Marcelo Martins
20095236

Mapping & Path planning

Orientador: Professor Doutor Nuno Pombo

Universidade Lusófona de Humanidades e Tecnologias Escola de Comunicação, Arquitectura, Artes e Tecnologias da Informação

Lisboa, 21 de Março de 2017

Indice

Glossário de Imagens

Resumo

Este trabalho teve como objectivo a pesquisa e implementação de um sistema autónomo de navegação.
Alguns dos pontos propostos para este sistema seriam:

- Capacidade de evitar colisões;
- Capacidade de aprender, identificar e lembrar de obstáculos que fosse encontrado;
- Capacidade de ao dar uma localização, conseguir com os dados já obtidos no ponto anterior, criar o melhor, mais rápido caminho até ao ponto de destino.

Abstract

This project had as objective, the research and implementation of an autonomous navigation system.
Some of the objectives for this system were:

- Ability to avoid collisions;
- Ability to learn and identify the obstacles that were found through the time
- Ability to use what was learned in the previous point to create the most optimized route to the destination.

Introdução

Objectivos

O objectivo deste trabalho seria a pesquisa, construção e implementação de um sistema autónomo de navegação que tivesse a capacidade de identificar e desviar de obstáculos dinamicamente, um carro robotizado que pudesse ter a capacidade de navegar em áreas desconhecidas para o mesmo, realizando uma “aprendizagem” pela área navegada para que pudesse usar esse conhecimento caso fosse necessário voltar a passar por caminhos que já tivessem sido previamente mapeados.

Este tema surgiu pela curiosidade e pelo gosto do conceito de inteligência artificial, bem como a aplicação que está a ter já nos dias de hoje na evolução da tecnologia. Ao dia de hoje, já existem carros criados pelas grandes empresas como a Google ou a Tesla Motors que começam a ser autónomos, existindo bastantes vídeos com demonstrações desta tecnologia. Este tipo de tecnologia poderá ser muito importante na perspectiva de eliminar por completo o fator número um em acidentes e vítimas rodoviárias, o erro humano.

Estado da Arte

Path Planning

Path Planning é um dos aspectos mais importantes para um sistema autónomo de navegação.(1)
Para um sistema autónomo de navegação, um dos algoritmos mais importantes a implementar, tem de ter como objectivo principal encontrar o caminho mais curto do ponto de partida **A**, até ao ponto de chegada **B**, ou seja, o algoritmo têm de encontrar o caminho mais optimizado de navegação.(1) Por caminho otimizado é possível ter várias variáveis a serem avaliadas, as mais comuns são a distância e o tempo, um exemplo mais comum e que uma grande maioria de pessoas poderam já ter usado, seria a pesquisa de um trajeto pelo Google Maps, em muitos casos a aplicação oferece mais do que um caminho diferente até ao destino desejado, sendo por norma um destes destinos o mais curto a nível de Km., e outro o mais rápido a nível de tempo. Existe outra métrica associada a estas duas já referidas que é a eficiência(2), sendo que de uma maneira geral, numa escala macro e dentro do exemplo dado a eficiência está diretamente relacionada com as métricas anteriores, numa escala mais pequena como num robot de pequenas dimensões, existem mais alguns factores que são importantes ter em consideração, como por exemplo, se sempre que o robot mudar de direcção, ou travar, existir um pequeno deslize resultante do momentum acumulado, o caminho mais otimizado pode ser além do trajecto mais curto ou mais rápido, o trajecto que envolve menos curvas ou travagens.(2)(3)
Dependendo do caso a aplicar este tipo de algoritmo, a importância de cada uma das métricas referidas pode variar, podendo existir casos em que o tempo seja mais importante que distância ou eficiência, irá depender de cada implementação os factores mais relevantes.

Mapa

Ao aplicar algoritmos de path planning a um robot, que têm como objectivo, navegar do ponto de partida **A** ao ponto de chegada **B**, é necessário que exista uma representação, um mapa que contenha um conjunto de informações necessárias para essa navegação, bem como a capacidade do robot se localizar nesse mesmo mapa.(1)(2)(3)(4)
A representação de um mapa é fundamental para a aplicação dos algoritmos.
No exemplo de um robot que precise navegar do ponto **A** ao ponto **B** é necessário o robot conseguir ao longo do movimento saber determinar na representação que têm da área:(1)(2)

- O ponto de partida - **A**;
- O ponto de destino - **B**;
- Obstáculos presentes no mapa;
- Localização do robot no mapa.

Só após estes fatores serem considerados é possível aplicar um algoritmo de path planning e encontrar o percurso mais curto, mais rápido e mais eficiente.

Algoritmos

Algoritmos que encontrem o caminho mais curto são importantes em várias áreas de IT para além da robótica, como em network routing ou jogos de vídeo.[1]
Existem duas maneiras complementares de abordar a representação de um mapa num sistema informático.[1],[2]

- Representação métrica;
- Representação topologica;

Numa representação métrica, o mapa é construído por um conjunto de blocos de tamanho semelhante, em que cada um desses blocos contém uma grelha que poderá por sua vez voltar a ser dividida se surgir necessidade de aumentar o detalhe necessário.
Dependendo de como se optar por representar o mapa, é possível ter blocos de diferentes tamanhos, por exemplo, se for determinado que uma área de grande dimensão contém um obstáculo, computacionalmente faz sentido agregar essa informação numa única célula do mapa para que esta tenha apenas ter de ser processada uma única vez, caso contrário seria necessário processar várias células de tamanho mais pequeno que iriam conter a mesma informação, e por estarem em maior quantidade, iria consumir mais recursos e levar mais tempo ao cpu para processar essa informação.
Numa representação topológica, o mapa contém os pontos de interesse sendo as várias ligações entre esses pontos, os caminhos por onde o robot poderá navegar. [1][2][4]

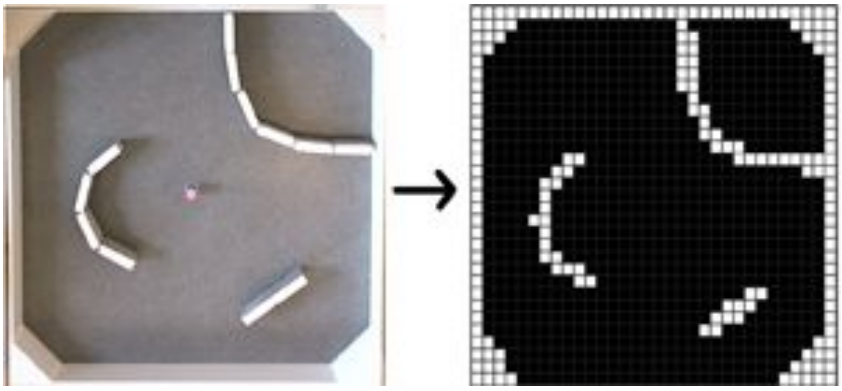


Figura 01 - Exemplo de um mapa métrico

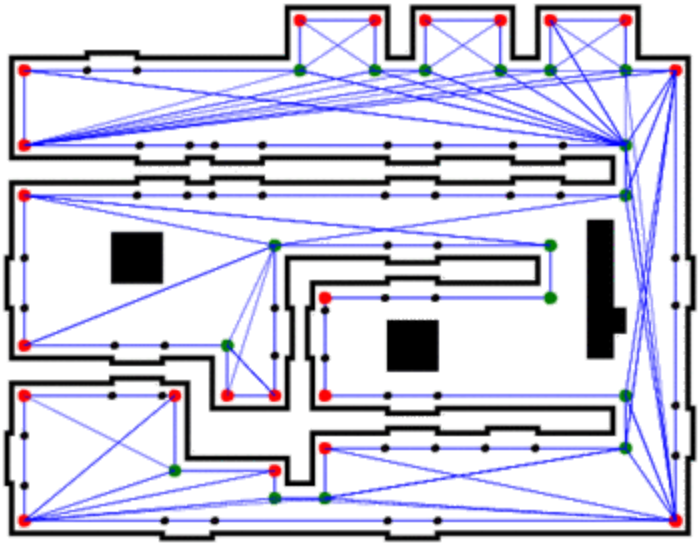


Figura 02 - Exemplo de um mapa Topológico

Atualmente o tipo de mapa mais utilizado para este tipo de robots é o *occupancy grid map*.
Um Occupancy grid map é um tipo de mapa métrico, em que existe uma grelha que representa o mapa, e nessa grelha, em cada célula é colocado todos os obstáculos conhecidos, bem como a probabilidade de alguma célula em específico conter um obstáculo. Este tipo de mapa usa a teoria estatística de Bayesian.[1]

Alguns exemplos de algoritmos de Path Planning são:[1]

- Dijkstra
- A*
- D*
- RRT

Dijkstra

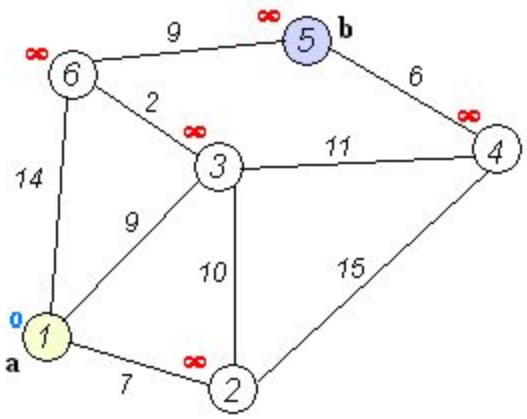
O algoritmo de Dijkstra foi desenvolvido em 1956 por Edsger W. Dijkstra e tem como objetivo encontra o caminho mais curto para um determinado ponto.
Este algoritmo tem uma representação topológica por nodes e para poder ser aplicado é necessário saber:

- Node de partida
- Node de chegada
- Custo de viagem entre nodes

Apenas após ter estes três dados, é possível aplicar este algoritmo.

Como funciona:

O algoritmo vai analisar qual o custo de viagem entre todos os nodes que estão ligados ao node de partida (1) e vai escolher viajar para o node que tem o custo mais baixo (2).



A* (A - Star)

É um algoritmo desenvolvido por Nils Nilsson para aplicar no robot que estava a ser desenvolvido “Shakey the Robot”, sendo visto como um algoritmo mais eficiente do que o algoritmo usado na altura (Dijkstra). O algoritmo A* é um algoritmo de procura informada, sendo que procura em todos os caminhos possíveis pela melhor solução dependendo do objectivo seja ele o menor tempo, custo ou distância.

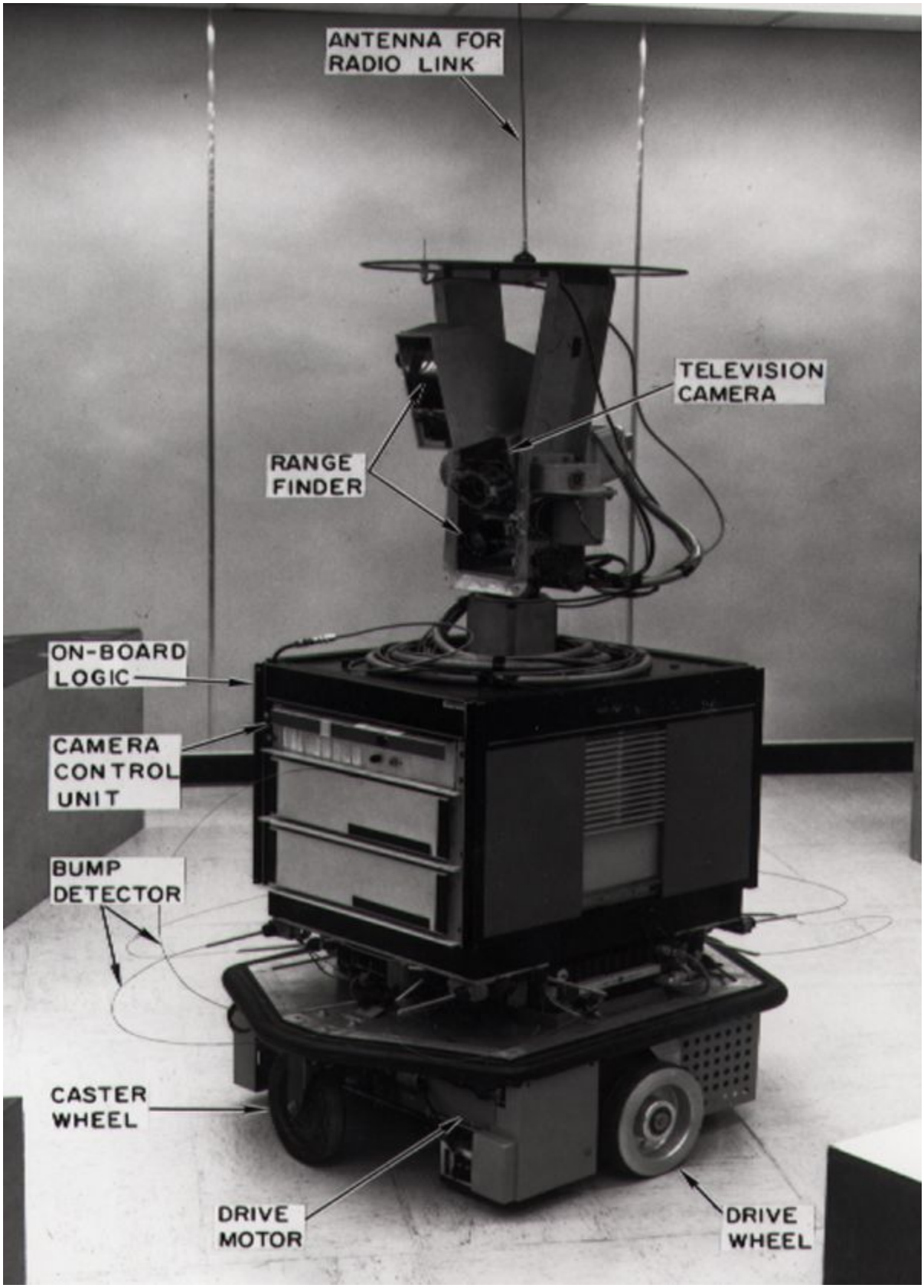


Figura 03 - Shakey the Robot

Projecto desenvolvido

Grande parte do esforço envolvido neste projecto foi a pesquisa dos equipamentos e respectivas compatibilidades que melhor poderiam realizar o objectivo.

Lista de material adquirido

Para este tipo de projecto, e como nunca tinha realizado nada do género, foi necessário adquirir um conjunto de material que uma pessoa que já tenha trabalhado nesta área não irá precisar, pois grande parte do material poderá ser usado para outras tarefas no futuro.

Lista de Material adquirido:

Material comprado	**Preço em €**
The Arduino Starter Kit	€72.10
2WD Arduino compatible Mobile platform	€36.00
MiniQ 2WD Plus	€27.40
Romeo BLE - Arduino Robot Control Board with Bluetooh 4.0	€41.25
Kit 50x - Cabos de ligação macho - fêmea	€9.90
Kit 40x - Cabos de ligação macho - macho	€6.90
Estação de soldar Profissional 45W	€27.90
Bomba dessoldadora	€3.50
Cabo unifilar preto 0.2MM	€0.20
Solda 60/40 17G	€2.00
Alicate de descarnar de ajuste 2-6MM	€8.90
Sensor sónico SRF05	€17.90
Bussula LSM303 - SEN079	€14.42
Bluno Link	€9.19
Total	**€277.56**

A longa lista de material que foi necessária foi uma das grandes dificuldades da implementação deste projecto, o tempo entre pesquisas de compatibilidades e encomendas foi um dos maiores consumidores de tempo deste projeto, sendo que para algumas peças foi preciso aguardar mais de um mês para que após realizar a encomenda, a mesma chegasse a Portugal.

2WD Arduino compatible Mobile platform[5]

Figura 04 - Peças do 2WD Arduino compatible Mobile platform

Specification[5]

- 2WD Arduino mobile robot development platform
- Low-cost Arduino microcontroller mobile platform
- Two differential drive
- Caster ball included
- Complete chassis with mounting hardware
- Dimensions: 170mm diameter base
- Weight: 400g

Motor Specification[5]

- Gear Ratio 1:120
- No-load speed(3V):100RPM
- No-load speed(6V):200RPM
- No-load current(3V):60mA
- No-load current(6V):71mA
- Stall current(3V):260mA
- Stall current(6V):470mA
- Torgue (3V): 1.2Kgcm
- Torque (6V): 1.92Kgcm
- Size: 55mm x 48.3mm x 23mm
- Weight:45g

RoMeo BLE (SKU:DFR0305)[6]

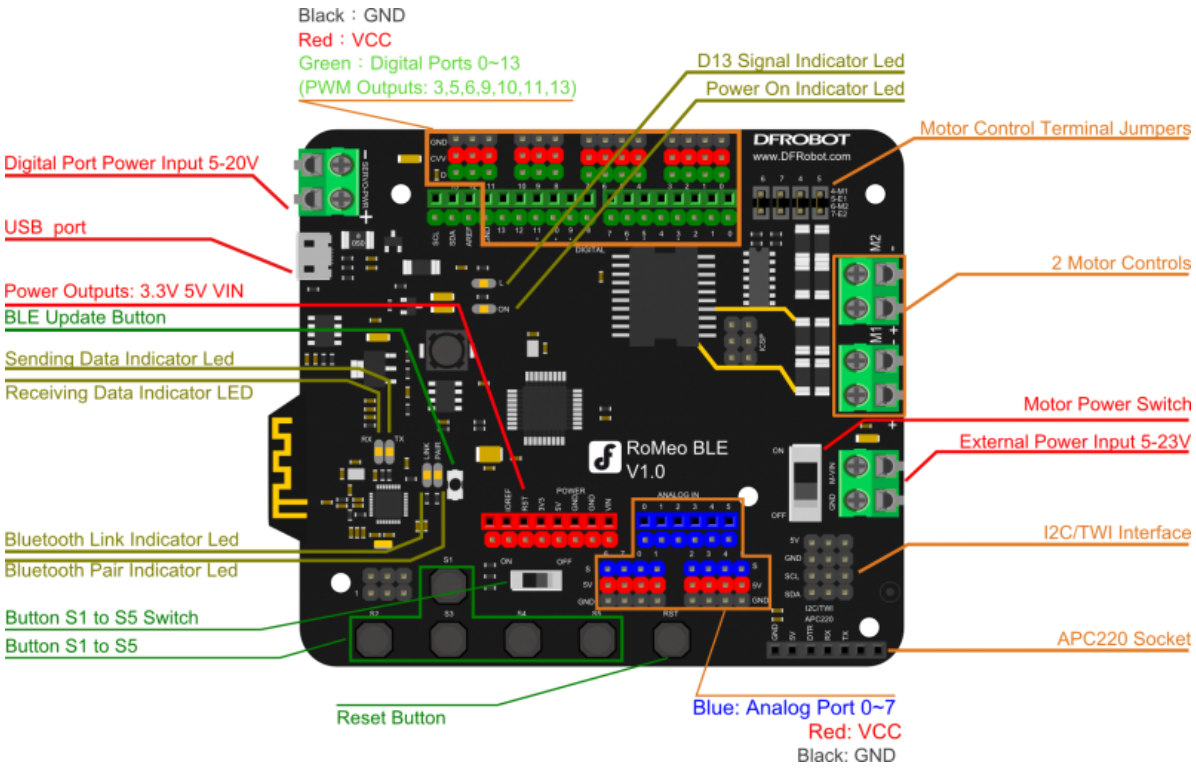


Figura 05 - Romeo BLE pinout info

Specifications[6]

Basic	Features
• Microcontroller: ATmega328P • Bootloader: Arduino UNO • On-board BLE chip: TI CC2540 • 14 Digital I/O ports • 6 PWM Outputs (Pin11, Pin10, Pin9, Pin6, Pin5, Pin3) • 8 10-bit analog input ports • 3 I2Cs • 5 Buttons • Power Supply Port: USB or DC2.1 • External Power Supply Range: 5-23V • DC output: 5V/3.3V • Size: 94mm x 80mm	• Auto sensing/switching external power input • Transmission range: 70m in free space • Support bluetooth remote update the Arduino program • Support Bluetooth HID • Support iBeacons • Support AT command to config the BLE • Support Transparent communication through Serial • Support the master-salve machine switch • Support usb update BLE chip program • Support Male and Female Pin Header • Two way H-bridgedMotor Driver with 2A maximum current

- Integrated sockets for APC220 RF Module

Sensor SRF05[7]

Figura 06 - Sensor SRF05

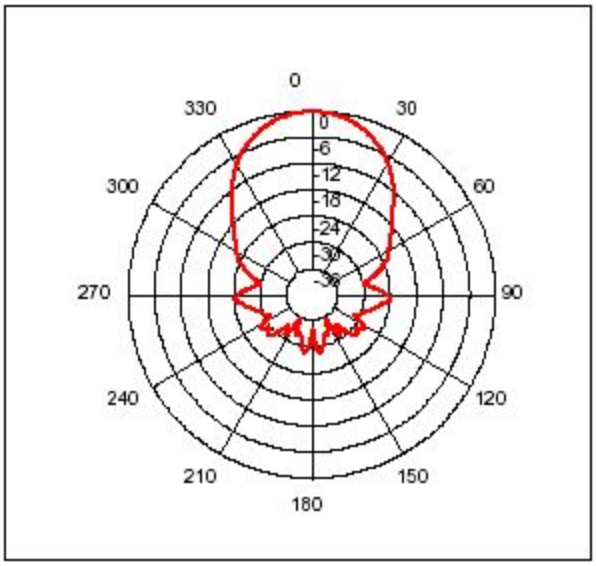


Figura 07 - SRF05 beam pattern and beam width

Specifications[8]

- Voltage - 5V
- Current - 30mA Typ. 50mA Max.
- Frequency - 40KHz
- Max Range - 3 m
- Min Range - 3 cm

- Sensitivity - Detect 3cm diameter broom handle at > 3 m
- Input Trigger - 10uS Min. TTL level pulse
- Echo Pulse - Positive TTL level signal, width proportional to range.
- Small Size - 43mm x 20mm x 17mm height

Estes três componentes:

- 2DE Arduino Mobile Platform;
- Placa RoMeo BLE;
- Sensor SRF05

Compõem os principais componentes deste projecto, sendo o microcontrolador RoMeo BLE o componente central de ligação.

Robot

Após a construção da plataforma mobile 2WD, o primeiro passo foi fazer a ligação dos motores ao Romeo BLE.
Tal como mostra a figura 4 existem duas entradas na placa Romeo BLE para fazer a ligação dos motores, M1 e M2, ambos com um polo positivo e outro negativo.
Na figura 7 é mostrado em maior detalhe essas entradas. Foi necessário identificar os polos positivos e negativos dos motores, e proceder à ligação correcta.

Alimentação e Motores[5][6]

A alimentação do robot podia ser feita unicamente com uma ligação por cabo USB a um computador, no entanto e como o objectivo era dar a maior autonomia possível ao equipamento, foi optado por fazer a alimentação do mesmo com um conjunto de cinco pilhas.
A alimentação do robot é realizada por um conjunto de cinco pilhas.
Esta ligação foi crítica pois o manual, apesar de não dizer como proceder à ligação, dizia claramente que se as polaridades tanto dos motores, como da alimentação externa como é o caso das pilhas, for mal executada, que tanto a placa como os motores iriam ser danificados.
Após alguma pesquisa foi nas normas da electricidade que as ligações foram baseadas.
Tanto os motores como as pilhas tinham dois cabos de ligação, um preto e outro vermelho.
Na placa Romeo BLE, no local de ligação do external power supply existe a designação GND que corresponde ao ground e a designação M-VIN que corresponde ao polo de corrente.
Resumindo, o cabo preto no caso das pilhas foi ligado ao GND, no caso dos motores, tanto para M1 Como para M2 foi ligado ao polo negativo (-). O cabo vermelho no caso das pilhas foi ligado ao polo M-VIN e no caso dos motores, tanto para M1 Como para M2 foi ligado ao polo positivo (+).

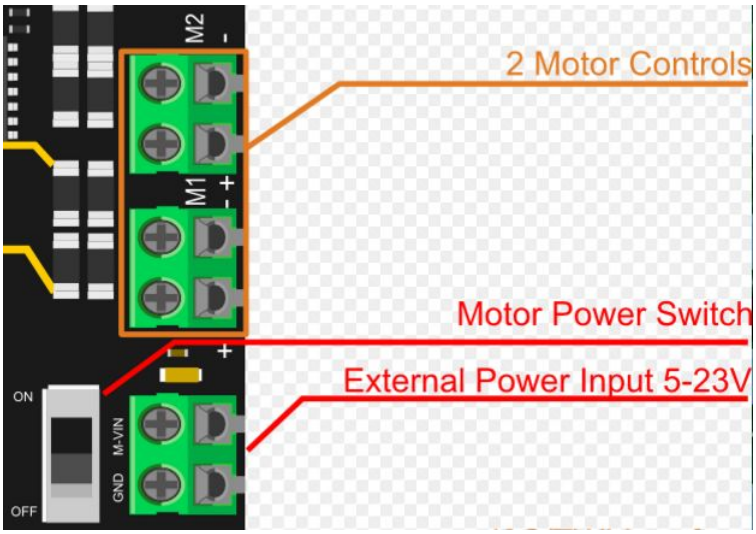


Figura 08 - Ligações de alimentação e motores

SRF05[7]

Este sensor ultra sónico é o componente que permite o robot detectar obstáculos.
A figura 6 mostra a onda sónica tipo que o sensor irá criar, e por consequência ter capacidade de detectar objectos dentro do padrão cónico que o sensor cria.
Existem dois modos distintos de configurar este sensor, ambos os modos com as suas vantagens e desvantagens.

- Modo 1- Separated Trigger and Echo

Neste modo é usado dois pins distintos para emitir e para receber o impulso do sensor.

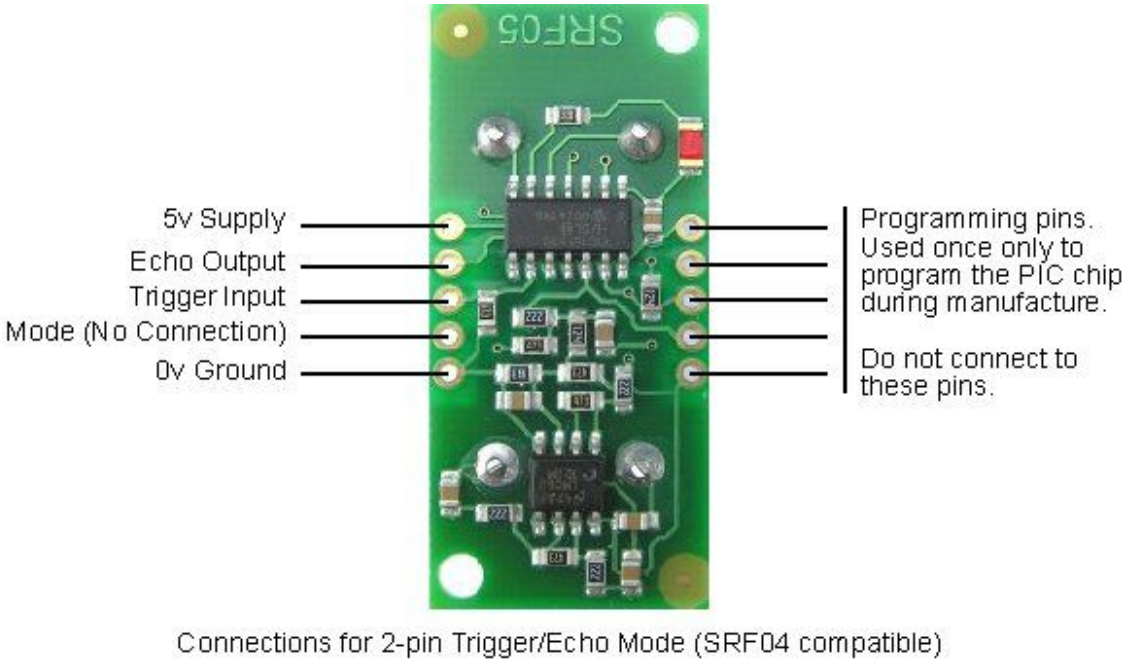


Figura 09 - Ligações do sensor SRF05 para o Modo 1

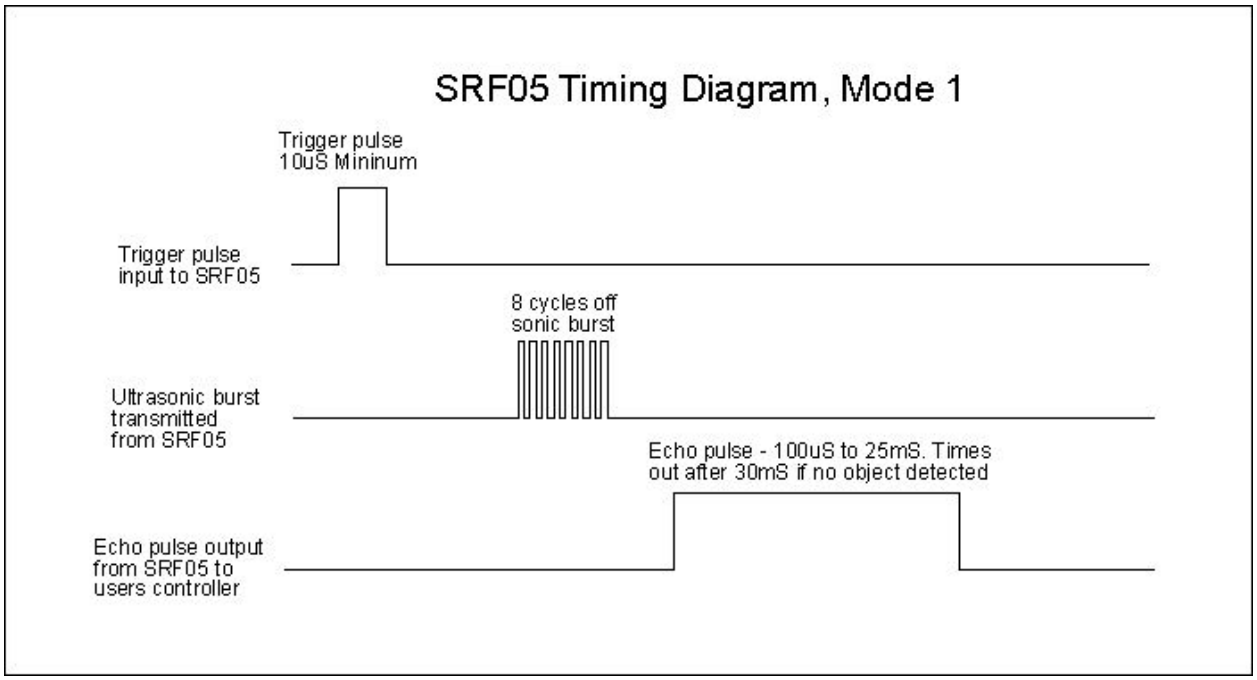


Figura 10 - Diagrama de tempo do sensor SRF05 para o Modo 1

- Modo 2 - Single Pin for both Trigger and Echo

Neste modo, ao contrário do modo 1, é usado um único pin para emitir e depois para receber o sinal transmitido.

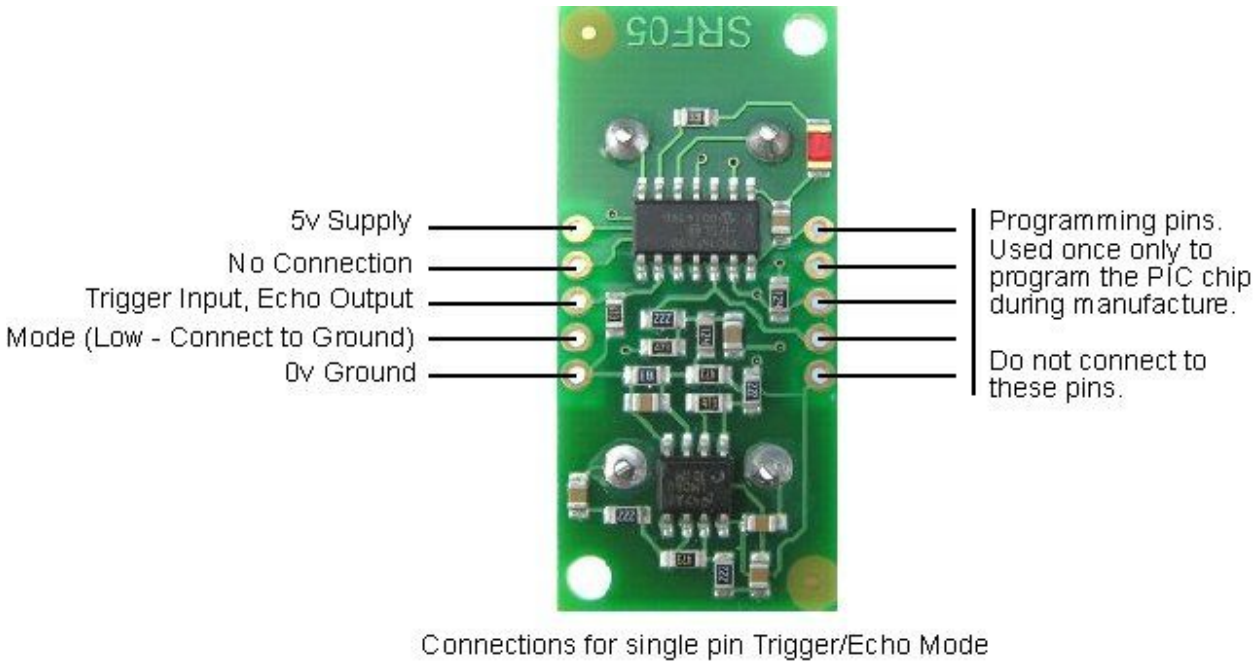


Figura 11 - Ligações do sensor SRF05 para o Modo 2

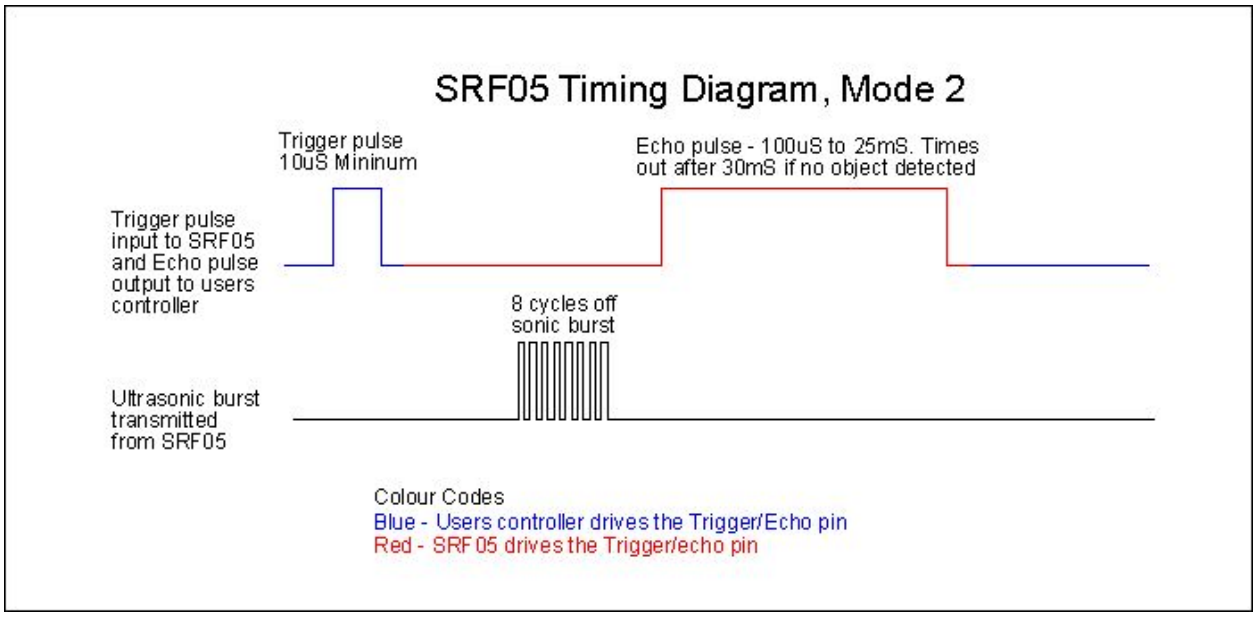


Figura 12 - Diagrama de tempo do sensor SRF05 para o Modo 2

Comparando ambos os modos, é possível verificar que o modo 1, além de ser mais simples a nível de implementação, é o mais fiável, pois consegue estar de uma forma contínua e em simultâneo a emitir e a receber o sinal, sem que uma ação vá interromper a outra. No caso do modo 2, isto não acontece, pois como o sinal tanto é emitido como recebido pelo mesmo pin, significa que as ações vão ser mutuamente exclusivas, não seria possível estar a emitir e a receber o sinal ao mesmo tempo. Neste último caso, teria de ser através do software a mudança de configuração do pin para Input ou Output, dependendo da ação a realizar. Logicamente e de uma maneira bastante general, as ações que seriam necessárias para configurar o sensor em modo 2, eram:

1. Configurar pin para output;
2. Emitir sinal;
3. Configurar pin para input;
4. Receber sinal;

Este conjunto de ações estaria continuamente a ser repetido nesta ordem para que o sensor pudesse estar a funcionar.

Devido ao modo 2 ter algumas restrições, e aumentar a necessidade da capacidade de processamento, o mesmo só seria aconselhado na eventualidade de não existir na placa Arduino a ser usada, pins livres suficientes, pois a única vantagem em relação ao modo 1 é apenas ser necessário a ligação de 3 pins, comparando com os 4 do modo 1.

Por todos os motivos apresentados, foi escolhido o modo 1 de funcionamento deste sensor.

Primeiros testes, observações, dificuldades e soluções

Numa fase inicial de testes, rápidamente foi possível realizar algumas observações sobre o comportamento do robot.
Observações:

1. O robot anda demasiado rápido para o sensor ter tempo de reagir e parar antes de chocar contra um obstáculo.

Esta foi a primeira observação, a mais óbvia.
Tinha configurado os motores do robot para funcionar com a potência máxima, e a velocidade instantânea que atingia era demasiada elevada para dar tempo ao sensor de reagir e de parar o robot antes de colidir com obstáculos.
Este problema foi resolvido numa fase inicial, baixando a potência que os motores estavam configurados para usar. Mais tarde no decorrer do projecto e igualmente numa tentativa de melhorar os dados que o robot estava a conseguir receber do sensor, foi aumentado a frequência com que o sensor estava a enviar o sinal de impulso, o que em conjunto com a baixa de velocidade dos motores, permitiu ao robot um maior controlo nos seus movimentos. Por um lado, o robot estava a andar mais lentamente, e por outro estava a obter informação útil dos obstáculos a um ritmo muito superior.

2. O robot não anda em linha recta.

Após o robot começar a movimentar-se mais lentamente, ficou claro outro problema grave que tinha. Não conseguia andar em linha recta. Ao analisar o robot fisicamente cheguei á conclusão que este problema poderia estar a acontecer por:

- Robot tinha sido mal montado, ou não tinha ficado simétrico;
- Robot tinha um defeito de construção numa das rodas, o que poderia justificar o desvio;
- O peso do hardware que está em cima do robot, não estando distribuído de forma igual, poderia estar a provocar esse desvio.

Desmontei o robot e voltei a montar o mesmo, para tentar que alguma falha no processo de montagem pudesse estar a provocar este desvio. Infelizmente o problema não ficou solucionado com uma segunda e mais cuidada montagem.
O defeito de construção presente na roda esquerda não é possível solucionar, sem encomendar uma nova plataforma, ou trocar a roda que tem o defeito, no entanto nada me iria garantir que as novas rodas estivessem 100% iguais.
O peso sobre a plataforma não podia ser controlado pois existem sítios específicos onde os componentes devem encaixar para que ficassem corretamente montados.
Neste momento foi encomendado o sensor LSM303 - SEN079[(9)] que em teoria iria funcionar como bússola e tentaria corrigir os desvios do robot através do software.
A encomenda desta bússula, ao dia em que este relatório está a ser escrito, ainda não chegou, pelo que a sua implementação não está concluída.

3. Com a deslocação do carro, existem pequenos desvios / deslocações resultantes da acomulação de momentum do carro.

O carro ao se deslocar, vai acomulando velocidade. Essa velocidade faz com que entre comandos, o carro realize uma deslocação mais prolongada do que o desejado.

Um exemplo seria a travagem do carro. Se o carro estiver em movimento e detectar um obstáculo, assumindo que neste caso ele está programado para parar, entre o obstáculo ser detectado, e o carro conseguir reverter a aceleração do mesmo, vai ocorrer um “derrape” do mesmo, resultando isso numa pequena movimentação (que é uma continuação da movimentação que o carro estava a ter até então) que não está prevista.
Resumindo, qualquer mudança de direcção vai aumentar a margem de erro que resulta do movimento.

4. Escolha de direção do desvio.

Um dos objectivos mais importantes é conseguir com que o robot seja capaz de, ao detectar um obstáculo no seu caminho, consiga fazer um desvio.
Para evitar definir o comportamento de desvio sempre da mesma forma, em vez de se escolher uma direcção específica, seja esquerda ou direita, escolhi implementar alguma aleatoriedade á essa escolha. Acontece que o código inicial, resultava num comportamento errante, o que acontecia é que o robot enquanto detectava um obstáculo, fazia uma escolha entre esquerda e direita, mas em obstáculos de grandes dimensões o robot começava a virar numa dessas direções, mas a meio, podia, ou não (como era aleatório) mudar de direção. O resultado era um robot que ao encontrar um obstáculo mudava de direcção várias vezes, até ter a “sorte” de escolher a mesma direção, algumas vezes seguidas para poder continuar a marcha.
A solução para esta questão foi a criação de uma função à parte em que, a escolha entre esquerda e direita continua a ser feita de forma aleatória, mas assim que é tomado uma decisão de direção, o robot vai continuar a virar nesse sentido até não existir nenhum obstáculo e seja possível continuar em frente.

Wireless

Com objetivo de aumentar a mobilidade do robot e de poder controlar o mesmo remotamente, sem ser preciso estar fisicamente ligado ao computador, bem como obter informação sobre os dados que estavam a ser recolhidos em real-time foi adquirido o componente Bluno Link.[(10)]

Figura 13 - Bluno Link

Specifications[11]

- bluetooth chip:TI CC2540
- Frequency: 2.4GHz
- Transfer rate: ≤1Mbps
- Modulation: GFSK, bluetooth low power, V4.0
- Power consumption: working:10.6mA average ,ready mode:8.7mA
- sensitivity: -93dB
- Input Voltage: +3.3 DC
- Operating temperature： -10 ℃ ~ +85 ℃
- Transmission distance: 15~20m indoor (30m in free space)
- size:32mm * 16mm
- support by the AT command to debug the BLE
- support the master-slave machine switch
- support transparent transmission serial port （only support UNO and Arduino Mega）
- support bluetooth remote update the Arduino program

A configuração do componente foi bastante complexa devido à falta de documentação sobre o assunto.
É necessário criar uma relação master-slave em que um dos componentes vai ter o comando sobre o outro.
Após algumas tentativas e erros, passou as er possível controlar o robot remotamente, usando para isso um pedaço de código em que os comandos inseridos pelo PC seriam obedecidas independente do fluxo de execução em que o robot se encontraria. Foi no entanto criado uma excepção para que mesmo ordenando o robot para colidir com um obstáculo, ele iria parar e desviar do mesmo.
As teclas usadas foram:
W - Andar em frente
A - Virar para a esquerda
S - Andar para trás
D - Virar para a direita
X - Parar.
Estes comandos teriam de ser inseridos através da consola de comandos.

Próximos passos e Dificuldades previstas

Apesar de já ter um robot funcional, ele ainda não consegue efectivamente “aprender” nada, mas como verificado na componente de investigação deste trabalho, isso vai ser impossível de implementar de forma tradicional com o tempo disponível para a conclusão deste trabalho.
Os próximos objetivos para implementação deste projecto seriam:

1. Implementar o componente que funciona como uma bussola.
    a. Tentar mitigar o desvio que o robot tem, provocado pelo defeito de fabrico da roda.
    b. Implementar mais ou menos força nos motores, dependendo da inclinação do plano. Neste momento a potência usada nos motores está pensada apenas para um plano liso e horizontal, sem inclinação nenhuma.
2. Implementação de um mapa em matriz, com granularidade variada de forma a tanto ter a capacidade de detalhe, numa granularidade mais fina, como de agrupar grandes conjuntos de informação semelhante de forma a poupar a capacidade de processamento.
    a. Possibilidade de criar o mapa dinamicamente. Para isso seria necessário adquirir um módulo de expansão de memória, pois a memória atual do robot é limitada.
3. Existe sempre a questão da localização do robot. Considerando um mapa não conhecido, de dimensões variáveis, sem usar mais sensores, vai ser necessário assumir sempre alguma margem de erro, e relacionar o mapa já “aprendido” pelo robot e os obstáculos que o robot está a localizar á sua volta, de forma a se poder localizar.
    a. Ao inserir um ponto de destino, será sempre em relação ao robot, no entanto, se esse ponto não for possível aceder, por exemplo, selecionando um ponto que estaria por detrás de uma parede, o robot tem de ter a capacidade de chegar o mais perto possível, sem nunca colidir com a parede.
4. Irá sempre existir alguma margem de erro. A implementação de mais e de melhores sensores, iriam mitigar muito a margem de erro.

Código implementado e testado

```
//SFR05
// variables to take x number of readings and then average them
// to remove the jitter/noise from the SRF05 sonar readings

const int numOfReadings = 10;                 // number of readings to take/ items in the array
int readings[numOfReadings];                  // stores the distance readings in an array
int arrayIndex = 0;                           // arrayIndex of the current item in the array
int total = 0;                                // stores the cumlative total
int averageDistance = 0;                      // stores the average value

// setup pins and variables for SRF05 sonar device

int echoPin = 2;                                        // SRF05 echo pin (digital 2)
```


```
int initPin = 3;                                    // SRF05 trigger pin (digital 3)
unsigned long pulseTime = 0;                        // stores the pulse in Micro Seconds
unsigned long distance = 0;                         // variable for storing the distance (cm)

//Motor
//Standard PWM DC control
int E1 = 5;                         //M1 Speed Control
int E2 = 6;                         //M2 Speed Control
int M1 = 4;                         //M1 Direction Control
int M2 = 7;                         //M1 Direction Control


void stop(void)                     //Stop
{
  digitalWrite(E1, LOW);
  digitalWrite(E2, LOW);
}
void advance(char a, char b)            //Move forward
{
  analogWrite (E1, a);              //PWM Speed Control
  digitalWrite(M1, HIGH);
  analogWrite (E2, b);
  digitalWrite(M2, HIGH);
}
void back_off (char a, char b)          //Move backward
{
  analogWrite (E1, a);
  digitalWrite(M1, LOW);
  analogWrite (E2, b);
  digitalWrite(M2, LOW);
}
void turn_L (char a, char b)            //Turn Left
{
  analogWrite (E1, a);
  digitalWrite(M1, LOW);
  analogWrite (E2, b);
  digitalWrite(M2, HIGH);
}
void turn_R (char a, char b)            //Turn Right
{
  analogWrite (E1, a);
  digitalWrite(M1, HIGH);
  analogWrite (E2, b);
```

```
  digitalWrite(M2, LOW);
}                                       //Setup

void sensor() {
  digitalWrite(initPin, HIGH);                // send 10 microsecond pulse
  delayMicroseconds(10);                      // wait 10 microseconds before turning off
  digitalWrite(initPin, LOW);                 // stop sending the pulse
  pulseTime = pulseIn(echoPin, HIGH);           // Look for a return pulse, it should be high as the
pulse goes low-high-low
  distance = pulseTime / 58;                  // Distance = pulse time / 58 to convert to cm.
  total = total - readings[arrayIndex];       // subtract the last distance
  readings[arrayIndex] = distance;            // add distance reading to array
  total = total + readings[arrayIndex];       // add the reading to the total
  arrayIndex = arrayIndex + 1;                // go to the next item in the array
  // At the end of the array (10 items) then start again
  if (arrayIndex >= numOfReadings)  {
    arrayIndex = 0;
  }
  averageDistance = total / numOfReadings;      // calculate the average distance

  Serial.println(averageDistance, DEC);         // print out the average distance to the debugger
  delay(25);
}

void setup() {
  // put your setup code here, to run once:
  pinMode(initPin, OUTPUT);                                   // set init pin 3 as output
  pinMode(echoPin, INPUT);                                    // set echo pin 2 as input

  // create array loop to iterate over every item in the array
  for (int thisReading = 0; thisReading < numOfReadings; thisReading++) {
    readings[thisReading] = 0;
  }
  // initialize the serial port, lets you view the
  // distances being pinged if connected to computer
  //Serial.begin(9600);
  int i;
  for (i = 4; i <= 7; i++)
    pinMode(i, OUTPUT);
  Serial.begin(115200);      //Set Baud Rate
  //Serial.println("Run keyboard control");
  Serial.println("--- End of Setup ---");
}
```

```
//loop
void loop() {
  // put your main code here, to run repeatedly:
  // execute
  digitalWrite(initPin, HIGH);                  // send 10 microsecond pulse
  delayMicroseconds(10);                        // wait 10 microseconds before turning off
  digitalWrite(initPin, LOW);                   // stop sending the pulse
  pulseTime = pulseIn(echoPin, HIGH);           // Look for a return pulse, it should be high as the
pulse goes low-high-low
  distance = pulseTime / 58;                    // Distance = pulse time / 58 to convert to cm.
  total = total - readings[arrayIndex];         // subtract the last distance
  readings[arrayIndex] = distance;              // add distance reading to array
  total = total + readings[arrayIndex];         // add the reading to the total
  arrayIndex = arrayIndex + 1;                  // go to the next item in the array
  // At the end of the array (10 items) then start again
  if (arrayIndex >= numOfReadings)  {
    arrayIndex = 0;
  }
  averageDistance = total / numOfReadings;      // calculate the average distance

  Serial.println(averageDistance, DEC);         // print out the average distance to the debugger
  delay(25);

  //sensor();


//-------------------------------------------------------------------------------------------------------------------------
-----------------------------
  // Failsafe - Se carro detectar que tem objectos á frente:
  if (averageDistance < 20)
  { stop();
    if (random(0, 2) == 0)                      // 0 - Left // 1 - Right
    { Serial.println("zero");
      do {
        turn_L (85, 85);
        sensor();
      } while (averageDistance < 20);
      stop();
    } else {
      Serial.println("one");
      do {
        turn_R (85, 85);
```

```
      sensor();
    } while (averageDistance < 20);
    stop();
  }
 }


//-------------------------------------------------------------------------------------------------------------------------
-----------------------------

 if (Serial.available()) {
   char val = Serial.read();

   //Serial.println("Valore Registado" + val);
   if (val != -1)
   {
     switch (val)
     {
       case 'w'://Move Forward
         advance (85, 85);                  //move forward in max speed = 255
         break;
       case 's'://Move Backward
         back_off (85, 85);                 //move back in max speed  = 255
         break;
       case 'a'://Turn Left
         turn_L (85, 85);
         break;
       case 'd'://Turn Right
         turn_R (85, 85);
         break;
       case 'z':
         Serial.println("Hello");
         break;
       case 'x':
         stop();
         break;
       case 'q':
         break;
     }
   }

   else stop();
```

```
  }

}
```

Figura 14 - Estado atual do robot

Bibliografia

1. Correll, Nikolaus. Introduction to Autonomous Robots. Createspace Independent Pub, 2014 , ISBN: 1493773070, 9781493773077.
2. Correll, Nikolaus. Introduction to Autonomous Robots: Kinematics, Perception, Localization and Planning. Magellan Scientific, 2016 , ISBN: 0692700870, 9780692700877.
3. LaValle, Steven M.. Planning Algorithms, Cambridge University Press, 2006, ISBN: 0-521-86205-1.
4. Zhi Yan, Nicolas Jouandeau, and Arab Ali Cherif. Towards a Probabilistic Roadmap for Multi-robot Coordination. Advanced Computing Laboratory of Saint-Denis (LIASD) Paris 8 University 93526 Saint-Denis, France
5. https://www.dfrobot.com/wiki/index.php/2WD_Mobile_Platform_for_Arduino_(SKU:ROB0005)
6. https://www.dfrobot.com/wiki/index.php/RoMeo_BLE_(SKU:DFR0305)
7. https://www.robot-electronics.co.uk/htm/srf05tech.htm
8. http://www.picaxe.com/docs/srf005.pdf
9. https://www.dfrobot.com/product-640.html
10. https://www.dfrobot.com/product-1220.html
11. https://www.dfrobot.com/wiki/index.php/USBBLE-LINK_Bluno_Wireless_Programming_Adapter_SKU:_TEL0087