Faculdade de Engenharia da Universidade do Porto

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Robô Seguidor de linha

Fábio Morais up201504257
Alberto Santos up201708979

21 de Dezembro de 2018

Abstract

In this assignment, the main purpose was to develop and implement a line follower robot that would have to follow the line as fast as possible. For that we projected a microcontroller based system with a user friendly mobile application.
With this project all the topics that were given during the lecturers were covered since the interruptions on ATMEGA to the Analog/Digital converter.
More specifically, we used the timer in PWM mode to control the motors and the USART protocol for the Bluetooth communication. The A/D converter was used to get the battery level and the EEPROM to record the track.
We consider this type of project very useful because it gives us knowledge in embedded systems in general but also because it gives the possibility to improve our working methods like project and time management.

Tags: line follower robot, Embebed systems, Microchips, Atmega328p

Índice

Abstract ..... **ii**

Índice ..... **iii**

Introdução 4

Materiais utilizados 4

APP 6
2.1 - Introdução 6
2.2 - Implementação com robô 7

Robô 11
3.1 - Introdução e funcionalidades 11
3.2 - Timers 13
3.3 - Implementação Segue Linha 14
3.4 - Implementação Bluetooth 17
3.5 - Implementação LCD 18
3.6 - Implementação Conversor A/D 20
3.7 - Implementação EEPROM 22
3.8 - Implementação Modo Manual 24

Discussion **25**

Anexos **26**

Introdução

Este projeto tem como principal objetivo desenvolver um robô seguidor de linha preta, em que teríamos de colocar o robô a seguir o mais rápido possível essa linha, com isso poderíamos colocar diversos extras para o projeto ficar mais completo e ser mais valorizado.

Após termos projetado o nosso robô decidimos que queríamos arriscar muito mais para além de um simples seguidor de linha, com isso desenvolvemos uma APP para o nosso robô e diversas funcionalidades interessantes, de maneira a ser um projeto bastante completo.

Com este projeto abordamos os temas todos, desde as interrupções à matéria do conversor A/D, colocando diversas funcionalidades que nos pareceu pertinentes de acordo com cada tema.

Na época em que vivemos muita gente troca o comando pelo smartphone, logo faria todo o sentido criarmos uma app para o nosso robô, para alem de tornar o projeto mais atrativo e interessante, pois até permite que seja um brinquedo para uma criança brincar, pois tem uma interface bastante intuitiva para o utilizador.

Tentamos também que o aspeto do nosso projeto seja bom, não ter os fios todos desorganizados, tendo cada componente a respetiva cor de fio.

Iremos implementar o timer no modo PWM para conseguirmos controlar os motores, comunicação USART para o Bluetooth, conversor A/D para conseguirmos ver o estado da bateria e a EEPROM para gravarmos o trajeto. Irá ser explicado em cada secção o respetivo detalhe de implementação.

Fazer este tipo de projetos é uma oportunidade excelente, tanto para melhorar o nosso conhecimento como para fazer currículo e por isso demos o nosso melhor para desenvolver este projeto.

1.Materiais utilizados

1.1 Arduino Uno
1.2 LCD Module Display 20x4
1.3 Modulo Bluetooth HC-06
1.4 Array de 5 sensores infrared
1.5 Motor DC 3V-6V c/Roda
1.6 Dual channel DC motor driver
1.7 Placa acrílico
1.8 Suporte 3x pilhas AA
1.9 Power Bank 4000mah
1.10 Interruptor ON/OFF
1.11 LED azul e vermelho

A pista foi desenvolvida recorrendo a 4 folhas A2, pintadas a spray preto mate.

- ✓ Largura do traço = 3,8cm
- ✓ A pista tem dimensões 1mx1,2m

O circuito elétrico irá ser disponibilizado em anexo, assim como a foto de alguns dos componentes utilizados, do protótipo da pista e tabela de bytes que será enviado/recebida via bluetooth.

Irá ser disponibilizado em anexo links que demos upload para uma drive sobre as funcionalidades do nosso robô em video.

Fig 1. Código QR da aplicação

2.APP

2.1 Introdução

A APP foi desenvolvida dando uso da ferramenta ***MIT App Inventor***, uma plataforma online em que é possível criar aplicação para smartphone de forma muito simples, recorrendo a programação em blocos, sendo muito intuitivo para todos.

Nesta plataforma existe 2 processos diferentes para criar a APP, a parte do *Designer* que consiste na interface do utilizador, em que é possível criarmos os botões, luzes, textos…etc

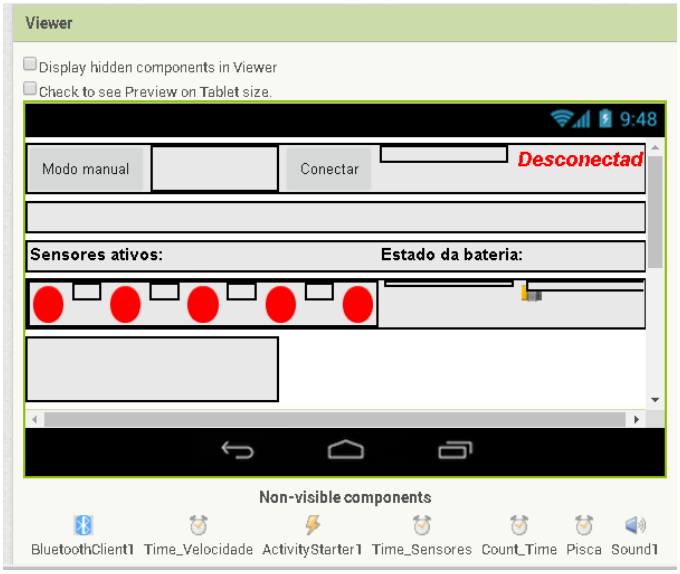

Fig 2. Exemplo do Designer, interface com o utilizador

Temos a parte da programação, chamada de *Blocks* em que com o Designer que criamos vamos colocar a APP funcional.

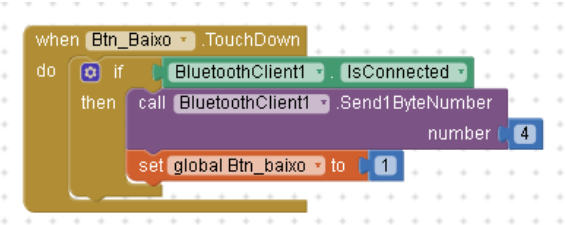

Fig 3. Exemplo de código para enviar dados via Bluetooth

Nesta plataforma é possível usar o Bluetooth para envio de dados como por wi-fi, foi escolhida o Bluetooth para não termos a limitação de termos de ter uma rede wi-fi por perto e assim tendo uma conexão mais estável para o nosso projeto, tendo um alcance de 10m.

Este projeto foi bastante ambicioso, pois com a APP temos o controlo absoluto do nosso Robô, sabendo exatamente tudo o que está a fazer.

A principal ideia ao termos escolhido fazer esta APP foi conseguirmos ter o controlo total do robô e por ser a maneira mais intuitiva e simpática de lidarmos com um robô. Estamos numa altura em que cada vez mais controlamos tudo pelo telemóvel, então faria todo o sentido fazermos o mesmo neste projeto.

Era possível termos a informação completa do nosso Robô sem estar perto dele, utilizando para isso a rede wi-fi, apenas teríamos de colocar o modulo Wi-fi no nosso projeto.

Outra ambição que queríamos implementar no nosso robô, mas que não tivemos tempo foi utilizar o sensor acelerômetro do telemóvel, sendo assim possível controlar o Robô fazendo movimentos com o telemóvel. Poderíamos usar esse sensor dando ativando essa opção no ***MIT App Inventor***.

2.2 Implementação com o robô

A APP, como já foi referido, vai comunicar via Bluetooth com uma Baud Rate de 9600, o que significa que o canal pode mudar o estado até 9600 vezes por segundo.

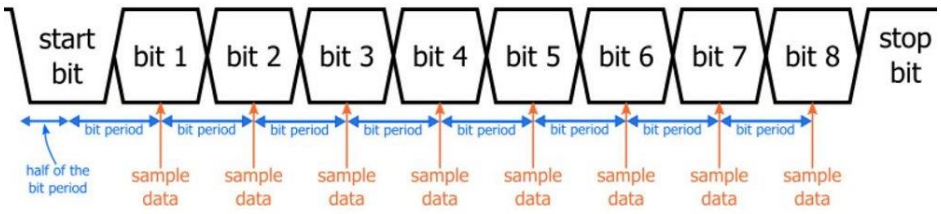

Fig 4.Formato de um trama de comunicação

Com este valor de Baud Rate default do HC-06, caso esteja constantemente a trocar de informação então vai ser ocorrido a cada 1ms no máximo.

Então foi definida na construção da APP que a cada 1ms tinha de verificar se havia algum byte para ser lido, desta forma evitamos que a APP fique carregada com bytes que deviam de já terem sido lidos.

A APP está dividida em 3 secções, a parte do Modo Manual, Modo Automático e os créditos.

Iremos disponibilizar um código QR em anexo para fazer download da APP, não garantindo que fique totalmente formatado em todos os smartphones.

- **Modo Manual**

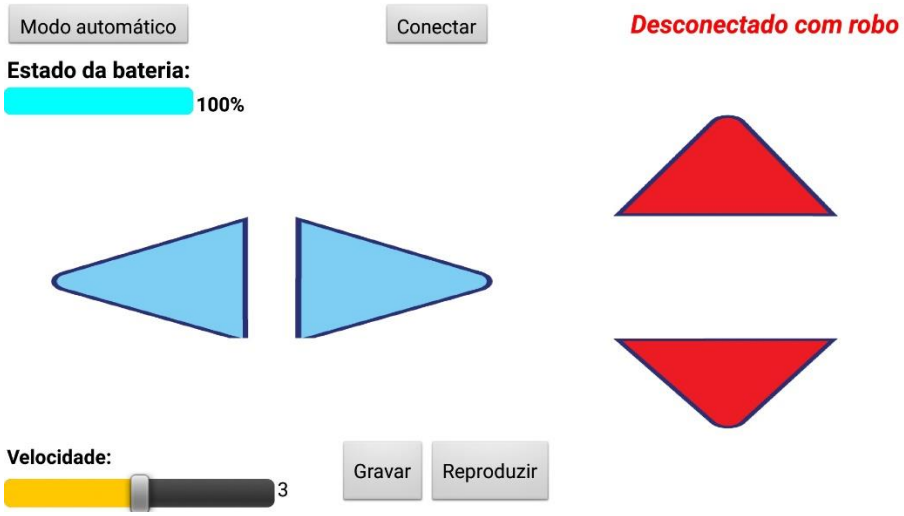

Fig 5.Modo Manual

Neste modo é onde é possível controlar o nosso robô de forma manual, onde é possível alterar a sua direção e velocidade.

Podemos saber neste ecrã se o robô está desconectado ou conectado ao robô e o estado de bateria do equipamento, caso o robô atinja menos de 20% de bateria então é avisado neste ecrã, piscando o nível de bateria, sinalizando que precisamos de mudar de pilhas.

É possível gravar e reproduzir o trajeto que efectuamos, sinalizando quando se está a gravar ou a reproduzir. Mais à frente iremos falar detalhadamente o que este modo faz.

Fig 6 Botões de gravar e reproduzir

Quando carregamos em algum dos botões de gravar/reproduzir é transmitido a informação do que está a ser feito, sendo impossível gravar e reproduzir ao mesmo tendo, evitando assim possíveis bugs.

- **Modo Automático**

Fig 7 Modo Automático

Neste modo é onde é recebida a informação quando o robô está a seguir a linha.

É possível saber os sensores ativos, quando o robô deteta a linha preta então o círculo passa para verde, sinalizando assim o reconhecimento da linha preta, o estado dos sensores é atualizado entre 15ms a 55ms, sendo quase instantâneo a mudança de estado.

Temos também a informação do estado da bateria e mais uma vez quando a bateria atinge menos de 20% é sinalizado a informação que é necessário mudar de pilhas quando a barra começa a piscar.

Podemos mudar a velocidade do robô, dar Start e Stop.

É transmitida a informação do numero de voltas que o robô fez e o tempo decorrido, quando completa uma volta é comparado com o melhor tempo já feito, caso tenha sido um tempo mais rápido então é colocado esse tempo na parte do "MELHOR" .

Caso seja premido o botão Start o tempo começa a contar, caso carregue em Stop o tempo é colocado em pausa.

É possível também dar reset ao melhor tempo e dar reset ao tempo decorrido no momento.

Quando o robô se perde da pista durante 2s é sinalizado um aviso na APP, tremendo o telemóvel e aparecendo uma mensagem de aviso.

Fig 8. Aviso de robô perdido

Neste modo é também possível gravar o trajeto que faz e posteriormente reproduzir os movimentos que fez.

- **Créditos**

Por último, temos a parte dos créditos, em que está os autores do trabalho e agradecimentos aos professores. Nesta parte há uma hiperligação também para o nosso linkedin.

Fig 9. Créditos

A comunicação entre o robô e APP vai ser através do envio e receção de 1 byte de dados.

3.Robô

3.1 Introdução e funcionalidades

O principal objetivo deste projeto é fazermos um robô segue linha, para isso vamos recorrer a um array de sensores para podermos detetar a linha e tomar as devidas ações.

Para além de ser um segue linha tínhamos ambição de fazer mais, de ficar um projeto completo e robusto.

Do ponto de vista do robô este vai oferecer várias funcionalidades, um modo que pode ser conduzido via telemóvel (Modo manual), um modo que segue a linha e transmite diversa informação tanto via LCD como Bluetooth (Modo automático) e por ultimo, temos o modo competição em que o robô desliga o LCD, Bluetooth e apenas fica projetado como um segue linha simples, colocando o robô na velocidade máxima e tendo reações mais rápidas, neste modo o erro de se perder é maior, devido a atingir velocidades maiores. A variável que vai conter a informação sobre o modo é a variável `Modo_Robo`, esta pode ter estes 3 modos

No modo automático o robô quando não encontra a linha vai fazer o seu ultimo movimento, desta forma é quase garantido que vá encontrar uma linha, caso não encontre em 2s então o robô fica no estado "perdido", em que os motores são parados, acende os leds a piscar e imprime no LCD a mensagem "Robot perdido", para alem disso recebemos via Bluetooth esse aviso, permitindo assim ao utilizador saber o estado do robô como foi explicado na secção da APP.

É possível também sabermos o nível de bateria em tempo real, tanto via telemóvel, como no LCD, caso o nível de bateria atinja menos de 20% então os motores são desligados, imprime a mensagem "Bateria fraca \n Mude as pilhas" e recebemos via telemóvel essa informação como foi explicado mais uma vez na secção APP.

Conta o número de voltas quando o array de sensores contem o valor 10101, que é a marca que o robô conhece ser a meta, imprimindo o numero de voltas via LCD e telemóvel.

Para alem destas funcionalidades, implementamos mais uma que achamos bastante interessante e com bastante potencial, pois conseguimos gravar o trajeto que o robô faz, tanto no modo manual, como no modo automático e depois reproduzimos as vezes que quisermos. É gravado na EEPROM e por isso é uma memória não-volátil em que o circuito fica guardado mesmo sem energia. Esta funcionalidade é interessante na medida que podemos gravar um circuito próprio sem necessidade de termos uma pista.

- **LCD**

No LCD é imprimida diversa informação sobre a atividade e estado do robô.

Fig 10. Mensagem LCD

- Na primeira linha é imprimido o estado logico dos sensores invertidos, com isso quer dizer que 1 corresponde a linha preta, 0 superfície branca.
- Na segunda linha é imprimido o estado do robô (Parado ou a andar), caso esteja a andar então é imprimido também nesta linha o número de voltas efetuadas.
- Na terceira linha é imprimido o estado do Bluetooth, se está conectado ou desconectado
- Por fim, na última linha é imprimido o estado de bateria, em que desce 10% a cada nível.

No modo manual, quando nos conectamos ao robô via telemóvel é imprimido a mensagem no LCD a dizer que está "conectado", se por alguma razão é perdida a ligação então é imprimido a mensagem "desconectado" e os motores são parados, para prevenir que ele fique descontrolado se tivesse desconectado.

Fig 11. Mensagem LCD desconectado

Fig 12. Mensagem LCD conectado

Quando estamos a reproduzir um trajeto gravado e chega ao fim da gravação é imprimida a mensagem "Reprodução acabada" parando os motores à espera de novos comandos.

- **LEDS**

Todos os comandos e estado do robô é sinalizado por LEDs que também tem um papel fundamental para percebermos o que o robô quer transmitir, sem precisarmos de ler qualquer tipo de aviso.

Estado Run	LED azul estático
Estado Stop	LED vermelho pisca
Robô perdido	LED vermelho pisca a cada 0,1s
Desconectado via bluetooth	LED azul pisca a cada 0,5s
Bateria fraca	LED azul e vermelho a piscar a cada 1s em simultâneo
Gravar trajeto	LED azul aceso e LED vermelho a piscar a cada 1s
Fim da reprodução	Pisca LED azul a cada 1s

Fig 13. Tabela de funções de cada LED

3.2 Timers

Para o desenvolvimento deste projeto iremos recorrer a 2 timers, um timer PWM que iremos falar mais à frente numa secção focada nisso e outro timer no modo normal, dando uso ao Timer 1 de 16 bits.

Tintr(ms)	Timer Prescaler (TP)						
1	1	8	32	64	128	256	1024
CLK Prescaler(CP) 1	16000	2000	500	250	125	62.5	15.625
2	8000	1000	250	125	62.5	31.25	7.8125
4	4000	500	125	62.5	31.25	15.625	3.90625
8	2000	250	62.5	31.25	15.625	7.8125	1.95313
16	1000	125	31.25	15.625	7.8125	3.90625	0.97656
32	500	62.5	15.625	7.8125	3.90625	1.95313	0.48828
64	250	31.25	7.8125	3.90625	1.95313	0.97656	0.24414
128	125	15.625	3.90625	1.95313	0.97656	0.48828	0.12207
256	62.5	7.8125	1.95313	0.97656	0.48828	0.24414	0.06104

Fig 14. Tabela de valor BOTTOM para TCNT1

Iremos ter este único timer que vai servir para diversas funções, desde as mais rápidas às mais lentas, vai ocorrer uma interrupção sempre que ocorre overflow e com isso incrementamos/decrementamos as respetivas variáveis.

Escolhendo o CLK prescaler e o timer prescaler igual a 1, dessa forma reduz mais a interrupção à main, ficando com CNT = 16000.

3.3 Implementação segue linha

O sensor segue linha é baseado em Led IR e um Photodiode de receção, em que as superfícies brancas refletem o que o Led IR transmite e o Photodiode recebe.

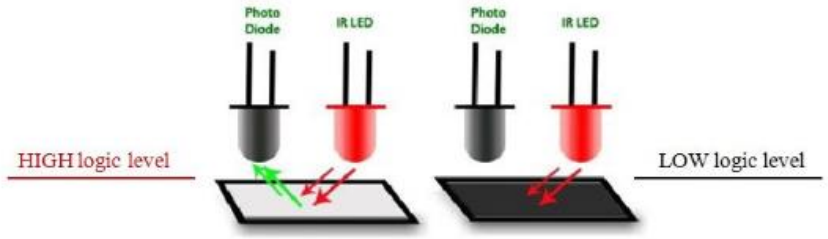

Fig 15. Funcionamento de IR LED e Photodiode

O sensor fica a 1 quando deteta superfície branca e fica a 0 quando deteta linha preta. Inicialmente fazemos uma rotina para colocar num array os valores do sensor ativos no momento, em seguida com os sensores ativos no array tomamos as decisões do movimento do motor.

Para conseguirmos mudar de direção é necessário recorrer ao PWM para regular o nível de tensão em cada motor e assim colocar mais tensão e menos tensão em cada lado. Iremos usar o timer 2 no modo PWM de 0 a 255 ativando a saída OCR2A e OCR2B, como estamos a usar 3 pilhas AA a tensão pode estar no máximo até 5V (assumindo que as pilhas têm uma tensão inicial de 1,7V no início)

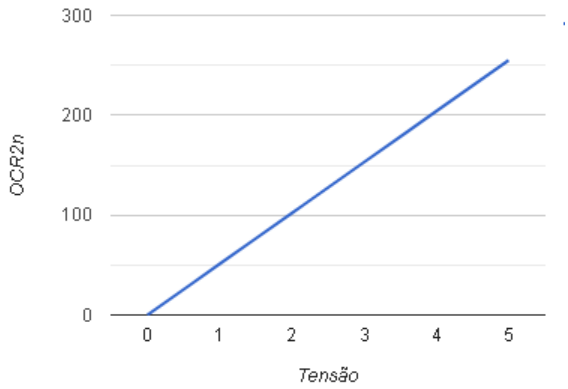

Fig 16. Variação da tensão com o OCCR2n

O motor apenas tem uma gama de 3V a 6V, logo o OCCR2n que faz com que o motor ande seja de 153. Logo a verdadeira gama de valores é com o OCCR2n entre 153 e 255. Esta variação da tensão é dada pela variação do duty cicle que o PWM faz na saída, alterando assim o valor medio da onda.

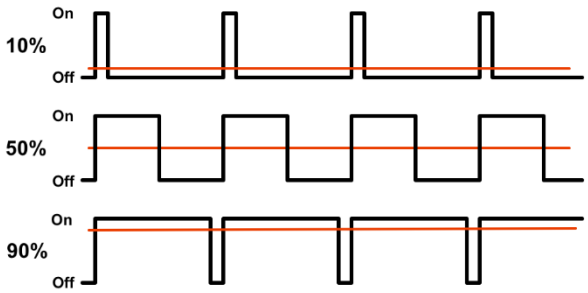

Fig 17. Variação da tensão com o Duty Cicle

Iremos ter um valor de velocidade default, como por exemplo 200.
Quando há uma mudança de direção iremos subtrair ao valor default de velocidade o respetivo erro gerado, fazendo assim que uma roda ande menos que outra, permitindo que o robô vire.
OCR2A corresponde ao motor da esquerda e o OCR2B corresponde ao motor da direita, caso queiramos virar para a esquerda então fazemos OCR2A= 200 – x, em que x corresponde ao erro gerado pela linha detetada, que mais à frente iremos explicar, desta forma colocamos menos tensão no motor da esquerda e consequentemente menos velocidade.
Com isto já temos as direções do motor.

Iremos inverter o valor logico dos sensores para entender melhor, diremos que 1 é quando deteta linha preta e 0 quando deteta superfície branca.
É definida um valor default para as mudanças de direção e é somado x a esse valor, x é um valor constante.

00000	Segue o último trajeto lido durante 2s
00100	Segue em frente
01000	Vira esquerda + x
10000	Vira esquerda + 3x
01100	Vira esquerda
11000	Vira esquerda +2x
11100	Vira esquerda + 4x
00110	Vira direita
00011	Vira direita + 2x
00111	Vira direita + 4x
00010	Vira direita + x
00001	Vira direita + 3x
10101	Deteta 1 volta, segue em frente

Fig 18. Erros de acordo com os sensores ativos

“Vira esquerda” e “vira direita” correspondem a valores defaults, quanto mais no extremo é detetado a linha preta mais terá de virar e por isso definimos 4x, caso apenas seja detetado como por exemplo, 01100 então significa que terá de virar apenas um pouco.

Vamos supor que “vira esquerda” e “vira direita” valem 10, com x=10.

Para o array de sensores 00001 a equação irá ficar (10+3*10), que vai dar um valor de 40, com esse valor subtraímos ao OCR2B (motor da direita), ficando OCR2B= velocidade_default – 40, virando dessa forma para a direita

Esta tabela serviu para termos ideia de como o robô iria reagir em pista, mas simplificamos mais o problema e definimos os valores padrões que foram ajustados por tentativa e vendo o comportamento do robô na pista, colocamos o robô na pista e íamos ajustando os valores até ele conseguir fazer uma trajetória com estabilidade, pois não podíamos linearizar o comportamento.

```
#define Velocidade_Padrao 200
#define Mudanca_Suave_Padrao 15
#define Mudanca_Media_Padrao 20
#define Mudanca_Media_Mais_Padrao 28
```

Fig 19. Valores default

Temos 3 valores de mudança, uns maiores que outros de maneira a colocar o robô estável, caso encontre a pista no extremo dos sensores a robô trava uma roda e vira com a outra, para assim conseguir virar rapidamente, salientar a importância de travar e não apenas colocar OCR2n=0, pois com a inercia do robô poderia sair mais da pista, então ao travarmos garantimos que faz um trajeto correto e rápido.

DC MOTOR	MODE	IN1	IN2	IN3	IN4
MOTOR-A	Forward	1/PWM	0		
	Reversion	0	1/PWM		
	Standby	0	0		
	Brake	1	1		
MOTOR-B	Forward			1/PWM	0
	Reversion			0	1/PWM
	Standby			0	0
	Brake			1	1

Fig 20. Valores default

Com esta tabela fornecida pela fabricante podemos travar um dos motores ao colocar o valor lógico 1 no IN1 e IN2 no caso do motor A. Foi desta forma que implementamos o travão no robô.

3.4 Implementação Bluetooth

Implementamos a biblioteca Bluetooth.h que recorre à comunicação assíncrona, usando as saídas TXD e RXD.

Com um baud rate de 9600 iremos utilizar no modo Duble Speed, com 8 data bits e 1 stop bit, que é a configuração do HC-06.

Para fazermos o envio dos dados iremos recorrer a uma função que verifica se os bits de estado (UDRE0) está a 0 e assim envia, caso ainda não esteja disponível para enviar então fica à espera.

Para recebermos os dados da APP para o robô vamos recorrer à interrupção, para deste modo pouparmos tempo e ser uma tarefa mais rápida.

Na função da interrupção é verificado o que está a ser recebido e coloca nas respetivas variáveis, uma outra implementação que poderíamos ter feito era um buffer circular para esta rotina, em que colocávamos um vetor com por exemplo 64 posições e tínhamos 2 variáveis, uma delas apontava para o elemento que queríamos retirar, outra variável para a posição livre do buffer.

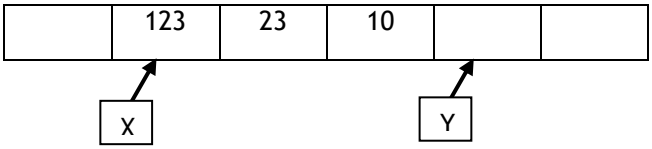

Fig 21. Buffer circular

Neste exemplo X corresponde à variável que aponta para a posição em que do primeiro elemento a ser retirado, Y corresponde à posição de onde devemos de adicionar.

Iremos fazer o envio de dados via Bluetooth a cada 15ms dando uso a variáveis que estão decrementando a cada 1ms do timer 1, ou seja, colocamos na variável 15 e quando esta chega a 0 então envia mais um dado via Bluetooth, iremos enviar dados alternadamente dando uso a flags.

3.5 Implementação LCD

Iremos implementar o LCD de 4bits para poupar fios, dessa forma teremos de fazer uma máscara de bits para conseguirmos enviar de 4 bits em 4 bits.

```
data_value1 = data_value & 0xF0;
lcd_data(data_value1);
_delay_us(10);
data_value1 = ((data_value << 4) & 0xF0);
lcd_data(data_value1);
```

Fig 22. Mascara de bits

Desta forma isolamos os 4 bits da esquerda e enviamos, depois fazemos um shift left dos bits que faltam e enviamos outra vez os 4 bits da esquerda.

Iremos exemplificar através de um esquema todo este raciocínio.

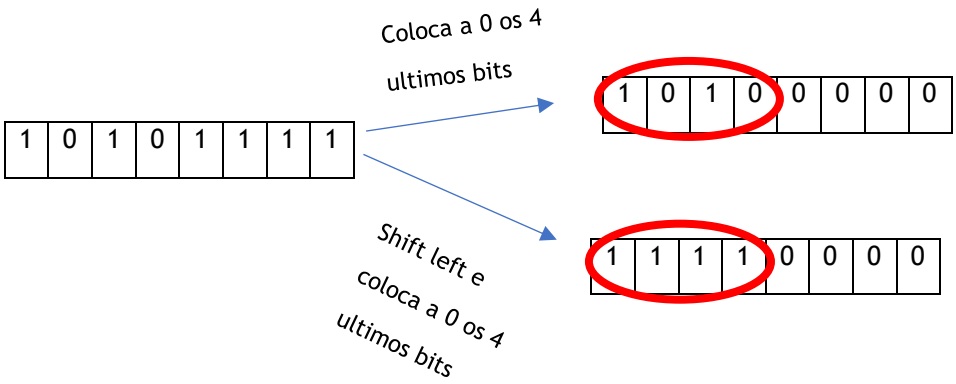

Fig 23. Mascara de bits

Analisando a datasheet do LCD podemos saber alguns dos endereços que podemos enviar, para fazer certas tarefas.

Caso seja necessário enviar um comando para o LCD então é ativado a entrada EN e desativamos o RS e o RW, desta forma entramos num modo em que o LCD deteta que está a enviar comandos para ele.

Este processo é aplicado para quando queremos fazer clean ao ecrã, ativar o cursor…etc

No caso de enviar caracteres é um processo semelhante, só que ativamos em simultâneo o RS e o EN, deixando o RW desativado, desta forma o LCD sabe que está a receber caracteres para imprimir.

Foi desenvolvida uma função que faz o gotoxy, que move o cursor para a posição que queremos, sendo bastante útil para fazermos uma boa formatação de texto, sendo possível dar print linha por linha.

O processo de inicialização do LCD foi feito do seguinte modo:

1. Inicializar LCD no modo 4 bits
2. Dar clean ao LCD
3. Inicializar no modo 4 linhas, 5x7 dots
4. Ativar cursor
5. Colocar o cursor no inicio

Iremos importar a biblioteca <stdio.h> para darmos uso do printf, para tornar o print numa tarefa mais fácil, tendo a noção que irá ocupar muito mais memoria, para fazer isso iremos definir que o nosso stdout irá dar uso à função "lcdData" que criamos, desta forma quando escrevemos o printf esta vai recorrer à nossa função para enviar a string via LCD, enviando caracter a caracter até encontrar o '\0' que indica que a string está terminada.

Vamos dar print no LCD a cada 20ms, recorrendo ao uso de timers, para desta maneira não atrasar no tempo de resposta dos sensores.

Foi desenvolvido desta forma porque a cada envio de caracter para o LCD é perdido um tempo de 2ms, ou seja, para preenchermos o LCD só de uma vez o micro ia ficar parado naquele estado durante 128ms a cada ciclo, preencher 16 colunas e 4 linhas, o que é um tempo enorme e que iria atrasar o tempo de reação.

Iremos imprimir linha por linha, recorrendo a flags e timers, reduzindo assim drasticamente o tempo perdido, sendo quase indiferente e impercetível.

3.6 Implementação conversor A/D

Iremos utilizar o conversor A/D para conseguirmos saber o nível de bateria do nosso robô.

Iremos ter 5V de VREF, pois iremos usar 3 pilhas AA, logo uma tensão máxima de 5V, utilizamos um pré-divisor de 128, pois a gama de valores que teríamos de ter era entre 50khz e 200khz. Com 16Mhz e um pré-divisor de 128 ficamos com 125khz o que é um valor aceitável.

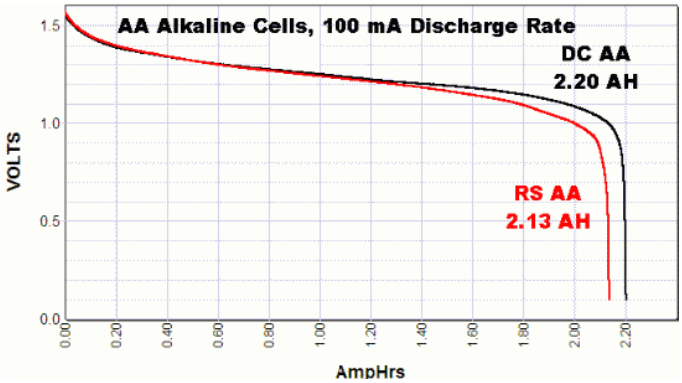

Fig 24. Gráfico da evolução da tensão numa pilha

Com esta evolução do gráfico e multiplicado por 3 pilhas decidimos colocar o nível de bateria a cada 100mv no início e a partir dos 4,3V é a cada 200mv. Ou seja, quando as pilhas têm uma tensão maior que 4,6V o nível de bateria é 10, quando tem 4,5V tem nível de bateria 9…etc O nível de bateria é depois multiplicado por 10, dando o 100% e 90% de bateria.

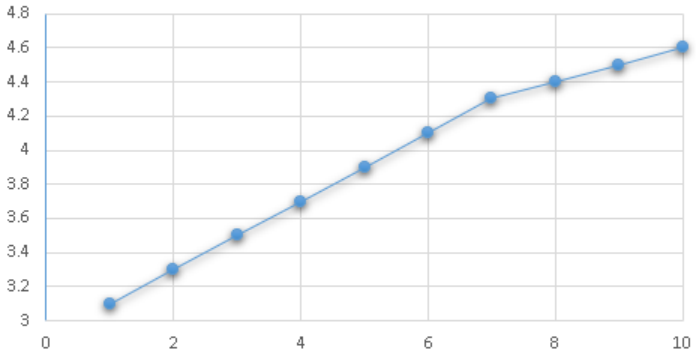

Fig 25. Gráfico da tensão nas pilhas por nível de bateria

Quando o valor de tensão está abaixo de 3,3V é quando é dado o aviso de bateria fraca e desligamos os motores, pois o motor apenas permite andar a partir dos 3V.

Se o nível de tensão está nos 3,1V então significa que o nível de bateria é 0.

Se quisermos implementar um robô que suporta 4 pilhas AA, então teríamos de fazer um divisor de tensão, pois o ATMEGA 328p apenas suporta leituras até 5V, como teríamos 4 pilhas AA iria dar uma tensão máxima de 7V caso cada pilha tivesse uma tensão inicial de 1,7V.

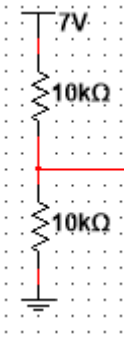

Fig 26. Divisor de tensão com 6V

Agora faríamos a nossa referência em 3,5V e a nossa escala não podia ser de 100mv em 100mv, teríamos de colocar no dobro, ou seja, se a tensão variar X com 3,5V de referência então significa que variou 2X na escala real (7V).

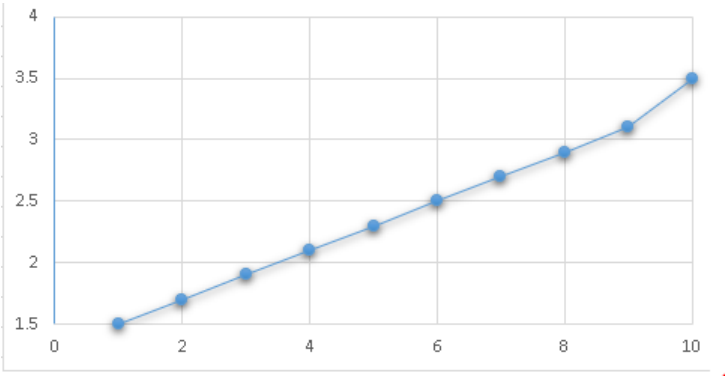

Fig 27. Gráfico da tensão nas pilhas por nível de bateria

Poderíamos implementar a gama de valores desta forma, quando a tensão fosse de 1,5V significava que a tensão real está em 3V, ou seja, é a tensão mínima para o motor, logo teríamos o nível de bateria a 0.

3.7 Implementação EEPROM

Iremos usar a EEPROM para gravarmos os trajetos quando o utilizador carrega no botão "gravar" na APP, deste modo fica guardado mesmo quando o robô está desligado, porque como já referimos é uma memória não-volátil, em que fica os dados guardados mesmo quando não há energia a alimentar.

O método de gravação consiste em gravar a cada movimento diferente, exemplo:

- Robô anda para a frente -> 1º movimento, grava na posição 1
- Robô vira para esquerda -> 2º movimento, grava na posição 2
- ….

Iremos ter 3 vetores declarados na EEPROM, um vetor para os movimentos do motor, outro vetor para a respetiva velocidade de cada movimento e por último um vetor para o tempo em que esteve a fazer o movimento X.

Sempre que vamos atualizar o valor na EEPROM perdemos 3,3ms por vetor, ou seja, iriamos perder cerca de 10ms a cada movimento. Para evitar isso vamos colocar primeiramente em vetores declarados globalmente, quando acabarmos a gravação então é enviado para a EEPROM, desta forma evitamos perdas de tempo na escrita da EEPROM enquanto o robô está a andar.

Existe uma variável de 16 bits declarada que é incrementada a cada 1ms, ou seja, é capaz de incrementar até 65, 535 segundos o que é mais que suficiente para guardar o tempo que esteve a fazer um tipo de movimento. Com essa variável vamos incrementando quando está a fazer o movimento X, quando se muda para o movimento Y então o tempo no movimento X é guardado.

Há uma variável auxiliar Count2 que aponta para a posição onde termina o último movimento efetuado no vetor.

Exemplo abaixo que explica o formato do vetor contido na EEPROM.

	[0]	**[1]**	**[2]**	**[3]**	**[4]**
Movimentos	Count2	Frente	Esquerda	Direita	LIXO
Velocidade	---------	200	230	200	LIXO
Tempo	--------	1000	500	1000	LIXO

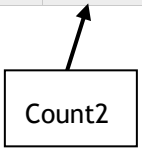

Fig 28. Vetores preenchidos em EEPROM

O primeiro elemento do vetor contem a posição do último elemento + 1 para ser lindo, sempre que carregamos em reproduzir vamos aos 3 vetores da EEPROM e fazemos um ciclo de 1 até Count2 e enviamos para os 3 vetores declarados globalmente e depois é reproduzido o trajeto com esses vetores.

No exemplo anterior indica que na posição 1 esteve a andar para a frente durante 1s à velocidade de OCR2n de 200, depois virou para a esquerda durante 0,5s a uma velocidade de OCR2n de 230 e assim por essa ordem.

Poderíamos ter poupado 2 posições ao preenchermos a posição 0 do vetor velocidade e tempo, porém não se tornava tão intuitivo e por isso decidimos fazer desta forma.

A EEPROM tem uma memoria de 1024 bytes, ou seja, não temos memoria infinita para gravarmos os movimentos todos, então declaramos vetores com 200 posições, sendo 3 vetores e levando a gravação ao limite iriamos ocupar no máximo 800 bytes de memoria na EEPROM. O que significa termos uma limitação de 200 movimentos diferentes que podemos fazer, para além de declararmos os vetores para um espaço de 200 posições também garantimos em todos os ciclos que não excede o valor máximo.

3.8 Implementação Modo Manual

Neste modo a implementação é simples, apenas iremos inicializar o modo manual imprimindo no LCD a informação que está no modo manual e acende a respetiva luz.

Em cada ciclo é imprimido a respetiva informação do modo manual e é enviado via Bluetooth a informação da bateria.

Sempre que há uma interrupção há um timer que começa a decrementar, começando em 2s, caso esse timer fique a 0 significa que perdeu a comunicação, pois a APP envia informação a cada 1s de confirmação. Então caso perca comunicação é imprimido a mensagem de aviso e entra num ciclo de modo de segurança, em que fica parado e acende as luzes, ficando à espera de alguma comunicação via Bluetooth para sair do ciclo.

A implementação para conseguirmos controlar o carro é bastante simples, sendo a APP a parte mais importante, pois terá de enviar um byte X caso carregue num botão e um byte Y caso tire o dedo do botão, para desta forma o robô só andar quando carregamos, ou seja, quando o utilizador carrega no botão é enviado o byte X que significa andar para a frente, como por exemplo, então o robô fica sempre nesse estado, quando tiramos o dedo do botão é enviado o byte Y, que coloca os motores no estado parado, dessa forma conseguimos controlar o robô de forma simples.

A implementação que fizemos foi simples, permitindo apenas 4 direções, frente, trás, esquerda, direita, não sendo possível andar na diagonal para um controlo mais fluido, mas isso teria de ser melhor programado na APP e seria uma implementação fácil, porém achamos desnecessária essa parte.

Discussion

Este projeto serviu para consolidar os nossos conhecimentos sobre todos os temas abordados e permitiu assim estudar para o teste fazendo o nosso projeto, pois demos uso a todos os temas abordados nas aulas.

Surgiu algumas dificuldade e bugs como em qualquer projeto, mas sempre conseguimos superar com êxito.

O nosso maior problema foi fazer uma pista com uma linha preta que o nosso sensor fosse detetado e foi um dos pontos onde perdemos bastante tempo, pois fizemos um protótipo da pista com papel vegetal e passamos para tamanho grande em 4 cartolinas A2 e pintamos com spray preto mate.

Com este projeto ocupamos 8240 bytes de memoria flash e 800 bytes de EEPROM, não tivemos muito preocupados em termos de otimização de memoria, pois tínhamos bastante margem, tivemos mais em consideração a parte do desempenho.

Para atingirmos velocidades maiores então temos de aumentar a nossa tensão, foi substituído o suporte de 3 pilhas AA por um suporte de 4 pilhas AA, colocando uma tensão de mais ou menos 7V, permitindo assim velocidades muito mais elevadas, como é demonstrado no modo speed do link em anexo [1] . Nesse anexo está compilado diversas funções do robô, as mesmas que foram apresentadas ao Professor. [VIDEO]

Infelizmente o tempo era curto para as ideias que gostávamos de ter realizado, mas demos o nosso melhor para realizar um projeto que fosse diferente do habitual.

Concluído o nosso projeto conseguimos verificar que foi um projeto realizado com êxito, bastante completo e com cuidado com todos os detalhes, ficamos bastante orgulhos com o projeto final.

Anexos

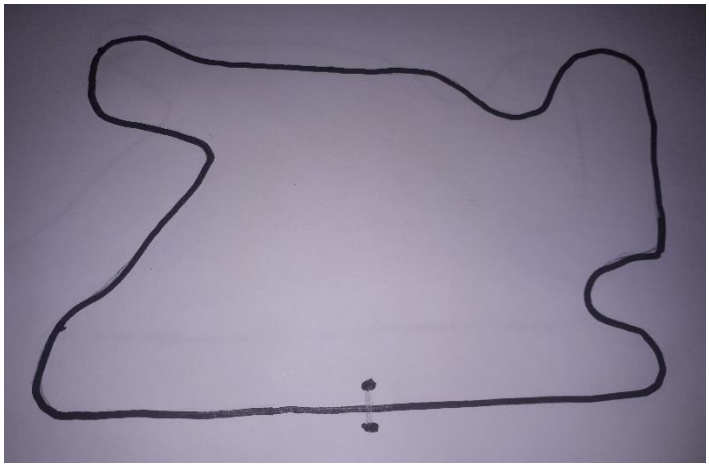

Fig 29. Protótipo da pista numa folha A4

Fig 30. LCD 20x4

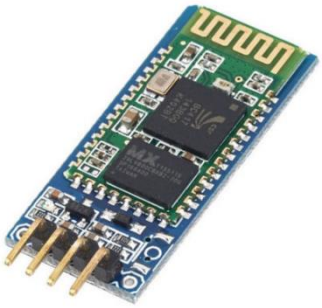

Fig 31. HC-06

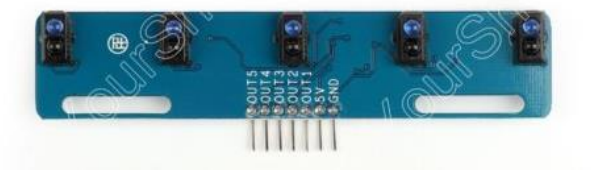

Fig 32. 5-Way Tracking Sensor Trace Sensor

Fig 33. DC Motor

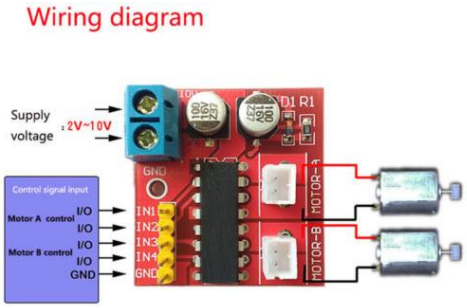

Fig 34. Dual Channel DC Motor Driver

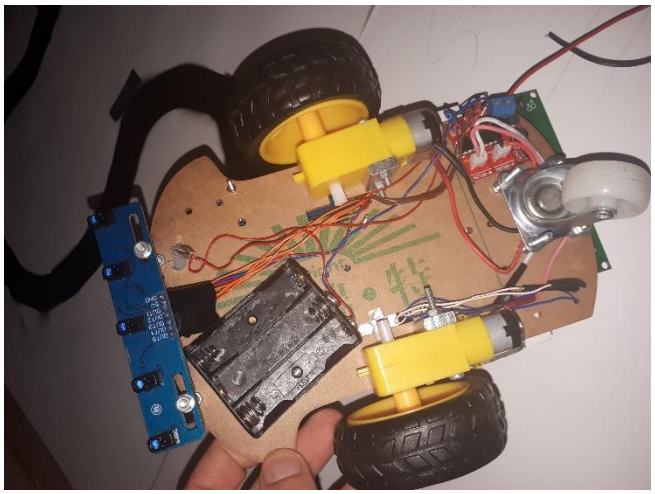

Fig 35. Parte traseira do robô

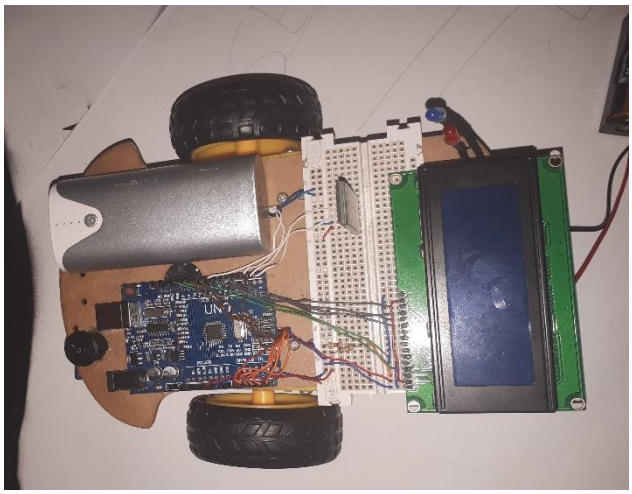

Fig 36. Parte frontal do robô

[1] Video : https://www.youtube.com/watch?v=CKVy9AVmKbY

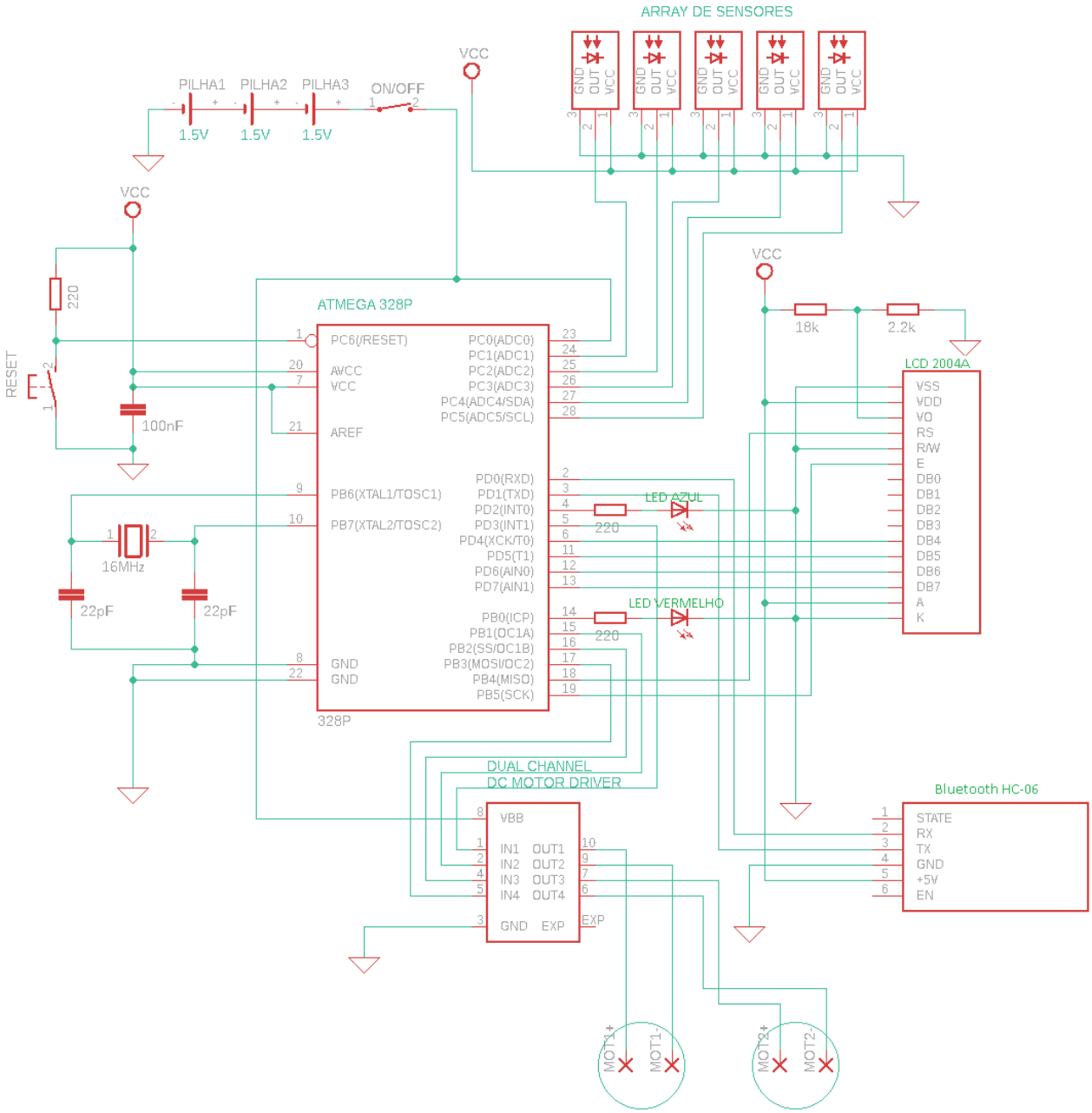

Fig 37. Esquema elétrico

Bytes Recebidos

61-65 -> Velocidade
150 -> Start
151-> Stop
41 -> A cada 1s para confirmar que está no modo automatico

Fig 38. Bytes recebidos do Modo Automático

1-> Carrega botão direito	11-> larga botao direito
2-> Carrega Esquerda	12->Larga Esquerda
3->Carrega Cima	13->Larga Cima
4->Carrega Baixo	14->Larga Baixo
5-> Sem carregar nada (PARADO)	40-> Cada 3s para confirmar que está no modo manual
41-> Modo competiçao	51-55-> Valor da velocidade

Fig 39. Bytes recebidos do Modo Manual

Bytes Enviados

1-> Deteta preto OUT1 (Sensor[0])	11-> Deteta branco OUT1 (Sensor[0])
2-> Deteta preto OUT2 (Sensor[1])	12-> Deteta branco OUT2 (Sensor[1])
3-> Deteta preto OUT3 (Sensor[2])	13-> Deteta branco OUT3 (Sensor[2])
4-> Deteta preto OUT4 (Sensor[3])	14-> Deteta branco OUT4 (Sensor[3])
5-> Deteta preto OUT5 (Sensor[4])	15-> Deteta branco OUT5 (Sensor[4])
122-> Robo perdido	123-> Robo Normal
125-> RUN	126->STOP
20-30 -> Bateria	70-> Aumenta 1 volta
71-> Dá reset ao numero de voltas	

Fig 40. Bytes Enviados do Modo Automático

20-30 -> Nivel de bateria

Fig 41. Bytes Enviados do Modo Manual