logo, grafik, grafik tasarım, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

**ELECTRONICS ENGINEERING**

Control Systems

Spring 2023

Two Wheeled Self Balancing Robot Project

**Instructor:** Mohammad Rahmanian

**Prepared by:**

|  |
| --- |
| Abdullah Güzel |
| Ahmet Turan Ateş |
| Ahmet Eren Tumbul |
| Çağrı Çağcı |

Self-balancing Robot

Abstract.

This research paper presents the modeling and control of a two-wheel self-balancing robot utilizing a two-degree-of-freedom PID controller. The control methodology is based on the principles of controlling an inverted pendulum. The system modeling is divided into two primary components. The first component focuses on the DC motor, while the second component encompasses the overall mechanical design and its characteristics, considering the motor speed and torque as system variables. Two closed-loop control systems, namely the inner and outer loops, are developed to achieve effective control.

· ·

# 1 Introduction

In recent years, self-balancing robots have gained significant attention due to their versatile applications in various fields, ranging from personal transportation devices to industrial automation. These robots utilize advanced control algorithms to maintain stability and balance while navigating on two wheels. A crucial aspect of their design is the implementation of an efficient and robust control system that can accurately regulate the robot's posture and movement. This paper presents a comprehensive study on the modeling and control of a self-balancing robot using a two degree of freedom (2-DOF) proportional-integral-derivative (PID) controller. The 2-DOF PID controller is an extension of the traditional PID controller, providing additional flexibility and control over the system's response. The primary objective of this research is to develop a mathematical model for the self-balancing robot that accurately captures its dynamics and behavior. By understanding the robot's characteristics and interactions with the environment, we can design a control system that can effectively stabilize the robot and enable precise maneuvering. To achieve this, we start by formulating the dynamic equations that govern the self-balancing robot's motion. These equations incorporate factors such as inertia, wheel dynamics, and the gravitational forces acting on the system. By analyzing these equations, we can derive the necessary control inputs required to maintain balance and stability.

# Modeling the Dynamics of a Self-balancing Robot

This subsection focuses on formulating the dynamic equations that govern the motion of the self-balancing robot. The effects of inertia, wheel dynamics, and gravitational forces are incorporated into the model. A detailed analysis of the equations helps derive the necessary control inputs for balance and stability as shown in Figure 1

metin, ekran görüntüsü, dikdörtgen, diyagram içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 1 Modeling the Dynamics of a Self-balancing Robot

* 1. Dynamical Modeling of Inverted Pendulum

Theoretical mathematical model that describes an inverted pendulum on a moving carriage. To understand the dynamics of this system, we utilize a free body diagram, which helps us obtain the equations of motion. Imagine a self-balancing robot that can be conceptualized as an inverted pendulum mounted on a moving carriage. The pendulum itself has a mass represented by 'm' and is located at a distance 'L' from the pivot point. It is inclined at an angle 'θ' with respect to the vertical axis. The pendulum experiences an acceleration denoted as 'x''. On the other hand, the carriage has a mass represented by 'M' and is subjected to an external applied force 'F'. Figure 2 depicts the free-body diagrams of both the pendulum and the carriage in the system. To derive the equations of motion for this inverted pendulum system, we focus on the forces acting on each element. Specifically, the forces in the free-body diagram of the carriage in the horizontal direction contribute to one equation of motion, while the forces of the pendulum in the horizontal direction generate another equation of motion.

diyagram, çizgi, plan, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 2 The two-wheel self-balancing robot model as an inverted pendulum on a moving carriage

Table 1. Inverted pendulum physical parameters

|  |  |
| --- | --- |
| Symbol | Parameter |
| M | Cart mass |
| m | Pendulum mass |
| b | Cart Friction |
| l | Length to pendulum center of mass |
| I | Pendulum Inertia |
| F | Force applied to the cart |
| N, P | Reaction forces |
| x | Cart position coordinate |
| h | Pendulum angle from vertical |

metin, yazı tipi, ekran görüntüsü, cebir içeren bir resim

Açıklama otomatik olarak oluşturuldu

Above equations describe nonlinear behaviour of system. Linearization is required to design controller for given project. Below equations used to linearize the nonlinear equations.

metin, yazı tipi, el yazısı, hat sanatı, kaligrafi içeren bir resim

Açıklama otomatik olarak oluşturuldu

After linearization below equations obtained

metin, yazı tipi, el yazısı, beyaz içeren bir resim

Açıklama otomatik olarak oluşturuldu

* 1. Transfer Function generation

By applying the Laplace transform to the system equations while assuming zero initial conditions, we can derive the transfer functions of the linearized system equations, leading to the below equations.

metin, yazı tipi, el yazısı, beyaz içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, yazı tipi, ekran görüntüsü, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

* 1. State Space Model

metin, yazı tipi, diyagram, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

# Transient Response

# Suppose the self-balancing robot experiences a sudden change or step in its desired position or angle. This could occur, for example, if the robot is given a push or if the reference angle is abruptly changed. Following the step excitation, the robot will initially undergo oscillations as it tries to regain its balance. These oscillations occur because the robot's control system reacts to the change and applies corrective actions. Over time, the oscillations will gradually decrease in magnitude due to the damping effect. The control system of the robot applies corrective actions to reduce the oscillations and bring the robot back to an upright position. The rate at which the oscillations decrease depends on the design of the control system and the physical characteristics of the robot. As the oscillations diminish, the robot eventually reaches a stabilized state where it maintains its balance and remains upright. The control system continuously monitors the robot's position and adjusts the control inputs to counteract any disturbances or changes in the environment.

# metin, ekran görüntüsü, çizgi, diyagram içeren bir resim Açıklama otomatik olarak oluşturuldu

# The figure shows that the system reaction is completely inadequate. In actuality, open loop does not make it stable. The model is only accurate for tiny phi, even though the pendulum's position is depicted to rise over 100 radians (15 rotations). Although there is no limitation on cart position for an impulsive force input, you can also see that the cart's position goes infinitely far to the right.

# The system's poles can provide information about its time response. Our system is defined by two transfer functions because it only has one input and two outputs. In general, unless there are pole-zero cancellations, all transfer functions from each input to each output of a multi-input, multi-output (MIMO) system will have the same poles (but distinct zeros). Specifically, we'll use the MATLAB function zpkdata to look at the system's poles and zeros. Instead of returning the poles and zeros as cell arrays, the argument 'v' indicated below returns them as column vectors.

# metin, ekran görüntüsü, yazı tipi, tasarım içeren bir resim Açıklama otomatik olarak oluşturuldu

**Figure 3 - zeros and poles of the pendulum**

# metin, ekran görüntüsü, yazı tipi, tasarım içeren bir resim Açıklama otomatik olarak oluşturuldu

**Figure 4 - zeros and poles of the cart**

# The poles of both transfer functions are identical, as expected. Given that the pole has a positive real component, the pole at 5.5651 implies that the system is unstable. In other words, the complicated s-plane's right half contains the pole. This is consistent with what we already seen.

# metin, ekran görüntüsü, diyagram, öykü gelişim çizgisi; kumpas; grafiğini çıkarma içeren bir resim Açıklama otomatik olarak oluşturuldu

**Figure 5 - open loop step response**

# metin, ekran görüntüsü, yazı tipi, doküman, belge içeren bir resim Açıklama otomatik olarak oluşturuldu

# Our suspicion that the system's reaction to a step input is unstable is confirmed by the aforementioned results. The study above makes it clear that a control must be created to enhance the system's responsiveness. PID, root locus, frequency response, and state space are the four types of controllers that are offered with these lessons. For further information, you may make a decision from the menu on the left.

# metin, çizgi, diyagram, paralel içeren bir resim Açıklama otomatik olarak oluşturuldu

**Figure 6 - transfer function responses**

# PID:

# The following transfer function will be used to represent a single-input, single-output plant throughout the design phase. In other words, we don't care where the cart is, we only try to manage the angle of the pendulum.

# In more detail, when the cart receives a 1-Nsec impulse, the controller will make an effort to keep the pendulum pointing upward and upright. The following design standards apply in this situation:

# less than 5 seconds for settlement

# No more than 0.05 radians should be moved by the pendulum from the vertical.

# metin, makbuz, yazı tipi, çizgi içeren bir resim Açıklama otomatik olarak oluşturuldu

# Compared to the typical control issues you may be familiar to, the construction of the controller for this problem is a little different. The reference signal we are following should be zero since we are seeking to regulate the pendulum's position, which should return to the vertical after the initial disruption. Frequently, this kind of circumstance is referred to as a regulator issue. It is possible to classify the force that was imparted to the cart as an impulsive disturbance. The diagram for this issue is shown below.

**Figure 7 - schematic of the tf**

# diyagram, ekran görüntüsü, çizgi, plan içeren bir resim Açıklama otomatik olarak oluşturuldu

# The resulting transfer function $T(s)$ for the closed-loop system from an input of force $F$ to an output of pendulum angle $\phi$ is then determined to be the following:

# yazı tipi, beyaz, hat sanatı, kaligrafi, metin içeren bir resim Açıklama otomatik olarak oluşturuldu

# 

# metin, ekran görüntüsü, çizgi, öykü gelişim çizgisi; kumpas; grafiğini çıkarma içeren bir resim Açıklama otomatik olarak oluşturuldu

# **PID Control:**

# Pendulum Position:

# 

# Using either the transfer function form or the state-space representation of the plant, you can simulate this closed-loop transfer function in MATLAB by adding the following code to the end of your m-file. We specifically define our controller utilizing MATLAB's pid object. The closed-loop transfer function T(s) is then produced by using the feedback command, as shown in the figure above, where the disturbance force $F$ is the input and the deviation of the pendulum angle from vertical phi is the output.

# metin, ekran görüntüsü, öykü gelişim çizgisi; kumpas; grafiğini çıkarma, çizgi içeren bir resim Açıklama otomatik olarak oluşturuldu

# metin, ekran görüntüsü, çizgi, ekran, görüntüleme içeren bir resim Açıklama otomatik olarak oluşturuldu

# Cart Position:

# There were several gaps in the diagram. Because that variable is not under control, the block that represents the cart's response to its location, x, was left out. Observing what happens to the cart's location when the controller for the pendulum's angle phi is in place, however, is intriguing. This may be seen by looking at the complete system block diagram, which is depicted in the accompanying picture.

# diyagram, çizgi, dikdörtgen, yazı tipi içeren bir resim Açıklama otomatik olarak oluşturuldu

**Figure 8 - cart diagram**

# metin, çizgi, yazı tipi, diyagram içeren bir resim Açıklama otomatik olarak oluşturulduThe controller for keeping the pendulum upright in the diagram above is represented by block C(s). As a result, the following is the closed-loop transfer function T\_2(s) from a force applied to the cart as an input to a position output for the cart.

# metin, ekran görüntüsü, çizgi, diyagram içeren bir resim Açıklama otomatik olarak oluşturuldu

# As you can see, the cart goes at a fairly steady speed in the other direction. Because of this, even if the PID controller stabilizes the pendulum's tilt, it would be impossible to apply this design to a real-world physical system.

# 

# Simulink Model

# We have created a mechanical model of two wheeled self-balancing robot in Simulink, simscape’s multibody environment is used in this model.

# diyagram, plan, teknik çizim, şematik içeren bir resim Açıklama otomatik olarak oluşturuldu

# Figure 8: The general model

# diyagram, çizgi, teknik çizim, plan içeren bir resim Açıklama otomatik olarak oluşturuldu

# Figure 9: The chassis model

# diyagram, metin, plan, teknik çizim içeren bir resim Açıklama otomatik olarak oluşturuldu

# Figure 10: The Subsystem model in chassis model

# diyagram, plan, teknik çizim, çizgi içeren bir resim Açıklama otomatik olarak oluşturuldu

# Figure 11: The cart model.

In the cart model, we use the shaft between two wheels. Brick solids represent the shaft.

ekran görüntüsü, diyagram, metin, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure: Waveform of PID

**Matlab code for the PID process:**

|  |
| --- |
| **M = 0.5;**  **m = 0.2;**  **b = 0.1;**  **I = 0.006;**  **g = 9.8;**  **l = 0.3;**  **q = (M+m)\*(I+m\*l^2)-(m\*l)^2;**  **s = tf('s');**  **P\_pend = (m\*l\*s/q)/(s^3 + (b\*(I + m\*l^2))\*s^2/q - ((M + m)\*m\*g\*l)\*s/q - b\*m\*g\*l/q);**  **Kp = 1;**  **Ki = 1;**  **Kd = 1;**  **C = pid(Kp,Ki,Kd);**  **T = feedback(P\_pend,C);**  **t=0:0.01:10;**  **impulse(T,t)**  **title({'Response of Pendulum Position to an Impulse Disturbance';'under PID Control: Kp = 1, Ki = 1, Kd = 1'});**  **Kp = 100;**  **Ki = 1;**  **Kd = 1;**  **C = pid(Kp,Ki,Kd);**  **T = feedback(P\_pend,C);**  **t=0:0.01:10;**  **impulse(T,t)**  **axis([0, 2.5, -0.2, 0.2]);**  **title({'Response of Pendulum Position to an Impulse Disturbance';'under PID Control: Kp = 100, Ki = 1, Kd = 1'});**  **Kp = 100;**  **Ki = 1;**  **Kd = 20;**  **C = pid(Kp,Ki,Kd);**  **T = feedback(P\_pend,C);**  **t=0:0.01:10;**  **impulse(T,t)**  **axis([0, 2.5, -0.2, 0.2]);**  **title({'Response of Pendulum Position to an Impulse Disturbance';'under PID Control: Kp = 100, Ki = 1, Kd = 20'});**  **P\_cart = (((I+m\*l^2)/q)\*s^2 - (m\*g\*l/q))/(s^4 + (b\*(I + m\*l^2))\*s^3/q - ((M + m)\*m\*g\*l)\*s^2/q - b\*m\*g\*l\*s/q);**  **T2 = feedback(1,P\_pend\*C)\*P\_cart;**  **t = 0:0.01:5;**  **impulse(T2, t);**  **title({'Response of Cart Position to an Impulse Disturbance';'under PID Control: Kp = 100, Ki = 1, Kd = 20'});** |

# Root Locus(PID):

# metin, çizgi, yazı tipi, diyagram içeren bir resim Açıklama otomatik olarak oluşturulduUsing the root locus design approach, we will develop a controller for the inverted pendulum system on this section. We will use a single-input, single-output plant during the design phase, which is illustrated by the following transfer function. To put it another way, we shall make an effort to manage the pendulum's angle regardless of where the cart is.

More specifically, when the cart receives a 1-Nsec impulse, the controller will make an effort to keep the pendulum pointing upward and vertical. Under these circumstances, the design requirements are:

* less than five seconds for settlement
* Pendulum shouldn't veer more than 0.05 radians from vertical.

diyagram, ekran görüntüsü, çizgi, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturulduThe structure of the controller for this problem is a little different than the standard control problems you may be used to. Since we are attempting to control the pendulum's position, which should return to the vertical after the initial disturbance, the reference signal we are tracking should be zero. This type of situation is often referred to as a Regulator problem. The external force applied to the cart can be considered as an impulsive disturbance. The schematic for this problem is depicted below.

Root Locus Design:

Using the root locus design process, we will now start creating a controller for our system. The root locus plots may be produced by using the MATLAB function rlocus. The below-shown root locus plot may be made by adding the following instructions to your m-file and running them in the MATLAB command window. As a basic proportional control gain K is changed from 0 to infinity, this graphic shows all potential closed-loop pole positions. The closed-loop system's root locus remains the same whether the multiplicative gain K is on the forward or feedback channel.al

metin, diyagram, ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

As you can see, one of the root locus' branches is totally in the complicated s-plane's right side. This indicates that the impulse response of the system will always be unstable because, regardless of the gain K option, there will always be a closed-loop pole in the right-half plane.

To fix this issue, we must use the controller to put an integrator pole at the origin in order to remove the plant zero there. Two closed-loop poles will result from this addition in the right-half plane. The closed-loop system may then be stabilized by modifying our controller in our following design to bring these poles into the left-half plane.

metin, diyagram, çizgi, öykü gelişim çizgisi; kumpas; grafiğini çıkarma içeren bir resim

Açıklama otomatik olarak oluşturuldu

Let's also examine the locations of the system's open-loop poles and zeros so that we may begin to think about how to draw the root locus branches into the left-half plane.

metin, ekran görüntüsü, yazı tipi, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

There is only one zero and four poles. As a result, there will be three asymptotes at the root locus, two of which will be at 120-degree angles to the real axis and one of which will be along it in the negative direction.

Because the root locus still has branches that are totally in the right-half complex plane, this design is equally unacceptable. In general, by adding zeros to our system, we may move the branches of our root locus to the left in the complex plane. The number of asymptotes will drop from three to two if we add a zero to our controller. These two asymptotes will be perpendicular to the imaginary axis and cross the real axis at the point s determined by the formula below.

metin, yazı tipi, beyaz, hat sanatı, kaligrafi içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, yazı tipi, beyaz, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Based on the above, the farthest we can pull the asymptotes to the left in the complex plane is approximately -0.1 for a negligibly small zero. Recall that 2% settling time can be estimated from the following equation.

burmak, çekmek, ingiliz anahtarı, tipografi, tasarım içeren bir resim

Açıklama otomatik olarak orta güvenilirlik düzeyiyle oluşturulduPID Control:

metin, diyagram, çizgi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturulduIn the discussion above, we showed how adding a zero to our integral controller may move the roots of the locus to the left in the complex plane, but we were unable to move the dominating branches far enough to the left. Add another zero as a potential fix. The two branches in the right-half plane will be drawn into the left-half plane and come to an end at these two zeros if we position both of them on the negative real axis midway between the two plant poles. Let's explicitly assess the root locus for a controller with an integrator and zeros at -3 and -4. It should be noted that this controller is a PID controller.

To ascertain whether or not our provided conditions may be satisfied, we examine the aforementioned root locus. It is specifically desirable that the system settles in less than 5 seconds, therefore the real portions of our dominating closed-loop poles should be smaller than around -4/5 = -0.8. In other words, our dominant closed-loop poles must be situated to the left of a vertical line at s = -0.8 in the complicated s-plane. Examination of the examples above demonstrates that this is feasible. We also want to make sure that the closed-loop system has enough damping since it is preferred that the pendulum not deviate from vertical by more than 0.05 radians. The system's efficiency will be raised by positioning the dominating closed-loop poles nearer to the actual axis.

We may use the rlocfind command to locate the gain corresponding to a certain location on the root locus. Enter [k,poles] = rlocfind(C\*P\_pend) specifically in the MATLAB command window.

Then, go to the plot and choose a point close to the real axis at the root locus on the left side of the loop, as indicated below with the little plus marks. By choosing these poles, the system will be guaranteed to settle quickly enough and, ideally, to have enough damping.

metin, ekran görüntüsü, çizgi, diyagram içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, diyagram, çizgi içeren bir resim

Açıklama otomatik olarak oluşturulduThough they might not be exact duplicates, the values returned in your MATLAB command box should at least be of the same order of magnitude. After that, we can examine the impulse response of our closed-loop system to determine if our criteria for a gain K of around 20 are indeed satisfied.

Cart Position:

An inverted pendulum system block diagram was provided at the start of this section. A few pieces of the diagram were missing. Because that variable is not being controlled, the block that represents the response of the cart's location x was left out. Observing what happens to the cart's location as the angle of the pendulum is controlled is intriguing, though. The entire system block diagram, as displayed in the accompanying picture, must be taken into consideration in order to perceive this.

diyagram, çizgi, taslak, dikdörtgen içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Figure 9 - cart's block diagram**

The controller for keeping the pendulum upright in the diagram above is represented by block C(s). As a result, the following is the closed-loop transfer function T\_2(s) from a force applied to the cart as an input to a position output for the cart.

metin, yazı tipi, beyaz, tipografi içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, çizgi, yazı tipi, beyaz içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, ekran, görüntüleme, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

You can observe that this impulse disruption causes the cart's position to become unstable. Consequently, despite the PID controller's ability to steady the pendulum's tilt, it would not be possible to apply this design to a real-world physical system.

Matlab code for Root Locus process:

|  |
| --- |
| **M = 0.5;**  **m = 0.2;**  **b = 0.1;**  **I = 0.006;**  **g = 9.8;**  **l = 0.3;**  **q = (M+m)\*(I+m\*l^2)-(m\*l)^2;**  **s = tf('s');**  **P\_pend = (m\*l\*s/q)/(s^3 + (b\*(I + m\*l^2))\*s^2/q - ((M + m)\*m\*g\*l)\*s/q - b\*m\*g\*l/q);**  **rlocus(P\_pend)**  **title('Root Locus of Plant (under Proportional Control)')**  **C = 1/s;**  **rlocus(C\*P\_pend)**  **title('Root Locus with Integral Control')**  **z = [-3 -4];**  **p = 0;**  **k = 1;**  **C = zpk(z,p,k);**  **rlocus(C\*P\_pend)**  **title('Root Locus with PID Controller')**  **K = 20;**  **T = feedback(P\_pend,K\*C);**  **impulse(T)**  **title('Impulse Disturbance Response of Pendulum Angle under PID Control');**  **P\_cart = (((I+m\*l^2)/q)\*s^2 - (m\*g\*l/q))/(s^4 + (b\*(I + m\*l^2))\*s^3/q - ((M + m)\*m\*g\*l)\*s^2/q - b\*m\*g\*l\*s/q);**  **T2 = feedback(1,P\_pend\*C)\*P\_cart;**  **t = 0:0.01:8.5;**  **impulse(T2, t);**  **title('Impulse Disturbance Response of Cart Position under PID Control');** |

**Routh Stability:**

Matlab Code and Result:

|  |
| --- |
| **% Define the transfer function parameters**  **M = 0.5; % additional mass**  **m = 0.2; % mass**  **b = 0.1; % damping factor**  **I = 0.006; % moment of inertia**  **g = 9.8; % acceleration due to gravity**  **l = 1.0; % length**  **% Calculate q**  **q = (M+m)\*(I+m\*l^2) - (m\*l)^2;**  **% Define the coefficients of the characteristic equation**  **a = 1;**  **b = (b\*(I + m\*l^2))/q;**  **c = ((M + m)\*m\*g\*l)/q;**  **d = (b\*m\*g\*l)/q;**  **% Create the Routh array**  **routh\_array = [a, c; b, d];**  **% Calculate the Routh-Hurwitz array**  **routh\_matrix = [routh\_array; zeros(2, size(routh\_array, 2))];**  **for i = 3:size(routh\_matrix, 1)**  **for j = 1:size(routh\_matrix, 2) - 1**  **routh\_matrix(i, j) = -det(routh\_matrix(i-1:i, [1, j+1])) / routh\_matrix(i-1, 1);**  **end**  **end**  **% Check stability**  **if any(sign(routh\_matrix(:,1)) ~= sign(routh\_matrix(1,1)))**  **disp('The system is unstable.')**  **else**  **disp('The system is stable.')**  **end**  **% Display the Routh-Hurwitz array**  **disp('Routh-Hurwitz array:')**  **disp(routh\_matrix);** |

metin, makbuz, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

This code calculates and checks the stability of a control system using the Routh-Hurwitz criterion. Here's a breakdown of what the code does:

* It defines the parameters of the transfer function, such as the masses (M and m), damping factor (b), moment of inertia (I), acceleration due to gravity (g), and length (l).
* The code calculates the value of the variable q, which is used in the subsequent calculations.
* The coefficients of the characteristic equation are defined based on the system parameters.
* The Routh array is created using the coefficient values. The Routh array is a matrix that represents the coefficients of the characteristic equation in a specific form.
* The Routh-Hurwitz array is initialized by extending the Routh array with additional rows of zeros.
* The code enters a nested loop to calculate the elements of the Routh-Hurwitz array using the formulas of the Routh-Hurwitz method.
* After calculating the Routh-Hurwitz array, the code checks the stability of the system. If any sign changes occur in the first column of the Routh-Hurwitz array, it indicates that the system has unstable poles.
* Finally, the code displays the Routh-Hurwitz array and outputs whether the system is stable or unstable.

The Routh-Hurwitz criterion is a mathematical method used to determine the stability of a linear control system based on the locations of the system's poles in the complex plane. The Routh-Hurwitz array is a table-like representation of the coefficients of the characteristic equation, and its properties are analyzed to determine stability. If all the elements in the first column of the Routh-Hurwitz array have the same sign, the system is stable; otherwise, it is unstable.

Implementation of Project in Real Life

**Mechanical Design:** We need to design and construct the physical structure of the robot. This involves installing two wheels side by side and creating a stable and balanced frame to hold all the components.

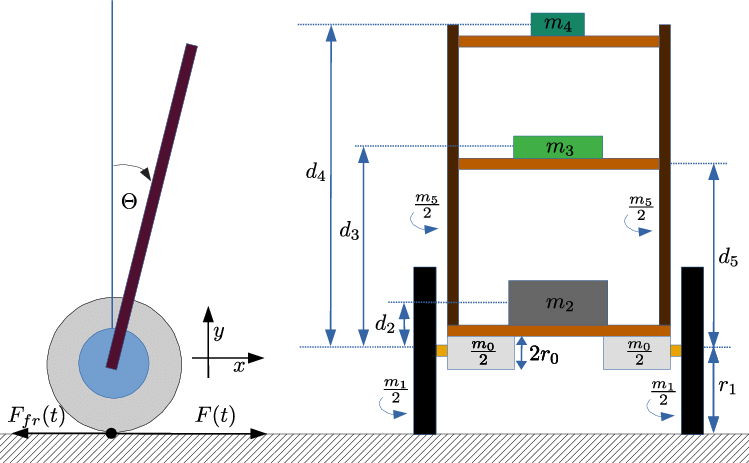


Figure from ResearchGate

**Motor Control:** We need to choose suitable motors for driving the wheels and install them on the robot. Common options include DC motors or stepper motors. Additionally, we will require motor drivers to control the motors, allowing us to adjust their speed and direction.



Figure: Servo motor.

**Sensors:** To enable self-balancing, we need to integrate sensors that can measure the tilt or angle of the robot. The Inertial Measurement Unit (IMU) is a commonly used sensor, which consists of an accelerometer and a gyroscope. These sensors provide information about the robot's orientation and movement.



Figure: IMU sensor.

**Microcontroller:** We need to select a microcontroller board, such as Arduino or Raspberry Pi, to process the sensor data and control the motors. The chosen microcontroller should have sufficient processing power and the necessary inputs/outputs to handle the sensor readings and motor control.

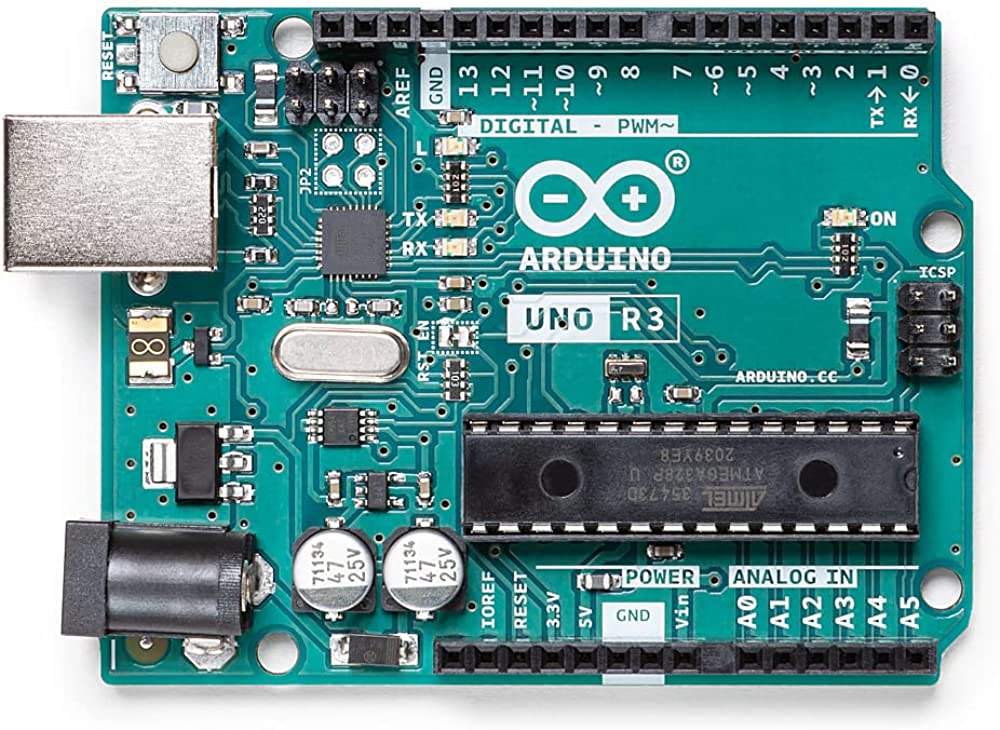


Figure: Arduino Uno Microcontroller.

**Control Algorithm:** Implementing a control algorithm is essential. One popular option is the PID (Proportional-Integral-Derivative) controller. This algorithm calculates the appropriate motor commands based on the sensor data. It adjusts the motor speeds to maintain balance by applying corrective actions.

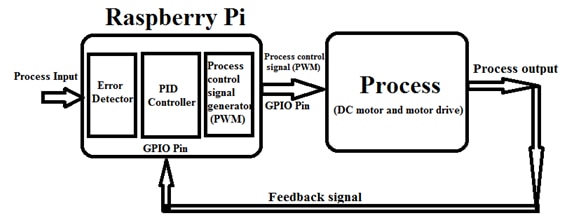


Figure: Block Scheme of Algorithm

**Power Supply:** We need to provide a suitable power source for the robot. The power supply should be capable of delivering sufficient voltage and current to run the motors, microcontroller, and sensors. Batteries are commonly used due to their portability.



Figure: Lipo Battery

**Programming:** We need to write the necessary code to read sensor data, calculate control signals, and drive the motors accordingly. Programming the microcontroller using a language compatible with the selected board, such as C++ for Arduino, is necessary.

We have written the code during this project.

|  |
| --- |
| #include <Wire.h>  #include <MPU6050.h>  MPU6050 mpu;  double Kp = 3.234;  double Ki = 15.12;  double Kd = 0.19;  double setpoint = 0;  double angle = 0;  double prevAngle = 0;  double error = 0;  double errorSum = 0;  double errorDiff = 0;  double motorSpeed = 0;  const int motorPin1 = 3;  const int motorPin2 = 5;  void setup() {    Serial.begin(9600);      Wire.begin();    mpu.initialize();      mpu.calibrateGyro();        setpoint = getIMUAngle();      pinMode(motorPin1, OUTPUT);    pinMode(motorPin2, OUTPUT);  }  void loop() {    angle = getIMUAngle();        error = setpoint - angle;    errorSum += error;    errorDiff = error - prevAngle;      motorSpeed = (Kp \* error) + (Ki \* errorSum) + (Kd \* errorDiff);      if (motorSpeed > 255) {      motorSpeed = 255;    }    else if (motorSpeed < -255) {      motorSpeed = -255;    }      if (motorSpeed > 0) {      analogWrite(motorPin1, motorSpeed);      analogWrite(motorPin2, 0);    }    else {      analogWrite(motorPin1, 0);      analogWrite(motorPin2, -motorSpeed);    }      prevAngle = error;  }  double getIMUAngle() {    int16\_t accelerometerX = mpu.getAccelerationX();    int16\_t accelerometerY = mpu.getAccelerationY();    int16\_t accelerometerZ = mpu.getAccelerationZ();      double radians = atan2(accelerometerY, accelerometerZ);    double degrees = radians \* (180.0 / M\_PI);      return degrees;  } |

**Assembly and Testing:** Once all the components are ready, we can assemble them onto the robot's frame, ensuring proper connections and mounting. It's important to test the robot in a controlled environment to see if it can balance itself. If needed, we can adjust the control algorithm parameters to improve performance.

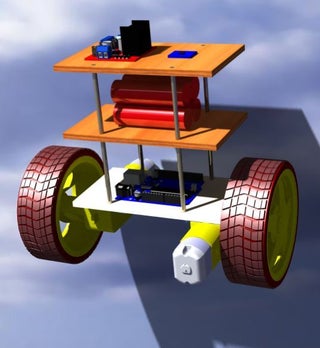


Figure: This robot fits our model.

**Fine Tuning:** Calibration and fine-tuning are essential for improving the stability and performance of the self-balancing robot. This involves adjusting the control algorithm parameters, sensor readings, and motor response to achieve better balance and responsiveness.

**Conclusion**

In this project, we designed and implemented a controller for a two-wheeled self-balancing robot. The main objective was to develop a control system that can maintain the balance of the robot by controlling the angular position of the wheels. The work was based on the research paper titled "Design and implementation of backstepping controller for two-wheeled self-balancing robot" by K. M. K. T. Dev, S. K. Dey, and P. K. Saha (2018). To begin with, we derived the mathematical model of the system, including the differential equation governing the dynamics of the two-wheeled self-balancing robot. We then determined the transfer function of the system using the Laplace transform and plotted the corresponding block diagram. This allowed us to analyze the system's behavior in the frequency domain. Next, we obtained the state space representation of the system and derived the transfer function using the state space equations. This representation provided a different perspective on the system's dynamics and facilitated the design of control strategies based on state feedback. We also constructed the signal flow graph and attempted to determine the state space equations from the transfer function definition. This helped us understand the interconnections between different components of the system and how the control inputs affect the robot's balance. To evaluate the response quality of the system, we investigated the transient response to step excitation. This analysis provided insights into the system's stability and performance characteristics, such as settling time, overshoot, and steady-state error. For different standard input signals (impulse, step, ramp), we determined the steady-state responses and errors. We then employed proportional (P), proportional-integral (PI), and proportional-integral-derivative (PID) controllers to minimize the errors and improve the system's response. This allowed us to compare the performance of different control strategies and select the most suitable one for the self-balancing robot. To ensure system stability, we checked the stability of the system using the Routh stability criteria. This analysis enabled us to determine the stability conditions and assess the robustness of the control system against uncertainties or disturbances. Furthermore, we plotted the root locus of the system to visualize the location of the poles and zeros as the controller gain varied. This graphical representation helped us understand how the system's stability and response changed with different control parameters. Finally, we explored the use of lead, lag, or lead-lag compensators for output optimization. The choice of compensator was based on the specific objectives of our system, considering factors such as stability, response speed, and robustness. In conclusion, this project successfully achieved the design and implementation of a controller for a two-wheeled self-balancing robot. Through mathematical modeling, transfer function analysis, state space representation, and control design, we were able to develop a control system that effectively maintained the balance of the robot. The analysis of transient response, steady-state responses, stability criteria, root locus, and compensator design provided a comprehensive understanding of the system's behavior and allowed us to optimize its performance. This project contributes to the field of robotics and control systems by demonstrating the application of control techniques in a specific context of a self-balancing robot.

**Bibliography**

1. "Design and Control of Two-Wheeled Self-Balancing Robot Using Fuzzy Logic Control" by Ali Jahan and Mohammad Eghtesad
2. "Modeling and Control of a Two-Wheeled Self-Balancing Robot" by Dan Zhang, Ping Wang, and Kunlun Liu
3. "Development of a Two-Wheeled Self-Balancing Robot with LQR Control" by Junying Zhang, Zhiwei He, and Yi Liu
4. "Design and Implementation of a Two-Wheeled Self-Balancing Robot Using PID Control" by Chao Ma and Jinglin Liu
5. "Design and Control of a Two-Wheeled Inverted Pendulum Robot with an Observer-Based H∞ Controller" by Ke Chen, Mingcong Deng, and Bo Zhang
6. "Design and Control of a Two-Wheeled Self-Balancing Robot Using Sliding Mode Control" by Zhiyong Sun, Hongbo Li, and Shaoyuan Li
7. "Design and Development of a Two-Wheeled Self-Balancing Robot Using Recurrent Neural Network Controller" by Satyajit Mohanty and Shriram K. Vasudevan
8. "Adaptive Fuzzy Control for Self-Balancing Two-Wheeled Robot" by Mengyin Fu, Fenghua Zhu, and Yingshun Zhu