Relatório Final

Robô seguidor de linha para limpeza industrial – JORDO.

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 um relatório 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 limpezas industriais.

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.

Com o intuito de auxiliar na limpeza industrial, especialmente em locais pequenos, de difícil acesso e locais que ofereçam risco à saúde, foi construído um robô seguidor de linha, denominado de JORDO, que realiza um trajeto especificado, programado na MSP430. Este robô foi capaz de se movimentar para frente, para trás, direita e esquerda e contou com 4 opções de velocidade. Com isso, o usuário poderá optar por uma limpeza mais superficial ou mais específica, dependendo da necessidade deste.

II. OBJETIVOS

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

III. DESCRIÇÃO

Primeiramente, é necessário descrever as funções dos componentes de hardware de alguns componentes utilizados no protótipo do JORDO.

A. HARDWARE

1) 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.

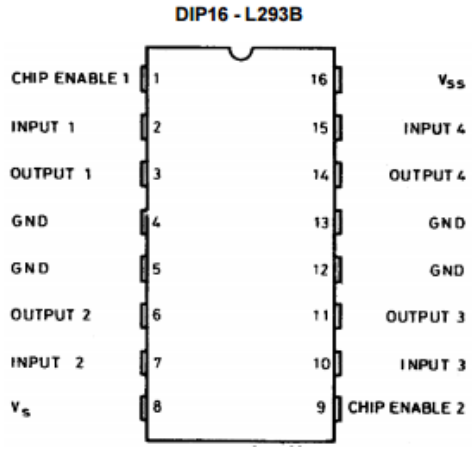

Figura 01. Pinagem do CI L293B

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 foi utilizado para controlar a direção do robô, conforme o acionamento dos sensores, e o modo de limpeza, alternando entre as velocidades dos motores.

2) 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.

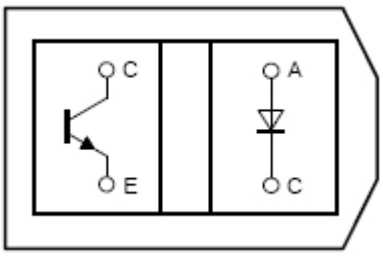

Figura 02. Vista superior do sensor TCRT5000.

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. No momento em que o sensor detectar uma superfície de cor branca, no caso a linha que ele deve seguir, significa que deve-se alterar a trajetória e o robô está prestes a desviar do caminho. Neste momento, será ativado um comando (vire à direita, por exemplo) para que o robô volte ao trajeto original.

O LED infravermelho deve ser ligado em série a um resistor de 330Ω e o fototransistor a um resistor 10kΩ.

3) Lista de Materiais

Abaixo na Tabela na Tabela 2 encontra-se a lista de materiais do projeto.

Tabela 02. Lista de Materiais do Projeto

Componente	**Quantidade**
MCU MSP430g2553	1
CI L293B	1
Sensor TCRT5000	2
Resistor de 10kΩ	2
Resistor de 330Ω	2
Resistor de 1kΩ	2
Motor DC 6V	2
Motor DC 3V	2
Push Button	2
Pilha AA	4
Pilha AAA	2
Chave (Switch)	2

4) Esquemático

Na figura 3 encontra-se o esquemático do projeto. É importante destacar que nesta figura o CI L293D faz o mesmo papel do CI L293B, que na verdade são equivalentes, mas com uma faixa de trabalho diferente.

Além disso, o CI CNY70 possui a mesma função que o TCRT5000. Os motores ligados ao CI L293D na figura 3 representam os motores que fazer o robô andar e os outros dois representam os motores que estão realizando a limpeza.

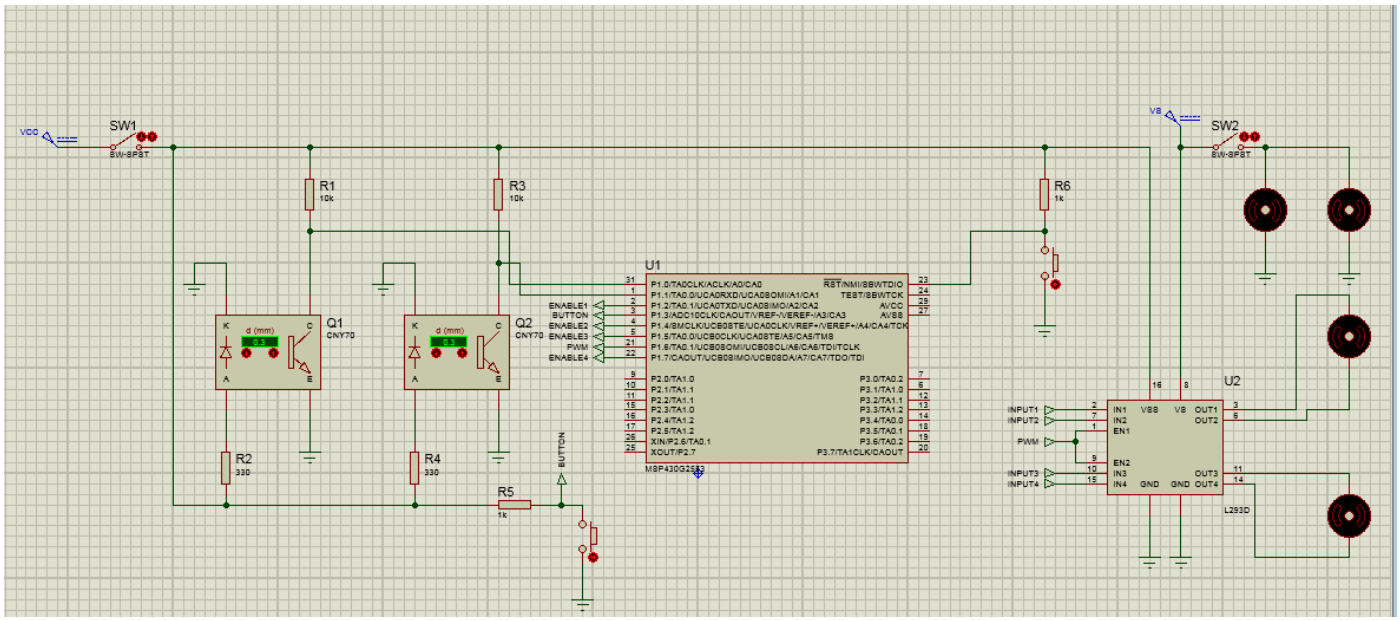

Figura 3. Esquemático Completo do Projeto

B. 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 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 *TA0CCR* 1 e 0 da MSP. Para este modo na MSP430G2553, EQU0 está na Porta P1.6, conforme o DATASHEET e será resetado em TA0CCR1 e setado TA0CCR0 (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[1] corresponde ao valor lido no sensor 1, porta A0 (sensor do motor direito), e adc[0] corresponde ao valor lido no sensor 2, porta A1 (sensor do motor esquerdo).

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 2,34 V, ou 800 como valor lido, então isto indica que existe luz sendo refletida e logo há uma curva a vista, sendo que deve-se seguir virar à esquerda ou a direita, ou parar, se os dois sensores tiverem valores < 800, o robô deve parar. Mas se um valor ≥ 900 for lido em algum dos sensores, continua-se seguindo reto. 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)
}
```

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. 04 na protoboard, foi possível simular as seguintes funções do robô:

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

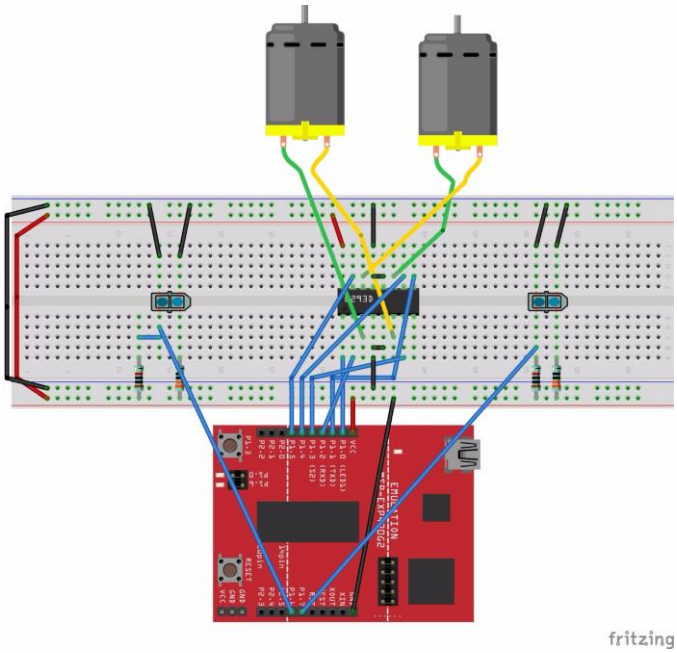

Figura 04. Simulação de algumas funções do robô seguidor de linha.

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. 05 mostra o robô construído.

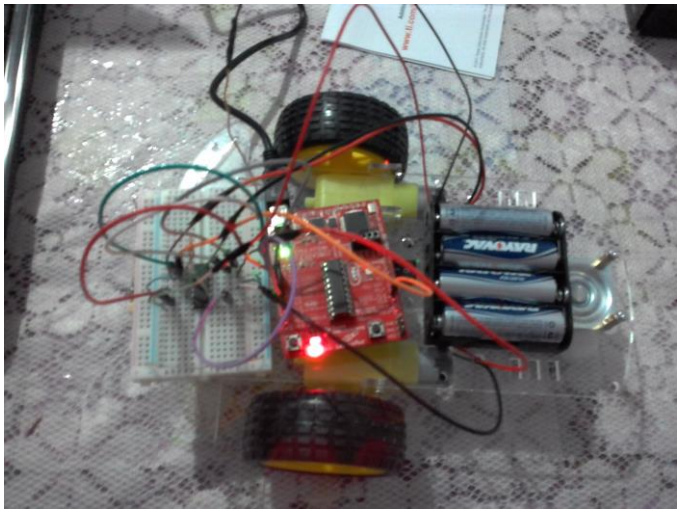

Figura 05a. JORDO – vista superior.

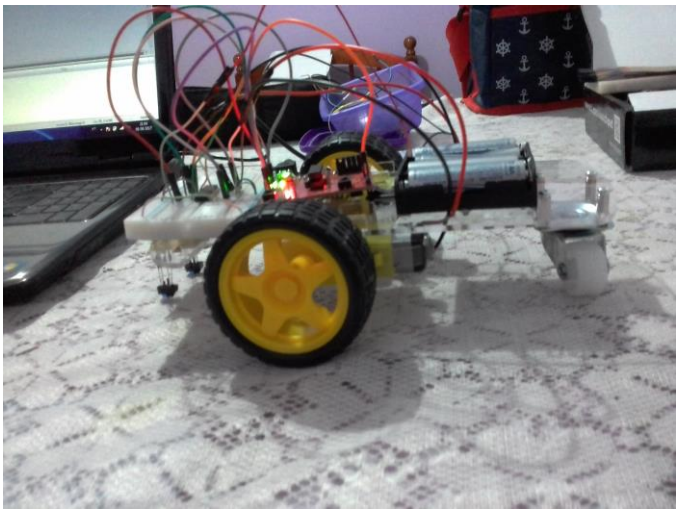

Figura 05b. JORDO – vista lateral.

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. 06.

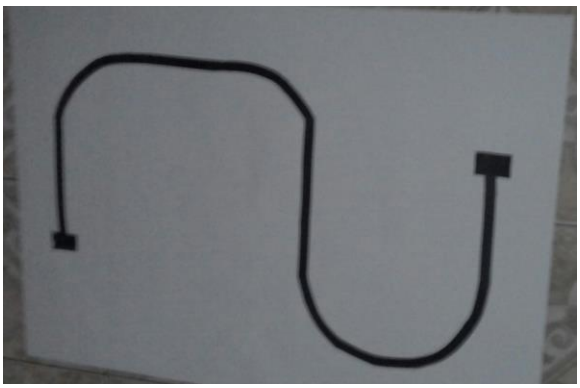

Figura 06. Caminho para o robô.

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.

Apesar do sucesso dos testes feitos no caminho mostrado na Fig. 06, observou-se a necessidade da implementação de uma nova pista, pois é preciso levar em consideração o fato de que o robô deve ser usado para limpeza industrial em locais onde o piso estará bastante sujo. Por isso, constatou-se que utilizar uma cartolina branca não é a melhor forma de representação do uso real do robô. Assim, implementou-se um caminho branco, feito com fita crepe em uma cartolina preta, que além de se adequar melhor à proposta de projeto, foi feito de forma que o robô tenha andado por toda a cartolina ao completar o caminho. A Fig. 07 mostra o novo caminho.

Figura 07. Caminho para o robô – v2.

Acrescentou-se ainda ao chassi, duas esponjas circulares acopladas à motores DC de 6V, a fim de simularem esfregões para a limpeza. A Fig. 08 mostra o chassi com as esponjas acopladas.

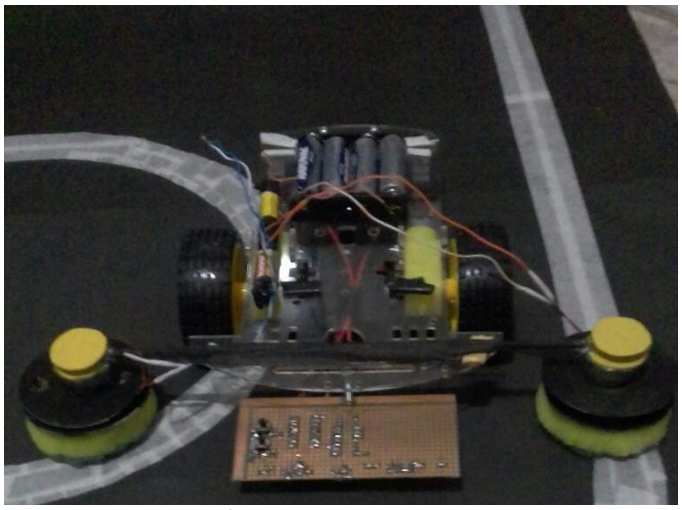

Figura 08. Robô JORDO com esponjas acopladas.

Por fim, foi de decisão da dupla que se implementasse o circuito do robô JORDO em uma placa de circuito perfurada, para demonstrar um possível circuito eletrônicos real para utilização do robô. As figuras 09 e 10 mostram como ficaram os resultados.

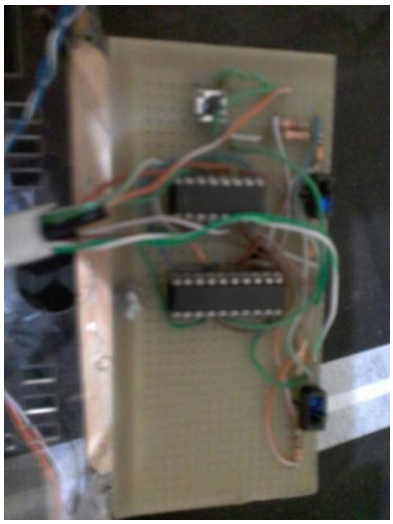

Figura 09. Vista superior da placa de circuito montada

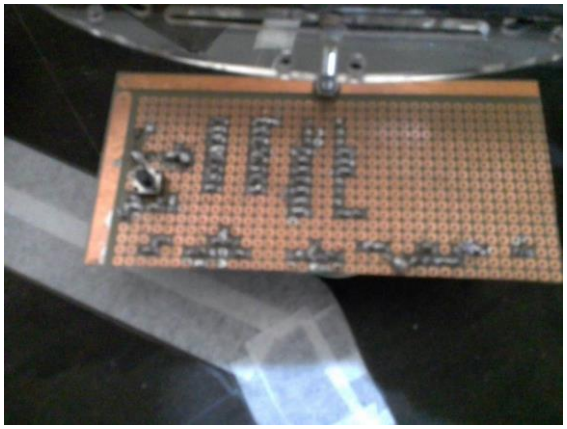

Figura 10. Vista inferior da placa de circuito montada mostrando as soldagens realizadas

V. CONCLUSÃO

Com o trabalho realizado ao longo do semestre 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, o que tornou possível o sucesso obtido pela dupla. Entretanto, é importante ressaltar que existem algumas limitações no robô construído, por exemplo, para que o seu funcionamento seja perfeito, não se deve ter um período de tempo muito longo entre uma limpeza e outra de um mesmo local, pois o acúmulo de sujeira pode cobrir a linha que o robô deve seguir, de forma a comprometer o seu funcionamento. Nesse caso, é importante fazer um estudo para descobrir quais são os possíveis locais onde esse robô pode ser utilizado, bem como pesquisa com clientes em potencial, e a partir desse estudo, descobrir qual o grau de risco que essa limitação apresenta para o sucesso desse robô como solução viável para limpeza industrial. Caso o problema comprometa de fato a implementação, faz-se necessária uma mudança nos circuitos e na programação constituintes do robô, a fim de contornar essa limitação.

É possível concluir ainda, que os conhecimentos adquiridos na disciplina, capacitam a dupla para construir um robô seguidor de linha com mais funções do que as apresentadas nesse 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.). Entretanto, a limitação de tempo do semestre letivo, tornou essa opção inviável.

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;
volatile unsigned int stop=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)
{
 //adc[0] = p1.1 - sensor esquerda
 //adc[1] = p1.0 - sensor direita
 volatile unsigned int i = 2000;
 if((adc[0] > 800) && (adc[1] > 800))
 {
  P1OUT &= ~INPUTS;
  P1OUT |= 0x84;
  if((impulse==0) && (com!=0))
  {
  volatile unsigned int delay = 8000;
  Control_Speed(5);
  while(delay--);
  Control_Speed(com);
  impulse++;
  }
 }
 else if((adc[1] < 800) && (adc[0] > 800))
 {
  //right motor stops
  P1OUT &= ~INPUTS;
  P1OUT |= 0x94; //Motor_Direction('S','R');
  impulse=0;
 }
 else if((adc[0] < 800) && (adc[1] > 800))
 {
  //left motor stops
  P1OUT &= ~INPUTS;
  P1OUT |= 0xA4; //Motor_Direction('R','S');
  impulse=0;
 }
 else
 {
```

```
    if(stop==1)
    {
    P1OUT &= ~INPUTS;
    P1OUT |= 0xB4; //Motor_Direction('S','S');
    stop=0;
    }
    else
    {
      stop=1;
      volatile unsigned int delay = 1000;
      while(delay--);
    }
    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