term, or I value, shown with the equation above. In some situations, proportional control may not produce sufficient force to fully balance the pendulum when the angle error is very small. Small continuous forces, such as gravity or friction, can still cause the pendulum to drift away from the centre over time. To fix this, the integral term tracks the total error over a period of time and adds a small extra correction to remove any remaining offset. However, the integral effect is normally kept quite low because if too much past error is accumulated, the system can become unstable and the pendulum movement may become even more erratic.

$$K_d \frac{de(t)}{dt}$$

To help control this erratic movement, the **derivative** term is added to the equation shown above. The D value looks at the rate of change of the error rather than just the size of the error itself. Acting as a braking force to slow the pendulum as it approaches the upright position. When the pendulum is moving rapidly towards the target angle, the derivative term produces an opposing effect that reduces the speed of the correction, as the rate of the error will be decreasing, creating a negative value. This makes the overall movement smoother and reduces overshoot and excessive oscillation.

Combining all these terms gives us the following equation, which can be used to control an inverted pendulum smoothly. (u(t) in a two-wheeled robot would be the motor voltage)

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

Potential Issue With PID And How To Fix It

Windup

One potential problem with using PID control is an **integral windup.** This is because in a real-world scenario, motors have a physical limit where they can reach a maximum RPM, and if the system experiences a constant error, such as the pendulum being knocked over, this error will build up through the integral, causing it to request a higher RPM than is physically possible by the motor. However, the main issue is that when the robot returns to its correct position, the controller has to unwind all the accumulated excess error, causing the pendulum to overshoot significantly and fall over. To fix this, an anti-windup method is used, with the most popular being clamping. Clamping is where it conditionally turns off the integrator by setting its error input to zero when two things happen: the controller outputs its saturation limit, and the error is trying to push the controller even further into that saturation limit, and then the integral is turned back on when the error sign changes from + to - or - to +.

Noise

Another huge issue is noise, which can be caused by motor vibration, sensor inaccuracies, external factors, etc. One of the biggest issues is high-frequency noise, as even when the amplitude is low, it can create a very high derivative value in the PID control, causing inaccuracies. This can be fixed through a filter. One of the best filters you can use is the extended Kalman filter. A Kalman filter removes noise by continuously comparing a mathematical prediction of a system's state, such as the precise tilt angle, against the actual, jittery sensor measurements it receives. It mathematically blends these two values using a dynamic weight called the Kalman Gain, trusting whichever source currently has less uncertainty to calculate a smooth, accurate estimate of the true state.

LQR Control System

An LQR system, compared to a PID control system, is a lot more complex. Developed 38 years after PID in the 1960s, an LQR system model offers much better performance in terms of stability and control. However, it requires more computation and, in robots, unlike PID, requires encoders on the motors.

Maths Behind LQR

State Vector

To begin with, you have a state vector X
And for a self-balancing robot, it would be represented with 4 variables, a 4x1 vector:

$$\begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix}$$

Figure 1

The state vector is useful because it encapsulates the system's exact physical state, allowing the Linear Quadratic Regulator (LQR) to compute the single, mathematically optimal motor torque required to keep the robot upright. These are the definitions for the terms shown:

Where x is the displacement from the robot's wheels to the designated reference origin point. Telling the robot where it is relative to its starting point, which is recorded using the robot's encoder. This value is used by the robot so it doesn't drift and tries to move back to its original starting position.

(x-dot) represents the first time derivative of the position and is used by the LQR for momentum, as the system has to know the instantaneous speed of the robot to calculate the counter torque required to apply without the system falling over.

The tilt angle theta is the angle deviation from the robot's perfect vertical position and is used by the LQR system to ensure stability, as any deviation from the robot's vertical position creates a gravitational torque pulling the robot down. The LQR heavily penalises this variable in its cost function (**Q** matrix) to keep the tilt as small as possible.

The Theta dot represents the first time derivative of the angle deviation, which represents the angular velocity of the robot, representing how fast the robot is falling/recovering. This is used by the LQR much like the derivative function in the PID controller, which ensures that the robot doesn't overshoot. As the robot has a large tilt angle and is already rotating back to the original position, as measured by theta dot, it knows it is recovering and will not overshoot with excessive motor power.

Equation: $\dot{X} = AX + Bu$

In terms of how the robot knows if it is stabilised or falling, it uses eigenvalues.
Which is a scalar value, λ. It is used to see if a robot is stabilised due to this equation:

$$\dot{x} = \lambda x$$

This is important because the rate of change of x with respect to time equals λx. Which says that the rate at which x changes is proportional to x.
Which is an exponential function and can be represented as:

$$x(t) = x(0)e^{\lambda t}$$

This shows that if $\lambda > 0$ the robot will be unstable as $e^{\lambda t}$ will increase exponentially as time passes. If $\lambda = 0$ or < 0, it will be stable or stabilising, respectively. As it would be decreasing to 0.

Where eigenvector A will have 4 distinct eigenvectors (v1,v2,v3,v4) and also 4 distinct eigenvalues($\lambda 1, \lambda 2, \lambda 3, \lambda 4$)
And to show the robot's stability, we can sum each mode:

$$x(t) = C_1 e^{\lambda_1 t} v_1 + C_2 e^{\lambda_2 t} v_2 + C_3 e^{\lambda_3 t} v_3 + C_4 e^{\lambda_4 t} v_4$$

Where C is the coefficient of the term when t=0

Putting everything together: $\dot{X} = AX + Bu$

Where:

- **X** is the state vector (4X1)
- **u** is the control input, which is just one number and is the force that the motor pushes at the ground.
- **A** is the system matrix (4x4) which captures the robot's own dynamics: which is what the robot will do naturally by itself.
- **B** is the input matrix (4X1), which shows that if the robot is pushed with force **u,** what states does it affect and by how much.

  So, to put the equations into words: the rate of change = the physics of the robot alone + what you are commanding the robot to do.

The A matrix: From the equation: $\dot{X} = AX + Bu$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{m^2 g l^2}{p} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{mgl(M+m)}{p} & 0 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix}, \quad \dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix}$$

Row 1:

The first row of the matrix A takes out the robot's change of rate of position (Xdott) from the state vector shown. As the first term in $\dot{X}$ vector in the equation: $\dot{X} = AX + Bu$ is $\dot{x}$.
Therefore, we have to simply take x^* from the state vector X as shown:
so row 1 of A = $\begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \Rightarrow \dot{x} = 0 * x + 1 * \dot{x} + 0 * \Theta + 0 * \dot{\Theta} = \dot{x}$

Row 3:

Row 3 will just be $\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$ *as we need* $\dot{\theta}$ from the state vector x, which is just its 4th variable, as the 3rd variable in the $\dot{X}$ vector is $\dot{\theta}$.

Row 2 and 4:

For row 2, we are looking to obtain $\ddot{x}$ in terms of θ (the angle deviation from the perfect vertical position). And for row 4, we are looking to get $\ddot{\theta}$ from θ.
In order to find these, we begin applying Newton's laws: one for the horizontal motion of the whole thing, one for the rotation of the body about its centre of mass

These are all the new terms in the following equations:
M: The total mass of the wheels and base.
m: The body mass.
l**:** The distance from the wheel axle up to the body's centre of mass.
I**:** The moment of inertia of the body about its own centre of mass.
g: The acceleration due to gravity (equal to 9.81).
F: The horizontal drive force generated at the ground by the wheel motors (interchangeable with the control input u).

$$(M + m)\ddot{x} - m\, l\ddot{\theta} \cos\theta + m\, l\, \dot{\theta}^2 \sin\theta = F \qquad (1)$$

$$(I + m\, l^2)\, \ddot{\theta} - m\, l\, \ddot{x} \cos\theta - m\, g\, l \sin\theta = 0 \qquad (2)$$

Equation (1) is total horizontal force = mass × acceleration for the system
Equation (2) is the rotational law for the body. The crucial term is $-mgl\sin\theta$, which is gravity and is what topples the robot.
However, the problem with these equations is that they are not linear due to the $\sin\theta$, $\cos\theta \wedge \dot{\theta}^2$ terms and eigenvalues and $AX + Bu$ is built for a linear system.

To make the equations linear, we can use the small-angle approximation, since a balancing robot only needs to deal with small values of θ, and a good control system will keep θ within a few degrees. ($sin\theta \approx \theta$, $cos\theta = 1$, $\theta^{\cdot 2} \approx 0$)

Now by substituting the small angle approximations into equations (1) and (2), we get these 2 linear equations:

$$(M + m)\ \ddot{x} - m\ l\ \ddot{\theta} = F \tag{2}$$

$$(I + m\ l^2)\ddot{\theta} - m\ l\ \ddot{x} = m\ g\ l\ \theta \tag{3}$$

Now we need to remove the two unknowns with one equation, that is $\ddot{x} =$, without having another $\ddot{x} \vee \ddot{\theta}$ inside the equation. To do this, we solve it simultaneously and get a common denominator p and get the following equations:

$$p \equiv I(M + m) + M\ m\ l^2 \tag{4}$$

And the results are:

$$\ddot{x} = \frac{m^2 g\ l^2}{p}\ \theta + \frac{I + m l^2}{p}\ F \tag{5}$$

$$\ddot{\theta} = \frac{mgl(M + m)}{p}\ \theta + \frac{ml}{p}\ F \tag{6}$$

Therefore, for row 2, it will be: $\begin{bmatrix} 0 & 0 & \frac{m^2 g\ l^2}{p} & 0 \end{bmatrix}$

And for row 4 it will be: $\begin{bmatrix} 0 & 0 & \frac{mgl(M + m)}{p} & 0 \end{bmatrix}$

Giving the final matrix of:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{m^2 g l^2}{p} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{mgl(M+m)}{p} & 0 \end{bmatrix}$$

The vector B: From the equation: $\dot{X} = AX + Bu$

$$\begin{bmatrix} 0 \\ \frac{I+ml^2}{p} \\ 0 \\ \frac{ml}{p} \end{bmatrix}$$

For vector B, the first row is 0, as we already get the $\dot{x}$ value from the AX part of the equation.
Row 3 is also 0, as once again we get the $\dot{\theta}$ from the AX part in $\dot{X}$ = AX + Bu

And for rows 2 and 3, we can use the values on the right-hand side of equations (5) and (6). (F can be interchanged with u).

$$\ddot{x} = \frac{m^2 g\ l^2}{p}\ \theta + \frac{I+m l^2}{p}\ F \qquad (5)$$

$$\ddot{\theta} = \frac{mgl(M+m)}{p}\ \theta + \frac{ml}{p}\ F \qquad (6)$$

Showing that the system with matrix A will be unstable:

By looking at just the angle and switching the motor off, so therefore f = 0, we get the equation

$$\ddot{\theta} = \frac{mgl(M+m)}{p}\ \theta = \omega^2\ \theta, \qquad \omega^2 \equiv \frac{mgl(M+m)}{p}\ \theta > 0$$

Unstable: $\ddot{\theta} = w\theta^2$ Stable: $\ddot{\theta} = -w\theta^2$

It is unstable because, when the growth rate coefficient is positive, it leads to exponential growth. So the angle doesn't swing back and forth; it just blows up instantly until the robot hits the floor.

Feedback Control: Steering λ to be negative

We know that the control system is naturally unstable, and we now need to make it stable through feedback, where it measures the state and feeds back to determine the control input.

Full-state feedback: u = −KX

One of the best ways to create a simple, powerful feedback rule is to make the control flow through a weighted combination of all the values in the state vector. And is shown as:

$$\mathrm{u} = \text{-}\mathrm{Kx} = \text{-}\begin{bmatrix} k_1 & k_2 & k_3 & k_4 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} = \text{-}(\mathrm{k}_1 x + k_2 \dot{x} + k_3 \theta + k_4 \dot{\theta})$$

In addition, we use a negative K to counter the robot's movement by pushing it in the opposite direction. And the k values inside the K vectors are referred to as gains.
Now, by substituting u =-KX into the original equation, we get:

$$\dot{X} = (A - BK)X$$

So, in order for the robot to be stable, the eigenvalue of (A - BK) needs to be negative using the same logic shown in the (Equation: Ẋ =AX + Bu) section, so the control system needs to use the BK value to steer the eigenvalue to be negative.

Is the robot controllable?

To see if the robot is controllable, we use the controllability matrix, which is the following:

$$C = \begin{bmatrix} B & AB & A^2B & A^3B \end{bmatrix}$$

Here, B is the direction your input pushes immediately, AB is where the push is a moment later, then A^2B is a moment after that, and so on. Then, using MATLAB, we can calculate the rank, which shows how many dimensions of the state equation can be controlled, so if the rank is equal to 4, the system is controllable, and if it is less than 4, it is not controllable. We will assume our robot is controllable.

Why this isn't the end

Firstly, an aggressive controller commands forces larger than the motor can output. As soon as the motor reaches its saturation limit, the linear design, which assumes the motor can output unlimited force, no longer describes what the LQR system wants.

In addition to this method, the gains are calculated through pole placement, where you first decide what you want your 4 eigenvalues to be (all negative to ensure the robot is stable) and write them as a polynomial's solutions, so $(\lambda - \lambda_1)(\lambda - \lambda_2)(\lambda - \lambda_3)(\lambda - \lambda_4)$ and through this you get a quartic. Then you take the quartic from the equation $\det(A - BK - \lambda I) = 0$, whose coefficients contain the unknown gains k_1, k_2, k_3, k_4 , and you set the coefficients to equal each other, creating four different equations with four unknowns, and through solving them simultaneously, you can find the k values. However, the issues with this are that when you set the two quartics to equal each other, you see that multiple k_1, k_2, k_3, k_4 values sit in one coefficient, so changing only one of the four gains shifts the other coefficients, which shifts all four eigenvalues at once. Therefore, the four gains cannot be treated as 4 independent dials, as there will be no single gain that will control a single eigenvalue for the states, and therefore you cannot control the controller by changing one gain and watching just one eigenvalue respond. Also, the pole placement method does not tell you where the four eigenvalues should be placed to make the robot behave well, as many different sets of negative eigenvalues all keep the robot stable, and the behaviours you actually care about how quickly the angle settles, how far the robot drifts along the floor and how sensitive it is to sensor noise don't correspond neatly to individual eigenvalues either. So designing a good controller this way becomes a process of trial and error: you guess a set of target eigenvalues, solve for the k values: k_1, k_2, k_3, k_4, simulate the result, and adjust. Giving no principled way of knowing whether the gains you finally settle on are the best ones possible, and also making the process very time-intensive.

The Linear Quadratic Regulator

To fix the issues, we need to find another way to compute which stabilising gain k_1, k_2, k_3, k_4 is the best out of all the stabilising gains.

Step 1: Scoring a controller with a cost function

So here we essentially try to compute a value that tells us how good the controller is, basically saying how good the k values are, where the lower the number, the better it is. There are two things that negatively impact the run:

1. The state being away from zero as we want θ and x to be as close to zero as possible as we want it to be upright, still and in place.
2. Using a lot of control effort. Because large motor forces drain the robot's battery, it can cause overheating, and the motors sometimes hit their physical limit, leading to harsh movements.

In order to be able to create this scoring system, we need to be able to turn a vector into a single number value and to do this, we use the quadratic form. Given a vector X and a square matrix Q:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}, \qquad Q = \begin{bmatrix} q_1 & 0 & 0 & 0 \\ 0 & q_2 & 0 & 0 \\ 0 & 0 & q_3 & 0 \\ 0 & 0 & 0 & q_4 \end{bmatrix}, \qquad \mathbf{x}^\mathsf{T} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}$$

And then by calculating Qx and then $x^T(Qx)$ we get the following, which will give us a single value:

$$X^T QX = = q_1 x^2_1 + q_2 x^2_2 + \ . \ . \ . \ + q_n x^2_n$$

As a 1 × n row times an n × n matrix times an n × 1 column gives a 1 × 1 result.

Now we can look at the LQR cost function, which is the following:

$$J = \int_0^\infty \left(x^T Qx + u^T Ru \right) \, dt$$

Where the $x^T Qx$ is the state deviation and $u^T Ru$ is the control effort.
Also, integration is used to sum up how bad the controller is over time. If the robot leans and moves for a long time, it will add up the badness from t=0 to the infinite future.

Step 2: What are Q and R?

We chose the matrices Q and R, which are the only factors within the LQR control system that we chose, and everything else is done automatically.

- For Q, each diagonal entry q_1, q_2, q_3, q_4 is how much we care about each state being zero. For example, setting q_3 as a large number means that you prioritise the tilt angle being zero and hate any tilt.

- For R, it is how much you care about saving control effort; the higher you set the value of R, the more you are going easier on the robot's motors.

- Overall, only the ratio matters, as it is the balance between Q and R and is what determines how the control system works.
  For example, a large (Q/R) ratio means that the control system will be fast, aggressive and effortful.
  And for a small (Q/R) ratio, the control system will be gentle and slow.

Step 3: the answer

Now, by minimising J, we use the following equation to find what the optimal value of K is:

$$K = R^{-1} B^T P$$

where the matrix P is the (symmetric, positive semi-definite) solution of the Continuous Algebraic Riccati Equation (CARE):

$$A^T P + PA - PB R^{-1} B^T P + Q = 0$$

Where all of A,B,R,Q are known, and you use the equation to solve for P. There is a long derivation for this formula, and also there is a way to solve for P; however, in our case

```
import numpy as np
from scipy.linalg import solve_continuous_are
P = solve_continuous_are(A, B, Q, R)
```

How it works

Overall, to show how it works, I will lay out the processes the robot goes through for the LQR system:

Step 1:

The physical parameters are chosen, for example:

Quantity	Symbol	Value
Wheel & base mass	M	0.5 kg
Body Mass	m	0.5 kg
Height of the body centre of mass above the axle	l	0.15 m
Body moment of inertia = $\frac{1}{3}ml^2$	I	0.00375 $kg\ \ m^2$
Gravity	g	9.81 m/s^2

And we also compute the value p:

$$p = I(M + m) + Mml^2 = 0.00375(1.0) + (0.5)(0.5)(0.15)^2 = 0.009375$$

Step 2:

Here we compute the A and B matrices:

$$\frac{m^2 g\ l^2}{p} = 5.886,\ \frac{mgl(M + m)}{p} = 78.48,\ \frac{I + ml^2}{p} = 1.6,\ \frac{ml}{p} = 8.0$$

So with these calculations, the A and B matrix is:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 5.886 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 78.48 & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ 1.6 \\ 0 \\ 8.0 \end{bmatrix}$$

Step 3:

In this step, we check if the robot is unstable and controllable, and to do this, we calculate the eigenvalues and use the controllability matrix:
First, to check whether the robot is unstable, we compute the eigenvalues of A and ignore B, since we are only looking at the robot's state with no input. In order to find the eigenvalue of A, we do $det(A - \lambda I) = 0$ which returns λ as 0, +8.86 and -8.86
The positive real part of the eigenvalue +8.86 shows that the robot is naturally unstable.
Then we use the controllability matrix: $C = \begin{bmatrix} B & AB & A^2B & A^3B \end{bmatrix}$ and compute its rank, which will return a rank of 4, showing that the robot is stable as the robot is able to steer all 4 states, so an LQR solution is guaranteed to exist.

Step 4:

Here we choose the values placed in the matrices Q and R. For Q, we care most about the tilt angle, as the robot falling is the main failure that we want to avoid, and we don't care as much about the position and the rates. For R, we set it to reflect that we are willing to expend a normal amount of motor effort.

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad , \quad R=1$$

So, essentially, we are saying we care 10 times more about the tilt angle than about all the other variables in the state vector.

Step 5:

Now compute the optimal K values:
First, we use the Riccati solver to compute the matrix P, and then $K = R^{-1}B^T P$ to return the K value, and in our case, it will equal:

$$K = \begin{bmatrix} -1.00 & -2.00 & 24.60 & 3.07 \end{bmatrix}$$

So, therefore, the motor command for each instant is:

$$u = -Kx = -(-1.00\ x - 2.00\ \dot{x} + 24.60\ \theta + 3.07\ \dot{\theta})$$

And by looking at the values, we can see that k_3 is by far the largest, as expected, indicating that a small deviation in the robot's tilt angle will produce the largest corrective force.
k_2 Reacts to how quickly the robot is tipping, catching a fall early rather than waiting for the tilt angle to increase. Without it, the robot would overshoot and oscillate. Finally, k_1, k_2 are smaller and gently keep the robot from drifting away while it balances.

Step 6:

This is the final step, where we check whether the robot is actually stable by computing the eigenvalues of A − BK, which return λ = −13.63, −6.07, −0.84 + 0.50 i, and −0.84 − 0.50 i.
Since all the real parts of the eigenvalues are negative, the system is stable. This is because the real part of the eigenvalue represents the decay of the $e^{\lambda t}$ The imaginary part of the eigenvalue is what represents the oscillations. For example, let's look at the eigenvalue -0.84 ± 0.50 here the real part is negative, so it will decay, but its magnitude is small, so it will decay slowly at around: 1 / 0.84 ≈ 1.2s and for the imaginary part of the eigenvalue, which is 0.05, means it oscillates at 0.5 rad/s. Though crucially, the decay time is much shorter than one oscillation period, so the motor dies away before the robot can complete one full swing, showing that it is well-damped and you will get roughly a single small overshoot which is then gone rather than a sustained wobble.

Overall, putting the four eigenvalues together, the two fast eigenvalues (-13.6 and -6.1) snap the angle back quickly and handle most of the initial error, while the slow complex pair ($-0.84 \pm 0.50i$) is what is left over and produces a gentle overshoot, followed by the slow final approach to rest.

Putting it into a robot

While the maths will compute K for us, the robot still needs to measure its 4 states many times per second, compute u = -kX, and convert it into a motor command.

A balancing robot runs the same short loop, typically 100–500 times every second, forever:

$$sense \rightarrow estimate\ the\ state\ x \rightarrow compute\ u = -kX \rightarrow drive\ motors \rightarrow repeat$$

And the u value is the force. Then we convert the force into PWM, allowing us to control the motors and stabilise it.