



**PROJETO DE GRADUAÇÃO**

**Plataforma móvel com detecção de  
obstáculos**

**Aline Barbosa Alves**

**Brasília, dezembro de 2014**

**UNIVERSIDADE DE BRASÍLIA**

FACULDADE DE TECNOLOGIA  
DEPARTAMENTO DE ENGENHARIA ELÉTRICA



UNIVERSIDADE DE BRASÍLIA  
Faculdade de Tecnologia  
ENE – Departamento de Engenharia Elétrica

## PROJETO DE GRADUAÇÃO

# Plataforma móvel com detecção de obstáculos

**Aline Barbosa Alves**

RELATÓRIO SUBMETIDO AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE  
TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO REQUISITO PARCIAL PARA A OBTENÇÃO DO  
GRAU DE ENGENHEIRO ELETRICISTA

**Aprovada por**

---

Prof. D. Sc. Ricardo Zelenovsky, PUC-RJ, UnB/ENE  
*Orientador*

---

Prof. PhD. Alexandre Romariz, UnB/ ENE  
Examinador interno

---

Prof. PhD. Eduardo Peixoto, Queen Mary – University of London, UnB/ ENE  
Examinador interno

Brasília, dezembro de 2014





## FICHA CATALOGRÁFICA

ALVES, ALINE  
Plataforma móvel com detecção de obstáculos [Distrito Federal] 2014  
72p., 210 x 297 mm (ENE/FT/UnB, Engenharia Elétrica).  
Monografia de Graduação – Universidade de Brasília, Faculdade de  
Tecnologia Departamento de Engenharia Elétrica

1. – Plataforma móvel
2. – Detecção de obstáculos

I. ENE/FT/UNB  
II. Título (Série)

## REFERÊNCIA BIBLIOGRÁFICA

ALVES, A. (2014). Plataforma móvel com detecção de obstáculos, Relatório de Graduação em Engenharia Elétrica, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 72p.

## CESSÃO DE DIREITOS

AUTORES: Aline Barbosa Alves

TÍTULO: Plataforma móvel com detecção de obstáculos

GRAU: Engenheira Eletricista

ANO: 2014

É permitida à Universidade de Brasília a reprodução desta monografia de graduação e o empréstimo ou venda de tais cópias somente para propósitos acadêmicos e científicos. O autor reserva outros direitos de publicação e nenhuma parte desta monografia pode ser reproduzida sem autorização escrita do autor.

---

Aline Barbosa Alves

UnB – Universidade de Brasília

Campus Universitário Darcy Ribeiro

FT – Faculdade de Tecnologia

ENE – Departamento de Engenharia Elétrica

Brasília – DF – 70919-970

Brasil

## **Dedicatória**

*Dedico este trabalho a todos que me apoiaram e acreditaram em mim nessa longa trajetória da graduação. Dedico especialmente a minha mãe, pois sem o seu esforço eu jamais conseguiria chegar até aqui.*

## **Agradecimentos**

*Agradeço primeiramente a Deus, sem o qual nada disso seria possível. Agradeço a minha família e amigos que sempre me apoiaram. Agradeço especialmente a minha mãe pelos longos anos dedicados a minha educação e a minha avó pela confiança e apoio incondicionais. Agradeço ao meu pai, meu irmão, minha cunhada e minha amiga, Juliana Mangi, por todo apoio, confiança, ajuda, paciência e suporte. Agradeço também ao meu colega Luiz Santana pelas longas horas no laboratório fazendo com que esse projeto acontecesse e ao meu orientador, Ricardo Zelenovsky, por toda prontidão e paciência, sem eles este trabalho jamais seria possível. Por último, agradeço a todos os professores, colegas de classe, servidores e técnicos da UnB por toda ajuda e pelos cinco cansativos, mas maravilhosos, anos que passei pela universidade.*

---

## RESUMO

Este texto apresenta um estudo sobre o desenvolvimento, programação e implementação de uma plataforma móvel capaz de localizar e se desviar de obstáculos. Após uma programação prévia, a plataforma deve trabalhar de forma autônoma. Para isso, a localização dos obstáculos é feita via sensores de ultrassom e o controle dos motores é feito via motor *driver shield*. O controle é feito pelo Arduino Mega. O objetivo do sistema é ser um módulo de fácil integração capaz de ser aplicado em diversas funções futuramente.

---

## ABSTRACT

This text briefly presents a study on the development, programming and implementation of a mobile platform that can locate and avoid obstacles. The platform can work autonomously after a preliminary programming. The location of obstacles is made by ultrasound sensors and the control of the motors is by motor driver shield. The control is done by the Arduino Mega. The purpose of the system is to be an easy integration module that can be applied in various roles in the future.

# SUMÁRIO

<b>LISTA DE FIGURAS</b> .....	i
<b>LISTA DE TABELAS</b> .....	iii
<b>1. INTRODUÇÃO</b> .....	1
1.1. Ambientação .....	1
1.2. Motivação.....	3
1.3. Objetivos.....	4
1.4. Estrutura do Trabalho.....	4
<b>2. HARDWARE</b> .....	6
2.1. Arduino .....	6
2.2. Sensor de Distância Ultrassônico HC-SR04.....	9
2.3. Motor <i>Driver Shield</i> L293D .....	11
2.4. Motor DC .....	12
<b>3. PROGRAMAÇÃO</b> .....	15
3.1. Programação básica .....	15
3.1.1. <i>Arduino</i> .....	15
3.1.2. <i>Motor Driver Shield</i> .....	17
3.2. Algoritmo para o cálculo da distância do objeto e do ângulo de fuga.....	19
3.3. Princípio do acionamento dos motores .....	21
3.4. Algoritmo teórico geral .....	24
<b>4. ENSAIOS E TESTES</b> .....	28
4.1. Escolha e montagem da plataforma .....	28
4.2. Escolha dos tipos de baterias e das tensões .....	30
4.3. Ensaios com a plataforma unificada.....	31
<b>5. RESULTADOS</b> .....	37
<b>6. CONCLUSÃO</b> .....	42
<b>REFERÊNCIAS BIBLIOGRÁFICAS</b> .....	44
<b>APÊNDICES</b> .....	45
<b>APÊNDICE A</b> .....	45
<b>APÊNDICE B</b> .....	52



# LISTA DE FIGURAS

Figura 1 - Exemplo de um braço robô .....	1
Figura 2 - Exemplo de AGV ( <i>Automatic Guided Vehicles</i> ) .....	2
Figura 3 - Sketch exemplo no espaço Arduino IDE.....	7
Figura 4 - Arduino Mega.....	8
Figura 5 - Exemplificações de lóbulos laterais (à esquerda) e reverberação (à direita) .....	9
Figura 6 - Sensor Ultrassônico HC-SR04 .....	10
Figura 7 - Motor <i>Driver Shield</i> L293D .....	11
Figura 8 - Esquemático de um Motor DC .....	12
Figura 9 - Motor DC usado no projeto .....	13
Figura 10 - Fluxogramas que exemplificam o uso das funções de programação do Arduino. Fonte: [8] .....	16
Figura 11 - Sketch exemplo de programação do Motor <i>Driver Shield</i> no ambiente Arduino IDE ...	18
Figura 12 - Esboço para melhor visualização do cálculo de $\theta_1$ .....	19
Figura 13 - Esboço para melhor visualização do cálculo de $\theta_T$ , $\alpha$ e $d_T$ .....	20
Figura 14 - Esboço para melhor localização dos ângulos.....	21
Figura 15 - Gráfico Velocidade vs Distância .....	22
Figura 16 - Esboço das localizações dos motores na plataforma e medição do ângulo de fuga .....	23
Figura 17 - Gráfico Tgiro vs Ângulo de fuga .....	24
Figura 18 - Fluxograma da interrupção .....	26
Figura 19 - Fluxograma do <i>loop</i> principal.....	27
Figura 20 - Plataforma mecânica montada .....	28
Figura 21 - Disposição dos sensores na placa superior .....	29
Figura 22 - Plataformas superior e inferior separadas.....	30
Figura 23 - Plataforma final montada .....	31
Figura 24 - Esboço do caso em que o objeto não é um ponto material .....	32
Figura 25 - <i>Sketch</i> do Motor <i>Driver Shield</i> . Fonte: [10].....	34
Figura 26 - Novo gráfico de Velocidade vs Distância .....	36
Figura 27 - Estrutura final do projeto.....	37
Figura 28 - Intervalo do ângulo de fuga para que a função <i>frente()</i> seja acionada .....	39
Figura 29 - Fluxograma final das interrupções.....	40
Figura 30 - Fluxograma final do <i>loop</i> principal.....	41





## LISTA DE TABELAS

Tabela 1 - Dados dos motores fornecidos pelo fabricante [7].....	14
Tabela 2 - Relação entre os pinos do Arduino Mega e os <i>timers</i> .....	17

# INTRODUÇÃO

## 1.1- Ambientação:

Nos dias atuais, fala-se cada vez mais do uso de robôs e plataformas para diversas funções cotidianas. Normalmente, estes são utilizados para a realização de trabalhos considerados pesados, sujos ou perigosos para o ser humano ou, ainda, para automatizar processos. Um robô pode ser considerado um dispositivo ou um conjunto de dispositivos que seja capaz de realizar uma tarefa de forma autônoma ou pré-programada.

A ideia de se construir robôs começou a tomar força no início do século XX com a necessidade de aumentar a produtividade e melhorar a qualidade dos produtos. É nesta época que o robô industrial encontrou suas primeiras aplicações. George Devol fez diversos trabalhos na área, registrando mais de 40 patentes, inclusive a primeira da história, e por essa razão ele é considerado o pai da robótica industrial. Atualmente, devido aos inúmeros recursos que os sistemas de microcomputadores nos oferecem, a robótica vive uma época de contínuo crescimento que, provavelmente, permitirá em um curto espaço de tempo o desenvolvimento de robôs inteligentes fazendo assim a ficção do homem antigo se tornar a realidade do homem atual [1].

Robótica é um ramo da tecnologia que engloba mecânica, eletrônica e computação tratando de sistemas compostos por máquinas e partes mecânicas automatizadas e controladas por circuitos integrados. Cada vez mais as pessoas utilizam os robôs para suas tarefas. Em breve, tudo poderá ser controlado por robôs. Os robôs são apenas máquinas: não sonham nem sentem e muito menos ficam cansados. Tecnologias, como a da Figura 1, são hoje adotadas por muitas fábricas e indústrias e tem obtido, de um modo geral, êxito em questões levantadas sobre a redução de custos e aumento de produtividade [2].



Figura 1 - Exemplo de um braço robô

O modo industrial continua sendo a forma mais comum de uso, porém o cenário atual está em constante mudança o que faz com que o uso doméstico esteja crescendo, além de outras aplicações de risco, como limpeza de lixo tóxico, explorações subaquática e espacial, cirurgia, mineração, busca e resgate de minas terrestres. Também existem aplicações nas áreas de cuidados de saúde e entretenimento.

Na indústria, principalmente automotiva, manipuladores industriais com capacidade de movimento semelhante ao braço humano (Figura 1) são comumente utilizados. Suas funções principais são soldagem, pintura e carregamento de máquinas. Esta tecnologia é utilizada para substituir a mão-de-obra humana em trabalhos repetitivos e perigosos, fazendo com que a empresa diminua a quantidade de acidentes de trabalho e atestados médicos e, assim, aumente a produção.

Outra utilização industrial são os AGVs (*Automatic Guided Vehicles*), como mostrado na Figura 2. AGVs são veículos guiados automaticamente usados em depósitos, hospitais, portos de containers, laboratórios, instalações de servidores e outras aplicações onde o risco, confiabilidade e segurança são os principais fatores a serem considerados. Da mesma forma, aumenta o número de prédios automatizados que fazem o uso de patrulhamento autônomo e plataformas de segurança.



**Figura 2 - Exemplo de AGV (*Automatic Guided Vehicles*)**

A robótica tem possibilitado às empresas redução de custos com o operariado e um significativo aumento na produção. O país que mais tem investido na robotização das atividades industriais é o Japão. Um exemplo disso observa-se na Toyota [2].

A expansão robótica em escritórios e residências com a substituição de aparelhos não inteligentes por seus equivalentes robóticos provavelmente continuará acontecendo, mas ainda não podemos contar com o aperfeiçoamento de robôs domésticos capazes de realizarem trabalhos caseiros como os descritos em histórias de ficção científica.

Para o trabalho com robótica, são necessários conhecimentos sobre eletrônica, mecânica e software. Noções de cinemática, pneumática, hidráulica e microcontroladores também podem ser requeridas dependendo do tamanho do projeto. O procedimento padrão para a criação de robôs começa pelo estudo dos sensores, algoritmos e atuadores que serão demandados no trabalho. Além disso, é necessário pensar o tamanho mais efetivo dos componentes para o melhor funcionamento do projeto e a fonte de alimentação primária. Assim que a plataforma básica estiver completa, os sensores e demais entradas e saídas do robô deverão ser conectados a um dispositivo capaz de tomar decisões, ou seja, o uso de um microcontrolador poderá ser necessário. Este circuito inicial será capaz de avaliar os sinais de entrada, calcular a resposta apropriada e enviar aos atuadores sinais de modo a causar uma reação da plataforma.

Pode-se notar certo grau de convergência entre humanos e robôs. Vários seres humanos têm alguma parte do corpo ou até mesmo parte do sistema nervoso substituído ou auxiliado por equivalentes artificiais como, por exemplo, o marcapasso. Em diversos casos uma mesma tecnologia pode ser utilizada tanto na medicina quanto na robótica.

Em diversos ramos a robótica gera impacto social positivo. Como, por exemplo, quando um robô é na realidade uma ferramenta para preservar o ser humano, como robôs bombeiros, submarinos, cirurgiões, entre outros tipos. O robô pode auxiliar a reintegrar algum profissional que teve parte de suas capacidades motoras reduzidas devido a doença ou acidente e, a partir utilização da ferramenta robótica ser reintegrado ao mercado. Além disto, estas ferramentas permitem que seja preservada a vida do operador.

## **1.2- Motivação:**

Este será o século da robótica e o grande salto para o desenvolvimento dessa tecnologia está sendo dado neste momento: enquanto os preços dos computadores caem a cada dia, assiste-se a um forte avanço da tecnologia sem fio e de sensores que possibilitam a corpos robóticos imitarem formas biológicas. Robôs cada vez mais sofisticados, com cérebros eletrônicos complexos, capazes de se comunicar entre si e de reagir em tempo real irão proliferar em meio às atividades mais diversas do século 21. Em breve eles poderão desempenhar funções tão delicadas quanto as que envolvem os cuidados com doentes em convalescença ou pessoas com necessidades especiais [1].

Com esse mercado crescente e com a chance de uma aceleração nas próximas décadas, vimos a necessidade de aumentar os estudos voltados para esta área. Como foi abordado na seção anterior, um robô pode ser um dispositivo ou conjunto de dispositivos que atua de forma autônoma ou pré-programada. Assim, são necessários módulos separados para a sua composição. Esses módulos podem ser projetados para diferentes usos em diversas plataformas e por isso são construídos separadamente.

Normalmente, esses dispositivos menores são projetados como plataformas independentes umas das outras e posteriormente são estudadas formas de se juntar tais partes para montar um novo robô mais completo, mais complexo e com maior funcionalidade ou uso específico para certa função.

Diante deste desafio, vimos a possibilidade de fazer um projeto que fosse uma dessas plataformas menores que pudesse ser aplicada posteriormente em sistemas maiores de segurança, limpeza ou instrumentação biomecânica.

Como atualmente a maior carência é na robótica doméstica, de segurança ou biomédica, propomos uma plataforma que fosse melhor implementada em sistemas desse tipo. Porém, com o devido estudo e os ajustes necessários, nosso projeto pode ser implementado em qualquer área da robótica.

### **1.3- Objetivos:**

O sistema proposto tem como objetivo fazer uma plataforma móvel capaz de se desviar de objetos. Assim, o sistema terá capacidade de identificar e se distanciar de obstáculos de forma autônoma considerando uma programação prévia de forma que uma vez que o objeto seja localizado, o robô se direcione para o lado oposto, ou seja, sempre 'fugindo' do obstáculo.

Para isso, é preciso definir a localização exata do obstáculo. A solução adotada para a medida de distância até o obstáculo foi por meio de sensores de ultrassom. Assim, o processador irá receber os sinais dos sensores, calcular a localização do obstáculo e o ângulo de fuga e enviar sinais para o acionador do motor fazer com que os motores sejam ativados de forma que a plataforma se mova na direção calculada.

Posteriormente, deseja-se aprimorar o projeto para torna-lo comercializável. Dessa forma, nosso projeto pode ser ajustado para fazer parte de robôs de segurança, limpeza doméstica e industrial, aprimoramento de cão-guias e cadeiras de rodas inteligentes, entre outras aplicações.

### **1.4- Estrutura do Trabalho:**

Esse trabalho conta com a seguinte estrutura de capítulos:

- *Capítulo 1 - Introdução:* conta com uma ambientação para que o leitor consiga um contexto para a leitura do projeto e apresenta quais foram as motivações para a escolha do trabalho e objetivos principais a serem alcançados pelo projeto.
- *Capítulo 2 - Hardware:* onde serão explicados os detalhes do funcionamento do hardware utilizado no projeto como, por exemplo, os sensores de ultrassom, Arduino, motores e placa de acionamento do motor (*motor shield*).

- *Capítulo 3 - Programas:* neste capítulo mostramos algoritmos de como é o funcionamento básico da plataforma. Assim, será possível um melhor entendimento da forma em que o robô atua.
- *Capítulo 4 - Ensaio e testes:* aqui mostramos toda a escolha de componentes, montagem e desenvolvimento do projeto, expondo os problemas encontrados e as soluções utilizadas.
- *Capítulo 5 - Resultados:* capítulo em que se pode observar o resultado final do projeto após toda a montagem, ensaios e testes. Nesta seção mostramos todos os detalhes finais, funcionamento efetivo da plataforma e formas de aprimorar.
- *Capítulo 6 - Conclusão:* concluiremos mostrando como a plataforma atinge os objetivos propostos, além de citar formas de melhorar e continuar o projeto e possíveis utilidades para o robô da forma como será apresentado.

# HARDWARE

## 2.1- Arduino:

Arduino é uma plataforma de prototipagem eletrônica de placa única *open-source* baseada em *hardware* e *software* flexíveis e de fácil utilização. Projetado com um microcontrolador Atmel AVR com suporte de entrada/saída (E/S) embutido e linguagem de programação padrão que é semelhante ao C/C++. É destinado a engenheiros, artistas, *designers*, estudantes e a qualquer pessoa interessada em criar objetos, ambientes, ou sistemas interativos. A principal finalidade do projeto é criar ferramentas acessíveis, com baixo custo, flexíveis e de fácil utilização principalmente para aqueles que não teriam acesso aos controladores mais sofisticados ou ferramentas mais complicadas [3]. Existem vários produtos dentro da família Arduino, como Arduino Mega, Arduino UNO, Arduino DUE, Arduino Leonardo, Arduino Nano, Arduino LilyPad, dentre muitas outras.

Um microcontrolador é um dispositivo capaz de controlar um *hardware* para fazer funções específicas. É um circuito integrado que contém processador, memória e periféricos de E/S. Eles são embarcados no interior de algum outro dispositivo para que possam controlar as funções ou ações do produto.

Uma placa Arduino é composta por um microcontrolador Atmel AVR de 8 bits, linhas E/S digitais e analógicas, além de uma interface serial ou USB que é utilizada para interligar-se a um computador hospedeiro que tem como objetivo programá-la em várias linguagens. Sua programação é feita com sintaxe baseada na linguagem de programação C++ e a programação é mantida na memória *flash* do microcontrolador, assim ela é mantida mesmo que a alimentação seja cortada. A placa possui componentes complementares para facilitar a programação e incorporação de novos circuitos. Por ter conectores expostos de forma padrão, é possível interligar a plataforma a outros módulos expansivos que são conhecidos como *shields*.

A maioria dos pinos e E/S podem ser utilizados em outros circuitos. Versões como a Duemilanove possuem 14 pinos digitais, sendo que 6 destes podem produzir sinais PWM, e 6 entradas analógicas disponíveis em cima da placa através de conectores fêmeas espaçados em 0,1 polegadas. Outros modelos, como o Nano, fornecem conectores machos na parte inferior da placa possibilitando que sejam conectados em *protoboards* [3].

Na maioria das vezes, as placas possuem um regulador de tensão linear de 5 volts em conjunto com um oscilador de cristal de 16 MHz (podendo haver variantes com um ressonador cerâmico), embora alguns esquemas, como o LilyPad, usam até 8 MHz e dispensam um regulador de tensão embutido devido a sua forma específica de restrições de fator. O controlador também é pré-programado com um *bootloader* que simplifica o carregamento de programas para a memória *flash* embutida.

Conceitualmente, quando seu software é utilizado, ele ordena todas as placas sobre uma programação com conexão serial RS-232, mas a maneira que é implementado no hardware varia em cada versão. Suas placas seriais contêm um simples circuito inversor para converter entre os sinais dos níveis RS-232 e TTL. Atualmente, existem alguns métodos diferentes para realizar a

transmissão dos dados, como por placas programáveis via USB, adicionadas através de um chip adaptador USB-para-Serial como o FTDI FT232. Algumas variantes, como o Arduino Mini e o não oficial Boarduino, usam um módulo, cabo adaptador USB, Bluetooth ou outros métodos. Nestes casos, são usados com ferramentas particulares ao invés do Arduino IDE, utilizando assim a programação padrão AVR ISP [3].

O Arduino IDE é um software escrito em Java derivado dos projetos *Processing* e *Wiring*. Seu objetivo é introduzir a programação a pessoas não familiarizadas com o desenvolvimento de software. A aplicação inclui um editor de código com recursos como realce de sintaxe, parênteses correspondentes e identificação automática, além de ser capaz de compilar e carregar programas para a placa facilmente, fazendo com que não haja necessidade de editar *Makefiles* ou rodar programas em ambientes de linha de comando.

Tendo uma biblioteca chamada *Wiring*, ele possui a capacidade de programar em C/C++. Isto permite criar com facilidade muitas operações de entrada e saída, tendo que definir apenas duas funções no pedido para fazer um programa funcional. A função *setup()*, inserida no início, na qual pode ser usada para inicializar configuração, e a função *loop()* chamada para repetir um bloco de comandos ou esperar até que seja desligada [3]. A seguir, temos uma tela do programa com um *sketch* exemplo:



```
Arduino - 0011 Alpha
File Edit Sketch Tools Help
Blink
/*
 * Blink
 *
 * The basic Arduino example. Turns on an LED on for one second,
 * then off for one second, and so on... We use pin 13 because,
 * depending on your Arduino board, it has either a built-in LED
 * or a built-in resistor so that you need only an LED.
 *
 * http://www.arduino.cc/en/Tutorial/Blink
 */

int ledPin = 13;           // LED connected to digital pin 13

void setup()              // run once, when the sketch starts
{
  pinMode(ledPin, OUTPUT); // sets the digital pin as output
}

void loop()               // run over and over again
{
  digitalWrite(ledPin, HIGH); // sets the LED on
  delay(1000);                // waits for a second
  digitalWrite(ledPin, LOW);  // sets the LED off
  delay(1000);                // waits for a second
}

Done compiling.

Binary sketch size: 1098 bytes (of a 14336 byte maximum)

22
```

Figura 3 - Sketch exemplo no espaço Arduino IDE.



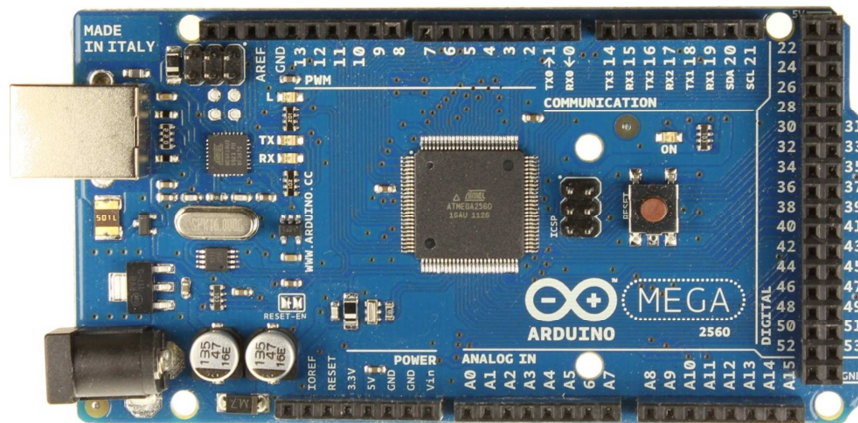


Figura 4 - Arduino Mega

Na Figura 4 temos o Arduino Mega que é uma placa baseada no microcontrolador ATmega1280 ou ATmega2560 e foi escolhida para este projeto. Possui 54 entradas/saídas digitais (das quais 14 podem ser usados como saídas PWM), 16 entradas analógicas, 4 UARTs (*Universal Asynchronous Receiver-Transmitter*), um oscilador de cristal de 16 MHz, uma conexão USB, um cabeçalho ICSP (conector para programação), um conector de alimentação e um botão de *reset*. Ele contém todo o necessário para suportar o microcontrolador. Basta conectá-lo a um computador com um cabo USB, ligá-lo com um adaptador AC ou bateria para funcionar. Além de ser compatível com a maioria dos *shields* feitos para o Arduino Duemilanove [4].

A placa pode operar com uma fonte externa de 6 a 20 volts. No entanto, se for fornecido menos que 7 V, o regulador de tensão poderá não funcionar adequadamente e fornecer uma tensão abaixo da necessária, assim, é possível que a placa fique instável. Se for usado mais que 12 V, o regulador de tensão pode superaquecer e danificar a placa. Assim, a faixa de tensão recomendada é entre 7 e 12 V. O microcontrolador tem 128KB de memória interna para armazenamento de código (sendo que 4 KB são usados para o *bootloader*), 8 KB de SRAM e 4 KB de EEPROM.

Neste projeto, o Arduino Mega é responsável pelas seguintes funções:

- Receber e analisar os sinais enviados pelos sensores de ultrassom;
- Calcular o ângulo/direção de fuga;
- Enviar sinais para o motor *shield* controlar os motores da forma adequada.

## 2.2- Sensor de Distância Ultrassônico HC-SR04:

Uma forma muito comum de medir distâncias, sem contato direto, é com o emprego de ultrassom (sons acima de aproximadamente 20 kHz). Assim como no sonar ou radar, um pulso ultrassônico é gerado na direção desejada. Se houver um objeto nessa direção, o pulso será refletido de volta como um eco e será detectado. Desta forma, é possível determinar a distância do objeto medindo a diferença de tempo entre a emissão e recepção do pulso. Este é o método que os morcegos utilizam para detectar suas presas e os obstáculos.

Para o funcionamento de sensores ultrassônicos, é necessária uma fonte emissora de ultrassons e um microfone ultrassônico. A função do emissor é emitir um pulso de curta duração. Este pulso será propagado e refletido no objeto que se deseja detectar. Assim, esses pulsos de reflexão irão retornar e serão captados pelo receptor.

Para aplicações práticas, os padrões de reflexão do ultrassom são determinados pelo formato e tipo do objeto. Ou seja, o reflexo em uma superfície plana na perpendicular será praticamente na mesma direção da emissão enquanto objetos de outros formatos provocarão reflexões com diversos padrões. Assim, um maior rendimento será obtido se for possível direcionar a maior parte do som para o objeto que se deseja detectar. Para isso, recursos como refletores e cornetas garantem as características direcionais para os pulsos emitidos pelos sensores, de forma semelhante às antenas de rádio.

Da mesma forma que nas antenas, existem pequenos lóbulos secundários (laterais) no diagrama de irradiação de um sensor. Isto ocorre devido ao fenômeno da refração das bordas do sistema acústico que direciona o ultrassom. Outro fato interessante é que dificilmente a forma do sinal refletido é a mesma do sinal emitido, devido às características do objeto a ser detectado. Este fenômeno que altera a forma original da onda é chamado reverberação e deve ser considerado ao projetar o circuito detector. Estes fenômenos podem ser visualizados na figura abaixo:

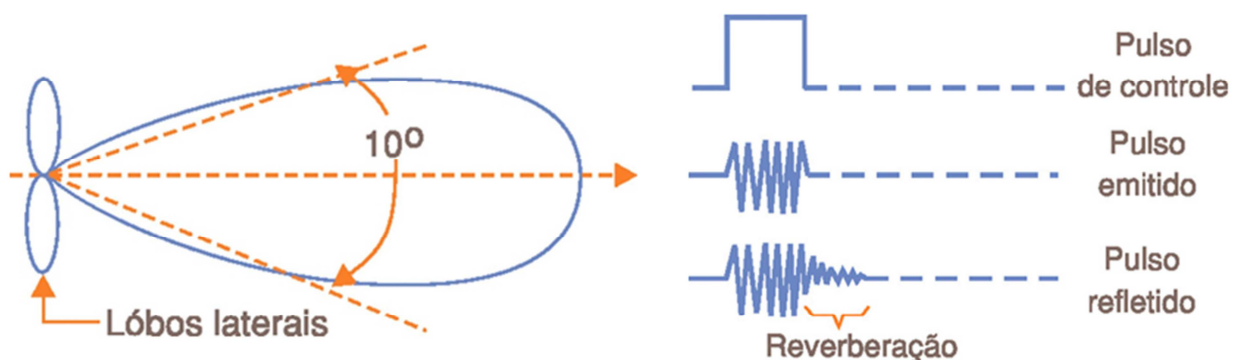


Figura 5 - Exemplificações de lóbulos laterais (à esquerda) e reverberação (à direita)

De forma geral, os sensores possuem um oscilador que gatilha um monoestável responsável por determinar o tempo previsto de retorno do eco para o receptor. Se o eco demorar mais do que o previsto, significa que o objeto está longe demais para ser detectado. Entretanto, se o eco for captado dentro do intervalo previsto, o circuito contador do monoestável é interrompido e se obtém a distância do objeto detectado. No entanto, se não houver necessidade dessa informação, o circuito poderá ser simplificado e ao se detectar o eco, é produzido um pulso único para disparo de algum circuito externo.

Para este projeto específico, utilizamos o Sensor Ultrassônico HC-SR04 (Figura 6). Este sensor, de dimensões 4,5 x 2 cm, é capaz de medir distâncias de 2 cm a 4 m com uma precisão de 3 mm e ângulo de efeito de 15°. Sua alimentação é de 5 V e, neste projeto, será utilizada a alimentação do próprio Arduino. O módulo possui um circuito pronto com emissor e receptor acoplados e 4 pinos (VCC, Trigger, ECHO, GND) para medição.



Figura 6 - Sensor Ultrassônico HC-SR04

Para realizar a medição é necessário colocar o pino *Trigger* em nível alto por pelo menos 10  $\mu$ s. Dessa forma, o sensor emite 8 ciclos de onda com uma frequência de 40 kHz que, após encontrar um obstáculo, são refletidas em direção ao módulo. O pino *Echo* fica em nível alto durante o tempo entre a emissão e o recebimento do sinal. Assim, é possível calcular a distância de acordo com o tempo em que o pino *Echo* permaneceu em nível alto após o *Trigger* ter sido colocado em nível alto.

Considerando a velocidade do som como aproximadamente 340 m/s, ajustando as unidades e lembrando o fato de que a onda é enviada e rebatida, ou seja, percorre duas vezes a distância a ser calculada, temos a seguinte fórmula para obter a distância:

$$Distância = \frac{(Tempo\ do\ Echo\ em\ nível\ alto * Velocidade\ do\ Som)}{2} \quad (1)$$

$$Distância = \frac{Tempo\ do\ Echo\ em\ nível\ alto}{58.2} \quad (2)$$

Sendo que a distância será dada em centímetros e o tempo do *Echo* em nível alto é dado em microssegundos.

### 2.3- Motor Driver Shield L293D:

*Shields* são módulos de fácil conexão e manuseamento, capazes de se encaixar no topo de determinadas placas de Arduino possibilitando a expansão de suas capacidades. Este nome, que significa escudo em inglês, vem da aparência semelhante a um escudo quando está acoplado ao Arduino. Como possuem seus próprios pinos para fazer o encaixe físico, são fáceis de manusear e bastante amigáveis, além de eliminar a necessidade de fios ou conectores o que reduz grande parte da chance de mau contato. A maior parte deles possui conectores não só na parte inferior, mas também na parte superior o que permite o encaixe de vários módulos um em cima do outro possibilitando, assim, a diversificação das funções do projeto. Existem diversos fabricantes de diferentes modelos de *shields*, com os mais diversos formatos e funcionalidades o que permite se criar uma grande quantidade de possibilidades.

Neste projeto usaremos o *Motor Driver Shield L293D* (Figura 7) projetado pela Adafruit em 2008. Este é um *shield* com componentes antigos, porém ainda muito útil, que é compatível com os Arduinos Mega, Diecimila e Duemilanove. O módulo contém dois motor *drivers* L293D e um registrador de deslocamento 74HC595. O registrador de deslocamento expande 3 pinos do Arduino para 8 pinos para o controle de direção para os motor *drivers*, enquanto as saídas dos L293D estão diretamente conectadas as saídas PWM do Arduino [5].

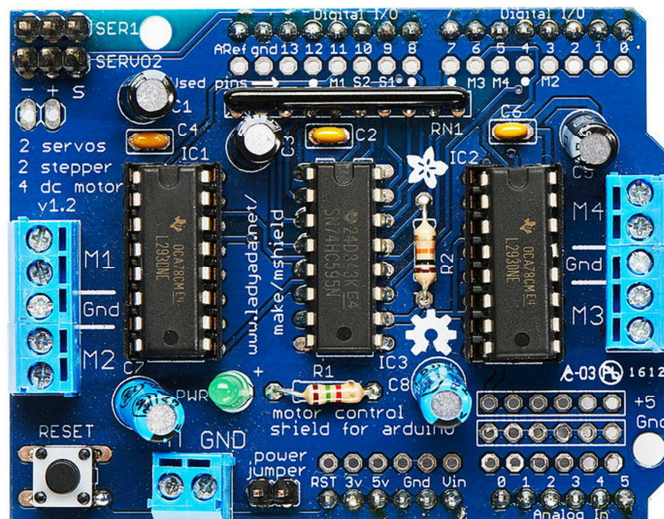


Figura 7 - Motor Driver Shield L293D

Este módulo é capaz de acionar 2 servo-motores e tem 8 saídas por uma Ponte H que podem ser usadas para controlar 2 motores de passo, 4 motores DC bidirecionalmente ou 8 motores DC unidirecionalmente (ou uma combinação entre os tipos de motores) e suporta motores com tensões de 4,5 V e 25 V. Os servo-motores usam a saída de 5 V da placa do Arduino, mas é preciso um cuidado especial para não esquentar o regulador de tensão da placa.

Cada L293D possui 4 pontes H, cada uma é capaz de fornecer uma corrente de 0,6 A, ou 1,2 A de pico, com uma proteção de temperatura. Para aumentar essa corrente, é possível colocar outros L293D como 'piggyback', ou seja, soldando outros CIs no topo do L293D do *shield* [5].

O *shield* pode ser alimentado pelo próprio Arduino ou de forma independente. Porém, é preciso lembrar que se for utilizar duas fontes de alimentação, é necessário retirar o *jumper* da parte superior da placa. Este procedimento é devido ao fato do Arduino possuir um diodo de proteção para o caso de fornecermos uma tensão diferente da que ele foi projetado, enquanto o *shield* não possui essa proteção. Assim, se não desconectarmos o *jumper*, é possível que se danifique o *shield* e/ou o Arduino.

#### 2.4- Motor DC:

Motor DC é uma máquina de corrente contínua capaz de converter energia elétrica em energia mecânica. Seu funcionamento se baseia no fato de que polos magnéticos do mesmo tipo se repelem enquanto polos opostos se atraem e que uma bobina, por onde passa corrente, gera um campo eletromagnético. Este campo só existirá na presença de corrente e poderá ser invertido com a troca do sentido da corrente [6].

Normalmente, motores possuem uma parte giratória que geralmente está no interior e é chamada de rotor, enquanto a parte estacionária é chamada de estator, como pode ser visto na Figura 8. O corpo do rotor é constituído por eletroímãs que são arranjados de modo que um torque seja desenvolvido na sua linha central, fazendo com que gire.

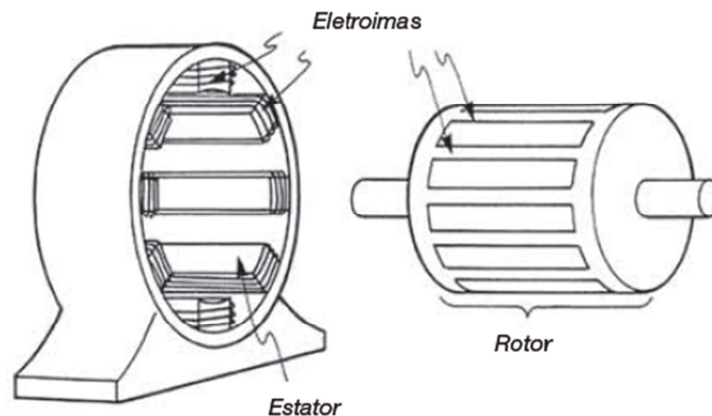


Figura 8 - Esquemático de um Motor DC

O funcionamento do motor é feito da seguinte maneira: uma tensão elétrica é aplicada aos terminais pelo anel comutador (responsável por fazer as inversões do sentido da corrente) fazendo com que circule uma corrente que produz um campo magnético no enrolamento da armadura (outra denominação do rotor). Ao se aplicar tensão nos terminais do enrolamento do estator, temos a intensificação dos campos magnéticos e, com isso, a produção de polos magnéticos por toda a sua extensão.

A orientação do campo e as posições dos polos norte e sul do rotor permanecem fixas. Assim, quando uma tensão é aplicada no estator, os polos tentarão se alinhar, ou seja, o polo norte de um dos campos tentará se aproximar do polo sul do outro. Como o eixo pode girar, surgem forças que tentam alinhar os polos e produzirão torque. Ao girar, o anel comutador irá mudar o sentido de aplicação da tensão fazendo com que a corrente circule no sentido contrário e invertendo o

campo magnético produzido. Ou seja, ao girar, as posições dos polos magnéticos mudam enquanto o campo do estator permanece fixo fazendo com que, novamente, surjam forças que irão manter o movimento do eixo da máquina [6].

As diferentes características de velocidade e torque de cada motor são devidas às quantidades de enrolamentos nos rotores e estatores. A velocidade do motor pode ser controlada pela alteração da tensão aplicada nos terminais da armadura. Assim, uma resistência variável no circuito da armadura pode servir para o controle de velocidade. Os motores mais modernos costumam ser controlados por sistemas eletrônicos que ajustam a tensão pelo controle da corrente do sistema.

Como esse tipo de motor desenvolve um maior torque para baixas velocidades, muitas vezes é utilizado para tracionar locomotivas elétricas ou bondes. Essa foi a única aplicação utilizada por muitos anos até a década de 1870, com a Segunda Revolução Industrial, quando começaram a aplicar tais motores em sistemas elétricos para o funcionamento de máquinas.

De modo geral, motores de corrente contínua podem operar com baterias recarregáveis, o que proporciona o funcionamento dos atuais carros elétricos/híbridos ou até mesmo de ferramentas sem fio. Atualmente, tais motores são comumente aplicados desde projetos menores como brinquedos até maiores como laminadores de aço e fabricação de papel.



**Figura 9 - Motor DC usado no projeto**

No nosso projeto, temos 4 motores de pequeno porte (Figura 9) que foram adquiridos juntos com o chassi da plataforma. Assim, a princípio, tivemos poucas informações pela falta de um *datasheet* que contivesse todas as características dos motores. Porém, os fabricantes disponibilizaram a tabela abaixo que foi o suficiente para o nosso trabalho.



Voltage	DC 3V	DC 5V	DC 6V
Current	100 mA	100mA	120mA
Reduction rate	48:1		
RPM (With tire)	100	190	240
Tire Diameter	66mm		
Car Speed(M/minute)	20	39	48
Motor Weight (g)	50		
Motor Size	70mm*22mm*18mm		
Noise	<65dB		

**Tabela 1 - Dados dos motores fornecidos pelo fabricante [7]**

Pela tabela, vemos que são motores projetados para atuarem de 3 a 6 V, com uma corrente máxima de 120 mA e uma velocidade máxima de 240 rpm, lembrando que tais características são para o motor atuando sem carga. Assim, pode-se observar que os motores são compatíveis com o motor *driver* [7].

# PROGRAMAÇÃO

## 3.1- Programação básica:

### 3.1.1- Arduino:

Toda a programação do sistema é feita e transferida pelo Ambiente de Desenvolvimento Integrado (IDE) do Arduino. O *software* é disponível livremente para *download* na página da plataforma na internet [4]. Como foi dito anteriormente, a linguagem de programação utilizada é baseada em C++. Os programas (ou *sketchs*) são formados por: declaração de bibliotecas, declaração de variáveis globais (as variáveis que poderão ser utilizadas em qualquer parte do programa), função *void setup*, função *void loop* e outras demais funções utilizadas do projeto. As funções *setup* e *loop* são, na verdade, blocos de funções, pois é possível chamar outras funções ou sub-rotinas dentro delas. Outra observação importante é que elas sempre aparecerão nessa ordem.

A função *setup* é a primeira a ser chamada quando o programa inicia e é chamada apenas uma única vez. É a função de preparação, ou seja, nela colocamos o comportamento dos pinos do Arduino e inicializamos a porta serial. Assim, suas principais aplicações são: configurar o Arduino, utilizar os parâmetros das bibliotecas, inicializar variáveis, definir o modo de operação dos pinos e a velocidade da comunicação serial.

A função *loop* é chamada logo a seguir e todas as funções embarcadas nela são executadas repetidamente. Ela lê os pinos de entrada, comandando os pinos de saída e a porta serial. Ou seja, é uma função executada em ciclo contínuo com objetivo de criar um laço infinito onde o programa se altere e se repita à medida que os parâmetros externos do hardware se modifiquem ao longo do tempo. Portanto, é a função que permite um controle ativo da plataforma.

As outras estruturas de comando são: *if*, *if...else*, *if...else if*, *switch...case*, *while*, *do...while* e *for*. Os três primeiros comandos listados tem o objetivo de comparação de casos, assim, definida uma expressão, o comando irá analisar se é verdadeira ou falsa e, baseado nisso, indicará quais instruções seguir. O comando *switch...case* é utilizado para simplificar casos onde seriam necessárias diversas estruturas *if...else*, fazendo que seja possível selecionar casos específicos, ou seja, permitindo comparar uma mesma expressão com vários valores possíveis. Os comandos *while* e *do...while* são semelhantes e sua principal operação é repetir um grupo de instruções até que uma certa condição mude. A diferença entre esses comandos é que no segundo, as instruções serão seguidas ao menos uma vez até a comparação, enquanto na primeira as instruções somente serão seguidas no caso da comparação for verdadeira, ou seja, se a comparação for falsa, o bloco de instrução não será lido. O comando *for* serve como um tipo de laço que deve ter uma variável de controle previamente inicializada com um tipo e um valor, normalmente é 0 ou 1, e uma expressão que deve conter o valor máximo ou mínimo que o contador deve alcançar quando o incremento é adicionado ou subtraído da variável a cada vez que o bloco de instruções é executado [3]. A seguir, temos a Figura 10 que mostra os fluxogramas destes comandos.



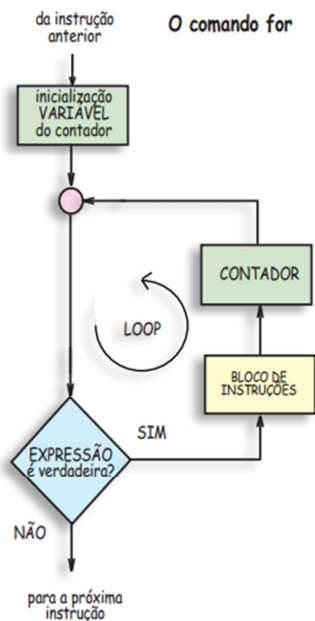
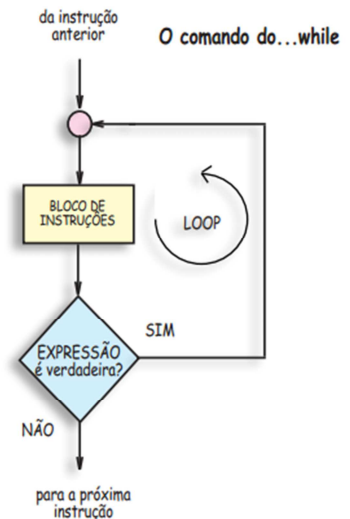
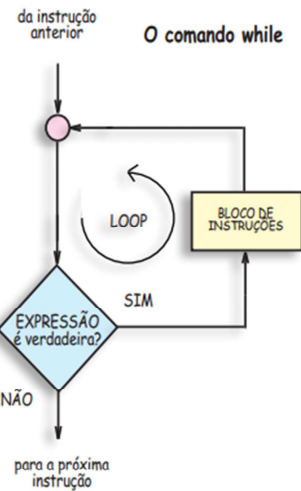
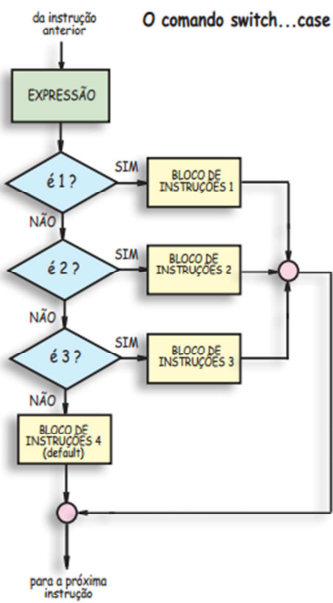
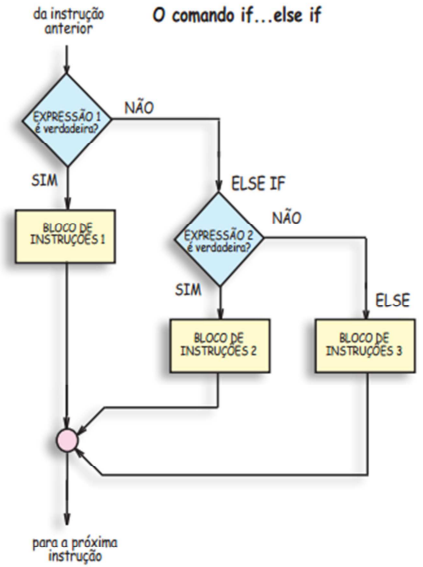
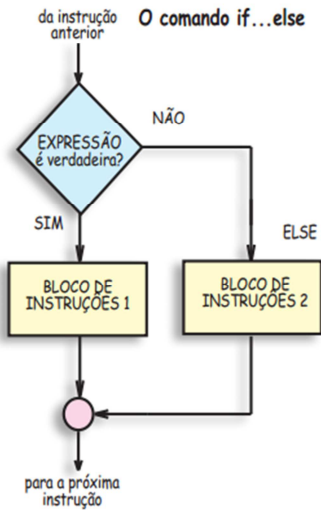
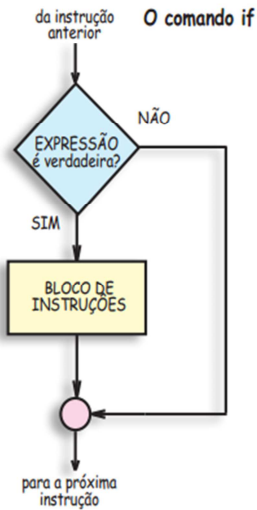


Figura 10 - Fluxogramas que exemplificam o uso das funções de programação do Arduino. Fonte: [8]

Outros conceitos importantes para a programação de microcontroladores, em especial o Arduino Mega, são os *timers* ou temporizadores e as interrupções.

Um *timer* nada mais é do que um contador que é incrementado a cada intervalo de tempo, onde esse intervalo pode ou não ser configurado. Os *timers* funcionam como uma espécie de relógio que pode ser usado para a contagem de tempo, medição da duração de certos eventos, etc [9].

Todos os modelos de Arduino vêm equipados com microcontroladores AVR que possuem os *timers* 0, 1 e 2. Os *timers* 0 e 2 são contadores de 8 bits, ou seja, contam de 0 a 255, enquanto o *timer* 1 é um contador de 16 bits, contando de 0 a 65535. O Arduino Mega, além de vir equipado com esses *timers*, ainda possui os *timers* 3, 4 e 5. Todos esses são contadores de 16 bits e somente serão utilizados pela biblioteca da placa caso o programa deseje controlar servos [9].

No Arduino o programa é executado sequencialmente, linha após linha, e uma interrupção é um evento externo que interrompe a execução do programa e chama uma função específica ou ISR (*Interrupt Service Routine*). Após o fim dessa rotina, a execução normal do programa é retomada de onde parou. Os *timers* podem ser configurados para quando atingirem o valor máximo da contagem (o que chamamos de *overflow*) gerarem essa interrupção e chamarem uma função específica. No caso do nosso projeto, a função chamada é a função responsável pela leitura de cada sensor.

Para o seu funcionamento, os *timers* utilizam certos pinos do Arduino. No caso do Arduino Mega, a listagem dos *timers* e pinos usados segue na tabela abaixo:

Timer	Pinos
0	4 e 13
1	11 e 12
2	9 e 10
3	2, 3 e 5
4	6, 7 e 8
5	44, 45 e 46

Tabela 2 - Relação entre os pinos do Arduino Mega e os *timers*

### 3.1.2- Motor Driver Shield:

Para a programação do *Motor Driver Shield*, também utilizamos o software do Arduino, ou seja, o Arduino IDE. Para isso, é preciso incluir a biblioteca *AFMotor.h* que é a biblioteca específica desse *shield*. Esta biblioteca disponibiliza funções que definem a velocidade e a direção de cada motor.

```
#include <AFMotor.h>
```

Depois da inclusão da biblioteca, é necessário definir cada motor, ou seja, definir os nomes e posições dos motores nas conexões M1, M2, M3 e M4 (como pode ser visto na Figura 7) da parte superior da placa. Para isso, existe a função *AF\_DCMotor* que especifica o nome do motor, onde ele está ligado e a frequência adotada para o sinal (as frequências possíveis são 1, 2, 8 e 64kHz e, caso não se especifique, a frequência será de 1 kHz).

`AF_DCMotor nomedomotor (posição, frequência)`

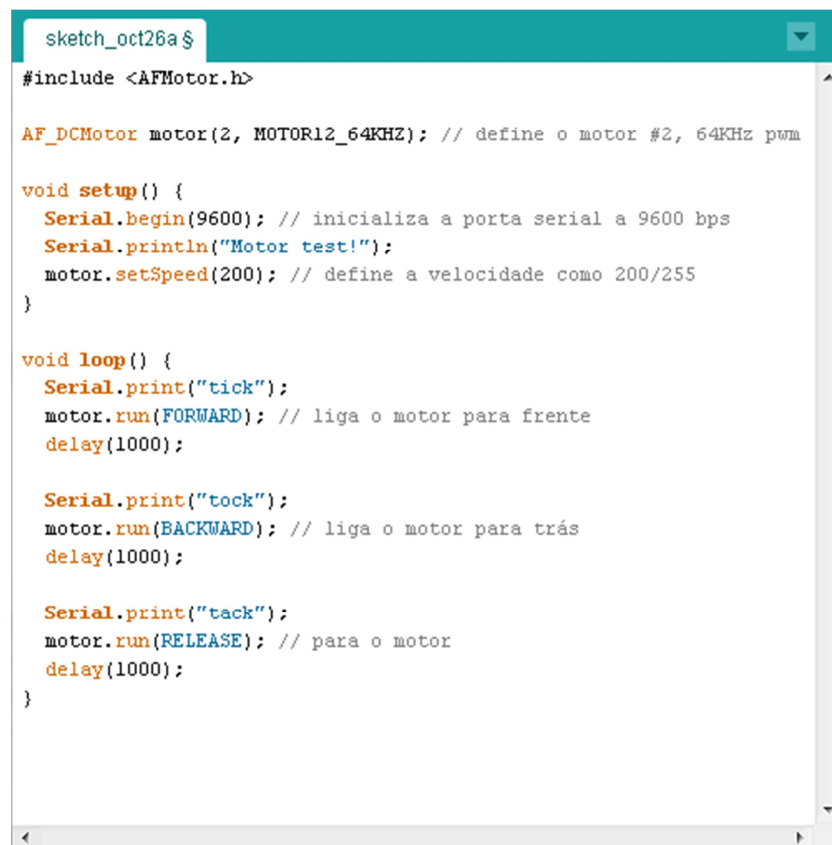
Após definir os motores, usamos a função *setSpeed* para definir as velocidades em que eles deverão funcionar, onde essas velocidades vão de 0 (parado) a 255 (velocidade máxima).

`nomedomotor.setSpeed (velocidade)`

Por último, usamos a função *run* para definir a direção em que os motores deverão girar. Essas direções são FORWARD, BACKWARD e RELEASE que irão girar o motor para frente, para trás e parar o motor, respectivamente. Lembrando que o Arduino não sabe a direção dos motores, simplesmente inverte a tensão dos seus terminais para fazê-lo girar na direção contrária [5].

`nomedomotor.run (direção)`

A seguir, temos um exemplo de *sketch* simples que liga em *loop* o motor para frente, para trás e para por 1 segundo em cada estado. O motor está na posição 2, em uma frequência de 64 kHz e uma velocidade de 200 ( 200/255 da velocidade máxima).



```
sketch_oct26a $
#include <AFMotor.h>

AF_DCMotor motor(2, MOTOR12_64KHZ); // define o motor #2, 64KHz pwm

void setup() {
  Serial.begin(9600); // inicializa a porta serial a 9600 bps
  Serial.println("Motor test!");
  motor.setSpeed(200); // define a velocidade como 200/255
}

void loop() {
  Serial.print("tick");
  motor.run(FORWARD); // liga o motor para frente
  delay(1000);

  Serial.print("tock");
  motor.run(BACKWARD); // liga o motor para trás
  delay(1000);

  Serial.print("tack");
  motor.run(RELEASE); // para o motor
  delay(1000);
}
```

Figura 11 - Sketch exemplo de programação do Motor *Driver Shield* no ambiente Arduino IDE

### 3.2- Algoritmo para o cálculo da distância do objeto e do ângulo de fuga:

Como foi dito anteriormente, o método utilizado para o cálculo da distância do obstáculo foi por meio de sensores de ultrassom. Para isso, utilizamos 8 sensores HC-SR04 nas laterais de um octógono inscrito (que pode ser visto na Figura 21) em uma circunferência de raio de 10 cm localizados na parte superior da plataforma. Cada sensor nos fornece a distância do objeto mais próximo encontrado na sua direção. Como vimos, essa distância (em centímetros) é dada pela Equação 2:

$$Distância = \frac{Tempo\ do\ ECHO\ em\ nível\ alto}{58.2} \quad (2)$$

Se considerarmos que o ângulo aumenta no sentido anti-horário e que os sensores estão arrumados nos vértices de um octógono de raio R e lado L, obtemos o esquema apresentado na Figura 12.

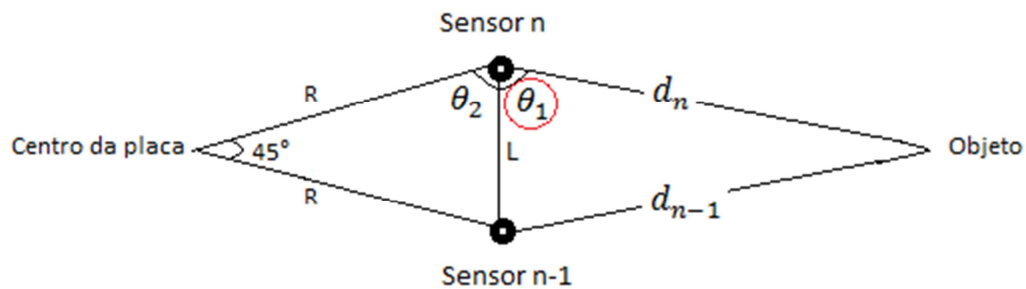


Figura 12 - Esboço para melhor visualização do cálculo de  $\theta_1$

Para calcularmos  $\theta_1$ , utilizamos uma manipulação da lei dos cossenos que resulta na seguinte equação:

$$\theta_1 = \cos^{-1} \left( \frac{d_n^2 + L^2 - d_{n-1}^2}{2d_n L} \right) \quad (3)$$

O ângulo  $\theta_2$  pode ser calculado da mesma forma que  $\theta_1$  pela equação a seguir, lembrando que, como R e L são valores constantes, o valor desse ângulo também será constante.

$$\theta_2 = \cos^{-1} \left( \frac{L}{2R} \right) \quad (4)$$

Conforme podemos observar na próxima figura, podemos obter o ângulo  $\theta_T$  da seguinte maneira:

$$\theta_T = \theta_1 + \theta_2 \quad (5)$$

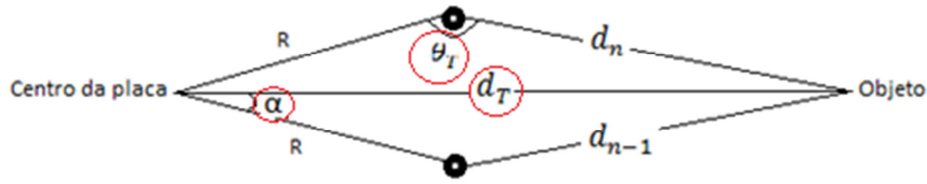


Figura 13 - Esboço para melhor visualização do cálculo de  $\theta_T$ ,  $\alpha$  e  $d_T$

Utilizando novamente uma manipulação da lei dos cossenos podemos calcular  $d_T$ , ou seja, calculamos a distância do objeto até o centro do octógono (que também é o centro da placa). Lembrando que os valores  $d_n$  e  $d_{n-1}$  não necessitam serem equivalentes (assim o objeto pode estar em outras posições diferentes das mostradas nas Figuras 12 e 13) e são fornecidos pelos sensores pela Equação 2 sendo que  $0 \leq n \leq 7$ , temos:

$$d_T = \sqrt{R^2 + d_n^2 - 2Rd_n \cos \theta_T} \quad (6)$$

Uma vez que calculamos todos esses parâmetros, podemos calcular o ângulo  $\alpha$ , também por manipulação da lei dos cossenos, da seguinte maneira:

$$\alpha = \cos^{-1} \left( \frac{R^2 + d_T^2 - d_{n-1}^2}{2Rd_T} \right) \quad (7)$$

Se transladarmos o ângulo  $\alpha$  para o intervalo dos sensores correspondentes, temos a seguinte equação:

$$\varphi = \alpha + (n - 1) \cdot \frac{\pi}{4} \quad (8)$$

Por último, para obtermos o ângulo de fuga, adicionamos  $\pi$  a  $\varphi$ , assim:

$$\varphi_{fuga} = \varphi + \pi \quad (9)$$

Lembrando que os ângulos estão em radianos e as distâncias em centímetros, obtemos os dados necessários para a programação do funcionamento dos motores. A figura a seguir mostra um exemplo onde o obstáculo está entre os sensores 3 e 4:

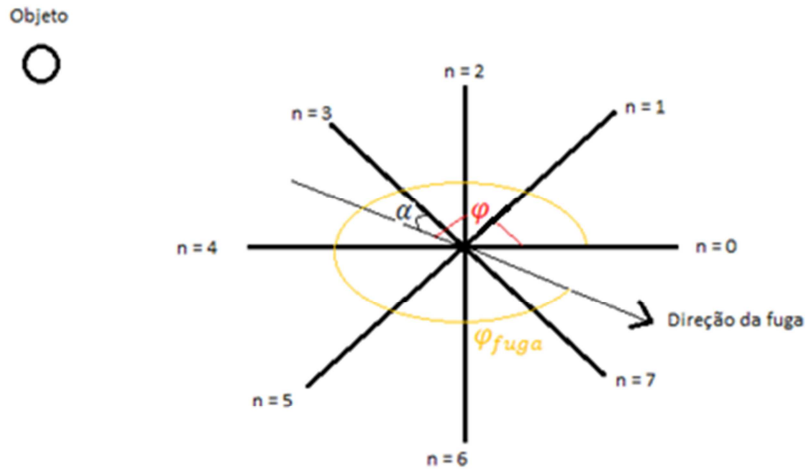


Figura 14 - Esboço para melhor localização dos ângulos

Após os cálculos descritos, as distâncias calculadas são colocadas em um vetor de médias móveis para minimizar possíveis erros de leitura. A média móvel é uma técnica usada para analisar dados em um intervalo de tempo, calculando a média através dos valores mais recentes em uma série de dados. Assim, com esse vetor calculando as médias das distâncias encontradas em cada sensor, os valores repassados para o motor não oscilam tanto fazendo com que a resposta do sistema seja mais homogênea.

### 3.3- Princípio do acionamento dos motores:

Com o avanço do projeto, estabelecemos que a plataforma 'fuja' de um determinado obstáculo com uma velocidade inversamente proporcional à sua distância, os parâmetros  $d_T$  e  $\varphi_{fuga}$  são essenciais para o funcionamento correto dos motores. Com esses parâmetros calculados, podemos programar o motor *shield* para acionar os motores da forma desejada.

Como a velocidade do robô é inversamente proporcional à distância do obstáculo, foi feita uma função como mostra a equação e o gráfico a seguir:

$$Velocidade = -0,6375 \cdot d_T + 255 \quad (10)$$

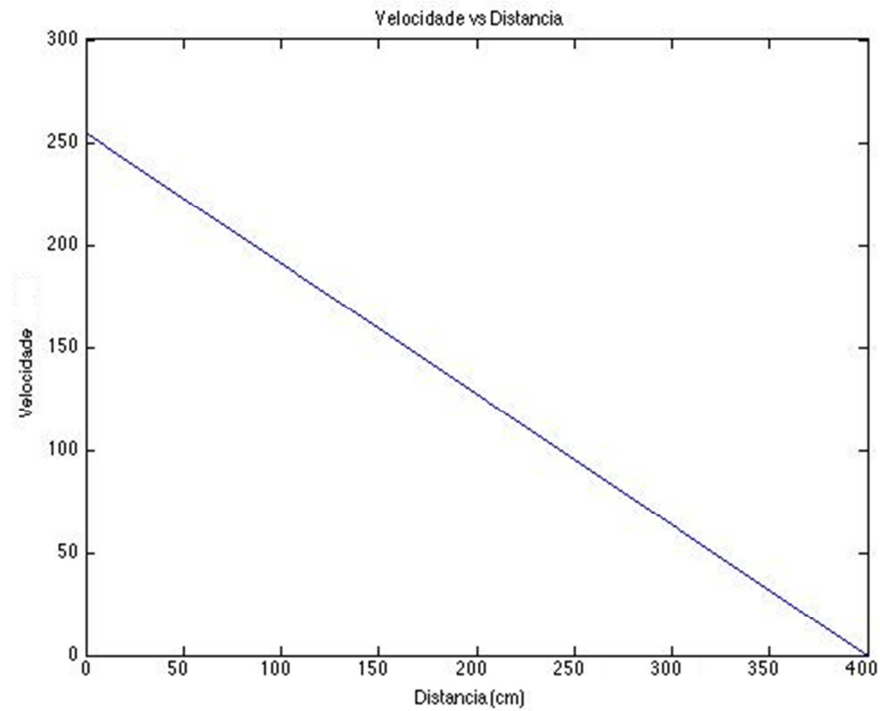


Figura 15 - Gráfico Velocidade vs Distância

Lembrando que essa velocidade indica, na realidade, uma proporção da velocidade máxima do motor. Como nossos motores serão alimentados com 6 V, sua velocidade máxima é 240 rpm, assim, temos:

$$Velocidade (rpm) = \frac{Velocidade}{255} * 240 \quad (11)$$

Definida a velocidade do motor, devemos agora nos preocupar com a direção da fuga, ou seja, devemos fazer com que os motores sigam a direção desejada. Para isso, temos o parâmetro  $\varphi_{fuga}$ .

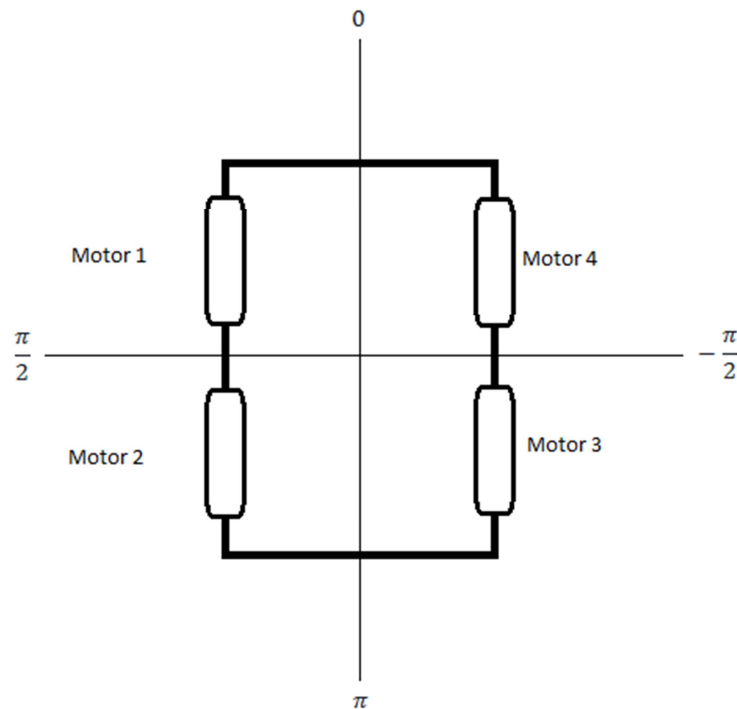


Figura 16 - Esboço das localizações dos motores na plataforma e medição do ângulo de fuga

Para seguir a direção de fuga, pode ser que os motores precisem fazer curvas e, para isso, optamos fazer com que a plataforma gire em torno do próprio eixo. Assim, definimos funções para os casos onde a plataforma deve ir para a direita, fazendo os motores 1 e 2 serem acionados para frente enquanto os motores 3 e 4 são acionados para trás, ou para a esquerda, onde os motores 1 e 2 são acionados para trás enquanto os motores 3 e 4 são acionados para frente. As posições dos motores e definição do ângulo de fuga são mostradas na Figura 16.

Para seguir o ângulo de fuga corretamente, supomos que este pode ir de  $-\pi$  a  $\pi$ . Desta forma, para ângulos entre 0 e  $\pi$  a plataforma deve ir para a esquerda, entre  $-\pi$  e 0 para a direita e seguir em frente para ângulos próximos de 0.

Caso a plataforma precise ir para frente, chamamos a função que aciona os quatro motores para frente. Porém quando a plataforma deve fazer uma curva, devemos chamar a função *direita()* ou *esquerda()* por determinado tempo  $t_{giro}$  e, então, chamamos a função *frente()*. Para determinarmos esse tempo, em que os motores devem ficar acionados conforme as funções que efetuam curvas, foram feitos testes de calibração onde os motores foram ligados na velocidade máxima e contamos o tempo que a plataforma levava para dar uma volta. O resultado foi:

$$1800 \text{ ms} \leftrightarrow 2\pi \quad (12)$$

Assim, calculamos uma função matemática que obedecesse tais critérios. A Equação 13 e a Figura 17 mostram a função e seu gráfico, respectivamente.

$$t_{giro} = 286,4873 * |\varphi_{fuga}| \quad (13)$$



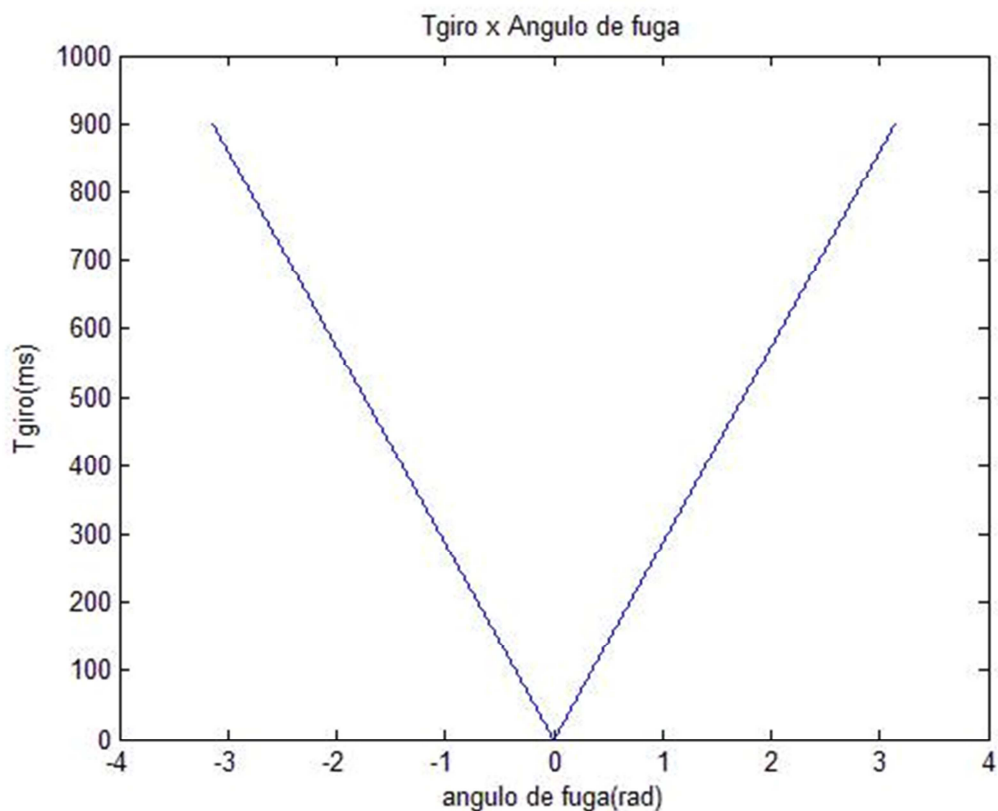


Figura 17 - Gráfico Tgiro vs Ângulo de fuga

Definidos os parâmetros *Velocidade* e  $t_{giro}$  e as funções *frente()*, *direita()* e *esquerda()*, temos nosso programa que define o princípio de funcionamento dos motores.

### 3.4- Algoritmo teórico geral:

A princípio, os programas dos sensores e dos motores foram elaborados separadamente para evitar erros maiores e para que se tornasse mais fácil de detectar possíveis falhas nos programas. Assim, o algoritmo geral com relação aos programas separados segue os seguintes passos:

1. O programa dos sensores começa com a inicialização do Timer 4 para gerar interrupções a cada 100 ms. A cada interrupção o sistema realiza a leitura de um sensor, atualiza o vetor de distâncias e as médias móveis dos sensores.
2. O pino de *trigger* do sensor, que normalmente está em nível baixo, é colocado em nível alto por 10  $\mu$ s e depois colocado novamente em nível baixo. O sinal de retorno do pulso chega pelo pino de *echo*. A CPU recebe esse sinal e o tempo transcorrido para tal é colocado na variável *duration* e, então, é calculada a distância do obstáculo pelo algoritmo descrito na seção 3.2 e este valor é colocado em uma posição no vetor daquele sensor.

3. O cálculo da média móvel é feito para cada sensor com um *loop for* que soma todos os valores do vetor do sensor e então divide por 8, que é o tamanho do vetor. O resultado é carregado no vetor de médias móveis.
4. O *sensor\_index* (variável que indica qual sensor está ativado) é incrementado para que o próximo sensor seja atualizado no próximo ciclo. Quando o indexador dos sensores atinge o número de sensores, o indexador retorna para o 0 e o *position\_index* (variável que indica qual é a atual posição no vetor) é incrementado mudando a posição no vetor das distâncias. Ao se atingir *position\_index* igual a 8, o indexador é retornado para 0 novamente.
5. Finalizando a rotina de interrupção, a variável de contagem de décimos de segundo é incrementada e utilizando funções condicionais as variáveis de contagem de segundos, minutos, horas e dias são atualizados quando se passa um segundo, um minuto, uma hora e um dia respectivamente.
6. A cada dois segundos uma *flag* é acionada indicando ao *loop* principal da passagem do tempo. Então, o *loop* chama a rotina de cálculo do ângulo de fuga.
7. A rotina ordena as médias móveis da menor para a maior, salvando as duas menores medidas e seus sensores correspondentes. Sendo que estes sensores devem ser adjacentes para a execução dos cálculos. Estes valores salvos são ordenados por ordem decrescente do número do sensor e seguem o algoritmo para o cálculo da distância do objeto ( $d_T$ ) e do ângulo de fuga ( $\varphi_{fuga}$ ) explicado anteriormente.
8. Uma vez calculados os parâmetros  $d_T$  e  $\varphi_{fuga}$ , o programa dos motores calcula a velocidade da plataforma e o  $t_{giro}$ . Sendo que, para isso,  $d_T$  precisa ser menor que 250, ou seja, a distância do objeto deve ser menor que 2,5 metros. Caso contrário, o programa entra na função *parar()*, assim, a plataforma ficará parada.
9. Caso  $d_T \leq 250$ , o programa irá analisar o  $\varphi_{fuga}$  para determinar qual direção a plataforma deverá seguir. Teremos três possíveis casos:
  - Caso  $-0,15 \leq \varphi_{fuga} \leq 0,15$ , o programa executa a função *frente()* na qual os motores andam na velocidade calculada no item anterior;
  - Caso  $-3,1415 \leq \varphi_{fuga} < -0,15$  ou  $3 \leq \varphi_{fuga} \leq 3,1415$ , o programa executa a função *direita()* durante o tempo  $t_{giro}$  e, então, executa a função *frente()*. Durante o tempo em que os motores obedecem a função *direita()*, a velocidade será a velocidade máxima, ou seja, 255. Então, quando os motores passam a obedecer a função *frente()*, a velocidade será a calculada no item anterior;
  - Caso não esteja em nenhum dos casos anteriores, ou seja,  $0,15 < \varphi_{fuga} < 3$ , o programa executa a função *esquerda()* durante o tempo  $t_{giro}$  e, então, executa a função *frente()*. Da mesma forma, durante o tempo em que os motores obedecem a função *esquerda()*, a velocidade será a velocidade máxima, ou seja, 255. Então, quando os motores passam a obedecer a função *frente()*, a velocidade será a calculada no item anterior.

A seguir temos as Figuras 18 e 19 que mostram os fluxogramas do algoritmo. Sendo que um fluxograma mostra o funcionamento da interrupção (ISR) e o outro do *loop* principal, respectivamente.

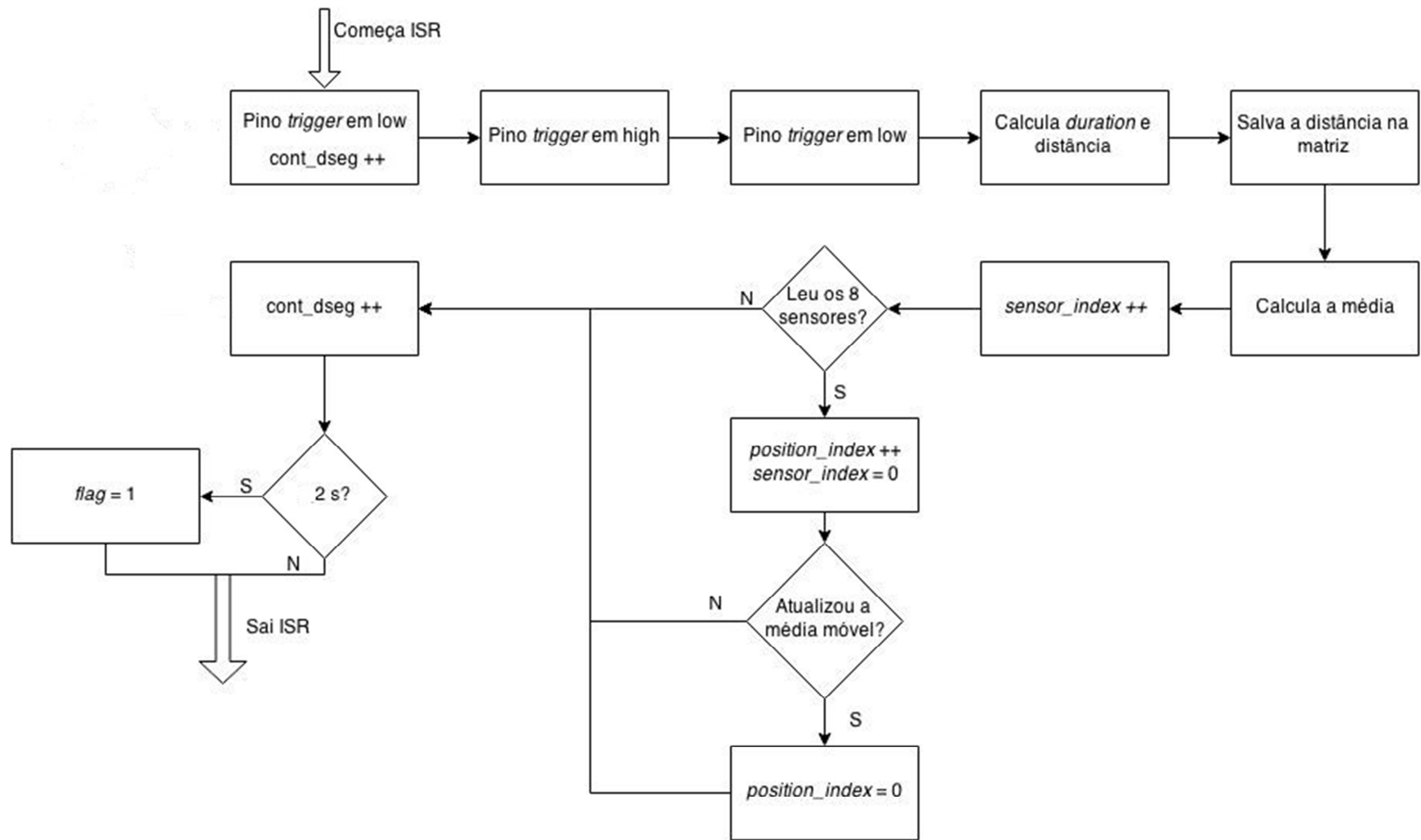


Figura 18 - Fluxograma da interrupção

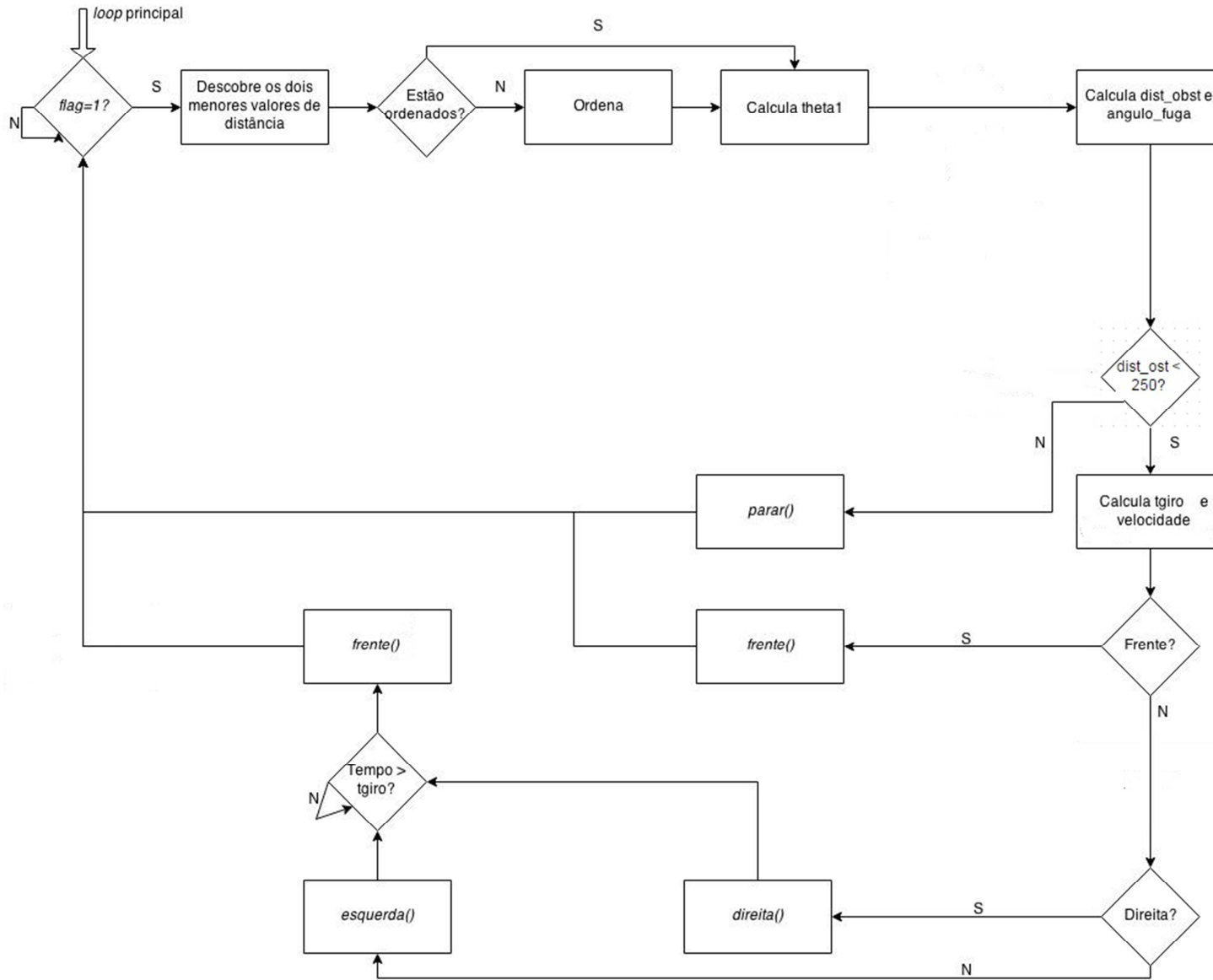


Figura 19 - Fluxograma do *loop principal*

# ENSAIOS E TESTES

## 4.1- Escolha e montagem da plataforma:

Logo após definidos os objetivos do projeto, o próximo passo foi a escolha da plataforma que seria utilizada. Tivemos que analisar as possíveis dimensões, formato e quantidade de rodas que estariam disponíveis.

Como sabíamos que possivelmente poderíamos colocar diversos tipos de hardware na plataforma, definimos que, independente do formato, as dimensões deveriam ser razoavelmente grandes. Também definimos o uso de 4 rodas para uma maior estabilidade da plataforma, visto que esta deveria estar em constante movimento e, para isso, seria necessário que tivesse grande estabilidade.

Após diversas pesquisas, principalmente *online*, encontramos uma plataforma que se adequasse as nossas necessidades. Porém, está plataforma só estava disponível no Brasil caso não incluísse o *encoder*, material que julgávamos necessário. Assim, voltamos às pesquisas e encontramos uma loja *online* americana [7] que tinha um produto adequado para nosso projeto e que continha o *encoder*.

O chassi escolhido era composto por duas placas transparentes de acrílico com diversos furos para fixação de componentes, quatro motores DC de pequeno porte, quatro rodas de plástico com pneus de borracha, seis espaçadores para a montagem, oito peças de acrílico para a fixação dos motores, quatro *encoders* para serem acoplados nos motores, uma caixa para quatro pilhas AA e diversos parafusos e porcas. Montado, teria dimensões de aproximadamente 28 cm de comprimento, 14,5 cm de largura e 10 cm de altura, além de um peso de 800 g. Abaixo, temos uma figura do chassi montado:

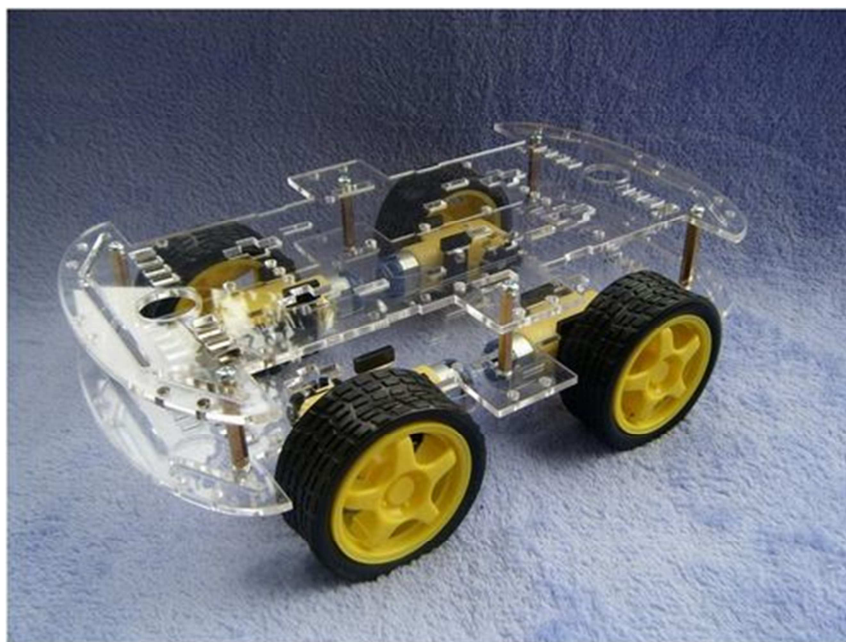
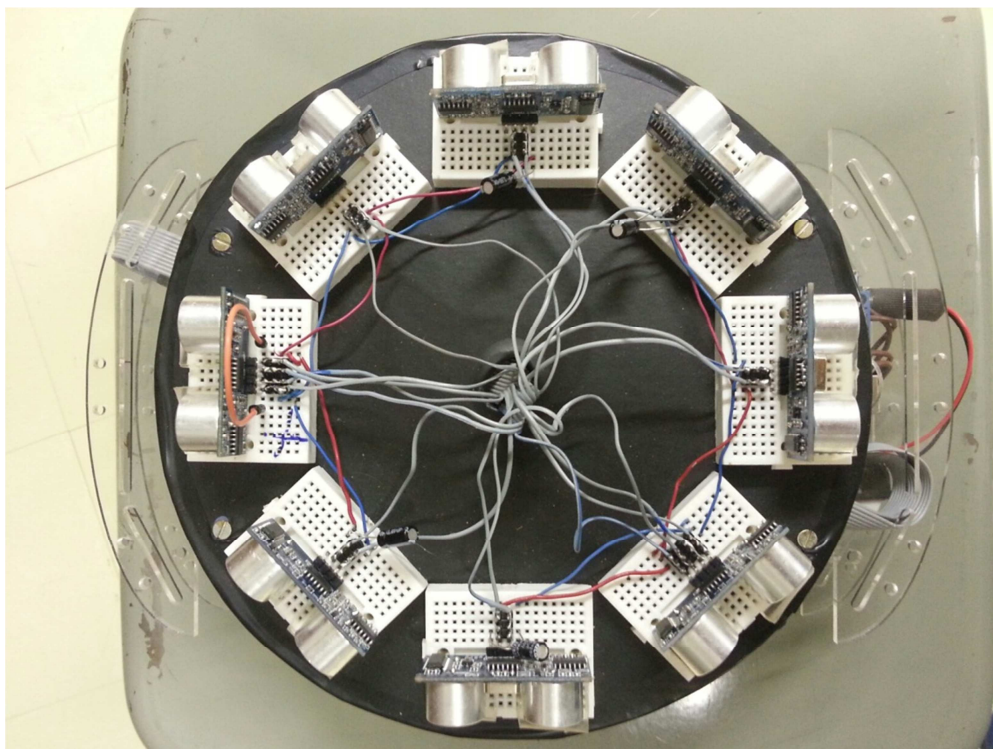


Figura 20 - Plataforma mecânica montada

Logo após a montagem, já detectamos pequenos problemas como a falta de estabilidade dos motores no chassi, visto que as peças de fixação eram maiores do que deveriam ser. Para solucionar este problema, utilizamos fita dupla face transparente de forte fixação para colarmos os motores à placa de acrílico de forma permanente. Também utilizamos esta fita para a fixação da caixa das pilhas AA.

Outro problema encontrado foi que, mesmo com a presença de diversos furos nas placas de acrílico, nenhum deles era ajustado aos furos do Arduino Mega. Para solucionarmos este novo problema, fomos à oficina do SG-09 onde obtivemos ajuda dos técnicos da mecânica para fazermos novos furos na placa.

Solucionados os problemas iniciais de montagem, tivemos que analisar a melhor forma de disposição dos oito sensores na plataforma. A melhor forma seria um octógono regular de lado 4,5 cm (largura do sensor), porém para isso era preciso uma área equivalente a de uma circunferência de raio 10 cm, ou seja, nossa placa de acrílico de 14,5 cm de largura não comportaria tal disposição dos sensores. Resolvemos fazer uma nova placa mais simples que servisse apenas para a fixação dos sensores na placa de acrílico e, para isso, utilizamos uma capa de caderno pela facilidade de obtenção e por ter um material firme. O arranjo final dos sensores é mostrado na figura abaixo:



**Figura 21 - Disposição dos sensores na placa superior**



#### 4.2- Escolha dos tipos de baterias e das tensões:

Após a montagem da plataforma, o próximo passo foi o de definição da forma de alimentação do projeto. Até esse momento, apenas usávamos a alimentação de 5 V disponibilizada pela porta serial USB do computador.

Os primeiros testes, nos quais os programas eram separados, foram realizados com dois Arduinos distintos sendo um para a parte inferior da plataforma, onde se encontram os motores e motor *shield*, e outro para a parte superior da plataforma, onde se encontram os sensores, como mostra a figura abaixo:

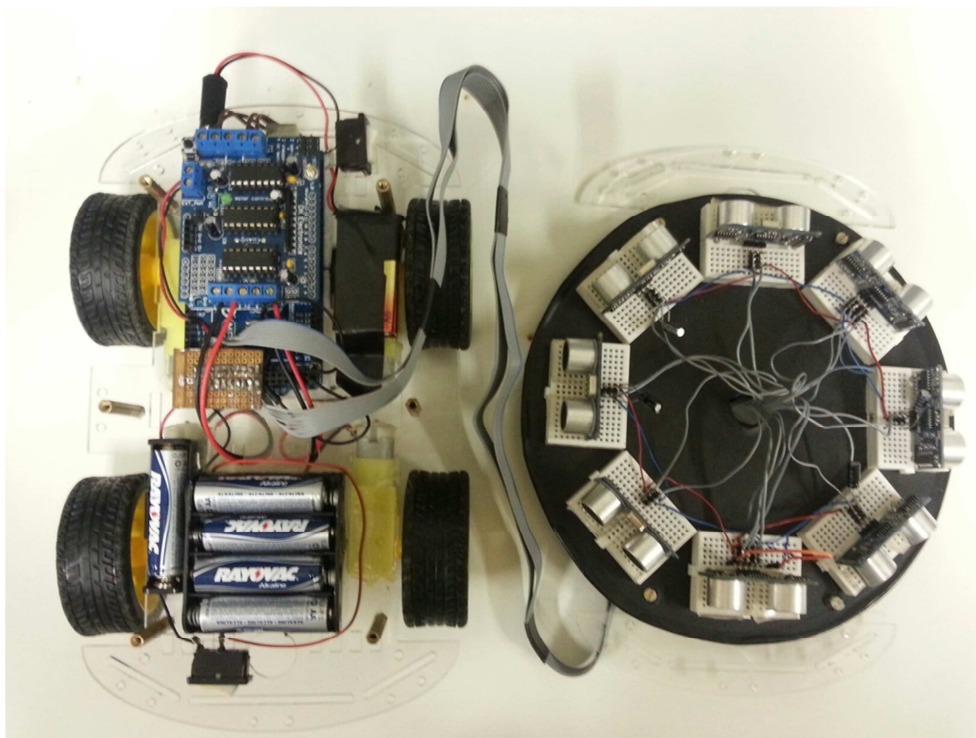


Figura 22 - Plataformas superior e inferior separadas

Já nos primeiros testes com a plataforma inferior, encontramos problemas para a alimentação dos motores visto que a tensão que o Arduino fornece para o motor *shield* é de 5 V, ou seja, a tensão entregue aos motores era menor que 5 V, pois existe uma queda de tensão entre a entrada e saída do *shield*. Outro problema encontrado foi que o Arduino não obedecia ao programa de forma perfeita quando era alimentado por uma fonte conectada à rede externa. Esse problema não existia quando conectado ao computador.

Após diversos testes e estudo aprofundado do *datasheet* do motor *shield*, foi decidido o uso de fontes de alimentação independentes para o Arduino e o motor *shield*, visto que o próprio fabricante da placa indicava que a maioria dos '*weird motor problems*' (problemas estranhos do motor) seriam devido à alimentação conjunta. [5]

Depois de decidido o uso da alimentação separada, o primeiro passo foi a remoção do *jumper* do motor *shield* e, então, foi preciso decidir qual tensão deveria ser utilizada para cada caso. Para a tensão do Arduino, foi decidido o uso de uma bateria de 9 V devido a sua facilidade de acesso e por estar dentro da sua faixa recomendada de alimentação (de 7 a 12 V). Para o *shield*, realizamos testes para achar a relação entre tensão de entrada e tensão de saída e encontramos a seguinte relação:

$$V_{out} = 0,8V_{in} \quad (14)$$

Com essa relação e com as informações da Tabela 1, foi definida uma tensão de 7,5 V para a alimentação do *shield*, ou seja, com a redução, os motores seriam alimentados com a tensão máxima de 6 V. Para o fornecimento da tensão desejada, foi escolhido o uso de 5 pilhas AA pela facilidade de obtenção e para o aproveitamento da caixa de pilhas já existente no chassi.

#### 4.3- Ensaios com a plataforma unificada:

Com os testes das plataformas inferior e superior feitos, montamos a plataforma unificada da forma final do projeto e acrescentamos 2 *switches* para o controle das alimentações do Arduino e do *shield*. Após toda a montagem, a plataforma ficou como mostra a Figura 23 abaixo:

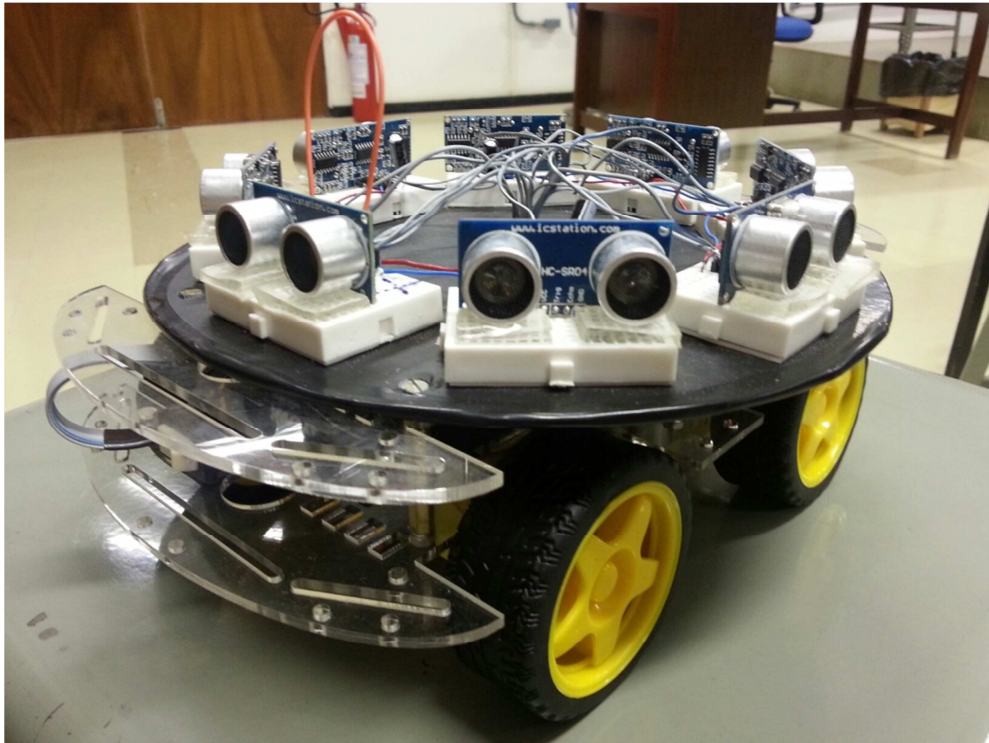


Figura 23 - Plataforma final montada



Depois de juntarmos a parte mecânica, precisávamos integrar os dois programas anteriores de forma que se formasse um único programa final do projeto. Para integração dos programas, houve uma alteração no cálculo do ângulo de fuga, já que cada programa utilizava uma margem diferente para os possíveis valores do parâmetro. No programa dos sensores, os valores possíveis eram de  $\pi$  a  $3\pi$ , enquanto no programa dos motores era de  $-\pi$  a  $\pi$ . Dessa forma, definimos que o ângulo de fuga teria valores entre  $-\pi$  e  $\pi$ , ou seja, seguiria o modelo mostrado na Figura 16. Assim, a Equação 9 foi alterada para a equação abaixo:

$$\varphi_{fuga} = \varphi - \pi \quad (15)$$

Ajustados os parâmetros, começamos a fazer testes mais próximos da realidade, ou seja, neste ponto iniciamos os ensaios com a plataforma unificada em movimento. Dessa forma conseguimos identificar problemas encontrados em um cenário real.

O primeiro problema encontrado foi o fato de diversas vezes os sensores conseguirem identificar um determinado obstáculo, porém não conseguirem fazer o cálculo para encontrar os parâmetros  $d_T$  e  $\varphi_{fuga}$ , ou seja, os motores não eram acionados uma vez que dependem desses parâmetros para o seu funcionamento. Depois de estudos para identificar a origem do problema, chegamos à conclusão de que era devido ao fato do algoritmo, para o cálculo da distância do objeto e do ângulo de fuga, considerar o obstáculo como um ponto material, ou seja, um objeto sem dimensões. Assim, nesses casos, cada sensor identificava uma parte do obstáculo e não conseguia formar um triângulo como mostrado na Figura 13. A figura abaixo mostra um exemplo aproximado do ocorrido:

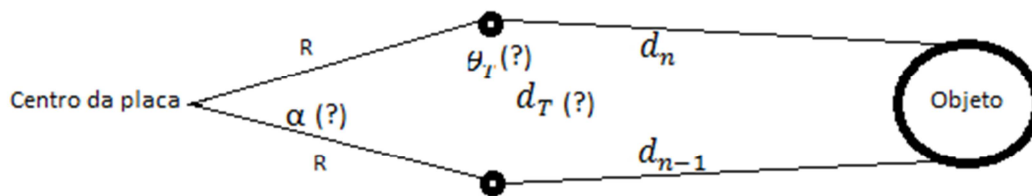


Figura 24 - Esboço do caso em que o objeto não é um ponto material

Pelo algoritmo temos que, se estivermos neste caso, o ângulo  $\theta_T$  não conseguirá ser calculado visto que o ângulo  $\theta_1$  é calculado pela lei dos cossenos, ou seja, é preciso a formação de um triângulo para que estes ângulos sejam calculados. Como todos os outros parâmetros dependem desses dois, o cálculo é interrompido e o programa não consegue chegar a um resultado.

Para solucionarmos este tipo de deformidade, ajustamos o programa para, após realizar as medições dos 8 sensores e considerar a possibilidade deste caso, tomar como verdadeira a distância ao objeto de menor distância encontrada pelos sensores. Assim, o ângulo de fuga será dado como o ângulo diametralmente oposto ao sensor com menor distância fazendo com que o robô se mova na direção aproximadamente oposta. Dessa forma, nesses casos temos:

$$d_T = d_{n-1} \quad \text{ou} \quad d_T = d_n \quad (16)$$

Implementada essa mudança percebemos que, dependendo do ambiente e de quão distante o obstáculo estiver, o sensor pode retornar valores exorbitantes como distância do obstáculo. Para minimizar esse erro definimos que, para distâncias maiores que 400 cm, o programa irá definir como sendo igual a 400 cm.

Outra razão para essa alteração é o fato do sensor perder a acurácia conforme o obstáculo está mais distante. Dessa forma, quando passamos do limite estabelecido pelo fabricante, existe uma diferença entre os sensores nesse quesito, pois cada um tem um limite real para a detecção dos obstáculos. Assim, com essa modificação, tornamos os 8 sensores o mais iguais possível.

Além disso, apenas um valor exorbitante já é capaz de uma grande alteração na média móvel dificultando o processamento da plataforma e fazendo com que seu tempo de resposta seja cada vez maior. Para obter um tempo de resposta ainda menor, determinamos que o vetor média móvel devesse ter 3 posições no lugar das 8 posições definidas anteriormente.

Ainda, em se tratando do tempo de resposta, mudamos o tempo das interrupções do *Timer* de 100 ms para 50 ms. Assim, alteramos o tempo de leitura dos 8 sensores de 800 ms para 400 ms. Como o vetor média móvel anterior era de 8 posições, o tempo para a alteração de todas as posições do vetor era de 6,4 s. Enquanto com as alterações no tamanho do vetor e no tempo das interrupções, o tempo atual para a alteração de todas as posições do vetor é de 1,2 s, ou seja, 25% do tempo anterior.

Com essa diminuição significativa no tempo da atualização do vetor média móvel, foi possível uma diminuição no tempo de acionamento da *flag* responsável por indicar a passagem de tempo para que o *loop* principal consiga chamar a rotina do cálculo do ângulo de fuga. O tempo anterior era de 2 s, ou seja, o *loop* principal chamava a rotina de cálculo com cerca de 31% de atualização do vetor média móvel. Alteramos esse valor para 1,2 s, fazendo com que o *loop* principal chamasse a rotina de cálculo com a atualização completa do novo vetor. Dessa forma, o cálculo será feito de forma o mais próximo possível da realidade atual da plataforma possível fazendo com que o tempo de resposta seja o menor possível.

Outra falha encontrada foi o fato do motor 3 não estar funcionando da forma adequada desde a integração dos programas. Após os estudos feitos para a possível causa de tal defeito, vimos uma disfunção no uso do *Timer 4* em conjunto com o motor *shield*. Como mostra a Tabela 2, o *timer 4* utiliza os pinos 6, 7 e 8. Enquanto o motor *shield*, quando acoplado na parte superior do Arduino Mega, utiliza os pinos de 0 a 13. Após encontrarmos esses pinos em comum, verificamos as funções de cada pino no *sketch* do motor *shield*.

Pelo *sketch* da Figura 25, vemos que os pinos 2 e 13 são usados para controlar os motores 1 e 2, enquanto os pinos 5 e 6 são usados para controlar os motores 3 e 4. Mais especificamente, o pino 6 controla o motor 3 [10]. Dessa forma, não poderíamos utilizar o *timer 4* e o motor *shield* (para controlar o motor 3) ao mesmo tempo.

Pela mesma razão não poderíamos utilizar o *timer 3*, pois pela Figura 25 vemos que alteraria o funcionamento do motor 4. Assim, alteramos o uso do *timer 4* para o *timer 5* que não possui pinos em conjunto com a placa acoplada. Como o *timer 5* também é um contador de 16 bits, não houve alteração no funcionamento do programa.

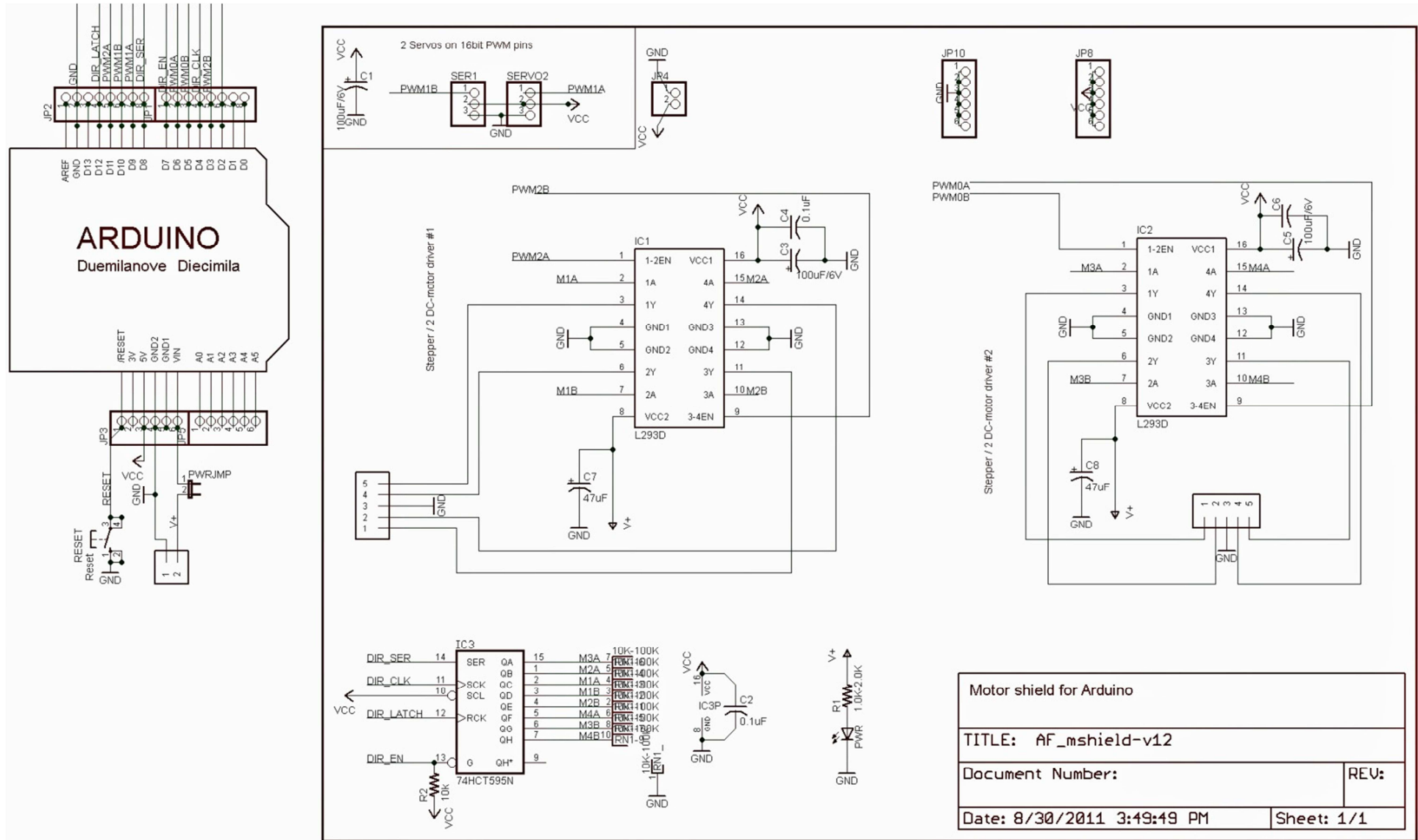


Figura 25 - Sketch do Motor Driver Shield. Fonte: [10]

Ainda sobre a contagem de tempo, tivemos problema com o *delay()* que estava dentro dos casos de curvas. Este estava travando o programa a cada novo acionamento do motor e contagem de tempo, não seguindo o tempo  $t_{giro}$  e tornando a curva mais demorada que o previsto. Esta falha era devido ao fato de tanto o *delay()* quanto o *delayMicroseconds()* utilizarem o *Timer 0*, assim, durante a contagem de tempo do  $t_{giro}$ , o primeiro era interrompido para que o segundo fosse realizado, alterando o valor armazenado.

Para solucionarmos este defeito, utilizamos a função *millis()* juntamente com outras duas variáveis para contar o tempo. Mesmo utilizando o *Timer 0*, esta função não interrompe o programa, assim, não tivemos problemas na sua aplicação.

Durante os testes finais, encontramos uma nova falha. Os tempos de curva da plataforma não eram consistentes, pois diversas vezes executava uma curva muito maior que a esperada. Após estudo do código, encontramos o erro. A função *pulseIn()*, responsável pela contagem de tempo de ida e volta do pulso lançado pelos sensores, estava levando um tempo consideravelmente grande para ser executada o que atrasava todo o restante do programa.

Para isso, colocamos uma nova *flag*. Quando esta *flag* estiver ativada, ocorrerá a leitura do sensor, cálculo e atualização do vetor média móvel. Então programamos os casos de curva para desativarem a *flag* durante o período de curva e para ativa-la novamente após isso. Dessa forma, a função *pulseIn()* não seria ativada durante as funções de curva e, assim, a curva seria feita com o tempo determinado.

Para finalizarmos os ensaios, mudamos o valor da distância limite para que o robô reagisse a certo obstáculo. Anteriormente este valor era de 250 cm, ou seja, a plataforma somente reagia para obstáculo com até 2,5 m de distância. Porém, mesmo depois de todos os testes feitos, o sistema ainda não tinha entrado em um caso com uma distância maior do que a limite, ou seja, a plataforma nunca tinha ficado parada por ter obstáculos distantes demais.

Dessa forma, decidimos alterar o limite para 100 cm, assim, se tivermos um objeto mais distante que 1 m, o robô permanecerá parado. Esse limite foi definido arbitrariamente, somente levando em consideração as distâncias reais que a plataforma estava sendo submetida. Tal mudança foi feita principalmente para podermos testar todos os casos possíveis do programa.

Como nosso projeto tinha como objetivo uma mudança de velocidade de acordo com uma mudança da distância, determinamos que, para uma melhor visualização da alteração da velocidade, a Equação 10 e o gráfico da Figura 15 fossem modificados para a equação e gráfico a seguir:

$$Velocidade = -1,85 \cdot d_T + 273,5 \quad (17)$$

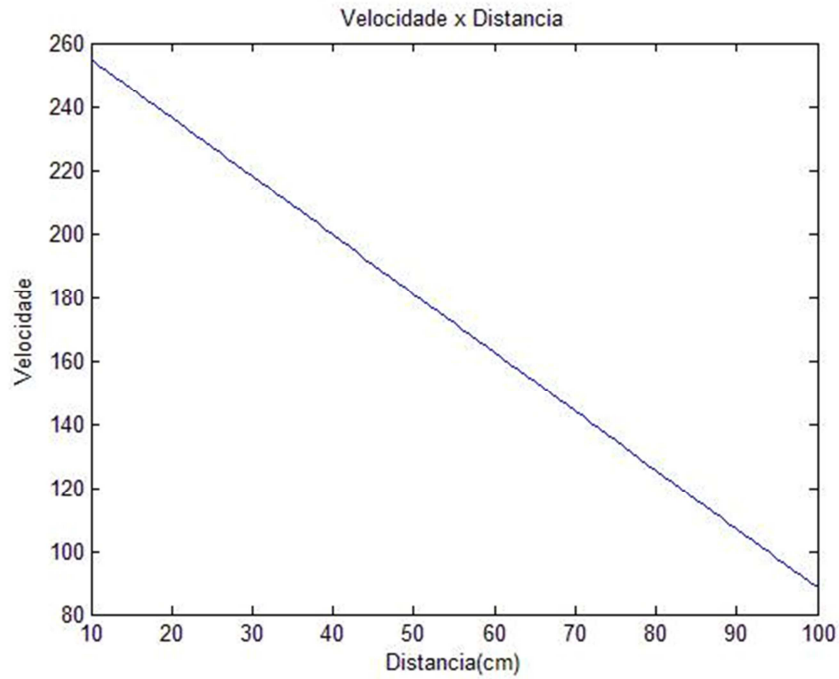


Figura 26 - Novo gráfico de Velocidade vs Distância

Nosso cálculo da distância do objeto é feito a partir do centro da placa dos sensores, assim, os valores sempre serão contados a partir de 10 cm, visto que este é o raio da placa, ou seja, distância entre o centro e os sensores. Dessa forma, a velocidade somente será calculada para  $10 \leq d_T \leq 100$ , então temos valores possíveis de velocidade dentro do intervalo  $88,5 \leq Velocidade \leq 255$ . Anteriormente, para a nova distância limite, esse intervalo seria  $191,25 \leq Velocidade \leq 248,63$ , intervalo consideravelmente menor que dificilmente seria possível de se notar com uma visão macro.

## RESULTADOS

Com a conclusão dos ensaios, chegamos à nossa plataforma final. Um robô com 8 sensores de ultrassom, Arduino, motor *driver shield*, 4 motores e toda uma estrutura mecânica formada por 2 placas de acrílico, 4 rodas, *case* para 5 pilhas AA, 8 mini *protoboards*, 2 *switches*, adaptador de bateria de 9 V para Arduino e diversos cabos, parafusos e porcas.

Esta plataforma, em funcionamento, é capaz de identificar e se desviar de obstáculos, mudando de velocidade dependendo da distância do objeto. Os testes feitos no Laboratório 2 do SG-11 da Universidade de Brasília, mostram uma plataforma que consegue identificar o obstáculo mais próximo e, em seguida, é capaz de 'fugir' dele. Porém, se a plataforma estiver 'fugindo' para certa direção e nesse percurso existir um obstáculo, ocorrerá um choque. O funcionamento do projeto pode ser visualizado em um vídeo disponível [11] no site *Youtube*. A estrutura final do projeto pode ser vista na figura a seguir:

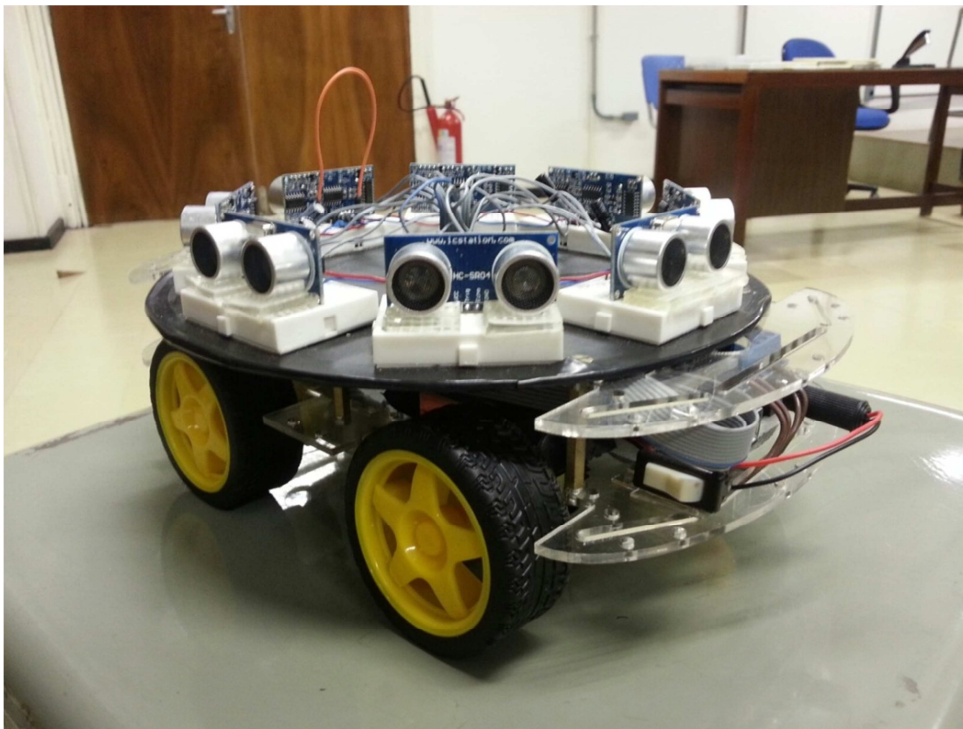


Figura 27 - Estrutura final do projeto

A plataforma final conta com certos aspectos que precisam ser aperfeiçoados para uma melhor aplicação futura. As principais melhorias a serem feitas são: aperfeiçoamento mecânico, melhora do sistema de alimentação e auto calibração da velocidade angular dependendo do tipo de superfície.

O aperfeiçoamento mecânico seria devido ao fato dos motores, ainda que colados na plataforma, não estarem perfeitamente firmes o que pode interferir nas direções em que o robô pretende andar. Com uma visão macro, vemos que a plataforma consegue andar em linha reta, porém com uma melhor fixação dos motores, e por consequência das rodas, a acurácia da trajetória seria o

mais ideal possível. Outro aspecto mecânico importante que se deve aprimorar é a troca do cabo que interliga os sensores e o Arduino. Este cabo está com certos maus contatos e tem um tamanho maior do que o necessário o que pode interferir no sinal recebido e, por consequência, interferir na detecção dos obstáculos.

A melhora no sistema de alimentação é devido ao fato das fontes atuais estarem se esgotando rapidamente, especialmente as pilhas AA. Este fato é devido à baixa corrente que elas conseguem fornecer comparado ao alto consumo do sistema dos motores. Embora a bateria de 9 V esteja com maior durabilidade, esta também está abaixo do esperado. Assim, recomenda-se um novo estudo de possíveis formas de alimentação como baterias de filmadora, baterias de aeromodelos ou pilhas recarregáveis. Além do custo, um dos problemas das pilhas recarregáveis é devido à tensão delas serem de 1,2 V, enquanto a pilha normal é de 1,5 V. Dessa forma, para o uso de pilhas recarregáveis, será necessário aumentar o banco de pilhas de 5 para 6 e, ainda assim, o fornecimento será menor do que o atual (que é de 7,5 V).

Uma possível auto calibração da plataforma irá torna-la mais confiável do ponto de vista da fuga dos obstáculos. Este é o principal problema atual, já que os outros dois citados tem menor interferência na eficiência do projeto. Atualmente o projeto identifica o obstáculo, calcula os parâmetros necessários e, caso precise, faz uma curva com um  $t_{giro}$  calculado. Porém esse tempo calculado para que o programa esteja na função *direita()* ou *esquerda()* deveria levar em consideração a superfície pela qual a plataforma deverá andar e a tensão das baterias, pois o robô poderá rodar mais ou menos que o esperado dependendo da aderência que as rodas terão com o chão e da tensão de alimentação dos motores. Assim, na realidade a Equação 13 deveria ser:

$$t_{giro} = A \cdot |\varphi_{fuga}| \quad (188)$$

Onde A é uma constante para cada tipo de superfície, considerando a tensão aplicada. Como não temos como conhecer a constante A antes de um teste, o sistema deveria ser auto calibrável e capaz de encontrar a constante de forma autônoma.

A forma encontrada para minimizar este erro foi o uso de um  $t_{giro}$  constante. O objetivo seria que a plataforma fizesse certa curva por um tempo determinado e parasse, aguardando nova leitura. Dessa forma a plataforma faria pequenos giros até que o obstáculo estivesse alinhado com a de trás e então ‘fugiria’ andando para frente.

Utilizando o parâmetro da Equação 12, determinamos um  $t_{giro}$  que fosse capaz de fazer uma curva de aproximadamente 40°. Assim, definimos:

$$t_{giro} = 180 \text{ ms} \leftrightarrow 0,6283 \text{ rad} = 36^\circ \quad (19)$$

Porém, o programa só iria entrar na função *frente()* quando o ângulo de fuga estivesse no intervalo  $-0,15 \leq \varphi_{fuga} \leq 0,15$ . Decidimos então alterar essa margem para  $-0,8 \leq \varphi_{fuga} \leq 0,8$ , assim o programa entraria na função *frente()* caso o obstáculo esteja do sensor 4 ao 6. A figura a seguir nos mostra a nova angulação para que a função *frente()* seja acionada:



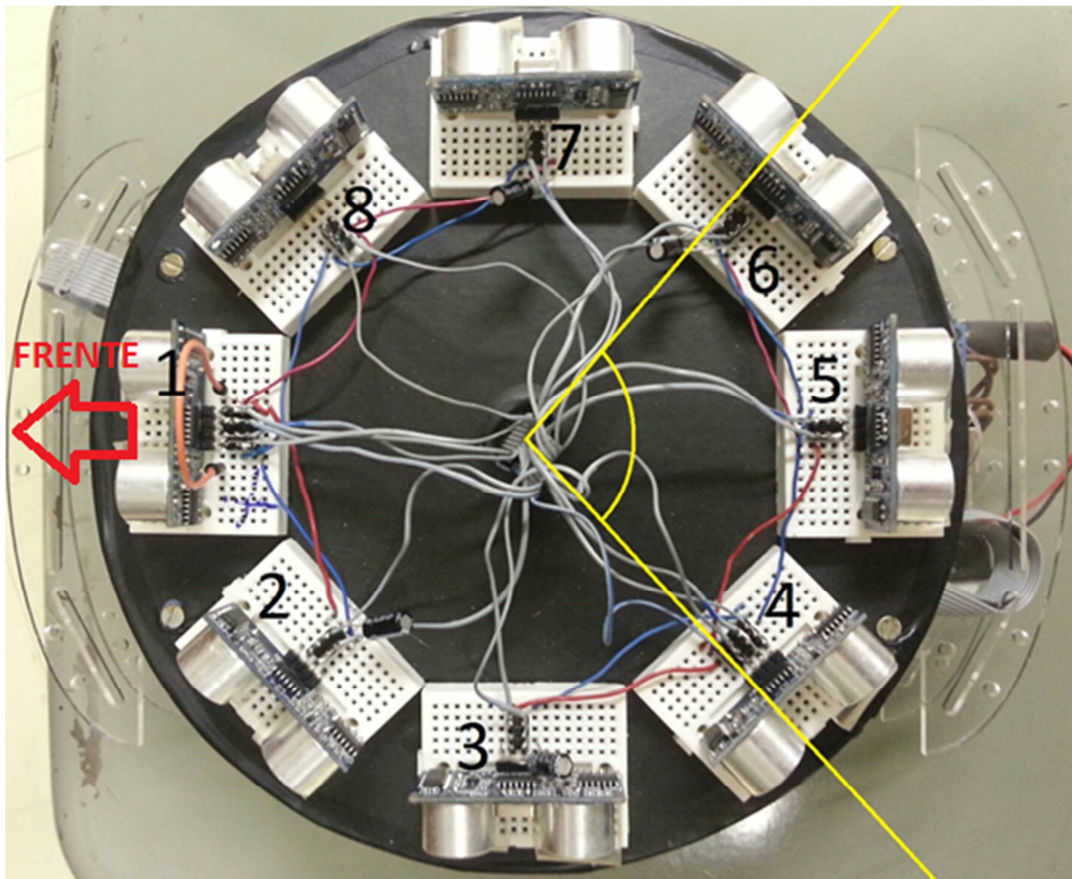


Figura 28 - Intervalo do ângulo de fuga para que a função *frente()* seja acionada

O novo programa se tornou mais preciso, porém ficou mais lento fazendo que seu tempo de resposta fosse muito inferior ao programa anterior. Por isso decidimos manter os dois programas em paralelo, visando uma melhora futura em algum deles ou em ambos. O código com  $t_{giro}$  calculado era mais dinâmico, porém menos acurado, enquanto o novo código com  $t_{giro}$  constante é mais lento, porém mais apurado. O funcionamento do projeto com o novo código pode ser visualizado no site *Youtube*, no *link* disponível em [12].

A plataforma final, independente de qual dos dois programas esteja sendo executado, conta com um tempo de 400 ms para a leitura dos 8 sensores, 1,2 s para a atualização do vetor média móvel e uma *flag* que indica a passagem de 1,2 s para o *loop* principal chamar a rotina de cálculo, fazendo com que os cálculos sejam feitos somente com o vetor média móvel atualizado.

Os algoritmos e códigos dos programas finais com  $t_{giro}$  calculado e constante são mostrados nos Apêndices A e B, respectivamente. Os fluxogramas das Figuras 29 e 30 mostram o funcionamento final das interrupções e do *loop* principal dos códigos, respectivamente. Na Figura 30 temos o funcionamento dos dois tipos de códigos diferenciados pelos asteriscos e parênteses, sendo que o que está entre parênteses é relativo ao código com o  $t_{giro}$  constante.



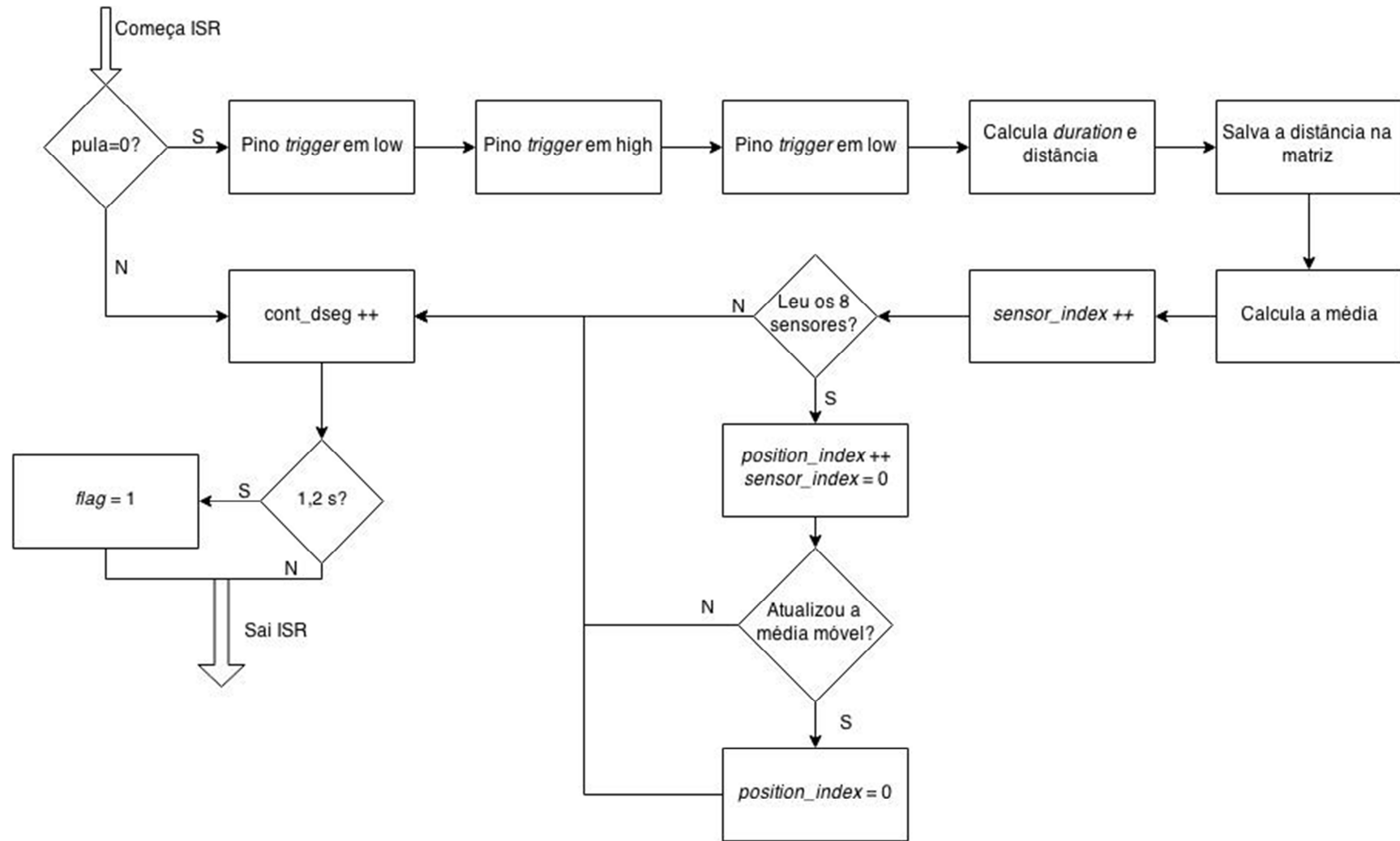


Figura 29 - Fluxograma final das interrupções

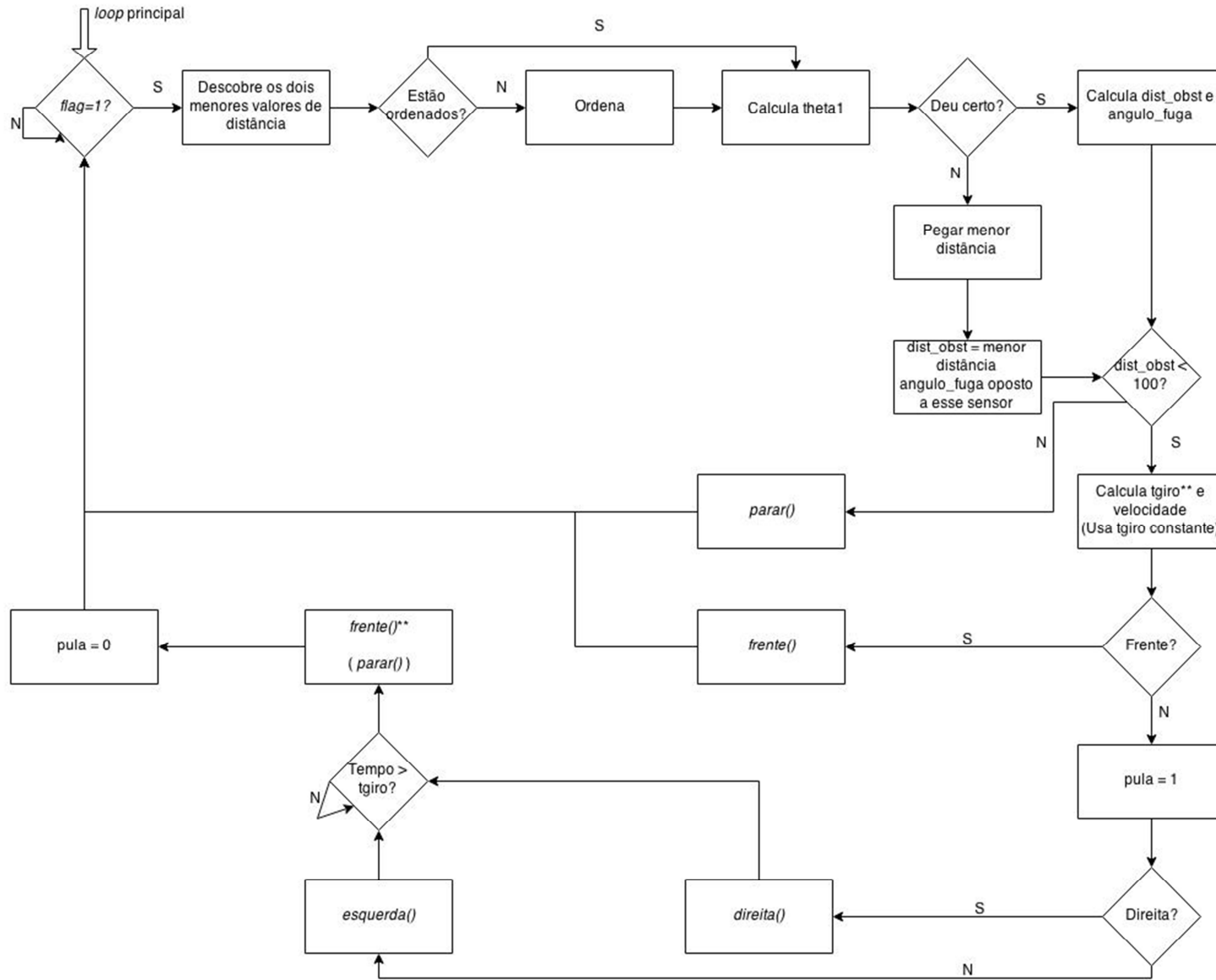


Figura 30 - Fluxograma final do *loop principal*

## CONCLUSÃO

Com o crescente mercado de robôs e plataformas móveis, vimos a oportunidade de aumentar os estudos voltados para a área. Como uma plataforma pode ser um conjunto de módulos independentes, surgiu o desafio de elaborar um desses módulos que fosse capaz de ser aplicado posteriormente em diversos tipos de robôs maiores. Lembrando que um mesmo módulo pode ser adequado em diferentes tipos de plataformas e com diversos tipos de função.

O sistema proposto teve como objetivo projetar e programar uma plataforma móvel autônoma capaz de se desviar de objetos. Para isso, foi proposto o uso de sensores de ultrassom para a localização do obstáculo, enviando sinais para o processador que calcula a sua distância e ângulo de fuga e manda sinais para o acionador dos motores.

Foram utilizados 8 sensores de ultrassom dispostos em um octógono regular inscrito em uma circunferência de raio 10 cm. Tais sensores foram dispostos em 8 mini *protoboards* de 4,5 cm de largura, ou seja, o octógono tem 4,5 cm de lado. Tal conjunto formava uma placa independente na parte superior do projeto. A parte inferior consiste em uma placa de Arduino com um motor *driver shield* acoplado, estojo para 5 pilhas AA e 4 conjuntos de motores-rodas.

A princípio, foram elaborados programas separados para o funcionamento dos sensores e dos motores. O programa dos sensores era capaz de localizar certo obstáculo, calcular sua distância e o ângulo de fuga que a plataforma deveria seguir. O programa dos motores dividia os possíveis valores do ângulo de fuga (que até então eram valores teóricos pré-determinados no próprio programa) em 3 casos: frente, direita e esquerda. Este programa também fazia o cálculo do tempo em que a plataforma deveria fazer certa curva e a velocidade em que os motores deveriam funcionar dependendo da distância do obstáculo.

Após testes independentes, houve a integração dos códigos. Junto com a integração, vieram diversos tipos de falhas. Houve falhas nas contagens de tempo, nos intervalos adotados para parâmetros, nos cálculos dos parâmetros, etc. Depois da resolução das falhas, fizemos uma mudança na distância limite a qual a plataforma deveria reagir e da velocidade em que deveria andar.

Cada sensor é lido a cada 50 ms, assim, são necessários 400 ms para que todos os 8 sensores sejam lidos. Como o tamanho do nosso vetor média móvel é de 3 posições, o tempo para a sua total atualização é de 1,2 s. Considerando este tempo, uma *flag* é acionada, após a atualização total do vetor, para indicar o instante em que o programa chama a rotina de cálculo e toma uma decisão de andar ou não e em qual direção.

Também foi feito um programa em paralelo com a intenção de minimizar o principal defeito atual da plataforma que é a dificuldade para determinar a velocidade angular do projeto dependendo da tensão aplicada aos motores e da superfície sobre a qual ele pretende se mover. Tal programa fazia com que o robô executasse pequenos giros sobre seu eixo até que o obstáculo estivesse na parte de trás da plataforma. O problema desse novo código foi o tempo de resposta, pois o robô gasta muito tempo fazendo tais giros.

Após todos os ajustes feitos até o momento, pode-se afirmar que o projeto atingiu os objetivos propostos. Foi elaborado um robô previamente programado que é capaz de localizar e se desviar de obstáculos de forma autônoma, conforme o previsto.

Da forma que está apresentado atualmente, para uma melhor implementação, o projeto precisa de certas melhorias. Como uma melhor fonte de alimentação, visto que a bateria de 9 V e as pilhas AA estão sendo consumidas rapidamente. Para a resolução deste problema, sugere-se a pesquisa de novos tipos de baterias como baterias de filmadoras, aeromodelos ou pilhas recarregáveis.

Outro tipo de melhoramento seria a implementação de um sistema ou rotina capaz de realizar uma auto calibração na plataforma. Para isso, deverão ser considerados os diversos tipos de superfície em que o robô poderia estar sujeito. Uma sugestão seria o uso de botões externos para a medição da velocidade angular na superfície desejada, porém seria necessário o uso de um operador externo o que tornaria a plataforma menos autônoma. Assim, é preciso um estudo apropriado para o melhor 'custo-benefício' deste caso.

É possível, também, realizar melhorias na parte mecânica do projeto, como diminuição do cabo que conecta os sensores ao Arduino e melhor fixação dos motores e rodas. Além de um aprimoramento do código para evitar que a plataforma se choque com outros obstáculos que podem aparecer no intervalo entre as leituras.

Feitas tais mudanças, o sistema poderá ser ajustado para o uso em sistemas de segurança, sistemas de captação de imagens, sistemas biomecânicos, entre outros. Assim, com o devido estudo e ajustes, temos um módulo de fácil implementação em diversos outros sistemas.

# REFERÊNCIAS BIBLIOGRÁFICAS

- [1] J. MATARIC, Maja. Introdução à Robótica. São Paulo: Blucher, 2014. 368p.
- [2] GROOVER, Milkell P. et al. Robótica: tecnologia e programação. São Paulo: McGraw-Hill, 1989. 401 p.
- [3] MCROBERTS, Michael. Arduino Básico. São Paulo: Novatec, 2011. 456p.
- [4] Arduino – Home – Disponível em [www.arduino.cc](http://www.arduino.cc) – Última data de acesso: 13/11/2014
- [5] Adafruit Motor *Shield Datasheet* – Disponível em <https://learn.adafruit.com/adafruit-motor-shield> - Última data de acesso: 09/11/2014
- [6] CARVALHO, Geraldo de. Máquinas Elétricas: Teoria e Ensaio. 2. ed. São Paulo: Érica, 2007. 264 p.
- [7] 4WD smart car chassis kits arduino robot chassis with speed encoder – Disponível em <http://www.tohobby.com/4wd-smart-car-chassis-kits-arduino-robot-chassis-with-speed-encoder.html> - Última data de acesso: 20/08/2014
- [8] Cartilha de programação em C - Arduino – Disponível em [http://ordemnatural.com.br/pdf-files/CartilhadoArduino\\_ed1.pdf](http://ordemnatural.com.br/pdf-files/CartilhadoArduino_ed1.pdf) - Última data de acesso: 13/11/2014
- [9] Atmel Microcontrolador *Datasheet* – Disponível em <http://www.atmel.com/images/doc2549.pdf> - Última data de acesso: 03/11/2014
- [10] Multi-Motor *Shield Schematic* – Disponível em <http://yourduino.com/docs/Multi-MotorShieldSchematic.jpg> - Última data de acesso: 13/11/2014
- [11] Plataforma móvel com detecção de obstáculos – Tgiro Calculado – Youtube – Disponível em <https://www.youtube.com/watch?v=eZGnRAUljgU> – Última data de acesso: 17/12/2014
- [12] Plataforma móvel com detecção de obstáculos - Tgiro Constante – Youtube – Disponível em <https://www.youtube.com/watch?v=wYFtz8Q1w> – Última data de acesso: 17/12/2014

# APÊNDICES

## APÊNDICE A:

Algoritmo do programa final com  $t_{giro}$  calculado:

1. O programa dos sensores começa com a inicialização do Timer 5 para gerar interrupções a cada 50 ms. Sendo que a cada interrupção o sistema realiza a leitura de um sensor, atualiza o vetor de distâncias e o vetor média móvel de cada sensor.
2. Caso a *flag* 'pula' esteja ativada, ou seja, igual a zero:
  - O pino de *trigger* do sensor encontra-se em nível baixo e em seguida gera um pulso em nível alto por 10  $\mu$ s, retornando ao nível baixo após isso. O sinal de recebimento do pulso retorna pelo pino de *echo*. A CPU recebe esse pulso e o tempo transcorrido para tal é colocado na variável *duration* e, assim, é calculada a distância do obstáculo e este valor é colocado em uma posição no vetor daquele sensor.
  - O cálculo da média móvel é feito com um *loop for* que soma todos os valores do vetor do sensor e então divide 3, o tamanho do vetor. O resultado é carregado no vetor de médias móveis.
  - O *sensor\_index* é incrementado para que o próximo sensor seja atualizado no próximo ciclo. Quando o indexador dos sensores atinge o número de sensores, o indexador retorna para o 0 e o *position\_index* é incrementado mudando a posição no vetor das distâncias. Ao se atingir *position\_index* igual a 8, o indexador é retornado para 0 novamente.
3. Finalizando a rotina de interrupção, a variável de contagem de décimos de segundo é incrementada.
4. A cada 1,2 segundos uma *flag* é acionada indicando ao *loop* principal da passagem do tempo. Então, o *loop* chama a rotina de cálculo do ângulo de fuga.
5. A rotina ordena as médias móveis da menor para a maior, salvando as duas menores medidas e seus sensores correspondentes. Estes valores salvos são ordenados por ordem decrescente do número do sensor e seguem o algoritmo para o cálculo da distância do objeto ( $d_T$ ) e do ângulo de fuga ( $\varphi_{fuga}$ ) explicado anteriormente.
6. Caso o programa não consiga calcular os parâmetros, ele define o ângulo de fuga como o ângulo diametralmente oposto ao sensor de menor distância e esta distância será tomada como distância do objeto.
7. Uma vez definidos os parâmetros  $d_T$  e  $\varphi_{fuga}$ , o programa dos motores calcula a velocidade da plataforma e o  $t_{giro}$ . Sendo que, para isso,  $d_T$  precisa ser menor que 100, ou seja, a distância do objeto deve ser menor que 1 metro. Caso contrário, o programa entra na função *parar()*, assim, a plataforma ficará parada.
8. Caso  $d_T \leq 100$ , o programa irá analisar o  $\varphi_{fuga}$  para determinar qual direção a plataforma deverá seguir. Teremos três possíveis casos:
  - Caso  $-0,15 \leq \varphi_{fuga} \leq 0,15$ , o programa executa a função *frente()* na qual os motores andam na velocidade calculada pela Equação 17;
  - Caso  $-3,1415 \leq \varphi_{fuga} < -0,15$  ou  $3 \leq \varphi_{fuga} \leq 3,1415$ , o programa desativa a *flag* 'pula', executa a função *direita()* durante o tempo  $t_{giro}$ , executa a função

*frente()* e, então, ativa a *flag* novamente. Durante o tempo em que os motores obedecem a função *direita()*, a velocidade será a velocidade máxima, ou seja, 255. Então, quando os motores passam a obedecer a função *frente()*, a velocidade será a calculada pela Equação 17;

- Caso não esteja em nenhum dos casos anteriores, ou seja,  $0,15 < \varphi_{fuga} < 3$ , o programa desativa a *flag* 'pula', executa a função *esquerda()* durante o tempo  $t_{giro}$ , executa a função *frente()* e, então, ativa a *flag* novamente. Da mesma forma, durante o tempo em que os motores obedecem a função *esquerda()*, a velocidade será a velocidade máxima, ou seja, 255. Então, quando os motores passam a obedecer a função *frente()*, a velocidade será a calculada pela Equação 17.

Código final do programa com  $t_{giro}$  calculado:

```
// Bibliotecas
#include <interrupt.h>
#include <io.h>
#include <math.h>
#include <AFMotor.h>

#define DELAY_50ms 3125 //gerar 50ms Timer 5

// Pinos dos sensores

const int trigPin[8] = {38, 40, 42, 44, 46, 48, 52, 50}; // Trigger Pin
const int echoPin[8] = {39, 41, 43, 45, 47, 49, 53, 51}; // Echo Pin

// Definição dos motores

AF_DCMotor motor1(4);
AF_DCMotor motor2(1);
AF_DCMotor motor3(2);
AF_DCMotor motor4(3);

// Constantes do sensores

const char N_sensor = 8; //Numero de sensores
const char NMM = 3; //Numero de posicoes no vetor de MM

// Variaveis para contar o tempo

volatile unsigned int cont_dseg = 0; // contador decimos de segundos
volatile unsigned int cont_seg = 0; // contador de segundos
volatile unsigned int pula = 0; // flag que ativa e desativa os sensores

//*****Variaveis globais*****

//Variáveis da média móvel
```

```

unsigned char sensor_index = 0x00; //Sensor que esta sendo atualizado
unsigned char position_index = 0x00; //Posicao do vetor do sensor que esta sendo atualizado
unsigned int V_sensor[N_sensor][NMM];
    float mediaM[N_sensor] = { 0 };
    int duration = 0;

//Variáveis de cálculo
    float angulo_fuga = 0;    //Angulo de fuga
    int dist_obst = 1000;    //Distancia do obstaculo
unsigned char raio=10;
    float lado2 = 2*pow(raio,2) - 2*pow(raio,2)*cos(M_PI/4);
    float lado = sqrt( lado2 );
    float theta2 = acos( lado / (2*raio) );

//Variáveis do motor
unsigned char flag = 0;
    int vel;
    float tgiro;

//Inicialização do Arduino
void setup() {

Serial.begin (9600);

    for (unsigned char i = 0x00; i < N_sensor; i++)
    {
        pinMode(trigPin[i], OUTPUT);
        pinMode(echoPin[i], INPUT);
    }

    timer5_inicializa();
}

// Timer 5 - Inicializacao
void timer5_inicializa(void){
    cli();
    TCCR5A=0;
    TCCR5B=0;
    OCR5A = DELAY_50ms; // Interromper a cada 50 ms
    TCCR5B=(1<<WGM52)|(1<<CS52); // modo CTC e Prescaler=1/256
    TIMSK5|= (1<<OCIE5A); // Hab interrup por comparacao
    sei();
}

void loop() {

if(flag == 1){
    flag = 0;
    dist_ang();

    if (dist_obst <= 100) {
        vel = (-1.85)*dist_obst + 273.5;           // velocidade vai de 88,5 a 255
    }
}
}

```



```

    tgiro = 286.4873*abs(angulo_fuga);          // baseado no teste que 900ms = pi, com uma
alimentação de 7,5V no motor shield (6V nos motores)

    Serial.println(angulo_fuga);

    if (angulo_fuga>=-0.15 && angulo_fuga<=0.15){ //considerando que o angulo de fuga pode
dar negativo (-pi a +pi) e esta em rad
        frente();
    }

    else if ((angulo_fuga<=-0.15 && angulo_fuga>=-3.1415) || (angulo_fuga<=3.1415 &&
angulo_fuga>=3)) {
        direita();

        unsigned int temp = millis();
        unsigned int temp2 = millis();
        pula = 1;
        while((temp2-temp) < tgiro) {
            temp2 = millis();
        }
        frente();
        pula = 0;
    }

    else
    {
        esquerda();
        pula = 1;
        unsigned int temp = millis();
        unsigned int temp2 = millis();

        while((temp2-temp) < tgiro) {
            temp2 = millis();
        }
        frente();
        pula = 0;
    }
}

else {
    parar();
}
}
}

// Interrupcao Timer 5 - 50ms
ISR(TIMER5_COMPA_vect){

    unsigned int sum = 0x00;

    if (pula == 0){
        digitalWrite(trigPin[sensor_index], LOW);
    }
}

```

```

delayMicroseconds(2);

digitalWrite(trigPin[sensor_index], HIGH);
delayMicroseconds(10);

digitalWrite(trigPin[sensor_index], LOW);

duration = pulseIn(echoPin[sensor_index], HIGH);

int distancia = duration/58.2;//distance

if (distancia > 400) {
  V_sensor[sensor_index][position_index] = 400;
}
else {
  V_sensor[sensor_index][position_index] = distancia;
}

for (unsigned char i=0x00; i<NMM; i++){
  sum = sum + V_sensor[sensor_index][i];
}
  mediaM[sensor_index] = sum/NMM;

  sensor_index++;

  if (sensor_index >= N_sensor) {
    sensor_index = 0;
    position_index++;

    if(position_index >= NMM) {
      position_index = 0;
    }
  }
}
  cont_dseg++;

  if(cont_dseg==24){    cont_dseg=0; cont_seg++;  flag = 1;

  }
}

//Rotina de cálculo

void dist_ang(){

  unsigned int rm[2] = {0};
  unsigned int pos[2] = {0};

  for (unsigned char n = 0; n < 2; n++){

```

```

rm[n] = 100000;
pos[n] = 9;

for (unsigned char m = 0; m < 8; m++) {

    if (n == 0){
        if (mediaM[m] < rm[n]){
            rm[n] = mediaM[m];
            pos[n] = m;
        }
    }
    else if (m != pos[0]){
        if (mediaM[m] < rm[1]){
            rm[n] = mediaM[m];
            pos[1] = m;
        }
    }
}

if (pos[0] < pos[1]){
    unsigned char temp = pos[0];
    pos[0] = pos[1];
    pos[1] = temp;

    temp = rm[0];
    rm[0] = rm[1];
    rm[1] = temp;
}
if ((pos[0] == 7) && (pos[1] == 0)){
    unsigned char temp = pos[0];
    pos[0] = pos[1];
    pos[1] = temp;

    temp = rm[0];
    rm[0] = rm[1];
    rm[1] = temp;
}

float costheta1 = ( pow(lado,2) + pow(rm[0],2) - pow(rm[1],2) ) / (2*lado*rm[0]);

float theta1 = acos( costheta1 );

if (theta1 >= 0){

    float thetaT = theta2 + theta1;

    dist_obst = sqrt(pow(raio,2) + pow(rm[0],2) - 2*raio*rm[0]*cos(thetaT));

    float alpha = acos( (pow(raio,2) + pow(dist_obst,2) - pow(rm[1],2))/(2*raio*dist_obst) );

```

```

float theta = alpha+(pos[1])*M_PI/4;

angulo_fuga = (theta-M_PI);
}
else {
if (rm[0] <= rm[1]){
dist_obst = rm[0]+10;
angulo_fuga =(pos[0])*M_PI/4 - M_PI;
}
else {
dist_obst = rm[1]+10;
angulo_fuga =(pos[1])*M_PI/4 - M_PI;
}
}
}
}

```

//Funções dos motores

```

void parar(){
motor1.run(RELEASE);
motor2.run(RELEASE);
motor3.run(RELEASE);
motor4.run(RELEASE);
}

```

```

void frente(){
motor1.run(FORWARD);
motor2.run(FORWARD);
motor3.run(FORWARD);
motor4.run(FORWARD);
motor1.setSpeed(vel);
motor2.setSpeed(vel);
motor3.setSpeed(vel);
motor4.setSpeed(vel);
}

```

```

void direita(){
motor1.run(FORWARD);
motor2.run(FORWARD);
motor3.run(BACKWARD);
motor4.run(BACKWARD);
motor1.setSpeed(255);
motor2.setSpeed(255);
motor3.setSpeed(255);
motor4.setSpeed(255);
}

```

```

void esquerda(){
motor1.run(BACKWARD);
motor2.run(BACKWARD);
motor3.run(FORWARD);
motor4.run(FORWARD);
}

```

```

motor1.setSpeed(255);
motor2.setSpeed(255);
motor3.setSpeed(255);
motor4.setSpeed(255);
}

```

## APÊNDICE B:

Algoritmo do programa final com  $t_{giro}$  constante:

1. O programa dos sensores começa com a inicialização do Timer 5 para gerar interrupções a cada 50 ms. Sendo que a cada interrupção o sistema realiza a leitura de um sensor, atualiza o vetor de distâncias e o vetor média móvel de cada sensor.
2. Caso a *flag* ‘pula’ esteja ativada, ou seja, igual a zero:
  - O pino de *trigger* do sensor encontra-se em nível baixo e em seguida gera um pulso em nível alto por 10  $\mu$ s, retornando ao nível baixo após isso. O sinal de recebimento do pulso retorna pelo pino de *echo*. A CPU recebe esse pulso e o tempo transcorrido para tal é colocado na variável *duration* e, assim, é calculada a distância do obstáculo e este valor é colocado em uma posição no vetor daquele sensor.
  - O cálculo da média móvel é feito com um *loop for* que soma todos os valores do vetor do sensor e então divide 3, o tamanho do vetor. O resultado é carregado no vetor de médias móveis.
  - O *sensor\_index* é incrementado para que o próximo sensor seja atualizado no próximo ciclo. Quando o indexador dos sensores atinge o número de sensores, o indexador retorna para o 0 e o *position\_index* é incrementado mudando a posição no vetor das distâncias. Ao se atingir *position\_index* igual a 8, o indexador é retornado para 0 novamente.
3. Finalizando a rotina de interrupção, a variável de contagem de décimos de segundo é incrementada.
4. A cada 1,2 segundos uma *flag* é acionada indicando ao *loop* principal da passagem do tempo. Então, o *loop* chama a rotina de cálculo do ângulo de fuga.
5. A rotina ordena as médias móveis da menor para a maior, salvando as duas menores medidas e seus sensores correspondentes. Estes valores salvos são ordenados por ordem decrescente do número do sensor e seguem o algoritmo para o cálculo da distância do objeto ( $d_T$ ) e do ângulo de fuga ( $\varphi_{fuga}$ ) explicado anteriormente.
6. Caso o programa não consiga calcular os parâmetros, ele define o ângulo de fuga como o ângulo diametralmente oposto ao sensor de menor distância e esta distância será tomada como distância do objeto.
7. Uma vez definidos os parâmetros  $d_T$  e  $\varphi_{fuga}$ , o programa dos motores calcula a velocidade da plataforma. Sendo que, para isso,  $d_T$  precisa ser menor que 100, ou seja, a distância do objeto deve ser menor que 1 metro. Caso contrário, o programa entra na função *parar()*, assim, a plataforma ficará parada.

8. Caso  $d_T \leq 100$ , o programa irá analisar o  $\varphi_{fuga}$  para determinar qual direção a plataforma deverá seguir. Teremos três possíveis casos:
- Caso  $-0,80 \leq \varphi_{fuga} \leq 0,80$ , o programa executa a função *frente()* na qual os motores andam na velocidade calculada pela Equação 17;
  - Caso  $-3,1415 \leq \varphi_{fuga} < -0,80$  ou  $3 \leq \varphi_{fuga} \leq 3,1415$ , o programa desativa a *flag* 'pula', executa a função *direita()* durante 180 ms, executa a função *parar()* e, então, ativa a *flag* novamente. Durante o tempo em que os motores obedecem a função *direita()*, a velocidade será a velocidade máxima, ou seja, 255;
  - Caso não esteja em nenhum dos casos anteriores, ou seja,  $0,80 < \varphi_{fuga} < 3$ , o programa desativa a *flag* 'pula', executa a função *esquerda()* durante 180 ms, executa a função *parar()* e, então, ativa a *flag* novamente. Da mesma forma, durante o tempo em que os motores obedecem a função *esquerda()*, a velocidade será a velocidade máxima, ou seja, 255.

Código final do programa com  $t_{giro}$  constante:

```
//Bibliotecas
#include <interrupt.h>
#include <io.h>
#include <math.h>
#include <AFMotor.h>

#define DELAY_50ms 3125 //gerar 50ms Timer 5

// Pinos dos sensores

const int trigPin[8] = {38, 40, 42, 44, 46, 48, 52, 50}; // Echo Pin
const int echoPin[8] = {39, 41, 43, 45, 47, 49, 53, 51}; // Trigger Pin

// Definição dos motores

AF_DCMotor motor1(4);
AF_DCMotor motor2(1);
AF_DCMotor motor3(2);
AF_DCMotor motor4(3);

// Constantes do sensores

const char N_sensor = 8; //Numero de sensores
const char NMM = 3; //Numero de posicoes no vetor de MM

// Variaveis para contar o tempo
volatile unsigned int cont_dseg = 0; // contador decimos de segundos
volatile unsigned int cont_seg = 0; // contador de segundos
volatile unsigned int pula = 0;

////***Variaveis globais
//Variáveis da média móvel
```

```

unsigned char sensor_index = 0x00; //Sensor q esta sendo atualizado
unsigned char position_index = 0x00; //Posicao do vetor do sensor q esta sendo atualizado

unsigned int V_sensor[N_sensor][NMM]; //8 linhas 8 colunas
float mediaM[N_sensor] = { 0 };
int duration = 0;

//Variáveis de cálculo
float angulo_fuga = 0; //Angulo de fuga
int dist_obst = 1000; //Distancia do obstaculo
unsigned char raio=10;
float lado2 = 2*pow(raio,2) - 2*pow(raio,2)*cos(M_PI/4);
float lado = sqrt( lado2 );

//Variáveis do motor
unsigned char flag = 0;
int vel;
float tgiro;

void setup() {

Serial.begin (9600);

for (unsigned char i = 0x00; i < N_sensor; i++)
{
pinMode(trigPin[i], OUTPUT);
pinMode(echoPin[i], INPUT);
}

timer5_inicializa();
}
// Timer 5 - Inicializacao
void timer5_inicializa(void){
cli();
TCCR5A=0;
TCCR5B=0;
OCR5A = DELAY_50ms; // Interromper a cada 50 ms
TCCR5B=(1<<WGM52)|(1<<CS52); // modo CTC e Prescaler=1/256
TIMSK5|= (1<<OCIE5A); // Hab interrup por comparacao
sei();
}

void loop() {

if(flag == 1){
flag = 0;
dist_ang();

if (dist_obst<=100) {
vel = (-1.85)*dist_obst + 273.5 ; // velocidade vai de 70 a 255
tgiro = 180; // tempo para uma curva de 36 graus
}
}
}

```

```

//Serial.println(angulo_fuga);

    if (angulo_fuga>=-0.80 && angulo_fuga<=0.80) //considerando que o angulo de fuga pode
dar negativo (-pi a +pi) e esta em rad
    {
        frente();
    }

    else if ((angulo_fuga<=-0.80 && angulo_fuga>=-3.1415) || (angulo_fuga<=3.1415 &&
angulo_fuga>=3))
    {
        direita();

        unsigned int temp = millis();
        unsigned int temp2 = millis();

        pula = 1;
        while((temp2-temp) <= tgiro) {
            temp2 = millis();
        }
        parar();
        pula = 0;
    }

    else
    {
        esquerda();

        unsigned int temp = millis();
        unsigned int temp2 = millis();

        pula = 1;
        while((temp2-temp) <= tgiro) {
            temp2 = millis();
        }
        parar();
        pula = 0;
    }
}

else
{
    parar();
}
}
}

// Interrupcao Timer 5 - 50ms
ISR(TIMER5_COMPA_vect){

    unsigned int sum = 0x00;

```



```

if (pula == 0){
digitalWrite(trigPin[sensor_index], LOW);
delayMicroseconds(2);

digitalWrite(trigPin[sensor_index], HIGH);
delayMicroseconds(10);

digitalWrite(trigPin[sensor_index], LOW);

duration = pulseIn(echoPin[sensor_index], HIGH);

int distancia = duration/58.2;//distance

if (distancia > 400) {
  V_sensor[sensor_index][position_index] = 400;
}
else {
  V_sensor[sensor_index][position_index] = distancia;
}

for (unsigned char i=0x00; i<NMM; i++){
sum = sum + V_sensor[sensor_index][i];
}
mediaM[sensor_index] = sum/NMM;

sensor_index++;

if (sensor_index >= N_sensor) {
  sensor_index = 0;
  position_index++;

  if(position_index >= NMM) {
    position_index = 0;
  }
}

cont_dseg++;

if(cont_dseg==24){  cont_dseg=0; cont_seg++; flag = 1;
}
}

//Rotina de cálculo

void dist_ang(){

unsigned int rm[2] = {0};
unsigned int pos[2] = {0};

```

```

for (unsigned char n = 0; n < 2; n++){

    rm[n] = 100000;
    pos[n] = 9;

    for (unsigned char m = 0; m < 8; m++) {

        if (n == 0){
            if (mediaM[m] < rm[n]){
                rm[n] = mediaM[m];
                pos[n] = m;
            }
        }
        else if (m != pos[0]){
            if (mediaM[m] < rm[1]){
                rm[n] = mediaM[m];
                pos[1] = m;
            }
        }
    }
}

if (pos[0] < pos[1]){
    unsigned char temp = pos[0];
    pos[0] = pos[1];
    pos[1] = temp;

    temp = rm[0];
    rm[0] = rm[1];
    rm[1] = temp;
}
if ((pos[0] == 7) && (pos[1] == 0)){
    unsigned char temp = pos[0];
    pos[0] = pos[1];
    pos[1] = temp;

    temp = rm[0];
    rm[0] = rm[1];
    rm[1] = temp;
}

float costheta1 = ( pow(lado,2) + pow(rm[0],2) - pow(rm[1],2) ) / (2*lado*rm[0]);

float theta1 = acos( costheta1 );

if (theta1 >= 0){

    float theta2 = acos( lado / (2*raio) );

    float thetaT = theta2 + theta1;

```

```

dist_obst = sqrt(pow(raio,2) + pow(rm[0],2) - 2*raio*rm[0]*cos(thetaT));

float alpha = acos( (pow(raio,2) + pow(dist_obst,2) - pow(rm[1],2))/(2*raio*dist_obst) );

float theta = alpha+(pos[1])*M_PI/4;

    angulo_fuga = (theta-M_PI);
}
else {
    if (rm[0] <= rm[1]){
        dist_obst = rm[0]+10;
        angulo_fuga =(pos[0])*M_PI/4 - M_PI;
    }
    else {
        dist_obst = rm[1]+10;
        angulo_fuga =(pos[1])*M_PI/4 - M_PI;
    }
}
}

//Funções dos motores

void parar(){
    motor1.run(RELEASE);
    motor2.run(RELEASE);
    motor3.run(RELEASE);
    motor4.run(RELEASE);
}

void frente(){
    motor1.run(FORWARD);
    motor2.run(FORWARD);
    motor3.run(FORWARD);
    motor4.run(FORWARD);
    motor1.setSpeed(vel);
    motor2.setSpeed(vel);
    motor3.setSpeed(vel);
    motor4.setSpeed(vel);
}

void direita(){
    motor1.run(FORWARD);
    motor2.run(FORWARD);
    motor3.run(BACKWARD);
    motor4.run(BACKWARD);
    motor1.setSpeed(255);
    motor2.setSpeed(255);
    motor3.setSpeed(255);
    motor4.setSpeed(255);
}

void esquerda(){

```

```
motor1.run(BACKWARD);  
motor2.run(BACKWARD);  
motor3.run(FORWARD);  
motor4.run(FORWARD);  
motor1.setSpeed(255);  
motor2.setSpeed(255);  
motor3.setSpeed(255);  
motor4.setSpeed(255);  
}
```