INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE PERNAMBUCO

CURSO TÉCNICO INTEGRADO EM MECATRÔNICA

PROJETO DE SEGUIDOR DE LINHA

Asafe dos Santos Silva

Daniel Queiroz Moraes Resende

Gabriel Tabosa da Silva

CARUARU

2016

Asafe dos Santos Silva
Daniel Queiroz Moraes Resende
Gabriel Tabosa da Silva

PROJETO DE SEGUIDOR DE LINHA

Relatório apresentado ao Professor Me. Eduardo Pereira, da disciplina de Sistemas Microcontrolados, do Curso Técnico Integrado em Mecatrônica do Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco, *Campus* Caruaru, como pré-requisito parcial para a conclusão do VII Período.

CARUARU
2016

SUMÁRIO

1 MICROCONTROLADOR

O Microcontrolador utilizado neste projeto está num sistema embarcado chamado de NXT fabricado pela *Lego Mindstroms*. O NXT parece um brinquedo e é muito utilizado desde escolas a universidades. Como pode ser visto na figura 1, o kit é composto de diversos componentes, como por exemplo, servomotores, sensores (luz, toque, som e ultrassom), bateria, rodas e peças que auxiliam nas montagens, como blocos, vigas, engrenagens, polias entre outros.

Figura 1. Kit NXT *Lego Mindstroms*.

O *Brick* NXT (Figura 2) é o cérebro do conjunto, onde está o microcontrolador (da marca *ATMEL*) e todo o circuito necessário para uso. Este kit permite que a criação de estruturas que são muito utilizados no aprendizado de conceitos básicos de ciência e engenharia, abrangendo as áreas de robóticas, controle, automação, física, matemática, programação e projetos.

Figura 2. *Brick NXT.*

2 SENSORES

Resumidamente, sensores são dispositivos capazes de responder a estímulos da natureza física, como umidade, luminosidade, temperatura, pressão e etc. No kit da Lego citado na seção anterior, estão presentes sensores de luminosidade, som, distância (ultrassom) e sensor de toque. No trabalho em questão serão usados somente os sensores de luminosidade (Figura 3).

O sensor de luz permite que o robô diferencie intensidades **entre** claro e escuro. Ou seja, qualquer valor intermediário entre o branco e preto, ele responde com um nível de intensidade proporcional.

Figura 3. Sensor de luminosidade.

3 LÓGICA GERAL

Para desenvolver toda a lógica do nosso seguidor de linha, foram necessários vários dias analisando os padrões que os obstáculos possuíam e tentando “encaixar” diversas disposições e quantidades de sensores de luz. Vale salientar que o robô não foi projeto para qualquer seguidor de linha. Este seguidor de linha se baseou nas regras da Olimpíada Brasileira de Robótica (OBR) 2015/2016, portanto qualquer outro obstáculo diferente da regra que o robô for submetido não terá bom desempenho.

Após várias discussões entre a equipe, observou que a melhor disposição dos sensores, para a nossa lógica de programação, é a mostrada na figura 4.

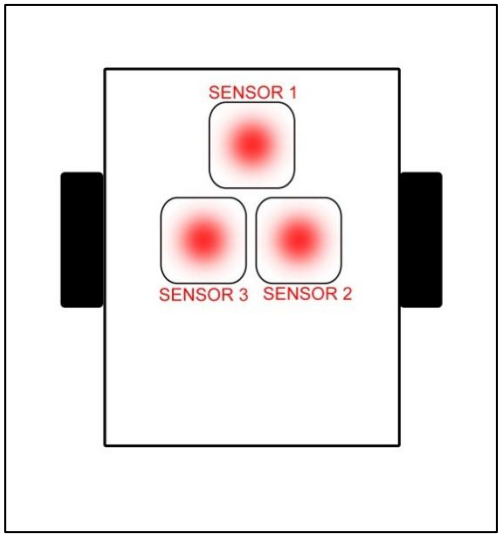


Figura 4. Esquema de disposição dos sensores em relação ao robô.

Para analisar os padrões foi empregado o método de desenhar todos os possíveis “obstáculos” e desenhar os sensores em cima dos mesmos e observar o que os sensores “veem”. Neste relatório os sensores 1, 2 e 3 da figura 4 serão interpretados como SLF (sensor de luz da frente), SLD (sensor de luz da direita) e SLE (sensor de luz da esquerda) respectivamente.

De acordo com a regra da OBR, as linhas poderão formar encruzilhadas e círculos. Encruzilhadas contêm uma marcação em fita verde na intersecção que indica a direção que o robô deverá seguir. O robô quando encontrar uma encruzilhada deve seguir pelo caminho indicado pela marcação verde, que pode indicar um caminho à direita ou à esquerda. A figura 5 apresenta opções de caminho a serem seguidos nestes casos.

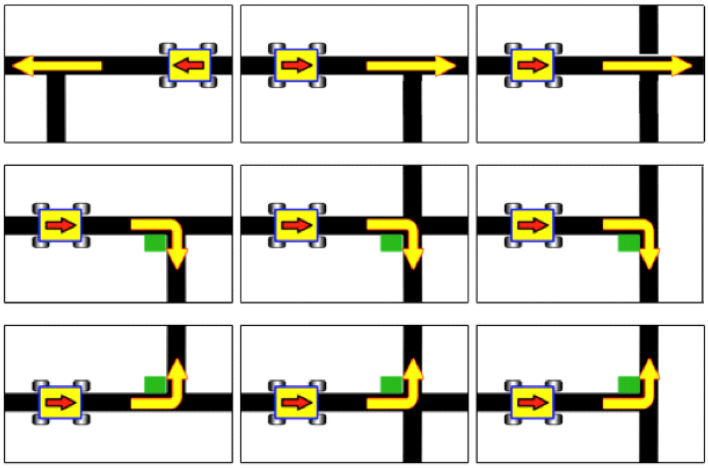


Figura 5. Caminhos obrigatórios que o robô deve seguir ao encontrar uma encruzilhada.

Outro obstáculo que também existe são os Gap's. Estes simulam situações onde o robô não consegue distinguir o caminho a ser seguido. Isto é feito com uma descontinuidade na linha preta como pode ser visto na Figura 6.

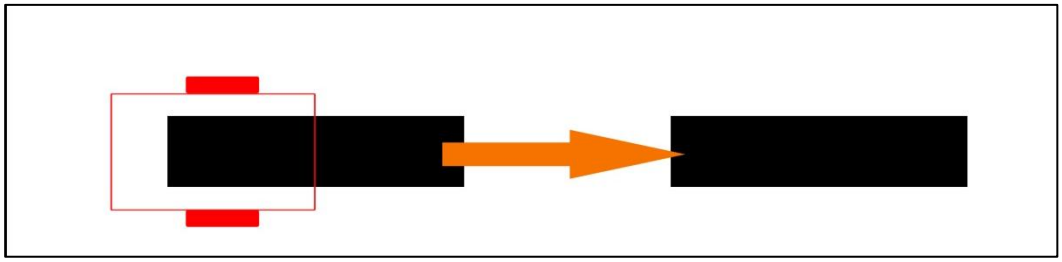


Figura 6. Representação dos Gap's que devem ser ultrapassados pelo robô.

Após o entendimento de todos os obstáculos que podem existir, foi feito o reconhecimento dos padrões que cada um dos três sensores "via" em cada situação. Na figura 7, pode ser visto o robô posicionado em cada um dos obstáculos.

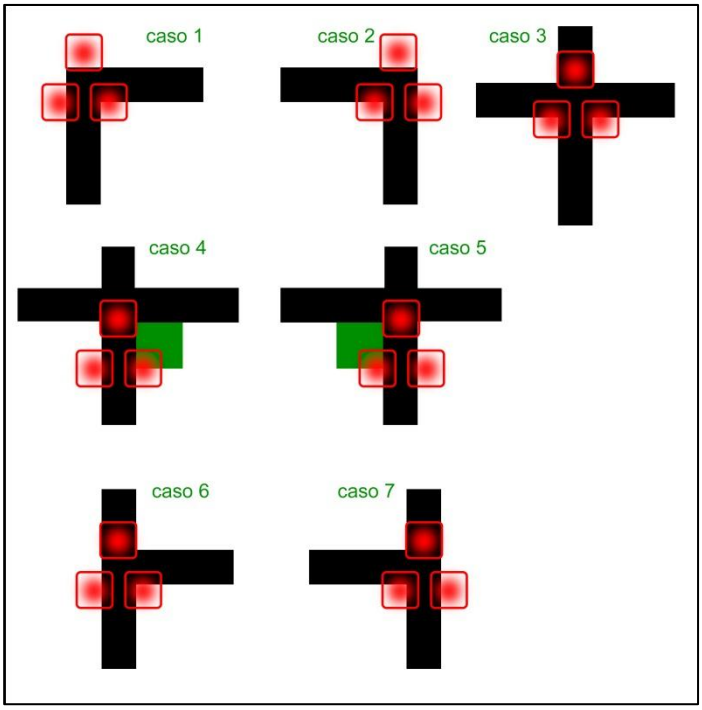


Figura 7. Sensores posicionados na entrada de todos os obstáculos.

Os sensores de luz passam por um conversor analógico/digital (ADC) de 10 bits, portanto retornam um valor que varia entre 0 a 1023 (2^{10}). Desse modo, devido a propriedade das cores, quando o sensor está posicionado no preto ele retorna um valor mínimo, não necessariamente 0, e quando está posicionado no branco retorna um valor máximo, que também não é necessariamente 1023. Portanto qualquer cor intermediária retornará também um valor intermediário entre o mínimo e o máximo. Pensando assim, para maior entendimento, pode-se traduzir estes valores como porcentagem, sendo 0% o mínimo (preto) e 100% o máximo (branco).

4 ANÁLISES E FUNÇÕES MATEMÁTICAS

Nesta seção, serão analisadas as funções matemáticas que resolvem cada um dos obstáculos.

4.1 CALIBRAÇÃO

A rotina de calibração consiste em efetuar alguns movimentos que fazem com que o robô se mova entre o preto e o branco, para que os valores máximos e mínimos

dos três sensores sejam armazenados em variáveis. A rotina a seguir mostra como fizemos para o robô se autocalibrar.

```
/********************* MOVIMENTO PARA CALIBRAÇÃO  **********************/
int sinal;
byte conte;
void motorRunCalibrate(){
  OnFwd(OUT_AB,sinal*40);
  if(sinal > 0){
     if(MotorRotationCount(OUT_A) > 300){
        sinal = -sinal;
        Off(OUT_AB);
     }
  }else{
     if(MotorRotationCount(OUT_A) < 0){
        conte++;
        sinal = -sinal;
        Off(OUT_AB);
     }
  }
}


/********************* FUNÇÃO PARA CALIBRAÇÃO DOS SENSORES  **********************/
void calibrate(){
    sinal = 1;
    conte = 0;
    do{
        read_sensors();
        motorRunCalibrate();

        if(value < min)      min  = value;
        if(value > max)      max = value;

    }while(conte < 1);
}
```

No nosso caso era necessário realizar três vezes a função '*calibrate()*', cada uma usando as variáveis de cada sensor e sempre "zerando" as variáveis '*sinal*' e '*conte'*. Nos anexos pode ser visto a programação completa de calibração.

4.2 SEGUIDOR DE LINHA

A nossa concepção para seguir a linha era usar os dois sensores trás, porém um controlando um motor individualmente, os sensores 2 e 3 controlando os motores da direita e esquerda respectivamente. No caso, pensou-se em uma função que quando os sensores estivessem no máximo ambos os motores aplicassem força "máxima" para naturalmente resolver o *gap.* Portanto, a função aplicada aos motores é mostrada na figura 8.

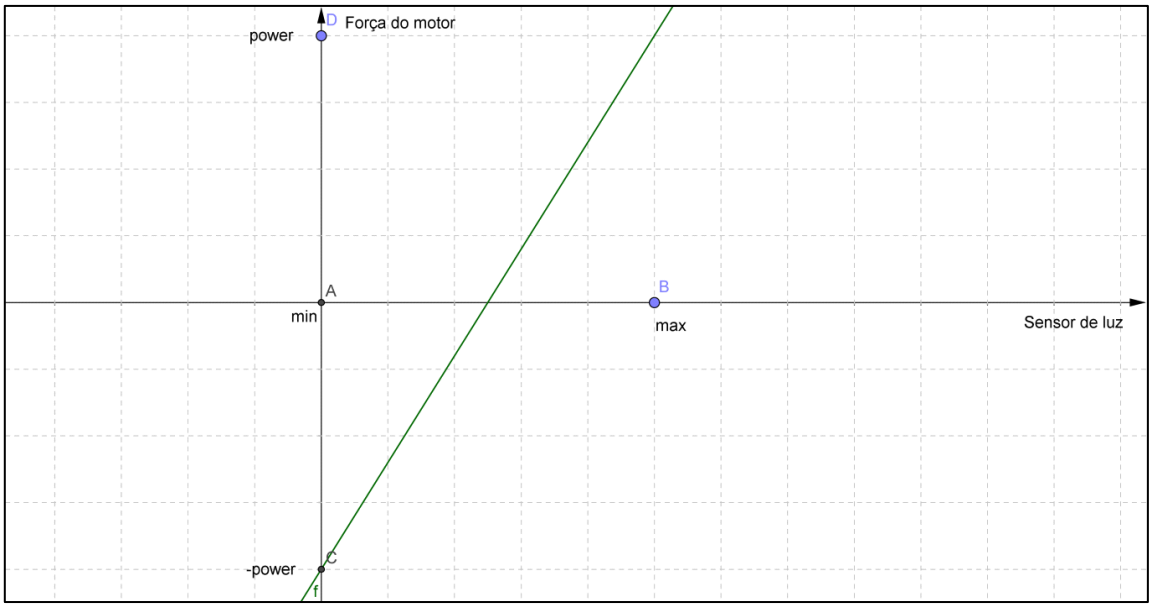


Figura 8. Função usada no seguidor de linha.

Percebe-se que quando o sensor está próximo ao máximo (branco) o motor está também com o máximo da força escolhida. E como no *gap* ambos os sensores estão "vendo" máximo, logo os motores também terão força máxima, e consequentemente o robô vai para frente. Mas para programar essa função é necessário obter uma equação, relacionando o valor do sensor, a força do motor e os valores mínimos e máximos. Temos que, $y = ax + b$, portanto:

$$power = a * max + b \ (1)$$

$$-power = a * min + b \ (2)$$

Substituindo (1) em (2):

$$-power = a * min + (power - a * \max)$$

$$-2 * power = a(min - \max)$$

$$a = -2 * \frac{power}{min - max} \ (3)$$

Substituindo (3) em (1):

$$power = \left(-2 * \frac{power}{min - max}\right) * max + b$$

$$b = power + \frac{2 * power * \max}{min - max} (4)$$

Agora substituindo (3) e (4) em ($y = ax + b$), temos que:

$$y = -\frac{2 * power}{min - max} * x + power + \frac{2 * power * \max}{min - max}$$

$$y = \frac{-2 * power * x + 2 * power * \max}{min - max} + power$$

Portanto:

$$y = 2 * \frac{power(-x + \max)}{min - max} + power$$

Agora, a equação anterior é a que relaciona todas as variáveis que o robô precisa; a força do motor, o valor do sensor, o valor máximo de força e os valores de calibração, onde ***y*** corresponde à força do motor e ***x*** o valor retornado pelo sensor. A partir disso a programação para o seguidor de linha linear pode ser feita com poucas linhas de código, como pode ser visto a seguir.

```
/******************* FUNÇÃO PARA SEGUIR A LINHA ***********************/
void seg_linha(int power){

  // aplica função de seguir linha nos motores
  powerA =  ( (2*power*(maxSLD-valueSLD)) / (minSLD-maxSLD) ) + power;
  powerB = ( (2*power*(maxSLE-valueSLE)) / (minSLE-maxSLE) ) + power;

  OnFwd(motorA, powerA);
  OnFwd(motorB, powerB);
}
```

Para os demais obstáculos, é necessário checar se o robô realmente está neles. Observando a figura 7, pode perceber o que cada sensor ler em cada obstáculo. Por exemplo, no caso 1, o SLD ler um valor abaixo de 50%, já o SLE e SLF “enxergam” valores maiores que 50%. Diferentemente do caso 4, onde o SLD e o SLF “veem” valores menor que 50% e o SLE “veem” valores na faixa de 50%. Desta forma, na programação é possível distinguir cada uma das encruzilhadas presente na arena.

Para melhor entendimento, a seguir é mostrado um *pseudo-código* para resolver os obstáculos, tanto para direita quanto para esquerda, relacionando com os casos da figura 7, que serão explicados nas seções posteriores.

PSEUDO-CÓDIGO (CURVAS PARA DIREITA):

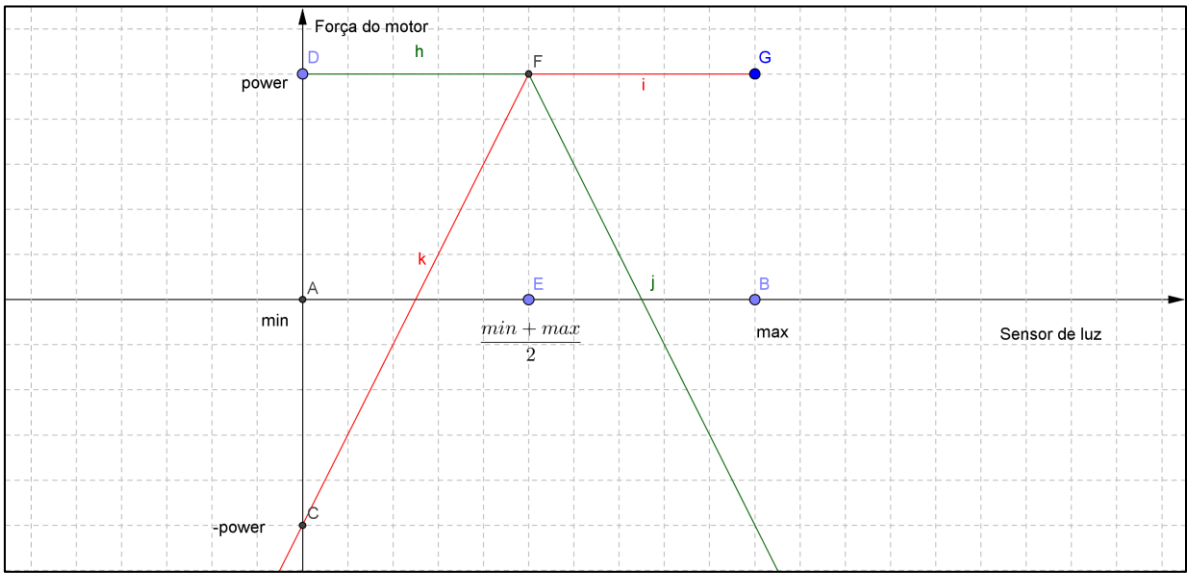
```
preto na direita e branco na esquerda?
sim:
  sensor da frente preto?
  sim:
    anda pra frente.
    sensor direito preto(verde)?
    sim:
      caso 4.
    não:
      caso 6.
  não:
    caso 1.
```

PSEUDO-CÓDIGO (CURVAS PARA ESQUERDA):

```
preto na esquerda e branco na direita?
sim:
  sensor da frente preto?
  sim:
    anda pra frente.
    sensor esquerdo preto(verde)?
    sim:
      caso 5.
    não:
      caso 7.
  não:
    caso 2.
```

4.3 CURVAS DE 90 GRAUS

Como já citado anteriormente o que diferencia as curvas de 90° com verde e sem verde, é basicamente o SLF. Portanto a lógica implementada foi a já citada no pseudo-código, que checa se quando perceber alguma curva (SLD = “preto” e SLE = “branco” ou SLD = “branco” e SLE = “preto”) o sensor da frete é “preto” ou “branco”; se for “branco”, o robô segue linha aprenas com o sensor que “viu branco”, até voltar a “ver preto”. Essa ação de seguir com um sensor, faz com que o robô faça uma curva no formato de tanque, com os motores em sentido contrário, devido a função aplicada a eles mostrada na figura 9[1].



[1] Curva para direita -> Em verde: Motor da direita; Em vermelho: Motor da esquerda.
Curva para esquerda -> Em verde: Motor da esquerda; Em vermelho: Motor da direita.

Figura 9. Função modular para seguir com um só sensor.

Para facilitar o entendimento de como é aplicada essa função, é mostrado um pseudo-código a seguir para curva à direita.

```
se(sensor > 50%){
        motorB = constante;
        motorA = função;
senao{
        motorA = função;
        motorB = constante;
}
```

Para encontrar a equação das funções usa-se o mesmo método que no tópico anterior. Antes de tudo, o valor de $\frac{\min+\max}{2}$ será abreviado para sp (*Setpoint*). Primeiramente vamos encontrar a equação da função em vermelho sabendo que $y = ax + b$, temos que:

$$-power = a * min + b(1)$$

$$power = a * sp + b(2)$$

Isolando *b* de (1) e substituindo em (2):

$$power = a * sp - power - a * min$$

$$a = 2 * \frac{power}{sp - min}$$

Agora substituindo em (2):

$$power = \frac{2 * power}{sp - \min} * sp + b$$

$$b = power - \frac{2 * power}{sp - \min} * sp$$

Assim, obtém-se a equação a seguir:

$$y = 2 * \frac{power}{sp - min} * x + power - \frac{2 * power}{sp - \min} * sp$$

$$y = \frac{2 * power}{sp - \min}(x - sp) + power$$

Agora, a partir do mesmo princípio, desenvolvendo para a função em verde:

$$power = a * sp + b \ (1)$$

$$-power = a * max + b \ (2)$$

Isolando *b* de (1) e substituindo em (2):

$$-power = a * max + power - a * sp$$

$$a = -2 * \frac{power}{max - sp}$$

Agora substituindo em (1):

$$power = -2 * \frac{power}{max - sp} * sp + b$$

$$b = 2 * \frac{power}{max - sp} * sp + power$$

Substituindo os termos a e b na equação $y = ax + b$, tem-se que:

$$y = -2 * \frac{power}{max - sp} * x + 2 * \frac{power}{max - sp} * sp + power$$

Logo:

$$y = \frac{2 * power}{max - sp}(sp - x) + power$$

Após realizar todos os cálculos para encontrar uma equação para tal função, já é possível converte-lo em programação. Porém antes de mostrar a programação, já se pode pensar sobre as curvas de 90 graus que contém verde, e as encruzilhadas que só contém um pedaço de fita à direita ou à esquerda, para um lugar que não leva a nada.

Como já citado anteriormente, o que diferencia as curvas com e sem verde, é o sensor da frente. Pois quando com verde, o sensor está “vendo” preto. Portanto, o código deve verificar se o robô entrou em alguma curva (SLD = “preto” e SLE = “branco” ou SLD = “branco” e SLE = “preto”); se ao entrar na curva o sensor da frente for “preto”, ele já sabe que não é curva sem verde. Porém existem dois casos que isso acontece. Nas curvas com verde, e nas encruzilhadas que não levam a lugar algum. Assim sendo, como visto na seção 4.2, a lógica utilizada para resolver este caso, foi fazer com que o robô, nessas situações, se

locomovesse um pouco para frente, o suficiente para ultrapassar a encruzilhada, porém não interferindo se estiver na curva com verde. Portanto, é necessário, após locomover-se um pouco, checar qual luminosidade que o sensor da direita (no caso de curva à direita) está lendo. Caso esteja medindo luminosidade maior, é porque era uma encruzilhada e o robô deve continuar seguindo linha. Mas se ele estiver vendo uma luminosidade maior, é porque é uma curva com verde. Esta lógica pode ser entendida melhor visualizando a figura 10.

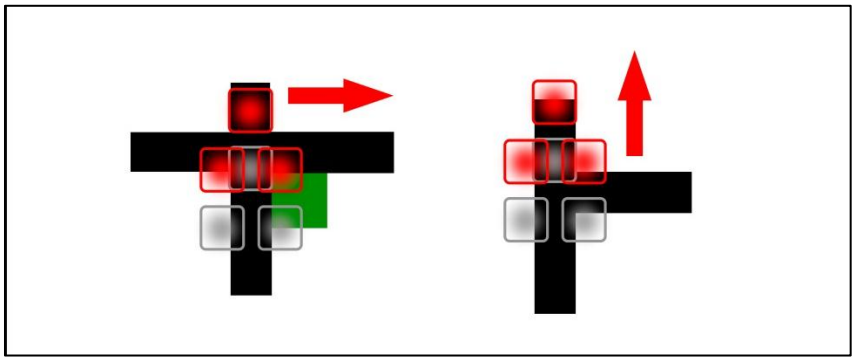


Figura 10. Apresentação da lógica das curvas e encruzilhadas.

No caso de ser curva com verde, a lógica implementada foi aplicar uma função de seguir linha, com o sensor da direita (para curva à direita). Esta função é a mesma que descrita anteriormente, porém não deteremos mais tempo desenvolvendo a equação matemática. Com toda a lógica descrita, a seguir é apresentada a programação contendo a parte de curvas de 90 graus e encruzilhadas à direita quanto à esquerda.

```
/******************* FUNÇÃO PARA TRATAR 'CURVAS' E 'T(s)' PELA DIREITA **********************/

void seg_SLF_direita(int power){

    /*
      PSEUDO-CÓDIGO:

      preto na direita e branco na esquerda?
      sim:
        sensor da frente preto?
        sim:
          anda pra frente.
          sensor direito preto(verde)?
          sim:
            curva verde.
```

```
          não:
            T.
        não:
          curva 90°
      não:
    */

if((valueSLD < (maxSLD-minSLD)*0.3 + minSLD) &&  // preto na direita e branco na esquerda?
 (valueSLE > (maxSLE-minSLE)*0.6 + minSLE)){

      /******************* CURVA VERDE ************************/

 if (valueSLF <= s_pSLF){
   PlayTone(440, 100);

   RotateMotor(motorAB, 50, 73); // 2.3 cm
   Off(motorAB);
   Wait(100);

   read_sensors();
   if (valueSLD < (maxSLD-minSLD)*0.4 + minSLD){ // é verde?

     time0 = CurrentTick();
     do{
       time_atual = CurrentTick() - time0;
       read_sensors();

       // função para seguir com o SLD
        if(valueSLD > s_pSLD){
         powerA = power;
         powerB = (-valueSLD*2 + s_pSLD + maxSLD)*power/(maxSLD-s_pSLD);
       }else{
         powerB = power;
         powerA = (-valueSLD*2 + minSLD + s_pSLD)*power/(minSLD - s_pSLD);
       }

       OnFwd(motorA, powerA);
       OnFwd(motorB, powerB);

   }while( time_atual < 1000 );

 }else if(valueSLD > (maxSLD-minSLD)*0.45 + minSLD){
       /* T */

 }


 /******************* CURVA 90° ************************/
 }else{

   PlayTone(1000, 100);

   do{

     read_sensors();
```

```
      if(valueSLE > s_pSLE){
        powerB = power;
        powerA = (-valueSLE*2 + s_pSLE + maxSLE)*power/(maxSLE-s_pSLE);
      }else{
        powerA = power;
        powerB = (-valueSLE*2  + minSLE + s_pSLE)*power/(minSLE - s_pSLE);
      }

      OnFwd(motorA, powerA);
      OnFwd(motorB, powerB);

    }while( (valueSLF < (maxSLF-minSLF)*0.45) || (valueSLF > (maxSLF-minSLF)*0.55)  );
  }
 }
}


/******************* FUNÇÃO PARA TRATAR 'CURVAS' E 'T(s)' PELA ESQUERDA ***********************/

void seg_SLF_esquerda(int power){

 /*
  PSEUDO-CÓDIGO:

  preto na esquerda e branco na direita?
  sim:
    sensor da frente preto?
    sim:
      anda pra frente.
      sensor esquerdo preto(verde)?
      sim:
        curva verde.
      não:
        T.
    não:
      curva 90°
  não:
 */

 if((valueSLE < (maxSLE-minSLE)*0.3 + minSLE) &&  // preto na esquerda e branco na direita?
    (valueSLD > (maxSLD-minSLD)*0.6 + minSLD)){

  /******************* CURVA VERDE ************************/
  if (valueSLF < s_pSLF){
     PlayTone(440, 100);

    RotateMotor(motorAB, 50, 73); // 2.3 cm
    Off(motorAB);
    Wait(100);

    read_sensors();
    if (valueSLE < (maxSLE-minSLE)*0.4 + minSLE){ // é verde?
        time0 = CurrentTick();
        do{
         time_atual = CurrentTick() - time0;
         read_sensors();
```

```
            if(valueSLE > s_pSLE){
                powerB = power;
                powerA = (-valueSLE*2 + s_pSLE + maxSLE)*power/(maxSLE-s_pSLE);
             }else{
                powerA = power;
                powerB = (-valueSLE*2  + minSLE + s_pSLE)*power/(minSLE - s_pSLE);
             }

                OnFwd(motorA, powerA);
                OnFwd(motorB, powerB);

          }while(time_atual < 1000);

      }else if (valueSLE > (maxSLE-minSLE)*0.45 + minSLE){
        /* T */
      }


     /****************** CURVA 90° ************************/
    }else{

     PlayTone(1000, 100);

     do{

      read_sensors();

        if(valueSLD > s_pSLD){
            powerA = power;
            powerB = (-valueSLD*2 + s_pSLD + maxSLD)*power/(maxSLD-s_pSLD);
        }else{
            powerB = power;
            powerA = (-valueSLD*2  + minSLD + s_pSLD)*power/(minSLD - s_pSLD);
        }

            OnFwd(motorA, powerA);
            OnFwd(motorB, powerB);

      }while( (valueSLF < (maxSLF-minSLF)*0.45) || (valueSLF > (maxSLF-minSLF)*0.55) );
    }
  }
}
```

ANEXO 1 – Explicação da sintaxe.

- Configura tipo e modo do sensor.
SetSensorType(porta, tipo);
SetSensorMode(porta, modo);
Ex.:
SetSensorType(IN_1, SENSOR_TYPE_LIGHT_ACTIVE);
SetSensorMode(IN_1, SENSOR_MODE_RAW);

- Ler valor retornado pelo sensor.
Sensor(porta);
Ex.:
int x = Sensor(IN_1);

- Emite um som em determinada frequência e por determinado tempo (milissegundos).
PlayTone(frequência, duração);
Ex.:
PlayTone(440, 1000);

- Rotaciona motor em determinado ângulo.
RotateMotor(porta, força, graus);
Ex.:
RotateMotor(OUT_A, 75, 90);

- Para motor(es).
Off(porta);
Ex.:
Off(OUT_A);

- Para o programa por determinado tempo (milissegundos).
Wait(tempo);
Ex.:
Wait(1000);

- Retorna o tempo que passou desde a inicialização da programação.
CurrentTick();
Ex.:
int time = CurrentTick();

- Move motor(es) com determinada força.

OnFwd(porta, força);
Ex.:
OnFwd(OUT_AB, 75);

ANEXO 2 – Programação completa.

```
// == DECLARAÇÃO DE VARIÁVEIS == //

#define SLF     IN_1            //Sensor de luz frente na porta 1
#define SLD     IN_2            //Sensor de luz direito na porta 2
#define SLE     IN_3            //Sensor de luz esquerdo na porta 3
#define motorA  OUT_A           // Comando do motor A  (direita)
#define motorB  OUT_B           // Comando do motor B  (esquerda)
#define motorAB OUT_AB          // Comando para os dois motores

unsigned int valueSLF; // guadar valor do sensor de luz frente
unsigned int valueSLD; // guadar valor do sensor de luz direito
unsigned int valueSLE; // guadar valor do sensor de luz esquerdo

int s_pSLF, s_pSLD, s_pSLE;  // valor para guardar setPoint

// váriáveis de calibração
int maxSLD = 0, maxSLF = 0, maxSLE = 0;
int minSLD = 9999, minSLF = 9999, minSLE = 9999;

int powerA, powerB;

/********************* MOVIMENTO PARA CALIBRAÇÃO  **********************/
int sinal;
byte conte;
void motorRunCalibrate(){
  OnFwd(OUT_AB,sinal*40);
  if(sinal > 0){
    if(MotorRotationCount(OUT_A) > 300){
      sinal = -sinal;
      Off(OUT_AB);
    }
  }else{
    if(MotorRotationCount(OUT_A) < 0){
      conte++;
      sinal = -sinal;
      Off(OUT_AB);
    }
```

```
  }
}


/********************* FUNÇÃO PARA CALIBRAÇÃO DOS SENSORES  **********************/
void calibrate(){
    sinal = 1;
    conte = 0;
    do{
        read_sensors();
        motorRunCalibrate();

        if(value < min)      min  = value;
        if(value > max)      max = value;

     }while(conte < 1);
}


/******************** FUNÇÃO PARA SEGUIR A LINHA ************************/
void seg_linha(int power){

  seg_SLF_direita(power);
  seg_SLF_esquerda(power);

  // aplica função de seguir linha nos motores
  powerA =  ( (2*power*(maxSLD-valueSLD)) / (minSLD-maxSLD) ) + power;
  powerB = ( (2*power*(maxSLE-valueSLE)) / (minSLE-maxSLE) ) + power;

  OnFwd(motorA, powerA);
  OnFwd(motorB, powerB);
}

/******************** FUNÇÃO PARA TRATAR 'CURVAS' E 'T(s)' PELA DIREITA ************************/

void seg_SLF_direita(int power){

     /*
       PSEUDO-CÓDIGO:

       preto na direita e branco na esquerda?
       sim:
         sensor da frente preto?
         sim:
           anda pra frente.
           sensor direito preto(verde)?
           sim:
             curva verde.
           não:
             T.
         não:
           curva 90°
       não:
     */

  if((valueSLD < (maxSLD-minSLD)*0.3 + minSLD) &&  // preto na direita e branco na esquerda?
```

```
(valueSLE > (maxSLE-minSLE)*0.6 + minSLE)){

    /******************* CURVA VERDE ************************/

 if (valueSLF <= s_pSLF){
  PlayTone(440, 100);

  RotateMotor(motorAB, 50, 73); // 2.3 cm
  Off(motorAB);
  Wait(100);

  read_sensors();
  if (valueSLD < (maxSLD-minSLD)*0.4 + minSLD){ // é verde?

    time0 = CurrentTick();
    do{
      time_atual = CurrentTick() - time0;
      read_sensors();

      // função para seguir com o SLD
       if(valueSLD > s_pSLD){
        powerA = power;
        powerB = (-valueSLD*2 + s_pSLD + maxSLD)*power/(maxSLD-s_pSLD);
      }else{
        powerB = power;
        powerA = (-valueSLD*2 + minSLD + s_pSLD)*power/(minSLD - s_pSLD);
      }

      OnFwd(motorA, powerA);
      OnFwd(motorB, powerB);

  }while( time_atual < 1000 );

 }else if(valueSLD > (maxSLD-minSLD)*0.45 + minSLD){
      /* T */

 }


 /******************* CURVA 90° ************************/
 }else{

  PlayTone(1000, 100);

  do{

    read_sensors();

    if(valueSLE > s_pSLE){
      powerB = power;
      powerA = (-valueSLE*2 + s_pSLE + maxSLE)*power/(maxSLE-s_pSLE);
    }else{
      powerA = power;
      powerB = (-valueSLE*2  + minSLE + s_pSLE)*power/(minSLE - s_pSLE);
    }
```

```
      OnFwd(motorA, powerA);
      OnFwd(motorB, powerB);

    }while( (valueSLF < (maxSLF-minSLF)*0.45) || (valueSLF > (maxSLF-minSLF)*0.55)  );
   }
  }
}


/******************* FUNÇÃO PARA TRATAR 'CURVAS' E 'T(s)' PELA ESQUERDA **********************/

void seg_SLF_esquerda(int power){

  /*
   PSEUDO-CÓDIGO:

   preto na esquerda e branco na direita?
   sim:
     sensor da frente preto?
     sim:
       anda pra frente.
       sensor esquerdo preto(verde)?
       sim:
         curva verde.
       não:
         T.
     não:
       curva 90°
   não:
  */

  if((valueSLE < (maxSLE-minSLE)*0.3 + minSLE) &&  // preto na esquerda e branco na direita?
     (valueSLD > (maxSLD-minSLD)*0.6 + minSLD)){

   /******************* CURVA VERDE *************************/
   if (valueSLF < s_pSLF){
      PlayTone(440, 100);

     RotateMotor(motorAB, 50, 73); // 2.3 cm
     Off(motorAB);
     Wait(100);

     read_sensors();
     if (valueSLE < (maxSLE-minSLE)*0.4 + minSLE){ // é verde?
         time0 = CurrentTick();
         do{
          time_atual = CurrentTick() - time0;
          read_sensors();

           if(valueSLE > s_pSLE){
              powerB = power;
              powerA = (-valueSLE*2 + s_pSLE + maxSLE)*power/(maxSLE-s_pSLE);
           }else{
              powerA = power;
              powerB = (-valueSLE*2  + minSLE + s_pSLE)*power/(minSLE - s_pSLE);
           }
```

```
            OnFwd(motorA, powerA);
            OnFwd(motorB, powerB);

        }while(time_atual < 1000);

    }else if (valueSLE > (maxSLE-minSLE)*0.45 + minSLE){
      /* T */
    }


    /******************* CURVA 90° ************************/
  }else{

    PlayTone(1000, 100);

    do{

     read_sensors();

       if(valueSLD > s_pSLD){
          powerA = power;
          powerB = (-valueSLD*2 + s_pSLD + maxSLD)*power/(maxSLD-s_pSLD);
       }else{
          powerB = power;
          powerA = (-valueSLD*2  + minSLD + s_pSLD)*power/(minSLD - s_pSLD);
       }

          OnFwd(motorA, powerA);
          OnFwd(motorB, powerB);

    }while( (valueSLF < (maxSLF-minSLF)*0.45) || (valueSLF > (maxSLF-minSLF)*0.55) );
  }
 }
}

/******************** CONFIGURA OS SENSORES  **********************/
void set_sensors(){

  SetSensorType(SLF, SENSOR_TYPE_LIGHT_ACTIVE);
  SetSensorMode(SLF, SENSOR_MODE_RAW);

  SetSensorType(SLD, SENSOR_TYPE_LIGHT_ACTIVE);
  SetSensorMode(SLD, SENSOR_MODE_RAW);

  SetSensorType(SLE, SENSOR_TYPE_LIGHT_ACTIVE);
  SetSensorMode(SLE, SENSOR_MODE_RAW);
}

/******************** LEITURA DOS SENSORES  **********************/
void read_sensors(){
 valueSLD = Sensor(SLD);
 valueSLF = Sensor(SLF);
 valueSLE = Sensor(SLE);
}
```

```
task main(){

        set_sensors();    // CONFIGURA OS SENSORES
        Wait(500);        // espera 500 milisegundos

        calibrate();

         s_pSLF = (minSLF+maxSLF)/2; // set point do SLF
         s_pSLD = (minSLD+maxSLD)/2; // set point do SLD
         s_pSLE = (minSLE+maxSLE)/2; // set point do SLE

         while(true){
                read_sensors();

                seg_linha(80);
         }
}
```