U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

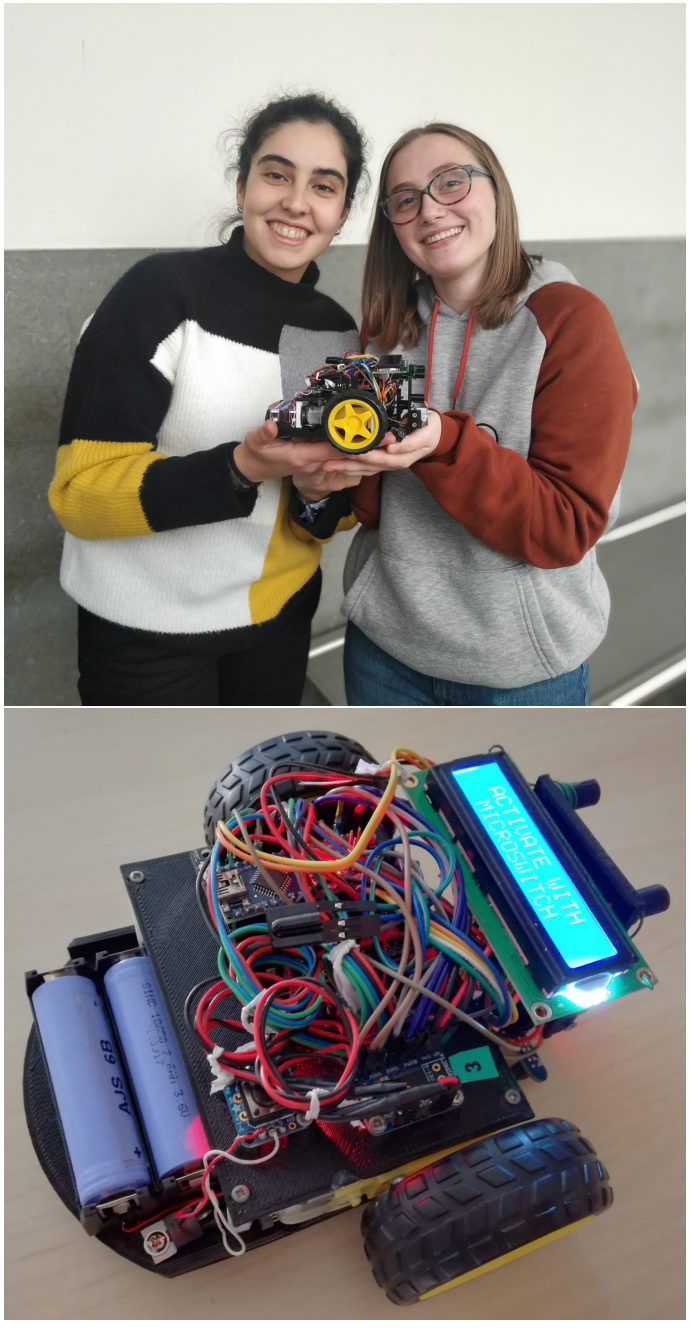
Robô seguidor de linha

Teresa Pintão e Leonor Gothen
(up201706703@fe.up.pt) (up201704974@fe.up.pt)

Turma A3

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Sistemas Baseados em Microprocessadores



Conteúdo

1 Introdução

No âmbito da unidade curricular de Sistemas Baseados em Microprocessadores, foi realizado um mini projeto que visou estudar e implementar um robô seguidor de uma linha preta espessura em fundo branco.

O robô foi testado numa pista com linha preta de espessura de 18mm. Caso se queira testar numa pista com uma espessura diferente, é sempre necessário atualizar o valor de HALFLINE, o valor de metade da espessura da linha utilizada.

Os extras colocados no robô foram os seguintes: um comando que foi programado com a funcionalidade de decidir o caminho escolhido, ou seja, numa bifurcação escolher a esquerda ou direita; um microswitch para começar e parar o movimento do robô; um LCD onde é apresentada a distância percorrida pelo robô (em cm), calculada com recurso aos encoders, a letra R ou L, indicando qual dos caminhos é que está selecionado para o robô seguir caso haja uma bifurcação e o número de voltas dadas. Este último parâmetro foi guardado na EEPROM. Para além disso, o robô já vinha equipado com um botão que funciona como interruptor para as pilhas, ou seja, liga ou desliga a energia do robô.

2 Contextualização

Para contextualizar os métodos utilizados neste trabalho, será feita uma breve explicação de alguns aspetos do microprocessador ATmega328P, que foi o microprocessador utilizado neste trabalho laboratorial através da utilização de um Arduino Nano.

2.1 ATmega328P

Este microprocessador tem 3 portas bidirecionais - PB (8 pinos), PC (7 pinos) e PD (8 pinos) -, sendo que cada uma destas portas tem três registos - DDR, PORT e PIN. O registo DDR permite configurar cada pino como entrada (1) ou saída (0), o registo PORT permite controlar as saídas e o registo PIN permite controlar as entradas.

O ATmega328P tem EEPROM (Electrically Erasable Programmable Read-Only Memory), ou seja, é possível guardar valores mesmo quando é feito um reset do microprocessador. É provocada uma paragem do CPU durante 2 ciclos do relógio para a escrita nesta memória e de 4 ciclos para a leitura. Deve-se ter em conta o desgaste da EEPROM, pois, mesmo havendo leituras ilimitadas, esta memória tem um limite de apagamentos ou escritas com um apagamentos finito ($> 10^5$).

2.2 PWM

PWM (Pulse Width Modulation) refere-se ao conceito de pulsar sinal digital a uma determinada frequência um , ou seja, é uma maneira de codificar digitalmente níveis de um sinal analógico. Portanto, um sinal PWM é uma onda quadrada que tem um determinada duty-cycle.

Neste trabalho, o OCR0A e o OCR0B definem o duty-cycle do PWM que é colocado em cada um dos motores. Ou seja, quanto maior o OCR0, maior será o duty-cycle da onda, ou seja está ativo durante um maior intervalo de tempo num período da onda, e consequentemente a velocidade do motor irá ser maior.

Por exemplo, se um PWM tem um duty-cycle de 10% da energia de entrada, e se a alimentação for de 10V, teremos um sinal analógico de 0,1V.

2.3 Timers e Interrupções

O microprocessador permite fazer interrupções ao seu funcionamento através de ISRs (Interrupt Service Routines). Estas interrupções podem ser associadas a vários acontecimentos.

O hardware Timer1 é utilizado para fazer contagens de tempo, sendo possível configurar as interrupções associadas de forma a terem uma periodicidade específica. Para isso, é necessário ter em conta a frequência a que o microprocessador está a operar e quantos ciclos do relógio devem ser contados antes de haver uma interrupção.

Existem 3 Timers, o Timer0, Timer1 e Timer2, que podem ser utilizados em 2 modos diferentes: modo normal ou modo CTC (clear timer on compare). Um Timer conta pulsos internos (derivados do CLK) ou externos (nos pins Tn), e pode gerar um pedido de interrupção quando ocorre overflow do TCNTn ou a um valor específico do TCNTn, guardado em OCRnA ou OCRnB. (n=0,1,2 identifica o Timer). Neste trabalho, vai ser utilizado o Timer1 no modo normal para os encoders e o LCD, e o Timer0 em modo CTC (clear timer on compare match) para o PWM.

2.3.1 Para PWM

O modo CTC permite fazer uma interrupção quando a contagem de ciclos chega ao valor definido em OCROA ou OCR0B, começando a contar a partir de um valor definido em TCNT0 pelo programador na função init_pwm().

TCNT0 - Contador de impulsos com 8 bits (pode contar até 256).
TCCR0A - define o modo de trabalho do timer.
TCCR0B - pode parar o timer e começá-lo com o TP pretendido.
TIFR0 - Timer 1 Flag Register.
TIMSK0 - permite pedidos de interrupção individuais.

O OCR0A e o OCR0B vão definir a velocidade da roda do motor A e da roda do motor B, respetivamente. Os seus valores são definidos nas funções set_speed_A e set_speed_B, sendo é escolhido o valor da percentagem da velocidade e a função coloca em OCR0A ou OCR0B o valor da multiplicação dessa percentagem por 255 (valor onde ocorre overflow dos OCR0).

2.3.2 Para encoders e LCD

O modo normal permite fazer uma interrupção quando a contagem de ciclos do relógio dá overflow (neste caso quando chega a 65536), começando a contar a partir de um valor definido em TCNT1 pelo programador na inicialização do Timer1, na função init_timer1().

TCNT1 - Contador de impulsos com 16 bits (pode contar até 65535).
TCCR1A - define o modo de trabalho do timer.
TCCR1B - pode parar o timer e começá-lo com o TP pretendido.
TIFR1 - Timer 1 Flag Register.
TIMSK1 - permite pedidos de interrupção individuais.

Para definir o valor de BOTTOM (valor inicial de TCNT1) , é preciso definir de quanto em quanto tempo é que acontece a rotina de interrupção (ISR - Interrupt Service Routine) e saber qual é a frequência interna do clock. Neste caso, a frequência de clock (F_{clk}) é de 16MHz, e decidiu-se fazer uma interrupção a cada 0.05ms (T_{intr}=0.05ms, base de tempo de 0.05ms). Para definir BOTTOM, é necessário utilizar a fórmula seguinte:

$$CP * TP * CNT = \frac{T_{intr}}{\frac{1}{T_{clk}}}, \text{ CP - pré-divisor do CLK, TP - pré-divisor do timer}$$

Sendo que CP=1, por defeito, neste caso $TP * CNT = \frac{0.05ms}{\frac{1}{16M}} = 800$, ou seja, para uma base de tempo de 0.05ms à frequência de 16MHz, é necessário contar 800 impulsos, o que é possível com o Timer1, visto que tem 16 bits (65535). Utilizando esse Timer, TP poderá ser igual a 1, 8, 64, 256 ou 1024. Para não existir um erro de arredondamento no cálculo do CNT, são eliminadas as opções de TP=1024, TP=256 e TP=64. As outras 2 são possíveis, mas deve-se escolher um TP o maior possível, porque um pré-divisor maior significa uma frequência mais baixa, o que implica uma menor potência, de maneira que foi escolhido TP=8, logo CNT=100. Portanto, BOTTOM vai ser definido com o valor 65436, de maneira a que até dar overflow (aos 65536), são contados 800 impulsos (CNT=100).

Para obter uma interrupção no tempo concreto, basta configurar o valor inicial de contagem (TCNT1=BOTTOM=65436=65536-CNT) e repor o valor inicial no final de cada contagem, utilizando para isso a rotina de atendimento da interrupção por overflow do TCCR1A.

A rotina de interrupção (ISR(TIMER1_OVF_vect)), a cada 0.05ms, faz o update da informação dos encoders, conta a distância pela roda A e B, com o posA e posB, respetivamente, e repõe o valor inicial da contagem (TCNT1=BOTTOM). O timer é inicializado na função init_timer1().

2.3.3 Para ADC

Esta interrupção serve para ler repetidamente os sensores de infravermelhos utilizados, um a um, de maneira a termos a informação correta e atualizada, a uma determinada frequência, do estado dos sensores. São lidos os sensores de infravermelhos para o seguimento da pista (pinos 1 a 3, ou seja do segundo ao quarto sensor, pino 0, primeiro sensor e pino 6, o quinto sensor), através da função read_analog. É também lido, com essa mesma função, o pino 7, onde foi ligado o sensor infravermelho que controla o comando.

A cada interrupção, o resultado da leitura dos sensores mencionados acima, é passado para a variável sensx (sensx=ADC), um sensor de cada vez, e à medida que é recebido um resultado, este é passado para o vetor IR_sensors (com a função read_sensors), utilizado depois para fazer o robô seguir a linha, com as funções follow_line_left e follow_line_right, e para a escolha do modo de seguimento com o comando.

2.4 Encoders

Os encoders são constituídos por dois sensores e um disco rotativo. O disco tem ranhuras que permitem obter diferentes leituras nos sensores, ou seja, os sensores lêem 0 ou 1, dependendo da posição do disco.

A sequência de 0s e 1s apresentada pelos encoders permite saber o sentido do movimento do motor e quantificá-lo. As sequências do movimento no sentido positivo e sentido negativo dependem de qual dos encoders está em qual posição e portanto deve ter-se isso em conta aquando da implementação de uma solução.

As sequências possíveis são 00 - 01 - 11 - 10 e 10 - 11 - 01 - 00, sendo cada par de bits a informação dos dois encoders, ou seja, o primeiro bit é informação de um encoder e o segundo bit é informação do outro encoder. Uma das sequências será para o movimento no sentido positivo e a outra será para o sentido negativo.

Dessa forma, incrementa-se a posição do motor quando há uma passagem de um elemento para o seguinte da sequência positiva e decrementa-se quando há uma passagem de um elemento para o seguinte da sequência negativa.

Para obter o deslocamento do robô, pode ser usado o perímetro da roda do robô, pois um perímetro da roda corresponde a uma rotação do motor. Também é possível obter o deslocamento empiricamente, testando qual a posição do motor após um deslocamento pré-definido.

2.5 I2C

O I2C (Inter-Integrated Circuit) é um protocolo de comunicação série (leitura de um bit de cada vez), utilizado na conexão de microcontroladores com outros dispositivos.

Neste trabalho, o I2C foi utilizado para interligar o micro ATmega328P (toma o papel de master) com o LCD (papel de slave). É um protocolo síncrono, ou seja, com sinal de relógio específico gerado pelo master, comandando a troca de dados com o slave.

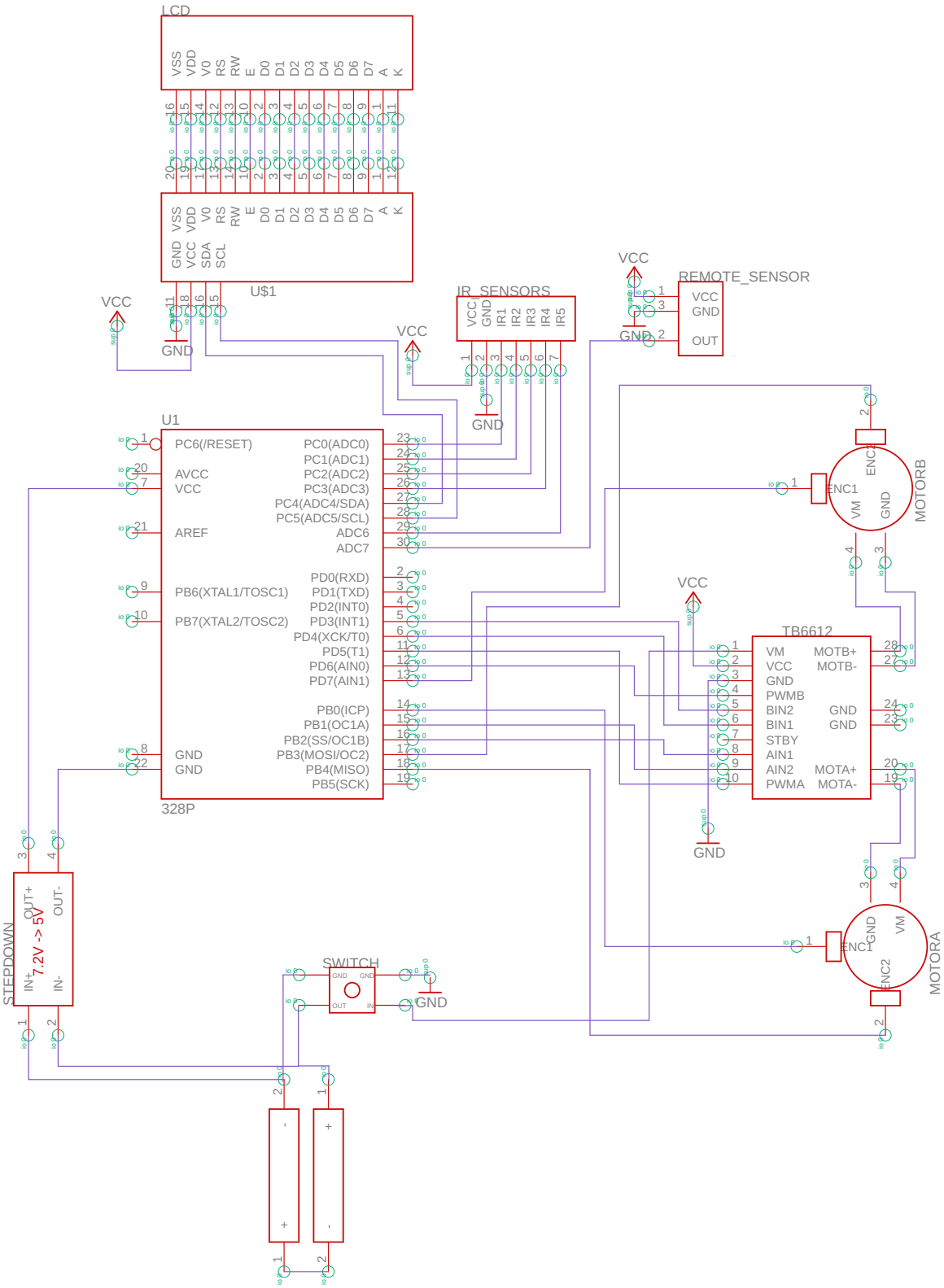
3 Procedimento

3.1 Hardware

3.1.1 Material

- 1 Arduino Nano;
- 1 Shield para Arduino Nano;
- 2 motores DC com encoders;
- 6 Sensores de infravermelhos;
- 1 Driver de motores;
- 1 *stepdow*;
- 1 LCD;
- 1 módulo I2C;
- 1 Interruptor fim de curso;
- 1 Suporte para pilhas;
- 2 pilhas;
- 1 botão;
- Parafusos;
- Fios.

3.1.2 Esquemático



3.2 Software

3.2.1 Funções

Criaram-se as seguintes funções:

1. **init_interrupts()**
   Esta função ativa o bit do *Status Register* que permite a execução de interrupções.

2. **init_IO()**
   Nesta função os pinos são definidos como inputs ou outputs, utilizando os DDR's B, C ou D. Os bits configurados como 1 correspondem a saídas (motores) e os bits configurados como 0 correspondem a entradas (encoders, sensores e touchswitch)

   Os pull-ups das entradas são ativados, colocando a 1 os PORT's dos respetivos pinos, e é definida a direção de rotação dos motores, utilizando os PORTS's B ou D que estão ligados ao driver de motores.

3. **read_analog(uint8_t bit)**
   Esta função é utilizada na interrupção "ADC_vect"para fazer a transformação de analógico para digital na leitura dos 6 sensores de infravermelhos (5 para seguimento da linha e 1 para controlo do comando). Na rotina de interrupção ISR(ADC_vect), são lidos os estados dos 6 sensores através desta função.

4. **init_analog()**
   Esta função define os parâmetros para executar corretamente a ISR(ADC_vect) e inicializa a função read_analog para ler o primeiro sensor.

5. **init_pwm()**
   Nesta função, são definidos os parâmetros para definir a onda do pwm (pulse width modulation).

6. **set_speed_A(float v) e set_speed_B(float v)**
   Nesta função, é definida a velocidade da roda A ou B através de OCR0A ou OCR0B, respetivamente, cujo valor, vai ser definido pelo programador. Isto é, o programador escolhe um valor v, entre o e 100, e o OCR0A ou OCR0B vão ser definidos com uma percentagem de 255, definida pelo valor v.

7. **follow_line_right(int *IR_sensors)**

   Cálculo do Erro
   Para calcular o erro associado à posição do robô em relação à linha, foram utilizados dois sensores para cada caso de seguimento. No caso do seguimento da linha pela esquerda foram os sensores 2 e 3 e no caso de seguimento da linha pela direita foram os sensores 3 e 4. A numeração dos sensores, apresentada anteriormente, foi feita considerando a contagem dos sensores de 1 a 5, sendo o 3 o sensor do meio.

   Para seguir a linha, o sensor 3 deve estar a transmitir um valor muito baixo (abaixo de 100) e o outro sensor, 2 ou 4, deve estar a transmitir um valor muito elevado (acima de 920).

   Considerando a transição da linha preta para o fundo branco como cinzento (cerca de 512) e sabendo a distância entre sensores, podem obter-se relações entre os valores transmitidos e as posições dos sensores.

Assim, é possível essas relações para calcular o desvio do robô em relação ao centro da pista, utilizando também o valor da largura da pista.

Os valores limites definidos para BLACK, WHITE e GREY são 100, 920 e 512, respetivamente. IR_D é a distância entre cada sensor e HALFLINE é o valor de metade da largura da linha.

Quando todos os sensores são expostos a branco, ou seja, todos os IR_sensors≥white, a velocidade das rodas é colocada a zero. Quando os sensores exteriores (0 e 4) ambos são expostos a preto (IR_sensors[x]≤black, em que x=[0,4]), quer dizer que o robô passou pela meta, e é contada mais uma volta. A velocidade das rodas é a VBASE (20%).

Erro Negativo
Caso seja uma curva à esquerda (vbase+error<SPEED_MIN e vbase-error>SPEED_MAX) → roda A com SPEED_MIN e roda B com SPEED_MAX. Caso seja para endireitar para a esquerda, a velocidade da roda A diminui para VBASE+erro.

Erro Positivo
Caso seja uma curva à direita (erro positivo e VBASE+error>SPEED_MAX e vbase-error<SPEED_MIN) → roda A com SPEED_MAX e roda B com SPEED_MIN. Caso seja para endireitar para a direita, a velocidade da roda B diminui para VBASE-erro.

8. **follow_line_left(int *IR_sensors)**
   Esta função segue o mesmo raciocínio que a função *follow_line_right*, inclusive para o cálculo do erro, mas para o lado oposto:

   Erro Negativo
   Caso seja uma curva à direita (erro positivo e VBASE-error>SPEED_MAX e VBASE+error <SPEED_MIN) → roda A com SPEED_MAX e roda B com SPEED_MIN. Caso seja para endireitar para a direita, a velocidade da roda B diminui para VBASE+erro.

   Erro Positivo
   Caso seja uma curva à esquerda (VBASE-error<SPEED_MIN e VBASE+error>SPEED_MAX) → roda A com SPEED_MIN e roda B com SPEED_MAX. Caso seja para endireitar para a esquerda, a velocidade da roda A diminui para VBASE-erro.

9. **read_sensors(int *IR_sensors)**
   Esta função faz a atualização dos dados dos sensores de infravermelhos.

   Os sensores para seguimento de linha transmitem um valor muito alto quando expostos a branco, sendo utilizado o valor 920 como o valor mínimo para identificar branco, transmitem um valor baixo quando expostos a preto, sendo utilizado o valor 100 como o valor máximo para identificar preto e transmitem um valor intermédio quando expostos a cinzento, definido como 512.

   O sensor de infravermelho utilizado para o comando remoto transmite valores elevados quando não está a receber nenhum sinal e quando recebe um sinal transmite 0.

10. **init_timer1()**
    Neste função são definidos os parâmetros TCCR1B, TIFR1, TCCR1A, TCNT1 e TIMSK1 da maneira correta para que o Timer1 seja definido no modo normal e permita a interrupção quando ocorre overflow de TCNT1, como é explicado na contextualização.

3.2.2 Bibliotecas

Recorreu-se também às seguintes bibliotecas:

1. **<serial_printf.h>**
   Necessária para facilitar o debug através de "printfs".

2. **<i2c.h>**
   Necessária para a biblioteca do LCD onde é usada a função i2c_send_packet(unsigned char value, unsigned char adress). Tem de ser inicializado na main através da função i2c_init(). Para além destas, esta biblioteca inclui as seguintes funções: i2c_start_condition(), i2c_stop_condition(), i2c_send_byte(unsigned char byte), unsigned char i2c_recv_byte(void) e unsigned char i2c_recv_last_byte(void).

3. **<lcd1602.h>**
   Para escrever no LCD, utilizaram-se as funções lcd1602_clear(), lcd1602_init(), lcd1602_goto_xy (char col, char row), lcd1602_send_string(const char str). Dentro desta biblioteca são utilizadas funções da biblioteca <i2c.h> para a transferência de informação para o LCD.

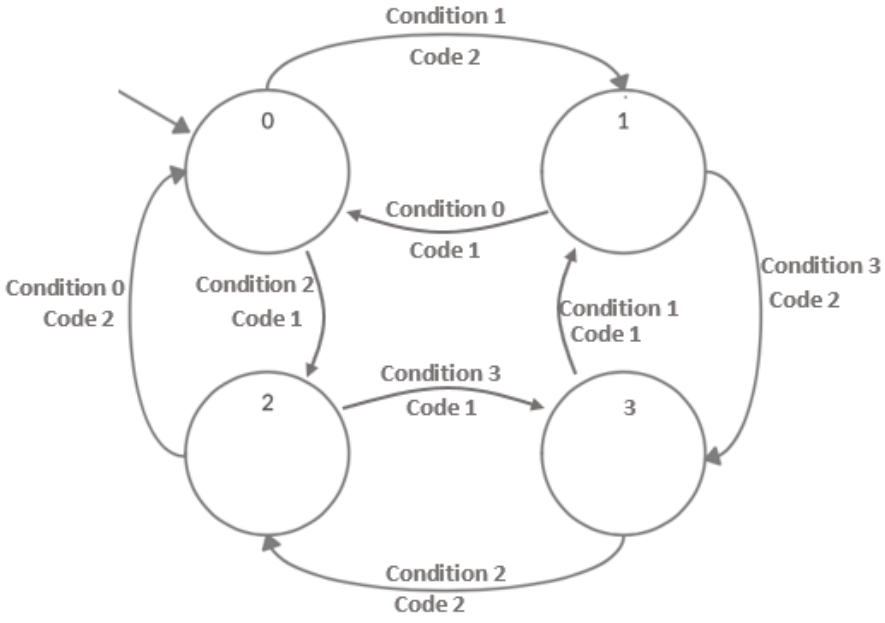
4. **<stdlib.h> e <string.h>**
   Necessárias para se poder utilizar as funções strcpy(string_destiny, string_source), itoa(int value, char * str, int base), e strcat(string_destiny, string_source).

5. **<avr/eeprom.h>**
   Necessária para se poder utilizar as funções eeprom_read_byte e eeprom_update_byte da variável uint8_t EEMEM laps_eeprom.

3.2.3 Encoders

Para utilizar os encoders, implementaram-se máquinas de estados.

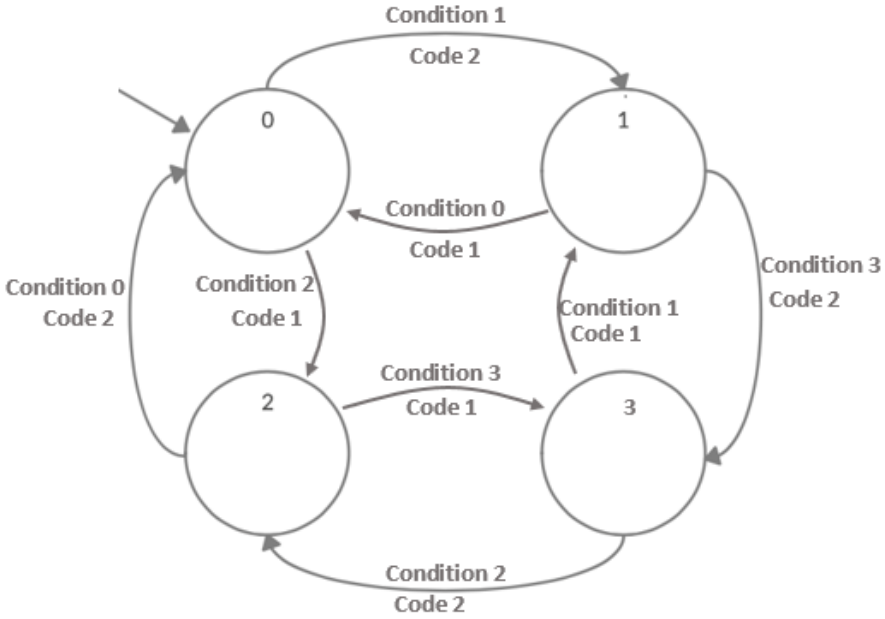
Motor A State



Condition 0: (enc1[0]==0) && (enc2[0]==0)

Condition 1: (enc1[0]==0) && (enc2[0]==1)
Condition 2: (enc1[0]==1) && (enc2[0]==0)
Condition 3: (enc1[0]==1) && (enc2[0]==1)
Code 1: {posA++;}
Code 2: {posA−−;}

Motor B State



Condition 0: (enc1[0]==0) && (enc2[0]==0)
Condition 1: (enc1[0]==0) && (enc2[0]==1)
Condition 2: (enc1[0]==1) && (enc2[0]==0)
Condition 3: (enc1[0]==1) && (enc2[0]==1)
Code 1: {posB−−;}
Code 2: {posB++;}

A diferença entre o local de incrementação e decrementação nas máquinas de estados deve-se apenas à diferença das ligações físicas.

3.2.4 Comando

Foi utilizado um sensor de infravermelhos TSOP31238 para receber o sinal do comando. Este sensor foi ligado a uma entrada analógica do Arduino Nano e verificou-se que, quando o sensor deteta o sinal do comando, transmite o valor 0, enquanto que quando não deteta nada, transmite um valor diferente de 0. Assim, considerou-se que sempre que o sensor transmitia 0, o estado do robô deveria mudar de maneira a alternar entre as funções "follow_line_right"e "follow_line_left".

4 Conclusão

O objetivo principal deste trabalho, fazer o robô seguir a linha, foi cumprido com sucesso. Para além disso, foram adicionados alguns extras, referidos ao longo do relatório, nomeadamente, o LCD, o comando e o microswitch.

Durante a execução deste trabalho foi possível aplicar os conhecimentos adquiridos no âmbito desta unidade curricular, nomeadamente, memória não volátil, interrupções, PWM, ADC, EEPROM e como programar um microprocessador, permitindo melhorar o nosso entendimento destes assuntos.

5 Agradecimentos

Agradecemos ao Professor Engenheiro Armando Sousa por ter cedido um robô da competição Robot@Factory no qual foram feitas as modificações necessárias ao projeto.

Agradecemos também ao Professor Engenheiro Paulo Costa pela ajuda para entender o funcionamento dos encoders e ao Professor Engenheiro Pedro Tavares pelo apoio e conselhos que deu durante as aulas.

6 Referências

Slides apresentados nas aulas teóricas
http://www.atmel.com/devices/atmega328p.aspx
https://www.citisystems.com.br/pwm/
https://paginas.fe.up.pt/ hsm/docencia/comp/spi-e-i2c/