UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA
DEPARTAMENTO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

IAGO PACHÊCO GOMES

DESENVOLVIMENTO DE UM CONTROLE ODOMÉTRICO PARA UM VEÍCULO TERRESTRE NÃO TRIPULADO COM MODELO DE DIREÇÃO *ACKERMANN*

Vitória da Conquista
2017

Iago Pachêco Gomes

Desenvolvimento de um Controle Odométrico para um Veículo Terrestre Não Tripulado com Modelo de Direção *Ackermann*

Trabalho de conclusão de curso, apresentado ao curso de Ciências da Computação, da Universidade do Sudoeste da Bahia, em Vitória da Conquista - Bahia, como requisito parcial para a obtenção do título de Bacharel em Ciências da Computação.

Orientador: Prof. Dr. Roque Mendes Prado Trindade

Coorientador: Prof. Dra. Alzira Ferreira da Silva

Bahia

Junho - 2017

Este trabalho é dedicado à minha família e
amigos da UESB

AGRADECIMENTOS

Quando ao final do curso, ao fim da escrita e desenvolvimento da monografia, muitos sentimentos nos permeiam, mas o mais gritante deles é a gratidão.

Gratidão porque nenhum trabalho pode ser feito sozinho, e este não foi uma exceção; Gratidão porque ao longo dos 4 anos cursando Ciência da Computação na UESB, diversas pessoas passaram por minha vida, e de alguma forma contribuiram para quem me tornei hoje.

Gostaria de agradecer imensamente a Deus, por toda a paciência e compeensão, principalmente nos momentos mais difíceis do curso, que necessitei de sua paz e tranquiladade, e ele soube amançar minha alma e me conduzir a este dia.

Agradeço também a todos que de alguma forma estiveram presentes nessa caminha, a minha família, principalmente meus pais por toda a calma e apoio; a meus amigos do ensino médio; a meus amigos do curso, do laboratório *LInDALVA* (Laboratório de Inteligência em Dispositivos de Arquitetura Livre e Veículos Autônomos), em especial à Matheus Thiago, Matheus Lima e a Yan, amigos (colegas) que me acompanham desde o primeiro semestre; aos que já formaram e para mim são grandes exemplos de pessoas, tia Jéssica, Vinicius, Thómas e Helber; a todos os amigos do LARA (Laboratório Remoto em Ava), Raphael, Pablo, Gustavo, Cássio, Rodrigo, João, Stefanie, Konai, Jorge e Luís; A Celina, uma pessoa realmente maravilhosa, que encanta o curso de Computação; a todos os professores, em especial à Professora Maísa, Alzira e ao Professor Roque, que me acolheram e orientaram desde que entrei no curso, e para mim são exemplos de profissionais e pessoas; E, novamente aos Professores Roque e Alzira, por me orientarem durante o processo da monografia; e, a Jenifer, uma pessoa muito especial que amo bastante, e que me fez uma pessoa muito mais feliz e melhor.

Em fim, são muitas as pessoas a quem devo minha gratidão, e isso só mostra o quão alto podemos chegar, pois o alicerce é muito forte. Muito obrigado a todos!!

"Houve um tempo em que o homem enfrentou o universo sozinho e sem amigos. Agora ele tem criaturas para ajudá-lo; criaturas mais fortes que ele próprio, mais fiéis, mais úteis e totalmente devota a eles. A humanidade não está mais sozinha. [...] Os robôs são uma espécie melhor e mais perfeita que a nossa"

Dra. Susan Calvin

(Eu, Robô. Isaac Asimov; tradução Aline Storto. 1º ed. ALEPH, pg. 15. 2014)

RESUMO

A localização é um componente fundamental para robôs móveis autônomos, que são capazes de navegar em diversos ambientes, e lidarem com o dinamismo destes. Existem diversas técnicas utilizadas para estimar a posição e orientação de um robô, elas podem ser divididas em relativas e absolutas. A implementação de ambas classes é essencial para um veículo robusto, pois cada uma possui uma especificidade característica. A odometria é um método de localização relativa classificada como *dead-reckoging* e é muito utilizada por conta de sua simplicidade e eficiência. Todavia ela possui um erro inerente à técnica que é acumulativo ao longo do tempo. Estes erros são oriundos de diferentes fontes, como o modelo cinemático do robô. Ainda assim, sua utilização continua sendo majoritária entre os sistemas robóticos móveis. Para desenvolver tal sistema é necessário, entretanto, o design da arquitetura de controle do robô, especificação de seus hardwares e softwares, simulação de cada componente, bem como algoritmos de navegação, localização, planejamento de trajetórias, e demais algoritmos que sejam pertinentes, como, por exemplo, os sistemas inteligentes para tomada de decisões e realização de tarefas. Este trabalho propôs a implementação de um controlador odométrico para o UGV – *Unmanned Ground Vehicle* com direção *Ackermann*, CELiNA da Universidade Estadual do Sudoeste da Bahia – UESB, campus Vitória da Conquista, com o auxílio do *framework* ROS – *Robot Operating System*, que além de possuir suporte a diversos simuladores de robótica, inclui diversas funcionalidades para o desenvolvimento de sistemas robóticos.

Palavras-chaves: odometria, localização de robôs móveis, veículos terrestres não tripulados - UGV.

ABSTRACT

The location is a key component for autonomous mobile robots that are able to navigate in different environments, and deal with the dynamism of these. There are several techniques used to estimate the position and orientation of a robot, they can be divided into absolute and relative location. The implementation of both classes is essential for a robust vehicle, because each class has a characteristic specificity. The odometry is a classified location relative method as dead-reckoging and is widely used because of its simplicity and efficiency. However th technique has an inherent error that is cumulative over time. These errors are from different sources, such as kinematic model robot. Still, its use remains majority among mobile robotic systems. To develop such system is necessary, however, the design of the robot control architecture, specification of your hardware and software, simulation of each component and navigation algorithms, location, trajectory planning, and other algorithms that are relevant, such as the intelligent systems for decision making and performing tasks. This job proposes the implementation of an odometer driver for UGV - Unmanned Ground Vehicle with *Ackermann* steering, *CELiNA* of the State University Southwest Bahia - UESB campus Vitoria da Conquista, with the help of the framework ROS - Robot Operating System, which in addition to has support for multiple robotic simulators, includes several features for the development of robotic systems.

Key-words: odometry, mobile robot localization, unmanned ground vehicle.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CELiNA	Carro Elétrico Livre com Navegação Autônoma
ROS	Robot Operation System
UMBmark	University of Michigan Benchmark test
UGV	Unmanned Ground Vehicles - Veículo Terrestre Não Tripulado
ICC	Instantaneous Center of Curvature - Centro de Curvatura Instantâneo
URDF	Universal Robot Description Format - Formato Univesal de Descrição de Robôs
IMU	Inertial Measurement Unit - Unidade de Medida Inercial
I^2C	Inter-Integrated Circuit- Circuito Inter-Integrado
CC	Corrente Contínua

LISTA DE SÍMBOLOS

P	Posição do robô
x	Posição do robô no eixo das abscissas
y	Posição do robô no eixo das ordenadas
θ	Orientação do robô
v_e	Velocidade linear da roda traseira esquerda
v_d	Velocidade linear da roda traseira direita
$d_{baseline}$	Distância entre os pontos de contatos das rodas traseiras do robô
θ_{left}	Angulação da roda frontal esquerda do modelo *Ackermann*
θ_{right}	Angulação da roda frontal direita do modelo *Ackermann*
W_{base}	Distância entre os eixos frontal e traseiro do veículo com modelo *Ackermann*
r_{left}	Raio da circunferência realizado pela roda esquerda (frontal ou traseira)
r_{right}	Raio da circunferência realizado pela roda direita (frontal ou traseira)
r_{center}	Raio da circunferência realizado pela roda virtual central (frontal ou traseira)
v	Velocidade linear
ω	Velocidade Angular
R	Raio da circunferência
α	Angulação de esterçamento do veículo
M_{rot}	Matriz de rotação
t	Instante de tempo
d_{left}	Distância percorrida pela roda traseira esquerda
d_{right}	Distância percorrida pela roda traseira direita
N_{left}	Quantidade de interrupções ocorridas em um intervalo de tempo para a roda traseira esquerda

N_{right}	Quantidade de interrupções ocorridas em um intervalo de tempo para a roda traseira direita
Re_{left}	Resolução do encoder da esquerda para uma revolução completa da roda
Re_{right}	Resolução do encoder da direita para uma revolução completa da roda
ρ_{left}	Perímetro da roda traseira esquerda
ρ_{right}	Perímetro da roda traseira direita

SUMÁRIO

1 INTRODUÇÃO

Os robôs estão cada vez mais presentes no dia-a-dia das pessoas, e eles se apresentam nas mais diversas formas, características, utilidades e aparências (WOLF et al., 2009). Essa presença é resultado de anos de desenvolvimento tantos dos componentes robóticos, como da sociedade. Por algum tempo os robôs foram usados para substituir humanos em processos monótonos e/ou perigosos, todavia, com o tempo também começaram a desempenhar outras atividades como educacionais, para monitoramento de ambiente, limpeza doméstica, transporte, fins militares, etc. (KANNIAH; ERCAN; CALDERON, 2013).

Desta forma, várias áreas da robótica recebem destaques na atualidade. A robótica industrial teve um crescimento bastante acentuado, acompanhando o desenvolvimento industrial em vários setores do mercado, como, por exemplo, o automobilístico e o de produção de bens de consumo. Por conta disso, as pesquisas voltadas aos robôs manipuladores e industriais receberam grande foco dos cientistas (WOLF et al., 2009; SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011). Os robôs industriais possuem como principal característica a capacidade de realizar tarefas repetitivas com muita velocidade e precisão. Eles atuam em ambientes controlados ou semi-controlados, dessa forma, seus *softwares* são projetados para que os robôs realizem tarefas "ótimas", reduzindo tempo e custo de produção, aumentando a qualidade dos produtos e a lucratividade do serviço (SECCHI, 2012). Além disso, por atuarem em tais ambientes, esses *software* não precisam considerar muitas variáveis externas ao robô e ao seu trabalho, uma vez que nesses ambientes é diminuta a quantidadede fatores que influenciam diretamente o controle dos robôs, em relação a ambientes dinâmicos (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

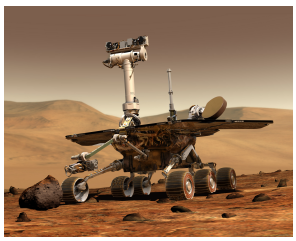
Apesar da precisão e eficiência, os robôs industriais possuem uma desvantagem quando estende-se sua aplicação. Os robôs manipuladores e industriais atuam em um espaço de trabalho fixo, ou seja, suas ações são restritas a um determinado espaço que é limitado pela geometria do robô. Ao contrário dos robôs industriais, os robôs móveis podem mudar de espaço e locomover-se por todos os ambientes permitidos por sua topologia física e arquitetura de software (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

Os robôs móveis autônomos, são uma classificação para robôs móveis que se distinguem das demais por possuírem autonomia em locomoção e em tomada de decisões. Um exemplo deles são os UGVs, do inglês *Unmanned Ground Vehicles* – Veículos Terrestres Não Tripulados. Os UGVs são capazes de se locomover em ruas de cidades, rodovias e em outros ambientes, sem a intervenção humana. Ao mesmo tempo em que planejam trajetórias, eles desviam de obstáculos, obedecem à sinalização e leis de trânsito, encontram vagas de estacionamento livres e realizam manobras autonomamente (GE, 2006).

Alguns desses veículos robóticos são bastantes conhecidos, como: Os Rovers enviados a Marte (Fig. 1(a); Stanley, vencedor da competição *Darpa Grand Challenge* de 2005 (Fig. 1(b)); Boss, UGV da *Carnegie Mellon University* vencedor da *Darpa Grand Challenge* de 2007 (Fig. 1(c)); e o UGVs da Google que transitam nas ruas e rodovias do estado da Califórnia – USA (Fig. 1(d)) (WOLF et al., 2009; BAJRACHARYA; MAIMONE; HELMICK, 2008; ODEDRA; PRIOR; KARAMANOGLU, 2009; ROMERO et al., 2014).

Existem alguns UGVs também no Brasil, que são objetos de pesquisa em algumas universidades, como: o CARINA – Carro Robótico Inteligente para Navegação Autônoma (Fig. 1(e)), do Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC-USP); o CADU da Universidade Federal de Minas Gerais - UFMG (Fig. 1(f)); e o IARA – *Intelligent Autonomous Robotic Automobile*, da Universidade Federal do Espírito Santos - UFES (Fig. 1(g)).

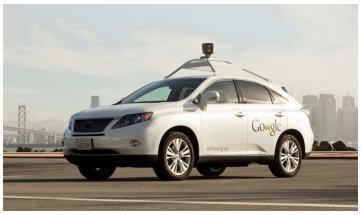
Figura 1 – Exemplos de veículos robóticos.



(a) Rovers da NASA

(b) Stanley

(c) Boss



(d) UGV Google

(e) CARINA

(f) CADU

(g) IARA

Fonte: (NASA, 2016a; WILLIAMS, 2016; INSTITUTE, 2016; MATSUBARA, 2016; ICMC, 2016; UFMG, 2016; MARCHIONI, 2016)

A habilidade de locomoção dos robôs móveis traz consigo grandes desafios tanto na concepção mecânica, quanto no desenvolvimento e implementação do software (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011). O robô móvel deve ser capaz de atuar nos mais

diversos tipos de ambientes, em que as posições dos obstáculos podem mudar a qualquer momento e novos objetos podem ser incluídos no ambiente, obrigando-os a atualizarem suas representações de mundo. Alcançar os objetivos com tamanho dinamismo no ambiente é um dos maiores desafios na área da robótica móvel. Essa meta exige que o controle do robô seja capaz de planejar trajetórias livres de obstáculos, executá-las corretamente e realizar suas tarefas satisfatoriamente(WOLF et al., 2009; GE, 2006).

Duas das características mais importantes para os robôs móveis inteligentes é a navegação e a localização (JENSFELT, 2001). Existem diversas técnicas desenvolvidas e estudadas para solucionar tais problemas, inclusive uma técnica que engloba os dois ao mesmo tempo, o SLAM - *Simultaneous Localization and Mapping*. Para solucionar o problema de navegação entretanto, é necessário que o robô seja capaz de se localizar. As técnicas de localização são divididas em locais ou relativas, e globais ou absolutas. A primeira se refere a uma localização em relação à posições já alcançadas pelo robô, com um intervalo de distância mais limitado. Já a localização global se refere à localização do robô em uma escala muito maior, e permite o planejamento para trajetórias de longas distâncias (BORENSTEIN et al., 1996).

Entretanto, segundo Siegwart, Nourbakhsh e Scaramuzza (2011) em um sistema robótico robusto não basta apenas implementar uma dessas técnicas de localização, pois o dinamismo no ambiente exige que ambas as abordagens estejam presentes, uma vez que as técnicas de localização relativas possuem uma eficiência maior para ambientes dinâmicos, como por exemplo locais de trânsito de pessoas. E as técnicas de localização global são indispensáveis para robôs que percorrem grandes distâncias.

Uma famosa técnica de localização relativa e bastante empregada nos sistemas robóticos, é a odometria (GONZALEZ et al., 2012). Sua popularidade é resultado de sua simplicidade e baixo custo (BORENSTEIN; FENG, 1995a). Esta consiste na estimação da posição do robô por meio de um acomulatório incremental entre as distâncias percorridas em função do tempo.

Para seu uso, geralmente, sensores conhecidos como encoders são acoplados as rodas do veículo. Estes sensores são capazes de aferir a quantidade de giros da roda, e por meio do modelo cinemático do veículo é possível conhecer seu deslocamento, bem como orientação. Diversos autores utilizam essa técnica em sistemas robóticos como: O'Kane (2006) que utiliza a odometria para estimar a localização global de um robô; Inthiam e Deelertpaiboon (2014) que utiliza a odometria para estimar a localização local de um robô omnidirecional; Reinstein, Kubelka e Zimmermann (2013) que utiliza a odometria para localização de robôs móveis que se locomovem por derrapagem, conhecidos como *Skid-steer*; Nourani-Vatani, Roberts e Srinivasan (2009), Chenavier e Crowley (1992) e Gonzalez et al. (2012) que utilizam a abordagem odométrica com visão computacional como outra alternativa aos encoders.

A maior preocupação em torno da técnica de odometria é o erro acumulativo que suas estimativas possuem, tal problema é objeto de estudo de diversos autores, como: Antonelli, Chiaverini e Fusco (2005), Chong e Kleeman (1997), Censi et al. (2013) e Borenstein e Feng

(1995a) . Por conta disso, apesar do erro inerente a técnica, que pode ser proveniente por exemplo do modelo cinemático do robô, a odometria ainda continua sendo uma técnica muito utilizada (BORENSTEIN et al., 1997).

Segundo Wolf et al. (2009) o projeto de um UGV é uma tarefa árdua, complexa e custosa. Pois envolve desde a concepção do *hardware* e mecânica do robô até o planejamento, construção e implementação de um *software* controlador. Para tanto, algumas técnicas são empregadas para diminuir o custo, o retrabalho e aumentar a eficiência do sistema. A principal técnica é a simulação dos dispositivos e seus softwares antes de sua implementação física ou de testes mais complexos com o robô.

No processo de simulação várias variáveis do ambiente podem ser recriadas dando uma fidelidade maior ao teste. Além disso, ajustes podem ser feitos ao longo dos experimentos, e condições de um teste podem ser repetidas em outros por meio de logs de simulações. Isso é conveniente, pois permite uma comparação mais fidedigna entre os programas, uma vez que não é possível repetir as mesmas condições de teste em um ambiente real.

Outra técnica é o uso de *frameworks* e *middlewares* de programação de robôs, nesse caso, esses *softwares* são utilizados para facilitar o entendimento e implementação de programas robóticos nos institutos de pesquisa e na comunidade científica em geral, mas também no desenvolvimento de robôs com fins industriais ou de mercado (WOLF et al., 2009). O ROS – *Robot Operating System* é um *framework*, *open-source*, que permite a implementação de programas controladores. Sua arquitetura é baseada em nós dando suporte à implementação de sistemas distribuídos. Outra característica desse *framework* é ser compatível com vários simuladores, como o Gazebo e o Morse, além de ser compatível com implementações nas linguagens C/C++, Python, etc. (QUIGLEY et al., 2009).

1.1 Objetivos

O objetivo geral deste trabalho é implementar um controle odométrico utilizando o *framework* ROS (*Robot Operating System*) para o controle de um UGV com direção *Ackermann*.

1.1.1 Objetivos Específicos

- Analisar arquiteturas de controle para robôs móveis; (Revisão Bibliográfica)
- Analisar os modelos disponíveis de controle cinemático de robôs móveis com direção *Ackermann*; (Revisão Bibliográfica)
- Analisar os modelos de controle odométrico para robôs móveis; (Revisão Bibliográfica)
- Analisar técnicas existentes para a correção do erro sistemáticos proveniente da localização por odometria; (Revisão Bibliográfica)

- Identificar os requisitos funcionais de um software para controle odométrico do UGV com direção *Ackermann* usando o *framework* ROS;
- Analisar a influência dos ruídos da cinemática no controle odométrico;
- Identificar a eficácia e eficiência do *software* desenvolvido;
- Simular o controle odométrico implementado com auxilio de um *software* de simulação de robôs móveis.

1.2 Justificativa

A robótica móvel é uma área em plena expansão, tanto em sua concepção acadêmica, no que diz respeito às suas áreas de estudos, quanto nas aplicações na sociedade. Suas aplicações são as mais diversas, pois um robô móvel deve ser capaz de se locomover em vários ambientes cujas variáveis são bastante dinâmicas, como, por exemplo, a posição dos obstáculos (ROMERO et al., 2014). Essa característica amplia a gama de aplicações dos robôs, ao passo que aumenta o nível de dificuldade na concepção de hardware e software dos mesmos.

Um sistema robótico é composto por vários módulos, cada qual responsável por uma funcionalidade, a exemplo dos planejadores de trajetória, que calculam um caminho livre de obstáculos de um ponto A a um ponto B; sistema de navegação, responsável por coordenar a navegação do robô nas trajetórias calculadas; sistema de localização, que orientam o robô em um ambiente, estimando sua posição e orientação; e, o sistemas inteligentes que planejam as tarefas a serem executadas.

O sistema de localização é um componente essencial aos robôs móveis inteligentes. Essa importância é evidenciada pela quantidade de pesquisadores que se dedicam ao estudo da problemática como Borenstein et al. (1996), Nistér, Naroditsky e Bergen (2006), Antonelli, Chiaverini e Fusco (2005), Olson (2000), Fox, Burgard e Thrun (1999), Bezerra (2004), Jensfelt (2001) e Dellaert et al. (1999).

Os algoritmos de localização de robôs são divididos em duas classes, localização relativa e a localização absoluta. Segundo Siegwart, Nourbakhsh e Scaramuzza (2011) ambas as partes são fundamentais e devem estar presentes em um sistema robótico robusto. A localização relativa, como a odometria, é aquela com maior taxa de atualização e fornece estimativas a uma velocidade bem maior que as técnicas de localização globais, apesar de estarem sujeitas ao acumulo de erros em função do tempo. Já a localização global, mesmo que demanda maior poder de processamento, é fundamental para navegação em mapas grandes, cujo deslocamento do robô também alcançam grandes distâncias (BORENSTEIN; FENG, 1995a).

Este trabalho objetiva desenvolver um sistema de localização relativa, a odometria, em um UGV – *Unmanned Ground Vehicle*, da Universidade Estadual do Sudoeste da Bahia – UESB,

chamado CELiNA – Carro Elétrico Livre com Navegação Autônoma. Será considerado tanto a modelagem odometríca para localização relativa do veículo, como também os modelos de diminuição das imprecisões inerentes a técnica. Além disso, o localizador odométrico será testado em um ambiente simulado, com o auxilio do framework ROS – *Robot Operating System*, cuja popularidade cresce no meio acadêmico por conta de sua versatilidade e funcionalidades agregadas (QUIGLEY et al., 2009).

1.3 Estrutura do Documento

Além deste capítulo, o trabalho está dividido em:

- No **Capítulo 2** é apresentado a fundamentação teórica necessária para a conclusão do trabalho;
- No **Capítulo 3** é descrito a metodologia e desenvolvimento do presente trabalho;
- No **Capítulo 4** é apresentado e discutido os resultados obtidos, detalhando os recursos utilizados e a solução proposta;
- No **Capítulo 5** apresenta-se a conclusão deste trabalho e os trabalhos futuros.

2 ROBÓTICA

Segundo a Organização Internacional de Padronização (ISO – *International Organization for Standardization*) um robô é um mecanismo programável que possui dois ou mais eixos, graus de autonomia, e é capaz de se locomover em um ambiente executando as tarefas pretendidas (ISO, 1994). O termo robô tem origem nas palavras tchecas *"robota"* e *"robotnik"* que significam *"trabalho"* e *"trabalhador"*, respectivamente (DUDEK; JENKIN, 2010).

Essa palavra surgiu em uma peça teatral do escritor tcheco Karel Čapek (1890 – 1938) conhecida como R.U.R. (*Rossus's Universal Robots* – Robôs Universais de Rossum) em 1923 (MURPHY, 2000), lançada no Brasil com o título de *"A Fábrica de Robôs*" pela editora Hedra. A contribuição literária e cinematográfica foram e continuam sendo importantíssimas para o processo histórico e desenvolvimento do campo de estudos sobre os robôs, conhecida como robótica. Podem-se citar diversas obras que abordaram o assunto, como os filmes: Metrópolis(1926); O Dia em que a Terra Parou (1951); O Planeta Proibido (1956); *Blade Runner*: O Caçador de Androides (1982); O Exterminador do Futuro (1984); O Homem Bicentenário (1999); A. I. Inteligência Artificial (2001); Eu, robô (2004); entre tantos outros. E as obras literárias: *The Sandman* (1817); *Frankenstein* (1818); *The Steam Man of the Pairies* (1868); *Liar!* (1941); *Runaround (1941)*; entre outros. Cada livro, peça teatral ou filme possui sua própria forma de abordar a ideia de um dispositivo mecânico ou biomecânico inteligente, submisso às ordens e que possui um objetivo ou propósito (MURPHY, 2000). Em algumas dessas obras os autores inclusive retratam um temor comum a humanidade, o fato de suas criações se tornarem mais inteligentes do que foi planejado, desenvolvendo alguma espécie de consciência e por conseguinte acabarem por voltar-se contra os humanos (DUDEK; JENKIN, 2010; SOLUTIONS, 2007).

De acordo com Romero et al. (2014) e Dudek e Jenkin (2010) esses temores são resultados das situações hipotéticas advindas das imaginações dos autores e da *"licença poética"* para a criação dos personagens, que, por exemplo, em alguns casos possuem componentes biológicos e/ou desenvolvem consciência. Todavia esses temores não são justificáveis, pois os robôs não usufruem de liberdade para pensarem como os humanos. Ou seja, os robôs são feitos para executarem ações para as quais foram programados, não sendo capazes de *"pensar"* ou ter *"livre-arbítrio"* (ROMERO et al., 2014).

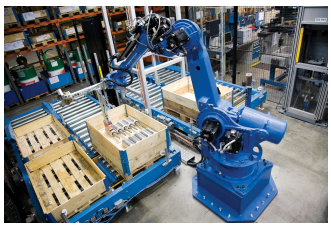
Apesar das obras de ficção retratarem ambientes amistosos ou fictícios a cerca da robótica, um fato é inegável, os robôs já estão bastante ambientados no cotidiano das pessoas, e cada vez mais tornam-se visíveis suas aplicações (WOLF et al., 2009; GATES, 2007). Segundo NIKU (2013) e Wolf et al. (2009) algumas das aplicações para os robôs são:

- **Aplicações industriais:** A aplicação industrial é uma das mais importantes aplicações

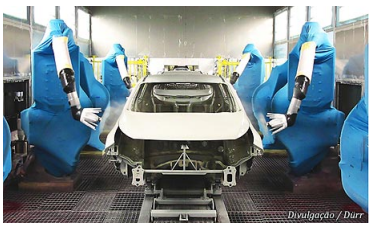
da robótica. Conforme Siegwart, Nourbakhsh e Scaramuzza (2011) os manipuladores industriais são robôs velozes e precisos, por conta disso alcançaram tanto sucesso na indústria. Esses robôs atuam em ambientes semi controlados ou controlados e são capazes de executar muito bem as tarefas que lhes são incumbidos. NIKU (2013) elenca algumas dessas tarefas, por exemplo: operações de retiradas e deposições; soldagem; pinturas; inspeção de peças; tarefas de montagem; entre outras (Figura 2).

Figura 2 – Exemplo de robôs industriais.

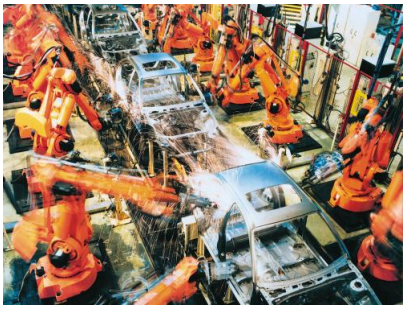
(a) robô de montagem



(b) robô de retiradas e deposições



(c) robô para pintura



(d) robô de soldagem

(e) robô de inspeção

Fonte: (BLACK, 2016; SOLUTION, 2016; FATOR, 2016; INDUSTRIAL, 2016; PHARMA, 2016)

- **Aplicações médicas:** Os componentes robóticos e seus software*s* dotam os robôs de uma precisão milimétrica, além de lhes proporcionarem uma visão muito mais ampliada e precisa na hora de procedimentos críticos. Por conta disso a aplicação dos robôs na medicina torna-se cada vez mais comum. Destes, destacam-se o Robodoc (Fig. 3(a)) da Curexo Technology Corporation e o Da Vinci (Fig. 3(b)) da Intuitive Surgical, ambos usados em uma variedade de operações cirúrgicas. Esses robôs podem ser teleoperados por médicos e são usados principalmente em microcirurgias, pois possuem grande precisão nos movimentos, diminuindo a incisão nos pacientes.

- **Aplicações domésticas:** O uso doméstico é outra situação aonde se encontram diversos tipos de robôs realizando atividades desde limpeza à assistência pessoal. Existem por exemplo: o robô Roomba (Fig. 4(a)) que aspira pó e o limpador de piscina Mirra (Fig. 4(b)), ambos da empresa iRobot; o robô ASIMO (Fig. 4(c)) da Honda, que foi criado para ser um assistente robótico domiciliar, capaz de realizar diversas tarefas como carregar objetos e

Figura 3 – Exemplo de robôs em aplicações médicas.

(a) Robodoc (b) Robô da Vinci

Fonte: (SURGICAL, 2016; NIKU, 2013)

fazer companhia a idosos e crianças; o robô Jibo (Fig. 4(d)), desenvolvido pela professora Cynthia Breazeal do Laboratório de Mídia do MIT e vendido pela companhia Indiegogo, que é capaz de ler histórias para crianças, ditar receitas culinárias, ler mensagens, tirar fotos, entre outras atividades domésticas; etc.

Figura 4 – Exemplo de robôs domésticos.

(a) Roomba (b) Mirra (c) ASIMO (d) Jibo

Fonte: (ROMERO et al., 2014; IROBOT, 2016; JEFFERSON, 2016; BREAZEAL, 2016)

- **Entretenimento e educação:** Kanniah, Ercan e Calderon (2013) destacam a robótica como uma área bastante interdisciplinar, que envolve conceitos da matemática, mecânica, física, biologia, ciência da computação, sociologia, engenharia elétrica, entre outras disciplinas; por conta disso é uma ferramenta com grande potencial para a educação. Além disso, os robôs também são utilizados para o entretenimento das pessoas. Alguns exemplos que se adéquam à estas áreas de aplicação são:Lego MindStorms (Fig. 5(a)), Aibo da Sony (Fig. 5(b)), Pleo da Innvo Labs (Fig. 5(c)), RoboSapiens da WowWee (Fig. 5(d)), NAO da Somai (Fig. 5(e)), entre outros (ROMERO et al., 2014).

- **Aplicações militares e segurança:** esta é uma das áreas da robótica com maior investimento (BRUMSON, 2011). Segundo Brumson (2011) as aplicações vão desde robôs que

Figura 5 – Exemplos de robôs que são aplicados na educação e no entretenimento.

(a) Lego MindStorms

(b) Aibo da Sony

(c) Pleo

(d) RoboSapiens Blue

(e) Robô NAO

Fonte: (LEGO, 2016; SONY, 2016; PLEO, 2016; WOWWEE, 2016; SOMAI, 2016)

realizam o monitoramento e patrulhas de áreas como os Drones (Fig 6(b)), à robôs que desarmam bombas e operaram armas de fogo (Fig 6(c)). Outro exemplo é o BigDog (Fig 6(a)), um quadrúpede para carregamento de cargas pesadas que é capaz de andar, correr, subir e descer barrancos em diversas condições, como terrenos pedregosos ou congelados (WOODEN et al., 2010).

Figura 6 – Exemplos de militares.

(a) BigDog

(b) drone militar

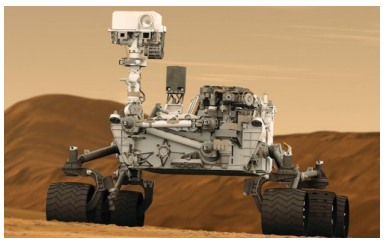
(c) robô HDMS

Fonte: (WOODEN et al., 2010; PEREZ, 2016; MOREIRA, 2016)

- **Transporte e mobilidade:** Os robôs móveis são capazes de se locomoverem em ambientes transportando cargas e/ou pessoas. A sua popularidade aumenta cada vez mais, principalmente no tocante aos autônomos (WOLF et al., 2009). Exemplos bastante famosos destes são os Rovers enviados a Marte (Fig. 7(a)); o drone da Amazon que realiza entrega de

produtos (Fig. 7(b)); o robô móvel Pioneer 3-DX utilizado para pesquisa e estudos sobre a robótica móvel (Fig. 7(c)); e o Ropits desenvolvido pela empresa japonesa Hitachi, para o transporte de passageiros (Fig. 7(d));

Figura 7 – Exemplos de robôs para transporte e mobilidade.



(a) Rovers da NASA

(b) Drone da Amazon

(c) Pionner 3-DX

(d) Ropits

Fonte: (MORAES, 2016; AMAZON, 2016; TECHNOLOGY, 2016; GUARDIAN, 2016)

Segundo Wolf et al. (2009) a aplicação da robótica em vários e heterogêneos segmentos da sociedade é mais uma evidência da importância do estudo na área e de seu futuro promissor. Gates (2007) esclarece que os maiores desafios da robótica atualmente são quesitos que dizem respeito à semânticas dos dados obtidos do ambiente, a acurácia do hardware e eficiência do software, como: interpretar palavras e expressões reconhecidas em um determinado contexto; o alto custo dos componentes de hardware como sensores e atuadores; e processar todas as informações obtidas pelo robô. O autor, contudo, salienta que apesar dessas dificuldades, os pesquisadores estão progredindo com a pesquisa e encontrando soluções inovadoras para cada problema. Outro ponto que colabora com o desenvolvimento da robótica é o próprio avanço da computação em geral, por exemplo, alguns algoritmos e análises de dados que despendiam muito poder de processamento foram otimizados ao passo que o próprio poder de computação e armazenamento dos hardware*s* aumentaram, além da queda de preço de muitos componentes (GATES, 2007).

2.1 Componentes Robóticos

De acordo com Dudek e Jenkin (2010) um robô pode ser compreendido sob diferentes perspectivas. São elas: perspectiva de *hardware*, que possui enfoque na parte física; perspectiva de interface *hardware*/*software*, que consiste na abstração dos componentes de hardware para

uso em software; e, perspectiva computacional, que consiste em toda sua arquitetura lógica de controle, como por exemplo: controle de movimento; planejamento de missões e objetivos; controle sensorial, interpretação e inferências sobre os dados obtidos.

A compreensão minuciosa dessas perspectivas permitem que os pesquisadores obtenham sucesso no desenvolvimento de um sistema robótico completo, analisando cada área conforme sua especificidade. Murphy (2000) enfatiza que criar um sistema robótico inteligente e autônomo não é uma tarefa trivial, uma vez que em um sistema autônomo, o robô deve ser capaz de agir em ambientes e lidar com situações inesperadas, como obstáculos que aparecem em seu percurso, e ainda assim continuar a operar até ter atingido sua meta. Para o autor existem três paradigmas que norteiam o controle desses, a deliberativa, reativa e a híbrida (deliberativa/reativa). Para entendê-las é necessário antes conhecer os componentes de um sistema robótico. Conforme Romero et al. (2014) os robôs são constituídos de diversos componentes, mas, destacam-se os sensores, atuadores e o sistema de controle ou controlador, que é responsável por integrar e gerir a operação desses componentes.

2.1.1 Sensores

A principal diferença dos robôs em relação a outras máquinas é que estes são capazes de interagir com o mundo, de se locomoverem, mudar seu ambiente, ao contrário, por exemplo, dos computadores, que não possuem essas capacidades (MURPHY, 2000; TÖLGYESSY; HUBINSKỲ, 2011). De acordo com Romero et al. (2014) tais habilidades são possíveis porque eles conseguem extrair informações do ambiente em que estão atuando e de si próprios, e isto é possível devido ao uso de sensores (KANNIAH; ERCAN; CALDERON, 2013).

Um sensor é um dispositivo eletromecânico que obtém informações sobre propriedades físicas do ambiente e as convertem em sinais eletromagnéticos. Assim são capazes de converter informações tais como, intensidade luminosa, temperatura, umidade, intensidade da corrente elétrica, distância, amplitude sonora, reflectância de materiais, e assim por diante, em sinal analógico ou digital, que por sua vez é interpretado como um dado.

Segundo Solutions (2007) existem basicamente duas formas de classificar os sensores: proprioceptivos, que obtém informações internas do robô; sensores exteroceptivos, que obtém informações externas ao robô; sensores ativos, que emitem energia no ambiente para então extrair as informações; e, sensores passivos, que obtém informações a partir da energia externa que os infligem.

2.1.2 Atuadores

Além de perceberem o ambiente em que estão atuando, os robôs também devem ser capazes de interagir com eles, seja se deslocando, mudando a posição de objetos, ou de alguma outra forma. Essa interação consciente é outra característica importantíssima para distingui-los

de outras máquinas (TÖLGYESSY; HUBINSKỲ, 2011). De acordo com Romero et al. (2014) os componentes robóticos responsáveis por esse tipo de interação são os atuadores. Tecnicamente os atuadores são dispositivos que convertem energia elétrica, hidráulica ou pneumática em energia mecânica, ou seja, em ações (ROMANO; DUTRA, 2002). Solutions (2007) classifica os atuadores como: pneumáticos, hidráulicos e elétricos.

2.1.3 Controladores

Obter informações sobre o ambiente e sobre si próprio é muito importante para a autonomia de um robô (MURPHY, 2000), todavia, para que tais dados sejam transformados em ações concretas, eles precisarão ser analisados sob diferentes perspectivas e objetivos, reorganizados para gerar novas informações a partir de inferências, e então definir e planejar ações.

O componente robótico responsável por todo esse processo é o controlador, que planeja as ações do robô a partir dos dados obtidos pelos sensores, além de coordenar as ações dos atuadores para que ao final o robô realize sua tarefa ou progrida em direção à realização, ou seja, em essência, um controlador é um computador que é capaz tratar e analisar dados, tomar decisões e coordenar suas execuções (NIKU, 2013).

2.2 Arquiteturas de Controle

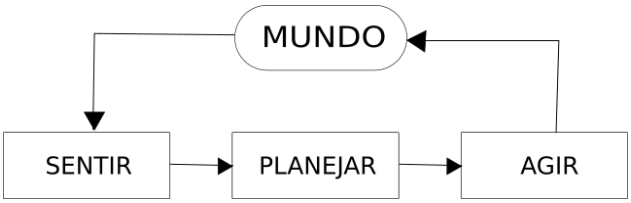
Segundo Albus (1990) uma arquitetura de controle de um robô é a composição de todos os seus componentes básicos (sensores, atuadores e controladores), ou seja, fornecem uma visão abstrata de como tais componentes estão interconectados e como ocorrem suas interações, assim é possível definir qual a capacidade do robô quanto a interação com o seu ambiente, planejamento de tarefas e reação à eventos (ALAMI et al., 1998).

Para Murphy (2000) as arquiteturas norteiam-se basicamente em três primitivas: sentir, planejar e agir. Cada uma dessas primitivas estão diretamente ligada aos componentes robóticos, uma vez que o robô sente por meio de seus sensores, planeja a partir dos algoritmos implementado em seu controlador e agi por intermédio dos atuadores. Assim, as arquiteturas de controle de robôs podem ser classificadas de acordo como o nível de prioridade e detalhe dado a cada uma dessas primitivas. Romero et al. (2014) e Murphy (2000) classificam-as como:

2.2.1 Arquiteturas Deliberativas

As arquiteturas deliberativas são modelos cuja principal abstração é o planejamento. Nessa arquitetura o robô percebe o ambiente por meio de seus sensores; planeja suas ações com base nas informações obtidas; e por fim realiza as ações planejadas (SIMPSON; JACOBSEN; JADUD, 2006). A Figura 8 ilustra o diagrama deste ciclo.

Figura 8 – Diagrama de uma arquitetura deliberativa.



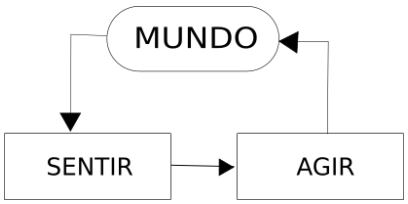
Fonte: o autor. Adaptado de (SIMPSON; JACOBSEN; JADUD, 2006)

2.2.2 Arquiteturas Reativas

As arquiteturas reativas são inspiradas no comportamento biológico de animais. Segundo Murphy (2000) o comportamento animal pode ser traduzido em percepção e ação, ou seja a entrada sensorial é automaticamente mapeada em uma ação, dispensando a etapa de cognição. Simpson, Jacobsen e Jadud (2006) esclarecem que essa abordagem descentraliza o sistema de controle dos robôs, mapeando cada entrada em uma saída, e, consequentemente, tornando mais rápidas suas reações à eventos.

Conforme Romero et al. (2014) as ações dos robôs devem ser previamente definidas, restrição que tornam os modelos dessa arquitetura restritivos, todavia, por conta de sua velocidade de resposta, elas são bastante indicadas à ambientes muito dinâmicos, cujas variáveis podem mudar a qualquer momento, como, um obstáculo aparecendo no percurso do robô enquanto este está em movimento acelerado. A Figura 9 mostra como as primitivas robóticas interagem neste modelo de arquitetura.

Figura 9 – Diagrama de uma arquitetura reativa.



Fonte: o autor. Adaptado de (MURPHY, 2000)

2.2.3 Arquiteturas Híbridas

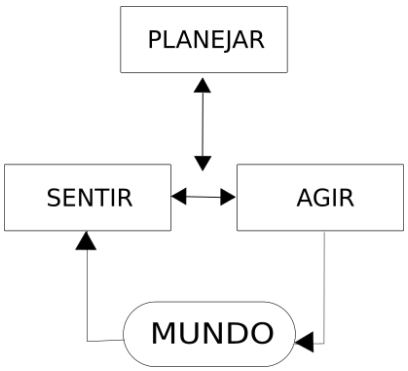
Segundo Murphy (2000) as arquiteturas reativas permitem aos robôs operarem em ambientes dinâmicos e reagirem com muita rapidez à eventos inesperados, pois a abordagem elimina a fase de planejamento, reagindo a cada entrada sensorial. O autor, contudo, salienta que essa característica também traz limitações à essas arquiteturas, principalmente quando as tarefas do robôs exigem maiores complexidades, como: conhecimento do estado global do robô, ou seja, o andamento da tarefa incumbido a ele; planejamento de trajetórias; avaliação de suas

ações, por exemplo, a escolha da melhor rota entre dois pontos; e, implementação de algoritmos de inteligência para aprendizado, etc.

Para esses casos que exigem um planejamento mais aprimorado, as arquiteturas deliberativas são as mais indicadas, contudo, elas também possuem desvantagens. Como essas arquiteturas são ruins ao lidarem com ambientes muito dinâmicos, pois demanda muito tempo na fase de planejamento.

Por conta disso, de acordo com Romero et al. (2014), nos casos mais complexos, que por ventura são mais comuns, é necessário uma arquitetura de controle que una a reatividade das arquiteturas reativas e a deliberatividade das arquiteturas deliberativas. Murphy (2000) esclarece que as arquiteturas híbridas são capazes que manter as qualidades de ambas as arquiteturas, uma vez que prega a modularidade das funções do robô. O *software* controlador do robô pode ser dividido em um planejador, responsável por planejar e calcular as novas metas, e um executor reativo, que o permitirá reagir a mudanças dinâmicas no ambiente. Assim, considerando as primitivas de um sistema robótico, uma arquitetura híbrida consiste basicamente em planejar, depois sentir e agir. A Figura 10 ilustra esta interação entre as primitivas.

Figura 10 – Diagrama de uma arquitetura híbrida.



Fonte: o autor. Adaptado de (MURPHY, 2000)

2.3 Classificação dos Robôs

Segundo Jung et al. (2005) e Kelly (2013) existem vária maneiras de classificar os robôs. Basicamente a classificação é feita seguindo alguns critérios como, entre outros, a autonomia do robô, sua mobilidade, sua funcionalidade, sua estrutura mecânica e seu campo de atuação.

Um forma de classificação, entre as mais importantes, é a classificação com base na mobilidade do robô, ou seja, os de base fixa e os móveis. Essa diferenciação é importante pois divide os estudos dos robôs em dois grandes campos, que possui aplicação distintas e importantes na sociedade, cada qual com seu nível de complexidade.

2.3.1 Robôs de Base Fixa

Os robôs de base fixas, também conhecidos como robôs industriais ou manipuladores por conta de sua grande aplicação nas indústrias, são dispositivos que atuam em ambiente controlados ou semi controlados. Esses são projetados para serem precisos e velozes em suas tarefas. Contudo, o fato de serem fixos limitam sua aplicabilidade e extensão (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

Segundo NIKU (2013), os robôs manipuladores possuem muitas articulações que lhes permitem mover dentro de seu espaço de trabalho, ou seja, dentro do espaço delimitado por sua geometria. A Figura 11 mostra robô TX40 que possui 6 graus de liberdade, produzido pela empresa suíça Stäubli.

Figura 11 – Robô manipulador TX40

Fonte: (STäUBLI, 2016)

2.3.2 Robôs Móveis

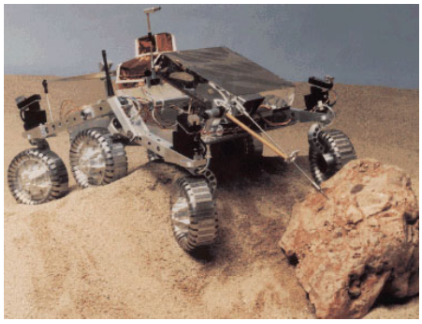
De acordo com Siegwart, Nourbakhsh e Scaramuzza (2011) os robôs de base fixas são bastante velozes e precisos no cumprimento de suas tarefas, contudo, faltam-lhes uma característica fundamental que aumentaria significativamente sua escala de aplicação, a mobilidade. Siegwart, Nourbakhsh e Scaramuzza (2011) e Kanniah, Ercan e Calderon (2013) afirmam que esta característica distingue o estudo sobre os robôs móveis das demais áreas da robótica.

A grande questão que norteia a mobilidade é o fato de que agora os robôs devem ser capazes de lidarem com ambientes diferentes nas mais diversas condições. Condições estas, muitas vezes imprevistas, assim, a quantidade de variáveis que deveriam ser controladas cresce de maneira significativa, obrigando-lhes a serem capazes de continuarem a atuar ainda que em ambientes imprevisíveis e dinâmicos (DUDEK; JENKIN, 2010). Essa nova problemática não diz respeito a apenas o *software* do robô, mas também ao seu *hardware*, haja vista que além de construir um modelo de mundo confiável com base nas informações de seus sensores, eles também devem se locomoverem em ambientes sinuosos e irregulares, além de interagirem com diversas formas e objetos que lhes forem apresentados.

Segundo Jones e Flynn (1993) existem várias mecanismos para locomoção dos robôs, os mais comuns são: pernas, a exemplo dos humanoides; esteiras; e rodas. Dentre todos, os robôs móveis com rodas são os mais comuns e fáceis de controlar. De acordo com Kelly (2013) e Murphy (2000) a classificação destes com base em seu grau de autonomia é:

- **Robôs teleoperados:** Segundo (MURPHY, 2000) os robôs teleoperados necessitam de um humano para controlá-los a distância. O operador deve possuir informações suficientes sobre as condições atuais, para então tomar decisões, ou seja, o sistema como um todo deve retornar por algum canal de comunicação, como a rádio frequência, informações sobre seus sensores, inclusive imagens de câmeras quando for mais conveniente. Um exemplo destes é o robô Sojourner (Fig 12) que explorou Marte em 1997 até parar de responder a comandos de rádio em 27 de setembro do mesmo ano.

Figura 12 – Rover Sojourner.



Fonte: (EDGERTON, 2016)

- **Robôs teleguiados:** Segundo Borenstein et al. (1997) os robôs teleguiados, ou AGVs[1] usam guias em sua orientação no ambiente, ou seja, são capazes de se locomoverem seguindo marcações pré-definidas no ambiente, como linhas pintadas no chão, fitas magnéticas, etc. Essas técnicas de locomoção os impedem de mudar seus percursos conforme estímulos externos em seus sensores. O seu uso é bastante comum em indústrias e, inclusive, em restaurantes, uma vez que para aplicações cuja tarefa e percurso são repetitivos esses robôs são bastante eficazes. A Figura 13 mostra um robô garçonete que "trabalha" em um restaurante na cidade de Harbin na China. Esse restaurante conta ainda com mais 18 tipos de robôs diferentes que recepcionam, cozinham e servem os clientes, todos desenvolvido pela empresa Harbin Haohai Robot (WRENN, 2016).

- **Robôs Autônomos:** os robôs autônomos são capazes de se locomoverem e de operarem sem nenhuma intervenção humana (WOLF et al., 2009). Para tanto, possuem *"habilidades"* mais sofisticadas, tais como: capacidade de percepção, propiciada por seus sensores; capacidade de agir, proporcionada por seus atuadores; robustez e inteligência, possibilitados

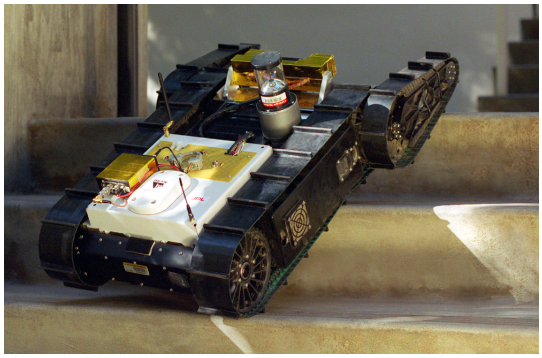
[1] do inglês Automated Guided Vehicles

Figura 13 – Robô Garçonete.

Fonte: (WRENN, 2016)

por algoritmos de inteligência artificial. Segundo Murphy (2000) a inteligência artificial é o principal componente que tornam inteligentes os robôs, e, por sua vez, autônomos. A Figura 14 mostra o *Urbie*, desenvolvido em um projeto conjunto entre várias instituições inclusive a empresa iRobot e o Instituto de Robótica da *Carnegie Mellon University*, com o propósito de ajudar no resgate de vítimas de tragédias urbanas.

Figura 14 – Robô autônomo Urbie.



Fonte: (NASA, 2016b)

De acordo com Odedra, Prior e Karamanoglu (2009) os robôs autônomos ainda podem ser separados de acordo com o ambiente em que eles operam:robôs autônomos aéreos, do inglês *Unmanned Aerial Vehicle* - UAV, que operam no espaço aéreo; robôs autônomos aquáticos, do inglês *Unmanned Underwater Vehicle* - UUV, que operam no espaço subaquático; e, robôs autônomos terrestres, do inglês *Unmanned Ground Vehicle* - UGV, que operam no espaço terrestre;

Ainda segundo os autores cada ambiente oferece um nível diferente de complexidade para a mobilidade do robô. No caso dos UAVs o maior desafio é a resistência do vento, uma vez que o espaço aéreo possui poucos obstáculos. No caso dos UUVs o maior desafio é a integridade física dos componentes, já que esta classe opera sob a água que pode danificar seus componentes eletrônicos. Por fim, os UGVs são os robôs que lidam com os maiores desafios, pois o ambiente

terrestre possui diferentes tipos de terreno e uma quantidade muito maior de obstáculos que os demais.

Para Wolf et al. (2009) por conta da versatilidade dos robôs móveis autônomos e seus complexos processos de desenvolvimentos, a área pode ser dividida em algumas subáreas de pesquisa, como: fusão de sensores; desvio de obstáculos; auto-localização; mapeamento do ambiente; planejamento de trajetórias; planejamento de ações; navegação autônoma; interação e comunicação. Segundo Siegwart, Nourbakhsh e Scaramuzza (2011) um bom mecanismo de percepção de mundo e de locomoção são capazes de fazer os robôs móveis atuarem em qualquer ambiente. Ainda de acordo com os autores, a localização é peça fundamental para as demais técnicas dos robôs móveis.

2.4 Localização

Segundo Jensfelt (2001), a localização é um componente essencial para os robôs móveis autônomos. O'Kane (2006) e Gonzalez et al. (2012) enfatizam que quase todos os algoritmos que dizem respeito ao planejamento de trajetórias, tarefas, e execução de rotas exigem que o robô saiba sua posição atual e orientação no ambiente.

O problema da localização é resumidamente simples. Por exemplo, imagine uma pessoa entrando em uma sala que nunca estivera; logo ao entrar, instantaneamente ela começa a reconhecer o ambiente e criar um mapa interno, destacando sua posição atual, obstáculos presentes, e um alvo em particular. Análogo aos humanos, os robôs ao entrarem em um ambiente desconhecido iniciam o mesmo processo, construção do mapa e reconhecimento do ambiente. Para os humanos navegar, ou seja, se deslocar neste novo local não é uma tarefa complicada, pois instintivamente conhece-se tudo o que diz respeito aos seus movimentos, assim, ao dar um passo saberá exatamente qual será sua nova posição e orientação, entretanto, para um robô essa tarefa pode ser mais complicada do que se imagina.

Jensfelt (2001) destaca que o problema da localização para robôs não é nada trivial, pois conta com vários fatores desfavoráveis como a limitação dos sensores, poder de processamento dos controladores, e ruídos provenientes tanto dos sensores quanto dos atuadores. Para o autor, a localização em robótica móvel consiste basicamente em lidar com a incerteza. Ele explica que esta técnica pode ser dividida em três partes: rastreamento de posição, onde a posição inicial do robô é conhecida, cabendo ao programa apenas acompanhar o deslocamento deste, atualizando sua posição; localização global, quando a priori a posição inicial do dispositivo é desconhecida, requerendo desta classe sua estimação inicial; e, construção de mapas, usados pelos algoritmos de planejamento de trajetória, pois representa o conhecimento atual do robô quanto ao ambiente que está operando (ROMERO et al., 2014).

Existem diversos algoritmos que objetivam solucionar o problema de localização dos robôs móveis (BORENSTEIN et al., 1996). Esses algoritmos podem ser categorizados em dois

campos:

- **Localização relativa ou local:** são técnicas que estimam a posição do robô com base em algum ponto referencial. As duas técnicas mais conhecidas são a odometria e a navegação inercial (BORENSTEIN et al., 1996);
- **Localização absoluta ou global:** são técnicas que estimam a localização do robô com base em um estado global. Segundo Gonzalez et al. (2012) a técnica mais famosa para essa classe é a utilização do sistema de posicionamento global - GPS, existem também os algoritmos baseados em marcos naturais ou artificiais (BORENSTEIN et al., 1996).

Para Jensfelt (2001) o projeto de um sistema de localização deve, fundamentalmente, levar em consideração as incertezas provenientes do conhecimento de mundo e da posição do robô. Ou seja, o quão confiável são as inferências realizadas com base nas informações sensoriais. Siegwart, Nourbakhsh e Scaramuzza (2011) enfatizam também que a localização abrange muito mais que conhecer sua posição absoluta ou relativa separadamente, pois para compor um sistema robusto ambas as técnicas são essenciais, uma vez que a local fornece informações mais precisas em relação ao espaço atual do robô, e, por sua vez, a global provê informações com uma dimensão muito maior, permitindo ao robô navegar entre cidades, por exemplo.

Os autores salientam também que a construção de um mapa do ambiente é fundamental para algoritmos de planejamento de trajetória, que objetivam encontrar um caminho livre de obstáculos entre a posição atual do robô e uma posição final desejada. Com base nisso, Jensfelt (2001) esclarece que o desafio da localização concentra-se em associar as informações sensoriais ao modelo de mundo criado, uma vez resolvido, o problema se resumiria em uma de estimação de padrão, haja vista que tais informações seriam apenas conferidas com modelo de mundo, fornecendo dados precisos sobre a localização do robô.

Entretanto, ainda que diante a tantas variáveis negativas acerca da localização de robôs móveis, diversas técnicas são utilizadas para estimar sua posição, todas considerando o problema de sua incerteza. Segundo Jensfelt (2001) todas as técnicas são divididas em duas etapas, a de previsão da posição atual e sua atualização. De acordo com Romero et al. (2014) o sistema de localização de um robô móvel começa com a estimativa inicial de sua posição, que segundo Borenstein e Feng (1995a) e Siegwart, Nourbakhsh e Scaramuzza (2011) podem, por exemplo, ser obtida pelo GPS ou por técnicas triangulação e marcos naturais ou artificiais; depois o robô planeja seu próximo movimento, baseado em um mapa ou outra representação do ambiente; e utilizando técnicas incrementais como a odometria, conhecidas por *dead-reckoning*, para estimar a posição atual.

Por fim outras técnicas são utilizadas para aumentar a crença sobre a posição atual do robô no mapa, como os métodos probabilísticos de localização de Markov, que mantém um

conjunto de posições possíveis para a localização do robô (FOX; BURGARD; THRUN, 1999), ou a localização de Monte Carlo, que, segundo Romero et al. (2014) e Fox, Burgard e Thrun (1999), utiliza amostragens para representar a crença do robô sobre sua posição. Essa técnica também é conhecida como filtro de partículas.

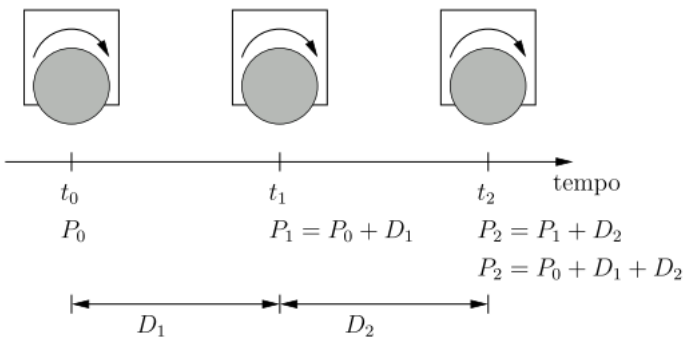
Segundo Borenstein e Feng (1995a) a técnica de odometria, classificada em *dead-reckoning*, é uma boa abordagem para a localização relativa do robô, pois é simples de ser implementada, barata e possui uma frequência de atualização bastante alta, ao contrário de outras técnicas que exigem maior taxa de processamento para estimarem a posição. Todavia, conta como desvantagem o erro acumulado sobre o deslocamento do robô, notável quando este percorre longas distâncias ou realiza muitas curvas (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011). Assim sendo, várias abordagens foram desenvolvidas visando amenizar essa incerteza em métodos de odometria.

2.4.1 Odometria

Segundo Gonzalez et al. (2012) uma das técnicas mais famosas e utilizadas para localização local dos robôs móveis é a odometria. Essa popularidade é justificável por conta da simplicidade e eficácia do método.

A odometria é uma técnica baseada em *dead-reckoning*, que por sua vez classifica um conjunto de métodos em que a posição atual é calculada com base em uma anterior. Basicamente os modelos odométricos calculam incrementalmente o deslocamento do robô a partir de sua posição inicial (BEZERRA, 2004). A Figura 15 ilustra um diagrama criado por Bezerra (2004) para exemplificar de forma sucinta a ideia por trás deste modelo. Nela é possível notar que ao longo do tempo a posição do robô vai sendo mensurada com base em seu deslocamento. Este é medido por sensores tais como, encoders (monofásicos ou quadráticos), tacômetro, giroscópio, etc., nomeados por Olson (2004) como sensores *dead-reckoning*, todavia, alguns autores utilizam informações visuais como em Nourani-Vatani, Roberts e Srinivasan (2009) e Nistér, Naroditsky e Bergen (2006), esta abordagem é conhecida como odometria visual.

Figura 15 – Deslocamento de um robô utilizando odometria



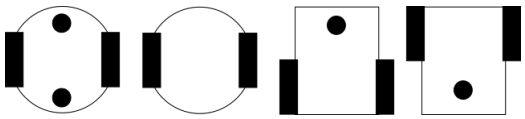
Fonte: (BEZERRA, 2004)

Os métodos classificados em *dead-reckoning* são bastante eficazes à medida que possui uma frequência de atualização alta, são baratos, precisos para pequenos deslocamentos, e simples de utilizar. Por conta disso, Jensfelt (2001) afirma que a odometria se tornou essencial para a maioria dos mecanismos de localização na robótica. Kanniah, Ercan e Calderon (2013) esclarecem que está técnica é um modelo matemático que estima a posição atual do robô integrando a velocidade de suas rodas.

Na prática, as leituras dos sensores responsáveis pelo monitoramento das rodas são feitas periodicamente. Isso reduz bastante a complexidade do modelo matemático, pois a integral passa a ser um simples somatório realizado em intervalos de tempo predefinidos (OLSON, 2004; KANNIAH; ERCAN; CALDERON, 2013).

Os modelos odométricos utilizam dos parâmetros cinemáticos do robô e sua velocidade linear e angular de cada roda para, então, mensurar sua posição e orientação (BEZERRA, 2004; GONZALEZ et al., 2012). Conforme Jensfelt (2001) existem diferentes tipos de designs de cinemática para robôs móveis, e estes influenciam no desempenho e precisão da estimação da posição por meio da odometria. A topologia diferencial é a mais comum delas. Esta é simples de implementar fisicamente e também de controlar. A Figura 16 exemplifica diversas configurações dessa topologia.

Figura 16 – Modelos da topologia diferencial de duas rodas tracionadas.



Fonte: o autor.

2.4.1.1 Modelo Odométrico para um robô com topologia diferencial

Para compreender o modelo odométrico, é importante, a princípio, entender a cinemática, que, segundo Siegwart, Nourbakhsh e Scaramuzza (2011), é o estudo mais básico e fundamental de como o sistema mecânico, ou seja, o corpo do robô, se comportar no ambiente, no tocante aos movimentos realizados por este. O autor ainda enfatiza que esse estudo permite a criação de um bom software controlador, pois esquematiza todas as contribuições e restrições advindas do chassi e rodas. Assim, é possível ter uma noção bem clara de como e quais os movimentos são realizados.

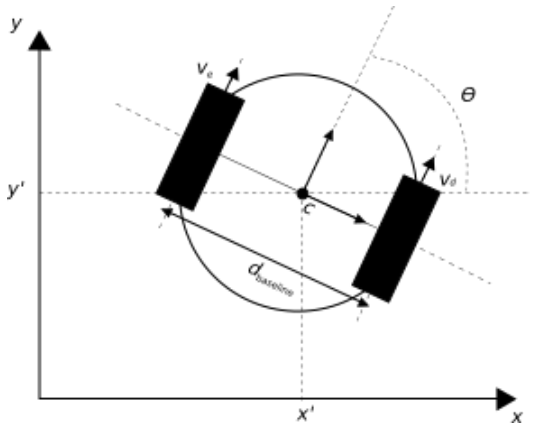
A cinemática considera o robô como um corpo rígido, ou seja, um conjunto finito de pontos, tal que a distância entre dois destes é constante ao longo do tempo (ROMERO et al., 2014). Esse está associado à rodas, e opera em um plano horizontal. Assim, sua posição no plano pode ser expressa por uma tripla, cujas duas primeiras dimensões definem a localização cartesiana do corpo, e o terceiro componente, sua orientação (ROMERO et al., 2014; SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).Desta forma, a posição P do robô é definida como:

Definição 1 (Posição do Robô) *A posição P de um robô que possui liberdade de movimentação restrita a um plano horizontal é definida pela tripla* (x, y, θ)*, cujo par ordenado* (x, y) *definem sua posição e* θ*, sua orientação em relação ao eixo vertical e ortogonal ao plano. Assim:*

$$P = [x, y, \theta]^T$$

A Figura 17 ilustra o modelo cinemático de um robô com topologia diferencial. Nela é possível notar alguns parâmetros relevantes para a definição do modelo odométrico e compreensão da dinâmica dos movimentos do robô. Por exemplo, nesta topologia cada roda possui uma velocidade associada (v_e, para a roda da esquerda; e, v_d para a roda da direita), que não precisam ser iguais, na prática isso quase nunca ocorre. Logo, é compreensível que ao alterar uma velocidade o robô fará uma curva. Por exemplo, considere v_d maior que v_e, isso resultará em uma desvio para a esquerda.

Figura 17 – Cinemática de um robô diferencial com duas rodas.



Fonte: o autor.

A Tabela 2.4.1.1 mostra as principais equações odométricas para o modelo diferencial, a dedução completa das fórmulas pode ser encontrada no Apêndice A.

Equação	Aplicação
$d = \frac{d_{left}+d_{right}}{2}$	calcula o deslocamento do robô
$\phi = \frac{d_{right}-d_{left}}{d_{baseline}}$	calcula a orientação momentânea do robô
$\theta' = \theta + \phi$	calcula a orientação do robô
$x' = x + d\cos(\theta)$	calcula a posição x do robô
$y' = y + d\sin(\theta)$	calcula a posição y do robô

$$d_{left} = \frac{P_{left} N_{left}}{Re_{left}} \tag{2.1}$$

$$d_{right} = \frac{P_{right} N_{right}}{Re_{right}} \tag{2.2}$$

2.4.1.2 Modelo Odométrico para um robô com direção *Ackermann*

A topologia diferencial é fácil de ser implementada e controlada, todavia, segundo Muniandy e Muthusamy (2012), neste modelo, fazer o robô seguir uma linha reta é uma tarefa difícil, mesmo aplicando o mesmo controle e tensão nos motores. Além disso, como as rodas responsáveis pela tração e direção estão desassociadas, o robô naturalmente tenderá para uma direção, possuindo um desnível lateral que acentua-se proporcional à distância percorrida.

Outro modelo mecânico para robôs móveis é a topologia *Ackermann*, baseado na geometria de mesmo nome. Este sistema foi criado por volta de 1800 d.C. com o intuito de corrigir o desnível presente nos diferenciais durante a realização de uma curva (GUPTA; DIVEKAR; AGRAWAL, 2010; HRBÁČEK; RIPEL; KREJSA, 2010). Conforme Borenstein et al. (1996) a geometria garante que a roda dianteira do veículo, interna a curva realizada, esteja em um ângulo maior que a roda externa, eliminando o desvio do pneu. A Figura 18(a) mostra a cinemática deste modelo mecânico e a Figura 19 apresenta um esquemático do sistema aplicado a um veículo realizando uma curva. O ângulo específico para cada roda pode ser calculado usando as seguintes fórmulas, sendo θ_{left} e θ_{right} para as rodas esquerda e direita, respectivamente.

$$\theta_{left} = \arctan2\left(W_{base}, r_{left}\right) \tag{2.3}$$

$$\theta_{right} = \arctan2\left(W_{base}, r_{right}\right) \tag{2.4}$$

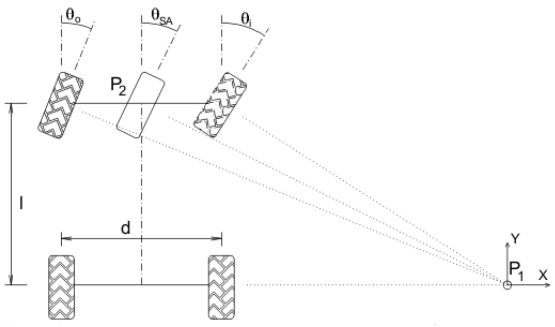
Por questão de conveniência e com o intuito de simplificar a modelagem, o modelo *Ackermann* (Fig. 18(a)) é reduzido a uma cinemática baseado no modelo de triciclo (Fig. 18(b)), que segundo Borenstein et al. (1996), para a odometria são equivalentes. Desta forma, as rodas dianteiras do veículo são substituídas por uma roda virtual, situada no ponto médio do eixo que as separam. Além disso, a mesma configuração usada no modelo diferencial é usada para estimar a posição do eixo traseiro.

Analisando a Figura 19, é possível notar que há um ponto comum (C) entre os eixos traçados pelas rodas traseiras e dianteiras. Este ponto é conhecido como *Centro de Rotação* (CR) e coincide com o centro das circunferências. Além disso, é notável também que o raio da circunferência da roda frontal interna (RFI) é menor que o da roda frontal externa (RFE), que por sua vez, é maior que a da roda frontal virtual (RFV). A diferença existente na circunferência realizada por RFV e a roda virtual traseira (RVT) é determinado pela distância L entre os os eixos do veículo. Assim, a odometria no modelo *Ackermann* objetiva determinar a orientação P do robô, conhecendo os valores de $d_{baseline}$ e L[2].

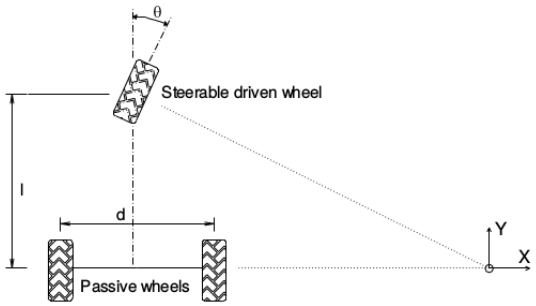
A Figura 20 apresenta o geometria para a odometria de robôs que usam a cinemática de triciclos. Nessa representação duas outras variáveis aparecem, a velocidade linear (v) e a veloci-

[2] Por convenção, a partir daqui, L será nomeado como W_{base}.

Figura 18 – Cinemática do modelo *Ackermann* e Triciclo.



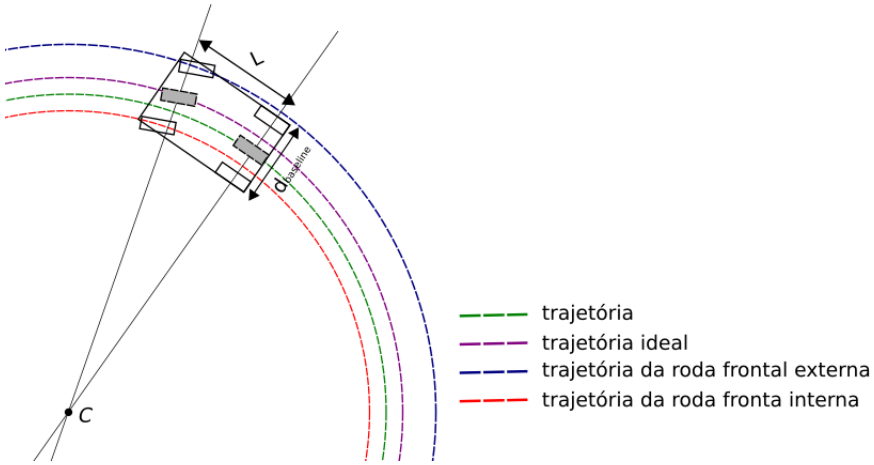
(a) Cinemática Ackermann



(b) Cinemática Triciclo

Fonte: (BORENSTEIN et al., 1996)

Figura 19 – Trajeto de um veículo usando a topologia *Ackermann*



Fonte: o autor

dade angular (ω). A relação em entre tais variáveis é como segue (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011):

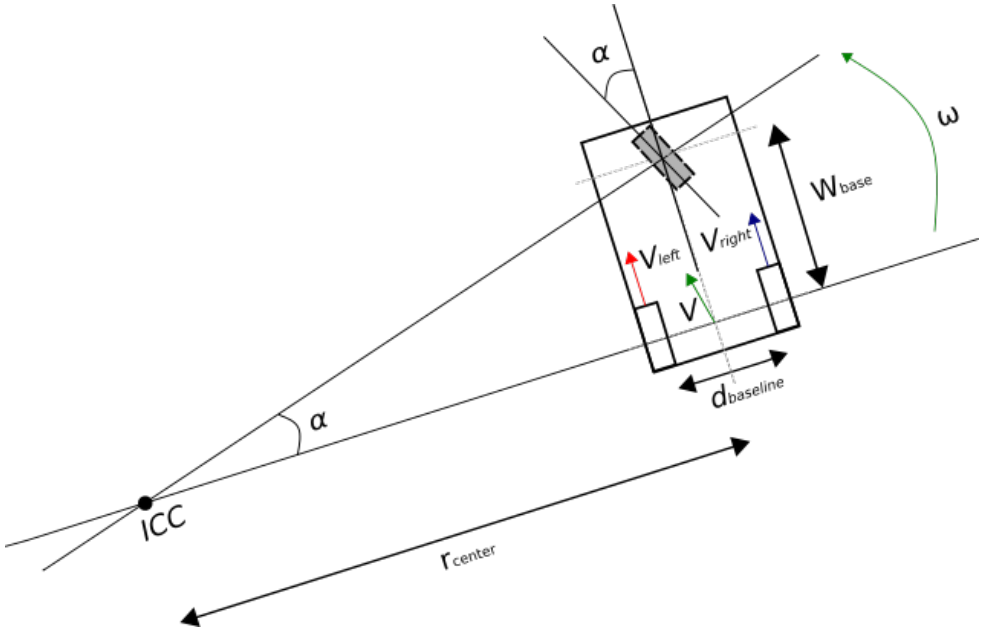
Definição 2 (Velocidade Angular e Linear) *A velocidade linear descreve a rapidez que uma partícula (corpo rígido) percorre uniformemente uma trajetória, enquanto a velocidade angular é sua rapidez percorrendo um segmento de circunferência de ângulo* δ *e raio R. A relação entre*

essas duas propriedades é entendida como:

$$v = \omega R$$

Onde v é a velocidade linear, ω *a velocidade angular e R o raio da circunferência.*

Figura 20 – Modelo geométrico da odometria de um triciclo.



Fonte: o autor

O robô percorre uma trajetória curva, cujo centro de circunferência é definido pelo ponto *ICC* (Centro de Curvatura Instantâneo - *Instantaneous Center of Curvature*). O raio desse trajeto é determinando analisando o triângulo formado pelos eixos traçados de cada roda, assim:

$$\tan\alpha = \frac{W_{base}}{r_{center}}$$

$$r_{center} = \frac{W_{base}}{\tan\alpha} \tag{2.5}$$

Calculado o raio da circunferência, então, é possível definir a velocidade angular do veículo, desta forma, usando 2.3, ω é entendida como:

$$v = \omega r_{center}$$

$$\omega = \frac{v}{r_{center}}$$

$$\omega = \frac{v}{\left(\frac{W_{base}}{\tan\alpha}\right)}$$

$$\omega = \frac{v\tan\alpha}{W_{base}} \tag{2.6}$$

Considerando apenas a posição cartesiana do robô, definido por $P^{'} = \begin{bmatrix} x & y & 0 \end{bmatrix}^T$, é possível calcular suas componentes, multiplicando a matriz de rotação M_{rot} e uma matriz para velocidade do movimento longitudinal $V = \begin{bmatrix} v & 0 & 0 \end{bmatrix}^T$, desconsiderando desníveis laterais (RILL, 2007). Assim:

$$P = V^T M_{rot}, \qquad sendo \quad M_{rot} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = \begin{bmatrix} v & 0 & 0 \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ 0 \end{bmatrix}$$

$$x = v\cos(\theta) \tag{2.7}$$

$$y = v\sin(\theta) \tag{2.8}$$

Como a posição do robô muda conforme seu deslocamento em função do tempo, então, esta passa a ser definida pela integral das equações 2.5 e 2.6, definidas pelo intervalo $[0,t]$. Dessa forma 2.5, 2.6 e a orientação θ do robô assumem os valores:

$$x(t) = \int_0^t v(t)\cos\big(\theta(t)\big)dt \tag{2.9}$$
$$y(t) = \int_0^t v(t)\sin\big(\theta(t)\big)dt \tag{2.10}$$
$$\theta(t) = \int_0^t \omega(t)dt \tag{2.11}$$

No entanto, conforme Olson (2004), Kanniah, Ercan e Calderon (2013), o intervalo de t é periódico, assim, as integrais em 2.7, 2.8 e 2.9 podem ser resolvidas como:

$$x(t) = x(t-1) + v(t)\cos\big(\theta(t)\big) \tag{2.12}$$
$$y(t) = y(t-1) + v(t)\sin\big(\theta(t)\big) \tag{2.13}$$
$$\theta(t) = \theta(t-1) + \omega(t) \tag{2.14}$$

Substituindo 2.3 em 2.12, tem-se:

$$\theta(t) = \theta(t-1) + \frac{v\tan\alpha}{W_{base}} \tag{2.15}$$

A partir das explanações, é possível, então, sumarizar as principais equações odométricas para o modelo *Ackermann*. A Tabela 1 apresenta-as.

Tabela 1 – Principais equações odométricas para o modelo Ackermann.

Nome	Equação
$\tan\alpha$	$\tan\alpha = \frac{W_{base}}{r_{center}}$
$x(t)$	$x(t) = x(t-1) + v\cos\big(\theta(t)\big)$
$y(t)$	$y(t) = y(t-1) + v\sin\big(\theta(t)\big)$
$\theta(t)$	$\theta(t) = \theta(t-1) + \frac{v\tan\alpha}{W_{base}}$

Fonte: O autor.

2.4.2 Erros da Odometria

Os métodos classificados em *dead-reckonign* são bastante eficazes à medida que possui uma frequência de atualização muito alta, assim, ao passo que o robô se desloca, sua posição é

atualizada diversas vezes, aumentando a precisão da estimativa. Como baseiam-se em sensores acoplados às rodas do veículo, e em funções integrais definidas por um intervalo de tempo, erros cumulativos podem ocorrer na estimação, aumentando ainda mais a imprecisão da posição final mensurada. Conforme Borenstein e Feng (1995a), Chong e Kleeman (1997) os erros odométricos podem ser categorizados em:

- **Erros sistemáticos:** Ocorrem por conta das propriedades cinemáticas do robô móvel. De acordo com Borenstein e Feng (1995a), esse tipo de erro pode ser reduzido com uma boa definição cinemática, e métodos de calibragem de seus parâmetros. Existem outras técnicas empregadas, o próprio autor propõe a utilização de um método de estimação de erro conhecida como *UMBmark* (*University of Michigan Benchmark test*).
- **Erros não sistemáticos:** Segundo Bezerra (2004) os erros não sistemáticos são aqueles que são causados por variáveis dinâmicas no ambiente, como por exemplo, terrenos irregulares, objetos inesperados no chão, pouco atrito e aderência entre as rodas e a superfície, entre outros. Borenstein e Feng (1995a) esclarece que tais erros são mais comuns para a odometria, apesar de que os erros sistemáticos possuem como desvantagem o fato de serem cumulativos ao longo do tempo, já os não sistemáticos são pontuais,assim, mais difíceis de prever e corrigir.

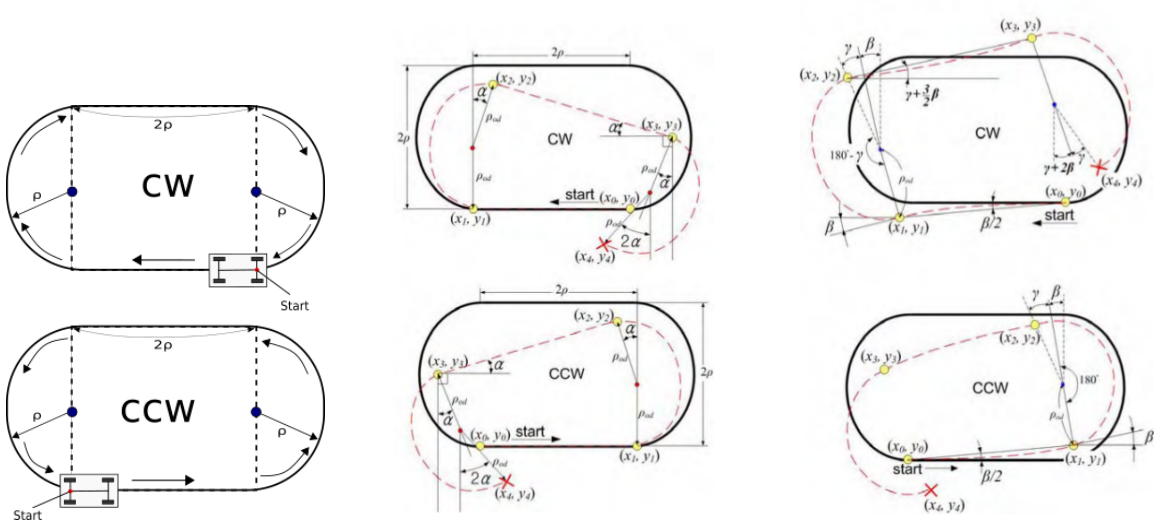
Apesar de possuírem tantos erros que tornam suas estimativas incertas, a odometria continua sendo uma das técnicas mais usadas para a localização relativa de robôs móveis (BORENSTEIN et al., 1996). Existem diversas técnicas para reduzir a interferência do ruído na estimativa da posição final do robô, entre elas, destaca-se a *UMBmark* definida para veículos com topologia diferencial (A demostração da técnica pode ser vista no Apêndice B), e sua adaptação para um veículo *Ackermann* proposta por Lee et al. (2007).

2.4.2.1 Correção de erro sistemático em robôs *Ackermann*

O teste *UMBmark* proposto por Borenstein e Feng (1995b) é prático e eficiente, contudo, não pode ser usado em outros modelos de robôs, como os *Ackermann*, por conta das características cinemáticas destes. Lee et al. (2007) esclarecem que estes veículos não podem realizar um movimento de rotação sem que haja uma translação em conjunto, e o *UMBmark* exige que o robô rotacione em 90° sobre seu próprio eixo (manobra facilmente implementada em diferenciais). Por conta disso, os autores propõem uma adaptação do percurso para a estimação e correção de erros sistemáticos em robôs que usam a cinemática *Ackermann*. A proposta dos autores é bastante semelhante ao feito na conceitualização do *UMBmark* por Borenstein e Feng (1995b), todavia, considera a nova geometria do percurso e as principais fontes de ruídos do modelo cinemático trabalhado. A Figura 21(a) apresenta o novo trajeto.

A cinemática de um modelo *Ackermann* possui parâmetros adicionais em relação ao diferencial. Assim, mais variáveis que contribuem para a imprecisão da estimação de posição do

Figura 21 – Percusso realizado no teste *UMBmark* para robôs Ackermann.



(a) Percurso (b) Tipo A (c) Tipo B

Fonte: o autor; e, Lee et al. (2007)

robô são adicionadas ao sistema. De acordo com Lee et al. (2007), os principais erros acometidos ao modelo são:

1. diferença de diâmetros das rodas;
2. tamanho do eixo traseiro ($d_{baseline}$);
3. desalinhamento das rodas;
4. o tamanho do eixo que separam as rodas frontais das traseira (W_{base}).

Entre estes, os que mais interferem na posição final são 1 e 2, assim como nos modelos diferenciais (LEE et al., 2007; BORENSTEIN; FENG, 1995b).

Seguindo os mesmos procedimentos que as demonstrações realizadas no caso de robôs diferenciais, Lee et al. (2007) obtiveram os valores finais da posição do robô após realizar o percurso no sentido horário e anti-horário, considerando os erros de Tipo A (Fig. 21(b)) e B (Fig. 21(b)), assim:

$$P_{A_{cw}}(x,y) = \left(\left[\frac{-2\pi}{\pi+\alpha} \right] \rho\alpha, -2\rho\alpha \right) \tag{2.16}$$

$$P_{A_{ccw}}(x,y) = \left(\left[\frac{2\pi}{\pi+\alpha} \right] \rho\alpha, -2\rho\alpha \right) \tag{2.17}$$

$$P_{B_{cw}}(x,y) = (2\rho\beta, 2\rho\beta) \tag{2.18}$$

$$P_{B_{ccw}}(x,y) = (2\rho\beta, -2\rho\beta) \tag{2.19}$$

Novamente, como os ambos os erros ocorrem simultaneamente, então somando 2.14 e 2.16, e 2.15 e 2.17, obtém-se o valor de P_{cw} e P_{ccw}.

$$P_{cw}(x,y) = \left(\left[\frac{-2\pi}{\pi+\alpha} \right] \rho\alpha + 2\rho\beta, -2\rho\alpha + 2\rho\beta \right) \tag{2.20}$$

$$P_{ccw}(x,y) = \left(\left[\frac{2\pi}{\pi+\alpha} \right] \rho\alpha + 2\rho\beta, -2\rho\alpha - 2\rho\beta \right) \tag{2.21}$$

Para encontrar o valor de α basta subtrair as componentes x de 2.18 e 2.19, por exemplo, assim:

$$\begin{aligned}
x_{cg_{cw}} - x_{cg_{ccw}} &= \left[\frac{-2\pi}{\pi+\alpha} \right] \rho\alpha + 2\rho\beta - \left[\frac{2\pi}{\pi+\alpha} \right] \rho\alpha - 2\rho\beta \\
x_{cg_{cw}} - x_{cg_{ccw}} &= \frac{-4\pi}{\pi+\alpha} \rho\alpha \\
(\pi+\alpha)(x_{cg_{cw}} - x_{cg_{ccw}}) &= -4\pi\rho\alpha \\
\pi(x_{cg_{cw}} - x_{cg_{ccw}}) &= -\alpha(4\pi\rho + (x_{cg_{cw}} - x_{cg_{ccw}})) \\
\alpha &= -\frac{\pi(x_{cg_{cw}} - x_{cg_{ccw}})}{(4\pi\rho + (x_{cg_{cw}} - x_{cg_{ccw}}))}
\end{aligned} \tag{2.22}$$

O valor de β, por sua vez, é definido pela soma de $x_{cg_{cw}}$ e $x_{cg_{ccw}}$, desta forma:

$$\begin{aligned}
x_{cg_{cw}} + x_{cg_{ccw}} &= \left[\frac{-2\pi}{\pi+\alpha} \right] \rho\alpha + 2\rho\beta + \left[\frac{2\pi}{\pi+\alpha} \right] \rho\alpha + 2\rho\beta \\
x_{cg_{cw}} + x_{cg_{ccw}} &= 4\rho\beta \\
\beta &= \frac{x_{cg_{cw}} + x_{cg_{ccw}}}{4\rho}
\end{aligned} \tag{2.23}$$

Com os valores de β e α é possível corrigir ambos os tipos de erro. O erro do Tipo A, é corrigido usando as equações B.27 e 2.20. Já para o Tipo B, usa-se , B.24, B.25, e 2.21 sendo

r_{center}, agora definido por:

$$r_{center} = \frac{\rho}{\sin\left(\frac{\beta}{2}\right)} \tag{2.24}$$

Por fim, a Tabela 2 apresentas as principais equações usadas para correção de erro sistemático, seguindo a técnica proposta por Lee et al. (2007), baseada na *UMBmark*.

Tabela 2 – Principais equações para estimação e correção de erro sistemático em robôs *Ackermann*.

Nome	Equação
α	$-\frac{\pi\left(x_{cg_{cw}} - x_{cg_{ccw}}\right)}{\left(4\pi\rho + \left(x_{cg_{cw}} - x_{cg_{ccw}}\right)\right)}$
β	$\frac{x_{cg_{cw}} + x_{cg_{ccw}}}{4\rho}$
r_{center}	$\frac{\rho}{\sin\left(\frac{\beta}{2}\right)}$

Fonte: O autor.

2.5 *Frameworks* para Robótica

É possível notar que o desenvolvimento de um sistema robótico exige muitas considerações, como o tratamento de seus sensores, controle de seus atuadores, planejamento de trajetórias, navegação, localização, planejamento de tarefas, entre tantos outros componentes. Segundo Wolf et al. (2009) técnicas de inteligência artificial, modelos matemáticos, estatísticos, modelos mecânicos, e outros métodos são utilizados pelos pesquisadores para criarem sistemas robóticos robustos e inteligentes.

Para tanto, outras abordagens são empregadas como a simulação, tendo em vista que sua utilização otimiza o processo de desenvolvimento e permite uma configuração melhor tanto do *hardware* quanto o *software* do robô (WOLF et al., 2009). O autor ainda esclarece que na simulação virtual é possível configurar os parâmetros reais do ambiente que o robô irá operar de fato, inclusive variáveis dinâmicas (randômicas), propriedades físicas, cinemáticas, etc. Essa característica propicia o uso inicial de ambientes simulados que permite testar diversas versões do mesmo algoritmo nas mesmas condições ambientais, algo impossível em condições reais. Assim, um algoritmo simulado estará pronto para ser embarcado em um robô real, em alguns casos bastará pequenas configurações (WOLF et al., 2009).

Segundo Quigley et al. (2009) outra poderosa ferramenta para o auxílio no desenvolvimento dos sistemas robóticos são os *frameworks* e *middlewares* para programação. Isto porque, os sistemas robóticos são os mais heterogêneos possíveis, pois contam com diferentes tipos de configuração de *hardware*, de *software*, de autonomia, entre outros componentes. Assim o entendimento de suas técnicas, bem como o desenvolvimento de outros robôs semelhantes aos

que já existiam, exigiam a re-implementação a partir do zero de todo o sistema, como a definição de toda a arquitetura de controle e modelagem, implementação e integração de seus *softwares*.

Por conta disso, o uso de *frameworks* e *middlewares* contribui para que os projetos sejam mais homogêneos e modulares possíveis, permitindo o reuso de estruturas já implementadas, disponibilizando uma interface prática e intuitiva entre os componentes de software, além de melhorar a legibilidade e manutenibilidade de todo o sistema. Estas características colaboram com todo o desenvolvimento do programa e avanço científico das técnicas empregadas, melhorando o entendimento do sistema para a própria equipe desenvolvedora, e facilitando sua compreensão na comunidade científica.

Quigley et al. (2009) apresenta um *framework* chamado ROS - *Robot Operating System*, como uma nova ferramenta para desenvolvimento de sistemas robóticos, e enfatiza que diferente de outros já existentes, o ROS possui um propósito mais geral, além de dar suporte a programação multi-linguagens, distribuída e modular.

2.5.1 ROS - *Robot Operating System*

Segundo Joseph (2015) o projeto ROS - *Robot Operating System*, iniciou em 2007 em uma parceria entre o projeto de robótica STAIR da Universidade de Stanford e o grupo *Willow Garage*. Quigley et al. (2009), um dos co-fundadores do projeto, afirma que o ROS surgiu como uma necessidade de infraestrutura básica para desenvolvimento de sistemas robóticos de larga escala, contemplando necessidades existentes nos *frameworks* de mesmo propósito.

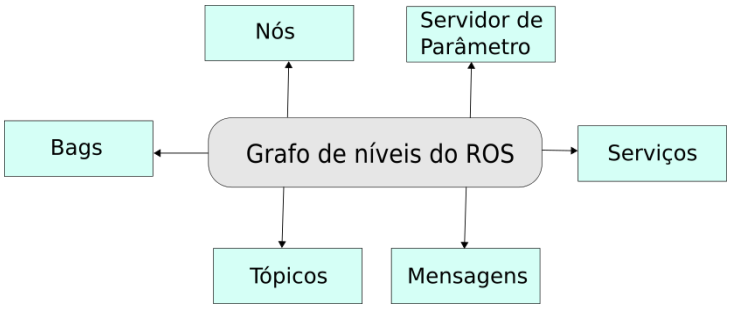
Os autores apontam que as principais características do sistema são: implementado sob uma arquitetura *peer-to-peer*; suporte a programação em multi-linguagens, ou seja, várias linguagens de programação; possui diversas ferramentas para *debuggin*, visualização e simulação dos sistemas desenvolvidos; modularidade, em que cada componente de *software* pode ser implementado em um módulo (nó ou processo), bem como, em linguagens de programação diferentes; suporte a concorrência e paralelismo; licenciado sob a licença BSD, que permite o livre uso comercial e não comercial; e, é um dos poucos *frameworks open-source* para robótica.

De acordo com Joseph (2015) e Quigley et al. (2009) os componente básicos do *framework* são os nós, tópicos, mensagens, serviços, *bags* e o servidor de parâmetros. Joseph (2015) apresenta estes componentes organizados em um grafo (Figura 22).

- **Nós:** Os *nodes*, ou nós, são os componentes onde estão implementados os algoritmos propriamente ditos. Cada um utiliza uma biblioteca cliente do ROS, como por exemplo a roscpp[3] e a rospy[4], para c++ e *python*, respectivamente, e é responsável por uma tarefa do sistema. Ademais, podem ser implementados em linguagens diferentes, a depender da familiaridade dos desenvolvedores.

[3] A documentação está disponível no sítio: <http://wiki.ros.org/roscpp>
[4] A documentação está disponível no sítio: <http://wiki.ros.org/rospy>

Figura 22 – Grafo de níveis do ROS.



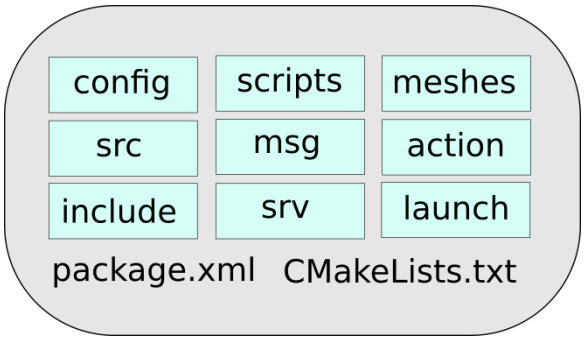
Fonte: o autor. Baseado em (JOSEPH, 2015)

Os *nodes* podem comunicar entre si utilizando as mensagens, parâmetros ou serviços. Além disso, o *framework* possui um nó especial, chamado de *master*. Em uma arquitetura distribuída, este é considerado um super nó, pois é responsável por coordenar e localizar outros. Assim, por exemplo, sem o *master*, as mensagens trocadas pelos componentes de software do sistema não chegam aos seus destinatários.

- **Tópicos:** Este é o formato padrão de comunicação entre os nós do ROS. Neste modelo, quando um nó, chamado de *publisher*, publica uma informação ou um tópico, todos aqueles que tiverem inscritos neste tópico, conhecidos como *subscribers*, receberá o dado;
- **Serviços:** Outro módulo de comunicação entres seus componentes de *software*, em que, semelhante a arquitetura cliente-servidor, um nó faz requisições de dados e serviços a outro;
- **Mensagens:** Os nós do ROS se comunicam por meio de mensagens. Segundo (JOSEPH, 2015) as mensagens são estruturas de dados que armazenam informações específicas;
- ***Bags:*** As *bags* permitem que mensagens e dados trocados no sistema sejam salvos, como em um *logger*. Assim, quando convier, por exemplo, esse *log* pode ser utilizado para re-executar uma versão de algoritmo nas mesmas condições, que fora testado anteriormente;
- **Servidor de parâmetros:** O servidor de parâmetros é um local, centralizado, que armazena parâmetros do sistema robótico. Qualquer nó pode obter ou alterar os valores do parâmetro, bem como, criar um novo e armazená-lo no servidor.

A estrutura básica de um programa no ROS é chamado pacote (JOSEPH, 2015). Segundo O'Kane (2014) em um pacote encontra-se presente todos os arquivos que possui um objetivo específico, ou seja, a implementação dos nós, bibliotecas, arquivos de configuração, modelo gráfico do robô, entre outros. A Figura 23 apresenta um esquema básico de um projeto e as unidades presentes em um pacote.

Figura 23 – Estrutura de um pacote no ROS.



Fonte: o autor. Baseado em (JOSEPH, 2015)

- **CMakeLists.txt:** Este arquivo apresenta a configuração do *cmake* para a compilação do pacote no ROS. Nele é possível encontrar as definições de: dependências do pacote; arquivos executáveis; mensagens criadas; serviços; arquivos de ações, outra funcionalidade do ROS; entre outras configurações;
- **package.xml:** Este arquivo contém metadados sobre o pacote, como o nome e *email* de seu criador, licença de uso, dependências, versão e uma descrição do pacote;
- **config:** Nesta pasta, encontram-se todos os arquivos de configuração do pacote. Por exemplo, é possível definir um arquivo de extensão *yaml*[5], para serialização de dados, como a descrição de parâmetros do sistema, que deverá ser carregado para o servidor de parâmetros do ROS.
- **include:** Organiza todos os cabeçalhos de programas em *c/c++*, também conhecido como *headers*. Um *header* é basicamente um arquivo que contém a descrição de protótipos de funções ou de uma classe.
- **scripts:** Esta pasta armazena *scripts* escrito em linguagens de programação como *pyhton*.
- **src:** Armazena código de programas em *c++*, por exemplo.
- **launch:** Contém arquivos conhecidos como *launch*, que basicamente servem para automatizar a execução de nós. Nesses arquivos é possível definir vários parâmetros pertinentes à execução destes nós, como carregar parâmetros para o servidor, ou, inclusive, inicializar mais de um nó.
- **msg:** Nesta pasta estão armazenadas as mensagens customizadas do projeto. O ROS dispões de várias mensagens padrões[6], mas permite também que os desenvolvedores criem suas próprias, as partir de pré-existentes.

[5] A extensão .yaml é utilizada para arquivos de serialização de dados inspirado em XML, que basea-se no conceito de identação, hashs, listas e dados escalares para representação de informações. Ele possui suporte à várias linguagens de programação. (<http://yaml.org/>)

[6] As mensagens padrões do ROS estão definidas no pacote *std_msgs*

- **srv:** Armazena as definições de serviços do projeto, usados na comunicação cliente-servidor.
- **action:** Está pasta contém a definição de ações do ROS. Uma ação ou *actionlib* é outro recurso do *framework*, assim como os mensagens e serviços.
- **meshes:** Contém arquivos de modelo 3D dos robôs simulados. Esta pasta não é um padrão estipulado pelo ROS, mas sim, uma convenção adotado pela comunidade desenvolvedora.

2.5.1.1 Simulação com o ROS e o simulador Gazebo

Outra característica que contribui para a popularidade do ROS, é sua integração a diversos simuladores de robótica, tais como VRep, MORSE e Gazebo, que segundo Joseph (2015) é o principal entre todos.

O Gazebo[7] é um simulador de robótica com suporte à ambientes virtuais 2D e 3D. Ele possui diversas ferramentas e *plugins* que permitem a simulação de diferentes variáveis, por exemplo: cinemática; dinâmica; temperatura; clima; fenômenos climáticos; etc. Além disso possui extensões para diversos sensores e atuadores que são usados na construção física do sistema robótico (CAÑAS; MARTN; VEGA, 2014). Segundo Staranowicz e Mariottini (2011), o simulador utiliza o motor de renderização gráfico OGRE, e, as bibliotecas de programação ODE, DART, BULLET e SIMBODY, para simulação de dinâmica de corpos rígidos, todas plataformas *open-source*.

De acordo com Joseph (2015) a interação entre o ROS e outros simuladores começa com a modelagem do corpo do robô, utilizando ferramentas própria de CAD (*Computer-aided design*). Esse etapa é crítica, pois o design do modelo influenciará na simulação, entretanto, este não precisa ter a aparência igual ao robô real, mas sim, todas as propriedades físicas. Além disso, é necessário criar uma descrição do modelo robótico, usando uma linguagem de descrição, baseada em XML, disponível no pacote *robot_model*[8], conhecida como URDF - *Unified Robot Description Format*.

Esta linguagem de marcação é responsável pela descrição em baixo nível do modelo do robô e todas as suas propriedades físicas, entretanto, possui desvantagens por conta dos recursos disponíveis para fazê-lo. Desta forma, o ROS possui outra linguagem, chamada XACRO, que utiliza a versão do XML Macro, e estende as funcionalidades do URDF, permitindo, por exemplo, a definição de macros, ou seja, funções, que permitem o reuso de código e melhora a legibilidade do arquivo.

Conforme o autor e Fernández et al. (2015) um robô para o *framework* é apenas as relações de juntas (*joint*) e articulações (*link*), que podem ser do tipo[9]: junta de revolução; contínua;

[7] sítio:<http://gazebosim.org/>

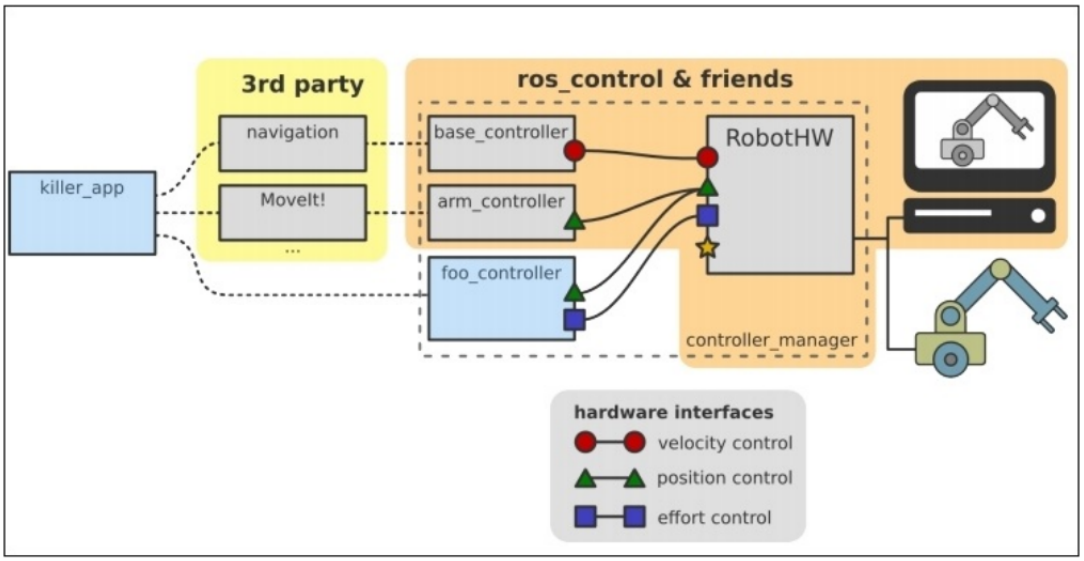
[8] este pacote é na realidade um conjunto de outros, como: *urdf, kdl_parser robot_state_publisher, collada_urdf*

[9] A descrição de cada tipo de junta pode ser vista em: <http://wiki.ros.org/urdf/XML/joint>

prismática; fixa; flutuante;e, planar. O ROS utiliza o meta pacote *robot_model*[10] para descrever e controlá-lo. Este engloba outros pacotes como o: URDF e Xacro; *"joint_state_publisher"*, responsável por examinar a descrição do robô e publicar informações sobre suas juntas móveis; kdl, um acrônimo para *Kinematic and Dynamic Library* (Biblioteca para Cinemática e Dinâmica), que permite publicação de informações sobre a cinemática direta e inversa de robôs, além de informações sobre a dinâmica destes; *"robot_state_publisher"*, implementa um nó com mesmo nome que publica transformações nas articulações do robô, baseado nas informações das juntas; etc.

Para controlar um robô, é preciso associar um controlador compatível com cada junta móvel (JOSEPH, 2015). Esse controlador é criado usando a tag *"hardwareInterface"*, e pode assumir os valor de *"hardware_interface/EffortJointInterface"*, *"hardware_interface/VelocityJointInterface"*, ou *"hardware_interface/PositionJointInterface"*. Esta é usada juntamente com tag *"transmission"* que, por sua vez, é responsável por ligar um atuador a uma junta. No *framework*, o pacote que implementa tais controladores é chamado de *"ros_control"*. A Figura 24 apresenta um diagrama que ilustra como os controladores do ROS interagem com o robô real e simulado.

Figura 24 – Interação entre os controladores do ROS e o robô real/simulado.



Fonte: Joseph (2015)

Este esquemático pode ser dividido em três partes: o programa; pacotes para planejamento; e os controladores do ROS. A primeira parte, representado por programa *"killer_app"*, é a implementação do software de controle pelo desenvolvedor. A segunda, *"3rd party"*, corresponde à utilização de ferramentas e bibliotecas do ROS para planejamento e definição de objetivos para o robô, por exemplo, a utilização do pacote *"navigation"* que possui nós para navegação de robôs, como planejamento de trajetória, SLAM, etc; e, a ferramenta MoveIt!, um *framework* para controle de robôs manipuladores, que possui integração com o ROS.

[10] A documentação do pacote pode ser encontrada no sítio: <http://wiki.ros.org/robot_model>

Por último, a terceira parte é a relação entre os controladores e o robô físico e/ou simulado. Esta relação é intermediada por uma interface, conhecida como *"hardware interface"* (interface de *hardware*), que é desacoplada à parte física e ao modelo simulável. De fato, a interface de *hardware* é uma abstração do robô, que inclui os seus sensores e atuadores. O intuito de usá-la é que as informações fornecidas à ela pode controlar tanto o robô físico, quanto o modelo simulável, evitando a necessidade de implementação de um controle separado para cada caso.

Para robôs móveis, o Gazebo possui alguns *plugins* que permitem o controle de movimento, facilitando a manipulação do mesmo. A exemplo destes, existem o *"libgazebo_ros_diff_drive.so"* para robôs diferenciais, e o *"libgazebo_ros_skid_steer_drive.so"* para robôs móveis com direção baseado em esteiras, ou que possuem 4 rodas, entretanto elas operam conjuntamente em cada lado, ou seja, as rodas da esquerda realizam movimentos sincronizados entre si, assim como os da direita.

Já para robôs *Ackermann*, o simulador não possui nenhuma biblioteca padrão, assim, cabe aos desenvolvedores implementarem um módulo que seja compatível com a cinemática, para mover o robô corretamente no ambiente.

3 DESENVOLVIMENTO DO *CELINA*

A localização de um robô móvel é peça fundamental para a elaboração e desenvolvimento de outras funcionalidades, entre elas a navegação, habilidade que distingue essa classe de robôs das demais. Este trabalho objetivou a implementação e simulação de um controle odométrico[1] para um UGV com direção *Ackermann*, usando o *framework* ROS - *Robot Operating System*, e o simulador Gazebo. A metodologia empregada fundamenta-se em pesquisa aplicada e tecnológica, cujas hipóteses foram avaliadas por meio de procedimentos experimentais realizadas em laboratório, mais especificamente, por meio de simulações.

Para tanto, conforme esclarece Wolf et al. (2009), o projeto de um sistema robótico envolve quesitos de *hardware* (sensores, atuadores e controladores) e de *software* (sistema de controle robótico). Assim, estabelecidos os requisitos, o modelo está apto para ser simulado e aplicado.

Os requisitos deste trabalho foram levantados por meio da análise do referencial teórico e estudos de trabalhos relacionados, como em: Juan e Cotarelo (2015) que desenvolvem um sistema de navegação para um veículo *Ackermann*, usando o ROS; Kaltenegger, Binder e Bader (2016), que apresentam um controle de movimentos de um robô *Ackermann* utilizando o ROS, e os controladores *raspberry pi* e arduino, aplicados e simulado; e, Allender et al. (2011) que descrevem o uso do ROS para o controle de um carro de golf, usando o arduino, encoders e bússola para prover a odometria.

Por meio desta, é possível determinar quais componentes são necessários para obter a odometria de um veículo, são elas: conhecer seus parâmetros cinemáticos; estimar o deslocamento de suas rodas; e, mensurar sua velocidade linear e angular. A partir de então, será possível realizar os cálculos, entretanto, considerando o uso do ROS como *framework* usado para o *software* de controle, é necessário incluí-lo nas condições avaliadas previamente ao desenvolvimento, assim, estudar a documentação das mensagens, nós e serviços existentes, bem como, as ferramentas que o compõem, como o RViz (visualizador de modelos em 3D e informações de sensores, como câmeras, nuvem de pontos, mapas criados, etc.), é uma premissa essencial para a continuação do desenvolvimento.

Este trabalho seguiu os passos abaixo listados:

1. **Definição de uma arquitetura de componentes de hardware:** O modelo de *hardware* é formado pela descrição dos componentes físicos presentes no veículo, suas especificação técnicas, disposição no robô, e interação entre elas, ou seja, a forma com que se comunicam. Assim, nesta fase foi desenvolvida uma arquitetura que descreve os elementos físicos

[1] Todo o código do controle implementado encontra-se no repositório do github no link: <https://github.com/IagoGomes/odometry_ros.git>

existentes, considerando os requisitos necessários para a realização deste trabalho, assim como, as limitações do veículo em sua atual fase de desenvolvimento;

2. **Desenvolvimento do Software:** Por sua vez, o *software* são os componentes lógicos presentes no veículo, ou seja, todo o programa implementado para controlá-lo, fazendo-o alcançar os objetivos pretendidos, no caso deste trabalho, estimar a localização por meio de cálculos odométricos, bem como, controlar seu modelo simulável.

   Por conseguinte, considera-se como parte integrante da especificação de *software*, todas as métricas e métodos utilizados para sua realização, como, modelos de arquitetura, metodologias de desenvolvimento, e processos que compõem a documentação.

   O ROS possui uma arquitetura baseada em pacotes, que garante a modularidade do sistema, e, melhora a legibilidade e manutenibilidade. Por conta disso, cada componente pode ser implementado seguindo um paradigma de programação específico (por exemplo, a orientação a objetos ou programação declarativa) a depender da familiaridade do programador e das funcionalidades pretendidas para o programa.

   Por conta dos modelos desenvolvidos, utilizou-se a programação orientada a objetos, que garante ao sistema, escalabilidade e uma documentação mais detalhada, seguindo os padrões da UML - *Unified Modeling Language*, Linguagem de Modelagem Unificada. O desenvolvimento do *software* seguiu os seguintes passos:

   a) **Levantamento de requisitos para a implementação da odometria e simulação do robô:** especificação e análise de requisitos para o projeto;
   b) **Criação da hierarquia de arquivos para o projeto:** estruturação da hierarquia de pacotes do ROS, responsáveis pelo controle odométrico e simulação;
   c) **Criação do modelo virtual do veículo:** desenvolvimento de um modelo simulável do robô, utilizando a linguagem baseada em XML conhecida como *Xacro*;
   d) **Desenvolvimento do modelo odométrico utilizando o ROS:** implementação do controle proposto, utilizando as fórmulas explanadas no capítulo anterior (sessão 2.4.1.2);
   e) **Planejamento e análise de teste:** planejamento, realização e análise de teste do sistema.

3.1 Arquitetura de *hardware*

O *CELiNA* - Carro Elétrico Livre com Navegação Autônoma (Fig. 25), é um projeto da Universidade Estadual do Sudoeste da Bahia, que objetiva desenvolver um UGV capaz de locomover-se pelo campus da instituição. O projeto centra-se em criar um modelo que seja objeto de pesquisa em robótica móvel para alunos dos cursos de Ciências Exatas, explorando as vastas aplicações, interdisciplinaridade e temas presente no campo de estudo.

Figura 25 – *CELiNA*

Fonte: o autor.

O veículo é baseado em um projetos abertos de *Kart-Cross*, cuja tração é realizada por um motor elétrico de corrente contínua (CC) utilizados em *Scooters*, com potência de 1000w e alimentação de 36V. A direção *Ackermann* foi implantada utilizando uma caixa de direção de Mini *Buggy*. A Tabela 3 apresenta as demais características do *CELiNA*.

Tabela 3 – Características do CELiNA

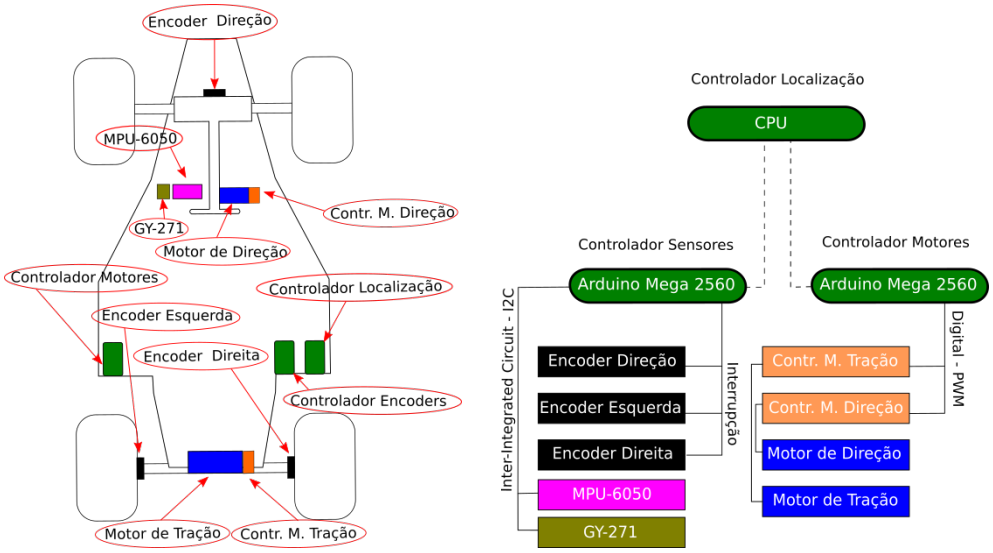
Característica	Valor
Velocidade Máxima	$5m/s$
Comprimento	$2.30m$
Largura	$1.22m$
Angulação máxima do volante	$[-30°, 30°]$
Distância entre eixos (W_{base})	$1.42m$
Diâmetro da Roda	$0.30m$
Distância entre os pontos médios das rodas ($d_{baseline}$)	

Fonte: O autor.

Além dessas descrições, foi desenvolvido uma arquitetura de componentes (Fig. 26) que apresenta os sensores, atuadores e controladores presentes no robô, utilizados na implementação do presente proposto. É importante salientar que o *CELiNA* ainda está em fase de desenvolvimento e construção mecânica, assim, nem todos os componentes apresentados a seguir foram incorporados no robô físico, trata-se apenas de seu projeto arquitetural, e modelo usado na simulação. A Figura 26(a) mostra a disposição dos componentes e, a Figura 26(b) a comunicação entre eles.

O veículo possui 3 encoders quadráticos (sensores passivos que são capazes de determinar a distância percorrida pela roda, bem como, o sentido da rotação), um ligado à roda traseira esquerda, outra na direita e o último no eixo da transmissão entre o volante e a caixa de direção frontal. Além dos encoders, o robô possui um sensor MPU-6050 e um GY-271 integrados no barramento I^2C (*Inter-Integrated Circuit*, Circuito Inter-Integrado), compondo um IMU

Figura 26 – Arquitetura de componentes do *CELiNA*



(a) Componentes do *CELiNA* (b) Comunicação dos componentes

Fonte: o autor.

- Unidade de Medida Inercial (*Inertial Measurement Unit*), que fornecem medições sobre a velocidade linear (acelerômetro), angular (giroscópio) e orientação (bússola), que são usados como redundância no cálculo de orientação, com o intuito de diminuir o erro odométrico, principalmente os não sistemáticos.

Um arduino mega 2560[2], baseado no microcontrolador *ATmega 2560*, é usado para ler os sensores, que são ligados à pinos de interrupção de *hardware* do controlador. Outro arduino é usado para controlar os dois motores existentes (motor de tração; e, motor de direção), que estão ligados à ESCs para controle de motores CC, e, por sua vez, são conectados ao controlador por pinos digitais com suporte a modulação de largura de pulso (PWM - *Pulse Width Modulation*), já que os ESCs são controlados pela variação desses pulsos, alterando a velocidade e direção da rotação do motor. Por fim, um último controlador é usado para determinar a posição do robô, neste serão implementados todos os algoritmos de localização, como é o caso da odometria explanado neste trabalho.

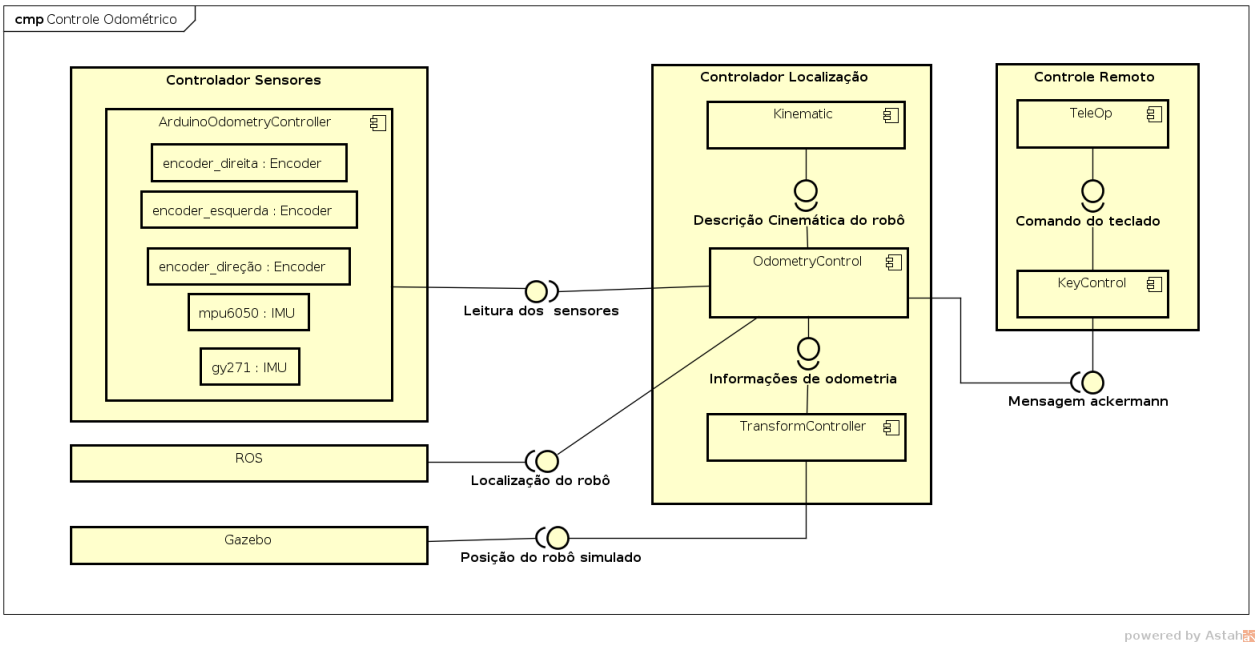
3.2 Requisitos para o controle odométrico

O controle de odometria utiliza as informações dos sensores (encoders) existentes no robô e da descrição cinemática do mesmo, para então, estimar sua posição e atualizá-la no modelo simulável (OLSON, 2000). A Figura 27 apresenta o diagrama de componentes do projeto que envolve e relaciona os sensores e os elementos responsáveis pelo controle. O sistema possui

[2] A descrição da plataforma pode ser encontrada no sítio: <https://www.arduino.cc/en/Main/arduinoBoardMega2560>

cinco componente principais, são eles: Controlador Sensores (CS); Controlador Localização (CL); Controle Remoto (CR); ROS; e, Gazebo.

Figura 27 – Diagrama de Componentes do Controle de Odometria



Fonte: o autor.

3.2.1 ROS

Neste contexto, o ROS representa todos os componentes do *software* controlador do robô que não estejam relacionados com o sistema de odometria, como, os algoritmos de navegação, processamento de imagem, inteligente artificial, etc. Como alguns desses elementos necessitam da posição do robô, então, este elemento requer esta informação, que é fornecida por *CL*;

3.2.2 Controlador Sensores

Este componente está ligado ao *hardware* do veículo, e é responsável por realizar as leituras dos sensores (*encoder_direita*, *encoder_esquerda*, *encoder_direção*, *mpu6050* e *gy271*), assim, fornece aos demais os valores lidos, que serão utilizados no cálculo da velocidade linear e angular do robô.

Além dos sensores, o Controlador Sensores é formado por um nó do ROS, o *ArduinoOdometryController*, que foi implementado usando a biblioteca cliente para a plataforma arduino[3]. Este, por sua vez, fornece aos demais componentes do sistema, as leituras dos sensores, publicando os tópicos *"/robot/odom/ticks"* que contém as leituras dos encoders, e *"/robot/odom/imu"* que dispõe das medições dos sensores inerciais.

[3] A documentação da biblioteca pode ser encontrada no sítio:<http://wiki.ros.org/rosserial_arduino>

3.2.3 Controlador Localização

O *Controdador Localização* é formado por três componentes: *Kinematic*; *OdometryControl*; e, *TransformController*. Juntos, são responsáveis pela estimação da posição do veículo e atualização desta, no modelo simulado.

OdometryControl e *TransformController* são nós implementados usando a biblioteca cliente do ROS para a linguagem de programação *C++*. O primeiro nó é responsável por calcular e publicar as informações de odometria, utilizando o tópico *"/robot/odom"* que publica mensagens do tipo *"nav_msg::Odometry"*. Está mensagem pertence ao pacote *nav_msg*[4], que é usado para algoritmos de navegação de robôs móveis. Já o segundo nó atualiza a posição do robô no ambiente simulado assinando o tópico *"/robot/simulation/tf"*, que publica mensagens do tipo *"ackermann_msgs/AckermannDriveStamped"*, e utiliza a classe *TransformBroadcaster* presente no pacote *tf2_ros*[5].

3.2.4 Controle Remoto

O controle remoto é uma funcionalidade do sistema que permite controlar o robô simulado por meio do teclado do computador. Assim, é formado por dois componentes: o *TeleOp* implementado usando a biblioteca cliente do ROS para a linguagem *python*, e responsável por obter do teclado os comandos inseridos, publicando-os por meio do tópico *"/robot/control/cmd_vel"*; e o nó *KeyControl*, implementado em C++, que assina o tópico publicado por *TeleOp*, e fornece ao controlador odométrico tais informações, por meio do tópico *"/robot/control/ackermann_command"*.

3.2.5 Gazebo

O componente Gazebo é tudo o que compreende a simulação do robô, ou seja, o modelo simulável e seu controle, que por sua vez, representa o recebimento das informações de transformações publicadas pelo nó *TransformController* e a atualização na visualização.

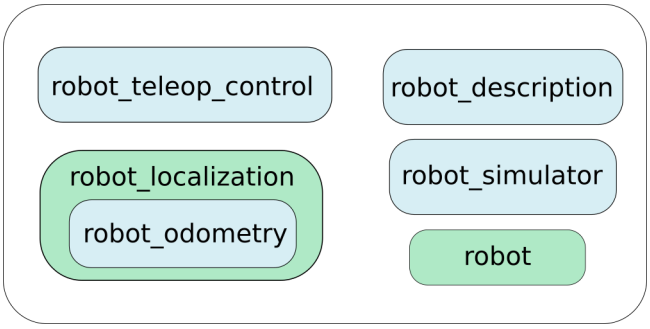
O modelo simulado é descrito usando uma linguagem de marcação baseada em XML chamada Xacro, que estende as funcionalidades do modelo padrão do ROS conhecida como URDF - *Universal Robotic Description Format* (Formato Universal de Descrição de Robôs) . O simulador Gazebo, no entanto, converte o Xacro em um arquivo SDF - *Simulation Description Format* (Formato de Descrição de Simulação), uma vez que no primeiro é permitido descrever apenas as configurações do robô, como formas, cinemática e algumas propriedades da dinâmica, já o SDF pode representar outros componentes do ambiente simulado, como a iluminação (GAZEBO, 2017).

[4] A documentação do pacote está disponível no sítio: <http://wiki.ros.org/nav_msgs>
[5] A documentação do pacote está disponível no sítio: <http://wiki.ros.org/tf2_ros>

3.3 Hierarquia de pacotes do projeto

Um software no ROS é organizado em pacotes, que garante sua modularização e escalabilidade (O'KANE, 2014; JOSEPH, 2015). Este pode, ou não, possuir um ou mais nós, que representam as unidades de programas responsáveis pelo funcionamento e controle do robô. Não há uma regra sobre como organizar os pacotes do sistema, mas, a partir da análise de projetos existentes que utilizam o *framework*, bem como os exemplos apresentados por Joseph (2015), O'Kane (2014) e Fernández et al. (2015) foi possível notar algumas convenções seguidas pela maioria dos pesquisadores. A Figura 28 apresenta a organização dos pacotes utilizados para desenvolver este trabalho.

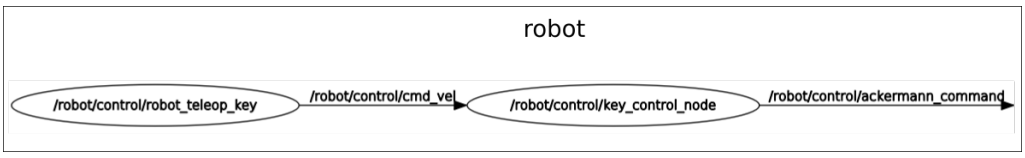
Figura 28 – Hierarquia de pacotes do projeto



Fonte: o autor.

- **robot_teleop_control:** Este pacote é responsável pelo controle do modelo simulável do robô, utilizando o teclado do computador. Para tanto, ele implementa dois nós, o *"robot_teleop_key"* (*TeleOp*) e *"key_control_node"* (*KeyControl*) (Figura 29). O primeiro nó é um *script* em *python* que obtém do teclado as teclas digitadas e as convertem em uma mensagem do tipo *"geometry_msgs/Twist"* publicadas pelo tópico *"/robot/control/cmd_vel"*, por conseguinte, o segundo nó, implementado em *C++*, assina esse tópico e converte seus valores em uma mensagem do tipo *"ackermann_msgs/AckermannDriveStamped"* que é publicado pelo tópico *"/robot/control/ackermann_command"*. Este tópico é assinado pelo controlador odométrico que é capaz de transformar suas informações em movimentos, e por fim, atualizá-los no robô simulado.

Figura 29 – Interação entre os nós *"robot_teleop_key"* e *"key_control_node"*.



Fonte: o autor.

- **robot_description:** Este pacote possui os arquivos que descrevem o robô usando o *Xacro*, e a modelagem 3D do chassi. Esses arquivos são usados para apresentar o veículo no simulador Gazebo e na ferramenta RViz, além disso, possui toda a descrição cinemática do mesmo, uma vez que a descrição em *Xacro* é também responsável pela definição de algumas limitações do robô, como esterçamento das rodas frontais, velocidade máxima, tamanho do chassi, etc.

- **robot_simulator:** Este pacote possui os *script* para automatizar as execuções de nós do ROS, conhecida como arquivo *launch*, responsáveis por iniciar a simulação do robô no Gazebo, bem como executar os nós encarregados pela interação entre o simulador e o *framework*. Para tanto, o pacote implementa também o *"transform_controller"* (*TransformController*) que assina o tópico *"/robot/simulation/tf"*, e o utiliza para atualizar a posição do robô simulado, publicando informações sobre cada articulação deste.

- **robot_localization:** Está pasta não se trata necessariamente de um pacote do projeto, mas sim um diretório para separar os pacotes destinados à localização do *CELiNA*. Inicialmente contém apenas o pacote *robot_odometry*, que possui os nós *"odometry_control"* (*OdometryControl*)e *"arduino_odometry_control"* (*ArduinoOdometryControl*), responsáveis pelo cálculo odométrico e leitura dos sensores do robô, respectivamente.

- **robot:** Este diretório também não é um pacote ROS, mas existe como pasta principal do sistema, assim, possui o arquivo *launch* que inicializa todos os componentes necessários para executar o sistema por completo, inicializando os nós para a odometria e simulação.

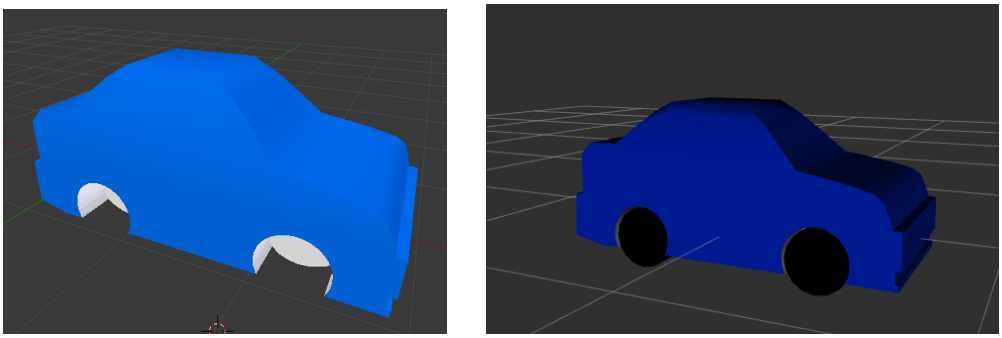
3.4 Modelagem e simulação do robô

Segundo Joseph (2015) a simulação de um robô começa com o desenvolvimento de sua aparência, que pode ser realizado com o auxílio de ferramentas de CAD, como o *Blender* 3D. O autor esclarece que esse modelo precisa ter todas as características do robô, principalmente aquelas que referem-se à cinemática e dinâmica, todavia, a aparência, ou seja, a parte visual, não precisa necessariamente parecer com o real.

3.4.1 Modelagem

O *CELiNA* é um veículo baseado em um *Kart-Cross*, entretanto, o modelo desenvolvido não possui a forma igual ao real, pois, optou-se por utilizar uma mais simples, que não possui tantos detalhes, mas implementa todas suas características físicas. A Figura 30(a) apresenta o chassi do robô, modelado usando o programa *Blender* 3D e exportado no formato DAE (Ativo Digital de Troca, do inglês *Digital Asset Exchange*) que é baseado no XML COLLADA. Este modelo será usado no arquivo Xacro, dando forma ao *CELiNA* (Fig. 30(b)).

Figura 30 – Modelagem do robô.



(a) Modelagem 3D (b) Modelagem Xacro

Fonte: o autor.

Após a modelagem 3D, o robô precisa de uma descrição para então ser simulado pelo ROS e Gazebo. Esse descrição é nada mais que um arquivo XML que mostra como as juntas e articulações do robô estão relacionadas, e quais suas propriedades. No entanto, algumas considerações precisam ser realizadas antes de descrever o robô usando o Xacro. O *CELiNA* é um robô com direção *Ackermann*, desta forma possui quatro rodas, duas frontais e duas traseiras. As traseiras são responsáveis pela tração do veículo, já as dianteiras, determinam a direção. A Figura 31 apresenta o modelo resultante do implementação da descrição do veículo. Esta representação é gerada pelo próprio *framework* a partir do arquivo URDF. A Tabela 4 expõe os detalhes de cada junta.

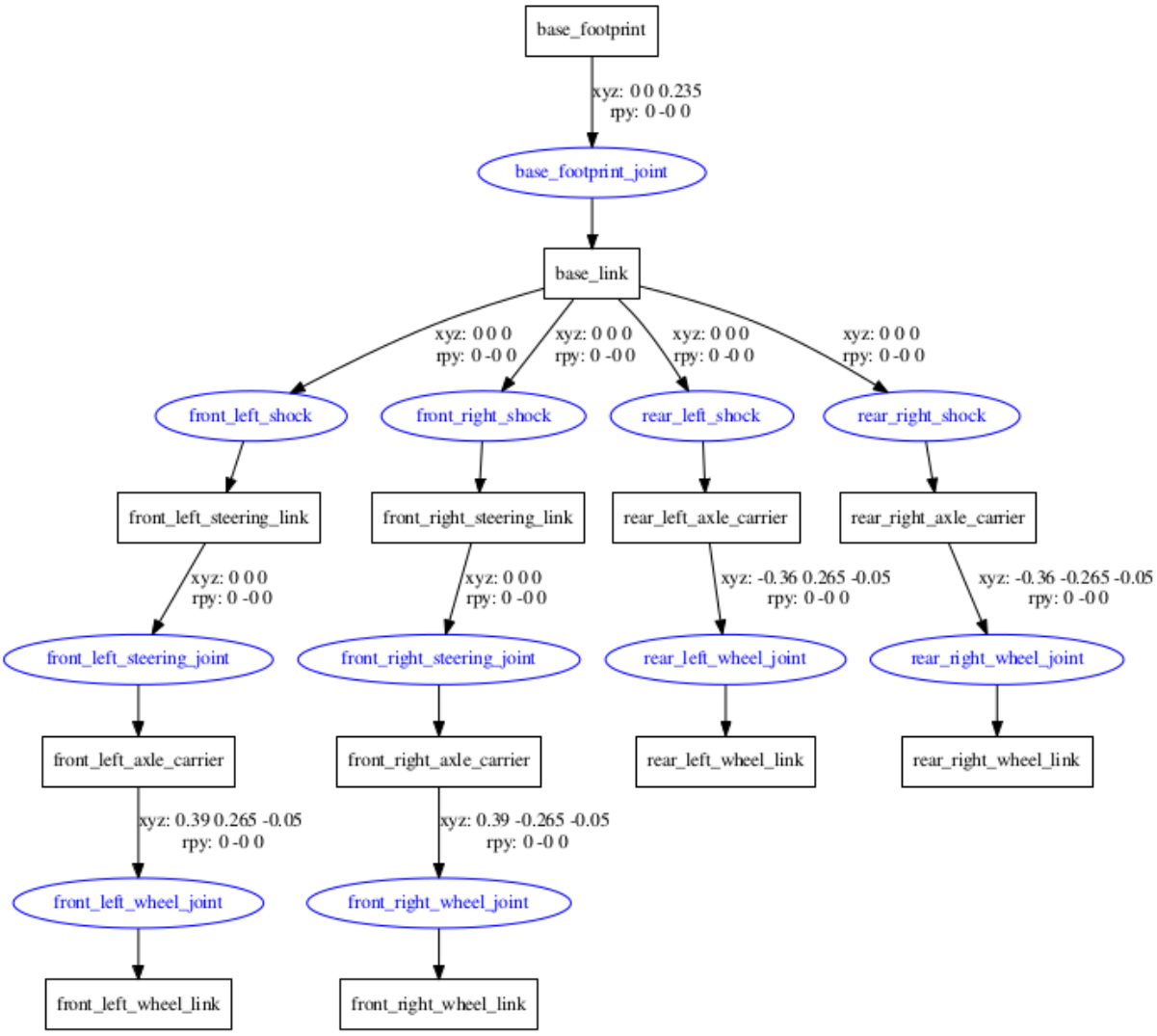
Tabela 4 – Descrição das juntas do modelo simulável do *CELiNA*

Nome	Tipo	Tipo do Controlador
base_footprint_joint	fixa	-
front_left_shock	prismática	*EffortJointInterface*
front_right_shock	prismática	*EffortJointInterface*
rear_left_shock	prismática	*EffortJointInterface*
rear_right_shock	prismática	*EffortJointInterface*
front_left_wheel_joint	contínua	*VelocityJointInterface*
front_right_wheel_joint	contínua	*VelocityJointInterface*
rear_left_wheel_joint	contínua	*VelocityJointInterface*
rear_right_wheel_joint	contínua	*VelocityJointInterface*
front_right_steering_joint	revolução	*EffortJointInterface*
front_left_steering_joint	revolução	*EffortJointInterface*

Fonte: O autor.

A implementação é dividida em 4 arquivos presentes no pacote *"robot_description"*: o *"base_link.urdf.xacro"*; *"wheel.urdf.xacro"*; *"robot.urdf.gazebo"*; e, *"robot.urdf.xacro"*. Os dois primeiros possui a implementação de macros, para criação das rodas e do chassi do robô. As macros são tais como as funções na programação, elas permitem criar objetos com propriedades semelhantes sem que haja a repetição de código. Por fim, o terceiro arquivo possui a inclusão de

Figura 31 – Modelo de descrição do robô.



Fonte: o autor.

algumas propriedades úteis para simular o robô no gazebo, e o quarto a descrição completa do *CELiNA*, que, por sua vez, utiliza os demais arquivos.

- ***base_link.urdf.xacro:*** Este arquivo possui a macro que define o chassi do robô. Para simular o robô no ROS usa-se uma articulação chamada de *"base_link"* responsável por ligar os demais componentes deste. No entanto, o gazebo exige que seja criado uma articulação e uma junta fixa a mais, chamados aqui de *"base_footprint"* e *"base_footprint_joint"*, para fixar o modelo no ambiente simulado. Esse dois últimos componentes não possuem nenhuma propriedade física, já *"base_link"* inclui as características do *CELiNA* como a massa, definição inercial, propriedade de colisão, e, aparência.

- ***wheel.urdf.xacro:*** Este arquivo possui a implementação da macro *"wheel"* usada para

descrever rodas genéricas do veículo, além disso, implementa também as macros *"shock"* para o amortecimento, *"front_wheel"* para as rodas frontais, e *"rear_wheel"* para as rodas traseiras. A diferença entre as rodas frontais e traseiras é que a primeira possui uma articulação e uma junta de revolução a mais, que são responsáveis pelo esterçamento do veículo.

- ***robot.urdf.xacro:*** Este é o arquivo principal da modelagem do robô, assim é chamado pelos launch do sistema para iniciar a simulação do veículo, incluindo-o no ambiente Gazebo e RViz, quando convier. É nele que são realizada as chamadas das macros *"front_wheel"*, *"rear_wheel"* e *"robot_base"*, dando forma ao *CELiNA* simulável.

- ***robot.urdf.gazebo:*** Basicamente a função deste arquivo é realizar configurações pertinentes à simulação do robô no ambiente Gazebo. Assim, encontra-se nele a inclusão de um plugin chamado *"gazebo_ros_control"*.

3.4.2 Simulação

O controle do *CELiNA* é realizado pelo nó de *"TransformController"* que converte as informações odométricas em posição e velocidade para o modelo simulável. O nó também faz uso da descrição cinemática do veículo, e é nele que são implementados os controladores do ROS para manipulação de juntas móveis (Como explanado na sessão 2.5.1.1).

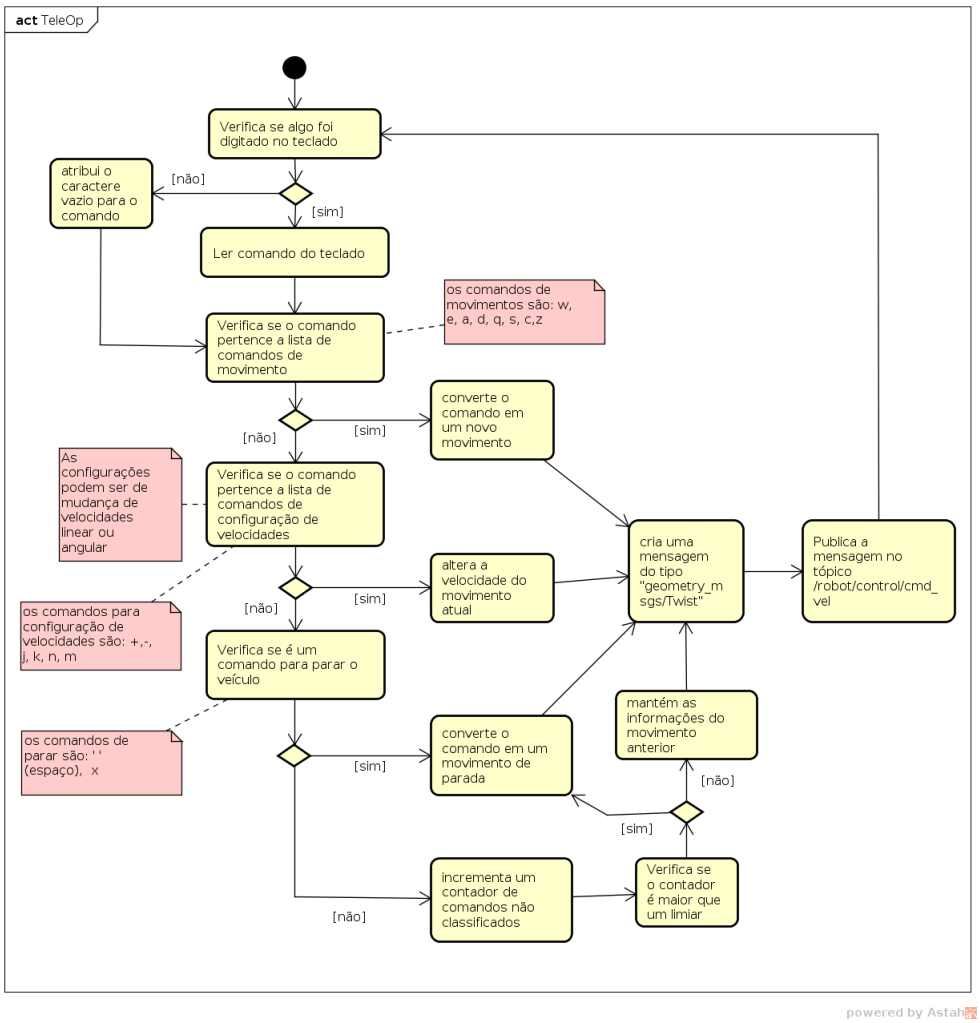
O *CELiNA* pode ser controlado de duas maneiras, a primeira é a partir de um controle remoto implementado via interface de teclado, e a segunda é a atualização do modelo simulável por meio das informações obtidas pelos sensores do robô real ou a simulação desses.

3.4.2.1 Controle Remoto

O controle remoto é realizado por meio de dois nós, o *TeleOp* e o *KeyControl*. A Figura 32 apresenta o diagrama de atividade que descreve o funcionamento do primeiro. *TeleOp* é responsável por obter do teclado o comando digitado, convertendo-o em uma mensagem do tipo *"geometry_msgs/Twist"*, que é assinada por *KeyControl* e convertida em uma mensagem do tipo *"ackermann_msgs/AckermannDriveStamped"*, que contém as informações quanto a velocidade linear do veículo e o ângulo de esterçamento das rodas frontais.

No fluxo de *TeleOp* (Fig. 32), a primeira parte consiste em verificar se alguma tecla foi digitada, e se sim, de qual se trata. Em seguida, esse comando é classificado em "movimentos", "ajustes de velocidades" ou "solicitação de parada". A Figura 33 apresenta uma captura de tela que expõe as teclas utilizadas para as manobras do robô. Como trata-se de um veículo *Ackermann*, não há disponível a possibilidade de realizar rotações sem que haja movimento de translação, assim, as teclas *'a'* e *'d'*, comumentemente usadas para tal propósito, apenas mudam o ângulo de esterçamento das rodas frontais do veículo, mantendo sua orientação. As teclas *'w'* e *'s'* movem o *CELiNA* para frente e trás, nessa ordem, e, *'q'* e *'e'* são as junções de movimentos

Figura 32 – Diagrama de atividades do nó *TeleOp*



Fonte: o autor.

de esterçamento das rodas frontais e translação longitudinal (frente e ré), capazes de mudar a orientação do veículo, deslocando-o ao mesmo tempo. Quanto à configuração de velocidades, os comandos mais importantes, são o *'+'* e *'-'*, que, respectivamente, aumentam e diminuem sua velocidade linear e angular. Por fim, as teclas de espaço e *'x'* realizam a parada imediata do robô. Na sequência, após classificar o tipo de ação solicitada, uma mensagem que desempenha o comando é criada e publicada.

Por sua vez, *KeyControl*, assina o tópico publicado por *TeleOp*, e converte a mensagem recebida em uma que contém as informações de velocidade linear e ângulo de esterçamento. Ademais, é sua responsabilidade também limitar tais informações para os máximos e mínimos estipulados pelas características cinemáticas. O tópico *"/robot/control/ackermann_command"* que carrega a mensagem criada será assinado pelo controlador odométrico e utilizado para estimar e atualizar a posição do veículo no ambiente simulado.

Figura 33 – Comandos do nó *TeleOp*

```
Control Your Robot!
---------------------------
Moving around:
   q    w    e
   a    s    d
   z    x    c

+/- : increase/decrease max speeds by 10%
j/k : increase/decrease only linear speed by 10%
n/m : increase/decrease only angular speed by 10%
space key, x : force stop
anything else : stop smoothly

CTRL-C to quit
```

Fonte: o autor.

3.4.2.1.1 Controle do modelo simulado

A atualização da posição e controles de ações no modelo simulado é realizado pelo nó *TransformController*, que obtém da odometria, assinando o tópico *"/robot/simulation/tf"*, as informações sobre a velocidade linear e ângulo de esterçamento das rodas frontais, publicando tais informações para cada junta móvel do robô.

Para mover o robô, segundo Joseph (2015), um controlador específico é associado às juntas móveis (como explanado no capítulo anterior, vide 2.5.1.1), que obtém o novo estado desta e a atualizá na simulação. A Tabela 4 apresentou os tipos de controladores para o *CELiNA*. Contudo, antes é necessário configurar alguns de seus parâmetros, por exemplo, os ganhos do P.I.D. (Proporcional, Integral e Derivativo) que é um algoritmo usado para suavizar a mudança de ações nos atuadores. Assim, para tal propósito, utiliza-se um arquivo de configuração que contém a declaração de variáveis que são carregadas para o servidor de parâmetros do ROS (ROS *Param*) antes que a simulação seja inicializada. Depois disso, um nó do ROS, conhecido como *"controller_spawner"*, é criado e é o responsável por dar início a todos os controladores necessários.

No *CELiNA*, o arquivo que contém essas informações está presente na pasta *"config"* do pacote *"robot_simulator"* e possui a extensão *".yaml"*, um acrônimo para "YAML não é linguagem de marcação" (*YAML Ain't Markup Language*), que trata-se de um formato de serialização de dados. A Tabela 5 apresenta os tópicos das juntas, e o tipo de mensagem associada a cada um. Esses valores são carregados para o servidor e os nós e a simulação são inicializados por meio dos arquivos launch nomeados de *"robot_control_gazebo.launch"* e *"robot_gazebo.launch"*.

O nó *TransformController* então consulta no servidor de parâmetros do ROS os tópicos para cada junta e cria um publicador (*publisher*) para cada um. Assim, ao assinar o tópico *"/robot/simulation/tf"* ele obtém do controlador odométrico os valores da velocidade linear e ângulo de esterçamento do veículo. A velocidade é publicada para às juntas *"front_left_wheel_joint"*, *"front_right_wheel_joint"*, *"rear_left_wheel_joint"* e, *"rear_right_wheel_joint"*, fazendo o robô se mover para frente e traz. Já a angulação é convertida em um ângulo para as rodas da esquerda

Tabela 5 – Tópicos e mensagens das juntas do *CELiNA*

Junta	Tópico	Tipo da Mensagem
front_left_wheel_joint	*/robot/simulation/front_left_axle_controller/command*	*std_msgs::Float64*
front_right_wheel_joint	*/robot/simulation/front_right_axle_controller/command*	*std_msgs::Float64*
rear_left_wheel_joint	*/robot/simulation/rear_left_axle_controller/command*	*std_msgs::Float64*
rear_right_wheel_joint	*/robot/simulation/rear_right_axle_controller/command*	*std_msgs::Float64*
front_right_steering_joint	*/robot/simulation/front_right_steering_joint_controller/command*	*std_msgs::Float64*
front_left_steering_joint	*/robot/simulation/front_left_steering_joint_controller/command*	*std_msgs::Float64*

Fonte: O autor.

e direita usando as fórmulas 2.1 e 2.2, e transmitidas às juntas *"front_left_steering_joint"* e *"front_right_steering_joint"*, respectivamente.

3.5 Desenvolvimento do controle odométrico

O controle odométrico do *CELiNA* é implementado no pacote *"robot_odometry"*, e é composto por dois nós, o *OdometryControl* e *ArduinoOdometryControl*, que realizam os cálculos da localização do robô e as leituras dos sensores, nesta ordem.

Os sensores, chamados por Olson (2004) de *dead-reckoning* (discutido no capítulo anterior, sessão 2.4.1), estão conectados ao arduino mega 2560, e suas medições são repassadas ao ROS por meio de um programa que utiliza a biblioteca cliente para comunicação serial, chamada de *rosserial*[6]. Esta, biblioteca possui uma adaptação do cliente ROS para C++, criada especificamente para sistemas embarcados, assim, ela emula um nó qualquer no arduino (ARAÚJO et al., 2013). Desta forma, o *"ArduinoOdometryControl"* realiza a leitura de 3 sensores encoders (posicionados conforme Figura 26) e cria uma mensagem do tipo *"odometry::odom_info"* (Fig. 34), que fora criada exclusivamente para carregar tais informações.

Figura 34 – Estrutura da mensagem *"odometry::odom_info"*

```
$ rosmsg show odometry/odom_info
int32 left_rw_enc
int32 right_rw_enc
int32 steering_enc
```

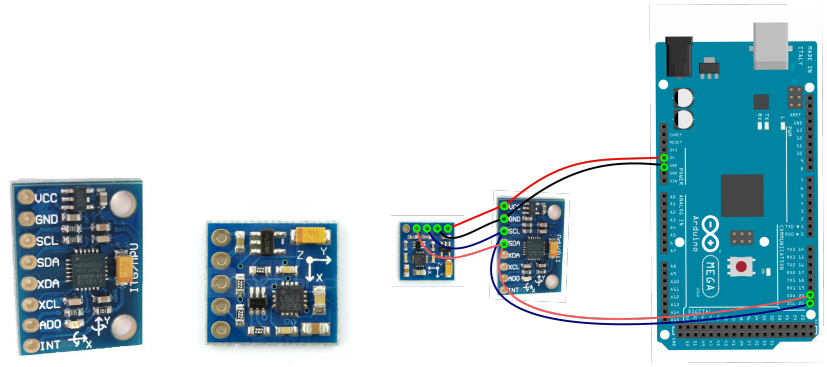
Fonte: o autor.

Da mesma maneira, este nó realiza a leitura dos dois sensores que compõem o IMU, e estão conectados no barramento I^2C do microcontrolador. Neste circuito o arduino é o master do protocolo e o MPU-6050 (Fig. 35(a)) e GY-271 (Fig. 35(b)) são os escravos, endereçados respectivamente nas posições $0x68$ e $0x1E$. O módulo do MPU possui uma conexão específica que permite ligar outro sensor ao protocolo, contudo a forma mais fácil e prática de utilizá-los é

[6] A documentação desta biblioteca pode ser encontrada no sítio <http://wiki.ros.org/rosserial>

ligando os pinos SDA (*Serial Data Line*) e SCL (*Serial Clock Line*) juntos, e conectá-los ao seu master. Como possuem endereços diferentes, o arduino sabe distinguir as leituras de cada um. Após obter suas leituras, o nó as armazenam em uma mensagem do tipo *"sensor_msgs/Imu"*, que pertence ao pacote *"sensor_msgs"*[7] , e a pública no tópico *"/robot/odom/imu"*. A Figura 35(c) apresenta o circuito prototipado para a realização deste experimento.

Figura 35 – Prototipação do IMU



(a) MPU6050 (b) GY-271 (c) Circuito

Fonte: Arduino (2017), botland (2017) e o autor.

Com os dados dos sensores providas por *ArduinoOdometryControl*, o nó *Odometry-Control*, responsável pelo cálculo odométrico, possui as ferramentas necessárias para estimar a posição do robô. Este, por sua vez, realiza tal ação em duas maneiras, a primeira é usando as informações vindas do controle remoto publicadas por *KeyControl*. A segunda forma é utilizando as leituras dos sensores supracitados. Ambos os métodos utilizam as características cinemáticas do *CELiNA* que foi carregada para um objeto da classe *Ackermann*, criada com as implementações das equações do modelo cinemático necessárias para o desenvolvimento do trabalho (explanadas no capítulo 2, sessão 2.4.1.2). A diferença entre os métodos é apenas a fonte das informações usadas, entretanto, a forma do cálculo é muito semelhante.

3.5.1 Odometria baseada no controle remoto

Ao receber uma mensagem do tópico *"/robot/control/ackermann_command"*, o controlador odométrico atualiza as informações de velocidade linear e ângulo de esterçamento das rodas dianteiras do veículo, e permite que esses dados sejam usados no laço principal do nó, para calcular a posição cartesiana e orientação, publicando a mensagem de odometria e de transformação do robô simulado.

Neste modo, não há nenhuma alteração ou manipulação dos dados, exceto limitá-los conforme a descrição cinemática, para, por exemplo, o robô não andar em uma velocidade maior que o máximo estipulado. Isto porque, a velocidade linear e esterçamento já estão prontos para

[7] A documentação deste pacote pode ser encontrado no sítio <http://wiki.ros.org/sensor_msgs>.

uso nas equações odométricas, assim, basta aplicá-los nas equações mostrada na Tabela 1 e a odometria estará calculada.

3.5.2 Odometria baseada nas leituras dos sensores

Nesta abordagem, o robô utiliza as informações sensoriais para então calcular a odometria. Os encoders são responsáveis por estimar o deslocamento das rodas, conforme as equações 3.3 e 3.4. Com essa informação é possível também computar a orientação do veículo a partir das fórmulas citada nas Tabelas 2.4.1.1 e 1, entretanto, como os modelos odométricos acumulam muitos erros (BORENSTEIN; FENG, 1996; MUNIANDY; MUTHUSAMY, 2012; CENSI et al., 2013), principalmente quando utilizam-se encoders, que estão acometidos às imperfeições cinemáticas e irregularidades dos terrenos, optou-se, então, por utilizar também o IMU, com o propósito de diminuir a influência do erro sistemático e não sistemático na estimação da orientação. Assim, uma média aritmética é realizada entre os ângulos computados por ambos sensores.

Para calcular a localização usando os encoders, algumas adaptações são realizadas nas equações, tendo em vista a forma com que esses estimam o deslocamento do robô. Segundo Lee, Chung e Yoo (2010), basicamente as equações são alteradas quanto a maneira que se calcula a distância individual das rodas, pois, esses sensores contam uma quantidade de interrupções ao longo do tempo, assim, sabendo a quantia que ocorre em um perímetro de roda do veículo, e, conhecendo também o tempo passado entre medições, é possível então determinar a distância, e por conseguinte, a velocidade linear do veículo.

Sabendo que a distância percorrida pelo robô é dado pela Equação A.1, conforme os autores, considerando o uso de encoders em cada roda traseira, d_{left} e d_{right} são definidos por :

$$d_{left} = \frac{\pi \oslash_{left} N_{left}}{Re_{left}} \tag{3.1}$$

$$d_{right} = \frac{\pi \oslash_{right} N_{right}}{Re_{right}} \tag{3.2}$$

Como a expressão $\pi \oslash_{left/right}$ é o mesmo que o perímetro da circunferência, já que $\oslash_{left/right}$ é o diâmetro nominal de cada roda, então, as fórmulas 3.1 e 3.2 podem ser rearranjadas como:

$$d_{left} = \frac{\rho_{left} N_{left}}{Re_{left}} \tag{3.3}$$

$$d_{right} = \frac{\rho_{right} N_{right}}{Re_{right}} \tag{3.4}$$

Sendo, $N_{left/right}$ a quantidade de interrupções ocorridas em um intervalo de tempo t para cada roda, $Re_{left/right}$ a resolução dos sensores para uma revolução completa das rodas, e $\rho_{left/right}$ o perímetro de cada roda.

Usando A.1, 3.3 e 3.4 é possível calcular a distância percorrida pela robô, e obtendo o intervalo t, por meio da biblioteca do ROS para tempo, conhecida como *ros::Time*[8], encontrar-se a velocidade linear do *CELiNA*. Salientando que, conforme a descrição cinemática (vide sessão 2.4.1.2) essa informação é obtida apenas analisando as rodas traseiras do veículo.

Quanto ao ângulo de esterçamento das rodas frontais, o cálculo é realizado com base nas leituras do encoder que foi colocado no eixo entre a transmissão do volante e a caixa de direção. Este sensor é responsável por determinar a quantidade de voltas que o volante efetuou, e utilizando uma regra de três simples entre este valor e os limites de angulação, conhecidos a priori, é factível então calcular o esterçamento do veículo. Além disso, considerando apenas os dados do IMU, o ROS possui uma classe chamada *Quaternion*[9], que pertence ao pacote *tf2*[10], que ao obter os valores lidos pelos sensores é capaz de determinar a angulação correspondente.

O diagrama da Figura 36 apresenta uma visão completa do controlador odométrico e sua estrutura de funcionamento a partir das fontes dos dados. É possível observar que este está dividido em quatro fases. A primeira consiste em obter os dados do teclado ou sensores; A segunda fase é responsável por convertê-los em informações (velocidade linear e esterçamento do veículo), possibilitando o uso das equações odométricas; Já na terceira fase o cálculo é realizado, obtendo a nova posição e orientação do *CELiNA*, e transmitida ao nó *TransformController*; E, por fim, na última fase, *TransformController* executa os procedimentos necessários e atualiza a posição do robô no Gazebo.

3.6 Planejamento de testes

Os testes executados neste trabalho tiveram o propósito de avaliar a eficácia das abordagens assumidas no desenvolvimento do controle odométrico. Assim, foram realizados dois testes, quanto a fonte dos dados que antecedem as operações realizadas por *OdometryControl*, e a atualização no modelo simulável.

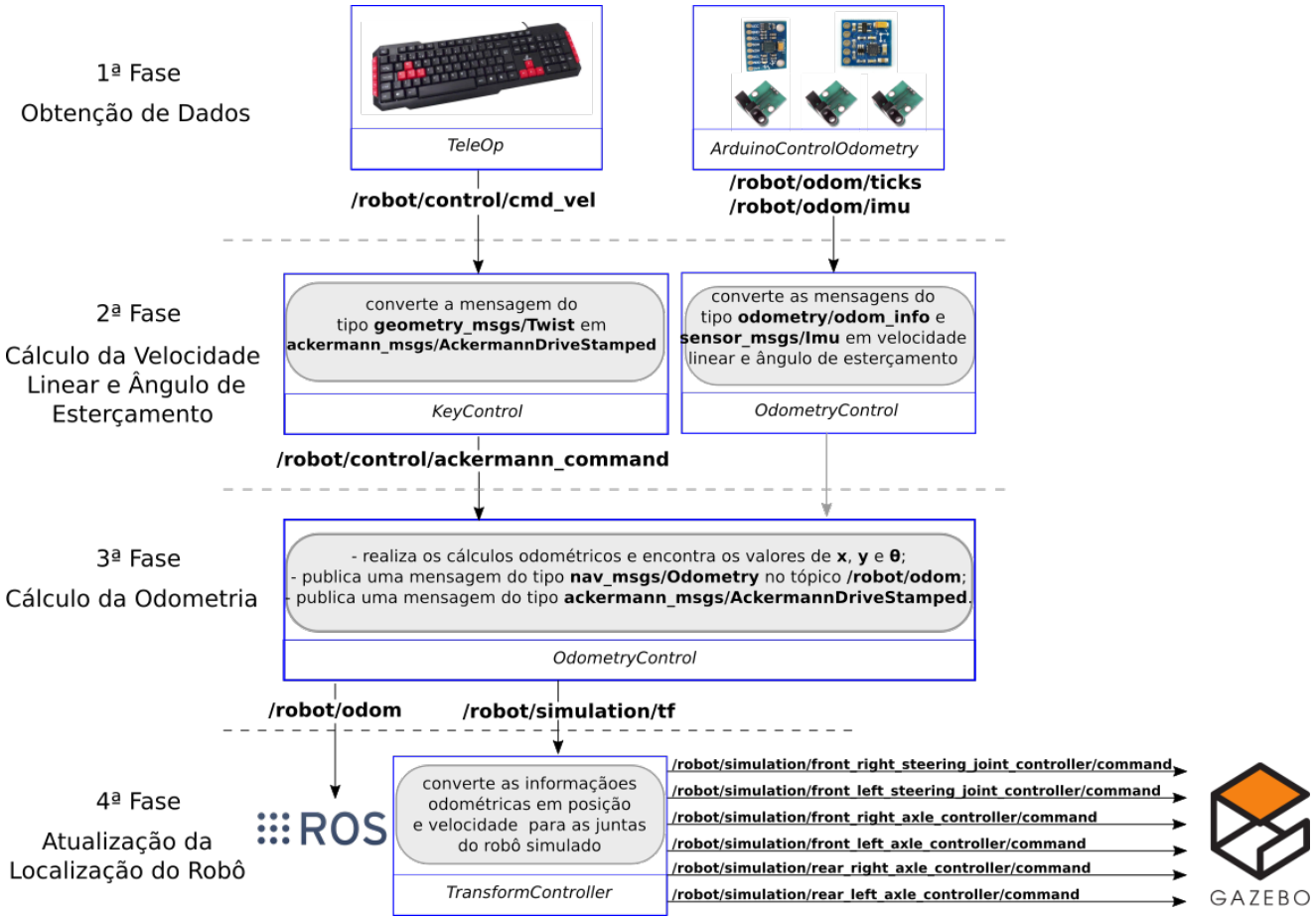
A metodologia de testes usadas é chamada de teste de caixa-cinza dinâmico, um método que utiliza conjuntamente os testes de caixa-branca, que tem o objetivo de verificar as unidades internas do *software*, ou seja, o código fonte, e o teste de caixa-preta, onde observa-se apenas a interface entre a entrada de um comando e sua saída do sistema (PATTON, 2001). O termo "dinâmico", significa que o teste foi realizado com o código rodando. Além disso, os testes desempenhados são classificados como testes-para-passar (*test-to-pass*) que, segundo Patton

[8] A documentação desse biblioteca está disponível no sítio: <http://wiki.ros.org/roscpp/Overview/Time>

[9] A documentação dessa classe pode ser vista no sítio: <http://docs.ros.org/jade/api/tf2/html/classtf2_1_1Quaternion.html>

[10] A documentação do pacote pode ser vista no sítio: <http://wiki.ros.org/tf2>

Figura 36 – Arquitetura do Controlador Odométrico



Fonte: o autor.

(2001), consistem na primeira parte do procedimento de avaliação de um novo produto, em que o objetivo é saber se este realiza o básico para qual foi projetado, assim, a meta não é forçar o programa para encontrar erros (chamado de teste-para-falhar, *test-to-fail*), e sim verificar se este realiza o que foi pretendido nas condições corretas. A seguir são descritos os procedimentos adotados em cada um.

3.6.1 Teste 1: Controle Remoto

- **Objetivo:** O objetivo deste primeiro teste é analisar se o controle remoto, desempenhado pelos nós *TeleOp*, *KeyControl*, *OdometryControl* e *TransformController*, nesta ordem, é capaz de mover o robô corretamente no ambiente simulável.
- **O que foi analisado:** Este teste analisou todo o circuito entre a obtenção de um comando de teclado e a atualização do robô no simulador.
- **Ferramentas utilizadas:** Para este teste foram utilizados as seguintes ferramentas:
  - Notebook DELL Inspiron 14R 5437;

  - Sistema Operacional Linux 16.04 LTS;
  - Intel® Core™ i5-4200U;
  - 8GB de memória RAM;
  - ROS Kinetic, versão 1.12.7;
  - Simulador Gazebo, versão 7.0.0.

- **Procedimentos:**

  1. *Iniciando os nós:* O primeiro procedimento consistiu em inicializar todos os nós necessários para a realização do teste, são ele: *TeleOp*; *KeyControl*; *OdometryControl*; *TransformController*; e, Gazebo.
  2. *Movendo o robô para frente e trás:* Utilizar o terminal para enviar os comandos *'w'* e *'s'*, com o objetivo de mover o robô para frente e trás, por uma determinada distância.
  3. *Movimentos conjuntos do robô:* Utilizar as teclas *'q'* e *'e'* para mover o robô utilizando manobras longitudinais (frente e ré) e laterais (direita e esquerda).

3.6.2 Teste 2: Odometria provida pelos sensores

- **Objetivo:** O objetivo deste teste é analisar o funcionamento do controle odometrico quando os dados usados são provenientes dos sensores, assim, os nós envolvidos são o *ArduinoOdometryControl*, *OdometryControl* e o *TransformController*.

- **O que foi analisado:** Este fluxo do funcionamento do controle odométrico é o mais crítico e importante entre todos, porque, como o *CELiNA* é um *UGV*, ou seja, um veículo autônomo, ele deverá guiar-se autonomamente, e tomar decisões baseados em informações sensoriais. Desta forma, neste teste foi analisado todo o circuito desde a captação das informações sensoriais pelo nó *ArduinoOdometryControl*, aos cálculos executados pelo *OdometryControl*, e a atualização do robô por *TransformController*.

- **Ferramentas utilizadas:** Para este teste foram utilizados as seguintes ferramentas:

  - Notebook DELL Inspiron 14R 5437;
  - Sistema Operacional Linux 16.04 LTS;
  - Intel® Core™ i5-4200U;
  - 8GB de memória RAM;
  - ROS Kinetic, versão 1.12.7;
  - Simulador Gazebo, versão 7.0.0;
  - Arduino Mega 2560;
  - Magnetômetro GY271;

  - Acelerômetro e Giroscópio MPU-2560;
  - Ferramenta rqt_gui[11];

- **Procedimentos:**

  1. *Movendo o robô pela ferramenta* rqt_gui*:* Usar a ferramenta *rqt_gui* para publicar informações no tópico *"/robot/odom/ticks*;
     - *Movimentos logintudinais*: utilizar a ferramenta para mover o robô para frente e trás;
     - *Movimentos conjuntos*: utilizar a ferramenta para fazer o robô realizar os movimentos de frente-esquerda e frente-direita;
  2. *Análise das informações odométricas:* Verificar se a odometria e movimentos do robô corresponde com o esperado.
     - *Movimentos longitudinais:* utilizar as informações publicadas pelo nó *ArduinoControlOdometry* para fazer o robô andar para frente trás;
     - *Movimentos conjuntos:* utilizar as informações publicadas pelo nó *ArduinoControlOdometry* para fazer o robô realizar os movimentos de frente-esquerda e frente-direita.

[11] A descrição dessa ferramenta pode ser encontrada no sítio: <http://wiki.ros.org/rqt_gui>

4 ANÁLISE DOS RESULTADOS

Neste trabalho foi desenvolvido um controlador odométrico para o UGV com direção *Ackermann* chamado *CELiNA*, construído na Universidade Estadual do Sudoeste da Bahia - UESB, campus de Vitória da Conquista. Para tanto, utilizou-se o *framework* de robótica conhecido como ROS - *Robotic Operating System* e o simulador Gazebo.

No capítulo anterior (Cap. 3) foi discutido todo o processo de desenvolvimento do *software* e ferramentas utilizadas, finalizando com a descrição dos testes aplicados para analisar a eficácia do projeto proposto. Como citado, esses seguiram a metodologia de teste de programas conhecida como caixa-cinza, que reune as características dos testes de caixa-preta e caixa-branca (PATTON, 2001).

A razão de se ter adotado essa metodologia é que o ROS possui uma estrutura bastante complexa de interação entre seus nós (unidades de programas), e na interação com o simulador utilizado. Por conta disso, analisar o controlador utilizando apenas a técnica de caixa-branca, que verifica o código-fonte, seria uma tarefa muito dispendiosa. Assim, a abordagem de caixa-preta é a melhor opção, uma vez que nesta não é necessário compreender o que ocorre no interior do sistema, e sim conferir se as entradas correspondem com as saídas desejadas. Entretanto, como foram implementados nós específicos para o controle do robô (*OdometryControl*, *ArduinoOdometryControl*, *TransformController*, *KeyControl* e *TeleOp*), assim, analisá-los do ponto de vista da implementação, ou seja, o código em si, é essencial para a conclusão deste trabalho. Desta forma, a escolha adotada torna-se bastante viável, pois a metodologia de caixa-cinza engloba a abstração e detalhismo da caixa-preta e baixa-branca, respectivamente (PATTON, 2001).

4.1 Teste 1: Odometria por Controle Remoto

Neste teste foram analisados o controle odométrico e simulação do robô, quando utilizados conjuntamente com o controle remoto, via interface de teclado. Para iniciá-lo é necessário executar o *script robot_remote_control.sh* localizado na pasta *robot*. Este programa foi implementado utilizando *Sheel Script* e automatiza a inicialização dos demais nós do ROS, executando em telas individuais[1] os seguintes comandos:

- **roscore:** Inicializa o nó *master* do ROS, responsável por coordenar a execução dos demais nós e encaminhar corretamente as mensagens do publicador (*puslisher*) aos assinantes(*subscriber*);

[1] Para executar esse script foi utilizado o comando *gnome_terminal*, cuja documentação pode ser vista no sítio: <https://help.gnome.org/users/gnome-terminal/stable/>

- **roslaunch *<pacote>* *<arquivo>*.launch:** O comando *roslaunch* executa um arquivo *launch* no ROS, sendo especificado através de parâmetros o pacote a qual pertence e seu nome. Os arquivos executados foram, nesta ordem: *keycontrol.launch*, do pacote *robot_teleop_control*; *odometry_control.launch*, do pacote *robot_odometry*; *robot_gazebo.launch* e *tf2_controller.launch*, do pacote *robot_simulator*.

Após a execução do *script* são abertas cinco janelas, cada uma executando um *launch*, na ordem que foi apresentado. Foi necessário colocar um tempo de 2 segundo entre o início de cada uma, para permitir que o sistema operacional interprete cada comando, caso contrário, um erro em sequência ocorreria e nenhum nó seria inicializado.

4.1.1 Movendo o robô para frente e trás

Para mover o robô para frente e trás é preciso digitar as teclas *'w'* e *'s'*. Por padrão a velocidade (v) estipulada vai de $0m/s$ a $2.5m/s$ quando uma entrada é recebida por *TeleOp* e *KeyControl*. Ao pressionar o botão por mais tempo, o valor chega a no máximo em $5m/s$. O movimento de ré é determinado pelo sinal de v, assim quando este é maior que 0 o robô irá para frente, e quando é menor para trás.

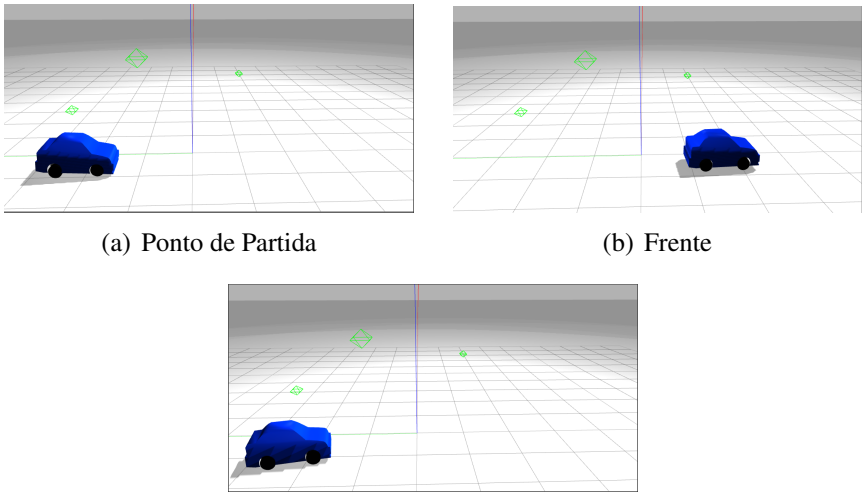
Neste procedimento foram realizadas as seguintes entradas: 50 entradas de *'w'* e 50 de *'s'*, com o único objetivo de verificar se o robô executa a manobra correta. A Figura 37(a) apresenta uma visão do *CELiNA* em seu ponto de partida, a Figura 37(b) depois da execução de *'w'*, e a Figura 37(c) após *'s'*. É possível notar então que tais movimentos foram realizados conforme esperado, no sentido que o robô foi para frente e trás, entretanto notou-se um desvio da orientação dele quando ao final de cada movimento. Esta desorientação ocorre em razão das rodas frontais não estarem sendo controladas, desta forma o robô acaba desviando em seu percurso.

4.1.2 Movimentos conjuntos do robô:

Os movimentos em conjunto reúnem as manobras de frente, ré, direita e esquerda, ou seja, movimentos longitudinais e laterais. Para usá-los digita-se as teclas *'q'* para frente-esquerda, e *'e'* para frente-direita. A Figura 2 (a) mostra o robô no ponto de partida e as Figura 2(b) e 2(c) após as entradas *'q'* e *'e'*, respectivamente. É perceptível que o robô executou os comandos corretamente, entretanto, durante o percurso foi notado que no esterçamento, as rodas frontais perpassavam os limites do robô, como se não estivessem ligadas à ele. A Figura 2 (d) mostra um exemplo do ocorrido.

As razões para tal erro não foram encontradas, aparentemente os centros de rotação das rodas não estão corretamente configurados, contudo, ao analisar o arquivo Xacro e as definições de momentos inerciais para rodas (semelhante ao tubo cilíndrico de paredes grossas,

Figura 37 – Movendo o *CELiNA* para 'frente' e 'trás'.



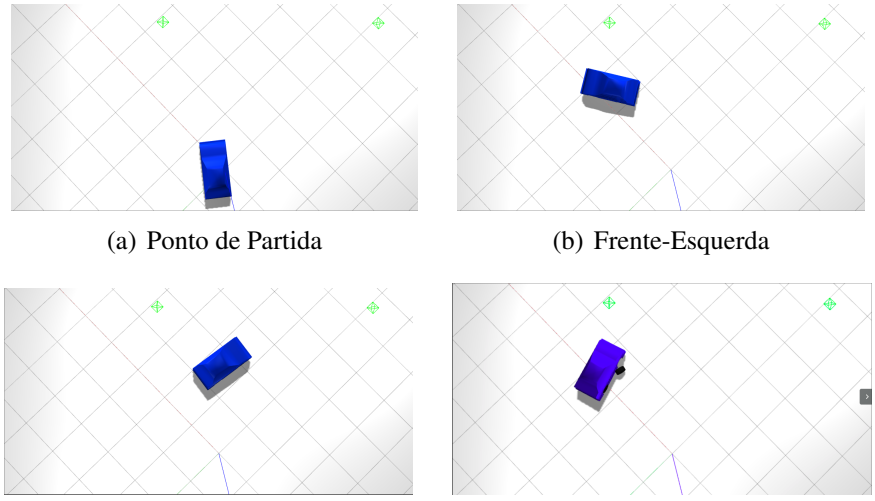
(a) Ponto de Partida (b) Frente

(c) Trás

Fonte: o autor.

ou *Thick-walled cylindrical tube*), nenhuma inconsistência foi encontrada, mesmo em comparações com outros trabalhos, como o RBCAR[2]. Um alternativa seria alterar as descrições das rodas do *CELiNA*, para um do Gazebo que possui 4 rodas, chamado de *cart_front_steer*, que diferencia-se do adotado atualmente por possuir juntas do tipo universal nas do tipo revolução (*front_left_steering_joint* e *front_right_steering_joint*). Esta abordagem pode ser testada em uma futura alteração do *CELiNA*.

Figura 38 – Movimentos conjuntos do *CELiNA*.



(a) Ponto de Partida (b) Frente-Esquerda

(c) Frente-Direita (d) Erro

Fonte: o autor.

[2] O RBCAR é um projeto aberto de um carro de golf que utiliza o ROS e Gazebo. A documentação do projeto pode ser encontrada no sítio: <http://wiki.ros.org/Robots/RBCAR>

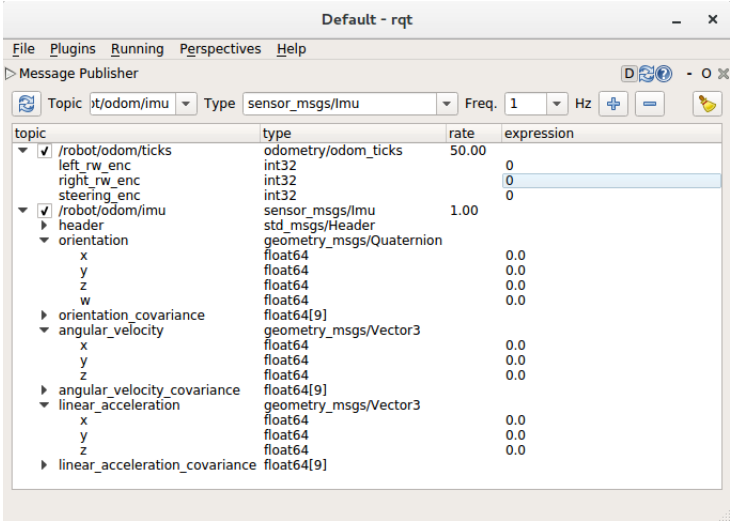
4.2 Teste 2: Odometria provida pelos sensores

O segundo teste verificou a corretude da interação entre o controle odométrico e as informações obtidas pelos sensores. Para tanto, optou-se a priori, pela simulação das informações fornecidadas pelo nó *ArduinoOdometryControl*, por meio da ferramenta *rqt_gui*. Esta abordagem foi assumida na primeira fase do teste, pois é uma forma mais prática de testar se as informações sensoriais movem o robô no ambiente simulado.

4.2.1 Movendo o robô pelo *rqt_gui*

A ferramenta *rqt_gui*, permite automatizar algumas funcionalidades do ROS, como publicar dados em tópicos, solicitar serviços, plotar gráficos, analisar a interação entre nós, etc. Nesse teste, ela foi usada com o proposito de simular o nó *ArduinoOdometryControl*, publicando mensagens no tópico */robot/odom/ticks* e */robot/odom/imu*. A Figura 39 mostra uma captura de tela do programa, quando configurado para ser um publicador dos tópicos citados. A Tabela 6 apresenta os dados que foram testados.

Figura 39 – *rqt_gui*



Fonte: o autor.

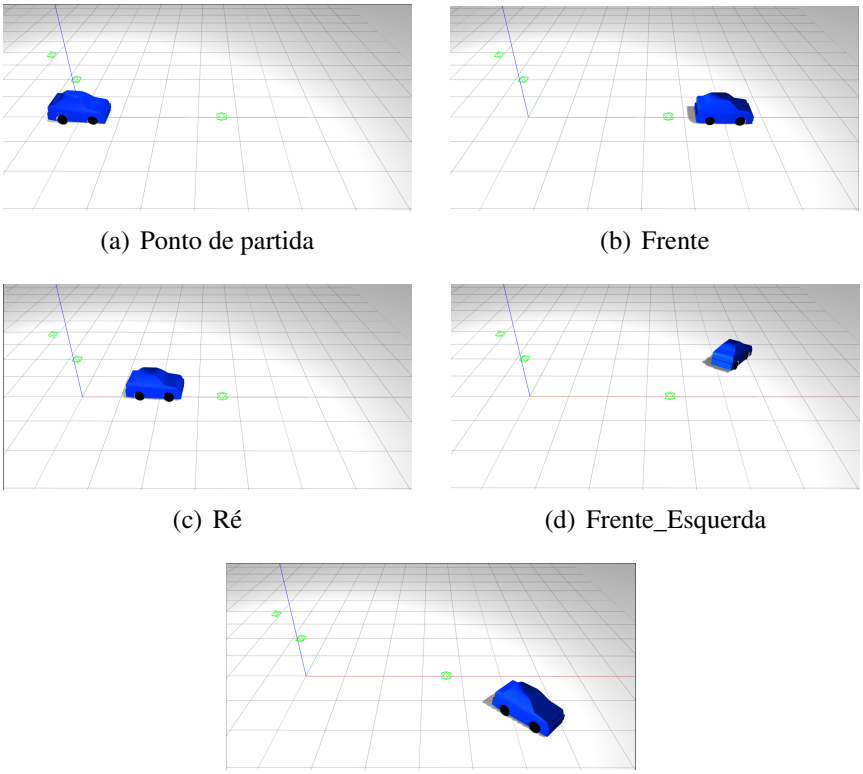
Tabela 6 – Dados aplicados ao rqt_gui

/robot/odom/ticks	frente	ré	frente_esquerda	frente_direita
left_rw_enc	20000	−10000	10000	10000
right_rw_enc	20000	−10000	10000	10000
steering_enc	0	0	1000	−1000

Fonte: O autor.

Os dados apresentados na Tabela 6 foram escolhidos com o propósito de replicar os movimentos realizados no primeiro teste, que foram: frente; ré; frente-esquerda; e, frente-direita. Ao aplica-lós é possível notar que os movimentos realizados pelo robô correspondem com o pretendido. O erro que foi observado na avaliação do primeiro teste persistiu, assim, ficou mais claro que o problema não está associado com a fonte de dados, e reforçou a possibilidade de inconsistência na rotação das rodas frontais quanto ao seus centros de rotação. A Figura 40 mostra os resultados dos movimentos do robô.

Figura 40 – Movendo o robô com o *rqt_gui*



(a) Ponto de partida (b) Frente

(c) Ré (d) Frente_Esquerda

(e) Frente_Direita

Fonte: o autor.

4.2.2 Análise das informações odométricas

O último teste realizado buscou verificar se a localização estimada pelo controlador odométrico condizia com o esperado. Nesta fase, optou-se por não simular o nó *ArduinoOdometryControl* como feito anteriormente, assim, os dados obtidos para uso nos cálculos odométricos foram publicados pelo arduino, ainda assim, algumas modificações tiveram que ser feitas na abordagem explanada neste trabalho.

Como o *CELiNA* ainda está em fase de desenvolvimento, nenhum sensor foi integrado ao robô físico, neste caso, os sensores de encoders não estavam prontos para a realização do presente teste, assim, os dados que seriam obtidos pelas leituras destes foram substituidos por

um valor fixo, fornecido ao controle odométrico de tal forma que a velocidade linear resultante fosse de $5m/s$. Esta segunda fase foi dividida em duas subpartes: movimentos longitudinais e movimentos conjuntos.

Para inicializar o experimento, foi implementado um outro *script* que automatiza a inicialização dos nós necessários. O arquivo, nomeado de *robot_odom_init.sh* encontra-se no diretório *robot*, e assim como o *shell* anterior, executa cada componente em uma janela individual do terminal linux, sendo eles: ROS *master*; *OdometryControl*; *ArduinoOdometryControl*; e, *rqt_gui*. Considerando que foram analisadas apenas as informações odométricas, o simulador Gazebo não foi utilizado, uma vez que já foi comprovado que as informações publicadas pelo controlador odométrico e assinada por *TransformController* de fato move o robô.

Outra modificação realizada foi a alteração do nó *OdometryControl* para imprimir em seu terminal a posição do robô a cada 5 segundos, usados aqui para análise dos cálculos. Para isso, utilizou-se a biblioteca de tempo como já citado e outra chamada *ros::Duration*[3], que permite definir durações em tempo para o *framework*, assim, garante-se melhor precisão nos dados colhidos, haja vista que este nó trabalha a uma frequência de *50Hz*.

4.2.2.1 Movimentos longitudinais

A primeira parte teve duração de 25 segundos. O *ArduinoOdometryControl* foi alterado para publicar informações apenas para o deslocamento linear do robô, ou seja, o ângulo de esterçamento mante-se com o valor 0 durante todo o procedimento. O propósito dessa escolha é isolar o cálculo odométrico dos desvios laterais, consequentemente, a distância percorrida pelo *CELiNA* deveria aproximar-se à multiplicação de sua velocidade (m/s) pelo tempo (s). A Tabela 7 apresenta os valores calculados pelo controle odométrico. É possível notar que a informação estimada aproximou-se bastante do ideal. Os erros foram respectivamente: $0.6m$; $0.7m$; $0.901m$; $1.301m$; e, $1.345m$. A análises destes retomam um tópico bastante discutido na literatura sobre a odometria, segundo autores como Siegwart, Nourbakhsh e Scaramuzza (2011), Borenstein et al. (1996), Muniandy e Muthusamy (2012), SECCHI (2012), entre outros, o modelo odométrico tende a acumular erro na medida que o deslocamento do robô aumenta, fato que ficou visível com os valores obtidos.

4.2.2.2 Movimentos conjuntos

No segundo procedimento, o robô realizou os movimentos de frente-esquerda e frente-direita, respectivamente. O teste teve duração de 35 segundos, sendo 15 para o primeiro movimentos, e o restante para o último. A Tabela 8 mostra os dados obtidos no experimento. Para conferir as informações e checar se estavam corretas, as fórmulas discutidas na sessão 2.4.1.2 foram aplicadas manualmente em um processo de *"debuggin"* dos cálculos realizados, não

[3] A documentação desta biblioteca pode ser encontrada no sítio: <http://docs.ros.org/diamondback/api/rostime/html/classros_1_1Duration.html>

Tabela 7 – Dados obtidos na 1° fase de testes dos valores odométricos

$1° fase$	5s	10s	15s	20s	25s
x (m)	24.400	49.300	74.099	98.699	123.655
y (m)	0	0	0	0	0
θ (graus)	0	0	0	0	0
velocidade linear (m/s)	5	5	5	5	5

Fonte: O autor.

havendo nenhuma diferença significante, apenas valores aproximados em 3 casas decimais para a angulação, por exemplo, que modificaram pouca coisa no resultado final.

Tabela 8 – Dados obtidos na 2° fase de testes dos valores odométricos

$2° fase$	5s	10s	15s	20s	25s	30s	35s
x (m)	24.278	48.391	71.071	93.304	117.239	142.153	166.866
y (m)	2.116	8.588	19.264	30.408	37.510	40.280	38.637
θ (graus)	9.920645	20.085239	30.290636	21.630280	11.465649	1.260412	−8.822879
velocidade linear (m/s)	5	5	5	5	5	5	5

Fonte: O autor.

5 CONCLUSÃO

Os robôs estão cada vez mais presentes no dia-a-dia da sociedade. Um dos principais motivos pela alta popularidade é que estes são mais precisos e produtivos na realização de atividades como, limpeza, segurança, utulidades médicas, mobilidade, etc. A exemplo disso são os veículos autônomos, robôs móveis que são dotados de inteligência e são capazes de locomover em diferentes ambientes adequando-se às leis específicas para transito, bem como, priorizando sempre a vida humana.

A mobilidade, característica que distingui a classe de robôs móveis das demais, traz consigo diversos desafios para a àrea, uma vez que tais dispositivos são projetados para lidarem com diferentes situação que poderão ocorrem enquanto desempenha suas atividades. Entre esses desafios encontra-se a localização, que nada mais é que dotar o robô da capacidade de se localizar no ambiente em que está inserido.

Existem diferentes técnicas que objetivam solucionar essa problemática, este trabalho estudou e desenvolveu um controle para localização de um veículo autônomo, chamado de *CELiNA*, desenvolvido pela Universidade Estadual do Sudoeste da Bahia - UESB. O robô é um UGV - *Unmanned Ground Vehicles* (Veículo Terrestre Não Tripulado), que possui direção *Ackermann*, e a técnica empregada para solucionar um dos problemas da localização (mais especificamente a localização local) é chamada de odometria. A odométria é uma técnica bastante empregada, pois é simples e eficiente, apesar de que acomula erros em função do deslocamento do dispositivo. No presente proposto, utilizou-se também o *framework* para robótica conhecido como ROS (*Robotic Operating System*), e o simulador Gazebo.

A abordagem de simulação advém da necessidade de otimizar o processo inicial de desenvolvimento de robôs, reduzindo custos, e permitindo aos pesquisadores um controle maior sobre o ambiente em que seus algoritmos são testados. E essa praticidade ficou evidente no trabalho, sendo que diversos teste foram repetidos nas mesmas condições, sem a necessidade do robô físico, e culminaram em uma experiência proveitosa e eficiente.

Apesar de alguns erros encontrados, alguns que a própria literatura aponta, o controlador desenvolvido está apto à integração com o *CELiNA* real, no entanto, é necessário realizar algumas intervenções pontuais para melhorar ainda mais a eficiência e utilidade do programa para o restante do projeto.

5.1 Trabalhos futuros

Dentre os trabalhos futuros encontram-se:

- Correção da simulação do esterçamento do veículo;
- Calibração dos parâmetros cinemáticos do *CELiNA* utilizando a técnica *UMBMark* para veículos *Ackermann*;
- Implementação de um modelo de correção de erros sistemáticos e não sistemáticos online;
- Integração dos sensores necessários no robô físico;
- Montagem da arquitetura de controle no robô físico;
- Modelagem de um chassi mais semelhante ao *CELiNA*;
- Implementação de outros sensores no robô, como câmera, sensores de distância, etc;
- Teste utilizando as informações odométricas puramente em algoritmos de navegação.

REFERÊNCIAS

ALAMI, R. et al. An architecture for autonomy. *The International Journal of Robotics Research*, SAGE Publications, v. 17, n. 4, p. 315–337, 1998. Acesso em: 31 ago. 2016. Disponível em: <http://ijr.sagepub.com/content/17/4/315.short>. Citado na página 28.

ALBUS, J. S. A theory of intelligent systems. In: IEEE. *Intelligent Control, 1990. Proceedings., 5th IEEE International Symposium on*. 1990. p. 866–875. Acesso em: 31 ago. 2016. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=128559>. Citado na página 28.

ALLENDER, D. et al. Autonomous golf cart navigation using ros. 2011. Acesso em: 06 mai. 2017. Disponível em: <http://users.csc.calpoly.edu/~clupo/teaching/Capstone/pastwork/avid_whitepaper.pdf>. Citado na página 53.

AMAZON. *Amazon Prime Air*. 2016. Figura. Disponível em: <https://www.amazon.com/b?node=8037720011>. Citado na página 26.

ANTONELLI, G.; CHIAVERINI, S.; FUSCO, G. A calibration method for odometry of mobile robots based on the least-squares technique: theory and experimental validation. *IEEE Transactions on Robotics*, IEEE, v. 21, n. 5, p. 994–1004, 2005. Acesso em 07 set. 2016. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1512356>. Citado 2 vezes nas páginas 18 e 20.

ARAÚJO, A. et al. Integrating arduino-based educational mobile robots in ros. In: IEEE. *Autonomous Robot Systems (Robotica), 2013 13th International Conference on*. 2013. p. 1–6. Acesso em: 13 jun. 2017. Disponível em: <http://ieeexplore.ieee.org/abstract/document/6623520/>. Citado na página 66.

ARDUINO. *MPU-6050 Accelerometer + Gyro*. 2017. Figura. Disponível em: <https://playground.arduino.cc/Main/MPU-6050>. Citado na página 67.

BAJRACHARYA, M.; MAIMONE, M. W.; HELMICK, D. Autonomy for mars rovers: Past, present, and future. *Computer*, IEEE, v. 41, n. 12, p. 44–50, 2008. Acesso em: 26 jul. 2016. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4712499>. Citado na página 17.

BEZERRA, C. G. *Localização de um robô móvel usando odometria e marcos naturais*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Norte, 2004. Acesso em: 13 set. 2016. Disponível em: <https://repositorio.ufrn.br/jspui/bitstream/123456789/15411/1/ClauberGB.pdf>. Citado 4 vezes nas páginas 20, 36, 37 e 43.

BLACK, T. *Robo-Gloves, Meat Hooks Help Ease Human-Machine Teamwork*. Robo-Gloves, Meat Hooks Help Ease Human-Machine Teamwork, 2016. Figura. Disponível em: <http://www.bloomberg.com/news/articles/2013-12-26/gm-robo-glove-to-meat-hook-smooth-human-machine-teamwork>. Citado na página 23.

BORENSTEIN, J. et al. *Where am I? Sensors and methods for mobile robot positioning*. University of Michigan, 1996. Acesso em: 07 set. 2016. Disponível em: <http:

//citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.444.5014&rep=rep1&type=pdf>. Citado 8 vezes nas páginas 18, 20, 34, 35, 39, 40, 44 e 78.

BORENSTEIN, J. et al. *Mobile robot positioning-sensors and techniques*. [S.l.], 1997. Acesso em: 06 set. 2016. Disponível em: <http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA422844>. Citado 2 vezes nas páginas 19 e 32.

BORENSTEIN, J.; FENG, L. Correction of systematic odometry errors in mobile robots. In: IEEE. *Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*. 1995. v. 3, p. 569–574. Acesso em: 10 ago. 2016. Disponível em: <http://ieeexplore.ieee.org/document/525942/>. Citado 7 vezes nas páginas 18, 19, 20, 35, 36, 43 e 44.

BORENSTEIN, J.; FENG, L. Umbmark: A benchmark test for measuring odometry errors in mobile robots. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *Photonics East'95*. 1995. p. 113–124. Acesso em: 29 mar. 2017. Disponível em: <http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=1009174>. Citado 3 vezes nas páginas 44, 93 e 94.

BORENSTEIN, J.; FENG, L. Measurement and correction of systematic odometry errors in mobile robots. *IEEE Transactions on robotics and automation*, IEEE, v. 12, n. 6, p. 869–880, 1996. Acesso em: 30 mar. 2017. Disponível em: <http://ieeexplore.ieee.org/abstract/document/544770/>. Citado 5 vezes nas páginas 68, 93, 94, 95 e 96.

BOTLAND. *HMC5883L 3-osiowy magnetometr cyfrowy I2C 3,3V / 5V - moduł GY-271*. 2017. Figura. Disponível em: <https://botland.com.pl/magnetometry/2722-hmc5883l-3-osiowy-magnetometr-cyfrowy-i2c-33v-5v-modul-gy-271.html>. Citado na página 67.

BREAZEAL, C. *JIBO, The World's First Social Robot for the Home*. 2016. Figura. Disponível em: <https://www.indiegogo.com/projects/jibo-the-world-s-first-social-robot-for-the-home#/>. Citado na página 24.

BRUMSON, B. *Robotics in Security and Military Applications*. 2011. Acesso em: 22 ago. 2016. Disponível em: <http://www.robotics.org/content-detail.cfm/Industrial-Robotics-Industry-Insights/Robotics-in-Security-and-Military-Applications/content_id/3112>. Citado na página 24.

CAÑAS, J. M.; MARTN, L.; VEGA, J. Innovating in robotics education with gazebo simulator and jderobot framework. In: *XII Congreso Universitario de Innovacion Educativa en las Ensenanzas Tecnicas*. [s.n.], 2014. Acesso em: 12 mar. 2017. Disponível em: <https://gsyc.urjc.es/jmplaza/papers/cuieet2014.pdf>. Citado na página 51.

CENSI, A. et al. Simultaneous calibration of odometry and sensor parameters for mobile robots. *IEEE Transactions on Robotics*, IEEE, v. 29, n. 2, p. 475–492, 2013. Acesso em: 12 set. 2016. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6482657>. Citado 2 vezes nas páginas 18 e 68.

CHENAVIER, F.; CROWLEY, J. L. Position estimation for a mobile robot using vision and odometry. In: IEEE. *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*. 1992. p. 2588–2593. Acesso em: 10 ago. 2016. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=220052>. Citado na página 18.

CHONG, K. S.; KLEEMAN, L. Accurate odometry and error modelling for a mobile robot. In: IEEE. *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*. 1997. v. 4, p. 2783–2788. Acesso em: 11 ago. 2016. Disponível em: <http://ieeexplore.ieee.org/document/606708/>. Citado 2 vezes nas páginas 18 e 43.

DELLAERT, F. et al. Monte carlo localization for mobile robots. In: IEEE. *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. [S.l.], 1999. v. 2, p. 1322–1328. Citado na página 20.

DUDEK, G.; JENKIN, M. *Computational principles of mobile robotics*. [S.l.]: Cambridge university press, 2010. Citado 3 vezes nas páginas 22, 26 e 31.

EDGERTON, R. *Virtual Sojourner*. 2016. Figura. Disponível em: <http://www-k12.atmos.washington.edu/k12/modules/Virtual_Sojourner/>. Citado na página 32.

FATOR, P. B. *Convertido com sucesso: pulverizador da Dürr economiza tempo e dinheirovelocímetro*. 2016. Figura. Disponível em: <http://www.revistafatorbrasil.com.br/ver_noticia.php?not=169794>. Citado na página 23.

FERNÁNDEZ, E. et al. *Learning ROS for Robotics Programming*. [S.l.]: Packt Publishing Ltd, 2015. Acesso em: 24 mai. 2017. Citado 2 vezes nas páginas 51 e 59.

FOX, D.; BURGARD, W.; THRUN, S. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, v. 11, p. 391–427, 1999. Acesso em: 12 set. 2016. Disponível em: <http://www.jair.org/media/616/live-616-1819-jair.pdf>. Citado 2 vezes nas páginas 20 e 36.

GATES, B. A robot in every home. *Scientific American*, Nature Publishing Group, v. 296, n. 1, p. 58–65, 2007. Acesso em: 23 ago. 2016. Disponível em: <http://www.master-ris.unina.it/attachments/058_A_Robot_in_Every_Home.pdf>. Citado 2 vezes nas páginas 22 e 26.

GAZEBO. *URDF in Gazebo*. 2017. Acesso em: 31 ago. 2017. Disponível em: <http://gazebosim.org/tutorials/?tut=ros_urdf>. Citado na página 58.

GE, S. S. *Autonomous mobile robots: sensing, control, decision making and applications*. [S.l.]: CRC press, 2006. v. 22. Citado 2 vezes nas páginas 16 e 18.

GONZALEZ, R. et al. Combined visual odometry and visual compass for off-road mobile robots localization. *Robotica*, Cambridge Univ Press, v. 30, n. 06, p. 865–878, 2012. Acesso em: 07 set. 2016. Disponível em: <http://journals.cambridge.org/abstract_S026357471100110X>. Citado 5 vezes nas páginas 18, 34, 35, 36 e 37.

GUARDIAN, T. *Meet Ropits, the Japanese robot car that drives itself*. 2016. Figura. Disponível em: <https://www.theguardian.com/artanddesign/architecture-design-blog/2013/mar/27/driverless-robot-car-elderly-disabled-mobility>. Citado na página 26.

GUPTA, A.; DIVEKAR, R.; AGRAWAL, M. Autonomous parallel parking system for ackerman steering four wheelers. In: IEEE. *Computational Intelligence and Computing Research (ICCIC), 2010 IEEE International Conference on*. 2010. p. 1–6. Acesso em: 22 mar. 2017. Disponível em: <http://ieeexplore.ieee.org/abstract/document/5705869/>. Citado na página 39.

HRBÁČEK, J.; RIPEL, T.; KREJSA, J. Ackermann mobile robot chassis with independent rear wheel drives. In: IEEE. *Power Electronics and Motion Control Conference (EPE/PEMC), 2010 14th International*. 2010. p. T5–46. Acesso em: 22 mar. 2017. Disponível em: <http://ieeexplore.ieee.org/abstract/document/5606853/>. Citado na página 39.

ICMC, L. de R. M. *Projeto CaRINA 2*. 2016. Figura. Disponível em: <http://lrm.icmc.usp.br/web/index.php?n=Port.ProjCarina2Info>. Citado na página 17.

INDUSTRIAL, B. *Processos de Fabricação – Robôs Industriais*. 2016. Figura. Disponível em: <http://www.boxindustrial.com.br/news/aula-20-processos-de-fabricacao-robos-industriais/>. Citado na página 23.

INSTITUTE, C. M. U. T. R. *Field Robotics*. 2016. Figura. Disponível em: <https://www.ri.cmu.edu/research_guide/field_robotics.html>. Citado na página 17.

INTHIAM, J.; DEELERTPAIBOON, C. Self-localization and navigation of holonomic mobile robot using omni-directional wheel odometry. In: IEEE. *TENCON 2014-2014 IEEE Region 10 Conference*. 2014. p. 1–5. Acesso em: 12 ago. 2016. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7022281>. Citado na página 18.

IROBOT. *iRobot Mirra: Pool Cleaning Robot*. 2016. Figura. Disponível em: <http://www.irobot.com/For-the-Home/Outdoor-Maintenance/Mirra.aspx>. Citado na página 24.

ISO. *Manipulating Industrial Robots–Vocabulary*. [S.l.], 1994. Acesso em: 17 ago. 2016. Disponível em: <https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-2:v1:en>. Citado na página 22.

JEFFERSON, A. *ASIMO by Honda: The World's Most Advanced Humanoid Robot*. 2016. Figura. Disponível em: <http://www.proctorhonda.com/blog/2013/august/6/asimo-the-honda-robot.htm>. Citado na página 24.

JENSFELT, P. *Approaches to mobile robot localization in indoor environments*. Tese (Doutorado), 2001. Citado 6 vezes nas páginas 18, 20, 34, 35, 37 e 90.

JONES, J. L.; FLYNN, A. M. *Mobile robots: inspiration to implementation*. [S.l.]: AK Peters, Ltd., 1993. Citado na página 32.

JOSEPH, L. *Mastering ROS for robotics programming*. [S.l.]: Packt Publishing Ltd, 2015. Citado 9 vezes nas páginas 47, 48, 49, 50, 51, 52, 59, 60 e 65.

JUAN, S. H.; COTARELO, F. H. Autonomous navigation framework for a car-like robot. 2015. Acesso em: 06 mai. 2017. Disponível em: <http://www.iri.upc.edu/files/scidoc/1658-Autonomous-navigation-framework-for-a-car-like-robot.pdf>. Citado na página 53.

JUNG, C. R. et al. Computação embarcada: Projeto e implementação de veículos autônomos inteligentes. *Anais do CSBC*, v. 5, p. 1358–1406, 2005. Acesso em: 04 set. 2016. Disponível em: <http://s3.amazonaws.com/academia.edu.documents/39620034/jai05-veiculos-inteligentes.pdf?AWSAccessKeyId=AKIAJ56TQJRTWSMTNPEA&Expires=1473013355&Signature=gzu8SMjl8BpomEvRDM7GTiQ1MMs=&response-content-disposition=inline;filename=Computacao_embarcada_Projeto_e_implement.pdf>. Citado na página 30.

KALTENEGGER, E.; BINDER, B.; BADER, M. Controlling and tracking an unmanned ground vehicle with ackermanndrive. 2016. Acesso em: 06 mai. 2017. Disponível em: <https://info.tuwien.ac.at/mbader/publications/downloads/kaltenegger2016a.pdf>. Citado na página 53.

KANNIAH, J.; ERCAN, M. F.; CALDERON, C. A. A. *Practical robot design: game playing robots*. [S.l.]: CRC Press, 2013. Citado 6 vezes nas páginas 16, 24, 27, 31, 37 e 42.

KELLY, A. *Mobile Robotics: Mathematics, Models, and Methods*. [S.l.]: Cambridge University Press, 2013. Citado 2 vezes nas páginas 30 e 32.

LEE, K. et al. Odometry calibration of a car-like mobile robot. In: IEEE. *Control, Automation and Systems, 2007. ICCAS'07. International Conference on*. 2007. p. 684–689. Acesso em: 4 abr. 2017. Disponível em: <http://ieeexplore.ieee.org/abstract/document/4406986/>. Citado 3 vezes nas páginas 44, 45 e 46.

LEE, K.; CHUNG, W.; YOO, K. Kinematic parameter calibration of a car-like mobile robot to improve odometry accuracy. *Mechatronics*, Elsevier, v. 20, n. 5, p. 582–595, 2010. Acesso em: 15 jun. 2017. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0957415810001042>. Citado na página 68.

LEGO. *EV3RSTORM*. 2016. Figura. Disponível em: <http://www.lego.com/en-us/mindstorms/build-a-robot/ev3rstorm>. Citado na página 25.

MARCHIONI, H. *Carro autônomo realiza "Volta da Ufes"em Goiabeiras*. 2016. Figura. Disponível em: <http://www.ufes.br/conteudo/carro-aut-C3-B4nomo-realiza-volta-da-ufes-em-goiabeiras>. Citado na página 17.

MATSUBARA, V. *Carro Autônomo do Google Causa Primeiro Acidente na Califórnia*. 2016. Figura. Disponível em: <http://quatrorodas.abril.com.br/materia/carro-autonomo-do-google-causa-primeiro-acidente-california>. Citado na página 17.

MCKERROW, P. J.; RATNER, D. Calibrating a 4-wheel mobile robot. In: IEEE. *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*. 2002. v. 1, p. 859–864. Acesso em: 01 abr. 2017. Disponível em: <http://ieeexplore.ieee.org/document/1041498/>. Citado na página 93.

MORAES, R. F. de L. *NASA revela o rover que estará na missão Marte 2020*. 2016. Figura. Disponível em: <http://www.tecmundo.com.br/nasa/59932-nasa-escolhe-instrumentos-missao-marte-em-2020.htm>. Citado na página 26.

MOREIRA, E. *HDMS, um robô que abre garrafas e desarma bombas*. 2016. Figura. Disponível em: <http://www.oeduardomoreira.com.br/hdms-um-robo-que-abre-garrafas-e-desarma-bombas/>. Citado na página 25.

MUNIANDY, M.; MUTHUSAMY, K. An innovative design to improve systematic odometry error in non-holonomic wheeled mobile robots. *Procedia Engineering*, Elsevier, v. 41, p. 436–442, 2012. Acesso em: 22 mar. 2017. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877705812025957>. Citado 3 vezes nas páginas 38, 68 e 78.

MURPHY, R. *Introduction to AI robotics*. [S.l.]: MIT press, 2000. Citado 7 vezes nas páginas 22, 27, 28, 29, 30, 32 e 33.

NASA. *Spirit (rover)*. 2016. Figura. Disponível em: <https://photojournal.jpl.nasa.gov/catalog/PIA04413>. Citado na página 17.

NASA. *Urbie, the Urban Robot*. 2016. Figura. Disponível em: <http://www.nasa.gov/vision/earth/technologies/urbie.html>. Citado na página 33.

NIKU, S. B. *Introdução à robótica–análise, controle, aplicações*. 1st edition. ed. São Paulo: LTC, 2013. Citado 5 vezes nas páginas 22, 23, 24, 28 e 31.

NISTÉR, D.; NARODITSKY, O.; BERGEN, J. Visual odometry for ground vehicle applications. *Journal of Field Robotics*, Wiley Online Library, v. 23, n. 1, p. 3–20, 2006. Acesso em: 12 ago. 2016. Disponível em: <http://onlinelibrary.wiley.com/doi/10.1002/rob.20103/abstract>. Citado 2 vezes nas páginas 20 e 36.

NOURANI-VATANI, N.; ROBERTS, J.; SRINIVASAN, M. V. Practical visual odometry for car-like vehicles. In: IEEE. *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. 2009. p. 3551–3557. Acesso em: 12 ago. 2016. Disponível em: <http://ieeexplore.ieee.org/document/5152403/>. Citado 2 vezes nas páginas 18 e 36.

ODEDRA, S.; PRIOR, S. D.; KARAMANOGLU, M. Investigating the mobility of unmanned ground vehicles. In: INTERNATIONAL CONFERENCE ON MANUFACTURING AND ENGINEERING SYSTEMS. *International Conference on Manufacturing and Engineering Systems. Proceedings*. 2009. p. 380–385. Acesso em: 07 set. 2016. Disponível em: <http://eprints.mdx.ac.uk/3863/1/Odedra,_S_-_MES_2009.pdf>. Citado 2 vezes nas páginas 17 e 33.

O'KANE, J. M. Global localization using odometry. In: IEEE. *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* 2006. p. 37–42. Acesso em: 11 set. 2016. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1641158>. Citado 2 vezes nas páginas 18 e 34.

O'KANE, J. M. *A gentle introduction to ros*. Jason M. O'Kane, 2014. Acesso em: 12 de mar. 2017. Disponível em: <https://cse.sc.edu/~jokane/agitr/agitr-small.pdf>. Citado 2 vezes nas páginas 49 e 59.

OLSON, C. F. Probabilistic self-localization for mobile robots. *IEEE Transactions on Robotics and Automation*, IEEE, v. 16, n. 1, p. 55–66, 2000. Citado 2 vezes nas páginas 20 e 56.

OLSON, E. A primer on odometry and motor control. 2004. Acesso em: 5 mar. 2017. Disponível em: <http://web.mit.edu/cathywu/6.186/2006/doc/odomtutorial/src/odomtutorial.pdf>. Citado 6 vezes nas páginas 36, 37, 42, 66, 90 e 92.

PATTON, R. *Software testing*. Sams publishing, 2001. Acesso em: 15 jun. 2017. Disponível em: <http://www.rehancodes.com/files/ron-patton-software-testing-1st-edition.pdf>. Citado 3 vezes nas páginas 69, 70 e 73.

PEREZ, M. *Robôs voadores: conheça um pouco sobre a tecnologia dos drones militares*. 2016. Figura. Disponível em: <http://canaltech.com.br/analise/seguranca/Robos-voadores-conheca-um-pouco-sobre-a-tecnologia-dos-drones-militares/>. Citado na página 25.

PHARMA, M. C. *A bright future for pharma robots*. 2016. Figura. Disponível em: <http://www.manufacturingchemist.com/news/article_page/A_bright_future_for_pharma_robots/104419>. Citado na página 23.

PLEO. *PLEO rb Basic Pack - Green*. 2016. Figura. Disponível em: <http://www.pleoworld.com/pleo_rb/eng/product.php?&c1id=1&id=4>. Citado na página 25.

QUIGLEY, M. et al. Ros: an open-source robot operating system. In: KOBE, JAPAN. *ICRA workshop on open source software*. 2009. v. 3, n. 3.2, p. 5. Acesso em: 11 ago. 2016. Disponível em: <https://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf>. Citado 4 vezes nas páginas 19, 21, 47 e 48.

REINSTEIN, M.; KUBELKA, V.; ZIMMERMANN, K. Terrain adaptive odometry for mobile skid-steer robots. In: IEEE. *Robotics and automation (icra), 2013 ieee international conference on*. 2013. p. 4706–4711. Acesso em: 11 ago. 2016. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6631247>. Citado na página 18.

RILL, G. Vehicle dynamics: fundamentals and modeling aspects. *Fachhochschule, Regensburg*, 2007. Acesso em: 28 mar. 2017. Disponível em: <http://ftp.demec.ufpr.br/disciplinas/EngMec_NOTURNO/TM355/Prof_Jorge_Erthal/Matlab/Dinamica_de_Veiculos/Short_Course_Brasil_2007.pdf>. Citado na página 42.

ROMANO, V. F.; DUTRA, M. S. Introdução a robótica industrial. *Robótica Industrial–Aplicação na Indústria de Manufaturas e de Processos*, v. 1, 2002. Acesso em: 25 ago. 2016. Disponível em: <http://www.soldaautomatica.com.br/index_arquivos/Arquivos/PDF16-UNICAMP-Cap1-Indroduç~aoàRobóticaIndustrial.pdf>. Citado na página 28.

ROMERO, R. A. F. et al. *Robótica Móvel*. 1st edition. ed. Rio de Janeiro: LTC, 2014. Citado 12 vezes nas páginas 17, 20, 22, 24, 27, 28, 29, 30, 34, 35, 36 e 37.

SECCHI, H. A. Uma introdução aos robôs móveis. *Instituto de Automática–INAUT. Universidade Nacional de San Juan–UNSJ–Argentina*, 2012. Acesso em: 26 jul. 2016. Disponível em: <http://www.obr.org.br/wp-content/uploads/2013/04/Uma_Introducao_aos_Robos_Moveis.pdf>. Citado 2 vezes nas páginas 16 e 78.

SIEGWART, R.; NOURBAKHSH, I. R.; SCARAMUZZA, D. *Introduction to autonomous mobile robots*. [S.l.]: MIT press, 2011. Citado 12 vezes nas páginas 16, 17, 18, 20, 23, 31, 34, 35, 36, 37, 39 e 78.

SIMPSON, J.; JACOBSEN, C. L.; JADUD, M. C. Mobile robot control. *Communicating Process Architectures*, p. 225, 2006. Citado 2 vezes nas páginas 28 e 29.

SOLUTION, B. R. *Sistema de Paletização Automático*. 2016. Figura. Disponível em: <http://www.boxindustrial.com.br/news/aula-20-processos-de-fabricacao-robos-industriais/>. Citado na página 23.

SOLUTIONS, A. K. *Robotics*. Hingham, Massachusetts: Infinity Science Press LLC, 2007. Citado 3 vezes nas páginas 22, 27 e 28.

SOMAI. *Robô NAO: um mundo de possibilidades*. 2016. Figura. Disponível em: <http://www.somai.com.br/robo-nao/>. Citado na página 25.

SONY, A. *Sony Aibo*. 2016. Figura. Disponível em: <http://www.sony-aibo.com/>. Citado na página 25.

STARANOWICZ, A.; MARIOTTINI, G. L. A survey and comparison of commercial and open-source robotic simulator software. In: ACM. *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments*. 2011. p. 56. Acesso em: 15 mar. 2017. Disponível em: <http://dl.acm.org/citation.cfm?id=2141689>. Citado na página 51.

STäUBLI. *TX40 6-axis industrial robot*. 2016. Figura. Disponível em: <http://www.staubli.com/br/robotics/braco-robotico/robos-de-baixa-carga/tx40/>. Citado na página 31.

SURGICAL, T. *TSolution One*. 2016. Figura. Disponível em: <http://thinksurgical.com/tsolution-one#tcat>. Citado na página 24.

TECHNOLOGY, A. *Pioneer 3 - DX*. 2016. Figura. Disponível em: <http://www.mobilerobots.com/Libraries/Downloads/Pioneer3DX-P3DX-RevA.sflb.ashx>. Citado na página 26.

TÖLGYESSY, M.; HUBINSKỲ, P. The kinect sensor in robotics education. In: *Proceedings of 2nd International Conference on Robotics in Education*. [s.n.], 2011. p. 143–146. Acesso em: 24 ago. 2016. Disponível em: <http://www.innoc.at/fileadmin/user_upload/_temp_/RiE/Proceedings/69.pdf>. Citado 2 vezes nas páginas 27 e 28.

UFMG, C. C. e R. *Fotos Carro Autônomo CADU*. 2016. Figura. Disponível em: <http://coro.cpdee.ufmg.br/index.php/fotos-e-videos/44-fotos/135-fotos-carro-autonomo-cadu-outubro-de-2010>. Citado na página 17.

WILLIAMS, M. *The Drive for Autonomous Vehicles: The DARPA Grand Challenge*. 2016. Figura. Disponível em: <https://herox.com/news/159-the-drive-for-autonomous-vehicles-the-darpa-grand>. Citado na página 17.

WOLF, D. F. et al. *Robótica móvel inteligente: Da simulação às aplicações no mundo real*. Brasil, 2009. Acesso em: 17 ago. 2016. Disponível em: <http://inct-sec.icmc.usp.br/actrep/sites/default/files/highlights/Tutorial-JAI.pdf>. Citado 11 vezes nas páginas 16, 17, 18, 19, 22, 25, 26, 32, 34, 47 e 53.

WOODEN, D. et al. Autonomous navigation for bigdog. In: IEEE. *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. 2010. p. 4736–4741. Acesso em: 22 ago. 2016. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5509226>. Citado na página 25.

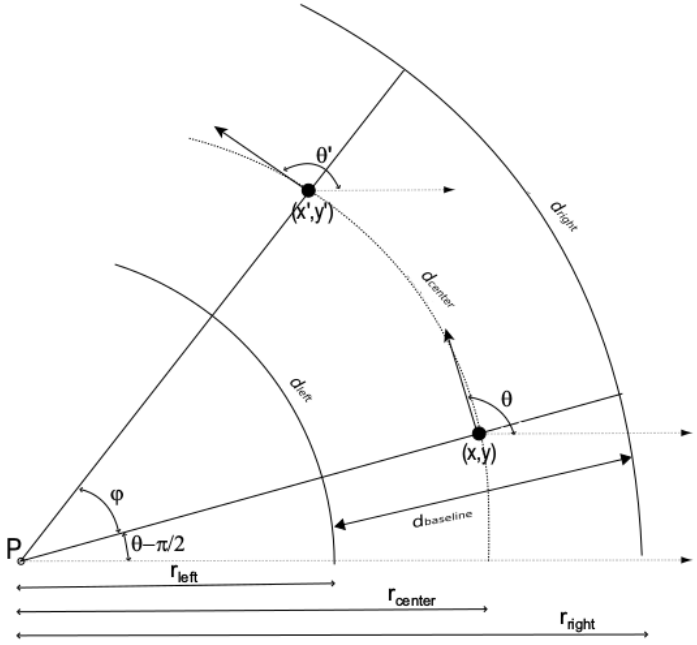
WOWWEE. *Robots*. 2016. Figura. Disponível em: <http://wowwee.com/products/robots>. Citado na página 25.

WRENN, E. *Serving humanity, one diner at a time: Chinese restaurant with robot staff delights noodle-lovers*. 2016. Figura. Disponível em: <http://www.dailymail.co.uk/sciencetech/article-2165339/Serving-humanity-diner-time-Chinese-restaurant-robot-staff-delights-noodle-lovers.html>. Citado 2 vezes nas páginas 32 e 33.

APÊNDICE A – ODOMETRIA PARA ROBÔS COM TOPOLOGIA DIFERENCIAL

De acordo com Jensfelt (2001) na definição geométrica do odometria, assume-se que o robô sempre percorre trajetórias curvas com raio *r*, pois, qualquer trajetória pode ser dividida em um conjunto finito de curvas. A Figura 41 apresenta um modelo geométrico para a odometria, nela é possível contrastar parâmetros que estão presentes na cinemática e influenciam diretamente na estimação de posição do robô, como o comprimento do eixo que separa as rodas tracionadas, chamado de $d_{baseline}$. Para Jensfelt (2001) o problema da odometria, *dead-reckoning*, é definir a posição (x', y', θ'), conhecendo (x, y, θ) e o valor de $d_{baseline}$.

Figura 41 – Modelo geométrico da odometria em um robô diferencial.



Fonte: o autor. Baseado em Olson (2004)

Neste esquemático, d_{left} e d_{right} representa a distância percorrida pelas rodas da esquerda e direita, respectivamente. A distância percorrida pelo robô, por sua vez, é definida pelo ponto médio da trajetória realizada por ambas as rodas, e chamada de d_{center}. Assim:

$$d_{centre} = \frac{d_{left} + d_{right}}{2} \tag{A.1}$$

É possível notar também que, d_{left}, d_{right} e d_{center} representam o comprimento de arcos

(vide Apêndice A - 1), de raio r_{left}, r_{right} e r_{center}, nesta ordem. Desta forma:

$$d_{left} = \phi r_{left} \tag{A.2}$$

$$d_{right} = \phi r_{right} \tag{A.3}$$

$$d_{center} = \phi r_{center} \tag{A.4}$$

Por sua vez, $d_{baseline}$ pode ser entendida por:

$$d_{baseline} = r_{right} - r_{left} \tag{A.5}$$

Substituindo A.2 e A.3 em A.5, temos:

$$d_{baseline} = \frac{d_{right} - d_{left}}{\phi} \tag{A.6}$$

$$\phi = \frac{d_{right} - d_{left}}{d_{baseline}} \tag{A.7}$$

Considerando que todo o movimento do robô realizado sob esse arco, tem o ponto $P(x_o, y_o)$ como origem, r_{center} como raio e os arcos d_{left} e d_{right} com mesmo ângulo ϕ. Assim, tendo como base a relações de circunferência e propriedades de cosseno e seno (vide Apêndice C - 2, 3 e 5), é possível definir as coordenadas de P:

$$\begin{aligned} x - x_o &= r_{center}\cos\left(\theta - \frac{\pi}{2}\right) \\ x_o &= x - r_{center}\cos\left(\theta - \frac{\pi}{2}\right) \\ x_o &= x - r_{center}[\cos(\theta)\cos\left(\frac{\pi}{2}\right) + \sin(\theta)\sin\left(\frac{\pi}{2}\right)] \\ x_o &= x - r_{center}\sin(\theta) \end{aligned} \tag{A.8}$$

$$\begin{aligned} y - y_o &= r_{center}\sin\left(\theta - \frac{\pi}{2}\right) \\ y_o &= y - r_{center}\sin\left(\theta - \frac{\pi}{2}\right) \\ y_o &= y - r_{center}[\sin(\theta)\cos\left(\frac{\pi}{2}\right) - \cos(\theta)\sin\left(\frac{\pi}{2}\right)] \\ y_o &= y + r_{center}\cos(\theta) \end{aligned} \tag{A.9}$$

Usando A.8, A.9 e as propriedades de cosseno e seno (vide Apêndice C - 4 e 6), qualquer ponto $P'(x', y')$ que pertence a circunferência é definido como:

$$
\begin{aligned}
x' &= x_o + r_{center}\cos\big(\phi+\theta-\frac{\pi}{2}\big) \\
&= x - r_{center}\sin\big(\theta\big) + r_{center}\cos\big(\phi+\theta-\frac{\pi}{2}\big) \\
&= x - r_{center}\sin\big(\theta\big) + r_{center}[\cos\big(\phi+\theta\big)\cos\big(\frac{\pi}{2}\big) + \sin\big(\phi+\theta\big)\sin\big(\frac{\pi}{2}\big)] \\
&= x - r_{center}\sin\big(\theta\big) + r_{center}\sin\big(\phi+\theta\big) \\
&= x + r_{center}[-\sin\big(\theta\big) + \sin\big(\phi+\theta\big)] \\
&= x + r_{center}[-\sin\big(\theta\big) + \sin\big(\phi\big)\cos\big(\theta\big) + \cos\big(\phi\big)\sin\big(\theta\big)] \qquad \text{(A.10)}
\end{aligned}
$$

$$
\begin{aligned}
y' &= y_o + r_{center}\sin\big(\phi+\theta-\frac{\pi}{2}\big) \\
&= y + r_{center}\cos\big(\theta\big) + r_{center}\sin\big(\phi+\theta-\frac{\pi}{2}\big) \\
&= y + r_{center}\cos\big(\theta\big) + r_{center}[\sin\big(\phi+\theta\big)\cos\big(\frac{\pi}{2}\big) - \sin\big(\frac{\pi}{2}\big)\cos\big(\phi+\theta\big)] \\
&= y + r_{center}\cos\big(\theta\big) - r_{center}\cos\big(\phi+\theta\big) \\
&= y + r_{center}[\cos\big(\theta\big) - \cos\big(\phi+\theta\big)] \\
&= y + r_{center}[\cos\big(\theta\big) - \cos\big(\phi\big)\cos\big(\theta\big) + \sin\big(\theta\big)\sin\big(\phi\big)] \qquad \text{(A.11)}
\end{aligned}
$$

Segundo Olson (2004) como a localização do robô é atualizada com uma frequência alta, então o ângulo ϕ, que justamente se refere à variação angular da posição anterior ao atual, é um valor pequeno, assim sendo, com o intuito de simplificar A.10 e A.11, as seguintes aproximações são consideradas:

$$\sin\big(\phi\big) = \phi \qquad \text{(A.12)}$$

$$\cos\big(\phi\big) = 1 \qquad \text{(A.13)}$$

Assim, substituindo A.12, A.13 e A.4 em A.10 e A.11, tem-se:

$$x' = x + d_{center}\cos\big(\theta\big) \qquad \text{(A.14)}$$

$$y' = y + d_{center}\sin\big(\theta\big) \qquad \text{(A.15)}$$

A partir das explanações sobre o modelo geométrico da odometria, é possível, então, sumarizar as principais equações odométricas. A Tabela 2.4.1.1 apresenta-as.

APÊNDICE B – CORREÇÃO DE ERRO SISTEMÁTICO USANDO A TÉCNICA *UMBMARK* PARA ROBÔS DIFERENCIAIS

O *UMBmark* foi desenvolvido por Borenstein e Feng (1995b), para quantificar o erro sistemático na estimação da posição de robôs diferenciais. Segundo os autores, a principal vantagem da técnica é sua simplicidade, praticidade e independência quanto ao uso de sensores externos, como em McKerrow e Ratner (2002), que usam sensores ultrassom para auxiliar na identificação e correção dos erros de odometria.

Antes de definir a técnica, os autores identificaram a partir de testes e experimentos, que os principais fatores que incrementam o erro sistemático são: a diferença de diâmetro entres as rodas, chamado de E_D (erro de diâmetro), que influencia apenas nos percursos retilíneos do robô; e, a inexatidão da medição do tamanho do eixo que separa as rodas do robô ($d_{baseline}$), nomeado de E_B (erro no tamanho do eixo entre as rodas), que ocasiona desníveis durante a realização de curvas. Desta forma, considerando tais erros, a posição $P(x, y, \theta)$ do robô, passa a ser $P'(x+\varepsilon x, y+\varepsilon y, \theta+\varepsilon\theta)$, sendo εx, εy e $\varepsilon\theta$ calculados como:

$$\varepsilon x = x_{abs} - x_{calc} \tag{B.1}$$

$$\varepsilon y = y_{abs} - y_{calc} \tag{B.2}$$

$$\varepsilon\theta = \theta_{abs} - \theta_{calc} \tag{B.3}$$

Onde x_{abs}, y_{abs} e θ_{abs} são os valores reais da posição do robô, e, x_{calc}, $ycalc$ e θ_{calc}, foram calculados pelas equações odométricas.

O algoritmo usado no teste é bastante simples, nele o robô deve percorrer n vezes[1] um percurso quadrangular (em ambos os sentidos; horário - *cw*, e anti-horário - *ccw*) que possui lados de mesma dimensão (L) e ângulos de 90°, como mostrado na Figura 20. Após cada i iteração, é calculado a posição $P'(x_{calc}, y_{calc}, \theta_{calc})$ e os valores de εx, εy e $\varepsilon\theta$.

Assim, conforme Borenstein e Feng (1995b), realizadas todas as n iterações, calcula-se um valor único, denominado centro de gravidade de *cluster*, de todas as estimações de erros. É importante frisar que serão realizados cálculos apenas em relação a posição cartesiana do veículo, pois, na prática os erros sistemáticos influenciam apenas na estimação de x e y (BORENSTEIN; FENG, 1996). Desta forma, usando B.1 e B.2, tem-se:

[1] Segundo Borenstein e Feng (1995b) um valor ideal para n é 5.

$$x_{c.g_{cw/ccw}} = \frac{1}{n}\sum_{i=1}^{n} \varepsilon x_{i_{cw/ccw}} \tag{B.4}$$

$$y_{c.g_{cw/ccw}} = \frac{1}{n}\sum_{i=1}^{n} \varepsilon y_{i_{cw/ccw}} \tag{B.5}$$

Por fim, usando B.4 e B.5 é possível determinar o valor da distância do centros $A'(x_{c.g_{cw}}, y_{c.g_{cw}})$ e $B'(x_{c.g_{ccw}}, y_{c.g_{ccw}})$ para o ponto $C(0,0)$ (posição deseja, pois ocorre apenas quando não há nenhum erro odométrico), denotados por $r_{c.g_{cw}}$ e $r_{c.g_{ccw}}$, e calculados como:

$$r_{cg_{cw}} = \sqrt{(x_{c.g_{cw}})^2 + (y_{c.g_{cw}})^2} \tag{B.6}$$

$$r_{cg_{ccw}} = \sqrt{(x_{c.g_{ccw}})^2 + (y_{c.g_{ccw}})^2} \tag{B.7}$$

Desta forma, considera-se como erro máximo sistemático $\left(E_{max}^{syst}\right)$ o valor máximo entre $r_{cg_{cw}}$ e $r_{cg_{ccw}}$, pois, segundo Borenstein e Feng (1995b) maior erro odométrico influenciará, na prática, na estimação da posição final do robô. Sendo assim:

$$E_{max}^{syst} = \max\{r_{cg_{cw}}, r_{cg_{ccw}}\} \tag{B.8}$$

Feitos os cálculos de $x_{cg_{cw/ccw}}$,$y_{cg_{cw/ccw}}$ é possível, então, calcular os fatores c_{left} e c_{right} que corrigem o erro ocasionados por E_D para as rodas esquerda e direita, respectivamente. Além disso, calcula-se o valor de E_B, usado na correção do mesmo. Borenstein e Feng (1996) esclarecem, no entanto, que é preciso compreender dois erros característicos que serão observados na aplicação do *UMBmark* (Fig 20b e 20c). Os autores os nomeiam de Tipo A e Tipo B, e eles são definidos por:

- **Tipo A:** O erro do Tipo A (fig 20b) ocorre quando, durante a execução do percurso quadrangular em ambas as direções (*cw* e *ccw*), a orientação do robô reduz ou é incrementada. Este é causado por E_B e, formalmente, pode ser definido como:

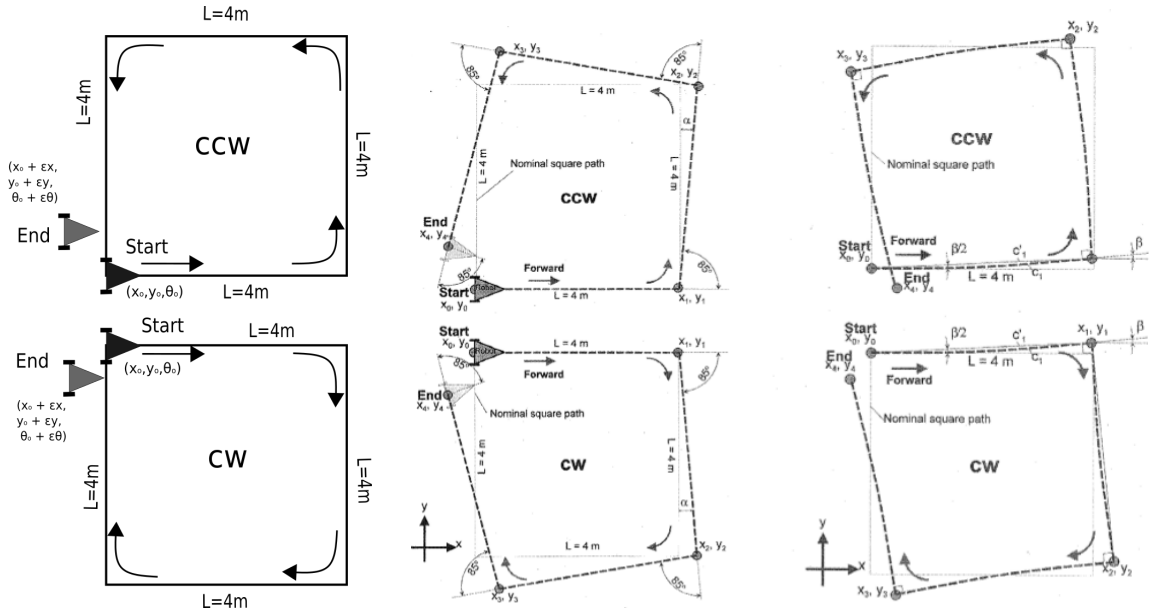
$$|\,\theta_{total_{cw}}| < |\theta_{nominal}| \quad e \quad |\theta_{total_{ccw}}| < |\theta_{nominal}|$$

ou

$$|\,\theta_{total_{cw}}| > |\theta_{nominal}| \quad e \quad |\theta_{total_{ccw}}| > |\theta_{nominal}|$$

- **Tipo B:** O erro do Tipo B (Fig 20c), por sua vez, ocorre quando, durante a execução do percurso, a orientação do robô reduz em uma direção e incrementa na outra. Este é

Figura 42 – Percusso realizado no teste *UMBmark*.



(a) UMBmark (b) Tipo A (c) Tipo B

Fonte: o autor; e, Borenstein e Feng (1996)

ocasionado por E_D, e, é definido como:

$$|\ \theta_{total_{cw}}| < |\theta_{nominal}| \quad e \quad |\theta_{total_{ccw}}| > |\theta_{nominal}|$$

ou

$$|\ \theta_{total_{cw}}| > |\theta_{nominal}| \quad e \quad |\theta_{total_{ccw}}| < |\theta_{nominal}|$$

Assim, segundo Borenstein e Feng (1996), corrigindo E_B e E_D, consequentemente, elimina-se quase que completamente os erros do Tipo A e B. Os autores demonstram que para ambos os tipos, a posição final do robô é definida pelas seguintes equações:

$$P_{A_{cw}}(x,y) = (-2L\alpha, -2L\alpha) \tag{B.9}$$

$$P_{A_{ccw}}(x,y) = (-2L\alpha, 2L\alpha) \tag{B.10}$$

$$P_{B_{cw}}(x,y) = (-2L\beta, -2L\beta) \tag{B.11}$$

$$P_{B_{ccw}}(x,y) = (2L\beta, -2L\beta) \tag{B.12}$$

Como ambos os erros ocorrem simultaneamente, somando B.9 com B.11 e B.10 com B.12, tem-se:

$$P_{cw}(x,y) = (-2L\alpha - 2L\beta, -2L\alpha - 2L\beta) \tag{B.13}$$

$$P_{ccw}(x,y) = (-2L\alpha + 2L\beta, 2L\alpha - 2L\beta) \tag{B.14}$$

Substituindo B.13 e B.14 em B.4 e B.5, é possível então definir os valores de os pontos $P_{cw}(x,y)$ e $P_{ccw}(x,y)$ como:

$$P_{cw}(x,y) = (x_{cg_{cw}}, y_{cg_{cw}})$$

$$x_{cg_{cw}} = -2L\alpha - 2L\beta \tag{B.15}$$

$$y_{cg_{cw}} = -2L\alpha - 2L\beta \tag{B.16}$$

$$P_{ccw}(x,y) = (x_{cg_{ccw}}, y_{cg_{ccw}})$$

$$x_{cg_{ccw}} = -2L\alpha + 2L\beta \tag{B.17}$$

$$y_{cg_{ccw}} = 2L\alpha - 2L\beta \tag{B.18}$$

O valor de β em radianos é definido pela subtração de B.15 e B.17, assim:

$$x_{cg_{cw}} - x_{cg_{cw}} = (-2L\alpha) - (-2L\alpha + 2L\beta)$$

$$\beta_{[radiano]} = \frac{(x_{cg_{cw}} - x_{cg_{ccw}})}{-4L}$$

$$\beta_{[grau]} = \beta_{[radiano]} \left(\frac{180^\circ}{\pi} \right) \tag{B.19}$$

Considerando que o raio da curvatura r_{center}, é dado pela equação 20, como explicitado em Borenstein e Feng (1996), usando relações geométricas (vide apêndice C.2) na análise dos percursos realizados pelo robô, é possível então definir o valor de E_D, que será usado na correção deste. Assim:

$$r_{center} = \frac{\left(\frac{L}{2}\right)}{\sin\left(\frac{\beta}{2}\right)} \tag{B.20}$$

$$E_D = \frac{D_r}{D_l} \tag{B.21}$$

$$D_r = r_{center} + \frac{d_{baseline}}{2}$$
$$D_l = r_{center} - \frac{d_{baseline}}{2}$$

$$E_D = \frac{\left(r_{center} + \frac{d_{baseline}}{2}\right)}{\left(r_{center} - \frac{d_{baseline}}{2}\right)} \tag{B.22}$$

Todavia, os autores afirmam que para corrigir E_D é preciso, no entanto, considerar o valor médio dos diâmetros das rodas do veículo (Eq. B.23), definido por D_a, e que $D_r = c_r D_a$ e $D_l = c_l D_a$. Assim, usando B.21 e B.23, computa-se os valores de c_l e c_r para o caso de E_D, como:

$$D_a = \frac{(D_r + D_l)}{2} \tag{B.23}$$

$$D_r + D_l = 2D_a \qquad e \qquad D_r = E_D D_l$$

$$E_D D_l + D_l = 2D_a$$

$$D_l = \frac{2D_a}{E_D + 1}$$

$$D_r + D_l = 2D_a \qquad e \qquad D_l = \frac{D_r}{E_D}$$

$$D_r + \frac{D_r}{E_D} = 2D_a$$

$$D_r = \frac{2D_a}{1 + \frac{1}{E_D}}$$

$$c_l = \frac{2}{E_D + 1} \tag{B.24}$$

$$c_r = \frac{2}{1 + \frac{1}{E_D}} \tag{B.25}$$

Por sua vez, para corrigir E_B, de acordo com os autores, é preciso computar o valor de α, usando B.15 e B.17, e o valor de $d_{baseline_{actual}}$, sabendo que:

$$\frac{d_{baseline_{actual}}}{90^\circ} = \frac{d_{baseline_{nominal}}}{(90^o - \alpha)} \tag{B.26}$$

$$d_{baseline_{actual}} = \frac{(90^\circ d_{baseline_{nominal}})}{90^\circ - \alpha} \tag{B.27}$$

$$x_{cg_{cw}} + x_{cg_{ccw}} = (-2L\alpha - 2L\beta) + (-2L\alpha + 2L\beta)$$

$$\alpha_{[radiano]} = \frac{(x_{cg_{cw}} + x_{cg_{ccw}})}{-4L}$$

$$\alpha_{[grau]} = \alpha_{[radiano]} \left(\frac{180^\circ}{\pi} \right) \tag{B.28}$$

Assim, considerando E_B como o fator de correção para o caso do erro de Tipo A, usando B.27, este pode ser definido por:

$$d_{baseline_{actual}} = E_B d_{baseline_{nominal}}$$

$$E_B = \frac{90^\circ}{(90^\circ - \alpha)} \tag{B.29}$$

Por fim, conforme Boreinsteins e Feng (1995), as equações B.24, B.25 e B.27 são usadas para corrigir ambos os erros (Tipo A e Tipo B). A Tabela 9 apresenta as principais equações da técnica de estimação e correção de erro sistemático para odometria usando o *UMBmark* para um robô diferencial.

Tabela 9 – Principais equações para estimação e correção de erro sistemático usando *UMBmark*.

Nome	Equação
$(\varepsilon x, \varepsilon y, \varepsilon\theta)$	$(x_{abs} - x_{calc}, y_{abs} - y_{calc}, \theta_{abs} - \theta_{calc})$
$\left(x_{c.g_{cw/ccw}}, y_{c.g_{cw/ccw}}\right)$	$\left(\frac{1}{n}\sum_{i=1}^{n} \varepsilon x_{i_{cw/ccw}}, \frac{1}{n}\sum_{i=1}^{n} \varepsilon y_{i_{cw/ccw}}\right)$
$(r_{cg_{cw}}, r_{cg_{ccw}})$	$\left(\sqrt{(x_{c.g_{cw}})^2 + (y_{c.g_{cw}})^2}, \sqrt{(x_{c.g_{ccw}})^2 + (y_{c.g_{ccw}})^2}\right)$
E_{max}^{syst}	$\max\{r_{cg_{cw}}, r_{cg_{ccw}}\}$
r_{center}	$\frac{\left(\frac{L}{2}\right)}{\sin\left(\frac{\beta}{2}\right)}$
$\left(\beta_{[grau]}, \alpha_{[grau]}\right)$	$\left(\left[\frac{x_{cg_{cw}} - x_{cg_{ccw}}}{-4L}\right]\left[\frac{180^\circ}{\pi}\right], \left[\frac{x_{cg_{cw}} + x_{cg_{ccw}}}{-4L}\right]\left[\frac{180^\circ}{\pi}\right]\right)$
E_D	$\frac{\left(r_{center} + \frac{d_{baseline}}{2}\right)}{\left(r_{center} - \frac{d_{baseline}}{2}\right)}$
(c_l, c_r)	$\left(\frac{2}{E_D+1}, \frac{2}{1+\frac{1}{E_D}}\right)$
E_B	$\frac{90^\circ}{(90^\circ - \alpha)}$
$d_{baseline_{actual}}$	$E_B d_{baseline_{nominal}}$

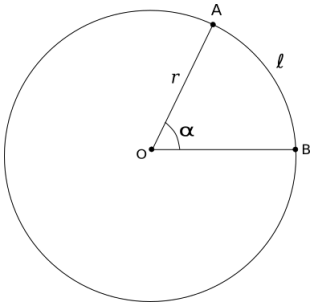
Fonte: O autor.

APÊNDICE C – CONCEITOS BÁSICOS DE MATEMÁTICA

1. **Comprimento do arco:** Seja uma circunferência de raio r (Figura 43). Sabendo que o comprimento desta é dado por $C = 2\pi r$, então, o comprimento de um segmento qualquer AB, chamado de arco, assumindo o ângulo central em radianos, é dado por:

$$l = \alpha r \tag{C.1}$$

Figura 43 – Circunferência



Fonte: o autor.

2. **Relação trigonométrica:** Seja a circunferência representada na Figura 44. As dimensões dos segmentos $y - y_o$ e $x - x_o$ são definidos como:

$$\begin{aligned} \sin(\alpha) &= \frac{y - y_o}{h} \\ y - y_o &= \sin(\alpha) h \end{aligned} \tag{C.2}$$

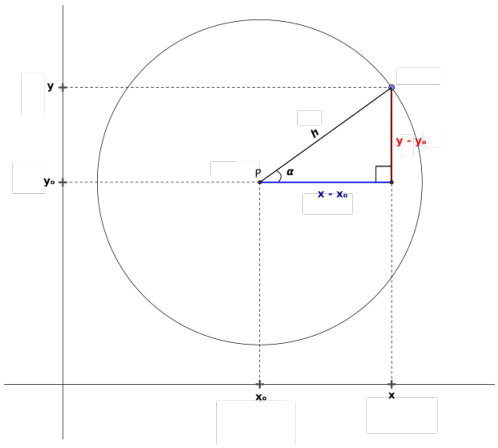
$$\begin{aligned} \cos(\alpha) &= \frac{x - x_o}{h} \\ x - x_o &= \cos(\alpha) h \end{aligned} \tag{C.3}$$

A partir de C.2 e C.3 é possível definir os pontos que representam o centro $P(x_o, y_o)$ da circunferência.

$$y_o = y - \sin\alpha h \tag{C.4}$$

$$x_o = x - \cos\alpha h \tag{C.5}$$

Figura 44 – Relação trigonométrica para o Ponto P



Fonte: o autor.

3. **Seno da diferença** O valor de seno resultante da direfença de dois ângulo, α e β , é definido como:

$$\sin(\alpha-\beta) = \sin(\alpha)\cos(\beta) - \cos(\alpha)\sin(\beta) \tag{C.6}$$

4. **Seno da soma** O valor de seno resultante da soma de dois ângulo, α e β , é definido como:

$$\sin(\alpha+\beta) = \sin(\alpha)\cos(\beta) + \cos(\alpha)\sin(\beta) \tag{C.7}$$

5. **Cosseno da diferença** O valor de cosseno resultante da diferença de dois ângulo, α e β , é definido como:

$$\cos(\alpha-\beta) = \cos(\alpha)\cos(\beta) + \sin(\alpha)\sin(\beta) \tag{C.8}$$

6. **Cosseno da soma** O valor de cosseno resultante da soma de dois ângulo, α e β , é definido como:

$$\cos(\alpha+\beta) = \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta) \tag{C.9}$$