```
// Define os pinos de utilizao do Arduino Mega
#define       motorA1    2
#define       motorA2    3
#define       motorB1    4
#define       motorB2    5
#define       trig_tras  6
#define       echo_tras  7
#define       trig_esq   8
#define       echo_esq   9
#define       trig_dir   10
#define       echo_dir   11
#define       trig_fren  12
#define       echo_fren  13
#define       BTState    19
//=================Algumas variveis teis=============

float Ref=5.04;
float V_in;
float relacao=22.133;
int amostra=20;

float distcm_fren;
float distcm_dir;
float distcm_esq;
float distcm_tras;
int state_rec;
int vSpeed;
char state;
char hist;

//=============SETUP================================

void setup() {
  vSpeed=255;
  // Inicializa as portas como entrada e sada.
  pinMode(motorA1, OUTPUT);
  pinMode(motorA2, OUTPUT);
  pinMode(motorB1, OUTPUT);
  pinMode(motorB2, OUTPUT);
  pinMode(BTState, INPUT);

  pinMode(trig_tras, OUTPUT);
  pinMode(echo_tras, INPUT);
  pinMode(trig_esq, OUTPUT);
  pinMode(echo_esq, INPUT);
  pinMode(trig_dir, OUTPUT);
  pinMode(echo_dir, INPUT);
  pinMode(trig_fren, OUTPUT);
  pinMode(echo_fren, INPUT);
  // Inicializa a  comunicao  serial em 9600 bits.
  Serial.begin(9600);
  mov_fren();                       // Inicializa o rob a ir para frente
}

//====================LOOP================================
  void loop(){
```

```
  distcm_fren= measureDistance_fren();
  distcm_dir= measureDistance_dir();
  distcm_esq= measureDistance_esq();
  distcm_tras= measureDistance_tras(); //armazena os valores de distancia dos ultrassons

  Serial.print("dist frente: ");
  Serial.print(distcm_fren);
  Serial.println("\t");

  Serial.print("dist atras: ");
  Serial.print(distcm_tras);
  Serial.println("\t");

  Serial.print("dist direita: ");
  Serial.print(distcm_dir);
  Serial.println("\t");

  Serial.print("dist esquerda: ");
  Serial.print(distcm_esq);
  Serial.println("\n");              //e printa eles no Serial Monitor


pid_control_fren(distcm_fren);
pid_control_tras( distcm_tras);
pid_control_dir( distcm_dir);
pid_control_esq( distcm_esq); //chama nossas funes de PID

if(distcm_fren>20 && distcm_tras>20 && distcm_esq>20 && distcm_dir>20 ){

/*if(hist=="f") mov_fren;
if(hist=="t") mov_tras;
if(hist=="d") mov_dir;
if(hist=="e") mov_esq;

if(hist=="fd") fren_dir;
if(hist=="fe") fren_esq;
if(hist=="td") tras_dir;
if(hist=="te") tras_esq;*/

switch(hist){
case 'f': mov_fren;break;
case 't':mov_tras;break;
case 'd':mov_dir;break;
case 'e':mov_esq;break;

case 'i':fren_dir;break;
case 'u':fren_esq;break;
case 'm':tras_dir;break;
case 'n':tras_esq;break;

}

}

  }
//=============================================================
```

```
//====================Funes ==============================
// Funes  de  movimentao
void mov_fren(){
    analogWrite(motorB1, vSpeed);
    analogWrite(motorA1, vSpeed);
    analogWrite(motorA2, 0);
    analogWrite(motorB2, 0);
    hist='f';
  }

void mov_tras(){
    analogWrite(motorA1, 0);
    analogWrite(motorB1, 0);
    analogWrite(motorB2, vSpeed);
    analogWrite(motorA2, vSpeed);
    hist='t';
}
void nao_mov(){
    analogWrite(motorA1, 0);
    analogWrite(motorA2, 0);
    analogWrite(motorB1, 0);
    analogWrite(motorB2, 0);
    hist='s';

}
void mov_dir(){
    analogWrite(motorA1, vSpeed);
    analogWrite(motorA2, 0);
    analogWrite(motorB1, 0);
    analogWrite(motorB2, vSpeed);
    hist='d';

}
void mov_esq(){
    analogWrite(motorA1, 0);
    analogWrite(motorA2, vSpeed);
    analogWrite(motorB1, vSpeed);
    analogWrite(motorB2, 0);
    hist='e';
}
void tras_dir(){
    analogWrite(motorA1, 0);
    analogWrite(motorA2, vSpeed/2);
    analogWrite(motorB1, 0);
    analogWrite(motorB2, vSpeed);
    hist='m';
}
void tras_esq(){

    analogWrite(motorA1, 0);
    analogWrite(motorA2, vSpeed);
    analogWrite(motorB1, 0);
    analogWrite(motorB2, vSpeed/2);
    hist='n';
}
void fren_dir(){
```

```
    analogWrite(motorA1, vSpeed/2);
    analogWrite(motorA2, 0);
    analogWrite(motorB1, vSpeed);
    analogWrite(motorB2, 0);
    hist='i';
}
void fren_esq(){

    analogWrite(motorA1, vSpeed);
    analogWrite(motorA2, 0);
    analogWrite(motorB1, vSpeed/2);
    analogWrite(motorB2, 0);
    hist='u';
}



//======== Funes  referente aos ultrassons======

void trigPulse_fren()                          // Funo  para gerar o pulso de trigger para
    o sensor HC-SR04
{

   digitalWrite(trig_fren,HIGH);                   // Sada de trigger em nvel alto
   delayMicroseconds(10);                     //Por 10 s ...
   digitalWrite(trig_fren,LOW);                    // Sada de trigger volta a nvel baixo

} //end trigPulse
void trigPulse_dir()                           // Funo  para gerar o pulso de trigger para
    o sensor HC-SR04
{

   digitalWrite(trig_dir,HIGH);                   // Sada de trigger em nvel alto
   delayMicroseconds(10);                     //Por 10 s ...
   digitalWrite(trig_dir,LOW);                    // Sada de trigger volta a nvel baixo

} //end trigPulse
void trigPulse_esq()                           // Funo  para gerar o pulso de trigger para
    o sensor HC-SR04
{

   digitalWrite(trig_esq,HIGH);                   // Sada de trigger em nvel alto
   delayMicroseconds(10);                     //Por 10 s ...
   digitalWrite(trig_esq,LOW);                    // Sada de trigger volta a nvel baixo

} //end trigPulse
void trigPulse_tras()                          // Funo  para gerar o pulso de trigger para
    o sensor HC-SR04
{

   digitalWrite(trig_tras,HIGH);                   // Sada de trigger em nvel alto
   delayMicroseconds(10);                     //Por 10 s ...
   digitalWrite(trig_tras,LOW);                    // Sada de trigger volta a nvel baixo

} //end trigPulse
```

```
float measureDistance_fren()                    // Funo  que retorna a distncia em
    centmetros
{

  float pulse_fren;                             //Armazena o valor de tempo em s que o pino
      echo fica em nvel alto

  trigPulse_fren();                             //Envia pulso de 10s para o pino de trigger
      do sensor

  pulse_fren = pulseIn(echo_fren, HIGH);            //Mede o tempo em que echo fica em nvel
      alto e armazena em pulse

  /*
    >>> Clculo da Converso de s para cm:

   Velocidade do som = 340 m/s = 34000 cm/s

   1 segundo = 1000000 micro segundos

      1000000 s - 34000 cm/s
           X  s - 1 cm

                1E6
           X = ------- = 29.41
               34000

    Para compensar o ECHO (ida e volta do ultrassom) multiplica-se por 2

    X' = 29.41 x 2 = 58.82

  */

  return (pulse_fren/58.82);                    //Calcula distncia em centmetros e retorna o
      valor


} //end measureDistante
float measureDistance_dir()                     // Funo  que retorna a distncia em centmetros
{

  float pulse_dir;                              //Armazena o valor de tempo em s que o pino
      echo fica em nvel alto

  trigPulse_dir();                              //Envia pulso de 10s para o pino de trigger
      do sensor

  pulse_dir = pulseIn(echo_dir, HIGH);              //Mede o tempo em que echo fica em nvel
      alto e armazena em pulse



  return (pulse_dir/58.82);                     //Calcula distncia em centmetros e retorna o
      valor
```

```
} //end measureDistante

float measureDistance_esq()                    // Funo  que retorna a distncia em centmetros
{

  float pulse_esq;                             //Armazena o valor de tempo em s que o pino
      echo fica em nvel alto

  trigPulse_esq();                             //Envia pulso de 10s para o pino de trigger
      do sensor

  pulse_esq = pulseIn(echo_esq, HIGH);            //Mede o tempo em que echo fica em nvel
      alto e armazena em pulse



  return (pulse_esq/58.82);                    //Calcula distncia em centmetros e retorna o
      valor


} //end measureDistante

float measureDistance_tras()                    // Funo  que retorna a distncia em
    centmetros
{

  float pulse_tras;                            //Armazena o valor de tempo em s que o pino
      echo fica em nvel alto

  trigPulse_tras();                            //Envia pulso de 10s para o pino de trigger
      do sensor

  pulse_tras = pulseIn(echo_tras, HIGH);             //Mede o tempo em que echo fica em nvel
      alto e armazena em pulse



  return (pulse_tras/58.82);                   //Calcula distncia em centmetros e retorna o
      valor


} //end measureDistante



//============= Funo  PID=================

void pid_control_fren(float measure)            // Funo  para algorimo PID
{

  float   error_meas,                       //armazena o erro
          kp = 1.0,                         //constante kp
          ki = 0.02,                        //constante ki
          kd = 0.0,                         //constante kd
```

```
        proportional,                       //armazena valor proporcional
        integral,                           //armazena valor integral
        derivative,                         //armazena valor derivativo
        PID,                                //armazena resultado PID
        ideal_value = 20.0,                 //valor ideal (setpoint), setado para 20cm
        lastMeasure;                        //armazena ltima medida



   error_meas = measure - ideal_value;      //calcula erro

   proportional = error_meas * kp;          //calcula proporcional

   integral += error_meas * ki;             //calcula integral

   derivative = (lastMeasure - measure) * kd; //calcula derivada ***h um erro de semntica aqui
                                            // ex.: corrija o erro de semntica para o clculo
                                                da derivada!
   lastMeasure = measure;                   //atualiza medida

   PID = proportional + integral + derivative; //calcula PID


   if(PID < 0)                              //PID menor que zero?
   {                                        //sim
     PID = map(PID, 0, -20, 60, 255);
     vSpeed=PID;
     //normaliza para PWM de 8 bits
     mov_tras();                          //move  rob  para  trs

   } //end if PID



} //end pid_control

//------------------------------controle quando o robo ta quase batendo
    atras---------------------------------------
void pid_control_tras(float measure)             //  Funo  para algorimo PID
{

  float   error_meas,                       //armazena o erro
        kp = 1.0,                           //constante kp
        ki = 0.02,                          //constante ki
        kd = 0.0,                           //constante kd
        proportional,                       //armazena valor proporcional
        integral,                           //armazena valor integral
        derivative,                         //armazena valor derivativo
        PID,                                //armazena resultado PID
        ideal_value = 20.0,                 //valor ideal (setpoint), setado para 20cm
        lastMeasure;                        //armazena ltima medida



   error_meas = measure - ideal_value;      //calcula erro
```

```
    proportional = error_meas * kp;          //calcula proporcional

    integral += error_meas * ki;             //calcula integral

    derivative = (lastMeasure - measure) * kd; //calcula derivada ***h um erro de semntica aqui
                                             // ex.: corrija o erro de semntica para o clculo
                                                 da derivada!
    lastMeasure = measure;                   //atualiza medida

    PID = proportional + integral + derivative; //calcula PID


    if(PID < 0)                              //PID menor que zero?
    {                                        //sim
      PID = map(PID, 0, -20, 60, 255);
      vSpeed=PID;
      //normaliza para PWM de 8 bits
      mov_fren();                           //move  rob  para  trs

    } //end if PID




} //end pid_control

//------------------------------controle quando o robo ta quase batendo
    dir--------------------------------------

void pid_control_dir(float measure)             // Funo  para algorimo PID
{

  float   error_meas,                        //armazena o erro
          kp = 1.0,                          //constante kp
          ki = 0.02,                         //constante ki
          kd = 0.0,                          //constante kd
          proportional,                      //armazena valor proporcional
          integral,                          //armazena valor integral
          derivative,                        //armazena valor derivativo
          PID,                               //armazena resultado PID
          ideal_value = 20.0,                //valor ideal (setpoint), setado para 20cm
          lastMeasure;                       //armazena ltima medida



    error_meas = measure - ideal_value;      //calcula erro

    proportional = error_meas * kp;          //calcula proporcional

    integral += error_meas * ki;             //calcula integral

    derivative = (lastMeasure - measure) * kd; //calcula derivada ***h um erro de semntica aqui
                                             // ex.: corrija o erro de semntica para o clculo
                                                 da derivada!
```

```
    lastMeasure = measure;                    //atualiza medida

    PID = proportional + integral + derivative; //calcula PID


    if(PID < 0)                               //PID menor que zero?
    {                                         //sim
      PID = map(PID, 0, -20, 60, 255);
      vSpeed=PID;
      if(hist='t'||hist=='n'){
      fren_dir();
      delay(100);
      mov_tras();
      }
      if(hist='f'||hist=='m'){
      fren_dir();
      delay(100);
      mov_fren();
      }
    } //end if PID




} //end pid_control


//------------------------------controle quando o robo ta quase batendo
    esquerda--------------------------------------


void pid_control_esq(float measure)             // Funo  para algorimo PID
{

  float   error_meas,                         //armazena o erro
          kp = 1.0,                           //constante kp
          ki = 0.02,                          //constante ki
          kd = 0.0,                           //constante kd
          proportional,                       //armazena valor proporcional
          integral,                           //armazena valor integral
          derivative,                         //armazena valor derivativo
          PID,                                //armazena resultado PID
          ideal_value = 20.0,                 //valor ideal (setpoint), setado para 20cm
          lastMeasure;                        //armazena ltima medida


    error_meas = measure - ideal_value;       //calcula erro

    proportional = error_meas * kp;           //calcula proporcional

    integral += error_meas * ki;              //calcula integral
```

```
    derivative = (lastMeasure - measure) * kd; //calcula derivada ***h um erro de semntica aqui
                                               // ex.: corrija o erro de semntica para o clculo
                                                   da derivada!
    lastMeasure = measure;                     //atualiza medida

    PID = proportional + integral + derivative; //calcula PID


    if(PID < 0)                                //PID menor que zero?
    {                                          //sim
      PID = map(PID, 0, -20, 60, 255);
      vSpeed=PID;
      //normaliza para PWM de 8 bits
      if(hist='t'||hist=='i'){
      fren_esq();
      delay(100);
      mov_tras();
      }
      if(hist='f'||hist=='u'){
      fren_esq();
      delay(100);
      mov_fren();
      }                          //move rob para trs

    } //end if PID

} //end pid_control
```