Carrinho de supermercado guiado por trilha

Rafael Alves Magalhães - 12/0020718
Graduando Engenharia Eletrônica
Faculdade UnB Gama - Universidade de Brasília
Área Especial de Indústria Projeção A, UnB - Df-480 - Gama Leste, Brasília - DF
e-mail: magalhaesrafael07@gmail.com

Renato C. S.Agnello - 12/0053896
Graduando Engenharia Eletrônica
Faculdade UnB Gama - Universidade de Brasília
Área Especial de Indústria Projeção A, UnB - Df-480 - Gama Leste, Brasília - DF
e-mail: ragnello19@gmail.com

Resumo — Criar um protótipo de um carrinho de compras automático, que seguirá uma linha (de cor preta) colocada no chão dos corredores de um supermercado.

Palavras-Chaves—MSP430, automação

I. Introdução

Em grande parte dos supermercados, sejam eles de pequeno, médio ou grande porte, podemos encontrar cestas plásticas ou carrinhos, que auxiliam o consumidor na hora da compra dos produtos. Apesar dos carrinhos terem rodas, que facilitam e muito o carregamento dos produtos, eles podem ficar bastante pesados para algumas pessoas, dificultando esse carregamento.

Esse projeto foi pensado após presenciar várias vezes a dificuldade de uma pessoa idosa na hora de fazer compras, por causa do aumento de peso do carrinho ao colocar os produtos dentro. Isso quando o idoso não estava acompanhado de uma outra pessoa, ajudando-o com a tarefa de empurrar o carrinho.

O projeto tem a finalidade de tornar o idoso mais independente, fazendo com que ele não precise de outros para carregar o carrinho.

II. Objetivos

Este projeto tem como principal objetivo desenvolver um sistema de controle de motores através de sensores infravermelhos, fazendo com que controlasse um carrinho de compras através de um percurso pré-definido. Outros dos objetivos são:

1. Estudar e analisar os melhores locais para a utilização do projeto (onde seria mais viável implementar);
2. Estudar sobre sensores infravermelhos e suas aplicações;
3. Estudar e aprender sobre o microcontrolador MSP430;
4. Testar e verificar o funcionamento do protótipo.

III. Desenvolvimento do Protótipo

A. Componentes

Componentes	**Quantidade**	**Preço Unitário(R$)**
Protoboard	1	21,90
Launchpad MSP430	1	50,00
CI L293D	1	8,10
Motor + Rodas	2	16,90
Sensor Infravermelho	2	1,50
LED	2	0,15
Transistor (BC547)	2	0,13
Resistor	4	0,04
Jumpers M-F (25)	1	14,90
Valor Total	**-**	**132,42**

B. *Descrição do Hardware*

Para simular o funcionamento do sistema desenvolvido, foi montado em uma protoboard o circuito que controlará o motor. O trabalho de alimentar o circuito e o motor, será feito pelo próprio microcontrolador MSP430, sabendo que ele tem uma alimentação de 3.3V nas saídas dos pinos VCC. Essa tensão é mais do que o suficiente para alimentar os componentes utilizados no protótipo.

Para verificar o funcionamento dos sensores, foram utilizados dois LEDs (vermelho para o motor da esquerda e verde para o motor da direita). Eles acendem quando seu respectivo sensor faz a leitura de uma superfície na qual não reflete bem a luz, ou seja, uma superfície mais escura.

Os motores funcionam quando os LEDs estão apagados. Esse mecanismo foi utilizado para verificar o funcionamento tanto dos sensores como dos motores.

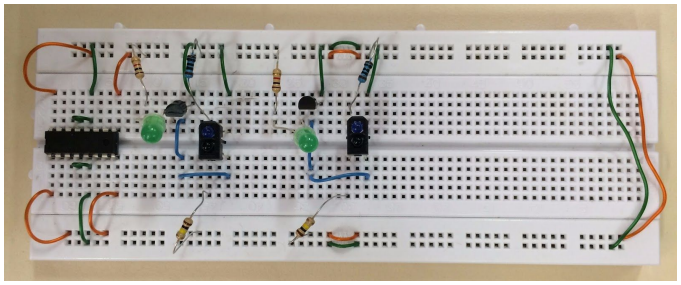
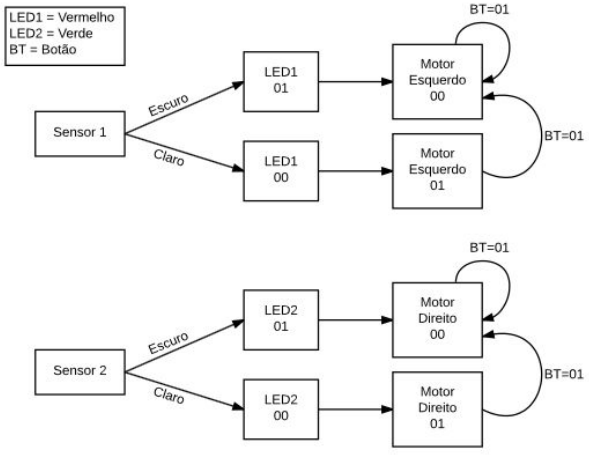


Figura 1 - Circuito de controle dos motores em protoboard

Os LEDs colocados no circuito servem para visualizar qual motor esta parado.

1. *Fluxograma do Projeto*



C. *Descrição do Software*

O código foi escrito na linguagem C, baseado no microcontrolador utilizado no projeto, que no caso foi o MSP430. Os sensores ficarão ativos o tempo todo em que o sistema estiver em funcionamento, fazendo a varredura para que o carrinho não saia do percurso. Essa varredura será feita no laço infinito, for(;;), que está localizado na função main() do código. Foram criadas também outra função, sendo ela:

– Converter_AD(): utilizada para fazer a configuração do conversor para retornar a melhor aproximação possível;

Além das funções criadas, foram utilizadas funções prontas, chamadas de Interrupção. Essa função é chamada quando ocorre algum evento no código, como por exemplo, quando
o contador do Timer da launchpad termina a contagem determinada e seta o flag ou quando um botão é pressionado.

Essa interrupção faz com que o microcontrolador salve na memória a parte do código que o programa estava lendo entre em modo de baixo consumo, fazendo com que reduza o consumo de energia, pule e execute as instruções determinadas na função de interrupção. Após terminar de executar as instruções da interrupção, o programa volta para a parte do código que estava salva na memória e continua a leitura e execução do código.

As duas interrupções utilizadas para o projeto foram as seguintes:

– __interrupt void Port_2(void): ativada pelo botão, ela executa a ação de parar os dois motores do carrinho;
– __interrupt void ADC10(void): interrupção do conversor, serve para quando a placa estiver fazendo a conversão A/D, ela se concentrar nessa tarefa, desativando temporariamente as demais funções do microcontrolador. Ele possui a configuração de um Timer, que serve para executar um pequeno *delay* entre as leituras dos sensores.

Resumindo, a placa faz a leitura analógica dos sensores infravermelhos através das portas P1.0 e P1.1 (BIT0 e BIT1), faz a conversão e, dependendo da leitura, faz os motores entrarem em funcionamento (ou não) através das portas P1.2/P1.4 (para o primeiro motor) e P1.5/P1.6 (para o segundo motor). Caso o botão seja pressionado, a interrupção é chamada fazendo com que seja enviado o nível lógico baixo para todas as portas dos motores, fazendo eles pararem. A leitura do botão é feita pela porta P1.3 da placa.

As figuras abaixo mostram em partes o código utilizado para o funcionamento do sistema, que está completo no Apêndice.

```c
#include <msp430g2553.h>    //Bibliotecas
#include <intrinsics.h>

#define SENSOR_1    BIT0
#define SENSOR_2    BIT1
#define M1_FRENTE   BIT2
#define M1_TRAS     BIT4          //Definições dos Bits
#define M2_FRENTE   BIT5
#define M2_TRAS     BIT6
#define ANALOG_INCH INCH_1
#define VALOR 400

int sensores[2];
void Converter_AD(void);
```

Figura 2 - Definição de bibliotecas, dos BITS e escopo das funções

```c
int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;

  BCSCTL1 = CALBC1_1MHZ; //MCLK e SMCLK definidos a 1MHz
  DCOCTL = CALDCO_1MHZ;

  P1DIR |= M1_FRENTE|M1_TRAS|M2_FRENTE|M2_TRAS;
  P1OUT &= ~(M1_FRENTE|M1_TRAS|M2_FRENTE|M2_TRAS);

  Converter_AD(); //Chamada do conversor

  _BIS_SR(GIE);
```

Figura 3 - Início da main, definição do MCLK e SMCLK e dos BITS de saída

```c
for(;;)
{
  if(sensores[0] >= VALOR && sensores[1] >= VALOR)
  {
    P1OUT |= M1_FRENTE|M2_FRENTE;
    P1OUT &= ~(M1_TRAS|M2_TRAS);
  }
  else if(sensores[0] >= VALOR && sensores[1] < VALOR)
  {
    P1OUT |= M1_FRENTE;
    P1OUT &= ~(M1_TRAS|M2_FRENTE|M2_TRAS);
  }
  else if(sensores[0] < VALOR && sensores[1] < VALOR)
  {
    P1OUT &= ~(M1_FRENTE|M1_TRAS|M2_FRENTE|M2_TRAS);
  }
  else
  {
    P1OUT |= M2_FRENTE;
    P1OUT &= ~(M1_FRENTE|M1_TRAS|M2_TRAS);
  }
 }
}
```

Figura 4 - Laço de repetição e estruturas condicionais

```c
void Converter_AD(void)
{
  TACCTL1 = OUTMOD_7;// Set/Reset
  TACCR0 = 400-1;
  TACCR1 = TACCR0/2; // Clock 50%
  TACTL = TASSEL_2 + ID_0 + MC_1; // Configuração para 5Khz

  ADC10CTL1 = ANALOG_INCH + SHS_1 + ADC10DIV_0 + ADC10SSEL_3 + CONSEQ_3;
  ADC10CTL0 = SREF_0 + ADC10SHT_2 + ADC10IE + ADC10ON;
  ADC10AE0 = 0x03;
  ADC10DTC1 = 0x02; // Transfere os dados de ADC10MEM para a variável
  ADC10SA = (unsigned int)sensores;
  ADC10CTL0 |= ENC;
}
```

Figura 5 - Configuração do conversor A/D

```c
#pragma vector=ADC10_VECTOR
__interrupt void ADC10(void)
{
  ADC10CTL0 &= ~ENC;
  ADC10CTL0 &= ~ADC10IFG;
  ADC10SA = (unsigned int)sensores;
  ADC10CTL0 |= ENC;
}
```

Figura 6 - Interrupção do conversor A/D

IV. CONCLUSÃO

O projeto realizado foi de grande importância para o grupo pois pode-se utilizar basicamente toda a matéria aprendida durante o semestre que inclui a comunicação de circuitos externos ao MSP, a utilização de cargas pesadas e a conversão analógica-digital, e aplicá-la em um sistema real. Além disso, houve um aprofundamento do estudo de alguns conteúdos como a conversão analógica-digital que foi de extrema necessidade para o projeto funcionar.

Pode-se observar também, melhorias que poderiam ser feitas em relação ao sistema, como a utilização de mais sensores para tornar o sistema mais preciso, a separação da fiação e também a utilização de uma comunicação serial.

V. Revisão Bibliográfica

[1] Davies, J., MSP430 Microcontroller Basics, Elsevier, 2008.
[2]Pereira, F., Microcontroladores MSP430: Teoria e Prática, 1a ed., Érica, 2005.

[3]

VI. Apêndice

A. Código do Projeto

```
#include <msp430g2553.h>    //Bibliotecas
#include <intrinsics.h>

#define SENSOR_1   BIT0
#define SENSOR_2   BIT1
#define M1_FRENTE  BIT2
#define M1_TRAS    BIT4          //Definições dos Bits
#define M2_FRENTE  BIT5
#define M2_TRAS    BIT6
#define ANALOG_INCH INCH_1
#define VALOR 400

int sensores[2];
void Converter_AD(void);

int main(void)
{
 WDTCTL = WDTPW + WDTHOLD;

    BCSCTL1 = CALBC1_1MHZ; //MCLK e SMCLK definidos a 1MHz
 DCOCTL = CALDCO_1MHZ;

                                    P1DIR |= M1_FRENTE|M1_TRAS|M2_FRENTE|M2_TRAS;
                                    P1OUT &= ~(M1_FRENTE|M1_TRAS|M2_FRENTE|M2_TRAS);

 Converter_AD(); //Chamada do conversor

 _BIS_SR(GIE);

 for(;;)
 {
  if(sensores[0] >= VALOR && sensores[1] >= VALOR)
  {
   P1OUT |= M1_FRENTE|M2_FRENTE;
   P1OUT &= ~(M1_TRAS|M2_TRAS);
  }
  else if(sensores[0] >= VALOR && sensores[1] < VALOR)
  {
   P1OUT |= M1_FRENTE;
   P1OUT &= ~(M1_TRAS|M2_FRENTE|M2_TRAS);
  }
  else if(sensores[0] < VALOR && sensores[1] < VALOR)
  {
                                    P1OUT &= ~(M1_FRENTE|M1_TRAS|M2_FRENTE|M2_TRAS);
  }
  else
  {
   P1OUT |= M2_FRENTE;
   P1OUT &= ~(M1_FRENTE|M1_TRAS|M2_TRAS);
  }
 }
}

void Converter_AD(void)
{
 TACCTL1 = OUTMOD_7;// Set/Reset
 TACCR0 = 400-1;
 TACCR1 = TACCR0/2; // Clock 50%
  TACTL = TASSEL_2 + ID_0 + MC_1; // Configuração para 5Khz

 ADC10CTL1 = ANALOG_INCH + SHS_1 + ADC10DIV_0 + ADC10SSEL_3 + CONSEQ_3;
  ADC10CTL0 = SREF_0 + ADC10SHT_2 + ADC10IE + ADC10ON;
 ADC10AE0 = 0x03;
  ADC10DTC1 = 0x02; // Transfere os dados de ADC10MEM para a variável (0x02 = 2 canais)
 ADC10SA = (unsigned int)sensores;
 ADC10CTL0 |= ENC;
}

#pragma vector=ADC10_VECTOR
__interrupt void ADC10(void)
{
 ADC10CTL0 &= ~ENC;
 ADC10CTL0 &= ~ADC10IFG;
 ADC10SA = (unsigned int)sensores;
 ADC10CTL0 |= ENC;
}

/*
#pragma vector=PORT2_VECTOR
__interrupt void Port_2(void)
{
 while((P2IN && BTN) == 0)
                                    P2OUT &= ~(M1_FRENTE|M1_TRAS|M2_FRENTE|M2_TRAS);

 P2IFG = 0;
}
*/
```

B. Referência Bibliográfica

- http://www.instructables.com/id/Line-follower-using-msp430g2-launchpad/
- http://www.instructables.com/id/Robot-Line-Follower/
- http://www.ti.com/lit/ds/symlink/l293.pdf
- https://www.sparkfun.com/datasheets/Components/BC546.pdf
-