# *Ponto de Controle 4*

## *Robô seguidor de linha para limpezas*

Leonardo Amorim de Araújo
15/0039921
Faculdade do Gama
Universidade de Brasília
St. Leste Projeção A - Gama Leste, Brasília - DF, 72444-240
Email: leonardoaraujodf@gmail.com

Josiane de Sousa Alves
15/0038895
Faculdade do Gama
Universidade de Brasília
St. Leste Projeção A – Gama Leste, Brasília – DF, 72444 – 240
Email: josianealves.18@gmail.com

***RESUMO – Este documento apresenta o ponto de controle III do projeto final para a disciplina de Microprocessadores e Microcontroladores. O projeto visa propor uma solução de um robô seguidor de linha para ser aplicado em limpeza.***

***Palavras-chave — MSP 430; Robô seguidor de linha; Robô Autônomo; Microprocessador;***

### I. INTRODUÇÃO

Robôs autônomos são máquinas inteligentes capazes de realizar tarefas sem controle humano contínuo e explícito sobre seus movimentos. Estes podem sentir e obter informações sobre seus arredores, trabalhar e desviar de obstáculos. Um tipo de robô autônomo é o robô seguidor de linha que é capaz de identificar uma trilha de dimensões definidas e percorrer por ela. Esse tipo de robô pode ser usado para muitas aplicações, dentre elas, na indústria como os Veículos Guiados Automaticamente – AGVs; para filmagens em campos, quadras; guias em estabelecimentos comerciais de grandes proporções, etc.

### II. OBJETIVOS

Construir um robô seguidor de linha programado na Lanchpad MSP 430 com o intuito de realizar tarefas de limpeza industrial.

### III. DESCRIÇÃO

Com o intuito de auxiliar na limpeza industrial, especialmente em locais pequenos, de difícil acesso e locais que ofereçam risco à saúde, será construído um robô seguidor de linha que realiza um trajeto especificado, programado na MSP430. Este robô será capaz de se movimentar para frente, para trás, direita e esquerda e contará com 5 opções de velocidade, com isso o usuário poderá optar por uma limpeza mais superficial ou mais específica, dependo da necessidade no momento da limpeza.

Para que as funções acima funcionem corretamente, é necessário utilizar alguns componentes e funções que serão especificados a seguir.

**HARDWARE**

- **CI L293B – Dual Ponte H**

O CI L293B é um *PUSH-PULL FOUR CHANNEL DRIVER* que consiste de dois drivers de ponte H que controlam até dois motores DC. A ponte H permite que se controle a direção de giro dos motores e a sua velocidade utilizando o microcontrolador. A pinagem do CI é mostrada na Fig. 01.

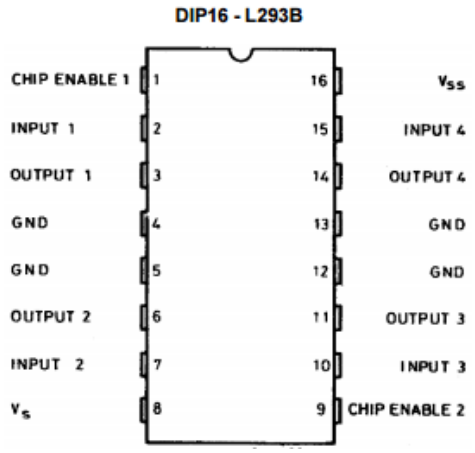


Figura 01. Pinagem do CI L293B

Os pinos 1 e 9 são chamados pinos de ENABLE pois habilitam o funcionamento do motor e devem estar em nível alto para que ocorra o funcionamento. Estes pinos podem ser controlados posteriormente para utilização do PWM para controle de velocidade do robô.

Os pinos 2, 3, 10, e 15 são os pinos que controlam o modo de giro do motor, que são três: direita, esquerda e freio.

A Tabela 01 mostra como deve ser utilizado o CI e os resultados esperados:

Tabela 01. Modo de utilização do CI L293B

| Pino 1 | Pino 2 | Pino 7 | Função |
|---|---|---|---|
| ALTO | BAIXO | ALTO | Girar no sentido horário |
| ALTO | ALTO | BAIXO | Girar no sentido anti-horário |
| ALTO | BAIXO | BAIXO | Parado |
| ALTO | ALTO | ALTO | Parado |
| BAIXO | Sem aplicabilidade | | |

Os pinos apresentados na Tabela 01 controlam o primeiro motor. Para o segundo motor a função é a mesma, mas os pinos 1, 2 e 7 equivalem respectivamente aos pinos 9, 10 e 15. O motor 1 tem seu polo positivo ligado no pino 3 e o negativo no pino 2. O mesmo para o motor 2 que deve ser ligado nos pinos 11 e 14. O pino 16 deve ser conectado no Vcc da MSP e o pino Vs na fonte de tensão do motor. Os GNDs da MSP e da fonte que alimentam o motor são ligados no mesmo nó, assim como os pinos denominados GND no CI.

No projeto, o CI L293B será utilizado para controlar a direção do robô, conforme o acionamento dos sensores, e o modo de limpeza, alternando entre as velocidades dos motores.

- **Sensor Óptico Reflexivo TCRT5000**

Este sensor, cuja a vista superior é mostrada na Fig. 02, é composto por dois componentes que funcionam em conjunto, um fototransistor e um led infravermelho envoltos em uma estrutura de plástico.

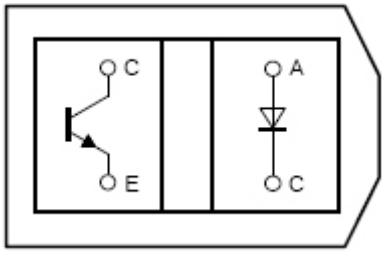


Figura 02. Vista superior do sensor TCRT5000.

O LED emite um feixe de luz infravermelha numa frequência não visível a olho nu, e o fototransistor tem a função de capturar o feixe de luz emitido pelo LED. O funcionamento do fototransistor é bem semelhante ao de um transistor, possui um coletor, um emissor, e a base, que é ativada ao receber um feixe de luz infravermelha.

O sensor possui quatro pinos, sendo dois do LED e dois do fototransistor. São eles:

-Anodo do led (A);
-Catodo do led (C);
-Coletor do fototransistor (C);
-Emissor do fototransistor (E).

Ao aplicar-se uma tensão no LED, ele emite o feixe de luz infravermelha, quando esse feixe de luz encontra algum objeto, essa luz é refletida na base do transistor, fazendo com que haja condução de corrente entre o coletor e o emissor do mesmo.

O LED emite um feixe de luz infravermelha que é refletida pelo objetivo que estiver a frente do sensor e que é, por sua vez, detectada pelo fototransistor. Quanto mais claro for o objeto, melhor será o funcionamento do sensor, pois mais luz será refletida. E é aproveitando-se dessa condição que o sensor será útil para o robô seguidor de linha, pois, no momento em que o sensor detectar uma superfície de cor preta, no caso a linha que ele deve seguir, significa que sua trajetória acabou e o robô está prestes a desviar do caminho. Nesse momento será ativado um comando (vire à direita, por exemplo) para que o robô volte ao trajeto original.

## SOFTWARE

Abaixo é mostrado as funções que realizam as configurações iniciais.

**Funções Setup:**

- Função ***Setup_ADC (void):*** Essa função tem por objetivo realizar a configuração inicial do conversor A/D;
- Função ***Setup_Button (void):*** Essa função configura o botão BTN e aciona uma interrupção na borda de descida;
- Função ***Setup_PWM (void***): Essa função define a porta P1.6 como a saída OUT 1 do TimerA;
- Função ***Setup_L293B (void):*** Essa função define quais as portas serão utilizadas como saídas para acionar as portas de entrada do CI L293B.

Antes de entrar em detalhes sobre o código, é importante destacar como ficou a definição da porta P1OUT da MSP para controle dos motores através do CI L293B.. A Tabela 02 mostra como ficou estruturado.

Tabela 02. Correspondência entre os bits em P1OUT e no CI L293B

| Bit em P1OUT | Especificação | Modo |
|---|---|---|
| BIT 0 (A0 ADC10) | Sensor 1 | ENTRADA ANALÓGICA |
| BIT 1 (A1 ADC10) | Sensor 2 | ENTRADA ANALÓGICA |
| BIT 2 (PORTA P1.2) | INPUT1 – CI L293B | SAÍDA |
| BIT 3 (PORTA P1.3) | Botão de Controle de Velocidade dos Motores | ENTRADA |
| BIT 4 (PORTA P1.4) | INPUT 2 – CI L293B | SAÍDA |
| BIT 5 (PORTA P1.5) | INPUT 3 – CI L293B | SAÍDA |
| BIT 6 (PORTA P1.6) | ENABLE1 e ENABLE2 - CI L293B (saída com PWM) | SAÍDA |
| BIT 7 (PORTA P1.7) | INPUT4 – CI L293B | SAÍDA |

A primeira função é que merece ser comentada com detalhes é a ***Control_Speed().*** O objetivo desta é acionar os pinos de ENABLE1 e ENABLE2 do CI L293B, usando PWM, onde é possível controlar a velocidade do robô. Esta aplicação é necessária para que o usuário possa escolher qual o tipo de limpeza deseja. Definimos que são 5 modos possíveis, onde em uma destas o robô permanece parado. Desta forma pode-se usar 4 velocidades. A função recebe como argumento um inteiro *Speed* que se refere a um dos modos requeridos e não retorna nenhum valor, somente configura o modo de comparação dos registradores de controle e captura *TACCR* 1 e 0 da MSP. Para este modo na MSP430G2553, EQU0 está na Porta P1.6, conforme o DATASHEET e será resetado em TACCR1 e setado TACCR0 (OUTMOD_7). O controle de velocidade é selecionado pelo botão da porta P1.3 na MSP430, e será discutido em detalhes a seguir. Um exemplo pode ser visto abaixo, caso o usuário deseje a velocidade 4.

```
else if(Speed == 4)
  {
   TACTL = TACLR;
   TACCR0 = 10000; //100 hz
   TACCR1 = 4000; //55% of the Duty Cycle
   TACCTL1 = OUTMOD_7;
   TACTL = TACTL = TASSEL_2 + ID_0 + MC_1;
  }
```

A função ***Read_ADC()*** tem o objetivo geral de pegar os valores das conversões realizadas nas portas analógicas A0 e A1 que possuem os valores atuais de tensão nos sensores para que os valores possam ser tratados na função Change_Direction(). Os valores das conversões armazenados na memória são lançados no vetor de inteiros adc[], sendo que adc[0] corresponde ao valor lido no sensor 1, porta A0, e adc[1] corresponde ao valor lido no sensor 2, porta A1.

Ainda na função ***Read_ADC()*** é possível ver a instrução

```
__bis_SR_register(CPUOFF + GIE);
```

Essa instrução foi definida dentro desta função de propósito, pois quando se dá início a uma conversão, o MCLK é desligado e só é ligado novamente com a interrupção do ADC10, que indica que uma conversão foi terminada. O MCLK será ligado também quando o botão P1.3 for acionado, o que será comentado a seguir.

A próxima função é a ***Control_Direction(),*** que tem o objetivo de trocar a direção do motor conforme os valores lidos nos sensores. A tensão de referência $V^+$ no conversor é 3V, representado pelo valor 1023. Se o valor de tensão sobre o fototransistor for menor que 2,63 V, ou 900 como valor lido, então isto indica que existe luz sendo refletida e logo nenhuma curva está a vista, sendo que deve-se seguir reto. Mas se um valor $\geq$ 900 for lido em algum dos sensores, deve-se virar ou à esquerda ou à direita, dependendo do sensor lido. Se, ainda, os dois valores de tensão lidos forem $\geq$ 900, isto indica um fim de percurso e o robô deve parar. Um outro ponto importante é o trecho a seguir da função

```
if((impulse==0) && (com!=0))
   {
   volatile unsigned int delay = 8000;
   Control_Speed(5);
   while(delay--);
   Control_Speed(com);
   impulse++;
```

Este trecho serve para que os motores possam romper a inércia inicializando com 75% do valor de seu torque máximo sempre que o robô for acionado pela primeira vez ou que terminar de realizar uma curva. Isto ocorrerá por um breve período de tempo.

A última função secundária que merece ser mencionada é a ***Read_Button()*** que serve para ler se o usuário deseja aumentar a velocidade do robô, sempre que pressiona o botão da porta P1.3. Esta função possui a variável int com, que indica qual a velocidade atual do robô e é inicializada toda vez que se está na velocidade 4 e pressiona-se o botão, indo novamente para 0, onde os motores são desativados.

Na função ***main()*** é feito a chamada a todas as funções de configuração mencionadas e também as funções que operam o

robô, mantendo em um laço infinito as funções Read_Button() e Read_ADC().

E por último, sempre que uma interrupção for acionada, esta acionará o MCLK novamente para realizar o comando desejado, seja uma conversão que acabou ou seja o pressionamento do botão, como pode ser visto abaixo.

```
//P1IN interrupt service routine
#pragma vector = PORT1_VECTOR
__interrupt void Port_1(void)
{
   P1IFG &=~BTN;
   __bic_SR_register_on_exit(CPUOFF);
}

// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
  __bic_SR_register_on_exit(CPUOFF);        // Clear CPUOFF bit from 0(SR)
}
```

**Funções Setup:**

- Função ***Setup_ADC (void):*** Essa função tem por objetivo realizar a configuração inicial do conversor A/D;
- Função ***Setup_Button (void):*** Essa função configura o botão BTN e aciona uma interrupção na borda de descida;
- Função ***Setup_PWM (void***): Essa função define a porta P1.6 como a saída OUT 1 do TimerA;
- Função ***Setup_L293B (void):*** Essa função define quais as portas serão utilizadas como saídas para acionar as portas de entrada do CI L293B.

## IV. RESULTADOS

Com o intuito de verificar a viabilidade do projeto pretendido pela dupla, alguns testes foram feitos com a MSP430, o CI L293B e os sensores ópticos reflexivos. Montando-se o circuito da Fig. 03 na protoboard, foi possível simular as seguintes funções do robô:

- Movimento em linha reta;
- Desvio para a direita;
- Desvio para a esquerda;
- Parada.

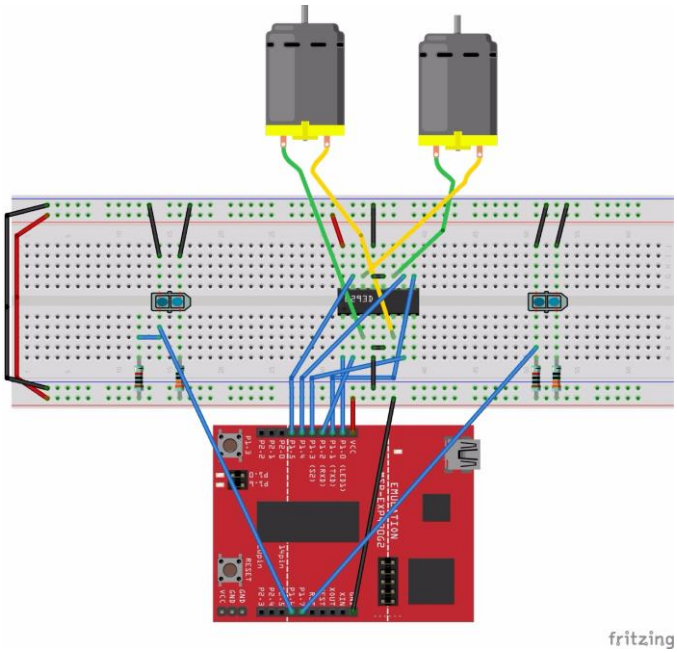


Figura 03. Simulação de algumas funções do robô seguidor de linha.

Dado o sucesso obtido nas simulações, deu-se início a construção do robô utilizando-se um chassi de acrílico com dois motores DC de 6V e dois sensores ópticos reflexivos acoplados.

A Fig. 04 mostra o robô construído.

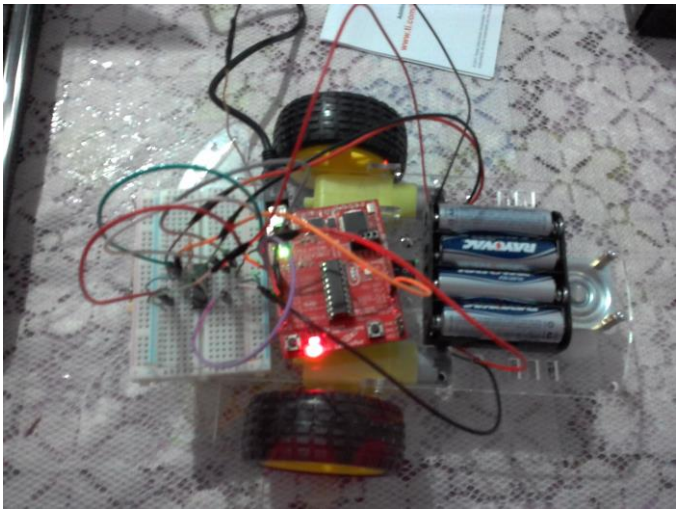

Figura 04a. Robô seguidor de linha – vista superior.

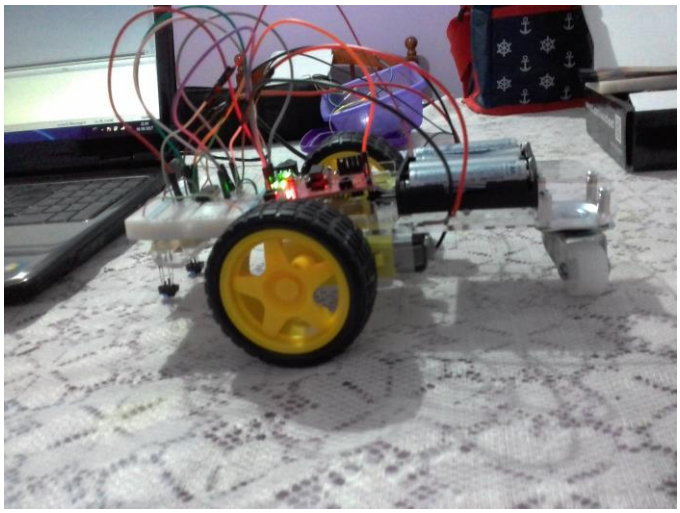

Figura 04b. Robô seguidor de linha – vista lateral.

Com o robô construído, e utilizando-se o código mostrado no Anexo A, verificou-se o funcionamento dos sensores ópticos reflexivos, pois através deles foi possível fazer com que o robô seguisse um caminho específico, feito com fita isolante, formando uma linha preta em uma cartolina branca, como mostra a Fig. 05.

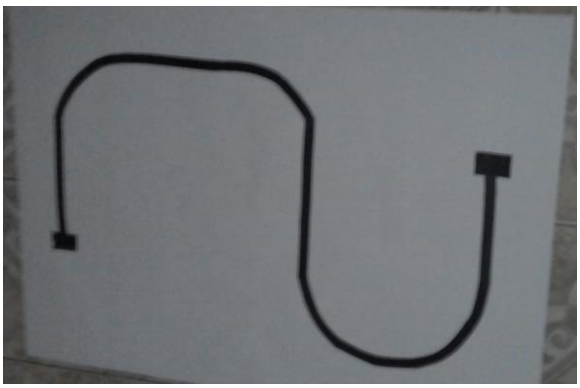

Figura 05. Caminho para o robô.

O uso dos sensores ópticos reflexivos, foi feito através de conversão A/D, como mostram as funções Setup ADC(); e Read ADC(); apresentadas no código do Anexo A e já explicadas no presente relatório.

## V. Conclusão

Com o trabalho realizado até o presente momento e o conteúdo apresentado na disciplina de Microprocessadores e Microcontroladores, é possível concluir que a implementação do robô seguidor de linha é totalmente compatível com o conteúdo aprendido durante o semestre, faltando então apenas algum refinamento simples no código e acréscimo de acessórios no robô, como por exemplo, algum material que faça o papel de uma vassoura ou um esfregão acoplado ao robô, uma vez que se trata de um robô destinado à limpeza.

É possível concluir ainda, que embora não haja tempo hábil, os conhecimentos adquiridos na disciplina, capacitam a dupla para construir um robô seguidor de linha com mais funções do que as apresentadas nessa proposta de projeto, como por exemplo, programar o robô para realizar a limpeza personalizada de um ambiente com vários cômodos (i.e. limpeza mais rápida no cômodo 1, mais lenta no cômodo 2, etc.).

## VI. Anexos

### *A. Código*

```
#include<msp430g2553.h>
#define BTN BIT3
#define INPUT1 BIT2 //INPUT FROM IC L293B
#define INPUT2 BIT4 //INPUT FROM IC L293B
#define INPUT3 BIT5 //INPUT FROM IC L293B
#define INPUT4 BIT7 //INPUT FROM IC L293B
#define INPUTS (INPUT1 + INPUT2 + INPUT3 + INPUT4)
#define ADC_CHANNELS 2
volatile unsigned int com=0;
volatile unsigned int impulse=0;
unsigned int adc[ADC_CHANNELS]=0;

void Control_Speed(int Speed)
{
  if(Speed == 0)
  {
   TACTL = TACLR;
   TACCR0 = 10000; //100 hz
   TACCR1 = 0;
   TACCTL1 = OUTMOD_7;
   TACTL = TASSEL_2 + ID_0 + MC_1;
  }
  else if(Speed == 1)
  {
   TACTL = TACLR;
   TACCR0 = 10000; //100 hz
   TACCR1 = 2500; //25% of the Duty Cycle
   TACCTL1 = OUTMOD_7;
   TACTL = TASSEL_2 + ID_0 + MC_1;
  }
  else if(Speed == 2)
  {
   TACTL = TACLR;
   TACCR0 = 10000; //100 hz
   TACCR1 = 3000; //35% of the Duty Cycle
   TACCTL1 = OUTMOD_7;
   TACTL = TASSEL_2 + ID_0 + MC_1;
  }
  else if(Speed == 3)
  {
   TACTL = TACLR;
   TACCR0 = 10000; //100 hz
   TACCR1 = 3500; //45% of the Duty Cycle
```

```
    TACCTL1 = OUTMOD_7;
    TACTL = TASSEL_2 + ID_0 + MC_1;

   }
   else if(Speed == 4)
   {
    TACTL = TACLR;
    TACCR0 = 10000; //100 hz
    TACCR1 = 4000; //55% of the Duty Cycle
    TACCTL1 = OUTMOD_7;
    TACTL = TACTL = TASSEL_2 + ID_0 + MC_1;
   }
   else if(Speed == 5)
   {
    TACTL = TACLR;
    TACCR0 = 10000; //100 hz
    TACCR1 = 6500; //65% of the Duty Cycle
    TACCTL1 = OUTMOD_7;
    TACTL = TACTL = TASSEL_2 + ID_0 + MC_1;
   }
}

void Change_Direction(void)
{
 volatile unsigned int i = 2000;
 if((adc[0] < 900) && (adc[1] < 900))
 {

  P1OUT &= ~INPUTS;
  P1OUT |= 0x24;
  if((impulse==0) && (com!=0))
  {
  volatile unsigned int delay = 8000;
  Control_Speed(5);
  while(delay--);
  Control_Speed(com);
  impulse++;
  }
 }
 else if((adc[0] > 900) && (adc[1] < 900))
 {
  P1OUT &= ~INPUTS;
  P1OUT |= 0x34; //Motor_Direction('S','R');
  impulse=0;
 }
 else if((adc[0] < 900) && (adc[1] > 900))
 {
  P1OUT &= ~INPUTS;
  P1OUT |= 0xA4; //Motor_Direction('R','S');
  impulse=0;
 }
 else
 {
  P1OUT &= ~INPUTS;
  P1OUT |= 0xB4; //Motor_Direction('S','S');
  impulse=0;
 }
  while(i--);
}

void Setup_ADC(void)
{
 ADC10CTL0 |=  SREF_0 +  ADC10SHT_0 + MSC +
ADC10ON + ADC10IE; //Reference from
 //Vcc and Vss, sampling time of 16×ADC10CLKs,
ADC10ON
 ADC10CTL1 |= INCH_1 + CONSEQ_3 + ADC10SSEL_3 +
SHS_0; //Input channel A1 and A0; repeated sequence
 ADC10AE0 = BIT0 + BIT1; // Analog Input in P1.0 and
P1.1;
 ADC10DTC1 = ADC_CHANNELS; // 2 conversions
 ADC10CTL0 |= ENC + ADC10SC; // Sampling and
conversion start
}

void Read_ADC(void)
{
 ADC10CTL0 &= ~ENC;
 while (ADC10CTL1 & BUSY);// Wait if ADC10 core is
active
 ADC10SA = (unsigned int)adc;// Copies data in ADC10SA
to unsigned int adc array
 ADC10CTL0 |= ENC + ADC10SC;
 __bis_SR_register(CPUOFF + GIE);
 Change_Direction();
}

void Setup_Button(void)
{
 P1DIR &= ~BTN;
 P1OUT |= BTN; /*Definindo o resistor de pull-up e pull-
down*/
 P1REN |= BTN; //Definindo como pull-down
 P1IE |= BTN; // Habilita Interrupcao na porta P1.0
 P1IES |= BTN; //Habilita Interrupcao na borda de descida
}

void Setup_PWM(void)
{
 P1DIR |= BIT6;
 P1SEL |= BIT6; //Usando a funcao TA0.1 na porta P1.6
 P1SEL2 &= ~BIT6;//Usando a funcao TA0.1 na porta P1.6
}

void Setup_L293B(void)
{
 P1OUT &= ~INPUTS;/*Inicializando as saidas em 0*/
 P1DIR |= INPUTS;
}

void Read_Button(void)
{
   volatile unsigned int i = 20000;
   //Controle de Velocidade dos Motores
```

```
   if((P1IN&BTN)==0)
   {
    if(com!=4)
    {
      com++;
      Control_Speed(com);
      while(i--);
    }
    else
    {
      com = 0;
      Control_Speed(0);
      while(i--);
    }
   }
}


int main(void)
{
  WDTCTL = WDTPW | WDTHOLD;
  BCSCTL1 = CALBC1_1MHZ;      //MCLK e SMCLK @
1MHz
  DCOCTL = CALDCO_1MHZ;       //MCLK e SMCLK @
1MHz
  Setup_L293B();
  Setup_Button();
  Setup_PWM();
  Setup_ADC();
  Read_ADC();
  Control_Speed(0); //Inicializar os motores parados
  while(1)
  {
    Read_Button();
    Read_ADC();
  }
}

//P1IN interrupt service routine
#pragma vector = PORT1_VECTOR
__interrupt void Port_1(void)
{
   P1IFG &=~BTN;
   __bic_SR_register_on_exit(CPUOFF);
}

// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
  __bic_SR_register_on_exit(CPUOFF);        // Clear CPUOFF
bit from 0(SR)
}
```

REFERÊNCIAS


[1] Davies, J., MSP430 Microcontroller Basics, Elsevier, 2008.

[2] Laboratório de Garagem, Carrinho seguidor de linha que desvia de obstáculos com plataforma Zumo e Arduino. Disponível em: <http://labdegaragem.com/profiles/blogs/tutorial-carrinho-seguidor-de-linha-que-desvia-de-obstaculos-com-> Acesso em 04 de Abril de 2017.

[3] Apostila: Oficina seguidor de linha. Vieira, Gabriel Meneses.

[4] McRoberts, Michael. Arduino Básico. [tradução Rafael Zanolli]. São Paulo: Novatec Editora, 2011