UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

THIAGO ALVES LIMA

APLICAÇÃO DO *FRAMEWORK* DE ROBÓTICA ROS NO CONTROLE DE SEGUIMENTO DE TRAJETÓRIA DE UM ROBÔ MÓVEL USANDO SENSOR LIDAR PARA ESTIMAR POSIÇÃO E ORIENTAÇÃO

FORTALEZA

2017

THIAGO ALVES LIMA

APLICAÇÃO DO *FRAMEWORK* DE ROBÓTICA ROS NO CONTROLE DE SEGUIMENTO DE TRAJETÓRIA DE UM ROBÔ MÓVEL USANDO SENSOR LIDAR PARA ESTIMAR POSIÇÃO E ORIENTAÇÃO

Monografia apresentada ao Departamento de Engenharia Elétrica da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Fabrício Gonzalez Nogueira.
Coorientador: Prof. Dr. Bismark Claure Torrico.

FORTALEZA

2017

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

A482a Alves Lima, Thiago.
Aplicação do framework de robótica ROS no controle de seguimento de trajetória de um robô móvel usando sensor lidar para estimar posição e orientação / Thiago Alves Lima. – 2017.
77 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia Elétrica, Fortaleza, 2017.
Orientação: Prof. Dr. Fabrício Gonzalez Nogueira.
Coorientação: Prof. Dr. Bismark Claure Torrico.

1. ROS. 2. Controle de Robô Móvel. 3. Seguimento de Trajetória. 4. Sensor LIDAR. I. Título.
CDD 621.3

THIAGO ALVES LIMA

APLICAÇÃO DO *FRAMEWORK* DE ROBÓTICA ROS NO CONTROLE DE SEGUIMENTO DE TRAJETÓRIA DE UM ROBÔ MÓVEL USANDO SENSOR LIDAR PARA ESTIMAR POSIÇÃO E ORIENTAÇÃO

Monografia apresentada ao Departamento de Engenharia Elétrica da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Bacharel em Engenharia Elétrica.

Aprovada em: ___/___/_____.

BANCA EXAMINADORA

_____
Prof. Dr. Fabrício Gonzalez Nogueira (Orientador)
Universidade Federal do Ceará (UFC)

_____
Prof. Dr. Bismark Claure Torrico (Coorientador)
Universidade Federal do Ceará (UFC)

_____
Prof. Dr. Paulo Peixoto Praça
Universidade Federal do Ceará (UFC)

_____
Eng. Adriano Rodrigues de Paula
Universidade Federal do Ceará (UFC)

Dedico este trabalho a minha mãe Antonia Freitas Alves Lima por ter sido a maior apoiadora dos meus estudos e a minha família por serem minha base sempre.

AGRADECIMENTOS

A Deus, por ter me abençoado com saúde, disposição e inúmeras oportunidades valiosas.

A minha mãe, Antonia, que é meu maior exemplo de persistência e de como enfrentar os desafios diários da vida com coragem e alegria, e que me ensinou a não desistir, mesmo nos momentos de maior dificuldade.

Ao meu pai, Antonio, e minhas irmãs, Carla e Glaubervânia, e meu cunhado Francisco, pelo incentivo diário e por compartilharem a alegria das minhas conquistas.

Ao meu primo, Junior, por ter partilhado da minha paixão pela Engenharia desde muito cedo, e ter me apresentado o ROS.

Aos meus amigos, Mateus e André, pela paciência em ouvir minhas reclamações diárias e por nunca desistirem de me fazer sair de casa e da faculdade para nos divertir nos momentos que mais precisei.

Aos meus amigos, Jander, Daniel e Dênio, que me acompanharam nessa jornada de noites acordado e imensas filas no restaurante universitário. Sou grato por contribuírem imensamente para o meu aprendizado e para o sucesso dos projetos que desenvolvemos juntos, sempre com muito bom humor e companheirismo.

Ao meu amigo, Nadson, por ter dividido e incentivado o meu sonho de realizar intercâmbio, ter sido meu primeiro professor de programação e ter generosamente colocado sua casa a disposição como ponto de apoio durante a graduação.

Ao meu amigo e cunhado, Ricardo, que me apoiou durante os meus estudos e me ajudou a atingir meu objetivo de participar do Ciência sem Fronteiras.

A minha amiga, Ana, que juntamente ao Nadson, foram meus companheiros de pesquisa nos Estados Unidos, no período em que iniciamos nosso trabalho com o robô Nanook da NASA, quando o meu interesse por robótica despertou.

À professora Natália, que depositou confiança em mim e escreveu a melhor carta de recomendação para intercâmbio que já li.

Ao professor Jae-Do Park, da *University of Colorado Denver*, que foi meu orientador no grupo de pesquisa *Energy Conversion Force*, quando tive meu primeiro contato com o universo da pesquisa.

Ao meu amigo e *roomate*, Antonio, ao Marco Figueiredo, aos professores Patrick Stakem e Alex Antunes, e ao Michael Comberiate, por terem me proporcionado a oportunidade de trabalhar no *Bootcamp* de Engenharia Aeroespacial da *Capitol Technology University*, quando

tive a oportunidade de realizar meu sonho de conhecer a NASA e aprender sobre engenharia aeroespacial.

Ao meu amigo e companheiro de laboratório, Marcus Davi, com quem desenvolvi a pesquisa que originou esse trabalho e nos rendeu uma publicação. Não teria chegado até aqui sem a sua formidável contribuição e companheirismo nas tardes de desenvolvimento no GPAR.

Ao meu orientador, Prof Dr. Fabrício Nogueira, e co-orientador, Prof Dr. Bismark Torrico, que deram todo suporte necessário para o desenvolvimento deste trabalho, e que souberam transmitir brilhantemente seus conhecimentos na área de controle, tanto em sala de aula, como no laboratório.

Aos meus professores, por terem me repassado o conhecimento tão precioso, muitas vezes demonstrando paixão pela área, e me mostrando que escolhi a profissão certa.

À Universidade Federal do Ceará, à *University of Colorado Denver* e à *Capitol Technology University* por terem me proporcionado a estrutura necessária para o meu desenvolvimento acadêmico e profissional, com diversas oportunidades e atividades que tornaram minha experiência na universidade completa.

Aos meus professores no estágio, André, Carlos, Vianey e Nonato, que me transmitiram conhecimentos únicos, advindos de muita experiência e que serão fundamentais na minha atuação no campo profissional.

À Capes e ao CNPq, que financiaram o meu intercâmbio acadêmico nos Estados Unidos, tornando realidade o sonho de estudar fora, que me proporcionou uma experiência única que trouxe desenvolvimento profissional, cultural e pessoal.

"*Two roads diverged in a wood, and I—I took the one less traveled by, and that has made all the difference.*"

Robert Frost

RESUMO

Neste trabalho é demonstrado a integração de um sensor LIDAR aplicando o *framework* de robótica *Robot Operating System* (ROS) no computador Raspberry PI para estimar a posição e orientação (postura) de um robô móvel com rodas. A medição é feita com o objetivo de implementar uma técnica de controle de seguimento de trajetória. Essa técnica é comparada com o método convencional de odometria em que os dados de *encoders* são utilizados para estimar a postura do robô. Neste trabalho também é apresentada a descrição do robô móvel e sua modelagem matemática. Dois controladores em cascata foram utilizados para implementar o controle de seguimento de trajetória. Um controlador cinemático gera as referências para o controlador de velocidade dos motores mais interno. São apresentados resultados experimentais em um ambiente fechado para trajetórias sem obstáculos.

Palavras-chave: ROS. Controle de Robô Móvel. Seguimento de Trajetória. Sensor LIDAR.

ABSTRACT

In this work it is shown the integration of a LIDAR sensor using the robotics framework Robot Operating System (ROS) in the Raspberry PI computer in order to estimate position and orientation (posture) of a wheeled mobile robot. The measuring is done with the goal of implementing a trajectory tracking control technique. This technique is compared with the conventional odometry method where the encoders' data are used for posture estimation. The paper also presents the description of the robot as well as its mathematical modeling. Two controllers in cascade were used to implement the tracking control. A kinematics controller generates the references for internal motor speed controllers. Experimental results in an indoor environment are presented for trajectories with no obstacles.

Keywords: ROS. Control of mobile robots. Trajectory tracking. LIDAR sensor.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CC	Corrente Contínua
NI	*National Instruments*
IMU	Inertial Measurement Unit
GPAR	Grupo de Pesquisa em Automação e Robótica
RAM	*Random Access Memory*
PC	*Personal Computer*
SLAM	*Simultaneous Localization and Mapping*
LIDAR	*Light Detection And Ranging* (Detecção e Telemetria de Luz)
USB	*Universal Serial Bus*
TOF	*Time-Of-Flight*
VNC	*Virtual Networking Computing*
UART	*Universal Asynchrounous Receiver/Transmiter* (Receptor/Transmissor Universal Assíncrono)
UFC	Universidade Federal do Ceará
ROS	*Robot Operating system*
USB	*Universal Serial Bus* (Porta Universal)
VCC	Tensão (Volts) em corrente contínua

SUMÁRIO

1 INTRODUÇÃO

A medição da posição e orientação de um robô móvel não-holonômico é um desafio importante a superar quando o robô móvel precisa seguir uma rota desejada em um ambiente previamente desconhecido. Métodos estimativos são constantemente utilizados para obter essa informação, como por exemplo o já bem conhecido método da odometria, onde os dados provenientes de *encoders* são usados para calcular as velocidades linear e angular do robô, e através do modelo cinemático do robô é possível estimar sua posição e orientação (postura). Este método simples apenas funciona quando o robô não está sujeito a distúrbios como deslizamento e derrapagem nas rodas. Portanto, a aplicabilidade desta técnica no mundo real demonstra limitações.

Ao longo dos últimos anos, um intensivo esforço de pesquisa foi aplicado nessa área. Uma das soluções mais buscadas foi a aplicação de fusão sensorial. Fuke e Kotrov (1996) apresentaram a utilização de um filtro de Kalman para realizar a fusão sensorial entre um acelerômetro, giroscópio e *encoders*. K. Srinivasan e J. Gu (2007), aplicaram fusão sensorial entre um sensor ultrassônico, *encoder* e giroscópio para estimar a localização de um robô e evitar obstáculos. Mais recentemente, H. Y. Chung, C. C. Hou e Y. S. Chen (2015), utilizaram um filtro de Kalman para fazer a fusão de dados obtidos por um magnetómetro e um giroscópio e lógica Fuzzy para estimar a localização de um robô móvel.

Existem diferentes abordagens para resolver o problema do erro de leitura de sensores causada pela derrapagem e escorregamento das rodas de um robô. Uma possível solução seria a inclusão do modelo matemático desses distúrbios no projeto dos controladores, considerando diferentes perturbações para diferentes tipos de terrenos (WANG e LOW, 2008). É possível ainda utilizar pneus altamente aderentes para mitigar o impacto dessas perturbações.

Nesse trabalho é proposto a utilização da tecnologia LIDAR (*Light Detection and Ranging*), por meio de um sensor laser, para estimar a postura de um robô móvel em tempo real usando um algoritmo SLAM (*Simultaneous Localization and Mapping*) e fechar a malha de controle de seguimento de trajetória.

Este sensor gira em uma linha horizontal e calcula as distâncias entre o centro do sensor e qualquer objeto atingido pelos raios infravermelhos. Enquanto o robô se move por ambientes não conhecidos previamente, um algoritmo de localização e mapeamento pode estimar, com alta velocidade e precisão, a posição e orientação do robô móvel. O algoritmo de localização e mapeamento foi utilizado através do *framework* de robótica *Robot Operating System* (ROS), embarcado no microcomputador Raspberry PI modelo 2B.

A postura estimada do robô pode então ser usada como realimentação na malha de controle mesmo se o robô deslizar ou derrapar, já que na presença de distúrbios, o sensor laser medirá um novo vetor de distâncias, e, portanto, o algoritmo de localização no plano 2D vai detectar uma nova postura. Ao usar este método para estimar a postura do robô, foi possível aplicar o controlador de seguimento de trajetórias demonstrado por Klančar, Matko e Blažič (2005), utilizando o sensor laser ao invés de *encoders* para fechar a malha de controle.

1.1 Justificativa

Robôs móveis estão cada dia mais presentes e possuem novas funções na indústria. Com o advento da chamada quarta revolução industrial, também conhecida como Indústria 4.0, tornou-se necessário que robôs móveis sejam capazes de se localizar e seguir uma trajetória pré-definida em diversos ambientes com bastante precisão para possibilitar, por exemplo, a total automação da estocagem de produtos em uma fábrica. Essas habilidades também são importantes em outras aplicações, como no auxílio ao resgate de vítimas em ambientes atingidos por desastres naturais como furacões e terremotos.

O uso de *encoders* para estimar a postura de um robô móvel por meio do acúmulo do número de vezes que a roda gira frequentemente é inadequado, pois gera erros devidos a impossibilidade de contabilizar o movimento do robô causado por distúrbios como o escorregamento e deslizamento. Por outro lado, a utilização de informação multissensorial exige um modelo preciso dos sensores e extensivo uso de filtros para fusão sensorial (MARÍN, L.; VALLÉS, M.; SORIANO, Á.; VALERA, Á.; ALBERTOS, P., 2014). É necessário, dessa forma, apresentar novas técnicas que possam mitigar ou eliminar o erro da medição da postura de robô móveis para que os sistemas de controle sejam capazes de acompanhar as novas demandas de utilização desses robôs.

1.2 Objetivos

Esse trabalho tem como principal objetivo comparar a utilização de dados obtidos por um sensor laser em substituição a *encoders* para realimentar a malha de controle de postura de um robô móvel no seguimento de diferentes trajetórias, por meio da utilização de um algoritmo de localização e mapeamento simultâneos. Para isso, o desempenho das duas estratégias (utilização de *encoders* e utilização de sensor laser) foram avaliadas através de testes com diferentes trajetórias de referência. Os testes foram conduzidos em um ambiente fechado

onde as rodas do robô estiveram sujeitas a distúrbios.

Em adição, este trabalho tem como objetivo secundário demonstrar como implementar o sistema de medição proposto aplicando o *framework* de robótica *Robot Operating System* (ROS), e como integrar a plataforma desenvolvida a um robô móvel real, que antes tinha realimentação de postura estimada apenas por meio de odometria utilizando *encoders*.

1.3 Contribuições

Esse trabalho demonstra uma metodologia de aplicação de sensores laser medidores de distância para controle na robótica móvel. Para isso, são demonstradas todas as etapas de desenvolvimento do projeto, desde o emprego da plataforma *raspbian* (sistema operacional baseado em Linux do microcomputador Raspberry PI) e do *framework* de robótica *Robot Operating System* (ROS) para obtenção da postura do robô até a descrição matemática do mesmo.

Também se destaca como contribuição deste trabalho a propagação da robótica como um laboratório de aprendizado que deve ser conhecido e empregado pelos alunos e professores nos cursos de controle nas universidades brasileiras. Por fim, a divulgação do ROS como uma plataforma de desenvolvimento de *software* para robótica no Departamento de Engenharia Elétrica da UFC (Universidade Federal do Ceará).

1.4 Organização do Texto

Este texto foi organizado em capítulos, proporcionando um aprendizado construtivo dos assuntos abordados. Esta monografia é dividida em seis capítulos conforme a seguinte estrutura adotada:

- Capítulo 1: Este capítulo apresenta uma introdução deste trabalho, esclarecendo a motivação de sua elaboração, objetivos almejados e contribuições decorrentes.
- Capítulo 2: Este capítulo traz a fundamentação teórica dos componentes do sistema de medição proposto. Assim, são abordados a tecnologia de medição laser, o computador Raspberry PI e o *framework* de robótica *Robot Operating System* (ROS).
- Capítulo 3: Este capítulo descreve o robô móvel utilizado durante os experimentos e a integração da nova plataforma de medição de posição e orientação proposta.

- Capítulo 4: Este capítulo apresenta a modelagem matemática do robô e aborda a arquitetura do controlador de seguimento de trajetórias.
- Capítulo 5: Este capítulo apresenta uma discussão geral dos testes experimentais e dos resultados obtidos.
- Capítulo 6: Este capítulo contém conclusões e propostas para trabalhos futuros.

2 SISTEMA DE LOCALIZAÇÃO DO ROBÔ MÓVEL

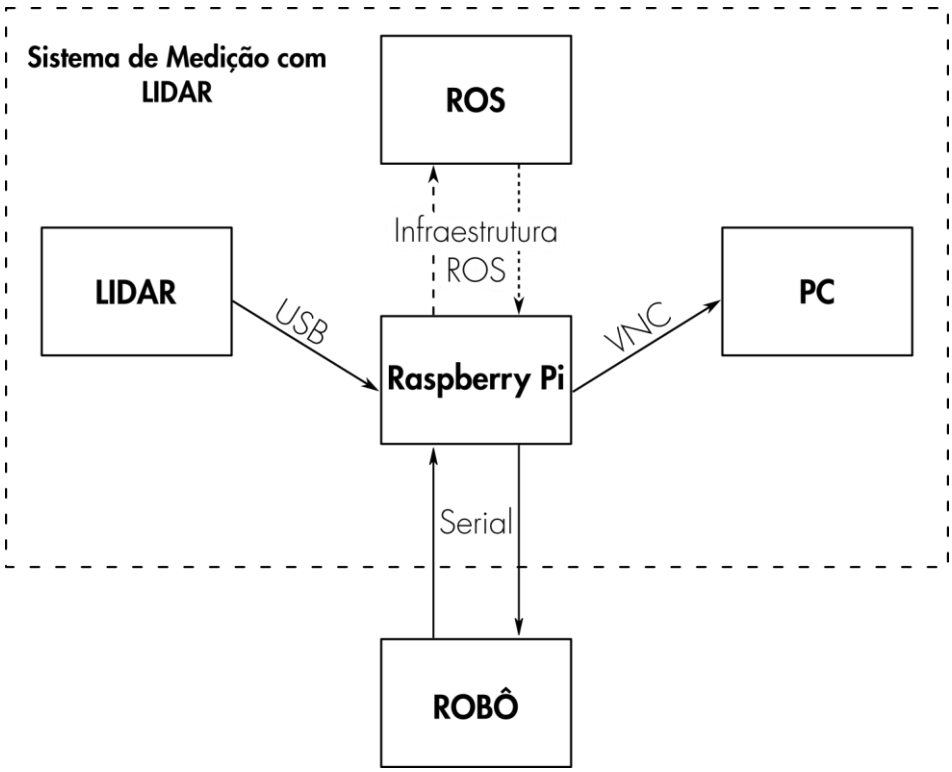
Embora a nomenclatura e concepção modernas da robótica só foram apresentadas no último século, conceitos primitivos que lembram os robôs atuais têm sido objeto de fascinação da classe literária há muitos anos. Prova disso são as interpretações primitivas de servos artificiais autônomos, que constam de longa data na cultura humana. Na literatura grega, por exemplo, há referências a imaginadas mesas autônomas de três pernas criadas pelo deus Hephaestus.

O campo de estudo da robótica móvel é relativamente jovem. É reconhecidamente um laboratório interdisciplinar, que conta com a contribuição de pesquisadores das engenharias mecânica, elétrica e eletrônica, e da ciência da computação. Robôs móveis são máquinas capazes de se locomover por um ambiente, interagir com o meio e realizar tarefas de maneira inteligente, com mínima ou nenhuma intervenção humana. Para realizar estas tarefas com precisão, os robôs devem contar com sistemas de sensoriamento que permitem o melhor reconhecimento possível do ambiente, e ferramentas computacionais compatíveis para o seu processamento (SIEGWART, ROLAND; NOURBAKHSH, ILLAH R.; SCARAMUZZA, DAVIDE, 2011).

Neste trabalho, propõe-se a utilização de ferramentas computacionais avançadas para melhoria do sistema de controle de um robô móvel por meio da inserção de um sistema de medição LIDAR (*Light Detection And Ranging*) e do uso do *framework* de robótica ROS. Os grandes blocos componentes do novo sistema proposto, delimitados pela linha pontilhada na Figura 1, serão detalhados neste capítulo. A proposta é integrar este novo bloco a um robô móvel, o qual será descrito no capítulo 3.

O sistema de medição de postura utilizando sensor LIDAR é composto por quatro grandes blocos, mostrados na Figura 1. No centro do sistema, encontra-se a unidade de processamento de dados, o microcomputador Raspberry PI, que lê os dados do sensor LIDAR via porta USB. O processamento dos dados no Raspberry PI é feito utilizando a infraestrutura oferecida pelo *framework* de robótica ROS. O *framework* é uma camada de abstração de software, um meta-sistema operacional, que utiliza como base o sistema operacional do microcomputador. O quarto bloco componente é um PC (*personal computer*) que realiza acesso remoto ao *desktop* do Raspberry PI, por meio do software VNC (*virtual computer networking*). Esse sistema de medição será integrado ao robô real utilizando comunicação via porta serial entre o Raspberry PI e o microcontrolador já embarcado previamente no robô móvel.

Figura 1 – Diagrama de blocos do sistema de medição proposto para adição ao robô móvel.



Fonte: Elaborado pelo autor.

2.1 Sensor LIDAR

A tecnologia LIDAR (*Light Detection And Ranging*) é aplicada por dispositivos capazes de medir distâncias até um objeto ou superfície os iluminando com raios laser (luz infravermelha invisível) e processando a onda de resposta refletida. Os instrumentos mais modernos possuem alta precisão e capacidade de medir distâncias a diversos pontos em alta velocidade. O sensor laser medidor de distância possui diversas aplicações, entre elas o levantamento topográfico de regiões em nuvens de pontos (para sensores de alto alcance), automação de processos industriais de armazenamento e aplicações balísticas militares (AMANN, BOSCH, MYLLYLA, RIOUX, 2001).

2.1.1 Princípio de Funcionamento

Sensores laser medidores de distância funcionam emitindo raios laser invisíveis em um objeto ou superfície, e processando a onda de resposta refletida pelo objeto alvo. Dois princípios comuns de funcionamento para o sensor são explicados a seguir.

A metodologia mais simples, chamada de *time-of-flight* (TOF), consiste em medir o tempo demorado para o pulso de energia viajar do transmissor até o objeto observado e então de volta ao transmissor (t_d). O raio laser viaja na velocidade da luz (aproximadamente $c = 30\ cm/ns$), assim é possível medir a distância ao alvo utilizando a equação (1):

$$D = \frac{c * t_d}{2} \tag{1}$$

Um segundo método, derivado do TOF, permite calcular a distância sem medição direta do tempo de viagem de ida e volta entre a emissão do pulso de luz infravermelho e o retorno do pulso resultante de sua refletância em um objeto (H. Y., CHUNG; C. C, HOU; Y. S, CHEN, 2015). Esse método, chamado de *phase-shift*, ou "deslocamento de fase" em tradução literal, consiste em excitar o diodo emissor de luz infravermelho com uma onda senoidal de frequência f_r. Após a reflexão no alvo, um fotodiodo coleta parte da onda, permitindo a dedução da distância D através da equação (2):

$$D = \frac{1}{2} c \frac{\Delta\varphi}{2\pi f_r} \tag{2}$$

Onde $\Delta\varphi = 2\pi f_r t_d$. Assim, é possível calcular a distância ao alvo sem medir diretamente o tempo de viagem da onda. Na Figura 2, é mostrado o diagrama de funcionamento desta técnica, que é utilizada para obter medições com resoluções milimétricas, em distâncias que variam até 20 metros (AMANN, BOSCH, MYLLYLA, RIOUX, 2001).

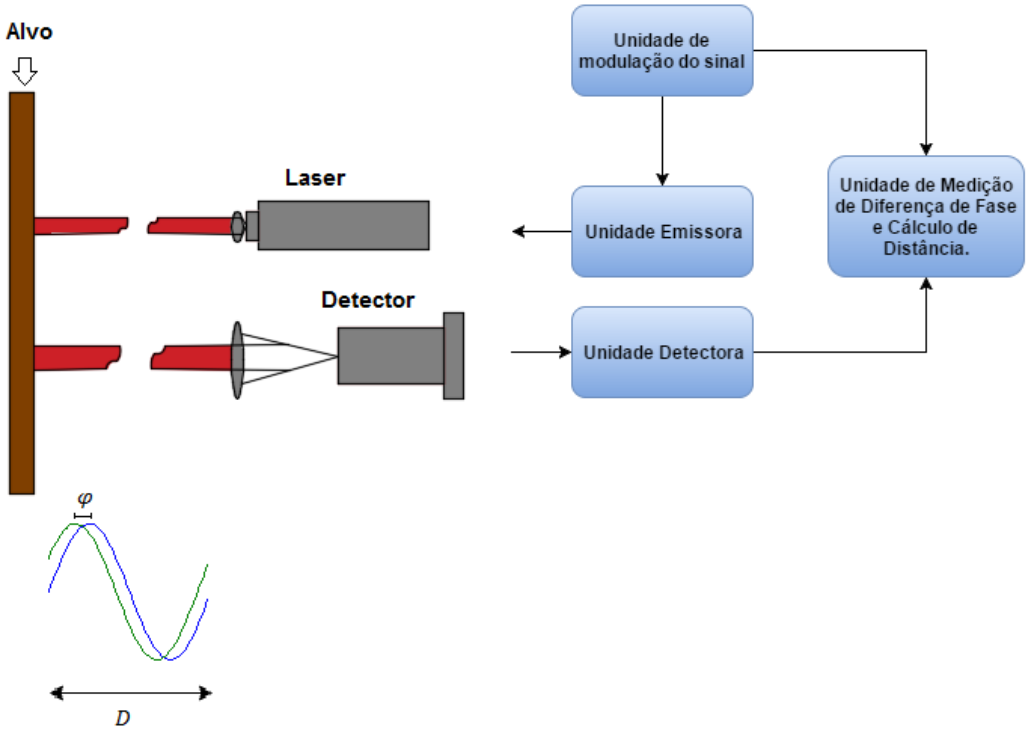
É importante ressaltar que para ambos os casos, é necessário que o sensor conte com uma unidade de processamento própria, capaz de calcular e transmitir os dados obtidos.

2.1.2 Especificações do Sensor LIDAR Utilizado

Neste trabalho, foi utilizado o sensor laser Hokuyo URG-04LX-UG01, que pode ser visto na Figura 3. O sensor tem área de escaneamento de 240 °, com precisão angular de 0.3515625 °. Ou seja, a cada escaneamento completo, o sensor obtém um vetor com 683 medidas de distância do ambiente, pois $Np = (240°/0.3515625°) + 1 = 683$. Além disso, esse sensor é capaz de realizar um escaneamento completo a cada 100 ms, ou seja, a uma taxa

de 10 Hz. A Figura 4 exemplifica a área detectável pelo sensor. A especificação completa do sensor laser utilizado nos experimentos é mostrada no Anexo A.

Figura 2 – Diagrama de blocos de funcionamento sensor laser – princípio de *phase-shift*.



Fonte: Elaborado pelo autor.

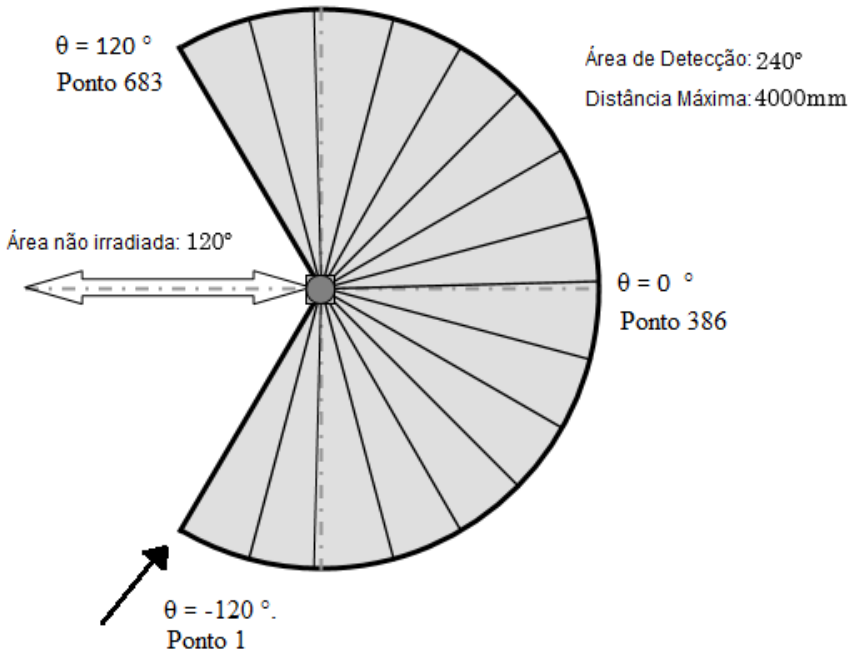
Figura 3 – Sensor laser Hokuyo URG-04LX-UG01.

Fonte: https://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx_ug01.html.

Dentro do sensor, um microprocessador calcula as distâncias aos 683 pontos utilizando o princípio de *phase-shift*, explicado na seção anterior. Sua precisão é bastante alta, detectando objetos a distâncias que vão de 2 centímetros até 4 metros. O protocolo de comunicação do sensor está disponível no site do fabricante e utiliza o padrão SPI 2.0.

O laser possui uma interface de comunicação com o *Robot Operating System* (ROS), como será explicado adiante, possibilitando a leitura dos dados no Raspberry PI 2B sem necessidade de uso do *software* do fabricante.

Figura 4 – Área de detecção do sensor laser Hokuyo URG-04LX-UG01.



Fonte: Adaptada de https://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx_ug01.html.

2.2 Computador Raspberry PI

Para leitura e processamento dos dados do sistema laser, foi utilizado o microcomputador Raspberry PI, modelo 2B, que pode ser visto na Figura 5. Este computador foi desenvolvido pela Raspberry PI Foundation, no Reino Unido, com o intuito de promover o ensino de computação para estudantes. Ele cabe na palma da mão, tem um bom poder de processamento, e baixo consumo de energia, o que o tornou uma opção bastante popular entre os desenvolvedores de *software* para robótica.

2.2.1 Informações de Hardware

O microcomputador utiliza um processador ARM Cortex-A7 quad-core de 900 MHz, e possui memória RAM de 1 GB. Também possui 4 portas USB, porta Ethernet, HDMI, e 40 pinos de comunicação de uso geral (GPIO).

Neste projeto, são utilizadas duas portas USB, sendo uma para uso do adaptador Wifi, que permite acesso à internet e acesso via computador remoto utilizando VNC, e a segunda para comunicação e recepção de dados do sensor LIDAR, como mostrado no diagrama de blocos do sistema de medição. Dentre os pinos de GPIO, são utilizados os 3 pinos para comunicação serial, que permitem transmitir a posição e orientação do robô calculadas pelo algoritmo SLAM 2D para o microcontrolador que embarca os códigos de controle de seguimento de trajetória.

Figura 5 – Computador Raspberry PI, Modelo 2B.

Fonte: http://www.materiel.net/barebone/raspberry-pi-type-b-106574.html.

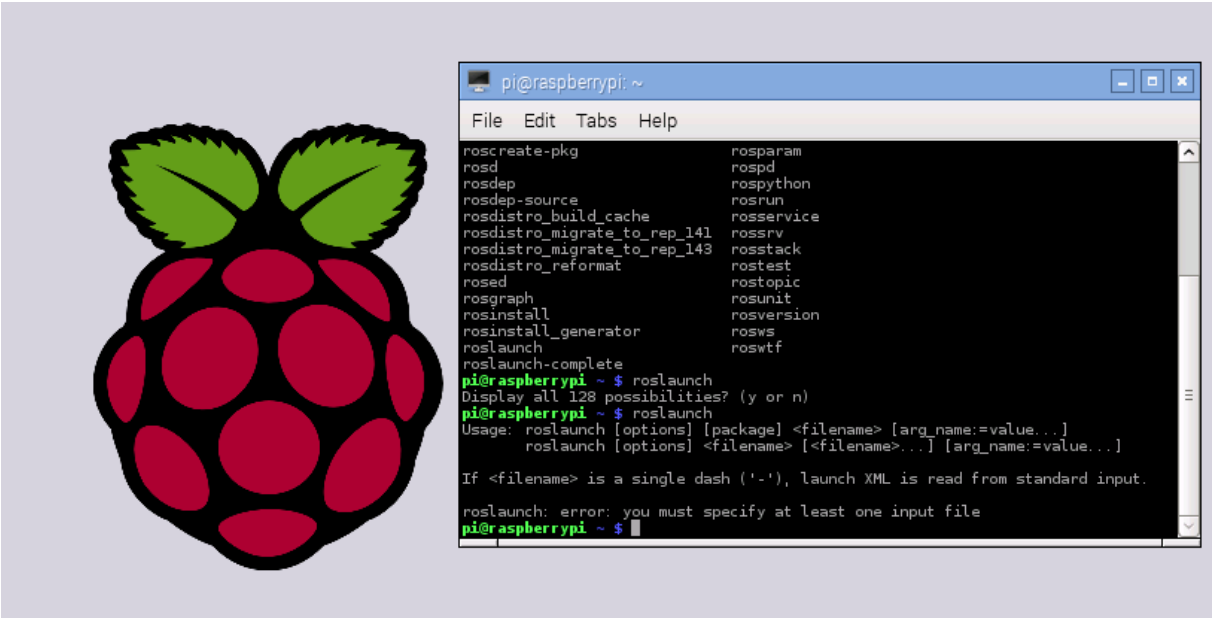
2.2.2 Sistema Operacional

O sistema operacional utilizado pelo Raspberry PI, chamado de Raspbian, é uma versão otimizada de Debian Linux oferecida pela fundação para utilização no dispositivo. Linux

é um dos sistemas operacionais mais utilizados para desenvolvimento de *software* de robótica no mundo, por ser de código livre (*Open Source*). Por ser o único sistema operacional que atualmente suporta o *framework* de robótica ROS, tornou-se fundamental para o desenvolvimento desta pesquisa o aprendizado de linha de comando em Linux.

A linha de comando do Raspberry PI é mostrada na Figura 6 (tela com fundo preto, ao lado da framboesa que simboliza a fundação Raspberry PI). A linha de comando é utilizada para instalar módulos, navegar por pastas e executar programas. Uma grande variedade de linguagens de programação é suportada pelo Raspberry PI, para isso somente é necessário instalar os pacotes que dão suporte a sua execução. Neste trabalho, opta-se por utilizar a linguagem de programação Python para desenvolvimento dos programas no ROS para tratamento dos dados do sensor LIDAR e envio da posição e orientação do robô para o microcontrolador previamente instalado no robô móvel.

Figura 6 – Terminal de comando no sistema operacional Raspbian.



Fonte: Elaborado pelo autor.

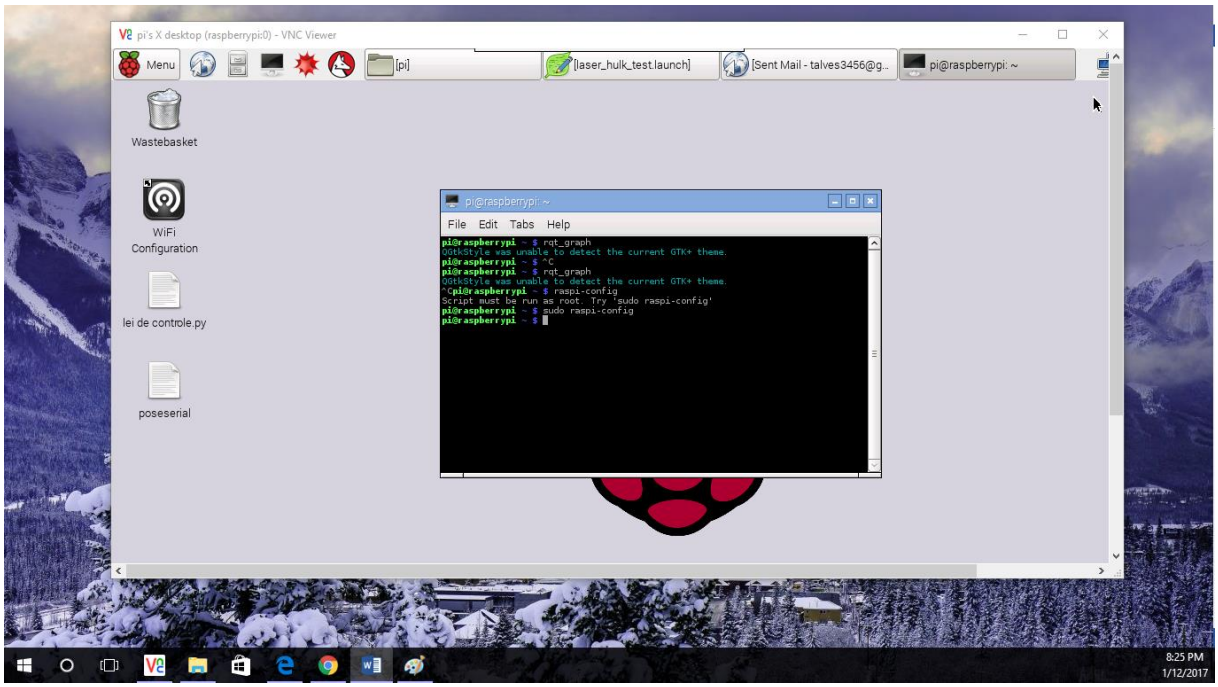
2.3 Acesso Remoto via VNC

O primeiro passo para preparar o Raspberry PI para ser embarcado no robô móvel é a instalação de um sistema de compartilhamento que permite acesso remoto. Desta forma, é possível desenvolver os programas necessários para interface e supervisão de dados do microcomputador a partir de um computador remoto de supervisão. O sistema de acesso remoto

VNC Viewer é utilizado para esse propósito. Apesar de que, por padrão, o servidor VNC já vir instalado no Raspberry PI, é necessário realizar sua ativação no menu de configurações, utilizando o caminho “Menu > Preferences > Raspberry Pi Configuration > Interfaces”.

Após configurar o acesso remoto no Raspberry PI, é necessário fazer o download do aplicativo VNC Viewer para o computador do qual se deseja realizar o acesso. A Figura 7 mostra a janela de acesso remoto a partir de um computador que usa Windows 10. O uso da ferramenta de acesso remoto é importante para acompanhamento dos dados e *debug* das funcionalidades embarcadas no microcomputador de forma prática. Caso contrário, seria necessário conectar teclado, *mouse* e monitor ao Raspberry PI, o que seria incompatível com sua aplicação em um robô móvel.

Figura 7 – Janela de acesso remoto ao Raspberry PI a partir de um computador Windows.



Fonte: Elaborado pelo autor.

2.4 *Robot Operating System* (ROS)

O ROS é, atualmente, um dos *frameworks* de robótica mais utilizado no mundo. Um *framework* é uma estrutura de *software* que contém códigos base que permitem que um programador desenvolva aplicações mais complexas sem a necessidade de desenvolver novamente as estruturas mais simples sempre que for escrever uma nova aplicação. Assim, um

framework é uma camada de abstração de *software* que é executado sobre um sistema operacional completo, não devendo ser confundido com o mesmo.

Consideremos a seguinte analogia: suponhamos que alguém necessite cortar um pedaço de papel com dimensões de 10 cm por 10 cm. Essa seria uma tarefa relativamente simples de ser realizada pois, com a ajuda de uma régua e uma tesoura, facilmente seria possível recortar o pedaço de papel nas dimensões desejadas. No entanto, imaginemos agora a árdua tarefa de ter que recortar 500 folhas de papel nessas dimensões. Seria exaustivo pegar uma régua e medir um quadrado de 10 cm por 10 cm repetidamente por 500 vezes. Ao invés disso, seria mais inteligente criar uma moldura (em inglês, *framework*) de madeira com essas dimensões, e utilizar apenas uma ferramenta de corte para realizar o trabalho, sem necessidade de realizar a operação de medição novamente.

Voltando ao mundo do *software*, um *framework* disponibiliza estruturas genéricas de *software* (as molduras da analogia anterior) que podem ser acrescidas de novas funcionalidades ou modificadas pelo código do usuário. A característica de promover *software* reutilizável torna os *frameworks* de robótica uma opção atraente como plataforma de desenvolvimento de novas aplicações. Neste trabalho, o *framework* ROS permitiu a fácil integração de uma plataforma de medição mais precisa, com utilização do sensor LIDAR, para aplicação no sistema de controle de seguimento de trajetória de um robô móvel.

Figura 8 – Logo do *framework* de robótica ROS.

Fonte: http://www.udoo.org/ros-on-udoo-neo-dual-quad/.

2.4.1 Estrutura do ROS

Diversos *frameworks* de robótica já foram desenvolvidos e estão disponíveis para uso, mas nenhum deles se tornou tão popular quanto o ROS nos últimos anos. O principal objetivo do ROS é dar suporte à reutilização de códigos na robótica e pesquisa. Conforme descrito no próprio site do ROS, o *framework* disponibiliza as seguintes funcionalidades:

> Abstração de *hardware*, controle de dispositivo em baixo nível, implementação de funcionalidades comumente usadas, troca de mensagens entre processos, e gerenciamento de pacotes. Também provê ferramentas e bibliotecas para obter, construir, escrever e executar códigos em vários computadores (ROS.wiki, 2014, tradução nossa).

O ROS foi desenvolvido por um grupo de alunos da *Stanford University*, nos Estados Unidos, baseado nas seguintes filosofias descritas abaixo. (QUIGLEY, GERKEY, CONLEY, FAUST, FOOTE, LEIBS, ERIC BERGER, WHEELER, NG, 2009).

- Ser *peer-to-peer*: Sistemas que utilizam ROS consistem de um número de processos (que podem estar rodando em computadores diferentes) que estão conectados em uma estrutura *peer-to-peer* (par a par). Isso significa que o ROS disponibiliza uma estrutura que permite que processos troquem informações facilmente, sem que o usuário necessite desenvolver esta funcionalidade, como será explicado mais adiante.

- Ser multilíngue: Isso significa que o ROS foi desenvolvido com o objetivo de ser neutro em relação a linguagens de programação. O ROS suporta, atualmente, diversas linguagens de programação, como Python, C++, e Octave. Essa flexibilidade permite que usuários escolham escrever programas na linguagem que já possuem maior afinidade. Além disso, por ser um sistema *peer-to-peer*, o ROS permite que processos escritos em linguagens de programação diferentes troquem mensagens facilmente, sem necessidade de utilização de estrutura adicional de *software*.

- Ser *tools-based*: O ROS apresenta uma arquitetura de microkernel, o que resulta na modularização do *framework* em diversos blocos pequenos usados para construir e rodar os vários componentes do ROS.

- Ser *thin*: A grande maioria dos *softwares* desenvolvidos para robótica se tornam não reutilizáveis fora do projeto para o qual foram escritos. Para combater esta tendência, o ROS estimula que o desenvolvimento de algoritmos e drivers ocorra em bibliotecas individualizadas, permitindo que o reuso de código alcance patamares além do objetivo inicial. O ROS reusa códigos de vários outros projetos *open-source*, como algoritmos de visão computacional do OpenCV e simuladores do projeto Player.

- Ser livre e *open-source*: Todo o código fonte do ROS está disponível publicamente. Essa característica vem da crença de seus desenvolvedores de que somente uma plataforma totalmente livre é capaz de oferecer suporte completo ao desenvolvimento de

aplicações. Isso é particularmente verdade quando vários programas são executados em paralelo. A licença do ROS permite o desenvolvimento de projetos com fins comerciais e não-comerciais.

2.4.2 Principais Conceitos do ROS

Os conceitos fundamentais para utilização e desenvolvimento do ROS são pacotes (*packages*), nós (*nodes*), mensagens (*messages*), tópicos (*topics*), assinaturas (*subscriptions*) e publicações (*publications*).

Um pacote é uma coleção coerente de arquivos, normalmente incluindo ambos executáveis e arquivos de suporte, que servem um propósito específico.

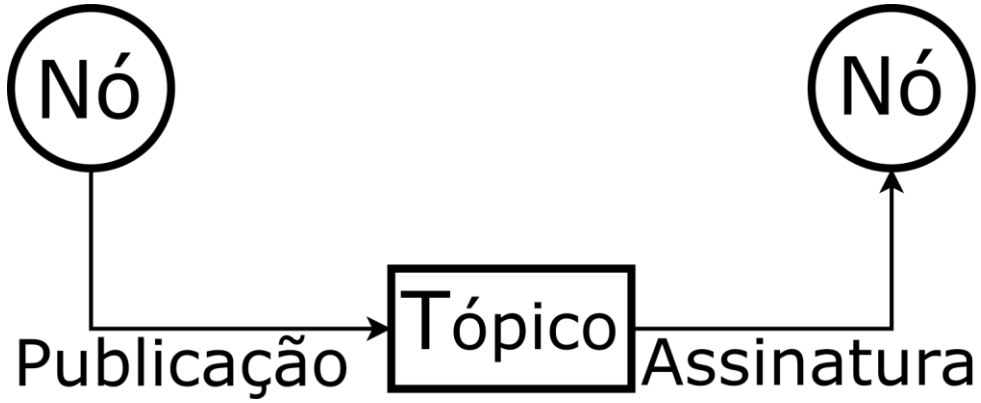
Nós são os executáveis, ou seja, os processos que efetuam computação. Como explicado na seção anterior, o ROS foi criado com a filosofia de modulação de processos. Dessa forma, um nó pode ser visto efetivamente como um módulo de *software* que executa uma função específica. Em geral, um sistema no ROS é composto por diversos nós que processam informações em paralelo e trocam dados por meio do envio de mensagens.

Mensagens são estruturas de dados pré-definidas. No ROS, existem diversos tipos de dados de mensagens, baseados em formatos de dados primitivos como inteiros, booleano, pontos flutuantes e *strings*. Também são suportados vetores de dados primitivos. Mensagens podem ser compostas de outras mensagens, ou partes de outras mensagens.

Para enviar uma mensagem, um nó a publica para um tópico. Um tópico é um nome utilizado para identificar um ponto de entrega e envio de mensagens. É importante notar que um nó não envia mensagens diretamente para outro nó, e tampouco sabe quais nós utilizam estes dados. Os nós interessados em um certo tipo de informação publicada devem se tornar assinantes do tópico apropriado. A troca de informações via estrutura de tópicos é exemplificada na Figura 9.

Podem existir diversos nós publicadores e assinantes para um único tópico. Também é possível que um único nó seja assinante e/ou publique para diversos tópicos. Pelo fato de publicadores e assinantes não saberem da existência um do outro, existe um desacoplamento da geração de informação e do seu consumo.

Figura 9 – Troca de mensagens via tópico no ROS.



Fonte: Adaptada de http://wiki.ros.org/ROS/Concepts.

2.4.3 Definição do Pacote Urg_Node

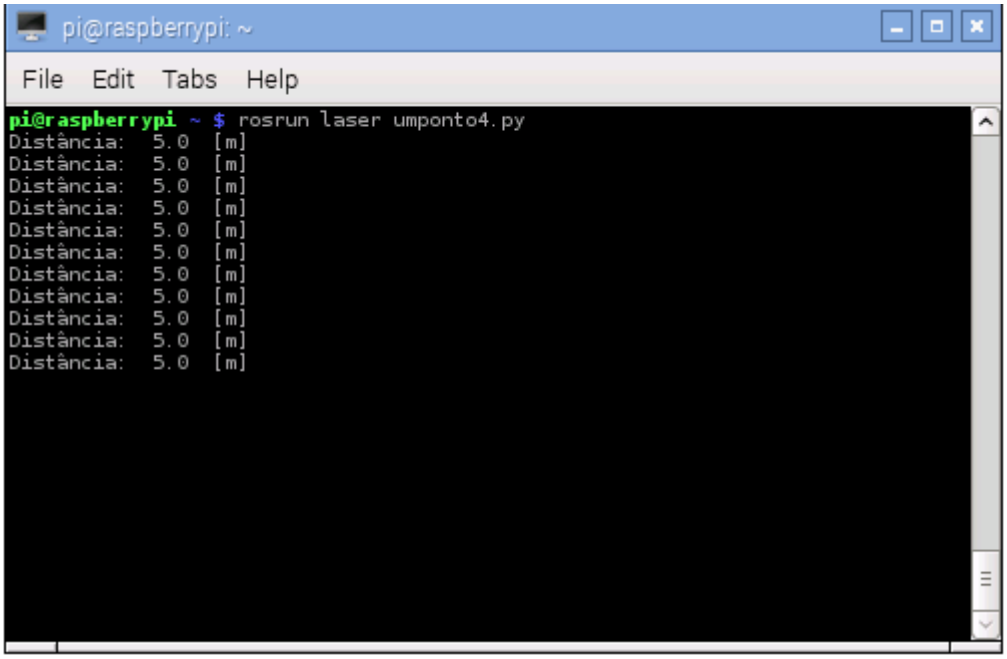
O pacote urg_node oferece um driver de comunicação com o sensor laser Hokuyo apresentado na seção 2.1.2. O pacote oferece um nó, de mesmo nome, que faz a leitura dos dados do sensor via porta USB. Após a leitura dos dados, que acontece a uma taxa de 10 Hz, o nó "/urg_node" publica um vetor de dados com todos os 683 pontos de distância medidos pelo sensor para o tópico "/scan". Dessa forma, roboticistas podem aproveitar da característica de modularização de *software* oferecido pelo ROS e escrever códigos (nós do ROS) que são assinantes do tópico "/scan", e utilizar esses dados para desenvolver aplicações mais específicas para o seu robô. É importante lembrar que dados trocados via tópicos precisam estar em formatos definidos de mensagens. O ROS utiliza o tipo de mensagen "sensor_msgs/LaserScan.msg" para dados de sensores laser. A definição completa da estrutura dessa mensagem se encontra no Anexo B.

Sem a utilização do ROS, o desenvolvedor teria que despender um tempo considerável para escrever o código de recepção de dados de acordo com o protocolo de comunicação do sensor. Em adição, o código possivelmente se tornaria inutilizável em outros projetos caso fosse necessária uma mudança de plataforma ou de linguagem de programação.

Como um simples exemplo de utilização do pacote, foi escrito um nó chamado "/ponto", que é assinante do tópico "/scan". Ao receber a mensagem do tópico, que contém os 683 pontos medidos do laser, o programa imprime na tela de comando o valor correspondente à posição 368 do vetor de distâncias. Esse ponto é a distância frontal do laser ao objeto refletor, ou seja, o ponto correspondente ao ângulo de zero grau, de acordo com a Figura 4.

Esse ponto poderia ser usado, por exemplo, para evitar que o robô colidisse com obstáculos a sua frente. A Figura 11 mostra como é feita a troca das mensagens no ROS, enquanto que a Figura 10 mostra a informação de distância sendo impressa no terminal do Raspberry PI para esse exemplo. O código do nó "/ponto" (escrito em Python), pode ser visto Apêndice A.

Figura 10 – Impressão da distância na tela de comando usando o nó "/ponto".



Fonte: Acervo do autor.

Figura 11 – Troca de mensagens do exemplo sobre o nó "/urg_node".

Fonte: Elaborado pelo autor.

2.4.4 Definição do Pacote Hector_Mapping

Técnicas de alinhamento de escaneamentos via tecnologia LIDAR têm sido o objeto de estudo de muitos pesquisadores ao longo dos últimos anos. O processo de se obter a posição de um robô por meio do alinhamento das medições de sensores LIDAR pode ser feito

em duas técnicas principais (ZHANG e SINGH, 2014. GONZALEZ, STENTZ, e OLLERO, 1995. DISSANAYAKE, DURRANT-WHYTE, e BAILEY, 2000. ELIAZAR e PARR, 2003).

A primeira técnica consiste em comparar cada novo escaneamento obtido pelo sensor com os escaneamentos precedentes para determinar o quanto o robô se moveu e rotou ao longo do tempo. Esse cálculo é limitado pela taxa de atualização do sensor LIDAR e pela quantidade de pontos obtidos em um escaneamento completo. Desta forma, quanto maior o número de pontos medidos e maior a taxa de atualização do sensor, mais preciso se torna o cálculo que determina a mudança de posição e orientação de um robô no espaço. Essa estimativa se baseia na mudança das medidas do sensor conforme o robô se move locomove. O problema dessa técnica é que ela requer poder de processamento muito grande, pois todos os vetores de pontos, ou pelo menos uma grande parte deles, devem ser armazenados e utilizados em um algoritmo de probabilidade para determinar a nova postura do robô ao longo do tempo.

Uma segunda abordagem, que exige menor capacidade do sensor e da unidade de processamento dos dados, é a utilização de um algoritmo de localização e mapeamento simultâneos (SLAM). Na robótica, o SLAM é o problema computacional de construir e atualizar um mapa de um ambiente desconhecido previamente, ao mesmo tempo em que a posição do agente de mapeamento é computada. Essa técnica é construída em dois passos. Um algoritmo mais lento e em uma camada de *software* inferior é responsável por construir um mapa do ambiente baseado nas medições do sensor. Enquanto isso, um algoritmo mais rápido e em uma camada de *software* superior é responsável por alinhar as novas medições com o mapa construído até o momento, assim obtendo a nova postura do robô no ambiente. Essa técnica dispensa a necessidade de armazenar as medições passadas, já que elas são indiretamente armazenadas no mapa já construído do ambiente, o qual é utilizado pelo algoritmo para estimar novas posturas.

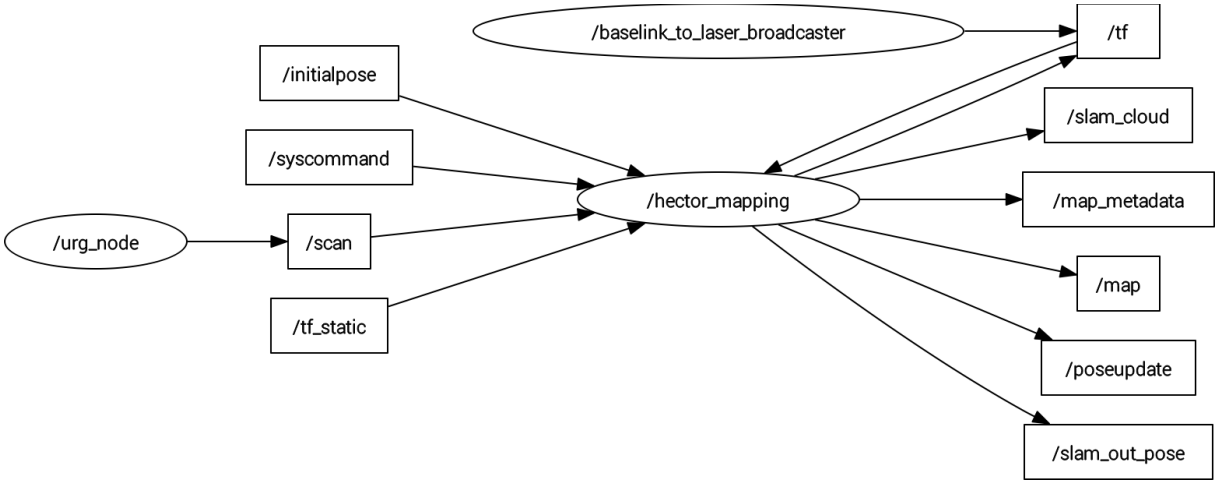
O ROS, mais uma vez aproveitando das suas características de modularização e reutilização de *software*, oferece um pacote para SLAM baseado na segunda estratégia. Esse pacote foi desenvolvido por Kohlbrecher, Von Stryk, Meyer, e Klingauf (2011), e é utilizado nesse trabalho para obter a postura do robô móvel por meio das leituras do sensor LIDAR.

O sistema oferecido por meio do pacote Hector_Mapping se aproveita da alta taxa de atualização de sensores LIDAR modernos, e utiliza seus dados para obter a postura de robôs móveis no ambiente. Esse sistema foi utilizado com sucesso em robôs terrestres não tripulados, veículos de superfície não tripulados, dispositivos portáteis de mapeamento e dados registrados

de UAVs (*unmanned aerial vehicle*) de quadrorotores (HECTOR_MAPPING WIKI, 2014).

Qualquer fonte de dados proveniente de sensores LIDAR pode ser usada junto ao pacote Hector_Mapping. O nó "/hector_mapping" é assinante do tópico "/scan", enquanto que o nó "/urg_node" publica para o tópico "/scan". Assim, a transmissão da informação do sensor LIDAR para o pacote de localização e mapeamento é simplificada, graças a estrutura de modularização de *software* oferecida pelo ROS. A Figura 12 mostra a estrutura de nós e tópicos utilizada na integração do sensor LIDAR com o sistema de localização e mapeamento.

Figura 12 – Estrutura de tópicos e nós com utilização do pacote hector_mapping.



Fonte: Elaborado pelo autor.

Além de ser assinante do tópico "/scan", o nó "/hector_mapping" é assinante dos tópicos de transformadas "/tf_static" e "/tf", com os quais o usuário define a relação de transformação entre a postura do centro do sensor LIDAR e a postura do centro do robô. Em todo este trabalho, foi considerado que o centro do sensor LIDAR coincide com os centros de massa e de gravidade do robô, sendo desnecessário, portanto, utilizar transformada entre eles.

O tópico "/initialpose" é usado para definir a posição inicial do robô (coincidente com a do sensor LIDAR). O tópico "/syscommand" é utilizado para restaurar a postura estimada do robô para seu valor inicial a qualquer momento. Para isso, basta que um nó publique a *string* "reset" para o tópico. Assim, é possível que durante os experimentos, a posição do robô seja reiniciada sem a necessidade de executar novamente o nó "/hector_mapping".

Os tópicos publicados pelo nó "/hector_mapping" são chamados de "/map_metadata", "/map", "/slam_cloud", "/pose_update" e "/slam_out_pose". O terceiro tópico serve para construção de nuvens de ponto, que não estão no contexto deste trabalho. Os dois primeiros tópicos contêm as informações do mapa construído do ambiente.

O mapa construído é uma grade retangular composto de células quadradas que representam a ocupação do ambiente. O usuário pode definir três parâmetros para o mapa construído: o número de células por linha e o número de células por coluna, que definem o número total de células ($n_{total} = n_{lin} * n_{col}$), e o tamanho do espaço que cada lado da célula quadrada representa em metros. O tópico "/map_metadata" traz essas informações características do mapa de ocupação do ambiente, enquanto que o tópico "/map" traz o valor de cada célula da grade de ocupação do mapa. O valor 100 representa certeza de ocupação da célula, o valor 0 significa certeza de não ocupação, enquanto que o valor -1 representa incerteza. As definições completas das mensagens publicadas nos tópicos "/map_metadata" e "/map" são mostradas nos Anexos C e D, respectivamente.

Os tópicos publicados "/pose_update" e "/slam_out_pose" trazem a postura do robô. A única diferença entre eles é que o primeiro oferece, além da postura, uma matrix de convariância entre os elementos do vetor de postura. A postura do robô é atualizada com a mesma taxa de atualização do sensor LIDAR. No caso do sensor utilizado neste trabalho, essa taxa é de 10 Hz, como explicado nas seções passadas. Já a atualização do mapa é mais devagar, e não acontece a uma taxa constante. Isso se deve ao fato da atualização do mapa ocorrer apenas quando um dos dois parâmetros estabelecidos de limiar ocorrem. Esses parâmetros, também definidos pelo o usuário, são o limiar de distância, e o limiar de ângulo. Assim, o mapa só é atualizado quando a plataforma LIDAR (e consequentemente o robô) houverem se movido ou rotado por um valor mínimo. As definições das mensagens publicadas nos tópicos "/pose_update" e "/slam_out_pose" são mostradas nos Anexos E e F, respectivamente.

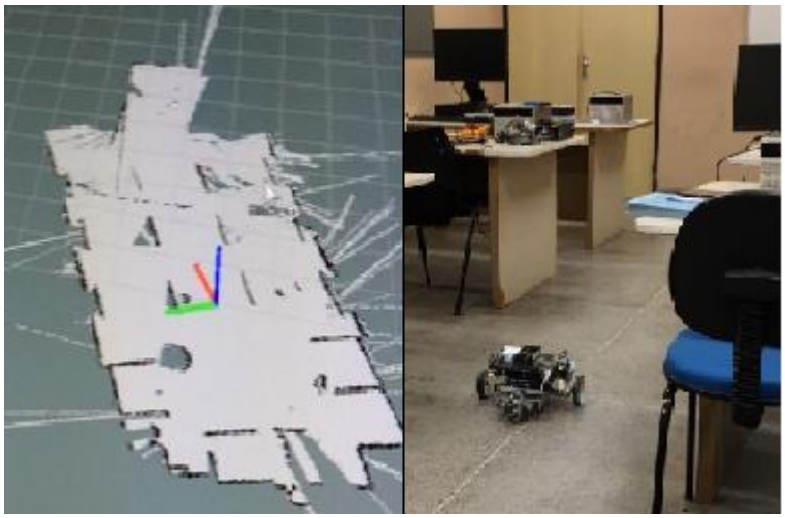
2.4.4.1 Exemplo de uso do pacote Hector_Mapping

Para exemplificar o uso da plataforma proposta no início deste capítulo, a mesma foi testada em um laboratório, tendo sido montada sobre um robô do kit de desenvolvimento de robótica da *National Instruments* (NI). Esse não é o robô final onde a plataforma será integrada, pois o mesmo ainda será apresentado no capítulo 3. Para este experimento, a plataforma estava completamente independente do robô, que foi utilizado apenas como meio de locomoção no ambiente. O mesmo experimento poderia ter sido realizado com uma pessoa andando pelo o laboratório com a plataforma em suas mãos. No entanto, a utilização do robô simula melhor o deslocamento pelo ambiente, em uma altura que permite a detecção de cadeiras e mesas pelo sensor LIDAR para construção do mapa. Enquanto o robô se movia no espaço, sem ajuda de

nenhum controlador de seguimento de trajetória, o nó “/hector_mapping” foi utilizado para criar o mapa do ambiente, ao mesmo tempo em que a postura do robô era rastreada.

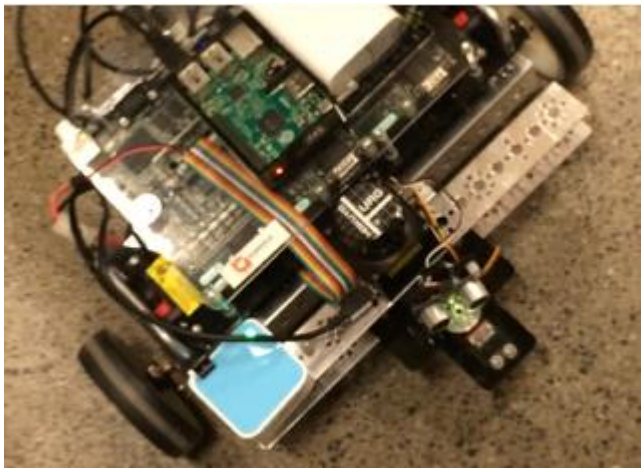
A Figura 13 mostra o robô no ambiente do laboratório, à direita, e o mapa criado e a posição do robô à esquerda. A visualização dos dados foi feita utilizando a ferramenta RVIZ do ROS. A Figura 14 mostra melhor a plataforma (sensor LIDAR + Raspberry PI) montada sobre o robô. No mapa, as regiões em cinza claro mostram as células com valor 0, ou seja, com certeza de não ocupação. Enquanto que as regiões em preto demonstram as células com valor 100, ou seja, com certeza de ocupação. As outras regiões são as de incerteza.

Figura 13 – Exemplo de mapeamento e localização usando o pacote hector_mapping.



Fonte: Elaborado pelo autor.

Figura 14 – Plataforma de mapeamento e localização montada sobre o robô da NI.



Fonte: Acervo do autor.

3 DESCRIÇÃO DO ROBÔ MÓVEL E INTEGRAÇÃO DO SISTEMA DE MEDIÇÃO

Na Figura 15 é mostrada a vista frontal do robô móvel utilizado para os experimentos, indicando a localização do sensor LIDAR, que fica centralizado na parte superior do robô. Uma visão inferior do robô é mostrada na Figura 16. O robô pode ser definido como um robô móvel diferencial, ou seja, um robô que tem movimento baseado em duas rodas com tração, colocadas em cada lado de massa do robô.

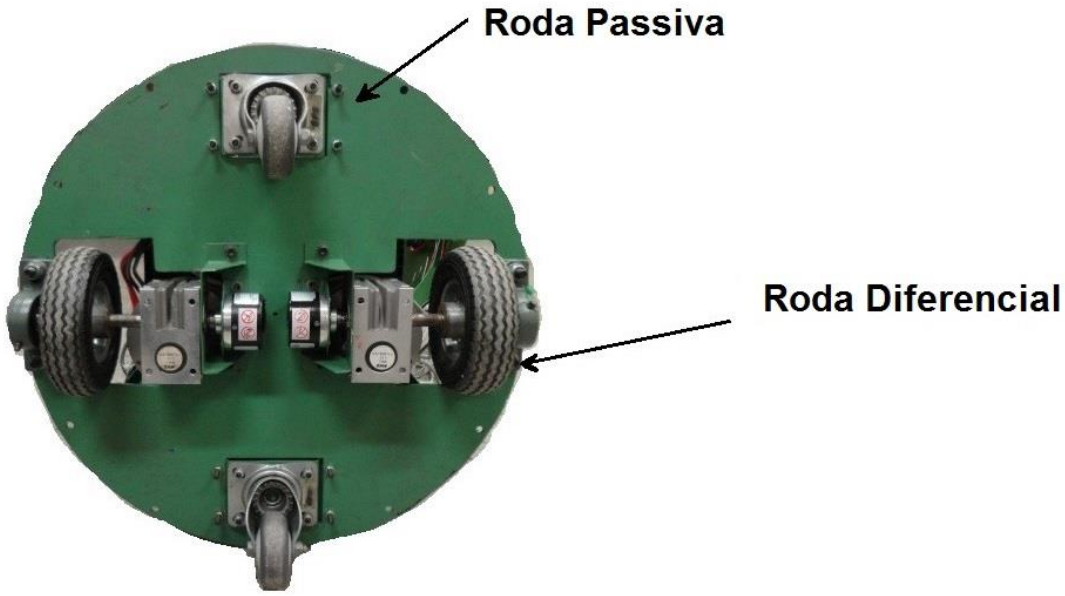
Figura 15 – Visão frontal do robô móvel.

Fonte: Acervo do autor.

O robô móvel é pronto para experimentos de controle e já foi utilizado por R. L. S. Sousa, M. D. do Nascimento Forte, F. G. Nogueira e B. C. Torrico (2016), quando o controlador não-linear apresentado neste trabalho foi utilizado para testes de controle de seguimento de trajetória. No entanto, o trabalho utilizou realimentação via *encoders* para estimar a postura do robô. Nesse trabalho, a realimentação da malha de controle via *encoders* será substituída, e então comparada com a nova técnica que utiliza o sensor LIDAR e o algoritmo SLAM para realimentar a malha. Em um outro trabalho com o mesmo robô, Diniz Lobo, T. (2016), obteve resultados satisfatórios no controle de seguimento de trajetória do robô, também utilizando

encoders, mas dessa vez a estratégia de controle foi alterada para um controlador adaptativo auto-ajustável.

Figura 16 – Visão Inferior do Robô Móvel.



Fonte: Acervo do autor.

3.1 Configuração inicial do robô

Na configuração inicial, dois motores de corrente contínua (motor cc) acionam as duas rodas diferenciais do robô, enquanto que outras duas rodas fixas são responsáveis por fornecer equilíbrio. A distância L entre as duas rodas diferenciais é de 0,4 metros, enquanto que o raio de cada roda é 0,08 metros. O sistema elétrico do robô é suprido por duas baterias em série recarregáveis 24 V 50 AH.

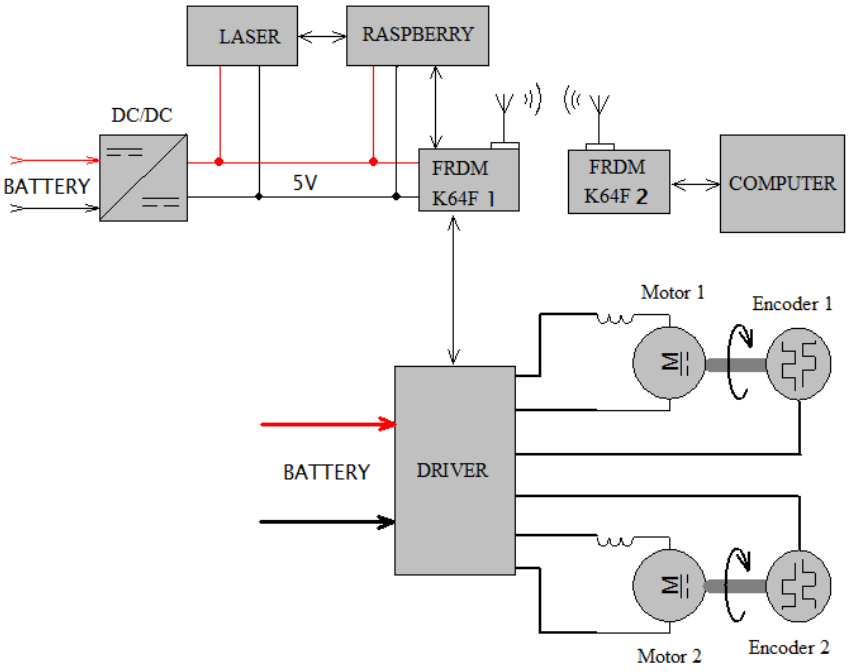
Um microcontrolador baseado em um processador ARM Cortex M4 (FRDM-K64F) da NXP embarca o algoritmo de controle de seguimento de trajetória, que utiliza integração da leitura dos *encoders* para estimar a postura do robô (o controlador será explicado com detalhes no capítulo 4). O controle de velocidade dos motores (CC) é realizado por um driver eletrônico (HDC2450, Roboteq), o qual se comunica com o microcontrolador por meio de protocolo serial RS-232. Um controlador PID interno ao driver dos motores é responsável por fechar a malha de controle de velocidade, utilizando realimentação provida pelos *encoders*.

O robô transmite e recebe dados de entrada e saída via um módulo de rádio-frequência que se comunica com o microcontrolador. Outro microcontrolador, conectado ao computador supervisor, recebe os dados transmitidos via um segundo módulo de rádio-frequência, salva em um arquivo, e plota em tempo real.

3.2 Integração do sistema de medição com sensor LIDAR

Foi adicionado ao robô o sistema de medição apresentado no capítulo 2, que em termos de *hardware* é composto do microcomputador Raspberry PI modelo 2B e do sensor Laser Hokuyo. O sensor LIDAR foi adicionado em uma posição central em cima do robô. O microcomputador Raspberry Pi lê os dados do sensor via porta USB utilizando o nó do ROS "/urg_node", enquanto que o nó de localização e mapeamento simultâneos "/hector_mapping" fornece a postura do robô. A cada amostra obtida, os dados do sensor LIDAR são processados no Raspberry PI, que então envia a postura atualizada para o microcontrolador que embarca o algoritmo de controle, usando um canal de comunicação serial de alta velocidade. O diagrama de blocos completo do robô após as modificações pode ser visto na Figura 17.

Figura 17 – Diagrama de blocos simplificado do robô móvel.



Fonte: Acervo do autor.

3.2.1 Conversão da orientação para ângulo de Euler

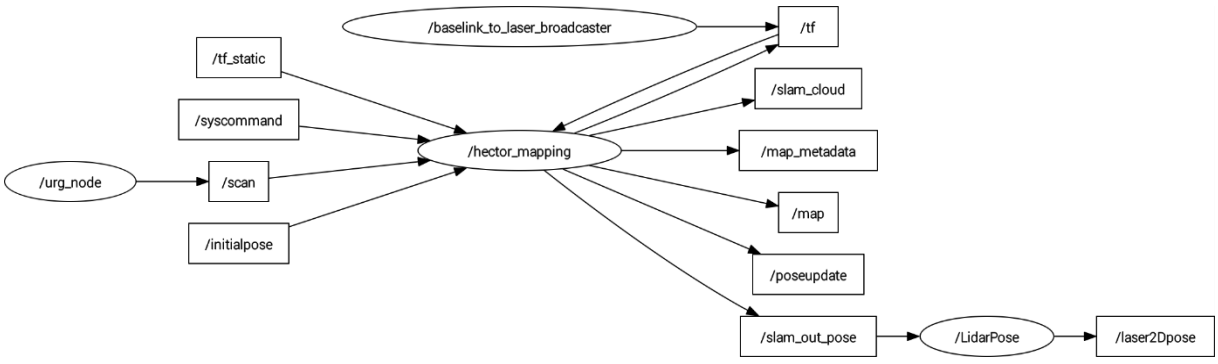
A mensagem publicada pelo nó "/hector_mapping" para o tópico "/slam_out_pose" contém a postura atualizada do robô móvel, atualizada a uma taxa de dez vezes por segundo. A mensagem é composta da posição, e orientação do robô. A posição do robô é dada pelas coordenadas $[x, y, z]$, em que $z = 0$, já que a localização é feita no plano 2D. A orientação do robô é dada no formato de quartênio, que é composto de quatro coordenadas que representam a rotação de um objeto no espaço tridimensional.

O controlador de seguimento de trajetórias, que será apresentado capítulo 4, recebe *feedback* de postura no formato de coordenadas de Euler $[x, y, \theta]$. Sendo assim, o primeiro passo para integração da nova plataforma de medição foi criar um novo nó do ROS para realizar a conversão de postura para o formato desejado. O código do nó, escrito em Python, pode ser visto no Apêndice B.

O nó criado para atender a necessidade de conversão de coordenadas foi chamado de "/LidarPose". Esse nó é assinante do tópico "/slam_out_pose", publicado pelo nó "/hector_mapping". Ao receber a mensagem de postura do tópico, o nó "/LidarPose" publica as coordenadas, já no formato requerido $[x, y, \theta]$, para um novo tópico, chamado de "laser2Dpose".

Com a adição do novo nó, a estrutura de nós e tópicos apresentada no exemplo da seção 2.4.2.1 é alterada. O novo nó é mostrado no canto inferior direito da Figura 18.

Figura 18 – Estrutura de nós e tópicos após adição do nó de conversão de coordenadas.



Fonte: Elaborado pelo autor.

3.2.2 Protocolo de envio de postura

Após a conversão para o sistema de coordenadas compatível com o controlador de seguimento de trajetórias, foi necessário criar mais um nó, responsável por enviar a postura do robô medida pelo algoritmo de localização e mapeamento simultâneos para o microcontrolador FRDM-K64F, que embarca o código de controle.

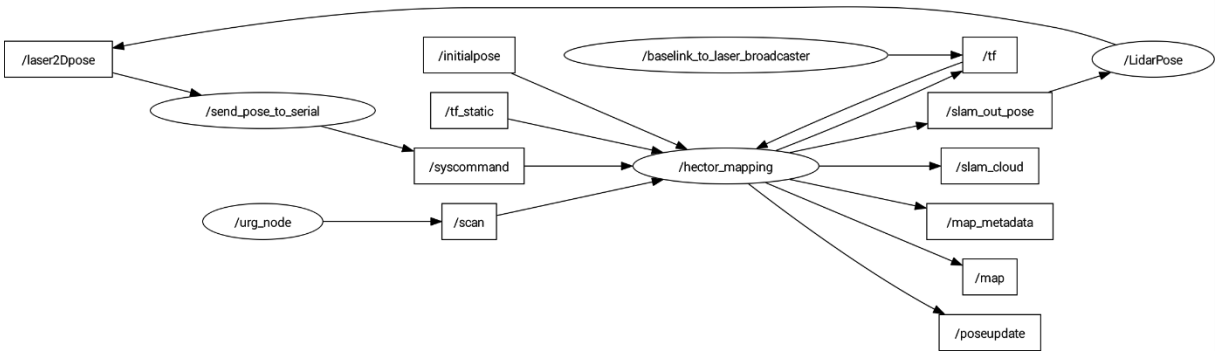
Para escrever o código do nó, foi necessário estabelecer um protocolo de comunicação que permitisse o envio dos dados de maneira que fosse facilmente descompactado pelo microcontrolador, de forma a não gerar atrasos na atuação do controlador. O novo nó abre a porta serial do microcomputador Raspberry PI em alta velocidade, com um *baud rate* de 115200 e espera pelo sinal de comando do microcontrolador, indicando que está pronto para receber um novo dado ou que deseja reiniciar a postura do robô.

A estrutura do dado de postura enviado foi definido como uma *string* de 15 caracteres no formato $"\pm x.xx \pm y.yy \pm \theta.\theta\theta"$. Esse formato indica que as medidas de posição e orientação são enviadas com uma precisão de duas casas decimais, e com o sinal indicador de positivo ou negativo. Foi decidido para o protocolo de comunicação que, caso a medição de qualquer um dos três valores fosse igual a zero, seria enviado com o símbolo de positivo, assim o número de caracteres da *string* é constante.

O novo nó criado para executar o protocolo de comunicação foi chamado de "/send_pose_to_serial". Este nó é assinante do tópico "/laser2Dpose", que contém a informação de postura no formato desejado $[x, y, \theta]$. Em adição, o nó é publicador do tópico "/syscommand", o que permite que a postura do robô seja reiniciada a qualquer momento, dependendo do comando enviado a partir do microcontrolador.

Foi definido no protocolo de comunicação que caso o caractere "R" fosse recebido via porta serial, o nó "/send_pose_to_serial" enviaria a palavra "reset" para o tópico "/syscommand", reiniciando a postura do robô no algorítmo SLAM do nó "/hector_mapping", como explicado no capítulo 2. Caso receba o caractere "$", a postura atual do robô é enviada via porta serial para o microcontrolador. A Figura 19 mostra a nova estrutura de nós e tópicos após a adição do nó de comunicação serial com o microcontrolador. O código do nó, que foi escrito na linguagem de programação Python, é apresentado no Apêndice C.

Figura 19 – Estrutura de nós e tópicos após adição do nó de comunicação serial.



Fonte: Elaborado pelo autor.

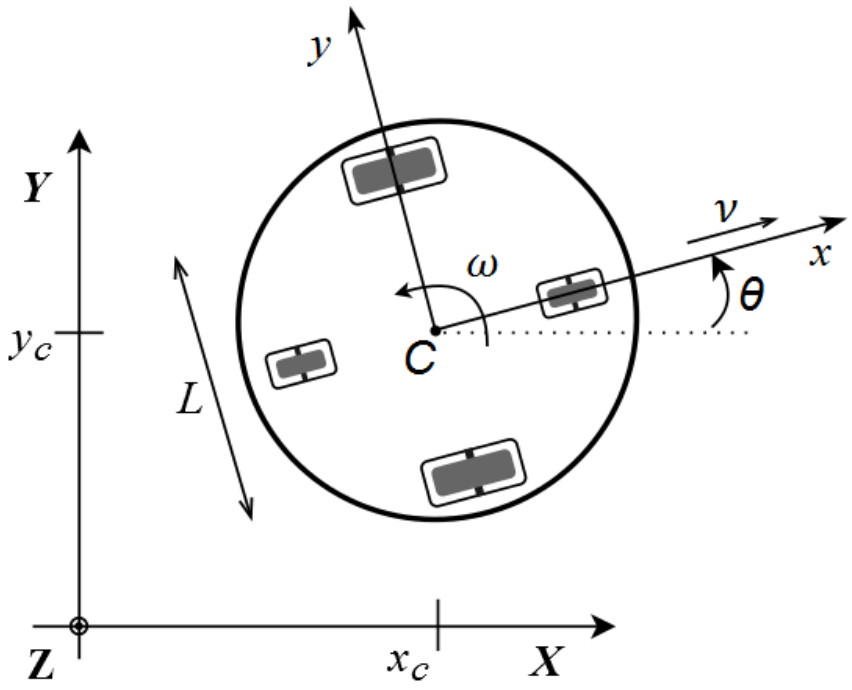
4 MODELAGEM MATEMÁTICA E PROJETO DO CONTROLADOR DE SEGUIMENTO DE TRAJETÓRIAS

Após o desenvolvimento e integração da nova plataforma de medição, a modelagem matemática do robô móvel não-holonômico e a arquitetura do controlador de seguimento de trajetórias são apresentados neste capítulo, como último passo necessário para apresentação e análise dos testes experimentais.

4.1 Modelagem Matemática do Robô Móvel

Os símbolos para descrição matemática do robô são mostrados na Figura 20.

Figura 20 – Arquitetura do robô e simbologia.



Fonte: Acervo do autor.

É suposto que o centro de gravidade e centro geométrico do robô são coincidentes, e L é a distância entre as duas rodas diferenciais. É importante notar que C é a origem (x, y) e a posição central entre as duas rodas, e θ é o ângulo de orientação do robô em relação ao eixo (X, Y). Usando as coordenadas de posição e o ângulo de orientação, é possível encontrar as

coordenadas absolutas (X, Y) na referência absoluta. A velocidade linear das rodas esquerda (v_L) e direita (v_R) podem ser descritas pelas equações (3) e (4).

$$v_L = v - \frac{\omega L}{2} \tag{3}$$

$$v_R = v + \frac{\omega L}{2} \tag{4}$$

Das equações (3) e (4), as velocidades linear e angular do robô (υ e ω respectivamente) podem ser obtidas por:

$$v = \frac{v_L + v_R}{2} \tag{5}$$

$$\omega = \frac{v_R - v_L}{L} \tag{6}$$

Robôs com essa arquitetura possuem uma limitação resultante da incapacidade de locomoção lateral da roda. Essa limitação é expressa pela equação (7).

$$[-\sin\theta\,(t)\ \cos\theta\,(t)]\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} = 0 \tag{7}$$

Consequentemente, o robô não pode se mover na direção lateral da roda, e então o modelo cinemático é dado por (8).

$$\begin{bmatrix}\dot{x}_c\\ \dot{y}_c\\ \dot{\theta}_c\end{bmatrix} = \begin{bmatrix}\cos\theta & 0\\ sen\ \theta & 0\\ 0 & 1\end{bmatrix}.\begin{bmatrix}v\\ \omega\end{bmatrix} \tag{8}$$

A partir de uma dada trajetória de referência $p_r(t) = (x_r(t), y_r(t), \theta_r(t))^t$, definida em um intervalo de tempo $T \in [0,\ T]$, e da cinemática inversa do robô, é possível derivar uma lei de controle não realimentado. No entanto, essas entradas calculadas só podem dirigir o robô por um caminho desejado se não houver presença de distúrbios (como deslizamentos e derrapagens) e erros no estado inicial.

A velocidade tangencial (v_r) e angular (ω_r) são as entradas necessárias para o robô, e são calculadas a partir da trajetória de referência. Assim, a velocidade tangencial de referência é obtida através da equação (9).

$$v_r(t) = \pm\sqrt{\dot{x}_r{}^2 + \dot{y}_r{}^2} \tag{9}$$

Onde o sinal $\pm$ depende da direção desejada de tração das rodas, (+ em frente, - para trás), e a velocidade angular do robô é calculada a partir da equação (10). E o ângulo de orientação em cada ponto de referência é dado pela equação (11).

$$\omega_r(t) = \frac{\dot{x}_r(t)\,\ddot{y}_r(t) - \dot{y}_r(t)\ddot{x}_r(t)}{\dot{x}_r^2(t) + \dot{y}_r^2(t)} \tag{10}$$

$$\theta_r(t) = \tan^{-1}(\dot{y}_r(t), \dot{x}_r(t)) + k\pi \tag{11}$$

Onde k define a direção desejada. Para k=0 representa um giro com velocidade angular positiva, e k=1 representa um giro na direção contrária.

4.2 Projeto do Controlador Não Linear

4.2.1 Visão Geral

A Figura 21 traz o resumo do esquema de controle de seguimento de trajetória utilizado neste trabalho. Existem dois controladores trabalhando em cascata. O controlador mais externo é a malha de controle de postura, nas qual a posição e orientação de referência são comparados com a postura obtida pelo sistema de localização e mapeamento simultâneos 2D apresentado no capítulo 2. O controlador cinemático então gera as referências de velocidade linear e angular para os motores da direita e esquerda, de forma a levar o robô para a postura de referência.

O segundo controlador é o controlador de velocidade dos motores. A velocidade dos motores (CC) é controlada através de um controlador PID já incluído no driver eletrônico

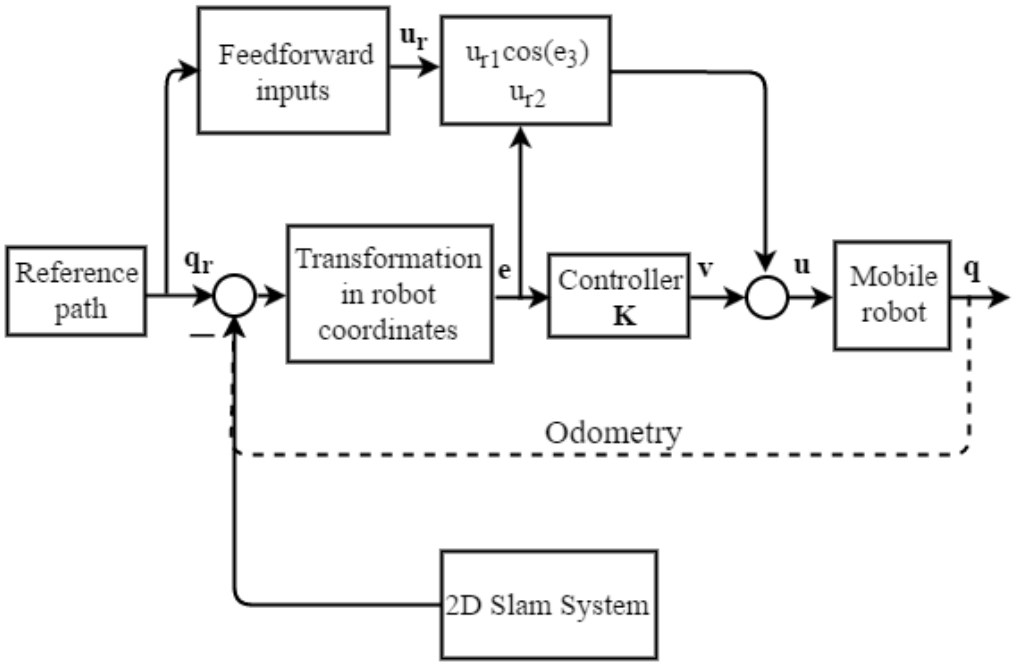
do motor. Durante todo o trabalho, o PID ficou ajustado com os valores mostrados na Tabela 1. A realimentação para o controlador de velocidade é obtida através dos *encoders* acoplados aos motores. O ajuste dos parâmetros do controlador de seguimento é discutido no capítulo de resultados experimentais.

Tabela 1 – Ajuste dos parâmetros do controlador de velocidade dos motores.

Variável	Valor
Kp	5
Ki	10
Kd	0

Fonte: Elaborado pelo autor.

Figura 21 – Esquema do controlador de seguimento de trajetória.



Fonte: Acervo do autor.

É importante notar que a linha tracejada mostra o método de odometria que era usado inicialmente para estimar a postura do robô, o qual está sendo substituído neste trabalho. Como mencionado no capítulo 2, foi introduzido um novo bloco, que converte as distâncias medidas pelo sensor LIDAR no vetor de postura do robô [x, y, θ], para ser usado como realimentação.

4.2.2 Desenvolvimento matemático

De acordo com Klančar, Matko e Blažič (2005), a trajetória de referência é definida pelo vetor $P_{ref}(\mathrm{t}) = (x_{ref}(t), y_{ref}(t), \theta_{ref}(\mathrm{t}))$. Omitindo a dependência das variáveis no tempo, o vetor erro de postura na referência do robô móvel é dado pela equação (12):

$$\begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} cos\theta & sen\theta & 0 \\ -sen\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} X_r - x \\ Y_r - y \\ \theta_r - \theta \end{bmatrix} \tag{12}$$

Das equações (8) e (12), o modelo do erro é definido com seguinte sistema de equações:

$$\begin{bmatrix} \dot{e}_1 \\ \dot{e}_2 \\ \dot{e}_3 \end{bmatrix} = \begin{bmatrix} cos\, e_3 & 0 \\ sen\, e_3 & 0 \\ 0 & 1 \end{bmatrix} . \begin{bmatrix} v_r \\ \omega_r \end{bmatrix} + \begin{bmatrix} -1 & e_2 \\ 0 & -e_1 \\ 0 & -1 \end{bmatrix} . \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{13}$$

Então, as entradas do robô (saídas do controlador de seguimento de trajetória) são definidas como:

$$\begin{aligned} u_1 &= v_r \cos e_3 - v_1 \\ u_2 &= \omega_r - v_2 \end{aligned} \tag{14}$$

Onde v_r e ω_r são as entradas de alimentação diretas de velocidades angular e tangencial, enquanto que v_1 and v_2 são as saídas da malha de controle realimentada. A equação de modelo do erro (13) é linearizada para obter a seguinte lei de controle:

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} -k_1 & 0 & 0 \\ 0 & -sign(v_r)k_2 & -k3 \end{bmatrix} . \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} \tag{15}$$

Para ajustar os ganhos do controlador, o usuário especifica os valores do coeficiente de amortecimento ζ e da frequência natural do sistema ω_n. A resposta em malha fechada aos sinais de erros está ligada à especificação desses parâmetros. Os ganhos do controlador são, então, calculados em tempo real através das seguintes equações:

$$k1 = k3 = 2\zeta\omega_n(t)$$
$$k2 = g.|v_r(t)| \quad (16)$$
$$\omega_n(t) = \sqrt{\omega_r(t)^2 + g.v_r(t)^2}$$

Prova e análise de estabilidade são apresentadas por Klančar, Matko e Blažič (2005).

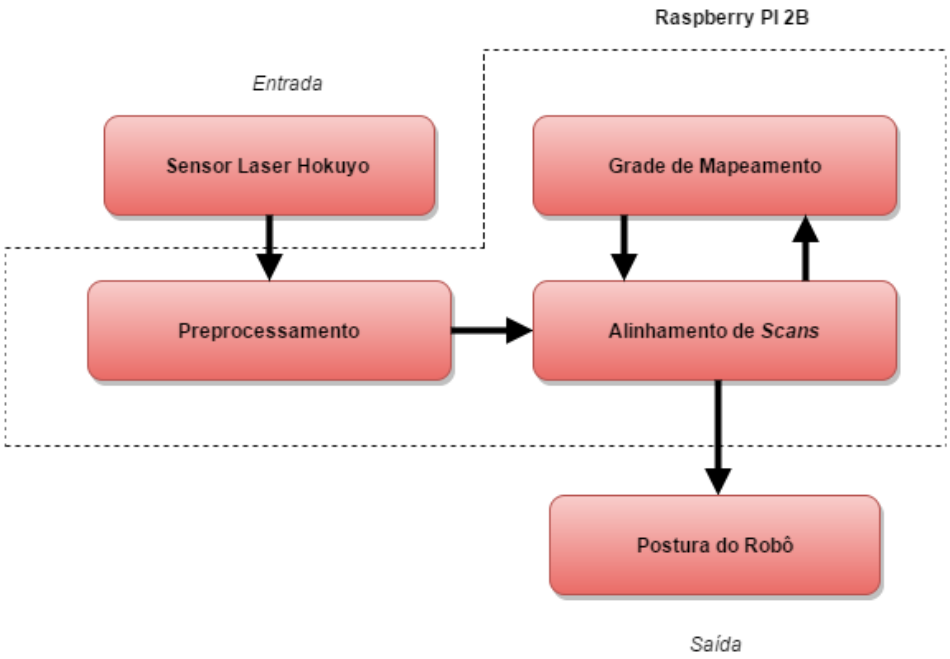
4.2.3 Resumo do sistema SLAM 2D usado para realimentar a malha de controle

Uma versão sistêmica e resumida do conjunto de localização e mapeamento utilizada é apresentado na Figura 22. Suponhamos que o sistema SLAM acaba de ser iniciado, e que não há nenhuma informação prévia do ambiente. As coordenadas do mundo serão ajustadas como na Figura 20, onde o centro do sensor LIDAR se torna a origem do plano cartesiano, e temos que (x,y,θ) = (0,0,0). O eixo X aposta na direção frontal do laser, enquanto que o eixo Y aponta na direção de quadratura, ou a 90 graus do eixo X.

O sensor LIDAR (Hokuyo Laser Urg-04lx-ug01) obtém novos pontos de distância a cada 100 ms, que são lidos por meio do nó do ROS "/urg_node". Uma rotina de pré-processamento mapeia os pontos nas coordenadas da grade de mapeamento, por meio do nó "/hector_mapping". Como explicado previamente, as rotinas de alinhamento dos pontos e mapeamento são interconectadas no SLAM.

A medida que o robô se move, a grade de mapeamento do ambiente é construída em paralelo com o algoritmo de correspondência que alinha os novos pontos de distância com o mapa aprendido até o momento, de forma que, implicitamente é realizado a correspondência com todos os escaneamentos passados. Assim, o nó "/hector_mapping" publica uma nova postura do robô, que é traduzida para coordenadas no sistema Euler pelo nó "/LidarPose". Finalmente, a postura é publicada via porta serial para o microcontrolador que embarca o controle de seguimento de trajetória do robô, usando o nó "/send_pose_to_serial", e é usada para fechar a realimentação da malha, como mostrado na Figura 21.

Figura 22 – Resumo da estrutura de localização e mapeamentos simultâneos 2D.



Fonte: Elaborado pelo autor.

É importante notar que o mapeamento do ambiente neste trabalho é usado exclusivamente com o objetivo de encontrar a postura do robô no espaço. No entanto, a grade de mapeamento pode ser usada futuramente em algoritmos de planejamento de trajetória e para evitar obstáculos.

5 TESTES EXPERIMENTAIS E RESULTADOS

Os parâmetros especificados para os quatro primeiros testes do controlador de seguimento de trajetória são mostrados na Tabela 2. Os valores foram selecionados visando um movimento suave do robô ao longo do trajeto de referência. A variável g representa um ganho proporcional, que multiplica a referência de velocidade e o erro calculado. O parâmetro ζ representa o fator de amortecimento que multiplica a frequência natural do polinômio que contêm as características desejadas da planta, que é de terceira ordem.

Tabela 2 – Parâmetros escolhidos para trajetória circular.

Variável	Valor
g	40
ζ	0.6

Fonte: Elaborado pelo autor.

5.1 Experimento usando *encoders* para fechar a malha de controle de posição de trajetória circular

O robô mostrado na Figura 15 foi usado para os testes experimentais. Em um primeiro teste, a postura do robô foi obtida utilizando o método convencional de integração dos dados dos *encoders*, através do modelo cinemático do robô.

A trajetória do robô de referência escolhida foi uma circunferência. Essa é uma boa trajetória de referência, pois o robô precisa mudar a sua posição e orientação em cada ponto da trajetória, o que adiciona um nível de complexidade maior. Neste primeiro teste, o robô realizou duas voltas.

Tabela 3 – Erros iniciais do robô para trajetória circular.

Erros Iniciais		
x [m]	y [m]	θ [rad]
0	0.5	-pi

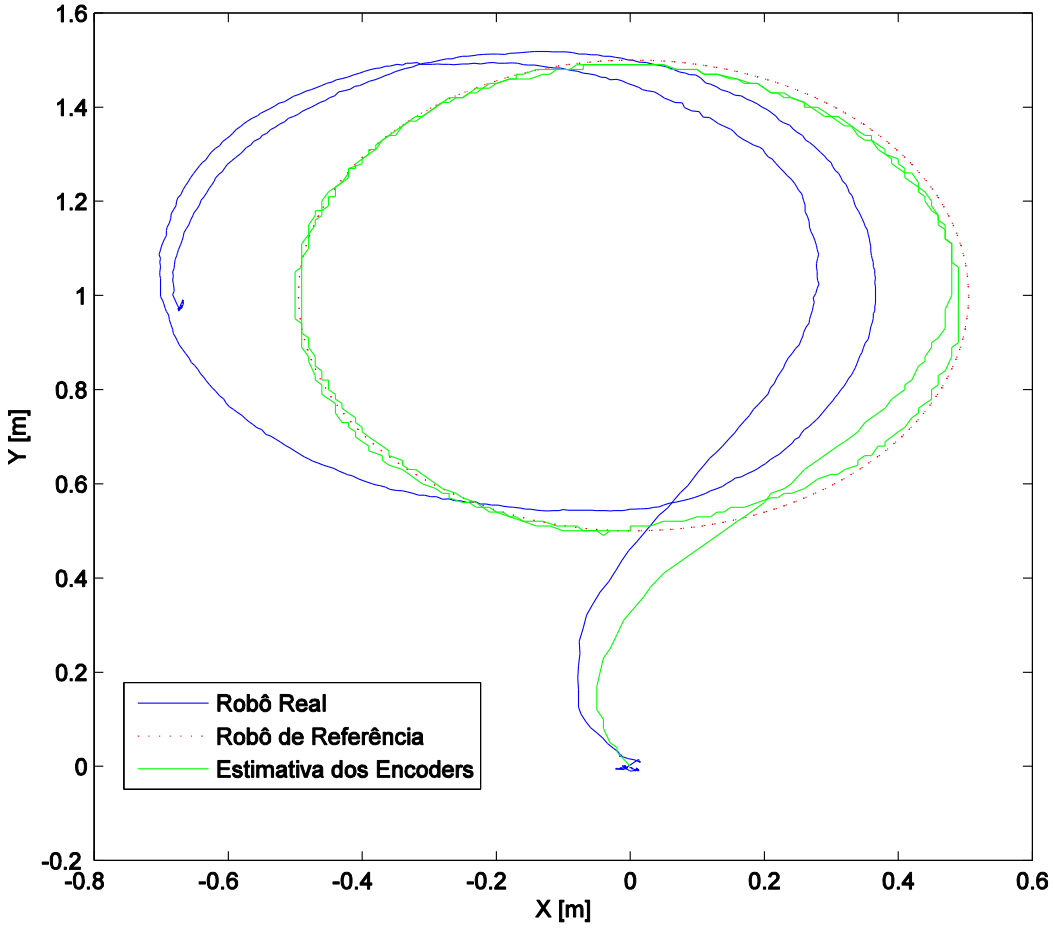
Fonte: Elaborado pelo autor.

No primeiro teste, a trajetória de referência (linha tracejada de vermelho da Figura 23) foi programada para um círculo. A postura inicial do robô [x=0, y=0, θ=pi] foi ajustada longe da referência [x=0, y=0.5, θ=0], forçando o erro inicial de postura da Tabela 3.

A linha azul mostra a posição real do robô durante o experimento de seguimento de trajetória. É notável que a curva azul se difere bastante da trajetória de referência devido ao escorregamento das rodas, um efeito que os *encoders* não conseguem detectar, demonstrando que a técnica de medição de postura por meio de odometria gera erro um em regime permanente.

Também é notável, ao analisar a figura, que a posição estimada pelos *encoders* segue a trajetória de referência, o que implica que o algoritmo fica "cego", pois acredita que está seguindo a trajetória sem erros, não havendo assim reação de correção do controlador. No entanto, a posição real do robô, medida pelo algoritmo de localização SLAM 2D, demonstra que isso não é o que realmente está acontecendo.

Figura 23 – Trajetória circular do robô utilizando posturas estimadas usando *encoders*.



Fonte: Acervo do autor.

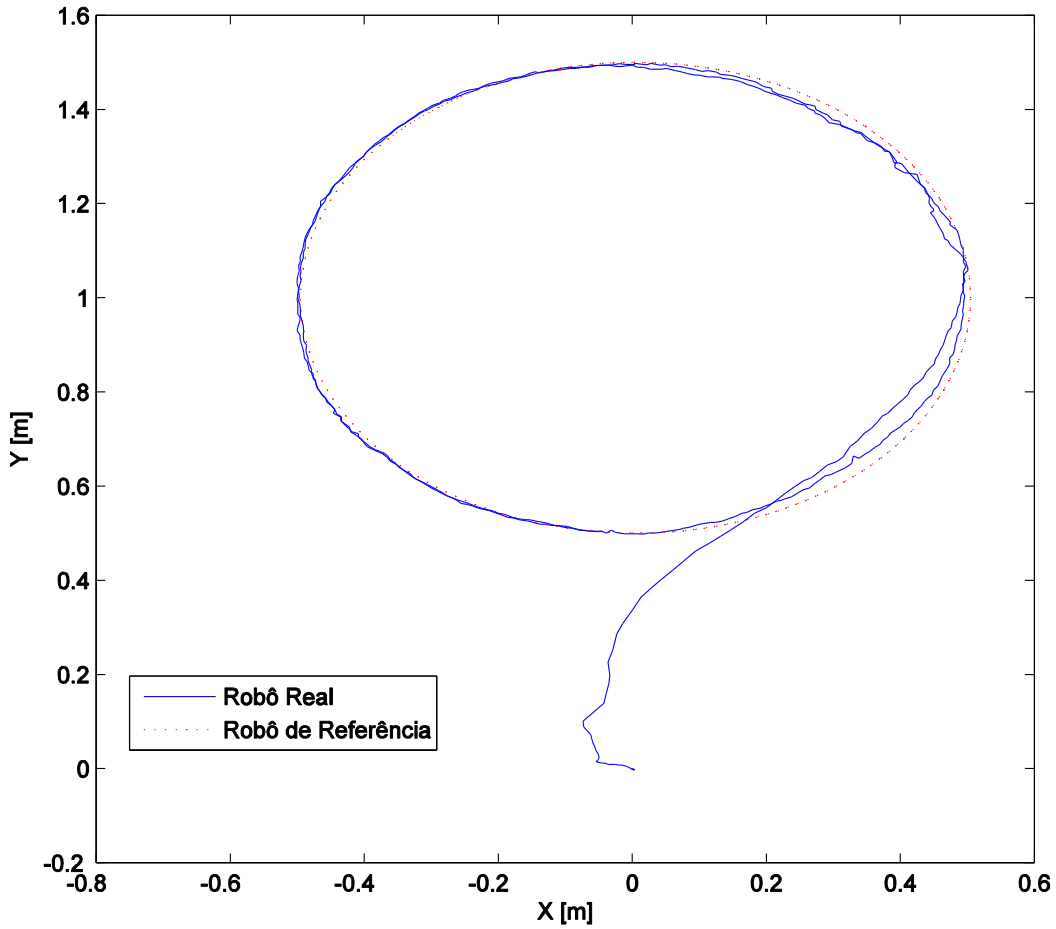
É importante notar que além de divergir da trajetória de referência, o robô para distante do ponto final da trajetória após as duas voltas. O robô para próximo do ponto de coordenadas cartesianas [x=0.6, y=1.0], quando deveria parar no ponto [x=0, y=0.5].

5.2 Experimento usando sistema SLAM 2D para fechar a malha de controle de posição de trajetória circular

Em um segundo teste, o algoritmo de localização e mapeamento simultâneo (SLAM) foi utilizado para obter a postura do robô e para realimentação do controlador cinemático. Os erros iniciais foram os mesmos da Tabela 3. A curva de trajetória resultante pode ser vista marcada pela linha azul na Figura 24. A trajetória real segue a de referência e o controlador se torna capaz de reduzir o erro de postura para zero.

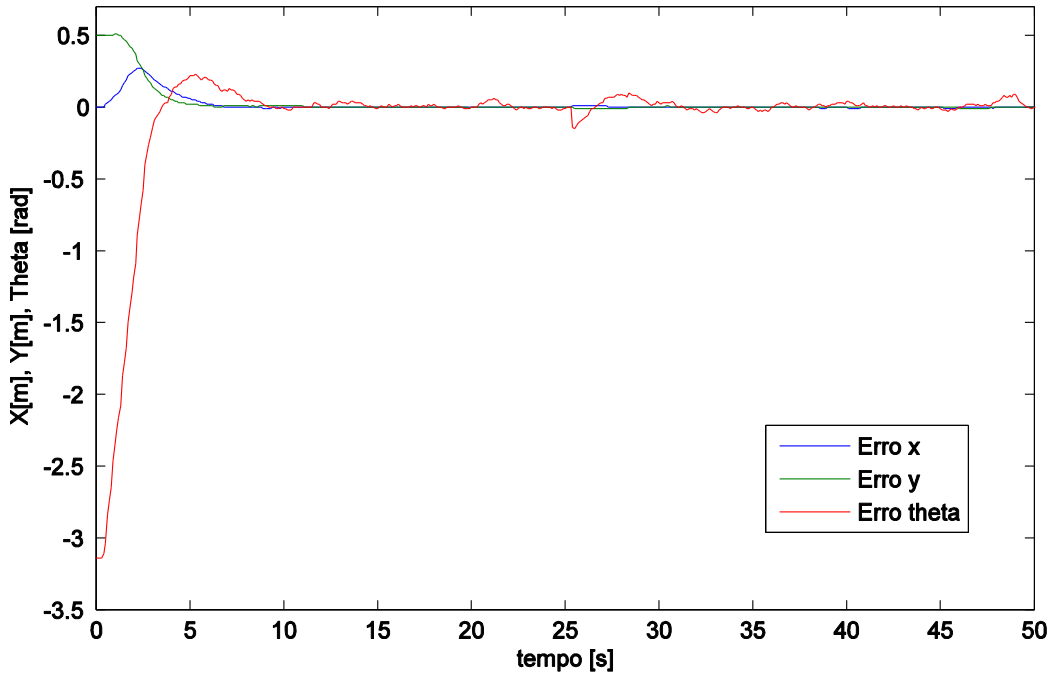
É notável que, além de seguir a posição especificada pelo robô de referência, o robô real completa as duas voltas no tempo determinado pelo controlador, parando no ponto de coordenadas [x=0, y=0.5], como esperado, não havendo erro de estado estacionário. A Figura 25 mostra os erros de x, y e θ para o segundo teste.

Figura 24 – Trajetória circular do robô utilizando posturas estimadas usando sensor laser.



Fonte: Acervo do autor.

Figura 25 – Erros de postura usando sensor laser para fechar malha de controle de seguimento de trajetória circular.



Fonte: Acervo do autor.

É possível notar que, no tempo aproximadamente igual a 25 segundos, um ruído de medição causa um erro na orientação do robô. Logo em seguida, o erro é novamente corrigido.

5.3 Experimento usando *encoders* para fechar a malha de controle de posição de trajetória quadrada

Em um terceiro teste, a trajetória de referência foi mudada para um quadrado de lados iguais a 1,5 metros. A postura de referência inicial foi [x=0, y=0.5, θ=0]. Mais uma vez, os *encoders* foram usados inicialmente para estimar a realimentação de postura. Os erros iniciais foram listados na Tabela 4. Os ganhos do controlador foram ajustados com os mesmos valores dos testes anteriores. Foi realizado apenas uma volta.

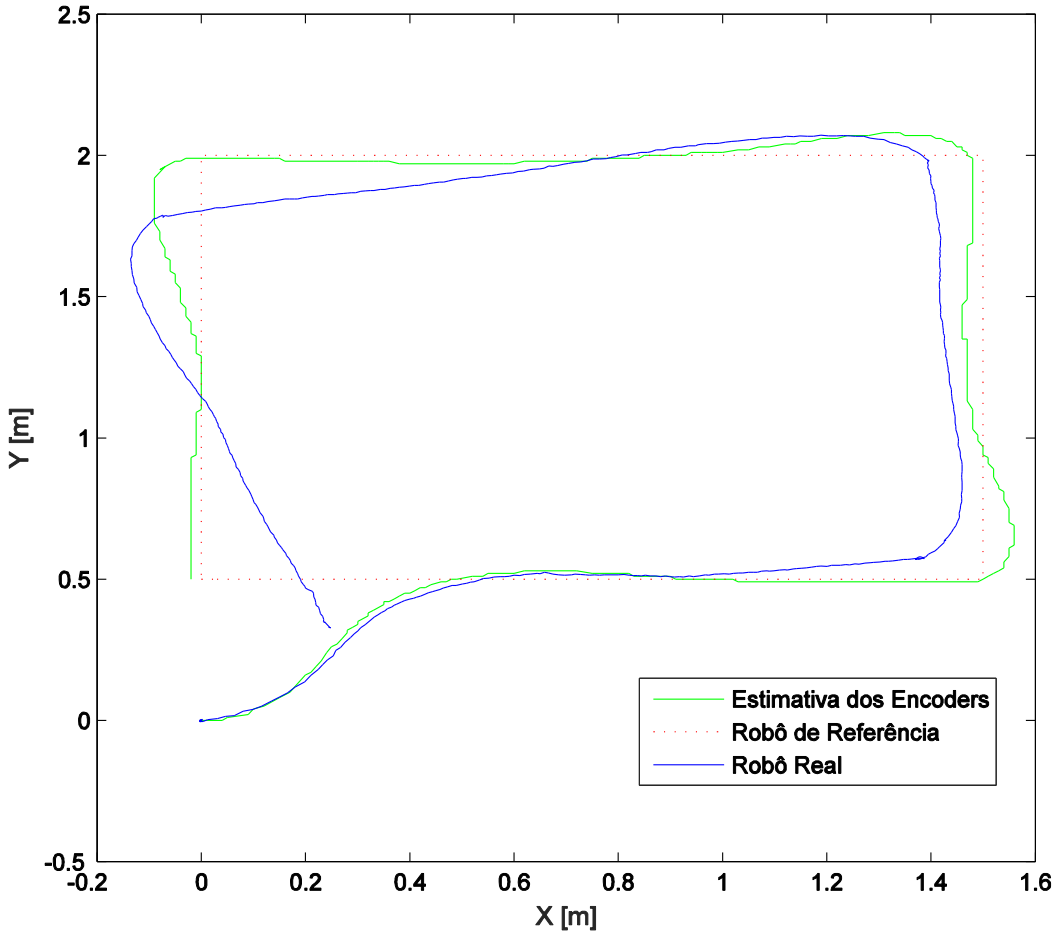
Tabela 4 – Erros iniciais do robô para trajetória quadrada.

Erros Iniciais		
x [m]	y [m]	θ [rad]
0	0.5	0

Fonte: Elaborado pelo autor.

A trajetória do robô foi representada na Figura 26. Os impactos do deslizamento e derrapagem são bastante notáveis nesse experimento, já que o robô teve que realizar uma curva mais acentuada para seguir a trajetória desejada. A curva verde mostra a postura estimada pelos *encoders*, enquanto que a linha tracejada em vermelho mostra a trajetória de referência, e a trajetória real do robô é mostrada pela curva em azul.

Figura 26 – Trajetória quadrado do robô utilizando posturas estimadas através dos *encoders*.

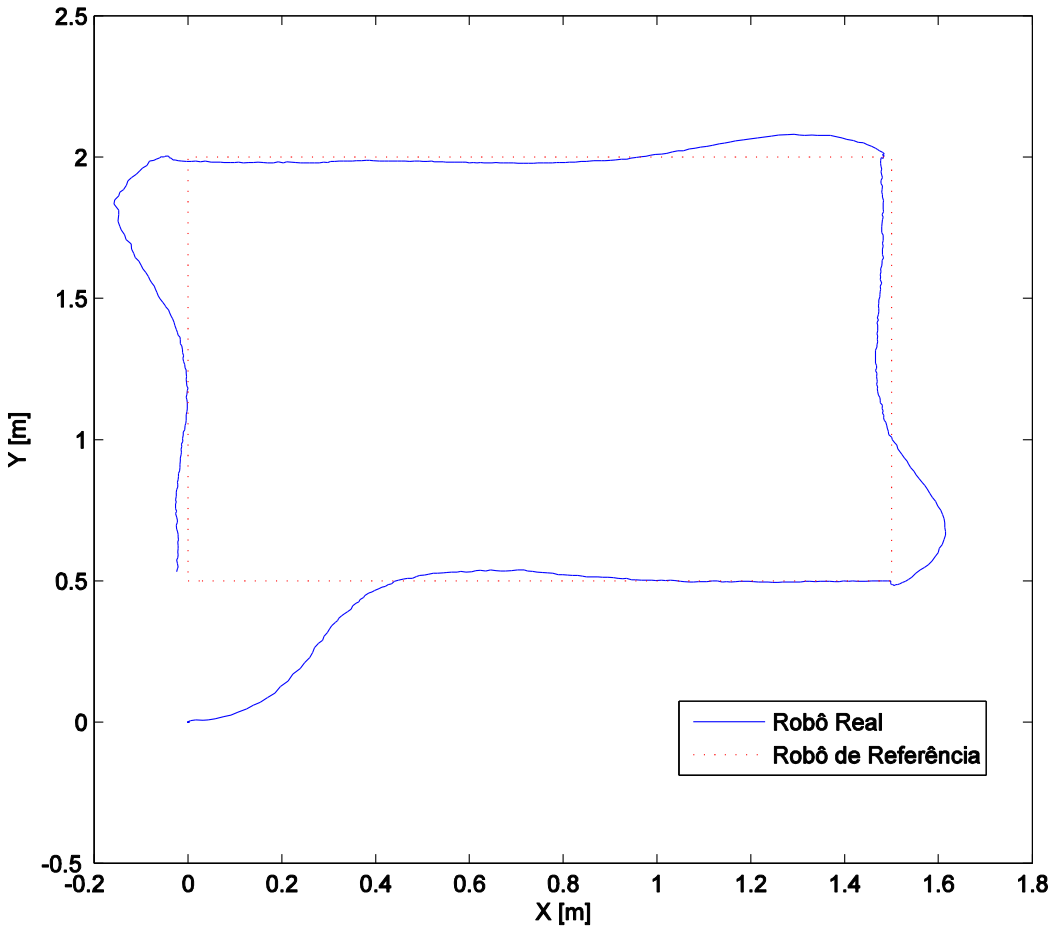


Fonte: Acervo do autor.

5.4 Experimento usando sistema SLAM 2D para fechar a malha de controle de posição de trajetória quadrada

No quarto experimento, foi utilizado a trajetória de referência quadrada novamente. Dessa vez, o algoritmo de localização e mapeamento simultâneo foi utilizado novamente para estimar a postura do robô através dos dados do sensor laser, e fechar a malha de controle.

Figura 27 – Trajetória quadrada do robô utilizando posturas estimadas usando sensor laser.



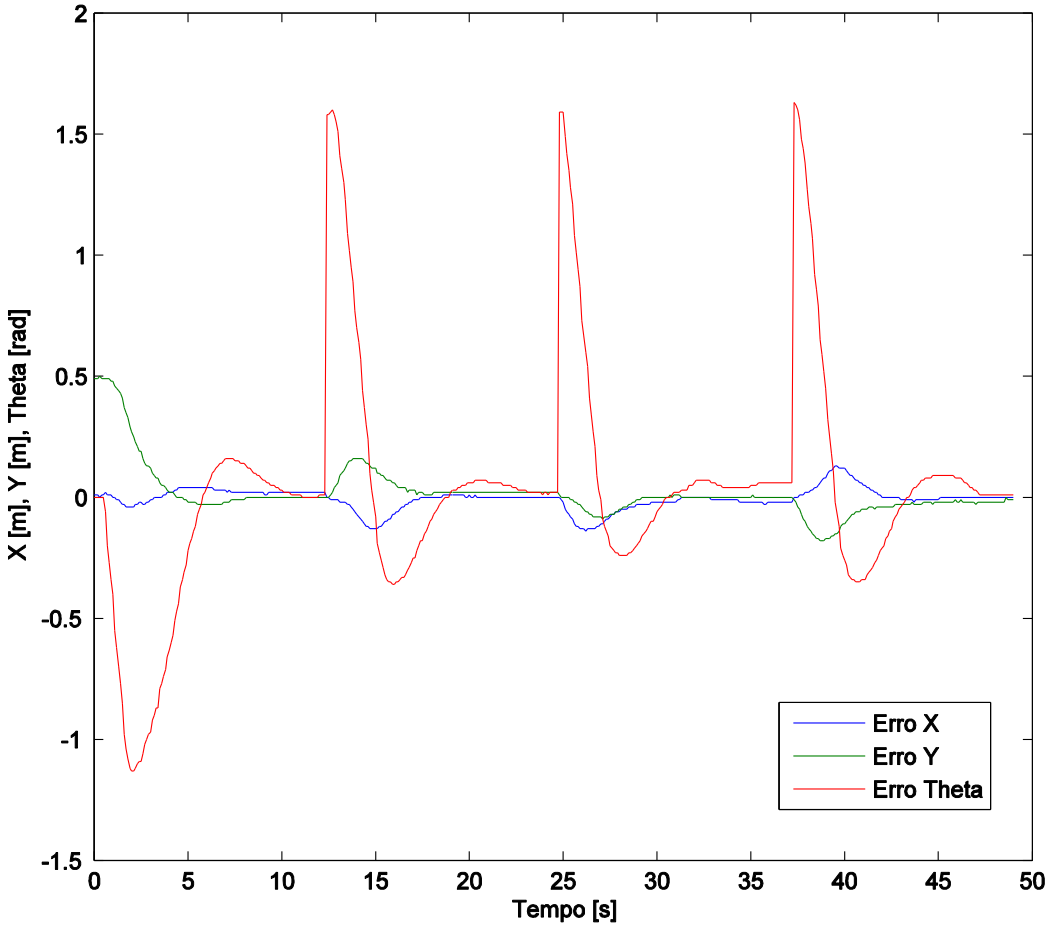
Fonte: Acervo do autor.

As curvas de trajetória foram apresentadas na Figura 27, enquanto que os erros são mostrados na Figura 28. A curva real do robô é mostrada pela curva em azul, e a trajetória desejada pela curva tracejada em vermelho.

A restrição de espaço no laboratório não permitiu a execução do teste utilizando lados maiores que o 1,5 m. Mais uma vez, é possível notar que o controlador consegue reduzir os erros de postura para zero ao longo do tempo, não havendo erro de estado estacionário com o uso de sistema de localização SLAM 2D. O erro na orientação do robô tem picos nos momentos quando o robô realiza curvas de 90 graus. No entanto, é possível notar nos tempos aproximados de 12 segundos e 47 segundos, que o esforço de controle consegue eliminar todos os três erros de postura simultaneamente.

O robô para quase que perfeitamente na posição desejada, de coordenadas [x=0, y=0.5].

Figura 28 – Erros de postura usando sensor LIDAR para fechar malha de controle de seguimento de trajetória quadrada.



Fonte: Acervo do autor.

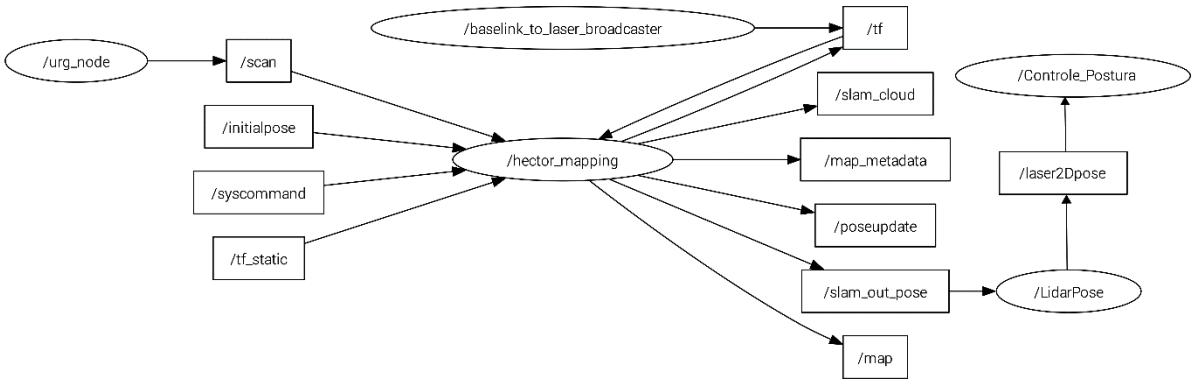
5.5 Experimento usando sistema SLAM 2D e com controlador de trajetórias embarcado no Raspberry PI

No quinto e último teste, o controlador de seguimento de trajetória foi embarcado no Raspberry PI como um novo nó do ROS (escrito em Python), deixando de ser executado no microcontrolador K64F da NXP. Para isso, a configuração de nós e tópicos foi modificada uma última vez, como mostrado na Figura 29. O código do novo nó está no Apêndice D.

O nó "/send_pose_to_serial" foi substituído pelo novo nó de controle e envio de comando de velocidades chamado "/Controle_Postura". Esse novo nó recebe os dados de postura do robô que são publicados ao tópico "/laser2Dpose". Em seguida, o nó executa a rotina de controle, e utiliza a porta serial para enviar as velocidades de comando dos motores da esquerda e da direita (saídas do controlador não-linear).

Dessa forma, a *string* utilizada no protocolo de comunicação com o microcontrolador foi alterada para o novo formato de oito caracteres "$\pm EEE \pm DDD$". Esse formato indica que as referências de velocidade dos motores da esquerda e da direita são valores com até três algarismos significativos, e com o sinal indicador de positivo ou negativo. As velocidades de referências enviadas para o controlador de velocidade dos motores (controlador interno ao driver dos motores) são dadas em rotações por minuto (RPM).

Figura 29 – Estrutura de nós e tópicos após adição do nó de controle de seguimento de trajetória.



Fonte: Elaborado pelo autor.

Mais uma vez, foi utilizada a trajetória de referência circular. Os erros iniciais foram os mesmos da Tabela 3. Os parâmetros especificados para este teste do controlador de seguimento de trajetória foram alterados, e são mostrados na Tabela 5. O parâmetro ζ, que representa o fator de amortecimento do controlador, foi alterado para 0.3 antes do início do experimento.

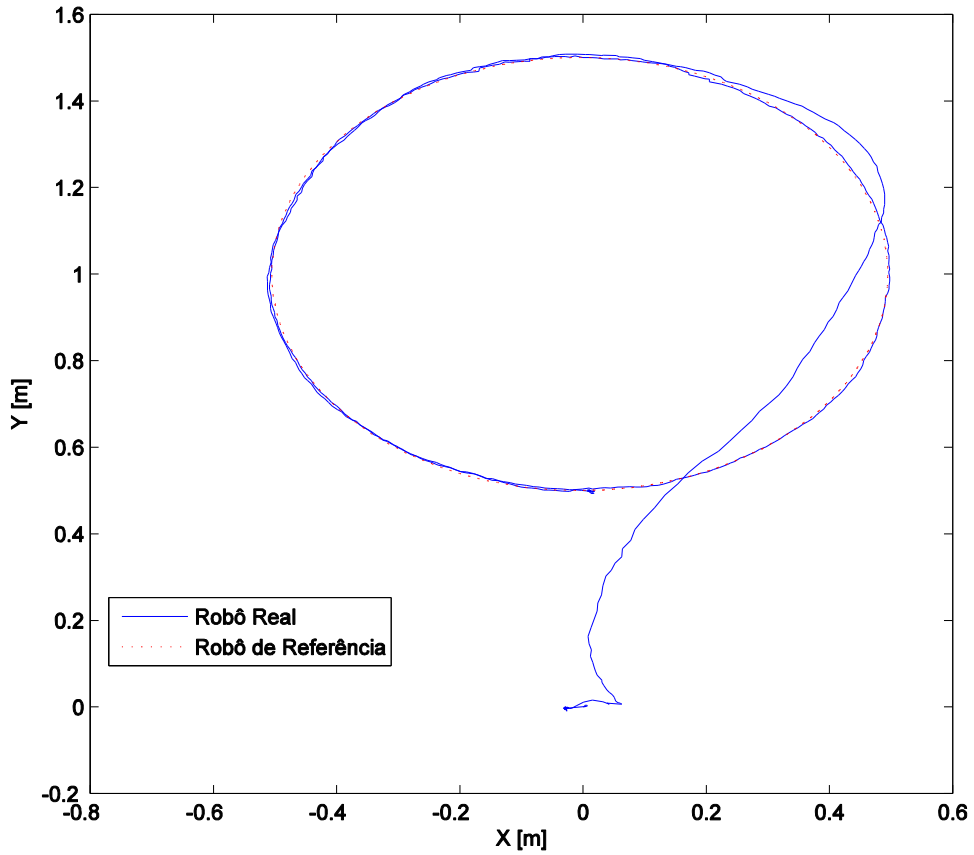
Tabela 5 – Parâmetros para o controlador de seguimento de trajetória embarcado no ROS.

Variável	Valor
g	40
ζ	0.3

Fonte: Elaborado pelo autor.

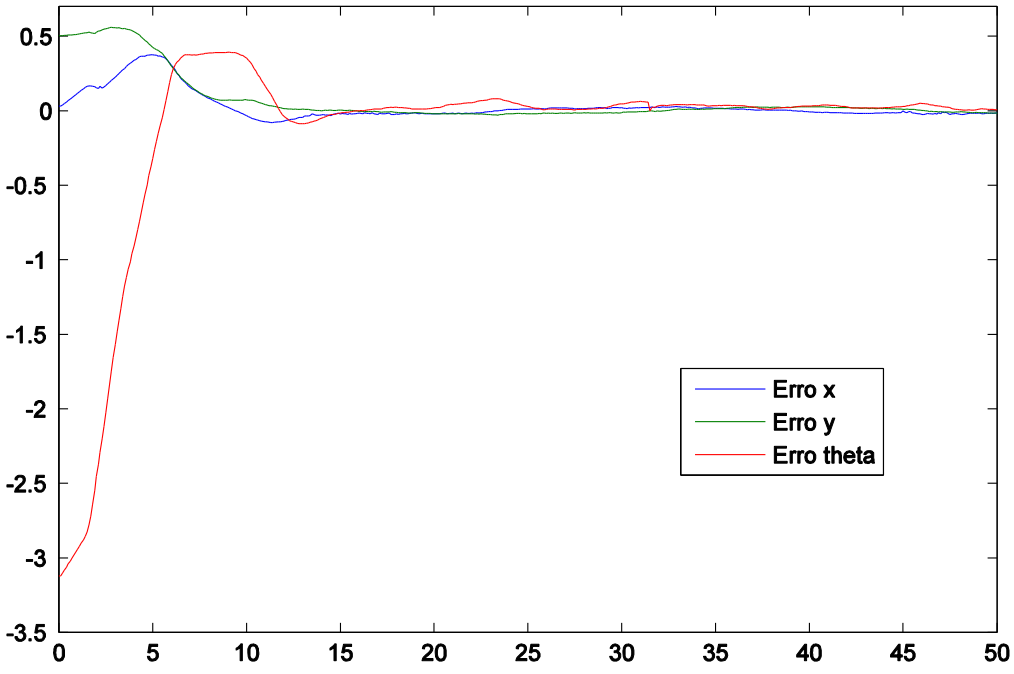
A curva de trajetória resultante pode ser vista marcada pela linha azul na Figura 30. A trajetória real segue a de referência e o controlador se torna, novamente, capaz de reduzir o erro de postura para zero, como mostrado na Figura 31. É possível notar que, ao diminuir o valor do parâmetro ζ para 0.3, o controlador se torna menos agressivo, corrigindo os erros no tempo igual a quinze segundos, sendo cinco segundos mais lento que no primeiro teste.

Figura 30 – Trajetória circular do robô com o controlador embarcado no ROS.



Fonte: Acervo do autor.

Figura 31 – Erros de postura com o controlador embarcado no ROS.



Fonte: Acervo do autor.

6 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho foi desenvolvido com o intuito de expor as principais desvantagens da utilização de *encoders* para fechar a malha de controle de posição de robôs móveis e propor, em sua substituição, o uso de um sensor LIDAR, mostrando suas principais vantagens e desempenho.

A estimação de posição e orientação do robô móvel por meio do uso da tecnologia LIDAR e um algoritmo SLAM se mostrou mais confiável do que o método antigo utilizado no robô, por meio de *encoders*. Isso se deve ao fato de que o robô se tornou bem mais inerte a distúrbios como escorregamentos, além de capaz de corrigir erros iniciais de postura. Até mesmo nessa situação, o robô se mostrou capaz de detectar sua postura no espaço, tornando o controlador de seguimento de trajetória capaz de encontrar o seu caminho de volta à trajetória desejada.

Também foi mostrado a utilização do *framework* de robótica *Robot Operating System* (ROS) como plataforma de integração do sensor e desenvolvimento para futuras aplicações de sistemas de controle na robótica.

Em adição, ficou demonstrado que a alta taxa de escaneamento de sensores LIDAR modernos e a performance melhorada dos processadores dos microcomputadores atuais tornaram o uso de algoritmos SLAM uma opção atraente para sensoriamento em sistemas de controle embarcado em robôs móveis.

6.1 Conclusões

O esquema de controle proposto se mostrou bastante efetivo, já que foi demonstrado que a utilização do sensor LIDAR permite implementação do controlador sem que o mesmo seja afetado pelo deslizamento das rodas do robô, como na realimentação por *encoders*. Além disso, foi demonstrado que o sistema pode ser implementado de maneira simples se aproveitando das características de modularização de *software* do *framework* ROS, sendo necessário apenas utilizar uma plataforma que tenha suporte a sistemas operacionais Linux, como o microcomputador Raspberry PI. Assim, é simples adaptar qualquer esquema de controle já existente ao modelo proposto, removendo a realimentação obtida por *encoders* ou outros sensores não tão precisos para medição de postura e adicionando a unidade de processamento habilitada para ler dados do sensor LIDAR e executar um algoritmo de mapeamento e localização simultâneos. A integração do sistema pode ser feita de maneira simples, por meio

de comunicação serial entre o Raspberry PI e qualquer microcontrolador já embarcado em um robô móvel. No último experimento foi demonstrado que, além da interface com o sensor LIDAR, o ROS pode ser usado também para embarcar os algoritmos de controle do robô.

Como possíveis limitações e desvantagens do sistema proposto, é possível citar a necessidade de utilização de um computador com alto poder de processamento, como o Raspberry PI, para obter dados do sensor LIDAR e aplicar o algoritmo SLAM. A utilização de outros sensores mais simples permite o uso de microcontroladores mais baratos, como é o caso quando a odometria via *encoders* é utilizada. Assim, a aplicação dessa tecnologia requer maior investimento, tendo em vista a melhoria do sistema de controle do robô móvel.

Em adição, é possível citar que o *hardware* utilizado tem aplicação limitada à ambientes fechados, já que a distância máxima medida pelo sensor laser utilizado é de quatro metros. Para conseguir aplicar a técnica proposta em ambientes externos, seria necessário realizar um investimento ainda maior, adquirindo um sensor LIDAR com maior alcance. A aplicação desta técnica também é limitada pela taxa de atualização e número de pontos obtidos pelo sensor.

6.2 Sugestões para Trabalhos Futuros

Como continuação desse trabalho, sugere-se a aplicação de novas estratégias de controle ao robô, em substituição ao controlador não-linear utilizado neste trabalho. Assim, seria mantida toda a estrutura de processamento de informação e obtenção de postura via sensor LIDAR, mas utilizando-se outras técnicas de controle, como modelos preditivos e adaptativos de controle.

Também se sugere o aproveitamento do mapa criado pelo algoritmo SLAM, para implementação de algoritmos para evitar obstáculos e para planejamento de trajetória.

Por fim, um acréscimo importante ao robô seria utilizar as bibliotecas já disponíveis no ROS para implementar fusão sensorial entre o sensor laser e uma unidade IMU (*Inertial Measurement Unit*), afim de obter estimativas de postura ainda mais precisas e no plano tridimensional, e implementar técnicas de controle mais robustas.

REFERÊNCIAS

AMANN, M.; BOSCH, T.; MYLLYLA, R.; RIOUX, M. “Laser ranging: a critical review of usual techniques for distance measurement”, **Optical Engineering**, v. 40, n. 1, jan. 2001.

DINIZ LOBO, T.; **Controle adaptativo auto-ajustável para controle de trajetórias de um robô móvel com rodas**. Dissertação (Mestrado em Engenharia Elétrica) – Universidade Federal do Ceará, 2016.

DISSANAYAKE, G.; DURRANT-WHYTE, H.; BAILEY, T. A computationally efficient solution to the simultaneous localization and map building (SLAM) problem. **IEEE International Conference on Robotics**, v. 2, p. 1009-1014, abr. 2000.

ELIAZAR, A.; PARR, R. DP-SLAM: Fast, robust simultaneous localization and mapping without predetermined landmarks. **International Joint Conference on Artificial Intelligence**, v. 3, p. 1135-1142, ago. 2003.

FUKE, Y.; KROTKOV, E., 1996, April. Dead reckoning for a lunar rover on uneven terrain. In Robotics and Automation, 1996. **Proceedings, 1996 IEEE International Conference on** (Vol. 1, pp. 411-416). IEEE.

GONZALEZ, J.; STENTZ, A.; OLLERO, A. A mobile robot iconic position estimator using a radial laser scanner. **Journal of Intelligent and Robotic Systems**, v.13, n. 2, p.161-179, jun. 1995.

H. Y., Chung; C. C, Hou; Y. S, Chen. Indoor Intelligent Mobile Robot Localization Using Fuzzy Compensation and Kalman Filter to Fuse the Data of Gyroscope and Magnetometer. **IEEE Transactions on Industrial Electronics**, v. 62, n. 10, p. 6436-6447, out. 2015.

HECTOR_MAPPING WIKI. Open Robotics Foundation. **ROS Wiki**. Disponível em: <http://wiki.ros.org/hector_mapping>. Acesso em: 15 de Janeiro, 2016.

HU, Jiubin Tan; HONGXING, Yang; XIPING, Zhao; SIYUAN, Liu. Phase-shift laser range finder based on high speed and high precision phase-measuring techniques. **The 10th International Symposium of Measurement Technology and Intelligent Instruments,** 2011.

KLANČAR, G.; MATKO, D.; BLAŽIČ, S. Mobile robot control on a reference path. **IEEE International Symposium on, Mediterrean Conference on Control and Automation**, (p. 1343-1348), jun. 2005.

KOHLBRECHER, S.; VON STRYK, O.; MEYER, J.; KLINGAUF, U., 2011, November. A flexible and scalable slam system with full 3d motion estimation. **In Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on** (pp. 155-160). IEEE.

MARÍN, L.; VALLÉS, M.; SORIANO, Á; VALERA, Á; ALBERTOS, P. Event-Based Localization in Ackermann Steering Limited Resource Mobile Robots. **IEEE/ASME Transactions on Mechatronics**, v. 19, n. 4, p. 1171-1182, ago. 2014.

QUIGLEY, Morgan; GERKEY, Brian; CONLEY, Ken; FAUST, Josh; FOOTE, Tully; LEIBS, Jeremy; BERGER, Eric; WHEELER, Rob; NG, Andrew. ROS: an open-source Robot Operating System. **ICRA Workshop on Open Source Software,** 2009.

R. L. S. Sousa; M. D. do Nascimento Forte; F. G. Nogueir; B. C. Torrico. Trajectory tracking control of a nonholonomic mobile robot with differential drive. **2016 IEEE Biennial Congress of Argentina (ARGENCON)**, Buenos Aires, Argentina, 2016, p. 1-6.

SIEGWART, Roland; NOURBAKHSH, Illah R.; SCARAMUZZA, Davide. **Introduction to Autonomous Mobile Robots, second edition**. The MIT Press, 2011.

SRINIVASAN, K.; GU, J. Multiple Sensor Fusion in Mobile Robot Localization. **Canadian Conference on Electrical and Computer Engineering**, Vancouver, BC, 2007, p. 1207-1210.

WANG, D.; LOW, C.B. Modeling and analysis of skidding and slipping in wheeled mobile robots: control design perspective. **IEEE Transactions on Robotics**, p.676-687, 2008.

ZHANG, J.; SINGH, S. Loam: Lidar odometry and mapping in real-time. **In Robotics: Science and Systems Conference**, p. 109-111, 2014.

APÊNDICE A – CÓDIGO-FONTE DO NÓ “/PONTO”

```
#!/usr/bin/env python
# coding=utf-8

import rospy
from std_msgs.msg import String
import serial
import sensor_msgs.msg
import time

i = 0

def callback(data):
    global i

    # Imprime a distância na tela.
    print "Distância: ", data.ranges[368], " [m]"

def listener():

    rospy.init_node('ponto', anonymous=True)
    rospy.Subscriber("scan", sensor_msgs.msg.LaserScan, callback)
    rospy.spin()

if __name__ == '__main__':
    listener()
```

APÊNDICE B – CÓDIGO-FONTE DO NÓ "/LIDAR2DPOSE"

```
#!/usr/bin/env python
# coding=utf-8

# Esse nó recebe a informação de posição do laser do tópico "slam_out_pose, que está em quartenion,
# e a transforma em coordenadas de euler, e então publica para o tópico /laser2Dpose.

import rospy
from std_msgs.msg import String
from geometry_msgs.msg import Pose2D
from geometry_msgs.msg import PoseStamped
import time
import tf

rospy.init_node('laser_pose', anonymous=True)
pub = rospy.Publisher('laser2Dpose', Pose2D, queue_size=10)

def callback(data):

    x = data.pose.orientation.x
    y = data.pose.orientation.y
    z = data.pose.orientation.z
    w = data.pose.orientation.w
    (r,p,yam) = tf.transformations.euler_from_quaternion([x,y,z,w])
    msg = Pose2D()
    msg.x = data.pose.position.x
    msg.y = data.pose.position.y
    msg.theta = yam
    pub.publish(msg)


def listener():

    rospy.Subscriber("slam_out_pose", PoseStamped, callback)
    rospy.spin()

if __name__ == '__main__':
    listener()
```

APÊNDICE C – CÓDIGO-FONTE DO NÓ "/SEND_POSE_TO_SERIAL"

```
#!/usr/bin/env python
# coding=utf-8

import rospy
from geometry_msgs.msg import Pose2D
from std_msgs.msg import String
import serial
import time
from math import pi
from math import sqrt

# Nessa versão foi adicionada a possibilidade de a posição do
# laser seja reinicida a partir do recebimento de um dado serial
# E também agora existe o envio da velocidade estimada do robô
# Salva dados enviados em um txt

pub = rospy.Publisher('syscommand', String, queue_size=10)

ser = serial.Serial ("/dev/ttyAMA0", 115200, timeout = 1)

f = open("/home/pi/pose&vel.txt","w")

# Variáveis globais

x_last = 0
y_last = 0
theta_last = 0
D = 0.4

def vel_roda(vlin,v_ohmega):

    ve = (vlin + v_ohmega*0.5*D)
    vd = (vlin - v_ohmega*0.5*D)

    return (ve,vd)

def vel(posetheta,posetheta_last,posex,posex_last,posey,posey_last):

    angular_increment = posetheta-posetheta_last


      if (posetheta <= 0.5 and posetheta_last >= pi):
          angular_increment = posetheta + 2*pi - posetheta_last

      if (posetheta >= pi and posetheta_last <= 0.5):
          angular_increment = posetheta - 2*pi - posetheta_last

      w_vel = (angular_increment)/0.1
      x_vel = (posex - posex_last)/0.1
      y_vel = (posey-posey_last)/0.1
      linear_vel = sqrt(x_vel**2 + y_vel**2)

      return (linear_vel,w_vel)
```

```
        def formata(x,y,theta,linear,angular,v_esquerda,v_direita):

            # Aproxima para duas casas decimais, e converte para string.
            string_x = "{:5.3f}".format(x)
            string_y = "{:5.3f}".format(y)
            string_theta = "{:5.3f}".format(theta)
            string_linear = "{:5.3f}".format(linear)
            string_angular = "{:5.3f}".format(angular)
            string_esquerda = "{:5.3f}".format(v_esquerda)
            string_direita = "{:5.3f}".format(v_direita)

            if (x>=0):
                string_x = '+'+string_x

            if (y>=0):
                string_y = '+'+string_y

            if (linear>=0):
                string_linear = '+'+string_linear

            if (angular>=0):
                string_angular = '+'+string_angular

            if (v_esquerda>=0):
                string_esquerda = '+'+string_esquerda

    if (v_direita>=0):
        string_direita = '+'+ string_direita

    string_theta = '+'+string_theta

    palavra1 = string_x+string_y+string_theta+string_linear+string_angular+'\r'
    aux1 = string_x + " " + string_y + " " + string_theta + " "
    aux2 = string_linear + " " + string_angular
    palavra2 =  aux1 + aux2

    return (palavra1,palavra2)

def callback(data):

    global theta_last, x_last, y_last

    # Faz com que theta fique no intervalo [0,2*pi]
    if (data.theta < 0):
        data.theta = data.theta + 2*pi

    # Calcula a velocidade
    vel_linear, vel_w = vel(data.theta,theta_last,data.x,x_last,data.y,y_last)
    ve, vd = vel_roda (vel_linear,vel_w)

    # Prepara os dados para serem enviados via serial
    palavra1, palavra2 = formata(data.x,data.y,data.theta,vel_linear,vel_w,ve,vd)

    # Le a porta serial para determinar se o dado pode ser enviado
    sinal = ser.read(1)
```

```
  # Reseta a posição ou envia o dado
  if (sinal == 'R'):
      msg = String()
      msg.data = "reset"
      pub.publish(msg)

  elif (sinal == '$'):
      ser.write(palavra1)  # Escreve na porta serial
      print palavra1
      f.write(palavra2+"\n") # Escreve o dado enviado em um arquivo de texto


    # Atualiza as variáveis globais

    theta_last = data.theta
    x_last = data.x
    y_last = data.y


def listener():

    rospy.init_node('SerialPose', anonymous=True)

    rospy.Subscriber("laser2Dpose", Pose2D, callback, queue_size = 10)

    rospy.spin()

if __name__ == '__main__':
    listener()
```

APÊNDICE D – CÓDIGO-FONTE DO NÓ "/CONTROLE_POSTURA"

```
#!/usr/bin/env python
# coding=utf-8

import rospy
from geometry_msgs.msg import Pose2D
import serial
import time
from math import radians
from math import pi
from math import cos
from math import sin
from numpy import sign

# Esse programa deve receber a informarção de posição do laser (x,y,theta),
# e calcular a velocidade dos motores esquerdo e direito do robô hulk de acordo
# Além disso, os erros são escritos em um arquivo de texto

ser = serial.Serial ("/dev/ttyAMA0", 115200, timeout = 1)
f = open("/home/pi/controle.txt","w")

# Variáveis

n = 0
indice = 0
w = 0.2
v = 0.1
Ts = 0.1
h = 0.1
R = 0.5/(2*pi)
D = 0.4


# Define a trajetória circular


# Declara listas

ref_x = []
ref_y = []
ref_t = []

ref_v = []
ref_w = []

size = 2*pi/(abs(w)*Ts)+1

# Valores iniciais
ref_x.append(0)
ref_y.append(0)
ref_t.append(0)
ref_v.append(0)
ref_w.append(0)
```

```
for i in range(1,int(size)):

    ref_v.append(v)
    ref_w.append(w)
    ref_t.append(ref_t[i-1] + h*w)
    ref_x.append(ref_x[i-1] + v*cos(ref_t[i])*h)
    ref_y.append(ref_y[i-1] + v*sin(ref_t[i])*h)


def callback (data):

    # recepção da posição

    global n
    global indice # Necessário para alterar o valor da variável
    x = data.x
    y = data.y
    theta = data.theta

    if (data.theta < 0):
        theta = (2*pi+data.theta)

    if (theta>6.2):
        theta = 0

    print x,y,theta

 # Calcula o erro
    e_x = ref_x[indice] - x
    e_y = ref_y[indice] - y
    e_theta = ref_t[indice] - theta


    if e_theta < -pi:
        e_theta = -e_theta

    e1 = cos(theta)*e_x + sin(theta)*e_y
    e2 = cos(theta)*e_y - sin(theta)*e_x
    e3 = e_theta

    # Lei de controle
    ksi = 0.3
    w_n = 1
    g = 40

    # Variaveis limite
    vu_max = 0.8
    wu_max = 0.8

    k1 = 2*ksi*w_n
    k3 = k1
    k2 = g*abs(ref_v[indice])

    v1 = -k1*e1
    v2 = -sign(ref_v[indice])*k2*e2 - k3*e3
```

```
  vu = ref_v[indice]*cos(e3)-v1
  wu = ref_w[indice] -v2;

  vu_antes = vu
  wu_antes = wu

  # Limitador

  if (vu > vu_max):
      vu = vu_max

  if (vu < 0):
      vu = -vu_max

  if (wu > wu_max):
      wu = wu_max

  if (wu < -wu_max):
      wu = -wu_max

  # Calculo da velocidade dos motores

  wd_f = (1/R)*(vu+wu*0.5*D)
  we_f = (1/R)*(vu-wu*0.5*D)

  wd_f = wd_f*60/(2*pi)
  we_f = we_f*60/(2*pi)

  if (n>=2):
      wd_f = 0
      we_f = 0

  # Enviar via serial as velocidades

  wd = '{0:03d}'.format(int(wd_f)) # 'É uma string'
  we = '{0:03d}'.format(int(we_f)) # 'É uma string'
  dado = wd + we

  sinal = ser.read(1)

  if (sinal == '$'):
      ser.write(dado+'\r')  # Escreve na porta seria
      indice += 1
  # Aproxima para três casas decimais, e converte para string.

palavra="{:6.3f}{:6.3f}{:6.3f}{:6.3f}{:6.3f}{:6.3f}{:6.3f}{:6.3f}{:6.3f}".format(
x,y,theta,ref_x[indice],ref_y[indice],ref_t[indice],e_x,e_y,e_theta)

  f.write(palavra+"\n")
  #print palavra
  # Monitora o indice
  #indice += 1

  if (indice == int(size)-1):
      indice = 0
      n += 1
```

```
def listener():

    rospy.init_node('SerialControle', anonymous=True)

    rospy.Subscriber("laser2Dpose", Pose2D, callback, queue_size = 10)

    rospy.spin()

if __name__ == '__main__':
    listener()
```

ANEXO A – ESPECIFICAÇÕES DO LASER HOKUYO URG-04LX-UG01

Product Name	Scanning Laser Range Finder
Model	URG-04LX-UG01
Light source	Semiconductor laser diode (λ=785nm), Laser safety Class 1 (IEC60825-1)
Power source	5V DC ±5% (USB buspower)
Current consumption	500mA or less (Rush current 800mA)
Detection distance	20mm ~ 4000mm
Accuracy	Distance 20mm ~ 1000mm : ±30mm* Distance 20mm ~ 4000mm : ±3% of measurement*
Resolution	1 mm
Scan Angle	240°
Angular Resolution	0.36°
Scan Time	100msec/scan
Interface	USB Version 2.0 FS mode (12Mbps)
Ambient (Temperature/Humidity)	-10 ~ 50°C / 85% or less (without dew and frost)
Preservation temperature	-25 ~ 75°C
Ambient Light Resistance	10000Lx or less
Vibration Resistance	Double amplitude 1.5mm 10 ~ 55Hz, 2 hours each in X, Y and Z direction, and 98m/s² 55Hz ~ 150Hz in 2 minutes sweep, 1 hours each in X, Y and Z direction
Impact Resistance	196 m/s², 10 times each in X, Y and Z direction
Protective Structure	Optics : IP64 Case : IP40
Insulation Resistance	10MΩ for DC 500Vmegger
Weight	Approx. 160 g
Case	Polycarbonate
External dimension (W×D×H)	50×50×70mm (Reference design sheet No.3502)

Fonte: https://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx_ug01.html.

ANEXO B –MENSAGEM DO ROS SENSOR_MSGS/LASERSCAN.MSG

File: sensor_msgs/LaserScan.msg

Raw Message Definition

```
# Single scan from a planar laser range-finder
#
# If you have another ranging device with different behavior (e.g. a sonar
# array), please find or create a different message, since applications
# will make fairly laser-specific assumptions about this data

Header header            # timestamp in the header is the acquisition time of
                         # the first ray in the scan.
                         #
                         # in frame frame_id, angles are measured around
                         # the positive Z axis (counterclockwise, if Z is up)
                         # with zero angle being forward along the x axis

float32 angle_min        # start angle of the scan [rad]
float32 angle_max        # end angle of the scan [rad]
float32 angle_increment  # angular distance between measurements [rad]

float32 time_increment   # time between measurements [seconds] - if your scanner
                         # is moving, this will be used in interpolating position
                         # of 3d points
float32 scan_time        # time between scans [seconds]

float32 range_min        # minimum range value [m]
float32 range_max        # maximum range value [m]

float32[] ranges         # range data [m] (Note: values < range_min or > range_max should be discarded)
float32[] intensities    # intensity data [device-specific units].  If your
                         # device does not provide intensities, please leave
                         # the array empty.
```

Fonte: http://docs.ros.org/api/sensor_msgs/html/msg/LaserScan.html.

ANEXO C –MENSAGEM PUBLICADA NO TÓPICO “/MAP_METADATA”

nav_msgs/MapMetaData Message

File: nav_msgs/MapMetaData.msg

Raw Message Definition

```
# This hold basic information about the characterists of the OccupancyGrid

# The time at which the map was loaded
time map_load_time
# The map resolution [m/cell]
float32 resolution
# Map width [cells]
uint32 width
# Map height [cells]
uint32 height
# The origin of the map [m, m, rad].  This is the real-world pose of the
# cell (0,0) in the map.
geometry_msgs/Pose origin
```

Compact Message Definition

```
time map_load_time
float32 resolution
uint32 width
uint32 height
geometry_msgs/Pose origin
```

Fonte: http://docs.ros.org/api/nav_msgs/html/msg/MapMetaData.html.

ANEXO D –MENSAGEM PUBLICADA NO TÓPICO "/MAP"

nav_msgs/OccupancyGrid Message

File: nav_msgs/OccupancyGrid.msg

Raw Message Definition

```
# This represents a 2-D grid map, in which each cell represents the probability of
# occupancy.

Header header

#MetaData for the map
MapMetaData info

# The map data, in row-major order, starting with (0,0).  Occupancy
# probabilities are in the range [0,100].  Unknown is -1.
int8[] data
```

Fonte: http://docs.ros.org/api/nav_msgs/html/msg/OccupancyGrid.html.

ANEXO E –MENSAGEM PUBLICADA NO TÓPICO “/POSEUPDATE”

geometry_msgs/PoseWithCovarianceStamped Message

File: geometry_msgs/PoseWithCovarianceStamped.msg

Raw Message Definition

```
# This expresses an estimated pose with a reference coordinate frame and timestamp

Header header
PoseWithCovariance pose
```

ANEXO F –MENSAGEM PUBLICADA NO TÓPICO “/SLAM_OUT_POSE”

geometry_msgs/PoseStamped Message

File: geometry_msgs/PoseStamped.msg

Raw Message Definition

```
# A Pose with reference coordinate frame and timestamp
Header header
Pose pose
```