PROJETO DE GRADUAÇÃO

Plataforma móvel com detecção de obstáculos

Lucas Ferreira Lima

Matheus de Moraes Soares

Brasília, 7 de Julho de 2017

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

Lucas Ferreira Lima
Matheus de Moraes Soares

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. Alexandre Romariz, UnB/ ENE
Examinador interno

Prof. Eduardo Peixoto Fernandes da Silva, UnB/ ENE
Examinador interno

Brasília, 7 de Julho de 2017

Dedicatória

Dedico esse trabalho à minha família que me apoiou incondicionalmente até aqui.

Lucas Ferreira Lima

Dedicatória

Dedico, com saudades, à vó Carmem, eternamente minha motivação.

Matheus de Moraes Soares

Agradecimentos

Agradeço a todos que contribuíram de forma direta ou indireta em minha caminhada até a graduação. Em especial ao meu orientador, Prof. Ricardo Zelenovsky, por seus ensinamentos, disposição e suporte no decorrer deste projeto.
A meus pais, Andrea e Neto, os alicerces da minha vida, pelos conselhos, pela paciência, pela educação e pelo amor incondicional que só eles poderiam dar. Sem os dois nada seria possível. Em especial a meu pai, que foi a minha fonte de inspiração e responsável por me despertar o interesse em seguir a carreira de engenheiro.
Ao meu irmão, Pedro, pela parceria e amizade e por me manter acordado de madrugada enquanto realizava o trabalho.
A meus grandes amigos, Artur e Marcos, que dividiram a dor das derrotas e comemoraram cada vitória da vida acadêmica no decorrer desses (longos) anos, vocês foram fundamentais para essa conquista.
Ao meu primo João Lucas, por ter me acolhido no ambiente universitário e por me ajudar nas diversas situações encontradas nele.
Agradeço ainda à minha namorada Camila, pela dedicação em me manter tranquilo e seguro na reta final do trabalho. Por nunca ter deixado de acreditar em mim e por todo o amor que conseguiu demonstrar na forma de paciência, equilíbrio e de palavras de motivação.
Por fim agradeço à todos os professores do Departamento de Engenharia Elétrica e aos meus colegas de curso, em especial à Elisa Monteiro, que me auxiliou com sugestões e ideias para este trabalho.

Matheus de Moraes Soares

Agradecimentos

Agradeço à minha família pelo apoio nos diversos momentos de dificuldades e pelo júbilo nos momentos de sucesso. Agradeço a todo corpo docente do Departamento de Engenharia Elétrica, em especial ao professor Ricardo Zelenovsky por sua competência e solicitude. Agradeço também minha companheira de luta Ellen Dourado que me apoiou em todos os momentos.

Lucas Ferreira Lima

RESUMO

Este projeto relata as modificações feitas para aprimoramento de uma plataforma móvel para detecção de obstáculos. Esta é composta por sensores ultrassônicos conectados a um microcontrolador (Arduino Due). A tomada de decisão é realizada pelo Arduino, que, por meio de um *Shield* (AD *Fruit*), imprime sinal PWM a motores de corrente contínua. Os aprimoramentos consistiram em melhoras da estabilidade do sistema e da usabilidade da plataforma.

Palavras-chave: *Arduino, Due, ARM, Bluetooth, ultrassom, controle, autônomo.*

ABSTRACT

This project describes the adjustments performed in order to upgrade the mobile platform with obstacle detection. This platform is composed by ultrasound sensors connected to a microcontroler (Arduino Due). The decision-making is performed by the Arduino, which, through the Motor Shield (AD Fruit), generates a PWM signal to the DC motors. The upgrade consists in improving the system stability and the platform usability.

Keywords: *Arduino, Due, ARM, Bluetooth, ultrasound, control, autonomous.*

SUMÁRIO

LISTA DE IMAGENS

1. Introdução

1.1. Ambientação

Microcontroladores são pequenos dispositivos providos de um ou mais processadores associados a uma ou mais memórias. Pode-se dizer que se trata de uma espécie de computador que controla diversas portas digitais e analógicas. São utilizados nos mais diversos sistemas automatizados.

O ritmo de desenvolvimento da indústria é muito acelerado. Projetos são constantemente renovados e geram novos produtos. É muito comum um produto se tornar supérfluo com a chegada de uma nova versão mais eficiente e econômica. Na indústria de microcontroladores não é diferente. A cada nova geração nota-se maior poder de processamento e maior facilidade de uso, possibilitando novas funções e/ou custos menores de implementação.

Outro fator que facilitou a aceitação desses produtos foi a globalização. É comum encontrar diversas páginas na *web* que oferecem uma espécie de suporte colaborativo. Usuários perguntam e expõem seus projetos a outros usuários, fomentando o conhecimento. Um exemplo disso foi a empresa Arduino, que lançou um processador juntamente com seu suporte colaborativo (Arduino.cc). Em apenas um ano, a empresa alcançou a marca de 50 mil placas vendidas.

A soma do número de usuários com o contexto globalizado resultou numa simbiose entre o aprimoramento dos microcontroladores e a facilidade de uso. Assim, a cada dia são apresentadas novas invenções. Um exemplo são os robôs (Figura 1). Sejam eles na forma humanoide, que normalmente são usados em competições e fins educativos, ou máquinas que se movem de forma autônoma, com objetivo de exercer uma função específica, os robôs exercem atividades previamente programadas.

O *Robotics Institute of America* (RIA), ou Instituto Americano de Robótica, define um robô como um manipulador reprogramável e multifuncional projetado para mover materiais, partes, ferramentas ou dispositivos especializados, por meio de movimentos variáveis programados para desempenhar uma variedade de tarefas [1].

Figura 1: Exemplo de robô [2].

Apesar de a primeira idealização do robô ter sido registrada no Egito antigo, somente em 1922 Karel Capek introduziu a palavra robô, que significa trabalhador em Tcheco. Em 1946, George Devol desenvolveu um tratamento magnético possibilitando Eckert e Mauchley construírem o computador ENIAC[1], que foi o primeiro robô programado da história. Já o primeiro robô de caráter industrial foi desenvolvido em 1962 para automatizar processos da fábrica da *General Motors* [2].

São várias as demandas que podem ser atendidas com o uso de robôs. A precisão e a sensibilidade nos movimentos atingiram um nível altíssimo. Não é difícil encontrar projetos de utilização de robôs na área de saúde, como o desenvolvido pela professora Mariana Costa Bernardes, da Universidade de Brasília, referente a posicionamento e guiagem de agulhas para cirurgias não assistidas [3].

Outro cenário em que predomina o uso de robôs é a indústria, em especial a automobilística. Nesse caso é desejável que a fabricação dos produtos seja rápida, em larga escala e que atendam a rigorosos controles de qualidade. A Figura 1, por exemplo, mostra um braço robótico que operam aparelhos de solda.

Embora o desenvolvimento da robótica seja constante, os preços dos produtos ainda são altos para o consumidor final. A sofisticação do projeto pode ser responsável por esse encarecimento, ou seja, o uso de sensores mais precisos, de motores mais potentes ou de processadores com melhor desempenho está diretamente ligado ao preço final.

[1] *Electronic Numerical Integrator and Computer* (em português: computador integrador numérico eletrônico).

1.2. Motivação

Esse projeto é uma plataforma móvel que faz detecção e desvio de obstáculos usando sensores de ultrassom e, por isso, é uma configuração muito versátil. Dentre as diversas aplicabilidades do dispositivo, destaca-se a automotiva. Até marcas populares como a *Ford* têm em seu portfólio carros com piloto automático adaptativo que utiliza sensores e câmeras para guiarem o veículo com a supervisão do motorista. Apesar de se orientar principalmente por câmera, o sistema utiliza sensores em redundância para garantir melhor segurança.

Outro ramo de interesse é o uso doméstico de robôs. Há diversas marcas que têm oferecido robôs aspiradores de poeira que percorrem o ambiente desviando-se dos objetos. Nesse caso, por serem sistemas mais modestos, dependem diretamente de sensores semelhantes àqueles utilizados no projeto proposto por esse trabalho.

A implementação de sensores ultrassônicos tem uma boa relação custo-benefício, ainda mais nesse caso, em que o projeto base dispõe de unidade de processamento compatível.

O principal desafio desse projeto é oferecer uma plataforma estável e que realize poucos movimentos quando não há obstáculos por perto. Além disso, pretende-se consumir pouca capacidade de processamento para possibilitar futuras modificações.

1.3. Principais problemas da versão anterior

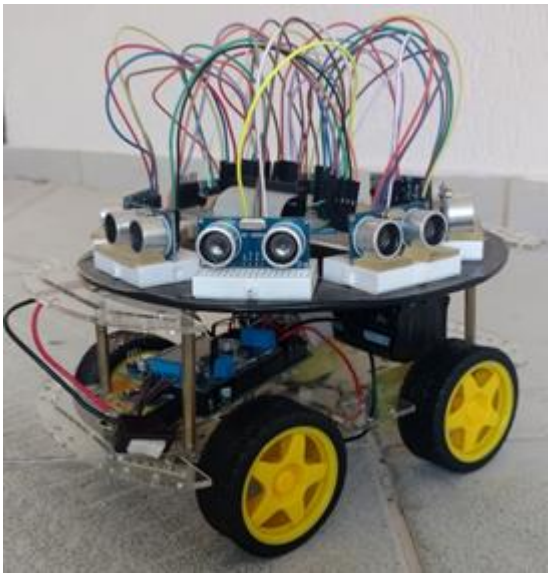

Figura 2: Versão anterior [5]

O projeto anterior (Figura 2) consistia na mesma plataforma usada neste trabalho, ela também fazia uso de sensores ultrassônicos para realizar a detecção de obstáculos e, em seguida, desviar do mesmo. O maior desafio do projeto passado era a mudança da placa para Arduino Due.

Embora atendesse àquilo que foi proposto, o projeto base possuía algumas falhas de performance. O sistema entrava em oscilação realizando pequenos movimentos como se ameaçasse girar mesmo sem nenhum objeto próximo. Como será melhor elucidado na seção 2.2.1 Detecção de obstáculos, na página 9, essa instabilidade provavelmente foi causada pela utilização inadequado de um temporizador responsável pelo funcionamento do sistema como um todo. Outra possível causa para essa oscilação é o fato de a rotina de giro estar associada à fuga. O sistema não identificava qual rotina deveria seguir, pois poderia haver interrupção de detecção de obstáculo em meio à fuga.

Na apresentação do projeto anterior aparentemente um motor da plataforma disparou por falha mecânica. Posteriormente se atribuiu essa falha recorrente ao código do programa. Verificou-se que havia dois laços temporais com um pequeno descompasso que, após somar um período de tempo se descompassava consideravelmente e causava esse tipo de problema.

Outro ponto que levantou questionamentos foi a baixa eficiência no processo de carregamento das baterias. No projeto base, era preciso remover todas as baterias da plataforma para que elas fossem carregadas individualmente na rede elétrica. Com isso foi visto que tal procedimento, apesar de atender seu objetivo, demandava esforço e tempo que poderiam ser evitados.

Quando os trabalhos foram iniciados, percebeu-se que as leituras dos sensores eram muito inconstantes e que havia alguns problemas de contato entre esses componentes e a placa Arduino.

Por fim, os motores eram presos na parte inferior da plataforma de baixo com fita dupla face. Agora fixados com parafusos, os motores ficam mais firme e com melhor sustentação. A inserção de mais uma base contribuiu para o reforço da estrutura, que era muito maleável e susceptível a torções quando manuseada.

1.4. Objetivos

De forma muito abrangente, o principal objetivo desse trabalho é o aperfeiçoamento da plataforma antiga. Seguem os objetivos dos principais pontos de modificação:

a) **Estabilidade:** Garantir que o sistema de temporização esteja adequado, evitando assim trepidações e possíveis outras falhas.
b) **Confiabilidade:** Entregar soluções mecânicas compatíveis com o grau de exigência para o bom funcionamento, substituindo e limpando peças defeituosas
c) **Usabilidade:** Anteriormente, o projeto continha um sistema muito primitivo de alimentação. Eram pilhas soldadas a fios que precisavam ser alimentadas por um carregador de parede que tinha capacidade de carregar somente uma bateria por vez. Desta maneira era necessário tirá-las da solda e esperar cerca de seis horas por pilha, totalizando 24 horas de espera para o carregamento. Também

era usado um carregador de celular para alimentar o Arduino. Foi necessária uma nova solução, de forma que o carregamento pudesse ser feito dentro da plataforma e em tempo razoável.

d) **Eficácia:** O sistema de sensores realizava muitas leituras erradas. Felizmente uma errada dificilmente tem valor pequeno (que poderia fazer a plataforma tentar se desviar). Nesse caso normalmente o sinal *echo* não chega ao Arduino e a medida de tempo é máxima. Espera-se encontrar uma maneira razoável de minimizar leituras defeituosas.

O desafio é promover as mudanças e manter o projeto dentro de um perfil de baixo custo. Para isso, foi necessário minimizar a compra de novos componentes e utilizar sucatas ou objetos descartados.

1.5. Resumo das modificações

A seção 2.2.1 Detecção de obstáculos, na página 9 relatará que o trabalho anterior utilizou a função *delay.Microsseconds()*, responsável pelas trepidações. A maior alteração consistiu na mudança completa da lógica de programação para abolir seu uso.

Foram descartadas duas peças do projeto anterior. Seguem a relação e o motivo:

a) **Placa de conexões:** impossibilitava redundância nos contatos dos sensores e aumentava o número de conexões susceptíveis ao mau contato.

b) **Carregador USB:** houve a necessidade de adquirir um novo componente, já que esse *power bank* era de propriedade de um dos componentes do grupo anterior de trabalho. Optou-se pelo uso de pilhas e conector *Jack*, por sua disponibilidade.

Foram inseridos três carregadores capazes de carregar seis pilhas ao mesmo tempo em uma tomada. Para isso, implementou-se um chaveamento que alterna o sistema entre dois estados: no primeiro, a bateria está sendo carregada na rede elétrica; no segundo ela está fornecendo energia aos motores e ao Arduino. Esse chaveamento será melhor explicado na seção 3.7 Baterias, na página 27. Essa configuração evita a remoção das baterias da plataforma para seu carregamento, além de tornar desnecessário o uso de soldas nas extremidades das baterias quando usadas na alimentação do projeto.

O funcionamento do módulo *Bluetooth* foi simplificado. Anteriormente, a comunicação era feita por meio do Matlab e mostrava uma série de aspectos da plataforma. Se por um lado isso permitia ensaios mais informativos, por outro apresentava problema de temporização, tornando uma rotina muito longa e provocando descompasso do Relógio. Apesar de ser possível implementar essa comunicação de forma mais exitosa, optou-se por simplificá-la para descartar possível causa de mau funcionamento. Caso se queira, futuramente, acrescentá-la na forma original, é aconselhável a verificação da viabilidade de sua coexistência com o sistema de triangulação de som (que será implementado no próximo projeto).

Isso porque a plataforma provavelmente conterá mais rotinas que podem provocar problemas de temporização.

Notou-se que pequenas mudanças implementadas causaram ganhos de performance. Por exemplo, diminuiu-se o tempo entre leituras de 80 para 73,4 milissegundos. Ressalta-se que para possibilitar o giro da plataforma, é necessário reduzir o atrito das rodas no solo. Por isso, foram executadas mudanças com a finalidade de tornar a superfície de contato mais lisa.

A lógica de detecção de objeto teve aprimoramentos pontuais. Antes, bastava que um sensor acusasse um objeto próximo para que houvesse tomada de decisão pelo processador. Hoje, ela é baseada na leitura de dois sensores. A mudança objetiva evitar que uma leitura errada cause decisão de giro. Outra mudança no mesmo sentido foi o descarte de medidas erradas. Antes, comparava-se a média atual, que, caso não fosse aceita, era substituída pela anterior. As condições para essa aceitação foram mantidas, porém agora é imposta sobre a medida atual e não sobre a média. Ou seja, se uma medida é caracterizada como errada por uma série de requisitos, ela é descartada e a média só é modificada quando houver um novo dado aceitável.

No projeto original, ambas as rotinas, detecção e fuga, ocorriam simultaneamente. Isso gerava uma certa instabilidade na plataforma, já que, ao longo do movimento, ela não detectava corretamente o obstáculo. Portanto, esta versão foi implementada de forma que as rotinas aconteçam separadamente.

1.6. Estrutura do Trabalho

O trabalho foi dividido da seguinte forma:

- *Capítulo 1 – Introdução*: Ambientação, Motivação, Principais problemas da versão anterior, Objetivos, Resumo das modificações e Estrutura do Trabalho
- *Capítulo 2 – Projeto Base:* Plataforma: Funcionamento da Plataforma, Detecção de obstáculos e Giro e Fuga
- Capítulo 3 – Hardware: Plataforma, Arduino, Sensor ultrassom, Arduino Motor Shield V2, Controle de Velocidade (PWM), I2C, Módulo *Bluetooth*, Motores DC, Baterias, Chaveamento e **Erro! Fonte de referência não encontrada.**
- *Capítulo 4 – Programação:* Firmware, Interrupções, Interrupções externas, Interrupções por temporização, Interrupção serial, Cálculo de distâncias e ângulo de fuga
- *Capítulo 5 – Considerações finais*

2. Projeto Base

O projeto original foi desenvolvido pela aluna Aline Barbosa em seu trabalho de conclusão de curso [4]. Em seguida, foi então aprimorado pelos alunos André Agostinho e Luiz Fernando para solucionar problemas de alimentação e de estrutura mecânica, no respectivo trabalho de conclusão de curso [5]. O projeto tinha como base uma plataforma móvel que, ao utilizar sensores de ultrassom, fosse capaz de, através de um algoritmo, identificar obstáculos, definir um ângulo de fuga e acionar motores para se afastar do obstáculo. O trabalho foi assumido posteriormente pelos alunos Renato Ferreira e Igor Barroso [6]. O projeto deles teve como principal mudança a troca da placa Arduino ATMega 2560 pela Arduino Due, que possui maior capacidade de processamento, além de permitir interrupções em todas as portas. Por fim, o trabalho foi assumido pelos alunos Lucas Ferreira Lima e Matheus de Moraes Soares, para cumprir os objetivos descritos na seção 1.4 página 4.

2.1. Plataforma:

Segue uma lista dos componentes usados diretamente:

- 8 *Protoboards* pequenas para os sensores
- 8 Sensores HC-SR04
- Arduino DUE
- Módulo *Bluetooth* HC-06
- 3 Placas de acrílico (chassi)
- 4 Motores com caixa de redução e roda acoplados
- 3 Carregadores duplos de baterias 18650
- 6 Baterias 18650
- Interruptor de parede (Sucata)
- Multi chave seletora 4P8T (Sucata)
- Multi chave seletora 8P16T (Sucata)
- Tomada de parede (Sucata)
- Suporte para Arduino feito à mão utilizando papelão reaproveitado
- Disco de sustentação dos sensores
- 2 Porta-fusíveis (sucata)
- 1 saco de fios tipo *Jumper* (macho-macho)
- 1 saco de fios tipo *Jumper* (fêmea-macho)
- 1 saco de fios tipo *Jumper* (fêmea-fêmea)
- Fio duplo 2,5 mm

- Parafusos para sustentação
- Fita isolante
- Solda
- Fita durex
- 2 *plugs* machos (Sucata)

As três placas de acrílico compõem os três andares da plataforma. No primeiro ficam os motores, o interruptor e a tomada de parede. No segundo, as barcas de pilhas auxiliadas pelas multichaves seletoras e os porta-fusíveis. Por fim, o terceiro andar contém o disco de sustentação dos sensores, as 8 protoboards, o suporte e o Arduino. Os fios tipo *Jumper* foram usados para ligar as chaves seletoras às barcas e também o Arduino aos sensores. Os itens descritos como sucata foram obtidos no SG11[2]. Lá existe uma sala com os mais diversos componentes reaproveitados do lixo eletrônico da Universidade de Brasília.

2.2. Funcionamento da Plataforma

O funcionamento da plataforma base pode ser dividido em duas partes principais: a detecção de obstáculos e a fuga. A detecção de obstáculos envolve a lógica do uso do sensor de ultrassom para o cálculo da distância e direção em que o obstáculo se encontra. A fuga inclui a lógica de acionamento dos motores e o intervalo de tempo em que eles permanecerão acionados. Essa lógica de fuga ainda foi subdividida em duas: o giro e o distanciamento do obstáculo

2.2.1 Detecção de obstáculos

A operação de detecção de obstáculo tem como dispositivo principal o sensor ultrassônico HC-SR04, que será abordado na seção 3.3, página 19. Seu funcionamento inicia quando sua porta *trigger* é acionada. De acordo com o fabricante, esse tempo de acionamento deve ser de 10µs em nível alto [7]. Quando isso ocorre, o sensor emite um pulso de ultrassom e espera o sinal de retorno como mostrado na Figura 3. A diferença de tempo entre o disparo e o recebimento do *echo* permite calcular a distância até o objeto mais próximo.

[2] Laboratório de Engenharia Elétrica da Universidade de Brasília.

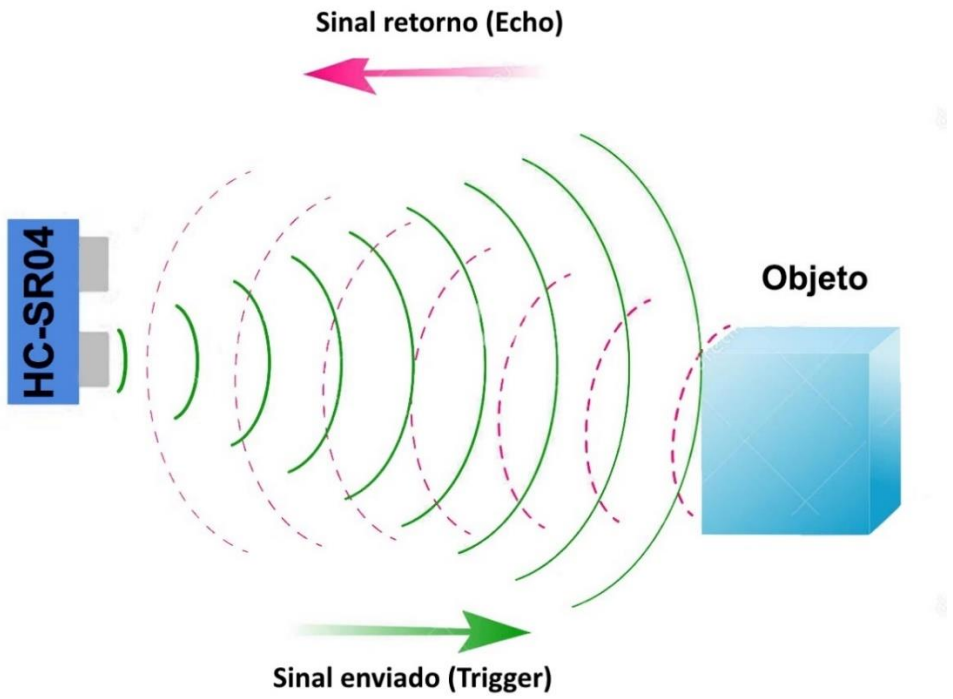

Figura 3: Ilustração do mecanismo utilizado para detecção de obstáculos [9]

Dado o funcionamento do sensor, fica evidente a necessidade de haver uma contagem muito precisa para aferir o tempo entre a emissão e a recepção da onda. Programando um dos contadores do Arduino Due para operar em 10,5 MHz, foi possível contar 105 batidas para integrar uma medida de tempo de 10µs. Esse tempo foi escolhido primeiramente por ser o de acionamento do *trigger* e, por coincidência, oferecia boa precisão para aferição de tempo.

Na versão anterior, para gerar o pulso de 10µs era usada uma função chamada delayMicroseconds(), como pode ser visto na Figura 4. Essa é uma função que interrompe o temporizador zero, responsável pelo funcionamento da placa. Assim, mesmo havendo outras rotinas em curso, há uma pequena interrupção temporal total do sistema, e a CPU fica parada por 10µs. Sabendo que isso ocorre oito vezes a cada 0,8 milissegundos, o produto final pode apresentar trepidações. A nova solução adotada faz essa espera de 10µs por meio de temporizadores. Isso permitiu que a plataforma opere sem trepidações e com desempenho sem engasgos.

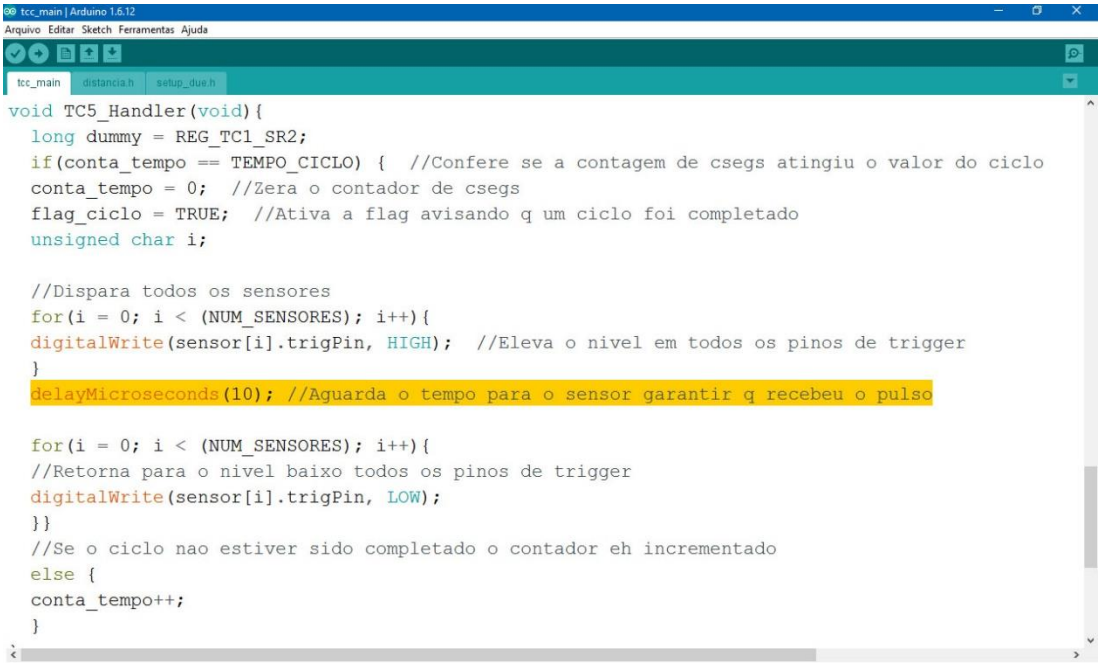

Figura 4: Captura de tela indicando em amarelo a função delayMicroseconds().

Foi escolhida a disposição da Figura 5 a fim de garantir que qualquer objeto possa ser detectado independentemente da face de aproximação:

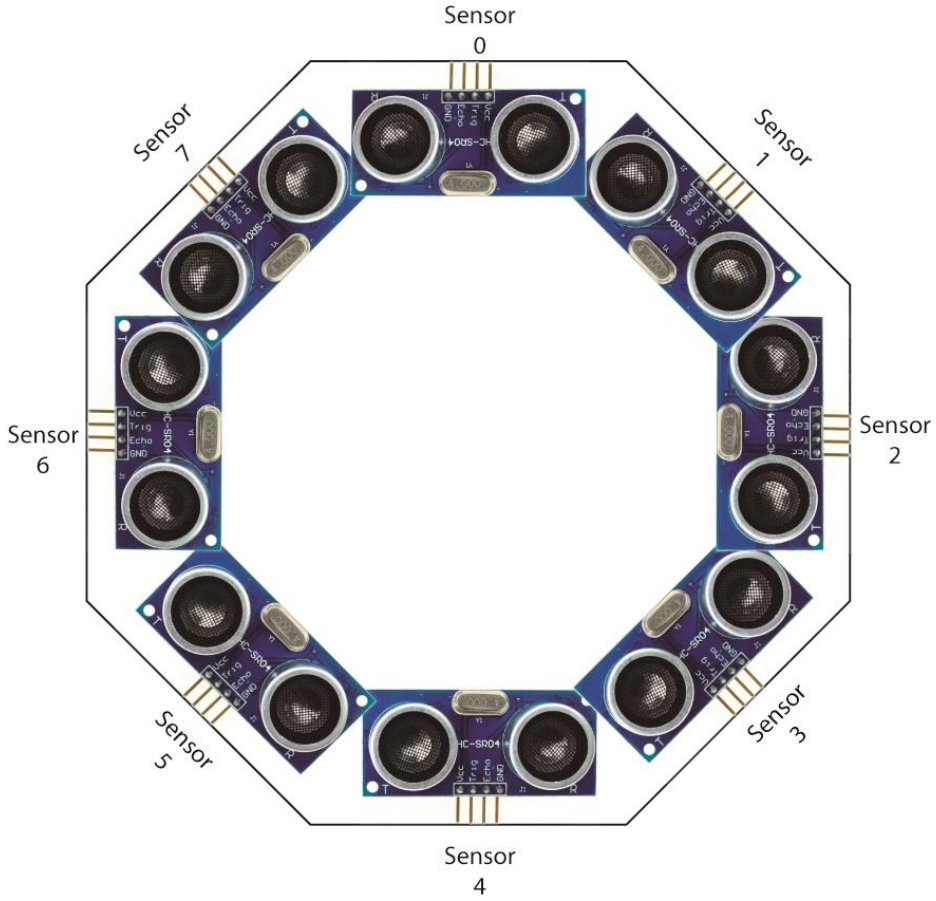

Figura 5: Disposição dos oito sensores usados para detecção de obstáculos

Primeiramente, foi especulado que tal disposição poderia gerar interferência lateral entre sensores adjacentes. O ângulo de medida[3] de cada sensor é de 15 graus [7] e, sabendo que o ângulo externo entre as faces do octógono (Figura 5) é de 225 graus, é fácil perceber que há um ângulo relativo de 105 graus entre as raias do ângulo de propagação de onda de dois sensores adjacentes. Como esse ângulo é maior que 90 graus, é pouco provável que haja interferência. Por vias experimentais foi percebido que, de fato, não há interferência relevante.

Outra preocupação era gerada pelo trabalho anterior. Nele foi afirmado que a alimentação via conector *Jack* causava perturbação nas medidas dos sensores. Contudo, apesar de possível, tal causalidade parecia pouco provável. Por meio de ensaios, verificou-se que havia muitos fatores que causavam erros e perturbações nas medidas, sendo impossível atribuí-los a apenas uma causa de forma isolada. A fim de minimizar esses erros, cada componente foi lavado usando-se um produto a base de solventes destilados de petróleo e propelente. Para descartar possíveis erros de comunicação entre a placa e os sensores, foram interconectadas todas as entradas *trigger*, todas as Vcc e todas as GND, com redundâncias sempre que possível. Assim, as leituras se estabilizaram e não se percebeu qualquer instabilidade nelas. Apesar do código separar os disparos de cada sensor, como as portas estão interconectadas, todas as entradas *Trigger* dos sensores são ativadas ao mesmo tempo. Já a resposta *Echo* é detectada via interrupção externa.

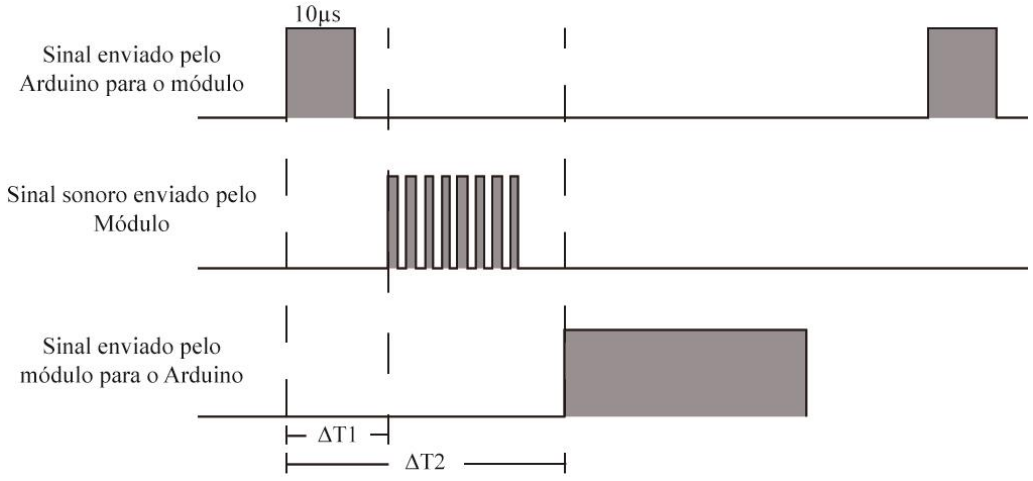

Figura 6: Diagrama de tempo referente ao funcionamento do módulo HC-SR04.

Para transformar o atraso do *echo* em distância, usa-se a velocidade de propagação do som. Assim, bastaria dividir esse tempo por uma constante. Porém, para isso é necessário saber a diferença entre ΔT2 e ΔT1 (como mostrado na Figura 6). A fim de estimar experimentalmente o valor dessa constante e do tempo ΔT1, foram realizados pequenos ensaios (Tabela 1). O procedimento consistiu em realizar contagens de tempo para distâncias já conhecidas. Foram feitas quatro medidas com cada um dos oito sensores. O tempo médio foi calculado a partir dessas 24 amostras por sensor. Foram realizados

[3] Ângulo de propagação do sinal no espaço

testes para as distâncias de 5, 10, 15, 20 e 25 centímetros. Usando-se o tempo médio, ΔT1 foi estimado empiricamente. Sabendo que a constante é dada pela Equação 1, foram tentados valores para ΔT1 que minimizassem o erro quadrático médio da constante.

$$Constante = \frac{Tempo\ médio - \Delta T1}{Distância}$$

Equação 1

Tabela 1:Resumo dos ensaios realizados para aferir as constantes e a distância.

Distância (cm)	Tempo Médio (µs)	ΔT1	Constante
5	690,00	404	57,20
10	987,50		58,35
15	1263,33		57,29
20	1553,33		57,47
25	1833,33		57,17
	Média		57,50

Assim, para se aferir a distância, basta usar a Equação 2 que foi usada no código que será abordado na seção 4 Programação, página 31.

$$Distância\ (cm) = \frac{\Delta T2 - 404}{57{,}5}$$

Equação 2

2.2.2 Giro e Fuga

O ângulo de fuga e a distância do obstáculo, calculados na etapa de detecção, são valores utilizados pelo Arduino para determinar o tempo de giro e a velocidade de acionamento do motor. Esse cálculo de tempo é baseado no ângulo relativo entre o sensor 4 e o objeto. Como foram utilizados oito sensores, criaram-se movimentos de giro múltiplos de 45º (divisão de 360 por 8). Quando os sensores 3 ou 5 acusam um objeto, a plataforma precisa girar 45º sentido anti-horário ou horário, respectivamente. Na secção 4.3 Cálculo de distâncias e ângulo de fuga, página 34, há uma tabela (Tabela 5) que mostra esses valores para cada sensor. Ainda na mesma seção podem ser vistos, na Tabela 6, os tempos de ativação das rodas para alcançar cada um desses giros. Esses valores foram tomados empiricamente.

Outra modificação interessante foi o desacoplamento das rotinas de giro e de fuga. Primeiramente, a plataforma realiza apenas giros até que o sensor 4 acuse o objeto. Após tê-lo feito, parte-se então para a fuga. Baseado na leitura de distância do objeto acusado no sensor 4, a plataforma aciona os motores com velocidade constante. A qualquer momento o processo pode ser interrompido pela detecção de um novo objeto, e a plataforma volta para o estado de giro ou, caso não seja acusado nenhum objeto, fica parada.

3. Hardware

Entre os componentes mecânicos do projeto, destaca-se o chassi, que é o esqueleto físico do projeto. Entre os eletrônicos, há a placa Arduino Due, a *motor shield* versão 2, os sensores de ultrassom, os motores que movimentam a plataforma e a alimentação do sistema. Uma descrição mais detalhada de cada um destes componentes será feita neste capítulo.

3.1. Plataforma

Todas as plataformas são feitas de acrílico e presas entre si por meio de parafusos. As duas plataformas originais foram herdadas do projeto anterior. Além disso, foi adicionada uma terceira, formando mais um andar, como mostrado na Figura 7. O objetivo é abrir espaço para as baterias que alimentam a *motor shield*, além de dividir melhor cada categoria de componentes, ficando da seguinte maneira:

- 1° andar para os motores, que, para dar mais estabilidade, são presos na plataforma com parafusos;
- 2° andar para as baterias. As barcas das baterias são fixadas com fita dupla face.
- 3° andar para os sensores e Arduino. Os sensores estão conectados a oito *protoboards*, que, por sua vez, estão fixadas na plataforma com fita dupla face. O Arduino fica na posição vertical para facilitar a conexão do cabo USB.

Figura 7: Plataforma atual.

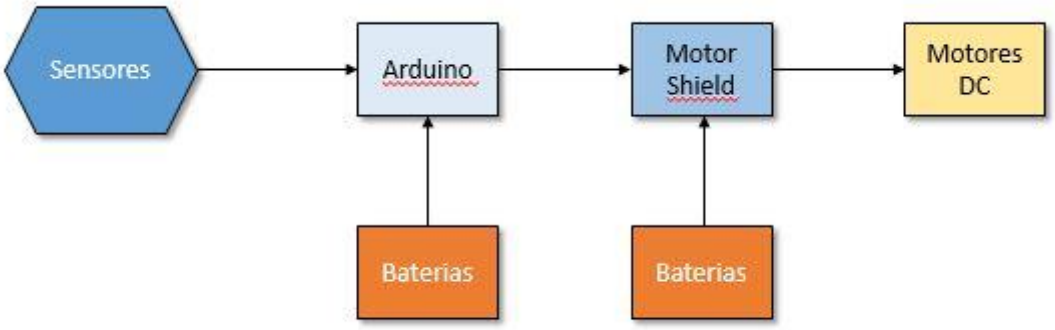

Figura 8: Diagrama de blocos da plataforma.

A Figura 8 ilustra as conexões de cada componente dispostos nas plataformas. A placa Arduino é o componente central, sendo que todos os outros conectam-se a ela de maneira direta. A alimentação foi feita com seis baterias, sendo quatro exclusivamente para a *motor shield* e as outras duas para o Arduino e, consequentemente, para sensores. O motivo dessa separação será abordado na seção 3.7 Baterias, página 27. A *motor shield* está conectada aos motores DC e é a responsável por controlá-los de acordo com o comando do Arduino [8].

Como elucidado na seção 2.2.1 Detecção de obstáculos, na página 9, foram adotadas soluções para minimizar maus contatos. Podia ter sido implementada também uma redundância na porta *echo* dos sensores ultrassônicos, porém isso acarretaria a duplicação das interrupções externas e poderia comprometer parte da capacidade de processamento.

A retirada da placa de conexões[4] culminou em leituras mais concisas. E, além disso, as redundâncias diminuíram a resistência elétrica do circuito. A tensão nas portas de alimentação dos sensores ficou mais alta e o sistema pôde ser melhor alimentado.

Usando uma folha de papel, foi criada uma barreira física para impedir que os fios interferissem na leitura dos sensores. Isso foi necessário porque a quantidade de fios aumentou consideravelmente com as redundâncias.

3.2. Arduino

O projeto Arduino se iniciou em 2005 e em 2006 já recebeu menção honrosa na categoria Comunidades Digitais pela Prix Ars Eletronica. Tal sucesso só foi possível devido ao

[4] Essa placa foi utilizada pelo projeto anterior, mas foi retirada no projeto atual. Ela ligava os sensores à placa Arduino.

altíssimo nível de aceitação do mercado devido ao fato de a placa ser *open source*. Ou seja, apesar de haver um produto final elaborado diretamente pela Arduino, qualquer empresa que se interessar pode lançar soluções idênticas. Além dessa possibilidade, quem quiser pode também se especializar em modificações da placa ou produção de adornos que a tornam muito versátil. São diversos módulos e *shields*[5] que possibilitam ao usuário encontrar soluções simples, estáveis e acessíveis.

Em especial o Due é uma placa da empresa italiana Arduino. É a primeira placa baseada na tecnologia ARM SAM3X8E. Assim como todas as outras placas Arduino, essa também utiliza um ambiente de desenvolvimento próprio chamada IDE, baseada em uma linguagem C adaptada e com suporte à linguagem C++. O ponto máximo de destaque dessa IDE é a facilidade de usar funções oriundas da comunidade Arduino.cc (assim como de qualquer comunidade). O apoio dessa comunidade é gratuito e muito utilizado pela maioria dos usuários. Além do compartilhamento de funções, o usuário encontra uma espécie de suporte colaborativo por meio de fóruns.

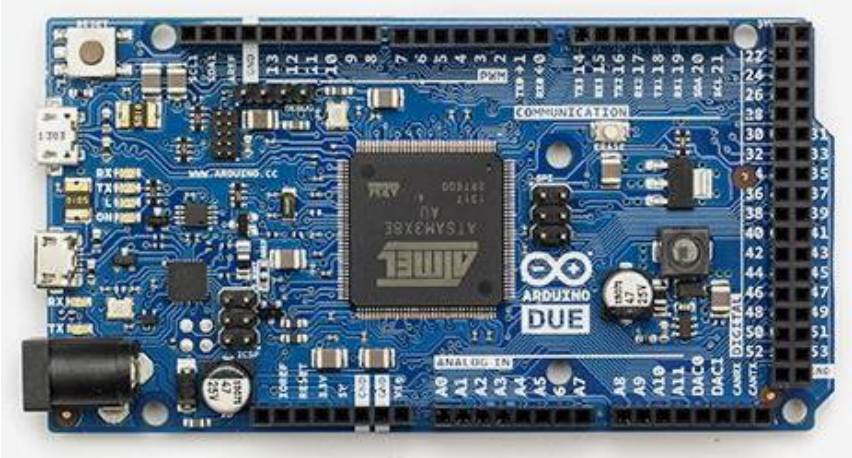

Figura 9: Arduino Due. [10]

O processador ARM é capaz de sincronizar até oito temporizadores com frequências de até 84 MHz. Esse *clock* da CPU permite que rotinas relativamente extensas e complexas possam ser temporizadas em intervalos pequenos de tempo. Já sua abundância de contadores permite que sejam feitas mais interrupções temporais independentes. Ainda oferece capacidade de priorizar essas temporizações. Quando há duas ou mais interrupções temporais no mesmo instante, a placa dá preferência para o temporizador de menor número de identificação (vide a primeira coluna da Tabela 2).

[5] São placas que se encaixam ao Arduino para promover novas funcionalidades.

Tabela 2: Temporizadores disponíveis no Arduino Due.

Temporizador	TC	Canal	NVIC "irq"	IRQ *handler function*	PMC id
0	TC0	0	TC0_IRQn	TC0_*Handler*	ID_TC0
1	TC0	1	TC1_IRQn	TC1_*Handler*	ID_TC1
2	TC0	2	TC2_IRQn	TC2_*Handler*	ID_TC2
3	TC1	0	TC3_IRQn	TC3_*Handler*	ID_TC3
4	TC1	1	TC4_IRQn	TC4_*Handler*	ID_TC4
5	TC1	2	TC5_IRQn	TC5_*Handler*	ID_TC5
6	TC2	0	TC6_IRQn	TC6_*Handler*	ID_TC6
7	TC2	1	TC7_IRQn	TC7_*Handler*	ID_TC7
8	TC2	2	TC8_IRQn	TC8_*Handler*	ID_TC8

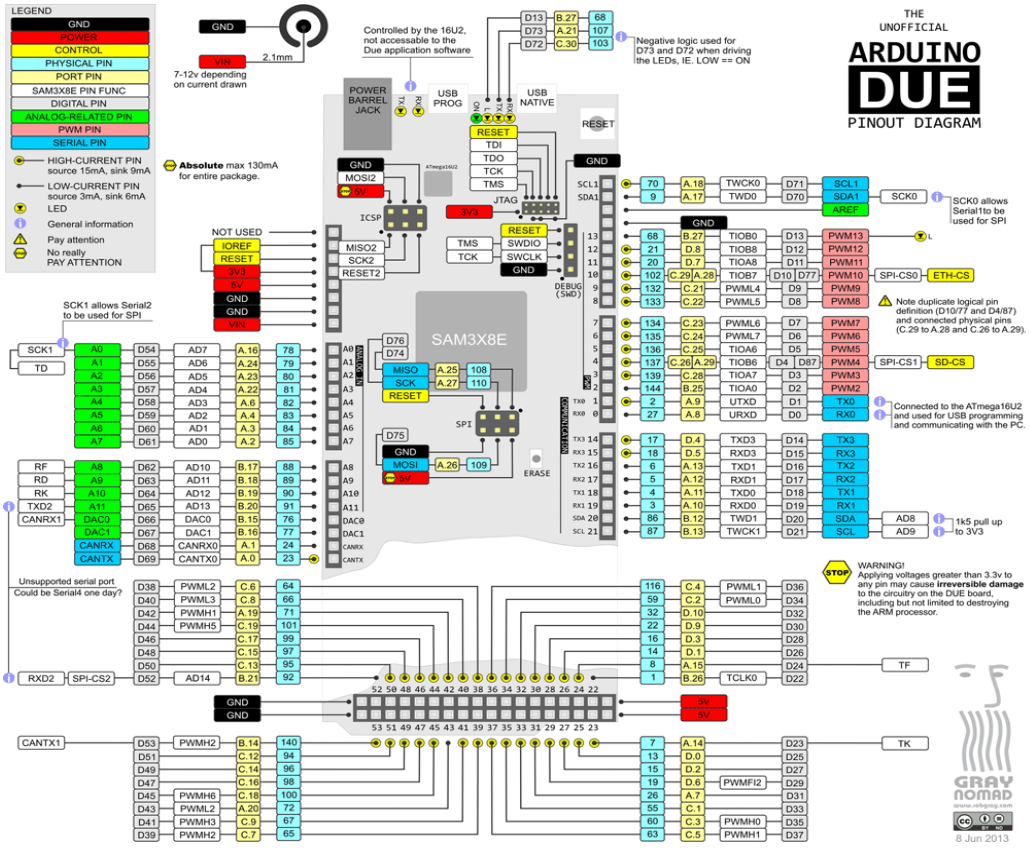

Figura 10: Mapa de pinos do Arduino DUE. [10]

3.3. Sensor ultrassom

Optou-se pela implementação do esquema de detecção de obstáculos utilizando o sensor HC-SR04. O funcionamento desse dispositivo se baseia em um mecanismo natural sofisticado. De forma similar, diversos mamíferos como morcegos, golfinhos e baleias, utilizam a ecolocalização por meio de biosonar que permite ao animal desviar-se de obstáculos sem utilizar sua visão propriamente dita. O morcego, por exemplo, emite um som com duração entre 25 e 90 milissegundos a uma frequência de 15 KHz a 72 KHz e, baseado no tempo de espera para captar a reflexão da onda no objeto, consegue determinar sua distância [8]. No caso do sensor HC-SR04, seu funcionamento se baseia no mesmo princípio, porém emite um trem de oito pulsos (duração total de 0,4 milissegundos) e opera na frequência de 40 KHz [7].

Este dispositivo foi utilizado desde o primeiro protótipo em 2014. O principal motivo da manutenção desse dispositivo foi sua alta eficiência energética: consome no máximo 15 mA [7]. Outro fator muito relevante foi seu baixo preço, pois torna fácil e acessível a troca de algum componente danificado.

A Figura 11 mostra os diversos dispositivos que compõem o sensor. Para melhor compreensão desse esquemático, seguem as descrições funcionais dos blocos:

a) **EM78P153N/S/SP/J**: Trata-se de um microprocessador CMOS de alta velocidade e baixo consumo de energia. Possui função de temporização, no caso, por meio de um *clock* externo acoplado às portas P64 e P65. O pino P67 é usado para despertar o dispositivo que entra em estado de hibernação quando não é ativo. A porta P50 está sendo usada como saída. Os pinos P51, P52 e P53 são pinos de entrada ou saída multipropósito. Já o pino P60 é de interrupção externa por borda de descida, enquanto que o pino P61 funciona como saída [9].
b) **MAX232:** É um dispositivo duplo que atua como conversor. É a união entre quatro dispositivos: dois *drivers* e dois receptores. Cada *driver* converte os níveis de entrada TTL/CMOS em níveis TIA/EIA-232-F, enquanto que os receptores fazem o inverso. Como se pode notar na Figura 11, as entradas C1+, C1-, C2+, C2-, VS+ e VS- estão conectadas de forma típica para esse dispositivo. Assim, as entradas utilizadas T1 e T2 são conectadas como *drivers*, ou seja, recebem padrões de tensão TTL/CMOS e exibem nas saídas T1 e T2 seus respectivos níveis TIA/EIA-232-F. Outra peculiaridade é a ligação VCC desse dispositivo, que está ligada a um transistor e, por isso, só há alimentação quando a saída do pino P53 do EM78P153N/S/SP/J alcança nível suficiente. [10]
c) **TL074**: Esse dispositivo é a junção de quatro amplificadores operacionais, sendo as portas IN+ e IN- entradas positivas e negativas e a saída amplificada OUT [11].

Assim, quando um sinal de 10 µs atinge a porta *trigger*, o dispositivo EM78P153N/S/SP/J imprime em suas portas um pulso de sinais gerado pelo temporizador externo (cristal de 4MHz). Na saída dessas portas, o dispositivo MAX232 converte esse sinal e o exprime na saída de som. Como pode ser visto na Figura 11, há dois dispositivos de som. Como foi dito, o primeiro atua como saída de som, enquanto que está abaixo desse atua como microfone. Assim, quando recebido de volta, o sinal é amplificado pelo TL074 que, por sua vez, o emite de volta para o EM78P153N/S/SP/J, que, por meio de laços temporais, abaixa o nível de tensão na porta *echo.*

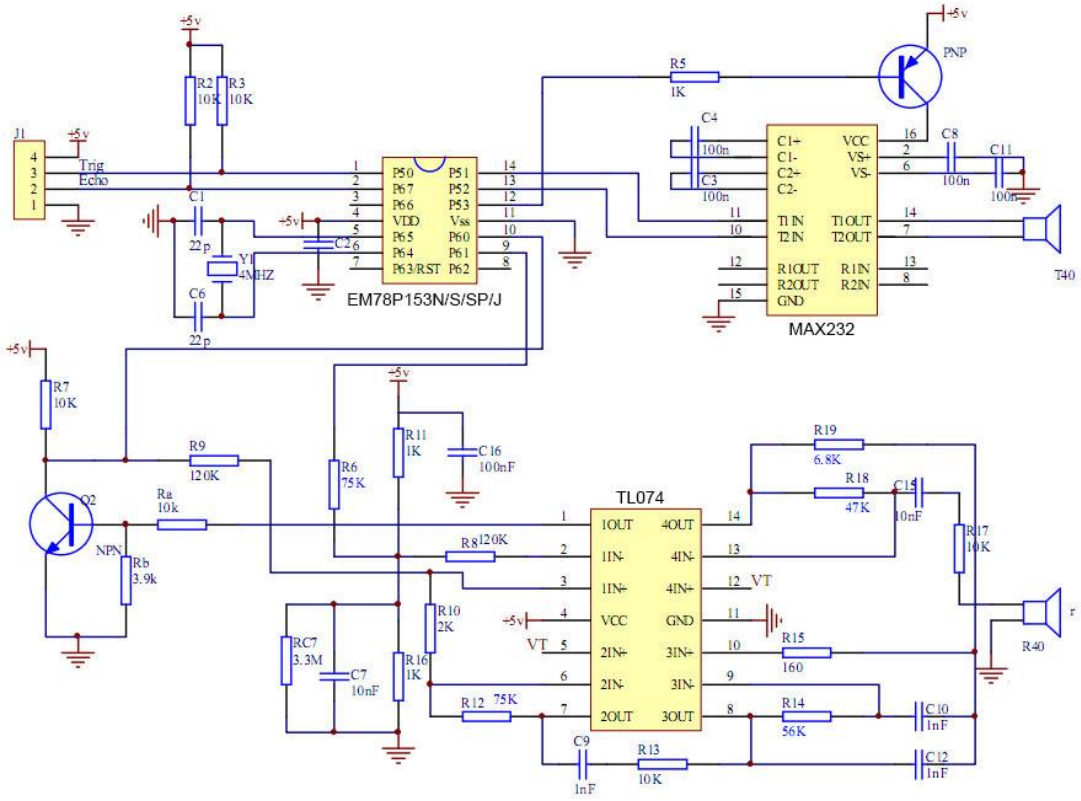

Figura 11: Esquemático do funcionamento do HC-SR04 [15].

É fácil perceber que em nenhum momento o dispositivo emitiu qualquer código que aferisse tempo ou espaço. Ele apenas abaixa o nível da porta *echo* quando detecta a entrada do som. Então é responsabilidade de outros dispositivos contar o tempo entre esses disparos para que se possa aferir distância.

3.4. Arduino *Motor Shield* V2

A Adafruit, empresa responsável pela criação da *shield*, fornece em seu site [13] a biblioteca com os comandos disponíveis para o controle dos motores. Foi realizada uma análise nessa biblioteca em busca de alguma função que utilizasse temporização no *timer* 0. Como nada foi encontrado, decidiu-se que seria seguro usá-la nas escolhas das velocidades e acionamentos dos motores.

Uma pequena alteração foi necessária na shield para usá-la com o Arduino Due. Já que o mesmo opera em 3,3V, em vez de 5V como em outros Arduinos. Conforme orientado no manual do usuário, deve-se encontrar um pequeno agrupamento de *pads*, identificado como *logic*, cortar a ligação entre o *pad* central e o da direita, identificado como 5V, e, por fim, com o auxílio de uma gota de solda refazer a conexão do *pad* central com o da esquerda, identificado como 3,3V. A figura 12 mostra de perto o local onde este procedimento foi realizado.

Para estipular a velocidade foi utilizada a função *setSpeed*(). Com ela a própria placa gera o sinal PWM, mencionado na seção 3.4.1 Controle de Velocidade (PWM) (página 22). A entrada dessa função pode variar em uma escala de 0 a 255. O valor escolhido no projeto foi de 100 e apesar de ser ligeiramente maior, comparado ao projeto anterior, ele foi necessário considerando que a plataforma está mais pesada.

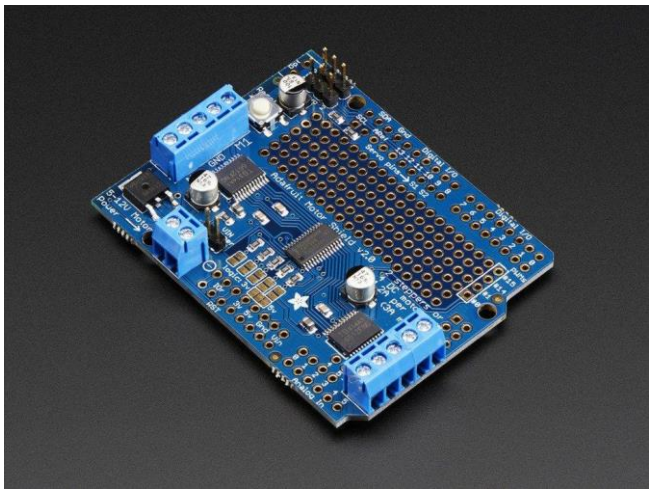

Figura 12: Adafruit Motor Shield V2 [17].

O acionamento dos motores é feito com a função *run()*. Ela dispõe de 3 comandos: *FORWARD* para acionar os motores para frente, *BACKWARD* para trás e *RELEASE* que interrompe o funcionamento dos motores. Ressalva-se que este último comando não configura um freio, mas sim a interrupção do funcionamento dos motores. Por isso, a plataforma tende a não parar imediatamente devido à sua inércia.

Existem oito rotinas para o acionamento dos motores, uma para cada sensor. Elas se diferenciam tanto no sentido de rotação dos motores quanto no intervalo de tempo que cada uma permanece em funcionamento. O programa principal é o responsável por checar as medidas de todos os sensores e indicar qual das rotinas melhor atende para a fuga daquele obstáculo. As funções dos acionamentos dos motores são: frente*()*, *turn_AH()*, *turn_SH()* e parar().

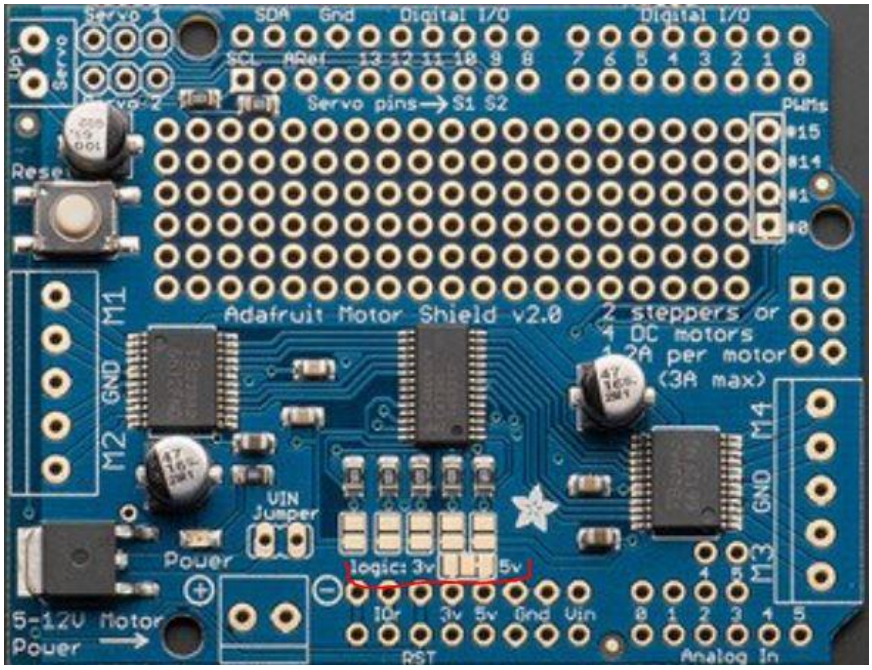

Figura 13: Detalhe da lógica em 3.3V e jumper de alimentação [17].

3.4.1 Controle de Velocidade (PWM)

O controle de velocidade do motor DC pode ser feito de diversas maneiras: uma delas é por meio de um reostato em série com a alimentação do motor; dessa forma, é possível alterar a corrente de armadura por meio da variação da resistência. Esse método apresenta algumas desvantagens – a principal é a dificuldade de manter a velocidade constante quando o motor está em baixa rotação.

Sabendo que a velocidade de um motor CC é proporcional à tensão quadrática média aplicada em seus terminais, uma segunda maneira, muito eficiente, é o uso de sinais PWM, *Pulse Width Modulation*. A ideia deste método consiste em trabalhar com o tempo como sendo uma variável de controle. A própria *shield* gera sinais de onda retangular cuja tensão máxima é a tensão nominal do motor, porém com tempo de ciclo reduzido. A Figura 14 ilustra o sinal enviado pela *shield*. Se, por exemplo, o sinal tiver um ciclo de 50%, a tensão quadrática média nos terminais do motor reduzirá pela metade, assim como sua velocidade. Em suma, esse método permite o controle preciso da velocidade do motor, além da manutenção de um torque próximo do valor máximo.

A biblioteca da *shield* já fornece a função *setSpeed*(), que gera o sinal. O argumento dessa função é a velocidade, que assume valores entre 0 e 255.

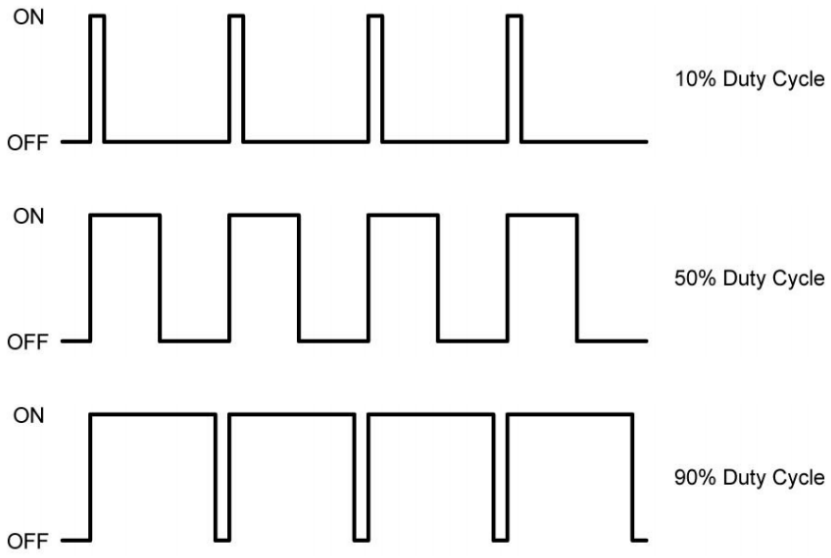

Figura 14: Exemplo de onda PWM com 10%, 50% 2 90% de ciclo de trabalho [18].

3.4.2 I²C

Em 1982, a Phillips criou esse protocolo de comunicação como um simples sistema de barramento interno para construir controles eletrônicos com vários chips. Desde então, foram lançadas seis versões. A quarta foi a última em que houve aumento de velocidade. Assim as versões 4, 5 e 6 também trabalham em até 5 MHz. Desde sua primeira versão, utilizam-se apenas duas linhas bidirecionais de dreno aberto[6], uma de dados seriais (*Serial Data* – SDA) e outra de *clock* serial (*Serial Clock* – SCL) [14].

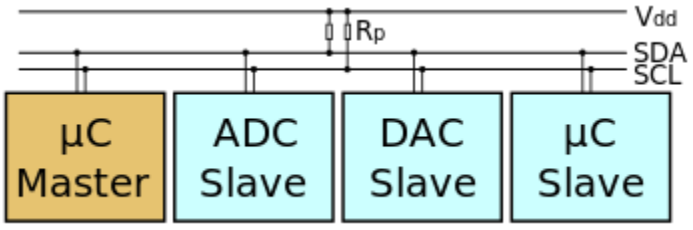

Figura 15: Linhas usadas pelo protocolo [20].

Como pode ser visto na Figura 15, esse barramento é composto por dois fios. O fio Vdd tem a função de alimentar o esquema de pull-up. Esse protocolo permite que uma unidade mestre (Master na Figura 15) controle os periféricos (*Slave* na Figura 15). Quando ambas linhas apresentam valor alto, diz-se que o barramento está no estado neutro. A escrita de dados no barramento é permitida pela oscilação

[6] Dreno aberto é um dos muitos tipos de padrões de entrada e saída usados em projetos digitais. Esta característica permite que vários dispositivos compartilhem o mesmo fio para transmitirem um nível lógico para outro dispositivo, ou combinar efetivamente as saídas dos dispositivos em uma função lógica AND.

do barramento SCL (Figura 16). A cada pulso o valor em SDA é lido como um bit começando pelo MSB[7].

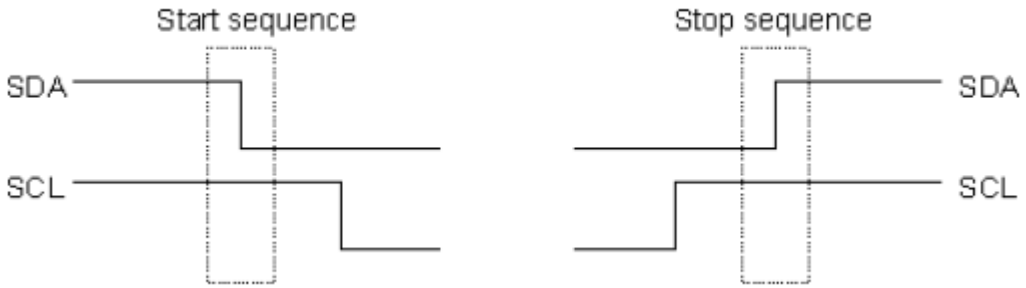

Figura 16: Níveis de SDA e SCL no início e no fim da comunicação [20]

Logo após SDA alcançar nível baixo, o mestre escreve o endereço do dispositivo que deseja se comunicar. Instantes após esse ocorrido, começa a transferência de dados. O mestre escreve o endereço do registrador do escravo que ele deseja ler ou escrever. Então, a operação começa possibilitando ao mestre ler ou escrever em mais de um registrador.

Para que um dispositivo mestre comunique com o Arduino (escravo) é necessário que se sincronize ambos. Essa sincronização só pode ser realizada com sucesso quando o barramento está ocioso, ou seja, não está sendo usado para comunicação. Quando isso acontece, é gerada a condição de início da Figura 16. O Arduino (escravo) aguarda as instruções do mestre. Ao fim da comunicação, este força o barramento para a situação de parada da Figura 16.

3.5. Módulo *Bluetooth*

O módulo *Bluetooth* da versão anterior foi trocado. Isso ocorreu por mero acaso e não trouxe nenhuma modificação prática ao projeto. Antes usava-se o modelo HC-05 e agora o HC-06. Apesar dos números de modelo indicarem o contrário, o módulo HC-05 é superior ao HC-06. Isso porque ele suporta dois tipos de configuração: pode ser usado como escravo ou como mestre. No caso deste projeto, o módulo é usado exclusivamente como escravo e, por isso, não é necessária a função extra do HC-05.

[7] Sigla para Most Significant Bit. No caso é o primeiro bit de uma sequência binária.

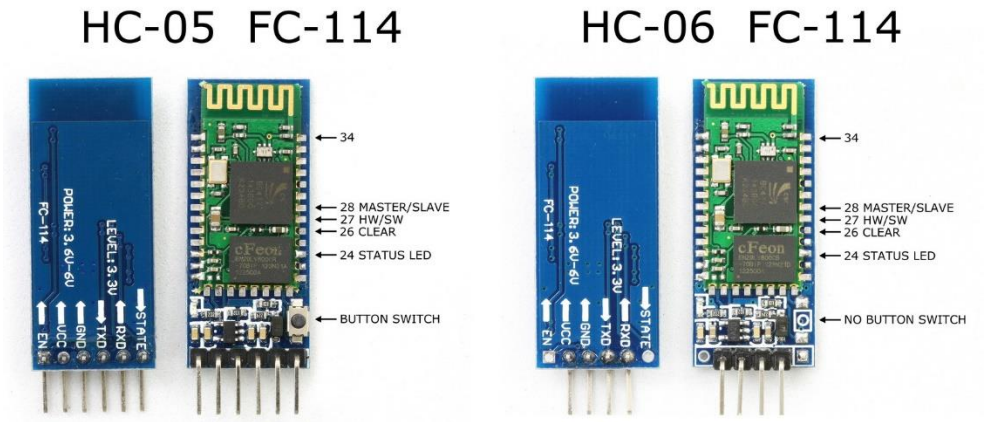

Figura 17: Módulo antigo (HC-05) e módulo novo (HC-06) [21].

Os pinos Vcc e GND são de alimentação enquanto que RXD e TXD são de envio e recebimento respectivamente. A conexão é feita de forma que o pino TXD da placa é conectado ao RXD do módulo, e o RXD da placa é conectado ao TXD do módulo. O Arduino possui 4 canais de comunicação serial, o canal TX0 e RX0 formam o primeiro canal enquanto que o TX3 e RX3 formam o último. O referido *Motor Shield* V2 utiliza o primeiro canal e, dado que o canal 3 está livre, este foi escolhido para a conexão do módulo.

3.6. Motores DC

Motores são máquinas que realizam a conversão de uma forma de energia em energia mecânica. No caso do motor de corrente contínua, ele converte energia elétrica em mecânica. [15]

O motor DC consiste em duas partes: o estator, parte externa e fixa do motor, onde se encontra o enrolamento de campo; e o rotor, parte móvel que contém o enrolamento de armadura. Seu funcionamento é baseado nas leis do eletromagnetismo; de acordo com Sen (1997, p. 125) [16]:

> "Uma corrente contínua corre através do enrolamento de campo para produzir um fluxo [eletromagnético] na máquina. A tensão induzida no enrolamento de armadura é alternada. Um comutador mecânico e um sistema de escovas funcionam como um retificador e inversor, tornando unidirecional a tensão terminal na armadura."

O projeto utiliza esses motores para mover as quatro rodas da plataforma, de forma individual. O conjunto de motores permite que a plataforma gire em torno do próprio eixo por meio da rotação de pequenas rodas conectadas diretamente no eixo e, assim, desvie dos obstáculos. Os motores são alimentados pelo *driver* presente na *motor shield*, conforme explicado na seção 3.4.

Figura 18: Motor DC utilizado no projeto [24].

Os motores são dispostos na plataforma conforme a Figura 19. A numeração dos motores é feita desta forma para facilitar a escrita do código.

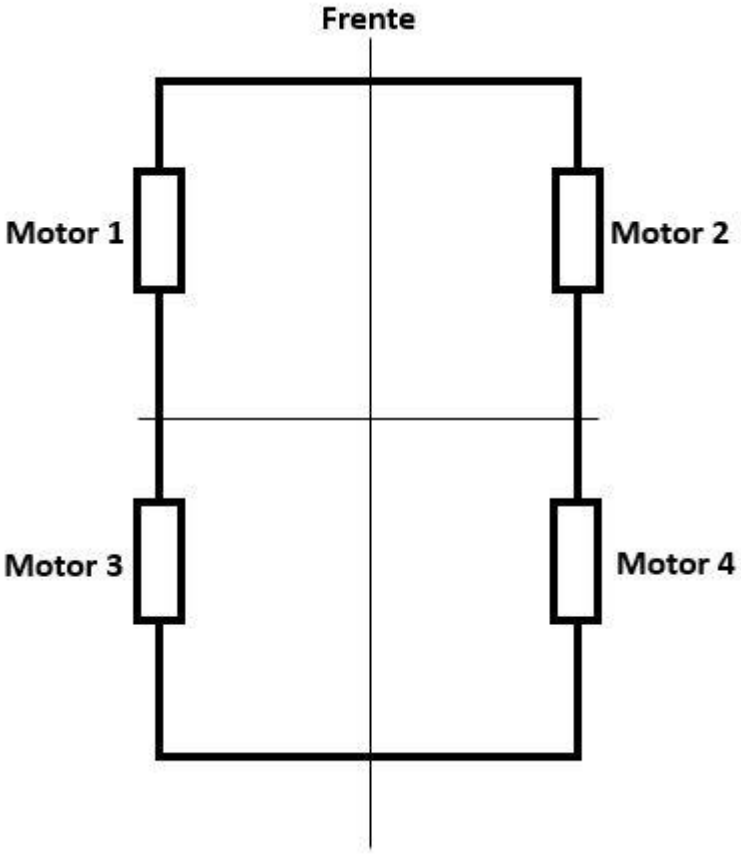

Figura 19: Disposição dos motores na plataforma.

3.7. Baterias

O sistema completo é alimentado por seis baterias recarregáveis de íon-lítio, modelo 18650, de 3,7V cada, fabricadas pela Ultrafire. Por questões de desempenho fez-se necessário separar as alimentações em dois subsistemas, a placa Arduino e a *motor shield*. Isso foi motivado pelas muitas falhas ocorridas quando a alimentação era única para todo o sistema.

Dessa forma, a alimentação ficou dividida assim:

- Arduino: 2 pilhas de 6000mAh cada;
- *Motor Shield*: 4 pilhas de 12000mAh cada.

A alimentação do Arduino consiste em duas baterias em série, fornecendo um total de 7,4V e 6000 mAh. Ao passo que a da placa shield consiste em baterias colocadas em série, duas a duas; depois, os pares são colocados em paralelo, a fim de fornecer os mesmos 7,4V. Como o motor requer aproximadamente 7,5V, este arranjo fornece a tensão necessária para o funcionamento do mesmo. O circuito da alimentação de ambos os subsistemas está ilustrado nas figuras 19 e 20.

.

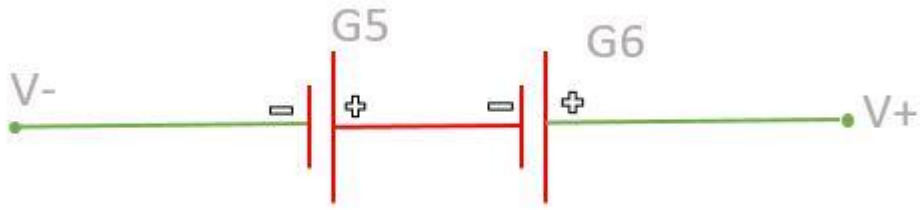

Figura 20: Sistema de carregamento das pilhas e alimentação do Arduino.

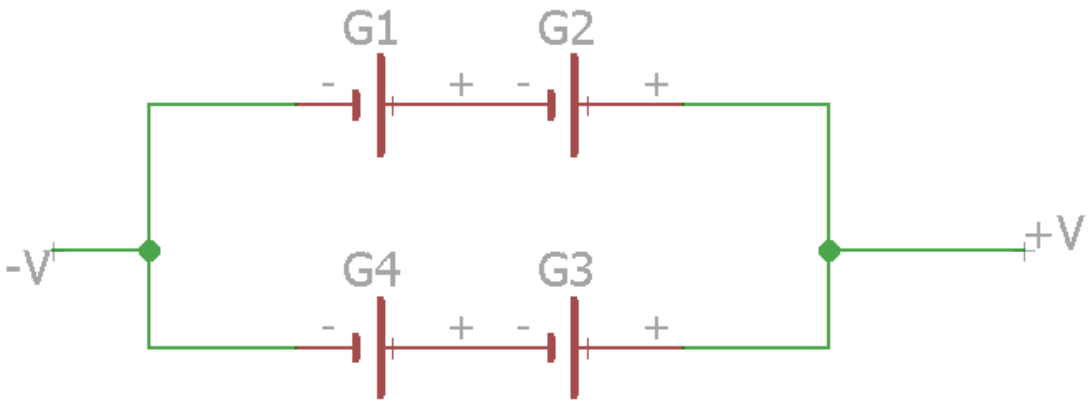

Figura 21: Sistema de carregamento das pilhas e alimentação do motor shield.

3.7.1 Chaveamento

Para sanar os problemas relacionados ao carregamento, foi decidido embarcar três carregadores. Esses carregadores foram modificados para atuar em dois estados:

- Estado 1: Bateria sendo carregada na rede elétrica;
- Estado 2: Bateria fornecendo energia aos componentes.

Figura 22: Chave seletora 4P8T.

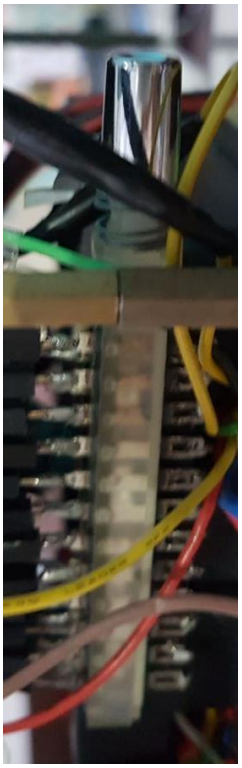

Figura 23: Chave seletora 8P16T.

Essa mudança ocorreu com o auxílio das chaves 4P8T e 8P16T (Figura 22 e Figura 23, respectivamente) das barcas de carregamento e de cabos tipo *jumpers*. Primeiramente, foi necessário abrir a barca, reconhecer os polos negativo e positivo de cada bateria e refazer as conexões, de modo a atender às funções de cada estado. Estas conexões estão ilustradas na Figura 25 e na 25, em que as linhas azuis representam o Estado 1 e as linhas vermelhas o Estado 2.

Figura 24: Barca original adaptada no projeto para oferecer carregamento embarcado [25].

As chaves 4P8T e 8P16T recebem este nome em referência ao número de polos e terminais de cada chave. Elas funcionam da seguinte maneira: os polos da chave alternam o contato, ora com os terminais da esquerda, ora com os da direita, a depender da posição em que botão se encontra. O mesmo raciocínio foi feito com o chaveamento da *shield* dos motores.

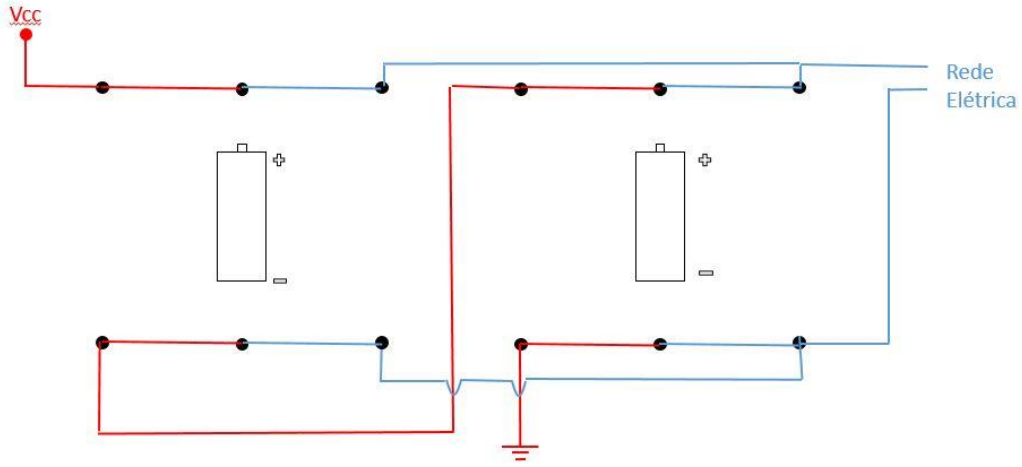

Figura 25: Esquema de ligação das pilhas que alimentam o Arduino.

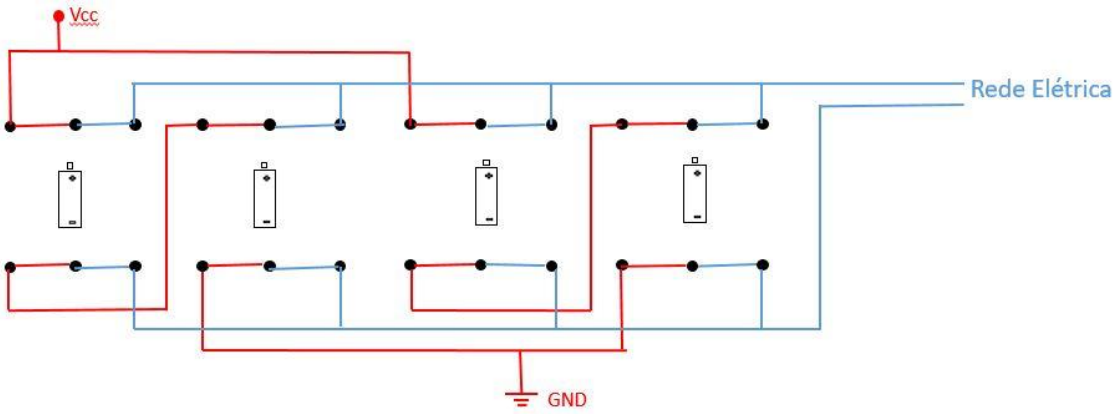

Figura 26: Esquema de ligação das pilhas que alimentam a Shield.

4. Programação

O programa está dividido em diversas funções, cada uma desempenhando um papel dentro do processo. Entre elas, destaca-se o *firmware*, as interrupções, o algoritmo de cálculo de distância e o algoritmo de controle dos motores.

4.1. *Firmware*

A proposta inicial deste projeto era que se mantivesse a mesma placa Arduino Due do projeto anterior, solucionando o problema da trepidação que a plataforma sofria durante o giro. Portanto, algumas mudanças foram necessárias, como, por exemplo, o pulso de 10 µs no *trigger*. O que antes era feito com a função *delayMicrosenconds()* agora é executado por meio de interrupções e do temporizador 7.

Uma prática importante realizada no projeto foi a declaração de algumas variáveis como *volatile*. Essa forma de declaração indica ao compilador para não otimizar a variável. É importante utilizar esse tipo de artifício quando a mesma variável é manipulada dentro e fora de uma interrupção.

Uma vez que durante a etapa de ensaios tornou-se necessário fazer diversas alterações em algumas variáveis, outra prática útil foi o emprego de variáveis parametrizadas: valores como número de sensores, número de amostras para o cálculo da média e distâncias mínimas para que os motores entrem em ação foram declarados como constantes (*#define*) e referenciada no programa pelo o nome designado.

O programa possui três grandes rotinas, dois temporizadores, TC7 e TC8, e o *loop* principal. Todos serão detalhados nas seções 4.3 e 4.4. Em termos gerais, o TC7 é o responsável por gerar os pulsos que acionam os sensores, bem como realizar o cálculo das distâncias, suas médias e ainda descartar as medições discrepantes. O *loop*, por sua vez, recebe estes valores calculados e gera um vetor que contém as medidas das distâncias de cada sensor. Em seguida, esses valores serão testados e, caso um deles resulte menor do que o estipulado, ainda dentro do *loop* é feito o reconhecimento de qual sensor realizou essa medida, levantam-se as *flags* referentes e, por fim, chama-se a função de acionamento dos motores correspondente.

O TC8 tem a tarefa de checar qual a *flag* que foi levantada no *loop* para, em seguida, realizar a contagem de tempo que os motores devem ficar acionados e, por fim, cessar seu funcionamento e zerar a *flag*.

A Figura 27 mostra o fluxograma do programa.

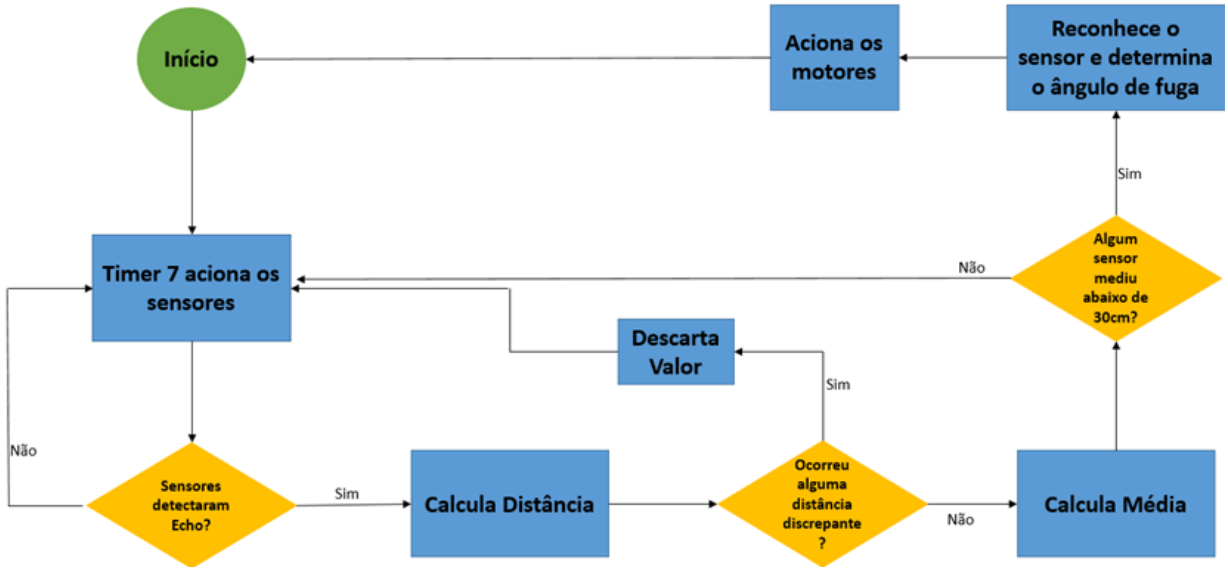

Figura 27: Fluxograma da lógica do programa.

4.2. Interrupções

A solução mais indicada para o controle temporal é a interrupção. Normalmente, o sistema é programado para realizar uma tarefa básica e contínua que pode ser interrompida para que o programa acesse alguma outra tarefa específica. Os tipos mais básicos de interrupções são as temporais e as externas. Por exemplo, o funcionamento de um relógio se dá por interrupções temporais que o fazem avançar na contagem de tempo, mas, quando um botão gera uma interrupção externa, essa contagem para e o usuário pode ajustar o horário atual. O Arduino Due apresenta até 67 tipos de interrupções:

- Até 54 externas
- 9 Temporizadores
- 4 Seriais

4.2.1 Interrupções externas

As interrupções externas são causadas por variações de tensão nas portas do Arduino. Existe a possibilidade de a detecção da variação ser realizada de cinco maneiras distintas:

- **Borda de subida:** Quando a porta está em qualquer estado, não há interrupção no programa. Porém, se houver passagem entre os níveis baixo e alto, a interrupção é ativada.
- **Borda de descida:** Análogo à borda de subida, só há interrupção quando a placa detecta passagem de estado da porta de alto para baixo.
- **Borda de descida e subida:** É a união entre os primeiros. Interrompe quando há mudança de estado. Seja de alto para baixo ou vice e versa.

- **Nível baixo:** A interrupção é feita sempre que a porta estiver em nível baixo. Caso a porta perpetue esse nível, a interrupção continuará sendo ativa seguidamente.
- **Nível alto:** Análogo à detecção por nível baixo, ativa interrupção quando detecta nível alto na porta. Ela também é ativa múltiplas vezes quando a porta se mantem em nível alto.

No Arduino Due as interrupções externas são configuradas em duas etapas. Primeiro, declara-se quais pinos serão usados. Isso é feito usando-se a função "*pinMode* (número do pino, *INPUT_PULLUP*)". A seguir, usa-se a função composta "*attachInterrupt* (*digitalPinToInterrupt* (número do pino), função, Tipo de detecção)". No campo função, indica-se o nome da função responsável pela tarefa que a rotina de interrupção guiará. O tipo de detecção é dado por *RISING*, *FALLING*, *CHANGE*, *LOW* e *HIGH* respectivamente associados aos referidos acima.

4.2.2 Interrupções por temporização

A placa possui três canais de temporização (TC Channel) que podem ser programados independentemente. Cada um desses canais possui três temporizadores independentes de 32 *bits* que aceitam mais de 4 milhões de estados. Esses contadores podem ser incrementados por *clock* interno ou externo. Esses *clocks* podem oferecer até cinco frequências diferentes (vide Tabela 3).

Tabela 3: Frequências padrão para os temporizadores.

Nome do tipo de temporizador	Velocidade
TIMER_CLOCK1	42 MHz
TIMER_CLOCK2	10,5 MHz
TIMER_CLOCK3	2,625 MHz
TIMER_CLOCK4	656,25 KHz
TIMER_CLOCK5	44,1 KHz

Foram escolhidos os temporizadores 7 e 8 da Tabela 2 seção 3.2 página 16. Essa escolha é justificada pelo fato de a placa priorizar os primeiros temporizadores. Assim, os temporizadores[8] de 1 a 6 podem ser usados para implementações mais complexas e urgentes. O temporizador 7 é usado para medir os intervalos de tempo das tarefas associadas a detecção de objeto e ao funcionamento dos sensores. Já o temporizador 8 é usado para ativar os motores. Ambos foram configurados para operar em 10,5 MHz, porém a interrupção do temporizador 7 ocorre ao incremento de 105 batidas enquanto a do 8 requer 10500.

Apesar de ser possível reunir ambas instruções em apenas um temporizador, essa configuração foi implementada para dividir melhor as partes do programa. Outro fator que estimulou foi a abundância de temporizadores.

[8] Cabe ressaltar que o temporizador zero tem seu uso muito restrito e, por isso, foi desconsiderado para implementações futuras.

4.2.3 Interrupção serial

Apesar de a comunicação serial oferecer bastante flexibilidade, sua configuração é muito simples. É feita em duas etapas. Na primeira (configuração), é declarada a função Serial.Begin(frequência[9]). Para que haja comunicação coesa é necessário que ambos dispositivos (transmissor e receptor) estejam na mesma frequência. A etapa seguinte é a comunicação. Ela pode acontecer em qualquer etapa do programa. A Tabela 4 reúne as funções que podem ser usadas nesse caso:

Tabela 4: Funções aceitas na comunicação serial.

Sintaxe	Função
Serial.available()	Retorna a quantidades de *bytes* disponíveis para leitura no *buffer*. Essa função auxilia em *loops* em que a leitura dos dados só é realizada quando há dados disponíveis. A quantidade máxima de bytes no *buffer* é 64.
Serial.print()	Escreve na serial texto em formato ASCII. Essa função tem muitas possibilidades. Números inteiros são escritos usando um caractere ASCII para cada dígito. O mesmo ocorre para números em flutuante e, por padrão, são escritas duas casas decimais. *Bytes* são enviados como caracteres únicos e strings e caracteres são enviados como escritos.
Serial.println()	Funciona praticamente igual a função Serial.print(), a única diferença é que esta função acrescenta ao fim da mensagem o caractere de retorno de carro (ASCII 13 ou '\r') e o caractere de nova linha (ASCII 10 ou '\n'). A sintaxe, os parâmetros e o retorno são os mesmos da função Serial.print().
Serial.write()	Escreve um *byte* na porta serial.
Serial.Event()	É chamada dentro do *loop*(). Conforme surjam novos dados seriais no *buffer* eles são adicionados a uma string até que encontre uma nova linha.

Quando cada uma dessas funções é ativa, a tarefa atual é interrompida. Rapidamente o programa realiza o que a função propõe e volta a sua tarefa anterior.

4.3. Cálculo de distâncias e ângulo de fuga

A distância em si é calculada conforme a Equação 2 da seção 2.2.1 página 9. Depois desse cálculo, o algoritmo segue para o cálculo da média. Para que um novo valor seja inserido nessa média é

[9] Os valores possíveis para essa variável são: 300, 1200, 2400, 9600, 19200, 38400, 57600, 74880, 115200, 230400 ou 250000

necessário que o valor mais antigo seja descartado. Essa substituição é condicionada por quatro fatores justificados abaixo:

- **Ser maior que 64**: Esse é o número de batidas do temporizador de 10 microssegundos, o que indica que a medida é muito pequena para ser levada em consideração.
- **Ser menor que 2330**: A cada nova leitura os registradores do vetor responsável é posto em 9999. Assim uma leitura entre 2330 e 9999 indica que a porta *echo* não imprimiu recebimento da onda enviada pelo sensor ou que fez uma leitura fora de seu valor máximo de quatro metros.
- **A diferença entre as duas últimas leituras deve ser maior que o valor TAM_DIFF**: Uma diferença muito grande entre duas leituras pode indicar uma leitura errada. Esse fator foi adicionado para que leituras espúrias fossem descartadas. Em contrapartida, caso algum objeto se aproxime rapidamente da plataforma, a leitura referente será descartada. Apesar de indesejável, esse fator não compromete o funcionamento da plataforma. Isso ocorre porque a variável TAM_DIFF é obtida empiricamente a partir da velocidade máxima da plataforma. Assim, para o referido caso de um objeto se aproximar velozmente, não haveria benefício em tentar fugir, já que a plataforma não oferece velocidade de fuga suficiente para fazê-lo. Nesse caso, a plataforma aguarda a colisão e depois começa a realizar o giro e, em seguida, a fuga.

O limite para rejeições de leituras para um sensor é TAM_ERROR. Caso haja algum estímulo externo ou quando a plataforma inicia o funcionamento, é possível que o sistema acuse indevidamente que a medida está fora do padrão acima descrito. Caso isso aconteça, um número razoável de vezes (TAM_ERROR) o sistema passa a aceitar esse valor e a média converge para ele após um determinado tempo.

O tamanho do filtro média móvel é dado pela constante TAM_M. Quanto maior esse valor mais o programa demora para convergir para um novo resultado.

Antes de entrar no programa principal foi determinada uma variável referente à distância mínima que um sensor deve medir para que ocorra uma reação nos motores, estabelecida em 25 cm. Além disso, foi declarada uma segunda variável, que corresponde às medidas dos sensores vizinhos, que é de 35 cm.

Em um determinado ciclo, o vetor que carrega as medidas das distâncias é varrido. Para que os motores sejam acionados, é necessário que ambas as condições sejam atendidas:

- O sensor s deve medir uma distância menor do que 25cm e;
- Os sensores s+1 ou s-1 devem medir uma distância menor que 35 cm.

Esses valores se justificam pelo tamanho da plataforma, que tem aproximadamente esse valor como a maior dimensão (entre a frente e a traseira).

Sabendo qual sensor realizou a menor medida, o programa determina em qual rotina de fuga entrar. O objetivo é sempre deixar a plataforma com o sensor 4 apontado para o obstáculo. Portanto, se

o sensor 1 fizer a detecção a plataforma primeiro irá girar 135° no sentido anti-horário, para em seguida se afastar do obstáculo. A Tabela 5 mostra o ângulo de giro para cada um dos sensores.

Tabela 5:Ângulo de giro de cada sensor.

Sensor	Giro	Sentido
0	180°	-
1	135°	AH
2	90°	AH
3	45°	AH
4	0°	SH
5	45°	SH
6	90°	SH
7	135°	SH

O tempo que cada um dos motores permanece em giro, até atingir o ângulo necessário, foi calculado empiricamente. A Tabela 6 e a Tabela 5 apresentam o tempo, em ms, para cada um dos giros. A fuga, então, segue a seguinte sequência: chama-se a função de giro (turn_AH e turn_SH), em seguida uma *flag* é levantada e o *timer* iniciado; a *flag* permanece alta até que o ângulo seja atingido.

Tabela 6: Tempo, em milissegundos, que cada motor funciona para cada rotina de giro.

Ângulo	180°	135°	90°	45°
tempo (ms)	1720	1290	860	430

A velocidade foi determinada com a função *setspeed()*, disponível na biblioteca da *shield*. O valor escolhido foi de 100. Por fim, para auxiliar o giro, foram criadas as funções *turn_AH()* e *turn_SH()*, onde estão estabelecidos os sentidos de rotação de cada motor.

5. Resultados

A fim de demonstrar resultados práticos foi feito um vídeo com a plataforma em pleno funcionamento, este vídeo encontra-se no seguinte link: https://youtu.be/Uvbmr9qimU8

Ressalva-se que devido à retirada do módulo *bluetooth*, que funcionava em comunicação direta com *matlab* no projeto anterior, não foi realizado um ensaio que mostrasse o tempo de reação dos motores para um determinado obstáculo.

Entretanto, para facilitar a visualização do desempenho dos sensores foi proposto um modelo de ensaio. Neste teste, foi imposta uma entrada do tipo degrau para cada sensor. Ou seja, iniciou-se com um obstáculo a exatamente 58cm do sensor por um período de 30 segundos. Em seguida, esta distância foi reduzida subitamente para 14cm, e assim ficava por mais 30 segundos, totalizando 1 minuto. Desejava-se saber o tempo que levaria para cada sensor corrigir o valor medido de 58cm para 14cm e quão próximo a leitura ficaria deste valor.

Inicialmente obteve-se a curva da Figura 29, claramente não é uma leitura muito estável. Nota-se que o sensor realizava algumas medidas nulas periodicamente. A origem deste problema não foi investigada a fundo, e, portanto, não foi possível explicar o motivo do mesmo. No entanto ele foi contornado por meio da modificação da variável TAM_ERROR, que é o juíz que determina quantas medidas discrepantes devem ser descartadas. Como ocorriam, em média 4 ou 5 leituras nulas, por segurança o valor do juíz foi colocado em 6. Após a modificação obteve-se o gráfico da Figura 28, dessa vez muito mais limpo. É possível perceber ainda que o tempo de reação, após a mudança, fica prejudicado. Ou seja, com o TAM_ERROR maior o sensor tende a demorar mais a responder. No entanto, este é um prejuízo irrelevante se for considerado que o problema das leituras nulas foi sanado completamente.

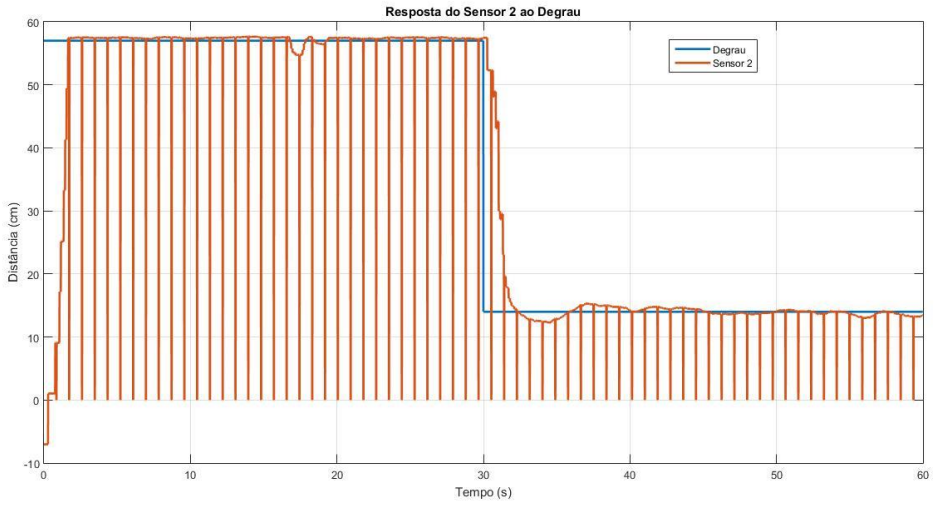

Figura 28: Reposta do sensor 2 ao degrau antes de mudar TAM_ERROR.

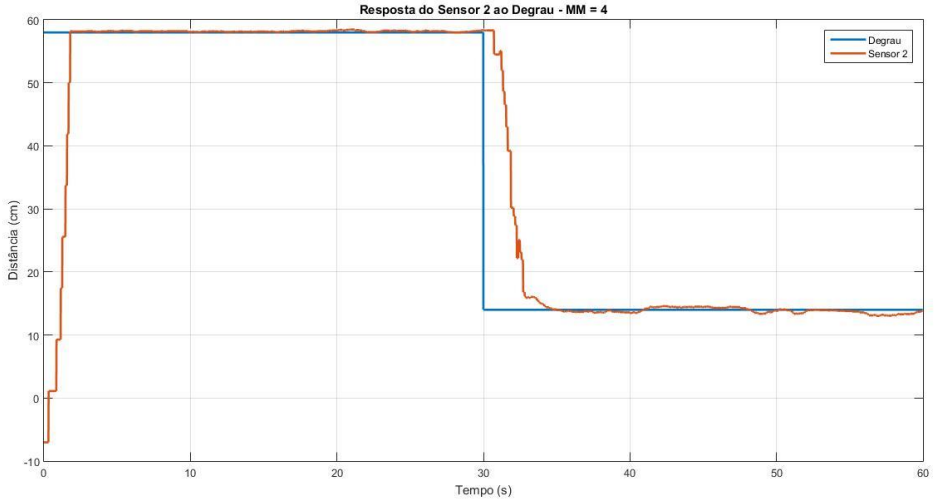

Figura 29: Reposta do sensor 2 ao degrau após a mudança de TAM_ERROR.

A curva da Figura 30 mostra a resposta dos oito sensores, após a modificação da TAM_ERROR. Nesta figura ressalta-se dois pontos importantes: o primeiro é que ao iniciar as medições todos os sensores realizam leituras discrepantes até estabilizarem em 58cm. Isso explica o comportamento da plataforma no vídeo, em que ao ligá-la a mesma gira de forma aleatória, por poucos segundos, antes de atingir o estado "estável". O segundo ponto é a queda no valor das medições, todos os 8 sensores apresentaram resposta satisfatória, em média 1,5 segundos para reconhecer que o obstáculo havia mudado de posição.

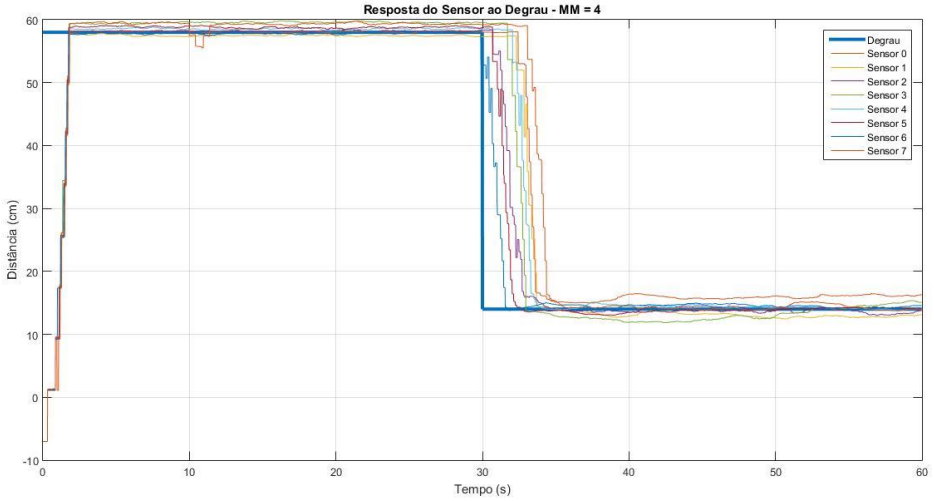

Figura 30: Reposta de todos os 8 sensores.

6. Considerações finais

Este projeto partiu da proposta de melhorar o desempenho da versão anterior, porém mantendo o uso da placa Arduino Due. Os objetivos iniciais eram corrigir as irregularidades durante a execução do giro, já que ela apresentava instabilidade e trepidava mesmo quando não detectava objetos próximos. Assim, a estabilidade da plataforma foi prioridade máxima no desenvolvimento deste projeto.

A primeira dificuldade encontrada foi dominar os procedimentos para configuração dos temporizadores. Os trabalhos foram iniciados no segundo semestre de 2016 e, nessa época, a placa era relativamente nova, e o suporte colaborativo ainda engatinhava. Uma vez superada essa dificuldade, foi possível configurar dois contadores. O primeiro seria usado como ferramenta para gerar o pulso que ativa os sensores. O segundo contador seria responsável por cronometrar o tempo de atuação dos motores e funcionaria simultaneamente ao primeiro. Outro problema recorrente aconteceu devido à compra de novos motores, incentivada pelo mau funcionamento da plataforma na apresentação do trabalho anterior. Apesar de novos, forneciam torque inferior aos anteriores. Por isso, a plataforma não girava mesmo aplicando-se a tensão máxima suportada. Primeiramente, a conclusão foi que o peso extra da solução de carregamento embarcado impossibilitava esse giro. Novas soluções mecânicas fracassaram até que foram colocados de volta os antigos motores.

Ensaios realizados mostraram que os sensores faziam leituras espúrias de distância nula periodicamente. Apesar de investigada, a origem desse problema não foi encontrada. Foi constatado que esse defeito não ocorria quando se realizava leitura de apenas um sensor; por isso, foi descartado erro de mau contato. Esse erro foi corrigido pelo algoritmo do programa, que descartava essas medidas ao se ajustar a constante TAM_ERROR com valor superior a quatro. O valor escolhido foi 6.

Essas leituras erradas também podem justificar o mau funcionamento da plataforma anterior. Isso porque é possível que o programa interpretasse essas leituras como um objeto muito próximo, que desaparecia antes que o giro fosse feito. Assim, a plataforma entrava em oscilação.

Apesar dos problemas encontrados, todos foram solucionados ou contornados de maneira satisfatória. Ainda durante dos ensaios foi observado que os objetivos propostos no começo foram alcançados. A plataforma não exibe mais as inconstâncias na execução do giro, bem como mantém-se imóvel quando não enxerga obstáculo próximo.

O último ajuste foi o da variável TAM_M, quantidade de leituras da média móvel. Um valor muito alto retarda a ação da plataforma quando há um objeto próximo. Porém, um valor muito baixo torna essa solução pouco eficaz. Assim foi escolhido o valor 4 para essa variável.

Desde o projeto anterior era desejado inserir à plataforma futura a detecção de som. Assim, seria acrescentado ao projeto um sistema que fosse capaz de se guiar em direção à fonte sonora. Como

primeiro passo para essa configuração, é sugerido que se modifique a rotina de movimento para que a plataforma guie-se sempre para frente enquanto não detecta um objeto. Ou seja, para que procure um novo obstáculo e desviar-se do mesmo. Após isso, poderiam ajustar-se as constantes referentes à detecção de obstáculos para essa nova realidade. Isso é necessário porque estas constantes dependem da rotina principal, que no caso desse projeto foram empiricamente obtidas para a situação atual.

7. Anexos

7.1. Anexo I - Manual para ajuste de temporização para o processador ARM Séries SAM3X e SAM3A

7.1.1 Introdução

Esse manual tem o propósito de auxiliar a programação de interrupções temporais para os referidos processadores. Isso é necessário já que o manual não reúne todas as informações em apenas uma seção, sendo necessário consultar diversas partes do documento de 1459 páginas.

7.1.2 Tipos de temporizadores

São 9 temporizadores subdivididos em em 3 *TC channels.* Cada temporizador pode ser programado independentemente. Assim, cada canal conta com 3 temporizadores. Na tabela abaixo seguem os temporizadores de cada canal. É possível programar os temporizadores para gerarem pulsos PWM, porém esse tipo de programação não será abordada aqui nesse manual.

TC	Channel	NVIC "irq"	IRQ handler function	PMC id
TC0	0	TC0_IRQn	TC0_Handler	ID_TC0
TC0	1	TC1_IRQn	TC1_Handler	ID_TC1
TC0	2	TC2_IRQn	TC2_Handler	ID_TC2
TC1	0	TC3_IRQn	TC3_Handler	ID_TC3
TC1	1	TC4_IRQn	TC4_Handler	ID_TC4
TC1	2	TC5_IRQn	TC5_Handler	ID_TC5
TC2	0	TC6_IRQn	TC6_Handler	ID_TC6
TC2	1	TC7_IRQn	TC7_Handler	ID_TC7
TC2	2	TC8_IRQn	TC8_Handler	ID_TC8

Tabela 7: Temporizadores disponíveis no processador ARM

O temporizador zero[10] é responsável pelo funcionamento da CPU em si. Funções de temporização já disponíveis na biblioteca Arduino controlam esse temporizador. Por exemplo: se for utilizada a função "*delaymicroseconds(10)"* o processador passará dez microssegundos parado sem realizar qualquer operação nesse tempo. Um leitor mais atento já deve ter compreendido então que dificilmente é aconselhado se escolher esse temporizador para interrupções.

[10] TC0, channel 0 cujo a identificação é ID_TC0

Outra forma de operação para temporizadores que não será abordada é o modo captura. Esse modo permite mensurar parâmetros de outros sinais e faz interrupções por meio desses parâmetros. Só será abordada aqui o temporizador em modo de onda, tipo responsável por gerar sinais periódicos. Há ainda uma subdivisão que deve também ser esclarecida: temporizadores em modo onda podem ser ativos por gatilho interno ou externo. Quando programado para operar em ativação por gatilho externo, depende de um sinal externo para incrementar contagem. Nesse manual serão abordados apenas contadores em modo onda com contagem ativada por gatilho interno, quando não há dependência de sinal exteno.

Valor	Nome	Frequência
0	TIMER_CLOCK1	42 MHz
1	TIMER_CLOCK2	10,5 MHz
2	TIMER_CLOCK3	2,625 MHz
3	TIMER_CLOCK4	656,25 KHz
4	TIMER_CLOCK5	32,768 KHz

Tabela 8: Velocidades disponíveis para temporização

7.1.3 Registradores

Na seção 36.7.3 do manual do processador (Anexo II) está detalhado cada bit do registrador, por isso não será detalhado aqui todos esses bits, de forma a esclarecer de forma prática quais são aqueles mais importantes. É recomendado que o leitor dê uma pausa para visualizar macroscopicamente o registrador.

Ainda nesse registrador é possível o ajuste da interrupção para incremento de variável (será elucidada a seguir). Esse valor é referido como *WAVSEL*. Seguem os possíveis valores que esse registrador pode assumir:

Valor	Nome	Descrição
0	UP	O valor do contador é incrementado por detecção de borda de subida. O registrador RC está inativo.
1	UPDOWN	O valor do contador é incrementado por detecção de borda de subida e descida. O registrador RC está inativo.
2	UP_RC	O valor do contador é incrementado por detecção de borda de subida quando uma variável atrelada ao registrador RC alcança o valor RC.
3	UPDOWN_RC	O valor do contador é incrementado por detecção de borda de subida e descida quando uma variável atrelada ao registrador RC alcança o valor RC.

Tabela 9: Valores para os bits 13 e 14 (WAVESEL) do registrador responsável pela configuração da temporização

Um destaque que deve ser feito é que nessa seção não há referência ao registrador RC. Só há referência desse importantíssimo registrador na seção 36.7.8 do manual (também em anexo). Esse registrador é referido como *Register c.* Apesar do manual não deixar explícito, essa letra (C) se refere a clear. Ou seja, há um incremento de uma variável até um limite estipulado por esse registrador. Após isso, a interrupção temporal é feita e essa variável volta ao valor zero, sendo incrementada novamente a seguir.

Outros registradores usado são o *RTC Interrupt Enable Register* e o *RTC Interrupt Desable Register*. O manual não deixa claro quais são suas funções, e por isso não será muito abordado aqui. As poucas informações sobre esses registradores estão, respectivamente nas seções 14.6.9 e 14.6.10 do manual do processador e seguem também em anexo

7.1.4 Passo a passo

É possível programar essa temporização pelo uso de registradores de duas maneiras diferentes: por valor numérico ou por identificação do registrador ativo. A primeira forma não é indicada porque não deixa explícita a programação do registrador tornando muito difícil qualquer modificação. A segunda forma mostra exatamente qual bit do registrador foi ativo pelo nome, assim fica mais fácil sua compreensão e ajustes. Seguem os passos:

1. Primeiro é necessário desabilitar a proteção de escrita para temporizadores do geral. Para isso basta a seguinte declaração:
2. pmc_set_writeprotect(false);
3. Em sequência, é necessário ativar o clock periférico do temporizador. Para isso, escolha na Tabela 7 qual o temporizador será utilizado. Vale destacar que os primeiros temporizadores têm preferência no caso de haver interrupção de vários temporizadores ao mesmo tempo. Por isso é recomendado que se use os primeiros temporizadores da tabela para soluções que demandem alta precisão na contagem de tempo, já que o ID_TC8 pode sofrer atraso de até 0,095 microssegundos. Para ativar o clock periférico do temporizador, use a declaração:
   Pmc_enable_periph_clk(ID_TCn)
   Lembrando que o valor ID_TCn é a identidade do n-ésimo temporizador destacada na última coluna da Tabela 7.
4. Agora chega a etapa principal: a configuração do registrador que orquestra o funcionamento do temporizador. A sintaxe utilizada é a seguinte:
   TC_Configure (<TC>, <canal do TC>, <Configuração>).
   Os campos <TC> e <canal do TC> foram escolhidos no passo anterior quando se escolheu qual temporizador. Ainda conforme a Tabela 7, se estabelece o TC e o canal nas colunas 1 e 2 para o temporizador escolhido. Já o campo configuração aceita todos os bits do registradaor da seção 36.7.3 do manual do processador. São ao todo 18 configurações separadas em 32 bits. Quando não se escolhe um valor para uma configuração o compilador entende que esse valor deve receber valor zero. De forma prática, ao invés de usar números, como já foi referenciado, usar-se-á o nome do bit a fim de atribuir ao registrador o valor referente. Assim, de maneira prática, são apenas 3 bits que terão valor diferente de zero. Segue a relação:

Bit	Descrição
TC_CMR_WAVE	Se ativo, o TC opera em modo de geração de onda, caso contrário opera no modo captura
TC_CMR_WAVSEL_<Comando>	Comando UP: Vide linha de valor zero da Tabela 9 Comando UPDOWN: Vide linha valor 1 da Tabela 9 Comando UP_RC: Vide linha valor 2 da Tabela 9 Comando UPDOWN_RC: Vide linha valor 3 da Tabela 9
TC_CMR_TCCLKS_TIMER_CLOCKn	Substitua n pela frequência desejada, conforme Tabela 8.

Tabela 10: Bits usados para configuração do temporizador via registrador

Exemplo:

Escolhendo-se o temporizador 7, ou ID_TC7, e frequência de 656,25 KHz para incrementar por borda de subida a variável atrelada ao registrador RC até seu valor:

TC_Configure (TC2,1, TC_CMR_WAVE | TC_CMR_WAVSEL_UP_RC | TC_CMR_TCCLKS_TIMER_CLOCK4)

5. Ajustar o valor do registrador C, responsável pela contagem. A sintaxe é bem simples:

   TC_SetRC(<TC>, <canal do TC>, Valor)

   Por exemplo, para gerar interrupção de 1 segundo usando-se o temporizador 7, e frequência de 656,25 KHz é necessário que o valor de RC seja 656250, então a declaração fica:

   TC_SetRC(TC2, 1, 656250);

6. Ainda são necessárias mais três declarações para configuração. Infelizmente não foi explicitado no manual a função dessas declarações. As sintaxes são as seguintes:
   TC2->TC_CHANNEL[1].TC_IER=TC_IER_CPCS;

   TC2->TC_CHANNEL[1].TC_IDR=~TC_IER_CPCS;

   NVIC_EnableIRQ(TC7_IRQn);

   Note que essas declarações já estão configuradas para o temporizador exemplo, no caso o sétimo. Para outro temporizador é necessário indicar no início da sentença os referidos TC, canal e ID da Tabela 7.

7. Finalmente usa-se o comando TC_Start(<TC>, <Canal>); para iniciar a temporização. Quando a CPU fizer a interrupção, o programa partirá para a leitura de uma função chamada “TCx_Handler()” que deve ser declarada, sendo “x” o temporizador. No caso do exemplo, o valor desse “x” seria 7.

7.2. Anexo II – Trechos do Manual Oficial do Processador

14.6.9 RTC Interrupt Enable Register

Name: RTC_IER

Address: 0x400E1A80

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	CALEN	TIMEN	SECEN	ALREN	ACKEN

- **ACKEN: Acknowledge Update Interrupt Enable**

0: No effect.

1: The acknowledge for update interrupt is enabled.

- **ALREN: Alarm Interrupt Enable**

0: No effect.

1: The alarm interrupt is enabled.

- **SECEN: Second Event Interrupt Enable**

0: No effect.

1: The second periodic interrupt is enabled.

- **TIMEN: Time Event Interrupt Enable**

0: No effect.

1: The selected time event interrupt is enabled.

- **CALEN: Calendar Event Interrupt Enable**

0: No effect.

1: The selected calendar event interrupt is enabled.

14.6.10 RTC Interrupt Disable Register

Name: RTC_IDR

Address: 0x400E1A84

Access: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	CALDIS	TIMDIS	SECDIS	ALRDIS	ACKDIS

- **ACKDIS: Acknowledge Update Interrupt Disable**

0: No effect.

1: The acknowledge for update interrupt is disabled.

- **ALRDIS: Alarm Interrupt Disable**

0: No effect.

1: The alarm interrupt is disabled.

- **SECDIS: Second Event Interrupt Disable**

0: No effect.

1: The second periodic interrupt is disabled.

- **TIMDIS: Time Event Interrupt Disable**

0: No effect.

1: The selected time event interrupt is disabled.

- **CALDIS: Calendar Event Interrupt Disable**

0: No effect.

1: The selected calendar event interrupt is disabled.

36.7.3 TC Channel Mode Register: Waveform Mode

Name: TC_CMRx [x=0..2] (WAVEFORM_MODE)

Access: Read/Write

31	30	29	28	27	26	25	24
BSWTRG		BEEVT		BCPC		BCPB	
23	22	21	20	19	18	17	16
ASWTRG		AEEVT		ACPC		ACPA	
15	14	13	12	11	10	9	8
WAVE	WAVSEL		ENETRG	EEVT		EEVTEDG	
7	6	5	4	3	2	1	0
CPCDIS	CPCSTOP	BURST		CLKI	TCCLKS		

This register can only be written if the WPEN bit is cleared in the TC Write Protection Mode Register.

- **TCCLKS: Clock Selection**

Value	Name	Description
0	TIMER_CLOCK1	Clock selected: internal MCK/2 clock signal (from PMC)
1	TIMER_CLOCK2	Clock selected: internal MCK/8 clock signal (from PMC)
2	TIMER_CLOCK3	Clock selected: internal MCK/32 clock signal (from PMC)
3	TIMER_CLOCK4	Clock selected: internal MCK/128 clock signal (from PMC)
4	TIMER_CLOCK5	Clock selected: internal SLCK clock signal (from PMC)
5	XC0	Clock selected: XC0
6	XC1	Clock selected: XC1
7	XC2	Clock selected: XC2

- **CLKI: Clock Invert**

0: Counter is incremented on rising edge of the clock.

1: Counter is incremented on falling edge of the clock.

- **BURST: Burst Signal Selection**

Value	Name	Description
0	NONE	The clock is not gated by an external signal.
1	XC0	XC0 is ANDed with the selected clock.
2	XC1	XC1 is ANDed with the selected clock.
3	XC2	XC2 is ANDed with the selected clock.

- **CPCSTOP: Counter Clock Stopped with RC Compare**

0: Counter clock is not stopped when counter reaches RC.

1: Counter clock is stopped when counter reaches RC.

- **CPCDIS: Counter Clock Disable with RC Compare**

0: Counter clock is not disabled when counter reaches RC.

1: Counter clock is disabled when counter reaches RC.

- **EEVTEDG: External Event Edge Selection**

Value	Name	Description
0	NONE	None
1	RISING	Rising edge
2	FALLING	Falling edge
3	EDGE	Each edge

- **EEVT: External Event Selection**

Signal selected as external event.

Value	Name	Description	TIOB Direction
0	TIOB	TIOB(1)	Input
1	XC0	XC0	Output
2	XC1	XC1	Output
3	XC2	XC2	Output

Note: 1. If TIOB is chosen as the external event signal, it is configured as an input and no longer generates waveforms and subsequently no IRQs.

- **ENETRG: External Event Trigger Enable**

0: The external event has no effect on the counter and its clock.

1: The external event resets the counter and starts the counter clock.

Note: Whatever the value programmed in ENETRG, the selected external event only controls the TIOA output and TIOB if not used as input (trigger event input or other input used).

- **WAVSEL: Waveform Selection**

Value	Name	Description
0	UP	UP mode without automatic trigger on RC Compare
1	UPDOWN	UPDOWN mode without automatic trigger on RC Compare
2	UP_RC	UP mode with automatic trigger on RC Compare
3	UPDOWN_RC	UPDOWN mode with automatic trigger on RC Compare

- **WAVE: Waveform Mode**

0: Waveform mode is disabled (Capture mode is enabled).

1: Waveform mode is enabled.

- **ACPA: RA Compare Effect on TIOA**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **ACPC: RC Compare Effect on TIOA**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **AEEVT: External Event Effect on TIOA**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **ASWTRG: Software Trigger Effect on TIOA**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **BCPB: RB Compare Effect on TIOB**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **BCPC: RC Compare Effect on TIOB**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **BEEVT: External Event Effect on TIOB**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

- **BSWTRG: Software Trigger Effect on TIOB**

Value	Name	Description
0	NONE	None
1	SET	Set
2	CLEAR	Clear
3	TOGGLE	Toggle

36.7.8 TC Register C

Name: TC_RCx [x=0..2]

Address: 0x4008001C (0)[0], 0x4008005C (0)[1], 0x4008009C (0)[2], 0x4008401C (1)[0], 0x4008405C (1)[1], 0x4008409C (1)[2], 0x4008801C (2)[0], 0x4008805C (2)[1], 0x4008809C (2)[2]

Access: Read/Write

31	30	29	28	27	26	25	24
RC							

23	22	21	20	19	18	17	16
RC							

15	14	13	12	11	10	9	8
RC							

7	6	5	4	3	2	1	0
RC							

This register can only be written if the WPEN bit is cleared in the TC Write Protection Mode Register.

- **RC: Register C**

RC contains the Register C value in real time.

7.3.

Referências bibliográficas

[1] *Robot Institute of America,* 1979.

[2] A. Silva, L. R. Silva, P. L. d. O. Almeida e L. C. Vidal, "Robótica na Indústria Atual," Associação Educacional Dom Bosco, Resende- RJ, 2015.

[3] M. C. Bernardes, GUIDAGE ROBOTISÉ DES AIGUILLES FLEXIBLES POUR DES PROCÉDURES PERCUTANÉES, Montpellier: Universidade de Brasília, 2012.

[4] A. B. Alves, "Plataforma móvel com detecção de obstáculos," Departamento de Engenharia Elétrica, Universidade de Brasilia, Brasília, DF, Brasil, 2014.

[5] A. Araújo e L. Santana, "Plataforma móvel com detecção de obstáculos," Brasília, 2015.

[6] I. B. d. S. Oliveira e R. F. Oliveira, Plataforma móvel com detecção de obstáculos, Brasília: Universidade de Brasília, 2016.

[7] ElecFreaks, "Ultrasonic Ranging Module HC - SR04," [Online]. Available: http://www.micropik.com/PDF/HCSR04.pdf. [Acesso em 13 Junho 2017].

[8] E. K. V. KALKO e M. A. CONDON, "Echolocation, olfaction and fruit display: how bats find fruit of flagellichorous Cucurbits," *Functional Ecology,* pp. 364-372, 1998.

[9] ELAN MICROELETRONICS CORP., *EM78P152/3S,* TAIWAN: ELAN MICROELETRONICS CORP., 2007.

[10] TEXAS INSTRUMENTS, *MAX232. MAX232I, DUALEIA-232 DRIVERS/RECEIVERS,* DALLAS,TEXAS: TEXAS INSTRUMENTS, 1989.

[11] TEXAS INSTRUMENTS, *TL071, TL071A, TL071B, TL072, TL072A, TL072B, TL074, TL074B LOW-NOISE JFET-INPUT OPERATIONAL AMPLIFIERS,* DALLAS, TEXAS: TEXAS INSTRUMENTS, 2004.

[12] SUWA KOUBOU, "Modificação da falha no circuito do módulo sensor de distância ultra-sônica "HC-SR04"," SUWA KOUBOU, [Online]. Available: http://www.suwa-koubou.jp/micom/HC-SR04/hc_sr04.html. [Acesso em 13 06 2017].

[13] L. Fried, “Adafruit Motor Shield v2 for Arduino,” [Online]. Available: https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino?view=all. [Acesso em 31 Maio 2017].

[14] NXP Semiconductors, “PCA9865,” 2015. [Online]. Available: http://www.nxp.com/documents/data_sheet/PCA9685.pdf. [Acesso em 31 Maio 2017].

[15] S. Chapman, Electric Machinery Fundamentals, 4ª ed., New York: McGraw-Hill, 2005.

[16] P. C. Sen, Principles of Electric Machines and Power Electronics, 2ª ed., Kingston: John Wiley & Sons, 1997.

[17] EMB, Lithium-ion Battery, Santa Cruz, California: EMB, 2010.

[18] Moog Components Group , Permanent Magnet DC Motors.