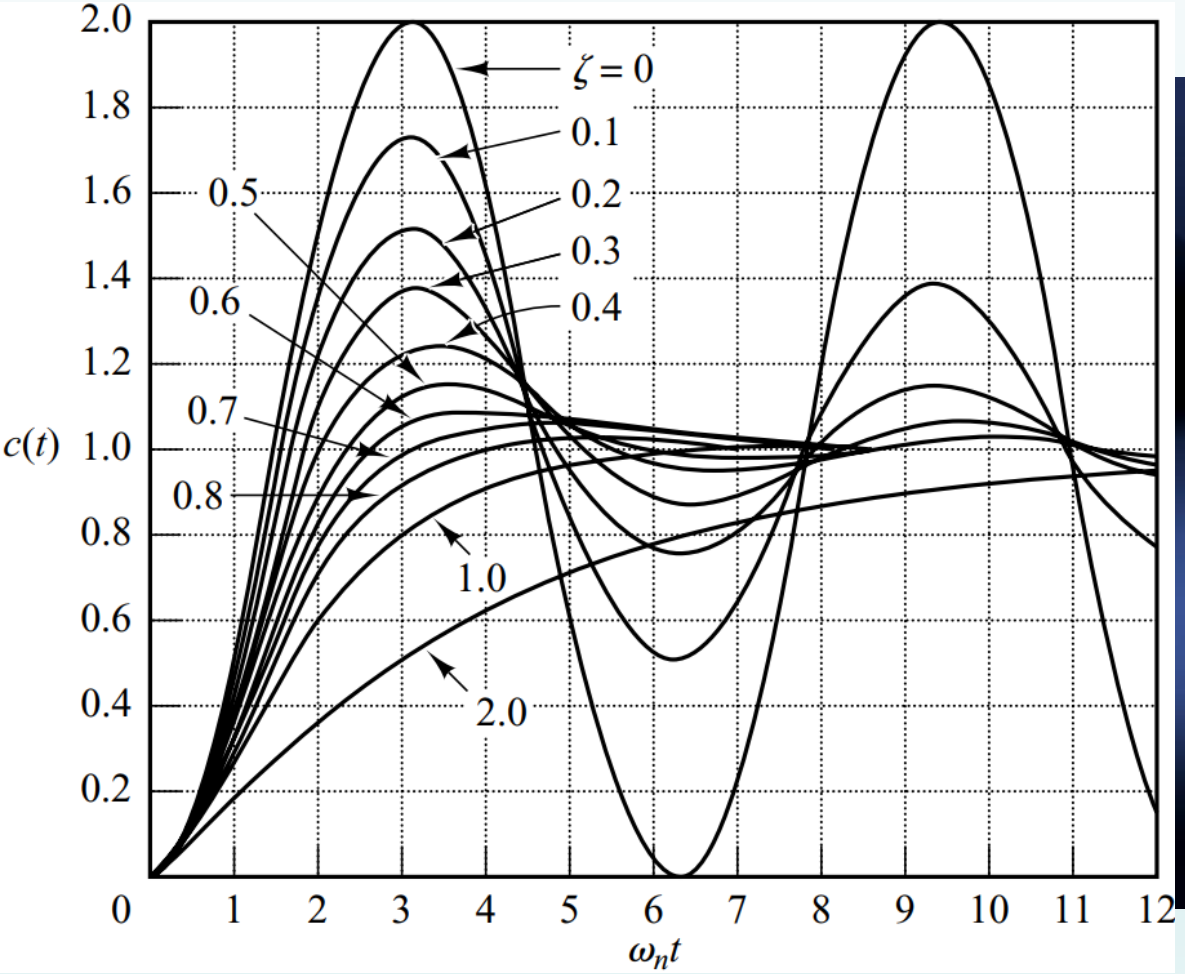


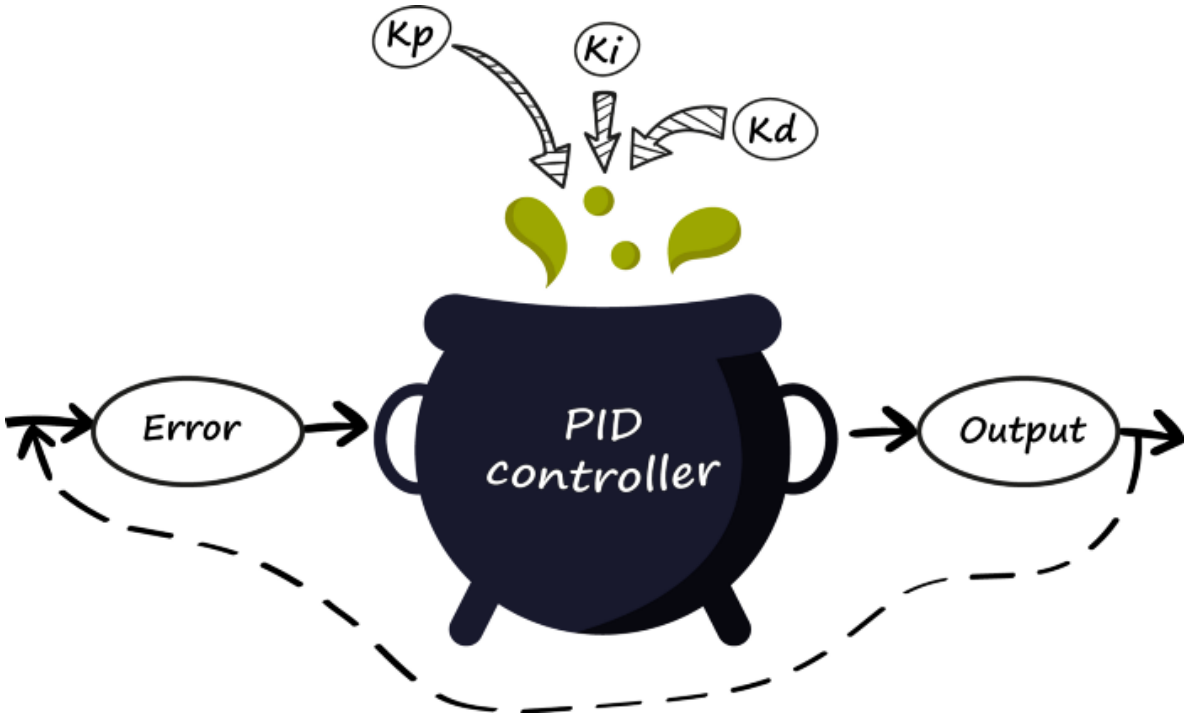
PMR 3100 – Introdução à Engenharia Mecatrônica

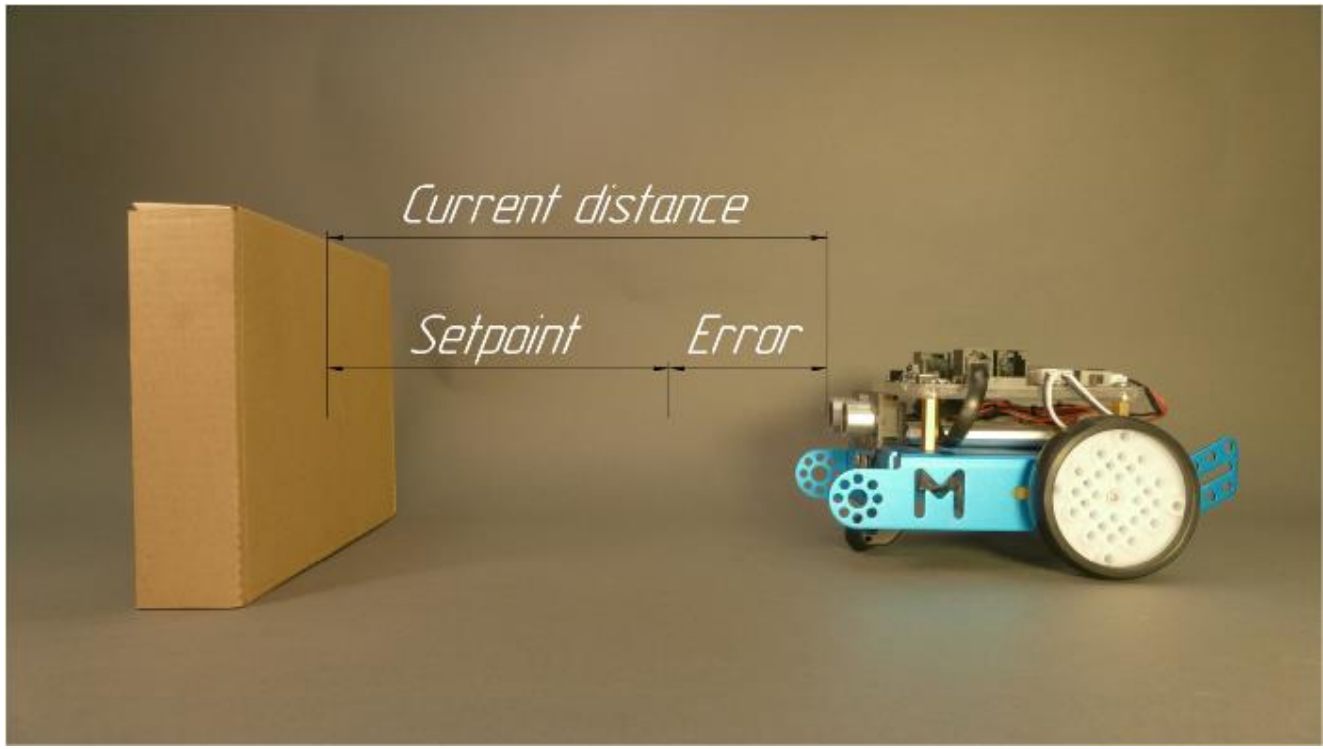
Módulo 04 – Meu Primeiro Robô

Aula 22 – Controle PID para seguidor de linha

Prof. Dr. Rafael Traldi Moura

Controle Proporcional Integrativo Derivativo





$$erro = x - setpoint$$

$$erro_i = \sum (x - setpoint)$$

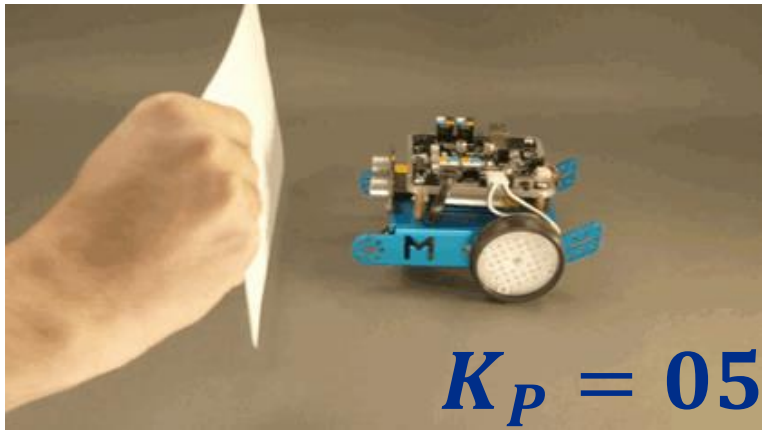
$$erro_d = \frac{\Delta erro}{\Delta t}$$

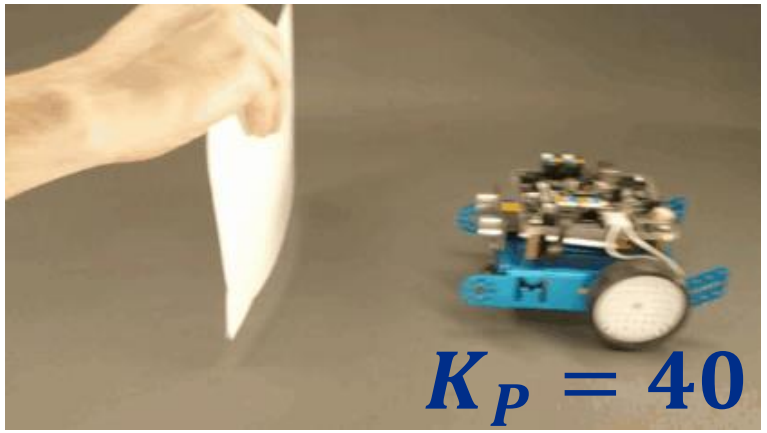
Fonte: XODlang

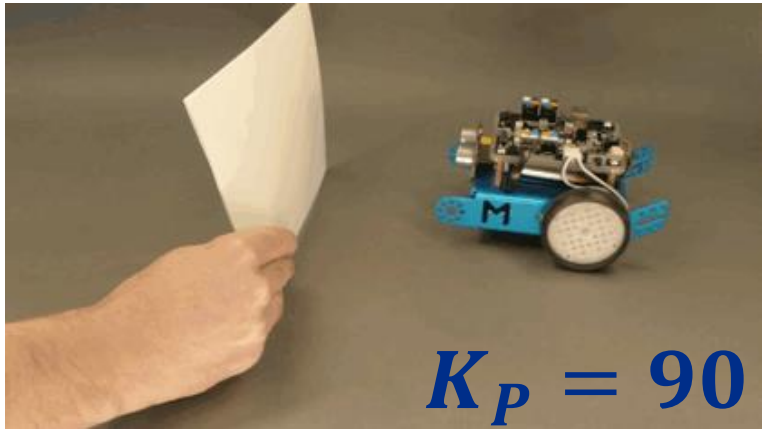
Controle Proporcional

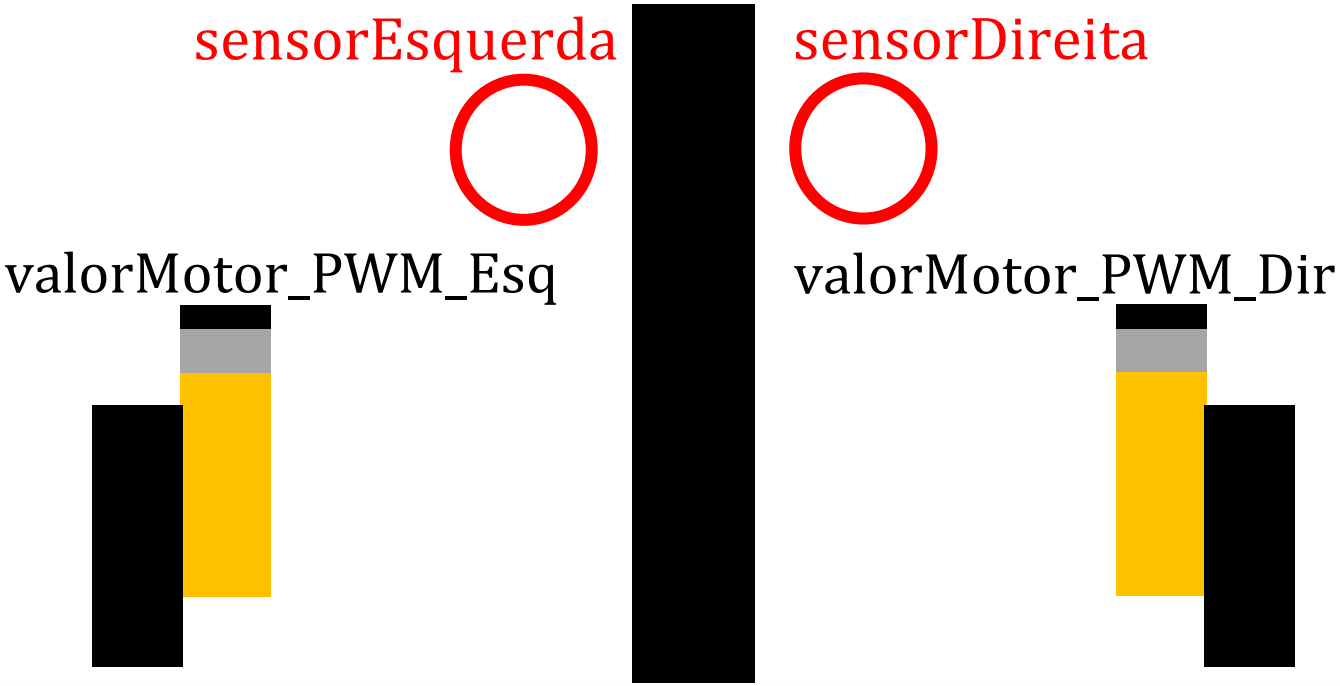
Fonte: XODlang

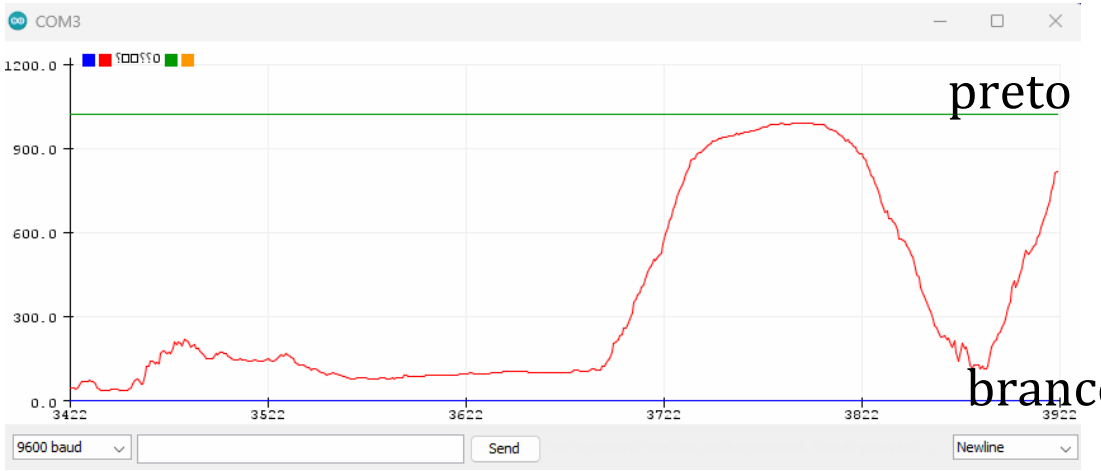
$$ação = K_P \cdot erro$$



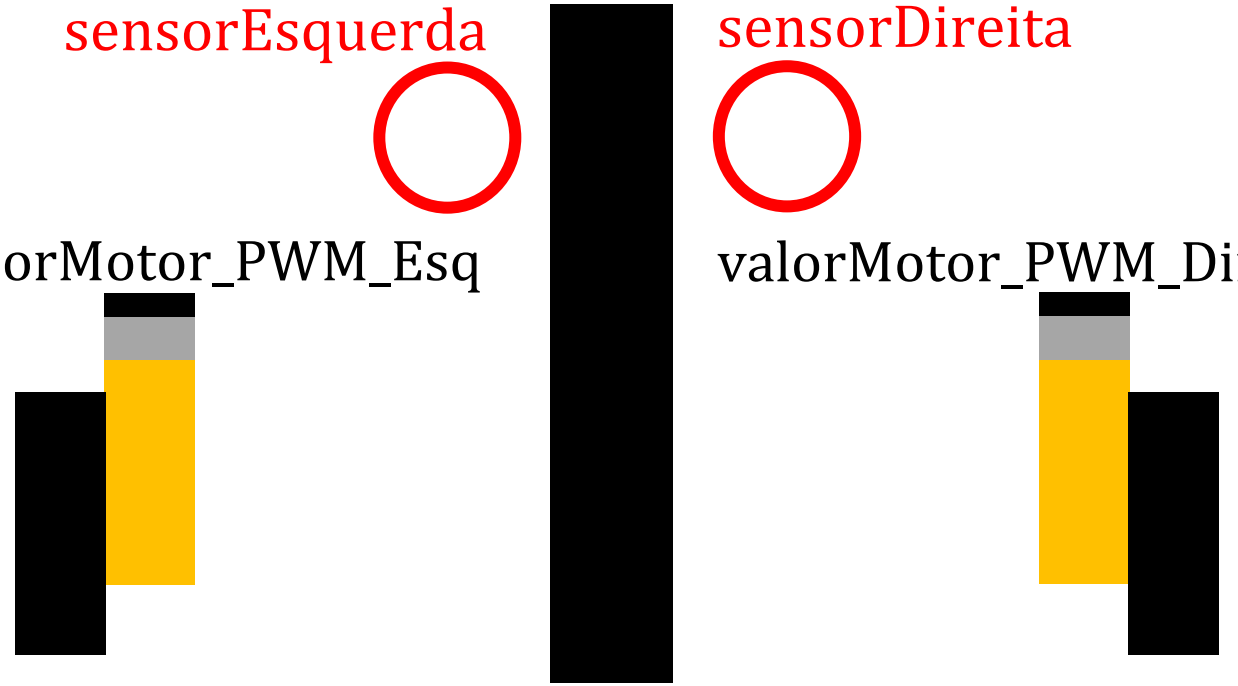


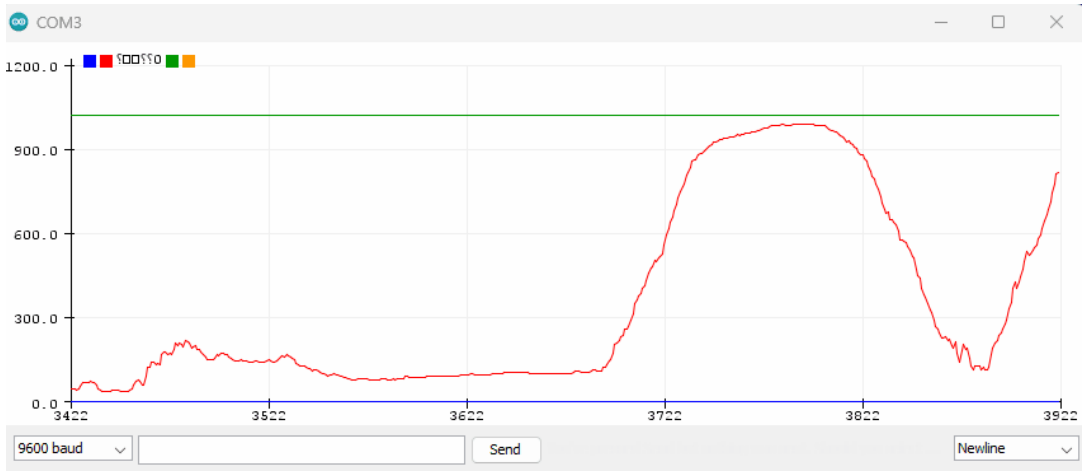






Controle Proporcional





$$ação = K_P \cdot erro$$

Se o sensorEsquerda e sensorDireita estão no branco, ambos tem por saída valores pequenos. Ou seja e devemos ter uma velocidade constante:

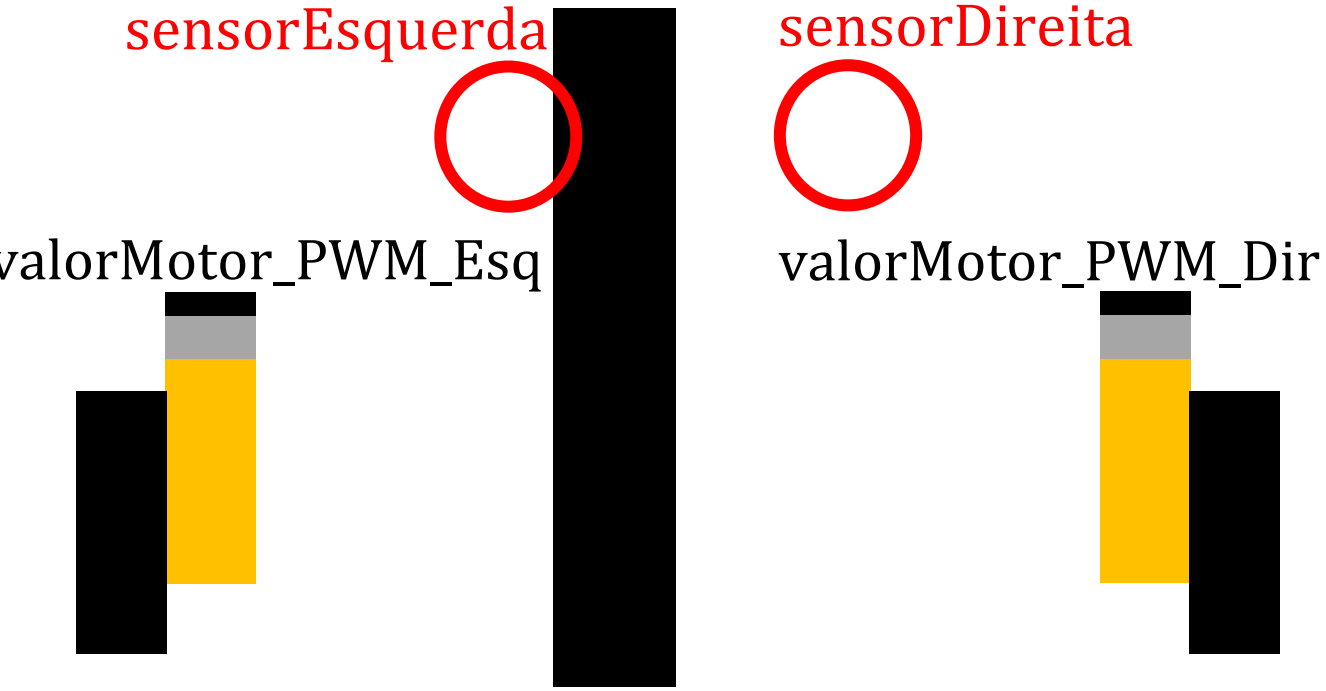
$$valorMotor\_PWM\_Esq = (int)(V0 + K_P \cdot erro)$$

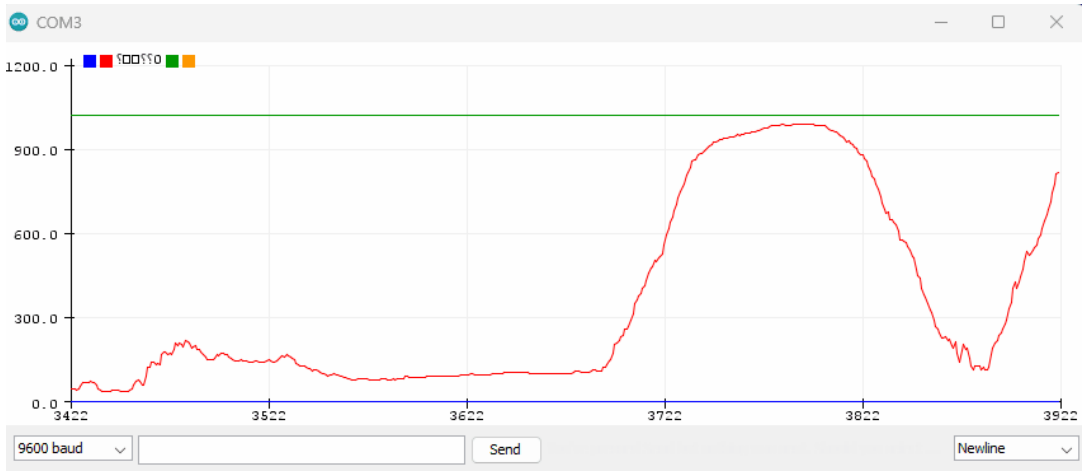
$$valorMotor\_PWM\_Dir = (int)(V0 + K_P \cdot erro)$$

Com o valor do erro podendo ser a soma dos valores dos sensores de linha, que são valores baixos. Podemos inclusive tirar esse valor constante de leitura do branco, chamando-o de desvio;

$$erro = sensorEsquerda + sensorDireita - 2 \cdot desvio$$

Controle Proporcional





$$ação = K_P \cdot erro$$

Se o sensorEsquerda detecta a linha, seu valor aumenta.

E devemos virar para a esquerda, ou seja:

- valorMotor_PWM_Esq deve diminuir;
- valorMotor_PWM_Dir deve aumentar;

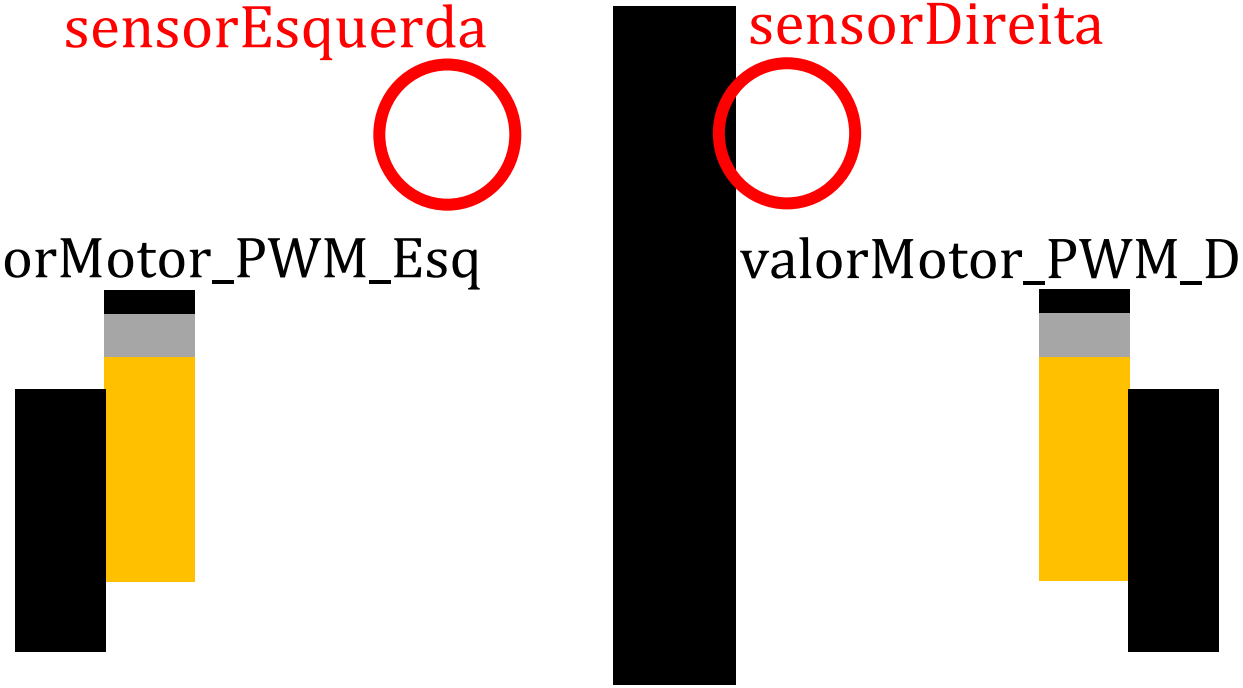
Adotando sinal positivo para virar para a esquerda, temos:

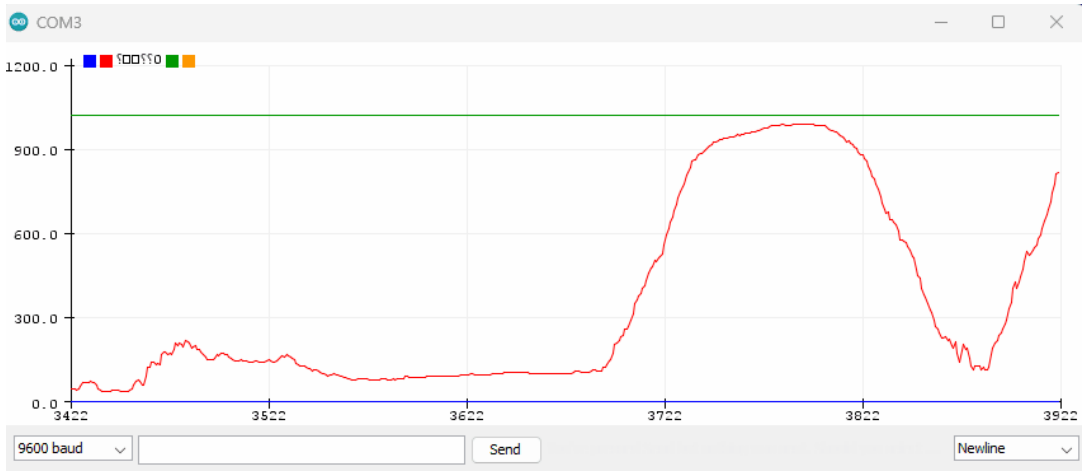
$$valorMotor\_PWM\_Esq = (int)(V0 + K_P \cdot erro)$$

$$erro = -(sensorEsquerda - desvio)$$

$$valorMotor\_PWM\_Dir = (int)(V0 - K_P \cdot erro)$$

Controle Proporcional





$$ação = K_P \cdot erro$$

Se o sensorDireita detecta a linha, seu valor aumenta.

E devemos virar para a direita, ou seja:

- valorMotor_PWM_Esq deve aumentar;
- valorMotor_PWM_Dir deve diminuir;

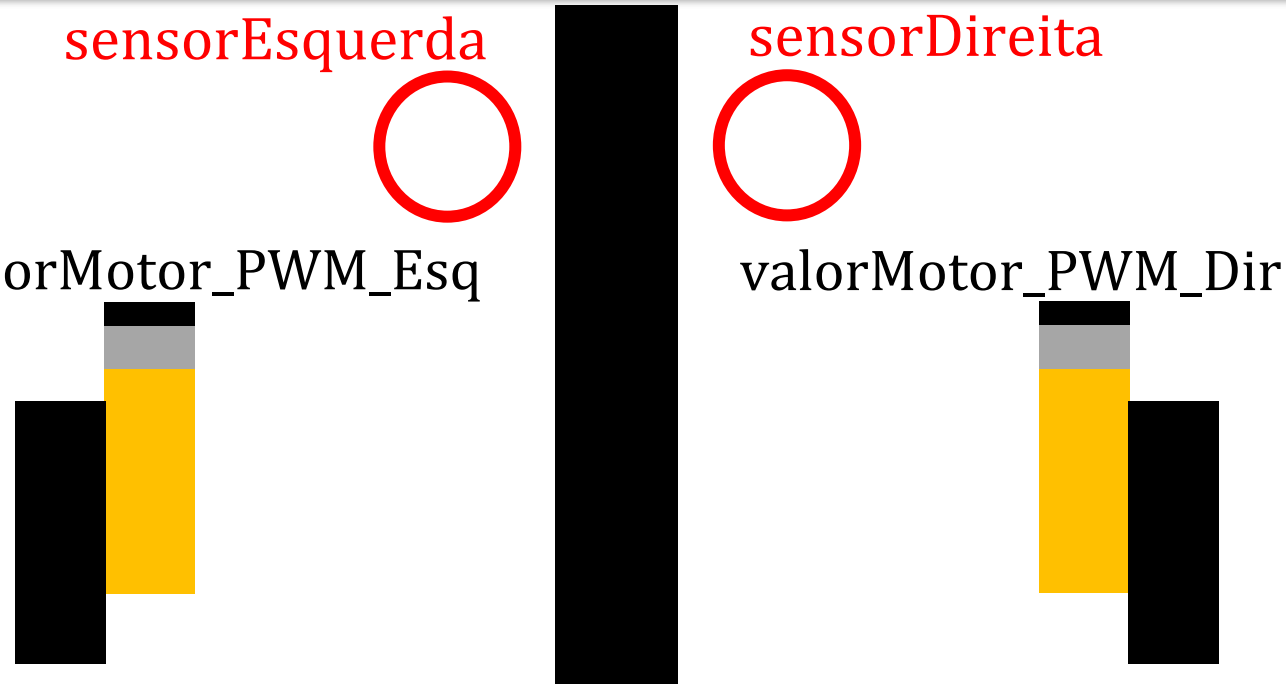
Adotando sinal positivo para virar para a esquerda, temos:

$$valorMotor\_PWM\_Esq = (int)(V0 + K_P \cdot erro)$$

$$erro = +(sensorDireita - desvio)$$

$$valorMotor\_PWM\_Dir = (int)(V0 - K_P \cdot erro)$$

Controle Proporcional



$$ação = K_P \cdot erro$$

Adotando sinal positivo para virar para a esquerda, e juntando as contribuições anteriores, temos:

$$valorMotor\_PWM\_Esq = (int)(V0 + K_P \cdot erro)$$

$$valorMotor\_PWM\_Dir = (int)(V0 - K_P \cdot erro)$$

$$erro = -sensorEsquerda + sensorDireita$$

Carrinho PID de distância – códido parte 1

- A primeira parte do código contém:
  - define
  - declarações de variáveis globais

```
#define bitMotor1A 2
#define bitMotor1B 3
#define bitMotor2A 1
#define bitMotor2B 4
#define pinMotor1PWM 11
#define pinMotor2PWM 3

#define pinSH_CP  4   //Pino Clock  DIR_CLK
#define pinST_CP  12  //Pino Latch  DIR_LATCH
#define pinDS     8   //Pino Data   DIR_SER
#define pinEnable 7   //Pino Enable DIR_EN

#define sensorDireita A0
#define sensorEsquerda A1
#define desvio 80
#define V0 100
#define V_max 250
#define V_min 0
```

Carrinho PID de distância – códido parte 2

- A primeira parte do código contém:
  - define
  - declarações de variáveis globais

```
void ci74HC595Write(byte pino, bool estado);
void inicializa_Motor_Shield();

float valorMotor_PWM_Esq,valorMotor_PWM_Dir;
int leituraEsquerda, leituraDireita;
float erro,integral,der,erro_anterior,PID;
float Kp,Ki,Kd;
```

Carrinho PID de distância – códido parte 3

- A segunda parte do código contém: a função `setup()` com:
  1. Inicialização do shield controlador de motores;
  2. Inicialização e configuração do sensor de linha;
  3. inicialização da comunicação serial em 9600kbps;
  4. Atribuição da configuração inicial para os motores;
  5. Inicialização das variáveis do PID;

```
void setup(){
//inicializa e configura pinos usados shield
de controle do motor
   inicializa_Motor_Shield();
//inicializa e configura pinos usados nos
sensores de linha
   pinMode(sensorEsquerda, INPUT);
   pinMode(sensorDireita, INPUT);
//inicializa e configura comunicação serial
   Serial.begin(9600);
//inicializa as variaveis dos motores
   valorMotor_PWM_Esq = V0;
   valorMotor_PWM_Dir = 0;
   ci74HC595Write(bitMotor1A, HIGH);
   ci74HC595Write(bitMotor1B, LOW);
   ci74HC595Write(bitMotor2A, HIGH);
   ci74HC595Write(bitMotor2B, LOW);
```

Carrinho PID de distância – códido parte 4

- A segunda parte do código contém: a função `setup()` com:
  1. Inicialização do shield controlador de motores;
  2. Inicialização e configuração do sensor de linha;
  3. inicialização da comunicação serial em 9600kbps;
  4. Atribuição da configuração inicial para os motores;
  5. Inicialização das variáveis do PID;

```
//inicializa as variaveis do PID
    erro = 0;
    integral = 0;
    der = 0;
    erro_anterior = 0;
    Kp = 1.0;
    Kd = 0.0;
    Ki = 0.0;
  Serial.println("Esperando 1 segundos");
  delay(1000);
  Serial.println("Setup completo");
}
```

Carrinho PID de distância – códido parte 5

- Na terceira parte do código começa a função loop.
- É implementada a leitura dos sensores de linha, tirando o desvio da leitura no branco;

```
void loop() {
// LÊ OS SENSORES ----------------------
  //Lê o senor de linha
  leituraDireita =
analogRead(sensorDireita)-desvio;
  leituraEsquerda =
analogRead(sensorEsquerda)-desvio;
  Serial.print("S_Esq= ");
  Serial.print(leituraEsquerda);
  Serial.print(" S_Dir= ");
  Serial.print(leituraDireita);
// fim de LÊ OS SENSORES---------------
```

Carrinho PID de distância – códido parte 6

- Na quarta parte do código continua a função loop.
- Nesta parte é implementado o controle;

```
// CALCULA O CONTROLE  ------------------------
  // NESTE controle, vamos assumir
  // que virar para a esquerda é positivo
  //calcula os termos do PID
  erro = leituraDireita - leituraEsquerda;
  integral = integral + erro;
  der = erro - erro_anterior;
  erro_anterior = erro;
  //calcula o valor do PID e a velocidade dos motores
  PID = Kp*erro + Ki*integral + Kd*der;
  valorMotor_PWM_Esq =(int)(V0 + PID);
  valorMotor_PWM_Dir =(int)(V0 - PID);
// fim de CALCULA O CONTROLE  ------------------------
```

Carrinho PID de distância – códido parte 7

- Na quinta parte do código termina a função loop.
- Nesta parte temos a verificação se os atuadores estão saturados;
- E atualizamos os valores dos atuadores.

```
// ATUALIZA VALOR DOS ATUADORES  --------------------
  //verifica saturacao dos motores
  if (valorMotor_PWM_Esq>V_max){
    valorMotor_PWM_Esq = V_max;
  }if (valorMotor_PWM_Esq<V_min){
    valorMotor_PWM_Esq = V_min;
  }if (valorMotor_PWM_Dir>V_max){
    valorMotor_PWM_Dir = V_max;
  }if (valorMotor_PWM_Dir<V_min){
    valorMotor_PWM_Dir = V_min;
  }
  //atualiza a velocidade dos motores
  analogWrite(pinMotor1PWM,int(valorMotor_PWM_Esq)); //MOTOR E PWM
  analogWrite(pinMotor2PWM,int(valorMotor_PWM_Dir)); //MOTOR D PWM
  Serial.print("  M_Esq= ");
  Serial.print(valorMotor_PWM_Esq);
  Serial.print(" M_Dir= ");
  Serial.println(valorMotor_PWM_Dir);
// fim de ATUALIZA VALOR DOS ATUADORES  --------------------
  delay(50);
}
```

Carrinho PID de distância – códido parte 8

- Ainda na sexta parte do código temos a função para inicializar os motores;

```
void inicializa_Motor_Shield(){
    pinMode(pinSH_CP, OUTPUT);
    pinMode(pinST_CP, OUTPUT);
    pinMode(pinEnable, OUTPUT);
    pinMode(pinDS, OUTPUT);

    pinMode(pinMotor1PWM, OUTPUT);
    pinMode(pinMotor2PWM, OUTPUT);
//    pinMode(pinMotor3PWM, OUTPUT);
//    pinMode(pinMotor4PWM, OUTPUT);
    digitalWrite(pinEnable, LOW);
}
```

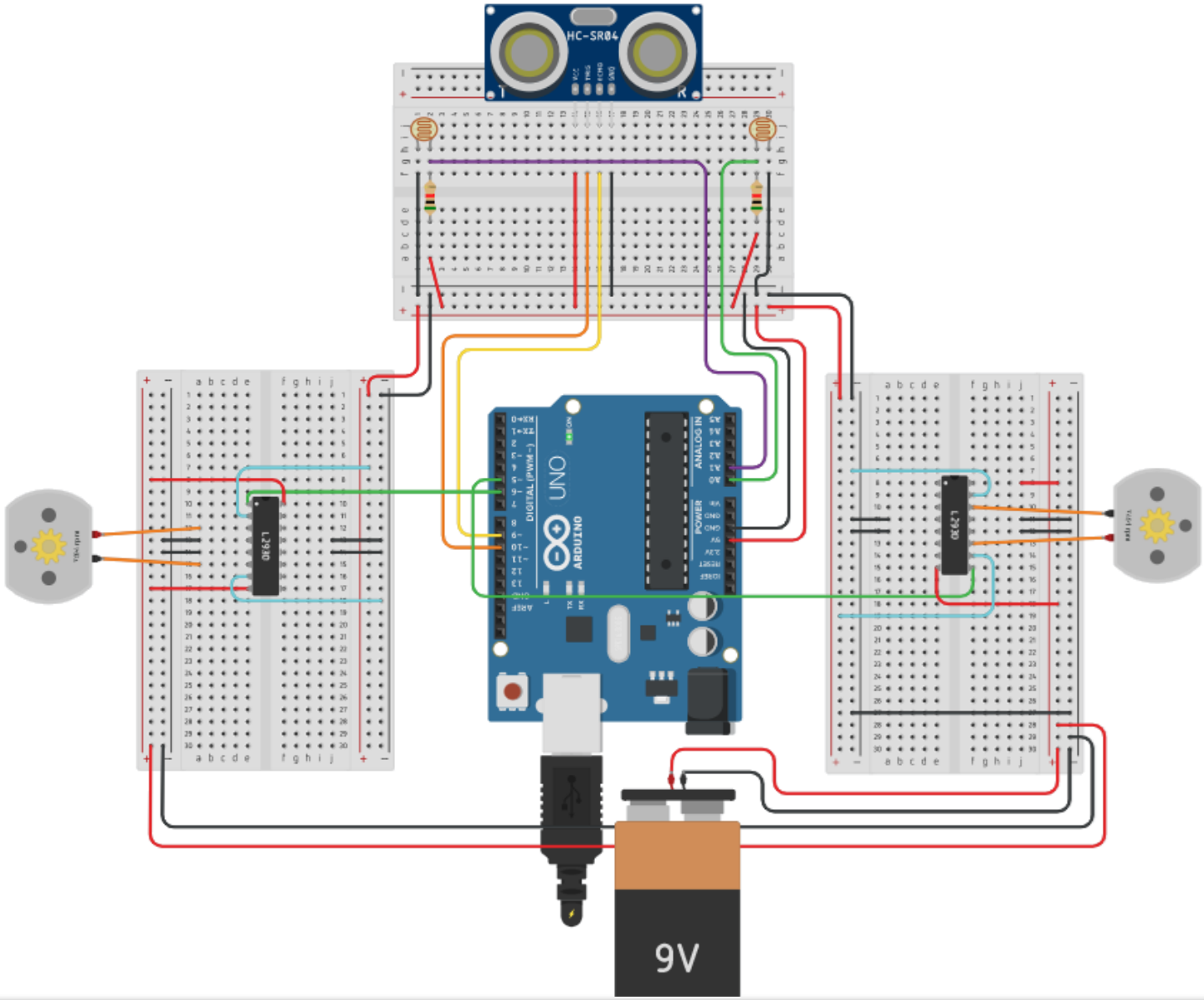
Carrinho PID de distância – códido parte 9

- A última função na sexta parte do código faz a comunicação com o shield controlador de motor para dizer a dizer a direção de rotação do motor;

```
void inicializa_Motor_Shield(){
     pinMode(pinSH_CP, OUTPUT);
     pinMode(pinST_CP, OUTPUT);
     pinMode(pinEnable, OUTPUT);
     pinMode(pinDS, OUTPUT);

     pinMode(pinMotor1PWM, OUTPUT);
     pinMode(pinMotor2PWM, OUTPUT);
//     pinMode(pinMotor3PWM, OUTPUT);
//     pinMode(pinMotor4PWM, OUTPUT);
     digitalWrite(pinEnable, LOW);
}
```

Para simular em casa:
Carrinho seguidor de linha – eletrônica



Para simular em casa:
Carrinho seguidor de linha – códido parte 1

- A primeira parte do código contém:
  - define
  - declarações de variáveis globais

```
#define EN_dir 5
#define EN_esq 6
#define trigPin 10
#define echoPin 9
#define sensorDireita A0
#define sensorEsquerda A1
#define V0 120
#define V_DEVAGAR 50
#define offset 94
#define V_max 240
#define V_min 0

float velE,velD;
int leituraEsquerda, leituraDireita;
float distancia;
float erro,integral,der,erro_anterior,PID;
float Kp,Ki,Kd;
```

Para simular em casa:
Carrinho seguidor de linha – códido parte 2

- A segunda parte do código contém a função `le_sensor_ultrassonico` que implementa a sequência necessária para emitir a onda, que bate no objeto e volta, medir o tempo necessário para a ida e volta da onda, além de transformar em distância a partir da velocidade do som no ar.

```
float le_sensor_ultrassonico() {
  long duracao;
//Limpa o pino de Trigger
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
//Coloca o pino de Trigger em HIGH por
//10 microseg
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
//Le o pino de echo, retornando o tempo
//em microseg da onda sonora ir e vir
  duracao = pulseIn(echoPin, HIGH);
//Calcula a distancia
  return ((float)duracao)*0.0174;
}
```

Para simular em casa:
Carrinho seguidor de linha – códido parte 3

- A terceira parte do código contém: a função `setup()` com:
  1. inicialização da comunicação serial em 9600kbps;
  2. definição dos pinos usados como entrada ou saída;
  3. Atribuição de valor inicial para cada variável.

```
void setup() {
  Serial.begin(9600);
  Serial.println("Esperando 5 segundos");
  pinMode (sensorEsquerda, INPUT);
  pinMode (sensorDireita, INPUT);
  pinMode (EN_dir, OUTPUT);
  pinMode (EN_esq, OUTPUT);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  velE = V0;
  velD = V0;
  erro = 0;
  integral = 0;
  der = 0;
  erro_anterior = 0;
  Kp = 0.1;
  Kd = 0.0;
  Ki = 0.0;
  Serial.println("Esperando 1 segundos");
  delay(1000);
  Serial.println("Setup completo");
}
```

Para simular em casa:
Carrinho seguidor de linha – códido parte 4

- Na quarta parte do código começa a função loop.
- Primeiro é implementada a leitura dos sensores de linha, com atenção para o offset devido ao valor lido quando o mesmo está totalmente no branco (devido ao divisor resistivo);
- Depois é implementada a leitura dos sensores ultrassônico de distância;

```
void loop() {
// LÊ OS SENSORES
  //Lê o senor de distancia ultrassonico
  leituraDireita =
analogRead(sensorDireita)-offset;
  leituraEsquerda =
analogRead(sensorEsquerda)-offset;
  Serial.print("S_Esq= ");
  Serial.print(leituraEsquerda);
  Serial.print(" S_Dir= ");
  Serial.print(leituraDireita);
  //Lê o senor de distancia ultrassonico
  distancia = le_sensor_ultrassonico();
  Serial.print(" Dist=");
  Serial.print(distancia);
  Serial.print("cm ");
// fim de LÊ OS SENSORES -------
```

Para simular em casa:
Carrinho seguidor de linha – códido parte 5

- Na quinta parte do código continua a função loop.
- Nesta parte é impleme ntado o controle;

```
// CALCULA O CONTROLE  ------------------------
// NESTE controle, vamos assumir
// que virar para a esquerda é positivo
//calcula os termos do PID
  erro = leituraDireita - leituraEsquerda;
  integral = integral + erro;
  der = erro - erro_anterior;
  erro_anterior = erro;
//calcula o valor do PID e a velocidade dos motores
  PID = Kp*erro + Ki*integral + Kd*der;
  velE =(int)(V0 + PID);
  velD =(int)(V0 - PID);
// se estiver muito perto da parede, desacelera ate a distancia limite
  if ((distancia > 10) && (distancia < 20)) {
    velE = V_DEVAGAR;
    velD = V_DEVAGAR;
  }if (distancia < 10) {
    velE = 0;
    velD = 0;
  }
// fim de CALCULA O CONTROLE  ------------------------
```

Para simular em casa:
Carrinho seguidor de linha – códido parte 6

- Na sexta parte do código termina a função loop.
- Nesta parte temos a verificação se os atuadores estão saturados;
- E atualizamos os valores dos atuadores.

```
// ATUALIZA VALOR DOS ATUADORES  -------------
  //verifica saturacao dos motores
  if (velE>V_max){
    velE = V_max;
  }if (velE<V_min){
    velE = V_min;
  }if (velD>V_max){
    velD = V_max;
  }if (velD<V_min){
    velD = V_min;
  }
  //atualiza a velocidade dos motores
  analogWrite(EN_esq,int(velE)); //MOTOR E PWM
  analogWrite(EN_dir,int(velD)); //MOTOR D PWM
  Serial.print("  M_Esq= ");
  Serial.print(velE);
  Serial.print(" M_Dir= ");
  Serial.println(velD);
// fim de ATUALIZA VALOR DOS ATUADORES  ------
  delay(50);
} //fim loop()
```