UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIAS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
GRADUAÇÃO EM ENGENHARIA ELÉTRICA

DANIEL VELOSO RIBEIRO

ROSELI: ROBÔ SEGUIDOR DE LINHA PARA MAPEAMENTO DE AMBIENTES INTERNOS

SÃO LUÍS
2017

DANIEL VELOSO RIBEIRO

ROSELI: ROBÔ SEGUIDOR DE LINHA PARA MAPEAMENTO DE AMBIENTES INTERNOS

Monografia apresentada ao Curso de Engenharia Elétrica da Universidade Federal do Maranhão como requisito para obtenção do grau de Bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Luciano Buonocore

SÃO LUÍS

2017

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Núcleo Integrado de Bibliotecas/UFMA

Ribeiro, Daniel Veloso.
ROSELI: ROBÔ SEGUIDOR DE LINHA PARA MAPEAMENTO DE AMBIENTES INTERNOS / Daniel Veloso Ribeiro. - 2017.
94 f.

Orientador(a): Luciano Buonocore.
Monografia (Graduação) - Curso de Engenharia Elétrica, Universidade Federal do Maranhão, São Luís, 2017.

1. Ambiente Interno Estruturado. 2. Multithreading. 3. OpenCV. 4. Raspberry Pi. 5. Robô seguidor de Linha. I. Buonocore, Luciano. II. Título.

DANIEL VELOSO RIBEIRO

ROSELI: ROBÔ SEGUIDOR DE LINHA PARA MAPEAMENTO DE AMBIENTES INTERNOS

Aprovada em 12/06/2017

BANCA EXAMINADORA

Prof. Dr. Luciano Buonocore (Orientador)
Departamento de Engenharia de Eletricidade - UFMA

Prof. Dr. Areolino de Almeida Neto (Membro)
Departamento de Informática - UFMA

Prof. Dr. André Borges Cavalcante (Membro)
Departamento de Engenharia de Eletricidade – UFMA

Prof. Dr. Denivaldo Cicero Pavão Lopes(Membro)
Departamento de Engenharia de Eletricidade - UFMA

AGRADECIMENTOS

Agradeço à minha família, principalmente meu pai e minha mãe, por todo amor, confiança e apoio ofertados, sempre aceitando minhas decisões e dando forças para continuar a perseguir o sucesso na minha carreira acadêmica. Serei eternamente grato.

Ao meu orientador Luciano Buonocore, pelo seu olhar crítico, opinião e percepção de cada detalhe, além de todos os esforços desprendidos para garantir que o LRC seja um ambiente propício para o desenvolvimento dos trabalhos propostos e de novos conhecimentos, muito obrigado.

Aos professores que contribuíram para a minha formação e aos amigos de curso, agradeço pelo apoio, ideias e críticas. Muito obrigado pelos momentos de estudos e de descontração.

Aos amigos do LRC, em especial Walber Lima e João Augusto, sempre incentivando e ajudando nas dificuldades que foram surgindo durante o desenvolvimento do trabalho.

Enfim, agradeço a todos que participaram direta ou indiretamente durante minha formação acadêmica.

RESUMO

Os avanços tecnológicos na produção de *hardware* e *software* nas últimas quatro décadas têm proporcionado um acentuado crescimento na automação de processos, que alcançam desde o setor industrial até o de entretenimento. Desde o primeiro processador em circuito integrado, lançado pela empresa Intel em novembro de 1971, as tecnologias de fabricação desses processadores e circuitos auxiliares vem proporcionando um acelerado aumento na capacidade de produção de sistemas computacionais. Uma linha especial dessa produção massificada é empregada em placas embarcadas e em dispositivos inteligentes usados em aplicação IoT (*Internet of Things*). A robótica móvel, articulada ou híbrida (robôs móveis manipuladores) é uma das áreas de pesquisa que vem se beneficiando desse acelerado avanço. Este trabalho de monografia aborda o tema de mapeamento de um ambiente interno formado por linhas, utilizando uma câmera simples para essa finalidade e, portanto, é um robô móvel considerado de baixo custo. Existem inúmeras aplicações reais onde existem este tipo de máquina, nos mais diversos segmentos econômicos com robôs usados em armazéns, hospitais e restaurantes. Uma plataforma móvel foi projetada e construída especificamente para esta finalidade, denominada RoSeLi (Robô Seguidor de Linha). A placa de controle utilizada foi o Raspberry Pi 3, que possibilitou a implementação de um ambiente *multithreading* para a solução da tarefa de mapeamento do ambiente, usando uma câmera comercializada para uso no Raspberry. A estruturação do ambiente foi feita através de *tags* que forneceram a pose do robô em relação a um sistema de coordenadas global. Foram feitos quatro testes de mapeamento, os quais construíram mapas muitos próximos do ambiente real, com erros menores que 1 cm nas coordenadas das poses do robô que, devido a estruturação do ambiente por *tags*, eliminou a acumulação de erros de odometria.

Palavras-chave: Robô seguidor de linha; Ambiente interno estruturado; *Multithreading*; Raspberry Pi 3 e OpenCV.

ABSTRACT

The technological advances in hardware and software production over the last four decades have made an accentuated growth in process automation possible, that reach from the industrial to the entertainment sector. Since the first integrated circuit processor, launched by Intel company in november 1971, the fabrication technology of these processors and auxiliary circuits have made the accelerated growth of production of these computational systems capacities possible. A special line of this massive production is applied in embedded boards and other smart devices used in IoT (Internet of Things). The mobile robotics, articulated or hybrid (mobile robots with manipulators) are one of the research areas that have been benefitted by this accelerated advance. This undergraduate thesis approaches the theme of mapping of an internal environment made by lines, using a simple camera for this purpose, and therefore, a mobile system considered of low cost. There are innumerous real applications where this kind of problem exists, in the more diverse economic segments with robots being used in storage houses, hospitals and restaurants. A mobile platform was designed and built specifically for this purpose, named RoSeLi (Line Follower Robot). The embedded system used was the Raspberry Pi 3 board, that made possible the implementation of a multithreading environment for the solution of mapping the environment, using the commercialized Raspberry Pi camera. The environment structuration was made through tags that provide the robot pose in relation to a global coordinate system. There were made four mapping tests, which built maps very close to the real environment, with error smaller than 1 cm in the coordinates of the robot poses that, due to the environment structuration by tags, eliminated the accumulation in odometry error.

Keywords: Mobile Robot; Internal structured environment; Multithreading; Raspberry Pi 3; OpenCV.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

IoT	*Internet of Things*
AURESIDE	Associação Brasileira de Automação Residencial e Predial
VANT	Veículo Autônomo Não Tripulado
RoSeLi	Robô Seguidor de Linha
SLAM	*Simultaneous Localization and Mapping*
LIDAR	*Light Detection and Ranging*
DSPs	*Digital Signal Processors*
OpenCV	*Open Source Computational Vision*
GPU	*Graphic Processor Unit*
PWM	*Pulse Width Modulation*
GPIO	*General Purpose Input Output*
LED	*Light Emitter Diode*
I2C	*Inter Integrated Circuit*
DAS	*Serial Data*
SCL	*Serial Clock*
ACK	*Acknowledgement*
NACK	*Negative-Acknowledgement*
SPI	*Serial Peripheral Interface*
UART	*Universal Asynchronous Receiver Transmitter*
MISO	*Master In Slave Out*
SCLK	*Serial Clock*
SS	*Slave Select*
MOSI	*Master Out Slave In*
CPOL	*Clock Polarity*
CPHA	*Clock Phase*
A/D	Analógico/Digital
SSH	*Secure Shell*
VNC	*Virtual Network Computting*
RFB	*Remote Framebuffer*
HDMI	*High-Definition Multimedia Interface*
CEC	*Consumer Electronics Control*

DPI	*Display Parallel Interface*
DSI	*Display Serial Interface*
DBI	*Display Bus Interface*
API	*Application Programming Interface*
FPS	*Frame per Second*
LRC	Laboratório de Robótica Móvel e Comunicação Sem Fio
DC	*Direct Current*
LiPo	Lítio Polímero
OCR	*Optical Character Recognition*
PCI	Placa de Circuito Impresso
RGB	*Red Green Blue*
HSV	*Hue Saturation Value*
ROS	*Robot Operating System*

SUMÁRIO

1. INTRODUÇÃO

Na sociedade atual, os meios de produção industrial, de serviço e de entretenimento tiveram um aumento bastante significativo na automação das tarefas necessárias em sua produção, em relação às duas décadas passadas. Vários fatores contribuíram para esse expressivo crescimento, entre eles:

a) O avanço tecnológico de produtos de *hardwares*, com uma consequente diminuição dos custos em virtude da massificação da demanda;
b) O aumento de ofertas de *softwares* para atender às mais variadas necessidades de automação existentes no mercado;
c) A introdução dos produtos IoT (*Internet of Things*) que permitem o intercâmbio de informações entre dispositivos inteligentes em residências, fomentando um grande aumento na demanda por equipamentos (*hardware* e *software*); e
d) O uso cada vez mais intenso de soluções robotizadas na execução das tarefas nos mais diferentes setores da economia.

Segundo a AURESIDE (Associação Brasileira de Automação Residencial e Predial): "Soluções de automação estão evoluindo constantemente nos dias de hoje em termos de eficiência, capacidade e desempenho geral". Uma pesquisa realizada no ano de 2015 em seis países, incluindo o Brasil, mostrou já um número significativo em automação residencial, sendo que no Brasil o potencial chega a ser mais de seis vezes o real instalado[1].

Nos fatores citados, a utilização de robôs nas soluções das tarefas industriais, de serviços e de entretenimentos é certamente um dos ramos mais expressivos que promovem um aumento significativo dos processos automatizados. Aproximadamente 85 mil robôs são inseridos nas industrias em todo mundo segundo dados divulgado pelas Organização das Nações Unidas (ONU), sendo motivado principalmente pela maximização da produção, chegando a ser quadruplicada em alguns segmentos[2].

Desde a década de 70 já havia pesquisa na área da robótica, inicialmente mais voltada ao uso de sistemas articulados, principalmente os utilizados em sistemas de manufatura. Os robôs móveis começaram a ganhar espaço na década seguinte, quando foram intensificadas

[1] Site: http://www.aureside.org.br/_pdf/potencial_2015.pdf
[2] Site: http://mundoeducacao.bol.uol.com.br/geografia/a-robotizacao-na-producao-industrial.htm

as pesquisas nas mais diversas tarefas realizadas pelas plataformas móveis. É muito comum a aplicação de equipamentos híbridos com capacidade de articulação e de mobilidade.

Atualmente, os veículos móveis e autônomos já são realidade em nosso cotidiano, tanto para navegação aérea, terrestre e subaquática. Esses veículos em algumas situações são teleoperados, ou seja, sua atuação na tarefa demanda o manuseio humano à distância. Entretanto, está se tornando cada vez mais comum a oferta de robôs autônomos, onde o ser humano tem pouca ou nenhuma intervenção no sistema. Como exemplo de sistemas autônomos tem-se os carros autoguiados, onde toda a sua operação está inserida nos veículos. Como exemplo para esta situação, o usuário de um sistema de taxi autônomo pode fazer a chamada através de um aparelho portátil (*tablet* ou celular), aguardar a chegada e o estacionamento do veículo sem motorista no local indicado pelo sistema GPS (*Global Position System*) e embarcar para que o veículo o leve ao local informado.

Serviços de vigilância ou de espionagem por VANT (Veículo Autônomo Não Tripulado) são exemplos de robôs móveis aéreos. Para ambientes subaquáticos tem-se a presença de veículos para tarefas como a de pesquisa submarina, na maioria teleoperados.

Um campo bastante pesquisado é o uso de robôs em ambientes internos (indústrias, restaurantes, hospitais, etc.). A realização de tarefas por essas máquinas pode dar-se em ambientes sem nenhuma estruturação, onde o local é totalmente desconhecido do robô ou em ambiente com um determinado grau de estruturação que servem de referências para o veículo na realização das tarefas atribuídas.

Nesse espectro de ambientes estruturados situam-se aqueles em que os robôs se movem guiados por linhas, visíveis ou invisíveis ao olho humano, localizados no piso, parede (marcos de referências) ou teto. Existem diversas aplicações reais para tais robôs, conhecidos como seguidor de linha.

1.1. Objetivos

O objetivo geral neste trabalho de monografia foi resolver o problema de mapeamento de ambientes internos estruturados, usando um robô de baixo custo seguidor de linha, denominado RoSeLi.

Os objetivos específicos relacionam-se às tarefas executadas pelo robô que possibilitaram a execução da sua tarefa principal de mapeamento do ambiente:

a) Implementação de algoritmo capaz de reconhecer e se guiar através de linhas pretas colocadas em fundo branco;
b) Reconhecer e extrair as informações contidas nas *tags* (poses do robô em relação ao *frame* cartesiano global no plano 2D) espalhadas pelo ambiente;
c) Capacidade de detecção de objetos que impeçam o movimento do robô na linha, possibilitando a interação com o usuário de modo a prosseguir com o mapeamento com a retirada do obstáculo; e

1.2. Motivação

Existem diversos setores da indústria, do comércio e mesmo do entretenimento que se beneficiam da implementação de robôs seguidores de linhas na solução de seus problemas. Por exemplo, na indústria se pode citar o uso dessas plataformas móveis em armazéns totalmente ou parcialmente automatizados, em hospitais como assistentes para transportes de material humano (amostras de sangue, urina, etc.) ou serviços gerais (alimentos, vestuários, etc.). No setor comercial é possível o emprego de tais sistemas em restaurantes, atendendo a pedidos de clientes, levando à mesa as comidas e bebidas encomendadas e trazendo as louças após as finalizações das refeições.

Quanto a área de entretenimento sua aplicação é ilimitada, a depender da imaginação dos desenvolvedores, que podem optar por soluções simuladas ou reais. No entretenimento simulado, pode-se citar jogos onde as tarefas realizadas por robôs, quaisquer que sejam elas, devem obedecer às demarcações das linhas no ambiente virtual, como jogos de saídas de labirintos. Para a aplicação em entretenimento com robôs reais têm-se os casos de gincanas, cujos trajetos de realizações das tarefas sejam restritos às linhas que estruturam os possíveis caminhos das disputas.

Essa gama de possíveis aplicações nas mais diversas áreas econômicas faz do problema tratado neste trabalho de monografia um tema atrativo de pesquisa. Ainda que limitado a um ambiente interno de pequenas proporções, a ideia é expansível a ambientes de mesmas características com áreas muito maiores.

1.3. Organização do trabalho

Este trabalho de monografia está divido em sete capítulos, incluindo este de introdução do tema. No Capítulo 2 é feita a fundamentação teórica, abordando os diversos temas que estiveram relacionados à solução ao problema do robô seguidor de linha em um ambiente interno e estruturado. O Capítulo 3 apresenta todas as especificações relativas ao projeto do robô, das três partes que o compõem: a mecânica, a eletrônica e a computacional. O Capítulo 4 detalha a implementação da solução proposta, fazendo na parte computacional a solução apenas em alto nível. O Capítulo 5 detalha a parte computacional embarcada no robô na solução das diversas operações realizadas pelo robô RoSeLi no mapeamento do ambiente. O Capítulo 6 apresenta os experimentos e os resultados obtidos nos diversos testes realizados com o objetivo final de mapear o ambiente. Finalmente, no Capítulo 7 são feitas as conclusões a respeito dos resultados obtidos para a solução ao problema de mapeamento proposto, sugerindo possíveis trabalhos futuros.

2. EMBASAMENTO TEÓRICO

Robótica é ciência de perceber e manipular o mundo físico através de dispositivos eletrônicos e eletromecânicos controlados por computador. Sendo um campo de estudo relativamente novo, apresenta um crescimento significativamente rápido desde a década de 1970, utilizando algoritmos determinísticos. Nesta fase, os modelos usados no movimento do robô e na aquisição de medidas do ambiente eram considerados precisos, sem a presença de incertezas. As construções computacionais de tais modelos requeriam um grande esforço de incorporar todas as variáveis significativas com o objetivo de minimizar as incertezas que surgiam com a movimentação do robô. Já na década de 80, passou a ser adotada de forma maciça a abordagem probabilística na construção dos modelos e algoritmos de execução de tarefas em robótica, facilitando a construção e a computação das incertezas (THRUN, 2005).

Os robôs podem ser classificados de acordo com critérios variados. Uma forma define questões como: autonomia do sistema de controle, estrutura cinemática, forma de acionamento, graus de liberdade, mobilidade, etc, conforme estrutura classificatória apresentada na Tabela 1.

Tabela 1 – classificação de robôs.

Sistema de controle	Mobilidade da base	Estrutura Cinemática	Espaço de trabalho
Equipamentos teleoperados	Veículo teleoperado		
	Manipulador teleop.		
Robôs	Móveis	Aquáticos	Submarinos
			Marinos
		Aéreos	
		Terrestres	Pernas
			Rodas
	Fixos	Paralelos	3 - 6 GdL
		Série	Cartesiano
			Cilíndrico
			Esférico
			Articulado
			SCARA

Fonte: http://pgene.ufabc.edu.br/docentes/Riascos/ensino/disciplinas/Robotica/FundamentosRobotica

Com o avanço da tecnologia em *hardware* e *software*, os robôs tornaram-se ferramentas muito mais comuns e acessíveis nas últimas décadas. As técnicas e os recursos disponíveis para o tratamento de problemas nesta área se desenvolveram e se propagaram pela comunidade. Neste capítulo serão tratados os fundamentos de algumas das tarefas e de ferramentas utilizadas para resolvê-las, como:

a) tarefas comuns em Robótica Móvel;

b) estruturação de ambientes;

c) sistemas embarcados em robôs;

d) a biblioteca OpenCV usada no tratamento de imagem e na aprendizagem de máquinas;

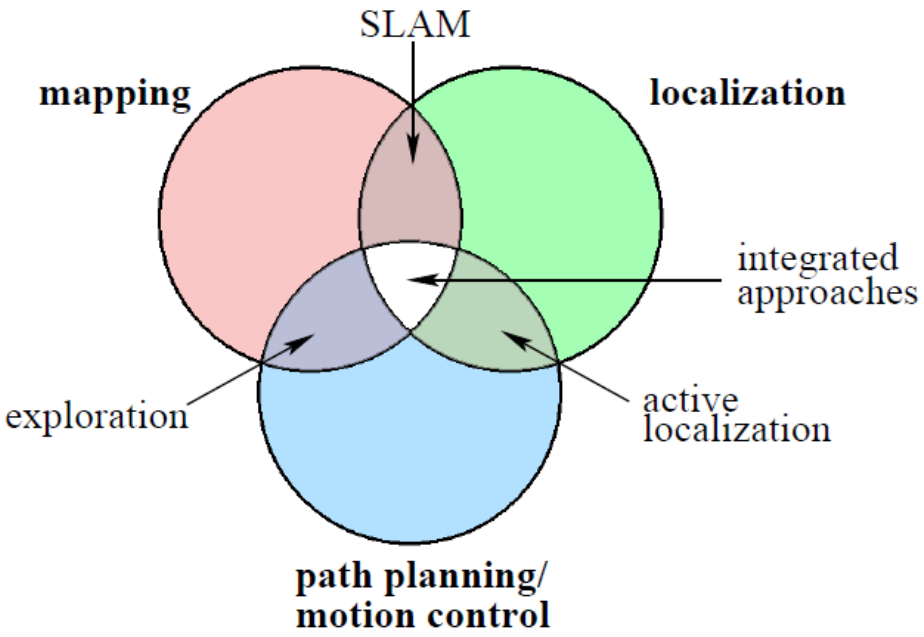
2.1. Tarefas comuns em Robótica Móvel

Quando seus sistemas de controle possuem autonomia (sem intervenção humana nas execuções das tarefas) são chamados de robôs; do contrário são dispositivos teleoperados. Além disso, se possuem cadeia cinemática fechada são chamados de manipuladores paralelos e quando as cadeias são abertas são denominados seriais. Os sistemas robóticos podem ser classificados, ainda, de acordo com a quantidade de graus de liberdade que possuem. Se este número for igual a seis são considerados de propósito geral, já que possuem articulações suficientes para realizar a maior parte das tarefas. Se possuírem mais que seis graus de liberdade, são considerados redundantes, já que possuem mais articulações que o necessário para a maioria das aplicações. Se o número de articulações for menor que seis, são considerados limitados, pois não possuem liberdade o suficiente para realizar certas aplicações[3].

A robótica móvel é um subconjunto de estudo da robótica, que pesquisa robôs capazes de se mover em um ambiente. Para a maior parte das aplicações desta área, existem tarefas que são recorrentes e que, se integradas, podem solucionar problemas maiores. Estas tarefas são: mapeamento, localização e planejamento de trajetórias, conforme ilustra a Figura 1 (STACHNISS, 2006).

[3] Site: http://graduacao.ufabc.edu.br/eiar/conteudo/ensino/disciplinas/Robotica/FundamentosRobotica.html

Figura 1 − Tarefas em robótica móvel.



Fonte: (STACHNISS, 2006).

A tarefa de mapeamento é o problema de integrar as informações adquiridas pelos sensores do robô, construindo um mapa que representa o espaço físico por onde navegou robô. Nesta tarefa as poses do robô (formado num plano pelas coordenadas <x, y> e orientação θ) são conhecidas. Para realizar a localização, o robô possui *a priori* o mapa do ambiente, e deve então estimar as suas poses durante a navegação pelo ambiente, baseando-se nas informações fornecidas por seus sensores. Já o planejamento de trajetórias é o que permite ao robô o movimento entre duas poses adjacentes (BUONOCORE, 2013).

Sendo o fundamento da maioria das aplicações em robótica móvel, é difícil realizar estas tarefas individualmente de forma prática. Quando se deseja conhecer o ambiente (mapeamento), baseando-se nas informações de seus sensores, o robô deve saber de que pose estas foram adquiridas. Da mesma forma, quando se deseja que a plataforma móvel saiba sua pose ou se movimente entre duas poses (localização), é necessário possuir dados do ambiente. Assim, ao executar duas ou mais destas tarefas, é possível resolver problemas mais complexos, como: exploração, localização ativa e SLAM (*Simultaneous Localization and Mapping*) (STACHNISS, 2006).

A exploração foca em guiar o robô de forma eficiente através do ambiente, com o intuito de construir um mapa que representa o espaço percorrido. Na localização ativa o robô

decide seu caminho para transcorrer entre dois locais do ambiente que navega com intuito de minimizar as incertezas de suas poses durante o trajeto efetuado. Já o SLAM é o problema de construir simultaneamente o mapa do ambiente enquanto o robô se localiza, podendo ser autônomo ou não. Quando existe autonomia, todas as três tarefas ilustradas na Figura 1 são feitas em conjunto (BUONOCORE, 2013).

2.2. Estruturação de ambientes

Na robótica, o ambiente em que a plataforma móvel está localizada desempenha um papel fundamental na realização de suas tarefas. Isso se deve ao fato de que, na maioria dos casos, o algoritmo deve "prever" as situações que podem ocorrer neste local para tomar as decisões mais adequadas. Em ambientes denominados estruturados, os objetos de referência à navegabilidade do robô não mudam de lugar e podem existir marcações que ajudam o robô em seus movimentos. Ambientes com certa estruturação podem ser encontrados em laboratórios (Figura 2), armazéns ou linhas de montagens. Assim, a complexidade do *software* existente no robô para esses tipos de ambientes é muito menor em tais circunstâncias[4].

Por outro lado, em ambientes não estruturados (Figura 3), as informações de referência para a circulação do robô inexistem e as tarefas que ele deve realizar tornam-se mais complexas de serem implementadas computacionalmente, como os casos da solução ao problema SLAM em ambientes internos (BUONOCORE, 2013) ou, no extremo, como na exploração de outro planeta[5]. Tais ambientes são muito mais comuns que os estruturados e é uma área atual de grande interesse para a pesquisa em robótica [6].

Na realidade, a maioria dos ambientes não se representam diretamente nos extremos, no sentido de que não são completamente estruturados ou desestruturados, tendo um certo grau de estruturação. Esta característica pode ser utilizada como forma de facilitar a resolução dos problemas propostos.

[4] Site: https://stanleyinnovation.com/robot-navigation-importance-of-environment/
[5] Site: https://www.nasa.gov/mission_pages/msl/overview/index.html
[6] Site: http://www.robotics.tu-berlin.de/fileadmin/fg170/publikationen_pdf/katz-08b.pdf

Figura 2 – Laboratório de robótica

Fonte: Site da Technische Universiteit Eindhoven.

Figura 3 – Robô guia em aeroporto

Fonte: http://www.spencer.eu/

2.3. Robô seguidor de linha

Em ambientes estruturados, é possível que existam marcos, que servem de guias para o robô utilizar em sua navegação, com o intuito de realizar suas tarefas. Se tais marcações forem linhas, visíveis ou não aos olhos humanos, que delimitam o caminho em que o robô pode navegar, este tipo de robô é chamado de seguidor de linha. Existem diversas variações de robôs seguidores de linha, desde os mais simples, cuja tarefa única é se mover pelas guias, até os mais complexos que as utilizam para alcançar outro objetivo, como o robô montador de *pallets* da Figura 4 (BAJESTANI, 2010).

Figura 4 – Robô seguidor de linhas

Fonte: http://www.hizook.com/blog/2014/07/08/logimover-clever-approach-pallet-moving-using-distributed-robots.

A fim de detectar as linhas marcadas no chão tais robôs fazem uso dos mais variados tipos de sensores, dos ópticos às câmeras, estes últimos requerendo tratamento de imagem. Ao utilizar a câmera, é possível fazer algoritmos que não seriam possíveis com um sensor mais simples, como demarcar caminhos por cores ou outro tipo de característica. Quando as linhas são demarcadas de forma invisível, é necessário utilizar um sensor adequado, geralmente um infravermelho (IR). Assim, a escolha dos componentes utilizados para alcançar o objetivo, no caso a detecção de linhas, depende das aplicações e da forma como se deseja realizá-las (PAKDAMAN, 2009).

Um dos maiores desafios para robôs desse tipo é o planejamento de trajetórias. Já que os mais simples não possuem capacidade de diferenciar vários ramos de um mesmo caminho, devido ao uso de sensores de baixo custo, é relativamente difícil implementar esta funcionalidade. Nos casos em que o sensor escolhido é capaz de identificar marcações que

diferenciam as rotas, esta tarefa é facilitada. No entanto, em aplicações reais as linhas podem ser bloqueadas por pessoas, objetos ou outras situações não previstas. Nestes casos, uma rota pode ser redefinida, com o auxílio de sensores extras, que detectam os bloqueios no caminho (ENGIN, 2012).

2.4. Sistemas embarcados em robôs

Conforme definição de KAMAL em 2011: "Um sistema embarcado é aquele que possui *hardware* computacional com *software* embarcado como um de seus principais componentes". Estendendo essa definição à robótica, percebe-se que a presença de um sistema embarcado é necessária na maioria dos robôs, para que estes possam desempenhar as funções desejadas. É necessária uma unidade de processamento de programas, capaz de controlar os seus atuadores e processar as informações adquiridas por seus sensores (KAMAL, 2011).

O processador é a unidade central de qualquer sistema embarcado. Para máquinas de uso geral, como em computadores, existe uma pequena quantidade de processadores diferentes que são utilizados, geralmente com mais de um núcleo e alto poder de processamento. No entanto em sistemas embarcados existe uma maior variedade, com as mais diversas características físicas e técnicas (SESHIA, 2012).

Entre os tipos de unidades de processamento mais utilizados em sistemas embarcados robóticos, estão os microcontroladores. Estes *chips* contém a unidade de processamento, geralmente *single-core*, e todos os periféricos necessários para o seu funcionamento, como memórias, gerenciadores de I/O, temporizadores e outros. Normalmente possuem baixíssimo consumo de energia, contando com modo *sleep*, que aumenta a sua eficiência energética. São disponibilizados em diversos tamanhos, sendo que é possível encontrar microcontroladores[7] no mercado com área menor que 2 mm^2. Assim, este tipo de unidade de processamento é ideal para robôs em que é necessário baixo consumo de energia e tamanho reduzido (SESHIA, 2012).

Em alguns robôs, existe uma grande carga de processamento de sinais, como tratamento de informações em aplicações multi-robôs ou quando a plataforma móvel tem que lidar com as incertezas do ambiente. Alguns sensores como *scanners* a laser, classificado como

[7] Site: http://www.atmel.com/products/microcontrollers/avr/tinyavr.aspx

LIDARs (*Light Detection and Ranging*), usados na medição de distâncias, exigem que o processador tenha alta capacidade de processamento de sinais, uma vez que recebe uma massa grande de informações por segundo para computar. Um sinal é uma coleção de medidas amostradas do mundo físico. Existem unidades de processamento próprias para lidar com sinais numericamente intensivos, como aplicações em MV (*Machine Vision*), chamados DSPs (*Digital Signal Processors*) (SESHIA, 2012).

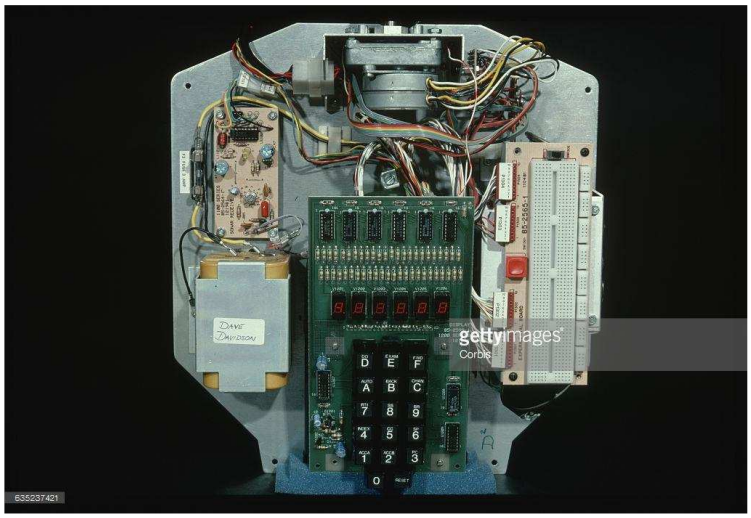
Com a disseminação de bibliotecas de tratamento de imagens, como o OpenCV (*Open Source Computational Vision*), robôs que utilizam câmeras como sensores são cada vez mais comuns. Nestes casos, é importante que o sistema embarcado possua uma GPU (*Graphic Processor Unit*), que se trata de um processador especializado que opera em paralelo ao principal, capaz de realizar esta tarefa com muito mais eficiência[8].

Atualmente é possível encontrar placas, processadas ou microcontroladas, que possuem vários recursos que facilitam o desenvolvimento de um sistema embarcado. Os recursos mais comuns são: a) acesso fácil as entradas e saídas através de *headers* (pinos de conexão), sendo esses pinos usados para implementação de barramentos específicos (SPI (*Serial Peripheral Interface*), UART (*Universal Asynchronous Receiver Transmitter*), I2C (*Inter-Integrated Circuit*) e outros); b) entradas analógicas; c) saídas PWM (*Pulse Width Modulation*); e d) GPIO (*General Purpose Input Output*). Essas características variam entre as placas, assim como a capacidade de processamento e outras especificações técnicas[9].

A aplicação em que o robô será utilizado tem um impacto direto na complexidade do sistema embarcado que será requerido para ele (Figura 5). Quando uma grande quantidade

[8] Site: https://www.ncbi.nlm.nih.gov/pubmed/24027619
[9] Site: http://hackaday.com/2011/02/01/what-development-board-to-use/

Figura 5 – Sistema embarcado do robô Hero 1



Fonte: http://www.gettyimages.com/detail/news-photo/electronic-circuitry-inside-the-hero-1-a-kit-robot-from-the-news-photo/635237421#electronic-circuitry-inside-the-hero-1-a-kit-robot-from-the-heath-picture-id635237421.

de dados é necessária, deverão ser utilizados sensores capazes de fazer maior quantidade de medidas ou grande quantidade de sensores simples.

Além disso, o ambiente onde a máquina irá movimentar-se, influência diretamente os tipos de atuadores que serão utilizados. Assim, diversas características e possibilidades devem ser levadas em consideração ao projetar um sistema robótico (SESHIA, 2012).

Os sensores são parte crucial do sistema. É através deles que o robô adquire informações do ambiente para realizar suas atividades. Existem diversos tipos de sensores disponíveis comercialmente, cada um deles possui características especificas e devem ser usados de acordo com a aplicação desejadas. Na tabela 2, é feita uma descrição de vários tipos de sensores, demonstrando suas aplicações e impactos de processamento no sistema de controle (YAP, 2009).

Tabela 2 – comparação entre sensores.

Sensor	Medida	Impacto no sistema
Infravermelho	Distância	Baixo
Sonar	Distância	Baixo
Par de câmeras	Distância	Médio
Laser Scanner	Distância	Alto

Os atuadores são os elementos do sistema robótico que lhe garantem seus graus de liberdade, ou seja, permitem que este interaja com o ambiente. Existem basicamente dois tipos de atuadores, os lineares e não lineares. Entre os lineares estão os motores de corrente contínua,

servos, motores de passo, etc. Já os não lineares são compostos por dispositivos não girantes, como cilindros pneumáticos e hidráulicos, solenoides, jugo escocês, entre outros[10].

2.5. Recursos de *hardware* empregados na implementação de Sistemas Embarcados

Os sistemas embarcados estão presentes em todos as áreas do mundo moderno, de aparelhos celulares aos sistemas de controle embutidos em veículos, nas mais diversas aplicações. Com tamanha gama de aplicações, é natural imaginar que são necessários diversos recursos diferentes, sendo as possibilidades para recursos de *hardware* serão detalhadas nesta seção.

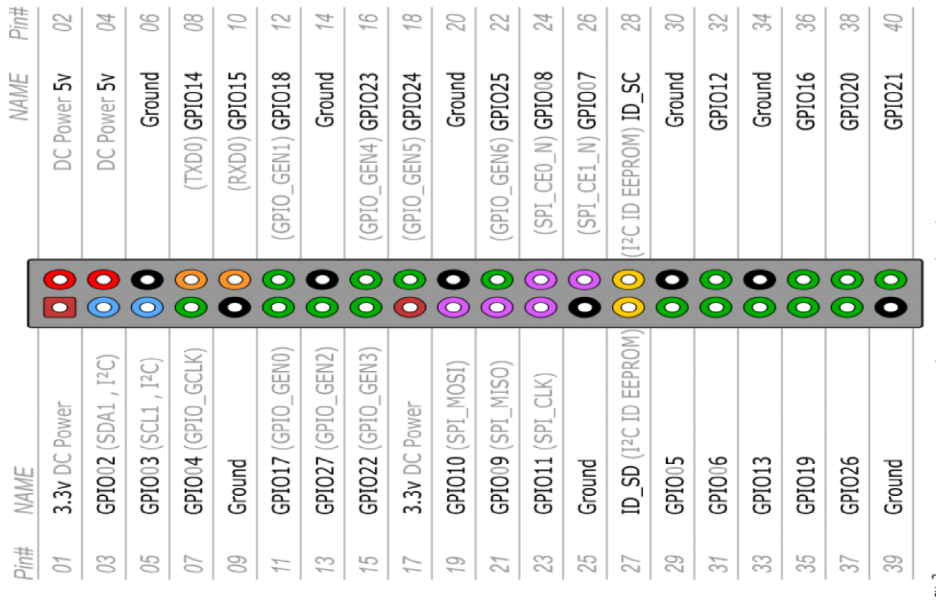
2.5.1. Entradas e saídas de propósito geral

Para muitas aplicações na robótica é necessário enviar ou ler um sinal digital, seja para ligar um LED (*Light Emitter Diode*), ler um botão ou sensor. Nestes casos, o processador deve possuir pinos de entradas e saídas de propósito geral (GPIO). Estes pinos são interfaces que permitem a personalização do sistema, permitindo ao programador o controle do *hardware*. Portanto, é interessante que a quantidade destes pinos seja suficiente para a aplicação desejada (BALACHANDRAN, 2009).

Além das aplicações padrões para estes pinos, geralmente eles podem ter uma segunda configuração. No computador portátil Raspberry Pi 3, por exemplo, todos os 40 pinos do *header* (com exceção daqueles de alimentação e terra), podem ser configurados como GPIO, mesmo os que possuem alguma função especifica, como pode ser visto na Figura 6.

[10] Site: https://www.robots.com/education/actuators

Figura 6 – Identificação de pinos do *header* no Raspberry Pi 3



Fonte: https://www.myelectronicslab.com/tutorial/raspberry-pi-3-gpio-model-b-block-pinout/

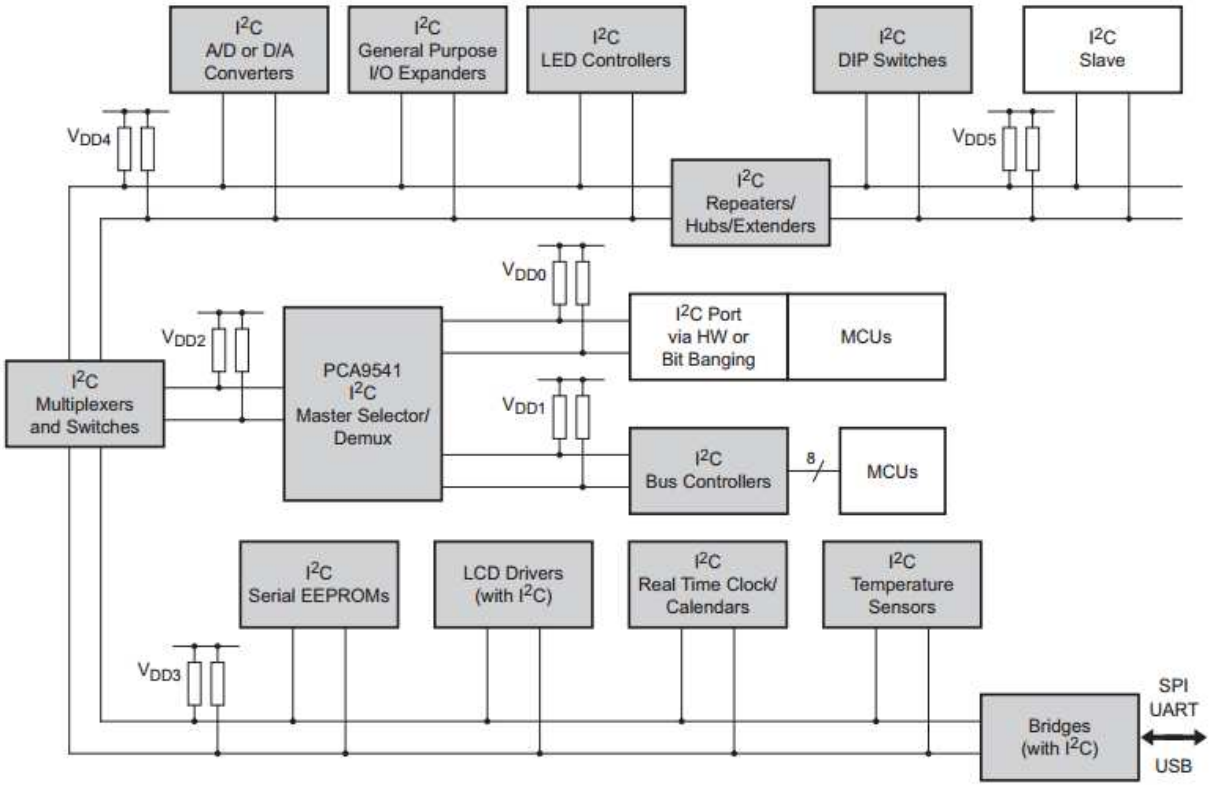
2.5.2. Entradas e saídas especificas

Alguns periféricos recebem e enviam informações através de protocolos específicos, exigindo pinos apropriados da placa do sistema. Existem vários recursos que podem ser utilizados em periféricos na robótica, como I2C, SPI e entradas analógicas. Nem sempre todos esses recursos são encontrados em apenas um sistema e, nestes casos, existem placas que fazem a interface entre o periférico e o processador, garantindo a comunicação necessária.

O protocolo I2C é um barramento serial, síncrono e *half-duplex* (tem capacidade de enviar e receber informações, mas não ao mesmo tempo), de simples implementação e muito difundido nos dispositivos sensores e atuadores. Precisa de apenas duas linhas de comunicação, a SDA (*Serial Data*) e a SCL (*Serial Clock*), ambas na configuração de coletor aberto. Cada dispositivo conectado ao barramento é endereçável via *software* e a comunicação é feita através do sistema mestre-escravo, onde o mestre controla o barramento (gerencia o sinal SCL), enviando dados para o escravo ou solicitando informações deste. Em um mesmo barramento podem ser conectados até 127 escravos, com a ajuda de buffers e repetidores. Um exemplo de

ligação pode ser visto na Figura 7 (I²C MANUAL E ESPECIFICAÇÕES DE USO, 2014).

Figura 7 – Esquema de ligação de dispositivos no barramento I2C

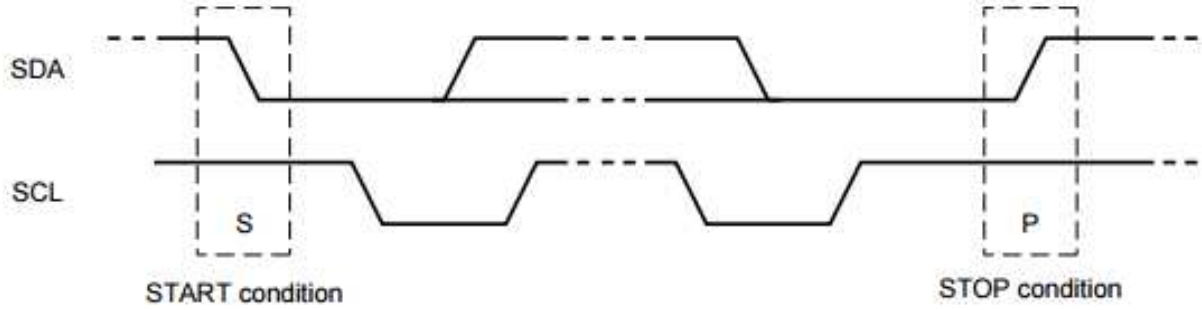


Fonte: Manual e guia de usuário I²C

Toda a comunicação é feita através das duas linhas citadas anteriormente, as quais devem ser conectadas ao positivo da fonte através de um resistor *pull-up* para garantir a definição do nível alto. A sinalização de que o barramento está livre (nenhum dispositivo mestre o está controlando) é feita colocando os dois pinos em nível alto. Todas as transações de dados começam com um START e terminam com um STOP (Figura 8). Uma condição de START é definida através de uma transição de nível alto para baixo no SDA, enquanto o SCL está em nível alto. A ocorrência do STOP acontece quando há uma transição de nível baixo para alto no SDA, enquanto o SCL está em nível alto (I²C MANUAL E ESPECIFICAÇÕES DE USO, 2014).

A comunicação através da linha de dados é feita com 8 bits enviados serialmente por vez, sendo o mais significativo do byte transmitido primeiro. Após o envio de cada byte, deve ser recebido um bit de reconhecimento ACK (*Acknowledgement*), que permite ao

Figura 8 – Condição de START e STOP no barramento I2C

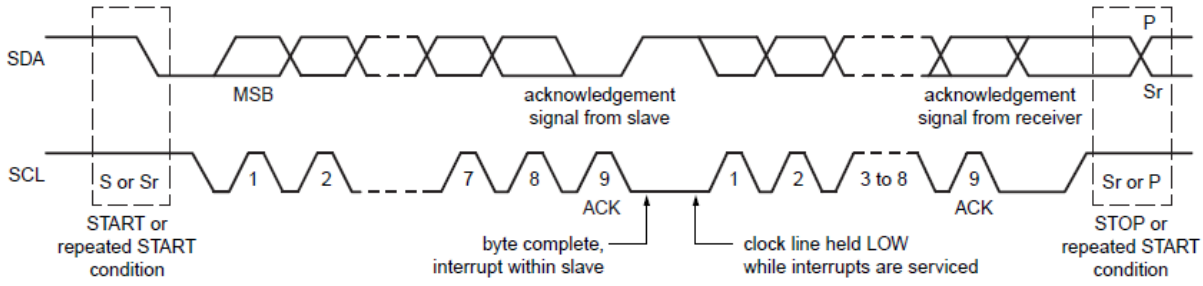


Fonte: Manual e guia de usuário I2C

dispositivo sinalizar que o byte foi recebido adequadamente. A quantidade de bytes que podem ser enviados por acesso é irrestrita, desde que após cada um deles exista o ACK, conforme mostra a Figura 9. Quando existe algum problema na transação dos dados, é gerado um bit de NACK (*Negative-Acknowledgement*). Essa situação ocorre quando: a) não existe um receptor no barramento com o endereço desejado; b) o receptor não pode receber o dado, pois está ocupado e não está pronto para estabelecer comunicação com o mestre; c) durante a transmissão o receptor recebe dados ou comandos que não entende; d) durante a transmissão o receptor não pode receber mais nenhum dado; e) um mestre-receptor precisa sinalizar o fim da transmissão com um escravo transmissor (I^2C MANUAL E ESPECIFICAÇÕES DE USO, 2014).

Originalmente, o barramento I2C comunicava-se com uma taxa única de 100 kbps (quilo bits por segundo ou 100.000 bps), que satisfazia todas as necessidades de velocidade da época de sua criação. No entanto, com o avanço da tecnologia e a demanda por barramentos mais velozes,

Figura 9 – Transferência de dados no barramento I2C



Fonte: Manual e guia de usuário I^2C

foram criados outros dois modos de operação. O *Fast-mode* opera com uma taxa de transmissão de 400 kbps, o *Fast-mode Plus* opera em 1 Mbps e o *High-speed mode* chega até 3,4 Mbps,

adequando o barramento as demandas modernas (I²C MANUAL E ESPECIFICAÇÕES DE USO, 2014).

O protocolo SPI é um barramento serial síncrono, também do tipo de conexão mestre-escravo. No entanto, necessita de quatro linhas de comunicações: MISO (*Master In Salve Out*), MOSI (*Master Out Slave In*), SCLK (*Serial Clock*) e SS (*Slave Select*). Dessa forma, sua implementação tem maior custo de *hardware* (mais pinos) que o protocolo I2C, porém por ter duas linhas de dados (MISO e MOSI) é possível enviar e receber dados com o escravo simultaneamente, possibilitando a comunicação em *full-duplex*. A quantidade de escravos ligados depende da quantidade de pinos SS (seleção do escravo) que o mestre aciona, já que não há endereçamento pela linha de dado, como no I2C, sendo os escravos identificados fisicamente por pinos[11].

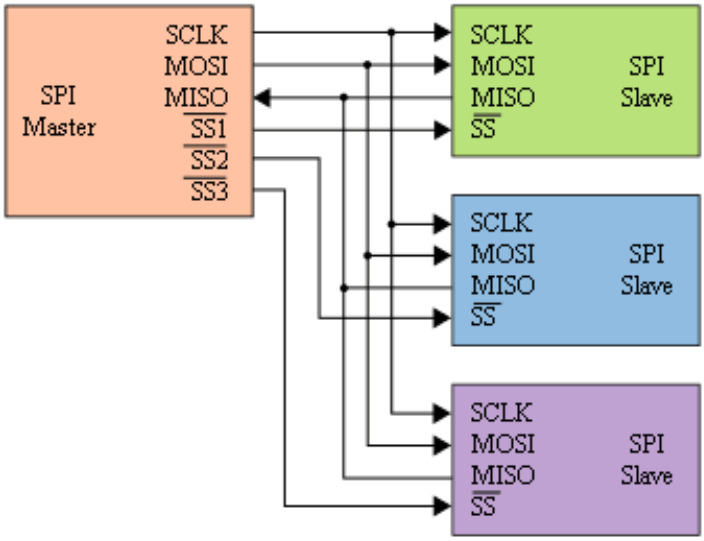
As linhas MISO, MOSI e SCLK são compartilhadas por todos os escravos, assim toda a informação transmitida nelas é percebida por eles, conforme ilustrado na Figura 10. É o nível de tensão do SS que define qual escravo deve tratar estas informações, sendo ativada em nível baixo, o que significa que enquanto este sinal estiver acima do valor mínimo toda a informação que trafega pelo barramento será ignorada. É possível programar as bordas de comunicação do *clock* em até quatro configurações, variando o CPOL (*Clock Polarity*) e CPHA (*Clock Phase*)[12].

Enquanto o barramento I2C traz a proposta de simples implementação e controle de vários dispositivos à uma taxa de velocidade média, o SPI sacrifica a quantidade de escravos que podem ser controlados simultaneamente (definidos pela quantidade de pinos SS) para ganhar em velocidade (transferências acima de 10 Mbps) e na capacidade de envio e recebimento simultâneo de informações. Assim, a escolha entre ambos deve ser feita de acordo com as necessidades do projeto e dos periféricos que serão empregados.

Os processadores são dispositivos que manipulam informações digitais (com apenas dois níveis, 0 e 1), enquanto a maior parte das grandezas do mundo real são analógicas (podem assumir um número infinito de valores diferentes). Em muitas aplicações da robótica é necessário processar informações do último tipo, sendo necessário fazer uma conversão do sinal da forma analógica para a digital. A conversão A/D (Analógico/Digital) é feita através de um

[11] Site: http://www.mct.net/faq/spi.html
[12] Site: https://www.embarcados.com.br/spi-parte-1/

Figura 10 – Ligação de dispositivos no barramento SPI



Fonte: *https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus*

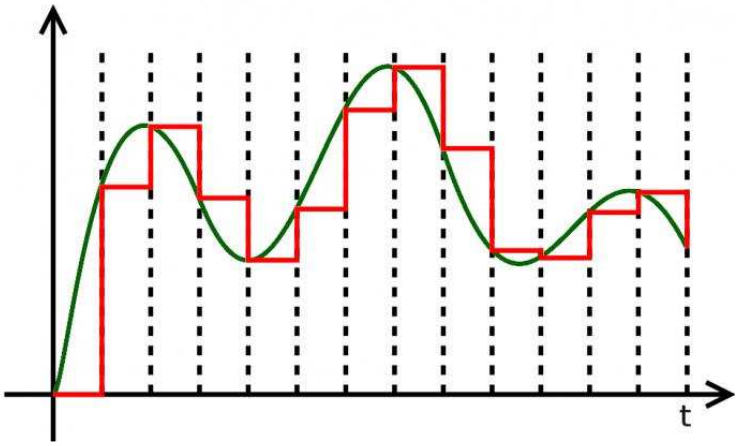
circuito capaz de dividir um sinal com muitos níveis intermediários e transformá-lo em um sinal binário, podendo estar contido ou não na placa embarcada (PELGROM, 2010).

Para fazer a conversão do sinal, o circuito adquire amostras do sinal de entrada, determinando o valor digital de cada uma dessas amostras através da conversão, conforme ilustra a Figura 11. Quanto mais amostras do sinal o conversor adquire (tempo de amostragem), mais precisa é a conversão. O conversor guarda as amostras obtidas em forma binária. Assim quanto maior o número de bits usado para guardar essa informação, mais o valor digitalizado se aproxima do sinal analógico (real), sendo que essa quantidade de bits define a resolução do conversor (PELGROM, 2010).

Muitos sistemas embarcados possuem conversores A/D nativamente; no entanto, nem sempre isso ocorre. Quando o *hardware* do sistema não possui tal recurso, a conversão deve ser feita através de um CI externo. A escolha deste circuito deve ser feita levando-se em consideração a quantidade de canais de entrada, a resolução e o tipo de comunicação utilizada (se é compatível com o controlador)[13].

[13] Site: http://web.mit.edu/rec/www/workshop/microcontrollers.html

Figura 11 – Amostragem de sinal analógico



Fonte: https://www.embarcados.com.br/conversor-a-d/

2.5.3. Entrada e saída de texto e imagem em dispositivo portátil

Ao embarcar um computador portátil em um sistema robótico, como o Raspberry Pi, muitas vezes existe a necessidade de trocar informações entre ele e o operador em forma de textos e/ou imagens. Para tanto, existem algumas formas de implementar essa comunicação, sendo as mais comuns a comunicação via SSH (*Secure Shell*), VNC (*Virtual Network Computing*) e *displays* portáteis. Nos dois primeiros tipos de sistemas de comunicação, o operador utiliza-se de um notebook e no último interage diretamente com o Raspberry.

O SSH é uma forma de acessar um computador remotamente, via rede, com o intuito de executar comandos e receber informações, além de mover arquivos entre as máquinas comunicantes. Este protocolo garante alta confiabilidade na autenticação e comunicação, mesmo através de redes consideradas não seguras. Com o tempo o SSH vem sendo cada vez menos utilizado, mas ainda encontra várias aplicações nas áreas de desenvolvimento de sistemas embarcados, principalmente com o Raspberry Pi, já que dispensa o uso de periféricos de entrada e saída, como o mouse e o teclado. Através do SSH não é possível ter acesso a interface gráfica do dispositivo, somente se tem acesso ao *shell*[14].

O protocolo VNC é um protocolo simples para acesso remoto da interface gráfica. Diferentemente do SSH, pode-se acessar a interface gráfica para visualizar as imagens capturadas por uma câmera, por exemplo, sem a necessidade de transferi-las do computador embarcado para a máquina principal.

[14] Site: https://docstore.mik.ua/orelly/networking_2ndEd/ssh/ch01_05.htm

VNC é um protocolo RFB (*Remote Framebuffer*), o que significa que trabalha ao nível do *framebuffer* e pode ser utilizado em qualquer sistema baseado em janelas. Um *framebuffer* é um conjunto de retângulos de *pixel* colocados em uma certa posição de uma matriz bidimensional, baseado nos dados que a imagem no *display* do cliente é formada O computador acessado remotamente (que gera o *framebuffer*) é chamado de servidor RFB, enquanto o *hardware* que o acessa é chamada de cliente RFB (RICHARDSON, 1998).

Quando conveniente, pode-se utilizar um *display touchscreen* (tela de toque) conectado ao computador embarcado, como forma de entrada e saída de texto. Existe uma diversidade de *displays* comercialmente disponíveis que utilizam muitos barramentos diferentes. Em alguns casos, não existe barramento e este deve ser controlado pino-a-pino, como nos *shields* (circuitos expansores) para a placa Arduino. Entre os barramentos de comunicação mais comuns, tem-se HDMI (*High-Definition Multimedia Interface*), DPI (*Display Parallel Interface*), DSI (*Display Serial Interface*) e DBI (*Display Bus Interface*)[15].

O barramento HDMI é muito comum nas televisões e monitores comerciais. Possui alta taxa de transferência, permitindo resoluções 4K na sua versão 2.0, com banda de 18 Gbps, possibilitando a comunicação entre o *display* e o controlador através de um bit chamado CEC (*Consumer Electronics Control*). No entanto, por ser tão robusto é um barramento caro e nem sempre adequado para os sistemas embarcados[16].

O barramento DPI é um barramento paralelo de 24 bits, com *clock* e bits de sincronização, totalizando 28 linhas que chaveiam à uma taxa de 70 MHz[17]. *Displays* que utilizam esse barramento tem preço acessível, mas exigem 28 linhas de GPIO para que seja feita a comunicação. Dependendo do controlador, essa demanda pode acabar com todas as linhas de propósito geral disponíveis. Por ser paralelo, o DPI sofre muitos problemas com interferência eletromagnética e consumo de potência.

DSI é a evolução do DPI, sendo um barramento serial de alta velocidade. Possui variação de valor em suas linhas entre os níveis lógicos 0 e 1 de apenas 200 mV[18], a fim de reduzir o consumo de energia e alcança velocidades de comunicação em até 1 Gbps. No entanto, por ser um barramento muito especifico e caro, não tem tanta disponibilidade no mercado. Para contornar esse problema, alguns fabricantes de placa de desenvolvimento, como o Raspberry

[15] Site: https://www.raspberrypi.org/blog/the-eagerly-awaited-raspberry-pi-display/
[16] Site: http://www.hdmi.org/
[17] Site: https://www.raspberrypi.org/documentation/hardware/raspberrypi/dpi/README.md
[18] Site: https://www.xilinx.com/support/documentation/application_notes/xapp894-d-phy-solutions.pdf

Pi, utilizam um misto dos dois barramentos, com entrada em DPI, que é convertido, através de um chip, para DSI[19].

2.6. Recursos de *software* empregados na implementação de Sistemas Embarcados

Para resolver os problemas propostos adequadamente, além de recursos de *hardware* apropriados, um sistema embarcado deve possuir algoritmos otimizados. Existem diversos recursos de *software* que podem ser utilizados para isso. Por exemplo, pode-se descentralizar o processamento do sistema, ao fazer execução do código do sistema robótico embarcado separado do código *offboard* desenvolvido em MATLAB®. A comunicação entre eles (notebook e Raspberry Pi) pode ser feita por SSH, conforme comentado anteriormente. Nesta seção, serão observadas algumas linguagens comuns para as soluções dos problemas em robótica e serão comentadas suas vantagens, além de uma breve explicação sobre a linguagem C++, escolhida na implementação do *software* embarcado no robô.

2.6.1. Tipos de linguagem

Uma linguagem é uma forma de transmitir informações. Para que a comunicação seja bem-sucedida, no entanto, o emissor deve transmitir a informação em uma linguagem que o receptor entenda (COOK, 2013). A linguagem de programação é uma forma de transmitir informações para uma máquina, a qual deve ser capaz de entender os comandos e executá-los de forma clara e objetiva. Em alguns casos, a linguagem pode tornar-se um impedimento, por ser muito complexa ou por não ter tantos recursos de acesso facilitado ao *hardware* quanto o desejado.

Para que a comunicação seja alcançada, uma linguagem deve ter sintaxe e semântica. Sintaxe é a estrutura da linguagem, a maneira como a informação deve ser formatada para que o receptor a entenda. Semântica são as regras que definem o significado da linguagem. Tanto a sintaxe quanto a semântica devem ser conhecidas pelo programador e pelo ambiente de programação, de maneira que o código gerado pelo ambiente deverá ser traduzido desta linguagem para o código de máquina (COOK, 2013).

[19] Site: http://mipi.org/specifications/display-interface

Geralmente as linguagens são classificadas em quatro tipos: imperativas, funcionais, lógicas e orientadas a objetos. Em uma linguagem imperativa, como o C e o Delphi, o algoritmo é extremamente detalhado, incluindo a ordem especifica de execução de cada instrução que ocorrem de forma definida no programa. Linguagens lógicas são aquelas em que são criadas regras que o programa deve seguir, mas não existe uma ordem de execução pré-definida pelo programador, por exemplo a lógica *fuzzy*. A linguagem do tipo funcional é aquela que se baseia na construção de funções matemáticas e evita estados e dados variáveis, como acontece com a linguagem Lisp, matemática simbólica, R (estatística) e K (análise financeira)[20]. Por fim, a linguagem de programação orientada a objeto, como C++, se baseia em estruturas de dados que contém informações (atributos do objeto) e procedimentos pertinentes ao objeto criado (SEBESTA, 2010).

Uma máquina não entende comandos fornecidos da mesma forma que os seres humanos, necessitando-os codificados em níveis binários. Logo, para que o computador execute as tarefas desejadas deve haver uma tradução. Existem três formas em que esta tarefa pode ser feita, através de: a) compiladores, b) interpretadores e c) híbridos entre os dois (SEBESTA, 2010).

Um compilador é um programa que recebe a linguagem escrita pelo programador e a converte em códigos de máquina, que estão prontos para serem executadas pelo computador (Figura 12(a)). Este método tem a vantagem de ter execução muito rápida. No entanto, possui o nível de programação mais próximo da máquina, tornando a implementação de certos algoritmos mais difícil. Linguagens como C, C++, PASCAL e Ada são compiladas (SEBESTA, 2010).

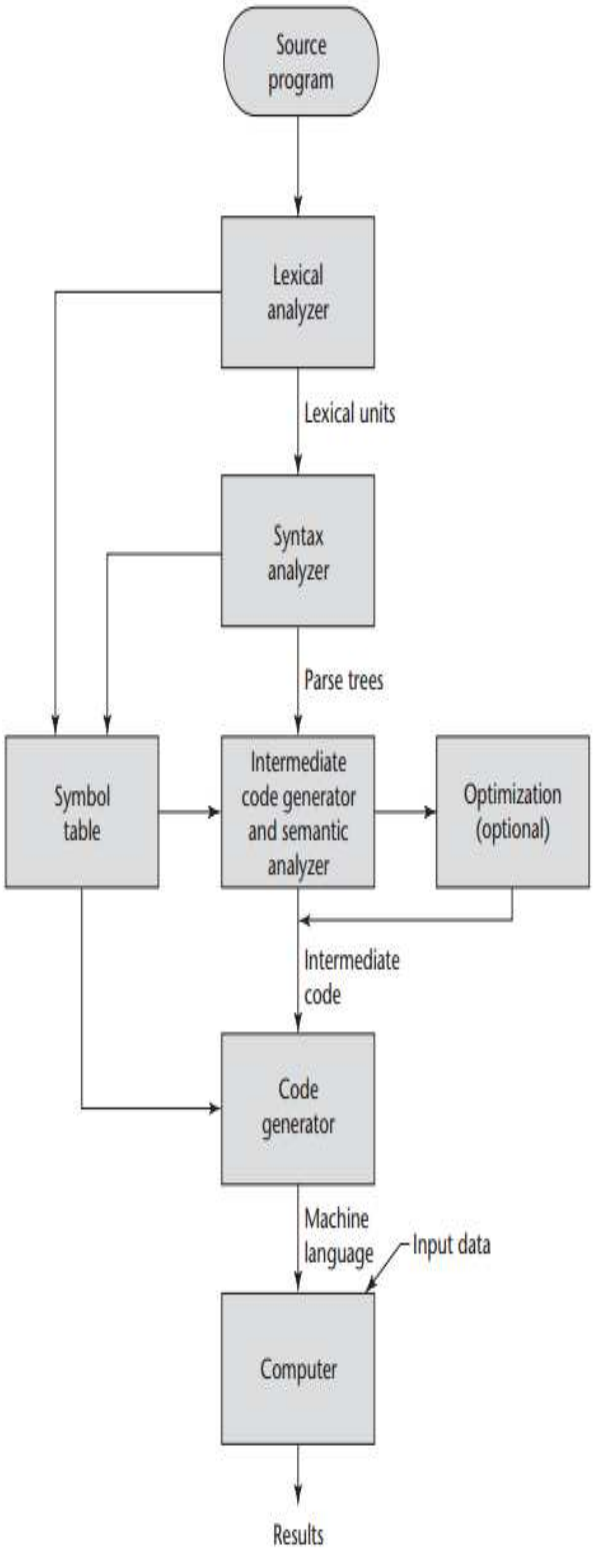
A interpretação pura está no extremo oposto dos métodos de implementação. Em linguagens deste tipo, um programa chamado interpretador, age como uma simulação da máquina, cujos ciclos de captura e execução de instruções operam com o código fonte, em vez de linguagem de máquina (Figura 12(b)). Códigos deste tipo permitem linguagens de nível mais elevado, facilitando a leitura e implementação para o programador. A interpretação garante maior agilidade no processo de *debugg* (depuração), já que todas as mensagens de erro se referem as unidades de código-fonte e não ao código de máquina. No entanto, a execução se torna entre 10 a 100 vezes mais lenta que as linguagens compiladas, podendo complicar seu uso em aplicações mais exigentes (SEBESTA, 2010).

[20] Site: http://www.ppgsc.ufrn.br/~rogerio/material_auxiliar/CLP20131_linguagens_imperativas_funcionais.pdf

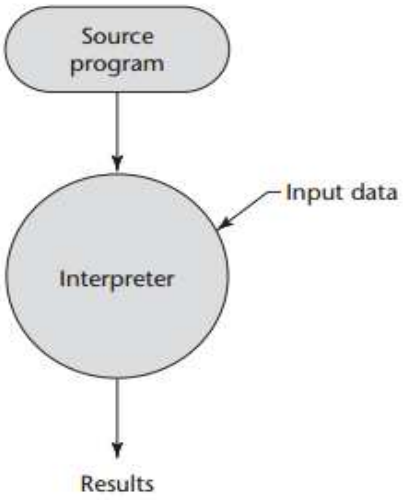
Essa situação ocorre principalmente onde as características tempo real de programação não admitem atrasos provocados pelos cálculos no algoritmo. Existe a possibilidade, ainda, de implementar linguagens que estão entre esses dois extremos, com o intuito de ganhar vantagens dos dois métodos, reduzindo as desvantagens citadas. Programas nessa categoria traduzem linguagem de alto-nível para uma linguagem intermediária, projetada para permitir fácil interpretação (Figura 12(c)). Esse processo é mais rápido que o anterior, mantendo suas características de fácil implementação e depuração (SEBESTA, 2010).

Existem diversas linguagens de programação que podem ser utilizadas para a implementação de programas, dentro de todos os tipos, alguns citados acima (C, C++, Delphi, Python, Java, MATLAB, etc.). Muitos fabricantes de robôs possuem, ainda, linguagens proprietárias que funcionam somente com os seus produtos. No entanto, no desenvolvimento, deve-se levar em conta quais linguagens o sistema de controle aceita. No caso de um sistema multi linguagens, como o Raspberry Pi, é interessante testar as possibilidades para obter o melhor desempenho possível.

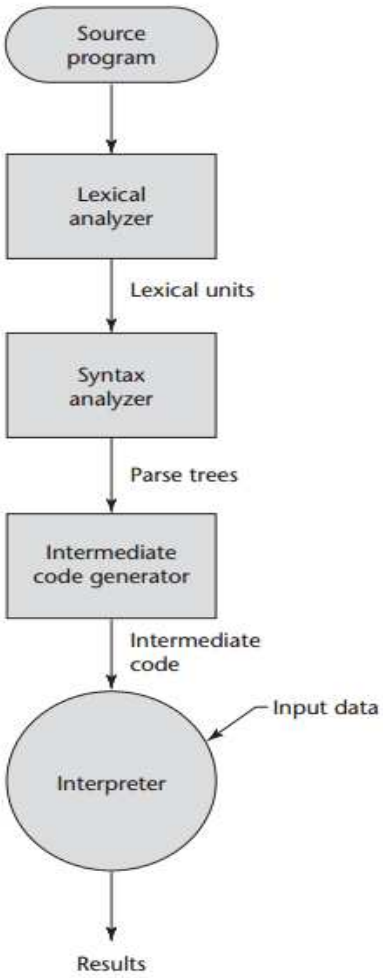
Figura 12 – Tipos de tradução de código fonte



a) Processo de compilação



b) Processo de interpretação



C) Processo hibrido

Fonte: SEBESTA, 2010

2.6.2. Linguagem C++

A linguagem C++ é derivada do C ANSI, devendo ser compilada. É uma linguagem considerada imperativa com suporte para orientação a objeto. Uma das primeiras modificações que levou a essa evolução com a orientação a objeto foi a introdução de classes que definem os dados e métodos desses objetos. Este aspecto permite maior organização dos programas, sem perder desempenho (HARPER, 2011).

A linguagem C++ possui suporte tanto para métodos quanto para funções aplicáveis aos objetos, o que significa que tem a capacidade de suportar tanto a programação procedural (apenas C) quanto a orientada a objeto (C++), possuindo total retro compatibilidade com o C ANSI (HARPER, 2011).

Existem diversas linguagens orientada a objeto atualmente. No entanto, por ser compilado, o C++ traz uma vantagem muito grande em termos de desempenho para aplicações em robótica. Além de disso, proporciona acesso ao mais baixo nível de programação que linguagens como Python e Java, permitindo ao desenvolvedor realizar o controle mais detalhado dos barramentos do processador utilizado, como o I2C e o SPI. Por essas características, é uma linguagem muito popular para a implementação dessas aplicações.

No desenvolvimento dos algoritmos tratados neste trabalho foi utilizada a linguagem C++. A escolha deu-se em função de sua maior capacidade de controle de *hardware* mantendo um bom desempenho de processamento, como comentado anteriormente. Apesar da ausência de uma API (*Application Programming Interface*) oficial para a câmera do Raspberry Pi na linguagem escolhida, a realização destes algoritmos foi possível através biblioteca Raspicam[21], a qual permite o controle da referida câmera através do C++ e foi criada pelo grupo AVA da Universidade de Córdoba.

2.7. Circuitos de interfaceamento e alimentação

Com mencionado em seções anteriores, um robô necessita de sensores e atuadores para interagir com o mundo físico. Muitas vezes estes atuadores, exigem correntes maiores que as fornecidas pelos pinos dos sistemas embarcados. Por exemplo, o motor com *encoder* da Devantech EMG30 especifica uma corrente de *stall* (com o eixo do motor travado) de até 2,5

[21] Site: https://www.uco.es/investiga/grupos/ava/node/40

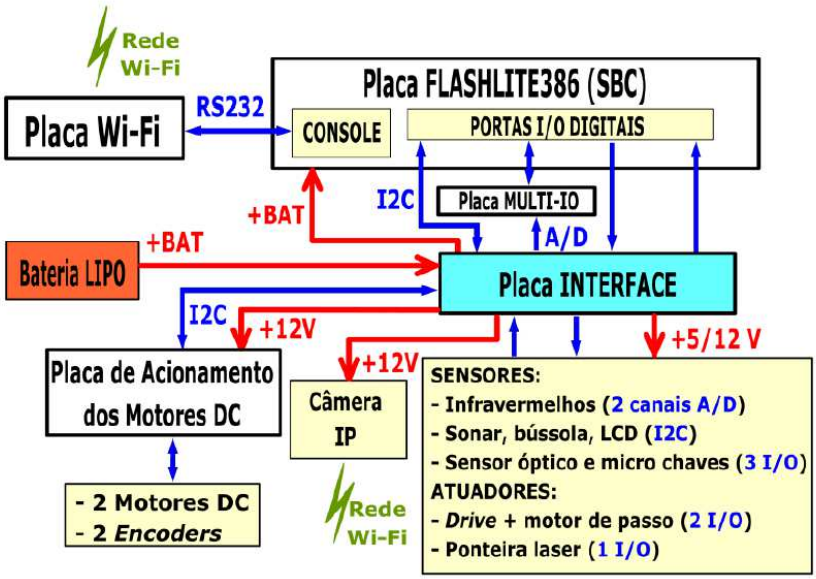
A[22], enquanto a maioria dos processadores fornecem, pelos seus pinos, correntes muito abaixo deste valor (poucos miliampères). Estes periféricos também podem se comunicar com protocolos para os quais o sistema de controle não está preparado. Nestes casos, deve-se utilizar placas de interface e alimentação. Existem diversas placas de interface no mercado que fazem a conversão dos padrões de comunicação mais comuns. O chip SC18IS600, por exemplo, é uma interface entre os padrões de comunicação SPI e I2C.

Um exemplo do uso das placas de interface pode ser encontrado no robô SLAMVITA (BUONOCORE, 2013), onde a mesma é utilizada para permitir a interação entre a placa controladora, os sensores, os atuadores e fazer o condicionamento de sinais, conforme mostra a Figura 13. Foi necessário, criar uma placa de interface para o robô de que trata este trabalho de monografia, a qual será detalhada no capítulo 4.

2.8. Tratamento de imagens usando OpenCV

Visão computacional é a análise de imagens e vídeos de forma automática por um computador. Apesar de ser algo natural para o ser-humano, é um grande desafio para uma máquina interpretar e tratar imagens da mesma maneira. Para o computador, uma imagem é uma matriz, onde cada valor é a densidade de luz do pixel naquele ponto (Figura 14). Logo,

Figura 13 – Esquema de interface do robô SLAMVITA



Fonte: BUONOCORE, 2013.

[22] Site: http://www.robot-electronics.co.uk/products/drive-systems/motors-brackets-and-wheels/emg30-gearmotor-with-encoder.html

entende-se que tratar uma imagem, na verdade, é analisar os dados dessa matriz. Quanto maior a matriz, maior a complexidade do problema. Além disso, essa matriz muda de valores de 30 a 60 vezes por segundo, denominado de quadros por segundo (*Frames Per Second* ou fps), para as câmeras mais comuns (BRADSKI, 2008).

A área da visão computacional tem diversas aplicações na indústria, sendo utilizada para verificar a qualidade das peças manufaturadas ou guiar carros de forma autônoma. Atualmente os desenvolvedores dessa área vem desafiando cada vez mais os limites do que se considerava impossível em aquisição de imagens e criando algoritmos que podem ser úteis em diversas áreas da ciência moderna (SZELISKI, 2010).

Com intuito de facilitar o desenvolvimento de algoritmos nestas áreas de pesquisas, foi desenvolvida e disponibilizada a biblioteca OpenCV. Ela possui mais de 2500 algoritmos otimizados, em todos os ramos de visão computacional e aprendizagem de máquinas. Em sua versão original OpenCV 1.0, lançado em 2006, utilizava a linguagem C, o que gerava um problema na criação de algoritmos mais complexos. Então, em 2009, foi lançada a segunda versão OpenCV 2.0, utilizando a linguagem C++, facilitando a implementação de algoritmos ao introduzir classes e objetos no código[23]. Atualmente a biblioteca está na sua versão 3.0, lançada em 2015, a qual trouxe diversas melhoras de API e otimização dos recursos já presentes na versão anterior, além de aumentar o desempenho em outras linguagens[24].

A biblioteca é estruturada em cinco componentes principais. O componente CV contém os algoritmos básicos para processamento de imagem e de visão computacional. Existe ainda um componente para aprendizagem de máquinas (MLL), que contém muitas ferramentas de classificações estatísticas e de *clustering*. O componente para interfaces gráficas '**Highgui**' possui rotinas de entrada e saída (E/S) e funções para armazenamento e carregamento tanto de vídeos quanto de imagens. Já o CXCORE trata de estruturas básicas, como operações com pontos e vetores, além dos conteúdos das matrizes que contém as imagens (BRADSKI, 2008).

Atualmente, o OpenCV opera em diversos sistemas operacionais, desde os oficialmente suportados, como Windows, Linux, Mac, Android e iOS, até aos adaptados pela comunidade como o Raspbian. Existem API desenvolvidas em diversas linguagens, C, C++, Python, MATLAB, entre outras. Todas essas características tornam esta biblioteca ideal para quem deseja trabalhar com visão computacional.

[23] Site: http://opencv.org/about.html
[24] Site: http://opencv.org/opencv-3-0.html

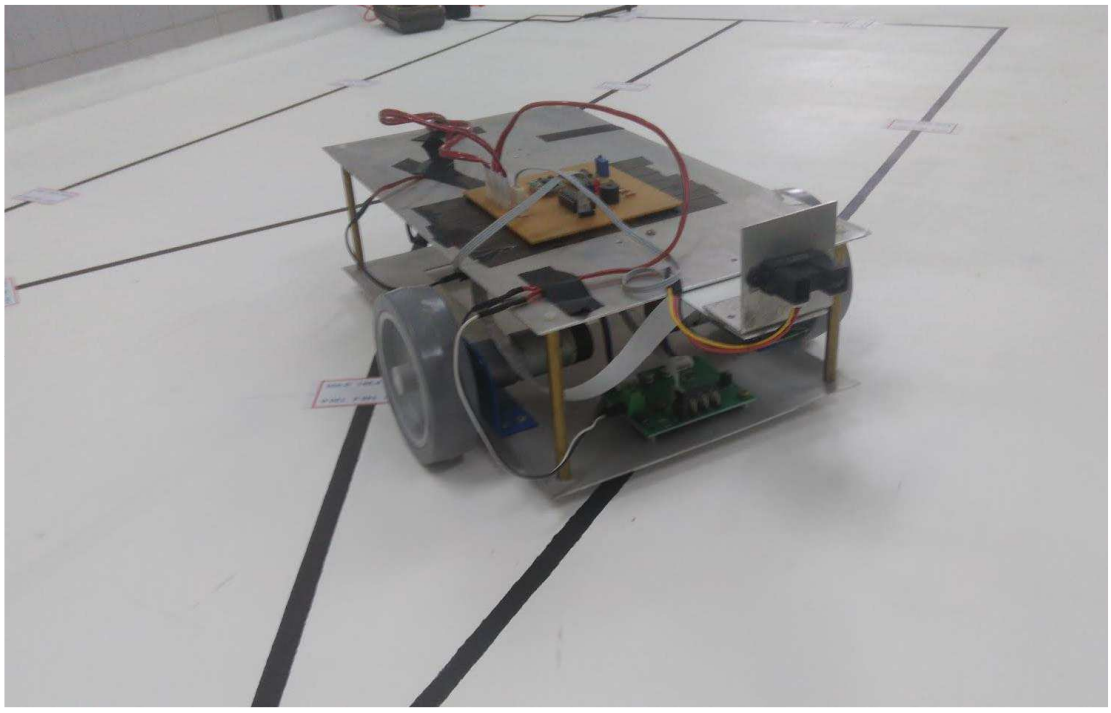
3. PROJETO DE ROBÔ SEGUIDOR DE LINHA RoSeLi

Neste capítulo, é detalhado o projeto do robô RoSeLi, nome atribuído pela forma que se move no ambiente, com caminhos definidos por linhas pretas sobre fundo branco, e que foi construído para a realização das tarefas propostas neste trabalho de monografia, conforme mostra a Figura 15. O robô é formado por um chassi de alumínio, uma placa de interface e um sistema de controle, idealizados e construídos, em sua maioria, no Laboratório de Robótica Móvel e Comunicação Sem Fio (LRC), localizado no Centro de Ciências Exatas e Tecnológicas da **Universidade Federal do Maranhão.**

3.1. Parte mecânica

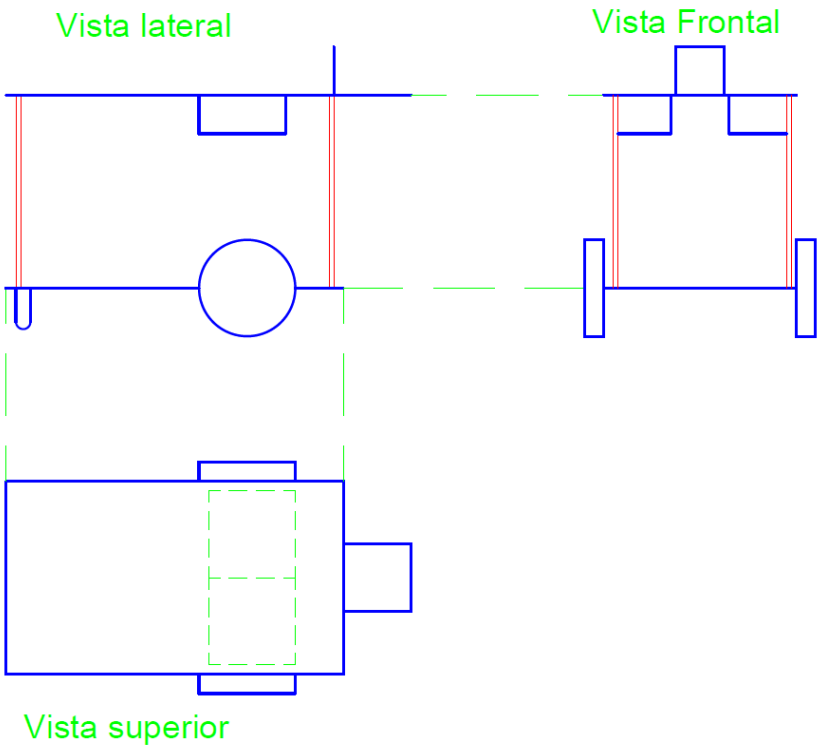
A parte mecânica do robô é composta por duas chapas de alumínio (Figura 16). Na chapa posicionada na parte superior são fixados os suportes para os sensores (infravermelho e câmera), além dos usados para acomodar o sistema Raspberry Pi 3 e a placa de *interface* eletrônica. A chapa inferior serve como apoio para o sistema de propulsão, contendo três rodas, sendo duas tracionadas e uma livre, além da placa de acionameanto diferencial para dois motores, todos partes de um kit de propulsão da Devantech (MD-25).

Figura 15 – Foto do robô RoSeLi no ambiente formado por linhas.

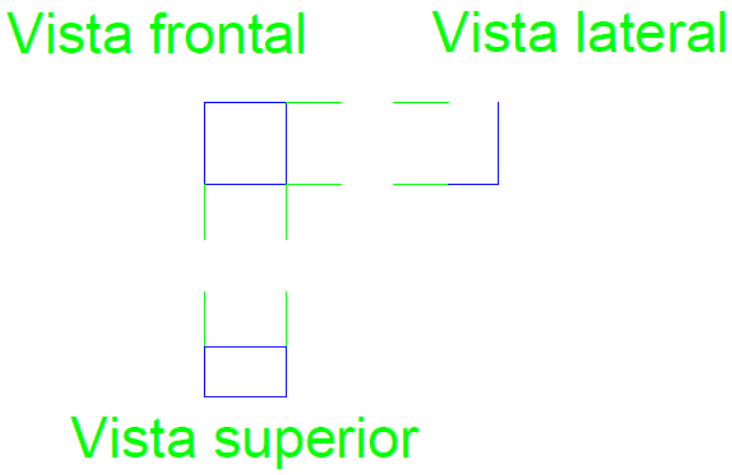


Fonte: Autor.

Figura 16 – Projeto do robô: a) estrutura e b) apoio do sensor infravermelho



(a)



(b)

Fonte: Autor.

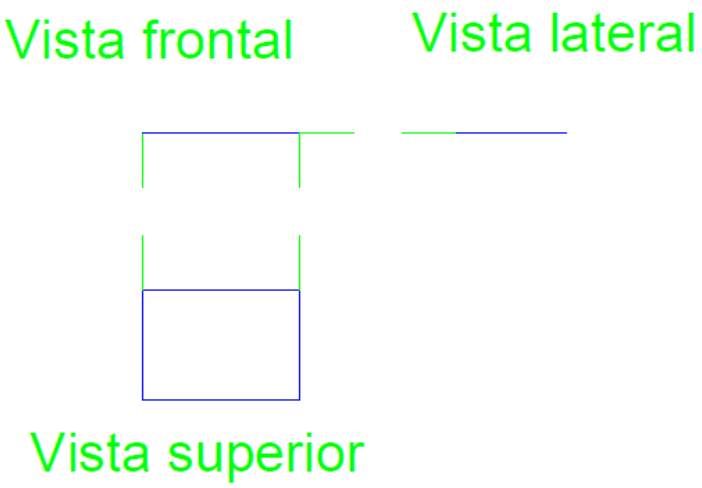
A utilização de rodas como pontos de apoio e forma de locomoção é muito comum na robótica móvel. Essa configuração de apoio permite boa estabilidade a um baixo custo, sendo de simples implementação. É possível alcançar bons níveis de estabilidade com dois ou mais pontos de apoio, sendo mais predominante o uso de três pela simplicidade em sua implementação, além de permitir a livre movimentação em rotação e translação (SIEGWART, 2004), motivo pelo qual foi escolhido para este projeto.

O principal sensor utilizado no projeto foi a câmera, já que possibilita ao sistema robótico realizar as suas tarefas de maneira satisfatória, fazendo uso de processamento de imagens em suas tarefas. Para a sustentação da câmera foi necessário construir uma peça de alumínio que foi fixada à frente do robô (Figura 17). Tal posição foi definida com a intenção de que o sensor capturasse o caminho (linha no ambiente) antes de passar por ele, podendo assim reagir de maneira adequada aos possíveis ângulos, interrupções ou *tags* globais existentes na linha, conforme será visto nos capítulos seguintes. Foi projetado ainda um apoio para o sensor infravermelho, responsável por detectar obstruções no caminho do robô, dispondo-o de forma perpendicular ao chassi, como demonstrado na figura 16b.

3.2. Parte eletrônica

O sistema eletrônico do robô consiste em três placas eletrônicas: a) a placa Raspberry Pi 3; b) a placa de interface e c) a placa de controle dos motores DC (*Direct Current*) de tração. A Placa Raspberry Pi 3 foi escolhida como sistema embarcado por possui uma boa relação custo-benefício pelos recursos que possui e que são indispensáveis na implementação do *software* de execução das tarefas pelo robô. Além de executar o programa que realiza as tarefas, faz a comunicação via *wifi* com o *notebook*, enviando informações como imagens para a definição das linhas no ambiente. A placa de interface foi projetada e construída com o objetivo de fornecer sinais de tensão regulados a todos os componentes do sistema robótico e fazer a compatibilidade de sinais entre os sensores e o Raspberry. A placa de controle diferencial de motores de tração (MD25) foi adquirida como sistema fechado e integrado ao robô. Informações detalhadas destas placas são apresentadas a seguir.

Figura 17 – Apoio para fixação do sensor da câmera no robô



Fonte: Autor.

3.2.1. Placa de Interface

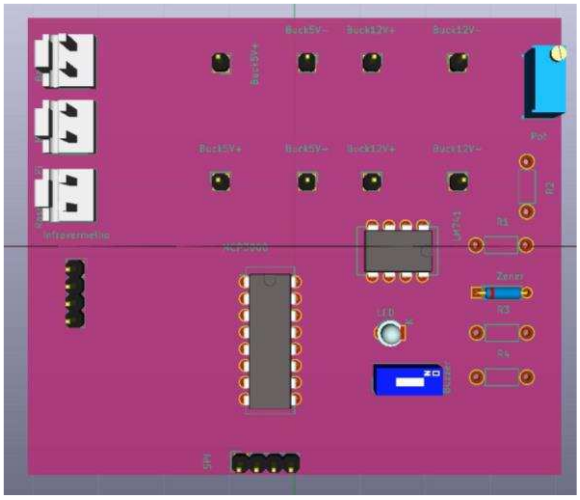
Por ser um robô móvel, o projeto prevê a utilização de uma bateria de Lítio Polímero (LiPo)[25] que possui tensão de saída máxima de 16,8 V continuo.

Para garantir o suprimento da tensão correta tanto para o sensor infravermelho (5 VDC) quanto para o driver (12 VDC), foi necessária a utilização da placa de interface (Figura 18), utilizando conversores *step-down*[26] (conversor DC-DC chaveado). Além disso, utilizando um conversor A/D, a placa de interface faz a intermediação na comunicação entre a placa Raspberry e o sensor infravermelho. Para implementar este recurso, foi escolhido o circuito integrado MCP3008[27], que possui 8 bits de resolução e utiliza o protocolo SPI para comunicação.

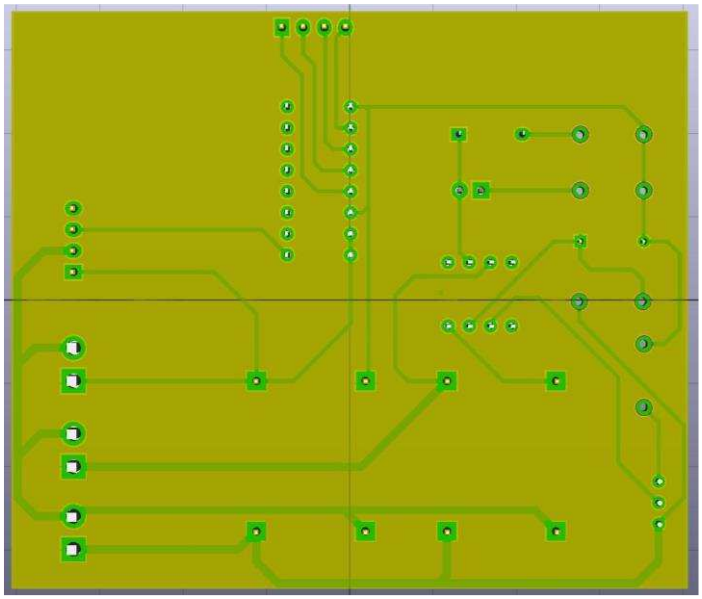
No sentido de evitar que o nível de tensão na bateria alcance nível crítico em sua descarga, foi implementado um circuito capaz de detectar seu nível de tensão e acionar uma

[25] Site: https://rogershobbycenter.com/lipoguide/
[26] Site: http://www.radio-electronics.com/info/power-management/switching-mode-power-supply/step-down-buck-regulator-converter-basics.php
[27] Site: https://cdn-shop.adafruit.com/datasheets/MCP3008.pdf

Figura 18 – Projeto de placa de interface: a) parte superior e b) parte inferior.



(a)



(b)

Fonte: Autor.

sinalização visual e sonora quando essa situação ocorrer. Este circuito foi projetado utilizando um comparador de tensão com amplificador operacional (CI LM741[28]) e um conjunto de resistores configurados de tal forma que sua saída será nível alto somente quando a tensão da bateria for menor que 14 VDC. Esse nível de tensão foi definido de forma a assegurar que a

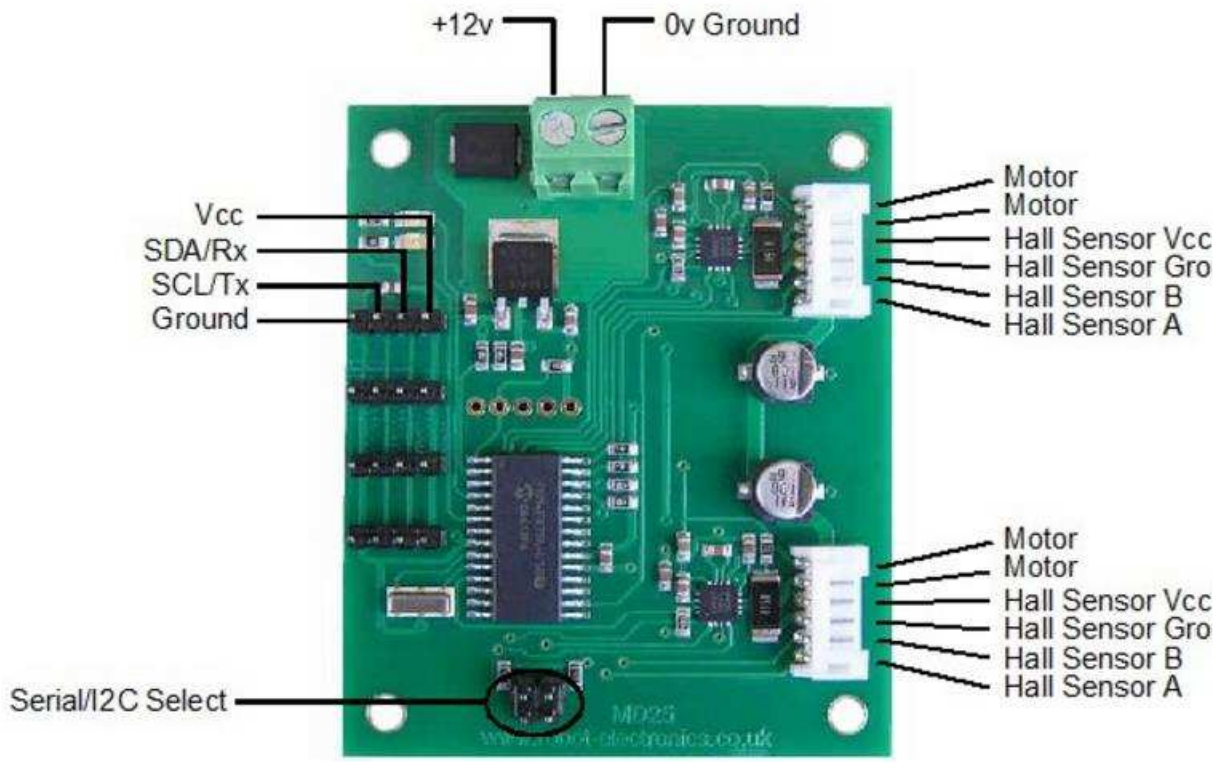
[28] Site: http://www.ti.com/lit/ds/symlink/lm741.pdf

tensão de saída nunca abaixe além dos 12V, valor fornecido pelo fabricante como crítico, abaixo do qual a bateria não consegue mais receber carga.

3.2.2. Placa de controle diferencial MD25

O driver MD25 (Figura 19) é responsável por identificar os comandos do controlador para acionar independentemente os motores DC, além de transmitir os dados da leitura dos *encoders* e estado de alimentação, através do protocolo I2C ou UART. Por ser compatível com o Raspberry Pi e possuir maior velocidade e maior capacidade de dispositivos acoplados simultaneamente foi escolhido o modo de operação por I2C

Figura 19 – Placa MD25



Fonte: https://www.robot-electronics.co.uk/htm/md25tech.htm

. Para operar desta forma, a placa MD25 possui um PIC18F2321 que serve como um *slave,* recebendo e mandando os dados e comandos, conforme a necessidade do controlador do sistema. A Tabela 3 mostra as possibilidades de acessos à placa fornecidas pelo fabricante.

Tabela 3 – Comandos e registradores I2C da placa MD25.

Register	Name	Read/Write	Description
0	Speed1	R/W	Motor1 speed (mode 0,1) or speed (mode 2,3)
1	Speed2/Turn	R/W	Motor2 speed (mode 0,1) or turn (mode 2,3)
2	Enc1a	Read only	Encoder 1 position, 1st byte (highest), capture count when read
3	Enc1b	Read only	Encoder 1 position, 2nd byte
4	Enc1c	Read only	Encoder 1 position, 3rd byte
5	Enc1d	Read only	Encoder 1 position, 4th (lowest byte)
6	Enc2a	Read only	Encoder 2 position, 1st byte (highest), capture count when read
7	Enc2b	Read only	Encoder 2 position, 2nd byte
8	Enc2c	Read only	Encoder 2 position, 3rd byte
9	Enc2d	Read only	Encoder 2 position, 4th byte (lowest byte)
10	Battery volts	Read only	The supply battery voltage
11	Motor 1 current	Read only	The current through motor 1
12	Motor 2 current	Read only	The current through motor 2
13	Software Revision	Read only	Software Revision Number
14	Acceleration rate	R/W	Optional Acceleration register
15	Mode	R/W	Mode of operation (see below)
16	Command	Write only	Used for reset of encoder counts and module address changes

Fonte: http://www.robot-electronics.co.uk/htm/md25i2c.htm

3.2.3. Placa Raspberry Pi 3

Para fazer o controle do robô é necessário um processador de baixo custo e alta capacidade de processamento, considerando que o programa embarcado executa repetidamente comandos de movimento do robô, leitura do sensor infravermelho, além da aquisição e tratamento de imagens para permanecer na linha e identificar valores das *tags* globais, conforme será melhor detalhado na próxima seção. Essas tarefas são executadas simultaneamente de modo que o robô possa permanecer na linha, parar para ler as *tags*, girar em bifurcações de caminho, etc. Essa forma de execução do programa embarcado demanda um ambiente *multithreading*, provido pelo Raspberry Pi 3.

Este computador de baixo custo possui alta capacidade de processamento com frequências de até 1.2 GHz, implementado com arquitetura 64 bits e quatro núcleos. Além do poder de processamento da placa, ela vem com um chip *wifi* embutido, facilitando o controle e supervisão do sistema embarcado de forma remota, via VNC como explicado no capítulo 2. Em contrapartida às vantagens mencionadas acima, a placa não possui entradas analógicas e possui consumo de energia relativamente alto, chegando a solicitar 2,5 A de sua fonte de alimentação. Para contornar a ausência da entrada analógica, foi implementado um conversor A/D na placa de interface, com o intuito de estabelecer a comunicação com o infravermelho (Seção 3.3.1). Na tabela 4 estão descritas todas as especificações técnicas da placa de controle.

3.3. Parte Computacional

Para realizar as tarefas foram desenvolvidos dois arquivos cabeçalhos (*headers*) que agruparam as funções de acordo com o seu tipo. Esse arquivo tem a função de reunir um conjunto de protótipos de funções do mesmo tipo em um arquivo separado, organizando o código e facilitando o seu entendimento. O primeiro arquivo, chamado de <**movimento.h**>, contém todos os protótipos das funções para movimentação do robô e leitura de *encoders*. O segundo, denominado <**sensores.h**> contém todos os protótipos das funções para controle da câmera e sensor infravermelho. As funções de movimento fazem uso da técnica de *pulling*, que consiste em acionar o motor e ler os valores dos *encoders* até que a distância desejada seja

Tabela 4 – Especificações técnicas do Raspberry Pi 3.

Componente	Especificação
Processador	ARMv8 Quad-Core 64 bits (1,2 GHz)
Processador de vídeo	Broadcom VideoCore IV
Memória RAM	1 GB LPDDR2 (900 MHz)
Adaptador Wi-Fi	802.11n
Bluetooth	4.1 (BLE)
Quantidade de pinos	40
Interface para camera	CSI – Câmera Serial Interface
Interface para display	DSI – Display Serial Interface/ HDMI – HIGH DEFINITION MULTIMEDIA INTERFACE

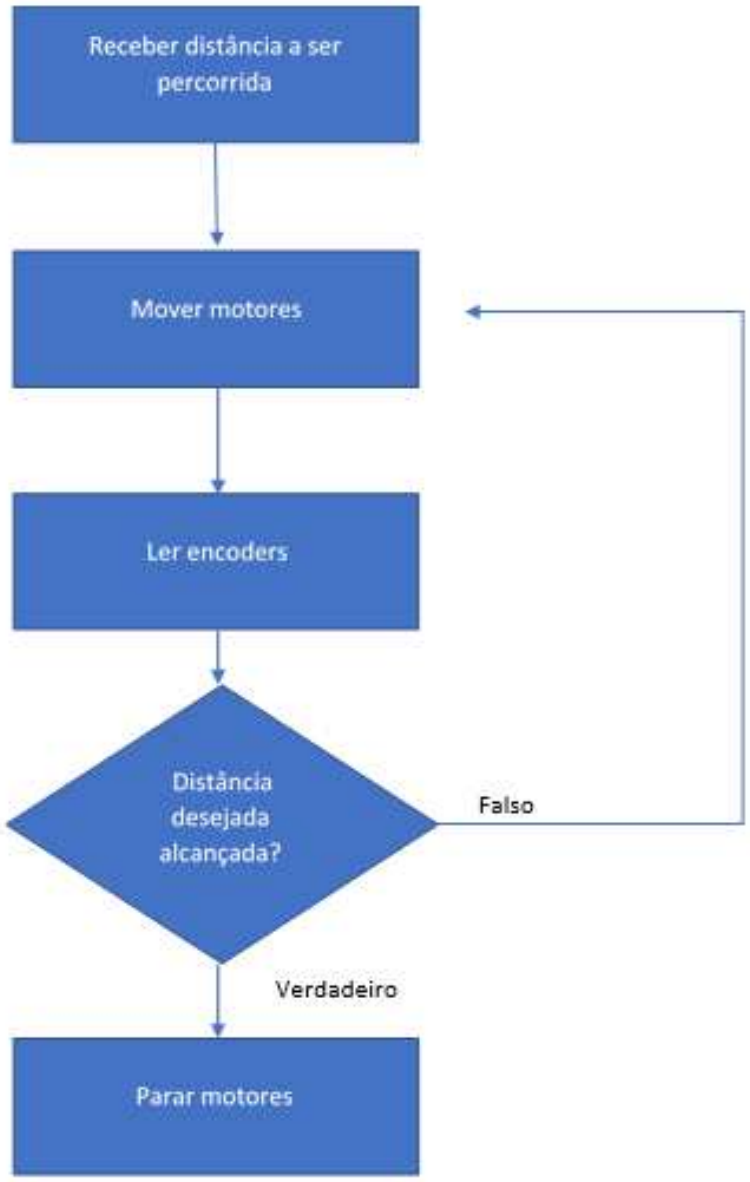
Fonte: https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/

alcançada, conforme mostrado na Figura 20. Já as funções da câmera foram feitas utilizando uma biblioteca própria chamada <**RaspiCam.h**>.

As tarefas executadas pelo robô, que operam em um ambiente *multithreading*, são:

a) Exploração da linha;
b) Leitura de *tags* globais;
c) Mapeamento do caminho; e
d) Leitura do sensor infravermelho.

Figura 20 – Técnica de pulling aplicada nas tarefas de movimentação do robô



Fonte: Autor

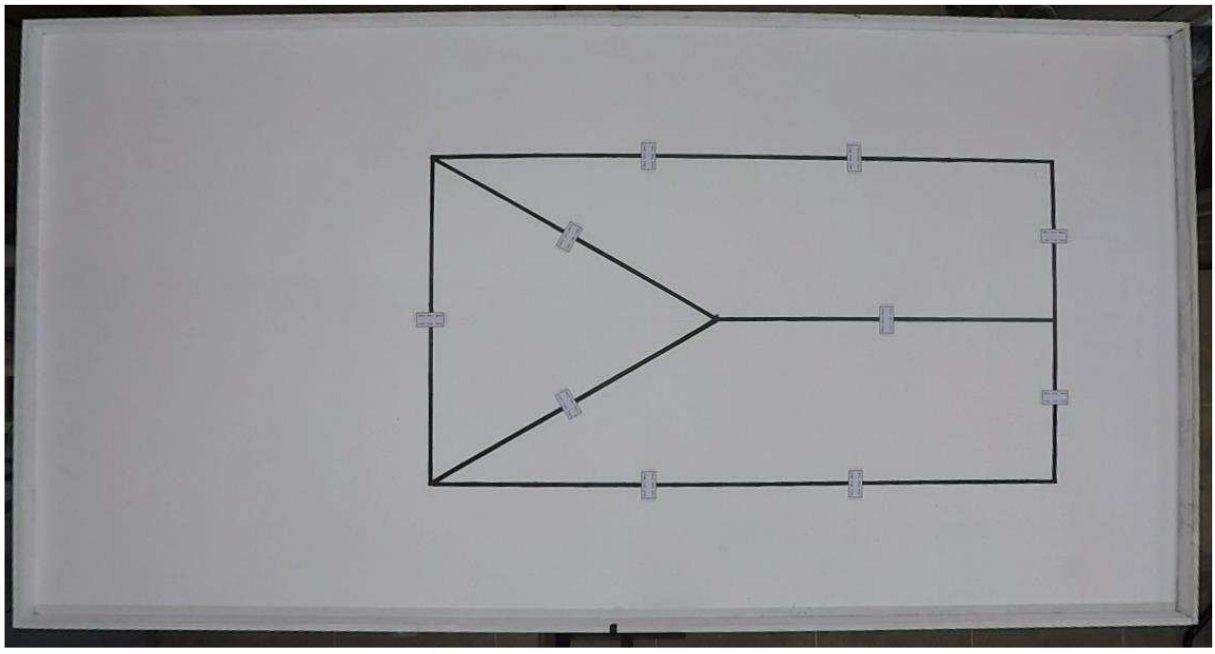
A tarefa de exploração de linha tem a finalidade de possibilitar o robô de seguir os possíveis caminhos do ambiente, demarcados por linhas pretas sobre um plano de fundo branco, de modo a realizar as tarefas definidas nos itens (b) e (c). O ambiente é representado por um tablado de 2,75 x 1,40 m, conforme mostra a Figura 21, onde são percebidas as linhas pretas dos caminhos.

As *tags* globais definem a pose do centro de giro do robô em relação a um sistema de coordenadas globais, definidos em um plano. Para realizar a leitura dessas *tags*, foi utilizado um algoritmo OCR (*Optical Character Recognition*), de forma que os dados capturados puderam ser transformados em variáveis tratáveis pelo programa. Com a informação da pose global, o sistema robótico conseguiu zerar os erros de odometria causados pelos movimentos comandados de translação e rotação, facilitando a tarefa de mapeamento do ambiente, a qual consiste nas definições das linhas.

A tarefa de mapeamento prioriza o mapeamento do caminho externo formado pelas linhas, seguido dos cruzamentos em T e finalmente aos cruzamentos em Y, este último consistindo em uma generalização de bifurcações.

Por fim, a tarefa de leitura do sensor infravermelho ocorre ininterruptamente e tem a finalidade de detectar possíveis obstáculos durante a fase de mapeamento ou mesmo,

Figura 21 – Mapa de linhas no tablado



Fonte: Autor

futuramente, quando o robô quiser se deslocar entre dois pontos distintos do ambiente pelo menor caminho, após o seu mapa ser construído (tarefa não implementada nesta monografia).

4. IMPLEMENTAÇÃO DO ROBÔ ROSELI

A implementação do robô foi feita seguindo o projeto descrito no capítulo 3, com o intuito de construí-lo com a capacidade de realizar as tarefas propostas. A estrutura mecânica foi construída em um estabelecimento especializado nesta cidade e a parte eletrônica e computacional no laboratório LRC.

4.1. Estrutura mecânica

Para construção da estrutura mecânica, o projeto foi levado à uma loja especializada localizada nesta cidade. Neste local foram adquiridas as chapas de alumínio necessárias, no total de três lâminas de 1/16", duas para a estrutura principal (base e superior) e uma para a sustentação dos sensores e placas (lâmina superior). As chapas de alumínio foram cortadas, dobradas e perfuradas para seguir as especificações desejadas e permitirem a montagem do sistema de propulsão, placas e sensores do robô (câmera e infravermelho).

Com as peças confeccionadas, foi possível montar o sistema de propulsão na estrutura, fixando os motores DC por parafusos. As duas placas de alumínio foram separadas através de um parafuso espaçador de cobre de 20 cm, altura determinada através de testes para garantir o foco da câmera utilizada na captura de informações do ambiente. A estrutura mecânica completa é mostrada na Figura 22.

4.2. Placas e componentes de interfaceamento

Com o intuito de garantir que a placa de interface seja capaz de cumprir sua função de forma satisfatória, fornecendo os níveis de tensão adequados e atingindo os seus outros objetivos, foram feitos testes de seus circuitos utilizando uma matriz de contatos (*protoboard*). O circuito foi feito de acordo com o projeto detalhado no capítulo 3 e pode ser visto na Figura 23 (foto e diagrama elétrico).

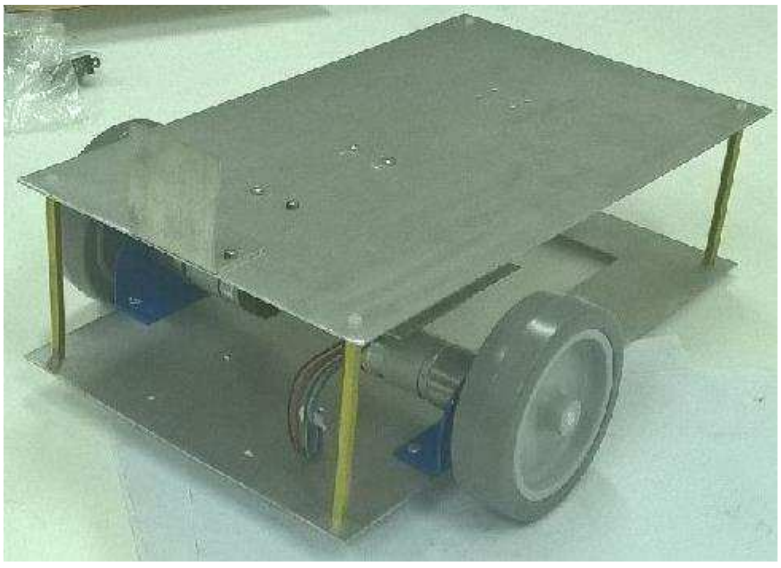
Para testar o circuito de interface I2C para comunicação com a placa MD25, foram feitos diversos testes enviando comandos de leitura e escrita para movimentação dos motores e leitura dos *encoders*. Os resultados obtidos nestes testes isolados foram sempre satisfatórios.

A placa apresentada na Figura 23 é dívida em setores por funções de alimentação, conversão analógico-digital, detecção de nível de bateria e conector do sensor infravermelho. Na parte superior esquerda da placa, são localizados três conectores: o primeiro é onde deve ser

conectada a alimentação geral do circuito, podendo variar entre 12 VDC e 18 VDC. Já o segundo e o terceiro conectores são para fornecer saída regulada em 12 e 5 VDC, para a alimentação dos motores e Raspberry Pi, respectivamente. Na região localizada à direita dos conectores são colocados dois conversores *step-down*, que convertem o valor de entrada para os valores desejados de saída mencionados acima. Abaixo dos conversores, existe um circuito comparador de tensão, para verificar o nível da bateria. Este utiliza um amplificador operacional LM741, uma malha resistiva, um diodo *zenner* e um LED em conjunto com um *buzzer* que servem como forma de sinalização. À esquerda do circuito comparador de tensão, existe um conversor A/D, que serve para fazer a comunicação entre o sensor infravermelho, cuja saída é analógica, e o Raspberry Pi, através do protocolo SPI.

Como mencionado anteriormente, a tensão de entrada da placa é regulada e convertida para os valores de alimentação padrão para a placa de motores (12 VDC) e para o Raspberry Pi 3 (5 VDC). Este processo é feito através de conversores *Buck* ou *step-down*; em contrapartida aos reguladores lineares, mais comuns, estes conversores possuem maior eficiência e esquentam menos[29]. Em um conversor chaveado, existe um interruptor que sofre comutação à altas frequências, de tal forma que, em conjunto com outros circuitos, fornece uma tensão de saída regulada (HART, 2010). Existem diversos tipos de conversores chaveados CC, os mais comuns são: a) o *step-down* reduz o nível de tensão na saída; b) o *step-up* aumenta o

Figura 22 – Estrutura mecânica completa

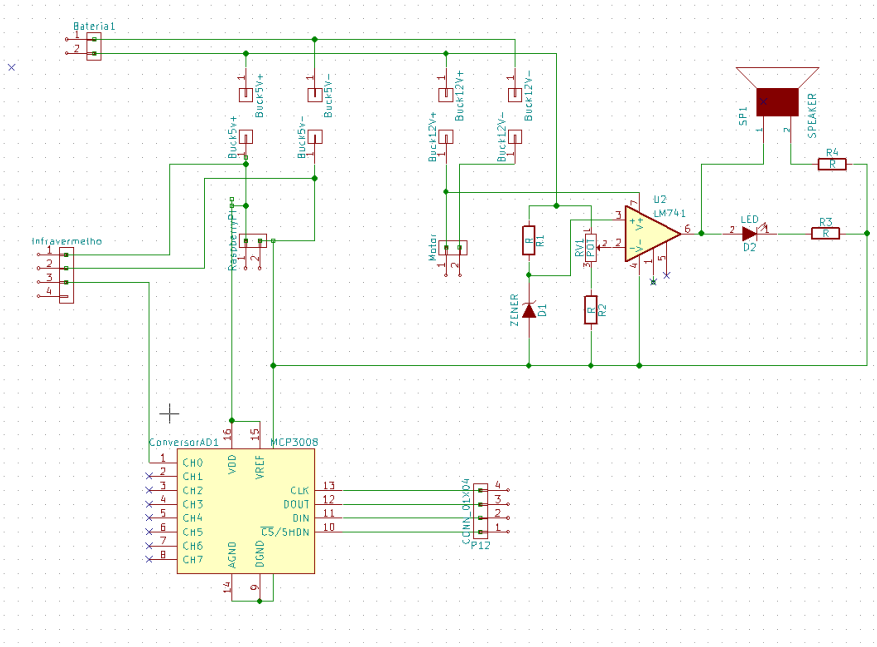


Fonte: Autor.

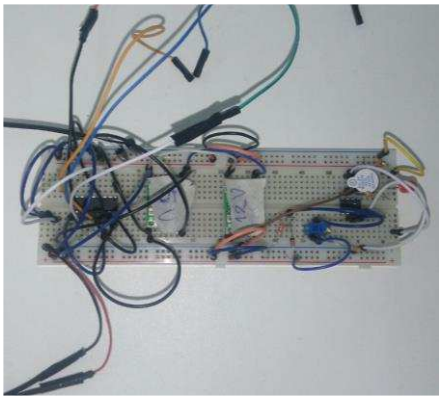
[29] Site: http://micro.rohm.com/en/techweb/knowledge/dcdc/s-dcdc/01-s-dcdc/78

nível de tensão na saída; e c) *buck-boost* que pode fazer as duas funções. O nível de tensão de saída nestes circuitos, é definida pela frequência de chaveamento que possuem, o ajuste dessa geralmente é feito através de um *trimpot* ligado à um oscilador.

Figura 23 – Testes do circuito da placa de interface



a) Diagrama elétrico



b) Teste em protoboard

Fonte: Autor.

O circuito para proteção de descarga da bateria foi inicialmente simulado no ambiente Proteus. Após a simulação, o circuito foi implementado em *protoboard* para testes reais, sendo o nível de tensão de detecção de bateria, que informa necessitar de recarga, ajustado para 14 volts DC. Este valor foi definido em função de testes feitos na bateria indicarem um rápido declínio no valor de tensão, podendo atingir rapidamente o valor de 12V, quando o fabricante previne sobre a possibilidade de não recarga da bateria, inutilizando-a[30]. Para sinalizar ao usuário que esse nível de 14V foi atingido, conforme informado no capítulo 3, foi utilizado um LED (aviso visual) e um *buzzer* (aviso sonoro), os quais são ativados. Por fim, toda a estrutura foi testada em conjunto no *protoboard*, obtendo resultados conforme especificado (LED e *buzzer* ligados quando a tensão da bateria chegar a 14V).

Para garantir que a bateria mantivesse em seu funcionamento adequado, com níveis de tensão de 16,8 V até 14 V, foi necessário comparar o valor instantâneo da mesma com um valor de referência. Com este intuito, foi utilizado um comparador com amplificador operacional (*ampop*). Quando um *ampop* é polarizado sem realimentação, passa a funcionar como um comparador de tensões de suas entradas, posicionando sua saída em tensão saturada negativa ou positiva, de acordo com a diferença entre as entradas (HOROWITZ e HILL, 2013). Desta forma, foi utilizado um CI LM741 sem realimentação, com um divisor de tensão utilizando diodo *zener* para gerar o sinal de referência, aplicado na entrada não inversora e um divisor resistivo com um *trimpot* aplicado à outra entrada. O potenciômetro foi ajustado para que a tensão na entrada inversora seja menor que o valor do diodo zener quando a bateria fornecesse 14 V. Nessa condição o *ampop* entra em saturação positiva, fornecendo 12 V em sua saída, em qualquer outro caso entra em saturação negativa e sua saída é 0 V.

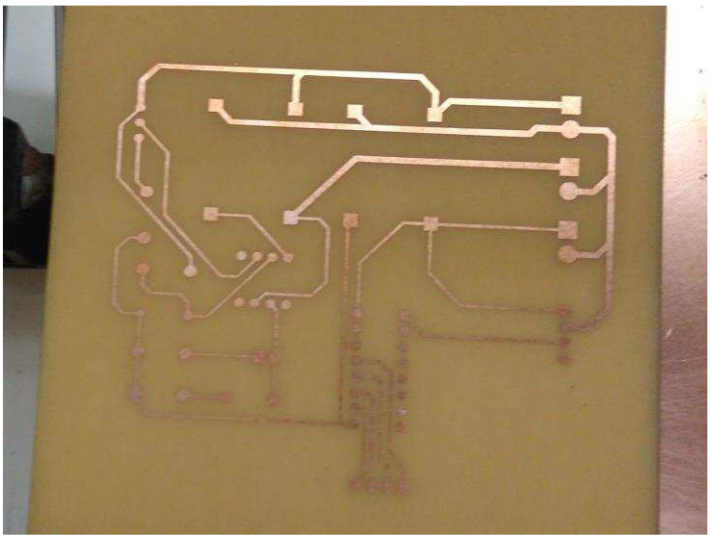
Assim que o circuito foi testado de forma a garantir seu funcionamento, o *layout* projetado na seção 3 foi impresso em uma transparência para confecção da placa de circuito impresso (PCI) final. Foi utilizada uma placa de cobre virgem na qual, através do método de luz fotossensível, foi gravado o circuito desejado na placa.

[30] Site: Manual da bateria thunder power.

Na etapa seguinte a placa foi exposta a luz branca de maneira a sensibilizar a tinta que não foi corroída em uma fase posterior. Esse processo durou aproximadamente 20 minutos. Após a fixação da tinta, a placa foi mergulhada em uma solução de água com barrilha para a remoção da tinta não sensibilizada na etapa anterior, de maneira a permanecer na placa apenas a tinta que fez a cobertura da área do circuito impresso.

Foi utilizado a solução de percloreto de ferro diluído em água para a remoção do cobre exposto, permanecendo na placa apenas a parte do circuito das trilhas e ilhas. Posteriormente, foi usado uma solução de soda cáustica com água para remover a tinta, revelando as trilhas em cobre para que os componentes eletrônicos pudessem ser soldados. Porém antes da soldagem, toda a face da placa com o cobre passou por uma impermeabilização com verniz, visando a proteger a parte útil de condução contra a ferrugem. A Figura 24 mostra a placa após a produção das trilhas e passado a impermeabilização e a Figura 25 com os componentes soldados. A Figura 15 mostrada no Capítulo 3 apresenta o robô montado com todos as placas (Raspberry Pi 3, interface e MD25) e componentes (motores DC e sensor infravermelho e bateria LIPO).

Figura 24 – Placa de fenotlite com circuito impresso



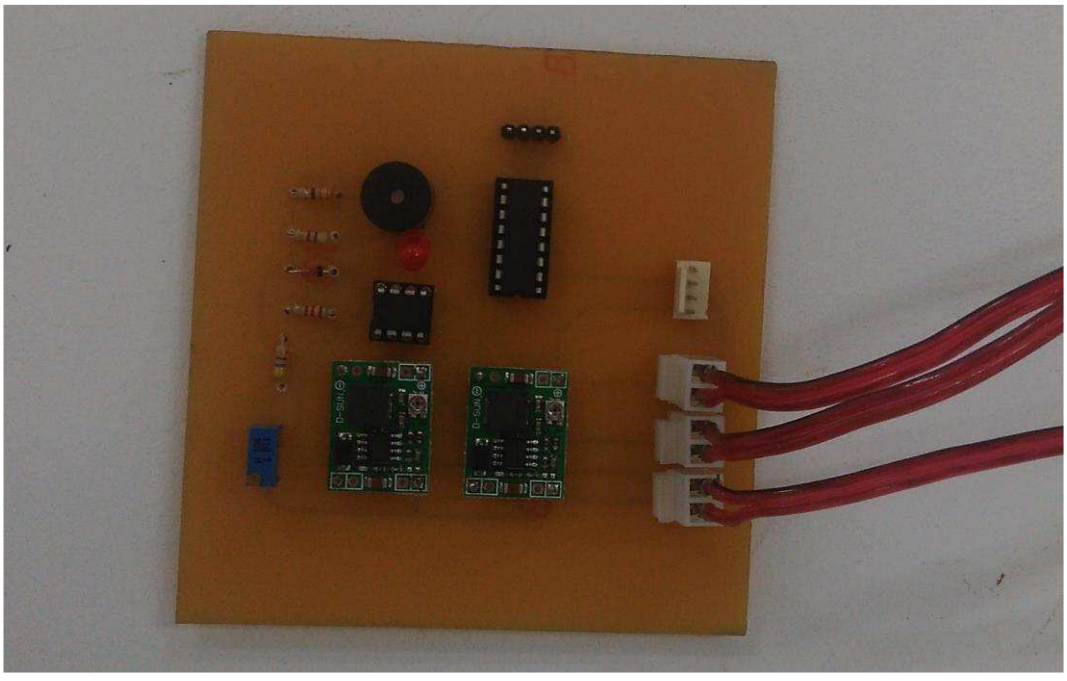
Fonte: Autor.

4.3. Visão geral dos módulos de *softwares*

Para melhor organização dos *softwares* desenvolvidos para o controle do robô, estes foram divididos em módulos, separados por funções que executam em arquivos diferentes. Esta organização facilita na depuração dos códigos e construção de algoritmos, além de ajudar nos seus entendimentos pelos alunos da disciplina TEEE – Robótica Móvel e Probabilística. Nesta perspectiva, este trabalho tem a finalidade também de fornecer material didático teórico e prático. Nesta seção são comentadas em alto nível as funções executadas nos módulos, sendo os detalhes de sua implementação feitos no Capítulo 5.

O primeiro módulo contém as funções de movimento do sistema de propulsão MD25 do robô. A comunicação pelo protocolo I2C é transparente neste código para o usuário, já que são criadas funções primitivas que contém todo este padrão embutido. Foram criadas funções para mover o robô em translação e rotação (em torno do próprio eixo), além de realizar sua parada, a leitura dos *encoders* nos dois tipos de movimentos e realizar o *reset* da contagem dos *encoders*. Essa última função é necessária, para garantir que os motores atuassem na distância pretendida e, para implementar essa funcionalidade, sempre antes do início de um movimento (rotação ou translação), foi feito o *reset* dos sensores.

Figura 25 - Placa de circuito impressa com components soldados



Fonte: Autor

O segundo módulo contém as funções para o sensor infravermelho. Devido a placa Raspberry Pi 3 não possuir pinos para leitura de valores analógicos, conforme comentado na Seção 4.2, a leitura do sensor infravermelho é feita através do barramento SPI no acesso ao conversor A/D. Também foram desenvolvidas funções primitivas para esta finalidade, facilitando a programação e o entendimento por parte do usuário em futuras fases de manutenção. Neste módulo existem funções para abertura e fechamento do protocolo SPI, além da função para medição de distância com o sensor, que retorna o valor medido em centímetros.

As funções desenvolvidas para a câmera utilizam uma API em C++, descrita no capítulo 2, que servem para a configuração do sensor. Essas funções permitem alterar parâmetros como brilho, ganho, contraste e cor. A referida API não possui uma função para captura de vídeo, o que dificultou a implementação do módulo de aquisição e processamento das imagens da câmera. Na captura, a quantidade de quadros por segundo que a câmera consegue alcançar é de apenas 10 fps, enquanto em um vídeo normalmente são atingidos 60 fps. Tal limitação teve influência direta na velocidade em que o robô se locomove no ambiente, pois se esta for alta mais informações sobre a linha serão perdidas.

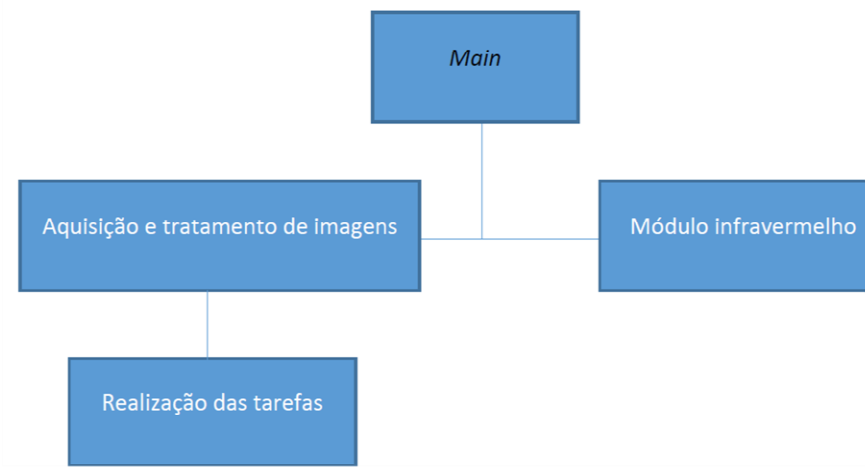
A execução desses módulos ocorre em um ambiente *multithreding* implementado na placa Raspberry Pi 3, conforme será detalhado no Capítulo 5 e demonstrado em alto nível (diagrama de blocos) na Figura 26.

5. MÓDULOS DE "SOFTWARE" DO ROBÔ RoSeLi

Neste trabalho foram criados diversos módulos com o objetivo de controlar o robô de forma eficiente e organizar o código com o objetivo de facilitar o seu entendimento, como mencionado no capítulo 4. Estes módulos são organizados em arquivos chamados *headers* (extensão <**.h**>), reunindo os protótipos de várias funções do mesmo propósito. As funções declaradas nos *headers* pertencem a arquivos com códigos que implementam as tarefas em linguagem C (extensão <**.c**>) e C++ (extensão <**.cpp**>). As funções dos cinco módulos descritos abaixo são chamadas pelo código principal, que organiza um ambiente *multithreading* com quatro fios de execução em um mesmo programa rodando em paralelo pelos 4 núcleos do Raspberry Pi 3:

1. *main*, o *thread* principal, que implementa o ambiente *multithreading*, inicializa suas variáveis, dispara e finaliza o controle do robô;
2. acionamento do robô, responsável por realizar os movimentos de translação ou rotação do robô (nunca ambos ao mesmo tempo);
3. tratamento da linha no ambiente, que captura e realiza o tratamento de imagens, responsável por iniciar e configurar a câmera, além de fazer o tratamento necessário dos *frames* de imagens adquiridos;
4. aquisição das *tags* globais, que realiza a interpretação das *tags* globais que definem as poses do robô em relação a um *frame* de coordenadas globais ao qual as linhas do ambiente estão referenciadas; e

Figura 26 - Estrutura de *multithreading* dos módulos do robô.



Fonte: Autor

5. leitor infravermelho, que lê a distância do robô em movimento ao objeto mais próximo e que se encontra sobre a linha que está seguindo.

Neste capítulo, todos os módulos que serão criados pelo *main* são detalhados, descrevendo as funções que realizam e seus objetivos dentro dos algoritmos responsáveis por executarem as tarefas propostas. Também será explicado como é feita a estruturação do ambiente *multithreading*.

O *thread main*, por realizar a inicialização das variáveis do ambiente, disparar e encerrar os demais *threads*, caso o mapa seja finalizado, não será apresentado em subseção como os demais. Além dessas poucas funções, o *main* faz protocolo com o *thread* infravermelho, conforme será detalhado na Seção 5.5.

5.1. Módulo de acionamento do robô

Para estruturar o módulo de acionamento do robô, responsável pela sua movimentação sobre as linhas que formam seus possíveis caminhos no ambiente, foram utilizados dois arquivos escritos na linguagem C++. Um deles é o <**movimento.h**> (*header*), o qual contém o protótipo de todas as funções de movimento. O segundo arquivo é o <**movimentofunctions.cpp**> (código), que reúne todas as definições das funções do módulo e é linkeditado juntamente com os arquivos de outros módulos na geração do programa executável que executa no Raspberry Pi 3. A Organização dos módulos desta forma facilita a leitura, a utilização e entendimento das funções. Existem seis funções de movimento: a) zerar *encoders*; b) ler *encoders* em movimento linear; c) ler *encoders* em movimento angular; d) acionamento de motores para translação; e) acionamento de motores para rotação; e f) parada de motores. Não existe a possibilidade de realizar ambos os tipos de movimento (translação e rotação) simultaneamente, já que as linhas do ambiente sempre são retas com possibilidades de intersecções. Os motores DC de tração acionados pela placa MD25 tem acionamentos diferencial, conforme citado no capítulo 3.

Quando o robô está parado e necessita fazer um movimento guiando-se pelas linhas do ambiente, tanto em movimento de translação quanto de rotação, é necessário zerar a contagem dos *encoders*, para determinar a distância ou ângulo comandado. Com este intuito foi criada uma função que faz uma operação de escrita pelo barramento I2C para a placa MD25,

zerando a contagem dos *encoders*. O pseudocódigo desta função pode ser encontrado na Tabela 5.

Quando o robô se movimenta, é fundamental que o código tenha uma noção da distância percorrida (translação) ou ângulo de giro efetuado (rotação). Para isso, pode-se obter a contagem dos dois *encoders*, existentes um em cada motor do sistema de propulsão. Tais operações são obtidas através de um acesso de leitura na placa MD-25 através do barramento I2C, onde quatro registradores da placa devem ser lidos para cada sensor, uma vez que a contagem total requer 4 bytes. Assim, para obter o total de movimento desejado, deve-se realizar uma operação de deslocamento de bytes antes que a soma seja realizada, conforme Equação 1. O resultado da soma anterior deve ser utilizado na Equação 2, para obtenção da leitura linear, ou na Equação 3, para leitura angular. O pseudocódigo para a função de leitura dos *encoders* em movimentos do robô é mostrado na Tabela 6.

$$Contagem\ dos\ encoders = (Primeiro\ byte \ll 24) + (Segundo\ byte \ll 16) + \\ + (Terceiro\ byte \ll 8) + (Quarto\ byte) \quad \text{(Eq. 1)}$$

$$Leitura\ linear = \frac{(Contagem\ dos\ encoders) * r * \pi}{360} \quad \text{(Eq. 2)}$$

$$Leitura\ angular = \frac{(Contagem\ dos\ encoders) * 50}{117.5} \quad \text{(Eq. 3)}$$

Na Equação 2, a variável r é o raio da roda do robô e 360 contagens dos *encoders* corresponde a uma revolução completa da roda (360°), implicando que a resolução de contagem

Tabela 5 – Algoritmo de *reset* para *encoders*.

resetencoders ()	
1.	Define dois bytes para serem escritos via I2C, byte do registrador (16) e do comando para resetar a contagem dos encoders (32)
2.	**If** (abertura da porta I2C malsucedida)
3.	Informa ao usuário sobre falha na abertura da porta
4.	**Endif**
5.	Define a placa MD25 como um escravo
6.	**If** (operação de escrita do comando for malsucedida)
7.	Informa ao usuário sobre falha na escrita dos bytes
8.	**Endif**
9.	

Fonte: Autor.

Tabela 6 – Algoritmo de leitura de *encoders*.

```
readencoders (encoder)
1.    Verifica qual encoder o usuário quer ler
2.    If (abertura da porta I2C malsucedida)
3.        Informa ao usuário sobre falha na abertura da porta
4.    Endif
5.    Define a placa MD25 como um escravo
6.    If (encoder é igual a 1)
7.        If (operação de escrita do registrador for malsucedida)
8.            Informa usuário sobre falha na escrita
9.        Endif
10.           If (operações de leitura dos registradores de encoder forem
malsucedidas)
11.               Informa usuário sobre falha de leitura
12.       Endif
13.       encoder = ((buf[0]<<24) + (buf[1]<<16) +  (buf[2]<<8) + buf[3])
14.   Elseif (encoder é igual a 2)
15.       If (operação de escrita do registrador for malsucedida)
16.           Informa usuário sobre falha na escrita
17.       Endif
18.           If (operações de leitura dos registradores de encoder forem
malsucedidas)
19.               Informa usuário sobre falha de leitura
20.           Endif
21.       encoder = ((buf[4]<<24) + (buf[5]<<16) +  (buf[6]<<8) + buf[7])
22.
23.   Else
24.       Informa o usuário de que o encoder acessado não existe
25.   Endif
26.   Retorna (encoder);
```

Fonte: Autor.

é de 1° por contagem. A razão 50/117.5 corresponde a conversão das contagens em ângulo de giro em função da resolução mencionada e da distância entre os centros das rodas e do robô.

Para movimentar o robô (linearmente ou angularmente) são feitas operações de escrita na placa MD25 em dois registradores que são responsáveis pelos acionamentos diferenciais dos respectivos motores DC. Quando se deseja fazer um movimento de translação, os valores das velocidades escritas nos dois registradores devem ser iguais; para rotação, os valores devem informar rotação dos eixos dos motores em sentidos opostos. Já a função de parada é feita ao definir a velocidade em ambas as rodas como zero. Existem três modos de operação possíveis para o controlador dos motores: zero, um e dois. Os modos são programados através de uma operação de escrita no registrador 15 da placa MD25. No modo zero os motores

são controlados de forma independente e possuem velocidades que variam de 0 (velocidade total em ré) a 255 (velocidade total à frente), sendo 128 o valor de parada. No modo um os motores são controlados em conjunto, o registrador 1 controla a velocidade de ambos para translação, enquanto o 2 controla para rotação. O modo dois é similar ao zero, com a diferença de que a variação de velocidade ocorre de -128 a 127. O pseudocódigo para estas funções é apresentado na Tabela 7.

5.2. Módulo de tratamento da linha no ambiente

Uma das tarefas principais do robô é centralizá-lo e guiá-lo através de uma linha preta sobre um fundo branco em um tablado de madeira. Para isso, o *thread* de tratamento da linha do ambiente realiza duas tarefas: a) captura da imagem da câmera; e b) tratamento da imagem definindo o alinhamento do robô.

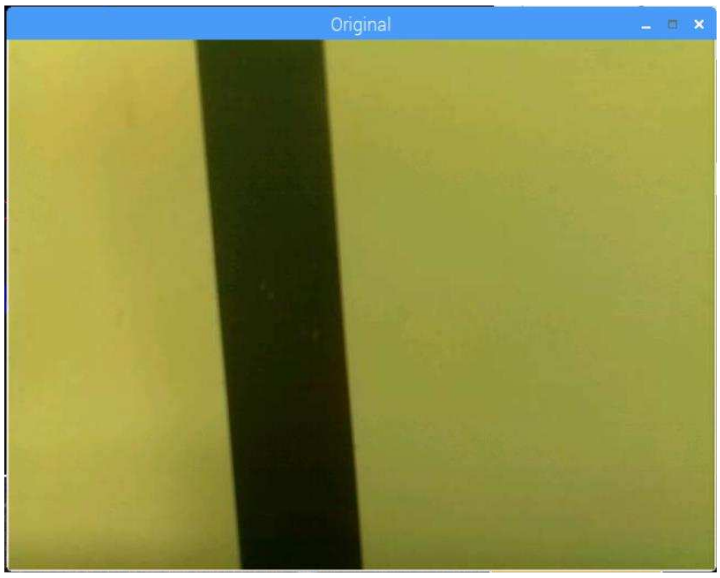
Na primeira tarefa é feita simplesmente a captura do *frame* no formato do padrão RGB (*Red Green Blue)*. Na segunda, essa imagem é convertida para o padrão HSV (*Hue Saturation Value*), como mostrado na Figura 27. O HSV é um espaço cilíndrico de coordenadas para representação de cores que separa o valor do brilho das cores, o que traz mais robustez à

Tabela 7 – Algoritmo de acionamento de motores.

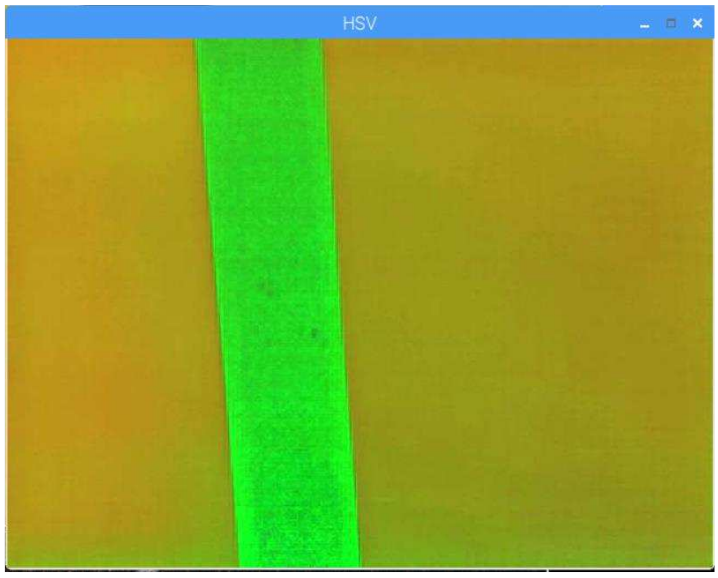
```
Acionarmotores (velocidade1, velocidade2)
1.    If (abertura da porta I2C malsucedida)
2.          Informa ao usuário sobre falha na abertura da porta
3.    Endif
4.    Define a placa MD25 como um escravo
5.    % Define o modo de operação da placa como 1
6.    If (Escrita do byte 1 no registrador 15 for malsucedida)
7.          Informa ao usuário sobre falha na operação de escrita
8.    Endif
9.    % Escreve a velocidade do motor 1
10.   If (Escrita da velocidade1 no registrador 1 for malsucedida)
11.         Informa ao usuário sobre falha na operação de escrita
12.   Endif
13.   % Escreve a velocidade do motor 2
14.   If (Escrita da velocidade2 no registrador 2 for malsucedida)
15.         Informa ao usuário sobre falha na operação de escrita
16.   Endif
```

Fonte: Autor.

Figura 27 – Conversão da imagem do padrão RGB para HSV



a) – Imagem RGB

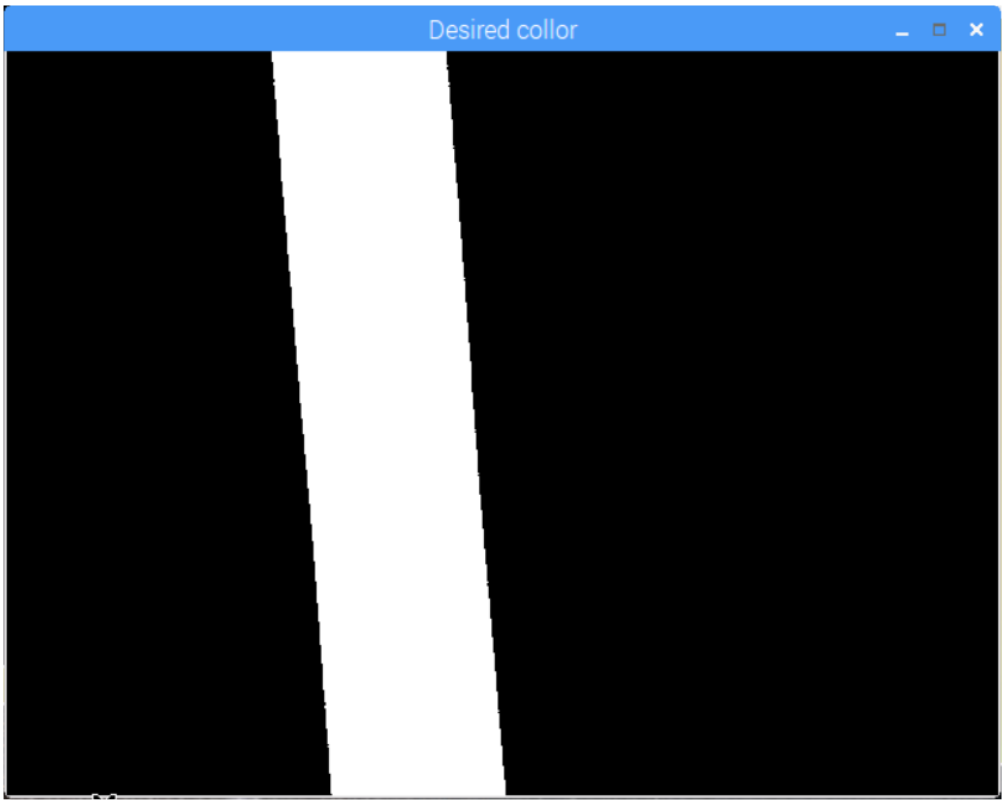


b) – Imagem HSV

Fonte: Autor.

algoritmos de identificação por cor (MacEvoy, 2010). Após a conversão, é feita uma verificação na imagem, buscando os *pixels* que estão dentro do valor que representam as cores preta e branca, separando-os, de maneira a obter a imagem conforme ilustra a Figura 28. Finalmente, é gerada a imagem que contém somente o setor com a linha destacada, onde se localizam os pontos sobre ela centralizados, como será descrito a seguir.

Figura 28 – Imagem em preto e branco usado na identificação da linha



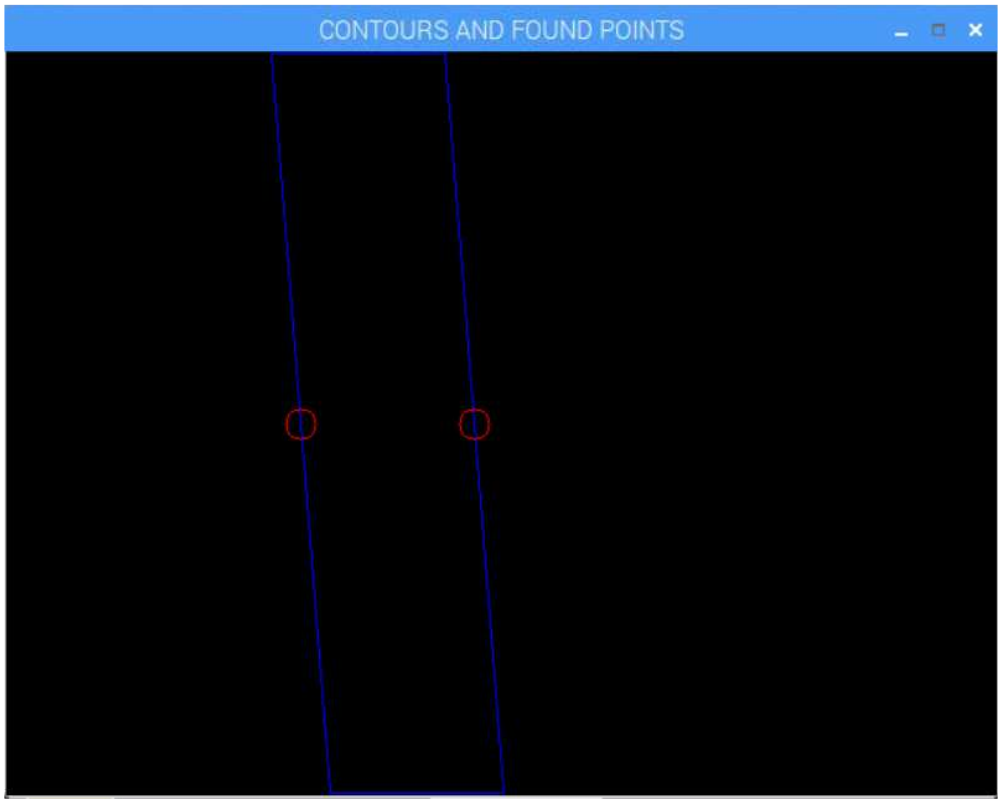
Fonte: Autor.

Para identificar se a linha está centralizada no *frame*, são buscados pelo código os pontos de contorno em uma linha traçada na metade da imagem, representados pelos círculos sobre as linhas na Figura 29. Dessa pesquisa, são extraídos dois pontos, um do contorno esquerdo e outro do direito, sendo comparados com os extremos da tela e calculando as distâncias para os respectivos pontos. Se a distância do contorno esquerdo da linha para o extremo esquerdo do *frame* for igual a do contorno direito para o extremo direito, então o robô está alinhado e deve continuar seguindo em frente. Caso contrário, o módulo de movimento deve realizar comandos de giros em quantidade adequada de maneira a conseguir esse alinhamento centralizado. O algoritmo em pseudocódigo é mostrado na Tabela 8.

5.3. Módulo de aquisição das *tags* globais

Uma das atividades principais do robô é mapear as linhas dispostas no tablado. Para isso, são necessárias informações que estão contidas em *tags* globais espalhadas pelo seu caminho (Figura 30). Existem três dados nestas *tags* (‘x’, ‘y’ e ‘θ’) que informam a pose atual do robô em relação a um sistema global de coordenadas ao qual todas as linhas do ambiente estão relacionadas. A *tag* possui uma moldura vermelha, utilizada para sua identificação, além

Figura 29 – Contornos das linhas e pontos extremos no centro da imagem



Fonte: Autor

de dois conjuntos de *poses*, uma para cada direção em que o robô pode estar seguindo. Esse sistema global foi definido em um dos cantos do tablado, nos experimentos realizados e apresentados no Capítulo 6. As informações contidas das poses são as coordenadas {x, y} no plano do tablado e a orientação do robô. Além disso, a composição e a disposição dos dados nas *tags* contém sempre três números antes do ponto (representa a virgula no código que irá processar a informação) e um após, representado números com uma casa decimal, por questões de precisão (principalmente nas coordenadas) e também de facilitação de correta leitura.

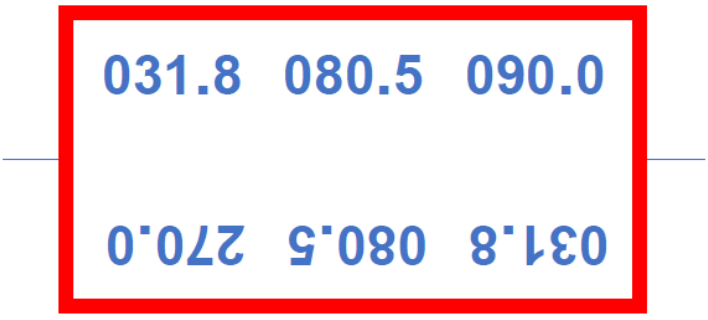
O algoritmo de aquisição da informação da pose inicia-se com a identificação da existência de uma *tag* na imagem. Isso ocorre ao verificar a área da moldura vermelha presente na imagem capturada com o robô em movimento. Caso a área seja suficientemente significativa para não ser considerada um ruído na imagem adquirida, o robô reduz sua velocidade até que o

Tabela 8 – Algoritmo de seguimento de linha.

```
seguirlinha ()
1.   Configuração do formato de captura da câmera para colorida
2.   Definição da resolução da câmera
3.   If (Abertura da câmera malsucedida)
4.       Informa ao usuário falha na abertura da câmera
5.   Endif
6.   While (1)
7.       Captura frame
8.       Converte frame para HSV
9.       Busca os pixels pretos na imagem e os separa, salvando em uma outra imagem
10.      Encontra contorno da linha preta
11.      Utiliza um iterador linear para descobrir os pontos do contorno na metade da altura do
frame
12.      Calcula a distância do primeiro ponto para o extremo esquerdo do frame e do
         segundo para o extremo direito
13.      If (distância do lado esquerdo igual ao lado direito)
14.          Manter velocidade dos dois motores iguais
15.      Elseif (distância do lado esquerdo maior que do lado direito)
16.          Aumentar a velocidade do motor direito e diminuir a do esquerdo
17.      Elseif (distância do lado direito maior que o do lado esquerdo)
18.          Aumentar a velocidade do motor esquerdo e diminuir a do direito
19.      Endif
20.  Endwhile
```

Fonte: Autor.

Figura 30 – Modelo da tag utilizada



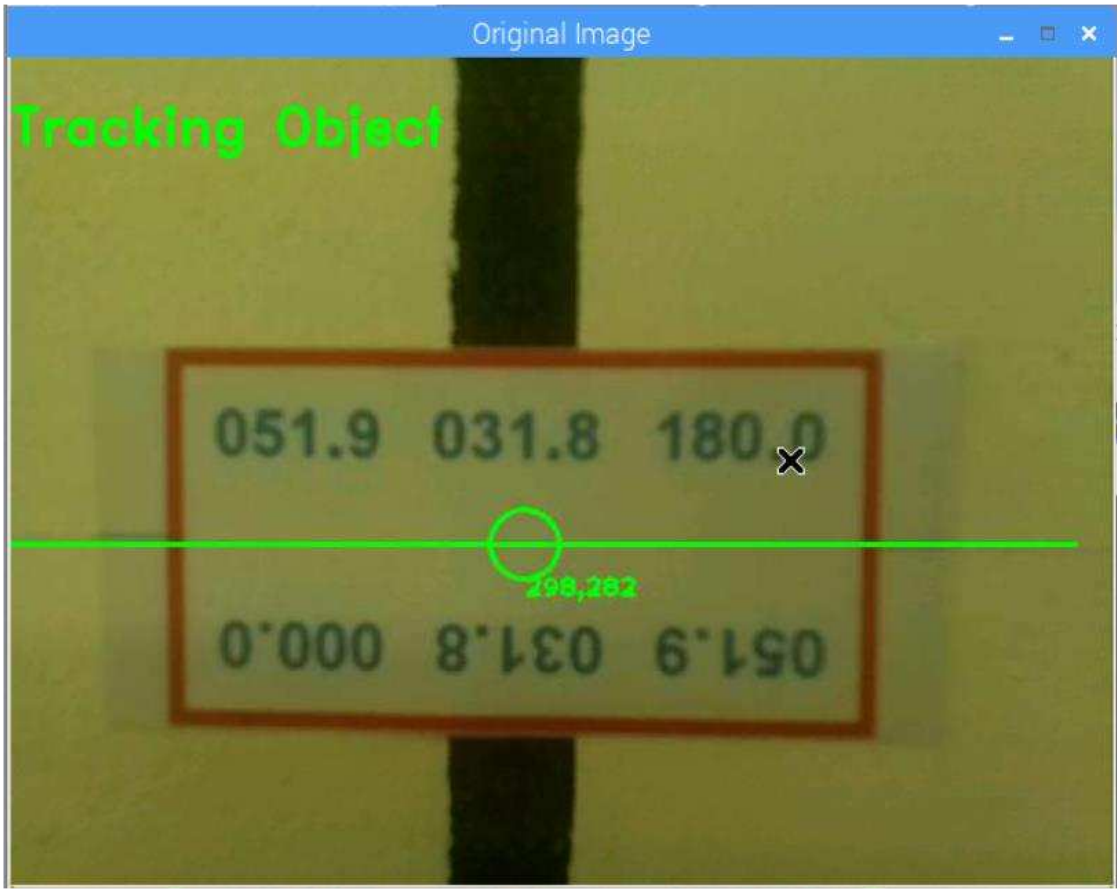
Fonte: Autor

inicio da *tag* coincida com o meio do *frame*. É importante que no momento em que for feita a aquisição das informações, o centro da moldura vermelha deve coincidir ou se aproximar o máximo possível do centro do *frame* da imagem, para que as medidas adquiridas e tratadas retratem uma pose global mais precisa do robô em relação ao sistema de coordenadas localizado

em um dos cantos do tablado. Portanto, o algoritmo tenta centralizar o mais preciso possível a *tag*, calculando o seu centroide (Figura 31).

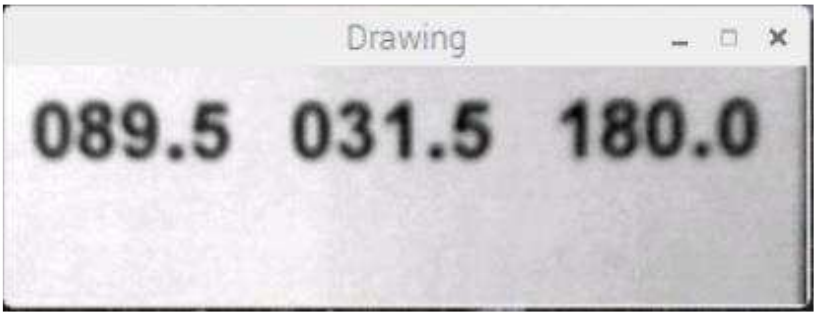
Assim que a *tag* é definida como centralizada, o algoritmo identifica o contorno em vermelho e corta a seção interior a ele, deixando somente as informações úteis da *pose* na imagem. Em seguida esta imagem é convertida para uma escala de cinza e um filtro é aplicado para que os contornos dos números sejam mais facilmente reconhecidos (Figura 32). Na sequência, a imagem tratada é fornecida ao <**Tesseract OCR**>, um módulo que contém a função que faz a conversão de informações na imagem em *strings*, operação conhecida por reconhecimento óptico de caracteres ou OCR. O OpenCV 3.0 já vem com funções prontas que

Figura 31 – Coordenadas (x,y) do centro da tag



Fonte: Autor

Figura 32 – Parte útil da tag cortada e tratada para reconhecimento do OCR



Fonte: Autor

facilitam a utilização de alguns *engines* OCR nos algoritmos, bastando que este esteja instalado no computador onde o código será executado.

A *string* obtida do tratamento com OCR é verificada de maneira a confirmar se a sua estrutura está de acordo com o formato esperado, ou seja, se possui três números antes e um após o ponto (XXX.X). Caso este teste seja positivo, os três dados são separados em *strings* diferentes e convertidos para variáveis *floats* pelo código. Os números resultantes são comparados com um vetor das *tags* presentes no ambiente que estão mantidos no código, de forma a estruturar o ambiente para que o robô possa realizar as tarefas a ele atribuídas. Caso não sejam encontrados no padrão citado, o processo reinicia (aquisição e tratamento) até que o resultado seja verdadeiro. O pseudocódigo para a leitura dos marcos pode ser encontrado na Tabela 9. Nos testes feitos e que são apresentados no Capítulo 6 não houve a situação de não definição da pose, sendo sempre obtidas as variáveis que a compõem.

5.4. Módulo de aquisição do mapa do ambiente

Um terceiro módulo foi criado, utilizando os dois anteriores, para realizar a tarefa de mapeamento. O algoritmo para esse código consiste em mapear primeiramente o caminho externo, marcando os pontos de interesse (interseções de linhas do ambiente) ao longo da linha externa percorrida. Em seguida o robô se movimenta até encontrar a primeira intersecção marcada no percurso externo. Quando existem duas opções de caminho em intersecções, esquerda ou direita, o robô sempre vai para a esquerda, a não ser que esteja procurando um ponto de interesse especifico, como o último cruzamento para completar o mapa. Quando não

Tabela 9 – Algoritmo de centralização e leitura das *tags*.

lertag ()

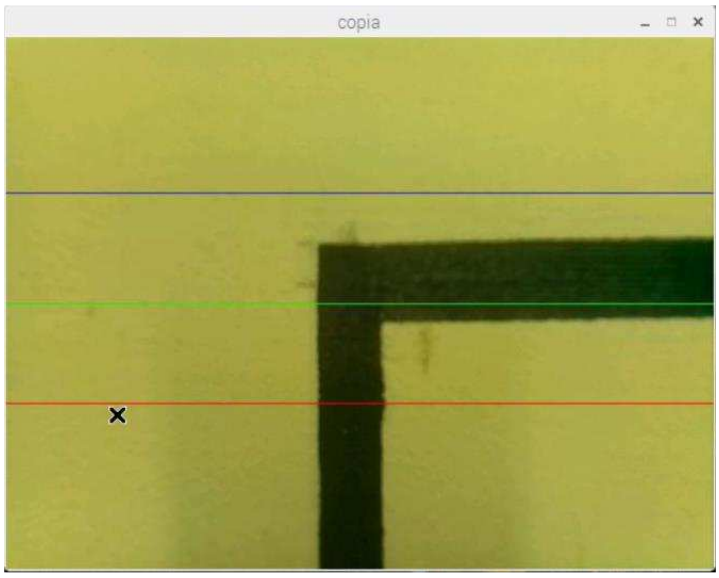
```
1.  Configuração do formato de captura da câmera para colorida
2.  Definição da resolução da câmera
3.  If (Abertura da câmera malsucedida)
4.      Informa ao usuário falha na abertura da câmera
5.  Endif
6.  While (1)
7.      Captura frame
8.      Converte frame para HSV
9.      Busca os pixels vermelhos na imagem (salva e separa em uma outra imagem)
10.     Encontra contorno dos pixels vermelhos
11.     Calcula o centroide da tag
12.     While (componente y do centroide da tag não coincidir com o centro do frame)
13.         Move o robô para frente
14.     Endwhile
15.     Corta a imagem original de acordo com o contorno encontrado
16.     Reduz a imagem cortada pela metade % para manter somente a pose superior
17.     Converte a imagem para cinza
18.     Aplica filtros gaussianos para melhorar a detecção
19.     Utiliza a imagem como entrada para o <Tesseract OCR>
20.     If (string fornecida pelo OCR está dentro do padrão)
21.         Separa os três valores em strings
22.         Converte os valores de strings para float e os armazena no vetor de poses
23.         Break; % sai do laço externo
24.     Else
25.         Realiza nova leitura
26.     Endif
27. Endwhile
```

Fonte: Autor.

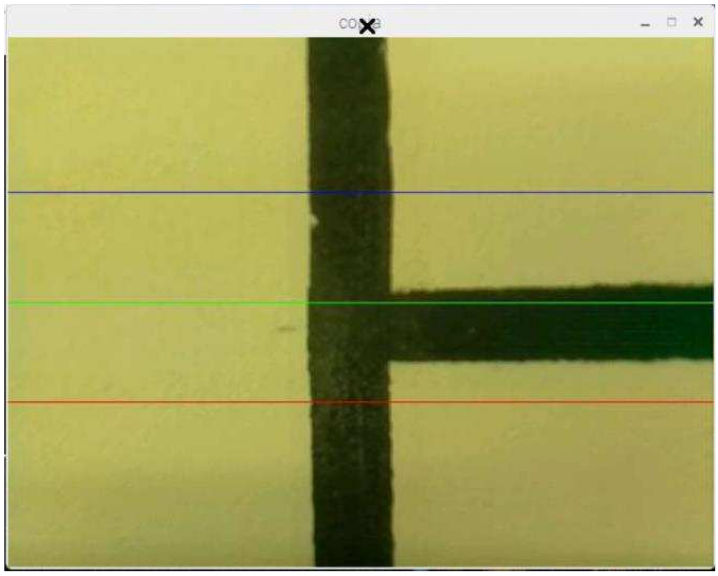
existem mais pontos de interesse, essa tarefa é dada por concluída e o ambiente de execução no Raspberry Pi 3 é finalizado.

No algoritmo de mapeamento estão previstos a identificação de três tipos de possíveis ângulos na formação das linhas do ambiente em situações de interseções. Para isso, são utilizadas três linhas desenhadas na imagem capturada pela câmera (Figura 33): uma no centro da tela, uma acima e outra abaixo da centralizada. As linhas servem para identificar a presença de um caminho possível, além de encontrar seus pontos de contorno. Se as três linhas identificam caminhos possíveis e os três pontos de contorno possuem a mesma componente '**x**',

Figura 33 – Identificação de ângulos



a) – Ângulo de 90º



b) – Cruzamento em T

Fonte: Autor.

então isso significa que não há angulação no trajeto. Se existem dois trajetos possíveis na linha superior, enquanto apenas um na inferior e na central, então há uma bifurcação em '**Y**'. No entanto, se a linha de cima não encontra um trajeto possível, enquanto a de baixo percebe somente um trajeto e a do meio percebe um prolongamento do caminho para a direita ou esquerda, existe um ângulo reto para a direita ou para a esquerda, respectivamente. O algoritmo trata diversos tipos de ângulos para torna o problema de seguir linhas o mais geral possível, como mostrado no pseudocódigo da Tabela 10.

Tabela 10 – Identificação de ângulos.

```
angulodalinha ()
1.    Configuração do formato de captura da câmera para colorida
2.    Definição da resolução da câmera
3.    If (Abertura da câmera malsucedida)
4.        Informa ao usuário falha na abertura da câmera
5.    Endif
6.    While (1)
7.        Captura frame
8.        Converte frame para HSV
9.        Busca os pixels pretos na imagem e os separa, salvando em uma outra imagem
10.       Encontra contorno dos pixels pretos
11.       Itera em linhas em três pontos na imagem, no centro, 100 pixels acima do centro e
          100 pixels abaixo para encontrar a presença de linhas pretas
12.       If ((linha superior possui um caminho) && (linha média possui um caminho) &&
          (linha inferior possui um caminho))
13.           Return (0) % caso 0, a linha não possui angulação
14.       Elseif ((linha superior não possui um caminho) && (linha média possui um
          caminho) && (linha inferior possui um caminho))
15.           If (ponto direito da linha média estiver coincidindo com o extremo do frame)
16.               Return (1) % caso 1, a linha com ângulo de 90 para direita
17.           Else
18.               Return (-1) % caso -1, a linha com ângulo de 90 para esquerda
19.           Endif
20.       Elseif ((linha superior possui um caminho) && (linha média possui um caminho
          largo) && (linha inferior possui um caminho))
21.           If (ponto direito da linha média estiver coincidindo com o extremo do frame)
22.               Return (3) % caso 3, a linha possui uma intersecção 'T' para direita
23.           Else
24.               Return (-3) % caso -3, a linha possui uma intersecção 'T' para esquerda
25.           Endif
26.       Elseif ((linha superior possui dois caminhos) && (linha média possui um caminho)
          && (linha inferior possui um caminho))
27.           Return (2) % caso 2, existe uma intersecção em 'Y'
28.       Elseif ((linha superior não possui caminho) && (linha média possui um caminho
          largo) && (linha inferior possui dois caminhos))
29.           If (ponto direito da linha média estiver coincidindo com o extremo do frame)
30.               Return (4) % caso 4, a linha possui um 'T' para direita
31.           Else
32.               Return (-4) % caso -4, a linha possui um 'T' para esquerda
33.           Endif
34.   °
35.       Endif
36.   Endwhile
```

Fonte: Autor.

Todas as poses do robô detectadas e tratadas pelo algoritmo são armazenadas em um vetor. Os pontos de interseção são guardados em vetores específicos que designam o tipo, podendo ser encontro em '**T**', bifurcação em '**Y**' ou canto com prolongação maior que 90º. Ao fim do mapeamento, os vetores são cruzados e um mapa é gerado com seus dados, que por sua vez é armazenado em um arquivo <**.txt**> e uma imagem. As Tabelas 11 e 12 mostram os pseudocódigos de mapeamento do perímetro externo e das linhas internas, respectivamente.

5.5. Módulo de infravermelho

Existe a possibilidade de um obstáculo interromper o caminho navegável pelo robô. Nestes casos existem formas de detecção do impedimento para que não aconteça uma colisão. Para realizar essa tarefa foi escolhido um sensor infravermelho, localizado na parte frontal do chassi. Existe uma opção melhor para detecção de obstáculos, que seria substitui-lo por um sonar, já que este pode detectar objetos sobre a linha e ao seu redor próximo, garantindo que não haja colisão, considerando-se a área útil de navegação que leva em conta a largura do robô.

A utilização de um sensor infravermelho e não de um sonar deu-se em virtude de haver disponível o primeiro e não o segundo. O uso do sonar pelo Raspberry Pi 3 é bastante simples e vem sendo testado de forma exaustiva em outros trabalhos no LRC. Dessa forma, sua substituição pode ser feita de forma simplificada com alterações no código.

Uma ilustração da forma de funcionamento do sonar nesta aplicação pode ser vista na Figura 34. Devido o sonar enviar uma onda ultrassônica na forma de um cone sólido, pode-se definir a distância mínima (altura do triângulo, '$\mathcal{D}$') que faria o robô passar com segurança pela condição do obstáculo detectado não oferecer empecilho à passagem do robô pela linha.(base do triângulo, '$\mathcal{L}$'). O cone em 3D foi representado no plano 2D por um triângulo para facilitar a visualização e os futuros cálculos, no caso desse sensor vir a ser utilizado.

A leitura do infravermelho é regida por um *thread* próprio, iniciado pela *main* com a passagem de uma variável. Quando a leitura do sensor é menor ou igual a 20cm, o valor da variável é setado para verdadeiro por esse *thread* e o código *main* interrompe os *threads* em execução, parando o robô e sinalizando ao usuário que existe um obstáculo. Assim que o caminho é liberado, as tarefas são reiniciadas de onde foram interrompidas.

Tabela 11 – Mapeamento da parte externa caminho navegável.

```
Mapeamento_externo ()
1.    Iniciar movimentação na linha
2.    Marcar primeira tag lida guardando-a em uma variável chamada pose inicial
3.    While(poseinicial diferente de pose atual)
4.        If(Existe linha no centro do frame)
5.          Seguir linha
6.          If(Ângulo ou tag se aproximando)
7.            Diminuir velocidade
8.          Endif
9.          Switch(ângulo da linha)
10.         Case(0): Seguir em frente
11.         Case(1):
12.           Guardar pose no vetor do mapa
13.           Rotacionar para direita até encontrar linha
14.         Case(-1):
15.           Guardar pose no vetor do mapa
16.           Rotacionar para a esquerda até encontrar linha
17.         Case(2):
18.           Guardar pose no vetor do mapa
19.           Rotacionar para a esquerda até encontrar linha
20.         Case(3):
21.           Guardar pose no vetor do mapa
22.           Guardar pose no vetor de pontos de interesse
23.           Rotacionar para direita até encontrar linha
24.         Case(-3):
25.           Guardar pose no vetor do mapa
26.           Guardar pose no vetor de pontos de interesse
27.           Rotacionar para esquerda até encontrar linha
28.         Case(4):
29.           Guardar pose no vetor do mapa
30.           Guardar pose no vetor de pontos de interesse
31.           Seguir em frente
32.         Case(-4):
33.           Guardar pose no vetor do mapa
34.           Guardar pose no vetor de pontos de interesse
35.           Seguir em frente
36.         Endswitch
37.       Else
38.         If(Existe tag)
39.           Centralizar tag
40.           Ler tag
41.           Guardar pose
42.           Sair da tag
43.         Endif
44.   Endwhile
```

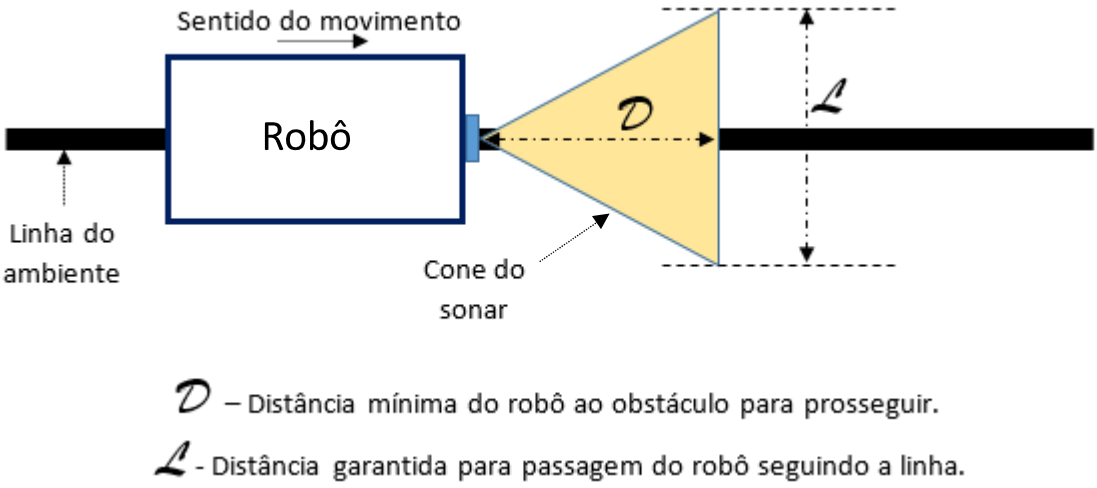
Fonte: Autor.

Tabela 12 – Mapeamento da parte interna caminho navegável.

```
mapeamento_interno ()
45.
46.     While(Vetor de pontos tratados diferente de vetor de pontos de interesse)
47.         If(Existe linha no centro do frame)
48.             Seguir linha
49.             If(Ângulo ou tag se aproximando)
50.                 Diminuir velocidade
51.             Endif
52.             Switch(ângulo da linha)
53.             Case(0):
54.                 Seguir em frente
55.             Case(1):
56.                 Rotacionar para direita até encontrar linha
57.             Case(-1):
58.                 Rotacionar para a esquerda até encontrar linha
59.             Case(2):
60.                 Guardar pose no vetor do mapa
61.                 Rotacionar para a esquerda até encontrar linha
62.             Case(3):
63.                 Guardar pose no vetor do mapa
64.                 Rotacionar para direita até encontrar linha
65.             Case(-3):
66.                 Rotacionar para esquerda até encontrar linha
67.             Case(4):
68.                 Guardar pose no vetor de pontos de tratados
69.                 Rotacionar para a direita até encontrar linha
70.             Case(-4):
71.                 Guardar pose no vetor de pontos de tratados
72.                 Rotacionar para a esquerda até encontrar linha
73.
74.             Endswitch
75.             Else
76.                 If(Existe tag)
77.                     Centralizar tag
78.                     Ler tag
79.                     Guardar pose no vetor do mapa
80.                     Sair da tag
81.                 Endif
82.     Endwhile
```

Fonte: Autor.

Figura 34 – Distância minima de segurança para passage do robô usando sonar



Fonte: Autor

6. EXPERIMENTOS E RESULTADOS

Diversos experimentos foram feitos com o objetivo de testar o robô e os algoritmos desenvolvidos que foram embarcados nele, para realizar a tarefa de mapeamento do ambiente interno formado por linhas. Estes testes têm a intenção de garantir o bom funcionamento da eletrônica e das partes mecânicas do sistema de propulsão, além de sua precisão ao realizar movimentos de translação e rotação. Além dos testes isolados da movimentação e da capacidade de guia sobre as linhas, os quais envolveram a aquisição e o tratamento da imagem adquirida pela câmera, foram realizados os testes de mapeamento. Esse último teste corresponde ao objetivo final deste trabalho de monografia, tendo sido feitos três vezes com o robô partindo do mesmo ponto do ambiente e uma vez de um ponto diferente, de maneira a garantir a robustez do algoritmo ao mapear o ambiente.

6.1. Avaliação dos erros de odometria em movimentos de translação e rotação

A fim de garantir que as medidas fornecidas pelos *encoders* do sistema de propulsão do robô tiveram dentro de limites aceitáveis, foram realizados testes de repetibilidade para movimentos de rotação e de translação. Em cada um dos experimentos foram tomadas diversas medidas, tendo sido obtida uma média do erro entre a medida real do deslocamento (movimento de translação) ou do ângulo de giro (movimento de rotação) realizados pelo robô e àquelas que o *encoder* forneceu ao sistema. Estes dados são importantes, pois a precisão na construção do mapa do ambiente é baseada nas medidas fornecidas por esse sensor.

6.1.1. Teste de repetibilidade para movimentos de translação

Para estimar os erros existentes nos movimentos de translação foram realizadas 50 repetições movimentando o robô, sem seguir a linha, em três distâncias diferentes: 30 cm, 80 cm e 120 cm. Nestes ensaios foi comparada a distância realmente percorrida pelo robô, medida através de uma trena, e a àquela calculada através das contagens dos *encoders*. Não foi levada em consideração a relação entre a distância realmente percorrida e àquela comandada via *software*, já que na aplicação esse erro não é importante, pois pode ser corrigido pela parada do robô com a centralização na *tag* global.

Existem dois possíveis erros em um movimento de translação, aquele em que o robô percorre (a menos ou a mais) do que o *encoder* indica que percorreu e aquele em que há uma derivação para qualquer um dos dois lados (erro de deriva). O primeiro acontece por problemas relacionados às rodas, como falta de atrito, enquanto o último está relacionado a outros fatores, como o acionamento diferencial dos motores, a diferença entre o diâmetro das rodas, pequenas diferenças construtivas entre os dois motores, entre outros, causando a derivação para um dos lados e não apenas em frente. Os resultados dos testes podem ser visualizados na Tabela 13.

Como mostrado pelos dados, os erros obtidos pelos ensaios feitos são suficientemente pequenos, de forma a possibilitar a utilização dos *encoders* como sensor para construção do mapa, especialmente para pequenas distâncias. Este erro deverá, ainda, ser reduzido já que para cada *tag* encontrada o algoritmo informará sua pose global real.

6.1.2. Avaliação dos erros de odometria em movimentos de translação e rotação

Para avaliar os erros de odometria em movimentos de rotação foram realizados 50 ensaios para os ângulos de 45º, 90º e 180º. Em cada uma das repetições a medida real do movimento de rotação, tirada através de um transferidor, foi comparada com a medida informada pelo sensor. Os dados tomados nos ensaios podem ser visualizados na Tabela 14.

Os erros demonstrados acima não influenciam diretamente nas atividades realizadas pelo robô. Isso ocorreu porque os movimentos de rotação realizados no desenvolver das

Tabela 13 – Ensaios para translação.

	Média do valor lido pelos *encoders* (cm)	Média do valor real deslocado (cm)	Media do erro linear (%)	Media da deriva (%)
Ensaio de 30 cm	30,105	30,08	0,08	0,07
Ensaio de 80 cm	80,11	80,01	0,12	2,11
Ensaio de 120 cm	120,07	119,8	0,26	7,14

Fonte: Autor.

Tabela 14 – Ensaios para rotação.

	Média do valor lido pelos *encoders* (°)	Media do valor real de giro (º)	Media do erro de giro (%)
Ensaio de 45°	45,15	45,07	0,17
Ensaio de 90º	90,117	89,45	0.74
Ensaio de 180º	180,19	178,9	0,72

Fonte: Autor.

diversas tarefas que envolvem movimentos de giros do robô são definidas pela linha, eliminando a necessidade da leitura dos sensores nestes casos.

6.2. Aquisição do mapa do ambiente

É importante que o algoritmo de mapeamento desenvolvido seja robusto o suficiente para adquirir mapas navegáveis de pontos de partida diferentes e com consistência, considerando que este é o objetivo principal deste trabalho de monografia. Assim, foram realizados testes de repetibilidade, ou seja, o mapeamento foi feito mais de uma vez, para verificar se as diferenças entre os três mapas construídos pelo sistema seriam diferentes a ponto de prejudicar a confiabilidade do algoritmo. Além disso, o código de mapeamento deve mostrar-se confiável e independente do ponto de partida do robô, apenas sobre a restrição de que o mesmo esteja sobre uma das linhas externas do ambiente. Essa restrição é insignificante e justificável, na medida em que nas situações práticas, essa posição pode ser escolhida pelo operador do sistema.

6.2.1. Testes de correção da odometria pelas *tags* globais

Para que o mapa adquirido pelo robô seja preciso em relação as dimensões reais do caminho no tablado é necessário que quando a *tag* for processada, o seu centro coincida com o centro do *frame* da imagem. Isso é importante para que a pose informada se aproxime em relação a pose real e assim o erro de odometria seja minimizado. As informações das poses existentes nos caminhos do ambiente são todas relativas ao sistema de coordenada global,

localizado em um dos cantos do tablado. Esse fato não possibilita a acumulação de erros de odometria durante os movimentos tantos de translação quanto de rotação.

Foram feitos 50 testes para cada um dos deslocamentos possíveis com o intuito de verificar a precisão entre a posição real do robô e a fornecida pela *tags* em processamento. Existem três tipos diferente de deslocamentos, aquele em que só há variação na componente x (movimentação horizontal), aquele em que só há variação na componente y (movimentação vertical) e aquele em que há variação nos dois (movimentação em diagonal). A Tabela 15 traz os resultados obtidos.

Como observado, os erros obtidos são pequenos e próximos a um valor, não possuindo grandes variações. Em termos estatísticos, se fosse utilizada a função gaussiana, seria possível afirmar que ela possui uma média (com desvio padrão médio de 4,44%). Tais resultados garantem que a pose obtida através da leitura corresponda de forma bastante aproximada com aquela em que o robô está localizada realmente.

6.2.2. Testes de aquisição do mapa partindo de diferentes pontos

Já que no algoritmo só existe uma condição de partida que é a saída do robô quando este já está centralizado na linha, então deve ser possível criar mapas consistentes partindo de dois pontos diferentes. Para testar esta proposição, foram feitos dois mapas partindo de duas coordenadas, cujos resultados (mapas e poses) são apresentados na Figura 35 e Tabela 16.

Tabela 15 – Ensaios de centralização nas *tags*.

	Valor informado pela *tag* (cm)	Media do valor real (cm)	Media do erro na componente x (%)	Media do erro na componente y (%)
Tag no eixo x	51,9	51,95	0,096	0
Tag no eixo y	31,6	31,63	0	0,09
Tag com variação nos dois eixos	(89,9;148,3)	(89,79;148,47)	0,122	0,11

Fonte: Autor.

6.2.3. Testes de repetibilidade partindo de um mesmo ponto

Foram feitos três testes de mapeamento com início centralizado no mesmo ponto do ambiente. Estas repetições tem o objetivo de avaliar a consistência dos resultados obtidos. Para que não houvessem influências adicionais, os testes foram feitos com a mesma bateria e sempre com partida centralizadas. Os resultados são demonstrados na Figura 36 e as respectivas *poses* na tabela 17.

Como pode-se observar pelas Figuras 35 e 36, os resultados obtidos tiveram variações de no máximo 1 cm na mesma pose nos três mapas. Assim, a qualidade dos mapas obtidos é considerada boa e pode ser aprimorada com alguns melhoramentos de *hardware* e *software*. Essa melhora ao nível do *hardware* é possível através da aquisição de novos modelos de câmeras de maior resolução lançadas para operar com o Raspberry Pi. Ao nível de *software* o código de aquisição pode ser otimizado, uma vez que não existe uma biblioteca nativa da câmera para a linguagem C++, tendo sido empregado uma biblioteca de terceiros que possui menor performance, alcançando no máximo 10 FPS. Esta redução de captura de *frames* de imagens por segundo causa a lentidão na ação do robô, pois alguns detalhes do caminho são perdidos. Se uma câmera USB (*Universal Serial Bus*) com suporte nativo do OpenCV fosse utilizada, seria possível obter uma taxa maior de *frames* de imagens, diminuindo o tempo de reação do algoritmo às mudanças na linha. Além disso, pode-se fazer uma operação de *overclocking* no processador do Raspberry Pi, aumentando sua capacidade de processamento. No entanto, isso não pôde ser feito, já que aumenta a temperatura de operação da placa, exigindo um sistema de ventilação ativo que não esteve disponível.

6.3. Testes de detecção de obstáculos

Foi verificado ainda a capacidade do robô de detectar e parar se algum obstáculo obstruir o seu caminho. Para isso, foi criado um *thread* responsável por fazer a leitura do sensor infravermelho em um laço infinito. Por ser direcional, o sensor verifica obstáculos diretamente em cima da linha, se a distância retornada pelo sensor for menor que 30 cm o robô deve parar, interrompendo a execução do algoritmo de mapeamento, até que o caminho esteja livre.

Para validar a capacidade de o robô perceber o obstáculo e parar, foi colocado um obstáculo (cano de plástico) na sua frente sobre a linha, durante a execução do mapeamento. Ao detectar o objeto, o *thread* "informa" o *main* através de uma variável que por sua vez

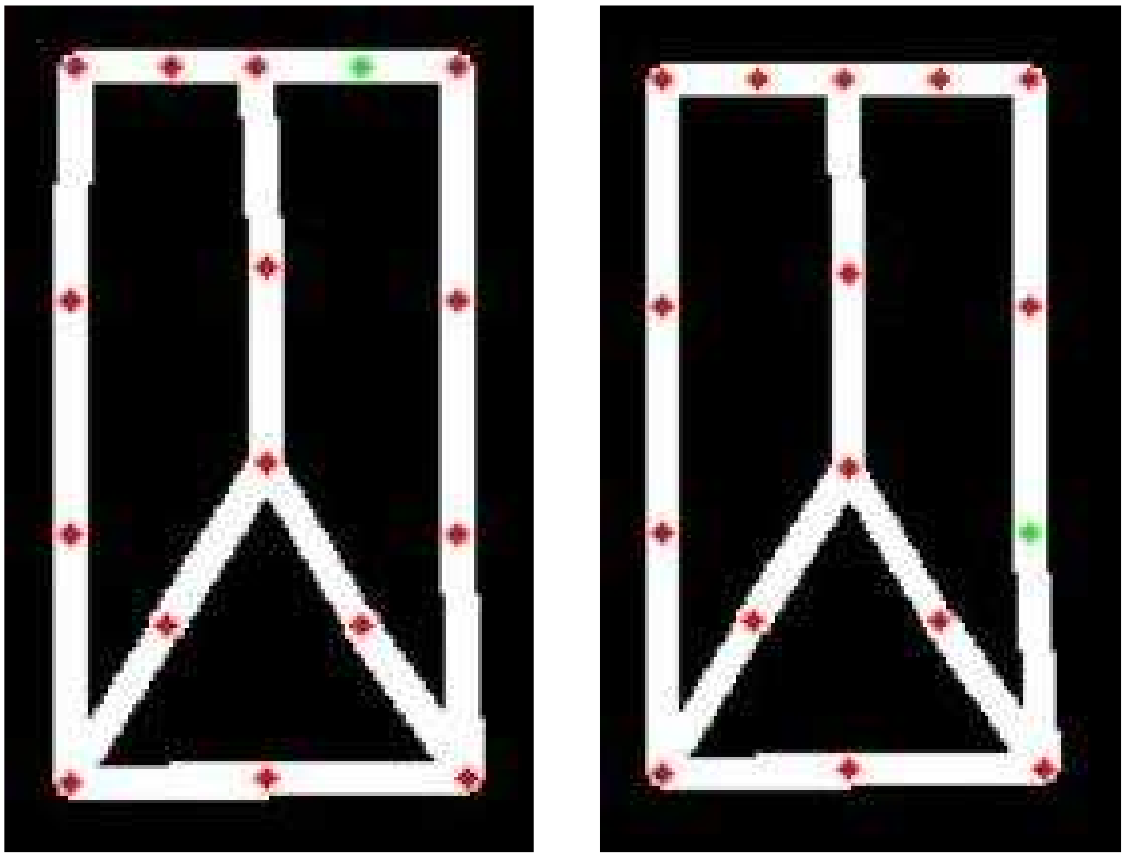
suspende o funcionamento dos outros fios de execução, até que o objeto seja retirado. Quando o caminho é liberado, o algoritmo volta a sua execução.

Tabela 16 – Conjunto de poses no ensaio de mapas partindo de pontos diferentes.

	Mapa 1 (x , y , θ)	**Mapa 2 (x , y , θ)**
Pose 1	089,5; 031,5; 180,0	108,5; 129,9; 270,0
Pose 2	068,8; 031,5; 180,0	108,5; 080,8; 270,0
Pose 3	051,9; 031,8; 180,0	108,5; 031,6; 270,0
Pose 4	031,5; 031,8; 180,0	089,5; 031,5; 180,0
Pose 5	031,6; 080,5; 090,0	069,8; 031,5; 180,0
Pose 6	031,6; 129,9; 090,0	051,9; 031,8; 180,0
Pose 7	031,6; 181,9; 090,0	031,5; 031,8; 180,0
Pose 8	070,0; 180,5; 000,0	031,6; 080,5; 090,0
Pose 9	111,2; 180,5; 000,0	031,6; 129,9; 090,0
Pose 10	108,5; 129,9; 270,0	031,6; 181,9; 090,0
Pose 11	108,5; 080,8; 270,0	070,0; 180,5; 000,0
Pose 12	108,5; 031,6; 270,0	111,2; 180,5; 000,0
Pose 13	070,4; 073,5; 090,0	070,4; 073,5; 090,0
Pose 14	070,4; 115,0; 090.0	070,4; 115,1; 090,0
Pose 16	050,0; 148,4; 120,0	050,0; 148,4; 120,0
Pose 17	089,9; 148,3; 240,0	089,0; 148,3; 240,0

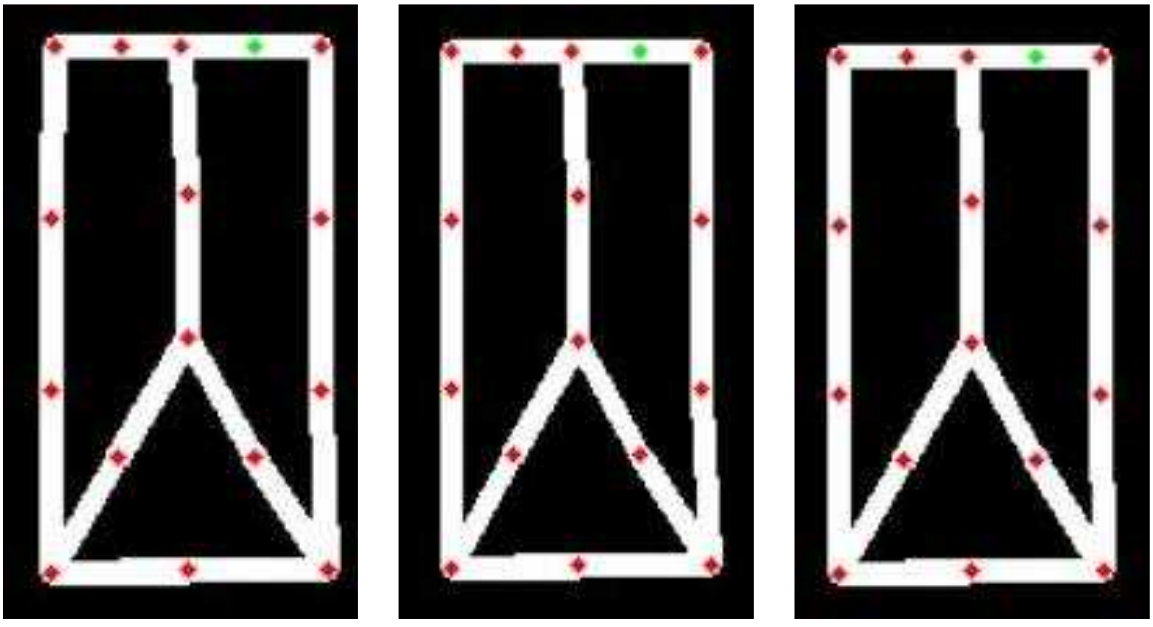
Fonte: Autor.

Figura 35 – Mapas tomados de duas posições iniciais diferentes (marcadas em verde)



Fonte: Autor.

Figura 36 – Mapas tomados da mesma posição inicial



Fonte: Autor.

Tabela 17 – Conjunto de poses no ensaio de mapas com mesmo ponto de partida.

	Mapa 1 (x , y , θ)	**Mapa 2 (x , y , θ)**	**Mapa 3 (x , y , θ)**
Pose 1	089,5; 031,5; 180,0	089,5; 031,5; 180,0	089,5; 031,5; 180,0
Pose 2	068,8; 031,5; 180,0	069,1; 031,5; 180,0	068,9; 031,5; 180,0
Pose 3	051,9; 031,8; 180,0	051,9; 031,8; 180,0	051,9; 031,8; 180,0
Pose 4	031,5; 031,8; 180,0	031,5; 031,8; 180,0	032,4; 031,8; 180,0
Pose 5	031,6; 080,5; 090,0	031,6; 080,5; 090,0	031,6; 080,5; 090,0
Pose 6	031,6; 129,9; 090,0	031,6; 129,9; 090,0	031,6; 129,9; 090,0
Pose 7	031,6; 181,9; 090,0	031,6; 181,6; 090,0	031,6; 181,9; 090,0
Pose 8	070,0; 180,5; 000,0	070,0; 180,5; 000,0	070,0; 180,5; 000,0
Pose 9	111,2; 180,5; 000,0	109,5; 180,5; 000,0	110,3; 180,5; 000,0
Pose 10	108,5; 129,9; 270,0	108,5; 129,9; 270,0	108,5; 129,9; 270,0
Pose 11	108,5; 080,8; 270,0	108,5; 080,8; 270,0	108,5; 080,8; 270,0
Pose 12	108,5; 031,6; 270,0	108,5; 031,7; 270,0	108,5; 031,6; 270,0
Pose 13	070,4; 073,5; 090,0	070,4; 073,5; 090,0	070,4; 073,5; 090,0
Pose 14	070,4; 115,0; 090,0	070,4; 114,2; 090,0	070,4; 114,9; 090,0
Pose 16	050,0; 148,4; 120,0	050,0; 148,4; 120,0	050,0; 148,4; 120,0
Pose 17	089,9; 148,3; 240,0	089,9; 148,3; 240,0	089,9; 148,3; 240,0

Fonte: Autor.

7. CONCLUSÕES E TRABALHOS FUTUROS

O presente trabalho de monografia abordou a solução de mapeamento de ambiente interno (tablado de 2,75 x 1,40), formado por linhas pretas sobre fundo branco, onde o robô RoSeLi foi especificamente projetado e construído para realizar o mapeamento das linhas do ambiente. O único sensor utilizado pelo sistema robótico para fazê-lo guiar-se pelas linhas do ambiente foi uma câmera de baixo custo. Nenhuma restrição, além das linhas serem sempre retas (não traçadas com formatos curvilíneos), foi imposta. Com isso pode-se testar situações de cantos à esquerda ou direita, interseções na forma de 'T' ou bifurcações em 'Y', generalizando a formatação do ambiente e adequando às aplicações práticas como o uso em armazéns ou hospitais.

As linhas do ambiente definem os possíveis caminhos por onde o robô pode circular. Sobre essas linhas foram posicionadas *tags* que informam, de forma indireta, a pose do centro de giro do robô em relação a um sistema de coordenadas global. Cada *tag* contêm as informações de duas poses que são lidas pela câmera, sendo aproveitada apenas a pose útil, a qual depende do sentido que o robô se movimenta sobre a linha.

Antes de realizar os experimentos de mapeamento do ambiente, foram feitos diversos ensaios isolados que testaram as diversas tarefas envolvidas: a) captura e processamento da imagem da câmera; b) movimentação sobre a linha sempre com objetivo de centralizar o robô sobre ela; c) movimento de centralização para leitura das *tags* com o mínimo erro de odometria; d) definição e escolhas das interseções ou bifurcações a seguir; e e) detecção de obstáculos sobre a linha.

Essas tarefas intermediárias foram executadas todas de forma satisfatórias, sendo a que centraliza a imagem para leitura das *tags* apresentados resultados bastantes satisfatórios com erros de no máximo 0,12% na componente x e 0,11% na componente y. Foram feitos testes isolados para o levantamento desses erros que se manteve dentro de valores baixos na faixa indicada. Como esses pequenos erros referem-se a poses globais, eles não são cumulativos. Dessa forma, os mapas gerados nos experimentos mostraram-se bastante próximos das linhas no ambiente, diferenciando-se das mesmas dentro do limite dos erros.

Como trabalhos futuros a serem desenvolvidos a partir do atual estágio alcançado neste trabalho de monografia podem-se citar:

a) A definição do menor caminho a ser percorrido pelo robô entre dois pontos do ambiente, tendo o mapa já sido gerado;
b) A implementação do ROS (*Robot Operating System*) no ambiente operacional embarcado no robô, de maneira a abstrair no *software* o acesso aos recursos de *hardware* do robô;
c) A implementação de uma iluminação embarcada para a câmera, além da otimização do *software* para aquisição de dados das *tags*;
d) Realização de *overclocking* no RaspberryPi, com o intuito de aumentar a taxa de imagens por segundo, melhorando a captura de detalhes da linha.

REFERÊNCIAS BIBLIOGRÁFICAS

APLICACIONES DE LA VISION ARITIFICIAL. **Raspicam: c api for using raspberry camera with/without opencv**. Disponível em: <https://www.uco.es/investiga/grupos/ava/node/40>. Acesso em: 23 dez. 2016.

AURESIDE. **Mercado de automação residencial**. Disponível em: <http://www.aureside.org.br/_pdf/potencial_2015.pdf>. Acesso em: 31 mai. 2017

BRADSKI, Gary; KAEHLER, Adrian. Learning OpenCV: Computer vision with the opencv library. California: O'really Media Inc, 2008. 580 p.

BUONOCORE, L. **SLAM em Ambientes Internos Utilizando Robô de Baixo Custo.** Tese de Doutorado em Dispositivos e Sistemas Eletrônicos, Instituto Tecnológico de Aeronáutica, São José dos Campos, 2013.

COOK, Willaim R.. **Anatomy of Programming Languages.** Austin: University Of Texas Print, 2013. 129 p.

DOCSTORE. **History of ssh**. Disponível em: <https://docstore.mik.ua/orelly/networking_2nded/ssh/ch01_05.htm>. Acesso em: 23 dez. 2016.

ENGIN, Mustafa; ENGIN, Dilsad. Path planning of line follower robot. **Proceedings of the 5th european dsp education and research conference,** Amsterdã, v. 5, p. 1-5, set. 2012.

EMBARCADOS. **Conversores a-d**. Disponível em: <https://www.embarcados.com.br/conversor-a-d/>. Acesso em: 23 dez. 2016.

GETTY IMAGES. **Circuitry of a hero 1 robot**. Disponível em: <http://www.gettyimages.com/detail/news-photo/electronic-circuitry-inside-the-hero-1-a-kit-robot-from-the-news-photo/635237421#electronic-circuitry-inside-the-hero-1-a-kit-robot-from-the-heath-picture-id635237421>. Acesso em: 23 dez. 2016.

HACKADAY. **What development board to use?**. Disponível em: <http://hackaday.com/2011/02/01/what-development-board-to-use/>. Acesso em: 23 dez. 2016

HDMI ORG. **Hdmi**. Disponível em: <http://www.hdmi.org/>. Acesso em: 23 dez. 2016.

HOROWITZ, Paul; HILL, Winfield. THE ART OF ELECTRONICS. 3. ed. Cambridge: Mit Press, 2015. 1219 p.

KAMAL, Raj. Embedded Systems: Architecture, Programming and Design. 2. ed. New Delhi: Tata Mcgraw-hill Publishing Company Limited, 2011. 662 p.

LEE, Edward Ashford; SESHIA, Sanjit Arunkumar. **Introduction to Embedded Systems:** A cyber-physical systems approach. California: University Berkley Print, 2012. 519 p.

MCT. **Spi - serial peripheral interface**. Disponível em: <http://www.mct.net/faq/spi.html>. Acesso em: 23 dez. 2016.

MIT. **Microcontrollers**. Disponível em: <http://web.mit.edu/rec/www/workshop/microcontrollers.htm>. Acesso em: 23 dez. 2016.

MUNDO EDUCAÇÃO. **A robotização na produção industrial**. Disponível em: <http://mundoeducacao.bol.uol.com.br/geografia/a-robotizacao-na-producao-industrial.htm>. Acesso em: 31 mai. 2017

MY ELECTRONICS LAB. **Raspberry pi 3 pinout**. Disponível em: <https://www.myelectronicslab.com/tutorial/raspberry-pi-3-gpio-model-b-block-pinout/>. Acesso em: 23 dez. 2016.

MIPI ALLIANCE. **Display and touch**. Disponível em: <http://mipi.org/specifications/display-interface>. Acesso em: 23 dez. 2016.

MICROCHIP, **MCP3004/3008 USER GUIDE.** Disponível em:< https://cdn-shop.adafruit.com/datasheets/MCP3008.pdf> Acesso em: 21 fev 2017

NXP SEMICONDUCTORS, **I2C BUS SPECIFICATION AND USER MANUAL**. Disponível em: <http://www.nxp.com/documents/user_manual/UM10204.pdf> Acesso em: 23 dez 2016

PELGROM, Marcel J. M.. **Analog-to-Digital Conversion.** New York: Springer Science, 2010. 17 p

PROGRAMA DE PÓS GRADUAÇÃO EM ENERGIA. **Fundamentos d robóticae**. Disponível em:<http://pgene.ufabc.edu.br/docentes/riascos/ensino/disciplinas/robotica/fundamentosrobotica>. Acesso em: 23 dez. 2016.

PPGSC. **Linguagens imperativas e funcionais**. Disponível em: <http://www.ppgsc.ufrn.br/~rogerio/material_auxiliar/clp20131_linguagens_imperativas_funcionais.pdf >. Acesso em: 23 dez. 2016.

OPENCV. **About**. Disponível em: <http://opencv.org/about.htm>. Acesso em: 23 dez. 2016.

OPENCV. **Opencv 3.0**. Disponível em: <http://opencv.org/opencv-3-0.html>. Acesso em: 23 dez. 2016.

RADIO ELECTRONICS. **Set down buck regulator/converter**. Disponível em: <http://www.radio-electronics.com/info/power-management/switching-mode-power-supply/step-down-buck-regulator-converter-basics.php>. Acesso em: 21 fev. 2017.

RASPBERRY PI ORG. **Dpi (parallel display interface)**. Disponível em: <https://www.raspberrypi.org/documentation/hardware/raspberrypi/dpi/readme.md>. Acesso em: 23 dez. 2016.

RASPBERRY PI ORG. **The eagerly awaited raspberry pi display**. Disponível em: <https://www.raspberrypi.org/blog/the-eagerly-awaited-raspberry-pi-display/>. Acesso em: 23 dez. 2016.

RASPBERRY PI ORG. **Raspberry pi 3 specs and benchmarks**. Disponível em: <https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/>. Acesso em: 21 fev. 2017.

ROBOTICS AND BIOLOGY LABORATORY. **How can robots succeed in unstructured environments?**. Disponível em: <http://www.robotics.tu-berlin.de/fileadmin/fg170/publikationen_pdf/katz-08b.pdf>. Acesso em: 23 dez. 2016.

ROBOTICS NEWS FOR ACADEMIC AND PROFESSIONALS. **Logimover: a clever approach to pallet moving using distributed robots**. Disponível em: <http://www.hizook.com/blog/2014/07/08/logimover-clever-approach-pallet-moving-using-distributed-robots>. Acesso em: 23 dez. 2016.

ROBOTS. **Robot actuators - different types**. Disponível em: <https://www.robots.com/education/actuators>. Acesso em: 23 dez. 2016.

ROBOT ELECTRONICS. **Emg30**. Disponível em: <http://www.robot-electronics.co.uk/products/drive-systems/motors-brackets-and-wheels/emg30-gearmotor-with-encoder.html>. Acesso em: 23 dez. 2016.

ROBOT ELECTRONICS. **Md25 tech**. Disponível em: <https://www.robot-electronics.co.uk/htm/md25tech.htm>. Acesso em: 21 fev. 2017.

ROBOT ELECTRONICS. **Md25 i2c**. Disponível em: <http://www.robot-electronics.co.uk/htm/md25i2c.htm>. Acesso em: 21 fev. 2017

ROGERS HOBBY CENTER. **A guide to understanding lipo batteries**. Disponível em: <https://rogershobbycenter.com/lipoguide/>. Acesso em: 21 fev. 2017.

SEBESTA, Robert W.. **Concepts of programming languages.** 10. ed. Boston: Pearson Prentice Hall, 2010.

SIEGWART, Roland; NOURBAKHSH, Illah Reza. **Inteligent robotics and autonomous agents.** Cambridge: Mit Press, 2004. 321 p.

STACHNISS, Cyrill. Exploration and Mapping with Mobile Robots. 2006. 244 f. Tese (Doutorado) - Curso de Computer Science, University Of Freiburg, Freiburg, 2006.

STANLEY INNOVATION. **Understanding your environment: an important step in robot navigation**. Disponível em: <https://stanleyinnovation.com/robot-navigation-importance-of-environment/>. Acesso em: 23 dez. 2016.

SZELISKI, Richard. **Computer Vision:** Algorithms and applications. Washington: Springer Science & Businesse Media, 2010. 812 p.

TECH WEB. **Linear regulator basics**. Disponível em: <http://micro.rohm.com/en/techweb/knowledge/dcdc/s-dcdc/01-s-dcdc/78>. Acesso em: 21 fev. 2017.

THRUN, Sebastian; BURGARD, Wolfram; FOX, Dieter. **PROBABILISTIC ROBOTICS.** Cambridge: The Mit Press, 2006. 647 p.

TEXAS INSTRUMENTS, **LM741 OPERATIONAL AMPLIFIER**. Disponível em <http://www.ti.com/lit/ds/symlink/lm741.pdf> Acesso em: 21 fev 2017

THUNDER RC. **Thunder power rc**. Disponível em: <https://system.netsuite.com/core/media/media.nl?id=132007>. Acesso em: 21 fev. 2017.

WIKIPEDIA. **Serial peripheral interface bus**. Disponível em: <https://en.wikipedia.org/wiki/serial_peripheral_interface_bus>. Acesso em: 23 dez. 2016.

XINLIX. **D-phy solutions**. Disponível em: <https://www.xilinx.com/support/documentation/application_notes/xapp894-d-phy-solutions.pdf>. Acesso em: 23 dez. 2016.

YAP, Teddy N.; SHELTON, Christian R.. Slam in large indoor environments with low-cost, noisy, and sparse sonars. **Robotics and automation,** Kobe, v. 1, p.111-222, 12./17. 05. Disponível em: <http://ieeexplore.ieee.org/abstract/document/5152192/>.Acesso em: 23 dez. 2016