Universidade Federal de São Carlos – UFSCar
Centro de Ciências Exatas e de Tecnologia – CCET
Departamento de Computação – DC
Programa de Pós-Graduação em Ciência da Computação – PPGCC

Jose Ceron Neto

Controle $\mathcal{H}_\infty$ não linear adaptativo baseado em aprendizado profundo para robôs móveis com rodas

São Carlos
2023

Jose Ceron Neto

Controle $\mathcal{H}_\infty$ não linear adaptativo baseado em aprendizado profundo para robôs móveis com rodas

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências Exatas e de Tecnologia da Universidade Federal de São Carlos , como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Metodologias e Técnicas de Computação

Orientador: Prof. Dr. Roberto Santos Inoue

São Carlos
2023

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Defesa de Dissertação de Mestrado do candidato José Ceron Neto, realizada em 09/02/2023.

Comissão Julgadora:

Prof. Dr. Roberto Santos Inoue (UFSCar)

Prof. Dr. Samuel Lourenço Nogueira (UFSCar)

Prof. Dr. Adriano Almeida Goncalves Siqueira (USP)

O Relatório de Defesa assinado pelos membros da Comissão Julgadora encontra-se arquivado junto ao Programa de Pós-Graduação em Ciência da Computação.

Dedico esta dissertação à minha esposa Alessandra e minha filha Lia,
pelo amor, apoio incondicional e presença essenciais para sua conclusão.
Grato pela compreensão com as minhas horas de ausência.

Agradecimentos

Ao meu orientador Prof. Dr. Roberto Santos Inoue, sempre presente, incentivando-me e apoiando-me e superando até a distância. Obrigado pela confiança, orientação, tempo e paciência dedicado à esse trabalho.

À minha esposa Alessandra Miguel Kapp, minha companheira, incentivadora e parceira. Agradeço à toda a paciência e dedicação que me oferece diariamente.

À nossa filha, Lia Kapp Ceron que nos ensinou uma nova forma de enxergar a vida e buscar um futuro melhor.

Aos meus pais, irmão e avós, obrigado pelo apoio.

Aos amigos que fiz durante a pós-graduação e aos de longa data que me apoiaram durante a realização deste trabalho.

Aos amigos e membros do LARIS pela amizade, paciência, suporte, companheirismo durante a realização deste trabalho.

Ao Programa de Pós Graduação em Ciência da Computação da Universidade Federal de São Carlos.

Aos professores e funcionários do Departamento de Computação da Universidade Federal de São Carlos pelas contribuições durante o mestrado.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pela concessão da bolsa de mestrado.

"Se você quer ser bem sucedido,
precisa ter dedicação total, buscar seu último limite
e dar o melhor de si."
(Ayrton Senna)

Resumo

A implementação de robôs móveis autônomos levanta questões importantes quando se deseja um sistema de navegação robusto para o transporte de cargas. Esses tipos de veículos estão frequentemente sujeitos a incertezas paramétricas e distúrbios externos. Incertezas de parâmetros geralmente surgem devido a dispositivos ou cargas extras que são anexadas ao robô, pois influenciam os parâmetros de massa, inércia, centro de massa e outros parâmetros levantados inicialmente para compor o modelo matemático do veículo. E as perturbações externas estão relacionadas à colisão do robô com obstáculos estáticos ou dinâmicos ou na superação de obstáculos no solo pelas rodas do robô, onde podem ocorrer escorregamentos e derrapagens das rodas. Com base nesse contexto, o presente trabalho propõe uma arquitetura de controle robusta e adaptativa para o problema de rastreamento de trajetória de um robô móvel sujeito a perturbações externas e incertezas paramétricas. A abordagem proposta compreenderá um Controle Adaptativo Não Linear $\mathcal{H}_\infty$. O controlador não linear $\mathcal{H}_\infty$ será responsável por atenuar distúrbios externos, e a parte adaptativa será baseada em Rede Neural Profunda para aprender as incertezas paramétricas relacionadas ao modelo matemático do robô. A simulação de um robô móvel de quatro rodas para o problema de rastreamento são usados para comparar as estratégias de controle propostas.

Palavras-chave: Robô Móvel. Controle Robusto Adaptativo. Controle $\mathcal{H}_\infty$. Rede Neural. Aprendizado Profundo.

Abstract

The implementation of autonomous mobile robots raises important issues when you want a robust navigation system to transport loads. These types of vehicles are often subject to parametric uncertainties and external disturbances. Parameter uncertainties usually arise due to extra devices or loads that are attached to the robot since they influence the parameters of mass, inertia, center of mass, and other parameters initially raised to compose the vehicle mathematical model. And the external disturbances are related to the robot's collision with static or dynamic obstacles or in overcoming obstacles on the ground by the robot's wheels, where skidding and slippage of the wheels can occur. Based on this context, the present work proposes a robust and adaptive control architecture for the trajectory tracking problem of a mobile robot subject to external disturbances and parametric uncertainties. The proposed approach will comprise a Nonlinear Adaptive Control $\mathcal{H}_\infty$ based on Deep Learning. The nonlinear $\mathcal{H}_\infty$ controller will be responsible for attenuating external disturbances, and the adaptive part will be based on Deep Neural Network for learning the parametric uncertainties related to the robot's mathematical model. Simulation results of a four-wheel mobile robot for the tracking problem are used to compare the control strategies proposed.

Keywords: Mobile Robot. Robust Adaptative Controler. $\mathcal{H}_\infty$ Controller. Neural Network. Deep Learning.

Lista de ilustrações

Lista de tabelas

Lista de siglas

AMR *Autonomous Mobile Robot* - Robô Móvel Autônomo

CM *Center of Mass* - Centro de Massa

CIR Centro Instantâneo de Rotação

DAC *Deep Adaptive Control* - Controle Adaptativo e Aprendizado Profundo

DNN *Deep Neural Network* - Rede Neural Profunda

DMRAC *Deep Model Reference Adaptive Control* - Controle Adaptativo por Modelo de Referência e Aprendizado Profundo

ROS *Robot Operating System* - Sistema operacional do robô

RMR Robô Móvel sobre Rodas

RMRD Robô Móvel com Rodas Deslizantes

SGD *Stochastic Gradient Descent* - Gradiente Descente Estocástico

SPI *Serial Peripheral Interface* - Interface Periférica Serial

SS *Slave Select* - Seleção de Chip

USB *Universal Serial Bus* - Barramento Serial Universal

XML *Extensible Markup Language*- Linguagem de marcação extensível

Sumário

Capítulo 1

Introdução

1.1 Motivação

Com o advento do paradigma da Indústria 4.0, os robôs móveis autônomos, conhecidos como *Autonomous Mobile Robot* - Robô Móvel Autônomo (AMR), têm desempenhado um papel cada vez mais crucial em diversos setores, como logística, manufatura e saúde. Esses robôs, equipados com sistemas de navegação autônomos, oferecem uma série de benefícios, incluindo maior segurança, flexibilidade e aumento da produtividade. A habilidade desses veículos de planejar e executar seus próprios caminhos, sem intervenção humana, com base em sensores embarcados, tem se mostrado uma das funções mais importantes para os robôs de serviço, atendendo à crescente demanda do mercado Lee et al. (2019).

Dessa forma, a modelagem dos robôs móveis como sistemas incertos não lineares torna-se essencial para lidar com as incertezas nos parâmetros cinemáticos e dinâmicos, além de abranger perturbações externas e ruídos de medição (MOHSENI; VORONOV; FRISK, 2018). O desenvolvimento de abordagens de controle robustas é fundamental para garantir um desempenho consistente e confiável desses sistemas em face das incertezas e distúrbios mencionados.

É nesse contexto que se destaca a importância de aprimorar a eficácia das abordagens de controle para robôs móveis autônomos. A pesquisa nessa área visa explorar técnicas e algoritmos que permitam uma adaptação robusta às incertezas paramétricas e aos distúrbios externos, garantindo um desempenho estável e preciso desses sistemas em diferentes ambientes e condições variáveis. Ao abordar essa necessidade, espera-se contribuir para o desenvolvimento de robôs móveis autônomos mais eficientes, confiáveis e versáteis, impulsionando ainda mais a sua aplicação em diversos setores.

No entanto, é importante destacar que a pesquisa atual sobre controle de robôs móveis

autônomos ainda apresenta lacunas significativas no tratamento adequado das incertezas paramétricas e dos distúrbios externos. Portanto, há uma demanda crescente por estudos que explorem novas abordagens ou aprimorem as existentes, a fim de garantir a eficácia e a robustez dos sistemas de controle, aperfeiçoando a capacidade dos robôs móveis autônomos de lidar com os desafios impostos pelas incertezas e pelos distúrbios.

Ao abordar essas questões, este trabalho visa contribuir para o avanço da pesquisa e desenvolvimento de abordagens de controle robustas para robôs móveis autônomos. Através de investigações teóricas e simulações, busca-se identificar soluções eficientes e confiáveis que possam aprimorar o desempenho desses sistemas em ambientes complexos e dinâmicos. Com isso, espera-se abrir caminho para a implementação mais ampla e efetiva dos robôs móveis autônomos, impulsionando a automação inteligente e revolucionando diversos setores industriais e de serviços.

1.2 Revisão Bibliográfica

Soluções robustas e adaptativas têm sido usadas para controlar robôs móveis sujeitos a incertezas paramétricas, veja por exemplo Mohseni, Voronov e Frisk (2018), Park et al. (2008), Hwang e Wu (2013), Martins et al. (2008), Hu, Ge e Su (2004), Khalaji e Moosavian (2014), Huang, Van Hung e Tseng (2015), Peng et al. (2017), Kanayama et al. (1990). Em Inoue, Siqueira e Terra (2009a), o controlador cinemático Kanayama Kanayama et al. (1990) foi usado para gerar trajetórias desejadas para um robô móvel, e estratégias de controle $\mathcal{H}_\infty$ baseadas no modelo dinâmico foram implementadas para o controle de um robô móvel para atenuar perturbações externas. Em Abu-Khalaf, Lewis e Huang (2004), um controle adaptativo robusto de comutação suave foi projetado para um robô móvel omnidirecional para lidar com incertezas estruturadas e não estruturadas. Em Mohseni, Voronov e Frisk (2018), foi proposta uma abordagem de controle robusta baseada na estratégia de controle de tensão, na qual a dinâmica dos motores foi levada em consideração. Em Peng et al. (2017), um esquema de controle de rastreamento híbrido robusto, que combina um controle de torque calculado e uma abordagem de controle híbrido robusto, é proposto para robôs móveis não holonômicos sujeitos a incertezas paramétricas. Em Park et al. (2010), Chen (2017), Li et al. (2018), Shen, Ma e Song (2018), restrições não holonômicas perturbadas na presença de deslizamento e distúrbios de deslizamento foram consideradas no modelo cinemático em que abordagens de controle robusto e adaptativo foram usadas para atenuá-las. Em Shen, Ma e Song (2018), além da dinâmica de deslizamento e derrapagem, a incerteza quanto o posicionamento do *Center of Mass* - Centro de Massa (CM), também foi considerada no projeto do controlador. Em Inoue et al. (2019), os autores propuseram o uso de uma abordagem robusta de controle recursivo Ishihara, Terra e Cerri (2015) para lidar com problemas de rastreamento de um robô móvel com rodas quando ele está sujeito a incertezas paramétricas e cargas variáveis no tempo.

Estratégias de controle $\mathcal{H}_\infty$ não lineares como propostas em Inoue, Siqueira e Terra (2009b) são menos efetivas quando comparadas com abordagens robustas que levam em consideração incertezas paramétricas Inoue et al. (2019) para robôs móveis com rodas, como apontado pelo estudo comparativo realizado em Inoue et al. (2019). No entanto, abordagens de controle $\mathcal{H}_\infty$ não lineares como a proposta por Chang e Chen (1997) já estão estabelecidas na rejeição de distúrbios externos e têm um custo computacional menor em comparação com controladores robustos como o proposto em Chang, Yen e Wang (2004). Nesse sentido, a adição de uma abordagem de controle adaptativo para melhorar o desempenho de controladores $\mathcal{H}_\infty$ não lineares como proposto por Chen, Chang e Lee (1997) na presença de incertezas torna-se interessante nesse cenário.

Um algoritmo de controle adaptativo $\mathcal{H}_\infty$ não linear é proposto em Chen, Chang e Lee (1997), onde um projeto robusto de controle path-following considera que uma adaptação atualizável de lei clássica pode aprender parâmetros desconhecidos. Controles adaptativos não lineares $\mathcal{H}_\infty$ baseados em técnicas inteligentes podem ser vistos em Chen, Chang e Lee (1997), Chang e Chen (1997) e Chang (2005). Redes neurais e lógica difusa são empregadas para estimar todo o modelo dinâmico do robô. Uma abordagem interessante desenvolvida em Lee e Ge (1998) utiliza um controlador adaptativo baseado no modelo nominal e redes neurais, sendo as redes neurais utilizadas apenas para estimar as incertezas paramétricas do sistema robótico. Em Inoue, Siqueira e Terra (2007), um controlador adaptativo não linear foi desenvolvido com base no modelo nominal e em redes neurais ou fuzzy usando o critério de desempenho robusto $\mathcal{H}_\infty$. Recentemente, Joshi e Chowdhary (2018) propôs uma nova arquitetura para controle adaptativo de um modelo de referência baseado em processos Gaussianos cujo treinamento é feito por redes generativas Goodfellow et al. (2014). Em Joshi e Chowdhary (2019), Joshi, Chowdhary e Waanders (2021), a arquitetura proposta em Joshi e Chowdhary (2018) agora é desenvolvida utilizando Redes Neurais Profundas que possuem o poder de aprender características complexas através da composição de redes profundas. Resultados experimentais com um quadrotor comercial são apresentados. Uma característica importante deste controlador adaptativo de modelo de referência é que ele possui um aprendizado rápido da rede adaptativa externa e um aprendizado lento da rede de recursos profundos. Vale ressaltar que os controladores adaptativos desenvolvidos em Joshi e Chowdhary (2019), Joshi, Chowdhary e Waanders (2021) não utilizam um critério de desempenho robusto, semelhante ao critério $\mathcal{H}_\infty$, por exemplo.

1.3 Objetivos

Considerando a motivação e a revisão bibliográfica anterior, este trabalho tem como objetivo principal a aplicação de leis de controle robusto adaptativo baseadas em Redes Neurais Profundas para estimar os termos incertos dos Robô Móvel sobre Rodas (RMR)

e a utilização do controle não linear $\mathcal{H}_\infty$ para atenuar distúrbios externos. Essas abordagens visam melhorar o desempenho e a robustez dos robôs móveis autônomos diante das incertezas paramétricas e dos distúrbios externos.

Para realizar a simulação dessas abordagens, utilizou-se o controle baseado em cinemática desenvolvido por Pazderski e Kozłowski (2004) para obter as velocidades desejadas, as quais serviram como referências de entrada para os controladores propostos. Por meio de simulações computacionais, será realizado um estudo comparativo entre as abordagens propostas e o controlador padrão $\mathcal{H}_\infty$ apresentado em Inoue, Siqueira e Terra (2009b).

Nesse estudo comparativo, serão analisadas métricas relevantes, como estabilidade, tempo de resposta, precisão do seguimento de trajetória e capacidade de rejeição de distúrbios externos, a fim de destacar as vantagens desses controladores robustos em relação ao controlador $\mathcal{H}_\infty$ padrão existente na literatura. Essa análise permitirá uma comparação mais abrangente e uma avaliação precisa do desempenho dos controladores propostos diante das incertezas e distúrbios externos.

Adicionalmente, este trabalho buscará fornecer diretrizes para possíveis melhorias e trabalhos futuros. Serão discutidas as limitações encontradas durante a implementação e simulação das abordagens propostas, como restrições computacionais e desafios na estimativa dos termos incertos dos robôs móveis sobre rodas. Sugestões para futuros trabalhos incluirão aprimoramentos nas técnicas de controle, como a exploração de algoritmos de aprendizado por reforço, a aplicação em ambientes mais complexos e a validação experimental em sistemas reais.

Com a realização desses objetivos, espera-se contribuir para o avanço do conhecimento na área de controle de robôs móveis autônomos, fornecendo soluções eficazes para lidar com as incertezas paramétricas e os distúrbios externos. Além disso, busca-se oferecer diretrizes para futuras pesquisas e inspirar o desenvolvimento de abordagens ainda mais eficientes e robustas para o controle desses sistemas em diferentes aplicações industriais e de serviços.

1.4 Organização dos Capítulos

O trabalho está organizado da seguinte forma: o Capítulo 2 apresentamos a formulação do problema de controle que será desenvolvido ao longo do trabalho. No Capitulo 3 estão apresentados a modelagem cinemática e dinâmica do robô móvel com rodas, o controlador cinemático, a representação em estado-espaço para o controle $\mathcal{H}_\infty$ não linear, o $\mathcal{H}_\infty$ não linear *Deep Model Reference Adaptive Control* - Controle Adaptativo por Modelo de Referência e Aprendizado Profundo (DMRAC) e o $\mathcal{H}_\infty$ não linear *Deep Adaptive Control* - Controle Adaptativo e Aprendizado Profundo (DAC). Já no Capítulo 4 serão apresentados os resultados simulados e experimentais que foram alcançados. E, por fim, o Capítulo 5 estão apresentadas as conclusões do trabalho e os trabalhos futuros.

Capítulo 2

Proposta do Trabalho

A proposta deste trabalho é modelar, desenvolver, simular e testar uma arquitetura robótica controle robusto sujeita a distúrbios externos, variações de CM, deslizamento, derrapagens e incertezas paramétricas modeladas ou não em ambientes desconhecidos.

Como forma de facilitar o entendimento da proposta da arquitetura do controlador robusto, optou-se pela representação em formato esquemático, como apresentado na Figura 1.

De forma resumida, inicialmente o bloco *Trajetória de Referência* será responsável por gerar q^{ref} e $\dot{q}^{ref}$, respectivamente posição e velocidades de referência através da resolução de um polinômio do quinto grau e suas derivadas ou através da resolução de uma equação ciclo-limite, conforme apresentado na Seção 3.2.1.

Em seguida, as saídas do bloco anterior q e $\dot{q}$, tornam-se entradas do bloco de *Erro de Postura*. Neste bloco, são calculados a posição e as velocidades estimadas, $\tilde{q}$ e $\tilde{\dot{q}}$.

O próximo bloco, *Controle Cinemático*, calcula as velocidades linear e angular desejadas v^d e ω^d, a partir $\tilde{q}$ e $\tilde{\dot{q}}$, e obtém as velocidades angulares desejadas das rodas ω_L^d e ω_R^d a partir das velocidades lineares e angulares desejadas Além disso, o bloco *Erro de Velocidade das Rodas* calcula o erro de deslocamento $\tilde{\delta}_L$ e $\tilde{\delta}_R$ e o erro de velocidade $\tilde{\omega}_L$ e $\tilde{\omega}_R$ das rodas usando a posição de referência q^{ref} e sua posição real q. Esses cálculos são mencionados na Seção 3.2.2.

O bloco *Controle $\mathcal{H}_\infty$ não linear adaptativo baseado em aprendizado profundo* calcula o torque τ a ser aplicado nas rodas do robô. É baseado em um Controle Não Linear $\mathcal{H}_\infty$ baseado em Taveira, Siqueira e Terra (2006) para lidar com distúrbios externos e uma Rede Neural que estima as incertezas do sistema baseado em Joshi, Chowdhary e Waanders (2021). O controle proposto é apresentado na Seção 3.2.3.

Por fim, o bloco *Robô Móvel* recebe como entrada o torque τ a ser aplicado nas rodas,

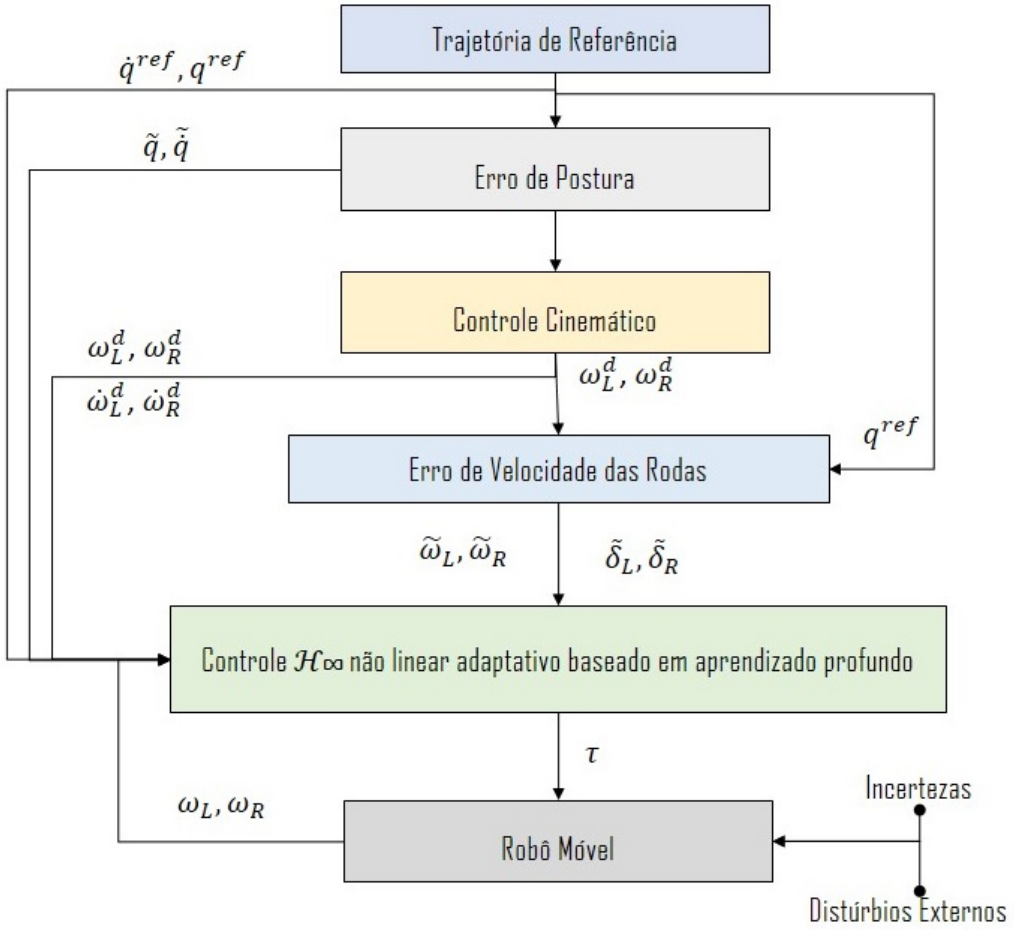


Figura 1 – Diagrama de Blocos Arquitetura Proposta

e fornece as velocidades angulares ω_L e ω_R e a posição q. Aqui o robô móvel está sujeito a perturbações externas e incertezas paramétricas. Distúrbios externos são devidos à dinâmica de derrapagens e deslizamento, e incertezas são devidas às variações do centro de massa, estimativa errônea de massa e inércia e mudanças na carga útil.

Capítulo 3

Conceitos Básicos

Neste capítulo, são apresentados os conceitos que serão utilizados nesse trabalho. Na Seção 3.1 aborda a modelagem Cinemática e Dinâmica do robô que será baseada nos trabalhos de Caracciolo, Luca e Iannitti (1999), Pazderski e Kozłowski (2004) e Cui et al. (2017), e também é apresentada a Representação em espaço-estado. Já na Seção 3.2, as técnicas de controle que serão utilizadas no robô móvel são apresentadas: desde o cálculo da trajetória de referência, passando pelo controle baseado na cinemática e por fim o controle baseado na dinâmica.

3.1 Modelagem

Nesta seção, inicia-se pela modelagem cinemática do robô móvel 3.1.1. Na sequência, as equações que definem o modelo dinâmico da movimentação do robô. E por fim, uma solução para conversão de torque para tensão de alimentação de atuadores em corrente contínua será introduzida.

3.1.1 Modelagem Cinemática

Assume-se que o RMRD esteja sob uma superfície plana, sendo (X, Y) o sistema de coordenadas de navegação. Já (X_{CM}, Y_{CM}) o sistema de coordenadas do corpo que possui como origem o centro de massa do RMRD.

Quanto ao movimento do robô, tem-se que a sua velocidade linear e angular podem ser expressas, no sistema de coordenadas do corpo, como $v = [v_x \; v_y \; 0]^T$ e $\omega = [0 \; 0 \; \omega]^T$. E considera-se como o vetor de estado $q = [x \; y \; \psi]^T$ descrevendo a posição CM, x e y, no sistema de coordenadas e ψ como sendo a rotação do corpo. Desta forma temos

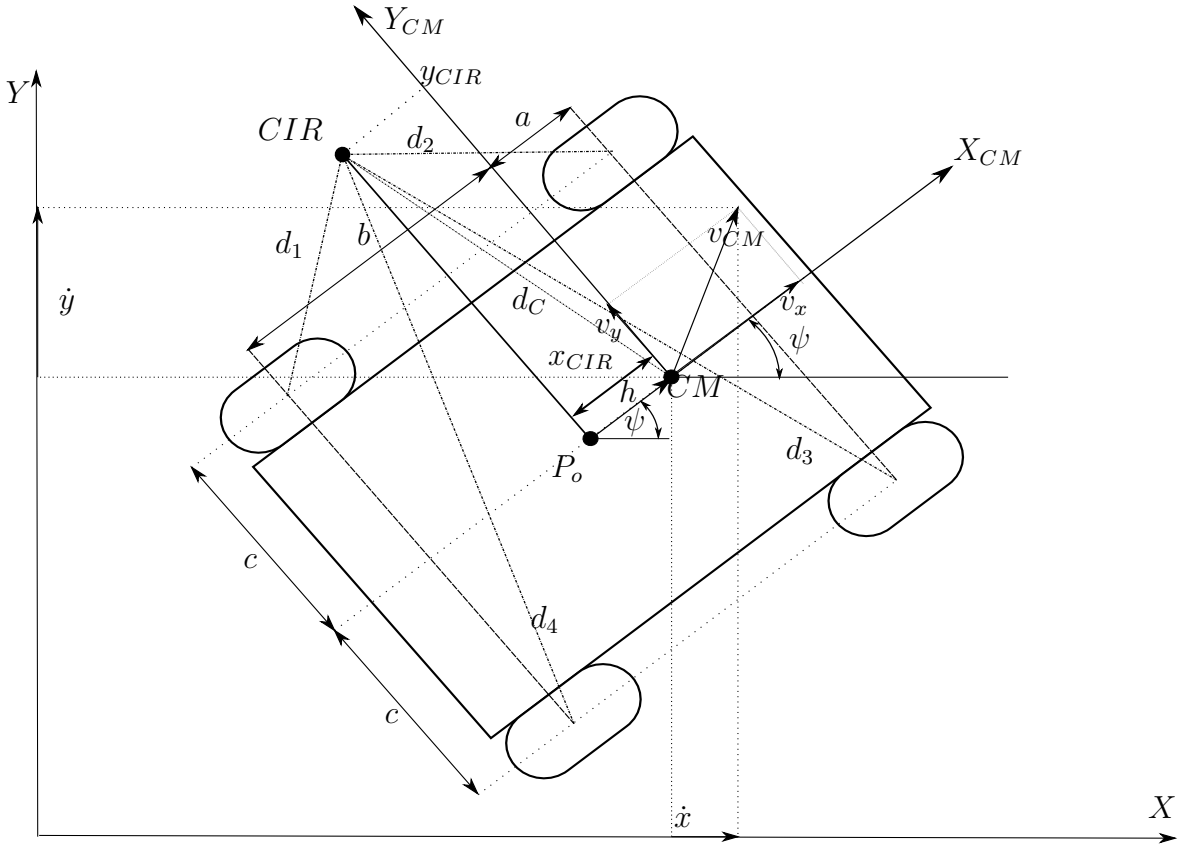


Figura 2 – Geometria do RMRD

$\dot{q} = [\dot{x}\ \dot{y}\ \dot{\psi}]^T$ e pelo movimento planar, $\dot{\psi} = \omega$. Portanto, as velocidades absolutas $\dot{x}$, $\dot{y}$ e $\dot{\psi}$ no sistema de coordenadas de navegação, são dadas por

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \cos\psi & -\text{sen}\psi & 0 \\ \text{sen}\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}. \tag{1}$$

Na equação (1) não há imposição de nenhuma restrição ao movimento do RMRD, sendo necessária a análise da relação das velocidades locais e das rodas. Desta forma, em seu trabalho, Pazderski e Kozłowski (2004) supõe que cada roda, i possua uma velocidade angular $\omega_i(t)$, com $i = 1, 2, \ldots, 4$, sendo estas consideradas como entradas de controle. Contrastando com a maioria dos veículos sobre rodas, a velocidade lateral do robô, v_y é geralmente diferente de zero. Tal propriedade deriva-se da estrutura mecânica do RMRD, fazendo com que o deslizamento lateral seja necessário caso o robô altere sua orientação. Portanto, as rodas são tangentes ao caminho apenas se $\omega = 0$, ou seja, o robô se move por uma linha reta.

Simplificando o movimento do robô, Pazderski e Kozłowski (2004) apresenta a seguinte relação:

$$v_{ix} = r_i \omega_i, \tag{2}$$

sendo v_{ix} componente longitudinal do vetor velocidade v_i expresso no sistema de coordenadas do robô, e r_i o raio de cada roda. E desta forma, para desenvolver um modelo cine-

mático é necessário considerar as quatro rodas juntas. Portanto, define-se $d_i = \begin{bmatrix} d_{ix} & d_{iy} \end{bmatrix}^T$ como sendo as distâncias do centro da roda i ao CIR decomposta em coordenadas de x e y; e $d_C = \begin{bmatrix} d_{Cx} & d_{Cy} \end{bmatrix}^T$ como a distância dentre o CM e o CIR, também decomposta em coordenadas de x e y, no sistema de coordenadas do robô, a partir do Centro Instantâneo de rotação. Com base na geometria do mesmo, a seguinte expressão pode ser deduzida:

$$\frac{v_{ix}}{-d_{iy}} = \frac{v_x}{-d_{Cy}} = \frac{v_{iy}}{d_{i_x}} = \frac{v_y}{d_{Cx}} = \omega. \tag{3}$$

Definindo as coordenadas do Centro Instantâneo de Rotação (CIR) no sistema de coordenadas do corpo como

$$CIR = (x_{CIR}, y_{CIR}) = (-d_{Cx}, -d_{Cy}), \tag{4}$$

é possível reescrever (3) como:

$$\frac{v_x}{y_{CIR}} = -\frac{v_y}{x_{CIR}} = \omega. \tag{5}$$

Em sua modelagem, Pazderski e Kozłowski (2004) afirma que as coordenadas dos vetores d_i satisfazem as seguintes relações:

$$\begin{aligned} d_{1y} = d_{2y} = d_{Cy} + c, \\ d_{3y} = d_{4y} = d_{Cy} - c, \\ d_{1x} = d_{4x} = d_{Cx} - a, \\ d_{2x} = d_{3x} = d_{Cx} + b, \end{aligned} \tag{6}$$

sendo a, b e c os parâmetros dimensionais cinemáticos expressos na Figura 2. Combinando as equações (3) e (6), a seguintes relações entre as velocidades das rodas são obtidas:

$$\begin{aligned} v_L = v_{1x} = v_{2x}, \\ v_R = v_{3x} = v_{4x}, \\ v_F = v_{2y} = v_{3y}, \\ v_B = v_{1y} = v_{4y}, \end{aligned} \tag{7}$$

sendo v_L e v_R coordenadas longitudinais das velocidades das rodas esquerdas e direitas. Já v_F e v_B são as coordenadas laterais das velocidades das rodas dianteiras e traseiras.

Fazendo o uso das equações (3)-(7), obtém-se a seguinte relação entre as velocidades das rodas e a velocidade do robô:

$$\begin{bmatrix} v_L \\ v_R \\ v_F \\ v_B \end{bmatrix} = \begin{bmatrix} 1 & -c \\ 1 & c \\ 0 & -x_{CIR} + b \\ 0 & -x_{CIR} - a \end{bmatrix} \begin{bmatrix} v_x \\ \omega \end{bmatrix}. \tag{8}$$

Assumindo que o raio r_i é o mesmo para todas as rodas, $r_i = r$ e utilizando as equações (2) e (8) é possível escrever:

$$\omega_w = \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix} = \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix} = \begin{bmatrix} \omega_3 \\ \omega_4 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} v_L \\ v_R \end{bmatrix}, \tag{9}$$

sendo ω_L e ω_R as velocidades angulares das rodas esquerda e direita, respectivamente.

Conforme apresentado por Cui et al. (2017), define-se i_L e i_R como a razão de escorregamento das rodas esquerda e direita de um robô móvel, em que

$$i_L = \frac{r\omega_L - v_L}{r\omega_L}, i_R = \frac{r\omega_R - v_R}{r\omega_R}, \tag{10}$$

sendo v_L e v_R as velocidades lineares das rodas esquerda e direita do robô móvel com escorregamento nas rodas.

Neste sentido é possível reescrever (10) da seguinte forma

$$\begin{aligned} r\omega_L i_L = r\omega_L v_L \Rightarrow v_L = r\omega_L(1 - i_L), \\ r\omega_R i_R = r\omega_r v_R \Rightarrow v_R = r\omega_R(1 - i_R). \end{aligned} \tag{11}$$

A partir de (8), tem-se

$$v_L = v_x - c\omega, \tag{12}$$

$$v_R = v_x + c\omega. \tag{13}$$

Somando (12) e (13), tem-se

$$v_L + v_R = 2v_x. \tag{14}$$

Substituindo (11) em (14), obtém-se

$$v_x = r\frac{\omega_L(1 - i_L) + \omega_R(1 - i_R)}{2}. \tag{15}$$

Subtraindo (12) e (13), tem-se

$$v_L - v_R = 2c\omega. \tag{16}$$

Substituindo (11) em (16), obtém-se

$$\omega = r\frac{-\omega_L(1 - i_L) + \omega_R(1 - i_R)}{2c}. \tag{17}$$

A partir da combinação de (15) e (17) é possível aproximar-se a uma relação entre velocidades angulares das rodas e a velocidade do robô, e pode ser considerada como uma nova entrada de controle em nível cinemático (PAZDERSKI; KOZŁOWSKI, 2004).

$$\begin{bmatrix} v_x \\ \omega \end{bmatrix} = r \begin{bmatrix} \frac{\omega_L(1-i_L)+\omega_R(1-i_R)}{2} \\ \frac{-\omega_L(1-i_L)+\omega_R(1-i_R)}{2c} \end{bmatrix}. \tag{18}$$

Desta maneira, por sua estrutura, o robô móvel é impedido de mover-se na direção de Y_{CM} e como forma de completar o modelo cinemático, esta restrição não holonômica é introduzida por (CARACCIOLO; LUCA; IANNITTI, 1999):

$$v_y = -x_{CIR}\dot{\psi}, \tag{19}$$

e como forma de manter a estabilidade do movimento x_{CIR} é limitado ao intervalo $(-b, a)$.

A partir de (1), tem-se que

$$v_y = -\text{sen}\psi\dot{x} + \cos\psi\dot{y}. \tag{20}$$

Substituindo (20) em (19) tem-se que

$$-\text{sen}\psi\dot{x} + \cos\psi\dot{y} + x_{CIR}\psi = 0. \tag{21}$$

Ou utilizando a forma matricial de Pfaffian, tem-se que

$$\begin{bmatrix} -\text{sen}\psi & \cos\psi & x_{CIR} \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = A(q)\dot{q} = 0, \tag{22}$$

sendo $q = [x\ y\ \psi]^T$.

Neste sentido, as velocidades $\dot{q}$ podem ser expressas como

$$\dot{q} = S(q)\eta, \tag{23}$$

sendo $\eta = [\omega_L\, \omega_R]^T$ e $S(q)$ uma matriz cujas colunas estão no espaço nulo de $A(q)$, ou seja

$$S^T(q)A^T(q) = 0, \tag{24}$$

e

$$S(q) = \begin{bmatrix} \frac{r}{2}(1-i_L)(\cos\psi - \frac{x_{CIR}}{c}\text{sen}\psi) & \frac{r}{2}(1-i_R)(\cos\psi + \frac{x_{CIR}}{c}\text{sen}\psi) \\ \frac{r}{2}(1-i_L)(\text{sen}\psi + \frac{x_{CIR}}{c}\cos\psi) & \frac{r}{2}(1-i_R)(\text{sen}\psi - \frac{x_{CIR}}{c}\cos\psi) \\ \frac{r}{2c}(1-i_L) & \frac{r}{2c}(1-i_R) \end{bmatrix}. \tag{25}$$

3.1.2 Modelagem Dinâmica

Segundo Pazderski e Kozłowski (2004), os efeitos dinâmicos desempenham um papel importante para os RMRD, fundamentalmente causados pelo escorregamento lateral e forças de interação com o solo. Neste sentido, esta seção descreverá as suas propriedades dinâmicas. Portanto, em um primeiro momento as forças atuantes sobre as rodas serão descritas, como na Figura 3.

A força de ação F_i e a força de reação N_i estão relacionadas ao torque na roda e a gravidade, respectivamente. Observa-se que F_i é linearmente dependente da entrada de controle da roda, ou seja,

$$F_i = \frac{\tau_i}{r}. \tag{26}$$

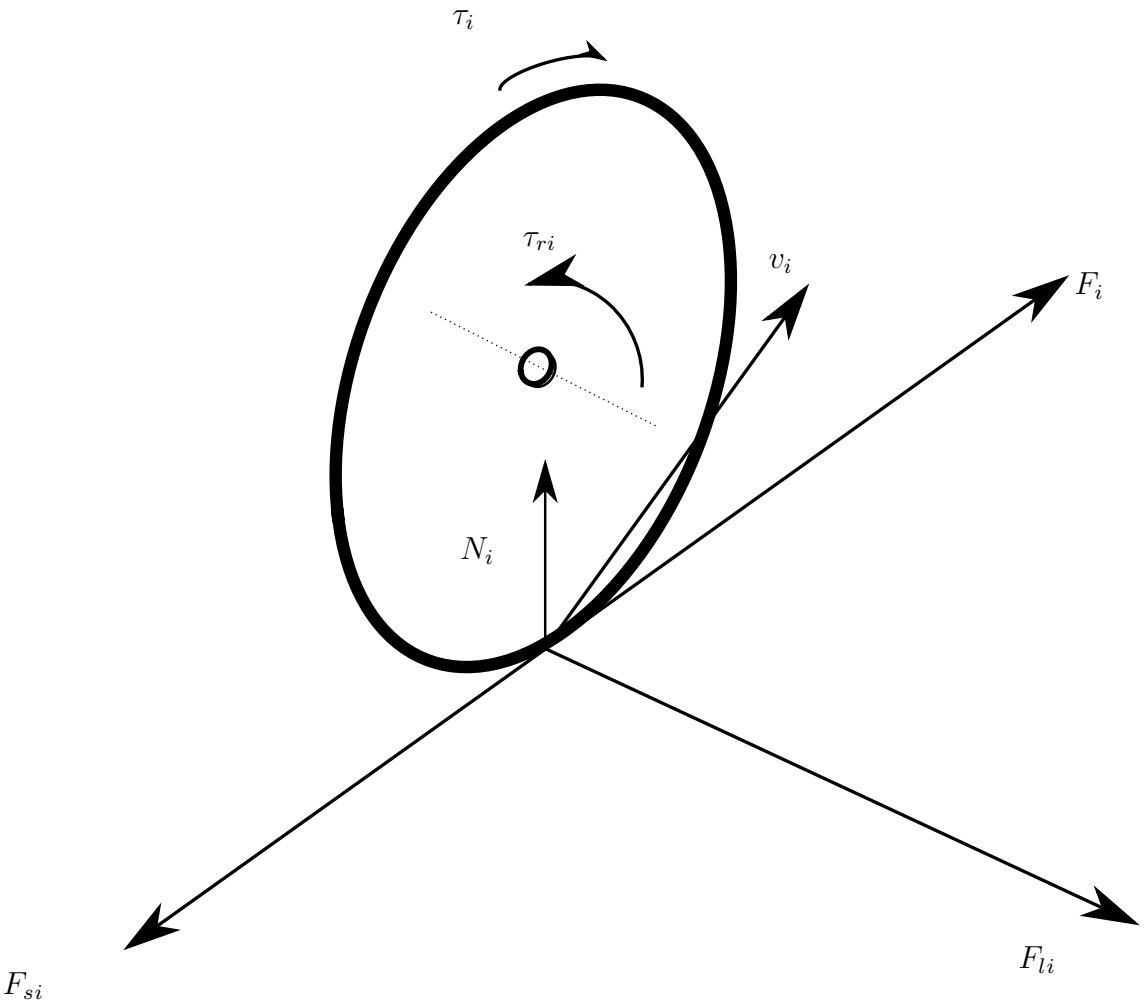


Figura 3 – Forças Atuantes nas Rodas do RMRD.

De acordo com Pazderski e Kozłowski (2004), assume-se que a força vertical N_i atua da superfície para o centro da roda, considerando as quatro rodas do robô (Figura 4) e desconsiderando propriedades dinâmicas adicionais, obtém-se as seguintes equações de equilíbrio:

$$\begin{aligned} N_1 &= N_2, \\ N_4 &= N_3, \\ \sum_{i=4}^{4} N_i &= mg, \end{aligned} \tag{27}$$

sendo m a massa do robô e g a aceleração da gravidade. Uma vez que existe simetria ao longo da linha média longitudinal, obtém-se

$$\begin{aligned} N_1 = N_4 &= \frac{b}{2(a+b)} mg, \\ N_2 = N_3 &= \frac{a}{2(a+b)} mg. \end{aligned} \tag{28}$$

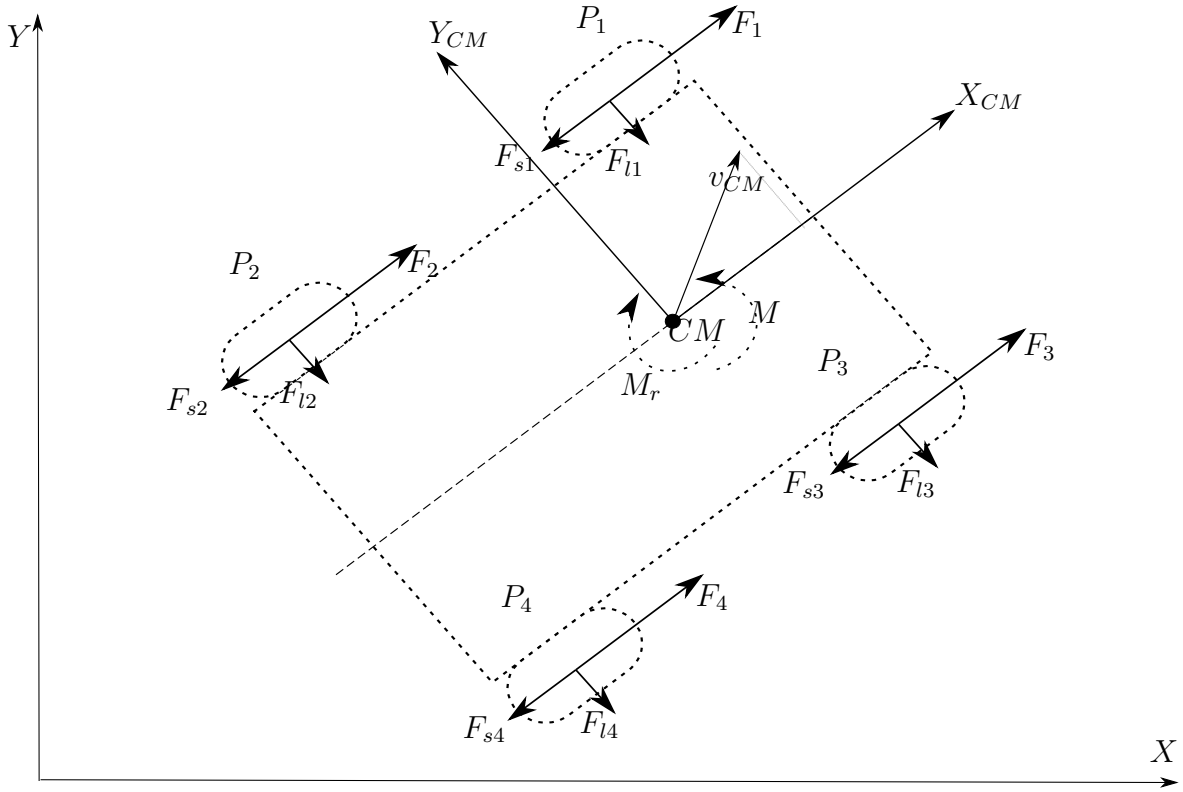


Figura 4 – Forças Ativas e Resistivas Atuantes no RMRD.

Define-se o vetor F_{si} como resultado do momento de resistência à rolagem τ_{ri} e o vetor F_{li} como a força de reação lateral. Desta forma, utilizando da fórmula de Lagrange-Euler, com multiplicadores de Lagrange como forma de incluir as constantes não-holonômicas (19), obtém-se a equação dinâmica do robô. Em seguida, devido ao movimento planar, assume-se que a energia potencial do RMRD é nula. Por conseguinte, o Lagrangiano L do sistema é igual à energia cinética:

$$L(q, \dot{q}) = T(q, \dot{q}). \tag{29}$$

Levando em consideração a energia cinética do veículo de desprezando a energia rotativa das rodas, Pazderski e Kozłowski (2004) define a seguinte equação:

$$T = \frac{1}{2} m v^T v + \frac{1}{2} I \omega^2, \tag{30}$$

sendo m a massa do robô, I o momento de inércia do mesmo, no CM. Como simplificação, assume-se que a distribuição de massa seja homogênea. Uma vez que $v^T v = v_x^2 + v_y^2 = \dot{x}^2 + \dot{y}^2$ é possível reescrever (30) da seguinte forma

$$T = \frac{1}{2} m(\dot{x}^2 + \dot{y}^2) + \frac{1}{2} I \dot{\psi}^2. \tag{31}$$

Calcula-se a derivada parcial da energia cinética e os seus derivados de tempo, as forças

inerciais podem ser obtidas como

$$\frac{d}{dt}\left(\frac{\partial E_c}{\partial \dot{q}}\right) = \begin{bmatrix} m\ddot{x} \\ m\ddot{y} \\ I\ddot{\psi} \end{bmatrix} = M\ddot{q}, \tag{32}$$

sendo

$$M = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I \end{bmatrix}. \tag{33}$$

Ainda segundo Pazderski e Kozłowski (2004), as forças que causam dissipação de energia devem ser consideradas, sendo assim, de acordo com a Figura 4 as seguintes forças resultantes são calculadas:

$$F_{rx}(\dot{q}) = \cos\psi \sum_{i=1}^{4} F_{si}(v_{xi}) - \text{sen}\psi \sum_{i=1}^{4} F_{li}(v_{yi}), \tag{34}$$

$$F_{ry}(\dot{q}) = \text{sen}\psi \sum_{i=1}^{4} F_{si}(v_{xi}) + \cos\psi \sum_{i=1}^{4} F_{li}(v_{yi}). \tag{35}$$

Já o momento de resistência em torno do CM M_r é obtido por

$$\begin{aligned} M_r(\dot{q}) = &- a \sum_{i=1,4} F_{li}(v_{yi}) + b \sum_{i=2,3} F_{li}(v_{yi}) \\ &+ c[\sum_{i=3,4} F_{si}(v_{xi}) - \sum_{i=1,2} F_{si}(v_{xi})]. \end{aligned} \tag{36}$$

Partindo de (34) e (36) é possível introduzir o vetor

$$R(\dot{q}) = \begin{bmatrix} F_{rx}(\dot{q}) & F_{ry}(\dot{q}) & M_r(\dot{q}) \end{bmatrix}. \tag{37}$$

As forças de ação gerada pelos quatro atuadores, que fazem com que o robô se mova, podem ser definidas

$$F_x = \cos\psi \sum_{i=1}^{4} F_i, \tag{38}$$

$$F_y = \text{sen}\psi \sum_{i=1}^{4} F_i. \tag{39}$$

O torque no CM é obtido por

$$M = c(F_3 + F_4 - F_1 - F_2). \tag{40}$$

Desta forma, o vetor das forças de ação, possui a seguinte forma:

$$F = \begin{bmatrix} F_x & F_y & M \end{bmatrix}^T. \tag{41}$$

Partindo de (26), (38) - (40) e assumindo que todas as rodas possuem o mesmo raio

$$F = \frac{1}{r}\begin{bmatrix} \cos\psi \sum_{i=1}^{4} \tau_i \\ \text{sen}\psi \sum_{i=1}^{4} \tau_i \\ c(\tau_3 + \tau_4 - \tau_1 - \tau_2) \end{bmatrix}. \tag{42}$$

Como forma de simplificar a notação, define-se a nova entrada de torque de controle τ como

$$\tau = \begin{bmatrix} \tau_L \\ \tau_R \end{bmatrix} = \begin{bmatrix} \tau_1 + \tau_2 \\ \tau_3 + \tau_4 \end{bmatrix}, \tag{43}$$

sendo τ_L e τ_R o torque produzido pelas rodas do lado equerdo e direito do RMRD, respectivamente. A partir da combinação de (42) e (43), obtém-se

$$F = B(q)\tau, \tag{44}$$

sendo B a matriz de transformação de entrada definida da seguinte forma

$$B(q) = \frac{1}{r}\begin{bmatrix} \cos\psi & \cos\psi \\ \text{sen}\psi & \text{sen}\psi \\ -c & c \end{bmatrix}. \tag{45}$$

Desta forma, usando de (32), (37) e (44), obtém-se o modelo dinâmico

$$M(q)\ddot{q} + R(\dot{q}) = B(q)\tau. \tag{46}$$

Porém o modelo expresso em (46) descreve apenas a dinâmica do corpo livre e não incluí restrições não holonômicas, como expressa em (20). Como forma de resolver este impasse, Pazderski e Kozłowski (2004) sugere a introdução de um vetor com multiplicadores de Lagrange λ, da seguinte maneira

$$M(q)\ddot{q} + R(\dot{q}) = B(q)\tau + A^T(q)\lambda. \tag{47}$$

Para os propósitos dos controladores, é melhor apresentado expressar (47) em termos do vetor de velocidades internas η. Sendo assim, multiplicando (47) por $S^T(q)$ obtém-se

$$S^T(q)M(q)\ddot{q} + S^T(q)R(\dot{q}) = S^T(q)B(q)\tau + S^T(q)A^T(q)\lambda. \tag{48}$$

Em seguida, derivando (23) em relação ao tempo, obtém-se

$$\ddot{q} = \dot{S}(q)\eta + S(q)\dot{\eta}. \tag{49}$$

Partindo de (49) e (23) em (47) resulta na seguinte equação dinâmica

$$\overline{M}\dot{\eta} + \overline{C}\eta + \overline{R} = \overline{B}\tau, \tag{50}$$

sendo

$$\overline{M} = S^T M S = \begin{bmatrix} m & 0 \\ 0 & mx_{CIR}^2 + I \end{bmatrix}, \tag{51}$$

$$\overline{C} = S^T M \dot{S} = m x_{CIR} \begin{bmatrix} 0 & \dot{\psi} \\ -\dot{\psi} & \dot{x}_{CIR} \end{bmatrix}, \tag{52}$$

$$\overline{R} = S^T R = \begin{bmatrix} F_{rx}(\dot{q}) \\ x_{CIR} F_{ry}(\dot{q}) + M_r \end{bmatrix}, \tag{53}$$

$$\overline{B} = S^T B = \frac{1}{r} \begin{bmatrix} 1 & 1 \\ -c & c \end{bmatrix}. \tag{54}$$

Isolando $\dot{\eta}$ e adicionando o distúrbio de torque $w = [w_L \; w_R]^T$ na (50), resulta em

$$\dot{\eta} = -\overline{M}^{-1}\overline{N}\eta - \overline{M}^{-1}\overline{R} + \overline{M}^{-1}\overline{B}\tau + \overline{M}^{-1}\overline{B}w. \tag{55}$$

Adicionando e subtraindo $\dot{\eta^d}$ and $\overline{M}^{-1}\overline{N}\eta^d$, respectivamente. Define-se o estado como sendo

$$\tilde{x} = \begin{bmatrix} \omega_L - \omega_L^d \\ \omega_R - \omega_R^d \\ \delta_L - \delta_L^d \\ \delta_R - \delta_R^d \end{bmatrix}, \tag{56}$$

em que ω_L e ω_R são as velocidades angulares nas rodas esquerda e direita do robô móvel ;ω_L^d e ω_R^d as velocidades angulares desejadas rodas esquerda e direita do robô móvel; δ_L e δ_R os deslocamentos das rodas esquerda e direita; δ_L^d e δ_R^d os deslocamentos desejados das rodas esquerda e direita do robô móvel; a representação em espaço de estado para o problema de acompanhamento de trajetória do robô é dado por

$$\dot{\tilde{x}} = \begin{bmatrix} A(\eta) & 0 \\ I & 0 \end{bmatrix} \tilde{x} + \begin{bmatrix} I \\ 0 \end{bmatrix} u + \begin{bmatrix} E \\ 0 \end{bmatrix} w, \tag{57}$$

em que

$$A(q) = \overline{M}^{-1}\overline{N}, \tag{58}$$

$$B = \overline{M}^{-1}\overline{B}, \tag{59}$$

$$\tau = E^{-1}(u + \dot{\eta}^d - A(q)\eta^d + \overline{M}^{-1}\overline{R}). \tag{60}$$

3.1.3 Representação em Espaço de Estado para o Controle não linear $\mathcal{H}_\infty$

Nesta seção será apresentada a representação em espaço de estados para o problema de controle $\mathcal{H}_\infty$ não linear para acompanhamento de trajetória de robôs móveis, como

apresentado em (CHEN; CHANG; LEE, 1997). Considere a seguinte transformação de estado

$$T_0\tilde{x} = \begin{bmatrix} T_{11} & T_{12} \\ 0 & I \end{bmatrix} \begin{bmatrix} \dot{\tilde{q}} \\ \tilde{q} \end{bmatrix}, \tag{61}$$

em que T_{11}, $T_{12} \in \Re^{2\times 2}$ matrizes constantes a serem determinadas. Supõe-se que T_{11} seja diagonal, ou seja, $T_{11} = t_{11}I$.

Define-se

$$A_T = \begin{bmatrix} A(\eta) & 0 \\ T_{11}^{-1} & -T_{11}^{-1}T_{12} \end{bmatrix} T_0, \tag{62}$$

$$E_T = T_0^{-1} \begin{bmatrix} E \\ 0 \end{bmatrix}, \tag{63}$$

$$F = E^{-1}(T_{11}^{-1}T_{12}\dot{\tilde{q}} - T_{11}^{-1}A(\eta)T_{12}\tilde{q} - \dot{\eta}^d + A(\eta)\eta^d - \overline{M}^{-1}\overline{R}). \tag{64}$$

Aplicando a transformação de estados (61) em (57) é possível reescrever a aplicação da transformação de estados da seguinte forma

$$\dot{\tilde{x}} = A_T\tilde{x} + E_T T_{11}(F + \Delta F + \tau) + E_T w. \tag{65}$$

Define-se como incertezas modeladas do sistema

$$\mathcal{U} = U + \Delta U, \tag{66}$$

$$\mathcal{V} = V + \Delta V, \tag{67}$$

sendo $\mathcal{U} = A(\eta)$ e $\mathcal{V} = \bar{M}^{-1}\bar{R}$, tem-se que

$$\Delta F = E^{-1}(-T_{11}\Delta U T_{11}\tilde{q} + \Delta U\eta^d - \Delta V). \tag{68}$$

Portanto as incertezas do sistema podem ser descritas na forma

$$\Delta F = T_{11}^{-1}\Delta U(-T_{11}^{-2}T_{12})\ddot{\tilde{q}} + \kappa, \tag{69}$$

em que κ são as incertezas não modeladas.

3.2 Controle

Nesta seção serão apresentados os conceitos básicos referentes às técnicas de controle utilizadas no trabalho. Inicialmente será apresentada a forma de cálculo de trajetórias de referência para robôs móveis. Em seguida, será apresentado um controle baseado no modelo cinemático para percorrer a trajetória de referência gerada anteriormente respeitando suas restrições. Na sequência, serão introduzidos os métodos de controle baseado na dinâmica do robô, o controle $\mathcal{H}_\infty$ não linear baseado em redes neurais.

3.2.1 Trajetória de Referência

Nesta seção serão apresentados como foram definidas as trajetórias de referência utilizadas durante o trabalho. Tais trajetórias podem ser geradas por uma função matemática. Inicialmente trabalhou-se com a resolução de um polinômio do quinto grau para gerar trajetórias entre dois pontos. Em seguida, como forma de elevar o grau de incertezas e distúrbios dos controladores, optou-se por gerar uma trajetória circular.

3.2.1.1 Trajetória Linear

Nesta seção será apresentado como é possível definir a trajetória de referência do robô móvel através de um polinômio de quinto grau, apresentado em Inoue, Siqueira e Terra (2009a). O polinômio é descrito por

$$q_i^{ref}(t) = a_{i0} + a_{i1}\Delta t + a_{i2}\Delta t^2 + a_{i3}\Delta t^3 + a_{i4}\Delta t^4 + a_{i5}\Delta t^5, \tag{70}$$

com $\Delta t = t - t_0$ e $i = 1, 2, ..., l$, sendo l o número de eixos que se deseja gerar trajetórias de referência. Suas derivadas são dadas por

$$\dot{q}_i^{ref}(t) = a_{i1} + 2a_{i2}\Delta t + 3a_{i3}\Delta t^2 + 4a_{i4}\Delta t^3 + 5a_{i5}\Delta t^4, \tag{71}$$

$$\ddot{q}_i^{ref}(t) = 2a_{i2}\Delta t + 6a_{i3}\Delta t + 12a_{i4}\Delta t^2 + 20a_{i5}\Delta t^3, \tag{72}$$

$$\dddot{q}_i^{ref}(t) = 6a_{i3}\Delta t + 24a_{i4}\Delta t + 60a_{i5}\Delta t^2, \tag{73}$$

da forma que as seguintes condições sejam satisfeitas:

$$q_i^{ref}(t_0) = q_{i0}, \qquad q_i^{ref}(t_f) = q_{if}, \tag{74}$$

$$\dot{q}_i^{ref}(t_0) = \dot{q}_{i0}, \qquad \dot{q}_i^{ref}(t_f) = \dot{q}_{if}, \tag{75}$$

$$\ddot{q}_i^{ref}(t_0) = \ddot{q}_{i0}, \qquad \ddot{q}_i^{ref}(t_f) = \ddot{q}_{if}, \tag{76}$$

$$\dddot{q}_i^{ref}(t_0) = \dddot{q}_{i0}, \qquad \dddot{q}_i^{ref}(t_f) = \dddot{q}_{if}. \tag{77}$$

Desse modo, os parâmetros do polinômio de quinto grau podem ser obtidos pela equação

$$a_i = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & \Delta t & \Delta t^2 & \Delta t^3 & \Delta t^4 & \Delta t^5 \\ 0 & 1 & 2\Delta t & 3\Delta t^2 & 4\Delta t^3 & 5\Delta t^4 \\ 0 & 0 & 2 & 6\Delta t & 12\Delta t^2 & 20\Delta t^3 \end{bmatrix}^{-1} q_i, \tag{78}$$

sendo $q_i = [q_{i0} \quad \dot{q}_{i0} \quad \ddot{q}_{i0} \quad q_{if} \quad \dot{q}_{if} \quad \ddot{q}_{if}]^T$ e $a_i = [a_{i0} \quad a_{i1} \quad a_{i2} \quad a_{i3} \quad a_{i4} \quad a_{i5}]^T$.

Para o robô aqui descrito, utiliza-se de $l = 2$, pois deseja-se gerar trajetórias para o eixo X e Y, considerando a condição inicial x_0, $\dot{x}_0$, $\ddot{x}_0$, y_0, $\dot{y}_0$, $\ddot{y}_0$ e a condição final x_f, $\dot{x}_f$, $\ddot{x}_f$, y_f,$\dot{y}_f$, $\ddot{y}_f$. Variando as equações em (71)-(73) no tempo t obtém-se a trajetória de referência do robô $x^{ref}(t)$, $y^{ref}(t)$ além de suas derivadas à cada instante.

3.2.1.2 Trajetória Circular

Nesta Seção será apresentada como é definida uma trajetória de referência do AMR em formato circular, como apresentado por Khalil (2002), possui formato circular no plano xy, baseada em um ciclo-limite, descrita pelas equações

$$\begin{aligned} \dot{r} &= \rho r - r^2, \\ \dot{\psi} &= \omega_s, \end{aligned} \tag{79}$$

sendo: r o raio da trajetória de referência, em relação a origem do sistema de coordenada inercial - para a simulação $r = 2m$; $\dot{\psi}$ a velocidade angular da trajetória de referência; ρ uma constante correspondente ao raio escolhido para o círculo; ω_s uma constante correspondente à velocidade angular definida como $\omega_s = v_r/\rho$, em que $v_r = 0.2m/s$ é a velocidade linear de referência.

A partir da integração das variáveis $\dot{r}$ e $\dot{\psi}$ e da utilização das relações trigonométricas $x^{ref} = r\cos\psi$ e $y^{ref} = r\text{sen}\psi$ obtém-se $q^{ref} = [x^{ref}, x^{ref}, \psi^{ref}]$. Quanto à orientação de referência ψ^{ref}, pode ser calculada como

$$\psi^{ref} = \arctan\left(\frac{\dot{y}}{\dot{x}}\right). \tag{80}$$

3.2.2 Controle baseado na cinemática

Nesta seção, será apresentado o controle baseado na cinemática apresentado por Kanayama et al. (1990), capaz de gerar as velocidades desejadas para o problema de acompanhamento de trajetória de referência de robôs móveis. Definindo que o erro $q_e = [x_e \ y_e \ \psi_e]^T$, entre a postura de referência $P^{ref} = [x^{ref} \ y^{ref} \ \psi^{ref}]^T$ e a postura real do RMRD $P_o = [x_o \ y_o \ \psi_o]^T$ como sendo

$$\begin{aligned} x_e &= \cos\psi(x^{ref} - x_o) + \text{sen}\psi(y^{ref} - y_o), \\ y_e &= -\text{sen}\psi(x^{ref} - x_o) + \cos\psi(y^{ref} - y_o), \\ \psi_e &= \psi^{ref} - \psi_o, \end{aligned} \tag{81}$$

sendo $q^{ref} = [x^{ref} \ y^{ref}]^T$ a trajetória de referência escolhida e $\psi^{ref} = \tan^{-1}(\dot{y}^{ref}/\dot{x}^{ref})$. Neste sentido, as velocidades linear e angular desejadas, respectivamente v^d e ω^d são dadas por

$$v^d = v^{ref}\cos\psi_e + k_x x_e, \tag{82}$$

$$\omega^d = \omega^{ref} + v^{ref}(k_y y_e + k_\psi \text{sen}\psi_e), \tag{83}$$

sendo $k_x,\ k_y,\ k_\psi$ constantes, e

$$v^{ref} = \sqrt{(\dot{x}^{ref})^2 + (\dot{y}^{ref})^2}, \tag{84}$$

$$\omega^{ref} = \dot{\psi}^{ref}. \tag{85}$$

No decorrer deste trabalho, o controlador baseado no modelo dinâmico do robô, são resolvidos levando em consideração as velocidades angulares das rodas, $\dot{q}_2^d$. Desta forma, define-se a relação de velocidades que segue

$$\dot{q}_2^d = \begin{bmatrix} \dot{\theta}_R^d \\ \dot{\theta}_L^d \end{bmatrix} = \begin{bmatrix} 1/r & b/r \\ 1/r & -b/r \end{bmatrix} \begin{bmatrix} v^d \\ \omega^d \end{bmatrix}, \tag{86}$$

sendo $\dot{\theta}_R^d$ e $\dot{\theta}_L^d$ as velocidades angulares desejadas das rodas direita e esquerda, respectivamente.

Neste sentido, o erro de velocidade das rodas é calculado seguindo

$$\tilde{x} = \begin{bmatrix} \omega_e \\ \delta_e \end{bmatrix}, \tag{87}$$

em que

$$\omega_e = \begin{bmatrix} \omega_L - \omega_L^d \\ \omega_R - \omega_R^d \end{bmatrix}, \delta_e = \begin{bmatrix} \delta_L - \delta_L^d \\ \delta_R - \delta_R^d \end{bmatrix} \approx S^+ \begin{bmatrix} x - x^{ref} \\ y - y^{ref} \\ \psi - \psi^{ref} \end{bmatrix},$$

e S^+ como a pseudo inversa de (25).

3.2.3 Controle $\mathcal{H}_\infty$ Não Linear baseado em aprendizado profundo

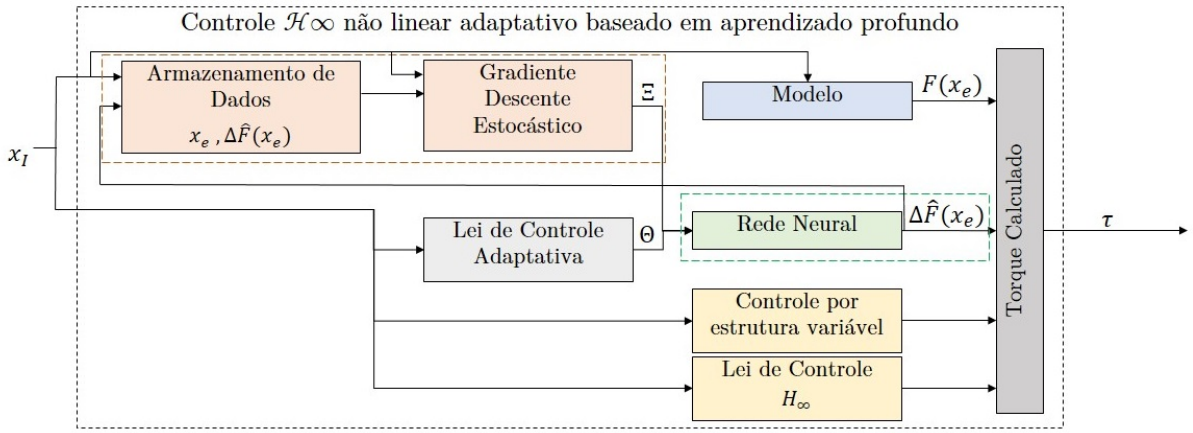


Figura 5 – Controle $\mathcal{H}_\infty$ não linear adaptativo baseado em aprendizado profundo

Nesta seção, será proposto o controlador $\mathcal{H}_\infty$ não linear DMRAC, como apresentado na Figura 5. Seu funcionamento é baseado no Controle $\mathcal{H}_\infty$ Não linear apresentado em Taveira, Siqueira e Terra (2006), para tratar os distúrbios externos e uma rede neural profunda para estimar as incertezas do sistema, conforme apresentado em Joshi, Chowdhary e Waanders (2021).

Neste sentido, tem-se que o bloco *Armazenamento de Dados* armazena *online* os valores de $\{x_e\ ,\ \Delta\widehat{F}(x_e)\}$ que são utilizados no processo de treinamento da rede neural. O vetor de entrada x_I pode ser descrito como

$$x_I = \left[x_e^T\ \Delta\widehat{F}(x_e)^T\right]^T, \tag{88}$$

em que $x_e = \left[\tilde{\omega}_L, \tilde{\omega}_R, \tilde{x}, \tilde{y}, \tilde{\psi}, \omega_L, \omega_R\right]^T$, $\tilde{\omega}_L$ e $\tilde{\omega}_R$ são os erros de velocidade das rodas esquerda e direita; $\tilde{x}$, $\tilde{y}$ e $\tilde{\psi}$ são erros de posição e orientação, respectivamente; ω_L e ω_R são as velocidades agulares atuais das rodas esquerda e direita. E $\Delta\widehat{F}(x_e)$é a estimativa das incertezas da Equação do sistema (68) dada pela Deep Neural Network (DNN) descrita na Seção 3.2.5.

O bloco *Rede Neural* fornece a rede neural $\Xi\Theta$ que estima a incerteza $\Delta\widehat{F}(x_e)$. O vetor Θ possui as principais equações da DNN, e a matriz Ξ possui parâmetros de escala.

O treinamento *offline* da DNN utiliza um conjunto de treinamento composto por $\{x_e\ , F(x_e)\}$. $F(x_e)$ é calculado usando a Equação 64. O conjunto de treinamento pode ser obtido através de um experimento usando, por exemplo, apenas a abordagem de controle não linear $\mathcal{H}_\infty$.

Além disso, agora, com base em Joshi, Chowdhary e Waanders (2021), um processo *online* de aprendizado rápido e lento para DNN para o Controle Não-linear $\mathcal{H}_\infty$ é proposto. O processo de aprendizado rápido é baseado em uma lei de controle adaptativo (bloco *Lei de Controle Adaptativo*) que sintoniza a DNN através de Θ. Além disso, o processo de aprendizado lento é baseado no algoritmo de *Stochastic Gradient Descent* - Gradiente Descente Estocástico (SGD) (bloco *Gradiente Descente Estocástico*), que fornece uma regra de atualização baseada em uma aproximação estocástica do valor de gradiente esperado da função de perda para o treinamento.

O bloco *Lei de Controle por Estrutura Variável* é utilizado para eliminar o efeito da estimativa da matriz Ξ. O bloco *Lei de Controle* $\mathcal{H}_\infty$ calcula a lei de controle não linear $\mathcal{H}_\infty$. E o bloco *Modelo* calcula a dinâmica F, Equação 64.

E por fim, o bloco *Torque calculado* calcula o torque τ que será enviado ao robô móvel.

Desta forma, na Seção 3.2.4 apresentamos o Controle $\mathcal{H}_\infty$ Não Linear baseado em DNN. E na Seção 3.2.5 apresentamos detalhes da DNN usado para o processo de aprendizado profundo.

3.2.4 Controle $\mathcal{H}_\infty$ Adaptativo Não Linear baseado em Rede Neural Profunda

O $\mathcal{H}_\infty$ não linear baseado em redes neurais profundas para sistemas robóticos, descrito em Lee e Ge (1998) e Chang (2000), pode ser formulado como: dado um nível de atenuação $\gamma > 0$ e uma matriz de ponderação $Q = Q^T > 0$. Se existe uma matriz definida positiva simétrica $K = K^T > 0$, e uma matriz T_0, satisfazendo a seguinte equação algébrica tipo

Riccati

$$\begin{bmatrix} 0 & K \\ K & 0 \end{bmatrix} - T_0^T B \left(R^{-1} - \frac{1}{\gamma^2} I \right)^{-1} B^T T_0 + Q = 0, \tag{89}$$

em que R é uma matriz de ganho tal que $R < \gamma^2 I$ e $B = [I \mid 0]^T$, então um modelo misto/adaptativo inteligente lei de controle $\mathcal{H}_\infty$ pode ser formulada da seguinte forma,

$$\tau = F(x_e) + \Xi\Theta + T_{11}^{-1} u + u_s, \tag{90}$$

com

$$\dot{\Theta} = Proj[-Z^{-1}\Xi^T T_{11}[I\ 0]T_0\tilde{x}], \tag{91}$$

$$u = -R^{-1}[I\ 0]T_0\tilde{x}, \tag{92}$$

$$u_s = -k(x_e)sgn(T_{11}[I\ 0]T_0\tilde{x}), \tag{93}$$

em que Z é o ganho adaptativo. E o seguinte índice de desempenho seja satisfeito

$$\int_0^T (\tilde{x}^T Q\tilde{x} + u^T Ru)dt \leq \tilde{x}^T(0)P\tilde{x}(0) + \tilde{\Theta}^T(0)Z\tilde{\Theta}(0) + \gamma^2 \int_0^T (d^T d)dt,$$

tendo que $Q = Q^T > 0$, $R = R^T > 0$, $P = P^T > 0$ *e* $Z = Z^T > 0$, matrizes de ponderação. $\tilde{\Theta} = \Theta - \Theta^*$ denota o erro de estimativa dos parâmetros neurais. O termo u_s em (96) é o controle de estrutura variável usado para eliminar o efeito do erro de aproximação. É possível afirmar em (96), o termo $\Xi\Theta$ é responsável por estimar $F(x_e)$ e $\Delta F(x_e)$.

3.2.5 Rede Neural Profunda

Nesta seção, a arquitetura da DNN $\Delta\hat{F}(x_e, \Theta)$ será definida coforme apresentado na Figura 6. Para tal, a DNN possui como entrada x_I, duas camadas ocultas, $H1$ e $H2$ com $p = 100$ neurônios em cada, além de $n = 2$ saídas. A sua função de ativação é do tipo tangente hiperbólica e pode ser destritas como (94)

$$tanh(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}. \tag{94}$$

O *bias* da primeira e segunda camadas ocultas foi definido como $b1$ e $b2$, respectivamente. E a saída do DNN é definida como $\xi_k\Theta_k$, onde $k = \{1, \ldots, n\}$, ξ_k é a saída do DNN, e Θ_k é um parâmetro de ajuste de escala.

Desta forma, a DNN completa é denotada por

$$\Delta\hat{F}(x_I, \Theta) = \begin{bmatrix} \Delta\hat{F}_1(x_I, \Theta_1) \\ \vdots \\ \Delta\hat{F}_n(x_I, \Theta_n) \end{bmatrix} = \begin{bmatrix} \xi_1\Theta_1 \\ \vdots \\ \xi_n\Theta_n \end{bmatrix} = \Xi\Theta. \tag{95}$$

Neste trabalho, a DNN será implementada usando o *Mathworks Matlab Neural Network toolbox* para construir e treinar uma rede neural adequada. Para tal, a DNN aqui utilizada

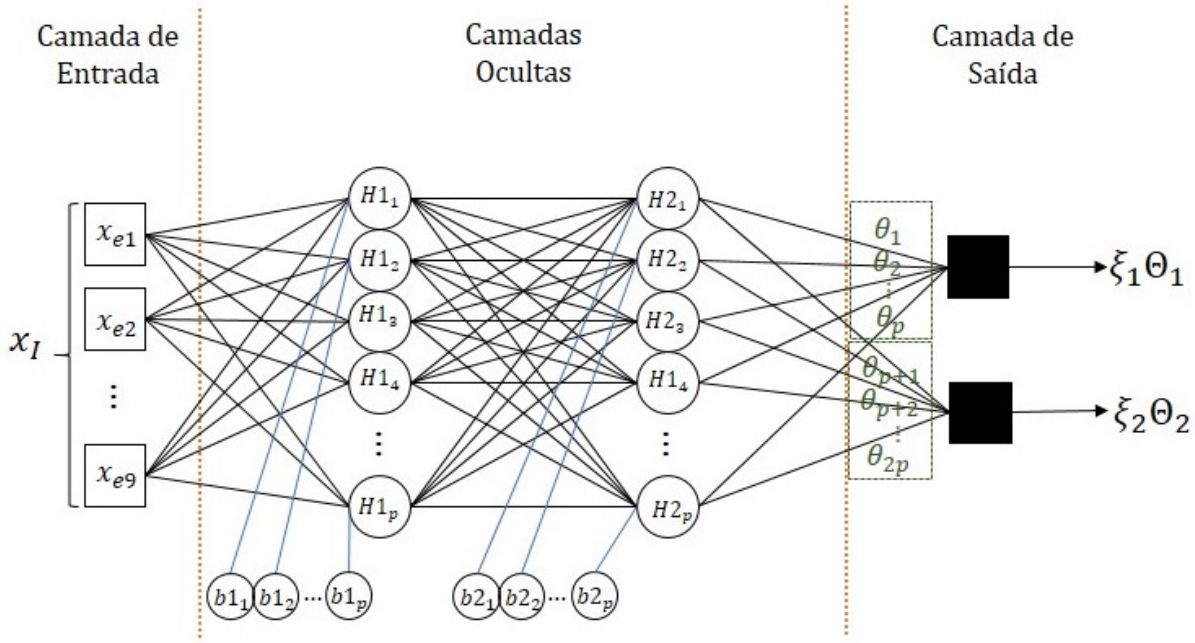


Figura 6 – Arquitetura da DNN

é uma rede *feedforward* multicamadas com duas camadas ocultas, e as conexões entre os nós não formam um ciclo. Também definiu-se função de treinamento como SGD, então os pesos e *bias* são atualizados na direção do negativo gradiente da função de desempenho (MATLAB, 2020).

Segundo Kim (2017), o SGD é um esquema para calcular a atualização de peso disponível para aprendizado supervisionado da rede neural. Ele calcula o erro para cada dado de treinamento e o ajusta imediatamente. Para a simulação apresentada no trabalho, o SGD realiza uma atualização os pesos e *bias* na direção do gradiente da função de desempenho.

Capítulo 4

Resultados

4.1 Resultados Simulados

Esta seção apresenta resultados simulados de rastreamento de trajetória quando o robô é afetado por incertezas paramétricas e distúrbios externos usando a abordagem proposta mostrada na Figura 1. Será apresentado um estudo comparativo entre os controladores descritos abaixo com base em experimentos de simulação.

4.1.1 Controladores

4.1.1.1 Controle $\mathcal{H}_\infty$ não linear

No Controle $\mathcal{H}_\infty$ não linear foi utilizado o mesmo *loop* de realimentação apresentado na Figura 1, mas o torque calculado (96) é calculado como

$$\tau = F(x_e) + T_{11}^{-1} u. \tag{96}$$

Para o Controle $\mathcal{H}_\infty$ não linear , os ganhos do controlador cinemático foram escolhidos heuristicamente, com base nas recomendações propostas por Kanayama et al. (1990) para garantir convergência rápida sem oscilações. Os ganhos do controlador cinemático são

$$k_x = 0.5 \ , \ k_y = 3 \ e \ k_\psi = 1.$$

E foram definidos os parâmetros de controle $\mathcal{H}_\infty$, através do algoritmo apresentado por Chen, Lee e Feng (1994), com $\gamma = 3.09$, $Q_1 = 10 I_{(2\times2)}$, $Q_2 = 1000 I_{(2\times2)}$, $Q_{12} = 0$, e $R = 0.49 I_{(2\times2)}$ para que o sinal de tensão enviado não sature.

4.1.1.2 Controle *Deep Model Reference Adaptive Nonlinear* $\mathcal{H}_\infty$

O Controle $\mathcal{H}_\infty$ não linear adaptativo baseado em aprendizado profundo ($\mathcal{H}_\infty$ DMRAC) possui os mesmos elementos da Figura 5 menos o processo de *aprendizado lento* realizado pelo bloco SGD.

A arquitetura da DNN utilizara é a mesma apresentada na Seção 3.2.5

O controlador $\mathcal{H}_\infty$ via DNN tem os mesmos parâmetros de ajuste da Seção 4.1.1.1 e os parâmetros adaptativos são $k(x_e) = 100,00$ e $Z = 20,00 \times I_{1\times 4}$. A lei de controle adaptativo para ajustar os parâmetros é implementada com base em (91).

4.1.1.3 Controle *Deep Model Reference Adaptive Nonlinear* $\mathcal{H}_\infty$ com Gradiente Descente Estocástico

O Controle $\mathcal{H}_\infty$ não linear adaptativo baseado em aprendizado profundo com Gradiente Descente Estocástico ($\mathcal{H}_\infty$ DMRAC + SGD para abreviar) é a abordagem proposta na Figura 5.

4.1.1.4 Controle *Deep Adaptive Nonlinear* $\mathcal{H}_\infty$ com Gradiente Descente Estocástico

O Controle Adaptativo Profundo $\mathcal{H}_\infty$ com Gradiente Descente Estocástico ($\mathcal{H}_\infty$ DAC+SGD para abreviar) é a abordagem proposta na Figura 5, menos que o bloco *Modelo*.

Os parâmetros de controle são os mesmos apresentados na Seção 4.1.1.3.

4.1.2 Metodologia

Nesta seção serão apresentados a metodologia e as ferramentas utilizadas no desenvolvimendo dos resultados simulados deste trabalho. Toda a arquitetura foi definida usando um desktop com sistema operacional Windows 11. O hardware utilizado conta com um processador Intel Core i9-10900K com 20 núcleos de 3,7 GHz, 32 GB de RAM DDR4 e GPU Nvidia Geforce GT1030.

4.1.2.1 Simulação

O ambiente de simulação para o robô foi desenvolvido em MATLAB R2020a. Os controladores foram simulados para o modelo apresentado nas Seções 3.1.1 e 3.1.2 . Os testes foram simulados com tempo de amostragem, $T_s = 25ms$. O tempo de simulação foi de 80s para cada experimento, ou seja, com duração de $N = 2000$ iterações.

Os testes consistem em $N_e = 1000$ simulações para cada um dos controladores, com incertezas no modelo descrito na Seção 4.1.2.3. O percurso do robô móvel consiste do movimento a partir de um ponto próximo a origem para um trajetória que converge em

uma curva periódica, circular de raio ρ, conforme descrito na Seção 3.2.1. A escolha dos parâmetros dos controladores são apresentados na Seção 4.1.1.

Foram simulamos quatro controladores: Controle $\mathcal{H}_\infty$ como o controle padrão para comparações. DMRAC $\mathcal{H}_\infty$ em uma versão sem a atualização online da DNN, também de forma comparativa. DMRAC $\mathcal{H}_\infty$+SGD com a atualização online da DNN e a sua versão sem modelagem matemática do sistema, DAC $\mathcal{H}_\infty$.

4.1.2.2 Treinamento e avaliação da DNN

Esta seção discute detalhes sobre o treinamento e avaliação das DNNs desenvolvidos. Vale ressaltar que, para desenvolver os DNNs apresentados, foi utilizada a linguagem Matlab além da *Neural Network Training Toolbox*[1].

Para o treinamento inicial da DNN, utilizamos do treinamento supervisionado, o qual um conjunto de amostras rotulado é necessário. Realizamos a execução da simulação em condição de zero incertezas, com a trajetória definida em 3.2.1. Assim, treinamos uma DNN, onde, após o treinamento inicial, foi utilizada como ponto de partida para cada um dos controladores que se propusemos utilizá-la: Adaptive $\mathcal{H}_\infty$, DMRAC e DMAC. Desta forma, garantimos que todos os controladores foram inicializadas a partir da mesma DNN. Obtivemos 2000 amostras para compor o conjunto de dados inicial, sendo que 90% foram usados para treinamento e 10% para validação, no tipo Gradiente Descente Estocástico.

Apesar os controles citados partirem da mesma DNN inicial, cada um possuem sua particularidade em relação a sua atualização. O controle $\mathcal{H}_\infty$ DMRAC realiza a atualização do treinamento da DNN off-line antes de iniciar a movimentação do robô a partir dos dados da simulação anterior, ou seja, treina-se a rede neural em $N_e - 1$ vezes. Já os controles $\mathcal{H}_\infty$ DMRAC + SGD e $\mathcal{H}_\infty$ DAC + SGD realizam a atualização do treinamento da DNN on-line, paralelamente a simulação a cada 4s. Desta forma, considerando t_s o tempo total da simulação e t_{upd} o intervalo de atualização, temos $N_e(t_s/t_{upd}) - 1$ novos treinamentos da DNN.

4.1.2.3 Incertezas paramétricas e distúrbios externos

Na prática, os parâmetros do robô não podem ser conhecidos precisamente, sendo assim, as incertezas paramétricas e não-linearidades negligenciadas (folga, histerese) aparecem com distúrbio do sistema w foi adicionado na (57).

Incertezas de massa, momento de inércia e variação do centro instantâneo de rotação

[1] Mais detalhes MATLAB Neural Network Training em https://www.mathworks.com /help/deeplearning/ref/network.train.html

do robô foram atribuídas como

$$m_{real} = (1 + \Delta)m_{model},$$
$$I_{real} = (1 + \Delta)I_{model},$$
$$CIR_{real} = (1 + \Delta)CIR_{model},$$

respectivamente, em que $\Delta = 10\%$. Enquanto uma função senoide suavizada pelo tempo gera perturbações externas para todas as simulações, nos intervalos de tempo $t = 10$s e $t = 70$s, com uma amplitude máxima de 3 $N.m$ se aplica, conforme mostrado na Figura 7. Além disso, usamos a taxa de escorregamento para as rodas esquerda e direita, i_L e i_R igual a $0,1$.

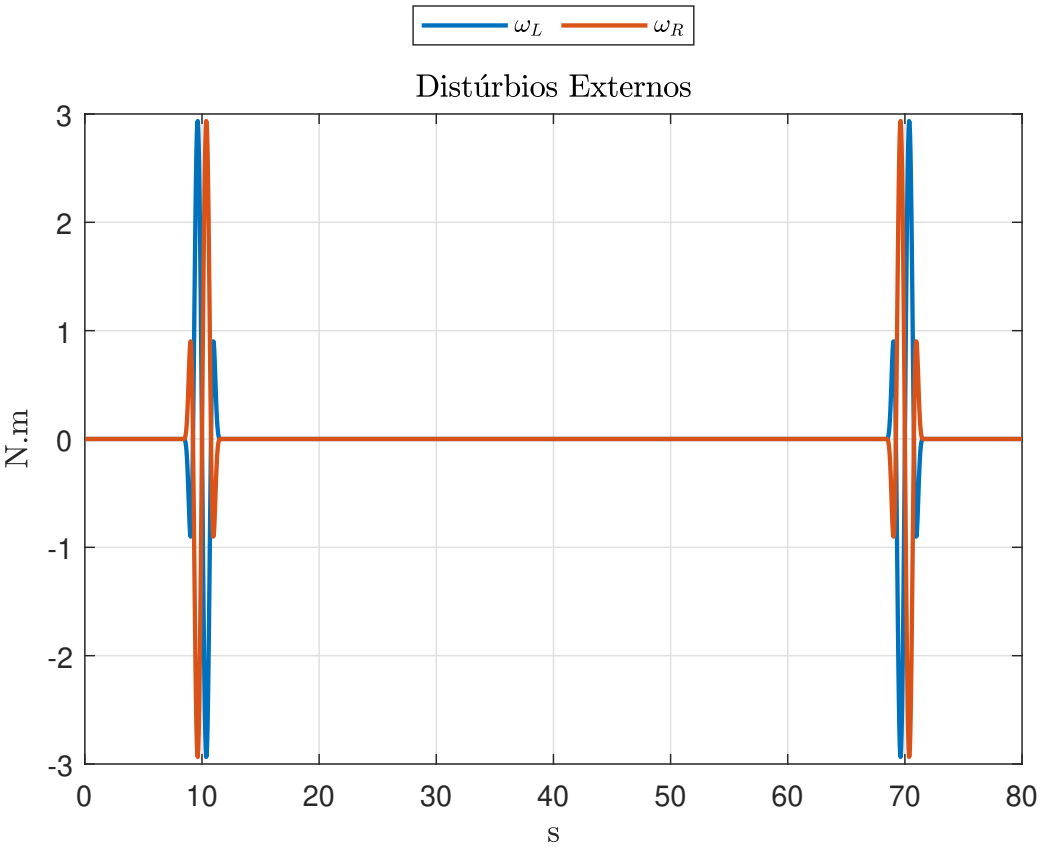


Figura 7 – Distúrbios externos ao longo do tempo

4.1.2.4 Métricas para o acompanhamento de trajetória

Como forma de avaliação estatística de desempenho dos controladores propostos, nós utilizamos da norma $\mathcal{L}_2$ e da energia total aplicada ao robô. Vale ressaltar que para melhor precisão dos dados, as métricas correspondem a média dos 1000 experimentos simulados, em que

$$E(i) = \frac{1}{N_e}\sum_{j=1}^{N_e}\left(\frac{1}{(t_r - t_0)}\int_{t_0}^{t_r}\|q - q^d\|_2^2 dt\right)^{\frac{1}{2}},$$

Tabela 1 – Parâmetros de Simulação do

Parâmetro	Valor
I	0,413 Kgm^2
m	40,0 Kg
X_{CIR}	0,0080 m
a	0,1380 m
b	0,1220 m
c	0,1975 m
r	0,1075 m

Tabela 2 – Simulação Inicial para Controle $\mathcal{H}_\infty$ Não Linear Padrão

Controle	MAE, $\overline{E}(m)$	Energia	Incertezas
$\mathcal{H}_\infty$ Não Linear	0,0448	3,1899	Não

a norma $\mathcal{L}_2$, o erro médio absoluto

$$\overline{E}(i) = \frac{1}{N_i}\sum_{i=1}^{N_i} E(i), \text{ e } E[\tau] = \sum_{i=0}^{2}\left(\int_{t_0}^{t_r} |\tau_i| dt\right),$$

a energia total utilizada. Temos que, q e q^d são os vetores de posição e posição desejada, respectivamente; j representa o número do experimento com $N_e = 1000$, o total de experimentos ; i corresponde ao valor de uma interação com N_i sendo o número máximo de iterações. Para calcular a porcentagem de melhoria das arquiteturas propostas em relação ao controlador padrão, usamos o seguinte índice percentual:

$$I^{\%} = \left(1 - \frac{\overline{E}}{\overline{E}_{ref}}\right) \times 100\,, \tag{97}$$

sendo $\overline{E}$ o erro absoluto de cada controlador e $\overline{E}_{ref}$ o erro absoluto do controlador escolhido com padrão.

4.1.3 Execução da Simulação

Inicialmente, simulamos um experimento com os parâmetros do robô apresentados na Tabela 1, utilizando o Controle Não Linear $\mathcal{H}_\infty$ e em condições ideais, ou seja, sem nenhuma incerteza paramétrica ou perturbação externa . Desta forma, foi possível armazenar os dados iniciais para o treinamento do DNN e os critérios das métricas iniciais, que apresentamos na Tabela 2 e o caminho percorrido apresentado na Figura 8.

A partir dos dados gerados pelo Controle $\mathcal{H}_\infty$ não linear, iniciamos a simulação comparativa, nas mesmas condições porém com adição de incertezas paramétricas. Ao final das N_e simulações, a trajetória desejada e a percorrida pelo robô nos quatro diferentes controladores pelo robô pode ser vista em Figura 9. As métricas das simulações estão sumarizados na Tabela 3 e a taxa de melhoria $I^{\%}$ na Tabela 4.

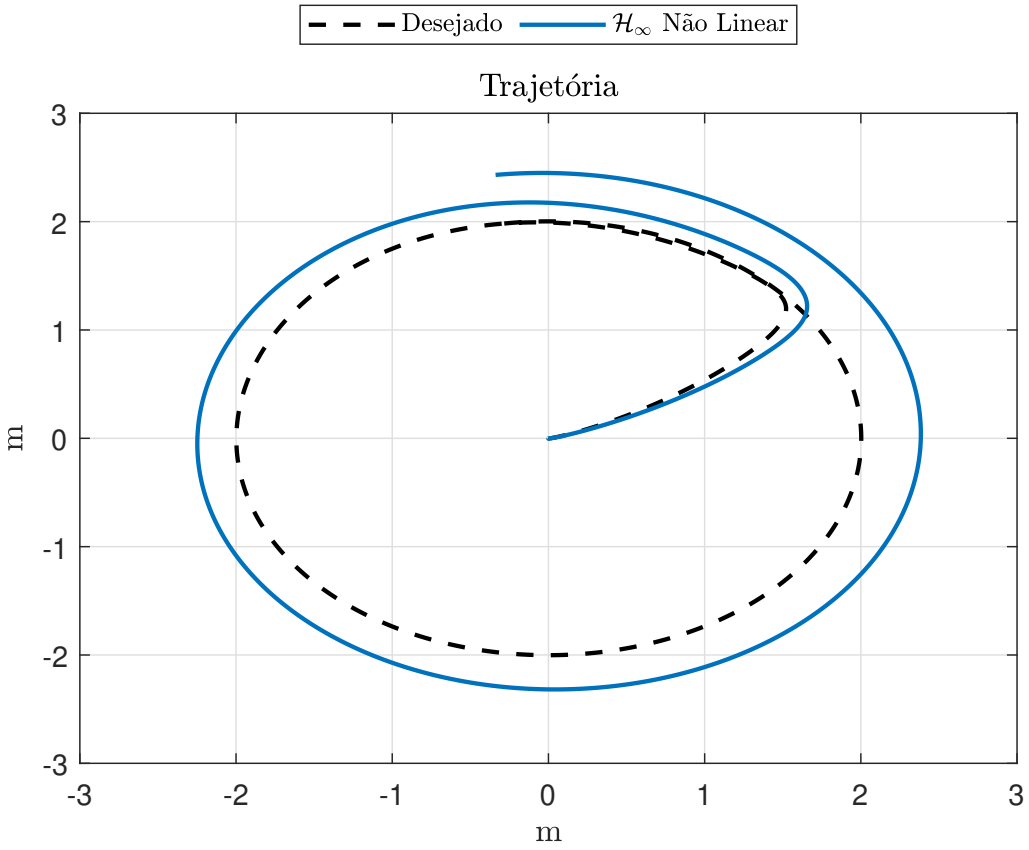


Figura 8 – Trajetória Inicial do Controle $\mathcal{H}_\infty$ Padrão Não Linear

Em Figura 10 é possível visualizar os valores de saída x dos controladores ao longo do tempo. Vale o destaque para o intervalo de simulação próximo aos 63 segundos, em que é possível verificar o controle $\mathcal{H}_\infty$ Não Linear apresenta um erro de posição em x de aproximadamente 20% enquanto os demais controladores não atingiram 10% de erro de posicionamento em x. Da mesma forma, em Figura 11, é possível visualizar a saída de controle em y. O intervalo próximo aos 47 segundos de execução, também ha um erro expressivo de posicionamento em y enquanto os demais controles apresentam menor erro. Já para ψ ao longo do tempo, tem-se os erros apresentados na Figura 12, destaca-se que o controle $\mathcal{H}_\infty$ Não Linear apresenta o menor erro. Quanto aos demais controles, apresentam erros em ψ ultrapassando, em alguns pontos $0,2\ rad$. Destaque para o controle $\mathcal{H}_\infty$ DAC + SGD que apresentou os menores erros em ψ. Neste sentido, novos ganhos para os controladores com redes neurais devem ser selecionados para melhorar tais controles nesta situação. O presente trabalho não cobre os novos ganhos de controle.

Considerando tais dados, é possível afirmar que a adição de incertezas paramétricas no Controle $\mathcal{H}_\infty$ manteve o consumo de energia ligeiramente menor; entretanto, houve uma piora no $I\%(\overline{E})$ de mais de 8%. Analisando a simulação de $\mathcal{H}_\infty$ DMRAC contra o controle padrão, notamos uma redução no erro médio absoluto com $I\%(\overline{E}) = 49,78\%$ enquanto energia consumo mais que dobrou. O controle $\mathcal{H}_\infty$ DAC + SGD teve menor

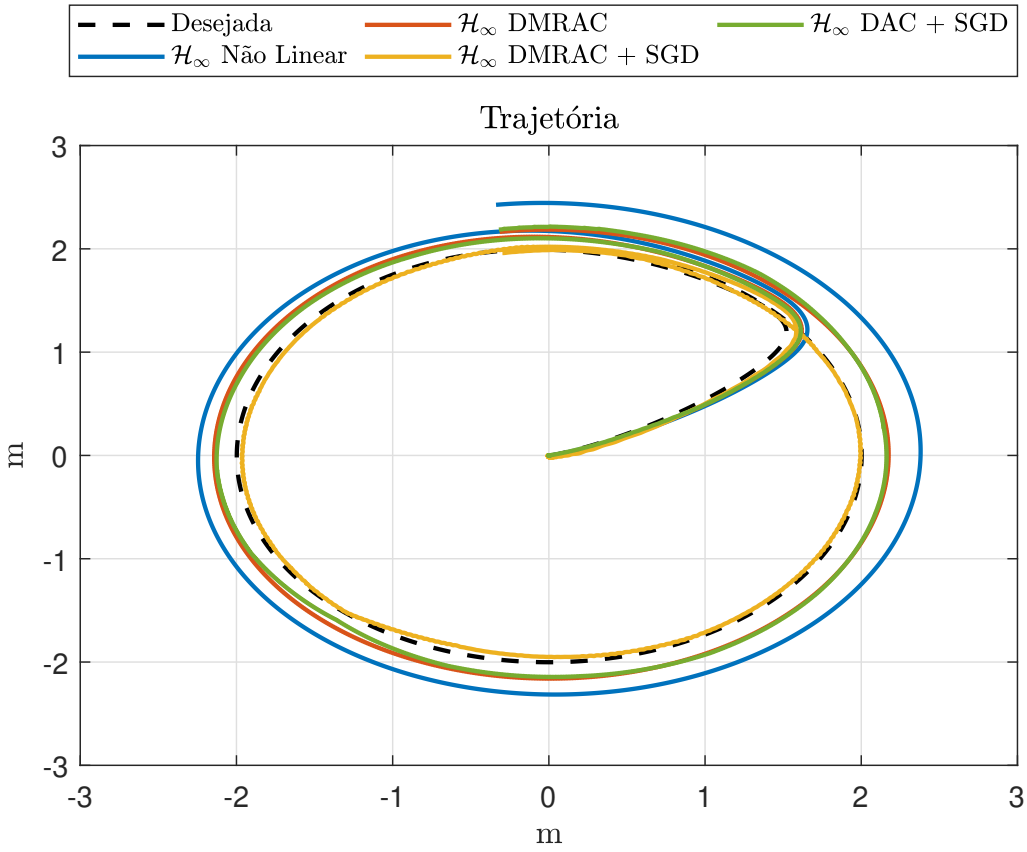


Figura 9 – Trajetória final dos controladores após 1000 experimentos.

consumo de energia do que o $\mathcal{H}_\infty$ DMRAC, mas ainda mais que o dobro do $\mathcal{H}_\infty$ Controle, o $I\%(\overline{E})$ reduziu em 76,34%. Finalmente, temos $\mathcal{H}_\infty$ DRAMC + SGD, com um consumo de energia 1,5 vezes maior que nosso controle padrão, mas com um erro médio absoluto da ordem de milímetros, e com um índice de melhora I

Em uma segunda análise, com base nos dados da Tabela 3, foram comparados $\mathcal{H}_\infty$ DMRAC + SGD e $\mathcal{H}_\infty$ DAC + SGD com o $\mathcal{H}_\infty$ DRAMC. O $\mathcal{H}_\infty$ DAC + SGD obteve uma melhora de $I\%(\overline{E}) = 52,89\%$ em relação ao $\mathcal{H}_\infty$ DMRAC com melhora do índice de consumo de energia de $I\%(Energia) = 0,42\%$. Por outro lado, o índice de consumo de energia do $\mathcal{H}_\infty$ DRAMC + SGD piorou $I\%(Energia) = 19,52\%$, mas melhorou quanto ao índice $I\%(\overline{E})$ foi de 88,44%. Apresentamos a análise na Tabela 5.

A partir das duas análises, é possível afirmar que os controladores $\mathcal{H}_\infty$ DAC + SGD e $\mathcal{H}_\infty$ DMRAC + SGD apresentaram taxas de melhoria de desempenho satisfatórias de mais de 94% como comparado a $\mathcal{H}_\infty$ DMRAC + SGD com $\mathcal{H}_\infty$ Controle. É necessário considerar o consumo de energia: O consumo de $\mathcal{H}_\infty$ DRAMC + SGD foi 20,03% maior que o de $\mathcal{H}_\infty$ DAC + SGD enquanto seu $I\%(\overline{E})$ é 75,47% melhor. Portanto, para escolher entre os dois controladores, é necessário analisar a precisão necessária da aplicação e o consumo de energia.

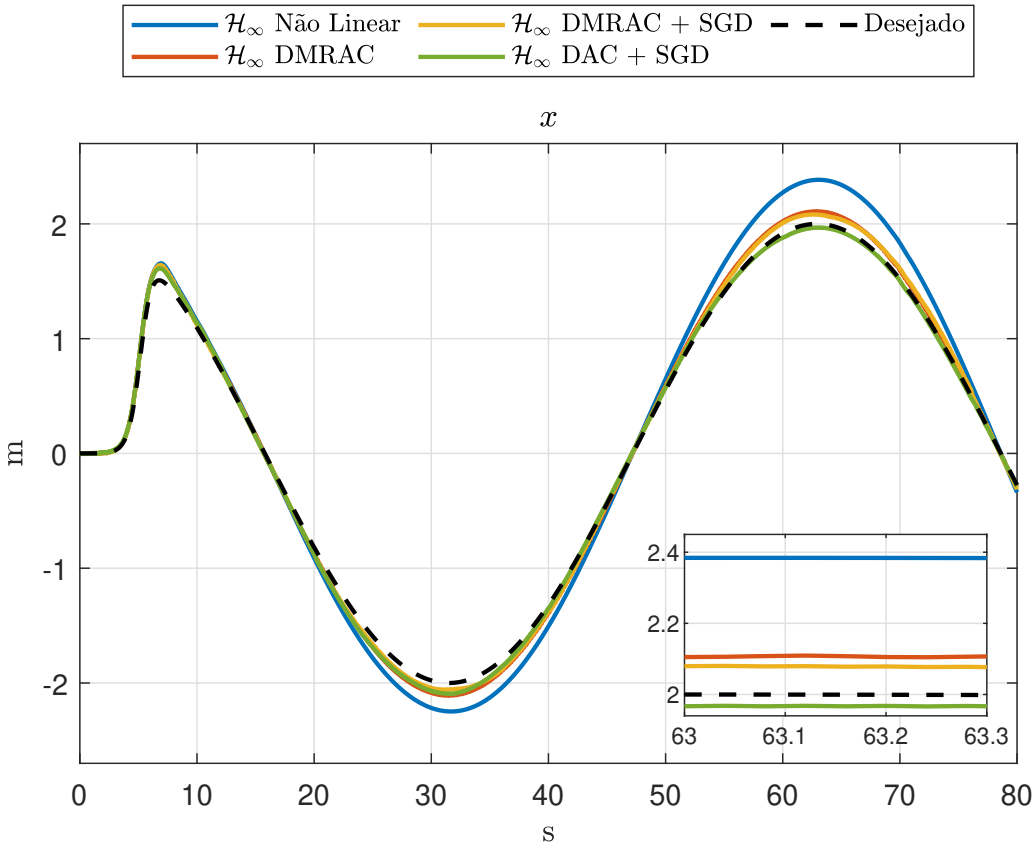


Figura 10 – Valores de x ao longo da simulação.

Tabela 3 – Métricas de desempenho após 1000 simulações

Controle	MAE, $\overline{E}(m)$	Energia	Incertezas
$\mathcal{H}_\infty$ Não Linear	0,0484	**3,1852**	Sim
$\mathcal{H}_\infty$ DMRAC	0,0225	6,8415	Sim
$\mathcal{H}_\infty$ DMRAC + SGD	**0,0026**	8,1772	Sim
$\mathcal{H}_\infty$ DAC + SGD	0,0106	6,8125	Sim

Tabela 4 – Percentual de Melhoria

Controle	$I^{\%}$ $(\overline{E})$	$I^{\%}$ (Energia)
$\mathcal{H}_\infty$ Não Linear	-8,04%	**0,15%**
$\mathcal{H}_\infty$ DMRAC	49,78%	-114,47%
$\mathcal{H}_\infty$ DMRAC + SGD	**94,20%**	-156,35%
$\mathcal{H}_\infty$ DAC + SGD	76,34%	-113,56%

4.2 Resultados Experimentais

Nesta seção serão apresentados os resultados experimentais obtidos neste projeto. Na Seção 4.2.1 será apresentada a montagem da plataforma robótica. Em seguida, na Seção 4.2.2 os atuadores e as rodas motrizes serão apresentados. Na sequência, detalhes do *driver* de acionamento dos motores serão apresentados na Seção 4.2.3. Já na Seção

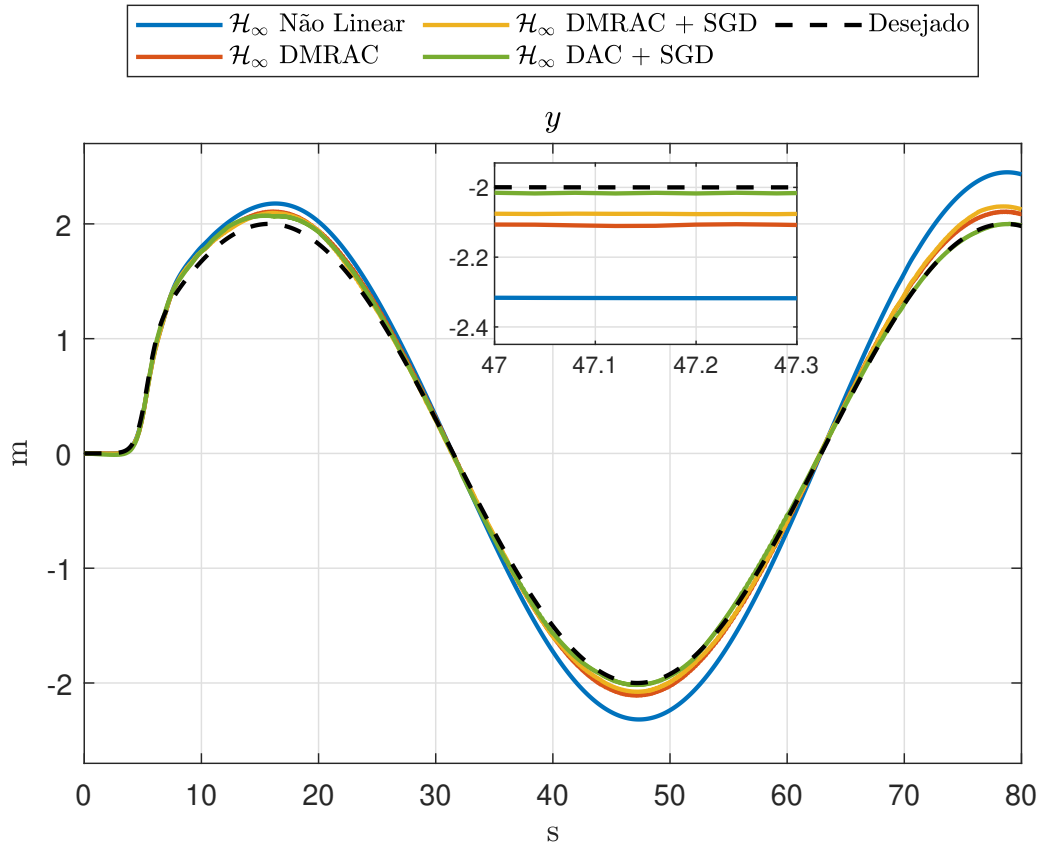


Figura 11 – Valores de y ao longo do simulação.

Tabela 5 – Percentual de Melhoria Controles DNN

Controle	$I^{\%}$ $(\overline{E})$	$I^{\%}$ (Energia)
$\mathcal{H}_\infty$ DMRAC + SGD	**88,44%**	-19,52%
$\mathcal{H}_\infty$ DAC + SGD	52,89%	**0,42%**

seguinte, 4.2.4 será apresentado o processo de leitura dos *encoders* dos atuadores das rodas. Em seguida, na Seção 4.2.5 serão apresentado os microcontroladores utilizados para o acionamento do robô em baixo nível. Em seguida, a estratégia de controle embarcado é apresentada na Seção 4.2.6. Por fim, uma explanação sobre o desenvolvimento de um pacote *Robot Operating System* - Sistema operacional do robô (ROS) e o computador embarcado utilizado, são apresentados nas Seções 4.2.7 e 4.2.12, respectivamente.

4.2.1 Montagem da plataforma

Foi utilizado o *Lynxmotion Aluminum A4WD1 Rover Kit* considerada uma plataforma robusta de desenvolvimento. Possui um chassi em alumínio, quatro rodas atuadas e calçadas com pneus de excelente tração sendo capaz de transportar aproximadamente $2,3kg$ de carga útil. Para ampliar e facilitar a instalação de novos componentes, a fabricante disponibiliza alguns *decks*, em que há a possibilidade de agregar novos equipamentos.

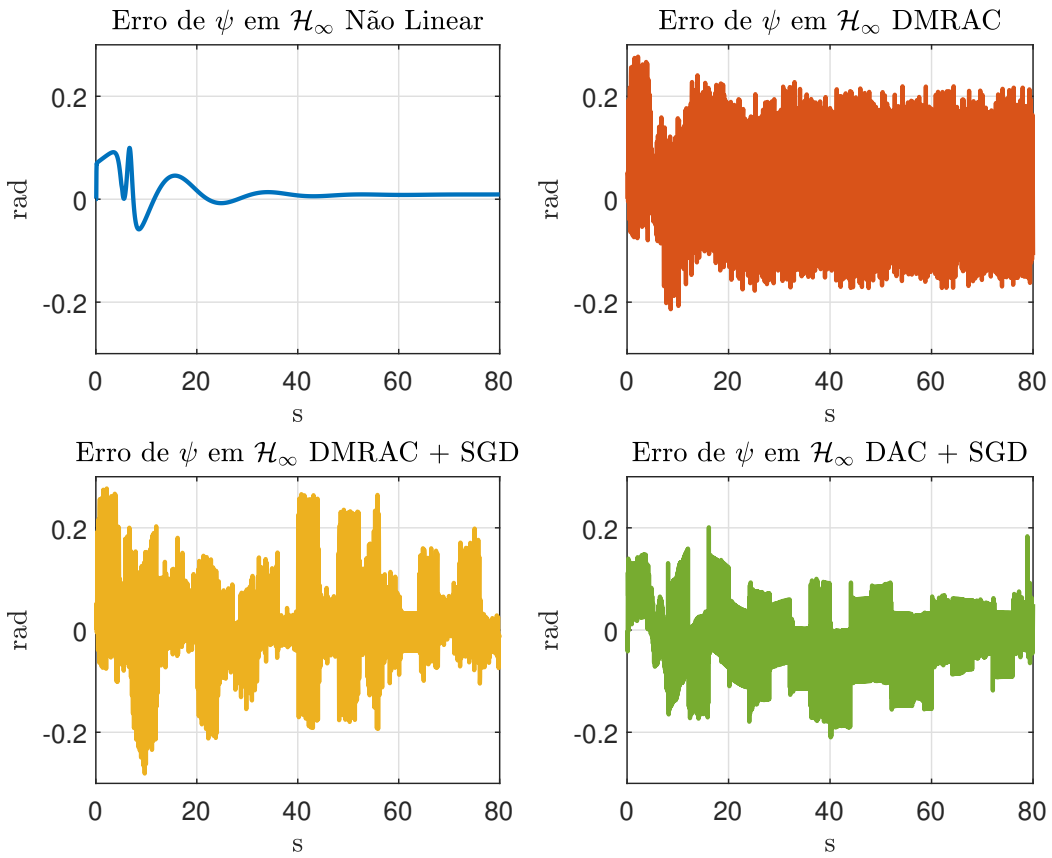


Figura 12 – Valores do Erro de ψ ao longo do simulação.

Para montagem, guiada pelo manual de instruções, basta unir as partes do chassi com os parafusos disponíveis. Mais detalhes sobre a plataforma robótica podem ser encontrados no Apêndice A.

Figura 13 – Lynxmotion Aluminum A4WD1 Rover Kit

4.2.2 Atuadores e Rodas

Quanto aos atuadores, utilizou-se de quatro motores de corrente contínua da fabricante *Hsiang Neng*, com tensão nominal de 12V e redução no eixo de saída de 30:1. Ademais, suas especificações técnicas apresentadas em Apêndice B. Como forma de mensurar a orientação e velocidade de cada motor, foram acoplados em seus eixos, quatro *encoders* de quadratura E4T da *US Digital* capaz de contar cem ciclos por revolução e quatrocentos pulsos por revolução. Seu detalhamento técnico pode ser encontrado no Apêndice C. Já em contato com o solo, estão quatro rodas *SportTraxxas Talon* de 121mm de diâmetro proporcionando excelente tração em diferentes tipos de piso. Um exemplo do chassi, atuadores e rodas montados, pode ser verificado na Figura 13.

4.2.3 Acionamento dos Motores

Para acionar e controlar os atuadores, foi utilizados um *driver Sabertooth 2x12A*, que possui 2 saídas (M1 e M2) para motores de corrente contínua independentes com 12A de corrente nominal. Em cada saída, foram conectador dois motores de cada lateral da plataforma. O *driver* também possui quatro modos de comandá-lo:

1. Entrada Analógica
   - ❏ Sinais de entrada variando de 0-5V;
   - ❏ Facilidade de controle através de um potenciômetro.
2. Controle Remoto
   - ❏ Utiliza de canais de rádio controle.
3. Serial Simplificado
   - ❏ Utiliza o modo serial padrão RS-232 de comunicação;
   - ❏ Interface PC/Microcontrolador com o *drive*
4. Serial Empacotado
   - ❏ Também utiliza o modo serial RS-232;
   - ❏ O pacote contem um *byte* de endereço, um *byte* de comando, um *byte* de dados e um *checksum*
   - ❏ Interface PC/Microcontrolador com até oito *drive*

Optou-se pelo modo serial simplificado devido à conexão em paralelo dos motores e alguns contratempos encontrados no desenvolvimento do decodificador de quadratura, que será abordado na Seção 4.2.4

Como fonte de energia, é possível escolher dentre uma variada gama de baterias, como NiMH, NiCd, LiPo, LiOn e Chumbo Ácido. Vale ressaltar que para as baterias que possuem Lítio em sua formulação (Lipo e Lion), o *driver* possui um circuito que evita seu esgotamento e consequentemente o dano à mesma. O *Sabertooth 2x12A* também possui um circuito eliminador de bateria, convertendo a entrada de 6-24V para 5V e corrente nominal de 1A. Contudo, todas as funcionalidades do *driver* são selecionadas a partir de 6 chaves numeradas alocadas no circuito principal. Para a aplicação aqui descrita - com um *baudrate* 9600, modo serial simplificado e bateria do tipo LiPO - a configuração das chaves é a sguinte: Chaves em *ON*: 1,4,5,6. Chaves em *OFF*: 2,3.

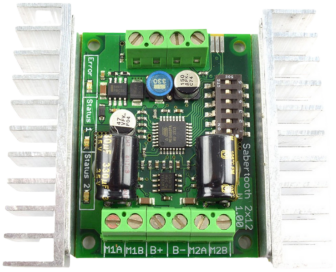


Figura 14 – Sabertooth 2x12A

Desta forma, com apenas um *driver Sabertooth 2x12A* controlando os quatro motores, a lógica de acionamento dos motores no modo serial simplificado pode ser resumida: Ao receber caracteres de 8 *bytes*, variando de 1 a 127, os motores na saída M1 serão acionados, sendo 1 reverso total, 64 parada, 127 à frente total. Já recebendo uma variação entre 128 e 255, acionará os motores na saída M2, sendo 128 reverso total, 192 parada, 255 à frente total. Vale a pena citar que ao receber o carácter 0, há o desligamento de ambas as saídas.

4.2.4 Decodificador de Quadratura

4.2.4.1 Baseado em *HCTL 2017*

Para obter a leitura de velocidade das rodas através dos *encoders* E4T acoplado ao eixo dos motores, foi necessário desenvolver o decodificador baseado no componente HCTL 2017. Seu funcionamento pode ser descrito da seguinte maneira: Os pulsos do canal A e B são monitorados e contados quando a porta *OE* está em nível lógico alto e no momento em que é desejado a leitura, altera-se seu nível para baixo. Através da porta *Sel* é possível selecionar a leitura dos 8 *bits* mais ou menos significativos, alterando o nível lógico para alto ou baixo, respectivamente. Após a leitura, reinicia-se os contadores utilizando a porta *RST* e o estado da porta *OE* para que o contador volte a contar os pulsos dos canais

de entrada (SGRIGNOLI, 2017). Desta forma, para realizar a leitura de velocidade de cada uma das quatro rodas seria necessária uma estratégia de multiplexação: ativando um decodificador de cada vez, reduzindo assim o número de portas utilizadas na aferição de velocidade do RMR. Na Figura 15 é apresentada a placa de circuito impresso com alguns componentes afixados.

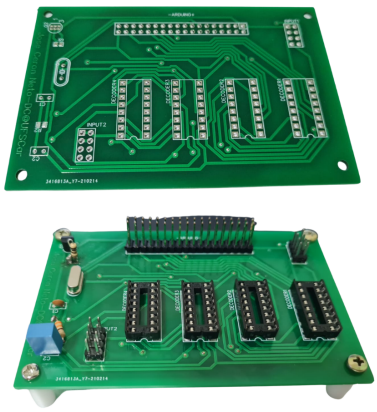


Figura 15 – Circuito Impresso do Decodificador de Quadratura para Quatro Encoders

No momento da montagem, a placa de circuito impresso apresentou alguns problemas que não foram possíveis de serem sanados rapidamente. Portanto, para evitar atrasos ao cronograma inicial, optou-se pela utilização do protótipo de decodificador anteriormente desenvolvido por Sgrignoli (2017), apresentado na Figura 16. Desta forma, há a leitura da velocidade apenas das rodas dianteiras esquerda e direita.

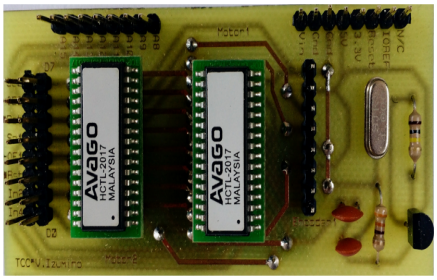


Figura 16 – Protótipo de Decodificador Elaborado por Sgrignoli (2017)

4.2.4.2 Baseado em LS7366R

Uma vez que o decodificador apresentado na Seção 4.2.4.1, desenvolvido por Sgrignoli (2017), é capaz de realizar a leitura apenas de dois dos quatro *encoders*, foi realizada a aquisição de um decodificador de quadratura quádruplo baseado no contador de quadratura LS7366R, de 32 bits. A placa é equipada com uma interface *Serial Peripheral Interface* - Interface Periférica Serial (SPI), tornando simples o monitoramento da posição dos motores, acompanhar a posição angular, velocidade, distância total percorrida

A interface de comunicação SPI simplifica muito seu uso, sendo que microcontrolador selecione qual dos *encoders* deve ser lido simplesmente abaixando o nível lógico de sua porta *Slave Select* - Seleção de Chip (SS)[2] conforme pode ser observado na Figura 17.

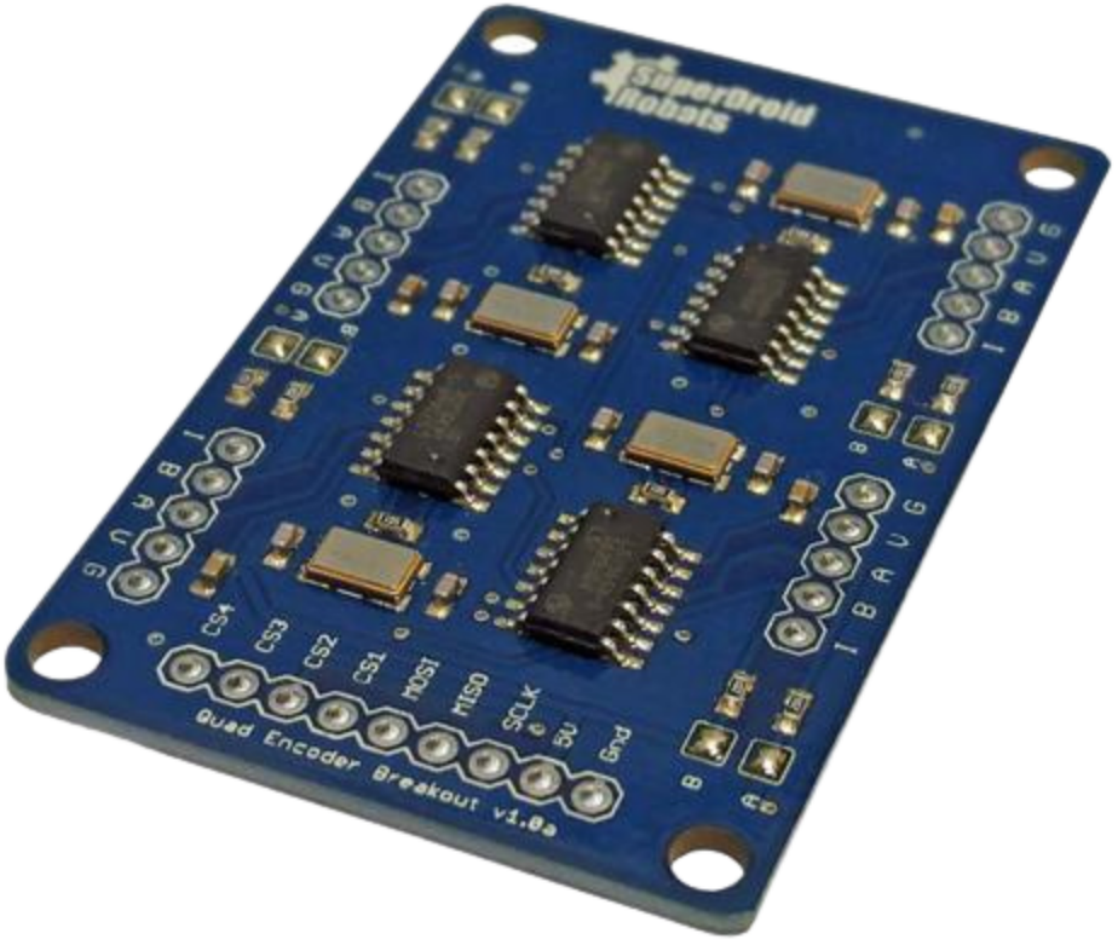


Figura 17 – Decodificador de Quadratura Quádruplo baseado em LS7366R

4.2.5 Microcontrolador

4.2.5.1 Arduíno

Com a finalidade de agregar os componentes apresentados nas seções anteriores, fez-se necessário a utilização de um microcontrolador *Arduíno Mega 2560*, devido aos seus 54 pinos de entrada e saída e portas seriais disponíveis. Primeiramente a placa decodificadora

[2] Mais detalhes do circuito impresso LS7366R, consulte seu *Datasheet*

foi conectada aos seus pinos de entrada e saída. Posteriormente um das portas seriais foi ligada à entrada de sinais do *driver* que também fornece os 5V de alimentação.

Comunicação serial com o *Sabertooth 2x12A*

Como citado na Seção 4.2.3, optou-se pelo modo serial simplificado para envio dos comandos de acionamento dos motores. No código executando no Arduíno, inicialmente define-se o pino serial de saída, *SABER_TX_PIN* e também a taxa de transmissão, *SABER_BAUDRATE*, em seguida é aberta a comunicação, utilizando a biblioteca *SoftwareSerial* disponível para Arduíno. Desta forma é possível enviar, através de um simples comando *write()* o caracter de velocidade para o *driver*.

Comunicação serial via USB

A utilização do Arduíno é restrita ao seu processamento e quantidade de memória disponível, portanto não suficiente para a realização de todos os cálculos que o projeto exige. Desta forma, será necessário realizar a comunicação com um equipamento externo. Para isso, optou-se pela solução de comunicação via USB apresentada por Sgrignoli (2017). Utilizando uma interrupção serial, disponível no microcontrolador, é executado um código faz a leitura de uma mensagem no padrão ($xxxxx,yyyyy#). Os caracteres $ e # indicam o início e fim da mensagem, respectivamente, o "xxxxx"contém a velocidade angular dos motores em M1, com sinal positivo ou negativo em ponto flutuante. Analogamente à "yyyyy"e M2.A vígula faz a separação entre os valores (SGRIGNOLI, 2017). Uma vez com a estrutura preenchida com as respectivas velocidades, elas são descarregadas em uma variável contendo o número de caracteres desejados para obedecer o protocolo de comunicação estabelecido, sendo então enviada via serial USB. Por sua vez, na via oposta da comunicação serial, a placa Arduíno recebe mensagens de velocidades através da USB, com um protocolo análogo. Então essa mensagem é interpretada e os valores obtidos são enviados como valor de referência para o controlador de cada roda.

4.2.5.2 *Raspberry Pi Pico*

Com a mesma finalidade de utilização do Arduíno, citado na Seção 4.2.5.1, também foi utilizado neste projeto um microcontrolador *Raspberry Pi Pico* de baixo custo e alto desempenho com interfaces digitais flexíveis, sendo que os principais recursos, que se destacam para o projeto, são: processador dois núcleos, 2MB de memória, bibliotecas de ponto flutuante aceleradas no chip,interface SPI para o novo decodificador apresentado na Seção 4.2.4.2 e um tamanho quatro vezes menor, em comparação com o Arduíno. Um microcontrolador é apresentado no Figura 18.

A *Raspberry Pi Pico* possui um kit de desenvolvimento de software para programação em MicroPython e C/C++ além de vasta documentação de suporte no site do fabricante [3].

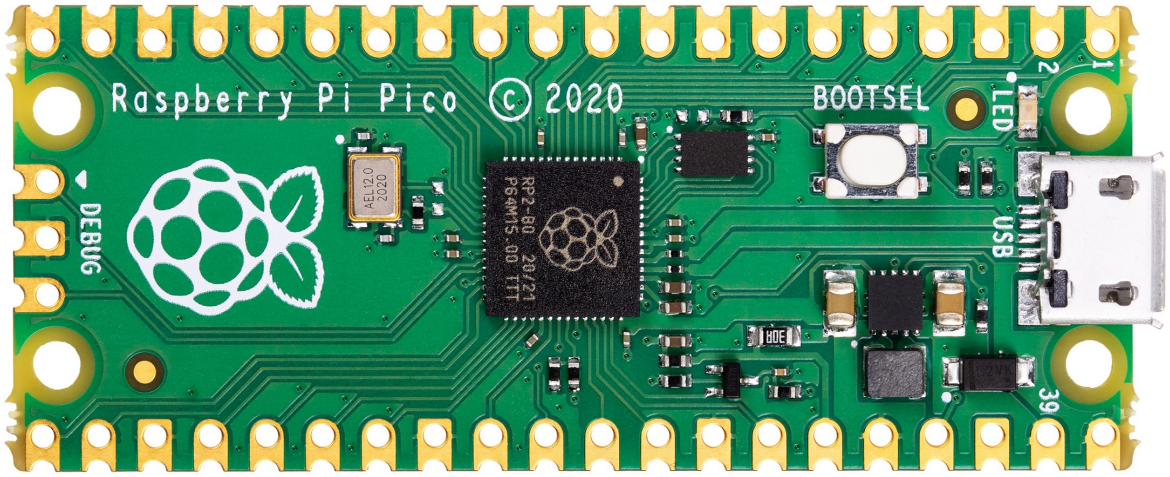


Figura 18 – *Raspberry Pi Pico*

Quanto à comunicação do microcontrolador *Raspberry Pi Pico* e o *driver* e também a comunicação entro o computador embarcado, seguem as mesmas orientações e protocolos apresentados na Seção 4.2.5.1.

4.2.6 Estratégia de controle embarcado

Baseado no trabalho de Sgrignoli (2017) é executado pelo microcontrolador Arduíno um controle PID sem a derivada de erro, com seus ganhos determinados através do método de Ziegler-Nichols. Sua estratégia alia resposta rápida da ação proporcional, com a remoção do *off-set* do termo integral e a estabilidade do termo derivativo. Neste sentido, a expressão do controlador PID é dada por:

$$u(t) = K_p \left[e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{de(t)}{dt} \right]. \tag{98}$$

Assume-se que os quatro motores possuem as mesmas características eletro-mecânicas. Selecionou-se um para o cálculo dos parâmetros do controle PID e utilizou-se do método da curva de reação, aplicando uma entrada constante (degrau) e o registrando a evolução da velocidade com o tempo, conhecido por Primeiro Método de Ziegler e Nichols ou Método de Malha Aberta. Desta forma, foi aplicado uma tensão de 12V e a velocidade angular foi medida com tempo de amostragem de 10 ms, até atingir a velocidade máxima definida pela construção do motor. Em seguida, determinou-se o atraso de transporte L

[3] Documentação disponível: *Raspberry Pi Pico*

K	L	T
1.0010	0.0275	0.0710

Tabela 6 – Valores das Constantes K, L e T

e a constante de tempo T e o ganho K.Para determinar o atraso e a constante de tempo deve-se traçar uma reta tangente passando pelo ponto de inflexão da resposta. O atraso de transporte L é o tempo decorrido entre a aplicação da entrada e o cruzamento da reta tangente com a linha base. A constante de tempo T é o tempo para a reta tangente subir de 0 a 100% da evolução do sinal de saída. Já o ganho K é determinado pela relação entre a variação da saída ΔY e variação da entrada ΔU, conforme mostrado na Equação 99 .

$$K = \frac{\Delta Y}{\Delta U} \tag{99}$$

A resposta no domínio do tempo é dada por

$$Y(t) = K\Delta U(1 - e^{-(\frac{t-L}{T})}) \tag{100}$$

sendo $Y(t)$ a velocidade do motor ao longo do tempo.

De acordo com o Gráfico apresentado em na Figura 19, os valores de K, L e T estão dispostos na Tabela 6

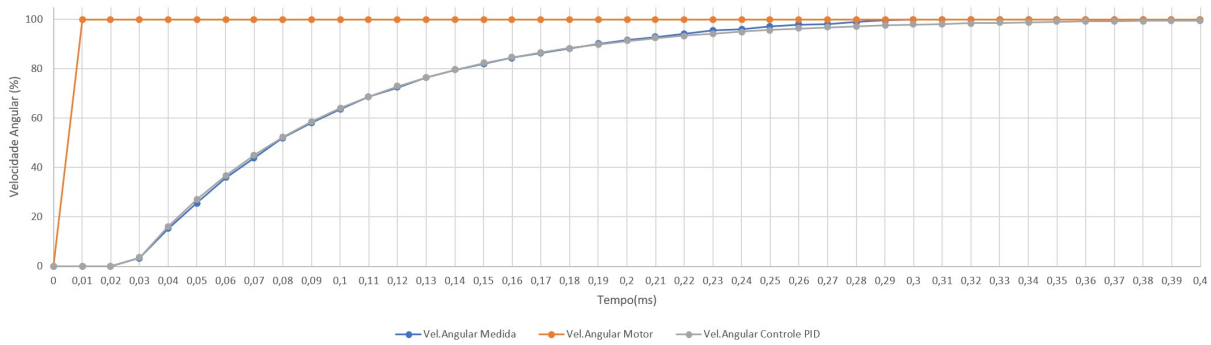


Figura 19 – Modelagem Matemática do Motor de Corrente Contínua

Desta maneira, baseado no ajuste de parâmetros sugerido pela Tabela 7, chegou-se aos parâmetros usados no controle PID que podem ser encontrados no Apêndice E.

Tipos de Controladores	K_p	T_i	T_d
P	$\frac{T}{KL}$	∞	0
PD	$0.9\frac{T}{KL}$	$\frac{L}{0.3}$	0
PID	$1.2\frac{T}{KL}$	$2L$	$0.5L$

Tabela 7 – Regra de Sintonia de Ziegler-Nichols Baseada na Resposta ao Degrau da Planta

4.2.7 Pacote ROS

A criação de robôs robustos e confiáveis é complexa, na perspectiva da robótica. Problemas que parecem triviais pra humanos, geralmente apresentam grandes dificuldades de implementação. Partindo deste princípio, o ROS, foi constituído como um *framework* com bibliotecas e ferramentas para criação de *software* robótico colaborativo, fornecendo ferramentas e bibliotecas com características que auxiliam os desenvolvedores em tarefas como a troca de mensagens, reuso de código e até mesmo computação distribuída (JOSEPH, 2015). Segundo Joseph (2015) enumera algumas vantagens de usar o *framework* no desenvolvimento, dentre elas: Capacidades robóticas de ponta já desenvolvidas e disponíveis para uso e reuso; a integração com os mais variados tipos de sensores disponíveis; ferramentas de desenvolvimento, visualização, correção de erros e simulação; a codificação pode ser desenvolvida em diversas linguagens - C++, Phyton e Java - aproveitando as características e familiaridade do desenvolvedor; reuso e modulação. Todas essas vantagens contam com um suporte e atualizações ativas da Comunidade ROS.

Em contrapartida, o *framework* apresenta também algumas desvantagens relevantes para a presente discussão, considerada por Joseph (2015), como a principal: A dificuldade de aprendizagem, já que os desenvolvedores serão introduzidos à vários novos conceitos e nomenclaturas do demandando uma curva de aprendizagem acentuada. As demais estão relacionadas a complexidade na modelagem dos robôs e no uso de ferramentas de simulação. Um maior detalhamento sobre o ROS pode ser consultado no Anexo A.

Neste sentido, optou-se pelo sua utilização para desenvolvimento do projeto aqui apresentado, desta forma priorizando a utilização de códigos previamente desenvolvidos, a integração com os diversos sensores e com os atuadores embarcados no robô e principalmente, considerando os resultados satisfatórios da pesquisa, disponibilizar e adaptar a arquitetura de controle desenvolvida aqui para futuros projetos na área.

Desta forma, fez-se necessário o desenvolvimento de um pacote ROS específico para o *Linxmotion*. Baseado no *drone_dev* de Benevides et al. (2019), foi estabelecido o *mobile_dev* com a finalidade de comunicar, controlar, movimentar e localizar o RMR nos seus diversos ambientes de funcionamento. Desta forma, até o presente momento, houve o desenvolvimento, utilizando a linguagem de programação *Python*, de quatro *nodes*, ou nós, do pacote que serão apresentados nas subseções a seguir.

4.2.8 Nó de Comunicação

Chamado de *LynxMotion*, este nó é responsável por estabelecer uma ponte de comunicação, via *Universal Serial Bus* - Barramento Serial Universal (USB), do pacote ROS com o *hardware* controlado pelo Arduíno. Após a abertura de conexão e seu estabelecimento, o programa é capaz de receberas velocidades das rodas, transformá-las em uma mensagem padrão odometria e publicá-las no ambiente ROS. Também é sua função, rea-

lizar o procedimento inverso, receber as mensagens com comando de velocidades de cada roda, empacotá-las no formado reconhecido pelo Arduíno e despachá-las via a ponte de comunicação estabelecida anteriormente.

4.2.9 Nó de Comandos por *Joystick*

Neste nó, houve o desenvolvimento de um programa para que o RMR possa ser controlado por um operador em casos excepcionais, como por exemplo uma falha de comandos. Desta forma, o *mobile_joy* é capaz de enviar os comandos velocidades de um *joystick* diretamente para o nó de Comunicação e assim passando o controle do robô para o operador humano.

4.2.10 Nó de Alvo

Nomeado aqui como *Target*, é responsável por receber os pontos de objetivo em um plano cartesiano, formatá-los para uma mensagem padrão ROS e despachá-las.

4.2.11 Nó de Controle

Aqui chamado de *trajectory*, este nó concentra o recebimento das mensagens oriundas de odometria, *joystick* e dos ponto alvo. Seu funcionamento tem início à partir do recebimento de uma mensagem de novas coordenadas de alvo. Uma trajetória de referência é gerada de acordo com os cálculos apresentados na Seção 3.2.1. Em seguida, o Controle baseado na Cinemática - definido na Seção 3.2.2 - gera as velocidades lineares e angulares desejadas, com a finalidade de percorrer a trajetória de referência gerada anteriormente, minimizando o erro. Esta calculado a partir das mensagens de odometria recebidas pelo nó.

4.2.12 Computador Embarcado

Apresentada as especificações do microcontrolador Arduíno utilizado no projeto, verificou-se algumas restrições em capacidade de processamento e disponibilidade de memória. Neste sentido, a utilização de um *hardware* computacional de maior poder e ao mesmo tempo compacto o suficiente para utilização embarcado na plataforma robótica. Para tal, optou-se pela plataforma computacional *NVIDIA Jetson Nano*. Com apenas 70x45mm e com um consumo de energia máximo de 10W, o módulo possui um processador *Quad-Core*, 4 GB de memória RAM e uma placa gráfica oferecendo 472 GFLOPs de processamento, sendo mais que suficiente para o trabalho aqui apresentado. Executando um sistema operacional baseado em Linux e adequado ao seu *hardware*, foi possível instalar e executar o *framework* de desenvolvimento robótico ROS na *NVIDIA Jetson Nano*.

Capítulo 5

Conclusão

Neste trabalho, controladores robustos inteligentes baseados no critério $\mathcal{H}_\infty$ foram desenvolvidos para robôs móveis incertos sujeitos a perturbações externas. Uma Rede Neural Profunda estima o modelo incerto do robô funcionando como um complemento ao modelo nominal e ao modelo total do robô. Resultados de simulação obtidos de um robô móvel com rodas foram apresentados. Foi realizado um estudo comparativo entre os controladores robustos propostos e o controlador não linear $\mathcal{H}_\infty$ e $\mathcal{H}_\infty$ DMRAC. Nesse sentido, avaliamos o desempenho dos controladores implementados e o consumo de energia com os diferentes controladores. Os resultados mostraram uma melhora de desempenho na tarefa de rastreamento de trajetória para todos os controladores implementados combinados com as DNNs.

Quanto aos resultados práticos, estão relacionados à montagem da estrutura do robô e integração de todos os componentes eletro-mecânicos com o microcontrolador, seu *software* embarcado capaz executar os comando de movimentação. Neste sentido, notou-se algumas limitações da primeira versão em Arduíno, principalmente quanto apenas possuir duas entradas para leitura. Desta forma, motivou-se uma nova pesquisa para integrar um novo microcontrolador e nova placa de leitura à estrutura previamente montada e o computador embarcado apresentado. Também desenvolvido os códigos responsáveis pelo acionamento do robô através do *framework* ROS, em especial o Controle Baseado na Cinemática (3.2.2), sendo esses as bases para o pacote *mobile_dev*.

5.1 Trabalhos Futuros

Quanto à trabalhos futuros, o principal objetivo é a aplicação dos controles simulados na plataforma robótica apresentada na Seção 4.2. O que também leva ao aperfeiçoamento

do pacote ROS *mobile_dev*, tornando-o uma ferramenta versátil para o desenvolvimento novos RMR e de pesquisas relacionadas ao tema.

Referências

ABU-KHALAF, M.; LEWIS, F.; HUANG, J. Hamilton-jacobi-isaacs formulation for constrained input nonlinear systems. In: **43rd IEEE Conference on Decision and Control (CDC) (IEEE)**. Kobe, Japan: IEEE, 2004.

BENEVIDES, J. R. S. et al. ROS-based robust and recursive optimal control of commercial quadrotors. In: **2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)**. [S.l.: s.n.], 2019.

CARACCIOLO, L.; LUCA, A. de; IANNITTI, S. Trajectory tracking control of a four-wheel differentially driven mobile robot. In: **Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)**. IEEE, 1999. Disponível em: <https://doi.org/10.1109/robot.1999.773994>.

CHANG, Y.-C. Neural network-based h tracking control for robotic systems. **IEE Proceedings of Control Theory Applications**, v. 147, n. 3, p. 303–311, 2000. ISSN 1350-2379.

_____. Intelligent robust control for uncertain nonlinear time-varying systems and its application to robotic systems. **IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)**, Institute of Electrical and Electronics Engineers (IEEE), v. 35, n. 6, p. 1108–1119, 2005.

CHANG, Y.-C.; CHEN, B.-S. A nonlinear adaptive h/sup /spl infin// tracking control design in robotic systems via neural networks. **IEEE Transactions on Control Systems Technology**, 1997. ISSN 1063-6536.

CHANG, Y.-C.; YEN, H.-M.; WANG, P.-T. An intelligent robust tracking control for a class of electrically driven mobile robots. **Asian Journal of Control**, 2004. ISSN 1561-8625.

CHEN, B.-S.; CHANG, Y.-C.; LEE, T.-C. Adaptive control in robotic systems with h∞ tracking performance. **Automatica**, 1997. ISSN 0005-1098.

CHEN, B. S.; LEE, T. S.; FENG, J. H. A nonlinear $\mathcal{H}_\infty$ control design in robotic systems under parameter perturbation and external disturbance. **International Journal of Control**, v. 59, n. 2, p. 439–461, 1994.

CHEN, M. Disturbance attenuation tracking control for wheeled mobile robots with skidding and slipping. **IEEE Transactions on Industrial Electronics**, 2017. ISSN 0278-0046.

CUI, M. et al. An adaptive unscented kalman filter-based controller for simultaneous obstacle avoidance and tracking of wheeled mobile robots with unknown slipping parameters. 2017.

GOODFELLOW, I. et al. Generative adversarial networks. Curran Associates, Inc., v. 63, p. 139–144, 2014. ISSN 0001-0782.

HU, Y.; GE, S. S.; SU, C.-Y. Stabilization of uncertain nonholonomic systems via time-varying sliding mode control. **IEEE Transactions on Automatic Control**, 2004. ISSN 0018-9286.

Huang, J.; Van Hung, T.; Tseng, M. Smooth switching robust adaptive control for omnidirectional mobile robots. **IEEE Transactions on Control Systems Technology**, 2015. ISSN 1063-6536.

HWANG, C.-L.; WU, H.-M. Trajectory tracking of a mobile robot with frictions and uncertainties using hierarchical sliding-mode under-actuated control. **IET Control Theory &amp Applications**, 2013. ISSN 1751-8652.

INOUE, R.; SIQUEIRA, A.; TERRA, M. Experimental results on the nonlinear h-infinity control via quasi-lpv representation and game theory for wheeled mobile robots. In: **2007 IEEE International Conference on Control Applications**. [S.l.: s.n.], 2007.

INOUE, R. S.; SIQUEIRA, A. A. G.; TERRA, M. H. Experimental results on the nonlinear∞control via quasi-LPV representation and game theory for wheeled mobile robots. 2009.

_____. Experimental results on the nonlinearsub∞/subcontrol via quasi-LPV representation and game theory for wheeled mobile robots. **Robotica**, 2009. ISSN 0263-5747.

INOUE, R. S. et al. Robust recursive linear quadratic regulator for wheeled mobile robots based on optical motion capture cameras. **Asian Journal of Control**, 2019.

ISHIHARA, J. Y.; TERRA, M. H.; CERRI, J. P. Optimal robust filtering for systems subject to uncertainties. **IEEE Transactions on Automatic Control**, 2015. ISSN 0005-1098.

JOSEPH, L. **Mastering ROS for Robotics Programming**. Packt Publishing, 2015. ISBN 9781783551798. Disponível em: <https://www.ebook.de/de/product/25685324/lentin_joseph_mastering_ros_for_robotics_programming.html>.

JOSHI, G.; CHOWDHARY, G. Adaptive control using gaussian-process with model reference generative network. In: **IEEE Conference on Decision and Control (CDC)**. [S.l.: s.n.], 2018.

_____. Deep model reference adaptive control. In: **2019 IEEE 58th Conference on Decision and Control (CDC)**. [S.l.: s.n.], 2019.

JOSHI, G.; CHOWDHARY, G.; WAANDERS, B. van B. Stochastic deep model reference adaptive control. In: **2021 60th IEEE Conference on Decision and Control (CDC)**. [S.l.: s.n.], 2021.

KANAYAMA, Y. et al. A stable tracking control method for an autonomous mobile robot. In: **Proceedings., IEEE International Conference on Robotics and Automation**. [S.l.]: IEEE Comput. Soc. Press, 1990. v. 13, n. 1, p. 289–291. ISSN 0169-1864.

KHALAJI, A. K.; MOOSAVIAN, S. A. A. Robust adaptive controller for a tractor–trailer mobile robot. **IEEE/ASME Transactions on Mechatronics**, 2014. ISSN 1083-4435.

KHALIL, H. K. **Nonlinear systems**. Second. [S.l.]: Prentice Hall, 2002. 750 p. ISBN 0130673893.

KIM, P. **MATLAB Deep Learning**. [S.l.]: Springer-Verlag GmbH, 2017. 103-120 p. ISBN 1484228456.

LEE, J.-Y. et al. Guidance of mobile robot navigation in urban environment using human-centered cloud map. In: IROS. [S.l.], 2019.

LEE, T. H.; GE, S. S. **Adaptive neural network control of robotic manipulators**. Singapore: World Scientific Publishing, 1998. (World Scientific Series In Robotics And Intelligent Systems).

LI, S. et al. Adaptive neural network tracking control-based reinforcement learning for wheeled mobile robots with skidding and slipping. **Neurocomputing**, 2018. ISSN 0925-2312.

MARTINS, N. et al. Trajectory tracking of a nonholonomic mobile robot with parametric and nonparametric uncertainties: A proposed neural control. 2008.

MATLAB. **version 9.9 (R2020b)**. Natick, Massachusetts: The MathWorks Inc., 2020.

MOHSENI, F.; VORONOV, S.; FRISK, E. Deep learning model predictive control for autonomous driving in unknown environments. **IFAC-PapersOnLine**, Elsevier BV, v. 51, n. 22, p. 447–452, 2018. ISSN 2405-8963.

PARK, B. S. et al. Adaptive neural sliding mode control of nonholonomic wheeled mobile robots with model uncertainty. In: **Control Systems Technology, IEEE Transactions on**. [S.l.]: Institute of Electrical and Electronics Engineers (IEEE), 2008. v. 18, n. 5, p. 1199 – 1206. ISSN 1063-6536.

_____. A simple adaptive control approach for trajectory tracking of electrically driven nonholonomic mobile robots. **IEEE Transactions on Control Systems Technology**, 2010. ISSN 1063-6536.

PAZDERSKI, D.; KOZŁOWSKI, K. Trajectory tracking control of skid-steering robot – experimental validation. **IFAC Proceedings Volumes**, Elsevier BV, v. 41, n. 2, p. 5377–5382, 2004. ISSN 1474-6670.

PENG, J. et al. Robust quadratic stabilization tracking control for mobile robot with nonholonomic constraint. In: **International Conference on Robotics and Automation Sciences**. Hong Kong: [s.n.], 2017.

SGRIGNOLI, V. **Prototipagem de um Robô de Tração Diferencial**. 2017. Trabalho de Conclusão de Curso (Engenharia Elétrica), UFSCar (Universidade Federal de São Carlos), São Carlos, Brasil.

SHEN, Z.; MA, Y.; SONG, Y. Robust adaptive fault-tolerant control of mobile robots with varying center of mass. **IEEE Transactions on Industrial Electronics**, 2018.

TAVEIRA, T. F. P. A.; SIQUEIRA, A. G.; TERRA, M. Adaptive nonlinear h-infinity controllers applied to a free-floating space manipulator. In: **IEEE International Conference on Control Applications**. [S.l.: s.n.], 2006.

Apêndices

APÊNDICE A

Especificações Plataforma

A plataforma robótica apresentada na Figura 13 é fabricada pela *Lynxmotion* e suas dimenções estão disponíveis na Tabela 8

Modelo	*Lynxmotion Aluminum A4WD1 Rover*
Comprimento	247mm (9.75")
Largura	203mm (8.0")
Altura	101mm (4.0")

Tabela 8 – Dimensões do Chassi do Robô

Os atuadores utilizados na movimentação da plataforma são fabricados pela *Hsiang Neng* modelo HN-GH12-1634T com redução no eixo de atuação de 30:1 e suas propriedades estão melhor descritas no Apêndice B.

O conjunto de rodas utilizado no experimento são fabricados pela *Traxxas* com suas especificações apresentadas na Tabela 9

Modelo	*SportTraxxas Talon* 4.8 x 2.7 x 2.8
Diâmetro Externo	121mm (4.8")
Largura	68.5mm (2.7")
Diâmetro Interno	71mm (2.8")

Tabela 9 – Especificações Técnicas das Rodas

APÊNDICE B

Especificações dos Motores

A propulsão do robô utilizado nos experimentos é realizada por quatro motores de corrente contínua, da fabricante *Hsiang Neng*, com as propriedades apresentadas na Tabela 10. Os parâmetros foram definidos sob as seguintes condições: Temperatura 25°, Umidade 60% e motor posicionado verticalmente. Vale a pena ressaltar que os motores não se destinam a reversão de rotação instantânea bem como suas engrenagens não possuem proteção contra água ou poeira. As dimensões dos motores são apresentados na Figura 20

Modelo	HN-GH12-1634T
Redução	30:1
Tensão Nominal	12V
Tensão de Operação	6 - 12V
Carga Nominal(12V)	0.78Kg-cm
Velocidade sem carga(12V)	200RPM +/- 10%
Velocidade à Carga Nominal	163RPM +/- 10%
Corrente sem carga à 12V	$<$ 115mA
Correte à à Carga Nominal	$<$ 285mA
Folga Axial	Máximo 0.8mm
Resistência de Isolamento	10 M ohm em 300V
Tensão Suportável	300V por 1 segundo

Tabela 10 – Especificações Técnicas do Motor DC

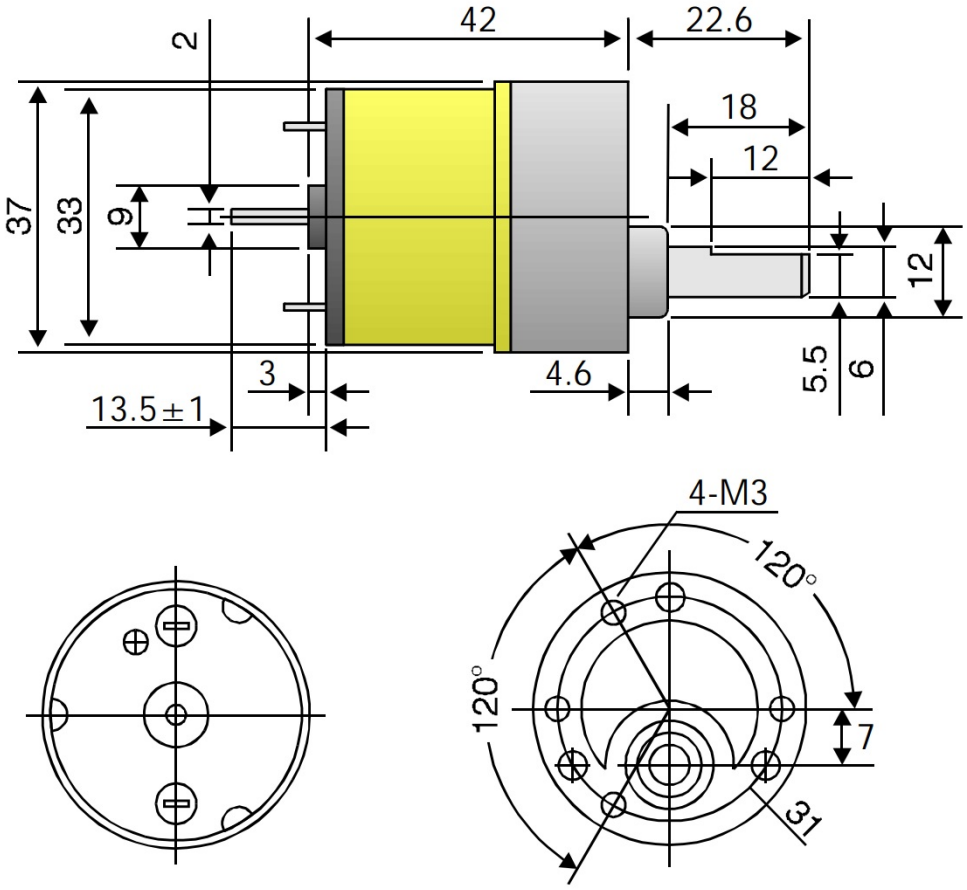


Figura 20 – Dimensões do Motor

APÊNDICE C

Encoders

Como forma de medir a velocidade e a direção da rotação dos motores, foram utilizados *Encoders* de Quadratura, da fabricante *US Digital* construídos especificamente para serem acoplados nos motores descritos no Apêndice B. As especificações técnicas podem ser encontradas na Tabela 11

Modelo	E4T
Ciclos por revolução (CPR)	100
Pulsos por revolução (PPR)	400
Frequência	30kHz

Tabela 11 – Especificações do *Encoder E4T*

Figura 21 – Encoder US Digital E4T

APÊNDICE D

Drivers

O controle de velocidade e direção dos motores do projeto é realizado através de *Drivers* da fabricante *Sabertooth*, sendo um dos mais eficientes e versáteis do mercado. Seu modo de operação pode ser selecionado na própria placa através de chaves *DIPs*. Possui três modos de operação: analógico, radio-controlado e serial, proteção contra superaquecimento e sobrecarga além freio-motor regenerativo para alguns tipos de baterias. Suas especificações técnicas estão disponíveis na Tabela 12

Modelo	*Sabertooth* 2x12
Tensão de Entrada (V)	6-24V
Corrente de Saída (A)	Contínuo 12A/Pico 25A
Fonte de Energia	NiMH, NiCd, LiOn, LiPo, Chumbo Ácido
Dimensões LxCXA (mm)	64 x 75 x 16

Tabela 12 – Especificações do *Driver*

APÊNDICE E

Código Arduíno

```
/* Jose Ceron Neto
/* Projeto de Mestrado
/*****************************************/
/*       Rob  M vel LynxMotionA4WD1        */
/*****************************************/

#include <TimerOne.h>
#include <TimerThree.h>
#include <string.h>
#include <stdio.h>
#include <SoftwareSerial.h>
//--------------------Declaring Global Variables--------------------

#define t_interrup 10000 // time interruption rate (10ms)
#define t_PWM 50 // Period of the PWM wave set to 50us or 20kHz

/*Port Selection for the Encoder*/
#define Sel2 53 // Bit Sel at the ports 36, 41, 47 and 53 commands the
    reading of the MSB or LSB from each encoder
#define OE2  51 // Bit OE at the ports  34, 39, 45 and 51 activates the
    reading mode for each encoder
#define RST2 49 // bit RST at the ports 32, 37, 43 and 49 resets the
    counter of the encoder
#define Sel1 47
#define OE1  45
#define RST1 43
#define Sel4 41
#define OE4  39
#define RST4 37
```

```
27 #define Sel3 36
   #define OE3  34
29 #define RST3 32

31 /* Encoder Reading Ports*/
   #define port52 52 //D7 //Reading ports
33 #define port50 50 //D6
   #define port48 48 //D5
35 #define port46 46 //D4
   #define port44 44 //D9
37 #define port42 42 //D2
   #define port40 40 //D1
39 #define port38 38 //D0
   boolean TimerFlag = LOW;
41
   /*Encoder Reading Variables*/
43 int EncoderPulses1 = 0; //Encoder 1 counting variable
   int EncoderPulses2 = 0; //Encoder 2 counting variable]
45 int EncoderPulses3 = 0; //Encoder 3 counting variable
   int EncoderPulses4 = 0; //Encoder 4 counting variable
47 float Speed1       = 0; //rad/s
   float Speed2       = 0; //rad/s
49 float Speed3       = 0; //rad/s
   float Speed4       = 0; //rad/s
51
   /*Motor Driving Variables*/
53 //Set the ports to drive the motor
   #define SBT1         35   //Sabertooth driver 1 - controls Left side Motors
       - HIGH activate command/LOW deactivate command
55 #define SBT2        33   //Sabertooth driver 2 - controls Right side Motors
      - HIGH activate command/LOW deactivate command
   #define SABER_TX_PIN 18   //Motor drive command
57
   #define SABER_RX_PIN   19   // Not in use - just for initialization
59 #define SABER_BAUDRATE 9600 // Baudrate defined by pins in drivers
   SoftwareSerial SaberSerial = SoftwareSerial( SABER_RX_PIN, SABER_TX_PIN );
61
   //Sets the variables to drive the motor
63 int SetPWM = 0; // from 0 to 255

65 #define SABER_MOTORA_FULL_FORWARD  127
   #define SABER_MOTORA_FULL_REVERSE  1
67 #define SABER_MOTORB_FULL_FORWARD  255
   #define SABER_MOTORB_FULL_REVERSE  128
69 /************************************
   For each Sabertooth:
71 - 0   STOP all motors
```

```
- 1   Set Motor A FULL Reverse
- 64  Set Motor A SOFT Stop
- 127 Set Motor A FULL Forward

- 128 Set Motor B FULL Reverse
- 192 Set Motor B SOFT Stop
- 255 Set Motor B FULL Forward
************************************/

/*Control Variables*/
const float Kc = 279.8404*0.5;
const float Td = 0;
const float Ti = 0.0132;
const float T  = 0.01;
const float Ki = T*Kc/Ti;
const float Kd = Kc*Td/T;
float prevError1 = 0;
float prevError2 = 0;
float prev1Speed1 = 0;
float prev1Speed2 = 0;
float prev2Speed1 = 0;
float prev2Speed2 = 0;
float SetPoint1 = 0; // from 0 to 27rad/s
float SetPoint2 = 0;
int uM1 = 0;
int uM2 = 0;

/*Communication Variables*/
#define TAM_MSG  15
char inputChar[TAM_MSG];
union Data {
  int i;
  char b[2];
  };

//----------------------Timer Interruption-----------------------

void TimeInterurpt(){ //Faz a EncoderReading dos Encoders 1 a 4
  EncoderPulses1 = EncoderReading(OE1, Sel1, RST1);
  EncoderPulses2 = EncoderReading(OE2, Sel2, RST2);
  EncoderPulses3 = EncoderReading(OE3, Sel3, RST3);
  EncoderPulses4 = EncoderReading(OE4, Sel4, RST4);
  TimerFlag = HIGH;
}
//----------------------Serial Interruption----------------------
```

```
119 void serialEvent(){
      getMsg ();
121 }

123 //--------------------------Ports Setup-------------------------------

125 void setup(){
      // put your setup code here, to run once:
127   Serial.begin(57600);
      Timer1.initialize(t_interrup);
129   Timer3.initialize(t_PWM);
      Timer1.attachInterrupt(TimeInterurpt);
131
      pinMode(RST1,OUTPUT); // Reset1 Pin
133   pinMode(RST2,OUTPUT); // Reset2 Pin
      pinMode(RST3,OUTPUT); // Reset3 Pin
135   pinMode(RST4,OUTPUT); // Reset4 Pin

137   pinMode(OE1,OUTPUT);  // OE1 Pin
      pinMode(OE2,OUTPUT);  // OE2 Pin
139   pinMode(OE3,OUTPUT);  // OE3 Pin
      pinMode(OE4,OUTPUT);  // OE4 Pin
141
      pinMode(Sel1,OUTPUT); // Sel1 Pin
143   pinMode(Sel2,OUTPUT); // Sel2 Pin
      pinMode(Sel3,OUTPUT); // Sel3 Pin
145   pinMode(Sel4,OUTPUT); // Sel4 Pin

147   pinMode(port38,INPUT);// Bit 00
      pinMode(port40,INPUT);// Bit 01
149   pinMode(port42,INPUT);// Bit 02
      pinMode(port44,INPUT);// Bit 03
151   pinMode(port46,INPUT);// Bit 04
      pinMode(port48,INPUT);// Bit 05
153   pinMode(port50,INPUT);// Bit 06
      pinMode(port52,INPUT);// Bit 07
155
      //Timer3.pwm(MotorPWM1,SetPWM1);
157   //Timer3.pwm(MotorPWM2,SetPWM2);
      pinMode( SABER_TX_PIN, OUTPUT );
159   pinMode( SBT1, OUTPUT );
      pinMode( SBT2, OUTPUT );
161   SaberSerial.begin( SABER_BAUDRATE );
      delay( 2000 ); // Time to initialize Sabertooth
163
      digitalWrite(OE1,HIGH);
165   digitalWrite(Sel1,HIGH);
```

```
  digitalWrite(OE2,HIGH);
  digitalWrite(Sel2 ,HIGH);
  digitalWrite(OE3,HIGH);
  digitalWrite(Sel3 ,HIGH);
  digitalWrite(OE4,HIGH);
  digitalWrite(Sel4 ,HIGH);
}

//-------------------------Loop Section-------------------------

void loop(){
   // put your main code here, to run repeatedly:
   if (TimerFlag == HIGH){
    //Velocidade [rad/s] = EncoderPulses*(2*pi/(18*2048*0.01))
    Speed1 = EncoderPulses1*(-0.01704423);
    Speed2 = EncoderPulses2*(+0.01704423);

    ControlePID1 ();
    ControlePID2 ();
    SetMotor ();

    TimerFlag = LOW;

    outputMsg (Speed1,Speed2);

    if(digitalRead(22)) {digitalWrite(22,LOW);}
    else {digitalWrite(22,HIGH);}
    }
  }
//-------------------------------------------------------------------
//                          Functions
//-------------------------------------------------------------------

//-------------------------Encoder Reading-------------------------

int EncoderReading(int OE, int Sel, int RST){
  digitalWrite(OE,LOW); //Initiate Reading
  digitalWrite(Sel,LOW);//Reading MSB
  boolean bit15 = digitalRead(port52);
  boolean bit14 = digitalRead(port50);
  boolean bit13 = digitalRead(port48);
  boolean bit12 = digitalRead(port46);
  boolean bit11 = digitalRead(port44);
  boolean bit10 = digitalRead(port42);
  boolean bit09 = digitalRead(port40);
  boolean bit08 = digitalRead(port38);
```

```
213   digitalWrite(Sel,HIGH); //Reading LSB
      boolean bit07 = digitalRead(port52);
215   boolean bit06 = digitalRead(port50);
      boolean bit05 = digitalRead(port48);
217   boolean bit04 = digitalRead(port46);
      boolean bit03 = digitalRead(port44);
219   boolean bit02 = digitalRead(port42);
      boolean bit01 = digitalRead(port40);
221   boolean bit00 = digitalRead(port38);

223   digitalWrite(OE,HIGH); // Finish Reading
      digitalWrite(RST,LOW); // Resets Counter
225   digitalWrite(RST,HIGH);// Restores Counter Reading

227   int EncoderPulses = 0  ; // Reset/create couter variable EncoderReading

229   // concatena  o dos bits
      EncoderPulses = EncoderPulses | (bit00 << 0 ); // bitXX goes to the
          position XX and than it is sumed up to the previous value ( Or
          operation )
231   EncoderPulses = EncoderPulses | (bit01 << 1 );
      EncoderPulses = EncoderPulses | (bit02 << 2 );
233   EncoderPulses = EncoderPulses | (bit03 << 3 );
      EncoderPulses = EncoderPulses | (bit04 << 4 );
235   EncoderPulses = EncoderPulses | (bit05 << 5 );
      EncoderPulses = EncoderPulses | (bit06 << 6 );
237   EncoderPulses = EncoderPulses | (bit07 << 7 );
      EncoderPulses = EncoderPulses | (bit08 << 8 );
239   EncoderPulses = EncoderPulses | (bit09 << 9 );
      EncoderPulses = EncoderPulses | (bit10 << 10 );
241   EncoderPulses = EncoderPulses | (bit11 << 11 );
      EncoderPulses = EncoderPulses | (bit12 << 12 );
243   EncoderPulses = EncoderPulses | (bit13 << 13 );
      EncoderPulses = EncoderPulses | (bit14 << 14 );
245   EncoderPulses = EncoderPulses | (bit15 << 15 );

247   if (EncoderPulses <= 32768){ // The bottom half of the counter is
          positive (the counter size is 2 to the power of 16)
       //EncoderPulses = EncoderPulses;
249   }
      else {
251     EncoderPulses = (EncoderPulses - 65535); // The top half of the counter
            is positive
      }
253   return EncoderPulses;
  }
255
```

```
//-------------------------Motor Driving-------------------------
257
void SetMotor (){
259  // Logic to convert the output of the controler to a Sabertooth reference
       .
   if (uM1 == 0){
261  digitalWrite(SBT1, HIGH);
   SaberSerial.write(byte(0));
263  digitalWrite(SBT1, LOW);
   }
265
   if (uM1 <> 0) {
267    digitalWrite(SBT1, HIGH);
   if(uM1 >= 100){
269    SaberSerial.write(byte(SABER_MOTORA_FULL_FORWARD));
     SaberSerial.write(byte(SABER_MOTORB_FULL_FORWARD));
271    digitalWrite(SBT1, LOW);
   }
273  else if (uM1 <= -100){
     SaberSerial.write(byte(SABER_MOTORA_FULL_REVERSE));
275    SaberSerial.write(byte(SABER_MOTORA_FULL_REVERSE));
     digitalWrite(SBT1, LOW);
277  }
   else {
279    unsigned char SpeedMotorA = map(uM1,-100,100,SABER_MOTORA_FULL_REVERSE,
         SABER_MOTORA_FULL_FORWARD);
     unsigned char SpeedMotorB = map(uM1,-100,100,SABER_MOTORB_FULL_REVERSE,
         SABER_MOTORB_FULL_FORWARD);
281    SaberSerial.write(byte(SpeedMotorA));
     SaberSerial.write(byte(SpeedMotorB));
283    digitalWrite(SBT1, LOW);
   }
285  }

287  if (uM2 == 0){
   digitalWrite(SBT2, HIGH);
289  SaberSerial.write(byte(0));
   digitalWrite(SBT2, LOW);
291  }


293
   if (uM2 <> 0) {
295    digitalWrite(SBT2, HIGH);
   if(uM2 >= 100){
297    SaberSerial.write(byte(SABER_MOTORA_FULL_FORWARD));
     SaberSerial.write(byte(SABER_MOTORB_FULL_FORWARD));
299    digitalWrite(SBT2, LOW);
```

```
  }
301  else if (uM2 <= -100){
    SaberSerial.write(byte(SABER_MOTORA_FULL_REVERSE));
303    SaberSerial.write(byte(SABER_MOTORA_FULL_REVERSE));
    digitalWrite(SBT2, LOW);
305  }
  else {
307    unsigned char SpeedMotorA = map(uM2,-100,100,SABER_MOTORA_FULL_REVERSE,
       SABER_MOTORA_FULL_FORWARD);
    unsigned char SpeedMotorB = map(uM2,-100,100,SABER_MOTORB_FULL_REVERSE,
       SABER_MOTORB_FULL_FORWARD);
309    SaberSerial.write(byte(SpeedMotorA));
    SaberSerial.write(byte(SpeedMotorB));
311    digitalWrite(SBT2, LOW);
  }
313  }
}
315

317 //------------------------PID Controller M1-------------------------

319 void ControlePID1 () {
  float error = SetPoint1-Speed1;
321

  float deltaU = Kc*(error - prevError1) + error*Ki - Kd*(Speed1 - 2*
     prev1Speed1 + prev2Speed1);
323  deltaU = floor(deltaU);
  uM1 = uM1 + deltaU;
325

  prevError1 = error;
327  prev1Speed1 = Speed1;
  prev2Speed1 = prev1Speed1;
329 }

331 //------------------------PID Controller M2-------------------------

333 void ControlePID2 () {
  float error = SetPoint2-Speed2;
335

  float deltaU = Kc*(error - prevError2) + error*Ki - Kd*(Speed2 - 2*
     prev1Speed2 + prev2Speed2);
337  deltaU = floor(deltaU);
  uM2 = uM2 + deltaU;
339

  prevError2 = error;
341  prev1Speed2 = Speed2;
  prev2Speed2 = prev1Speed2;
```

```
343 }

345 //------------------------Send Output Message----------------------

347 void outputMsg (float Speed1, float Speed2){
    char outputChar[8];
349 Data PrintVel1, PrintVel2;

351 PrintVel1.i = (int)(Speed1*100);
    PrintVel2.i = (int)(Speed2*100);
353
    outputChar[0] = '$';
355 outputChar[1] = PrintVel1.b[0];
    outputChar[2] = PrintVel1.b[1];
357 outputChar[3] = PrintVel2.b[0];
    outputChar[4] = PrintVel2.b[1];
359 outputChar[5] = '#';
    outputChar[6] = '\n';
361 outputChar[7] = '\0';

363 Serial.write(outputChar,8);
}
365
//------------------------Get Input Message----------------------
367
void getMsg (){
369 char inputChar[6];
    Serial.readBytes(inputChar,1);
371 Data getVel1, getVel2;

373 if (inputChar[0] == '$'){
      Serial.readBytes(inputChar,5);
375     if (inputChar[4] == '#'){
        getVel1.b[0] = inputChar[0];
377     getVel1.b[1] = inputChar[1];
        getVel2.b[0] = inputChar[2];
379     getVel2.b[1] = inputChar[3];

381     SetPoint1 = (float)getVel1.i/100;
        SetPoint2 = (float)getVel2.i/100;
383   }
  }
385 else{
    //clearSTR();
387 }
}
389
```

```
//------------------------Clear Input Buffer-------------------------
391
void clearSTR (){
393   int cnt = 0;
      for (int i = 0; i < TAM_MSG; i++)
395   {
        inputChar[i] = ' ';
397   }
}
```

codigoFonte/LynxMotion.ino

APÊNDICE F

Resultados da Simulação ROS-Gazebo

Nesta Seção serão apresentados os resultados preliminares da implementação do controle baseado na cinemática do RMR com base nas simulação realizada em ambiente *Gazebo*. Desta forma foram definidos seis coordenadas cartesianas (x, y) e enviadas ao polinômio gerador de trajetória de referência, descrito na Seção 3.2.1, as quais o robô deveria alcançar, com a condição iniciais $(x_o, y_0, \psi_o) = (0, 0, 0)$. Os pontos considerados foram, partindo do ponto inicial, $(5, 5), (5, -5), (-5, -5), (5, -5)$ e retornando à ele, $(0, 0)$.

Para o Controlador baseado na cinemática, definido na Seção 3.2.2, os ganhos foram selecionados empiricamente, como apresentado na Tabela 13

Ganho	Valor
k_x	0.5
k_y	3.0
k_ψ	1.0

Tabela 13 – Ganhos do Controle Baseado na Cinemática

Neste sentido é possível verificar que o Controle baseado na Cinemática cumpre o seu papel no acompanhamento da trajetória de referência, como é possível verificar através da Figura 22.

Sendo assim, na Figura 23 é possível verificar o acompanhamento da trajetória de referência ao longo do eixos de coordenadas. Tem-se que a Figura 23a, apresenta o acompanhamento da trajetória de referência ao longo do eixo X, e, de modo análogo, a Figura 23b, ao longo do eixo y.

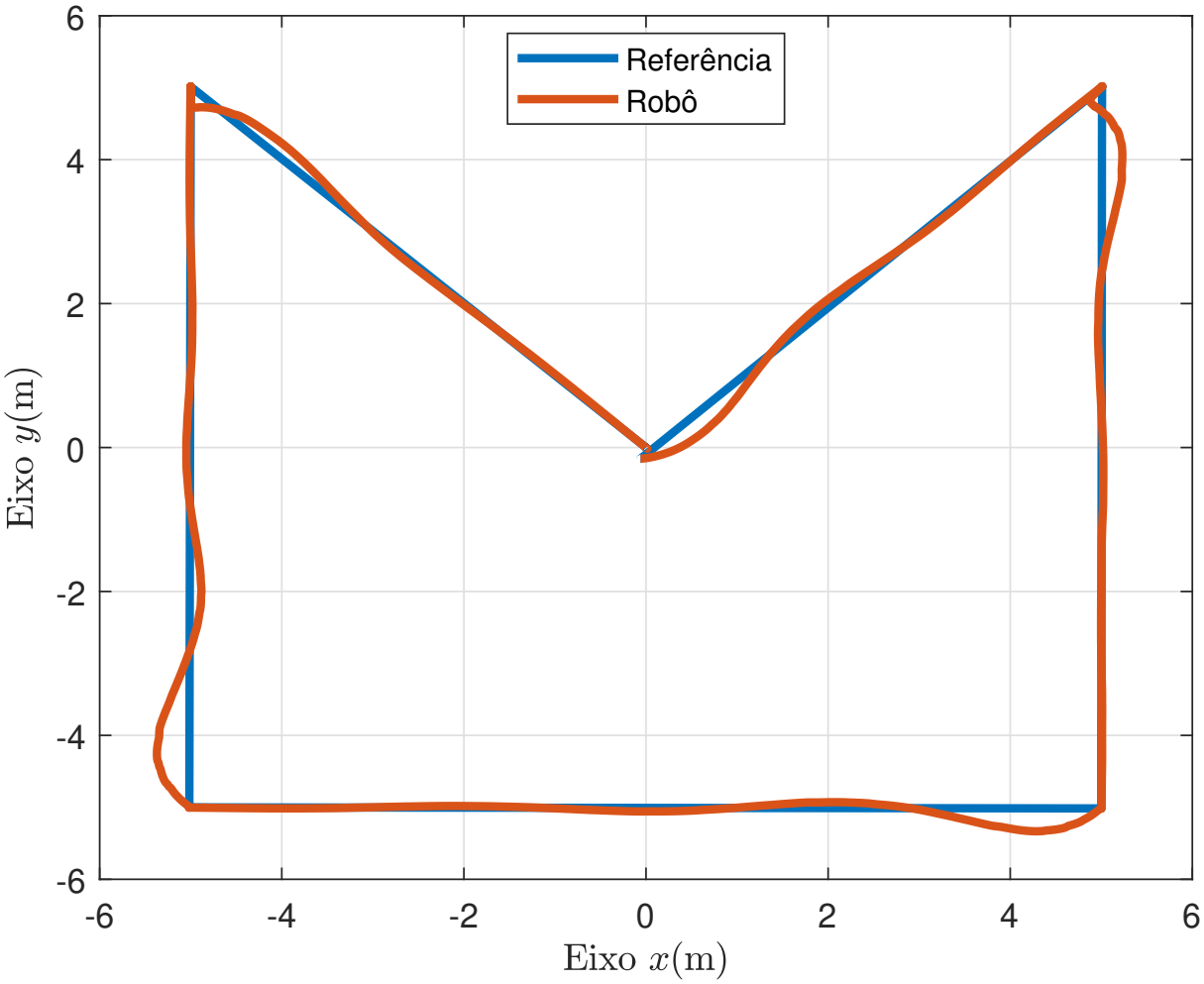
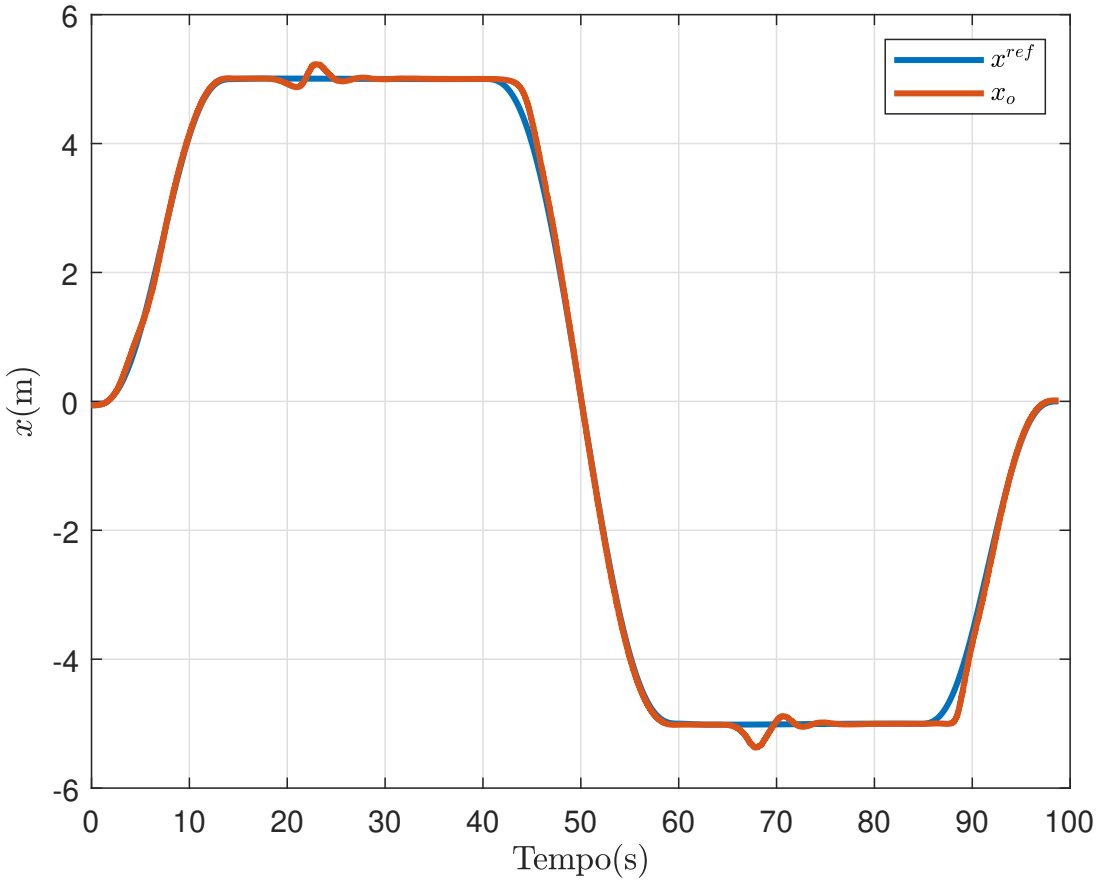


Figura 22 – Acompanhamento da Trajetória de Referência pelo RMR

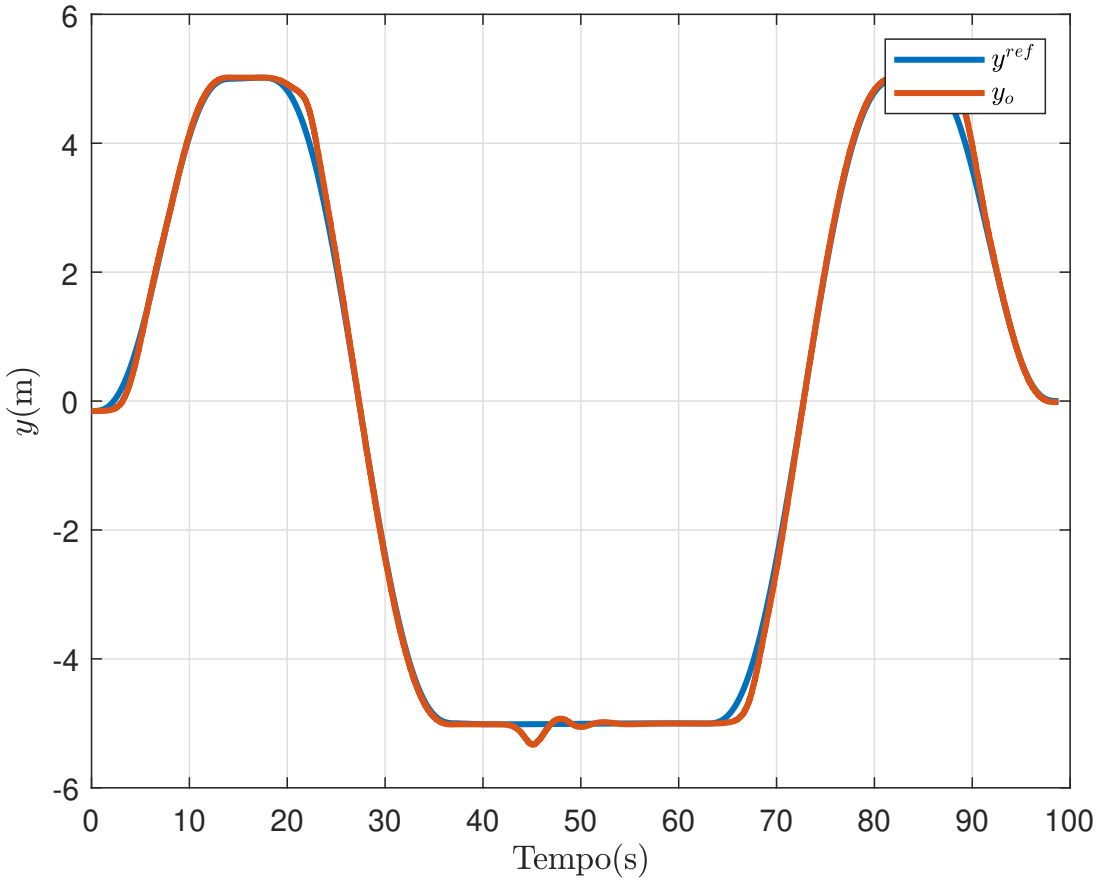
O erro de acompanhamento de acompanhamento da trajetória de referência ao longo do eixos de coordenadas é apresentado na Figura 24 e encontra-se dentro dos limites definidos como aceitáveis, durante esta simulação. É possível verificar que há um acúmulo de erro ao passar do tempo, já previsto e que deve ser atenuado com a utilização do controle adaptativo.

Em seguida, é possível observar na Figura 25 as saídas do controlador baseado em cinemática, as velocidades linear e angular desejadas, v^d e ω^d, respectivamente em Figura 25a e Figura 25b.

Portanto é possível afirmar que, durante o período de simulação, o controlador baseado na cinemática conseguiu percorrer a trajetória de referência, apesar do acúmulo de erro de posição, respeitando as restrições não holonômicas do robô. Sendo assim, pretende-se realizar a simulação para trajetórias em formato de círculo e posteriormente os testes com a plataforma robótica em solo.



(a) Acompanhamento da Trajetória em x



(b) Acompanhamento da Trajetória em y

Figura 23 – Acompanhamento da Trajetória nos Eixos x e y.

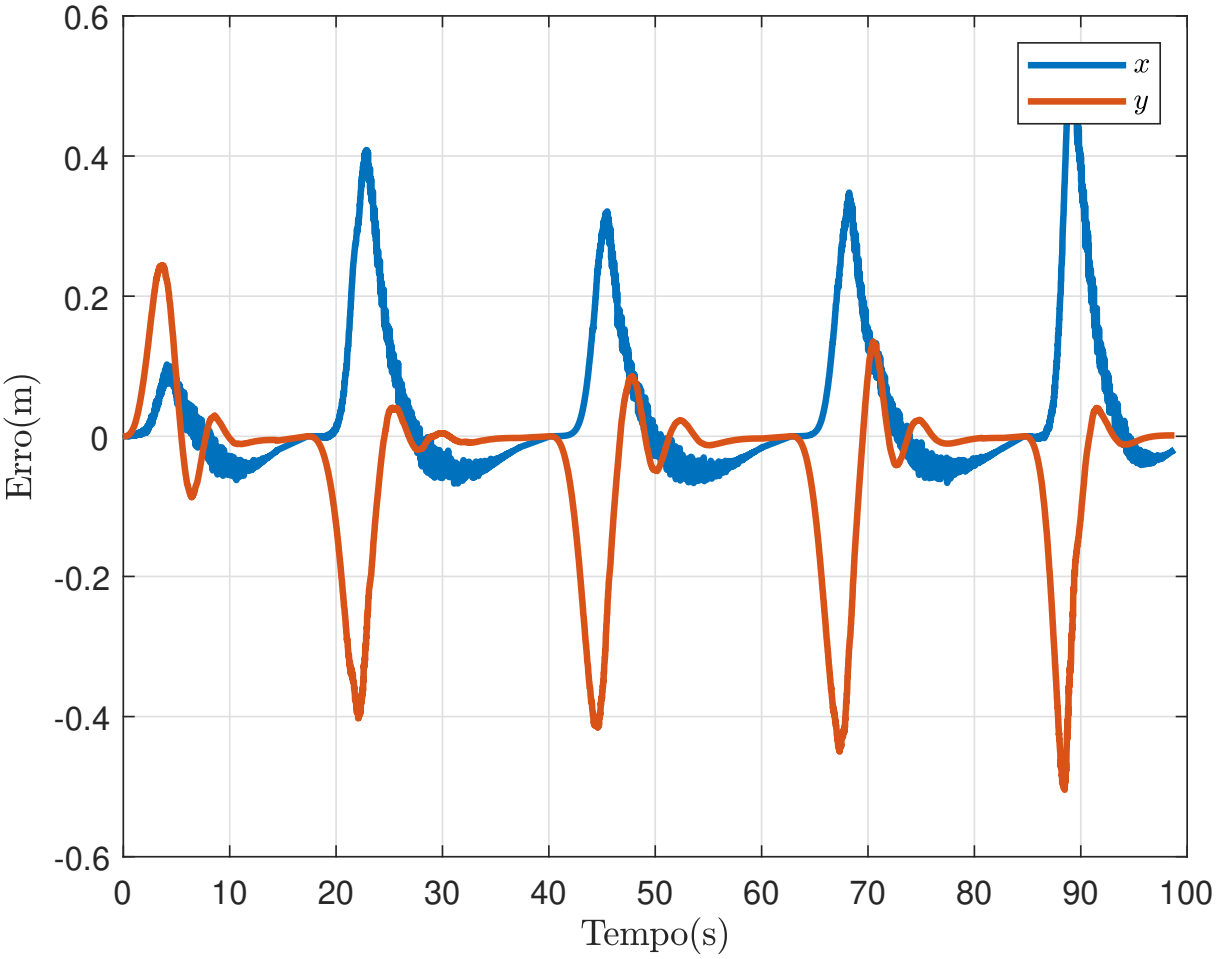
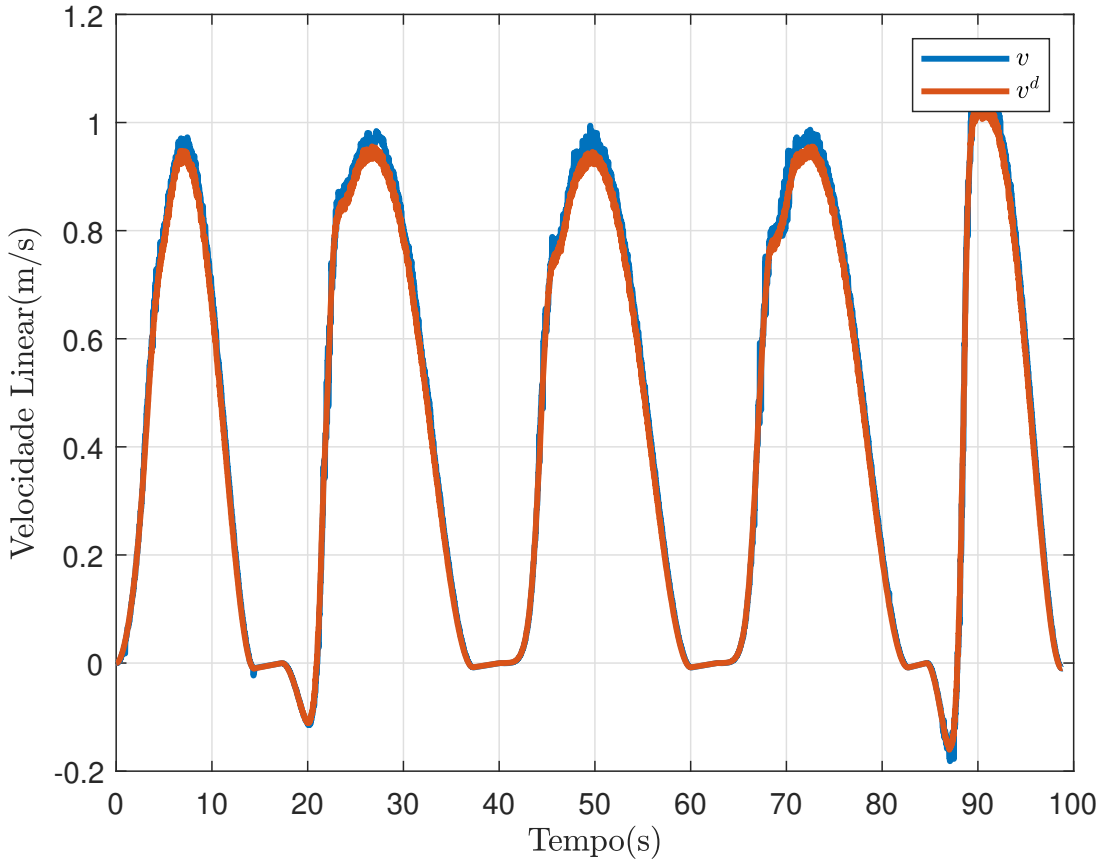
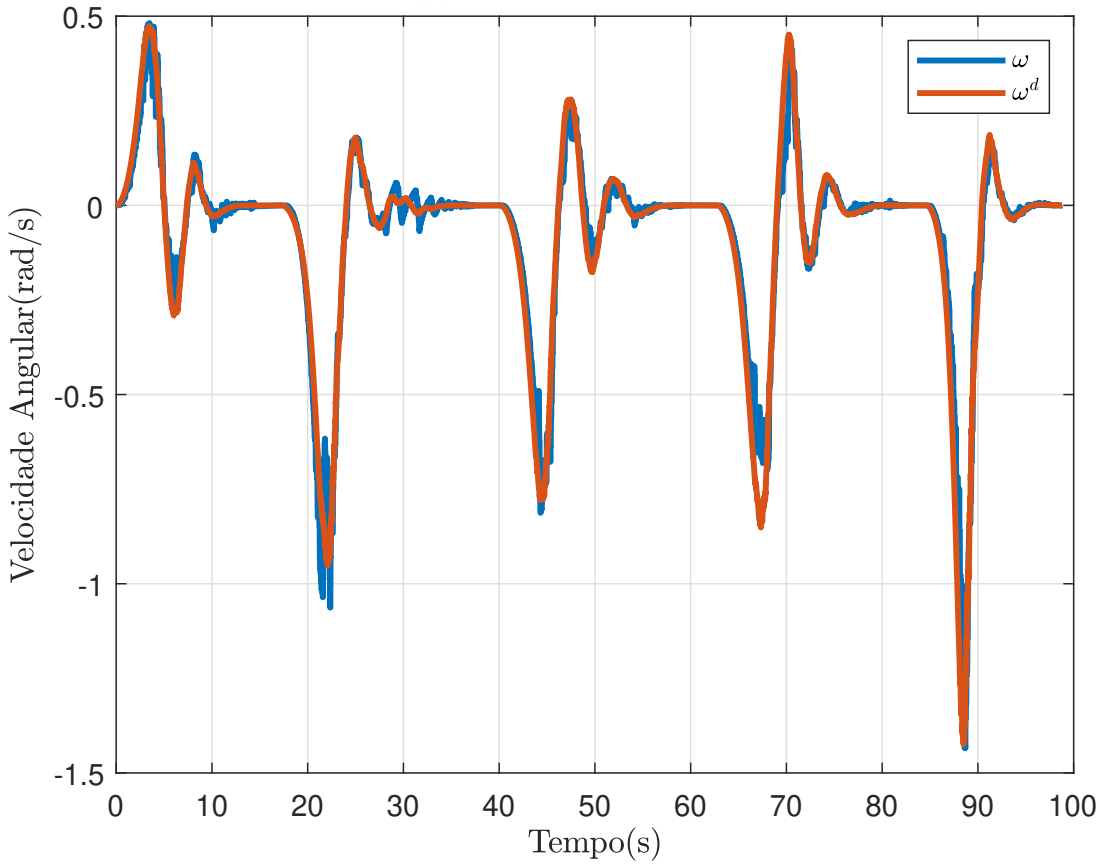


Figura 24 – Erro de Posição



(a) Velocidade Linear



(b) Velocidade Angular

Figura 25 – Velocidades Linear e Angular do RMR Durante o Acompanhamento da Trajetória de Referência

Anexos

ANEXO A

Robot Operating System

Este anexo tem por objetivo expandir o entendimento sobre o *framework*. Neste sentido, serão apresentadas as têS principais camadas que compõem a sua estrutura base.

A.1 Camadas ROS

As camadas ROS podem ser subdivididos em três: Primeiramente, trata-se do *Filesystem* ou Sistema de Arquivos, compreendendo os recursos ROS encontrados em disco e são nomeados e organizados de uma forma particular. Já a segunda camada, chamada de *Computational Graph* ou Grafo Computacional asemelha-se à uma rede *peer-to-peer*, onde os nós se comunicam par a par. A terceira apresenta-se como a parte colaborativa, chamado *Community*, oferece desde o intercâmbio de informações até a compilação de novas versões do sistema operacional (JOSEPH, 2015).

A.1.1 *Filesystem*

A camada de *Filesystem* ou Aquivos de Sistema do ROS é muito similar a um sistema operacional utilizado no dia a dia. Os arquivos são organizados logicamente no disco de uma forma particular, e possuem algumas categorias de classificação. Inicialmente tem-se a unidade básica do ROS , os *Packages* que podem conter desde as linhas de código do programa até detalhes de configuração do mesmo, a fim de constituir um módulo totalmente útil e reutilizável (JOSEPH, 2015). Em conjunto com os Pacotes ROS, observa-se o *Package manifest*, constituído de um arquivo com extensão *Extensible Markup Language*- Linguagem de marcação extensível (XML) contendo informações sobre o pacote, autor, licença, dependências e até mesmo *flags* de compilação (JOSEPH, 2015). Compreendida

a unidade base, há uma a maneira de referenciar um ou mais *packages* ROS distantes de uma forma conveniente e lógica, os *Metapackages* e de forma paralela encontram-se os *Metapackages Manifests* (JOSEPH, 2015).

Ainda quanto à organização, tem-se a estruturação dos dois tipos disponíveis de comunicação ROS: Mensagens e Serviços. No sistema de arquivos, para cada pacote há dois diretórios onde encontra-se a definição da estrutura de dados para realização de troca de mensagens ou requisição e resposta dos serviços: *msg* e um *srv*, respectivamente (JOSEPH, 2015). Para um melhor controle e versionamento disponibiliza-se o *Repository*, permitindo assim uma liberação automatizada dos pacotes que compartilham a mesma versão.

A.1.2 *Computational Graph*

Este nível conceitual do ROS compreende uma visão abstrata e de simples compreensão para o usuário, tratando a organização computacional como um grafo, onde os vértices representas os *Nodes* ROS e as arestas representam as trocas de informação através das *Messages* e *Services* tecendo assim uma rede de processos par a par. Por definição os *Nodes* ou Nós são os processos que realizam a computação (JOSEPH, 2015) e foram desenvolvidos para serem a menor unidade de computação, isolando e tolerando falhas além da redução de complexidade em comparação aos sistemas monolíticos (JOSEPH, 2015).

Um sistema robótico desenvolvido em ROS geralmente opera com vários *Nodes* - por exemplo, um nó controlará o sistema de movimentação, outro realizará o planejamento de trajetória e outro receberá os dados uma câmera - e assim a comunicação deve ser efetiva entre eles. Neste sentido, tem-se as *Messages* que são estruturas de dados simples contendo campos tipados, onde são permitidos desde tipos de dados primitivos, como inteiro, ponto flutuante e booleano até vetores (JOSEPH, 2015). Como forma de realizar a comunicação, os *Nodes* enviam suas mensagens para os *Topcis* que funcionam como encaminhadores nomeados onde pode haver a troca de mensagens de forma unidirecional, desassociando a produção de informação de seu consumo. Como forma de guardar os dados das mensagens, utiliza-se os *Bags* permitindo além do armazenamento, a análise, o processamento e a sua visualização (JOSEPH, 2015).

O conceito de mensagens é um paradigma de comunicação muito flexível, mas considerando o envio unidirecional e de muitos *publishers* para muitos assinantes não é muito apropriado para sistemas distribuídos, estabeleceu-se um outro tipo de troca de informações - os *Services*. Estes são baseados no paradigma de requisição e resposta, um nó oferece o serviço e um segundo nó chama por esse serviço enviando um requisição e aguardando a resposta do nó provedor(JOSEPH, 2015).

A fim de permitir a localização e comunicação dos *Nodes* entre si, utiliza-se do ROS *Master*. Desta forma é possível rastrear os nós que publicam mensagens aos *Topics* bem

como os nós assinantes assim como os provedores e o clientes de serviço e uma vez que os nós localizaram-se está estabelecida uma rede de comunicação par-a-par. Outo papel do ROS *Master* é prover o *Server Parameter*, que basicamente consiste em um dicionário onde os nós podem armazenar e receber parâmetros em tempo de execução com algumas restrições, por exemplo, para parâmetros de configuração (JOSEPH, 2015).

A.1.3 *Community*

No terceiro e último nível das Camadas ROS, destaca-se a troca de informações, conhecimento e até mesmo *softwares*. Uma série de recursos distintos e categorizados permitem este intercâmbio, como: Um fórum, contendo a documentação, manuais e repositório de códigos chamado ROS *Wiki* de acesso simples e com tradução em várias línguas é ideal para iniciantes e solução dos problemas mais comuns. Já o o ROS *Answers* apresenta mais de 53mil publicações no estilo perguntas e repostas catalogadas e com vários filtros para pesquisa.Entretanto quando há a identificação de um erro, ou pedido de um novo recurso a partir das questões levantadas nas páginas de perguntas e respostas, entre em cena o ROS *Issue Trackers*. Sobre novos projetos, notícias e novidades o ROS *Discourse* apresenta atualizações regulares(JOSEPH, 2015).

Neste sentido, uma das principais atividades desta camada é a liberação das Distribuições ROS, que são o conjunto versionado e estável de pacotes organizados de forma a facilitar a instalação e permitindo que os desenvolvedores trabalhem em uma base relativamente estável (JOSEPH, 2015).