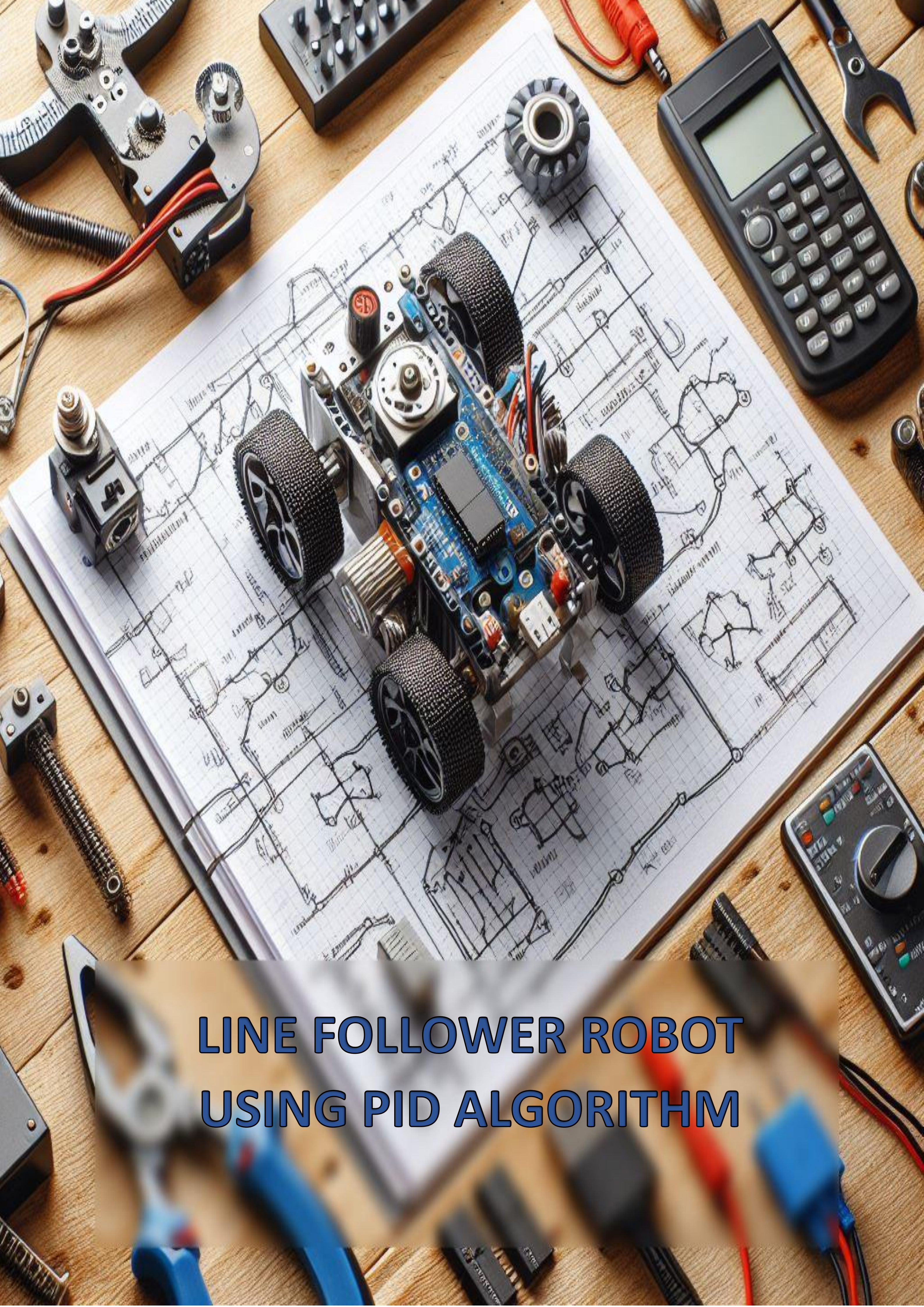
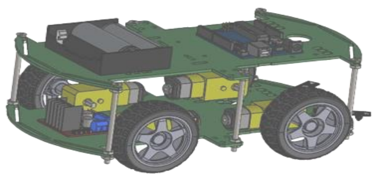


LINE FOLLOWER ROBOT USING PID ALGORITHM

Under Control

Team 1

18011819	مصطفى يسري محمد عبدالرءوف
18010872	طارق حسام السيد
19017494	سيف الاسلام ابراهيم حسن محمد
19016255	محمد ابراهيم ابواليزيد محمود
19016895	يمنى زكريا محمود عبدالشافي
18011727	مروان عبدالمنعم السيد
20010952	علي محمود محمد كمال عطيه
19016675	مصطفى محمد محمد خليل
19017491	محمد توحيد محمد نديم
19017050	عائشة ناجد عبد السلام غريب



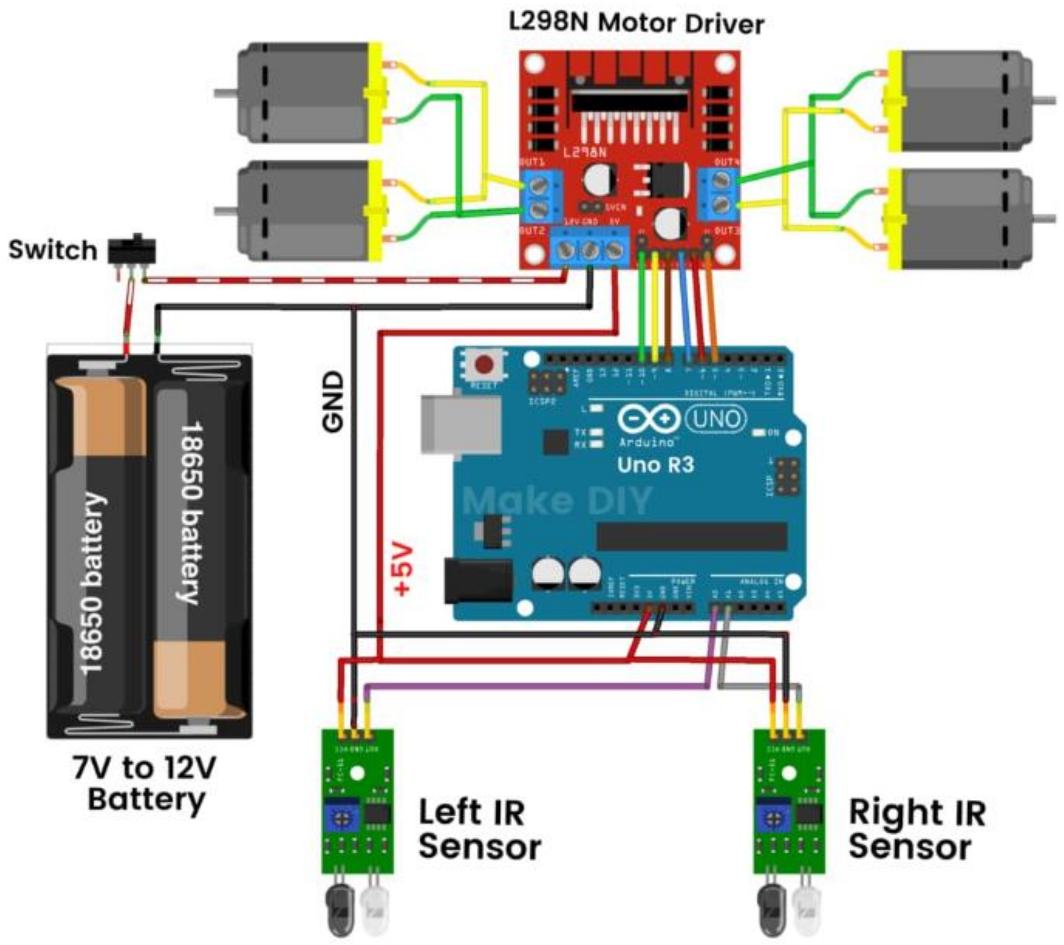
Contents

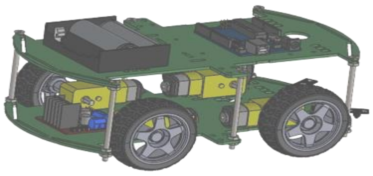
Introduction

The Line Follower Robot Car is a fascinating example of how technology can mimic human behavior to perform tasks autonomously. Operating on the principles of reflective light sensing and microcontroller programming, this simple yet ingenious device can navigate predefined paths with remarkable accuracy. At its core, the Line Follower Robot Car employs infrared (IR) sensors to detect changes in surface reflectivity, enabling it to follow lines drawn on a contrasting background.

Main Components:

1) Arduino UNO
2) IR Sensors
3) L298N Motor Driver Module
4) Battery 7~12V
5) DC Gearbox Motor

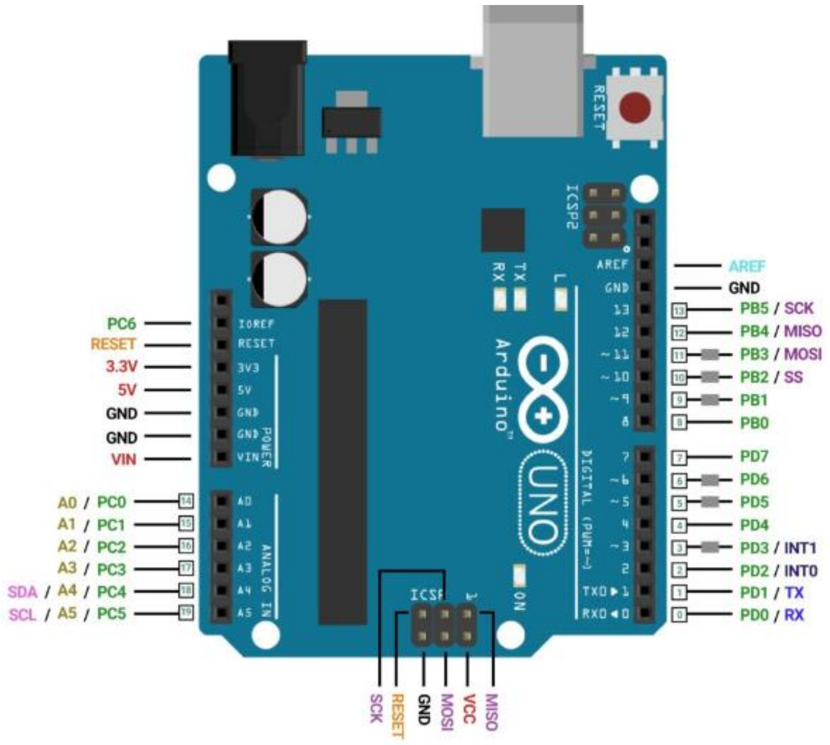




Components

1) Arduino UNO:

Arduino Uno is a popular microcontroller board used in various electronics projects, prototyping, and DIY applications. It's a key component in the maker and hobbyist community due to its versatility, ease of use, and extensive support resources.



Microcontroller: At the heart of Arduino Uno is a microcontroller chip, typically an Atmel ATmega328P or similar model. This microcontroller acts as the brain of the board, executing instructions and controlling the interactions between various components connected to it.

Input/Output (I/O) Pins: Arduino Uno features a set of digital and analog input/output pins that can be used to connect sensors, actuators, displays, and other electronic components. These pins allow the microcontroller to interact with the external world by reading inputs from sensors (such as temperature sensors, motion sensors, or buttons) and sending outputs to control actuators (such as motors, LEDs, or displays).

Power Supply: Arduino Uno can be powered via a USB connection, which also allows for programming the board, or through an external power source such as a battery or a DC adapter. It typically operates at 5 volts, although some pins support both 3.3V and 5V operation.

USB Interface: Arduino Uno includes a built-in USB interface that facilitates communication between the board and a computer. This USB connection is used for

programming the microcontroller and uploading code (sketches) to the board. It also enables serial communication for debugging and exchanging data between the board and external devices.

Reset Button: Arduino Uno features a reset button that allows you to restart the microcontroller, restarting the execution of the uploaded code from the beginning.

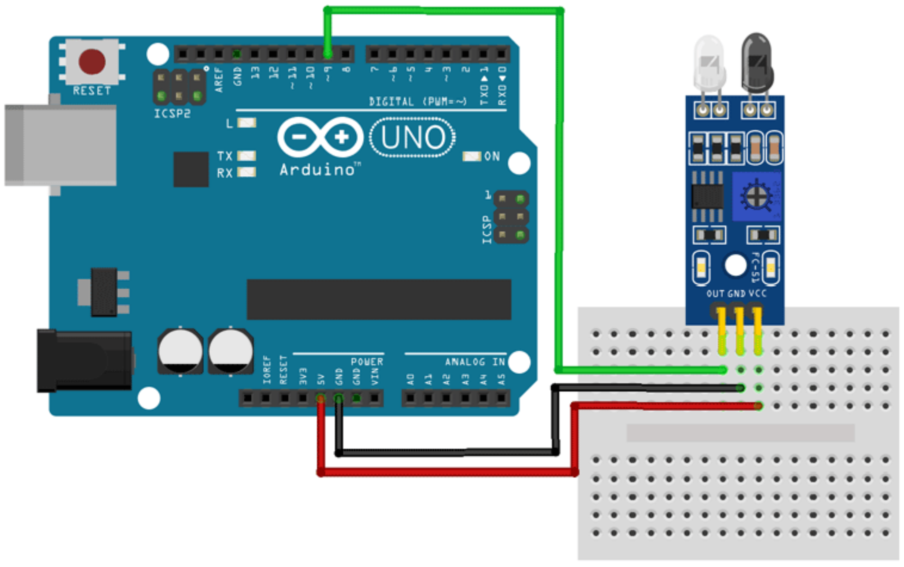
Voltage Regulator: Arduino Uno incorporates a voltage regulator that ensures a stable power supply to the microcontroller and other components connected to the board. This regulator allows Arduino Uno to accept a wide range of input voltages (typically 7-12 volts) and provide a consistent output voltage (5 volts) to the microcontroller and connected devices.

Compatibility: One of the significant advantages of Arduino Uno is its compatibility with the Arduino Integrated Development Environment (IDE), a user-friendly software tool used for writing, compiling, and uploading code to the board. Arduino Uno supports a vast library of pre-written code (known as sketches) and numerous tutorials and examples, making it accessible even to beginners.

2) IR Sensors:

The Line Follower Robot Car is equipped with two IR sensors positioned underneath its chassis. These sensors are designed to detect changes in surface reflectivity, distinguishing between reflective (typically white) surfaces and non-reflective (typically black) lines.

When the sensor detects a reflective surface, such as white, it sends back a signal indicating high reflectivity. Conversely, when it encounters a non-reflective surface like a black line, it doesn't reflect any light, resulting in a low or no signal output.

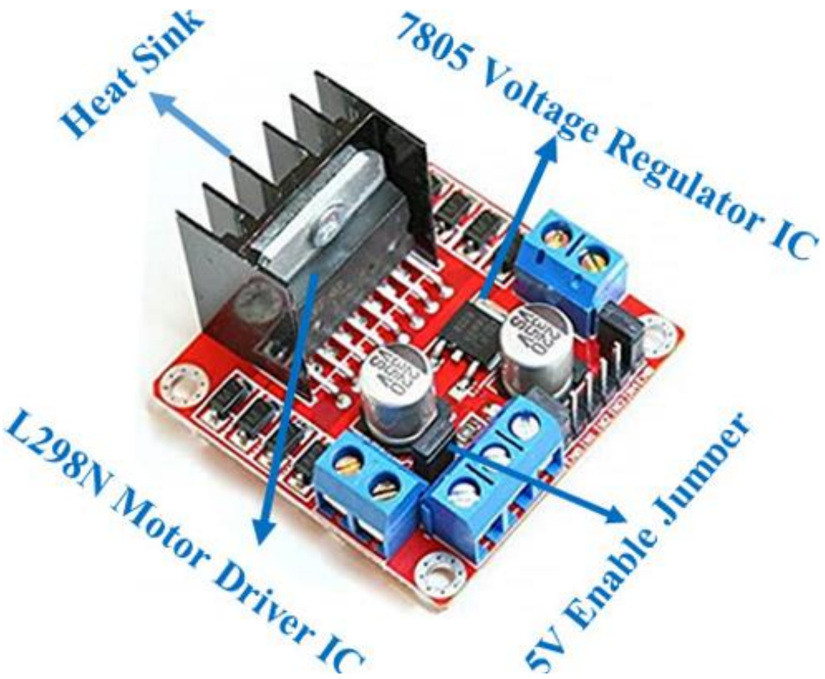


Connecting the IR sensor to any microcontroller is really simple. As we know this sensor outputs a digital signal and processing this signal is very easy. There exist two methods to do so first, you can always check the port in an infinite loop to see when the port changes its state from high to low, or the other way is to do it with an interrupt if you are making a complicated project the interrupt method is recommended. Power the IR with 5V or 3.3V and connect ground to ground. Then connect the output to a digital pin D9. We have just used a Male to Female Jumper wire to connect the IR sensor module with Arduino board.

3) L298N Motor Driver Module:

This L298N Motor Driver Module is a high-power motor driver module for driving DC and Stepper Motors. This module consists of an L298 motor driver IC and a 78M05 5V regulator. L298N Module can control up to 4 DC motors, or 2 DC motors with directional and speed control.

This module consists of an L298 Motor Driver IC, 78M05 Voltage Regulator, resistors, capacitor, Power LED, 5V jumper in an integrated circuit.



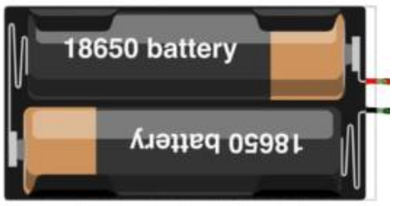
78M05 Voltage regulator will be enabled only when the jumper is placed. When the power supply is less than or equal to 12V, then the internal circuitry will be powered by the voltage regulator and the 5V pin can be used as an output pin to power the microcontroller. The jumper should not be placed when the power supply is greater than 12V and separate 5V should be given through 5V terminal to power the internal circuitry.

ENA & ENB pins are speed control pins for Motor A and Motor B while IN1& IN2 and IN3 & IN4 are direction control pins for Motor A and Motor B.

4) Battery 7~12V

An 18650 battery is a type of lithium-ion rechargeable battery. The numbers "18650" refer to the battery's dimensions: it is 18mm in diameter and 65mm in length. 18650 batteries are commonly used in electronic devices such as laptops and flashlights, as well as in electric vehicles and other high-power applications. They are known for their high energy density, long lifespan, and relatively low self-discharge rate.

Some types of 18650 have been modified adding either a button top and/or internal protection circuit. This can increase the physical length of an "18650" battery from 65mm to 70mm or in certain cases even longer.

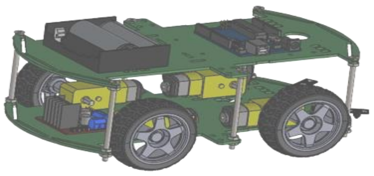


5) DC Gearbox Motor

DC Gearbox Motor with a gear ratio of 1:48. Perfect for plugging into a breadboard or terminal blocks. You can power these motors with 3VDC up to 12VDC, they'll of course go a little faster at the higher voltages.

Motor specifications:

- Power supply voltage: 3-12 V.
- Power consumption: 190 mA (max. 250 mA).
- Gearbox: 1:48.
- Speed: 600 ± 10 rpm (at 12V).
- Shaft diameter: 5.4 mm.
- The shaft is cut on both sides.
- The motor has a double shaft.



Controller

In control systems, a controller is a mechanism that seeks to minimize the difference between the actual value of a system (i.e. the process variable) and the desired value of the system (i.e. the setpoint). Controllers are a fundamental part of control engineering and used in all complex control systems.

The important uses of the controllers include:

- Controllers improve the steady-state accuracy by decreasing the steady state error.
- As the steady-state accuracy improves, the stability also improves.
- Controllers also help in reducing the unwanted offsets produced by the system.
- Controllers can control the maximum overshoot of the system.
- Controllers can help in reducing the noise signals produced by the system.
- Controllers can help to speed up the slow response of an overdamped system.

Types of Controllers:

The types of controllers are as follows:

- Proportional Controller (P-Controller)
- Derivative Controller (D-Controller)
- Integral Controller (I-Controller)

We use the combination of these modes to control our system such that the process variable is equal to the setpoint (or as close as we can get it). These three types of controllers can be combined into new controllers:

- Proportional and integral controllers (PI Controller)
- Proportional and derivative controllers (PD Controller)
- Proportional integral derivative control (PID Controller)

Proportional Controller (P-Controller)

The proportional controller adjusts the control output in direct proportion to error signal in which is the difference between desired setpoint and the actual process variable. The Proportional Controller produces an output that is proportional to error signal.

- ➢ The control output (u(t)) is calculated as u(t) = KP * e(t)
  where KP is the proportional constant.

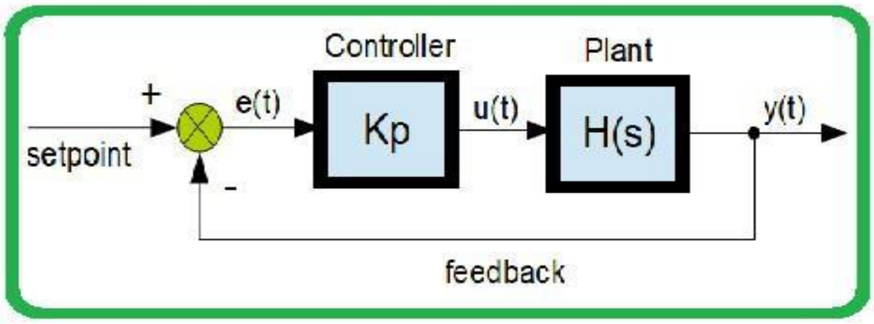
The Proportional Controller aims to reduce the error and bring the system closer to setpoint. It is effective in reducing steady-state error but may lead to oscillations and overshoot in response.

Advantages

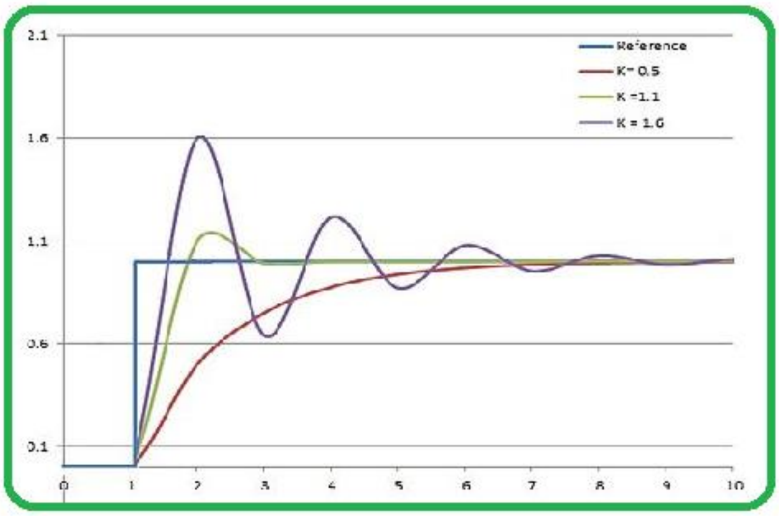
- Simple and easy to implement.
- Provides fast response to errors.
- Reduces steady-state error.

Disadvantages

- Cannot eliminate steady-state error entirely.
- May lead to oscillations or instability in system if not tuned correctly.



The P-controller's block diagram features a direct connection from input to the controller in which then directly influences the output. The P-controller multiplies the error signal by constant proportional gain (Kp). The resulting control signal is added to system input to correct the error.



The P-controller reduces the steady-state error but introduces the oscillations and overshoot. It cannot eliminate all error.

Derivative Controller (D-Controller)

The derivative controller reacts to rate of change of the error signal. It anticipates future error trends and provides control action to the counteract them.

The Derivative Controller produces an output that is derivative of the error signal with the respect to time.

- The control output is calculated as u(t) = KD * (de(t)/dt)
  where, KD is the derivative constant.

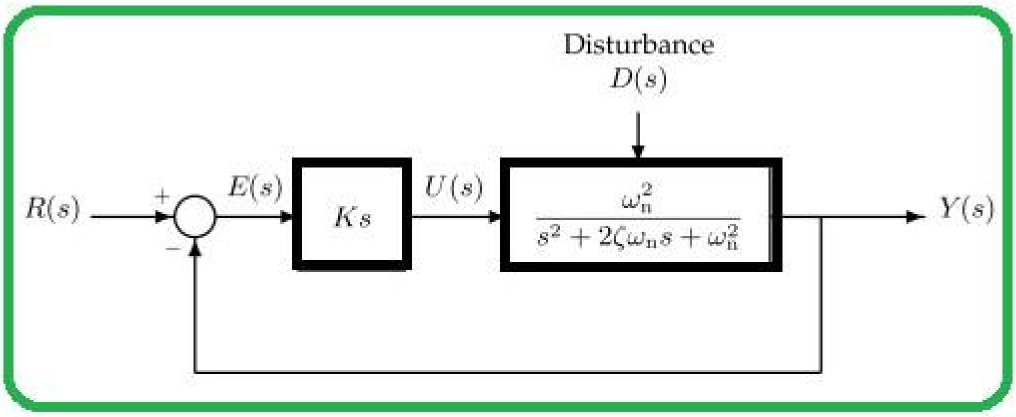
The Derivative Controller helps in the damping oscillations and improving system stability. It is anticipating future errors based on the rate of change of error.

Advantages

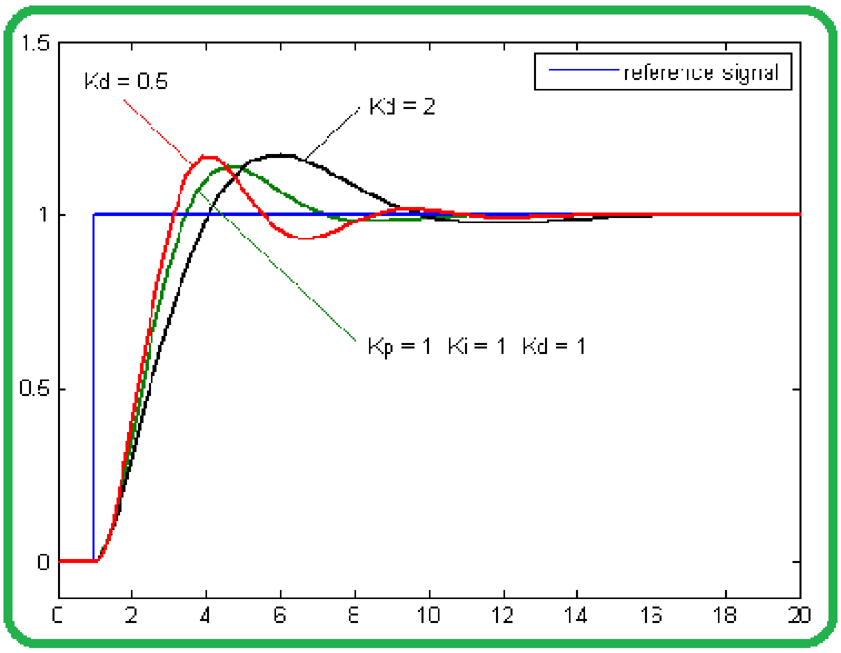
- Provides rapid response to the changing errors.
- To Reduces overshoot and oscillations.

Disadvantages

- Amplifies noise and measurement errors.
- Can be sensitive to the parameter variations.



The D-controller's block diagram features a differentiation block between input and the controller. The D-controller calculates the rate of change of error signal (derivative) and multiplies it by a constant derivative gain (Kd). This derivative term is added to control signal and helping to reduce overshoot and dampen oscillations.



The D-controller improves system stability and transient response reducing overshoot and oscillations.

Integral Controller (I-Controller)

The integral controller responds to cumulative sum of past errors. It continuously adjusts the control output to eliminate any steady-state error.

The Integral Controller produces an output that is the integral of the error signal with respect to time.

- The control output is calculated as u(t) = KI * ∫e(t)dt
  where, KI is the integral constant.

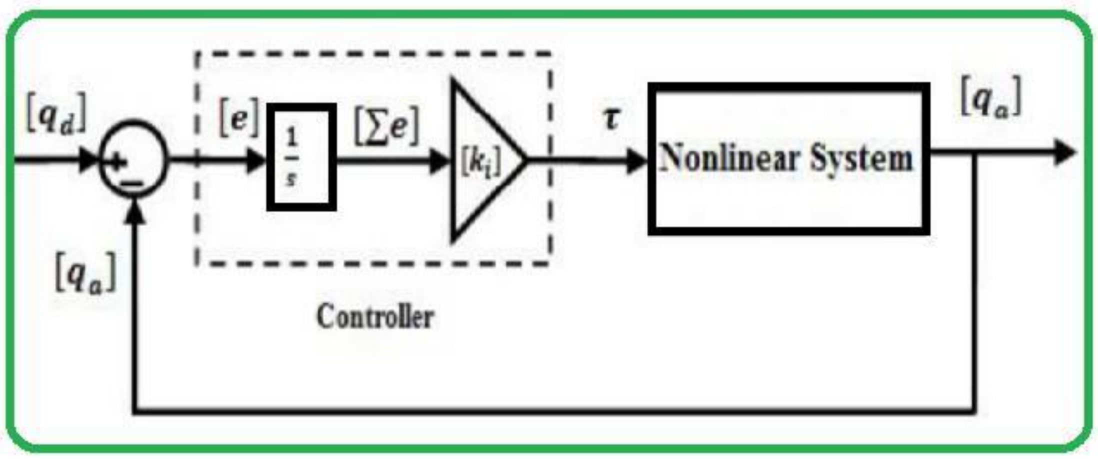
The Integral Controller helps in the eliminating steady-state error by continuously integrating past errors. It ensures that the system reaches and maintains setpoint over time.

Advantages

- Eliminates steady-state error completely.
- Increases system accuracy.

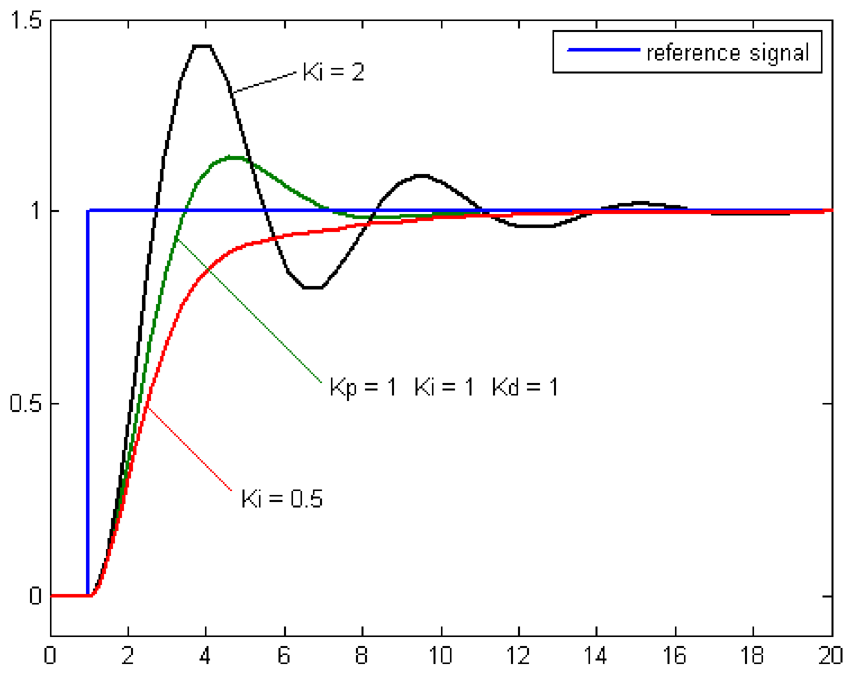
Disadvantages

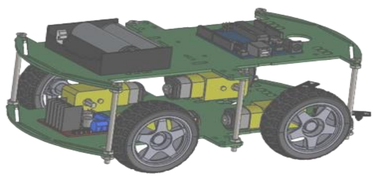
- Slower response to sudden changes in setpoint.
- lead to instability if excessively tuned.



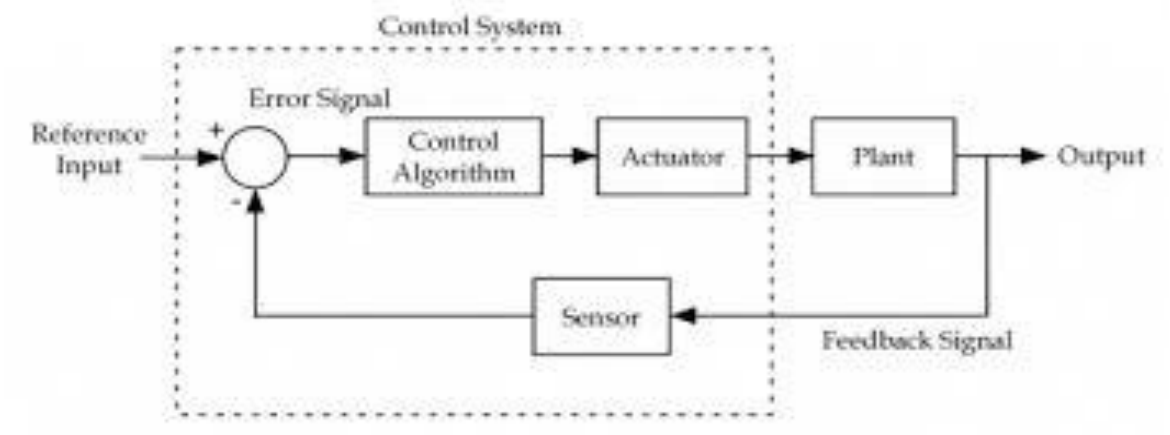
The I-controller's block diagram features an integration block between input and the controller. The I-controller integrates the error signal over time multiplying it by constant integral gain (Ki). This accumulated error correction is added to control signal and gradually eliminating steady-state errors.

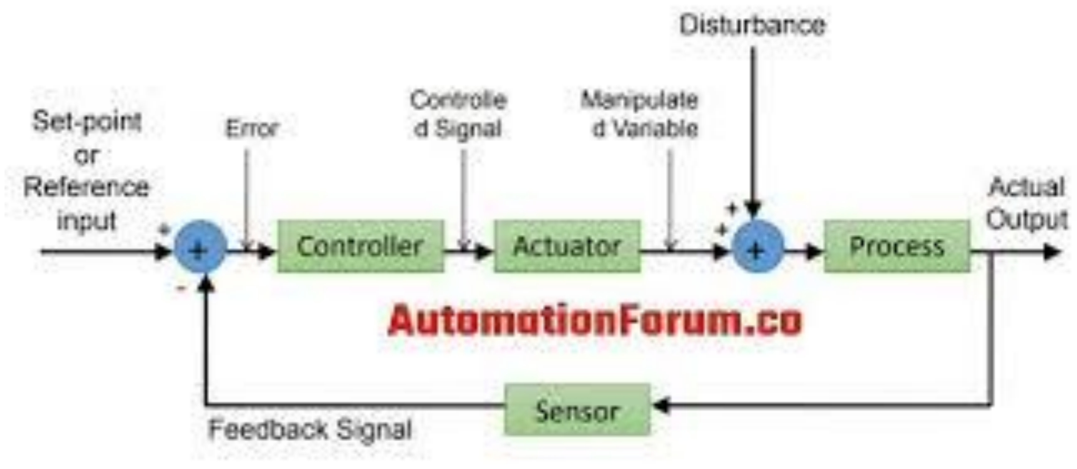
The I-controller eliminates steady-state error but can lead to the slower responses and overshoot if not tuned properly.

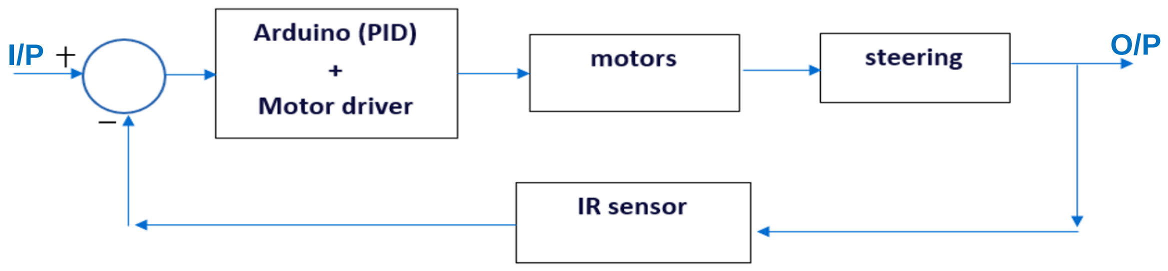




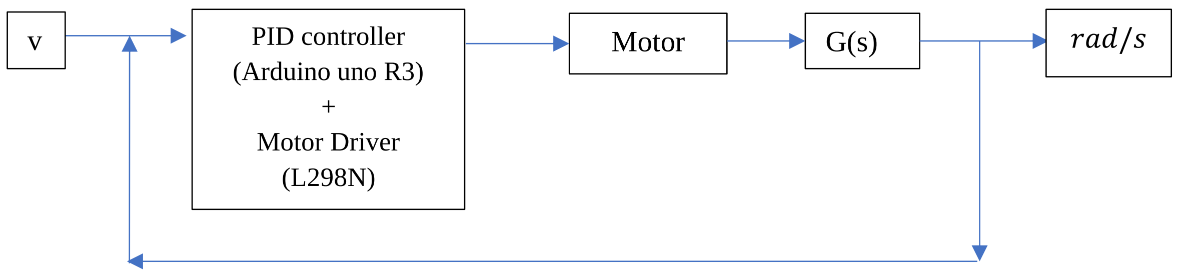
Control system diagram

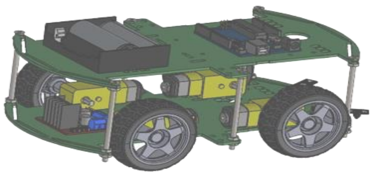






The steering process requires simulation of the whole components that participate in the process. For simplicity we will only simulate the variation of speed for one motor.





Motor Driver

What is a Motor Driver?

A motor driver is a device that acts as an interface between a motor and a controller circuit like a microcontroller. It is needed because motors require higher current and different voltage levels than microcontrollers and other low-power logic chips can provide.

The key role of the motor driver is to take the low current control signals from the controller and convert them into higher power signals that can properly drive the coils of the motor. Essentially it acts as a power amplifier and regulates the power flowing to the motor to enable speed and torque control.

Motor drivers come in the form of integrated circuits (ICs) that are designed for specific motor types like brushed DC motors, stepper motors, and brushless DC motors. Each type requires a different driver architecture and control method tailored to that motor.

Inside the driver IC, transistor configurations like H-bridges allow the driven motors bidirectionally by actively controlling the current and polarity of the coils. Features like pulse width modulation (PWM), current limiting, and braking are commonly included to allow fine motor control.

Why is a Motor Driver Needed?

A motor driver is needed because microcontrollers operate at lower voltages than motors. The microcontroller cannot power the direct drive motors. A driver is required in between to step up the current from the microcontroller to match the higher current needed by the motor. Here are some key reasons why a motor driver is needed in a motor control board system:

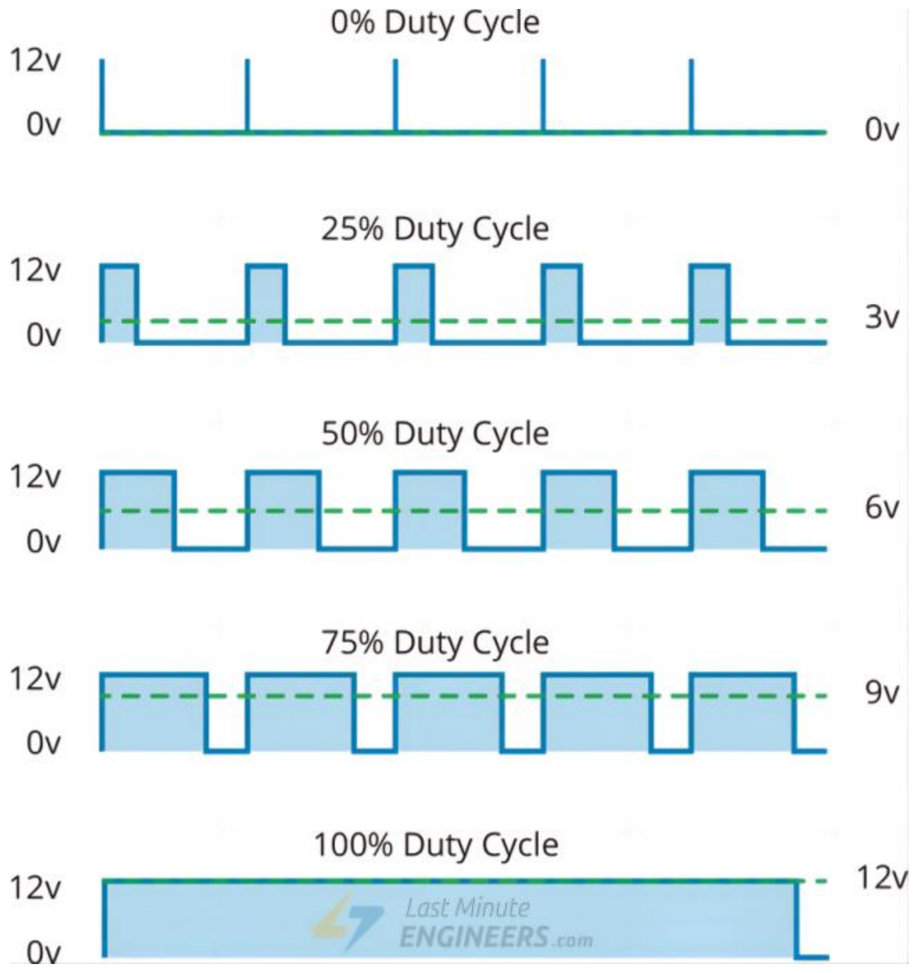
- **Amplification:** Microcontrollers and other logic-level control circuits cannot provide enough current to directly drive a motor. Motor drivers can amplify low-level signals to the higher voltages and currents required by the motor.
- **Voltage Regulation:** The motor usually needs a different operating voltage than the logic-level control circuitry. The driver regulates the voltage as needed by the motor.
- **Precision Control:** Motor drivers allow for control techniques like PWM to vary motor speed and torque accurately. The discrete outputs of the driver translate control signals into proportional power delivery.
- **Overload Protection:** Built-in protections in motor drivers prevent damage from over-current, short circuits, and overheating. This protects both the driver and the motor.

- **Noise Isolation:** The high-power switching inside the motor driver can generate electrical noise. The driver isolates the low-power control circuitry from this noise.
- **Compatibility:** Motor drivers match the requirements of different motor types like brushed DC, stepper, BLDC, etc. This ensures optimal performance.
- **Bidirectional Control:** H-bridge drivers allow both forward and reverse motor operation. The control circuit only needs to command direction digitally.

There are two main techniques used to control the speed and direction of DC motors – PWM (pulse width modulation) and H-bridge circuits.

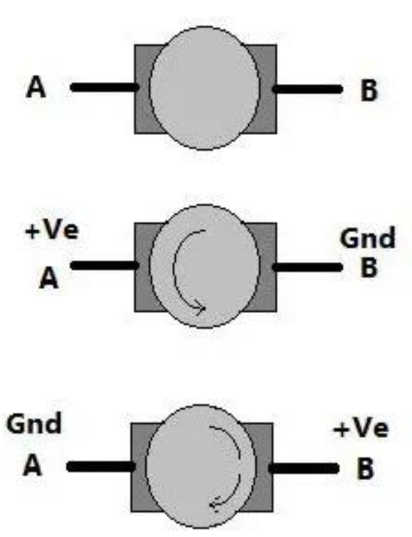
Method 1: PWM – To Control Speed

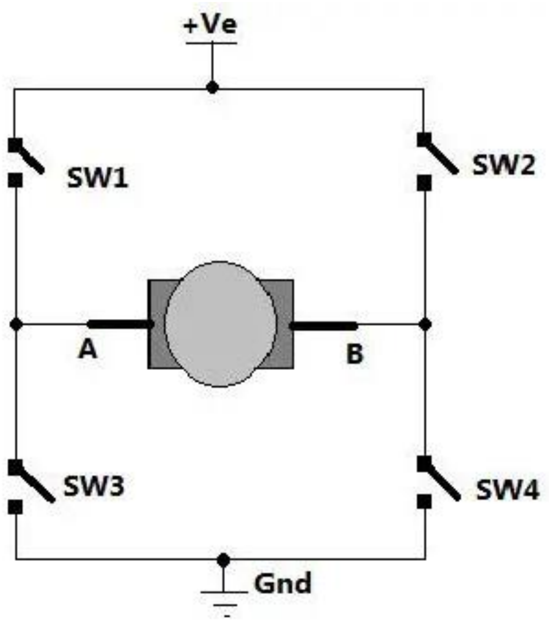
PWM allows variable speed control of a DC motor by rapidly pulsing the power supplied to the motor. By increasing the duty cycle, or the percentage of time the power is on versus off, the more average power is supplied to the motor and it speeds up. Shorter duty cycles provide less power and slow the motor down. PWM creates an adjustable analog-like output pin from a digital signal.



Method 2: H-Bridge – To Control the Spinning Direction

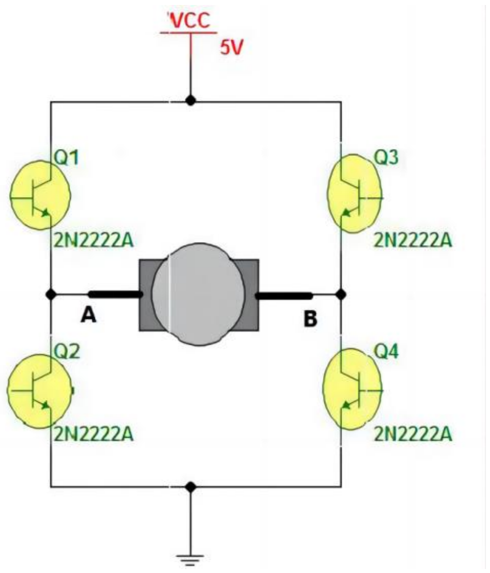
DC motors are natively unidirectional, but their rotation can be reversed by changing the polarity of the voltage applied to their terminals. This direction control can be achieved using a circuit called an H-bridge.

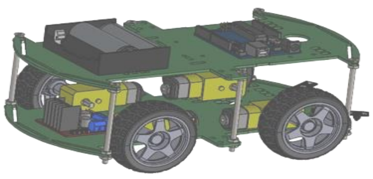




An H-bridge contains four switching elements arranged in the shape of the letter ‘H’, with the motor connected between the poles. The switches are commonly implemented using transistors. By activating the switches on the left or right side of the H, the motor supply voltage polarity can be reversed, thus reversing rotation.

For example, turning on the top left and bottom right switches allows current to flow through the motor in one direction, spinning it clockwise. Activating the bottom left and top right switches reverses the current, reversing the motor direction. The center switches are turned off to prevent short circuits when reversing.





Using an H-bridge gives digital control over motor direction. Only one switch pair is activated at a time, so the circuit electronically switches the polarity applied to the motor. This eliminates the need to manually swap wiring to change rotation.

By using transistors instead of mechanical switches, the bridge can be easily controlled by a microcontroller. The H-bridge provides bidirectional control vital for motorized robotics and mechatronic applications. Combined with PWM speed control, it allows complete control of DC motor direction and speed.

Comparator

A comparator compares two input voltages and outputs a binary signal indicating which is larger. If the non-inverting (+) input is greater than the inverting (-) input, the output goes high. If the inverting input is greater than the non-inverting, the output goes low.

The two basic types of voltage comparator are inverting and non-inverting, depending on which terminal the input signal is applied to.

In an inverting comparator (or negative voltage comparator), the input signal is applied to the inverting terminal and the reference voltage is at the non-inverting terminal. This creates a positive voltage output if the input voltage is less than the reference voltage.

In a non-inverting comparator, the input signal is applied to the non-inverting terminal and the reference voltage is at the inverting terminal. This creates a positive voltage output if the input voltage is greater than the reference voltage.

Car Modeling

Calculate Constants

From motor specification: Max speed happens at very low loads. At 12v and 600 rpm torque is very low so as current

GIVING: (in case of free rotation of motor)

$Power = \ v * I \ = \ 12 * 0.19 = 2.28\, W$

$0.8 * Power = \ T * w = Eb * i$ (Assume efficiency for motor 80%)

- $W \ = \frac{2*\pi*N(rpm)}{60} \ = 20\pi\, rad/s$
- $T \ = 0.029\, N.m = 0.296\, kg.cm = \ 2.96 * 10^{-3}\, kg.m$

$\boldsymbol{k_t \ = k_e = \frac{T}{i} \ = 0.15\, N.M/AMP}$

$j_m = T * r = 2.96 * 10^{-3} * 0.0325 = 9.62 * 10^{-5}\, kg.m^2$ (for motor shaft)

$j_w = 0.5 * m * r^2 = 0.5 * 0.3 * (.0325)^2 = 1.58 * 10^{-4}\, kg.m^2$ (for the wheel)

Every wheel support 0.25 of total mass of the robot car which weigh about 1.1 kg

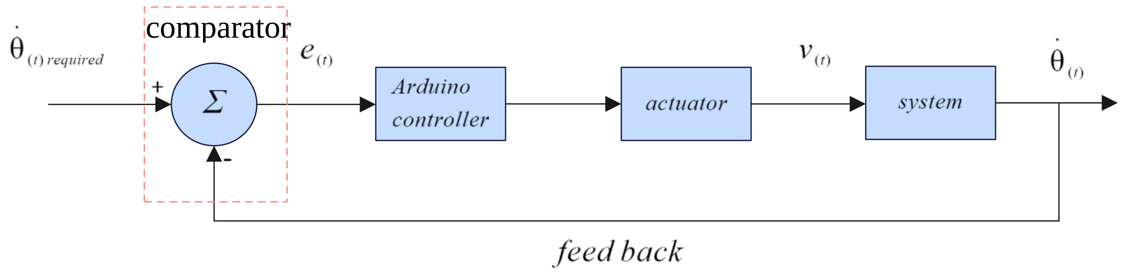
$\mu = 0.4$ (Friction coefficient)

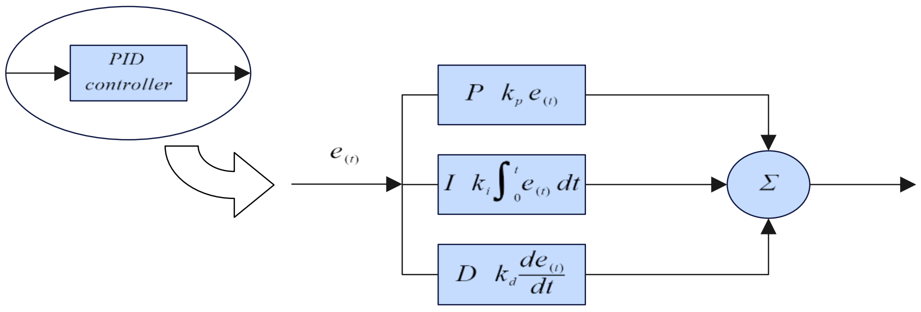
We can calculate impedance by:

$$I \ = 0.19\, A\, , \qquad v = i^2 * z = 0.2 * 2.28$$

(impedance power is the losses that decreased efficiency of motor to 80% so it is 20% of the total power generated)

$z = \ 12.63\, ohm$





PID controller
$e_{(t)}$
$P \quad k_p e_{(t)}$
$I \quad k_i \int_0^t e_{(t)}\,dt$
$D \quad k_d \frac{de_{(t)}}{dt}$
Σ

Mathematical modeling

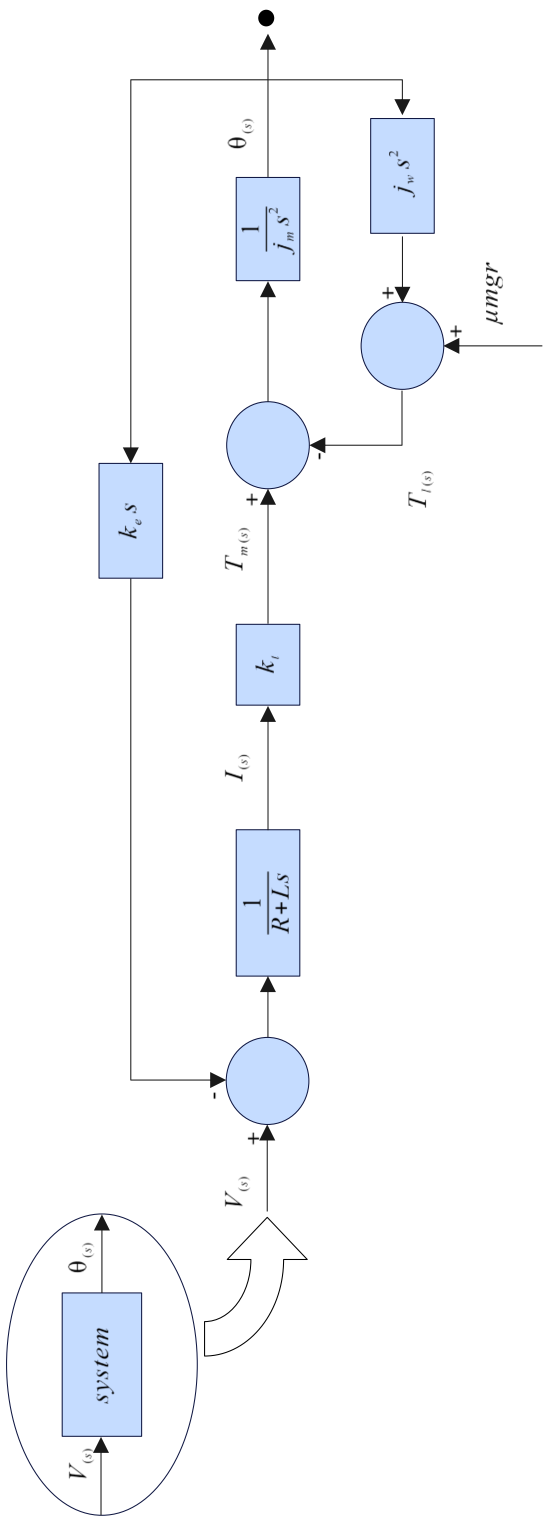
- $V_{(t)} - V_{emf(t)} = Ri_{(t)} + L\frac{di}{dt}$
- $V_{emf(t)} = k_e\dot{\theta}_{(t)}$
- $T_{m(t)} = k_t i_{(t)}$
- $T_{m(t)} - T_{L(t)} = j_m\ddot{\theta}_{(t)}$
- $T_{L(t)} = j_w\ddot{\theta}_{(t)} + \mu mgr$

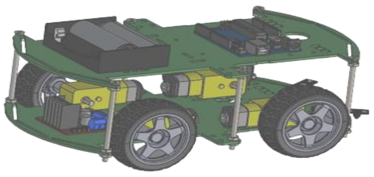
By Applying Laplace inverse to these equations, we get a set of equations in the s-domain which are much easier to deal with:

- $V_{(s)} - V_{emf(s)} = I_{(s)}[R + LS]$
- $V_{emf(s)} = k_e\ S\ \theta_{(s)}$
- $T_{m(s)} = k_t\ I_{(s)}$
- $T_{m(s)} - T_{L(s)} = j_m\ S^2\ \theta_{(s)}$
- $T_{L(s)} = j_w\ S^2\ \theta_{(s)} + \mu mgr$

Now, we set our block diagram to solve it and get the transfer function of the system.

The block diagram





By implying mason's rule, we get the transfer function for the system:

- Forward paths

$$P_1 = \frac{k_t}{(R+LS)(j_m\, S^2)}$$

- Loops

$$L_1 = \frac{-\,k_t\, k_e\, S}{(R+LS)(j_m\, S^2)}$$

$$L_2 = \frac{-\,j_w}{j_m}$$

- Non-Touching loops {All loops touch}

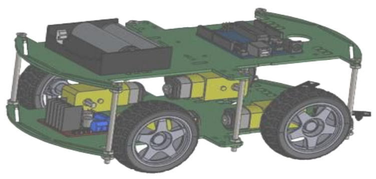
$$\Delta = 1 - \sum l + \sum ll$$

$$\Delta = 1 + \left[\frac{k_t\, k_e\, S}{(R+LS)(j_m\, S^2)}\right] + \left[\frac{j_w}{j_m}\right]$$

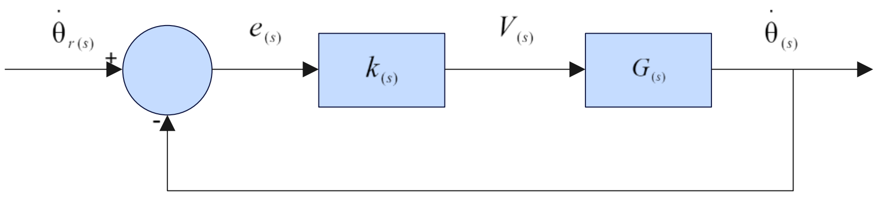
- $\Delta_1 = 1$ {All loops touch the path}

$$T.F = \frac{P_1 \Delta_1}{\Delta}$$

$$= \frac{\frac{k_t}{(R+LS)(j_m\, S^2)}}{1 + \left[\frac{k_t\, k_e\, S}{(R+LS)(j_m\, S^2)}\right] + \left[\frac{j_w}{j_m}\right]} = \frac{k_t}{(R+LS)\,(j_w S^2 + j_m S^2) + k_t\, k_e\, S}$$



We can reduce our block diagram to a simpler form:



Where k$_{(s)}$ is considered to be the controller gain, Therefore, the controller will send a signal according the PID gain values which I gave him to also send a signal to the actuator to make an action to which the system responds with a certain degree, so I compare the system output with the desired input to estimate to what degree our gain values achieved the best performance for our system.

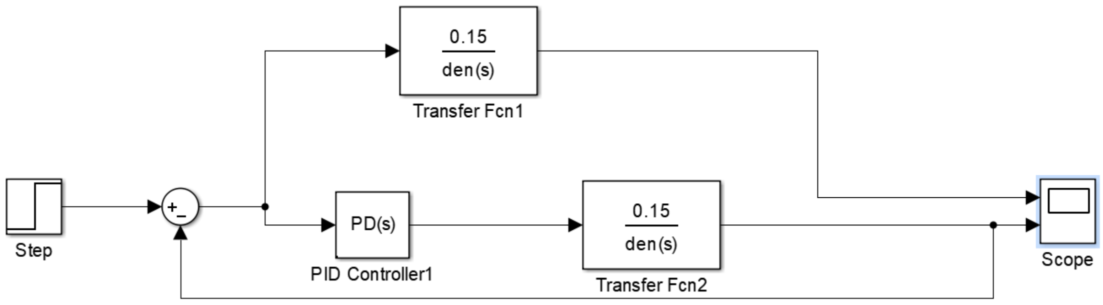
MATLAB

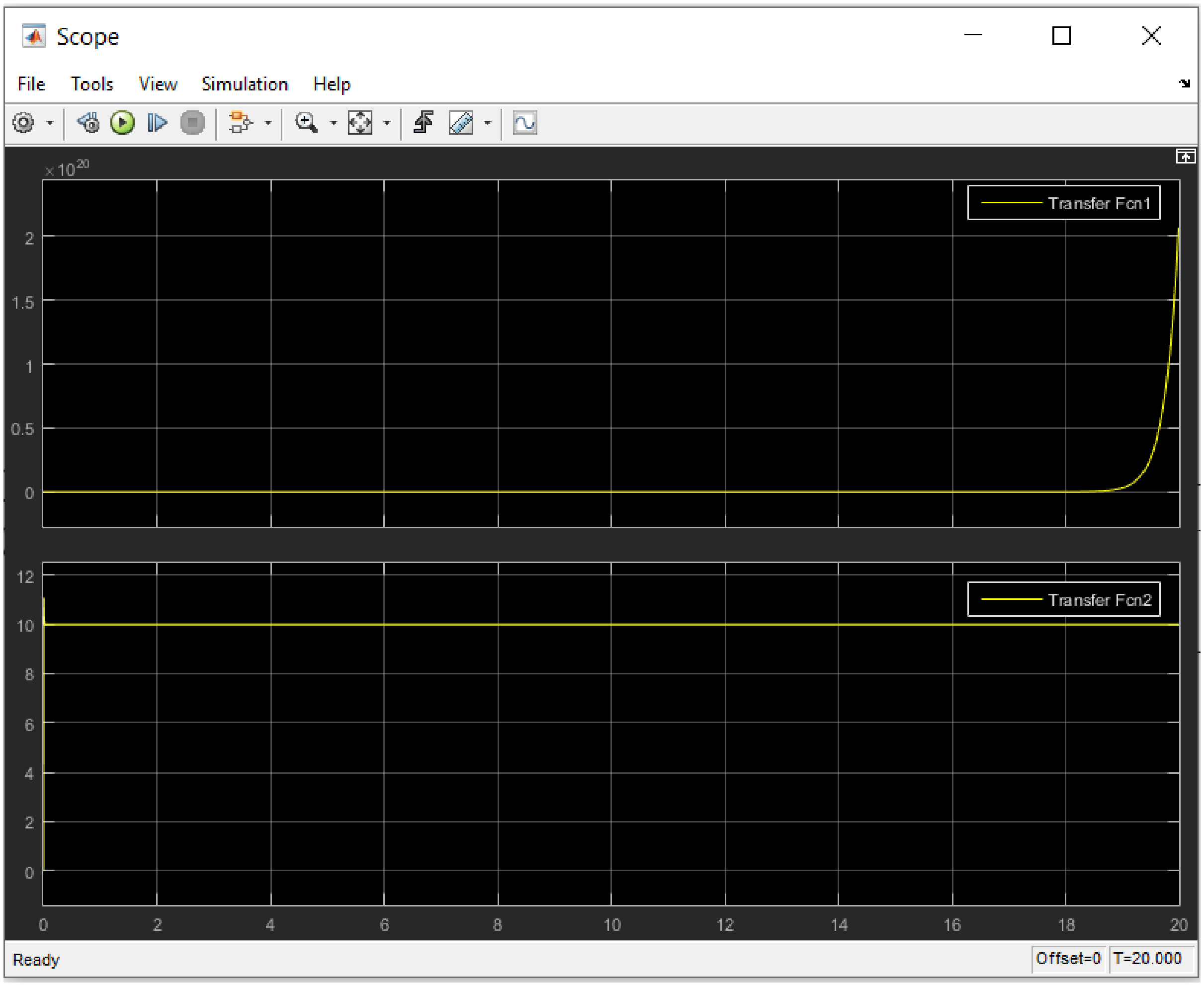
By replacing (R+Ls) by Z Getting forward path transfer function as:

$$G(s) = \frac{\boldsymbol{k_t}}{\boldsymbol{(R+LS)\,(j_w S^2 + j_m S^2) + k_t\, k_e\, S - k_t}}$$

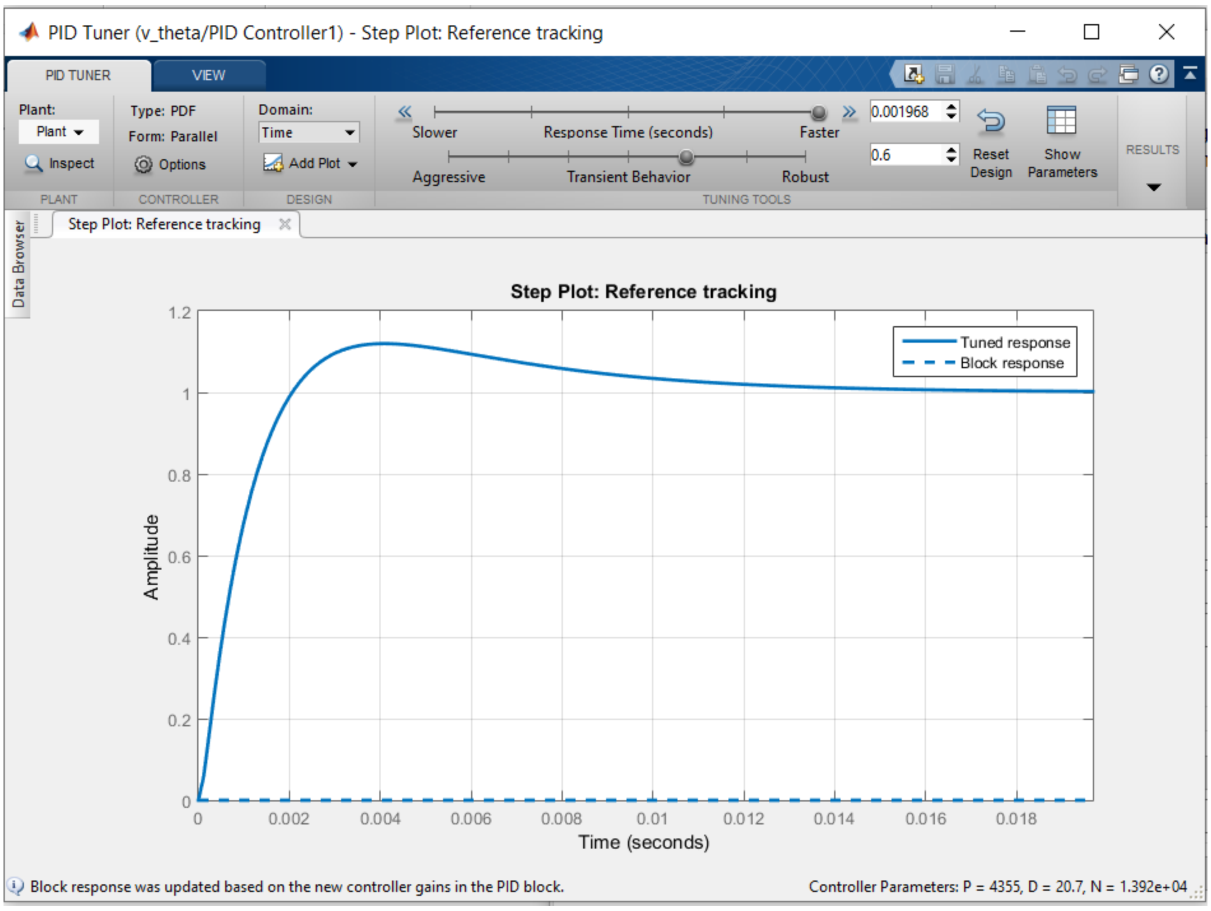
$$G_{(s)} = \frac{0.15}{12.63 * (2.5 * 10^{-4}) * S^2 + (0.0225) * S - 0.15}$$

Input: volt, output:θ





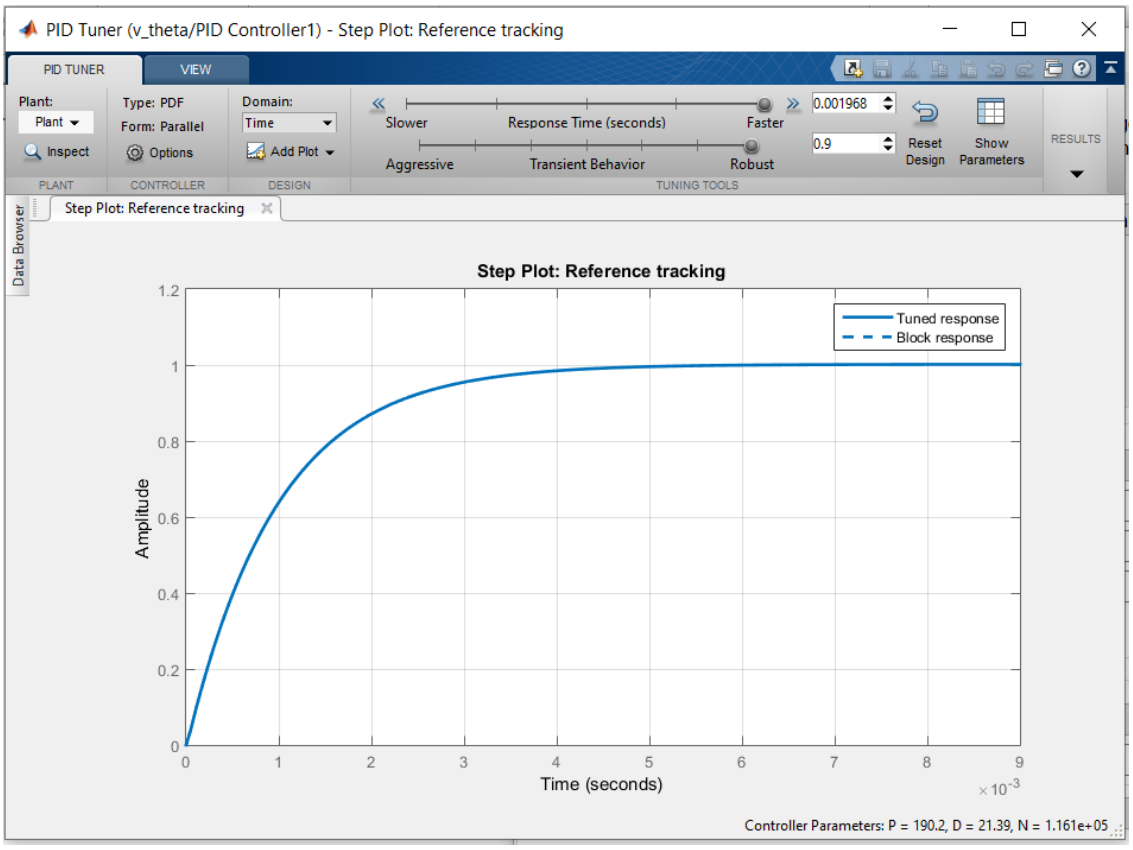
The system without PID controller Is unstable.



Block response was updated based on the new controller gains in the PID block.

Controller Parameters: P = 4355, D = 20.7, N = 1.392e+04

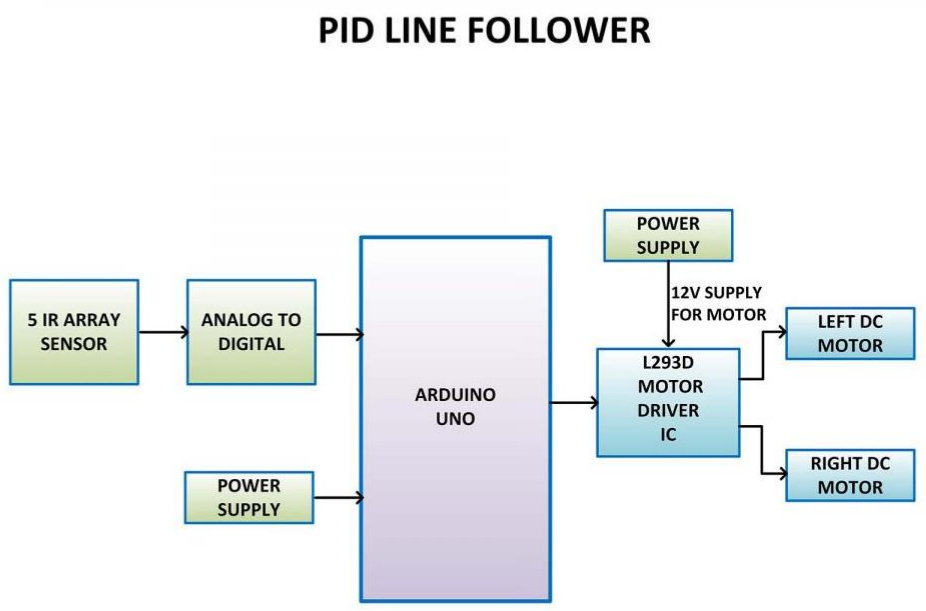
After tuning

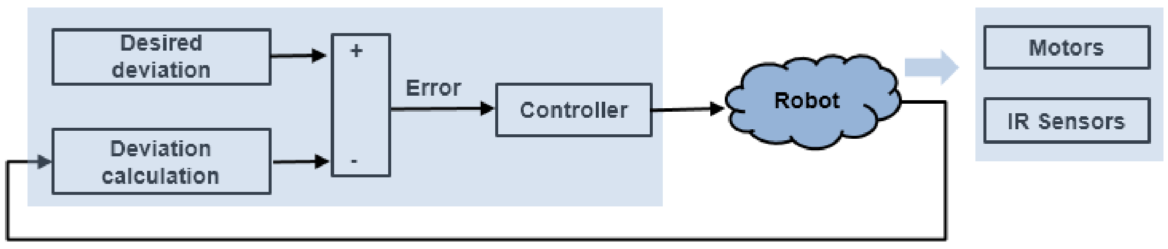


Controller Parameters: P = 190.2, D = 21.39, N = 1.161e+05

Technical part

PID stands for Proportional, Integral and Derivative control. This is a control system algorithm. It is basically a feedback mechanism which can predict error by comparing an output variable with a reference value and apply a correction based on proportional, integral and derivative terms. The algorithm aims to predict and minimize error in an output variable. The algorithm is usually implemented by an electronic controller and it can be implemented in several ways in a programming language. There are even many open-source libraries available which implement the PID algorithm. The PID algorithm is widely used in the industrial control systems like in the robotic arms and assembly lines.

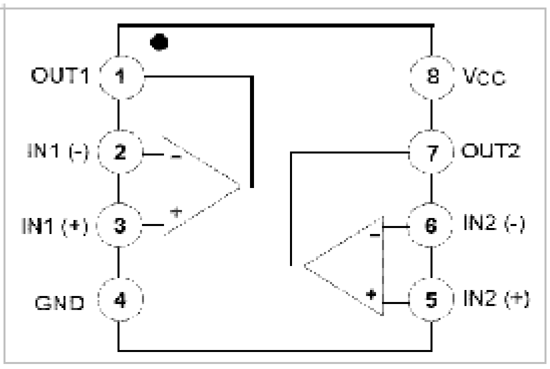




L293N DC Motor Driver IC

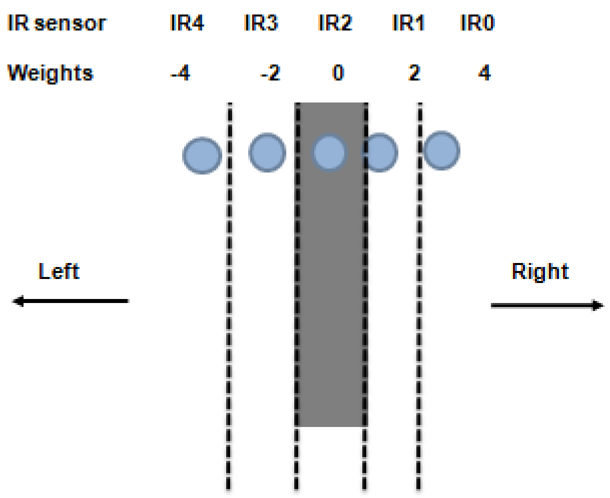
The L293N is a dual H-bridge motor driver integrated circuit (IC). The Motor drivers act as current amplifiers since they take a low-current control signal and provide a higher-current signal. This higher current signal is used to drive the motors. It has 16 pins with following pin configuration:

Pin No	Function	Name
1	Enable pin of Input 1 and 2 of Motor 1	Enable 1,2
2	Input 1 for Motor 1	Input 1
3	Output 1 for Motor 1	Output 1
4	Ground	Ground
5	Ground	Ground
6	Output 2 for Motor 1	Output 2
7	Input 2 for Motor 1	Input 2
8	Supply Voltage for Motor 1 and 2	VS
9	Enable Pin of Input 1 and 2 of Motor 2	Enable 3,4
10	Input 1 for Motor 2	Input 3
11	Output 1 for Motor 2	Output 3
12	Ground	Ground
13	Ground	Ground
14	Output 2 for Motor 2	Output 4
15	Input 2 for Motor 2	Input 4
16	Logic Supply Voltage	VSS



How It Works

we are using 5 array sensors with gap on a way so that only one sensor is centered to the black line and 2 sensors can cover the full width simultaneously. A sensor in the black means it is producing a HIGH or digital 1. So, all the possible the possible sensor array output when following a line is:



0 0 0 0 1

0 0 0 1 1

0 0 0 1 0

0 0 1 1 0

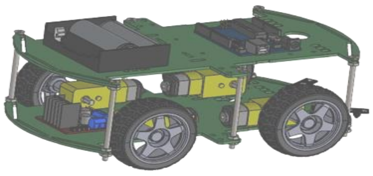
0 0 1 0 0

0 1 1 0 0

0 1 0 0 0

1 1 0 0 0

1 0 0 0 0



Let's consider that the optimum condition is when the robot is centered, The output of the array will be: 0 0 1 0 0 and in this situation, the "error" will be "zero". Let's call 0 0 0 0 1 is the maximum positive error so 1 0 0 0 0 is the minimum negative error. It can be changed upon user's perspective. To make a linear set we counted the error like the following one. So, the possible error would be like:

0 0 1 0 0 ==> Error = 0

0 0 0 0 1 ==> Error = 4

0 0 0 1 1 ==> Error = 3

0 0 0 1 0 ==> Error = 2

0 0 1 1 0 ==> Error = 1

0 1 1 0 0 ==> Error = -1

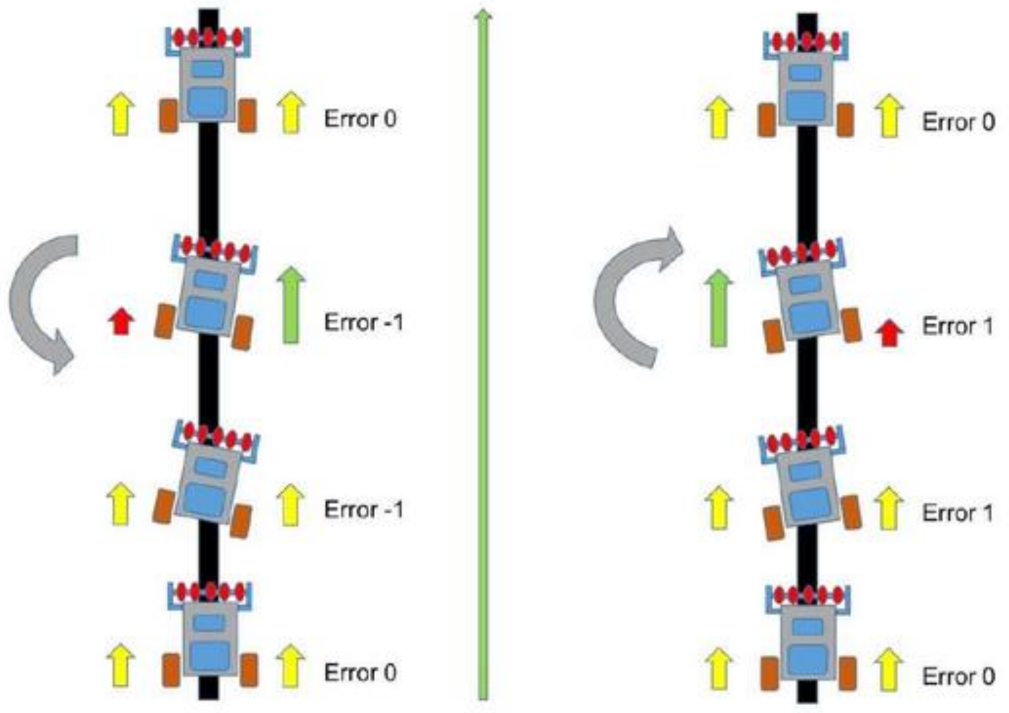
0 1 0 0 0 ==> Error = -2

1 1 0 0 0 ==> Error = -3

1 0 0 0 0 ==> Error = -4

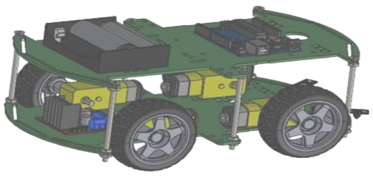
If we look at the following picture the error would be canceled like this

One:



Now we get that at different position the bot will face different error. The bot has to always stick to the line. We see that at positive error the bot has driven (slightly or greatly according to the error) left and at negative error the bot has driven right. So, at our problem, during at the positive error the bot will have to increase its left motor speed and decrease its right motor speed so that the bot can take a quick right turn. At negative error it will have to speed up its right motor and speed down its left motor so that it can take a quick left turn since it has driven right. When this error becomes larger the speed changing value will have to be larger. Like we set up our base speed according to pwm value left and right motor equal speed when error is zero. We saw that in this case there would be 9 possible

6



conditions and we have to set different speed according to their error. The bigger error they are obtained, the bigger difference will generate.

Because the bot will have to come back to line faster. When error is zero we set equal speed to the left and right motor. When maximum error is positive 4 we have to give a maximum fair speed to left motor and minimum fair speed to the right motor to come back to line. This is called simple proportional control.

We can write in our code total nine if. else if condition but the fun thing is that we can cover up these 9 conditions in this simple linear equation:

"motor_speed = base speed ± (error * proportional constant) "

we can see that when the error is zero the speed becomes the base speed and when the error arises the motor speed increases or decreases proportionally. We can set a maximum speed and a base speed and can measure this proportional constant which is called the **Kp** value. By comparing two desired value we can easily measure this **Kp** and control our bot easily. When I wrote code in this all if else if condition I felt about some kind of equation that will handle all the logics that will increase speed at the center at decrease speed at the border. When I got the logic to calculate this **Kp** value and got a superb result.

We have known about the Proportional parameter of the PID equation. With this we can control any control mechanism easily, but with this in can come close to zero error but never obtains it because the **oscillation continues** always.

the PID's main error equation:

$$PIDvalue = (Kp*P) + (Ki*I) + (Kd*D)$$

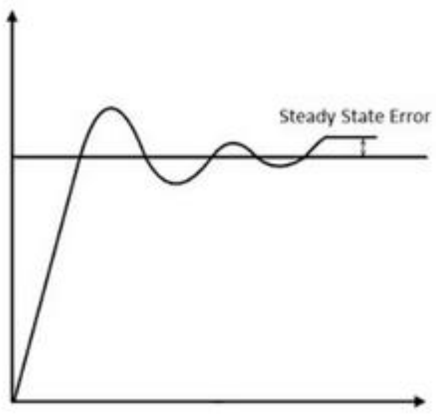
Ki is the constant used to vary the rate at which the change should be brought in the physical quantity to achieve the set point. **Kd** is the constant used to vary the stability of the system.

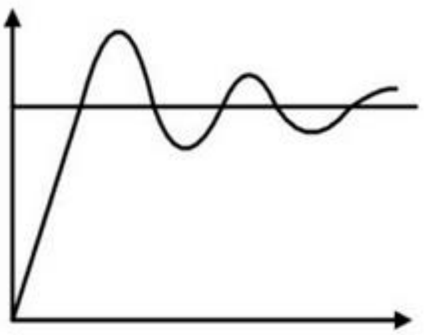
Differential or Derivative Term (D):

This term is the difference between the instantaneous error from the set point, and the error from the previous instant.

D = error - previousError

This value is responsible to slow down the rate of change of the physical quantity when it comes close to the set point. The derivative term is used to **reduce the overshoot.**





PID controller Proportional response response

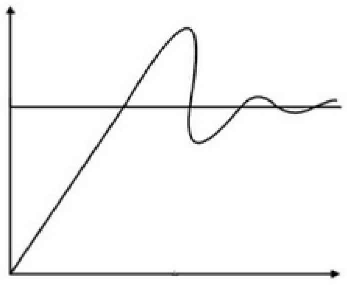
PID controller Derivative

Integral Term (I):

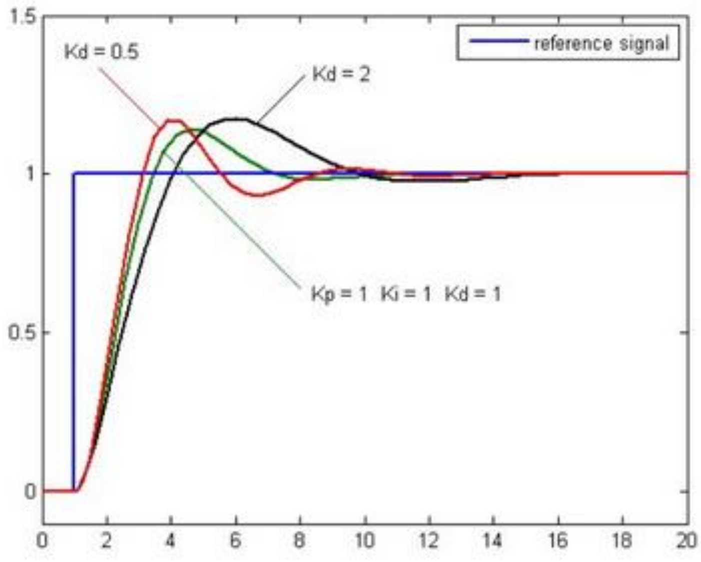
Integral controller is generally used to decrease the steady state error.

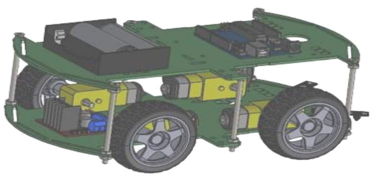
Term ‘I’ is integrate (with respect to time) to the actual value of the error.

Because of integration, very small value of error, results very high integral response. Integral controller action continues to change until error becomes zero.



The final result



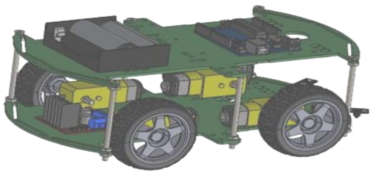


simple Line follower robot with two IR Sensors

It works on the simple principle of a **bang-bang controller**. If the left sensor is on black then it turns right. And if the right sensor is on black then it turns left. tested it out using 12V LiPo battery.

The old code:

```
// Arduino Line Follower Robot Code
#define enA 3//Enable1 L293 Pin enA
#define in1 6 //Motor1  L293 Pin in1
#define in2 7 //Motor1  L293 Pin in1
#define in3 10 //Motor2  L293 Pin in1
#define in4 11 //Motor2  L293 Pin in1
#define enB 9 //Enable2 L293 Pin enB
#define R_S 4//ir sensor Right
#define L_S 2 //ir sensor Left
void setup(){
Serial.begin(9600);
pinMode(R_S, INPUT);
pinMode(L_S, INPUT);
pinMode(enA, OUTPUT);
pinMode(in1, OUTPUT);
pinMode(in2, OUTPUT);
pinMode(in3, OUTPUT);
pinMode(in4, OUTPUT);
pinMode(enB, OUTPUT);
digitalWrite(enA, HIGH);
digitalWrite(enB, HIGH);
delay(1000);
}
void loop(){
if((digitalRead(R_S) == 0)&&(digitalRead(L_S) == 0)){forward();}   //if
Right Sensor and Left Sensor are at White color then it will call forword
function
if((digitalRead(R_S) == 1)&&(digitalRead(L_S) == 0)){turnRight();} //if
Right Sensor is Black and Left Sensor is White then it will call turn Right
function
if((digitalRead(R_S) == 0)&&(digitalRead(L_S) == 1)){turnLeft();}  //if
Right Sensor is White and Left Sensor is Black then it will call turn Left
function
if((digitalRead(R_S) == 1)&&(digitalRead(L_S) == 1)){Stop();} //if Right
Sensor and Left Sensor are at Black color then it will call Stop function
}
void forward(){   //forword
        analogWrite(enA, 50);
        analogWrite(enB, 50);
digitalWrite(in1, LOW); //Right Motor forword Pin
digitalWrite(in2, HIGH);  //Right Motor backword Pin
digitalWrite(in3, HIGH);  //Left Motor backword Pin
digitalWrite(in4, LOW); //Left Motor forword Pin
}
```



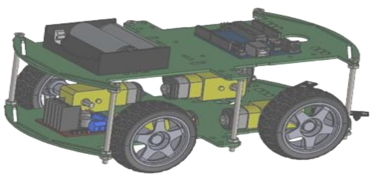
```
void turnRight(){ //turnRight
        analogWrite(enA, 50);
        analogWrite(enB, 50);
digitalWrite(in1, LOW); //Right Motor forword Pin
digitalWrite(in2, HIGH);  //Right Motor backword Pin
digitalWrite(in3, LOW); //Left Motor backword Pin
digitalWrite(in4, HIGH);  //Left Motor forword Pin
}
void turnLeft(){ //turnLeft
        analogWrite(enA, 50);
        analogWrite(enB, 50);
digitalWrite(in1, HIGH);  //Right Motor forword Pin
digitalWrite(in2, LOW); //Right Motor backword Pin
digitalWrite(in3, HIGH);  //Left Motor backword Pin
digitalWrite(in4, LOW); //Left Motor forword Pin
}
void Stop(){ //stop
digitalWrite(in1, LOW); //Right Motor forword Pin
digitalWrite(in2, LOW); //Right Motor backword Pin
digitalWrite(in3, LOW); //Left Motor backword Pin
digitalWrite(in4, LOW); //Left Motor forword Pin

}
```

After Using PID and add 5 IR sensors

The Arduino Code

```
// Author: Yossri
#include <L298N.h>
#include <QTRSensors.h>
//*************************************************
// Motors Pins
#define PWMA 10
#define AIN1 8  // Assign the motor pins
#define AIN2 9

#define BIN1 6
#define BIN2 7
#define PWMB 5

const int offsetA = 1;
const int offsetB = 1;

L298N motor1(PWMA, AIN1, AIN2);
L298N motor2(PWMB, BIN1, BIN2);
//*************************************************
//Sensors defintions
QTRSensors qtr;

const uint8_t SensorCount = 5;
uint16_t sensorValues[SensorCount];
int threshold[SensorCount];
//******************************************************
//PID k values and intial speeed
float Kp = 2.0;  // Initial PID parameters
float Ki = 0;
float Kd =0.09;

int lfspeed = 200;
//*************************************
// TO Make values integer
uint8_t multiP = 1;
uint8_t multiI  = 1;
uint8_t multiD = 1;
uint8_t Kpfinal;
uint8_t Kifinal;
uint8_t Kdfinal;
float Pvalue;
float Ivalue;
float Dvalue;
```

```
boolean onoff = false;

int val, cnt = 0, v[3];
uint16_t position;
int P, D, I, previousError, PIDvalue, error;
int lsp, rsp;

//**************************
//Calibration of sensors

void setup() {
  qtr.setTypeAnalog();
  qtr.setSensorPins((const uint8_t[]){A0, A1, A2, A3, A4}, SensorCount);

  delay(500);
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, HIGH); // Turn on Arduino's LED to indicate
calibration mode

  Serial.begin(115200);

  for (uint16_t i = 0; i < 200; i++) {
    qtr.calibrate();
  }
  digitalWrite(LED_BUILTIN, LOW); // Turn off Arduino's LED after
calibration
//************************************
  // Compute the threshold values

  for (uint8_t i = 0; i < SensorCount; i++) {
double thresholdRatio = 0.2;  // Adjust this value between 0 and 1

    threshold[i] = (qtr.calibrationOn.minimum[i] +
qtr.calibrationOn.maximum[i]) / 2;
      Serial.print(threshold[i]);
    Serial.print("  ");
  }

  delay(3000);
}

void loop() {
  // Read sensor values and perform line-following
  position = qtr.readLineBlack(sensorValues);
  error = 2000 - position;
    // Check if all sensors detect white (indicating the end of the line)
  if (sensorValues[0] >= 980 && sensorValues[1] >= 980 && sensorValues[2]
>= 980 && sensorValues[3] >= 980 && sensorValues[4] >= 980) {
```



```
      // Stop the motors
      motor_drive(0, 0);
      return; // Exit the loop
    }
    PID_Linefollow(error);
}
//**********************
//calculate PID value
void PID_Linefollow(int error) {
    P = error;
    I = I + error;
    D = error - previousError;

      Pvalue = (Kp/pow(10,multiP))*P;
      Ivalue = (Ki/pow(10,multiI))*I;
      Dvalue = (Kd/pow(10,multiD))*D;

    float PIDvalue = Pvalue + Ivalue + Dvalue;
    previousError = error;

    lsp = lfspeed - PIDvalue;
    rsp = lfspeed + PIDvalue;

int m=220;
    if (lsp > m) {
      lsp = m;
    }
    if (lsp < -m) {
      lsp = -m;
    }
    if (rsp > m) {
      rsp = m;
    }
    if (rsp < -m) {
      rsp = -m;
    }
    motor_drive(lsp, rsp);
}
//************************
//set motor directions
void motor_drive(int left, int right) {
    if (right > 0) {
      motor2.setSpeed(right);
      motor2.forward();
    } else {
      motor2.setSpeed(abs(right)); // Set speed using absolute value for
backward movement
      motor2.backward();
    }
```

```
  if (left > 0) {
    motor1.setSpeed(left);
    motor1.forward();
  } else {
    motor1.setSpeed(abs(left)); // Set speed using absolute value for
backward movement
    motor1.backward();
  }
  // Print the motor speeds
  Serial.print("L: ");
  Serial.print(left);
  Serial.print("  ||  ");
  Serial.print("R:  ");
  Serial.println(right);
}
```