

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA – PATOS DE MINAS
ENGENHARIA ELETRÔNICA E DE TELECOMUNICAÇÕES

CÉSAR HELLY SOARES SANTOS JÚNIO

**SISTEMA DE MONITORAMENTO E MAPEAMENTO POR
ROBÔ MÓVEL**

Patos de Minas - MG
2019

CÉSAR HELLY SOARES SANTOS JÚNIO

SISTEMA DE MONITORAMENTO E MAPEAMENTO POR ROBÔ MÓVEL

Trabalho de conclusão de curso apresentado à banca examinadora como requisito de avaliação da disciplina de TCC2 da graduação em Engenharia Eletrônica e de Telecomunicações, da Faculdade de Engenharia Elétrica, da Universidade Federal de Uberlândia, *Campus* Patos de Minas.
Orientador: Prof. Dr. Daniel Costa Ramos

CÉSAR HELLY SOARES SANTOS JÚNIO

SISTEMA DE MONITORAMENTO E MAPEAMENTO POR ROBÔ MÓVEL

Trabalho de conclusão de curso apresentado à banca examinadora como requisito de avaliação da disciplina de TCC2 da graduação em Engenharia Eletrônica e de Telecomunicações, da Faculdade de Engenharia Elétrica, da Universidade Federal de Uberlândia, *Campus* Patos de Minas.

Orientador: Prof. Dr. Daniel Costa Ramos

Patos de Minas, 12 de agosto de 2019

Banca Examinadora

Prof. Dr. Daniel Costa Ramos – FEELT/UFU (Orientador)

Prof. Dr. Júlio César Coelho – FEELT/UFU (Membro 1)

Prof. Dr. Davi Sabbag Roveri – FEELT/UFU (Membro 2)

RESUMO

A robótica móvel é um campo da robótica útil na sociedade atualmente, que surgiu a partir do crescimento da robótica, uma ciência relativamente recente. Útil por ser necessária em diversas áreas, como em indústrias de mineração, aeroespacial, agricultura, entre outras. É um campo que está em desenvolvimento, onde uma das aplicações comuns é de navegar em ambientes desconhecidos e com isso surge a necessidade de mapeamento territorial, uma vez é necessário conhecer um local quando se trata de um ambiente de risco, como mapeamento para segurança em minas, ou quando se deseja adquirir informações de lugares de difícil acesso humano, como explorações interplanetárias ou marinhas, ou para trabalhos de inspeção. Este trabalho aborda o mapeamento feito por robô móvel de forma autônoma-manual, e o seu monitoramento, podendo ser assistido por um usuário, pois mesmo em mapeamentos exclusivamente autônomo, erros podem ser cometidos, necessitando uma intervenção humana para corrigir esses erros ou problemas. O mapeamento é feito de forma simultânea com o monitoramento. Para isso, são usados sensores de velocidade e sensor ultrassônico, mapeamento métrico por grade, placa de processamento baseado no Arduino e tecnologia de comunicação Bluetooth. Os resultados demonstram a capacidade do robô em mapear um ambiente desconhecido de forma satisfatória, levando em conta a utilização de materiais de baixo custo.

Palavras-chave: Mapeamento. Monitoramento. Robótica.

ABSTRACT

Mobile robotics is a field of robotic useful to the society today, which came from the growth of robotics, a relatively recent science. Useful because it is necessary in various areas as the mining, aerospace, agriculture and other industries. It is a field that is in development, where navigation in unknown environment is a common task. This urges the necessity to have a map of the territory, once it is necessary to know place when it is a risk environment, such as the mapping for security in mines, or when information from specific places that is difficult for humans to access is desired, such as interplanetary or marine explorations, or for inspection work. This work deals with the mapping process made by a mobile robot in an autonomous-manual way, and its monitoring, being able to be assisted by a user, since in the autonomous exclusive mappings errors may occur, requiring a human action to correct these errors or problems. In this work, the mapping is done simultaneously with the monitoring. For this end, speed and ultrasonic sensor are utilized along with a grid mapping mode, an Arduino-based processing board, and a Bluetooth communication technology. Results shown the capacity of the robot to satisfactory map an unknown environment, taking in account the limitations of a low-budget platform.

Keywords: Mapping. Monitoring. Robotic.

LISTA DE FIGURAS

Figura 2.1 - Mapeamento topológico em etapas graduais.....	25
Figura 2.2 - Mapa topológico.	26
Figura 2.3 - Sonar em mapa de grade de ocupação.	27
Figura 2.4 - Resultado de mapeamento por mapa de características.	28
Figura 2.5 - Mapa métrico.	30
Figura 2.6 - Mapa topológico.	30
Figura 2.7 - <i>Encoder</i> óptico.....	31
Figura 2.8 - Exemplo de atuação de um sensor ultrassom.	32
Figura 2.9 - Alcance do sensor ultrassom.	32
Figura 2.10 - Fluxograma do SLAM.	33
Figura 2.11 - Interface gráfica para monitoramento e mapeamento.....	34
Figura 2.12 - Modelos de placas Arduino	35
Figura 4.1 - Fluxograma de funcionamento do robô.	40
Figura 4.2 – Peças da estrutura de acrílico utilizada no projeto.	41
Figura 4.3 – Vista superior (esquerda) e inferior (direita) da estrutura de acrílico montada. ..	41
Figura 4.4 - Diagrama de pinos da placa MBZ Pro Mega Wi-Fi Edition.	43
Figura 4.5 - Diagramas de pinos do Arduino UNO.....	43
Figura 4.6 - Placa FTDI FT232RL Conversor USB Serial	44
Figura 4.7 - IDE do arduino e processo de seleção das configurações.....	45
Figura 4.8 - Sensor HC-SR04 real e o diagrama de pinagem utilizada.	46
Figura 4.9 - Motor <i>Shield</i> L293D <i>Driver</i> Ponte H.....	50
Figura 4.10 - Pinagem do Motor <i>Shield</i> L293D.	50
Figura 4.11 - Solda das conexões no motor.	51
Figura 4.12 - Ligação dos motores no <i>shield</i> da ponte H.....	51
Figura 4.13 - Posicionamento das rodas e vista superior do robô.	52
Figura 4.14 - Sentido de rotação dos motores.	53
Figura 4.15 – Projecção das rodas no robô.	53
Figura 4.16 - Locomoção do robô devido a ação das rodas.....	54
Figura 4.17 - Deslocamento do robô para trás.....	56
Figura 4.18 - Ação dos motores para giro do robô para direita.....	56
Figura 4.19 - Ação dos motores para giro do robô para esquerda.....	57

Figura 4.20 – Módulo <i>Bluetooth</i> HC06.....	60
Figura 4.21 - Diagrama das ligações do módulo <i>Bluetooth</i>	61
Figura 4.22 - Vetor deslocamento do robô.....	63
Figura 4.23 - Projeção de u e sua relação com o ângulo.....	64
Figura 4.24 - Projeções em X e Y de um vetor a	64
Figura 4.25 - Exemplo de movimentação do robô e de geração de vetores.....	66
Figura 4.26 - Segundo exemplo de movimento do robô e geração de vetores.....	67
Figura 4.27 - Terceiro exemplo de movimentação e geração dos vetores.	68
Figura 4.28 – Quarto exemplo de movimentação e geração de vetores.	68
Figura 4.29 - Encoder HC-020K.	69
Figura 4.30 - Diagrama da ligação dos pinos para o <i>encoder</i>	69
Figura 4.31 - Demonstração do aspecto do robô, mira, extremidade, range e seus estados no mapa.	82
Figura 4.32 - Demonstração do efeito descrito no Quadro 4.48.....	83
Figura 5.1 - Aspecto final do protótipo, com posicionamento do sensor de ultrassom na parte frontal do robô.	84
Figura 5.2 - Visão geral dos componentes do robô e suas conexões.....	84
Figura 5.3 - Tela inicial do mapeamento do robô em seu estado inicial, considerando que não foi detectado nenhum obstáculo e que o robô está parado.....	85
Figura 5.4 - Indicação dos elementos do mapa, incluindo o robô, a mira e o ambiente.....	86
Figura 5.5 - Leitura do ambiente real sem obstáculos com o robô parado e sua representação no mapa.	86
Figura 5.6 - Exibição de obstáculo com o robô parado.	87
Figura 5.7 - Teste de giro do robô e suas respectivas leituras com o ambiente livre de obstáculos.	88
Figura 5.8 - Teste de giro do robô e suas respectivas leituras com o ambiente com a presença de obstáculos.....	88
Figura 5.9 - Representação do ambiente com a presença de rastros indesejáveis do movimento do robô.	89
Figura 5.10 - Remoção dos traços indesejados, sem afetar a representação do robô.	90
Figura 5.11 - Primeiro exemplo da representação da posição inicial no mapa e seus efeitos..	91
Figura 5.12 - Segundo exemplo da representação da posição inicial no mapa e seus efeitos..	91
Figura 5.13 - Teste em ambiente real com a verificação do mapeamento e de sua exibição na interface.	91

LISTA DE TABELAS E QUADROS

Tabela 3.1 - Lista de componentes do projeto e respectivos valores.....	39
Quadro 4.1- Comparação entre a placa MBZ e UNO.....	44
Quadro 4.2 - Código de teste da serial.	46
Quadro 4.3 - Código de medição utilizando o sensor de ultrassom.	48
Quadro 4.4 - Código da função trigPulse().	48
Quadro 4.5 - Código exemplo da função pulseIn (echo, High).....	49
Quadro 4.6 - Código do valor limite de distância.....	49
Quadro 4.7 - Código referente ao motores.....	52
Quadro 4.8 - Código de teste dos motores.	53
Quadro 4.9 - Código de movimentos do robô.	54
Quadro 4.10 - Código para movimentação do robô.....	55
Quadro 4.11 - Balanceamento da velocidade do robô.	55
Quadro 4.12 - Código com o movimento balanceado para trás.	55
Quadro 4.13 - Código para rotação do robô para direita.....	56
Quadro 4.14 - Código para rotação do robô para esquerda.....	57
Quadro 4.15 - Código para o robô parar.	57
Quadro 4.16 - Código para importar código serial no <i>Processing</i>	58
Quadro 4.17 - Parâmetros iniciais de teste da serial.	58
Quadro 4.18 - Leitura de <i>string</i> pela serial.....	59
Quadro 4.19 - Código para os cliques do mouse no <i>Processing</i>	59
Quadro 4.20 - Inicialização da comunicação serial com o <i>Processing</i>	59
Quadro 4.21 – Teste de funcionamento da serial com acender e apagar do LED.....	60
Quadro 4.22 - Código para as condições de estado das teclas.	62
Quadro 4.23 - Código para cada movimento do robô.....	63
Quadro 4.24 - Código de configuração dos pinos do <i>encoder</i>	70
Quadro 4.25 - Código do limitador de tempo.....	70
Quadro 4.26 - Código com marcadores.	72
Quadro 4.27 - Código do acréscimo no contador.	73
Quadro 4.28 - Código do decréscimo no contador	73
Quadro 4.29 - Código dos limites do ambiente.	73

Quadro 4.30 - Estabelecendo os limites do ambiente	74
Quadro 4.31 - Contagem para direita e para esquerda, respectivamente.	74
Quadro 4.32 - Conversão de pulsos em ângulo.	75
Quadro 4.33 - Reduzindo o processamento pelo “zeramento” da variável.	75
Quadro 4.34 - Envio dos dados do robô pela serial.	76
Quadro 4.35 - Envio do comando manual ao robô.	76
Quadro 4.36 - Função de controle automático.	76
Quadro 4.37 - Atribuição do estado "5", referente ao controle automático.	77
Quadro 4.38 - Ações atribuídas ao estado automático.	77
Quadro 4.39 - Ações no contador no estado automático.	77
Quadro 4.40 - Ajuste de 16 pulsos para que o robô gire 90°.	78
Quadro 4.41 - Parâmetros para criação do mapa.	78
Quadro 4.42 - Comando para receber e separar <i>strings</i>	79
Quadro 4.43 - Utilização dos valores x e y do ultrassom.	80
Quadro 4.44 - Distância do robô até o obstáculo via ultrassom.	80
Quadro 4.45 - Código da criação gráfica do robô.	81
Quadro 4.46 - Preenchimento das cores no mapa.	81
Quadro 4.47 - Preenchimento dos obstáculos no mapa.	82
Quadro 4.48 - Remoção das cores indesejadas do mapa.	82

LISTA DE ABREVIATURAS E SIGLAS

DC	Direct Current
FTDI	Future Technology Devices International
IDE	Integrated Development Environment
IoT	Internet of Things
LED	Light-Emitting Diode
PC	Personal Computer
PWM	Pulse Width Modulation
RFID	Radio-Frequency Identification
RTC	Real Time Clock
SLAM	Simultaneous Localization And Mapping
USB	Universal Serial Bus
Wi-Fi	Wireless Fidelity

SUMÁRIO

1	INTRODUÇÃO.....	17
1.1	TEMA	18
1.2	PROBLEMATIZAÇÃO	18
1.3	HIPÓTESES.....	19
1.4	OBJETIVOS.....	20
1.4.1	Objetivos gerais.....	20
1.4.2	Objetivos específicos	20
1.5	JUSTIFICATIVA	21
1.6	CONSIDERAÇÕES FINAIS	22
2	DESENVOLVIMENTO TEÓRICO	23
2.1	MAPEAMENTO	23
2.1.1	Mapeamento topológico.....	23
2.1.2	Mapeamento métrico	26
2.1.3	Comparação entre mapeamento topológico e mapeamento métrico	28
2.2	MAPEAMENTO UTILIZANDO SENSORES	30
2.2.1	Sensor de velocidade (<i>encoder</i> óptico).....	31
2.2.2	Sensor de ultrassom	31
2.3	MONITORAMENTO.....	33
2.4	PROCESSAMENTO	34
2.5	COMUNICAÇÃO	36
2.6	CONSIDERAÇÕES FINAIS	37
3	MATERIAIS E MÉTODOS.....	38
3.1	MATERIAIS E CUSTOS	39
3.2	CONSIDERAÇÕES FINAIS	39
4	DESENVOLVIMENTO	40
4.1	CONSTRUÇÃO MECÂNICA.....	41
4.2	COMPONENTES ELETRÔNICOS.....	42
4.2.1	Placa MBZ Pro Mega	42
4.2.2	Sensor ultrassônico	46
4.2.3	Motores	50
4.2.4	Interação entre motores e a estrutura.....	52
4.2.5	Locomoção do robô.....	54
4.2.6	Interface gráfica no <i>Processing</i>	57
4.2.7	Módulo <i>Bluetooth</i>	60
4.3	PROGRAMAÇÃO DO MICROCONTROLADOR	61

4.3.1	Implementação do controle manual	62
4.3.2	Lógica de deslocamento no mapa	63
4.3.3	<i>Encoder</i> e implementação da interrupção.....	69
4.3.4	Implementação da lógica de deslocamento no mapa	71
4.3.5	Implementação do controle automático	76
4.3.6	Implementação da criação de mapas	78
4.4	CONSIDERAÇÕES FINAIS	83
5	RESULTADOS	84
5.1	DETECÇÃO ESTÁTICA DE OBSTÁCULOS	87
5.2	MOVIMENTAÇÃO E DETECÇÃO DE OBSTÁCULOS	89
5.3	EXECUÇÃO EM AMBIENTE REAL DE FORMA REMOTA.....	90
5.4	LIMITAÇÕES ENCONTRADAS	92
6	CONCLUSÃO, CONTRIBUIÇÕES E TRABALHOS FUTUROS	95
	REFERÊNCIAS.....	97
	APÊNDICE I.....	101
	APÊNDICE II	105

1 INTRODUÇÃO

A robótica móvel é um campo da robótica que está em constante crescimento, consequência da evolução tecnológica e de sua necessidade na parte das indústrias de mineração, aeroespacial, agricultura, bélicas, para pesquisas e estudos. Ela permite que o usuário realize de forma segura as tarefas de alto risco como a inspeção de ambientes com exposição à gases tóxicos, de temperaturas extremas ou com ausência de oxigênio, auxilia em trabalhos que exigem grande esforço físico, de velocidade ou de repetição, reduzindo a chance de acidentes e de erros durante os processos de manufatura.

Os tópicos de localização e mapeamento são campos de pesquisas clássicas na robótica móvel, que quando tratados simultaneamente são denominados de *SLAM* (*Simultaneous Localization and Mapping*) (SIEGWART, 2011). Neste caso, o robô deve se localizar em um ambiente desconhecido e realizar o seu mapeamento, a partir das informações obtidas por sensores. Saber para onde o robô está se locomovendo é importante quando se trata de robótica móvel, pois em um sistema autônomo é necessário ter conhecimento do trajeto do robô e seu destino, possibilitando em uma tarefa específica, como por exemplo, a tarefa de reconhecimento territorial, saber se seguiu a orientação correta.

Os algoritmos de SLAM são limitados aos recursos disponíveis no robô e no ambiente e, portanto, não almejam a perfeição e sim um funcionamento operacional (RAMOS, 2012). As principais aplicações atuais incluem carros autônomos (ROOSE, 2018), veículos aéreos não tripulados (BURKE e GREINER, 2018) e veículos de exploração interplanetária (NASA, 2018).

Uma aplicação particular do SLAM é a questão de mapeamento em tempo real do ambiente, assistido por um usuário. Neste caso, a navegação do robô pode ser inicialmente autônoma e quando é detectado uma anomalia no ambiente, o operador pode passar a controlar o robô de forma manual, iniciando um processo mais delicado e detalhado de inspeção do ambiente.

Esta forma de operação de robôs móveis é geralmente utilizada para longas e cansativas inspeções, sejam estas de dutos de ventilação, de exploração submarina ou mapeamento de ambientes. Nestas inspeções, a exploração é feita de forma autônoma exaustivamente pelo robô, o fator humano apenas observa e intervém nos casos onde são detectadas anomalias que requerem uma inspeção mais cautelosa.

1.1 TEMA

Este trabalho tem como tema a utilização e exemplificação de técnicas de mapeamento de ambientes e de localização de robôs móveis, com foco na inspeção. É proposto a utilização inicial de um protótipo robótico capaz de realizar um simples mapeamento do ambiente, onde as informações do ambiente são adquiridas pelo robô móvel e uma representação do ambiente (mapa) é reconstruído em um computador para o usuário.

1.2 PROBLEMATIZAÇÃO

O campo da robótica está em grande desenvolvimento (ARAÚJO, 2013) e suas tecnologias estão cada vez mais necessárias nas indústrias, medicina, e outras áreas de atuação. Em especial, o tópico de mapeamento e localização com um robô móvel é um problema clássico da robótica móvel e pode ser amplamente utilizado para pesquisas em sensoriamento, prototipagem de robôs e de sistemas de comunicação.

Este campo de pesquisa pode ser aplicado nas mais variadas atividades, como exploração de ambientes desconhecidos, ambientes de desastres naturais ou que ofereçam riscos a presença humana.

Um robô com deslocamento por rodas é capaz de executar tal movimento, não necessitando de deslocamento humano e nem de outros equipamentos para mapear (MONTEMERLO *et al*, 2002), podendo o robô agir de forma autônoma ou de forma teleoperada. Neste contexto da exploração e mapeamento assistido, é necessário que o sistema de monitoramento lide com alguns desafios de forma satisfatória.

Primeiramente, é necessário que haja uma boa estrutura de comunicação entre o robô e o terminal do usuário. Essa comunicação deverá levar em conta a quantidade de dados que serão trocados, a disponibilidade da rede no local da aplicação e uma forma de lidar com um possível atraso ou perda de informações. Para uma transmissão de dados o robô deve usar um sistema de comunicação sem fio, pois como o robô se deslocará pelo ambiente, não é adequado e nem viável fixar um cabeamento de comunicação em sua base.

Para captar informações do meio físico, como a distância de obstáculos ou o deslocamento do robô, sensores devem ser utilizados, pois essas informações devem ser usadas para cálculo do deslocamento, indicação de presença de obstáculo ou limitação da área.

Além da comunicação robô-usuário, é necessário desenvolver uma ferramenta para integrar as leituras realizadas pelo robô, associando a sua atual posição com as leituras relativas de distância dos sensores, elaborando assim um mapa do ambiente e exibindo este mapa ao usuário.

Quanto ao robô, é necessário planejar a forma com que o mesmo irá processar as informações. Um microcontrolador fazendo ligação com os circuitos eletrônicos do robô pode ser utilizado, sendo que o microcontrolador será o “cérebro” do robô, pois toda informação e dados são processados por ele. Este, combinado com sensores e transmissores sem fio, convertem as informações analógicas obtidas em dados digitais e os enviam. O microcontrolador também organiza e executa as ordens programadas ou recebidas do terminal do usuário.

Na questão de monitoramento, ao usar um PC (*PC– Personal Computer*) como terminal, proporciona acessibilidade e redução de custo, não necessitando projetar um terminal exclusivo para tal tarefa. O mesmo proporciona facilidade de visualização do usuário, por utilizar interface própria para visualização, permite uma boa versatilidade de alcance se for utilizar a internet para controle do protótipo, por exemplo, e permite versatilidade na programação utilizada, pois os softwares utilizados para programação e execução dos sistemas no projeto, podem ser usados em outros PC's.

O monitoramento do mapeamento em uma interface em um PC, além de permitir o usuário visualizar o mapeamento e localização do robô, permite economizar memória, pois os dados coletados são transmitidos enquanto são coletados, ao invés de armazenados no robô. Se os dados não fossem transmitidos, o robô precisaria de uma memória relativamente grande para o armazenamento dos dados e sua área de cobertura seria proporcional a capacidade de armazenamento, limitando a capacidade de mapeamento ou elevando o custo do protótipo.

Desta forma, considerando todos os pontos mencionados acima, é proposto a construção de um protótipo robótico capaz de se comunicar por uma rede de comunicação sem fio e enviar dados de mapeamento até uma interface com interação com o usuário, onde o mesmo poderá acompanhar a tarefa do robô em tempo real.

1.3 HIPÓTESES

É considerada a hipótese de que é possível desenvolver um robô de baixo custo para obter informações do ambiente, podendo este ser utilizado para pesquisas na área de robótica e de redes de comunicação.

1.4 OBJETIVOS

1.4.1 Objetivos gerais

Obter um protótipo robótico funcional capaz de realizar um mapeamento do ambiente e transmitir as informações em tempo real para um computador, onde os dados serão tratados e o mapa resultante visualizado pelo usuário.

1.4.2 Objetivos específicos

Para construir um protótipo robótico que realiza a função de mapeamento, é necessário usar um sistema de comunicação entre usuário e protótipo, projetar o circuito eletrônico e mecânico do protótipo, elaborar os sistemas de monitoramento para o usuário, de mapeamento e de comando para as ações do robô. Desta forma, é necessário concluir uma série de objetivos específicos, estes descritos a seguir.

Objetivo específico 1: Pesquisar e analisar tecnologias para protótipo.

Antes de projetar ou criar um protótipo robótico para suprir o objetivo deste trabalho, é necessário pesquisar e levantar opções sobre o tema tratado. Pesquisar opções de mapeamento robótico territorial. Pesquisar estilos de protótipo robótico que economizam espaço e que sejam viáveis em relação a custo. Pesquisar tipos de comunicação sem fio entre protótipo e usuário.

Objetivo específico 2: Projetar e montar o robô e a sua comunicação sem fio

O sistema de comunicação entre o usuário e o robô será com transmissão e recepção por rede sem fio. O protótipo do robô móvel terá sua mobilidade por meio de rodas, microcontrolador para gerenciar suas ações e terá sensores para obter dados físicos do ambiente.

Objetivo específico 3: Projetar e montar o sistema de monitoramento.

O sistema de monitoramento deve ser criado por *software* para a visualização da mobilidade do protótipo em um computador pelo usuário. Uma interface gráfica deve ser

criada para esta visualização, nesta interface deve exibir, por meio de preenchimento de cores, uma representação da área real percorrida pelo protótipo.

1.5 JUSTIFICATIVA

Segundo Thrun (2002), o mapeamento na robótica aborda a obtenção de modelos de representação física do espaço de um ambiente. Thrun também afirma que nas construções de robôs móveis que trabalham de forma autônoma um dos principais problemas é o mapeamento. A utilização de sistemas de mapeamento e localização é relevante em outras áreas dentro da robótica. O sistema de mapeamento e localização permite orientação de robôs móveis em trabalhos feitos de forma autônoma.

Em julho de 2002, em Somerset (Pensilvânia, EUA), na mina de Quecreek, nove mineiros quase morreram afogados com a inundação da mina após um erro de perfuramento (PAULEY, 2002). Este acidente exemplifica a necessidade de um sistema de mapeamento em minas abandonadas, por possuir condições de risco e difíceis rotas de acesso. Mapeamento utilizando a robótica seria uma opção segura (THRUN *et al*, 2003). Ao mapear as áreas de riscos de desabamento, inundações, intoxicação, as informações adquiridas previnem o trabalhador de possíveis acidentes. O mapeamento feito por um robô permite acesso a locais difíceis e substitui o ser humano prevenindo caso aconteça algum acidente.

As obras de Thrun serviram como conceito, exemplificação ou modelos para os trabalhos realizados de Yukinobu (2010), Ramos (2012) e Abner (2012), assim como para este trabalho, por mostrar utilidade em mapeamento usando robô móvel ou propor melhorias em técnicas de mapeamento.

No trabalho de Yukinobu (2010), exemplifica ambientes externos utilizando robôs móveis assim como o trabalho de Abner (2012), ambos os trabalhos foram executados utilizando sensores de captação do ambiente externo. Os autores fizeram mapeamento externo, criando mapas 3D ou 2,5D, mas enquanto Yukinobu utilizou sensor laser para 3D, Abner utilizou uma câmera, esta foi usada para captura de cores considerando grades de elevação. Ambos conseguiram resultados satisfatórios, porém o mapa gerado por Abner era um mapa 2,5D com detalhes de elevação do solo, proporcionado por uma variação de cores no mapa representando a elevação.

O trabalho de Abner (2008) mostra resultado de mapeamento territorial 2D, utilizando ultrassom e odometria, uma abordagem barata que permitiu bons resultados de mapeamento. Trabalhos feitos com simulação utilizando algoritmos adaptados como o algoritmo de SLAM,

feito por Ramos (2012), mostrou uma otimização do mapeamento, quando fundido aos dados obtidos por sensores de odometria e sensores inerciais, para compensar os erros de mapeamento provocados por erros de sensores.

Murphy (2000) também cita modelos probabilísticos para localização de objeto em uma grade de ocupação usando sonares. Mas este trabalho não possui como foco encontrar um modelo probabilístico de otimização, mas exercer a aplicação, sendo de forma prática o mapeamento territorial, mas com adição de capacidade de intervenção humana.

Os trabalhos de mapeamento implementados de forma prática, feitos por Abner e Yukinobu, exemplificam a capacidade robótica de mapear, um dos objetivos desse trabalho. Murphy, Ramos e Abner mostram conceitos de mapeamento simultâneo com a localização, que serão usados para o monitoramento neste trabalho, por usar uma interface que receberá informações a cada atualização do ciclo de mapeamento local.

Os trabalhos mencionados anteriormente demonstram a abrangência de assuntos relacionados a este tópico, demonstrando a multidisciplinaridade e a grande possibilidade de desenvolvimento de pesquisas por meio do protótipo funcional.

1.6 CONSIDERAÇÕES FINAIS

Após as considerações iniciais expostas neste capítulo, é apresentado no capítulo 2 o Desenvolvimento Teórico, onde é apresentada a teoria e conceitos utilizados neste trabalho. No capítulo 3, Materiais e Métodos, é apresentado os materiais utilizados e seus respectivos custos, assim como a metodologia utilizada para construção do protótipo. No capítulo 4, é apresentado o Desenvolvimento, sendo os Resultados discutidos no capítulo 5. Ao final, são apresentadas as principais conclusões deste trabalho.

2 DESENVOLVIMENTO TEÓRICO

Neste item serão apresentados os principais conceitos teóricos utilizados neste trabalho e necessários para compreensão do mesmo.

2.1 MAPEAMENTO

O mapeamento de uma área significa exploração e (ou) registros de informações da área e locais percorridos. O mapeamento territorial pode ser realizado utilizando a robótica e proporciona, por exemplo, segurança dependendo do risco de sua aplicação, e automatização de trabalhos que exigiriam esforço físico.

Através de mapeamento é possível obter informações úteis do ambiente e planejar de forma mais adequada o trajeto do robô, evitando obstáculos e otimizando o desempenho (KORTENKAMP *et al.*, 1998). O erro em um mapeamento é a ausência de informação necessária que era para ser adquirida no ato de mapear. Uma rota imprecisa em um mapeamento territorial gera um mapeamento com erros. O erro de mapeamento de um território também pode ter origem na imprecisão dos sensores utilizados e falta de informações no mapa devido ao não sensoriamento de uma região caso não tenha como localizar o robô (problema de cobertura).

Existem dois tipos gerais de mapeamento na robótica, o mapeamento por método topológico e o mapeamento por método métrico (SIEGWART, 2011). Duas formas diferentes de mapeamento, ambas possuindo suas vantagens e desvantagens (THRUN, 1998).

2.1.1 Mapeamento topológico

O mapeamento topológico consiste no mapeamento através de captura de pontos de referência territoriais para a orientação. Assemelham-se como as pessoas agem quando perdidas em lugares desconhecidos. Por exemplo, se alguém está perdido, provavelmente, irá adquirir informações necessárias para orientação, como o nome de uma rua ou local, ou talvez algum local ou uma rua próxima popularmente conhecida, este será o ponto de referência territorial.

No mapeamento robótico informações relevantes são coletadas através de sensores, estas são os pontos de referências deste mapeamento, informações como objetos próximos, ou objetos específicos (com cores específicas, no caso de sensores para captura de cor), tipos de informações coletadas que dependem do tipo de sensor usado, e sua aplicação.

Segundo Yukinobu (2010), essas referências podem ter dois tipos de representação, pontos de referência naturais e artificiais. Pontos de referência naturais são pontos de referência que não foram construídos para este propósito, mas podem servir como referência de localização, como por exemplo, casas, lojas, edifícios. No mapeamento por robô móvel dentro de uma casa seria, por exemplo, móveis, portas, quinas, podendo ser encontrados por sensores ultrassônicos, ou infravermelho.

Pontos de referência artificiais são pontos de referências que foram criados para este propósito de referência, como por exemplo, adesivos refletivos em placas de trânsito, gravuras coloridas, ou seja, características que foram adicionadas propositalmente a um objeto para sua identificação. No mapeamento por robô móvel, pontos de referência artificiais seriam, por exemplo, adesivos de uma cor específica colocado em objetos para fácil identificação por câmera ou utilização de sensores RFID (*Radio-Frequency IDentification*, Identificação por Radiofrequência) nas portas de salas onde o robô transita.

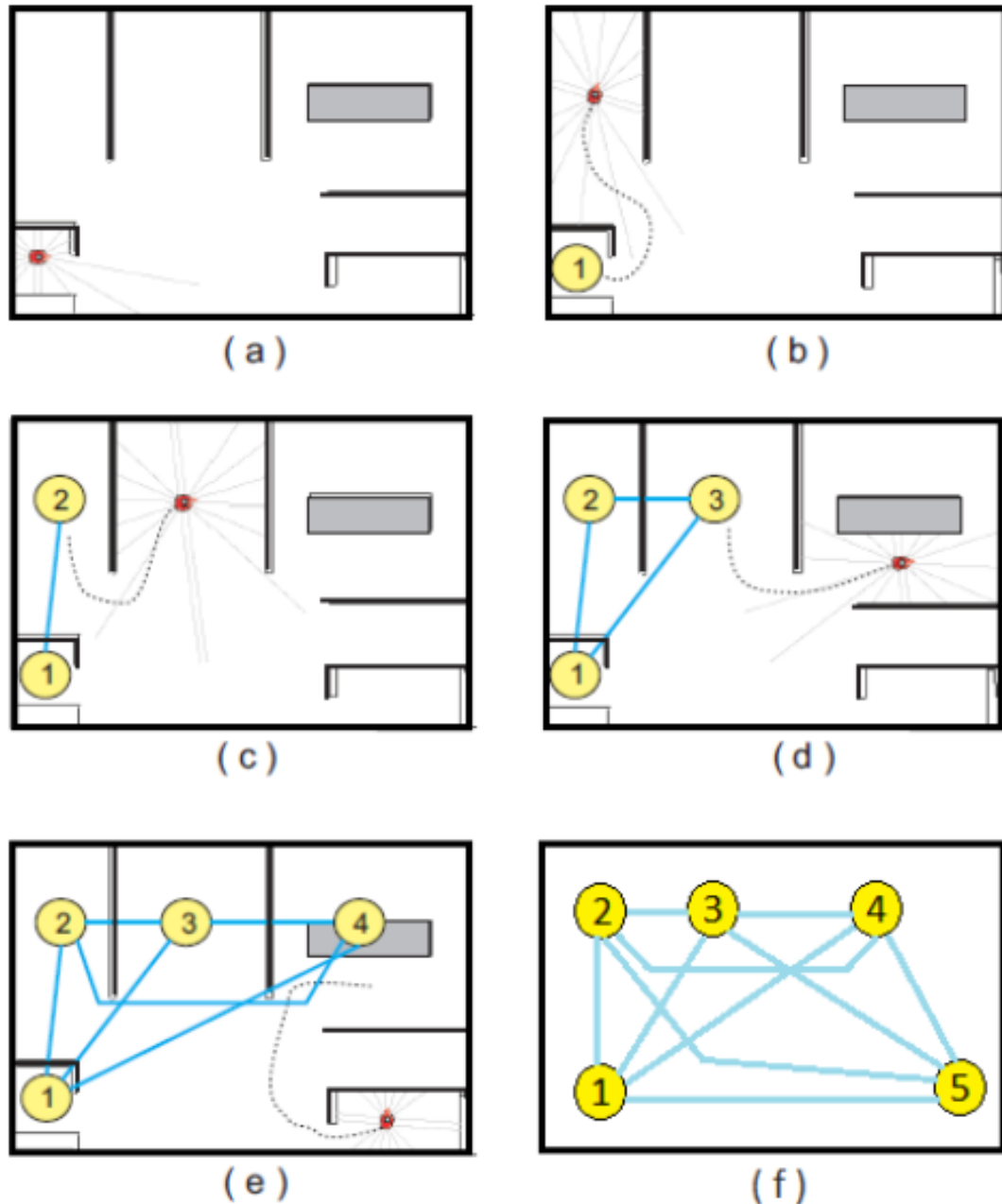
O mapa topológico é basicamente desenhado em grafos destacando os pontos de referências encontrados. Os nós entre as linhas nos grafos representam os pontos de referência, ou objetos encontrados, denominando o lugar, local ou cômodo. As arestas entre os nós representam um caminho ou passagem. O método no mapeamento topológico utiliza de caminhos para ligação de arestas e os nós entre as arestas para representar locais de acesso. Exemplo de mapa topológico na Figura 2.1 retirado da dissertação de Oliveira (2015).

O mapa da Figura 2.1 representa um escritório. Na Figura 2.1(a) o robô móvel (representado pelo objeto vermelho) está em sua posição inicial sem movimento, sendo seu ponto de partida. Os traços saindo do robô representam sua área de obtenção de dados por sensores. Na Figura 2.1(b), o robô saiu do seu ponto inicial e se dirigiu a outra área, a varredura pelos sensores delimitam as paredes ou objetos da área.

O primeiro nó representado na figura pelo círculo amarelo “1” indica o local inicial registrado, o primeiro ponto de referência que denomina um lugar, e também o primeiro nó no mapa topológico. Na Figura 2.1(c), o robô saiu do local “2” e partiu para outra área, o segundo local foi registrado e o mapa topológico foi atualizado. Agora com dois nós interligados por uma aresta, indicando um caminho. Na Figura 2.1(d), o robô saiu do local “3” e partiu para outra área, o terceiro local foi registrado e o mapa topológico foi atualizado. Os nós “1”, “2” e “3”, foram interligados por arestas, pois como existe caminho do ponto de partida “1” para o “2”, e do “2” para o “3”, também é possível um caminho direto do “1” para o “3”. Na Figura 2.1(e), um novo local é registrado, outro nó é adicionado ao mapa

topológico, possibilitando maior variedade de caminhos pela combinação de nós. A Figura 2.1(f) mostra o mapa de grafos final formado pelos nós e arestas.

Figura 2.1 - Mapeamento topológico em etapas graduais.

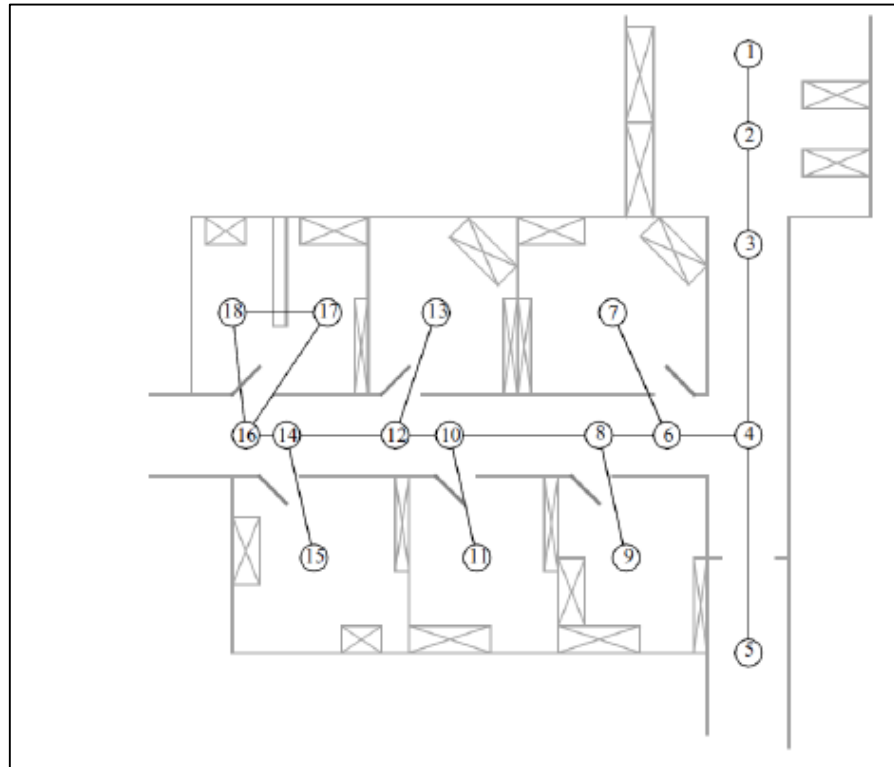


Fonte: OLIVEIRA (2015)

Dependendo da estratégia (OLIVEIRA, 2015), o mapeamento topológico pode ter formato como na Figura 2.1, com nós representando áreas (pontos de referência) e arestas representando caminhos entre eles. E também, como na Figura 2.2, mapa de um escritório, onde os nós não são colocados apenas nos cômodos, mas possuindo nós em áreas com opções de caminhos disponíveis, criando nós em corredores onde há possibilidade de quebra de caminho.

Sua ligação por arestas nesta estratégia demonstra também os caminhos onde há passagem disponível (porta). A simplicidade de um mapa topológico representado por grafos não representa a geometria do ambiente, podendo gerar imprecisão no mapeamento.

Figura 2.2 - Mapa topológico.



Fonte: SIEGWART (2011)

2.1.2 Mapeamento métrico

O mapeamento métrico consiste no mapeamento do ambiente captando suas estruturas geométricas, ou seja, comparando com o mapeamento topológico, o mapeamento possui maior fidelidade ao mundo real, pois capturam, por exemplo, paredes, objetos relevantes, formas do mundo físico que são exibidas no mapa métrico, semelhante a um mapa com seu conteúdo real, divergente do mapa topológico que é representado em grafos.

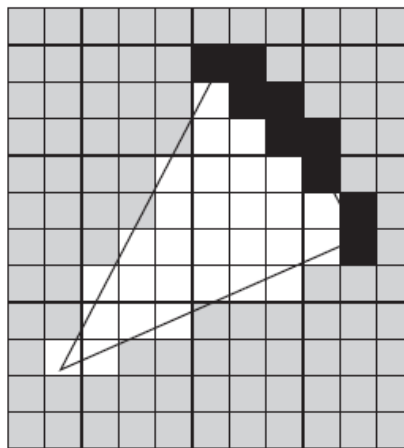
Como o nome sugere o mapeamento métrico captura as características métricas do ambiente (THRUN, 2003), e podem ser classificados de dois tipos, mapas de grade de ocupação e mapas de características (ABNER, 2012). O mapeamento métrico usando grade de ocupação (ou células) é o mapeamento mais utilizado na robótica móvel (MURPHY, 2000) e também é o tipo de mapeamento usado neste trabalho.

2.1.2.1 Mapa de grade de ocupação

No mapa de grade de ocupação o mapa é subdividido em células de mesmas dimensões, as quais representam um espaço em uma área. Essas células representam locais de acesso e locais sem acesso, diferenciadas por coloração.

Quando existe um objeto em um espaço específico, e aquele objeto ocupa uma ou mais células, por exemplo, estas células serão marcadas de uma cor para diferenciar das demais células desocupadas. Paredes e outros obstáculos também podem ser delimitados como marcação de célula de ocupação. O mapeamento neste trabalho foi feito por grade de ocupação

Figura 2.3 - Sonar em mapa de grade de ocupação.



Fonte: ABNER (2012)

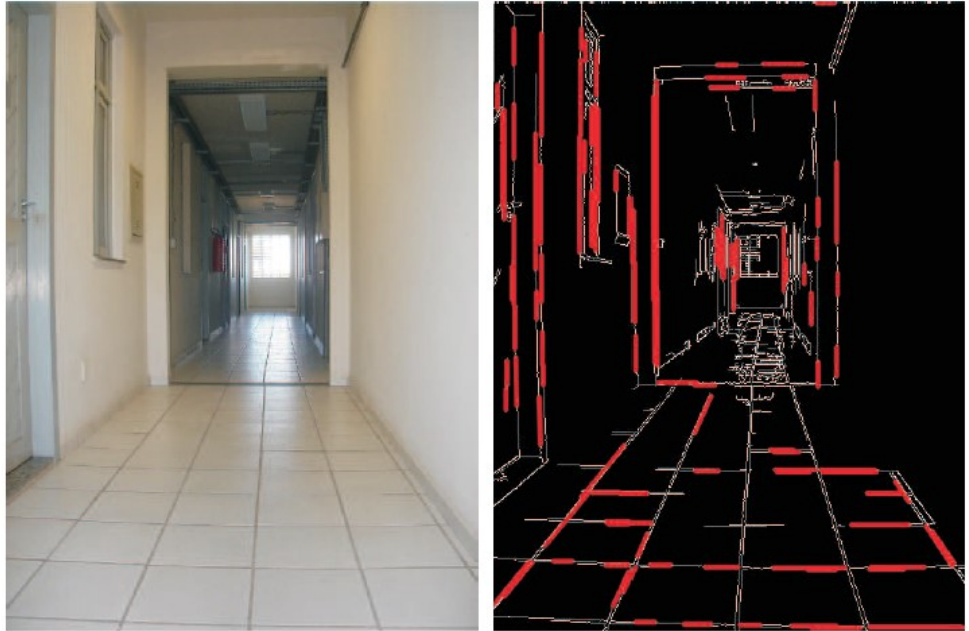
A Figura 2.3, é um exemplo de mapeamento métrico por grade de ocupação utilizando um sonar (ABNER, 2012). As células na figura estão com mesmas dimensões, cada uma representando uma área espacial no mundo físico. Nesta figura as células brancas representam células desocupadas captadas pelo sonar, indicando não ocupação no espaço físico na qual ela representa. As células pretas representam células ocupadas, indicando espaço ocupado no mundo físico. As células cinza indicam indeterminação, ou seja, essas células ainda não foram captadas pelo sonar para determinar a ocupação ou não ocupação.

2.1.2.2 Mapa de características

O mapa de características é um tipo de mapa métrico que captura informações do ambiente de forma representativa dos formatos geométricos que o ambiente possui. Por exemplo, quinas, pontas, círculos, podem ser destacados no mapa, dependendo da estratégia de utilização dos sensores. A Figura 2.4, representa um mapa de características formado pela

aquisição de formas geométricas relevantes no para o usuário, no caso, um mapa gerado por detecção de retas de um corredor.

Figura 2.4 - Resultado de mapeamento por mapa de características.



Fonte: ABNER (2012)

2.1.3 Comparação entre mapeamento topológico e mapeamento métrico

No artigo de Thrun (1998) foi comparado os dois tipos de mapas, mapa topológico e mapa métrico, ressaltando vantagens e desvantagens de cada um em relação ao outro. Thrun destacou algumas vantagens e desvantagens de cada mapa, o que implica nos seus respectivos mapeamentos e suas facilidades e dificuldades em obter, manter e compreender informações.

O mapeamento métrico por gerar mapas que possuem uma fidelidade maior com o ambiente físico, considerando células de ocupação e não ocupação em um mapa de grade possui maior confiabilidade em relação a um mapa de grafos gerado por mapeamento topológico. Possui uma fácil construção por depender de presença ou ausência espacial, sendo facilmente representada por células.

A utilização do mapa por células permite uma representação de ambientes em larga escala com diferentes ocupações, obtendo uma construção de acordo com as variações geométricas do ambiente. Também possui maior facilidade em sua manutenção, pois sua construção geralmente (ABNER, 2012) é feita por incremento de célula, com base na obtenção de dados dos sensores e odometria.

Em um mapeamento por grade de ocupação, como visto na Figura 2.3, enquanto um robô está se movendo, os sonares captam obstáculos e esses obstáculos são representados. A posição atual do robô também pode ser calculada através de odometria e com outros tipos de sensores. As informações adquiridas do meio físico que irão definir uma aproximação da localização no mapeamento métrico, estimando a posição da célula ocupada por distância percorrida e distância dos obstáculos. Esse tipo de mapeamento permite obter uma localização atual do robô precisa (ABNER, 2012).

O mapeamento métrico permite captação da geometria do ambiente e sua geometria espacial possui características diferentes de acordo com a variação da posição no mapa, essas variações geométricas em diferentes lugares permitem distinguir lugares mesmo que sejam próximos. Por exemplo, se em um mesmo cômodo o robô que usa mapeamento métrico se moveu uma distância X, essa variação de distância é representada no mapa.

No mapeamento topológico, apenas informações relevantes para a criação de um nó serão coletadas, criando um mapa de grafos baseados em nós e arestas, sem uma representação física fiel ao território, carente de informações territoriais do meio, podendo ocorrer casos de ambiguidade quando existem locais com grande semelhança, não distinguindo estes lugares.

Comparado ao mapeamento métrico, o mapeamento topológico mesmo carecendo de informações geométricas do ambiente, pela sua simplicidade de abordagem, não necessita de alto processamento e não necessita de um maior espaço para dados coletados, pois coletam dados simples para formar pontos de referência e a ligações entre eles.

O mapeamento métrico possui maior custo computacional comparado ao mapeamento topológico (THRUN, 1998), pois necessita de grande número de informações do território para formar o grupo de células para o mapa.

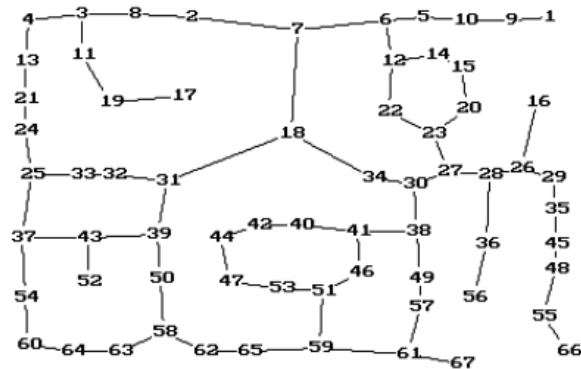
O mapeamento topológico por utilizar simbologias para representação de locais e caminhos o torna intuitivo para resolver problemas que envolva planejamento de rotas e compreensão simplificada. Por exemplo, na Figura 2.1, supõe que o robô esteja no início e procura uma rota nesse mapa para levar o menor tempo possível para chegar a “5”, com este tipo de abordagem ele irá de “1” para “5”. A simbologia deste mapa permitiu fácil compreensão para resolver este tipo de problema.

Figura 2.5 - Mapa métrico.



Fonte: Adaptado por YUKINOBU (2010)

Figura 2.6 - Mapa topológico.



Fonte: Adaptado por YUKINOBU (2010)

Os mapas apresentados nas Figura 2.5 e Figura 2.6, representam os dois tipos de mapa respectivamente, mapa métrico e mapa topológico. Duas representações de um mesmo ambiente usando mapas diferentes na robótica.

2.2 MAPEAMENTO UTILIZANDO SENSORES

Receber informações do ambiente é necessário quando tratam de mapeamento, dependendo da aplicação o mapeamento pode ser feito de forma métrica ou topológica, mesmo que existam mapeamentos híbridos envolvendo os dois tipos de mapeamento para uma otimização, compensando as desvantagens em relação de um mapeamento com outro.

As informações coletadas pelos sensores permitem uma interpretação do mundo físico para o mapeamento, através de dados coletados do meio é possível construir uma representação 3-D ou 2-D do ambiente. Na tese de mestrado de Abner (2008), ele utiliza sensor sônico para a construção do mapa, usando mapeamento métrico por grade ocupação.

Gonçalves (2011) utilizou como sensores a laser e de odometria para mapeamento. Neste trabalho, para obter informações, foi necessário utilizar sensores. Os sensores mais comuns para obtenção de informações para mapeamento de um protótipo compacto são lasers de infravermelho, câmera, sensores de velocidade (YUKINOBU, 2010) e ultrassônico (ABNER, 2008), estes dois últimos foram utilizados neste trabalho.

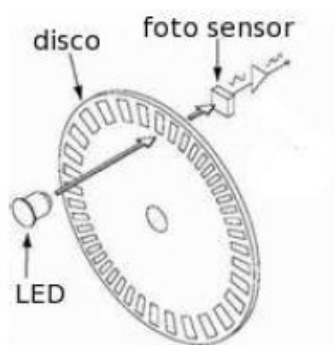
2.2.1 Sensor de velocidade (*encoder* óptico)

O sensor de velocidade mais comumente utilizado é o *encoder* óptico (RAMOS, 2012), sensor que foi utilizado neste trabalho. Este sensor é capaz de informar o número de pulsos durante a rotação de uma roda por meio de leitura de feixe de luz. Na roda (ou motor), é anexado um disco perfurado, o feixe de luz ao passar no disco em estado de rotação, resulta um feixe com inúmeras interrupções.

Segundo Ramos, sensores de velocidade que utilizam *encoders* estão sujeitos a erros dependendo de fatores externos atuantes, como deslizamento de pneu ou imperfeição do solo que provoca erros de rotação.

A Figura 2.7 representa um *encoder*, o disco com os seus furos espaçados igualmente para a criação de pulsos enquanto o disco gira. Provocado por alternância de estados de recepção ou não recepção de luz infravermelha emitida. Recepção feita por um sensor óptico.

Figura 2.7 - *Encoder* óptico



Fonte: ABNER (2008)

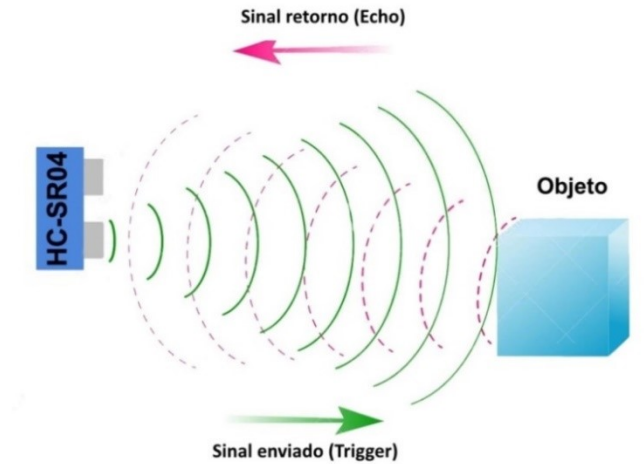
Os dados obtidos pelo sensor foram usados para determinar a localização do robô no mapeamento.

2.2.2 Sensor de ultrassom

O sensor ultrassônico é capaz de medir a distância de um ponto (onde está fixado o sensor) a um objeto. Para realizar a medição, o sensor emite ondas sonoras (*trigger*) que

refletem no objeto e captam seu retorno (*echo*). De acordo com a velocidade de propagação do som e o tempo de emissão e captação do sensor é possível calcular a distância a um objeto.

Figura 2.8 - Exemplo de atuação de um sensor ultrassom.

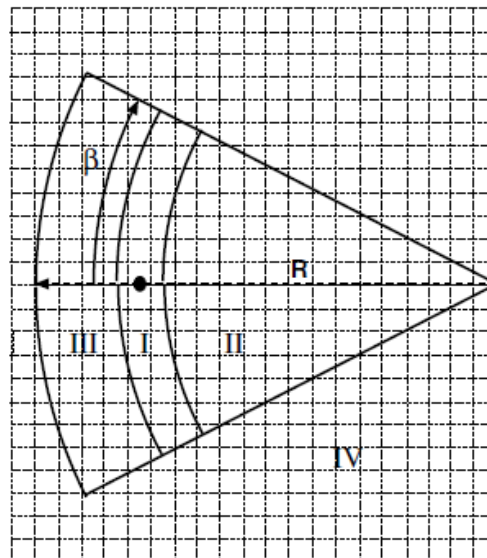


Fonte: THOMSEN (2011)

A Figura 2.8 exemplifica a atuação do sensor de ultrassom. O sensor nesta figura é o HS-SR04, bastante utilizados em projetos eletrônicos, as ondas sonoras emitidas pelo sensor são o sinal enviado (*trigger*), e as ondas refletidas são o sinal de retorno (*echo*).

A emissão de ondas sonoras pelo ultrassom acoplado a um robô permite a detecção de paredes, ou barreiras no meio físico. O ultrassom foi utilizado no mapa de grade de ocupação.

Figura 2.9 - Alcance do sensor ultrassom.



Fonte: MURPHY (2000)

A Figura 2.9 mostra o *range* do ultrassom, ou seja, a área de visão em um ambiente em duas dimensões. R na figura é o raio máximo do alcance que o ultrassom pode detectar, β é a metade do ângulo da largura do cone de visão. Essa área de visão pode ser projetada em

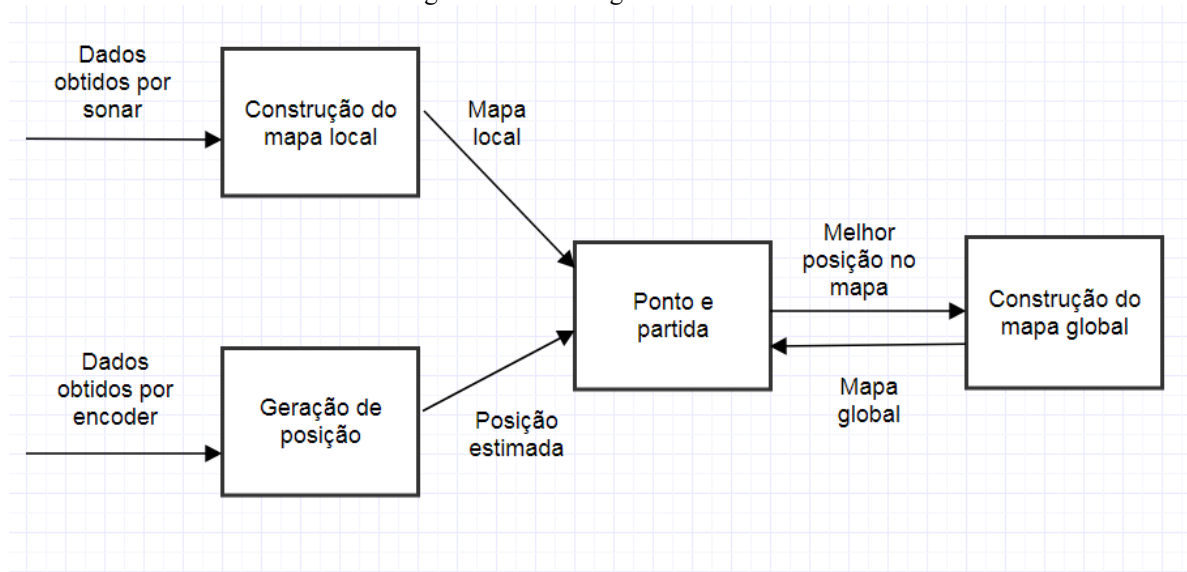
grade de ocupação, pois para cada célula na grade, manterá um estado que representa se o local está ocupado ou vazio (MURPHY, 2000).

Como mostrado na Figura 2.9, a área de visão, pode ser dividida em 3 regiões. Região I, é a região que provavelmente a célula da grade está ocupada, detectado pelo sensor. Região II, é a região de detecção de células vazias. Região III, é a região de incerteza, pois não possui confirmação da ocupação ou não das células. A leitura do sensor possui maior chance de estar correta ao longo do eixo linear originado no ultrassom, do que nas bordas, pois podem existir obstáculos nas bordas que refletem as ondas emitidas (MURPHY, 2000).

2.3 MONITORAMENTO

O conceito de monitoramento neste trabalho é a capacidade de um usuário assistir o mapeamento sendo feito graficamente em uma interface de visualização enquanto o robô mapeia o ambiente.

Figura 2.10 - Fluxograma do SLAM.



Fonte: Autor.

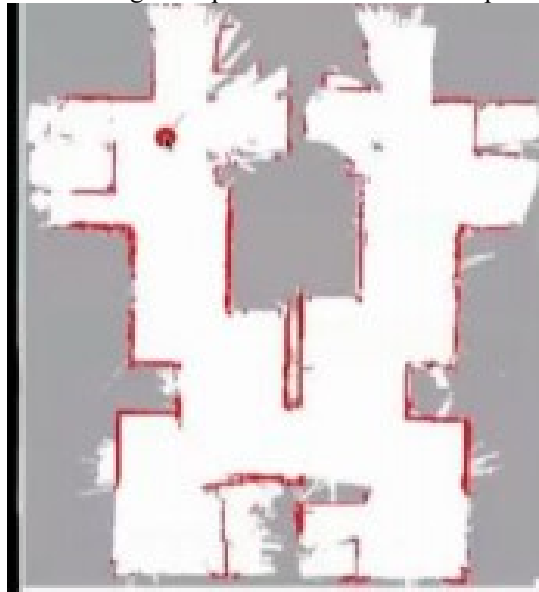
A Figura 2.10, mostra um esquema de SLAM. Nesta figura, os dados obtidos por um sonar, permitem a construção de um mapa local, ou seja, um mapa limitado pelo alcance do sonar. Os dados obtidos pelo *encoder* permitem gerar uma posição estimada do robô. A união das informações de posição estimada com o mapa local construído permite gerar uma posição que o robô está situado no mapa local, esta posição do robô é a sua localização. Esta localização será também a posição de partida do robô para a próxima área que será mapeada conforme o seu deslocamento.

O acúmulo de informações de mapeamento obtidas a partir da posição do robô e seu deslocamento permite construir um mapa mais extenso, denominado na figura como “mapa global”, as informações do mapa global fundidas com a localização permite encontrar a posição do robô não apenas no mapa local, mas também no mapa global, permitindo assim uma melhor posição no mapa. Esta posição no mapa por sua vez remete também a posição de partida do robô para a próxima área que será mapeada, formando ciclos de atualização de mapa.

A cada ciclo de atualização as informações são enviadas a um computador. O mapeamento pode ser observado pelo usuário utilizando uma interface gráfica programada para mostrar o mapa visualmente conforme cada atualização.

A imagem na Figura 2.11 representa um exemplo de interface gráfica para monitoramento e mapeamento do ambiente. O robô está como o “ponto” no mapa gerado e as paredes estão como “linhas” de vermelho, áreas brancas foram mapeadas e áreas cinza, não mapeadas.

Figura 2.11 - Interface gráfica para monitoramento e mapeamento.



Fonte: MACHARET (2017)

2.4 PROCESSAMENTO

Para processar as informações adquiridas pelos sensores e controlar suas ações em um exemplo prático de robótica, é necessária uma plataforma de *hardware* e *software*. Neste trabalho, será utilizada a plataforma Arduino para o processamento, por ser uma plataforma de baixo custo, popular em projetos eletrônicos, fácil acesso, fácil uso, seu *hardware* possui

um microcontrolador com a placa que permite o encaixe de módulos, vem com o *Software* de programação Arduino IDE e usa linguagem de programação.

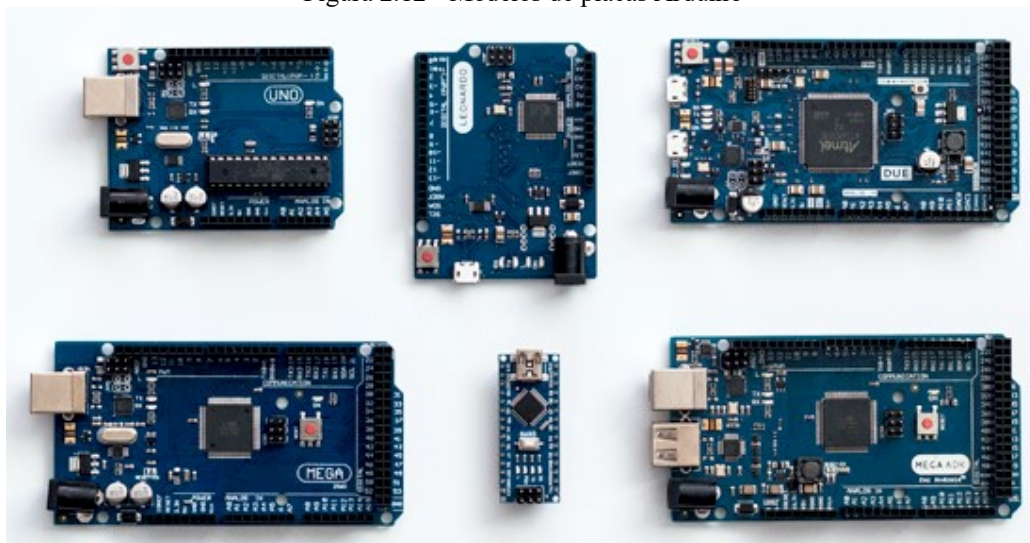
A placa de Arduino possui um microcontrolador capaz de processar informações coletadas pelos sensores para diferentes aplicações. Como por exemplo, para monitoramento, no artigo de Nasution, Siagian e Tanjung (2018), um Arduino foi usado com um sensor ultrassônico para monitorar o nível de um rio.

O Arduino consiste de uma placa composta por um microcontrolador Atmel, com circuitos de entrada/saída e os componentes necessários para funcionamento do microcontrolador. A placa pode ser facilmente conectada à um computador e programada via IDE (*Integrated Development Environment*) utilizando uma linguagem baseada em C/C++, sem a necessidade de equipamentos adicionais além de um cabo USB (*Universal Serial Bus*) e adaptador, caso precise.

Se o Arduino é capaz de ser usado para processar dados de distância do nível de um rio usando ultrassom, é considerado que a placa também atende as necessidades para processamento de dados da aplicação desejada de monitoramento e mapeamento.

O tipo de placa Arduino a ser utilizada depende do projeto a ser desenvolvido e principalmente, do tamanho e do número de portas digitais e analógicas necessárias. As opções variam desde a placa padrão Uno e suas 14 portas digitais e 6 analógicas, passando por placas com maior poder de processamento e de portas, como o Arduino Mega, com microcontrolador ATmega2560 e 54 portas digitais, e o Arduino Due, baseado em processador ARM de 32 bits (Figura 2.12).

Figura 2.12 - Modelos de placas Arduino



Fonte: Adaptado pelo Autor

2.5 COMUNICAÇÃO

Em um mapeamento feito por robô à longa distância, uma comunicação sem fio entre usuário e máquina deve ser escolhida para melhor obtenção de resultados. No artigo de Alshazly e Hassaballah (2016), foi criado um robô móvel com comunicação *Bluetooth*, seu controle é feito por um usuário à distância. Este robô também é capaz de captar informações de aproximação pelo sensor infravermelho acoplado. A tecnologia *Bluetooth* usada neste caso é para objetivos simples de controle de um robô compacto para andar em pequenas regiões.

Outra possibilidade de tecnologia para a comunicação é a utilização do Zigbee, tecnologia baseada no padrão IEEE 802.15.4. O Zigbee é voltado para aplicações de menos potência, menor taxa de dados e menor ciclo de trabalho do que *Bluetooth*. Embora contra intuitivo, nem todas as aplicações de rede necessitam de muita largura de banda e dos custos mais altos decorrentes disto. Zigbee define taxas de canal de 20, 40, 100 e 250 Kbits/s, dependendo da frequência do canal.

Em objetivos de mapeamento que procura uma área de cobertura maior, a tecnologia de comunicação sem fio deve possuir uma capacidade de transmissão de alcance considerável, e por tratar na possibilidade de mapeamento em áreas de risco, essa transmissão deve possuir capacidade de monitoramento não presencial na área de risco.

No artigo de Reddy (2014), ele projetou robôs mestre e escravos por comunicação *wireless*, com o intuito da sua aplicação em áreas de risco, destacando principalmente ambientes de guerra para levar quites médicos, detectar minas terrestres e entrar em áreas de riscos em geral. Um sistema desse patamar para controlar ou monitorar mais de um robô por distância, é viável utilizar um sistema com conexão a Internet, como sugestão utilizar Wi-Fi (*Wireless Fidelity*). Similar ao sistema de mapeamento, usando comunicação Wi-Fi poderia controlar inúmeros robôs para mapear um local, ou locais distintos pela Internet.

Há diversos padrões 802.11 (Wi-Fi), dentre eles o 802.11 b, n e g. O 802.11g é, de longe, a tecnologia mais popular e fácil de encontrar. Os padrões 802.11 compartilham muitas características. Todos usam o mesmo protocolo de acesso ao meio, CSMA/CA, usam a mesma estrutura de quadro para seus quadros de camada de enlace, têm a capacidade de reduzir sua taxa de transmissão para alcançar distâncias maiores e permitem a funcionalidade “modo de infraestrutura” e “modo ad hoc”.

A comunicação feita neste trabalho foi feita inicialmente por comunicação *Bluetooth* devido a complexidade do projeto. A comunicação por Wi-Fi será posta como uma meta a ser

alcançada em um trabalho posterior a este, quando o sistema de mapeamento e o protótipo robótico já estiverem operacionais.

2.6 CONSIDERAÇÕES FINAIS

Neste capítulo, foram apresentados teorias e exemplos de trabalhos feitos sobre mapeamento robótico feito por robôs móveis, os tipos de mapeamento disponíveis, os sensores que serão utilizados e suas respectivas funções. Também foram apresentados, conceito teórico e exemplos de comunicação sem fio para robótica móvel. Teorias e exemplos que foram utilizados para a construção prática do protótipo deste trabalho.

O foco deste trabalho é o mapeamento de um ambiente e sua representação em mapa de grade de ocupação em uma interface enquanto o robô mapeia o ambiente, assim podendo ser assistida por um usuário com possibilidade de ele intervir no controle do robô. A comunicação foi por meio de *Bluetooth*, tendo como meta futura, a implementação de uma comunicação Wi-Fi.

No capítulo seguinte será abordado os materiais e métodos que foram utilizados neste trabalho.

3 MATERIAIS E MÉTODOS

Nesta seção é discutida a etapa de preparação e aquisição das peças do protótipo. A metodologia adotada no trabalho está alinhada com a conclusão dos objetivos específicos, seguindo os itens descritos a seguir.

- **Pesquisa bibliográfica:** A pesquisa foi realizada por meio da leitura de livros, *sites* e artigos da área, selecionando as referências mais relevantes e analisando as principais vantagens e desvantagens de cada tecnologia utilizada.
- **Construção mecânica do protótipo:** O segundo passo foi o projeto e construção do protótipo robótico, definindo a utilização de uma placa baseada no Arduino e o sistema de comunicação via *Bluetooth*, motivado pela redução da dificuldade de uma primeira implementação. O protótipo também deve contar com seu sistema de energia (bateria), placa de acionamento dos motores, sensor de posição e o sensor de distância. O sistema de energia escolhido foi bateria recarregável de 9v, a configuração dos motores DC no formato diferencial, *encoder* para posicionamento e como sensor de distância, o sensor ultrassônico.
- **Comunicação e interface de usuário** A comunicação é feita utilizando o *Bluetooth*, no qual, se corretamente configurado e instalado, é transparente para comunicação serial, sendo tratada da mesma forma que uma comunicação via cabo USB. Após estabelecer a comunicação, é escolhida uma interface gráfica de usuário usando o *software Processing*.
- **Finalização do desenvolvimento:** Após a montagem do protótipo robótico e do sistema de comunicação, com estes funcionando adequadamente separadamente, a programação do microcontrolador do protótipo e da interface é feita para interligar todos estes componentes.
- **Avaliação e teste final:** Para o teste final do robô, é realizado um teste prático de mapeamento. Para o teste, a construção do mapa é assistida enquanto o robô explora de forma autônoma e (ou) manual.

3.1 MATERIAIS E CUSTOS

Para executar de forma prática o objetivo proposto neste TCC foi necessário a pesquisa de componentes que atenderiam a demanda do projeto e adquirir os materiais. A lista dos principais componentes e de seus custos são listados na Tabela 3.1.

Tabela 3.1 - Lista de componentes do projeto e respectivos valores.

Item	Preço (R\$)
MBZ Pro Mega Wi-Fi Edition	30,00
Sensor Ultrassônico SR05	5,00
Motor <i>Shield</i> L293D <i>Driver</i> Ponte H para Arduino	8,00
Bateria 9V recarregável com carregador	35,00
Estrutura Mecânica em acrílico com motores e sensores HC-020K	45,00
Módulo Bluetooth HC 06	10,00
Módulo Bluetooth para Computador	20,00

Fonte: Autor.

3.2 CONSIDERAÇÕES FINAIS

Neste capítulo, foram apresentados de forma resumida os materiais e métodos utilizados para elaboração do projeto, considerando um protótipo com sensoramento simples por ultrassom e comunicação por *Bluetooth*. O passo a passo do projeto, ou seja, o seu desenvolvimento será descrito na próxima seção.

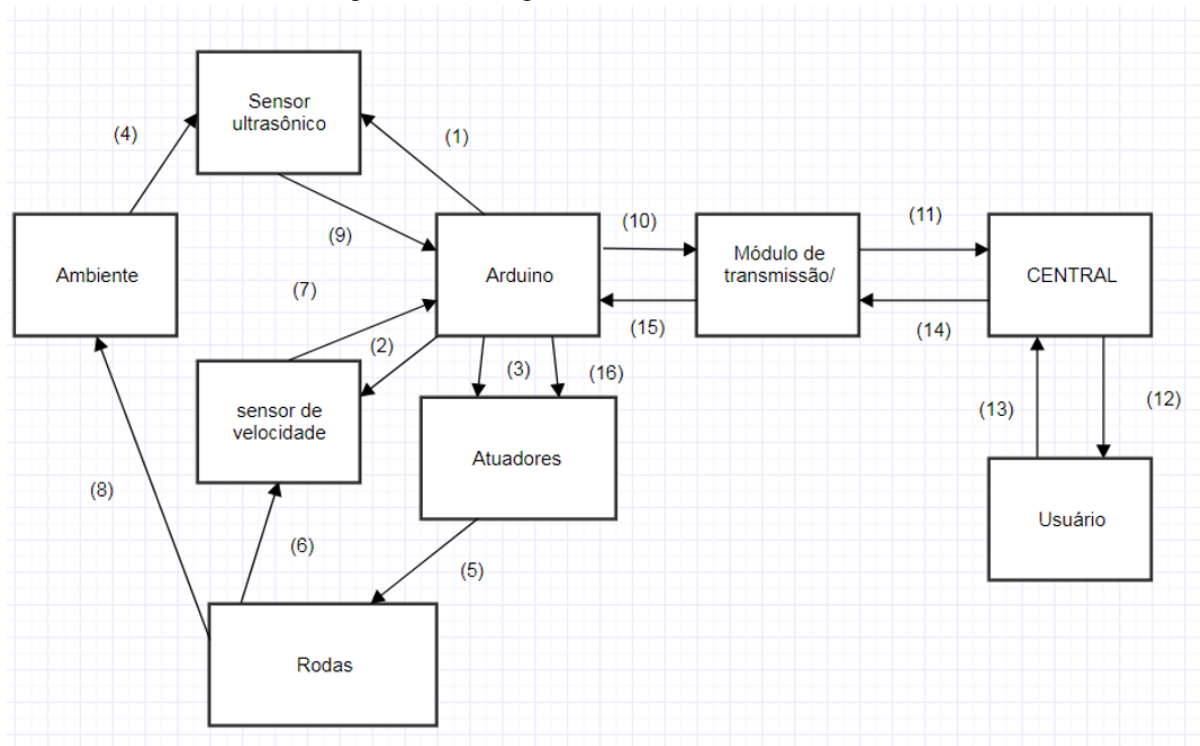
4 DESENVOLVIMENTO

O desenvolvimento do protótipo foi realizado conforme descrito na metodologia e possui a estrutura indicada na Figura 4.1. Nesta, (1), (2), (3) são comandos programados para a função de desempenho dos sensores e atuadores (motores DC). O (5) é a atuação da roda, que está acoplada a um *encoder*.

O (8) é o deslocamento do robô pelo movimento das rodas provocando variação territorial no ambiente. O (6) e (4) são dados físicos obtidos pelo deslocamento do robô e dados do ambiente respectivamente.

Já os itens (9) e (7), ambos se referem ao envio da informação dos sensores para o Arduino, este que irá trabalhar conforme os dados recebidos. O (10) se refere à conexão do Arduino com o módulo *Bluetooth*, representando o módulo para o envio dos dados.

Figura 4.1 - Fluxograma de funcionamento do robô.



Fonte: Autor.

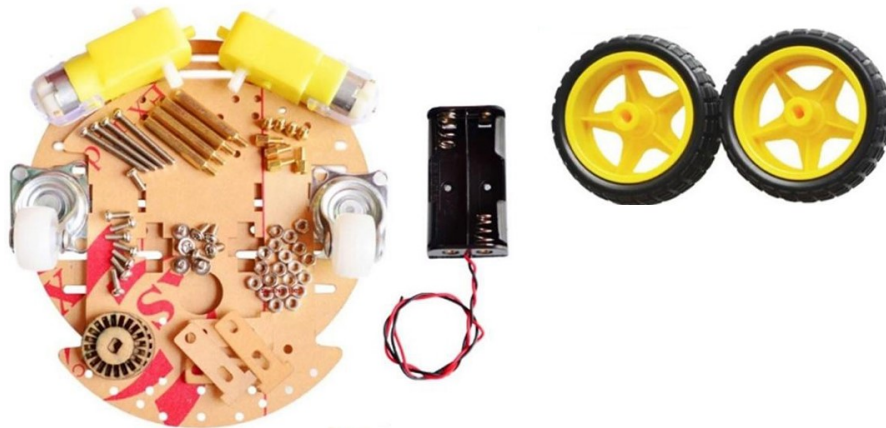
A etapa (11) representa o envio dos dados de distância por *Bluetooth* para uma central. Neste o mapa gráfico visual é gerado conforme a programação do Arduino em conjunto com os sensores, atuadores e deslocamento do robô no ambiente. O (12) se refere ao monitoramento pelo usuário, pela capacidade de assistir o mapeamento em uma interface gráfica. O (13) refere-se à interação do usuário, caso ele queira dar comandos ao robô.

O (14) é o comando da central que é recebido pelo módulo. O (15) representa a conexão do módulo com a placa do Arduino, representando a recepção dos dados pelo robô. O (16) é o comando manual do usuário que é convertido do Arduino para ação dos atuadores.

4.1 CONSTRUÇÃO MECÂNICA

A estrutura mecânica do protótipo não necessitou ser construída, pois feita a aquisição de uma estrutura de acrílico já com as furações, motores e apoios. Esta estrutura de baixo custo (Figura 4.2) não possui grande resistência mecânica e nem é recomendada para uso em terrenos irregulares, mas atende ao propósito do projeto ao fornecer suporte aos motores, aos discos do *encoder* e para a eletrônica.

Figura 4.2 – Peças da estrutura de acrílico utilizada no projeto.



Fonte: Adaptado pelo Autor.

A montagem da estrutura é simples e necessita apenas posicionar os parafusos e encaixar a duas placas de acrílico (Figura 4.3).

Figura 4.3 – Vista superior (esquerda) e inferior (direita) da estrutura de acrílico montada.



Fonte: Adaptado pelo Autor.

As rodas livres (frontal e traseira) servem apenas de apoio para o robô não tombar durante a sua movimentação e foi considerada que a sua utilização não interfere na movimentação do robô. O suporte para pilhas foi desconsiderado no projeto, tendo em vista que foi utilizado uma bateria de 9v para alimentar a placa e os motores.

4.2 COMPONENTES ELETRÔNICOS

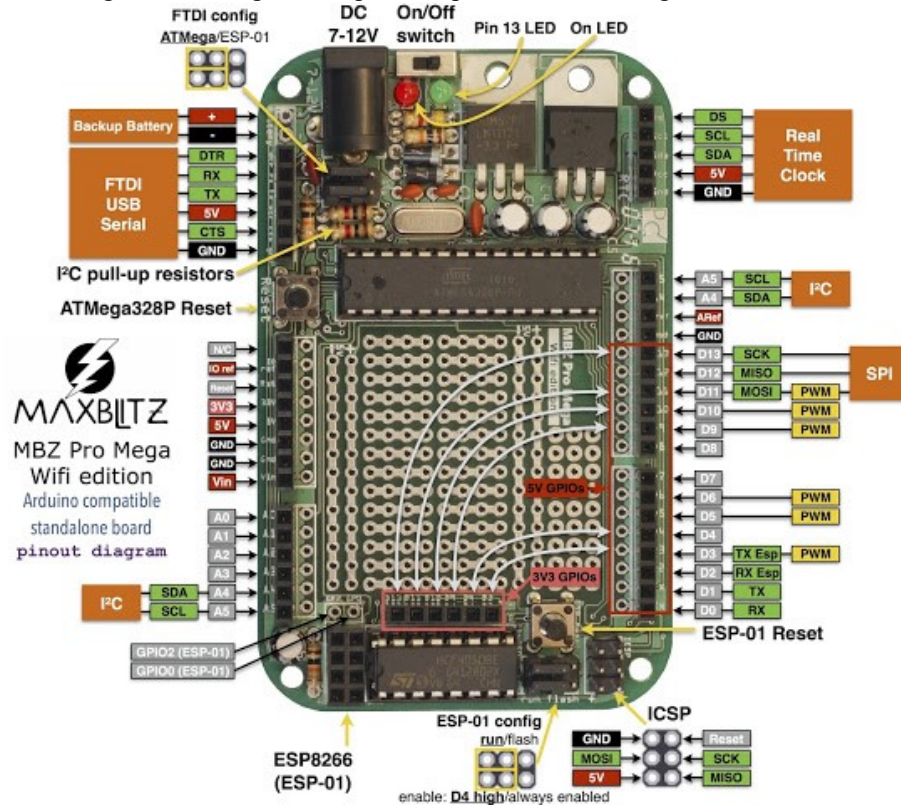
Os componentes eletrônicos listados na seção anterior necessitam um estudo mais detalhado de seu funcionamento, de suas ligações e interações com outros componentes. Estes são descritos neste item com seus respectivos códigos de teste, antes de serem integrados no projeto.

4.2.1 Placa MBZ Pro Mega

O protótipo foi construído por etapas, para cada parte do protótipo ser testada separadamente. A placa de circuito usada no trabalho foi o MBZ Pro Mega Wi-Fi Edition (Figura 4.4), é uma versão adaptada do Arduino Uno (Figura 4.5). Criada por Marcelo Maximiano (<https://www.blogger.com/profile/06251877058894470676>), a placa contém mais portas comparada com a placa do Arduino Uno, saídas digitais com 3,3 V e uma entrada para o módulo Wi-Fi ESP-01, é uma placa voltada para projetos de IoT (*Internet of Things*). Ela possui o mesmo microcontrolador do Arduino Uno, ATmega328P, e possui áreas próprias para soldagem (MAXIMIANO, 2016).

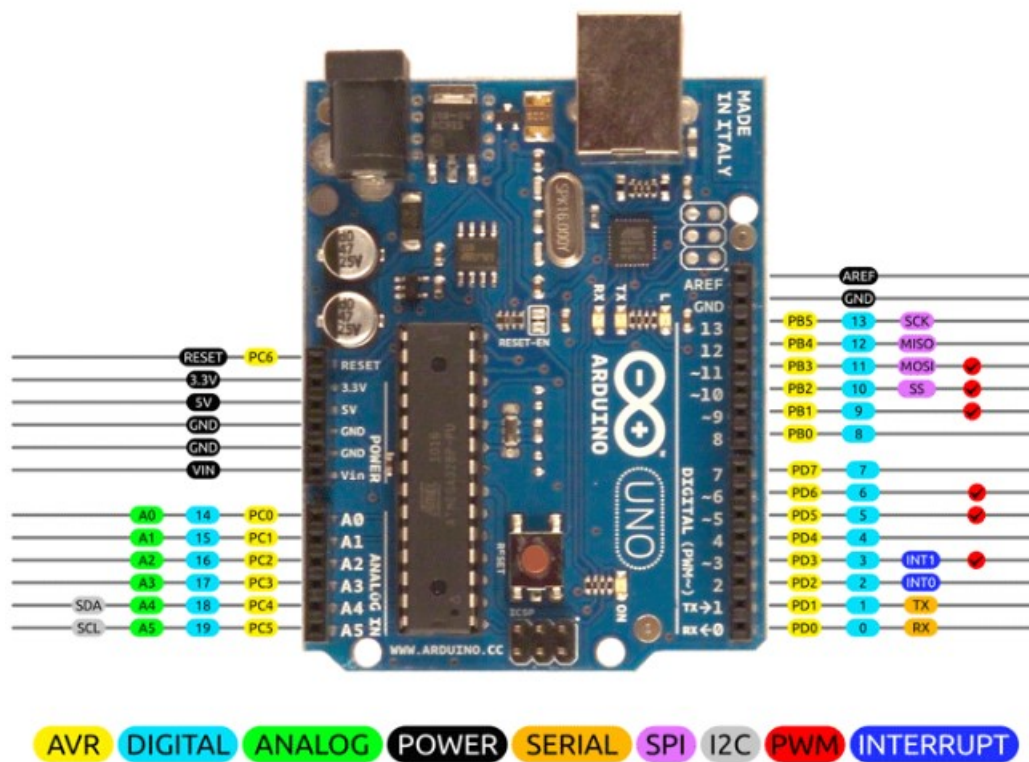
Essa placa foi escolhida inicialmente por possuir uma entrada própria para o ESP-01, para futuramente ser usada com uma comunicação Wi-Fi, mas neste trabalho a comunicação usada foi o *Bluetooth* pela sua simplicidade. Uma placa Arduino Uno também poderia ser usada neste trabalho substituindo a MBZ. O quadro a seguir compara a placa MBZ, com a placa Arduino Uno (MAXIMIANO, 2016).

Figura 4.4 - Diagrama de pinos da placa MBZ Pro Mega Wi-Fi Edition.



Fonte: MAXIMIANO (2016)

Figura 4.5 - Diagramas de pinos do Arduino UNO.



Fonte: THE ROBOTICS BACK-END (2019).

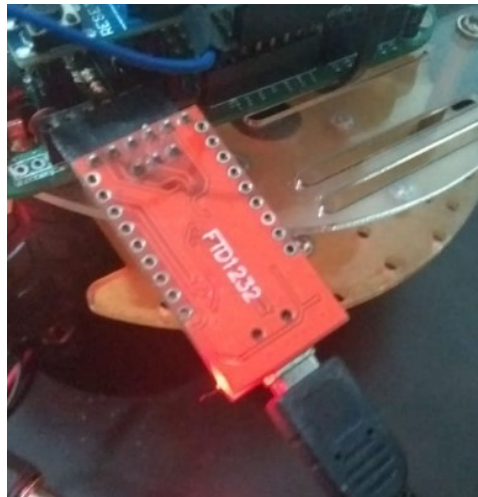
Quadro 4.1- Comparação entre a placa MBZ e UNO.

Características	MBZ Pro Wi-Fi	Arduino Uno
Microcontrolador	ATMega328P	ATMega328P
Interface USB	Não	Sim (ATMega16U2)
Regulador de voltagem 5 V	LM7805, máximo 1000 mA	NCP1117ST50T3G, máximo 800 mA
Regulador de voltagem 3,3 V	LM1117, máximo 800 mA	LP2985-33DBVR, máximo 150 mA
Compatível com <i>Shields</i>	Sim	Sim
Conector para RTC (<i>Real Time Clock</i>)	Sim	Não
Conexão Wi-Fi	Sim (ESP-01)	Não
Portas digitais 3,3 V	6 portas (D3/D4/D9/D10/D11/D13)	Não
Permite personalização	Sim (Possui área para soldagem de componentes e módulos)	Não

Fonte: MAXIMIANO (2016)

A placa de Arduino MBZ foi testada separadamente, usando a interface para programação e gravação de dados, Arduino IDE. Por não possuir entrada para porta USB, foi usada uma Placa FTDI (*Future Technology Devices International*) FT232RL Conversor USB Serial (Figura 4.6) para a comunicação com o PC. Segundo Maximiano, se o adaptador FTDI possuir a sequência de pinos “DTR, RX, TX, VCC, CTS e GND”, ele pode ser ligado de forma direta na placa. Caso contrário pode ser conectado através de fios *jumpers*.

Figura 4.6 - Placa FTDI FT232RL Conversor USB Serial



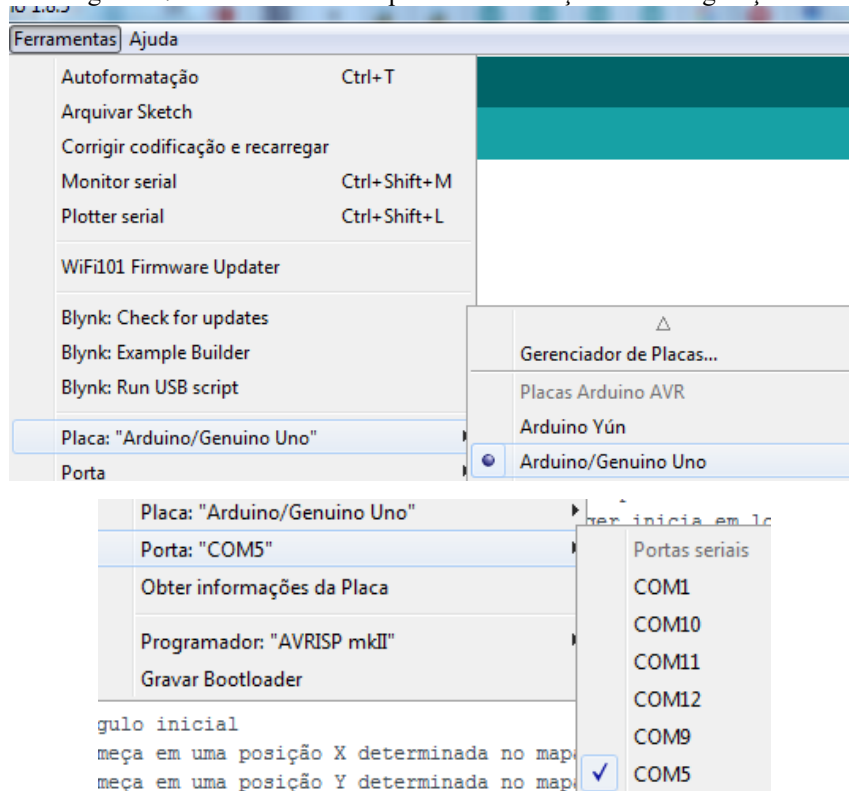
Fonte: Autor.

Foi necessário um cabo de dados para conexão do PC com a placa Arduino pelo FTDI. Para conseguir executar o programa corretamente foi necessário fazer o *Download* do *Driver* FTDI, este para não ocorrer erros de porta na IDE do Arduino quando o FTDI está conectado a placa de Arduino.

O *software* Arduino IDE usa o “dialetto” C/C++ (ARDUINO, 2019). Um código simples para testar a placa é o “Blink”, disponível nos exemplos do Arduino IDE

(ARDUINO, 2015) verificando se o LED (*Light-Emitting Diode*) da placa pisca de forma alternada. Para gravar na placa, a “placa” selecionada no Arduino IDE foi “Arduino/ Genuino Uno”, o mesmo usado para gravação em placas Arduino Uno (Figura 4.7). A porta serial “COM” deve ser selecionada corretamente, esta é a porta de comunicação entre a placa Arduino e o PC, no caso a porta usada foi a “COM5” (Figura 4.7).

Figura 4.7 - IDE do arduino e processo de seleção das configurações.



Fonte: Autor.

Após a verificação da iluminação do LED na placa, outro teste foi imprimir dados na porta serial (Quadro 4.2). A variável usada, “x”, foi declarada antes da função “setup()” e a função de impressão foi feita dentro da função “loop()”. O código feito dentro da função “setup()” foi inicializado para configurar as variáveis, a função é executada apenas uma vez cada vez que a placa de Arduino é ligada (ARDUINO, 2019). O código feito dentro da função “loop()” se repete em *loop*, acontecendo repetições a cada término (ARDUINO, 2019).

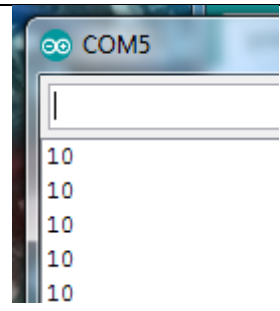
A taxa de dados da comunicação serial foi declarada em “Serial.begin()”, entre os parênteses é colocado a taxa de dados em bits/s. A função frequentemente usada neste trabalho foi “Serial.println()” que imprime os dados da porta serial e a próxima impressão segue na próxima linha (ARDUINO, 2019).

Quadro 4.2 - Código de teste da serial.

```

int x;
void setup() {
  // put your setup code here, to run once:
  x=10;
  Serial.begin(9600);
}
void loop() {
  // put your main code here, to run repeatedly:
  Serial.println(x);
}

```



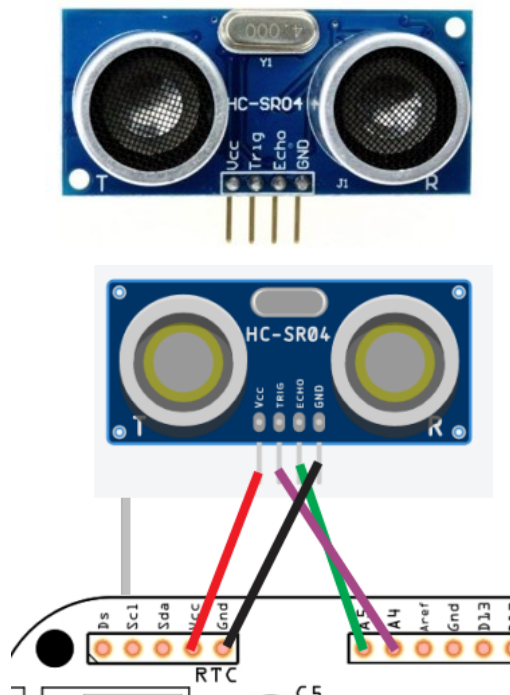
Fonte: Autor.

O teste resultou inúmeros “10” impressos no “Monitor Serial” (Quadro 4.2). Após sucesso no teste de funcionamento da placa, os itens necessários para o robô também foram testados de forma separada no decorrer do trabalho.

4.2.2 Sensor ultrassônico

O sensor ultrassônico utilizado no protótipo foi HC-SR04 (Figura 4.8). Para testar o sensor, este foi conectado a placa Arduino, os pinos do Arduino “A4” e “A5”, foram conectados respectivamente aos pinos “Trig” e “Echo” do sensor ultrassônico, os pinos GND e Vcc do sensor foram conectados aos pinos GND e Vcc do Arduino. Os pinos “GND” e “Vcc”, são pinos de terra e alimentação respectivamente (o Arduino fornece uma alimentação de 5V), os pinos “A4” e “A5” são pinos de entrada ou saída analógicas (depende da programação) (Figura 4.8).

Figura 4.8 - Sensor HC-SR04 real e o diagrama de pinagem utilizada.



Fonte: Adaptado pelo Autor.

O sensor HC-SR04 possui limites de alcance, uma distância de no mínimo 2 cm e no máximo 4 m. Considerando as dimensões do protótipo robótico, e em relação ao alcance do sensor ultrassônico, para uma melhor visualização de medição, a unidade escolhida para as medições foram centímetros.

O som viaja a velocidade aproximada de 340 m/s, com variações dependendo da temperatura (GOUVEIA, 2018), convertendo esta velocidade para centímetros por microssegundos (por causa da unidade de armazenamento do Arduino), resulta em 0,034 cm/μs.

$$340 \frac{\text{m}}{\text{s}} = \frac{34000 \text{ cm}}{10^6 \mu\text{s}} = 0,034 \frac{\text{cm}}{\mu\text{s}} \quad (4.1)$$

Em um microssegundo o som percorre 0,034 centímetros. Para saber quanto tempo leva para percorrer um centímetro:

$$\frac{1 \mu\text{s}}{0,034 \text{ cm}} = 29,4117647059 \frac{\mu\text{s}}{\text{cm}} \quad (4.2)$$

O sensor capta a reflexão do pulso, sendo o tempo de reflexão o tempo de ida do pulso somado com o tempo de retorno do pulso.

$$\text{Tempo de reflexão} = \text{Tempo de ida do pulso} + \text{Tempo de retorno do pulso} \quad (4.3)$$

O tempo de reflexão necessário para o sensor ultrassônico captar o pulso é o dobro do tempo que leva para um pulso alcançar um obstáculo, considerando que não irá ocorrer variação na velocidade do som durante sua reflexão. O tempo que leva para a onda refletir:

$$\begin{aligned} \text{Tempo de ida do pulso} &= \text{Tempo de retorno do pulso} \\ \text{Tempo de reflexão} &= 2 \times \text{Tempo de ida do pulso} \end{aligned} \quad (4.4)$$

O tempo calculado para um pulso ultrassônico percorrer um centímetro de distância é 29,4117647059 μs. Deste modo, leva 58.8235294118 μs a reflexão do pulso para obter 1 cm de distância.

$$\text{Tempo de reflexão (1 cm)} = 2 \times \text{Tempo de ida do pulso (1 cm)}$$

$$\begin{aligned} \text{Tempo de ida do pulso (1 cm)} &= 29,4117647059 \mu\text{s} \\ \text{Tempo reflexão (1 cm)} &= 2 \times 29,4117647059 \mu\text{s} \end{aligned} \quad (4.5)$$

$$\text{Tempo reflexão (1 cm)} = 58,8235294118 \mu\text{s}$$

O teste do ultrassom utilizou o código baseado em um projeto de automação feito por Rambo(2016). A configuração das variáveis e pinagens foram feitas, o pino de “trigger” foi definido como “A4” e pino “echo” como “A5”. O código de medição foi feito dentro da função denominada “measureDistance()”, esta função foi declarada dentro do “loop()” para ser constantemente chamada. Nesta função (“measureDistance()”) ocorre o cálculo de medição do sensor em centímetros, o valor desta medição foi guardada em uma variável denominada “dist_cm”, por estar em loop, o ultrassom se mantém em execução continuamente (Quadro 4.3).

Quadro 4.3 - Código de medição utilizando o sensor de ultrassom.

```
#define trig    A4           //Saída para o pino de trigger do sensor
#define echo    A5           //Saída para o pino echo do sensor
float dist_cm;
float measureDistance();     //Função para medir, calcular e retornar a distância em cm
void trigPulse();            //Função que gera o pulso de trigger de 10µs
void setup() {
    pinMode(trig, OUTPUT);    //Saída para o pulso de trigger
    pinMode(echo, INPUT);     //Entrada para o pulso de echo
    digitalWrite(trig, LOW);  //Pino de trigger inicia em low
    delay(500);
    Serial.begin(9600);
}
void loop() {
    dist_cm=measureDistance();
    Serial.println(dist_cm);
}
```

Fonte: Autor.

A função “void trigPulse()” faz o sensor ultrassônico emitir ondas sonoras por 10µs, estes são pulsos ultrassônicos (Quadro 4.4). O “digitalWrite(trig,HIGH)” deixa o pino de saída “trigger” no nível alto emitindo a onda sonora, durante um *delay* de 10 µs pelo “delayMicroseconds(10)”, após este tempo a saída se torna nível baixo, substituindo o “HIGH” por “LOW”, parando a emissão (REIS, 2018).

Quadro 4.4 - Código da função trigPulse().

```
void trigPulse()             //Função para gerar o pulso de trigger para o sensor HC-SR04
{
    digitalWrite(trig,HIGH);  //Saída de trigger em nível alto
    delayMicroseconds(10);    //Por 10µs
    digitalWrite(trig,LOW);   //Saída de trigger volta a nível baixo
} //end trigPulse
```

Fonte: Autor.

O “pulseIn(echo, HIGH)” permite obter o tempo de reflexão de um pulso ultrassônico quando o receptor está em nível lógico alto em µs (ARDUINO, 2019). Esse tempo foi armazenado em uma variável chamada “pulse” (Quadro 4.5).

Quadro 4.5 - Código exemplo da função pulseIn (echo, High).

```
float measureDistance()           //Função que retorna a distância em centímetros
{
  float pulse; //Variável que armazena o valor de tempo em μs
  float H;      //Limitador do ultrassom;
  trigPulse();  //Envia pulso de 10μs para o pino de trigger do sensor
  pulse = pulseIn(echo, HIGH); //Mede o tempo em que echo fica em nível alto e armazena
```

Fonte: Autor.

Para obter a distância a um obstáculo em centímetros, bastou dividir o tempo de reflexão neste objeto pelo tempo de reflexão necessário para obtenção de um centímetro, como indicado abaixo.

$$\text{Distância a um obstáculo} = \frac{\text{Tempo de reflexão}}{\text{Tempo de reflexão (1 cm)}} \quad (4.6)$$

$$\text{Distância a um obstáculo} = \frac{\text{Tempo de reflexão}}{58,8235294118 \mu s} \quad (4.7)$$

Os cálculos foram feitos no código e a distância foi armazenada em uma variável “H”. Foi atribuído um valor limite de distância de 70 cm mesmo com o ultrassom captando distâncias maiores (Quadro 4.6).

Quadro 4.6 - Código do valor limite de distância.

```
H= pulse/58.82; //Tempo que leva para a onda sonora ir e voltar
if(H>=70)       //Limitado um valor máximo de 70cm;
H=70;
return (H);     //Retorna o valor
} //end measureDistante
```

Fonte: Autor.

Esse valor foi escolhido para que durante a exibição no mapeamento o alcance máximo do sensor seja pouco menor que cinco vezes o comprimento do robô (aproximadamente 15 cm). Outro motivo para limitar o valor de medição, é a limitação do número de dígitos durante a comunicação serial entre robô e PC, quanto mais dígitos mais retarda a comunicação.

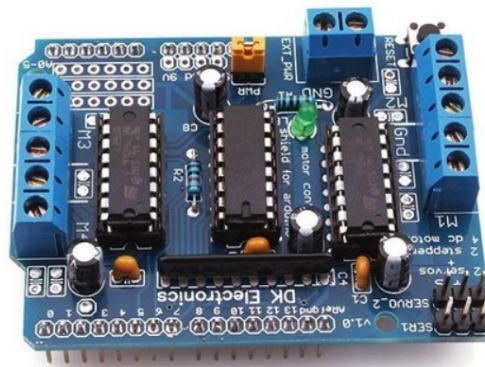
Durante a execução, pulsos constantemente são captados pelo sensor, retornando as distâncias no “Monitor Serial”. Para conferir se a distância foi medida corretamente, foi comparada a distância medida pelo sensor com a distância medida por uma régua. Ambas as medições foram aproximadamente 20 cm, o ultrassom funcionou corretamente.

4.2.3 Motores

Para o controle dos motores, uma placa com circuito ponte H foi necessária. O circuito ponte H serve para controlar o sentido de entrada da corrente DC no motor, assim controlando o sentido de sua rotação.

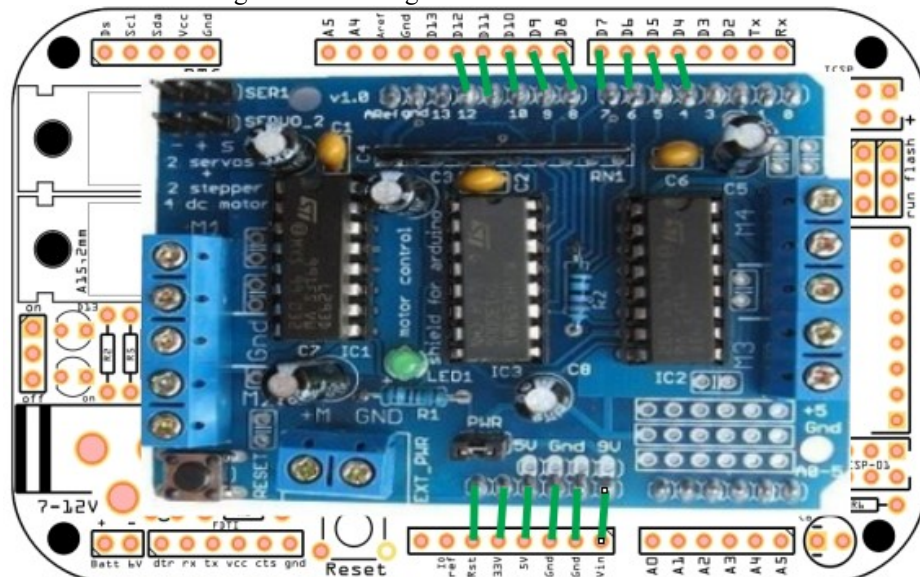
No protótipo foi usada a placa *Motor Shield L293D Driver Ponte H* para Arduino (Figura 4.9), essa placa possui dois chips de circuito integrado L293D, cada L293D possui duas pontes H internamente e suporta uma corrente de saída de 600 mA, permitindo a essa placa uma conexão com até quatro motores DC com até 600mA cada motor. Além de permitir conexão até quatro motores DC, essa placa possui duas saídas para motor de passo, duas saídas para servo motor e uma entrada para alimentação externa de até 16 V, embora possa ser alimentada internamente com a conexão de um *jumper*.

Figura 4.9 - Motor *Shield* L293D Driver Ponte H.



Fonte: Adaptado pelo Autor.

Figura 4.10 - Pinagem do Motor *Shield* L293D.



Fonte: Autor.

Essa placa *Driver* foi conectada a placa Arduino. Os pinos digitais do Arduino “D4”, “D5”, “D6”, “D7”, “D8”, “D11”, “D12” e os pinos “Vin”, “Gnd”, “5v”, “3.3v”, “Rst”, foram conectados aos pinos correspondentes de mesmo nome da placa *Driver* (Figura 4.10). Os pinos digitais “D3”, “D5”, “D6” e “D11”, são pinos para PWM (*Pulse Width Modulation*), que podem controlar a velocidade de rotação do motor dependendo da tensão aplicada. Os pinos “D3”, “D5”, “D6” e “D11”, correspondem aos motores “2”, “3”, “4” e “1”, respectivamente. Os pinos “D4”, “D7”, “D8”, “D12” são responsáveis para acionamento de motores DC e motores de passo.

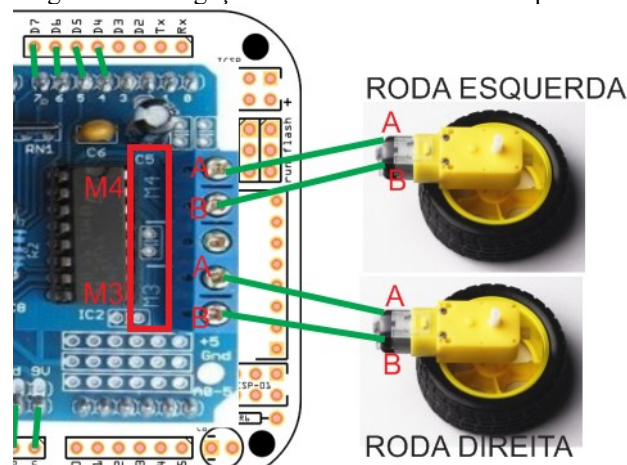
O protótipo robótico possui dois motores DC. Em cada motor possui um par de conexões (que foram denominadas como A e B), nessas conexões foram soldados *jumper*s (Figura 4.11), estes que se conectam aos respectivos pares de canais, “MX” na placa (X corresponde à numeração de motores de 1 a 4). Durante a montagem física, foram escolhidos os motores “3” e “4”. O par de canais “M3” foi conectado ao motor direito (roda direita), e o par de canais “M4” foi conectado ao motor esquerdo (roda esquerda) (Figura 4.12). A referência de “esquerda e direita” é feita observando o robô por trás, inverso da visão de frente.

Figura 4.11 - Solda das conexões no motor.



Fonte: Autor.

Figura 4.12 - Ligação dos motores no *shield* da ponte H.



Fonte: Autor.

Foram feitos testes de acionamento dos motores e análises do sentido das rotações, para testar corretamente as conexões A e B dos motores. O teste foi feito para manter a rotação das rodas conforme o tipo de movimento desejado. O código de teste foi baseado no projeto de automação de Rambo(2016).

O código usa a biblioteca “AFMotor.h”, os motores DC foram declarados como “motor1” e “motor2”, motor esquerdo e direito respectivamente. Para selecionar os motores foi usado a função “AF_DCMotor” Para o “AF_DCMotor motor1()”, entre parentes foi colocado a numeração do canal conectado ao motor, analogamente foi usado “AF_DCMotor motor2()”, para o motor direito.

Quadro 4.7 - Código referente ao motores.

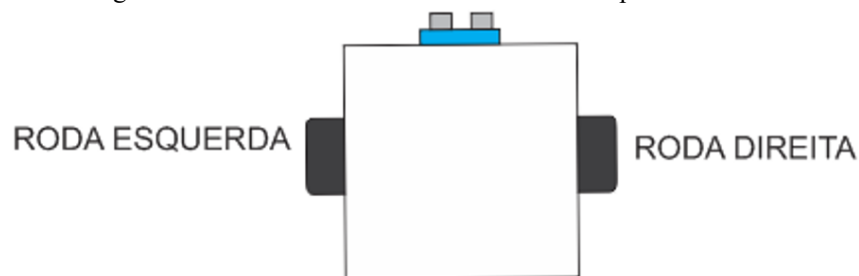
```
#include <AFMotor.h>
AF_DCMotor motor1(4);//Seleção do Motor 1
AF_DCMotor motor2(3);//Seleção do Motor 2
void setup() {
}
```

Fonte: Autor.

4.2.4 Interação entre motores e a estrutura

O robô possui duas rodas, uma situada no lado esquerdo e outra situada no lado direito (Figura 4.13), cada motor DC é responsável por girar uma roda diferente. Através do código usado para girar a roda, foi verificado o sentido de rotação da roda. A função “setSpeed()” define a velocidade do motor, sendo 0 o valor mínimo e 255 o valor máximo.

Figura 4.13 - Posicionamento das rodas e vista superior do robô.



Fonte: Autor.

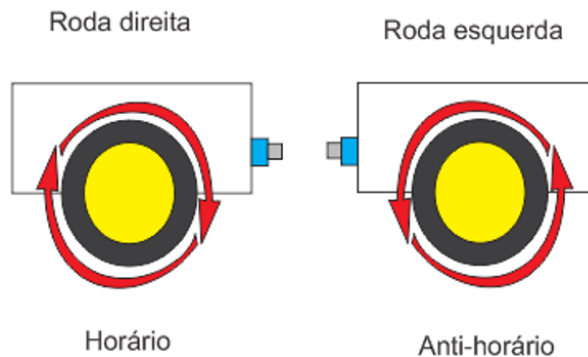
Para a roda se mover, foi usado a função “run()”, entre os parênteses foi colocado o parâmetro desejado para o motor. Os parâmetros são: “FORWARD”, para a roda girar para “frente”, “BACKWARD” para a roda girar para “trás” (a direção de rotação depende da fiação) e “RELEASE”, para a roda parar de girar (LEARN.ADAFRUIT, 2019).

Para a roda direita se mover para “frente” conforme a figura 4.15, esta deve rotacionar em sentido horário (Figura 4.14). Foi usado “motor2.run(FORWARD)” (Quadro 4.8) e observado se realmente a roda girava em sentido horário conforme a ligação dos fios A e B

do motor direito conectado ao par de canais do “M3”, caso girasse anti-horário, as conexões dos fios no “M3” seriam invertidos para corrigir o sentido de rotação conforme a programação.

Para a roda esquerda se mover para “frente” conforme a figura 4.15, esta deve rotacionar em sentido anti-horário (Figura 4.14). Foi usado o código “motor1.run(FORWARD)”, e verificado o sentido de rotação da roda esquerda (Quadro 4.8). Rotacionando de modo anti-horário, a conexão A e B no par “M4” está correta, caso contrário as conexões A e B seriam invertidas. O teste de rotação verificou a conexão correta dos motores.

Figura 4.14 - Sentido de rotação dos motores.



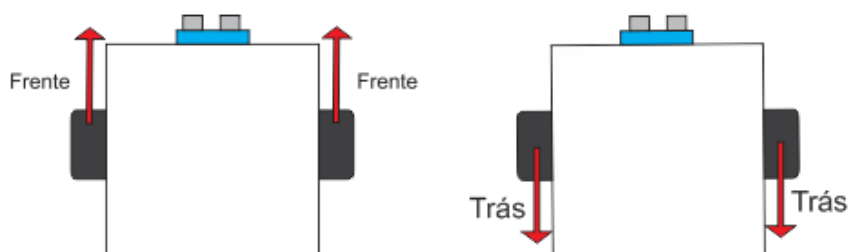
Fonte: Autor.

Quadro 4.8 - Código de teste dos motores.

```
void loop() {
  motor2.setSpeed(255);
  motor2.run(FORWARD);
  motor1.setSpeed(255);
  motor1.run(FORWARD);
}
```

Fonte: Autor.

Figura 4.15 – Projeção das rodas no robô.



Fonte: Autor.

De forma análoga, substituindo o “FORWARD” nas funções “run()” por “BACKWARD” foi verificado o sentido de rotação inverso das rodas, deste modo as rodas se

moveram para “trás” (Figura 4.15), e substituindo por “RELEASE” as rodas pararam de se mover. Com as combinações de rotação das rodas foi possível criar os movimentos do robô.

4.2.5 Locomoção do robô

Para o robô se mover, foram criados cinco tipos de movimento: se mover para frente normalmente, se mover para trás em ré, girar para direita em torno do próprio eixo, girar para esquerda em torno do próprio eixo e parar. As combinações de rotações das rodas permitiram criar estes movimentos.

Estes movimentos foram criados em funções diferentes com o propósito de cada uma ser usada em situações específicas. Por exemplo, para testar cada uma das funções bastou retirar o “//” antes da função de movimento desejado (Quadro 4.9). Funções que foram criadas de acordo com o código de rotação das rodas.

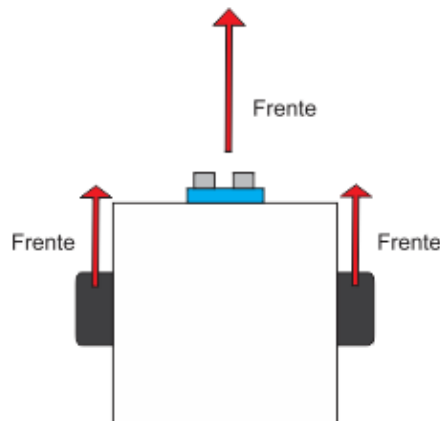
Quadro 4.9 - Código de movimentos do robô.

```
void robot_forward(); //Função para movimentar robô para frente
void robot_backward(); //Função para movimentar robô para trás
void robot_left(); //Função para movimentar robô para esquerda
void robot_right(); //Função para movimentar robô para direita
void robot_stop(); //Função para parar o robô
void setup() {
}
void loop() {
  //robot_forward(); //Usado para teste
  //robot_left(); //Usado para teste
  //robot_backward(); //Usado para teste
  //robot_right(); //Usado para teste
  //robot_stop(); //Usado para teste
}
```

Fonte: Autor.

O movimento do robô de ir para frente foi possível pela combinação de ambas as rodas se moverem para frente (Figura 4.16). Para este movimento foi usado o código do Quadro 4.10, criado na função denominada “robot_forward()”.

Figura 4.16 - Locomoção do robô devido a ação das rodas.



Fonte: Autor.

Quadro 4.10 - Código para movimentação do robô.

```
void robot_forward()
{
    motor1.setSpeed(255);
    motor1.run(FORWARD);
    motor2.setSpeed(255);
    motor2.run(FORWARD);
} //end robot forward
```

Fonte: Autor.

Foi notado durante o teste para robô se mover para frente, ao usar a mesma velocidade para ambos os motores, o robô se deslocava inclinado para direita, deste modo foi observado que o motor usado para a roda esquerda era mais potente que o motor da direita. Portanto, foi necessário atribuir um valor menor de velocidade para o motor da esquerda do que o da direita para calibrar os motores. Em um caso hipotético que ambos os motores funcionam de mesma forma com a mesma potência, os valores atribuídos de velocidade seriam iguais.

Valores de diferentes velocidades foram atribuídos e testados observando a inclinação do robô ao se mover para frente, sempre mantendo a velocidade do “motor1” maior que do “motor2”. Também, para evitar deslizamentos das rodas no solo, consequência da inércia do robô em uma velocidade alta, e para que também não seja um deslocamento lento, foram escolhidas as velocidades “205” e “170” para os motores. O valor de diferença “35” entre as velocidades foi suficiente para manter uma calibração aproximada. A função de movimento para frente foi modificada (Quadro 4.11).

Quadro 4.11 - Balanceamento da velocidade do robô.

```
void robot_forward()
{
    motor1.setSpeed(170);
    motor1.run(FORWARD);
    motor2.setSpeed(205);
    motor2.run(FORWARD);
} //end robot forward
```

Fonte: Autor.

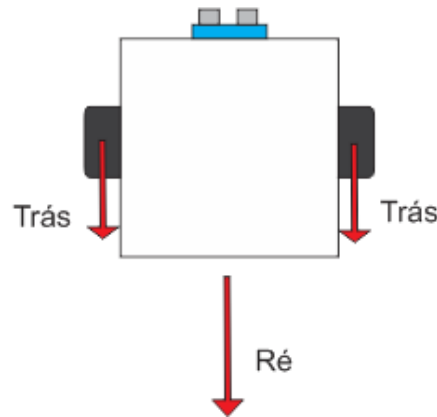
O movimento de ir para trás em ré foi possível pela combinação de ambas as rodas se moverem para trás (Figura 4.17). Para este movimento foi usado o código do Quadro 4.12, criado na função denominada “robot_backward()”.

Quadro 4.12 - Código com o movimento balanceado para trás.

```
void robot_backward()
{
    motor1.setSpeed(170);
    motor1.run(BACKWARD);
    motor2.setSpeed(205);
    motor2.run(BACKWARD);
} //end robot backward
```

Fonte: Autor.

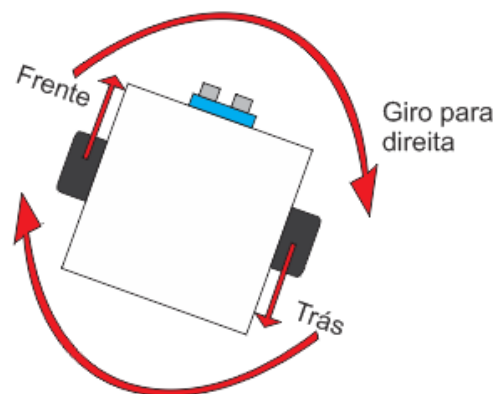
Figura 4.17 - Deslocamento do robô para trás.



Fonte: Autor.

O movimento de girar para a direita em torno do próprio eixo foi possível pela combinação do movimento da roda esquerda ir para frente e da roda direita ir para trás (Figura 4.18). Para este movimento foi usado o código do Quadro 4.13, criado na função denominada “robot_right()”.

Figura 4.18 - Ação dos motores para giro do robô para direita.



Fonte: Autor.

Para os movimentos de giro em torno do próprio do próprio eixo, foram escolhidas velocidades “120” e “155”, a fim de manter um giro suave para proporcionar maior controle na direção do robô.

Quadro 4.13 - Código para rotação do robô para direita.

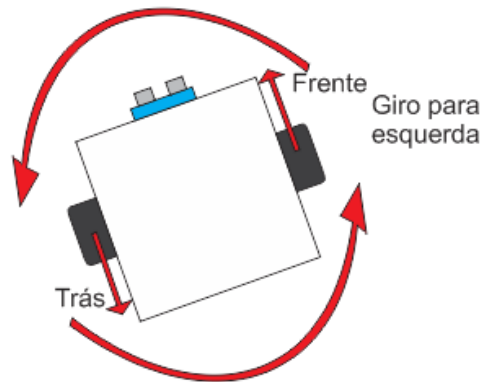
```
void robot_right()
{
  motor1.setSpeed(120);
  motor1.run(FORWARD);
  motor2.setSpeed(155);
  motor2.run(BACKWARD);
} //end robot right
```

Fonte: Autor.

O movimento de girar para a esquerda em torno do próprio eixo foi possível pela combinação do movimento da roda direita ir para frente e da roda esquerda ir para trás (Figura

4.19). Para este movimento foi usado o código do Quadro 4.14, criado na função denominada “robot_left()”.

Figura 4.19 - Ação dos motores para giro do robô para esquerda



Fonte: Autor.

Quadro 4.14 - Código para rotação do robô para esquerda.

```
void robot_left()
{
  motor1.setSpeed(120);
  motor1.run(BACKWARD);
  motor2.setSpeed(155);
  motor2.run(FORWARD);
} //end robot left
```

Fonte: Autor.

Para o robô parar de se mover foi usado “run(RELEASE)” em ambos os motores em na função denominada “robot_stop()” (Quadro 4.15).

Quadro 4.15 - Código para o robô parar.

```
void robot_stop()
{
  motor1.setSpeed(0);
  motor1.run(RELEASE);
  motor2.setSpeed(0);
  motor2.run(RELEASE);
} //end robot stop
```

Fonte: Autor.

4.2.6 Interface gráfica no *Processing*

Para monitorar a posição do robô e mapear o local, foi feita a construção de um mapa visual, exibindo no computador em tempo real o deslocamento do robô, sua posição atual, distâncias do robô a obstáculos, áreas livres ou com obstáculos e áreas que não foram exploradas.

Para criar a interface gráfica de mapeamento foi necessário utilizar um software para esta função. O *Processing* foi uma das principais ferramentas usadas neste trabalho, assim como Arduino IDE, este software é usado para programar, com o foco voltado para criar interfaces gráficas (REAS e FRY, 2010).

A linguagem para programar no *Processing* é baseada em Java. O software consegue se comunicar via serial com a placa de Arduino (HAMZA, 2018), permitindo assim, a execução de projetos que envolvam a interação da placa, seus módulos e sensores, com as interfaces gráficas criadas no próprio *Processing*, característica importante para este trabalho.

O primeiro teste para comunicar o *Processing* com a placa de Arduino foi feito usando um potenciômetro e um LED. A implementação do código de teste foi feita nos softwares *Processing* e Arduino IDE. O teste foi feito usando um tutorial criado por Hanza (2018).

Para testar o envio de dados do Arduino ao *Processing*, à medida que o potenciômetro conectado a placa foi girado, a interface gráfica gerada no *Processing* alterou suas cores. Na mesma interface gráfica, para testar o envio de dados do *Processing* ao Arduino, quando a interface foi clicada com o botão esquerdo do mouse o LED acendeu, quando foi clicada com o botão direito do mouse o LED apagou.

Para acontecer a comunicação entre *Processing* e Arduino foi necessário importar a biblioteca serial no *Processing* e inicializar uma variável para a comunicação serial. No teste, a variável serial foi denominada “myPort” (Quadro 4.16).

Quadro 4.16 - Código para importar código serial no *Processing*.

```
import processing.serial.*;
Serial myPort;
```

Fonte: Autor.

Foi declarada a porta do PC usada para comunicação serial, e a taxa de transmissão. A porta usada para a comunicação no caso foi “COM5”, mas esta é substituída de acordo com a porta usada pelo Arduino (Quadro 4.17).

Para a recepção de dados do Arduino, foi usado o “bufferUntil()”, este guarda os dados recebidos até um parâmetro colocado entre os parênteses, em um buffer serial (PROCESSING, 2019). No teste, o “myPort.bufferUntil('\n')” faz o “myPort” receber os dados enviados pelo Arduino até o final da linha de dígitos . Se o “Monitor Serial” do Arduino estiver ativo utilizando a mesma porta “COM” enquanto o *Processing* tenta se comunicar com o Arduino, irá ocorrer falha na comunicação.

Quadro 4.17 - Parâmetros iniciais de teste da serial.

```
myPort = new Serial(this, "COM5", 9600);
myPort.bufferUntil ( '\n' );
```

Fonte: Autor.

Dentro da função “serialEvent()”(Quadro 4.18), função chamada quando dados seriais estão disponíveis (PROCESSING, 2019), os dados de “myPort” foram lidos e movidos para uma *string*, que depois foram convertidos em números racionais e guardados na variável “background_color”. Esta variável foi utilizada para a coloração de uma matriz de pixels.

Através do “`readStringUntil()`” é possível ler os dados guardados no buffer serial até o caractere dentro do parênteses e converter em *strings* (PROCESSING, 2019). O “`float()`” converte um número inteiro ou uma *string* em números racionais.

Quadro 4.18 - Leitura de *string* pela serial.

```
void serialEvent (Serial myPort) {
  background_color = float (myPort.readStringUntil ( '\n' ) ) ;
}
```

Fonte: Autor.

A função “`draw()`” é uma função que é chamada repetidamente e automaticamente, por este modo esta função foi utilizada no projeto do protótipo robótico para executar uma série de comandos . Neste teste, dentro da função “`draw()`” foram utilizados comandos para alterar as cores da matriz de pixels e enviar dados de acordo com o estado de clique do mouse.

A condição “`mousePressed`” reconhece quando o botão do mouse é pressionado, e a condição “`mouseButton == LEFT`” ou “`mouseButton == RIGHT`” identificam qual botão do mouse foi pressionado, botão esquerdo “`LEFT`” ou o botão direito “`RIGTH`”. Condições usadas para definir os estados de clique do mouse.

Para enviar dados na porta serial foram usados nos estados de clique o comando “`write()`”. No código (Quadro 4.19), “`myPort.write()`” escreve na porta serial “`myPort`” o dado entre parênteses, sendo ‘1’ quando o botão esquerdo do mouse é clicado e ‘0’ quando o botão direito do mouse é clicado.

Quadro 4.19 - Código para os cliques do mouse no *Processing*.

```
void draw ( ) {
  background ( 150, 50, background_color );
  if ( mousePressed && ( mouseButton == LEFT ) ) { // se botão esquerdo for pressionado
    myPort.write ( '1' ); // envia '1'
  }
  if ( mousePressed && ( mouseButton == RIGHT ) ) { // se botão direito for pressionado
    myPort.write ( '0' ); // envia '0'
  }
}
```

Fonte: Autor.

No Arduino IDE, para inicializar uma comunicação serial com o *Processing*, dentro da função “`setup()`” a porta serial deve ser inicializada com uma taxa de transmissão igual a taxa usada no *Processing*: “`Serial.begin(9600)`” (Quadro 4.20).

Quadro 4.20 - Inicialização da comunicação serial com o *Processing*.

```
void setup ( ) {
  pinMode(led_pin, OUTPUT);
  Serial.begin(9600);
}
```

Fonte: Autor.

Para o Arduino receber os dados do *Processing* foi usado a condição “`if (Serial.available ( ) > 0)`” dentro da função `loop()`. Esta condição verifica no Arduino se

possui algum valor enviado do *Processing* para a porta serial Para o Arduino enviar os dados ao *Processing*, ele envia através do “Serial.println()” ou “Serial.print()”.

Ao receber os dados do *Processing* o valor é lido e guardado em uma variável chamada de “state” que acessa as condições de apagar ou acender o LED de acordo com seu valor. Apaga quando recebe “0”, através do “digitalWrite (led_pin, LOW)”, e o acende quando recebe “1” através do “digitalWrite (led_pin, HIGH)” (Quadro 4.21).

Quadro 4.21 – Teste de funcionamento da serial com acender e apagar do LED.

```
if (Serial.available () > 0) { //Verificação
char state = Serial.read (); // lendo dados
if(state == '1')          // se for 1, acende led
{
digitalWrite (led_pin, HIGH);
}
if (state == '0') {      // se for zero, apaga led
digitalWrite (led_pin, LOW);
}
}
```

Fonte: Autor.

4.2.7 Módulo *Bluetooth*

A comunicação serial por *Bluetooth* pode substituir a comunicação serial via cabo, porém para a gravação de código na placa Arduino é necessário o cabo de dados. Foi testado utilizando o “código do potenciômetro” anterior a comunicação *Bluetooth* entre a placa Arduino e PC.

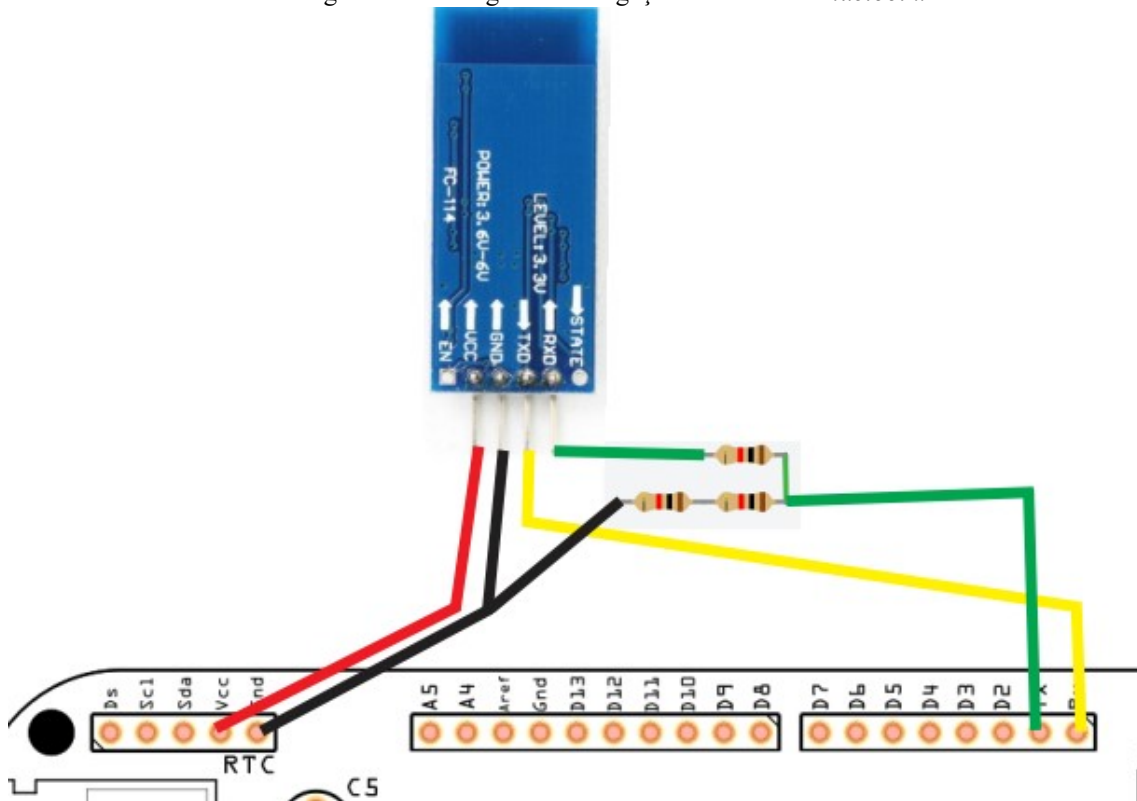
O módulo de transmissão *Bluetooth* usado foi o HC06 (Figura 4.20), as portas “TX” e “RX” do módulo foram conectadas as portas “RX” e “TX” da placa Arduino respectivamente. No pino “RX” do módulo foi necessário um fazer um divisor de tensão para converter os 5 V de entrada em 3,3 V. A placa Arduino foi alimentada por uma bateria de 9 V (Figura 4.21).

Figura 4.20 – Módulo *Bluetooth* HC06.



Fonte: Adaptado pelo Autor.

Figura 4.21 - Diagrama das ligações do módulo *Bluetooth*.



Fonte: Autor.

Foi ativada no PC a comunicação por *Bluetooth*, criando uma porta COM Serial *Bluetooth*. O computador usado não possui transmissor *Bluetooth*, deste modo foi necessário usar um adaptador *Bluetooth* USB. O adaptador usado foi o *Bluetooth* USB Dongle. Este adaptador usa tecnologia *Bluetooth* 2.0 e possui alcance de 100 metros.

A porta COM Serial *Bluetooth* foi usada substituindo a porta “COM” no código do *Processing*. O teste permitiu variar as cores da interface pelo potenciômetro e ligar/desligar o LED à distância. Ao término do teste, para voltar a gravar códigos na placa, o módulo *Bluetooth* foi desconectado, pois interfere na gravação.

4.3 PROGRAMAÇÃO DO MICROCONTROLADOR

Nos itens anteriores foram discutidos os componentes individualmente e os respectivos testes e ligações. Neste item serão discutidos os códigos elaborados para o microcontrolador com objetivo de operar de forma unificada, ou seja, são os códigos que definem os comportamentos do robô e a interação entre os componentes.

4.3.1 Implementação do controle manual

O controle manual do robô foi feito para que o robô se movimentasse de acordo com teclas específicas pressionadas no PC. O controle foi criado na *Processing*, quando o código está em execução, ao apertar as teclas de seta: “cima”, o robô anda para frente, “baixo”, o robô anda para trás em ré, “direita”, gira o robô para direita em torno do próprio eixo, “esquerda”, o gira para esquerda em torno do próprio eixo. Pressionando a tecla “Alt”, o robô pára o movimento.

Foi usada a função “keyPressed()”, função chamada sempre que alguma tecla é pressionada. Nesta função as condições de estado de tecla foram determinadas (Quadro 4.22). Na condição “if(keyCode ==UP){ myPort.write ('0') ;”, se a seta do teclado “cima” for pressionada, escreve na porta serial “myPort” o dado ‘0’. Na condição “if(keyCode ==DOWN){ myPort.write ('1') ;”, se a seta do teclado “baixo” for pressionada, escreve na porta serial “myPort” o dado ‘1’. Na condição “if(keyCode ==LEFT){ myPort.write ('2') ;”, se a seta do teclado “esquerda” for pressionada, escreve na porta serial “myPort” o dado ‘2’. Na condição “if(keyCode ==RIGHT){ myPort.write ('3') ;”, se a seta do teclado “direita” for pressionada, escreve na porta serial “myPort” o dado ‘3’. Na condição “if(keyCode ==ALT){ myPort.write ('4') ;”, se a tecla “Alt” for pressionada, escreve na porta serial “myPort” o dado ‘4’.

Quadro 4.22 - Código para as condições de estado das teclas.

```
void keyPressed(){
  if(key==CODED){
    if(keyCode ==UP){ myPort.write ( '0' ) ;
    }
    if(keyCode==DOWN){myPort.write ( '1' ) ;
    }
    if(keyCode==LEFT){myPort.write ( '2' ) ;
    }
    if(keyCode==RIGHT){myPort.write ( '3' ) ;
    }
    if(keyCode==ALT){myPort.write ( '4' ) ;
    }
  }
}
```

Fonte: Autor.

No Arduino, os números de 0 a 4 correspondentes as teclas pressionadas, enviados pelo *Processing*, foram lidos e guardados na variável “state”. De acordo com o valor da variável “state” o robô executa diferentes funções que foram agregadas a um valor específico desta. Se “state” valer 0, o robô executa a função dos motores que faz o robô se mover para a frente. Se “state” valer 1 o robô irá para trás, se for 2 o robô irá para a esquerda, se for 3 o

robô gira para direita e se for 4 o robô executa a função dos motores que faz o robô parar (Quadro 4.23).

Quadro 4.23 - Código para cada movimento do robô.

```
void loop() {
  if (Serial.available () > 0) { //
    char state = Serial.read (); //
    if (state == '0') {
      robot_forward(); }
    if (state == '1') {
      robot_backward(); }
    if (state == '2') {
      robot_left(); }
    if (state == '3') {
      robot_right(); }
    if (state == '4') {
      robot_stop(); }
  }
}
```

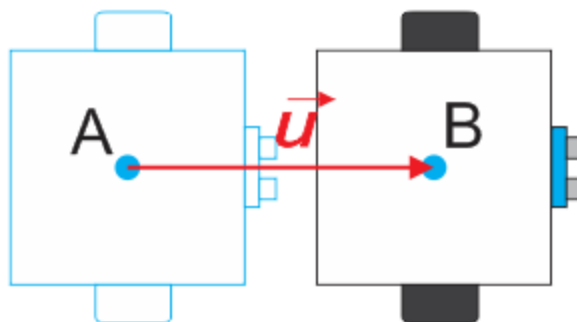
Fonte: Autor.

4.3.2 Lógica de deslocamento no mapa

O deslocamento do robô segue uma lógica vetorial que depende do módulo, e orientação. Um vetor pode ser representado pela mudança de coordenada de posição, e ao indicar um deslocamento, é denominado como vetor de deslocamento. Uma partícula quando se move um ponto A ao um ponto B, sofre um deslocamento de A para B, que pode ser representada por uma seta apontando de A para B (HALLIDAY, 2012).

O robô ao se deslocar durante um único pulso, ele se desloca de um ponto A (ponto anterior) ao ponto B (ponto atual), a distância de um único pulso foi representada como uma constante de valor unitário. O vetor deste deslocamento foi representado como $\vec{u}$ (Figura 4.22).

Figura 4.22 - Vetor deslocamento do robô.



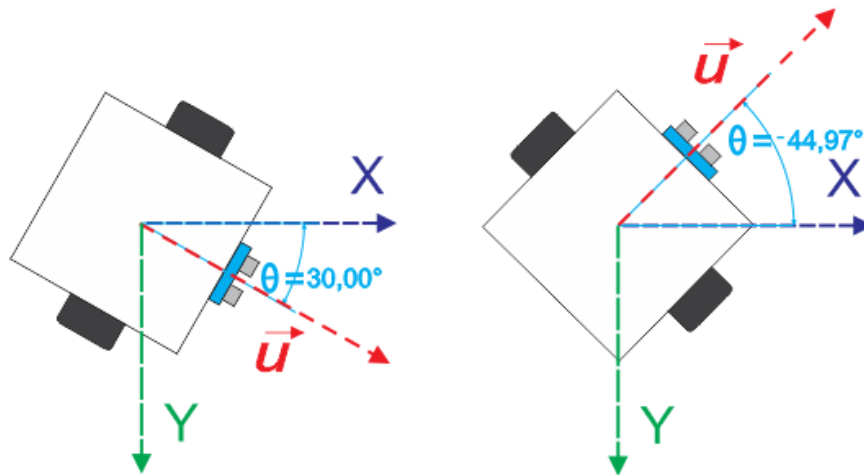
Fonte: Autor.

A posição do robô usa um sistema de coordenadas X, Y, este depende da orientação do vetor $\vec{u}$. O robô quando está se movendo em linha reta para frente, ele é representado como um vetor de sentido da traseira até a frente, formando um vetor conforme a Figura 4.22 Quando o

robô está se movendo em linha reta para trás, ele é representado como um vetor de sentido frente até traseira formando um vetor de sentido oposto.

O robô para determinar sua direção, ele gira em torno do próprio eixo de girando para direita ou girando para esquerda formando um ângulo θ em relação da projeção de um vetor $\vec{u}$ para frente e um semieixo X positivo. O ângulo cresce no sentido horário e decresce no anti-horário (Figura 4.23).

Figura 4.23 - Projeção de $\vec{u}$ e sua relação com o ângulo.

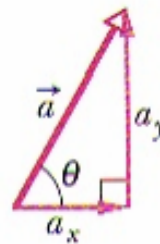


Fonte: Autor.

No eixo Y, o crescimento é positivo para baixo (devido à interface usada para o mapeamento), por este motivo o ângulo cresce no sentido horário e decresce no sentido anti-horário.

Segundo Halliday (2012), para calcular a componente X de um vetor, basta multiplicar o módulo do vetor pelo cosseno do ângulo formado entre o vetor e o semieixo X positivo, e para calcular a componente Y, basta multiplicar o módulo do vetor pelo seno do ângulo (Figura 4.24). Podem existir componentes negativas se o vetor aponta para o eixo negativo.

Figura 4.24 - Projeções em X e Y de um vetor $\vec{a}$.



$$a_x = a \cos \theta \quad \text{e} \quad a_y = a \sin \theta,$$

Fonte: Halliday (2012)

As componentes X e Y são projeções do vetor $\vec{u}$. Como o módulo do vetor $\vec{u}$ é unitário (possui valor igual a um), as componentes serão $\cos(\theta)$ e $\sin(\theta)$.

$$\begin{aligned} u_x &= ||u|| \cos(\theta) \\ u_x &= \cos(\theta) \\ u_y &= ||u|| \sin(\theta) \\ u_y &= \sin(\theta) \end{aligned} \tag{4.8}$$

Enquanto o robô se move de uma posição para outra, uma quantidade de vetores $\vec{u}$ é formada conforme a quantidade de pulsos gerados. O deslocamento do robô é o resultado da soma geométrica desses vetores e a posição do robô é a soma das componentes X e Y destes vetores. Com o vetor $\vec{u} = u_x \hat{i} + u_y \hat{j}$, reescrevendo as variáveis “x” e “y” como componentes do vetor $\vec{u}$: $u_x = x$, $u_y = y$, fica $\vec{u}_n = x_n \hat{i} + y_n \hat{j}$. Sendo $\vec{S}_n$ o vetor resultante da soma geométrica dos vetores $\vec{u}$ a uma quantidade n de pulsos já captados: $\vec{S}_n = \sum_1^n \vec{u}_n$. Através da combinação das componentes eixo por eixo é possível obter a soma de vetores (Halliday, 2012). Segue:

$$\begin{aligned} S_{x_n} &= \sum_1^n x_n \\ S_{y_n} &= \sum_1^n y_n \\ \vec{S}_n &= S_{x_n} \hat{i} + S_{y_n} \hat{j} \end{aligned} \tag{4.9}$$

Na lógica feita para a programação do robô, as posições X e Y (x e y maiúsculos) que o robô se encontra são feitas pela soma das componentes dos vetores $\vec{u}$:

$$\begin{aligned} X &= S_{x_n} \\ Y &= S_{y_n} \end{aligned} \tag{4.10}$$

A cada novo pulso, vai acrescentando no somatório uma nova componente x_n e y_n , fazendo o resultado do somatório anterior (antes do novo pulso) ser o valor de $S_{x_{n-1}}$ e $S_{y_{n-1}}$.

$$\begin{aligned} X &= S_{x_{n-1}} + x_n \\ Y &= S_{y_{n-1}} + y_n \end{aligned} \tag{4.11}$$

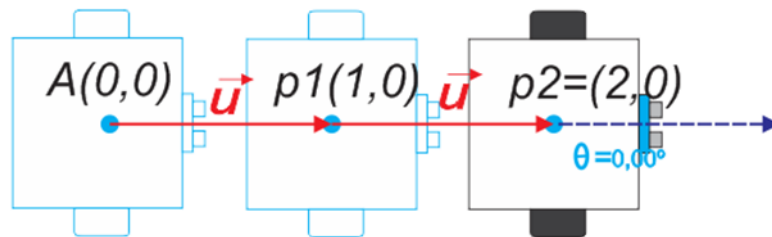
Com $x_n = \cos(\theta_n)$, $y_n = \sin(\theta_n)$, e $\theta_n = \theta$. (A angulação do novo pulso não muda durante o deslocamento X, Y do robô, a angulação só muda quando o robô gira em torno do próprio eixo para decidir sua orientação.) Resulta:

$$\begin{aligned} X &= S_{x_{n-1}} + \cos(\theta) \\ Y &= S_{y_{n-1}} + \sin(\theta) \end{aligned} \tag{4.12}$$

Deste modo a atualização das posições X e Y a cada novo pulso gerado/captado pode ter a escrita simplificada como: $X_{atual} = X_{anterior} + \cos(\theta)$ e $Y_{atual} = Y_{anterior} + \sin(\theta)$. Caso o robô se movimente dando ré, a nova componente será negativa, então: $X_{atual} = X_{anterior} - \cos(\theta)$ e $Y_{atual} = Y_{anterior} - \sin(\theta)$.

Por exemplo, o robô se move de um ponto no espaço na coordenada $A(0,0)$ para um ponto $p2$. Durante esta deslocamento foram gerados 2 vetores $\vec{u}$, passando pelos pontos $p1$ e $p2$ (Figura 4.25).

Figura 4.25 - Exemplo de movimentação do robô e de geração de vetores.



Fonte: Autor.

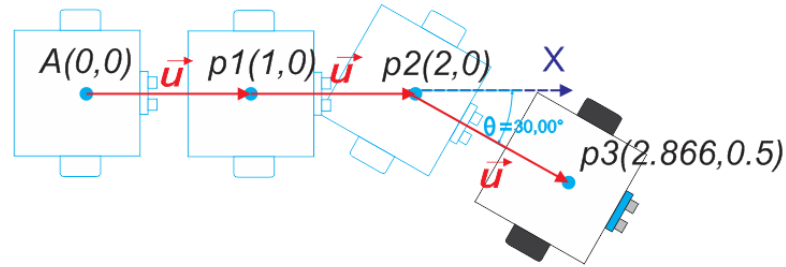
Na origem as coordenadas X e Y são $(0,0)$ e no primeiro ponto $p1$ é formado o primeiro vetor $\vec{u}$, com ângulo $\theta = 0^\circ$, portando em $p1$: $x_1 = \cos(0) = 1$, $y_1 = \sin(0) = 0$. No ponto $p2$ não acontece mudança de orientação, permanecendo $\theta = 0^\circ$, então do ponto $p1$ ao $p2$: $x_2 = \cos(0) = 1$, $y_2 = \sin(0) = 0$. Significa que aconteceu deslocamento de valor 1 no eixo X, e 0 no eixo Y, da origem ao $p1$, e também deslocamento de valor 1 no eixo X, e 0 no eixo Y, de $p1$ a $p2$. A soma geométrica destes vetores pela soma das componentes resulta na posição $X=2$ e $Y=0$, no plano de coordenadas.

$$\begin{aligned}
 x_1 &= 1 & x_2 &= 1 \\
 y_1 &= 0 & y_2 &= 0 \\
 X &= S_{x_n} \\
 X &= x_1 + x_2 = 1 + 1 = 2 \\
 Y &= S_{y_n} \\
 Y &= y_1 + y_2 = 0 + 0 = 0 \\
 & p2(2,0)
 \end{aligned} \tag{4.13}$$

Se o robô move da posição atual $p2$ para uma posição $p3$, girando 30 graus sentido horário antes de se mover (Figura 4.26), o ângulo se torna $\theta = 0^\circ + 30^\circ = 30^\circ$. De $p2$ para $p3$, as componentes do vetor $\vec{u}$ são:

$$\begin{aligned}x_3 &= \cos(30^\circ) = 0.866 \\y_3 &= \sin(30^\circ) = 0.5\end{aligned}\tag{4.14}$$

Figura 4.26 - Segundo exemplo de movimento do robô e geração de vetores.



Fonte: Autor.

Portanto a soma para um novo pulso resulta na posição $p3$. As posições X e Y anteriores foram do ponto $p2$.

$$\begin{aligned}X_{anterior} &= 2 \\Y_{anterior} &= 0 \\X_{atual} &= X_{anterior} + \cos(\theta) = 2 + 0.866 = 2.866 \\Y_{atual} &= Y_{anterior} + \sin(\theta) = 0 + 0.5 = 0.5 \\p3(2.866, 0.5)\end{aligned}\tag{4.15}$$

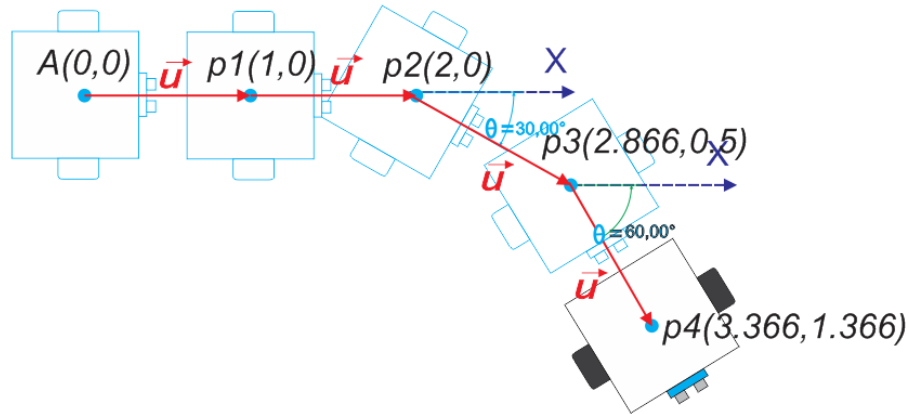
Se o robô move da posição atual $p3$ para uma posição $p4$, girando 30 graus no sentido horário, o ângulo se torna $\theta=60^\circ$ (). De $p3$ para $p4$, as componentes do vetor u serão:

$$\begin{aligned}x_4 &= \cos(60^\circ) = 0.5 \\y_4 &= \sin(60^\circ) = 0.866\end{aligned}\tag{4.16}$$

A soma das componentes resulta na posição $p4$. As posições X e Y anteriores foram do ponto $p3$.

$$\begin{aligned}X_{anterior} &= 2.866 \\Y_{anterior} &= 0.5 \\X_{atual} &= X_{anterior} + \cos(\theta) = 2.866 + 0.5 = 3.366 \\Y_{atual} &= Y_{anterior} + \sin(\theta) = 0.5 + 0.866 = 1.366 \\p4(3.366, 1.366)\end{aligned}\tag{4.17}$$

Figura 4.27 - Terceiro exemplo de movimentação e geração dos vetores.



Fonte: Autor.

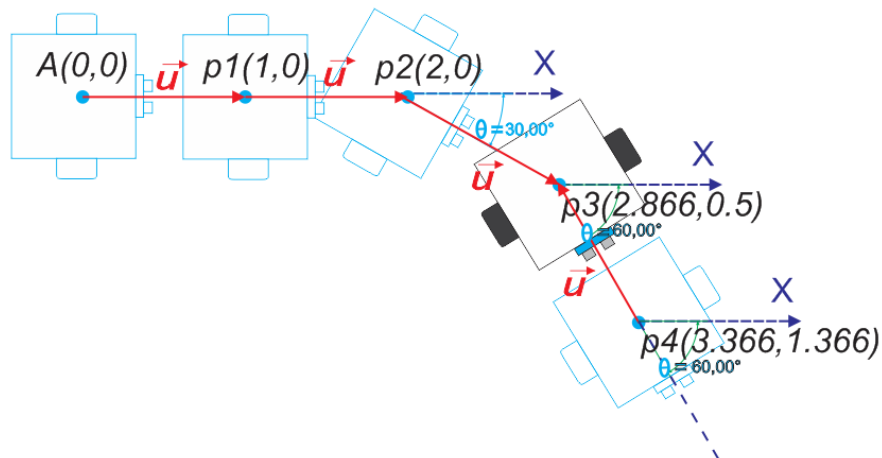
Se o robô se move da posição $p4$ em ré, voltando para $p3$, cria um vetor $\vec{u}$ oposto. Portanto, de $p4$ a $p3$:

$$\begin{aligned} x_5 &= -\cos(60^\circ) = -0.5 \\ y_5 &= -\sin(60^\circ) = -0.866 \end{aligned} \quad (4.18)$$

A soma das componentes resulta novamente na posição $p3$. As posições X e Y anteriores foram do ponto $p4$.

$$\begin{aligned} X_{anterior} &= 3.366 \\ Y_{anterior} &= 1.366 \\ X_{atual} &= X_{anterior} - \cos(\theta) = 3.366 - 0.5 = 2.866 \\ Y_{atual} &= Y_{anterior} - \sin(\theta) = 1.366 - 0.866 = 0.5 \\ p3 &(2.866, 0.5) \end{aligned} \quad (4.19)$$

Figura 4.28 – Quarto exemplo de movimentação e geração de vetores.



Fonte: Autor.

4.3.3 Encoder e implementação da interrupção

Para obter dados do deslocamento do robô foi necessário utilizar um sensor que retornasse para a placa Arduino dados que variam de acordo com a movimentação. O sensor escolhido foi o sensor de velocidade *encoder* HC-020K (Figura 4.29). O chassi do robô usado possui encaixe nas rodas para este sensor. Foi utilizado apenas um sensor, este sensor foi usado para capturar pulsos enquanto o robô se movimenta. A roda ao girar, gera pulsos proporcionados pela captura de luz no disco perfurado, estes pulsos foram traduzidos em dados pelo Arduino.

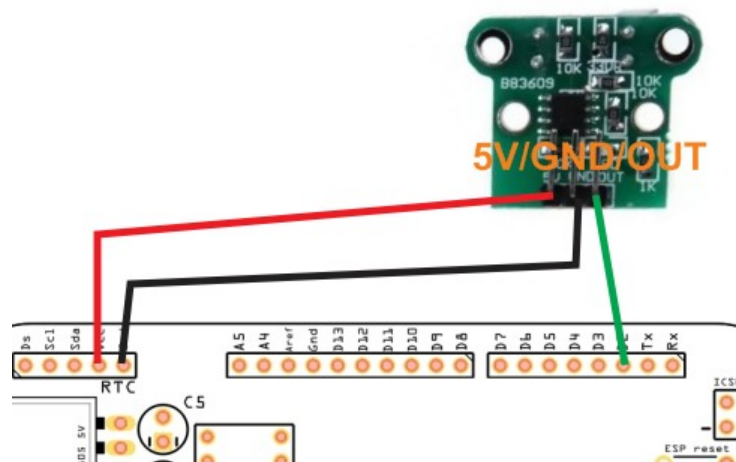
Figura 4.29 - Encoder HC-020K.



Fonte: Adaptado pelo Autor.

O sensor HC-020k possui três fios e um disco perfurado (20 furos) que foi acoplado à roda do robô. Os fios são: “5V”, “GND” e “OUT”, o fio “5V” foi conectando ao pino “Vcc” do Arduino, “GND” foi conectado ao pino “Gnd” do arduino e “OUT” foi conectado ao pino “D3” do Arduino (Figura 4.30). Os pino “D3” e “D4” do Arduino são usados para sistemas que envolvem interrupção (*datasheet*).

Figura 4.30 - Diagrama da ligação dos pinos para o *encoder*.



Fonte: Autor.

A interrupção é usada em casos que placa de Arduino precise receber vários dados de forma rápida sem a necessidade terminar o loop de leitura de dados, acessando os dados de forma paralela enquanto o loop acontece. A interrupção pode ser feita fora do loop, criando uma função que possa ser acessada de forma independente. O robô precisou de um sistema que receba vários dados em curto espaço de tempo, sendo necessário usar interrupção, estes dados são os pulsos gerados. Se os dados fossem lidos dentro da função de loop no Arduino, o robô não conseguiria contabilizar todos os dados recebidos a tempo, assim geraria um erro de contagem de pulsos resultando um número menor que o real.

Na programação no Arduino IDE, a interrupção foi colocada no “setup()” quando o código se inicializa configurando a pinagem e sua operação. A função “attachInterrupt()” habilita a interrupção de acordo com os parâmetros usados. O primeiro parâmetro usado é o pino, seguido da função executada sempre que acontece uma interrupção, o terceiro é o modo que a interrupção acontece.

O pino “D3” foi declarado como entrada, pois o sensor recebe os dados de pulso. No primeiro parâmetro, foi colocado como “0”, a função reconhece “0” como “D2” e “1” como “D3”. O segundo parâmetro foi a função criada “ContarAnguloPos()” e o terceiro parâmetro foi o modo “RISING”, este modo acessa uma interrupção quando o estado do pino vai de “LOW” para “HIGH” (Quadro 4.24).

Quadro 4.24 - Código de configuração dos pinos do *encoder*.

```
pinMode(2, INPUT); // Primeiro pino de interrupção
attachInterrupt(0, ContarAnguloPos, RISING); //
```

Fonte: Autor.

A interrupção é sensível, ocorreu de 4 a 6 contagens de pulsos quando o LED do sensor mudou de estado (apagado/aceso) uma única vez. O LED do sensor fica aceso quando o fotorreceptor do HC-020k recebe luz (quando a luz passa pelo furo), e apagado quando não recebe (quando a luz não passa pelo furo). Foi verificado diretamente e comparado o acender e apagar do LED com o número resultado de contagem de pulsos (feito dentro da função “ContarAnguloPos()”). Este problema foi resolvido com um limitador de tempo (Quadro 4.25).

Quadro 4.25 - Código do limitador de tempo.

```
void ContarAnguloPos() { // Função para valor de Angulo e Posição
    static unsigned long delayest; // delay falso para retornar um estado sem "tremida"
    if (millis() - delayest > 1) {
        contador = contador + 1;
    }
    delayest = millis();
}
```

Fonte: Autor.

A função “millis()” faz uma contagem de tempo em milissegundos (ARDUINO, 2019), a variável “delayest” guarda o atual tempo ao final da execução do código dentro da condição. Se a diferença entre o tempo atual e o tempo que o código foi executado for maior que 1 milissegundo o código da condição é executado. Isso significa que o código dentro da condição não será executando antes de 1 milissegundo, mesmo se ocorrer 4 a 6 capturas de pulsos, contando 1 pulso por execução. Foi notada deste modo a contagem de um pulso a cada transição de estado de recepção e não recepção de luz no sensor, totalizando uma contagem de 40 pulsos por volta completa da roda em um disco de 20 furos.

4.3.4 Implementação da lógica de deslocamento no mapa

A lógica vetorial para o deslocamento foi usada para a criação do código no Arduino IDE para enviar da placa Arduino ao *Processing* dados das componentes X, Y, e o ângulo θ . No *Processing* a matriz de pixels possuem posições X e Y, assim como no plano cartesiano, porém o eixo Y possui um crescimento invertido. O deslocamento do robô no mapa é a frequente atualização dos valores de posição.

A mudança dos valores X e Y da posição do robô dependem do *encoder*. O *encoder* enquanto gira, não distingue o sentido de rotação da roda, portanto foi necessário utilizar os comandos de movimento que são utilizados para controle para também distinguir separadamente os tipos de dados coletados pelo sensor, se naquele momento está coletando pulsos enquanto anda para frente, se está coletando enquanto da ré, ou se está coletando enquanto o robô gira de forma horária ou anti-horária.

As funções void robot_forward() , void robot_backward(), void robot_left(), void robot_right() (Quadro 4.26), possuem marcadores denominados “mark” estes marcadores possuem *strings* de nomes: “frente”, “re”, “esquerda”, “direita”, de acordo com os as funções de movimento que faz o robô ir para frente, trás, girar para esquerda ou girar para direita respectivamente. Estas funções são acessadas de acordo com o comando de movimento do robô. Os comandos de movimento podem ser acessados pelo código criado para o controle manual ou para o controle automático.

Os marcadores servem de condição dentro da função void ContarAnguloPos(), função acessada por interrupção sempre que o *encoder* recebe um pulso, nesta função são armazenados e calculados os valores de X, Y e ângulo θ .

Se o robô vai para frente ou para trás, é enviado o marcador “frente” ou “re”, respectivamente para a função “ContarAnguloPos()”, a condição acessada do marcador “frente” e do marcador “re”, armazenam o valor das contagens de pulsos nas variáveis

“contadorX” e “contadorY”. Por exemplo, o robô executa “robot_forward()”, o marcador “mark” adquire valor igual a “frente”, acessando a condição “if (mark==“frente”)”. Dentro desta condição são armazenadas e calculadas as variáveis “contadorX” e “contadorY”.

Quadro 4.26 - Código com marcadores.

```
void robot_forward()
{ mark="frente";
  motor1.setSpeed(170);
  motor1.run(FORWARD);
  motor2.setSpeed(205);
  motor2.run(FORWARD);
} //end robot forward

void robot_backward()
{
  mark="re";
  motor1.setSpeed(170);
  motor1.run(BACKWARD);
  motor2.setSpeed(205);
  motor2.run(BACKWARD);
} //end robot backward

void robot_left()
{ mark="esquerda";
  motor1.setSpeed(120);
  motor1.run(BACKWARD);
  motor2.setSpeed(155);
  motor2.run(FORWARD);
} //end robot left

void robot_right()
{ mark="direita";
  motor1.setSpeed(120);
  motor1.run(FORWARD);
  motor2.setSpeed(155);
  motor2.run(BACKWARD);
} //end robot right

void robot_stop()
{ // sem mark
  motor1.setSpeed(0);
  motor1.run(RELEASE);
  motor2.setSpeed(0);
  motor2.run(RELEASE);
} //end robot stop
```

Fonte: Autor.

Seguindo a lógica das equações: $X_{atual} = X_{anterior} \mp \cos(\theta)$ e $Y_{atual} = Y_{anterior} \mp \sin(\theta)$, quando a condição é “frente” o sinal é positivo, por causa do comportamento vetorial de soma geométrica e quando é “re” o sinal é negativo, pois o novo vetor adicionado possui sentido oposto. Essa lógica foi usada para calcular as novas posições a cada movimento.

A variável “contadorX” antes da equação ser resolvida pelo microprocessador, esta já possui um valor anterior, mas ao ser resolvida possui um novo valor atualizado, isso acontece a cada recebimento de pulso. Como na equação, o “contadorX” antes do processamento do cálculo é o $X_{anterior}$ e ao ser resolvido se torna o X_{atual} . Deste modo, esta variável sempre calcula um novo valor X, somando (ou subtraindo) $\cos(\theta)$ do X anterior. De forma análoga acontece o mesmo para o “contadorY” com a soma(ou subtração) de $\sin(\theta)$ (Quadro 4.27 e Quadro 4.28).

Quadro 4.27 - Código do acréscimo no contador.

```
contadorX= contadorX+1*cos(angulo);
contadorY= contadorY+1*sin(angulo);
```

Fonte: Autor.

Quadro 4.28 - Código do decréscimo no contador .

```
contadorX= contadorX-1*cos(angulo);
contadorY= contadorY-1*sin(angulo);
```

Fonte: Autor.

O código foi feito para limitar o valor do “contadorX” e “contadorY” a um número entre 0 e um valor fixo, e o robô para de se mover caso ultrapasse os limites (Quadro 4.29). Quando “contadorX” adquire um valor superior a um valor máximo “MaxX”, ou quando “contadorY” adquire um valor maior que “MaxY”, o valor da variável em questão se torna o máximo. E quando “contadorX” ou “contadorY” adquirem valor negativo, a variável que adquiriu este valor negativo se torna 0.

Quadro 4.29 - Código dos limites do ambiente.

```
if(contadorX>MaxX)
{robot_stop();
contadorX=MaxX;}
if(contadorX<0)
{robot_stop();
contadorX=0;}
if(contadorY>MaxY)
{robot_stop();
contadorY=MaxY;}
if(contadorY<0)
{robot_stop();
contadorY=0;}
```

Fonte: Autor.

As limitações máximas “MaxX” e “MaxY” são os valores máximos de X e Y calculados dado as dimensões máximas da área do mapa. O cálculo foi criado após testes visuais no *Processing*. Foi considerado no projeto uma área quadrada, fazendo assim “MaxX=MaxY”. No cálculo “MaxX=MaxY=2*(MaxPixel-15)” dentro do “setup()” (Quadro 4.30), o “MaxPixel” representa a dimensão máxima da área do mapa, foi subtraído 15 cm pela dimensão do robô (para a imagem do robô não sair do mapa) e este valor foi dobrado

pela conversão do valor, pois uma unidade do “contadorX” ou “contadorY” equivale aproximadamente 0,5 centímetros.

Quadro 4.30 - Estabelecendo os limites do ambiente

MaxPixel=500; //Dimensão máxima do mapa da matriz em cm MaxX=MaxY=2*(MaxPixel-15);

Esta limitação foi necessária para que o robô não ultrapasse os valores limites do mapa para sua exibição. Caso o robô ultrapassasse os limites de desenho no mapa, ele não iria ser exibido mesmo realizando suas funções normalmente.

As condições “esquerda” e “direita” dentro da função “ContarAnguloPos()”, foram usadas apenas para a formação do ângulo θ . O robô ao girar para direita o ângulo θ cresce e ao girar para esquerda o ângulo decresce.

Foi criada uma variável denominada “contadorr” para armazenar uma quantidade de pulsos durante o giro. O robô ao se mover na condição de “direita”, é somado uma unidade no “contadorr” a cada novo pulso, e ao se mover na condição de “esquerda” é subtraído uma unidade a cada novo pulso (Quadro 4.31).

Quadro 4.31 - Contagem para direita e para esquerda, respectivamente.

contadorr= contadorr+1;
contadorr= contadorr-1;

Fonte: Autor.

O valor armazenado no “contadorr” foi convertido em um ângulo θ (em radianos) e armazenada em uma variável chamada “ângulo”. Para converter em θ , foi necessário calcular a quantidade de pulsos suficiente para que o robô gire 360° em torno do próprio eixo.

O robô ao girar em torno do próprio eixo faz um movimento circular. O diâmetro roda a roda do robô foi medido com aproximadamente 13 cm. Para a roda completar uma volta completa em torno do eixo do robô, ela percorre a distância da circunferência roda a roda. Esta distância foi chamada de A, e o número de pulsos gerados para esta distância foi chamado de N_A .

$$\text{Circunferência roda a roda} = 13 \pi \text{ cm} = A \quad (4.20)$$

A roda possui 6.6 centímetros de diâmetro, e sua rotação completa gera 40 pulsos. Durante o movimento circular em torno do próprio eixo do robô, em um giro completo da roda, ela se locomove uma distância igual à sua circunferência. Esta distância foi chamada de B e o número de pulsos gerados nessa distância foi chamado de $N_B = 40 \text{ pulsos}$.

$$\text{Circunferência da roda} = 6,6 \pi \text{ cm} = B \quad (4.21)$$

A razão entre o número de pulsos gerados em uma determinada distância e a esta distância é um valor constante.

$$\frac{\text{número de pulsos gerados ao percorrer } D \text{ cm}}{D \text{ cm}} = \text{constante} \quad (4.22)$$

Então a razão entre o número de pulsos N_A por A, é igual a razão entre o número de pulsos N_B por B.

$$\begin{aligned} \frac{N_A}{A} &= \frac{N_B}{B} \\ \frac{N_A}{13 \pi \text{ cm}} &= \frac{40 \text{ pulsos}}{6,6 \pi \text{ cm}} \\ N_A &= \frac{40 \times 13 \pi}{6,6 \pi} = 78,78 \text{ pulsos} = 78 \text{ pulsos inteiros} \end{aligned} \quad (4.23)$$

Durante a execução do movimento de giro, com 78 pulsos o robô faz aproximadamente uma volta completa em torno do próprio eixo, significa que com 39 pulsos o robô gira aproximadamente π radianos. A proporção entre o ângulo de giro θ para π rad é igual à proporção do número de pulsos gerados para 39 pulsos.

$$\begin{aligned} \frac{\theta}{\pi} &= \frac{\text{número de pulsos gerados}}{39} \\ \theta &= \text{número de pulsos gerados} \times \frac{\pi}{39} \\ \theta \text{ (radianos)} &= \text{número de pulsos gerados} \times \frac{3,1415}{39} \end{aligned} \quad (4.24)$$

Esta fórmula foi usada para converter os pulsos do “contadorr” em “ângulo” (Quadro 4.32).

Quadro 4.32 - Conversão de pulsos em ângulo.

angulo=contadorr*3.1415/39;

Fonte: Autor.

Quando θ chega a 2π ele retorna ao valor 0 rad, através do zeramento da variável “contadorr” (Quadro 4.33), pois sem esse zeramento, a contagem de pulsos em “contadorr” e consequentemente o valor do “ângulo”, retornariam um resultado cada vez maior em dígitos, deixando o processamento de cálculo da placa Arduino cada vez mais lento.

Quadro 4.33 - Reduzindo o processamento pelo “zeramento” da variável.

if(angulo>=6.283) contadorr=0;

Fonte: Autor.

Paralelamente, a medição do ultrassom também fica em execução em loop, imprimindo na porta serial as coordenadas X, Y, o ângulo θ e a medição do ultrassom, dados que são enviados ao *Processing* (Quadro 4.34).

Quadro 4.34 - Envio dos dados do robô pela serial.

```
Serial.println((String)contadorX+","+contadorY+","+angulo+","+dist_cm);
```

Fonte: Autor.

4.3.5 Implementação do controle automático

No modo de operação automático o robô não necessita de controle manual do usuário, fazendo o mapeamento de forma autônoma. A lógica do mapeamento autônomo não foi desenvolvida totalmente, pela complexidade de desenvolver uma inteligência de mapeamento autônomo, o melhoramento desta função foi deixado como trabalho futuro. Mas o básico do mapeamento autônomo foi desenvolvido, o robô se movimenta linearmente, desvia de obstáculos sozinhos e se mantém dentro dos limites do mapa, enquanto todo percurso e obstáculos são registrados no mapa gráfico.

O controle manual, executado por comandos enviados do *Processing* ao Arduino ao pressionar teclas, retornam a variável “Auto=0”. Para controle automático, deve retornar “Auto=5”. Esta variável “Auto” foi criada para diferenciar os dois modos de controle. Por exemplo, se a tecla “cima” é pressionada na interface do *Processing*, é enviado para o Arduino o número 0, que habilita a condição “state==’0’” retornando “Auto=0” e faz o robô se mover para a frente (Quadro 4.35).

Quadro 4.35 - Envio do comando manual ao robô.

```
if(state == '0')
{
  Auto=0;
  robot_forward();
}
```

Fonte: Autor.

Para o controle automático foi criada uma função (Quadro 4.36) no *Processing* para obter uma condição de “state” específica no Arduino. Quando o botão direito do mouse é clicado envia o número 5 para a porta serial, então o robô adquire a condição “state==’5’” no Arduino.

Quadro 4.36 - Função de controle automático.

```
if ( mousePressed && ( mouseButton == RIGHT ) ) { // if the right mouse button is pressed
myPort.write ( '5' ); // Send a '5' to the Arduino IDE
}
```

Fonte: Autor.

Quando a condição “state” adquire o valor 5, é atribuído o valor 5 a variável “Auto” (Quadro 4.37). Esta variável ao possuir o valor 5, faz o robô percorrer em linha reta e acessa a

função de desvio “decision()” caso se sujeite a uma das seguintes condições: detecte um obstáculo a menos de 20 centímetros ou uma das componentes X ou Y ultrapassou o valor limite (Quadro 4.38).

Quadro 4.37 - Atribuição do estado "5", referente ao controle automático.

```
if(state=='5'){
    Auto=5;
}
```

Fonte: Autor.

Quadro 4.38 - Ações atribuídas ao estado automático.

```
if(Auto==5){
    robot_forward();
    if((dist_cm<20) || (contadorX>=MaxX) || (contadorY>=MaxY) || (contadorX<=0) || (contadorY<=0) )
    decision();
}
```

Fonte: Autor.

Foi testada a tomada de decisão para desvio virando o robô em um sentido específico quando encontra um obstáculo. Caso robô detecte um obstáculo ou quando chega aos limites do mapa, este desvia girando o robô para direita (poderia ser para esquerda, mas direita foi o sentido escolhido).

No começo da função de desvio “decision()”, a variável “contadorNew” é zerada, significa que sempre que o robô acessar esta função a variável irá começar com o valor igual a 0 (Quadro 4.40). As condições acessadas pelos marcadores “direita” e “esquerda” também zeram o “contadorNew” quando não estão em modo automático (“Auto=0”).

Esta variável serve para que o robô possa girar para direita durante a contagem de 19 pulsos. A contagem é feita dentro da condição “direita” ou “esquerda” acessada por interrupção na função “ContarAnguloPos()” quando o robô está em modo automático (“Auto=5”) (Quadro 4.39).

Os 19 pulsos equivalem aproximadamente um quarto dos 78 pulsos necessários para o robô girar uma volta completa, ou seja, aproximadamente 19 pulsos são necessários para o robô girar 90°. No código foi colocado o valor de 16, pois durante os testes da execução foi notado que o robô capta três pulsos a mais ao executar a função, então foi necessário o número 16 como limite para totalizar os 19 pulsos (Quadro 4.40).

Quadro 4.39 - Ações no contador no estado automático.

```
if(Auto==5)
    contadorNew=contadorNew+1;
else
    contadorNew=0;
```

Fonte: Autor.

Quadro 4.40 - Ajuste de 16 pulsos para que o robô gire 90°.

```

void decision()
{
  contadorNew=0;

  if (a==1) //
  {
    do { robot_right();
        dist_cm = measureDistance();

        Serial.println((String)contadorX+","+contadorY+","+angulo+","+dist_cm);
    } while(contadorNew<16);
    robot_forward();
  } //end if

```

Fonte: Autor.

Enquanto o robô gira, capta as distâncias dentro da condição “while” e envia para a porta Serial, os dados: “contadorX”, “contadorY”, “angulo” e “dist_cm”. Ao terminar a condição o robô se move para frente. A condição “if (a==1)”, foi uma condição criada que permanece sempre ativa pelo “a=1” no “setup()”. Esta condição foi colocada para inutilizar o modo automático caso o usuário quera, trocando o valor da variável “a” por um valor diferente de 1, ou trocando o valor na condição “if (a==1)”.

4.3.6 Implementação da criação de mapas

O mapa foi feito com as dimensões reais aproximadas representadas por números de pixels na tela da interface formando um mapa de matriz de pixels, na qual um pixel representa um centímetro. No mapa gerado pelo *Processing*, fatores como: a posição do robô, a distância captada pelo ultrassom, as áreas sem obstáculos, as áreas com obstáculos e áreas inexploradas, foram representados por cores e formatos geométricos diferentes. No mapa as cores brancas representam espaços vazios sem a presença de obstáculos, as cores pretas representam a presença de obstáculos, e a cor azul representa a área não explorada.

No *Processing*, foi criada uma tela interface azul com 500x500 pixels, representando uma área real de 5x5 metros (Quadro 4.41). Esta tela é o mapa gráfico, onde são desenhados o robô e os obstáculos. No “size()”, foram colocados as dimensões em pixels, e em “background()” foi colocado o valor da cor de fundo.

Quadro 4.41 - Parâmetros para criação do mapa.

```

void setup () {
  size (500, 500); // Size of the serial window
  background ( 1000 );

```

Fonte: Autor.

O robô capta as informações do ambiente que são traduzidos em dados. O Arduino envia os seguintes dados em ordem: “contadorX,contadorY,angulo,dist_cm” estes são recebidos pelo *Processing* pela porta serial. Os dados recebidos foram guardados em uma variável denominada “arduino”, estes dados foram separados em diferentes elementos em um vetor.

A função “split()”, é usada para separar *strings*, esta possui dois parâmetros, o primeiro é a *string*, o segundo é o caractere delimitador da *string* (PROCESSING, 2019). O “split(arduino, ',')” separou a *string* única “contadorX,contadorY,angulo,dist_cm”, em outras *strings*: “contadorX”, “contadorY”, “angulo” e “dist_cm”. Estes dados de foram convertidos em números racionais pelo “float()” em “float(split(arduino, ','));” e armazenados em um vetor “p” (Quadro 4.42).

Quadro 4.42 - Comando para receber e separar *strings*.

```
void serialEvent (Serial myPort) {
  arduino = myPort.readStringUntil ( '\n' ); // received data
  float p[] = float(split(arduino, ','));
```

Fonte: Autor.

O primeiro elemento do vetor “p” foi o dado X, o segundo elemento foi o dado Y, o terceiro elemento foi o ângulo, e o quarto elemento foi a distância captada pelo ultrassom. Os dados X, Y são valores de componentes X,Y enviado pelo Arduino, valores resultantes da quantidade de pulsos gerados/coletados, não significando valores de posições em escala real, deste modo foi necessária uma conversão destes valores para centímetros.

A circunferência da roda foi calculada, pois ao se mover durante um giro completo das rodas o robô se move a distância da circunferência:

$$\text{Circunferência da roda} = 6,6 \text{ cm} \times \pi = 20,73451151 \text{ cm} \quad (4.25)$$

Um giro completo da roda gera 40 pulsos. Uma unidade enviada pelo Arduino para o *Processing* é traduzido para um pixel. O robô ao se mover em linha reta durante 40 pulsos com a angulação igual a 0° envia 40 unidades da componente X para o *Processing* assim se movendo 20,73451151 centímetros no espaço real em um eixo X, o mesmo acontece com o robô se movendo em linha reta no eixo Y. Portanto uma unidade da componente X ou componente Y enviada pelo Arduino equivale a 0.51836278784 centímetros.

$$\frac{20,73451151 \text{ cm}}{40 \text{ unidades}} = 0,51836278784 \frac{\text{cm}}{\text{unidade}} \quad (4.26)$$

$$1 (\text{Arduino}) = 0,51836278784 \text{ cm}$$

Deste modo o valor das componentes enviadas do Arduino para o *Processing* foi multiplicado por 0.51836278784 para converter em centímetros.

$$\begin{aligned} X \text{ cm} &= X (\text{Arduino}) \times 0,51836278784 \text{ cm} \\ Y \text{ cm} &= Y (\text{Arduino}) \times 0,51836278784 \text{ cm} \end{aligned} \quad (4.27)$$

O ângulo e a distância do ultrassom não precisaram de conversão. Deste modo, foram feitos a conversão do X e Y e armazenados em variáveis denominadas de “x” e “y”, o dado do ângulo foi armazenado na variável “teta” e o dado da distância captada pelo ultrassom foi armazenada na variável “estado” (Quadro 4.43).

Quadro 4.43 - Utilização dos valores x e y do ultrassom.

```
x=p[0]*0.51836278784; //40*x=6,6pi // x=6,6pi/40 0.51836278784
y=p[1]*0.51836278784;
teta=p[2];
estado=p[3];
```

Fonte: Autor.

Foi criado um objeto de “mira”, este objeto acompanha o robô. A “mira” serve para indicar a orientação do robô visualmente, mostrando a sua “frente”. O centro do objeto “mira” foi situado a 10 pixels de distância do centro do robô e gira de acordo com o dado de angulação recebido. As coordenadas “miraX” e “miraY” são os resultados das projeções do módulo de 10 pixels, somados com as coordenadas da posição atual do robô.

Acompanhando a “mira”, foi criado o objeto “ultra” que indica a distância do robô a um obstáculo, se deslocando conforme a angulação, do mesmo modo que a “mira”. Porém para o cálculo das projeções do “ultra”, o módulo é dado pelo valor da distância captada do ultrassom (Quadro 4.44).

Quadro 4.44 - Distância do robô até o obstáculo via ultrassom.

```
miraX=x+10*cos(teta);
miraY=y+10*sin(teta);
ultraX=miraX+estado*cos(teta);
ultraY=miraY+estado*sin(teta);
}
```

Fonte: Autor.

Na função “draw()” foram criadas as representações do robô, mira e distância do ultrassom. Para representar o robô no mapa foi criado um quadrado 15x15 pixels de cor branca e borda vermelha, representando aproximadamente 15x15 centímetros o robô real. A posição do centro do quadrado na matriz de pixels varia de acordo com os valores “x” e “y” calculados pelo *Processing*, estas são aproximadamente as coordenadas X e Y em centímetros de uma área real (Quadro 4.45).

Quadro 4.45 - Código da criação gráfica do robô.

```
fill(255);
stroke(#ff0000);
rectMode(CENTER);
rect(x,y,15,15);
```

Fonte: Autor.

Sobre o código, “fill()” é o preenchimento de cores, o valor “255” corresponde ao branco. A função “stroke()” é para a coloração da borda, o “#ff0000” corresponde ao vermelho. O “rectMode(CENTER)” declara o ponto central do objeto “rect” como o ponto usado para posicionamento do objeto. O “rect(x,y,15,15)” cria um retângulo com quatro parâmetros, os dois primeiros são a posição x e y na interface, e os dois últimos as dimensões x e y do retângulo, no caso 15x15 pixels, formando um quadrado.

Para representar o alcance do ultrassom foi criada uma “linha” que tem início no centro da “mira” e termina no centro do objeto “ultra”. A posição do objeto “ultra” é a posição do obstáculo captado, deste modo o tamanho da “linha” é a distância entre o robô e o obstáculo. No código (Quadro 4.46), “fill(#ff0000)” preenche o objeto “mira” com a cor vermelha. “strokeWeight(2)” define a espessura da “linha” para 2 pixels. Logo após foram definidas a cor da “linha” e a borda da “mira” para a cor branca (“stroke(255)”), e os parâmetros de início e final da “linha” pela função “line(miraX,miraY,ultraX,ultraY)”.

A função “strokeWeight(1)” retorna as espessuras das linhas em 1 pixel. A função “rectMode(CENTER)” declara o ponto central da “mira” como o ponto usado para posicionamento. Na linha seguinte, “rect(miraX,miraY,5,5)” usa as variáveis “miraX” e “miraY” como as coordenadas para o posicionamento e cria um quadrado 5x5 pixels na interface para representar o objeto “mira”.

Quadro 4.46 - Preenchimento das cores no mapa.

```
fill(#ff0000); // preenchimento da mira
strokeWeight(2);
stroke(255);
line(miraX,miraY,ultraX,ultraY);
strokeWeight(1);
rectMode(CENTER);
rect(miraX,miraY,5,5);
```

Fonte: Autor.

O objeto “ultra” foi criado para se posicionar na extremidade da “linha”, este objeto serviu para verificação se contém ou não obstáculos. A diferenciação na presença ou ausência de obstáculos foi feita através de alteração de cores no mapa. Quando não há obstáculos a uma distância de 70 centímetros a coloração do objeto “ultra” é branca com a borda vermelha, e quando detecta um obstáculo há uma distância de 70 centímetros ou menos, a coloração do

“ultra” se torna preta com borda verde. Tanto as cores brancas quanto as pretas foram demarcadas no mapa para diferenciar a presença ou não presença de obstáculos.

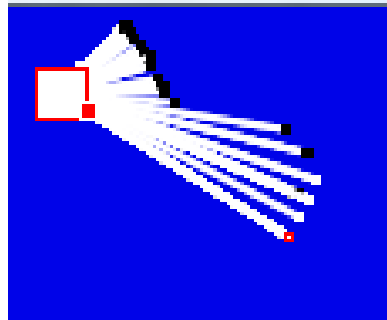
No código (Quadro 4.47), a condição “if(estados<70)” é acessada quando o ultrassom capta uma distância menor que 70 cm, dentro desta condição foram definidas as variáveis para coloração, “cor=0” (preto) e “cor1=#00ff00” (verde). Sendo a “cor” a variável usada para o preenchimento e a variável “cor1” para a borda. A condição “else” é acessada quando o ultrassom capta uma distância maior ou igual a 70 cm, dentro desta condição foram definidas as variáveis “cor=255” (branco) e “cor1=#ff0000” (vermelho).

Quadro 4.47 - Preenchimento dos obstáculos no mapa.

```
if(estados<70)
{cor=0;
cor1=#00ff00;}
else
{cor=255;
cor1=#ff0000;}
fill(cor);
stroke(cor1);//
rectMode(CENTER);
rect(ultraX,ultraY,2,2);
```

Fonte: Autor.

Figura 4.31 - Demonstração do aspecto do robô, mira, extremidade, range e seus estados no mapa.



Fonte: Autor.

O robô ao se mover cria rastros de “clones” no mapa gráfico. Nestes “clones”, as cores vermelha e verde são indesejadas, pois não possuem significância para diferenciar a presença ou não de obstáculos, e também pode confundir o usuário na visualização. Foi criada uma função “apagar()” que substitui a cor vermelha por branca e a cor verde por preta nos rastros indesejados deixados (Quadro 4.48).

Quadro 4.48 - Remoção das cores indesejadas do mapa.

```
void apagar()
{
float tamanho;
color white = color(255);
color red = color(#ff0000);
color black = color(0);
color verde = color(#00ff00);
color atual;
tamanho = 1000;
```

```

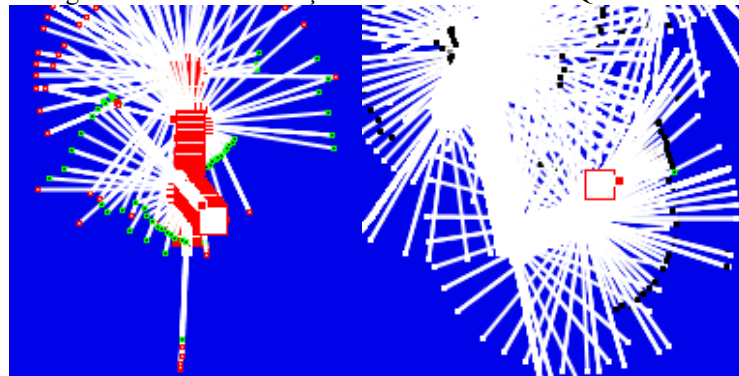
for (int i=0; i<tamanho; i++)
{
    for (int j=0; j<tamanho; j++)
    {
        atual = get(i,j);
        if (atual == red) {set(i,j, white);}
        if (atual == verde) {set(i,j, black);}
    }
}
}

```

Fonte: Autor.

Nesta função, o laço “for()” verifica toda a matriz de pixels da interface, armazena cada cor de pixel em uma variável denominada “atual” e verifica sua cor, se for vermelha, a transforma em branco, se verde, a transforma em preto. A função responsável “get(i,j)”, captura a cor do pixel, caso o pixel tenha a cor “red” ou “verde”, as as funções “set(i,j, black)” e “set(i,j, white)”, atribuem a cor “black” ou “white” a este pixel.

Figura 4.32 - Demonstração do efeito descrito no Quadro 4.48.



Fonte: Autor.

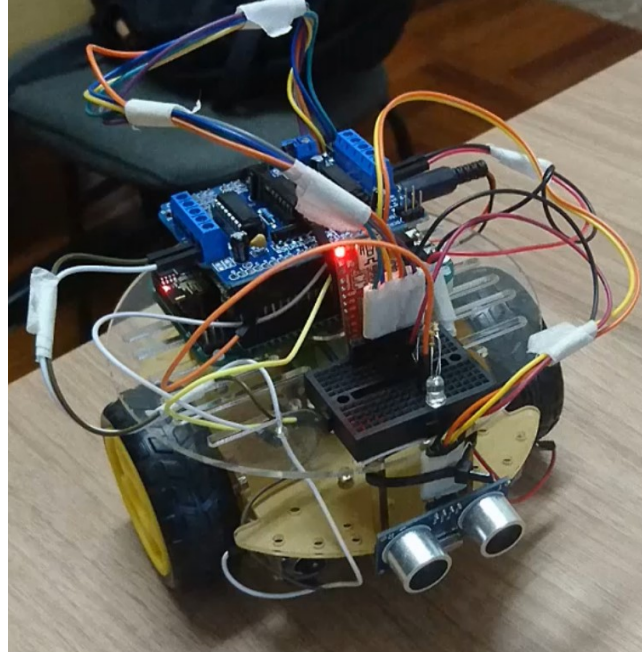
4.4 CONSIDERAÇÕES FINAIS

Nesta seção foram apresentadas as etapas de desenvolvimento do projeto final, etapas nas quais foram detalhadamente descritas culminando na conclusão bem-sucedida da construção do protótipo. Na próxima seção, Resultados, será abordado a operação do protótipo, explorando o seu funcionamento e identificando suas limitações. Também será mostrado com mais detalhes a interface de mapeamento.

5 RESULTADOS

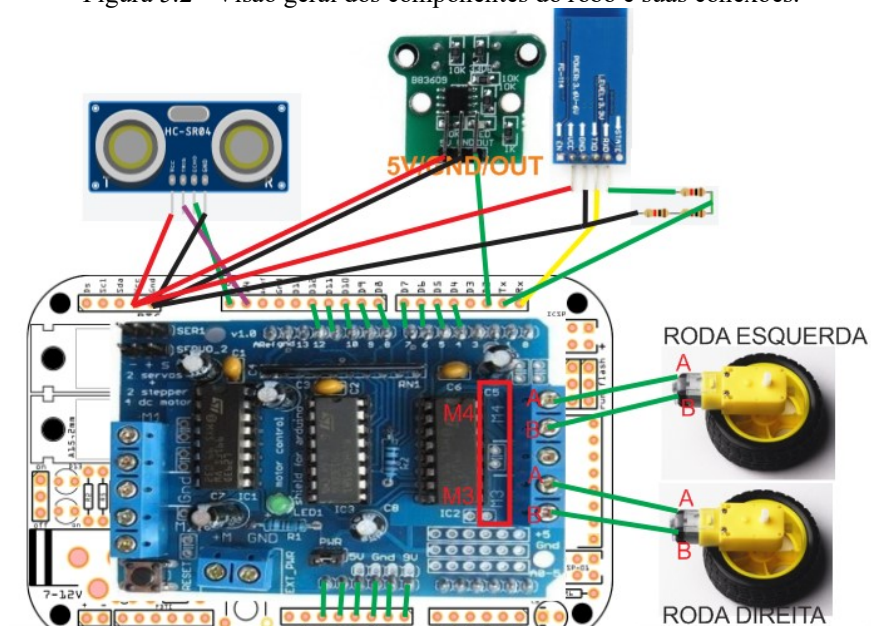
O aspecto final do protótipo é indicado na Figura 5.1, onde é possível observar a placa MBZ e com o motor *shield* na parte traseira do robô, a fixação do sensor de ultrassom na parte frontal do robô e um protoboard auxiliar para o módulo *Bluetooth*. A visão geral das conexões é indicada na Figura 5.2.

Figura 5.1 - Aspecto final do protótipo, com posicionamento do sensor de ultrassom na parte frontal do robô.



Fonte: Autor.

Figura 5.2 - Visão geral dos componentes do robô e suas conexões.

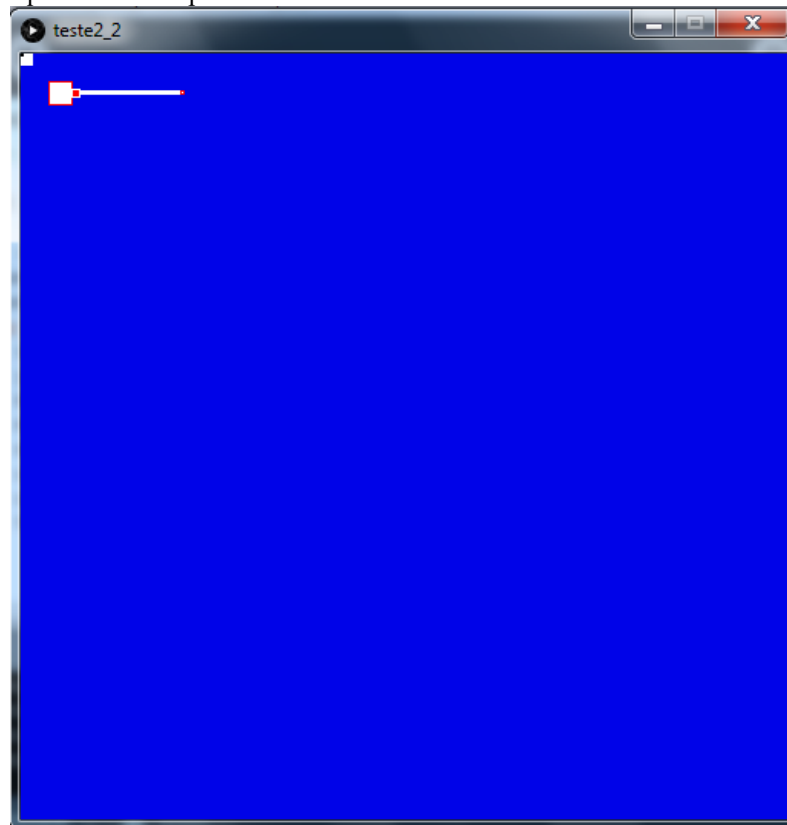


Fonte: Autor.

Considerando os códigos finais para o Arduino e *Processing* (APÊNDICE I e APÊNDICE II), o primeiro foi gravado no microcontrolador por meio da IDE do Arduino e o segundo é executado em um computador com possibilidade comunicação via Bluetooth.

Ao executar o programa com as configurações das portas COM corretas, é estabelecida a conexão entre a interface e o robô. A interface consiste na exibição de um mapa e da representação do robô no mesmo, onde é possível receber e enviar comandos diretamente ao robô (Figura 5.3).

Figura 5.3 - Tela inicial do mapeamento do robô em seu estado inicial, considerando que não foi detectado nenhum obstáculo e que o robô está parado.

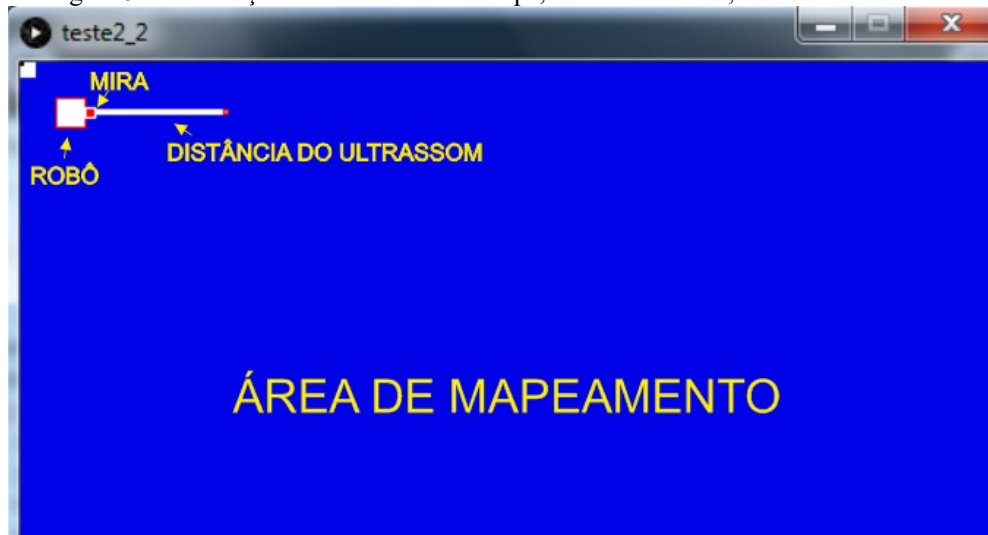


Fonte: Autor.

Para a tela inicial, foi empiricamente estabelecido uma dimensão de 500x500 pixels, onde cada pixel corresponde aproximadamente a 1 cm no mundo real. As cores representam os seguintes estados: área não mapeada (azul), área livre de obstáculo (branco), área com obstáculo (preto), contorno do robô (quadrado vermelho preenchimento branco) e a mira para indicar para qual lado o sensor e o robô estão apontados (quadrado branco com preenchimento vermelho).

A dimensão do robô foi representada como um quadrado branco com borda vermelha de 15x15 pixels com uma mira na frente, representado por um quadrado vermelho e borda branca de 5x5 pixels. A mira possui o centro a 10 pixels de distância do centro do robô. O alcance do ultrassom foi representado por uma linha branca com 70 pixels de tamanho máximo com um quadrado de 2x2 pixels na sua extremidade

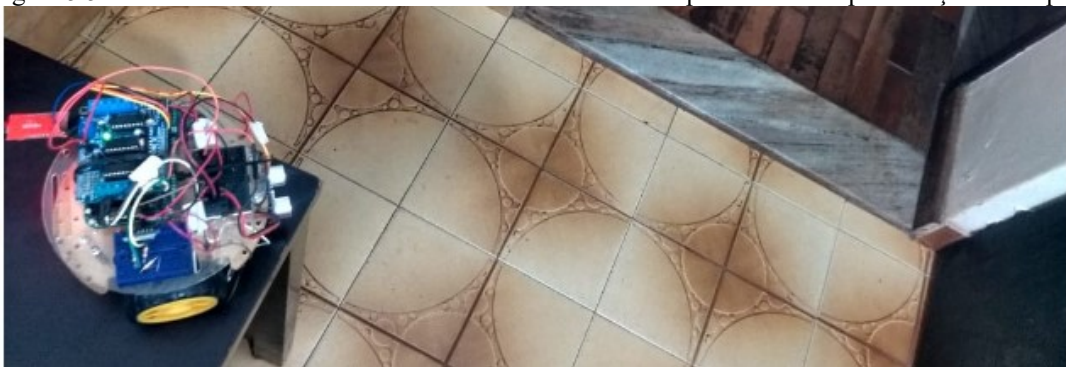
Figura 5.4 - Indicação dos elementos do mapa, incluindo o robô, a mira e o ambiente.



Fonte: Autor.

O quadrado 2x2 que varia a coloração dependendo se encontra ou não obstáculo, coloração branca com borda vermelha se não encontra obstáculo, coloração preta com borda verde se encontra obstáculo. Foi testada em duas situações, sem obstáculo à frente do robô e com obstáculo à frente do robô.

Figura 5.5 - Leitura do ambiente real sem obstáculos com o robô parado e sua representação no mapa.



Fonte: Autor.

5.1 DETECÇÃO ESTÁTICA DE OBSTÁCULOS

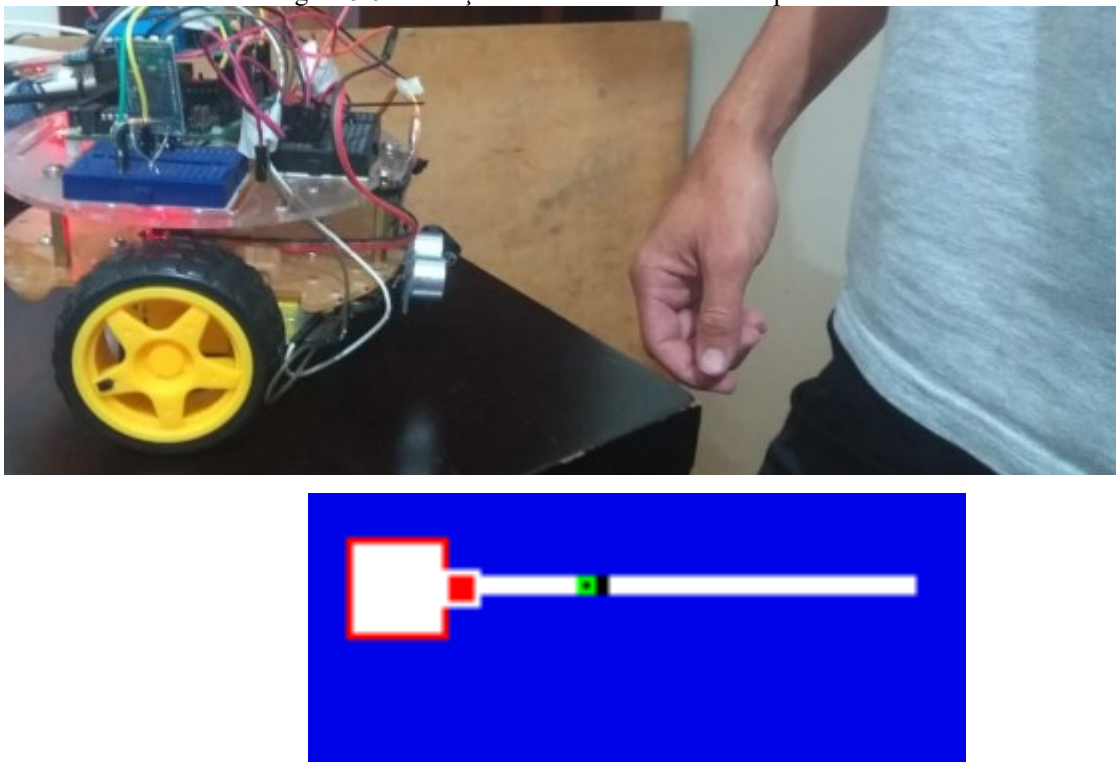
O primeiro teste completo executado no protótipo foi a detecção de obstáculo e a verificação de sua representação no mapa.

O alcance do sensor ultrassônico e sua extremidade marcam a área por onde passam. A linha de alcance sempre marca com a cor branca, e sua extremidade marca com a mesma cor do estado de detecção (preto ou branco, excluindo as bordas verdes e vermelhas).

O alcance da linha é 70 pixels na tela, mas é reduzido dependendo da distância do robô ao obstáculo. Quando existe um obstáculo detectado a uma distância menor de 70 centímetros, o tamanho da linha de alcance se torna igual à distância do robô ao obstáculo (em centímetros) e sua extremidade marca a posição do local de preto.

O primeiro teste foi executado com o robô parado e o código em execução, com o posicionamento de um obstáculo à frente do sensor. Como visto na Figura 5.6, a representação do mapa adicionou um marcador preto na posição referente ao obstáculo, indicando a obstrução do caminho.

Figura 5.6 - Exibição de obstáculo com o robô parado.

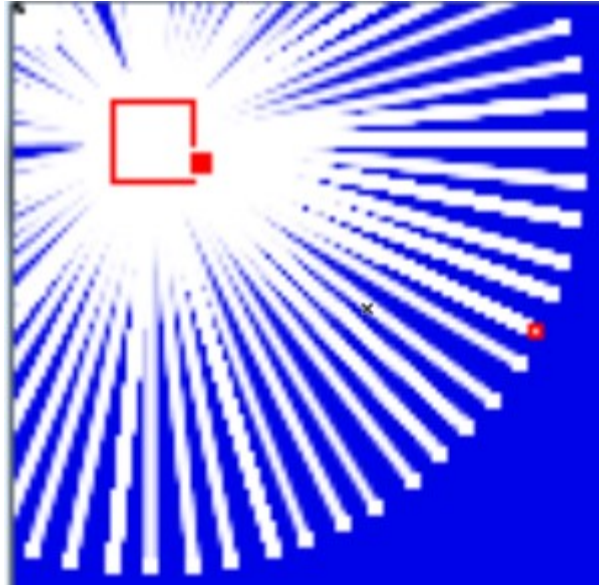


Fonte: Autor.

Em um segundo momento, foi feito o giro do robô manualmente para verificação se ao girar, o robô iria fazer as representações adequadas utilizando as informações de rotação do robô. Em um primeiro momento, o teste foi elaborado com o ambiente livre de obstáculo

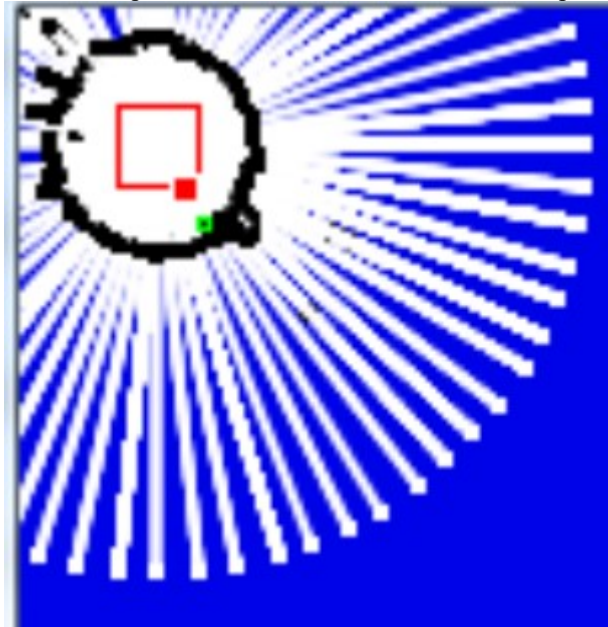
(Figura 5.7) e com a presença de um obstáculo em uma distância relativamente fixa (Figura 5.8).

Figura 5.7 - Teste de giro do robô e suas respectivas leituras com o ambiente livre de obstáculos.



Fonte: Autor.

Figura 5.8 - Teste de giro do robô e suas respectivas leituras com o ambiente com a presença de obstáculos.



Fonte: Autor.

É possível notar na primeira figura que como em volta do robô não possuía obstáculos, foi demarcado a área em branco, não ultrapassando o alcance máximo do sensor delimitado de 70 centímetros. Na segunda figura, ao inserir o obstáculo em volta do robô a uma distância próxima, foi demarcado de preto o local que foi detectado do obstáculo. Foi notado que a distância do robô até os obstáculos era próxima observando a distância dos pixels pretos.

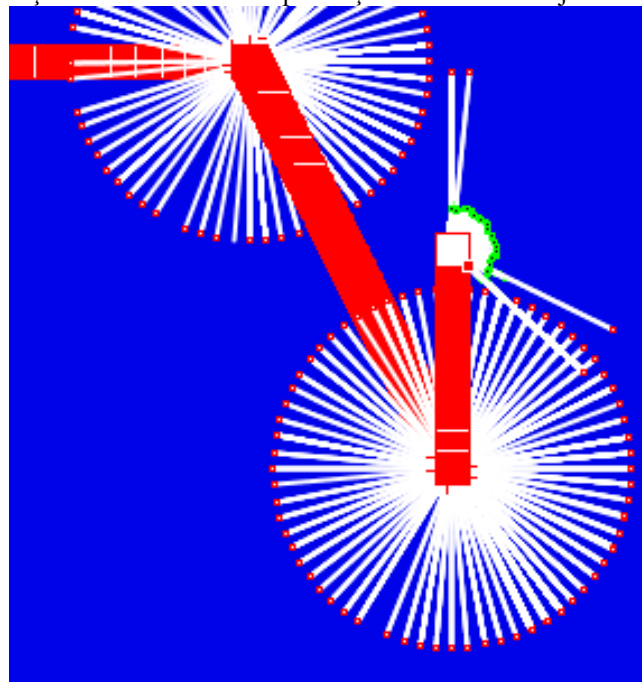
5.2 MOVIMENTAÇÃO E DETECÇÃO DE OBSTÁCULOS

Inicialmente quando o robô está em repouso só existem áreas inexploradas. Enquanto o robô se move no espaço físico, o quadrado que o representa no mapa também se move, deixando um rastro de cor branca na área por onde ele passou, indicando uma área explorada.

Após ter verificado que o robô estava mapeando de forma correta estando parado (ou apenas girando), foi feita uma segunda avaliação referente à combinação de movimentos e o mapeamento.

Nos testes iniciais de mapeamento (código sem a função “apagar()”), o movimento do robô no mapa do *Processing* gerava uma marcação indesejada de forma contínua das cores do robô, mira, linha e extremidade. Desta forma, as cores de borda, vermelho e verde deixavam rastros (Figura 5.9) confundindo assim o usuário sobre a posição atual do robô provocado pelos “clones” do robô.

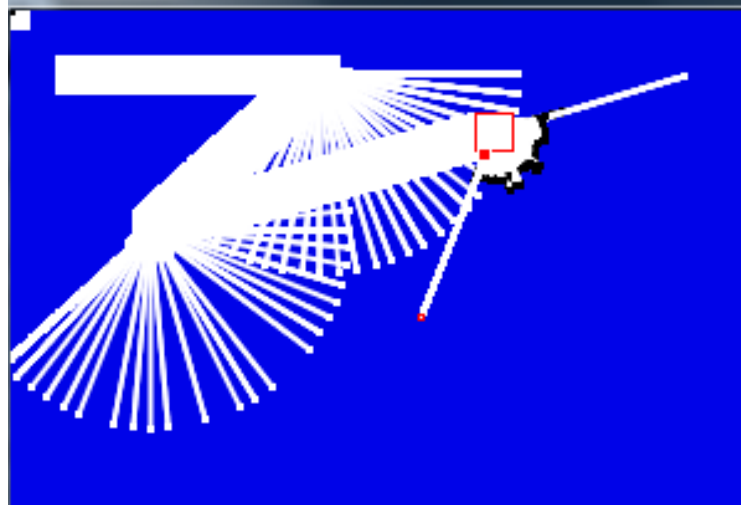
Figura 5.9 - Representação do ambiente com a presença de rastros indesejáveis do movimento do robô.



Fonte: Autor.

Entretanto, os rastros indesejáveis foram eliminados através da substituição das cores de borda indesejáveis. Deste modo foi possível visualizar a posição atual do robô e mapear normalmente, conforme indicado na Figura 5.10.

Figura 5.10 - Remoção dos traços indesejados, sem afetar a representação do robô.



Fonte: Autor.

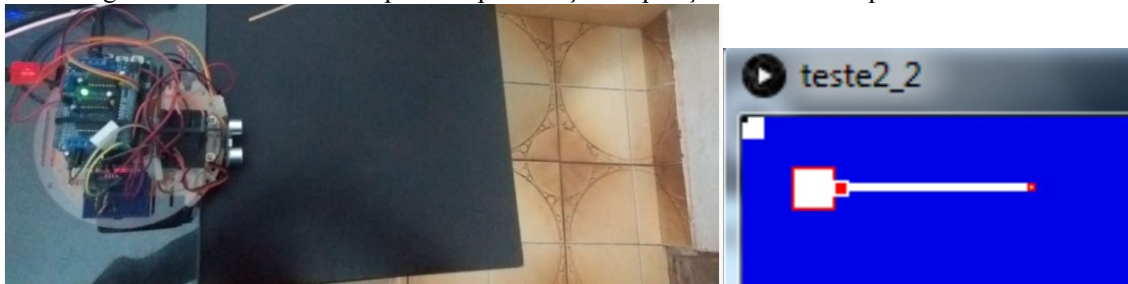
5.3 EXECUÇÃO EM AMBIENTE REAL DE FORMA REMOTA

A posição, direção e sentido inicial do robô no mapa gráfico são pré-determinados de acordo com a inicialização do robô. O local onde o robô começa a executar o mapeamento sempre é a posição X e Y inicial, considerando os valores X e Y atribuídos para começo (no teste foi colocado aproximadamente 25 cm para X e Y, para o robô não começar na borda do mapa). A angulação inicial também é pré-determinada, começando como 0°. Isso significa que o formato do mapa construído depende de onde o robô começa no ambiente e onde sua “frente” aponta.

Por exemplo, se o robô começa em uma posição específica e se desloca em linha reta para frente, é registrado no mapa o percurso do robô. Mas se o robô começar virado em uma direção diferente ou começar em outro ponto do espaço e executar o mesmo percurso, o mapa fará o mesmo desenho.

Para demonstrar este efeito da posição inicial, o robô foi colocado em uma posição de partida específica Figura 5.11, onde o alcance do sensor não detecta um obstáculo inicial, gerando um mapa inicial sem obstáculo. Logo após, o robô foi colocado em outro ponto (Figura 5.12) e foi executado o código. Nota-se que o ponto de partida no mapa continuou o mesmo, mas agora a representação iniciou com a detecção de obstáculo.

Figura 5.11 - Primeiro exemplo da representação da posição inicial no mapa e seus efeitos.



Fonte: Autor.

Figura 5.12 - Segundo exemplo da representação da posição inicial no mapa e seus efeitos.



Fonte: Autor.

Para testar o robô em campo, por meio da comunicação por *Bluetooth*, ao pressionar as teclas de movimento o robô se movimenta de forma correta conforme os comandos dados e captou a posição dos obstáculos com certa precisão. O teste foi feito em um local controlado de área pequena devido às limitações do ambiente (Figura 5.13).

Figura 5.13 - Teste em ambiente real com a verificação do mapeamento e de sua exibição na interface.



Fonte: Autor.

O funcionamento do mapeamento também funcionou de forma correta para o controle automático. Neste controle, o robô automaticamente segue em linha reta mapeando o ambiente até detectar um obstáculo à 20 cm de distância. Quando detecta este obstáculo nesta distância o robô gira para direita até o caminho estar livre para continuar mapeando. Este comportamento foi feito de forma empírica, apenas para demonstrar que o robô é capaz de realizar de forma contínua uma ação de mapeamento autônoma.

Por fim, pode ser concluído que a interação do robô com o ambiente permitiu coletar dados pelos sensores. Paralelamente, a comunicação entre a placa Arduino e PC, permitiu corretamente traduzir estes dados coletados em pixels. Em testes isolados do robô como posições X,Y, ultrassom e angulação em um ambiente favorável, por exemplo, em cima de uma mesa sendo alimentado pelo cabo USB o robô conseguiu mapear corretamente.

5.4 LIMITAÇÕES ENCONTRADAS

Algumas limitações atenuantes foram encontradas durante a execução do mapeamento:

- **Problema de deslizamento da roda e solo irregular:** Devido ao piso de teste possuir relevos e o robô ser leve, aconteceu que em alguns momentos as rodas ao girar, deslizar no solo e o robô não sair do lugar, acontecendo uma “patinação”, ou o fato do solo ser irregular e travar o deslocamento do robô mesmo com as rodas girando. Deste modo o mapa capta o movimento do robô mesmo este não ter se movido conforme o comando dado. A “patinação” costuma ser frequente em situações de mudança de direção, ou início de movimentação, acarretando principalmente erros que dependem da contagem de pulsos no giro do robô. Se o robô fosse maior e mais pesado, ou o atrito ao solo maior, ou o robô usar chassi com rodas de esteira, poderia evitar tal problema.
- **Diferença dos motores DC:** Por causa da diferença de potência dos motores foi necessário fazer testes para calibrá-los para que tenham a mesma velocidade. No entanto a calibração não é perfeita, assim, durante uma movimentação reta por um tempo muito prolongado, o robô tende um pouco para uma direção. Também, ao testar a movimentação só com uso da bateria, no caso da bateria ficar com menos energia, o motor DC menos potente começa a falhar. Nesta ocasião ocorre erro de movimentação do robô ou a roda interrompe o giro, provocando problemas nas contagens de pulsos, acarretando principalmente problemas no cálculo da angulação. Este problema poderia

ser evitado se ambos os motores forem de melhor qualidade, ou usar um sistema de compensação para regulação de giro da roda ou regulação de movimento do robô.

- **Problema de comunicação PC com Arduino:** A interface de mapeamento no *Processing* pode falhar no recebimento de dados do Arduino, esse erro acontece com mais frequência quando é usado a comunicação *Bluetooth*. Também ocorre um pequeno atraso para a interface receber os dados. Entretanto, para enviar dados do PC ao Arduino não houve falhas, mesmo quando acontece falha no recebimento de dados. Para resolver este problema poderia ser feita uma troca de comunicação sem fio substituindo o *Bluetooth* que seja eficiente.
- **Forma física do chassi:** A roda dianteira do chassi pode travar o movimento, acarretando mudança de direção ou orientação do robô e não mudando na interface. Um chassi com “roda-bola” poderia suprir este problema.
- **Diâmetro do robô:** O cálculo da angulação gerada através de captura de pulsos depende dos diâmetros da roda e do robô, na qual um valor mesmo com erro baixo faz diferença na angulação. O robô por possuir o diâmetro difícil de medir foi necessário fazer várias medições usando vários diâmetros aproximados e testados para um melhor resultado. Um sistema eficiente para regular o giro do robô em torno dele mesmo contabilizando os graus de giro de forma eficiente poderia suprir este problema.
- **Movimento falso:** O *encoder* óptico captura o pulso provocado pelo movimento da roda mesmo que o movimento não seja intencional. O movimento da roda provocado pela inércia em uma parada ou troca de direção pode contabilizar um ou dois pulsos. O problema provocado pela inércia pode parcialmente ser suprimido diminuindo a velocidade dos motores, porém ao diminuir a velocidade, o motor pode falhar. Foi necessário testar uma velocidade mediana. Um chassi com rodas de esteira, para um giro não sucessível a movimentos falsos poderia resolver o problema.
- **Contagem de interrupções:** Os valores na contagem de interrupções podem variar, gerando múltiplas contagens em um pulso. Foi necessário criar um “falso *delay*” para limitar a quantidade de pulsos captados em uma interrupção e usar a função de enviar os dados na função de loop. Quando os dados são enviados diretamente da função de interrupção ou são enviados do *loop*, os valores destes dados também são diferentes, quando isso ocorre, dependendo da aplicação pode não ser problema. O problema foi resolvido com o “falso *delay*”, porém um contador de pulsos exatos supriria este problema.

- **Problema com a reflexão do ultrassom:** A onda sonora refletida se não for captada pelo ultrassom, durante o mapeamento é mostrada como distância máxima, indicando caminho livre. Um sensor de distância eficiente poderia suprir este problema.

6 CONCLUSÃO, CONTRIBUIÇÕES E TRABALHOS FUTUROS

A robótica é um campo em grande desenvolvimento por surgir novos métodos e alternativas para solucionar problemas que necessitam a substituição humana ou facilitam suas atividades. Este trabalho exemplifica a utilidade do mapeamento por robô móvel e o seu monitoramento. Diferentes tipos de trabalhos que envolvem mapeamento robótico possuem diversas finalidades finais, sendo estas educativas ou por segurança por exemplo, mas em todas é necessário a correta apresentação de um mapeamento territorial.

Neste trabalho é proposto a elaboração de um protótipo robótico capaz de realizar mapeamento simples, tanto de forma manual quanto de forma automática.

Várias teorias de mapeamento foram estudadas, técnicas de mapeamento foram escolhidas com base nesses estudos, e com base em trabalhos de diferentes autores com finalidades parecidas. Foram também escolhidos o sensor e a comunicação que melhor satisfazia os requisitos do protótipo. A construção do mesmo foi apresentada em detalhes de forma didática para facilitar a reprodução do mesmo, assim como foram apresentados os resultados comprovando o seu correto funcionamento.

Estes resultados do protótipo foram satisfatórios ao considerar a quantidade de fatores limitantes, como as limitações impostas ao se utilizar uma plataforma de baixo custo. O mapeamento é capaz de identificar objetos a frente do robô e realiza o correto mapeamento a partir do ponto de partida conhecido no mapa.

Como já mencionado, o problema de localização e mapeamento é complexo, é uma área que demanda conhecimentos multidisciplinares avançados, mas que pôde ser inicialmente abordada de forma satisfatória neste trabalho.

Além das limitações citadas no capítulo anterior, houve dificuldades em lidar com sistemas de comunicações mais avançados, devido ao tempo requerido para lidar com todos os problemas encontrados. Desta forma, a mesma foi simplificada por meio da utilização do *Bluetooth*, sendo escolhido neste trabalho focar no robô e na geração do mapa.

As sugestões de continuidade deste trabalho são diversas. É possível utilizar algoritmos mais avançados para realizar o mapeamento automático, como por exemplo, a utilização de algoritmos genéticos e de redes Bayesianas. Nestes casos, é necessário também um estudo para determinar se é possível implementar estes algoritmos em um microcontrolador como o Arduino.

Outra sugestão é o melhoramento da plataforma robótica ao atualizar sua comunicação para Wi-Fi em vez do *Bluetooth*, ao instalar rodas livres que não interfiram no movimento do robô, melhorar a aderência do robô ao solo, melhorar a qualidade dos motores ou criar formas de melhor compensar a diferença entre os mesmos e uso de mais sensores de distância e de estados internos, como por exemplo, um sensor inercial.

O trabalho também pode ser aproveitado e expandido para um mapeamento tridimensional e (ou) utilizar outros tipos de robôs para realizar o mapeamento, como por exemplo, o uso de minidrones, utilizando o mesmo conceito de mapeamento por grade de ocupação e ter sua localização monitorada por um usuário. E para verificar a otimização do mapeamento, aplicar métricas para avaliação de mapas.

Por fim, a construção do protótipo robótico foi uma forma de enriquecer o conhecimento multidisciplinar, pois consiste em pensar, estudar diversos conceitos nas engenharias, planejar sistemas de comunicação, sistemas de mapeamento, entre outros. É preciso estudar os conceitos de mapeamento, eletrônica, mecânica, programação e de microcontroladores, e colocar todo este conhecimento para trabalhar em conjunto, contribuindo assim para a formação acadêmica do aluno.

REFERÊNCIAS

ABNER, A. S. S.; MARCOS, L. G. G. e ADELINO, A. D. M. **Mapeamento com Sonar Usando Grade de Ocupação Baseado em Modelagem Probabilística**. Dissertação de mestrado em engenharia de elétrica e de computação, UFRN, Natal, RN, fev. 2008.

ABNER, A. S. S. e MARCOS, L. G. G. **Mapeamento Robótico 2,5-D com representação em Grade de Ocupação-Elevação**. Tese de doutorado em engenharia elétrica e de computação, UFRN, Natal, RN, ago. 2012.

ADA, L. *AF_DCMotor Class*, 27 Ago. 2012. Disponível em: <<https://learn.adafruit.com/adafruit-motor-shield/af-dcmotor-class>>. Acesso em: 15 jul 2019.

ALSHAZLY, H. A. e HASSABALLAH, M. *An Embedded System for a Bluetooth Controlled Mobile Robot Based on the ATmega8535 Microcontroller*. In: *Egyptian Computer Science Journal*, Vol. 40, ISSN: 1110-2586. 2016.

ARAUJO, A. *et al.* *Integrating Arduino-Based Educational Mobile Robots in ROS*. In: *Journal of Intelligent Robot System*, DOI: 10.1007/s10846-013-0007-4. 2014.

ARDUINO. **Documentação de Referência da Linguagem Arduino**. Arduino, 2019. Disponível em: <<https://www.arduino.cc/reference/>>. Acesso em: 15 jul 2019.

BURKE, S. e GREINER, A. *These self-flying drones keep a constant eye on your business*. *CNN Tech*, New York, 01 maio 2018. Disponível em: <<http://money.cnn.com/2018/05/01/technology/business/self-flying-drones/index.html>>. Acesso em: 08 maio 2018.

CHOSSET, H. *Principles of Robot Motion*. MIT Press. 2005

FRY, B. e REAS, C. *Reference. Processing was designed to be a flexible software sketchbook*. *Processing*, 2019. Disponível em: <<https://www.processing.org/reference/>>. Acesso em: 15 jul 2019.

FRY, B. e REAS, C. *Getting Started with Processing*. O'Reilly Media, Inc. 1005 Gravenstein Highway North, Sebastopol, CA 95472, Estados Unidos, 2010.

GONÇALVES, R. C. e CHAIMOWICZ, L. **Uma abordagem para o mapeamento e localização simultâneos utilizando informação topológica**. Dissertação de mestrado em ciência da computação, UFMG, BH, MG, 2011.

GOUVEIA, R. **Velocidade do Som**. TodaMatéria, 10 set. 2018. Disponível em: <<https://www.todamateria.com.br/velocidade-do-som/>>. Acesso em: 15 jul 2019.

HALLIDAY, D.; RESNICK, R. e WALKER, J. *Fundamentals of Physics EXTENDED. 9th ed.* Cleveland State University, 2012.

HANZA, A. *How to Make Arduino and Processing IDE Communicate. MAKER PRO*, 21, mar. 2018. Disponível em: <<https://maker.pro/arduino/tutorial/how-to-make-arduino-and-processing-ide-communicate>>. Acesso em: 15 jul 2019.

KORTENKAMP, D.; BONASSO, R. P.; MURPHY, R., eds. *Artificial intelligence and mobile robots: case studies of successful robot systems*. Cambridge, MA, USA: MIT Press, 1998.

MACHARET, D. G. **Introdução à Robótica: Robótica Móvel-Mapeamento. Departamento de Ciência da Computação, UFMG, BH, MG, 2017.** Disponível em: <<http://homepages.dcc.ufmg.br/~doug/cursos/lib/exe/fetch.php?media=cursos:introrobotica:2017-1:aula19-mapeamento.pdf>>. Acesso em: 07 jun 2018.

MATARIC, M. J. *et al.* **Introdução à Robótica.** UNESP. 2014

MAXIMIANO, M. **MBZ Pro Mega Wi-Fi Edition.** Disponível em: <<https://www.maxblitz.com.br/p/mbz-pro-mega-wifi-edition.html>>. Acesso em: 5 jun 2019.

MONTEMERLO, M. *et al.* **A System for Three-Dimensional Robotic Mapping of Underground Mines.** *Comput. Sci. Dept.*, Carnegie Mellon Univ. 2002.

MURPHY, R. R. *Introduction to ai robotics.* Cambridge, MA, USA: MIT Press, 2000.

NASUTION, T. H.; SIAGIAN, E. C. e TANJUNG, K. **Design of river height and speed monitoring system by using Arduino.** *IOP Conference Series: Materials Science and Engineering, Department of Electrical Engineering, Universitas Sumatera Utara, Medan, Indonesia*, 2018.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION. **NASA's twin robot geologists, the Mars Exploration Rovers.** NASA, Washington, DC, 2018. Disponível em: <<https://mars.nasa.gov/mer/overview/>>. Acesso em : 02 maio 2018.

OLIVEIRA, J. R.; ABNER, A. S. S. e SILVA, R. M. **Reconhecimento de Objetos no Desenvolvimento de um Sistema de Navegação Inteligente para Robôs Móveis.** UERN, Mossoró, RN, 2015.

PAULEY, E; SHUMAKER, T. e COLE, B. **Preliminary report of investigation: Underground bituminous coal mine, non-injury mine inundation accident (entrapment), Black Wolf Coal Company.** Quecreek, Pennsylvania, 24 jul. 2002.

PROCESSING. *Reference. Processing was designed to be a flexible software sketchbook.* Processing, 2019. Disponível em: <<https://processing.org/reference/>>. Acesso em: 15 jul 2019.

RAMBO, W. **Vídeo: Robô com motor shield – Parte 2.** FILIPEFLOP, 26 jul 2016. Disponível em: <<https://www.filipeflop.com/blog/video-robo-com-motor-shield-parte-2/>>. Acesso em: 15 jul 2019.

RAMOS, D. C. e MORENO, U. F. **Aplicação de Técnicas de Fusão Sensorial para Mapeamento e Localização Simultâneos para Robôs Terrestres.** Dissertação de mestrado em engenharia de automação e sistemas, UFSC, 2012.

REDDY, B. R. *Wireless Technology Based Robot Communication System.* In: *International Journal of Engineering Research & Technology*, Vol. 3(10), ISSN: 2278-0181, 2017.

REIS, F. **Usando um Sensor Ultrassônico de Distância HC-SR04 com Arduino.** BÓSON TREINAMENTOS EM TECNOLOGIA, 25 fev. 2018. Disponível em: <<https://www.bosontreinamentos.com.br/electronica/arduino/usando-um-sensor-ultrassonico-hc-sr04-com-arduino/>>. Acesso em: 15 jul 2019.

ROMERO, R. A. F. *et al.* **Robótica Móvel.** LTC. 2014

ROOSE, K. *The Self-Driving Car Industry's Biggest Turning Point Yet,* *The New York Times*, 30 mar. 2018. Disponível em: <<https://www.nytimes.com/2018/03/30/technology/self-driving-cars.html>>. Acesso em: 06 maio 2018.

SIEGWART, R. *et al.* **Introduction to Autonomous Mobile Robots.** Segunda edição, MIT Press, 2011.

THE ROBOTICS BACK-END. *Arduino Uno Pins – A Complete Practical Guide.* *The Robotics Back-End.* Disponível em: <https://roboticsbackend.com/arduino-uno-pins-a-complete-practical-guide/#Arduino_Uno_pin_diagram>. Acesso em: 05 jun 2019.

THOMSEN, A. **Como conectar o Sensor Ultrassônico HC-SR04 ao Arduino.** FILIPEFLOP, 23 jul 2011. Disponível em: <<https://www.filipeflop.com/blog/sensor-ultrassonico-hc-sr04-ao-arduino/>>. Acesso em: 07 jun 2018.

THRUN, S. *Robotic Mapping: A Survey.* Sebastian Thrun, *School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213*, 22 fev. 2002. Disponível em: <robots.stanford.edu/papers/thrun.mapping-tr.pdf>. Acesso em: 07 maio 2018.

THRUN, S. *Learning metric-topological maps for indoor mobile robot navigation,* *Artificial Intelligence* 99(1), 21–71. 1998

THRUN, S. *et al.* ***A System for Volumetric Robotic Mapping of Abandoned Mines.*** *The Robotic Institute, Carnegie Mellon University, Pittsburgh, PA (EUA)*, 2003. Disponível em: <https://ri.cmu.edu/pub_files/...1/thrun_sebastian_2003_1.pdf>. Acesso em: 07 maio 2018.

YUKINOBU, A. H. **Mapeamento de ambientes externos utilizando robôs móveis.** Dissertação de mestrado em ciências de computação e matemática, USP- São Carlos. 21 Mar. 2010.

APÊNDICE I

Código do Arduino IDE.

```
#include <AFMotor.h>
AF_DCMotor motor1(4); //Seleção do Motor 1
AF_DCMotor motor2(3); //Seleção do Motor 2
#define trig    A4          //Saída para o pino de trigger do sensor
#define echo    A5          //Saída para o pino echo do sensor
int a;                    //Variavel qualquer para entrar em modo automático
int Auto;                //Variavel que determina se está em modo automático ou manual
float measureDistance(); //Função para medir, calcular e retornar a distância em cm
void trigPulse();        //Função que gera o pulso de trigger de 10µs
void decision();         //Função para tomada de decisão. Qual melhor caminho?
void ContarAnguloPos();
void robot_forward();    //Função para movimentar robô para frente
void robot_backward();   //Função para movimentar robô para trás
void robot_left();       //Função para movimentar robô para esquerda
void robot_right();      //Função para movimentar robô para direita
void robot_stop();       //Função para parar o robô
float dist_cm;           //Armazena a distância em centímetros entre o robô e o obstáculo
float angulo;            //Armazena angulo em radiandos do giro do robô
int MaxX;                //Armazena a distância X maxima no mapa em centímetros
int MaxY;                //Armazena a distância Y maxima no mapa em centímetros
int MaxPixel;           //Armazena a largura cm do mapa grafico considerando uma matriz quadrada
float contadorNew;       //Contador usado para giro durante o modo automático
float contadorX;         //Contador usado para calculo de distância X
float contadorY;         //Contador usado para calculo de distância Y
int contadorr;           //Contador usado para calculo da angulação em radianos
String mark;             //Marca o sentido do robô

void setup() {
  // put your setup code here, to run once:
  pinMode(trig, OUTPUT); //Saída para o pulso de trigger
  pinMode(echo, INPUT);  //Entrada para o pulso de echo
  digitalWrite(trig, LOW); //Pino de trigger inicia em low
  delay(500);
  Serial.begin(9600); // Starting the serial communication at 9600 baud rate
  // = 0xFF; //Inicia no valor máximo 0xFF
  pinMode(2, INPUT); // Primeiro pino de interrupção
  attachInterrupt(0, ContarAnguloPos, RISING); //
  angulo=0; //Angulo inicial
  contadorX=50; //Começa em uma posição X determinada no mapa
  contadorY=50; //Começa em uma posição Y determinada no mapa
  Auto=0; //Começa em modo manual (0) ou automático (5)
  a=1; // 1 para entrar no if usado no modo automatico
  MaxPixel=500; //Dimensão máxima do mapa da matriz em cm
  MaxX=MaxY=2*(MaxPixel-15);
}
void loop() {
  //robot_forward(); //Usado para teste
  //robot_left(); //Usado para teste
  //robot_backward(); //Usado para teste
  //robot_right(); //Usado para teste
  if (Serial.available ( ) > 0) { // Checking if the Processing IDE has send a value or not
    char state = Serial.read ( ); // Reading the data received and saving in the state variable
    if(state == '0')
    {
      Auto=0;
      robot_forward();
    }
  }
}
```

```

}
if (state == '1') {
  Auto=0;
  robot_backward();
}
if (state == '2') {
  Auto=0;
  robot_left();
}
if (state == '3') {
  Auto=0;
  robot_right();
}
if (state == '4') {
  Auto=0;
  robot_stop();
}
if (state == '5') {
  Auto=5;
}
}
dist_cm=measureDistance();
if(Auto==5){
  robot_forward();
  if((dist_cm<20) || (contadorX>=MaxX) || (contadorY>=MaxY) || (contadorX<=0) || (contadorY<=0) )
  decision();
}
//Serial.println((String)contadorX+","+contadorY+","+anguloA+",,,,,,,,,,,,,,,",+anguloB); // Usado para teste
// Serial.println((String)contadorX+","+contadorY+","+dist_cm+",,,,,,,,,,yyyy,,,,,,,,",+contadorr); // para teste
Serial.println((String)contadorX+","+contadorY+","+angulo+","+dist_cm); // Deve ir para porta serial
}

float measureDistance() //Função que retorna a distância em centímetros
{
  float pulse; //Armazena o valor de tempo em µs que o pino echo fica em nível alto
  float H; //Limitador do ultrassom;
  trigPulse(); //Envia pulso de 10µs para o pino de trigger do sensor
  pulse = pulseIn(echo, HIGH); //Mede o tempo em que echo fica em nível alto e armazena na variável pulse
  H= pulse/58.82; //Tempo que leva para a onda sonora ir e voltar
  if(H>=70) //Limitado um valor máximo de 70cm;
  H=70;
  return (H); //Retorna o valor
} //end measureDistant
void trigPulse() //Função para gerar o pulso de trigger para o sensor HC-SR04
{
  digitalWrite(trig,HIGH); //Saída de trigger em nível alto
  delayMicroseconds(10); //Por 10µs ...
  digitalWrite(trig,LOW); //Saída de trigger volta a nível baixo
} //end trigPulse
void decision()
{
  contadorNew=0;
  if (a==1) // Foi colocado para teste
  {
    do { robot_right();
      dist_cm = measureDistance();
      // Serial.println((String)contadorX+","+contadorY+","+dist_cm+",,,,,,,,,",+contadorr); // Usado para teste
      Serial.println((String)contadorX+","+contadorY+","+angulo+","+dist_cm); // Deve ir para porta serial, para
      que processing receba;
      // Serial.println((String)contadorX+","+anguloB+","+contadorNew+",,,,,,,,,,,,,,,",+contadorr); // Usado teste
    }
  }
}

```

```

    } while(contadorNew<16); //Vai pra direita enquanto o contador de giro não alcança 19
    robot_forward();
    } //end if
    } //end decision
void ContarAnguloPos() { // Função para valor de Angulo e Posição
    static unsigned long delayest; // delay falso para retornar um estado sem "tremida"
    if ( mark=="direita") {
    if (millis()-delayest>1){
    contadorr= contadorr+1;
    angulo=contadorr*3.1415/39;
    if(Auto==5)
    contadorNew=contadorNew+1;
    else
    contadorNew=0;
    if(angulo>=6.283)
    contadorr=0;
    }
    delayest=millis();
    } // end mark direita
    if (mark=="esquerda") {
    if (millis()-delayest>1){
    contadorr= contadorr-1;
    angulo=contadorr*3.1415/39;
    if(Auto==5)
    contadorNew=contadorNew+1;
    else
    contadorNew=0;
    if(angulo<=-6.283)
    contadorr=0;
    }
    delayest=millis();
    } // end mark esquerda
    if (mark=="re") {
    if (millis()-delayest>1){
    contadorX= contadorX-1*cos(angulo);
    contadorY= contadorY-1*sin(angulo);
    if(contadorX>MaxX)
    {robot_stop();
    contadorX=MaxX;}
    if(contadorX<0)
    {robot_stop();
    contadorX=0;}
    if(contadorY>MaxY)
    {robot_stop();
    contadorY=MaxY;}
    if(contadorY<0)
    {robot_stop();
    contadorY=0;}
    }
    delayest=millis();
    } // end mark re
    if (mark=="frente") {
    if (millis()-delayest>1){
    contadorX= contadorX+1*cos(angulo);
    contadorY= contadorY+1*sin(angulo);
    if(contadorX>MaxX)
    {robot_stop();
    contadorX=MaxX;}
    if(contadorX<0)
    {robot_stop();

```

```

contadorX=0;}
if(contadorY>MaxY)
{robot_stop();
contadorY=MaxY;}
if(contadorY<0)
{robot_stop();
contadorY=0;}
}
delayest=millis();
} //end mark frente
}
void robot_forward()
{ mark="frente";
  motor1.setSpeed(180);
  motor1.run(FORWARD);
  motor2.setSpeed(215);
  motor2.run(FORWARD);
} //end robot forward
void robot_backward()
{
  mark="re";
  motor1.setSpeed(180);
  motor1.run(BACKWARD);
  motor2.setSpeed(215);
  motor2.run(BACKWARD);
} //end robot backward
void robot_left()
{ mark="esquerda";
  motor1.setSpeed(130);
  motor1.run(BACKWARD);
  motor2.setSpeed(165);
  motor2.run(FORWARD);
} //end robot left
void robot_right()
{ mark="direita";
  motor1.setSpeed(130);
  motor1.run(FORWARD);
  motor2.setSpeed(165);
  motor2.run(BACKWARD);
} //end robot right
void robot_stop()
{ // sem mark
  motor1.setSpeed(0);
  motor1.run(RELEASE);
  motor2.setSpeed(0);
  motor2.run(RELEASE);
} //end robot stop

```

APÊNDICE II

Código do *Processing*.

```
import processing.serial.*; // Importing the serial library to communicate with the Arduino
Serial myPort; // Initializing a variable named 'myPort' for serial communication
int cor;
int cor1;
String arduino;
float x;
float y;
float miraX;
float miraY;
float estado;
float ultraX;
float ultraY;
float teta;

void setup ( ) {
  size (500, 500); // Size of the serial window, you can increase or decrease as you want
  background ( 1000 );
  myPort = new Serial (this, "COM9", 9600); // Set the com port and the baud rate according to the Arduino
  myPort.bufferUntil ( '\n' ); // Receiving the data from the Arduino IDE
}
void serialEvent (Serial myPort) {
  arduino = myPort.readStringUntil ( '\n' ); // Changing the background color according to received data
  float p[] = float(split(arduino, ','));
  x=p[0]*0.51836278784; //40*x=6,6pi // x=6,6pi/40 0.51836278784
  y=p[1]*0.51836278784;
  teta=p[2];
  estado=p[3];
  miraX=x+10*cos(teta);
  miraY=y+10*sin(teta);
  ultraX=miraX+estado*cos(teta);
  ultraY=miraY+estado*sin(teta);
}

void apagar()
{
  float tamanho;
  color white = color(255);
  color red = color(#ff0000);
  color black = color(0);
  color verde = color(#00ff00);
  color atual;
  tamanho = 1000;
  for (int i=0; i<tamanho; i++)
  {
    for (int j=0; j<tamanho; j++)
    {
      atual = get(i,j);
      if (atual == red) {set(i,j, white);}
      if (atual == verde) {set(i,j, black);}
    }
  }
}

void draw ( ) {
  apagar();
  smooth();
  fill(255);
```

```

stroke(#ff0000);
rectMode(CENTER);
rect(x,y,15,15);
if(estado<70)
{cor=0;
cor1=#00ff00;}
else
{cor=255;
cor1=#ff0000;}
fill(#ff0000); // preenchimento da mira
strokeWeight(2);
stroke(255);
line(miraX,miraY,ultraX,ultraY);
strokeWeight(1);
rectMode(CENTER);
rect(miraX,miraY,5,5);
fill(cor);
stroke(cor1);//
rectMode(CENTER);
rect(ultraX,ultraY,2,2);

print(x);
print("..");
print(y);
print("..");
println(estado);
//////////estado de click
if ( mousePressed && ( mouseButton == RIGHT ) ) { // if the right mouse button is pressed
myPort.write ( '5' ) ;    // Send a '0' to the Arduino IDE
}
}
void keyPressed(){
if(key==CODED){
if(keyCode ==UP){ myPort.write ( '0' ) ;
}
if(keyCode==DOWN){myPort.write ( '1' ) ;
}
if(keyCode==LEFT){myPort.write ( '2' ) ;
}
if(keyCode==RIGHT){myPort.write ( '3' ) ;
}
if(keyCode==ALT){myPort.write ( '4' ) ;
} } }

```