PROJETO DE GRADUAÇÃO

Plataforma móvel com detecção de obstáculos

Igor Barroso de Sá Oliveira

Renato Ferreira Oliveira

Brasília, 05 de Julho de 2016

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

Igor Barroso de Sá Oliveira
Renato Ferreira Oliveira

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, 05 de Julho de 2016

FICHA CATALOGRÁFICA

OLIVEIRA, IGOR; OLIVEIRA, RENATO
Plataforma móvel com detecção de obstáculos [Distrito Federal] 2016
X, 59 p., 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

OLIVEIRA, I.; OLIVEIRA, R (2016). Plataforma móvel com detecção de obstáculos, Relatório de Graduação em Engenharia Elétrica, publicação XXXXXX, Departamento de Engenharia Elétrica, Universidade de Brasília, DF, 59 p.

CESSÃO DE DIREITOS

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 à minha família, que me acompanhou durante toda a minha vida e que, no final desta etapa, demonstrou novamente todo o seu amor e paciência.

Igor Barroso de Sá Oliveira

Dedicatória

Dedico este trabalho a minha família, sem o apoio da qual não seria a pessoa que sou hoje.

Renato Ferreira Oliveira

Agradecimentos

Agradeço primeiramente a Deus pelo dom da vida e pelas graças por Ele concedidas. Agradeço à minha família, pelo apoio incondicional aos meus estudos e pela confiança que depositaram em mim; ao meu pai, pelos ensinamentos e conselhos; e à minha mãe, por sua paciência e seus cuidados. Agradeço ao professor Ricardo Zelenovsky por sua valiosa orientação e sua prontidão em auxiliar-nos no decorrer deste projeto. Agradeço ao Renato Oliveira por dividir comigo este projeto e, com isto, a última etapa em nossas vidas acadêmicas. Agradeço por fim aos meus colegas de faculdade por seu apoio e amizade durante toda a duração do curso.

Igor Barroso de Sá Oliveira

Agradecimentos

Agradeço a minha família, por todo o apoio e auxílio em todos esses anos de curso e no decorrer da minha vida. Agradeço ao professor Ricardo Zelenovsky por todo o apoio no decorrer desse projeto, por toda a compreensão e paciência e por todas as ajudas em momentos de necessidade. Agradeço ao Igor, pela parceria nesse último semestre e por todas as ajudas prestadas no decorrer desse projeto. Agradeço aos meus amigos de curso, sem os quais não teria memórias boas desses cinco anos.

Renato Ferreira Oliveira

RESUMO

Este projeto apresenta o aprimoramento de uma plataforma móvel para detecção de obstáculos, através de sensores ultrassônicos, e fuga, através de motores DC. A tomada de decisões e o processamento de informações são feitos por uma placa Arduino. O projeto tem como proposta a troca da placa Arduino Mega por uma Arduino Due, realizando todas as alterações de software e hardware necessárias para essa troca.

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

ABSTRACT

This project shows the improvements made to a mobile platform capable of identifying obstacles, through ultra-sonic sensors, and running away from them, through DC motors. The decision-making process and information processing are made by an Arduino board. The objective of this project is the exchange of the Arduino Mega board for an Arduino Due, making all the necessary modifications, both on hardware and software.

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

SUMÁRIO

LISTA DE IMAGENS

1. Introdução

1.1. Ambientação:

Vivemos em uma época em que a evolução tecnológica se dá de maneira extremamente rápida. Não se passa muito tempo sem que uma invenção recente passe a ser ultrapassada por outra que a realize melhor. Uma dessas invenções são os robôs: sejam eles em forma anatômica ou em forma de ferramentas, como na Figura 1. Independente da forma assumida, cada vez mais os robôs se fazem presentes no cotidiano, realizando diversas funções.

Segundo o R.I.A., *Robotics Institute of America* (ou Instituto Americano de Robótica), um robô é definido como sendo um manipulador reprogramável e multifuncional projetado para mover materiais, partes, ferramentas ou dispositivos especializados através de movimentos variáveis programados para desempenhar uma variedade de tarefas [1].

No geral, eles têm como função a realização de atividades consideradas repetitivas, cansativas, sujas, perigosas, de alta precisão onde o ser humano pode não conseguir manter a mesma eficiência ou sequer realizá-la.

Figura 1 - Exemplo de Robô

Eles são compostos por diversas peças e dispositivos, sendo as de maior importância os sensores, os atuadores e as unidades de processamento. Os sensores captam informações externas relacionadas à atividade realizada pelo robô. Essas informações são enviadas à unidade de processamento que as interpretará e tomará decisões segundo algoritmos pré-estabelecidos. Os atuadores são então acionados de acordo com a decisão tomada [2].

A ideia de robôs pode ser vista desde a antiguidade. O golem, Figura 2, era uma criatura da mitologia judaica, um robô de argila que respondia a comandos [3]. Na renascença italiana, Leonardo da Vinci fez esboços de um robô humanoide [4]. Apesar dessas presenças na história, o termo robô só surgiu em 1920, com a peça R.U.R. de Karel Čapek. A peça de ficção científica não mostrava a concepção atual de robôs, mas sim de androides e *cyborgs*, pois eles possuíam pensamento próprio e podiam ser confundidos com humanos [5].

Figura 2 - Golem como representado no filme de1920, O Golem

No século XVIII, durante a revolução industrial, que o desenvolvimento de robôs sofreu vários avanços significativos. Visto que havia a necessidade do aumento na produção fabril, houve um aumento de esforços na criação de máquinas que conseguissem manipular, juntar e movimentar peças de forma automática. Mas foi somente em 1951 que surgiu o primeiro robô automático comercial, o Unimate, desenvolvido por George Devol e Joe Engleberger, que trabalhava na linha de produção da General Motors, movimentando pedaços quentes de metal e soldando-os aos chassis dos carros [6].

Atualmente, os robôs possuem forte presença na produção industrial, onde reduzem o custo de produção, melhoram a qualidade do produto e poupam trabalhadores de condições de trabalho insalubres. Apesar de serem os mais comuns, os robôs industriais, não são os únicos tipos de robôs que existem.

Um fator que contribui muito para a popularização da robótica e seu constante desenvolvimento são os microcontroladores, Figura 3. Microcontroladores são pequenos computadores usados em aplicações onde é necessário um controle automatizado. Entre suas principais características estão a programação e as interrupções. A parte essencial de um microcontrolador é sua programação, pois é nela que são definidas as ações a serem tomados no decorrer de um processo. Além disso, o sistema necessita de uma resposta em tempo real para que o sistema possa ser controlado, para isso existem as interrupções. Elas permitem que, no momento em que um evento pré-programado ocorra, o programa seja interrompido e realize uma função de maior prioridade [7].

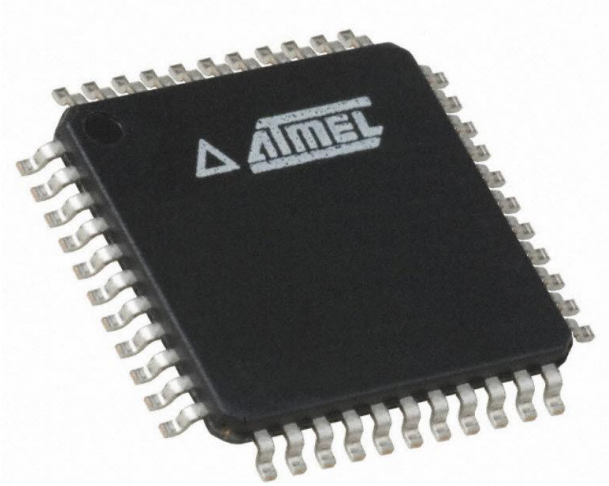

Figura 3 - Microcontrolador Atmel

A popularização de microcontroladores acessíveis, de baixo custo, fácil manipulação e aprendizado, permite a entrada de mais pessoas nesse ramo, o que por sua vez, faz com que haja uma evolução crescente da robótica. A junção da criatividade do programador com a constante evolução de componentes e processadores torna possível o surgimento de propostas inéditas. Robôs capazes de interagir com seres humanos de forma emocional ou interagir entre si com tomada de decisão em grupo, já mostra um grande avanço na robótica.

1.2. Motivação:

Apesar do desenvolvimento rápido da robótica e o aumento considerável de projetos na área, existe também um aumento no preço para o consumidor final. Projetos muito sofisticados tenderão a ter preços mais elevados. Dessa forma, o aprofundamento do estudo nessa área, visando um projeto economicamente viável, é algo de grande interesse. Outro grande objetivo dos trabalhos em robótica é o aumento da eficiência.

Encontramos um projeto disponível no Departamento de Engenharia Elétrica cujo funcionamento apresentava margem para melhorias nestes aspectos. O projeto tratava de um carro que detectava e fugia de obstáculos. Os aspectos envolvidos nesse projeto e suas possíveis aplicações, como sistemas de transporte sem a presença de motoristas ou mecanismos de freio e desvio com velocidade de resposta mais rápida que a humana, foram pontos importantes para a decisão de aprimorar esse projeto.

Com tudo isto em mente, decidimos adotar esse projeto e modificá-lo, utilizando novos componentes e funcionalidades que o valorizassem ainda mais e o deixassem mais rápido e eficiente. Vários desafios se apresentaram nesse trabalho. Podemos citar como principal a programação do novo processador, que se mostrou muito diferente dos processadores geralmente utilizados nas plataformas Arduino.

1.3. Objetivos:

Este projeto tem por objetivo o aperfeiçoamento da plataforma móvel projetada para desviar de obstáculos de forma autônoma, segundo uma programação em uma placa Arduino. A plataforma é, mecanicamente, composta por quatro motores DC, responsáveis pelo controle de quatro rodas, e uma base, onde se encontram os sensores de distância ultrassônicos, dentre outros elementos que serão detalhados nos próximos capítulos.

Em um primeiro momento, a plataforma identifica qual o obstáculo mais próximo, se direciona para o lado oposto ao obstáculo afastando-se com velocidade inversamente proporcional à distância entre a plataforma e o obstáculo. Realizado por equipes anteriores, a plataforma atingia de forma razoável todos os objetivos a ela proposta.

O foco desse projeto é o aperfeiçoamento da plataforma através da substituição da placa Arduino por outra, também Arduino, mas com um processador mais eficiente. Dessa forma, almejamos uma resposta mais rápida, um menor tempo de processamento e um menor consumo de memória.

1.4. Nova Versão:

O projeto anterior utilizava uma placa Arduino ATMega2560 [8]. Essa placa se mostrava lenta e com poucas interrupções, que eram necessárias ao projeto. Esse problema foi solucionado trocando a placa por uma Arduino Due que contava com um maior poder de processamento e permitia interrupção em todas as suas portas.

A *shield*, placa responsável por realizar operações mais específicas e mais pesadas do ponto de vista do programa, dos motores foi projetada para o Arduino Duemilanove e já se mostrava pouco eficiente no projeto anterior. Devido a sua configuração de pinos, eram necessários três *timers* para a geração dos sinais PWM, onde seriam precisos somente dois. A conexão da *shield* com o Due também se mostrava complicada, pois seria necessária a adição de uma placa de conexões entre a Due e a *shield* para que as duas se comunicassem. Dessa forma trocou-se a *shield* por uma indicada para a Due.

O software do carro utilizava de curvas suaves enquanto realizava a fuga, optamos por fazer com que ele rodasse em seu eixo antes de ativar a fuga. O objetivo era que a plataforma posicionasse o obstáculo exatamente nas suas costas antes de fugir, para que não houvesse erros de ecos indesejados enquanto ele se preparava pra fugir.

Outra mudança importante foi a troca da placa de conexões entre os sensores e a Arduino. A ATMega possuía apenas seis pinos de interrupção externa, o que fazia necessária o uso de um multiplexador para receber os dados dos sensores de ultrassom. Como a Due pode utilizar todos os seus pinos como interrupção externa, o uso do multiplexador se mostrou desnecessário. Assim foi confeccionada uma nova placa para a conexão dos sensores a placa Arduino.

Uma última mudança a ser mencionada foi a troca da alimentação. O Arduino apontava resultados diferentes dependendo se estivesse alimentado via USB ou via conecto P4. Observou-se que a alimentação via conector P4 gerava algum tipo de erro no resultado dos sensores. Dessa forma optou-se pela adição de uma alimentação USB para o Arduino. A alimentação anterior foi mantida, mas utilizada somente para a *Motor Driver Shield*.

1.5. Estrutura do Trabalho:

O trabalho se encontra dividido da seguinte forma:

- *Capítulo 1 – Introdução*: Ambientação, motivação e apresentação dos objetivos do projeto.
- *Capítulo 2 – Projeto* : Descrição do projeto original.
- *Capítulo 3 – Hardware*: Apresentação e detalhamento do projeto do ponto de vista mecânico.
- *Capítulo 4 – Programação*: Apresentação e detalhamento do ponto de vista da programação.
- *Capítulo 5 – Ensaios e Testes*: Apresentação dos resultados obtidos com a plataforma após as alterações.
- *Capítulo 6 – Considerações finais*: Resumo do desenvolvimento do projeto e conclusões.

2. Projeto Base

O projeto original foi desenvolvido pela aluna Aline Barbosa em seu projeto de conclusão de curso [9]. O projeto foi então aprimorado pelos alunos André Agostinho e Luiz Fernando para solucionar problemas de alimentação e de estrutura mecânica, no respectivo projeto de conclusão de curso [8]. 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. Nesse capítulo é detalhado o projeto modificado pelos alunos André Agostinho e Luiz Fernando.

2.1. Plataforma:

A plataforma utilizada como base possui uma estrutura mecânica formada por duas placas de acrílico, quatro rodas, oito *protoboards*, oito sensores de ultrassom, cabos, resistores, capacitores, parafusos e porcas utilizados nas conexões e a placa Arduino Mega2560 ligada ao *motor driver shield*. Do projeto original para o projeto base foram adicionadas duas chaves, conjunto de baterias, adaptador de baterias para Arduino, placa para conexão dos sensores e o módulo Bluetooth HC-05. O robô utilizado como base é mostrado na Figura 4.

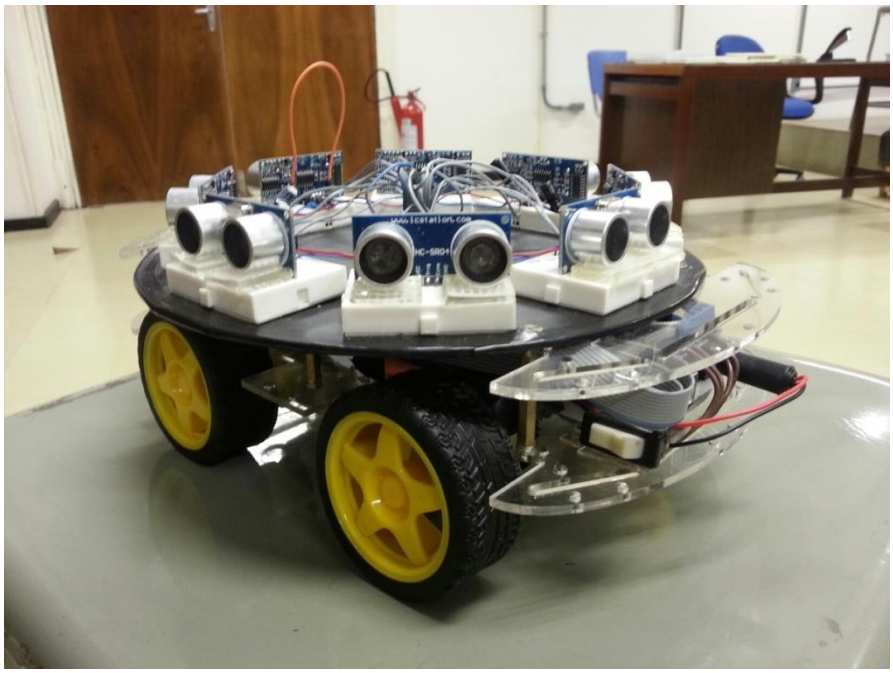

Figura 4 - Plataforma base

2.2. Arduino:

O projeto base utiliza como sua unidade de processamento uma placa Arduino Mega 2560, que possui um microcontrolador Atmega 2560 embarcado [10]. Esse microcontrolador possui seis *timers*, sendo um de oito *bits* e os demais de dezesseis *bits*. Todos os *timers* podem ser configurados para gerar sinais PWM em até três pinos do micro controlador cada, entretanto nem todos os pinos estão disponíveis ao usuário. O Atmega 2560 também dispõe de quatro módulos de comunicação serial UART e um módulo de comunicação *two wire interface* (TWI, também conhecida como I^2C). Apesar de o microcontrolador permitir o uso de até oito pinos de interrupção externa, o Arduino disponibiliza apenas seis para o usuário final [11].

A placa dispõe também de conexão USB, um *jack* de energia, um botão *reset*, entre outros. Ela vem pronta para uso, bastando alimentá-la diretamente com uma entrada USB, uma bateria ou um adaptador AC-DC conectado a uma tomada. Para sua programação, o Arduino oferece outra facilidade através do IDE, *Integrated Development Environment* (ou Ambiente de Desenvolvimento Integrado), como mostrado na Figura 5, que é um software próprio do Arduino e que dispõe de funções previamente definidas e uma linguagem C simples e acessível.

Figura 5 - Arduino IDE

2.3. 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 toda a lógica do uso do sensor de ultrassom para o cálculo da distância e direção do obstáculo. A fuga inclui a lógica de acionamento dos motores e o intervalo em que os mesmos são acionados.

2.3.1 Detecção de obstáculos

Na plataforma original, os oito sensores de ultrassom são dispostos em uma circunferência na plataforma superior sendo responsáveis pela detecção de obstáculos. Os sensores são disparados um a cada ciclo [9]. O Arduino então calcula a distância até um obstáculo utilizando uma fórmula fixa fornecida na própria *datasheet* dos sensores [12].

No projeto base a detecção dos sensores é dividida em dois ciclos, onde através de um multiplexador são disparados quatro sensores a cada ciclo [8]. No entanto o cálculo se mostrava impreciso, pois a qualidade dos ecos interferia nas medições realizadas pelos sensores. Dessa forma, o algoritmo anterior previa ainda a detecção de um mesmo obstáculo por dois sensores vizinhos, fazendo um cálculo baseado na Lei dos Cossenos para obter a real distância até o objeto [8].

O ciclo do programa funciona da seguinte forma: em um primeiro momento são disparados os sensores, depois se aguarda a interrupção gerada pelos sensores que confirma a realização de uma leitura. Esses dados de leitura são armazenados em vetores usados para calcular a média dos dados recebidos, que consideram as três últimas leituras; a média é, então, utilizada como entrada para o cálculo do ângulo de fuga e o consequente acionamento dos motores.

2.3.2 Fuga

O ângulo de fuga e a distância ao obstáculo, calculados na etapa de detecção, são valores utilizados pelo Arduino para determinar o tempo de giro e a velocidade de acionamento do motor. O tempo de giro se refere ao tempo necessário para o alinhamento da plataforma com a referência de ângulo, que é o ângulo referente ao sensor dianteiro do carro. O carro deve então acionar os motores, para que a referência de ângulo esteja diametralmente oposta ao obstáculo, ou seja, o obstáculo deve estar localizado no sensor traseiro.

Depois de alinhada a plataforma, a velocidade é utilizada para fugir do obstáculo. A velocidade é proporcional à distância entre a plataforma e o obstáculo, sendo maior quanto mais próximo o obstáculo estiver. Caso o obstáculo estivesse na parte dianteira do carro, o mesmo girava sobre o próprio eixo antes de fugir. Nos demais casos a plataforma realizava uma curva suave, andando para frente, através do acionamento dos motores com velocidades diferentes.

2.4. Principais problemas da versão anterior

Apesar de o projeto base realizar tudo aquilo que ele propunha, ele possuía algumas falhas. Sua resposta era instável e a plataforma tendia a não responder bem aos obstáculos. Sua fuga em curva suave tendia a deslocar o obstáculo dificultando ao carro identificar onde exatamente o objeto se encontrava.

Outro problema era a capacidade de processamento da Arduino utilizada. Um interesse em qualquer projeto é poder realizar implementações futuras. O projeto base, no entanto, não possuía essa mais essa capacidade, o Arduino utilizado só possuía capacidade para lidar com o que já havia sido programado.

Nos próximos capítulos, apresentaremos as modificações realizadas neste projeto para aprimorar o funcionamento da plataforma, detalhando cada uma tanto do ponto de vista do hardware quanto da programação.

3. Hardware

Entre os componentes mecânicos do projeto, destaca-se a plataforma que é o esqueleto físico do projeto; dentre os eletrônicos, citamos a placa Arduino DUE, a motor *shield*, os sensores de ultrassom, o módulo Bluetooth e os motores que movimentam a plataforma. Uma descrição mais detalhada de cada um destes componentes será feita nesse capítulo.

3.1. Plataforma

Herdada do projeto original, a plataforma, como pode ser vista na Figura 6, é composta de duas placas de acrílico sobrepostas, presas através de parafusos. Entre elas existe um vão no qual são colocados componentes como a bateria, o Arduino e a *shield* dos motores. Estes componentes são fixados à placa através de fita dupla face.

Na face superior da plataforma são posicionados os sensores de ultrassom, fixados em *protoboards*, dispostos em uma circunferência e a placa de conexões descrita na seção 3.9.

Na face inferior da plataforma são colados com fita dupla-face, os motores DC que movimentam pequenas rodas de plástico. Estas rodas também servem o propósito de fornecer sustentação ao carro.

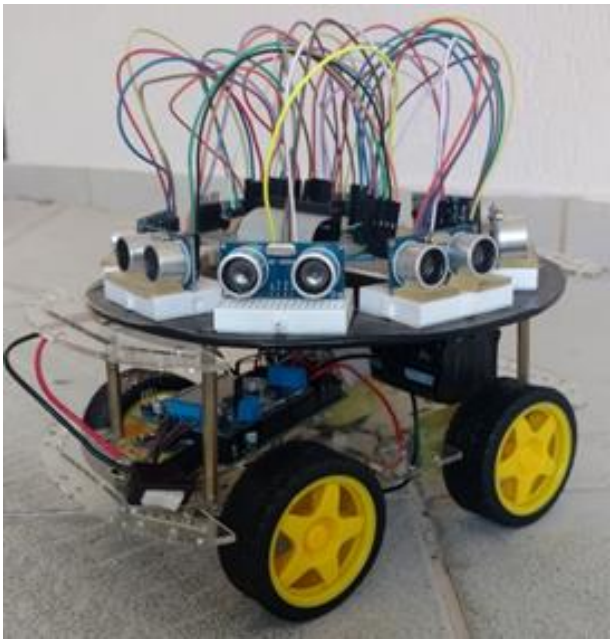

Figura 6 - Plataforma atual

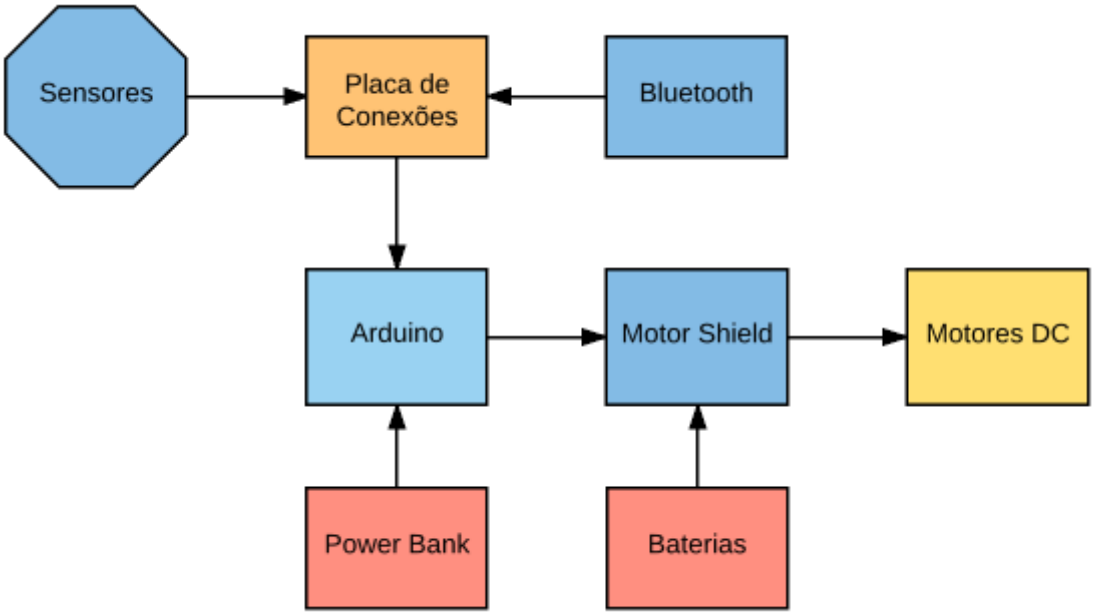

Figura 7 - Diagrama de blocos da plataforma

A Figura 7 ilustra as conexões entre os dispositivos que formam a plataforma. O Arduino é a parte central da plataforma e todos os demais componentes, com exceção das baterias, se conectam a ela de forma direta ou indireta. O *Power Bank* é responsável pela alimentação do Arduino e por consequência dos sensores e do Bluetooth. As baterias alimentam a *Motor Shield* e por consequência os motores. A *Motor Shield* está conectada aos motores DC e controla sua ativação de acordo com os comandos do Arduino. A placa de conexão facilita a ligação do módulo Bluetooth e dos sensores de ultrassom ao Arduino.

3.2. Arduino

O Arduino é uma família de placas de desenvolvimento de propriedade da empresa italiana de mesmo nome. A marca possui diversos modelos que podem ser aplicados a uma vasta gama de projetos, o que ajudou a difundir o seu uso pelo mundo. A grande maioria dos modelos é baseada em microcontroladores da empresa norte-americana Atmel, sendo os da família AVR os mais comuns [13]. As placas contam com um ambiente de desenvolvimento (IDE) próprio, que contém um conjunto de funções pré-programadas que facilitam a programação, embora nem sempre sejam particularmente eficientes. A principal linguagem utilizada para programação é uma forma simplificada de C, embora a IDE forneça suporte ao C++ [14].

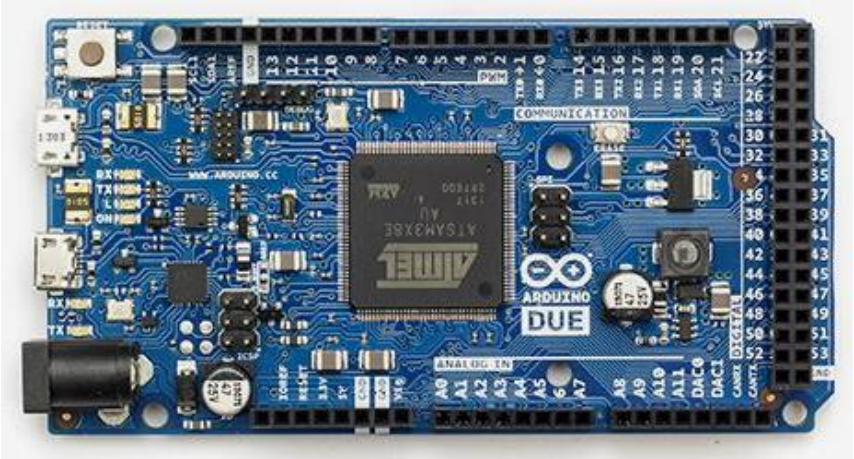

Figura 8 - Arduino Due

O Arduino DUE, Figura 8, utilizada neste projeto, é baseada no ARM SAM3X8E, o primeiro da linha ARM a integrar uma placa Arduino. As funcionalidades desta placa foram as grandes motivadoras deste projeto: ele possui uma arquitetura RISC (*Reduced Instructions Set Computer* ou Computador com Conjunto de Instruções Reduzido, em tradução livre) de 32 bits; conta com 54 pinos digitais, mostrados na Figura 9, 512kB de memória Flash, além de 4 portas UART e duas interfaces I²C. Outros diferenciais são os nove *timers* disponíveis, o *clock* de 84MHz e o recurso de interrupção externa, presente em todos os pinos digitais [15].

O *clock* de 84MHz e a disponibilidade de um grande numero de interrupções externas, foram determinantes na escolha da placa. O primeiro, pois a velocidade de processamento implica numa resposta mais rápida e eficiente para a plataforma. O segundo, pois a maior disponibilidade de interrupções externas permite uma simplificação tanto na parte mecânica quanto na parte lógica do carro.

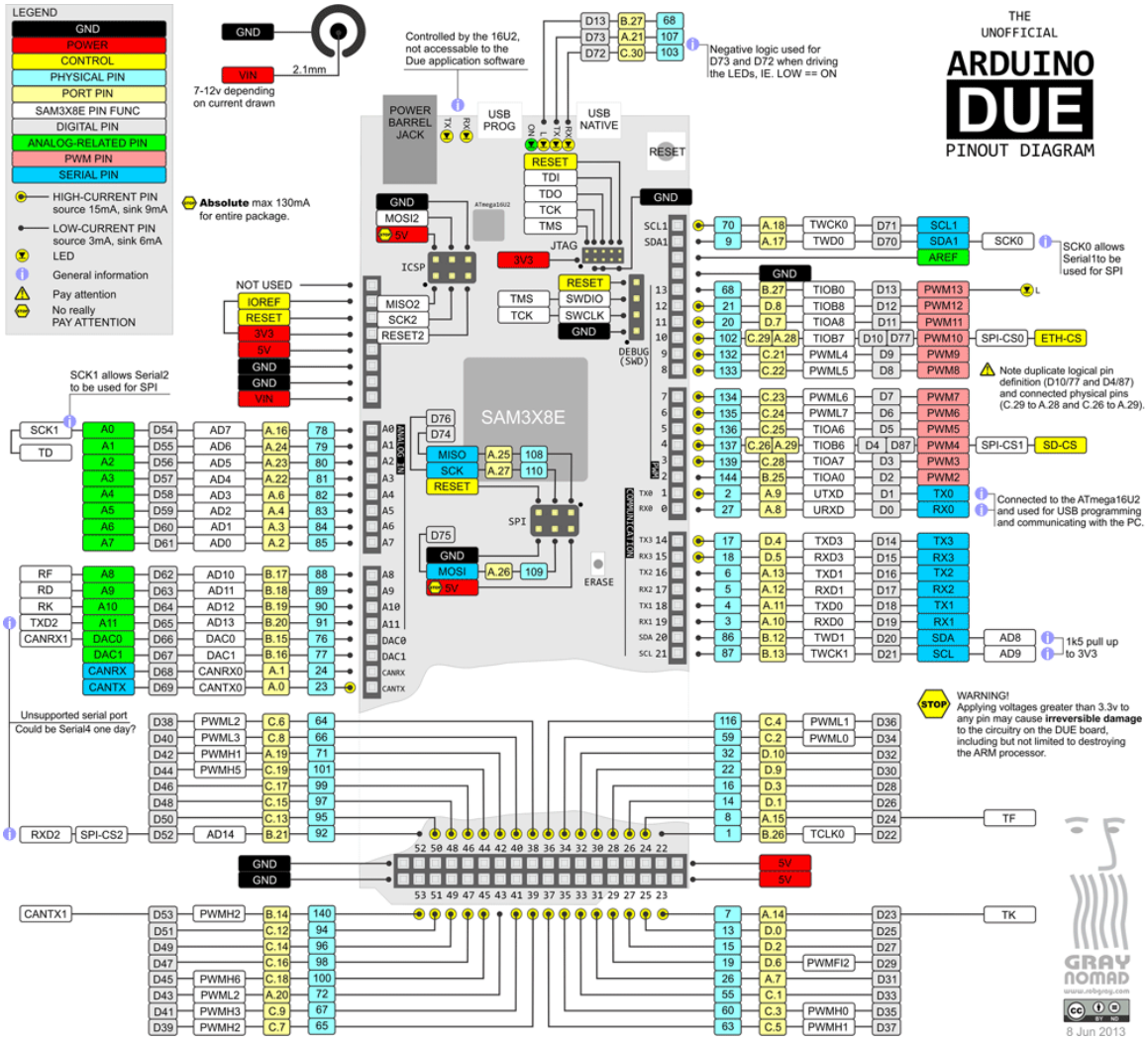

Figura 9 - Mapa de pinos do Arduino DUE

3.3. Sensor ultrassom

O sensor utilizado no projeto é o HC-SR04, vide Figura 10, também uma continuação do projeto anterior. Optou-se pela tecnologia de ultrassom, pois ela é simples, eficiente e de baixo custo. O princípio de funcionamento do sensor é o mesmo de um sonar: emite-se um trem de pulsos de som com uma frequência de 40kHz – acima da capacidade auditiva humana, motivo pelo qual não ouvimos o som - e espera-se que as ondas sejam refletidas em um obstáculo. O receptor embutido no sensor é capaz de detectar este eco e com isto o microcontrolador calcula a distância do objeto [12].

Figura 10 - Sensor de ultrassom HC-SR04

Para emitir o trem de pulsos no transmissor, é preciso deixar o pino de *trigger* em nível alto por 10µs e em seguida abaixá-lo novamente. Oito pulsos serão gerados e, 400µs após o começo do sinal de *trigger*, o pino de *echo* muda para o nível alto. O tempo em que o pino permanece neste estado é proporcional à distância até o objeto [12], conforme será demonstrado na seção 4.3. Este processo encontra-se ilustrado na Figura 11, retirada do *datasheet* do sensor [12].

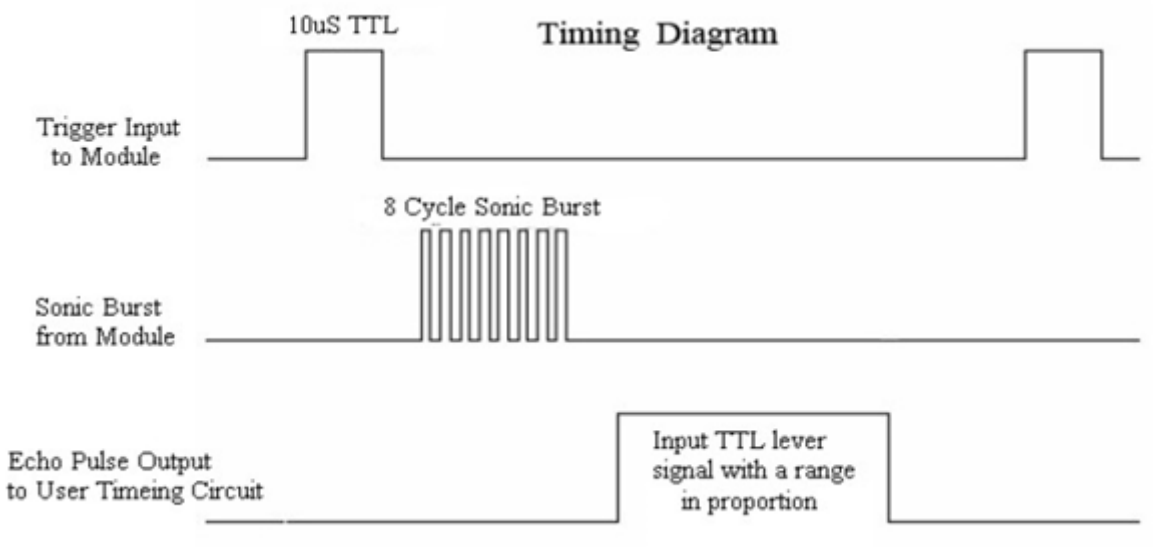

Figura 11 - Diagrama de tempo do sensor HC-SR04

Como já descrito, os sensores estão distribuídos igualmente ao redor da plataforma, a cada 45°, cada um deles fixado em uma *protoboard*. A disposição e a numeração dos sensores são apresentadas na Figura 12.

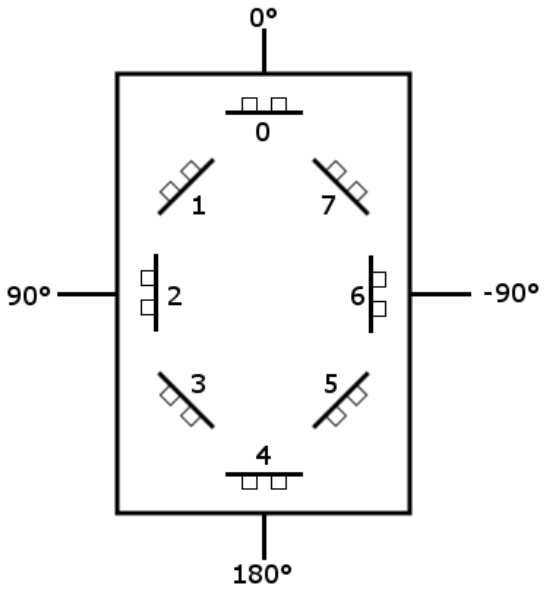

Figura 12 - Disposição dos sensores de ultrassom

3.4. Arduino Motor Shield V2

Devido à grande popularidade do Arduino, é comum que empresas e engenheiros independentes fabriquem placas adaptadoras para conectar um Arduino a um componente externo. Estes adaptadores são chamadas de *Shields* e podem servir aos mais variados propósitos. No presente caso, a finalidade da placa é fornecer uma ligação entre o Arduino e o conjunto de quatro motores DC que movimentam a plataforma.

A *Adafruit Motor Shield for Arduino V2*, que pode ser vista na Figura 13, é uma atualização da placa que vinha sendo utilizada no projeto. A troca foi motivada principalmente pela disposição dos pinos da *shield*: a primeira versão da placa foi desenhada para o Arduino Duemilanove [8], levando em conta os pinos de *Pulse Width Modulation* (PWM) específicos deste Arduino. Portanto, a antiga *shield* é incompatível com o Arduino Due; seria necessário o uso de *jumpers* para realizar as conexões de modo correto. Em vez disso optou-se pela nova versão da placa, que utiliza comunicação I²C, o que a torna compatível com a maioria dos Arduinos já que este tipo de comunicação requer apenas dois pinos [16] e está disponível na grande maioria das placas; uma explicação mais detalhada sobre o protocolo I²C pode ser encontrada na seção 3.4.1.

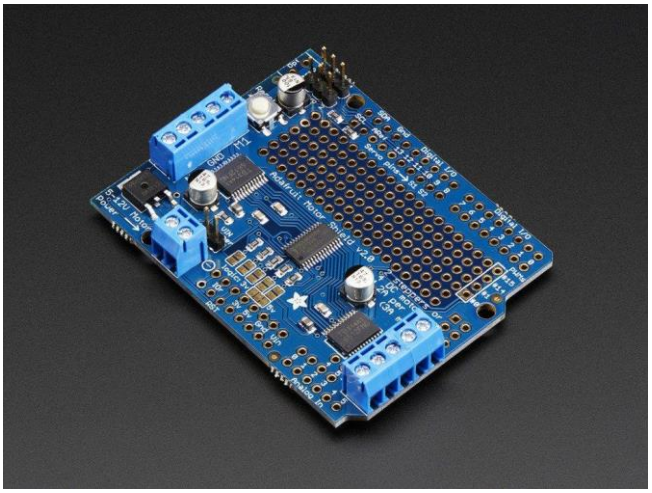

Figura 13 - Adafruit Motor Shield V2

A *shield* suporta a conexão simultânea de quatro motores DC, que devem ser conectados aos *bornes* presentes na placa. A placa utiliza dois drivers TB6612FNG, cada um com capacidade para alimentar dois motores DC [17]. Um chip PCA9685 (originalmente criado como um driver de LEDs) fornece a comunicação I²C e gera os sinais de PWM que movimentam o motor, aliviando o Arduino desta função [18].

A alimentação da *shield* pode ser feita pelo Arduino ou por uma fonte externa nos terminais de alimentação; por simplicidade, escolheu-se primeiramente alimentar a placa diretamente do Arduino, colocando-se um *jumper* nos pinos indicados pelo manual da placa; porém, esta solução apresentou problemas de leitura, que eram solucionadas ao alimentar o Arduino através do conector USB. Então, utilizou-se uma bateria portátil para alimentar o Arduino e o conjunto de baterias para alimentar os motores, conforme explicado nas seções 3.7 e 3.8.

Uma pequena alteração na placa é necessária para o uso com o Arduino DUE: porque ele opera com níveis lógicos altos nos pinos em 3,3V, em vez dos 5V comuns a outros Arduinos. Para fazer essa alteração é necessário localizar um pequeno agrupamento de *pads* na *shield*, identificado como *logic;* corta-se, então, a ligação entre o *pad* do meio e o da direita, identificado como 5V, e cria-se uma ligação (com o auxílio de uma gota de solda) com o pino da esquerda, identificado como 3V [19]. Assim, seleciona-se o uso desta tensão como nível alto nos pinos da *shield*. A Figura 14 ilustra esta mudança.

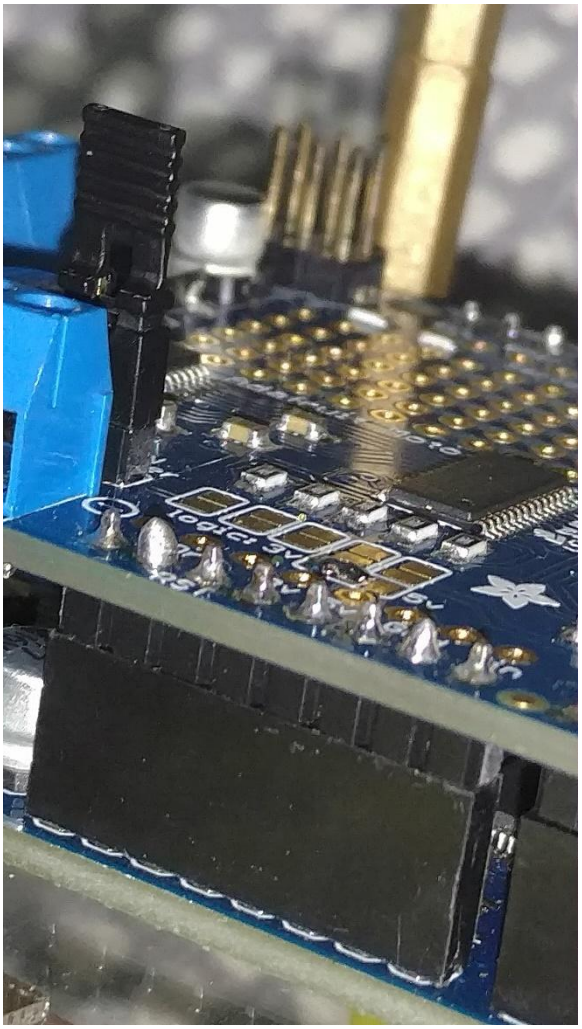

Figura 14 - Detalhe da lógica em 3.3V e jumper de alimentação

3.4.1 I²C

O I²C (*Inter-Integrated Circuit*) é um protocolo de comunicação desenvolvido pela Phillips cuja característica principal é o número reduzido de pinos: apenas dois são utilizados em cada dispositivo. Além disso, o I²C possui uma topologia de barramento, o que permite que vários dispositivos "escravos" se conectem a um "mestre" simultaneamente. Um sistema de endereços é utilizado para efetuar a comunicação em situações onde mais de um escravo está presente [16].

Os pinos utilizados são chamados de *Serial Clock Line* (SCL) e *Serial Data Line* (SDA). O SCL contém um sinal de *clock* fornecido pelo dispositivo mestre: a cada "batida" do *clock* um novo bit do SDA pode ser lido [16].

Todo o protocolo de comunicação é efetivamente controlado pelo dispositivo mestre (neste projeto, o Arduino). Para iniciar a comunicação, o mestre precisa esperar até que o barramento esteja livre (ou seja, que toda comunicação no barramento tenha sido encerrada); então, ele gera uma condição de *start*, comunicando a seus escravos que deseja utilizar a linha. Os escravos entram em modo de escuta; a seguir, o mestre envia pelo barramento o endereço do escravo com quem deseja se comunicar, juntamente com um bit que indica se quer transmitir dados ou recebê-los do escravo. O escravo deve responder com um *acknowledge* (ACK). Em seguida a transferência de dados pode então ser iniciada. A cada byte de informação, o dispositivo receptor deve enviar um ACK. Uma vez terminada a comunicação, o mestre gera uma condição de *stop*, encerrando a comunicação [16].

3.5. Módulo Bluetooth

Componente adquirido ainda no projeto anterior. Sua função é prover a comunicação entre a plataforma e um computador com algum grau de liberdade, já que o uso de cabos seria dificultado pelo movimento do carro.

O módulo usado é o HC-05, que pode ser visto na Figura 15, e seu funcionamento é equivalente ao de uma porta serial comum: ele permite a comunicação serial (um bit de cada vez) entre dois dispositivos, que assumem a condição de mestre ou escravo. Transmissão e recepção são realizadas através de uma pequena antena embutida no módulo, que funciona como uma "ponte" entre o Arduino e o computador. O módulo possui dois pinos de alimentação, dois pinos de comunicação serial UART (TX e RX, que são conectados ao RX e TX do Arduino, respectivamente), um pino de WAKEUP e um pino de STATE, que não são utilizados neste projeto [20]. Uma explicação mais detalhada sobre UART pode ser encontrada na seção 3.5.2.

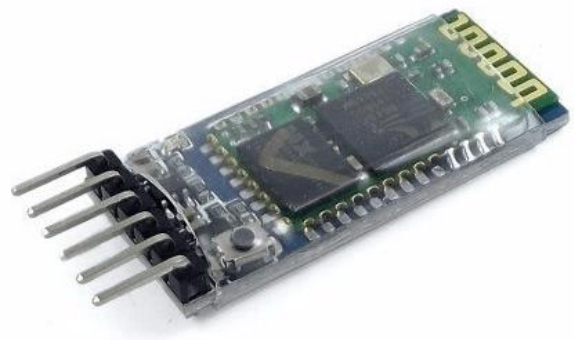

Figura 15 - Módulo Bluetooth HC-05

3.5.1 Bluetooth

O Bluetooth é um padrão *wireless* criado em 1994 que utiliza ondas de rádio para estabelecer uma comunicação entre dispositivos relativamente próximos. Foi desenvolvido e é regulado pelo Bluetooth Special Interest Group (SIG) [21]; a tecnologia opera na frequência de 2,4 GHz, pois esta é uma faixa de frequência livre (em outras palavras, qualquer produto pode utilizá-la sem necessidade de se obter uma licença para isto) [22]; de fato, esta é a mesma frequência utilizada em muitos outros aparelhos comuns (roteadores *WiFi*, mouses e teclados sem fio, portões de garagem, etc.).

A tecnologia Bluetooth se destaca de outros protocolos de comunicação sem fio por ter, em geral, um baixo consumo de energia e possuir uma operação fácil e intuitiva para o usuário. Estas características tornam o Bluetooth uma boa opção para aparelhos como telefones celulares, computadores e *tablets* [22].

A comunicação em si ocorre através de uma pequena "rede" criada por qualquer dispositivo Bluetooth, à qual podem se conectar outros aparelhos; uma vez estabelecida a conexão, um dos dispositivos deve tomar para si o papel de mestre, sendo, portanto, responsável por iniciar a comunicação com os demais, chamados de escravos [22].

3.5.2 UART

O UART, *Universal asynchronous receiver/transmitter* (ou transmissor/receptor universal assíncrono), é o componente responsável por receber dados de um sistema e os transmitir de forma serial. Também é responsável por realizar o processo inverso, ou seja, receber dados serialmente e remontá-los. Os UARTs contam com um registrador de deslocamento, que é responsável pela conversão serial/paralela [23].

No processo de envio do byte são adicionados bits especiais no início e no fim do campo de dado que se deseja enviar. O primeiro bit indica que um novo sinal está sendo enviado, depois seguem os próximos cinco a nove bits que representam a informação que se deseja enviar e por último são enviados os bits de parada. Através desse último bit o receptor é capaz de identificar que recebeu a informação completa [23].

As operações realizadas pelo UART são controladas por um sinal de *clock* interno. No processo de recepção, o módulo testa continuamente a linha em busca de um sinal de partida. Uma vez detectada a partida, os bits são recebidos e o byte é decomposto. Quando chegam todos os bits de informação eles são então enviados na forma de byte para o sistema ao qual estão conectados. No processo de transmissão, ao receber o byte, o módulo gera e envia na ordem o bit de partida, os bits de informação e o bit de parada; durante esse processo o UART sinaliza através de uma *flag* que está ocupado, para que o sistema aguarde antes do envio de novas informações [23].

3.6. Motores DC

O motor é um mecanismo que transforma diversas formas de energia (térmica, elétrica, etc.) em energia mecânica. Um motor de corrente contínua, ou motor DC, utiliza energia elétrica na forma de uma corrente contínua para realizar essa conversão [24].

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, tradução nossa) [25]:

> "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."

A função do conjunto de quatro motores DC utilizado no projeto é fornecer a movimentação da plataforma através da rotação de pequenas rodas, encaixadas diretamente no eixo de rotação dos motores. A Figura 16 apresenta o motor escolhido para o projeto. Os motores são alimentados pelo *driver* presente na *motor shield*, conforme explicado na seção 3.4.

Figura 16 - Motor DC utilizado no projeto

Os motores são dispostos na plataforma conforme a Figura 17. A numeração dos motores é feita desta forma para facilitar a escrita do código.

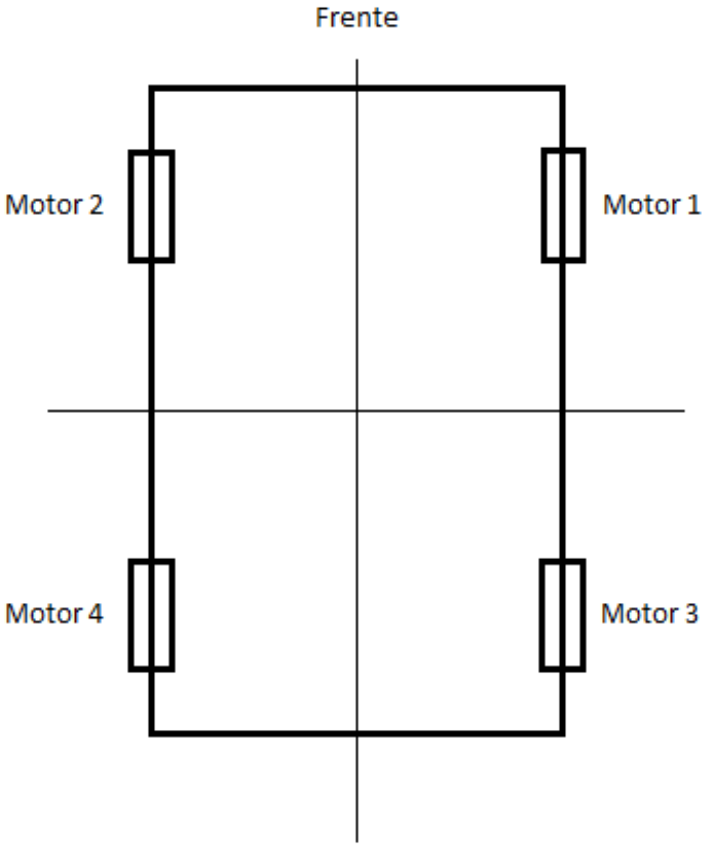

Figura 17 - Disposição dos motores na plataforma

3.7. Baterias

O sistema é alimentado por um conjunto de quatro baterias de íon-lítio de 3,7V cada, produzidas pela UltraFire. Cada bateria é do modelo 18650 e possui uma capacidade de 6000mAh; são baterias recarregáveis, o que suprime a necessidade de se trocar a alimentação do circuito de tempos em tempos.

Os motores exigem uma tensão de aproximadamente 7,5V. Para alcançar esta tensão, as baterias são colocadas em série, duas a duas; depois, os pares são colocados em paralelo, a fim de aumentar a capacidade de carga do circuito. Este arranjo é ilustrado na Figura 18.

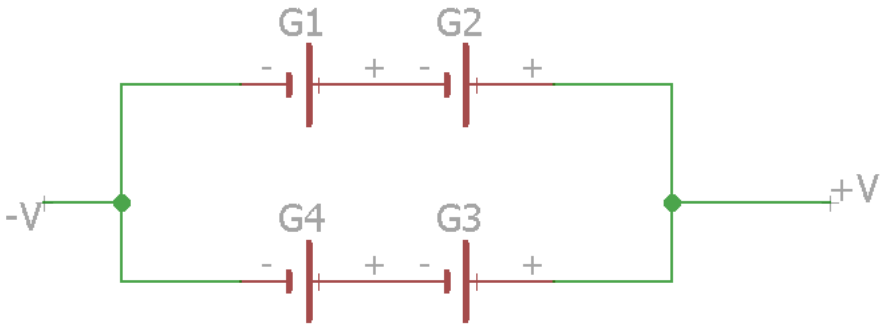

Figura 18 - Arranjo de baterias

As baterias são ligadas a um chicote para facilitar a entrega de alimentação. Por sua vez, o chicote é ligado a um interruptor, que controla a conexão entre a entrada de alimentação da *Motor Shield* e as baterias. Assim, é possível controlar a ativação dos motores sem retirar a alimentação da *Shield*.

3.8. Carregador de baterias portátil

Power Banks são dispositivos recentes no mercado. *Smart-phones* e *tablets* atuais não possuem a capacidade de durar por mais de um dia longe da tomada. Uma solução para esse problema foi o uso dos *Power Banks*. Esses dispositivos são carregados eletricamente em algum momento prévio e podem ser transportados por serem baterias portáteis. Como seu uso eventualmente elas devem ser carregadas novamente, mas garantem um maior tempo de utilização de aparelhos eletrônicos [26].

A necessidade de diferenciar a alimentação da *shield* dos motores e a Arduino, fez com que fosse escolhido esse tipo de equipamento. Por ser um produto comum no mercado e por sua saída USB ele era de fácil acesso e implementação. A Figura 19 mostra esse aparelho. A alimentação via USB se mostrou mais eficaz para a Arduino, pois existe algum problema, não explorado por esse trabalho, com a alimentação via conector P4. No capítulo 5, entramos em maiores detalhes com relação a esses problemas.

Figura 19 - Carregador de baterias portátil

Esse dispositivo é composto de uma bateria e de um circuito simples. O circuito é formado basicamente por resistores e capacitores e um CI para regulação da carga e descarga das baterias. A carga das baterias pode ser feita por um carregador de parede ou um computador via USB. Algumas versões apresentam LEDs para indicar quando as baterias estão sendo utilizadas, seja em carga ou em descarga.

3.9. Placa de conexões

O projeto anterior contava com uma placa que fornecia a comunicação entre o Arduino, os sensores, o módulo Bluetooth e o multiplexador que determinava os sensores que estavam sendo lidos. Como o projeto atual eliminou a necessidade desse multiplexador, foi necessário confeccionar uma nova placa para estabelecer esta comunicação.

Foi usada uma placa padrão quadrada com lado de 10cm. Este tipo de placa possui uma estrutura de fenolite, um isolante elétrico. Em uma de suas faces ela possui pequenas ilhas condutoras, que servem como base para a solda de componentes ou fios.

A placa utilizada neste projeto possui um desenho simples, que inclui um conector macho com 26 pinos dispostos em duas fileiras de 13, que fornece a comunicação com o Arduino através de um cabo plano; 8 conectores fêmea de 4 pinos (a cada qual se conecta um sensor de ultrassom); um conector fêmea de 6 pinos que acomoda o módulo Bluetooth; e um regulador de tensão. O esquemático da placa está ilustrado na Figura 20 e o desenho da placa na Figura 21, ambos gerados com o software de design EAGLE.

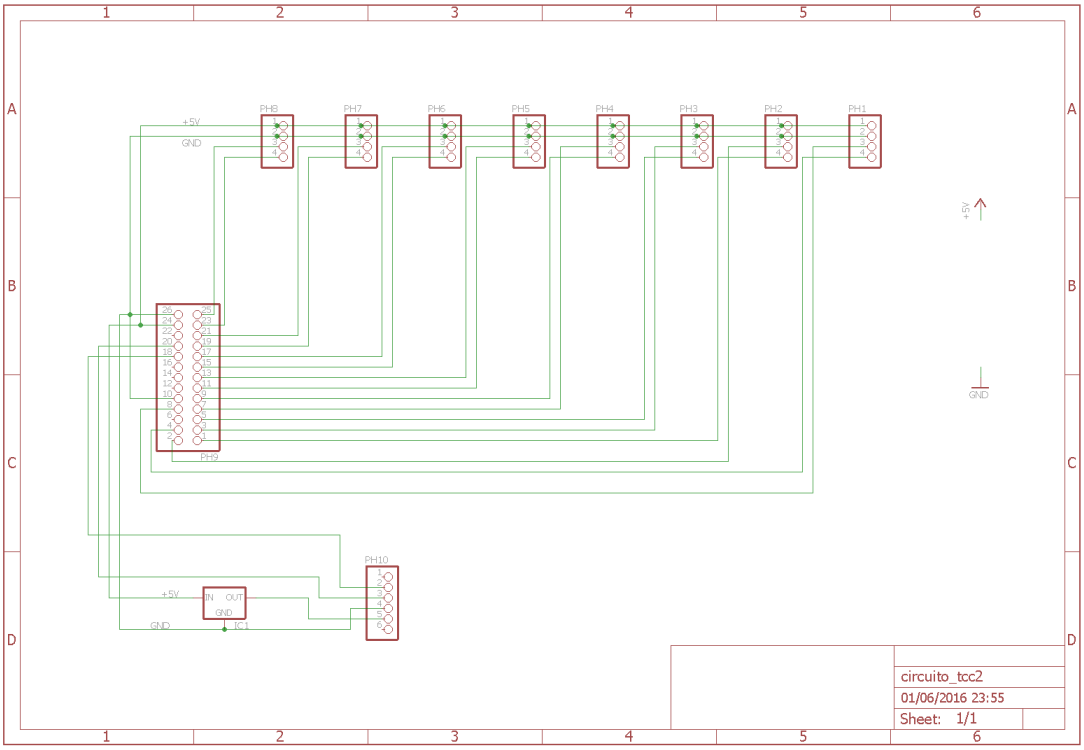

Figura 20 - Esquemático da placa de circuitos

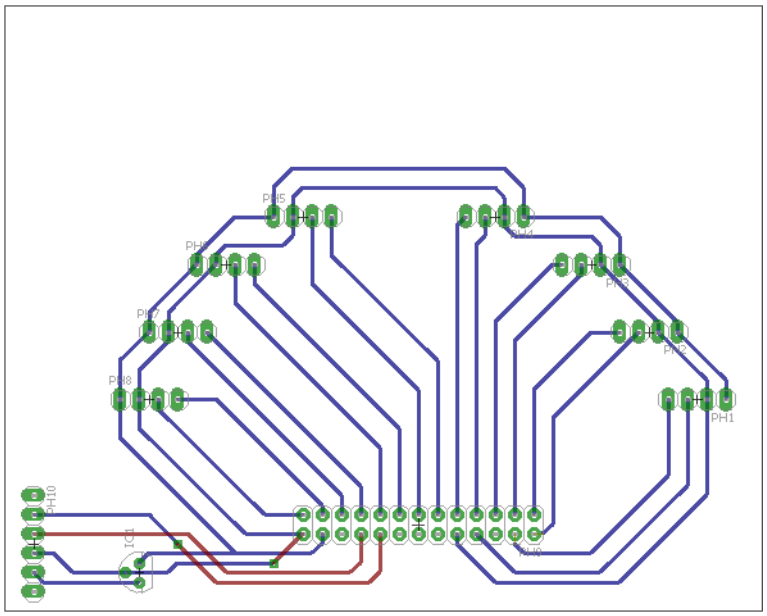

Figura 21 - Desenho da placa de circuitos, somente para ilustração

O desenho presente Figura 21 é somente para ilustração de como seria confeccionada a placa. Não houve impressão da mesma, pois optamos pela utilização de uma placa padrão. Foram utilizados fios em lugar das trilhas impressas para formar os caminhos condutores. No desenho, as vias (buracos condutores que atravessam a placa conectando duas camadas) representam os locais onde os fios se sobrepõem na placa real.

4. Programação

O programa está dividido em diversas funções, das quais podemos destacar o *firmware*, as interrupções, o algoritmo de cálculo de distância, o algoritmo de controle dos motores e a transmissão serial. Uma descrição mais detalhada sobre cada uma dessas partes será feita nesse capítulo.

4.1. Firmware

A mudança mais significativa implementada neste projeto tenha sido a programação do microcontrolador. A maior parte da lógica de operação da plataforma foi herdada do trabalho anterior; porém, a troca do Arduino Mega pelo Arduino DUE implicou em mudanças na sintaxe, no nome de registradores e na forma de acessá-los.

Entre as características do trabalho original que foram preservadas destaca-se o uso de bibliotecas para melhor compartimentalização do código – a comunicação Bluetooth, a inicialização dos sensores de ultrassom, a comunicação com a *motor shield* e o mecanismo de cálculo de distância e ângulo de fuga do carro. Dentre eles, os dois primeiros sofreram apenas as alterações necessárias para a transição entre placas, levando em conta a diferença entre os registradores envolvidos. Os dois últimos sofreram mudanças mais significativas, conforme explicado nas próximas seções.

Devido à extensão do código, outra prática continuada neste projeto foi a adoção de variáveis parametrizadas: quantidades como o tempo de interrupção do *timer* 5, número de sensores (alterado inúmeras vezes durante a fase de testes), número de entradas no vetor para o cálculo da média foram declaradas como constantes (#defines) e referidas diversas vezes no código pelo nome designado. Esta prática possibilita uma fácil manipulação destas variáveis e economiza tempo na fase de testes, pois na necessidade de uma mudança de parâmetros, basta mudar o valor da constante.

O *loop* principal do código é desenhado para esperar um ciclo de 50ms, sinalizado através de uma *flag*. O *timer* 5 do Arduino DUE (escolhido por continuar a convenção adotada pelo grupo anterior) é utilizado para cronometrar este tempo. Ele dispara uma interrupção a cada 0,8ms, incrementando um contador que, quando atinge o valor equivalente a 50ms, modifica o valor da *flag* e inicia um novo pulso nos sensores de ultrassom. Os oito sensores são disparados de forma sequencial dentro de um mesmo ciclo, ou seja, seus

disparos, para efeitos gerais, são ao mesmo tempo Paralelamente, as interrupções de porta ficam ativas, esperando a borda de descida no pino de *echo* de cada um dos sensores. Quando isto ocorre, a rotina de interrupção ativada por aquele sensor salva o valor do contador e também o valor do registrador do *timer* 5 para o cálculo da distância.

Enquanto não houver nenhum objeto em suas redondezas, a *flag* é mantida em zero, a plataforma fica parada aguardando a aproximação de algum obstáculo. Quando o *loop* principal percebe a *flag* com valor diferente de zero, ele realiza o cálculo necessário para a transformação do tempo de *echo* dos sensores para valores de distância. A seguir, como forma de prevenção de erros de leitura, estes valores passam por dois processos: o primeiro compara o valor atual com o último valor lido; se a diferença for discrepante, a nova leitura é ignorada. Este processo se repete duas vezes antes de aceitar valores díspares, a fim de se eliminar possíveis erros de leitura. O segundo processo é um cálculo de média levando em conta os últimos valores lidos; este processo filtra ruídos de alta frequência, eliminando ou diminuindo picos.

O valor da média com as últimas três leituras é então utilizado para o cálculo de ângulo de fuga do carro. Esta informação é utilizada para decidir o acionamento dos motores e a direção da rotação da plataforma. Completando o ciclo, dados como a direção adotada e o acionamento dos motores são enviados via Bluetooth para um computador. A Figura 22 ilustra como funciona a lógica do programa.

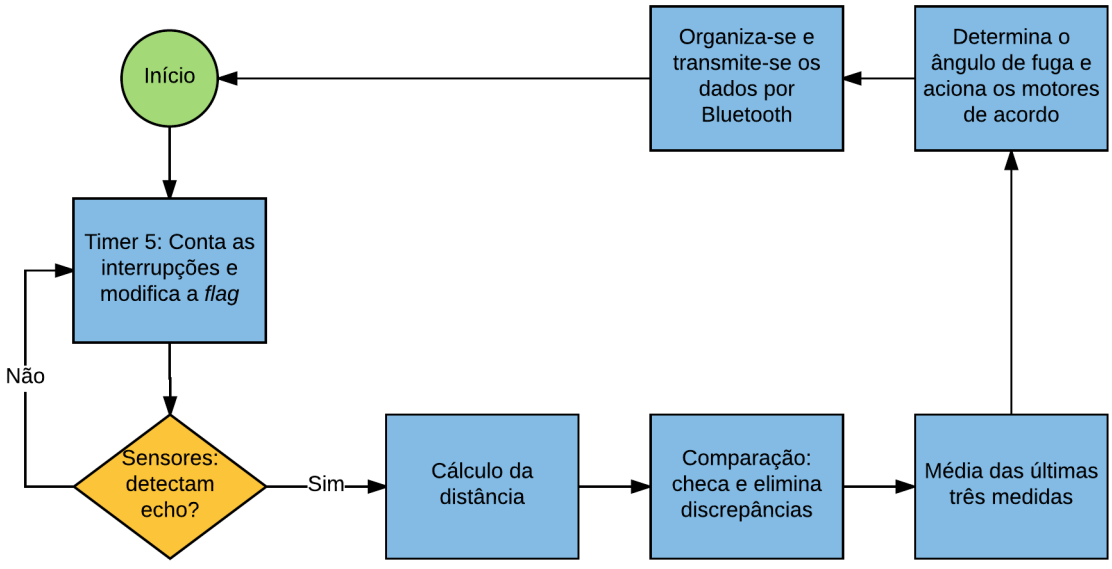

Figura 22 - Fluxograma da lógica do programa

4.2. Interrupções

Grande parte do projeto é baseado em interrupções, sejam elas de *timer* ou interrupções externas. Os dois tipos possuem as mesmas propriedades, mudando-se apenas a condição para que sejam acionadas. Uma interrupção é um recurso que permite ao microcontrolador "interromper" a tarefa atual (imediatamente após completar a instrução que está cumprindo), salvar o estado atual do código, colocando o ponteiro de programa em uma pilha, e chamar uma rotina especial, denominada de rotina de interrupção (ISR). Uma vez completa a rotina, o microcontrolador retorna para a próxima instrução do programa principal.

As interrupções de *timer* podem ser configuradas, como foram neste caso, para serem acionadas toda vez que o contador interno do *timer* atinge um valor pré-determinado. Uma vez que isto aconteça, o contador é reiniciado e o processo se repete.

As interrupções externas são processos semelhantes que são ativadas a partir de certa condição no pino. O SAM3X permite que esta condição seja uma mudança no nível (subida, descida ou ambos) ou um nível lógico (alto ou baixo). No presente caso, devido ao funcionamento próprio dos sensores de ultrassom, os pinos foram configurados para aguardar a borda de descida.

A interrupção externa foi um dos grandes motivadores da mudança de placa, pois o Arduino DUE permite interrupção em todos os pinos digitais, enquanto o Arduino MEGA disponibiliza apenas 6 pinos de interrupção externa (apesar de o microcontrolador ATMEGA2560 disponibilizar 8 pinos de interrupção, nem todos correspondem a pinos externos do Arduino MEGA) [11].

Quanto às rotinas de interrupção (também chamados de *handlers*), é preciso fazer uma observação. A IDE do Arduino disponibiliza a função *attachInterrupt*() para se ativar uma interrupção de pino. Esta função carrega como parâmetros o número do pino, o nome que se escolhe dar à ISR e o flanco ou nível lógico que ativará a interrupção. No Arduino DUE, esta função não é muito eficiente, pois os pinos são agrupados em blocos de 32 e cada um desses blocos possui apenas um vetor de interrupção, o que significa que uma interrupção em qualquer dos 32 pinos chamará o mesmo ISR padrão. Este ISR, por sua vez, realizará uma busca dentro do vetor de interrupção e chamará a rotina correspondente criada com a função *attachInterrupt*() [27].

A alternativa ao uso desta função seria modificar o código presente na IDE para reduzir o intervalo da busca; porém, esta prática dificultaria a portabilidade do código. Ao modificar o código presente na IDE, sua modificação se mantém somente no computador que fez a implementação no Arduino. Dessa forma, qualquer computador diferente do originalmente utilizado teria de fazer essa mudança nas bibliotecas da IDE. Além desse problema seria um trabalho exaustivo de modificar o código novamente caso se decida mudar os pinos destinados à interrupção. Por este motivo, decidiu-se adotar a função *attachInterrupt*(), confiando que a alta velocidade do SAM3X compensasse esta baixa eficiência.

4.3. Cálculo de distâncias e ângulo de fuga

O cálculo de distâncias não sofreu alterações. A fórmula disponibilizada na *datasheet* do sensor HC-SR04 é a seguinte:

$$d(cm) = \frac{w(\mu s)}{58},$$

em que *d* é a distância em centímetros e *w* é a duração do pulso de *echo*. Para descobrir *w*, utilizam-se os valores de tempos que foram armazenados durante a rotina de interrupção do *timer* 5 subtraídos de 400 microssegundos (intervalo de tempo entre o final do pulso de *trigger* e o início do pulso de *echo*). Este valor ainda passa por processos de "limpeza", conforme explicado anteriormente.

O cálculo do ângulo de fuga é um processo simples: o menor valor de distância entre os oito sensores deve ser identificado, juntamente com o índice daquele sensor. Depois disto, basta escolher o ângulo diametralmente oposto à posição ocupada pelo sensor. O cálculo considerando leituras dos sensores vizinhos foi cogitado, mas deixado de lado em seguida, devido ao estreito "cone" de leitura dos sensores: as boas leituras se dão, no melhor dos casos (quando o obstáculo está virado na direção do sensor), em ângulos inferiores a 30°. Este ângulo máximo é mencionado na *datasheet* do sensor e confirmado através de ensaios próprios [28].

Uma *flag* é utilizada para indicar que o carro está girando, o que ele continuará fazendo até que o obstáculo mais próximo esteja às suas costas, momento em que ele iniciará a fuga do objeto. Enquanto a plataforma está girando, o código considerará as leituras dos sensores, mas não ativará rotinas dos motores, até que a menor leitura seja do sensor 4, na parte traseira do carro. Isto evita certa confusão entre os sensores: quando a plataforma gira, o obstáculo se aproxima ora de um sensor, ora de seu vizinho, que por vezes chamam rotinas de fuga diferentes; com esta medida, porém, o código entende que as leituras se referem a um mesmo obstáculo e só muda seu comportamento no momento da fuga.

4.4. *Motor Shield*

A Adafruit, empresa responsável pela fabricação da placa que aciona os motores, disponibiliza uma biblioteca própria para sua utilização. Como a decisão em favor da troca da placa foi mais ou menos tardia, escolheu-se utilizar esta biblioteca para otimizar o tempo de desenvolvimento.

Os principais métodos (já que a biblioteca é escrita em C++ e não em C como todo o resto do código) utilizados na manipulação dos motores são o *run()*, em que se define a direção de rotação do motor (para a "frente", para "trás" ou parado) e o *setSpeed()*, em que se define a velocidade da rotação em uma escala de 0 a 255.

O programa principal analisa condições como o ângulo de fuga e a distância do obstáculo mais próximo para determinar a forma como os motores serão acionados. Se um obstáculo é detectado na lateral dianteira direita da plataforma, por exemplo, os motores 1 e 4 (colocados do lado direito do carrinho) giram para a frente enquanto os motores 2 e 3 (à esquerda) giram para trás, fazendo o carrinho girar sobre o próprio eixo. Optou-se por herdar da nomenclatura do projeto anterior para a descrição das funções.

Existem seis rotinas que controlam acionamento dos motores e, portanto, o movimento da plataforma. Elas são acionadas conforme o ângulo de fuga calculado. A função *andar_esquerda()* é chamada quando há um objeto próximo aos sensores 5 e 6 (na parte traseira direita e na lateral direita, respectivamente), e aciona os motores com velocidade "baixa" (convencionou-se o valor de 40), dependendo do lado em que estão posicionados, conforme explicação acima. A função *andar_direita()*, acionada pelos sensores 2 e 3

(simétricos dos sensores 5 e 6), funciona exatamente da mesma maneira, apenas invertendo a direção de rotação dos motores.

As funções *virada_brusca_esquerda()* e *virada_brusca_direita()*, acionadas por leituras nos sensores 7 e 1, respectivamente, funcionam de forma semelhante às funções descritas acima; porém, adota-se uma velocidade de rotação maior para os motores (convencionou-se 70 para este caso).

A função *andar_frente()*, ativada quando o obstáculo se encontra atrás da plataforma, é a responsável por sua fuga. A velocidade de fuga é formulada de modo a permitir uma desaceleração do carro: inicialmente com uma velocidade alta, ele ficará mais lento conforme o objeto às suas costas se distancie. Por fim, a função *parado()* libera os motores, fazendo com que a plataforma pare. A liberação dos motores não implica em nenhum tipo de freio, portanto a plataforma continua se movendo por um breve período de tempo, devido à sua própria inércia.

4.5. Bluetooth

Apesar de o módulo Bluetooth ser o mesmo do projeto anterior, foram necessárias alterações no código, pois a Due não reconhecia alguns dos comandos escritos. Mas seu funcionamento continua semelhante ao do projeto anterior. Ele transmite dados de forma serial para o software Matlab [8].

Matlab é um software de cálculo numérico. Ele possui diversas opções de uso, cálculos matriciais, processamento de sinais, simulações de controle, entre outras. Um dos seus diferencias são as *toolbox*, que são pacotes de ferramentas específicos para determinada aplicação. No caso deste projeto uma *toolbox* utilizada foi a *Instrument Control Toolbox*, que dentre suas várias funções, gerencia as comunicações do software com o Bluetooth.

O código do Bluetooth funciona da seguinte forma, os dados escolhidos para transmissão são organizados em um vetor. O primeiro dado, gerado no ciclo do programa, é a *flag* que indica ao Matlab que novos dados estão sendo enviados. Em seguida são enviados os dados, armazenados nos *buffers*, na seguinte ordem: médias de cada sensor (do 0 ao 7), velocidades de cada motor (do 1 ao 4), direção decidida, distância do obstáculo, parte inteira do ângulo de fuga e parte fracionária do ângulo de fuga. Ao todo 17 dados diferentes, incluindo a *flag*, são enviados de forma serial, conforme pode ser visto na Figura 23.

SEQUÊNCIA DOS DADOS ENVIADOS	INFORMAÇÃO
DADO[0]	*Flag*
DADO[1]	Sensor[0].media
DADO[2]	Sensor[1].media
DADO[3]	Sensor[2].media
DADO[4]	Sensor[3].media
DADO[5]	Sensor[4].media
DADO[6]	Sensor[5].media
DADO[7]	Sensor[6].media
DADO[8]	Sensor[7].media
DADO[9]	VELOCIDADE_1
DADO[10]	VELOCIDADE_2
DADO[11]	VELOCIDADE_3
DADO[12]	VELOCIDADE_4
DADO[13]	Direção Decidida
DADO[14]	Distância do Obstáculo
DADO[15]	Parte inteira do *float* ângulo de fuga
DADO[16]	Parte fracionária do *float* ângulo de fuga

Figura 23 - Sequência de dados enviados

O código do Matlab foi herdado do projeto anterior, sem muitas modificações necessárias para que o código atendesse aos objetivos propostos [8]. Os dados são recebidos e organizados em gráficos para uma melhor visualização dos resultados obtidos. São gerados 4 gráficos: distância lida por cada sensor, posição e distância do objeto em relação ao carro, direção decidida para a fuga da plataforma e velocidade dos motores.

Uma mudança a ser mencionada foi a mudança do ângulo de fuga, anteriormente ele aparecia nas coordenadas do Matlab, sendo a frente o eixo x no sentido crescente. Optou-se por colocar a frente do carrinho como o eixo y crescente, de forma a termos na tela do software uma representação vista de cima do carrinho. Na Figura 24 a frente do carro agora se localiza na parte positiva do eixo x, ou seja, o obstáculo está na parte dianteira esquerda do carro.

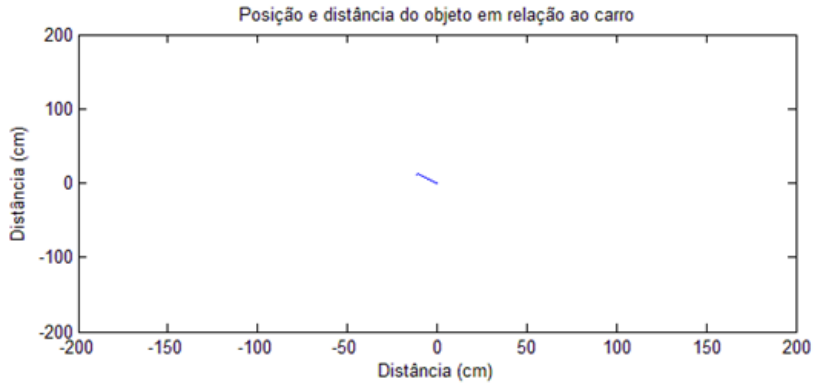

Figura 24 - Plot do ângulo de fuga

5. Ensaios e Testes

Após montar a plataforma e carregar o Arduino com o programa descrito na seção anterior, foram realizados alguns testes para verificar o funcionamento da plataforma e a comunicação entre ela e o computador. Os resultados dos testes são exibidos na tela do computador a partir de um *script* de Matlab, cujo conteúdo é apresentado no Apêndice B.6.

Observou-se que o sistema ao ser alimentado somente pelas baterias apresentava erros nas medições dos sensores. Um teste mais específico mostrou que a alimentação via USB não apresentava tais erros, enquanto a alimentação via P4 deixava alguns sensores com média mais baixa do que a real.

A adição de um carregador de bateria portátil com saída USB foi a solução tomada para resolver esse problema. Anteriormente as medidas dos sensores ficavam baixas, mesmo sem a presença de objetos nas suas imediações. A Figura 25 mostra os resultados dos sensores antes da troca da alimentação.

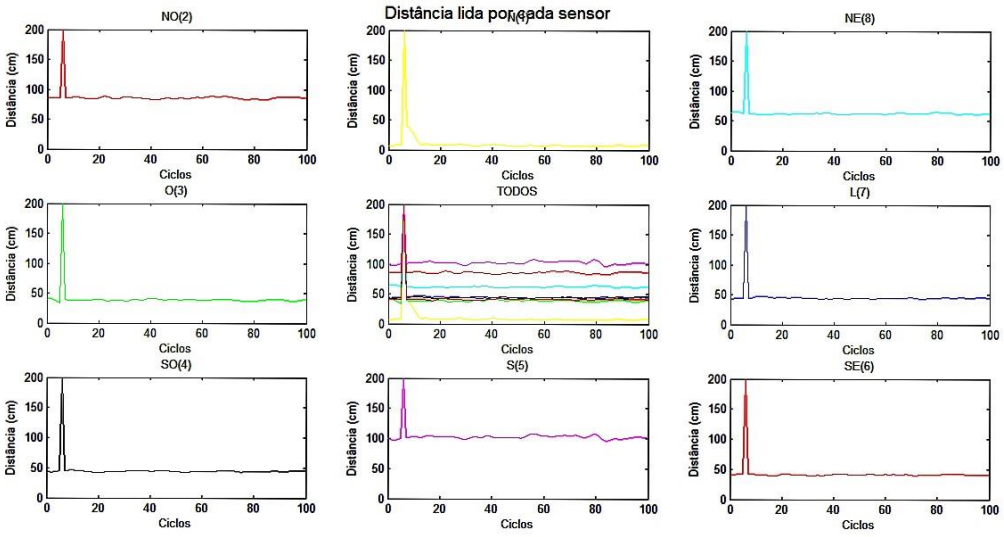

Figura 25 - Leitura dos sensores com alimentação via conector P4

Depois da troca os sensores começaram a se comportar da maneira desejada, dando os valores corretos das distâncias dos obstáculos em suas imediações. A Figura 26 mostra os resultados dos sensores após a troca da alimentação.

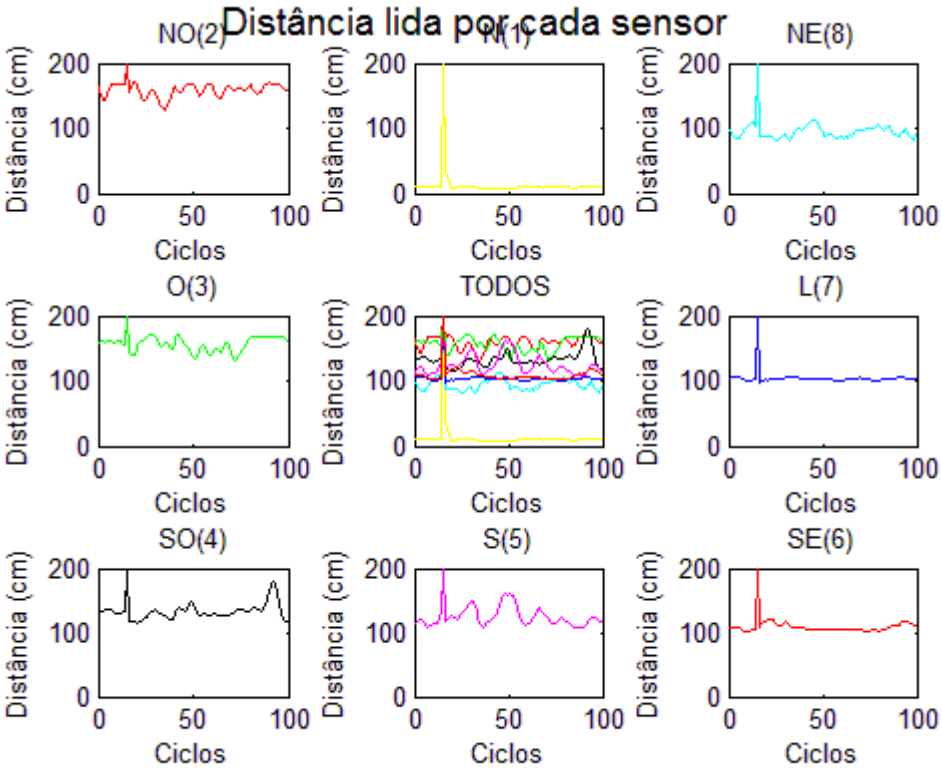

Figura 26 - Leitura dos sensores com conexão USB

O principal teste consiste em simular a aproximação de objetos: a plataforma é posicionada em uma área livre de obstáculos e o jumper de alimentação da *motor shield* é retirado, fazendo com que a plataforma fique parada. Então, um objeto (foi utilizado uma folha de papel para este propósito) é levado até as proximidades de um sensor na região frontal da plataforma. Simulando a rotação desta, aproxima-se o objeto dos próximos sensores, até que ele seja identificado pelo sensor traseiro. Quando isto ocorre, o objeto é lentamente afastado da plataforma, como ocorre quando os motores estão ligados.

Os primeiros resultados obtidos com este teste são apresentados acima na Figura 26. Ela mostra o histórico de leitura de cada um dos oito sensores; o Sensor 1, à frente da plataforma, identifica o objeto e, portanto, apresenta leituras mais baixas que os demais. O gráfico da Figura 27 confirma a posição do objeto à frente do carro.

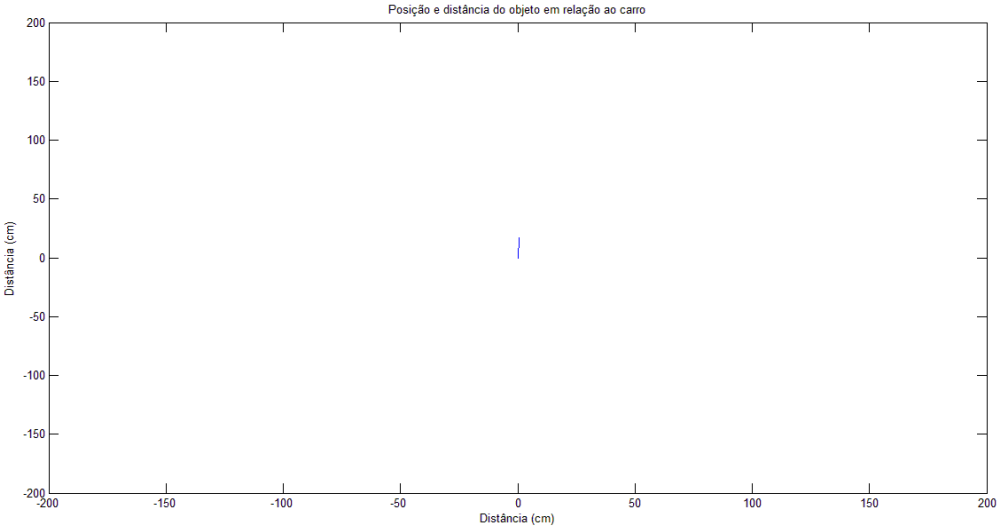

Figura 27 - Posição do objeto em relação à plataforma

A seta aponta para a frente, onde o objeto se encontra; o comprimento da seta é proporcional à distância até o obstáculo, e neste caso o obstáculo está muito próximo, conforme demonstrado na Figura 27.

Também são capturadas informações sobre a direção adotada para a fuga do carro. Se o obstáculo estiver à frente da plataforma, convencionou-se que a plataforma giraria para a direita. De fato, a direção exibida na Figura 28 é a direita.

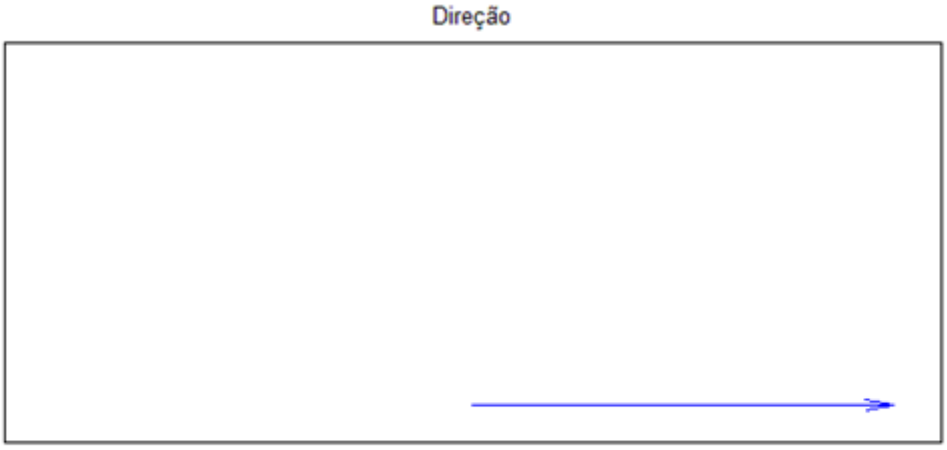

Figura 28 - Direção de fuga da plataforma com objeto à frente

O último dado coletado é a velocidade dos motores, ilustrada no gráfico da Figura 29.

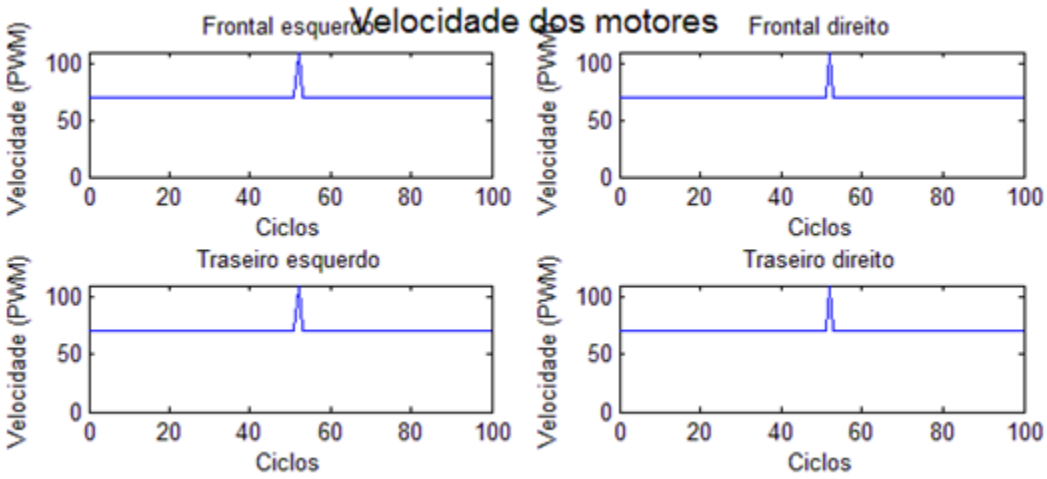

Figura 29 - Velocidade de acionamento dos motores com objeto à frente

Os motores são acionados com velocidade 70, uma velocidade relativamente alta, pois a fuga de um objeto à frente é considerada “mais urgente” pela plataforma. É importante notar que estes gráficos mostram apenas o módulo da velocidade de cada motor, e não a direção de rotação. Como o veículo deve girar para a direita, os motores do lado direito rodam “para trás”, enquanto os do lado esquerdo giram para “a frente”.

A seguir, o objeto é aproximado do Sensor 2 para simular a rotação da plataforma. O gráfico da Figura 30 mostra a mudança nas leituras.

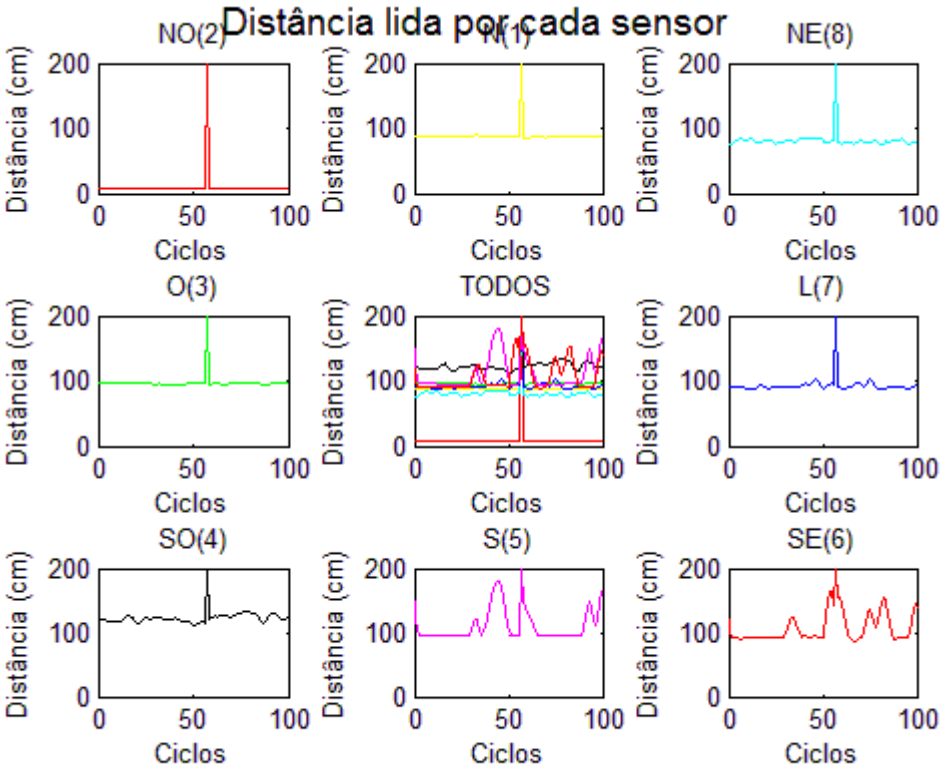

Figura 30 - Leitura dos sensores com objeto próximo ao Sensor 2

Desta vez, a leitura do Sensor 2 é a mais baixa. De fato, o objeto é reconhecido na lateral dianteira esquerda da plataforma, de acordo com o gráfico da Figura 31.

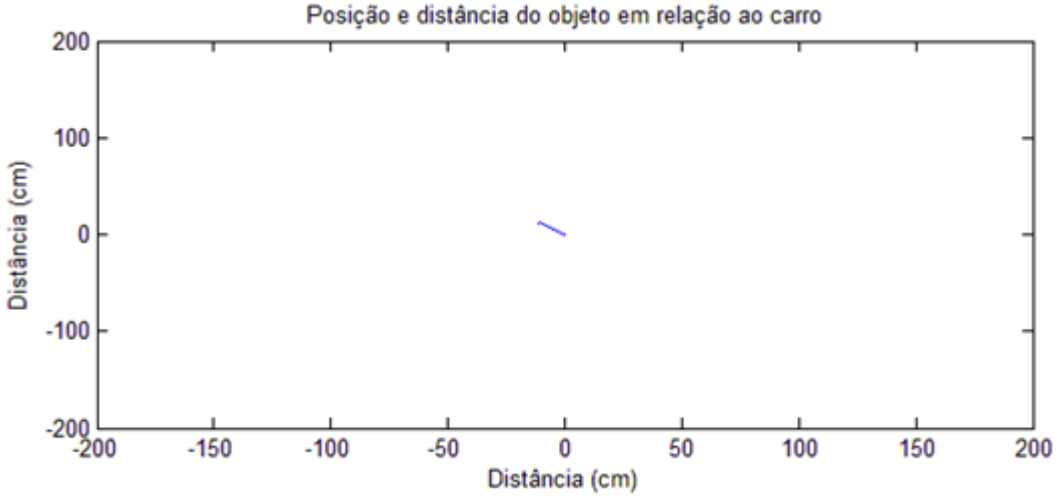

Figura 31 - Posição do objeto quando próximo ao Sensor 2

Os gráficos de velocidade dos motores e da direção de fuga da plataforma são idênticos aos anteriores e estão exibidos nas Figura 32 e Figura 33.

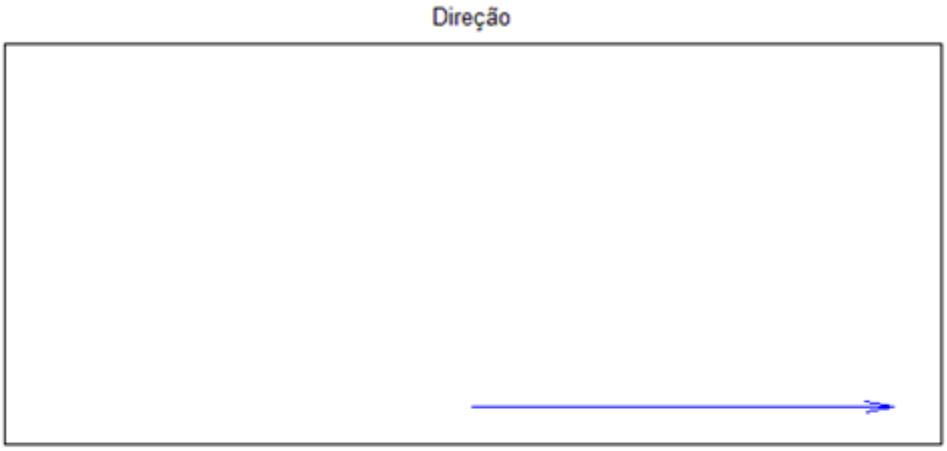

Figura 32 - Direção de fuga da plataforma com obstáculo no sensor 2

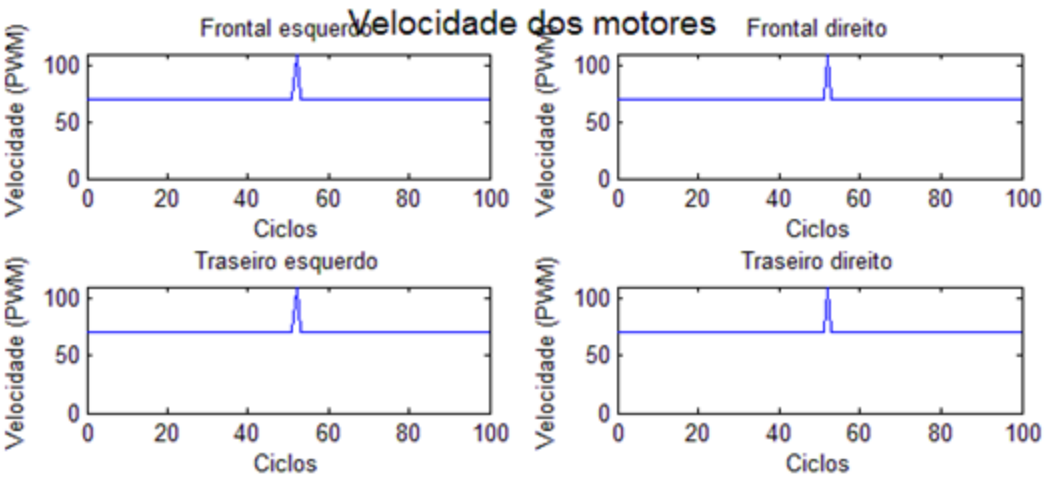

Figura 33 - Velocidade de acionamento dos motores com obstáculo no sensor 2

Continuando a simulação de rotação da plataforma, aproximou-se o objeto do Sensor 3, na lateral esquerda do carro, o que gera os gráficos apresentados na Figura 34.

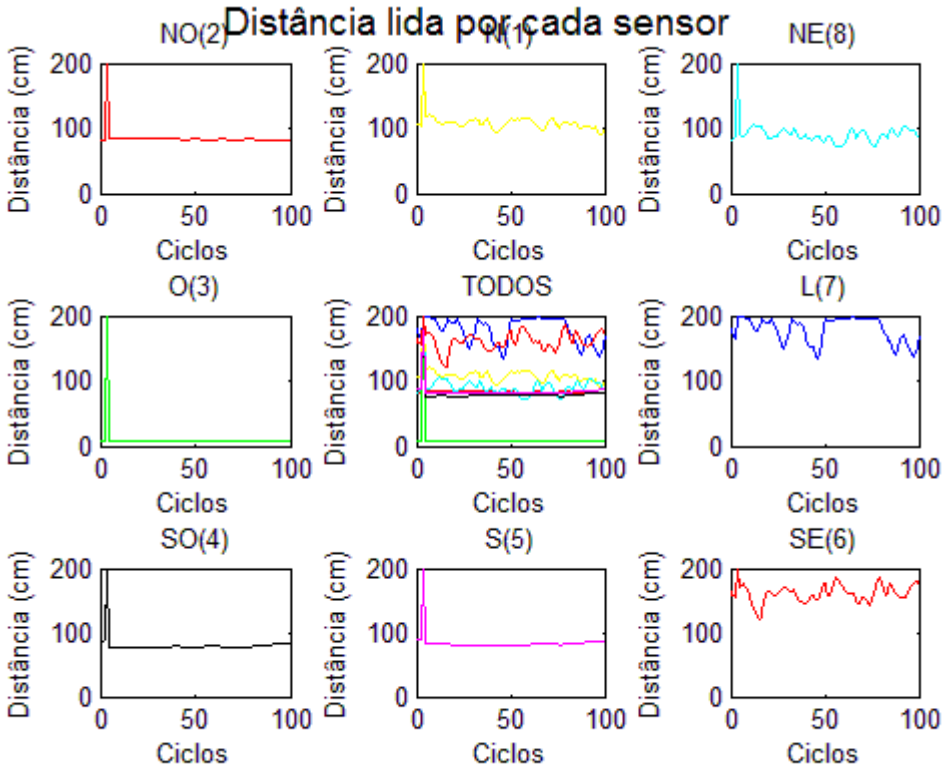

Figura 34 - Leitura dos sensores com objeto no lateral esquerda

A Figura 34 mostra a leitura do Sensor 3 mais baixa que as demais, indicando a presença de um obstáculo naquela região. O gráfico da Figura 34 também acusa a proximidade do objeto nesta direção, conforme pode ser visto na Figura 35.

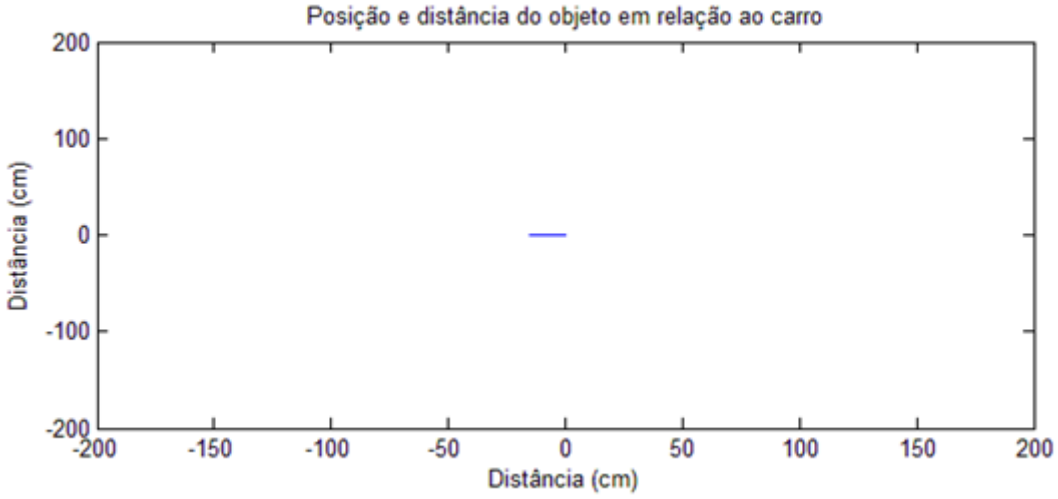

Figura 35 - Posição do objeto quando próximo ao Sensor 3

Neste caso, a velocidade dos motores é alterada: como o obstáculo não está mais na parte frontal da plataforma, deixa de ser considerado uma ameaça “urgente”, e a velocidade de rotação dos motores é reduzida. Este efeito pode ser contemplado na Figura 36.

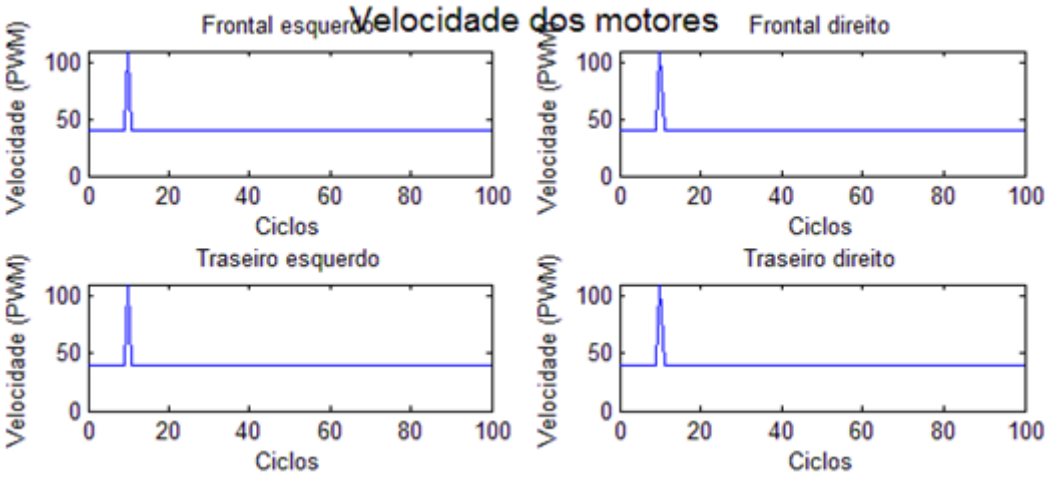

Figura 36 - Velocidade de acionamento dos motores com objeto à esquerda

Quando o objeto é colocado próximo ao Sensor 4, na lateral traseira esquerda da plataforma, obtém-se os gráficos exibidos na Figura 37.

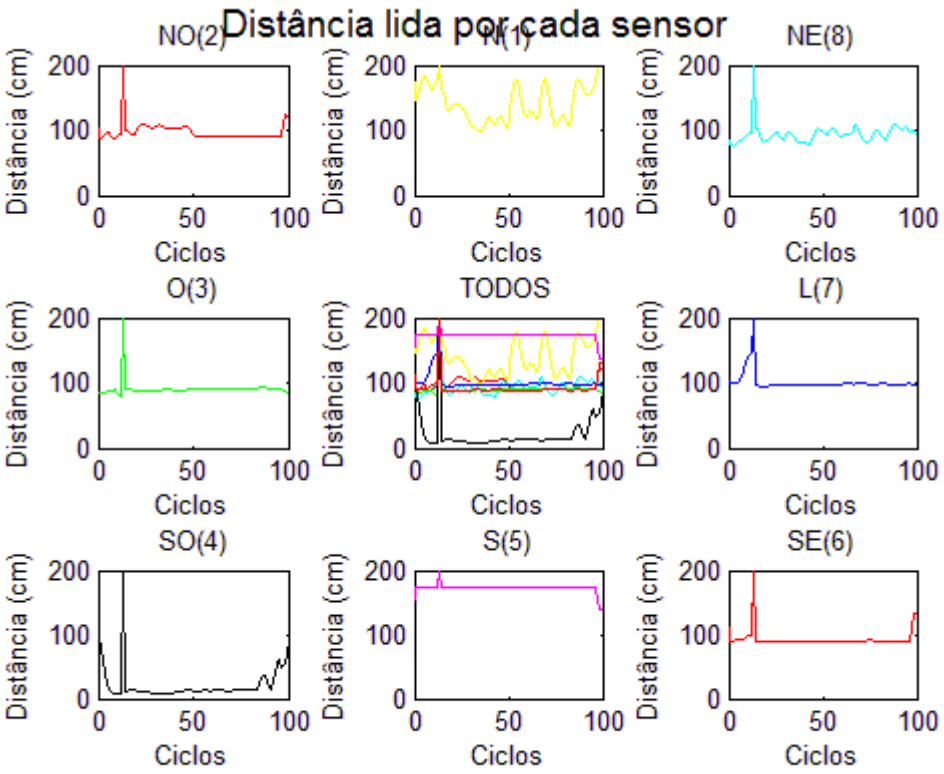

Figura 37 - Leitura dos sensores com objeto próximo ao Sensor 4

Como esperado, a leitura do Sensor 4 apresenta valores baixos, indicando a pequena distância até o obstáculo. A Figura 38 ilustra o vetor de distância até o objeto.

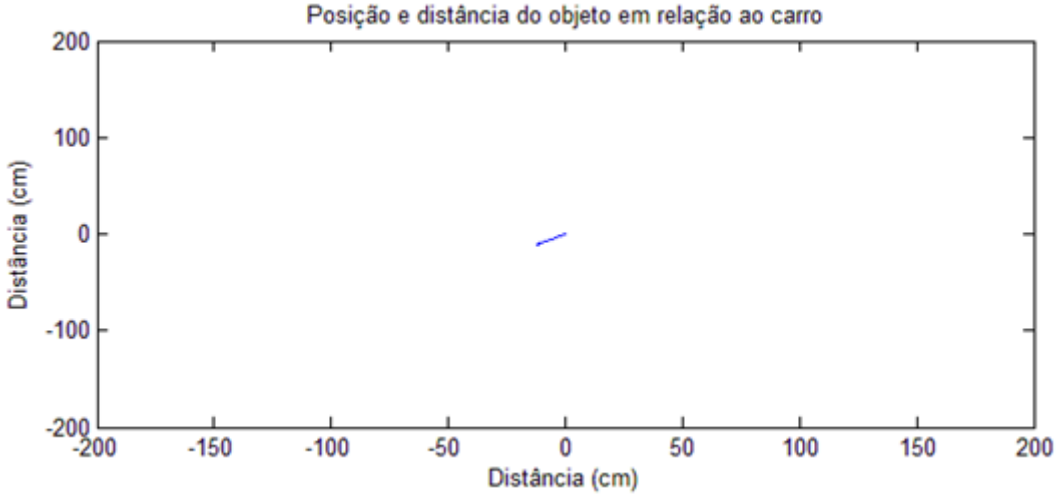

Figura 38 - Posição do objeto próximo ao Sensor 4

A velocidade dos motores é mantida baixa e a direção de fuga continua sendo a direita. Por fim, aproxima-se o objeto do sensor traseiro, de número 5. A detecção do obstáculo é ilustrada na Figura 39 e na Figura 40.

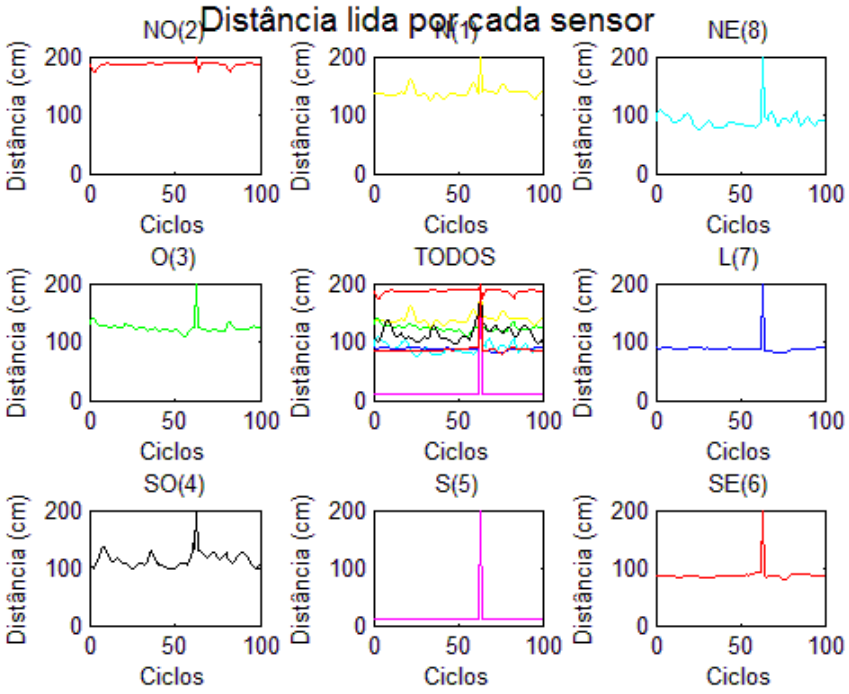

Figura 39 - Leitura dos sensores com objeto atrás da plataforma

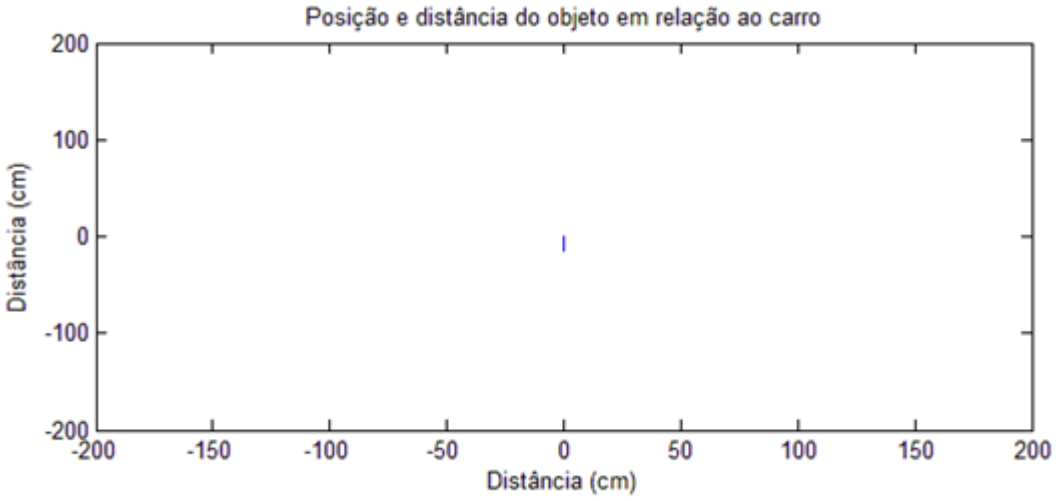

Figura 40 - Posição do objeto atrás da plataforma

Como em todas as leituras anteriores, os resultados estão de acordo com o esperado e são coerentes entre si. Finalmente, a direção de fuga é alterada, conforme Figura 41.

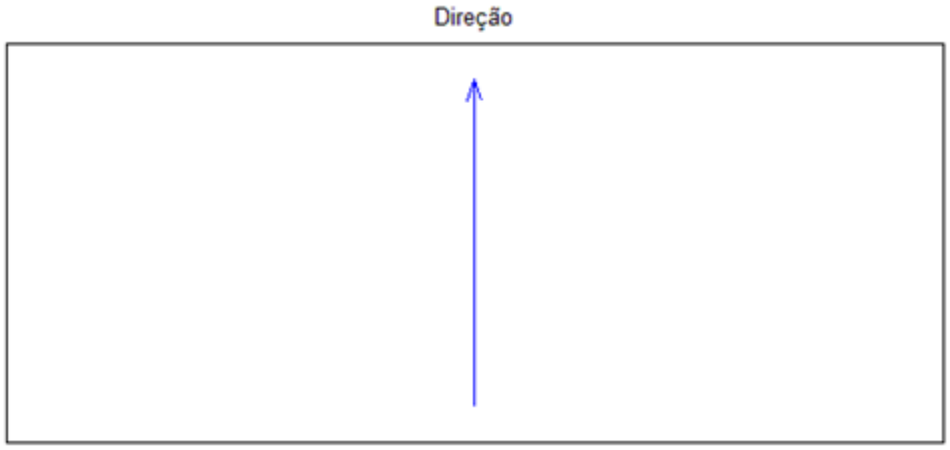

Figura 41 - Direção de fuga com objeto atrás da plataforma

A rotação dos motores também é alterada. Conforme explicado na seção 4.4, a velocidade dos motores será alta, pois o objeto se encontra muito próximo à plataforma. De fato, o gráfico da Figura 42 exibe altos valores para a velocidade.

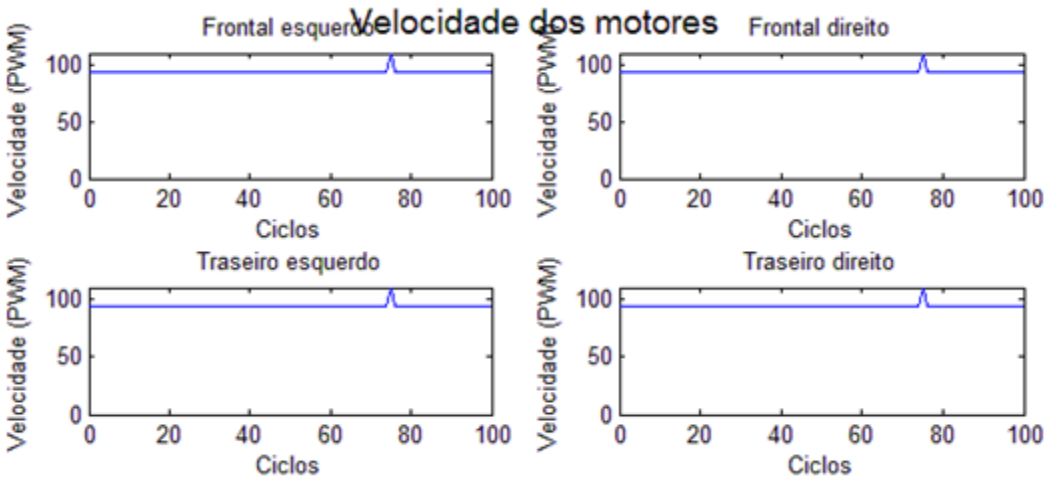

Figura 42 - Velocidade dos motores com objeto atrás da plataforma

Estes resultados nos permitem concluir que o veículo reage conforme o previsto à aproximação de obstáculos. Quando um obstáculo se aproxima pelo lado esquerdo da plataforma (ou, neste caso, pela frente), o programa se encarrega de rotacionar a plataforma para a direita até que o obstáculo esteja atrás do veículo. Então, os motores são acionados com alta velocidade para que o carro se mova para a frente.

Um teste semelhante foi realizado com os sensores do lado direito, com resultados idênticos ou simétricos, quanto aos gráficos de direção de fuga e posição do objeto. Optou-se por não apresentar tais gráficos neste trabalho; por possuírem resultados simétricos aos já obtidos, sua inclusão seria repetitiva.

Os testes exibidos acima foram estáticos, isto é, com a plataforma parada; é difícil, a partir apenas deste ensaio, de visualizar as melhorias implementadas neste projeto. De fato, tais melhorias só são evidenciadas com a plataforma em movimento, caso em que é perceptível o reconhecimento de objetos de maneira mais rápida do que ocorria no projeto anterior.

Para observar essa maior velocidade de detecção dos obstáculos, foi confeccionado um vídeo. O vídeo mostra o comportamento da nova plataforma com relação a obstáculos localizados em sua proximidade e a velocidade de resposta do carro a esses. O link para o vídeo é https://youtu.be/vYUb5Fh1jgQ.

6. Considerações finais

A robótica, atualmente, vem sofrendo grandes saltos em períodos curtos de tempo. Todo dia somos surpreendidos com alguma inovação que cria maior interesse nas pessoas pela área. A facilidade com que alguns microcontroladores estão disponíveis para acesso e uso imediato aumenta o número de avanços que podem ser feitos. O projeto do “carrinho”, nome carinhoso da plataforma, chamou nossa atenção. Era um projeto que já havia sido alterado por outro grupo, mas que ainda possuía muita possibilidade de melhora.

O maior problema da plataforma era a eficiência da mesma e optou-se por resolver esse problema com a troca do processador. Um processador com maior capacidade de *clock* e com mais interrupções externas simplificaria o código, consequentemente, deixando a resposta mais rápida e precisa.

Foram feitas as alterações necessárias no projeto e sua implementação se mostrou correta. No entanto houve vários problemas no decorrer do projeto. O principal problema sendo a Arduino Due. Diferente dos outros processadores da Arduino, que trabalham em sua maioria com a família AVR, o Due trabalha com a ARM. Dessa forma as bibliotecas e os *forums* de outras linhas de Arduino não serviriam para o nosso caso. Foi necessária uma leitura a fundo da *datasheet* do processador ARM para que fosse possível o desenvolvimento do código da plataforma.

O *datasheet* da ARM foi outro dos problemas encontrados. Diferente dos outros *datasheets* de processadores do Arduino, o da ARM não contava com exemplos para suas implementações. Ele apresentava várias informações e como alterá-las, mas sem em nenhum momento exemplificando como fazê-las.

O projeto ainda possui diversos aspectos a serem melhorados e várias novas funções a serem adicionadas. O novo processador, por sua maior eficiência, permitirá uma série de novas aplicações.

Algumas dessas melhorias que podem vir de projetos futuros são a criação de uma aplicação em Java para o controle do Bluetooth, a implementação de sensores de som para identificar sinais como assovio e a criação da função freio. O Matlab, apesar de sua grande capacidade de processamento, só é usado para a geração dos gráficos de dados. Uma aplicação em Java, responsável por receber esses dados e plotar os gráficos, seria mais rápida

e de maior portabilidade. O uso de sensores de som permitiria que a plataforma identificasse um sinal e se movesse na direção do mesmo, desviando de obstáculos no caminho. Uma função freio permitiria que o carro, enquanto foge de um obstáculo, ao identificar algo a sua frente, ele freie e evite a colisão.

As ideias descritas sobre projetos futuros são só alguns dos exemplos do que a plataforma ainda tem a oferecer. A área de robótica ainda possui muita capacidade de crescimento, assim como a plataforma. O próximo grupo a trabalhar com o projeto contribuirá com novas ideias, tornando o carro mais robusto, inteligente e prático.

Referências bibliográficas

[1] (1979). Robot Institute of America.

[2] *CCSI - Parts of a Robot*. (s.d.). Acesso em 12 de 05 de 2016, disponível em http://www.mind.ilstu.edu/curriculum/medical_robotics/parts_of_robots.php

[3] *Jewish Encyclopedia*. (s.d.). Acesso em 12 de 05 de 2016, disponível em http://www.jewishencyclopedia.com/articles/6777-golem#1137

[4] *Leonardo3*. (s.d.). Acesso em 12 de 05 de 2016, disponível em http://www.leonardo3.net/leonardo/books%20I%20robot%20di%20Leonardo%20-%20Taddei%20Mario%20-%20english%20Leonardo%20robots%201.html

[5] Čapek, K. (1920). *R.U.R. (Rossumovi Univerzální Roboti).*

[6] *Robotics.org*. (s.d.). Acesso em 12 de 05 de 2016, disponível em http://www.robotics.org/joseph-engelberger/unimate.cfm

[7] Heath, S. (2003). *Embedded systems design.* Newnes.

[8] Araújo, A., & Santana, L. (2015). *Plataforma móvel com detecção de obstáculos.* Universidade de Brasília, Brasília.

[9] Alves, A. B. (2014). *Plataforma móvel com detecção de obstáculos.* Brasília, DF, Brasil: Departamento de Engenharia Elétrica, Universidade de Brasilia.

[10] Atmel Corporation. (2012). *8-bit Atmel Microcontroller with 64K/128K/256K Bytes In-System Programmable Flash.* San Jose, CA, EUA: Atmel Corporation.

[11] Arduino. (s.d.). *ATmega2560-Arduino Pin Mapping*. Acesso em 30 de Maio de 2016, disponível em Arduino: https://www.arduino.cc/en/Hacking/PinMapping2560

[12] ElecFreaks. (s.d.). *Ultrasonic Ranging Module HC - SR04*. Acesso em 31 de Maio de 2016, disponível em http://www.micropik.com/PDF/HCSR04.pdf

[13] Arduino. (s.d.). *Compare Board Specs*. Acesso em 31 de Maio de 2016, disponível em Arduino: https://www.arduino.cc/en/Products/Compare

[14] Arduino. (s.d.). *Introduction*. Acesso em 07 de Junho de 2016, disponível em https://www.arduino.cc/en/Guide/Introduction

[15] Arduino. (s.d.). *Arduino Due*. Acesso em 31 de Maio de 2016, disponível em Arduino: https://www.arduino.cc/en/Main/ArduinoBoardDue

[16] Phillips Semiconductors. (2003). *AN10216-01 I2*. Acesso em 31 de Maio de 2016, disponível em http://www.nxp.com/documents/application_note/AN10216.pdf

[17] Toshiba. (s.d.). *TB6612FNG*. Acesso em 31 de Maio de 2016, disponível em https://cdn-shop.adafruit.com/datasheets/TB6612FNG_datasheet_en_20121101.pdf

[18] NXP Semiconductors. (2015). *PCA9865*. Acesso em 31 de Maio de 2016, disponível em http://www.nxp.com/documents/data_sheet/PCA9685.pdf

[19] Fried, L. (s.d.). *Adafruit Motor Shield v2 for Arduino*. Acesso em 31 de Maio de 2016, disponível em Adafruit Learn: https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino?view=all

[20] HC. (s.d.). *User Instructional Manual*. Acesso em 10 de Junho de 2015, disponível em http://www.rcscomponents.kiev.ua/datasheets/hc_hc-05-user-instructions-bluetooth.pdf

[21] Bluetooth. (s.d.). *What is Bluetooth Technology*. Acesso em 2 de Junho de 2016, disponível em https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth

[22] Electronical Code of Federal Regulations. (2016). *Part 15 - RAdio Frequency Devices*. Acesso em 8 de Junho de 2016, disponível em http://www.ecfr.gov/cgi-bin/text-idx?SID=a49d79db23ad60e02ee258da7668084e&mc=true&node=pt47.1.15&rgn=div5#se47.1.15_1247

[23] Texas Instruments. (2010). *Universal Asynchronous Receiver/Transmitter (UART)*. Acesso em 8 de Junho de 2016, disponível em http://www.ti.com.cn/cn/lit/ug/sprugp1/sprugp1.pdf

[24] Chapman, S. (2005). *Electric Machinery Fundamentals* (4ª ed.). New York: McGraw-Hill.

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

[26] *EasyAcc*. (s.d.). Acesso em 31 de 06 de 2016, disponível em https://www.easyacc.com/media-center/what-is-power-bank/

[27] Atmel. (2015). *SAM3X/SAM3A Datasheet Rev 11057C.* San Jose.

Referências de ilustrações

Figura 1 – Brokk 400. Acesso em 31 de Maio de 2016, disponível em http://www.brokk.com/us/400/

Figura 2 – Gary Lucas. Acesso em 31 de Maio de 2016, disponível em http://www.garylucas.com/www/golem/

Figura 3 – Share In. Acesso em 31 de Maio de 2016, disponível em http://sharein.org/microprocessor-microcontroller/

Figura 4 – [8]

Figura 8 – Arduino. Acesso em 31 de Maio de 2016, disponível em http://www.arduino.cc/en/uploads/Main/A000062_featured.jpg

Figura 9 – Arduino, Acesso em 31 de Maio de 2016, disponível em http://forum.arduino.cc/index.php?topic=132130.0

Figura 10 – FelipeFlop. Acesso em 31 de Maio de 2016, disponível em http://www.filipeflop.com/pd-6b8a2-sensor-de-distancia-ultrassonico-hc-sr04.html

Figura 11 – [12]

Figura 13 – Adafruit. Acesso em 31 de Maio de 2016, disponível em https://www.adafruit.com/product/1438

Figura 15 – Embarcados. Acesso em 31 de Maio de 2016, disponível em http://www.embarcados.com.br/modulos-bluetooth-hc-05-e-hc-06/

Figura 16 – Mercado Livre. Acesso em 31 de Maio de 2016, disponível em http://produto.mercadolivre.com.br/MLB-707472556-motor-dc-3-a-6v-dc-com-caixa-de-reducoeixo-duplo-arduino-_JM

Figura 17 – [8]

Figura 18 – [8]

Figura 19 – The Hunt. Acesso em 31 de Maio de 2016, disponível em https://www.thehunt.com/finds/wfumrB-lmnt-portable-phone-charger-252285100---lanyards---keychains

APÊNDICES

A. Programação do SAM3X

O processador SAM3X, principal componente do Arduino Due, possui uma programação levemente diferenciada dos processadores AVR presentes em outros tipos de Arduino. A principal diferença diz respeito à lógica dos registradores e a forma de nomeá-los. Estes pontos serão detalhados a seguir.

A manipulação de registradores é uma parte fundamental da programação de circuitos embarcados. Os registradores podem controlar diversas funções do microcontrolador (*timers*, RTCs, nível lógico de pinos, comunicação serial, etc.). Cada bit de um registrador possui função própria, como é explicado na *datasheet* do processador. Na família AVR, a maior parte dos registradores pode ser utilizado para escrita e leitura; é possível "setar" (levar o nível lógico a 1) ou "resetar" (levar o nível lógico a 0) um bit qualquer e com isso alterar a ação realizada pelo registrador. Isso também ocorre nos processadores MSP430, da Texas Instruments, e por muitos outros processadores. Entretanto, no SAM3X, são necessários três registradores para realizar esta mesma atividade: um apenas de leitura (normalmente sob o nome de *status register*), um para "setar" bits (geralmente chamado de *enable register*) e outro para "resetar" bits (pode ser chamado de *disable register*). A ação de "resetar" um bit em um *enable register* não produz efeito algum, e vice-versa.

Um programador familiarizado com outros microcontroladores também pode encontrar dificuldade em referenciar estes registradores ao longo do código. Embora a *datasheet* do SAM3X faça um bom trabalho em evidenciar o nome de cada registrador, não deixa claro que, durante o código, o nome deve ser acompanhado do prefixo "REG_". Cada campo formado por um ou mais bits dentro de um registrador também possui um nome, que deve ser acompanhado pelo nome do próprio registrador. Por exemplo, se se deseja desabilitar o contador do "*timer* 5", utilizado no programa, deve-se acessar o registrador TC1_CCR2 (a *datasheet* explica que existem três grupos de *timers*, numerados de 0 a 2. Cada grupo é dividido em três *timers*, também numerados de 0 a 2. Assim, o *timer* 2 do grupo 1 é equivalente ao *timer* 5) e se "setar" o bit CLKDIS. A linha de código para executar esta ação seria algo como:

REG_TC1_CCR2 |= TC_CCR_CLKDIS;

O operador *bitwise OR* (representado pelo caractere '|') é opcional, dependendo da intenção do programador: seu uso deixará os demais bits do registrador inalterados, mas se se deseja "resetar" os demais bits, o operador deve ser retirado.

B. Códigos

B.1 TCC_main.ino

```
#include "setup_due.h"
#include "distancia.h"
#include "motor_nova_shield.h"
#include "bluetooth.h"

int atual = 0;
int critica = 0;

void setup() {
  //  Serial.begin(9600); //para debug
  Serial2.begin(9600);  // Utilizado para a comunicação
com o módulo bluetooth
  timer5_inicializa();
  pinMode(39, INPUT_PULLUP);
  pinMode(37, INPUT_PULLUP);
  pinMode(35, INPUT_PULLUP);
  pinMode(33, INPUT_PULLUP);
  pinMode(31, INPUT_PULLUP);
  pinMode(29, INPUT_PULLUP);
  pinMode(27, INPUT_PULLUP);
  pinMode(25, INPUT_PULLUP);
  attachInterrupt(39, sensor0_handler, FALLING);
  attachInterrupt(37, sensor1_handler, FALLING);
  attachInterrupt(35, sensor2_handler, FALLING);
  attachInterrupt(33, sensor3_handler, FALLING);  //
habilita as interrupções de pino e cria os ISRs
  attachInterrupt(31, sensor4_handler, FALLING);
  attachInterrupt(29, sensor5_handler, FALLING);
  attachInterrupt(27, sensor6_handler, FALLING);
  attachInterrupt(25, sensor7_handler, FALLING);
  sensor_inicializa();
  motor_inicializa();
  //  pinMode(13, OUTPUT);  // para debug
}

void loop() {
  unsigned int i, j;

  if (flag_ciclo) { //Confere se o ciclo de tempo foi
completado
    flag_ciclo = 0; //Limpa a flag de ciclo

    //Realiza a transformacao das variaveis de tempo dos
sensores em microsegundoss
    for (int i = 0; i < NUM_SENSORES; i++) {
      sensor[i].duracao = sensor[i].contagem_int *
TEMPO_INTERRUPT + sensor[i].contagem_reg *
TEMPO_CLOCK - TEMPO_ESPERA_FIM_PULSO;

      //Verifica se o tempo é superior ao tempo
equivalente a distancia de 200cm
      if (sensor[i].duracao > TEMPO_MAXIMO)
sensor[i].distancia[MM_Posicao] = TEMPO_MAXIMO /
58;
      else       sensor[i].distancia[MM_Posicao] =
sensor[i].duracao / 58;

          eliminaCritica(i);  // verifica se a nova leitura é
muito diferente da leitura anterior

      //Calcula a media
      sensor[i].media = 0;

      for (j = 0; j < POSICOES; j++) {
        sensor[i].media += sensor[i].distancia[j];
      }
      sensor[i].media /= POSICOES;

//      Serial.print(sensor[i].contagem_int);
//      Serial.print(sensor[i].contagem_reg);
//      Serial.print(sensor[i].duracao);
//      Serial.print(sensor[i].distancia[MM_Posicao]);
//      Serial.print(sensor[i].media);
//      Serial.print("\t");
    }

    MM_Posicao++; // incrementa posição no vetor de
distâncias
    MM_Posicao = MM_Posicao % POSICOES; // zera a
posição se chegar a 3

    calcula_distancia_angulo();
    //    Serial.print(angulo_fuga,10);
    //    Serial.println("");
    decide_motores();
    //    String debug1=girando? "Girando" : "Não
Girando";
    //    Serial.print(debug1);
    //    Serial.print("\tDistancia: ");
    //    Serial.print(distancia_obstaculo);
    //    Serial.print("\tMovimento:");
    //    Serial.println(Movimento);
    envia_informacao();
  }
}

void eliminaCritica(int i) {
  int anterior = 0;
  if (atual != 0) anterior = atual - 1;
  else anterior = POSICOES - 1;
  if (abs(sensor[i].distancia[atual] - sensor[i].media) > 50
&& critica < 3) {
    critica++;
    if (atual != 0)       sensor[i].distancia[atual] =
sensor[i].distancia[anterior];
    else sensor[i].distancia[atual] =
sensor[i].distancia[POSICOES - 1];
  } else critica = 0;
}

void TC5_Handler(void) {
  long dummy = REG_TC1_SR2; // Ler esse registrador
apaga a flag de interrupção

  if (conta_tempo == TEMPO_CICLO) { //Confere se a
contagem de csegs atingiu o valor do ciclo
    conta_tempo = 0;  //Zera o contador
```

```
   flag_ciclo = 1;  //Ativa a flag avisando q um ciclo foi
completado
   unsigned char i;

//    Dispara todos os sensores
   for (i = 0; i < (NUM_SENSORES); i++) {
     digitalWrite(sensor[i].trigPin, HIGH);  //Eleva o nivel
em todos os pinos de trigger
   }

   delayMicroseconds(10); //Aguarda o tempo para o
sensor garantir que recebeu o pulso

   for (i = 0; i < (NUM_SENSORES); i++) {
     //Retorna para o nivel baixo todos os pinos de trigger
     digitalWrite(sensor[i].trigPin, LOW);
   }
 }
 //Se o ciclo nao estiver sido completado o contador é
incrementado
 else {
   conta_tempo++;
 }

}

/*
 * Cada rotina de interrupção salva o valor do contador e
o valor do registrador de contagem do timer. No
decorrer do programa, estes valores serão convertidos
em microsegundos
 */

void sensor0_handler() {
 sensor[0].contagem_int = conta_tempo;
 sensor[0].contagem_reg = REG_TC1_CV2;
}

void sensor1_handler() {
 sensor[1].contagem_int = conta_tempo;
 sensor[1].contagem_reg = REG_TC1_CV2;
}

void sensor2_handler() {
 sensor[2].contagem_int = conta_tempo;
 sensor[2].contagem_reg = REG_TC1_CV2;
}

void sensor3_handler() {
 sensor[3].contagem_int = conta_tempo;
 sensor[3].contagem_reg = REG_TC1_CV2;
}

void sensor4_handler() {
 sensor[4].contagem_int = conta_tempo;
 sensor[4].contagem_reg = REG_TC1_CV2;
}

void sensor5_handler() {
 sensor[5].contagem_int = conta_tempo;
 sensor[5].contagem_reg = REG_TC1_CV2;
}

void sensor6_handler() {
 sensor[6].contagem_int = conta_tempo;
 sensor[6].contagem_reg = REG_TC1_CV2;
}

void sensor7_handler() {
 sensor[7].contagem_int = conta_tempo;
 sensor[7].contagem_reg = REG_TC1_CV2;
}
```

B.2 setup_due.h

```
//#include <io.h>
#include <Arduino.h>
// DEFINES //

#ifndef _SETUP_H
#define _SETUP_H

#define DELAY 525 //Tempo entre as interrupcoes de tempo t=0.8ms
#define TEMPO_CICLO 62 //Tempo de um ciclo t=50ms
#define TEMPO_MAXIMO 11640 //Tempo maximo de espera do pulso de echo para 2m
#define TEMPO_ESPERA_FIM_PULSO 400 //Tempo entre o fim do pulso de disparo e o incio do pulso de echo
#define TEMPO_INTERRUPT 800 //Tempo entre interrupções em us
#define TEMPO_CLOCK 2 //Tempo entre incrementos no contador do timer 5
#define NUM_SENSORES 8 //Quantidade de sensores
#define POSICOES 3 //Tamanho do vetor de média móvel
#define PINO_INICIAL 38//Pino de Trigger do primeiro sensor
#define DISTANCIA_MAXIMA 100 //Distancia maxima em cm para reação do veículo
#define QUANTIDADE_DADOS 17 //Quantidade de dados a serem enviados pelo Bluetooth


//Declara��o da estrutura de um sensor
typedef struct _Sensor {
  unsigned int trigPin;          //Pino de Trigger
  unsigned int distancia[POSICOES]; //Vetor de Distancias em cm
  unsigned int contagem_int; //Armazena quantas interrupcoes se passaram
  unsigned int contagem_reg; //Armazena valor do contador do timer 5
  unsigned int duracao; //Valor total do pulso em us
  unsigned int media; //Valor da media das distancias
}          Sensor;


////////////////////////////////////
////// VARIABLES DECLARATION //////
////////////////////////////////////
enum moviment {Frente, Esquerda, Direita, Atras_Direita, Atras_Esquerda, Parado} Movimento;


// FLAGS //
volatile unsigned char flag_ciclo = 0;  //Flag que indica q um ciclo de contagem de 50ms foi concluido
// COUNTERS
volatile unsigned int conta_tempo = 0; //Variavel responsavel pela contagem de ciclos de 0.8ms
// ELSE
unsigned int MM_Posicao = 0; //Variavel que controla a posiçãoo no vetor da média móvel
char dados[QUANTIDADE_DADOS];          //Vetor com os dados a serem enviados

//////////////////////////
```

```
////// TIMER 5 SETUP //////
//////////////////////////
// Rotina de inicializacao do timer 5 para gerar interrupcoes a cada 0.8ms
void timer5_inicializa(void) {
  noInterrupts();
  REG_PMC_PCER1 = (1); // habilita o clock periférico do timer
  REG_TC1_CCR2 = TC_CCR_CLKDIS; // desabilita o contador do timer
  REG_TC1_IDR2 = 0xFF; // desabilita as interrupções do timer
  REG_TC1_SR2;  // limpa as flags de interrupção
  REG_TC1_CMR2                    = TC_CMR_TCCLKS_TIMER_CLOCK4                    | TC_CMR_WAVE    | TC_CMR_WAVSEL_UP_RC; //prescaler = 1/128, modo UP (conta até RC2, depois zera o contador)
  REG_TC1_RC2 = DELAY; // interrupções a cada 0,8 ms
  NVIC_EnableIRQ(TC5_IRQn);  // cria ISR para interrupção do timer
  REG_TC1_IER2 = TC_IER_CPCS;  // habilita interrupção por comparação com RC2
  REG_TC1_CCR2 = TC_CCR_CLKEN | TC_CCR_SWTRG; // habilita o contador o timer e força inicialização
  interrupts();
}


//////////////////////////
////// SENSOR SETUP //////
//////////////////////////
Sensor sensor[NUM_SENSORES];
//incialia todos os sensores
void sensor_inicializa() {
  unsigned int i, j;
  for (i = 0; i < NUM_SENSORES; i++) {
    sensor[i].trigPin  = PINO_INICIAL - (2 * i);   //Define o numero do pino de trigger
    pinMode(sensor[i].trigPin, OUTPUT); //Define o pino de trigger como saida

    for (j = 0; j < POSICOES; j++) sensor[i].distancia[j] = 200; //Inicializa o vetor de distancias com o valor maximo
    //Zera os contadores e variaveis
    sensor[i].contagem_int  = 0;
    sensor[i].contagem_reg  = 0;
    sensor[i].duracao                = TEMPO_MAXIMO;
    sensor[i].media                  = 200;
  }
}
#endif
```

B.3 motor_nova_shield.h

```
#ifndef _MOTOR_H
#define _MOTOR_H

#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include "setup_due.h"
#include "distancia.h"

#define velocidade_baixa 60
#define velocidade_alta 90

char VELOCIDADE_1 = 0;
char VELOCIDADE_2 = 0;
char VELOCIDADE_3 = 0;
char VELOCIDADE_4 = 0;

extern char girando = 0;

Adafruit_MotorShield shield = Adafruit_MotorShield();
// Cria um objeto shield com endereço padrão
Adafruit_DCMotor *motores[4] = {shield.getMotor(1),
shield.getMotor(2), shield.getMotor(4),
shield.getMotor(3)}; // Os motores são incluídos nesta
ordem para facilitar o controle simultâneo de todos

void andar_frente(int _velocidade) {
  int i;
  for (i = 0; i < 4; i++) {
    motores[i]->setSpeed(_velocidade);
    motores[i]->run(FORWARD);
    delay(10);
  }
  VELOCIDADE_1 = _velocidade;
  VELOCIDADE_2 = _velocidade;
  VELOCIDADE_3 = _velocidade;
  VELOCIDADE_4 = _velocidade;
  //Serial.println("Andar Frente");
}
void andar_direita(void) {
  int i;
  for (i = 0; i < 4; i++) {
    motores[i]->setSpeed(velocidade_baixa);
    if (i % 2) {  // motores 2 e 3
      motores[i]->run(FORWARD);
    } else {    //motores 1 e 4
      motores[i]->run(BACKWARD);
    }
    delay(10);
  }
  VELOCIDADE_1 = velocidade_baixa;
  VELOCIDADE_2 = velocidade_baixa;
  VELOCIDADE_3 = velocidade_baixa;
  VELOCIDADE_4 = velocidade_baixa;
  //Serial.println("Andar Direita");
}
void andar_esquerda(void) {
  int i;
  for (i = 0; i < 4; i++) {
    motores[i]->setSpeed(velocidade_baixa);
    if (i % 2) {
      motores[i]->run(BACKWARD);
    } else {
      motores[i]->run(FORWARD);
    }
    delay(10);
  }
  VELOCIDADE_1 = velocidade_baixa;
  VELOCIDADE_2 = velocidade_baixa;
  VELOCIDADE_3 = velocidade_baixa;
  VELOCIDADE_4 = velocidade_baixa;
  //Serial.println("Andar Esquerda");
}
void parado(void) {
  int i;
  for (i = 0; i < 4; i++) {
    motores[i]->setSpeed(0);
    delay(10);
  }
  VELOCIDADE_1 = 0;
  VELOCIDADE_2 = 0;
  VELOCIDADE_3 = 0;
  VELOCIDADE_4 = 0;
  //Serial.println("Parado");
}
void virada_brusca_esquerda(void) {
  int i;
  for (i = 0; i < 4; i++) {
    motores[i]->setSpeed(velocidade_alta);
    if (i % 2) {
      motores[i]->run(BACKWARD);
    } else {
      motores[i]->run(FORWARD);
    }
    delay(10);
  }
  VELOCIDADE_1 = velocidade_alta;
  VELOCIDADE_2 = velocidade_alta;
  VELOCIDADE_3 = velocidade_alta;
  VELOCIDADE_4 = velocidade_alta;
  //Serial.println("Virada brusca esquerda");
}
void virada_brusca_direita(void) {
  int i;
  for (i = 0; i < 4; i++) {
    motores[i]->setSpeed(velocidade_alta);
    if (i % 2) {
      motores[i]->run(FORWARD);
    } else {
      motores[i]->run(BACKWARD);
    }
    delay(10);
  }
    VELOCIDADE_1 = velocidade_alta;
    VELOCIDADE_2 = velocidade_alta;
    VELOCIDADE_3 = velocidade_alta;
    VELOCIDADE_4 = velocidade_alta;
  //Serial.println("Virada brusca direita");
}

void motor_inicializa() {
  shield.begin(); // Inicializa comunicação com shield
usando frequência padrão (1.6kHz)
  parado();
}

//Rotina de decisao de como sera o acionamento dos
motores
void decide_motores(void)
{
  char movimento_copia = Movimento;
```

```
  /*Verifica-se a distancia do obstaculo, se menor q 1m entao havera algum acionamento
    o programa verifica o angulo de saida e entao realiza a mudanca de estado.
    O acionamento é feito dentro do Switch case */
  if (!girando) {                         //Andando para a frente ou parado
    if (distancia_obstaculo > DISTANCIA_MAXIMA) {
      Movimento = Parado;
    }
    else if ((angulo_fuga <= M_PI_2) && (angulo_fuga > M_PI / 12)) {
      Movimento = Esquerda;
    }
    else if ((angulo_fuga >= 3 * M_PI_2) && (angulo_fuga < (23 * M_PI / 12))) {
      Movimento = Direita;
    }
    else if ((angulo_fuga > M_PI_2) && (angulo_fuga < M_PI)) {
      Movimento = Atras_Esquerda;
    }
    else if ((angulo_fuga < (3 * M_PI_2)) && (angulo_fuga >= (M_PI))) {  // se o obstáculo estiver à frente da plataforma, ela gira para a direita
      Movimento = Atras_Direita;
    }
    else {
      Movimento = Frente;
    }

    if (Movimento != Frente && Movimento != Parado) girando = 1;        // Nenhum obstáculo se aproximou e ainda não está suficientemente longe do obstáculo às costas do carro
    if (Movimento != movimento_copia) {
      switch (Movimento) {
        case Parado:
          parado();
          dados[13] = 0;
          break;

        case Atras_Direita:
          virada_brusca_direita();
          dados[13] = 2;
          break;

        case Atras_Esquerda:
          virada_brusca_esquerda();
          dados[13] = 3;
          break;

        case Direita:
          andar_direita();
          dados[13] = 2;
          break;

        case Esquerda:
          andar_esquerda();
          dados[13] = 3;
          break;

        case Frente:
          int rapidez_fuga = 100 - 0.3 * distancia_obstaculo; //Formula para definicao da velocidade de fuga. A velocidade é maior quando o objeto está mais próximo
          andar_frente(rapidez_fuga);
          dados[13] = 1;
          break;
      }
    }
  }
  else {  // O carro está girando
    if (angulo_fuga < (M_PI / 12) && angulo_fuga > (-M_PI / 12)) {  // Momento em que o carro dá as costas ao obstáculo mais próximo
      girando = 0;
      int rapidez_fuga = 100 - 0.3 * distancia_obstaculo; //Formula para definicao da velocidade de fuga
      andar_frente(rapidez_fuga);
    }
  }
}

#endif
```

B.4 distancia.h

```
#ifndef _DISTANCIA_H
#define _DISTANCIA_H


//A.4.      distancia_angulo.h
double angulo_fuga = 0;       //Angulo de fuga
int distancia_obstaculo = 1000; //distancia do obst�culo

void calcula_distancia_angulo(void){
   unsigned int Menores_distancias = 1000000; //Vetor de menores distancias medidas. Iniciado com valor alto
   unsigned int posicao_sensor_distancia = 9; //Identificacao dos sensores de menores distancias medidas. Iniciado com sensor inexistente
   for (unsigned char m = 0; m < NUM_SENSORES; m++) {
     //Encontra o menor valor medido
     if (sensor[m].media < Menores_distancias){
       Menores_distancias = sensor[m].media;          // Substitui pela menor média
       posicao_sensor_distancia = m;
     }
   }
   distancia_obstaculo = Menores_distancias+10;

   angulo_fuga = (posicao_sensor_distancia) * M_PI_4-M_PI; //calcula o ângulo diametralmente oposto ao sensor com menor média
   if (angulo_fuga<0) angulo_fuga+=(2*M_PI); // garante que o ângulo de fuga esteja entre 0 e 2pi
}

#endif
```

B.5 bluetooth.h

```
#ifndef _BLUETOOTH_H
#define _BLUETOOTH_H

#include "distancia.h"
#include "setup_due.h"
#include "motor_nova_shield.h"

//Rotina que arruma o vetor que sera transmitido com os valores do ultimo ciclo
void envia_informacao (void) {
  dados[0] = 255;    //Flag de inicialização

  for (int i = 0; i < NUM_SENSORES; i++) {
    dados[i + 1] = sensor[i].media; //Media de cada sensor
  }
  dados[9]  = VELOCIDADE_1; //Velocidade 1
  dados[10] = VELOCIDADE_2; //Velocidade 2
  dados[11] = VELOCIDADE_3; //Velocidade 3
  dados[12] = VELOCIDADE_4; //Velocidade 4
  //dados[13] = direcao; esta informação está sendo atualizada dentro das funções
  dados[14] = distancia_obstaculo; //valor da distancia calculada

  float fdata = angulo_fuga;
  char ftemp = (fdata - (int)fdata) * 10;
  dados[15] = (int)fdata+1; //parte inteira do float angulo_fuga. O caracter '0' não é corretamente entendido pelo Matlab, por isso soma-se 1. Essa transformação é revertida no Matlab
  dados[16] = ftemp+1; //parte fracionaria do float angulo_fuga

  Serial2.print(dados); // envia os dados pela porta serial 2, nos pinos 14 e 15 do Arduino DUE
}

#endif
```

B.6 leitura_dados.m

```
%%LEITURA DE DADOS DO ARDUINO VIA
BLUETOOTH%%
clear all
clc
close all
%%Parâmetros para os gráficos
nciclos = 100;
maxdist = 200;
maxvelm = 110;
mmovel = 4;
%%Vetores de dados
x = (0:1:nciclos);
s1 = zeros(1,nciclos+1);
s2 = zeros(1,nciclos+1);
s3 = zeros(1,nciclos+1);
s4 = zeros(1,nciclos+1);
s5 = zeros(1,nciclos+1);
s6 = zeros(1,nciclos+1);
s7 = zeros(1,nciclos+1);
s8 = zeros(1,nciclos+1);
m1 = zeros(1,nciclos+1);
m2 = zeros(1,nciclos+1);
m3 = zeros(1,nciclos+1);
m4 = zeros(1,nciclos+1);
d = 0;
a = 0;
u1 = 0;
v1 = 0;
u2 = 0;
v2 = 0;
%Matriz de armazenamento
temporária
t = zeros(16,mmovel);
ind = 1;

%%Figura 1: Distâncias lidas por
cada sensor%%
f1 = figure(1);
suptitle('Distância lida por cada
sensor');

subplot(3,3,1);
gs1 = plot(x,s2,'r');
title('NO(2)');
axis([0 nciclos 0 maxdist]);
xlabel('Ciclos');
ylabel('Distância (cm)');

subplot(3,3,2);
gs2 = plot(x,s1,'y');
title('N(1)');
axis([0 nciclos 0 maxdist]);
xlabel('Ciclos');
ylabel('Distância (cm)');

subplot(3,3,3);
gs3 = plot(x,s8,'c');
title('NE(8)');
axis([0 nciclos 0 maxdist]);
xlabel('Ciclos');
ylabel('Distância (cm)');

subplot(3,3,4);
gs4 = plot(x,s3,'g');
title('O(3)');
axis([0 nciclos 0 maxdist]);
xlabel('Ciclos');
ylabel('Distância (cm)');
subplot(3,3,6);
gs5 = plot(x,s7,'b');
title('L(7)');
axis([0 nciclos 0 maxdist]);
xlabel('Ciclos');
ylabel('Distância (cm)');

subplot(3,3,7);
gs6 = plot(x,s4,'k');
title('SO(4)');
axis([0 nciclos 0 maxdist]);
xlabel('Ciclos');
ylabel('Distância (cm)');

subplot(3,3,8);
gs7 = plot(x,s5,'m');
title('S(5)');
axis([0 nciclos 0 maxdist]);
xlabel('Ciclos');
ylabel('Distância (cm)');

subplot(3,3,9);
gs8 = plot(x,s6,'r');
title('SE(6)');
axis([0 nciclos 0 maxdist]);
xlabel('Ciclos');
ylabel('Distância (cm)');

subplot(3,3,5);
ga1 = plot(x,s2,'r');
hold on;
ga2 = plot(x,s1,'y');
hold on;
ga3 = plot(x,s8,'c');
hold on;
ga4 = plot(x,s3,'g');
hold on;
ga5 = plot(x,s7,'b');
hold on;
ga6 = plot(x,s4,'k');
hold on;
ga7 = plot(x,s5,'m');
hold on;
ga8 = plot(x,s6,'r');
title('TODOS');
axis([0 nciclos 0 maxdist]);
xlabel('Ciclos');
ylabel('Distância (cm)');
hold off;
%%Figura 2: velocidade dos motores
```

```
f2= figure(2);
suptitle('Velocidade dos
motores');

subplot(2,2,1);
gp1 = plot(x,m1);
title('Frontal esquerdo');
axis([0 nciclos 0 maxvelm]);
xlabel('Ciclos');
ylabel('Velocidade (PWM)');

subplot(2,2,2);
gp2 = plot(x,m2);
title('Frontal direito');
axis([0 nciclos 0 maxvelm]);
xlabel('Ciclos');
ylabel('Velocidade (PWM)');

subplot(2,2,3);
gp3 = plot(x,m4);
title('Traseiro esquerdo');
axis([0 nciclos 0 maxvelm]);
xlabel('Ciclos');
ylabel('Velocidade (PWM)');

subplot(2,2,4);
gp4 = plot(x,m3);
title('Traseiro direito');
axis([0 nciclos 0 maxvelm]);
xlabel('Ciclos');
ylabel('Velocidade (PWM)');
%%Figura 3: representação da
direção do carro em tempo real
f3 = figure(3);
q1 = quiver(0,0,u1,v1);
axis([-1 1 -0.1 1]);
set(gca,'xtick',[]);
set(gca,'ytick',[]);
title('Direção');
%%Figura 4: representação vetorial
do obstáculo
f4 = figure(4);
title('Distância e ângulo ao
obstáculo mais próximo');
q2 = quiver(0,0,u2,v2);
title('Posição e distância do
objeto em relação ao carro');
axis([-maxdist maxdist -maxdist
maxdist]);
xlabel('Distância (cm)');
ylabel('Distância (cm)');
%%Conexão com o Bluetooth
%instrhwinfo('BLuetooth')
h = Bluetooth('HC-05',1);
fopen(h);
h
disp('Programa em andamento');

while (1)
k = get(f1,'CurrentCharacter');
if k=='s'
fclose(h);
h
disp('Programa encerrado');
break;
end
%Leitura dos dados
if fread(h,1) == 255
for m=1:mmovel
for n=1:16
if n==15
t(n,m)=((fread(h,1)-1)+
((fread(h,1)-1)/10));
elseif (n==16 && m==mmovel)
t(n,m) = 0;
else
t(n,m) = fread(h,1);
end
end

%Cálculo de média móvel
s1(ind) = (sum(t(1,:)))/mmovel;
s2(ind) = (sum(t(2,:)))/mmovel;
s3(ind) = (sum(t(3,:)))/mmovel;
s4(ind) = (sum(t(4,:)))/mmovel;
s5(ind) = (sum(t(5,:)))/mmovel;
s6(ind) = (sum(t(6,:)))/mmovel;
s7(ind) = (sum(t(7,:)))/mmovel;
s8(ind) = (sum(t(8,:)))/mmovel;
m1(ind) = (sum(t(9,:)))/mmovel;
m2(ind) = (sum(t(10,:)))/mmovel;
m3(ind) = (sum(t(11,:)))/mmovel;
m4(ind) = (sum(t(12,:)))/mmovel;

switch t(13,m)
case 1
%Frente
u1 = 0;
v1 = 1;
case 2
%Direita
u1 = 1;
v1 = 0;
case 3
%Esquerda
u1 = -1;
v1 = 0;
case 0
%Parado
u1 = 0;
v1 = 0;
end

d = (sum(t(14,:)))/mmovel;
a = (sum(t(15,:)))/mmovel;
[u2,v2] = pol2cart((a-pi/2),d);
ind = ind + 1;
if ind==nciclos+2
ind=1;
end
```

```
%%Referência de atualização
s1(ind) = maxdist;
s2(ind) = maxdist;
s3(ind) = maxdist;
s4(ind) = maxdist;
s5(ind) = maxdist;
s6(ind) = maxdist;
s7(ind) = maxdist;
s8(ind) = maxdist;
m1(ind) = maxvelm;
m2(ind) = maxvelm;
m3(ind) = maxvelm;
m4(ind) = maxvelm;
%%Atualização dos dados nos
gráficos
set(gs1,'YData',s2);
set(gs2,'YData',s1);
set(gs3,'YData',s8);
set(gs4,'YData',s3);
set(gs5,'YData',s7);
set(gs6,'YData',s4);
set(gs7,'YData',s5);
set(gs8,'YData',s6);
set(ga1,'YData',s2);
set(ga2,'YData',s1);
set(ga3,'YData',s8);
set(ga4,'YData',s3);
set(ga5,'YData',s7);
set(ga6,'YData',s4);
set(ga7,'YData',s5);
set(ga8,'YData',s6);
set(gp1,'YData',m1);
set(gp2,'YData',m2);
set(gp3,'YData',m4);
set(gp4,'YData',m3);
set(q1,'udata',u1,'vdata',v1);
set(q2,'udata',u2,'vdata',v2);
drawnow;
end
end
end
```