UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

Ricardo Bedin Grando

ESTUDO DE REDES NEURAIS ARTIFICIAIS COMO ALTERNATIVA AO MÉTODO DO JACOBIANO PARA A CINEMÁTICA INVERSA

Santa Maria, RS
2019

Ricardo Bedin Grando

ESTUDO DE REDES NEURAIS ARTIFICIAIS COMO ALTERNATIVA AO MÉTODO DO JACOBIANO PARA A CINEMÁTICA INVERSA

Trabalho de Graduação apresentado à Graduação em Engenharia de Computaçãoda Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Engenharia de Computação**. Área de Concentração em Robótica, Inteligência Artificial, Automação, Cinemática,

ORIENTADOR: Prof. Rodrigo da Silva Guerra

Santa Maria, RS
2019

Ricardo Bedin Grando

ESTUDO DE REDES NEURAIS ARTIFICIAIS COMO ALTERNATIVA AO MÉTODO DO JACOBIANO PARA A CINEMÁTICA INVERSA

Trabalho de Graduação apresentado à Graduação em Engenharia de Computaçãoda Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Engenharia de Computação**. Área de Concentração em Robótica, Inteligência Artificial, Automação, Cinemática,

Aprovado em 6 de fevereiro de 2019:

Rodrigo da Silva Guerra, Dr. (UFSM)
(Presidente/Orientador)

Paulo Lilles Drews Junior, Dr. (FURG) (videoconferência)

Frederico Menine Schaf, Dr. (UFSM)

Santa Maria, RS
2019

DEDICATÓRIA

Nesse espaço gostaria de agradecer a todos que para a construção desse trabalho. Primeiramente à minha família. Minha mãe Arleide e meu pai Amelio foram a minha base de sustentação durante todo esse processo. Eles, juntamente com minha irmã Juliana, me motivaram a ser o melhor aluno possível. Também quero agradecer ao meus amigos da ECOMP e do Taura Bots, com quem compartilhei tantos bons momentos durante a graduação. Em especial, quero agradecer ao professor e orientador Rodrigo e ao aluno de mestrado Facrício por terem me ajudado no desenvolvimento desse trabalho. Por último, quero agradecer a UFSM e a cidade de Santa Maria por terem contribuído no meu desenvolvimento pessoal e profissional.

RESUMO

ESTUDO DE REDES NEURAIS ARTIFICIAIS COMO ALTERNATIVA AO MÉTODO DO JACOBIANO PARA A CINEMÁTICA INVERSA

AUTOR: Ricardo Bedin Grando
ORIENTADOR: Rodrigo da Silva Guerra

O problema da cinemática inversa é geralmente complexo e muitas soluções tradicionais são desenvolvidas somente para robôs de topologias específicas. O método iterativo baseado na Matriz Jacobiana (pseudo)inversa é uma abordagem bastante conhecida, provada e confiável que pode ser aplicada a uma grande variedade de manipuladores. Entretanto, esse método se baseia em linearizações que são válidas somente dentro de uma vizinhança bem próxima da pose atual do manipulador. Isso requer que o robô se mova em passos bem pequenos, recalculando intensivamente sua trajetória durante o caminho, fazendo essa abordagem ser ineficiente para certas aplicações. Redes neurais artificiais, dada sua capacidade conhecida de modelar sistemas não lineares, surgem como uma alternativa interessante. Neste trabalho é demonstrado que redes neurais artificiais podem de fato ser treinadas com sucesso para mapear deslocamentos no espaço em incrementos nas juntas, superando o método baseado na matriz jacobiana inversa quando trabalhando com grande incrementos de deslocamentos. O estudo é validado mostrando resultados comparativos para um braço planar hipotético de 3 juntas, um braço tridimensional de 3 juntas e no robô Thormang3.

Palavras-chave: Robótica, Rede Neural Artificial, Matriz Jacobiana, Cinemática Diferencial, Cinemática Inversa.

ABSTRACT

STUDY OF ARTIFICIAL NEURAL NETWORKS AS AN ALTERNATIVE TO THE JACOBIAN METHOD FOR INVERSE KINEMATICS

AUTHOR: Ricardo Bedin Grando
ADVISOR: Rodrigo da Silva Guerra

The inverse kinematics problem is generally very complex and many traditional solutions are targeted only to robots of certain specific topologies. The iterative method based on the (pseudo)inverse of the Jacobian matrix is a well-known, proven, and reliable general approach that can be applied to a wide variety of manipulators. However, it relies on linearizations that are only valid within a very tight neighborhood around the current pose of the manipulator. This requires the robot to move at very short steps, intensively recalculating its trajectory along the way, making this approach inefficient for certain applications. Neural networks, for their known capacity of modelling highly non-linear systems, appear as an interesting alternative. In this work is demonstrated that neural networks can indeed be successfully trained to map task space displacements into joint angle increments, outperforming the method based on the inverse of the Jacobian when dealing with larger displacement increments. The study is validated showing comparative results for hypothetical 3 joint planar arm, 3 joint 3D arm and the Thormang3 robot.

Keywords: Robotics, Neural Network, Jacobian Matrix, Differential Kinematics, Inverse Kinematics.

LISTA DE FIGURAS

Figura 2.1 – Braço robótico tridimensional de 3 DoFs. 22
Figura 2.2 – Braço robótico planar de 3 Dofs. 25
Figura 2.3 – Exemplo de comunicação entre dois nós no ROS. 31
Figura 2.4 – Arquitetura e Organização do Gazebo. 33
Figura 2.5 – Comparação dp robô Pionner2AT descrito no Gazebo e real 34
Figura 2.6 – Dimensões do Thormang3. 35
Figura 2.7 – Exemplo da descrição cinemática de uma junta em código. 37
Figura 2.8 – Representação gŕafica de uma célula perceptron. 40
Figura 2.9 – Representação gŕafica de uma ANN perceptron múlti-camada de uma camada escondida. 40
Figura 2.10 – Função de ativação sigmoid na forma de tangente hiperbólica. 42
Figura 2.11 – Função de ativação ReLU. 42
Figura 2.12 – Exemplo de ANN para *backpropagation*. 44
Figura 3.1 – Arquitetura de ANN utilizada. 50
Figura 3.2 – Pose inicial do braço no simulador. As linhas pretas representam os segmentos dos braços e o circulo azul representa a trajetória. 53
Figura 3.3 – Sumário da ANN utilizada para o braço planar 54
Figura 3.4 – Histograma mostrando o número de amostras para cada tamanho de passo dos *datasets* do braço planar 55
Figura 3.5 – Braço tridimensional de três tuntas no simulador Gazebo 57
Figura 3.6 – Sumário da ANN utilizada para o braço tridimensional 59
Figura 3.7 – Histograma mostrando o número de amostras para cada tamanho de passo do primeiro conjunto de *datasets* do braço tridimensional 61
Figura 3.8 – Histograma mostrando o número de amostras para cada tamanho de passo do primeiro conjunto de *datasets* do braço tridimensional 62
Figura 3.9 – Exemplo de trajetória no ambiente de teste. 63
Figura 3.10 – Sumário da ANN utilizada para o estudo de caso do Thormang3. 65
Figura 3.11 – Posição inicial do Thormang3 fixado no mundo 66
Figura 3.12 – Thormang3 com as caixas para verificar auto-colisões 67
Figura 3.13 – Histograma de um dos *datasets* do Thormang3. 68
Figura 3.14 – Exemplo de realização da trajetória. 69
Figura 4.1 – *Loss* de treinamento e validação dos modelos de ANN treinados. 70
Figura 4.2 – Resultados obtidos para o braço planar de três juntas. 71
Figura 4.3 – Comparação do desempenho entre a ANN e a JM inversa quando próximo a uma singularidade. 72
Figura 4.4 – *Loss* da ANN e comparação com JM inversa no alcance de 0 mm a 20 mm com 5° de delta. 72
Figura 4.5 – *Loss* da ANN e comparação com JM inversa no alcance de 20 mm a 50 mm com 15° de delta. 73
Figura 4.6 – *Loss* da ANN e comparação com JM inversa no alcance de 0 mm a 100 mm com 20° de delta. 73
Figura 4.7 – *Loss* da ANN e comparação com JM inversa no alcance de 50 mm a 100 mm com 20° de delta. 73
Figura 4.8 – *Loss* da ANN e comparação com JM inversa no alcance de 0 mm a 140 mm com 5° de delta. 74

Figura 4.9 – *Loss* da ANN e comparação com JM inversa no alcance de 140 mm a 350 mm com 15° de delta. 74
Figura 4.10 – *Loss* da ANN e comparação com JM inversa no alcance de 0 mm a 700 mm com 20° de delta. 74
Figura 4.11 – *Loss* da ANN e comparação com JM inversa no alcance de 350 mm a 700 mm com 20° de delta. 75
Figura 4.12 – *Loss* da ANN durante o treinamento. 76
Figura 4.13 – Comparação da trajetória realizada com um tamanho de passo de 1 mm 76
Figura 4.14 – Comparação da trajetória realizada com um tamanho de passo de 10 mm 77
Figura 4.15 – Comparação da trajetória realizada com um tamanho de passo de 25 mm 77
Figura 4.16 – Comparação da trajetória realizada com um tamanho de passo de 50 mm 77
Figura 4.17 – Comparação da trajetória realizada com um tamanho de passo de 75 mm 78
Figura 4.18 – Comparação da trajetória realizada com um tamanho de passo de 90 mm 78
Figura 4.19 – Comparação do MSE para todo alcance de tamanhos de passo treinados. 79
Figura 4.20 – Comparação do MSE com tamanho de passo entre 15 mm e 60 mm. .. 80

LISTA DE TABELAS

Tabela 2.1 – Alcance de ângulos para cada atuador do braço 36
Tabela 3.1 – Entradas e saídas da ANN para o braço planar 53
Tabela 3.2 – Limites dos *datasets* do braço planar 54
Tabela 3.3 – Entradas e saídas da ANN para o braço tridimensional de três juntas ... 59
Tabela 3.4 – Limites do segundo conjunto de *datasets* do braço tridimensional 60
Tabela 3.5 – Entradas e saídas da ANN para o estudo no Thormang3 64
Tabela 3.6 – Pontos no espaço da trajetória por pontos do ambiente de teste do Thormang3..... 68

LISTA DE ABREVIATURAS E SIGLAS

ANN	Artificial Neural Network - Rede Neural Artificial
GPU	Graphics Processing Unit - Unidade de Processamento Gráfico
FK	Forward Kinematics - Cinemática Direta
IK	Inverse Kinematics - Cinemática Inversa
DK	Differential Kinematics - Cinemática Diferencial
JM	Jacobian Matrix - Matriz Jacobiana
DoF	Degrees of Freedom - Graus de Liberdade
MSE	Mean Squared Error - Erro médio Quadrático
ROS	Robotics Operating System - Sistema Operacional de Robótica
SGD	Stochastic Gradient Descent - Gradiente Descendente Estocástico
LR	Learning Rate - Taxa de Aprendizagem
cm	Centimeters - Centímetros
mm	Millimeters - Milimetros
m	Meters - Metros

LISTA DE SÍMBOLOS

J	Jacobian Matrix - Matriz Jacobiana
s	Sino - Seno
c	Cosine - Cosseno
L	Length of the link - Comprimento do segmento

SUMÁRIO

1 INTRODUÇÃO 14
1.1 CONTEXTUALIZAÇÃO 14
1.2 RELEVÂNCIA 15
1.3 ABORDAGENS ANTERIORES 16
1.4 OBJETIVOS GERAIS E ESPECíFICOS 17
1.5 SEÇÕES DO TRABALHO 18
2 FUNDAMENTAÇÃO TEÓRICA 19
2.1 CINEMÁTICA DE ROBÔS 19
2.1.1 Cinemática Direta 20
2.1.2 Cinemática Inversa 24
2.1.2.1 Cinemática Diferencial - Jacobiano 25
2.2 ROS - ROBOTICS OPERATING SYSTEM 28
2.2.1 Conceitos Fundamentais 29
2.2.2 Desenvolvimento Colaborativo e Composição de Funcionalidade 30
2.2.3 Simulador Gazebo 31
2.2.3.1 Arquitetura e Ambiente 32
2.3 THORMANG3 34
2.3.1 Pacotes ROS e Simulação no Gazebo 36
2.4 REDES NEURAIS ARTIFICIAIS 38
2.4.1 Perceptron 39
2.4.2 Função de ativação 39
2.4.3 Treinamento Supervisionado 41
2.4.3.1 Função de Custo Erro Médio Quadrático 43
2.4.3.2 Backpropagation 44
2.4.3.3 Otimizador Adam 46
2.4.3.4 Regularização Dropout 47
3 METODOLOGIA 48
3.1 ARQUITETURA DA ANN 48
3.2 GERAÇÃO E NORMALIZAÇÃO DOS DADOS 50
3.3 AMBIENTE DE TESTE 51
3.4 BRAÇO PLANAR DE TRÊS JUNTAS 52
3.4.1 ANN 53
3.4.2 Geração de dados 53
3.4.3 Ambiente de Teste 56
3.5 BRAÇO TRIDIMENSIONAL DE TRÊS JUNTAS 56
3.5.1 ANN 59
3.5.2 Geração de dados 59
3.5.3 Ambiente de Teste 61
3.6 THORMANG3 - GAZEBO 61
3.6.1 ANN 64
3.6.2 Geração de dados 65
3.6.3 Ambiente de Teste 68
4 RESULTADOS 70
4.1 BRAÇO PLANAR DE TRÊS JUNTAS 70
4.2 BRAÇO TRIDIMENSIONAL DE TRÊS JUNTAS 72

4.3 BRAÇO DO THORMANG3 75
5 CONCLUSÃO E TRABALHOS FUTUROS 81
REFERÊNCIAS BIBLIOGRÁFICAS 83

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Nos últimos anos ocorreu o renascimento das redes neurais artificiais (ANNs) com o desenvolvimento do aprendizado profundo (LECUN; BENGIO; HINTON, 2015). Esse fenômeno, gerado essencialmente pela evolução da capacidade de processamento das unidades de processamento gráfico (GPUs) e pela *big data* (ZIKOPOULOS; EATON et al., 2011), tem sido um fator determinante no progresso de diversas áreas da tecnologia.

Muitos ambientes de desenvolvimento de *software* estão disponíveis para o treinamento e implantação de ANNs, e milhares de dispositivos de *hardware* customizados possuem fácil acesso, possibilitando a existência de um ambiente poderoso e versátil, que propícia um vasto número de aplicações computacionais. Algumas ANNs, por exemplo, têm se demonstrado muito eficientes em tarefas que possuem necessidade de automação. Identificar uma pessoa em uma imagem ou buscar por semelhanças de faces se tornou uma tarefa muito trabalhada nesse ambiente. Fundamental em redes sociais e locais de grande aglomeração de pessoas, o reconhecimento de faces chegou a ter uma acurácia de 99.63% com a FaceNet (SCHROFF; KALENICHENKO; PHILBIN, 2015). Analogamente, em Chiu et al. (2017) a tarefa de reconhecimento de voz utilizando uma um ANN do tipo LSTM (HOCHREITER; SCHMIDHUBER, 1997) apresentou erro WER (LIPPMANN et al., 1997) abaixo de 5% enquanto que em Alom et al. (2017) reconhecer o que alguém escreveu teve acerto superior a 98%. Mercado de ações (Rabiul Islam, 2018), geração de música (Quadrana et al., 2018), detecção de objetos (REDMON et al., 2016) e até mesmo imaginação e sonho (HA; SCHMIDHUBER, 2018); de modo geral, ANNs conseguem impactar em vários aspectos a sociedade atual.

A grande área da robótica, especificamente, tem sido um campo de grande extensão na aplicação de ANNs. Zhou e Wachs (2018), Martins, Custódio e Ventura (2018), Tang e Michmizos (2018), Eppe et al. (2018) são alguns exemplos recentes, dentre centenas. Mesmo com essa pequena amostra já é possível perceber que ANNs estão intrinsecamente presentes nas diversas subáreas da robótica, da robótica da medicina à robótica de serviços; de robôs autônomos aos robôs humanoides.

1.2 RELEVÂNCIA

Para realizar tarefas no mundo real, geralmente, as juntas que compõem um robô precisam se mover de um modo que o ponto final dessa configuração (orgão terminal) chegue a um certo ponto no espaço cartesiano ou siga uma certa trajetória no espaço. Essa relação entre posições das juntas dos robôs e posição cartesiana, e vice-versa, são estudadas, respectivamente, pela cinemática direta (FK) e pela cinemática inversa (IK) (SPONG et al., 2006), (CRAIG, 2005). Portanto, para encontrar a posição e orientação cartesiana do orgão terminal de um conjunto de ângulos das juntas se utiliza a FK enquanto que IK busca encontrar um conjunto de ângulos das juntas que representa em uma posição e orientação cartesiana desejada para o orgão terminal. Isso é a base para a geração de movimentos em robôs.

Como está abordado mais detalhadamente no Capítulo 2, enquanto que o problema da FK pode ser resolvido com regras simples, tipicamente multiplicando uma cadeia de matrizes de transformação homogêneas, o problema da IK é muito mais complexo por um número de razões. IK geralmente envolve um conjunto de equações não lineares, acopladas e algébricas que não possuem um algoritmo geral para serem solucionadas (CHAPELLE; BIDAUD, 2004). Além disso, geralmente uma solução fechada não pode ser encontrada (SCIAVICCO; SICILIANO, 2012).

Para algumas configurações de juntas de determinados membros de um robô, como um braço por exemplo, uma posição e orientação cartesiana do orgão terminal deste membro pode possuir mais de um conjunto de valores de juntas que à representa. Nesse caso de redundância, um conjunto infinito de soluções podem ser encontrados que mapeiam a mesma posição e orientação cartesiana (SCIAVICCO; SICILIANO, 2012). Também, pode não haver solução admissível, do ponto de vista da estrutura cinemática, se a posição e orientação desejada para o orgão terminal está fora da área de trabalho do robô.

Uma das formas mais comuns de resolver o problema da IK é através da cinemática diferencial (DK). A DK utiliza a matriz jacobiana (JM) para fazer uma relação entre uma pequena variação no espaço de juntas do robô e a variação correspondente na posição e orientação cartesiana do orgão terminal dessa configuração de juntas (BUSS, 2004). A JM depende da configuração inicial das juntas do robô que compõem o membro em questão e da posição do orgão terminal desse membro. Dessa forma, os componentes da JM são derivadas parciais que, para a pose atual, relaciona a taxa de mudança do valor das juntas com a taxa de variação da posição e orientação cartesiana do orgão terminal (SPONG et al., 2006).

A JM funciona como uma ferramenta para calcular a solução para a IK de forma iterativa quando invertida ou pseudo-invertida (caso a matriz não seja quadrada). Entretanto, devido a característica de não linearidade da IK, essa inversão ou pseudo-inversão é válida somente dentro de um pequeno intervalo próximo da posição inicial. Dessa forma, os

ângulos das juntas são incrementados a cada iteração, gerando uma trajetória que vai lentamente, passo a passo, para a posição final desejada. Além disso, tanto a matriz inversa ou a matriz psedo-inversa são computacionalmente custosas e não produzem resultados confiáveis quando próximos a singularidades, como discutido em Buss (2004).

Partindo dessa análise, este trabalho discute a implementação de ANNs para modelar a DK. Utilizando ANNs ao invés da JM inversa, busca-se expandir a vizinhança na qual a resultante da IK funciona e evitar a utilização de inversão de matrizes, que é numericamente instável próximo a singularidades. Esta abordagem foi aplicada em três estudos de caso: em um braço planar de três juntas/graus de liberdade (DoFs), em um braço tridimensional de três DoFs e no braço robótico de sete DoFs do robô Thormang3 (THORMANG3, 2018).

1.3 ABORDAGENS ANTERIORES

Muitos dos trabalhos que utilizam ANNs na cinemática de robôs têm focado em resolver o problema da IK de modo geral, buscando, essencialmente, mapear um conjunto de juntas para o seu respectivo orgão terminal, sem levar em consideração a DK. Tejomurtula e Kak (1999) fez uma análise profunda da abordagem de ANNs para resolver o problema da IK onde, assim como pode ser visto em Duka (2014) e em Hasan et al. (2010), uma ANN é capaz de resolver o problema. Entretando em todos a abordagem se resume a sistemas simples de dois e três DoFs que, como discutido em Goldenberg, Benhabib e Fenton (1985), já possuem solução fechada. O problema da IK com ANNs abordado dessa forma, de modo geral, tem um espaço não linear muito grande para ser modelado por uma ANN, mesmo quando testado com um braço robótico tridimensional de 3 DoFs, como concluído em Köker et al. (2004).

Como mencionado em Bingul, Ertunc e Oysu (2005), quando aplicado a um manipulador de 6 DoFs, que não possui solução fechada, uma abordagem que utiliza ANNs possui dificuldade para aproximar uma função descontínua e de muitas variáveis. Bingul, Ertunc e Oysu (2005) ainda conclui que um mapeamento grosseiro é obtido facilmente, mas um um representação precisa é muito difícil. Devido ao vasto alcance dos dados de treinamento, uma arquitetura de ANN muito grande é necessária para obter boa precisão. Os resultados de Bingul, Ertunc e Oysu (2005) mostram que bons resultados somente são obtidos dentro de uma resolução de 10 graus para as juntas.

Estudos mais recentes, entretanto, como em Almusawi, Dülger e Kapucu (2016), o problema da IK buscando mapear um conjunto de juntas para o seu respectivo orgão terminal começou a ter melhores resultados. O sistema Denso proposto por este foi aplicado em um braço robótico de 6 DoFs que não possui solução fechada. Entretanto, a boa precisão deste sistema e de sistemas semelhantes discutidos na comparação foi possível utilizando

ANNs muito grandes, que acarretam, dentre outras coisas, no difícil treinamento. Apesar de não mencionado, as 10 camadas da ANN do sistema Denso evidenciam a necessidade de um número elevado de parâmetros; da mesma forma para as 30 camadas do sistema PUMA 560, principal sistema na comparação.

Outros trabalhos que buscam modelar a DK com ANNs já foram desenvolvidos, tais como Aggarwal, Aggarwal e Urbanic (2014). Em Aggarwal, Aggarwal e Urbanic (2014) são alimentados à ANN somente a posição cartesiana para determinar a variação dos ângulos, sendo ignorado a orientação cartesiana pois o sistema possui somente três juntas. Além de levar em consideração esse detalhe importante no estudo de caso do Thormang3, o presente trabalho também se diferencia no tipo de dado alimentado à ANN, sendo a variação cartesiana do orgão terminal e o conjunto das posições angulares as entradas.

De modo geral, a ideia de mapear um conjunto de juntas para o seu respectivo orgão terminal atingiu os melhores resultados quando explorando uma abordagem híbrido entre ANNs e outras técnicas, como algoritmos genéticos por exemplo (KÖKER, 2013). Partindo dessa análise, o presente trabalho também propõe o desenvolvimento de um sistema híbrido entre ANNs e DK para um trabalho futuro, como está discutido com mais detalhes no Capítulo 5.

1.4 OBJETIVOS GERAIS E ESPECÍFICOS

O objetivo geral para os três estudos de caso é comparar o desempenho da JM inversa com um modelo de ANN na solução da IK. Para cada estudo, busca-se validar essa comparação utilizando como métrica o erro médio quadrático em tarefas que, iterativamente, descrevem uma trajetória no espaço.

Em todas as tarefas busca-se definir uma trajetória perfeita dividida entre vários pontos a uma distância fixa entre eles. Comparando a distância entre a posição cartesiana esperada para cada ponto e a posição que foi calculada tanto pela ANN quanto pela JM inversa, em média, espera-se que a ANN chegue mais próximo ao ponto desejado do que o método clássico; principalmente para distâncias muito grande entre os pontos. O resultado geral final esperado é que enquanto o erro da JM inversa é pequeno para pequenos tamanhos de passo e cresça de forma exponencial conforme o aumento da distância entre os pontos, o erro da ANN cresça a um ritmo menor, tendendo a uma curva linear de crescimento e superando o erro médio do método clássico a partir de um determinado tamanho de passo.

Mais especificamente, nos estudos de caso do braço planar e do braço tridimensional busca-se realizar uma trajetória circular e espiral no espaço, respectivamente, e usar os resultados obtidos para validar o estudo de caso do Thormang3, que possui um braço robótico típico. Para este último estudo, a tarefa utilizada para validação, trajetória

por pontos, serve de base para a comparação com tarefas típicas que um manipulador do tipo pode realizar, uma vez que vários pontos no espaço são definidos e o orgão terminal precisa alcançá-los.

Também no estudo de caso do braço planar, busca-se comparar o desempenho na proximidade de singularides. Espera-se que a ANN se comporte melhor e realize uma trajetória mais próxima da trajetória esperada do que o método clássico.

1.5 SEÇÕES DO TRABALHO

No Capítulo 2 do trabalho é realizada uma discussão da fundamentação teórica do trabalho. Na primeira parte desse capítulo é discutida a cinemática de robôs aplicada aos estudos de casos do trabalho e o sistema operacional de robôs (ROS) e sua interface com o simulador Gazebo. Na segunda parte desse capítulo discute-se sobre o robô Thormang3, utilizado no terceiro estudo de caso, e todos os conceitos envolvendo redes neurais artificias e técnicas utilizadas para treiná-las.

Na sequência, no Capítulo 3 ocorre a discussão de como a proposta do trabalho foi desenvolvida. Nesse capítulo, primeiramente é discutido termos comuns aos três estudos de caso e depois especificidades de cada caso.

Nos dois últimos Capítulos, 4 e 5, são apresentados os resultados obtidos e a conclusão que foi inferida destes no trabalho e como esta pode influenciar o desenvolvimento de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta Seção estão discutidos conceitos e técnicas utilizadas para o desenvolvimento deste trabalho.

2.1 CINEMÁTICA DE ROBÔS

A cinemática de robôs busca descrever movimentos de um robô sem considerar as forças e torques do movimento. Ela aplica geometria para estudar cadeias de juntas que compõem a estrutura de sistemas robóticos para encontrar equações que as descrevam. Nesta Seção é abordado com detalhes a FK e a IK.

Como é descrito em Spong et al. (2006), um robô manipulador é composto por um conjunto de segmentos ou corpos conectados por juntas. Existem vários tipos de juntas, sendo as juntas de revolução e prismáticas as mais comuns. As juntas de revolução permitem uma rotação sobre um eixo, enquanto que as juntas prismáticas permitem um movimento linear sobre um eixo, gerando uma extensão ou retração.

Cada junta possui um único grau de liberdade (DoF), que é um nó mecânico que permite movimento. O ângulo de cada junta é um valor real; ângulo de rotação para juntas de revolução e o deslocamento no caso de juntas prismáticas, por exemplo. Um robô manipulador de N juntas terá N + 1 segmentos, já que uma junta de conecta com dois segmentos. Desta forma a junta i conecta o segmento $i-1$ e i. Quando a junta i é atuada, o segmento i move. Consequentemente o segmento 0 é fixa, não se movendo quando as juntas atuam.

Nesta Seção é analisada a cinemática do primeiro e do segundo estudo de caso, braço planar de 3 DoFs e braço tridimensional de 3 DoFs, respectivamente. A análise é genérica e pode ser aplicada para qualquer robô manipulador. Como está descrito brevemente na introdução, o presente trabalho também é aplicado no braço do robô Thormang3 no simulador 3D Gazebo, o terceiro estudo de caso. Para esse estudo de caso, que possui uma descrição cinemática bastante complexa, foi utilizado uma descrição cinemática pronta, escrita em código[1]. Mais detalhes sobre as ferramentas nesse estudo de caso está apresentado nas seções de ROS e Thormang3.

Primeiramente é discutido sobre o problema da FK, que busca determinar a posição e orientação cartesiana do orgão terminal das juntas de um robô. Após, é discutido sobre o problema de determinar o valor das juntas, de acordo com uma posição e orientação cartesiana, o problema da IK.

[1] Pacote ROS com as funções de cinemática: https://github.com/ROBOTIS-GIT/ROBOTIS-THORMANG-MPC/blob/master/thormang3_kinematics_dynamics/src/kinematics_dynamics.cpp

2.1.1 Cinemática Direta

A FK se baseia no uso de transformações afins, parte da álgebra linear que busca mapear um espaço em outro. Para representar uma transformação afim são usadas coordenadas homogêneas, tornando possível a representação de um vetor de duas dimensões (x, y) em um vetor de três dimensões, $(x, y, 1)$.

Usando este sistema, uma translação, por exemplo, pode ser definida com uma multiplicação de matrizes. Ou seja, a sua forma funcional $x' = x + t_x$; $y' = y + t_y$ é representada como na Equação 2.1.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.1}$$

Todas as transformações lineares arbitrárias estão inclusas no conjunto de transformações afins e podem ser descritas como uma forma simplificada de transformações afins. Assim, qualquer transformação linear pode ser representada por uma matriz de transformação geral expandindo a matriz em uma coluna e uma linha, preenchendo o espaço extra com zeros com exceção do canto o direito baixo, que precisa ser preenchido com 1. Por exemplo, uma matriz de rotação no sentido anti-horário, onde $x' = xcos(\theta) - ysen(\theta)$ e $y' = xsen(\theta) + ycos(\theta)$, nessa representação pode ser definida na forma da Equação 2.2.

$$\begin{bmatrix} cos_\theta & -sen_\theta & 0 \\ sen_\theta & cos_\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

A ideia da FK gira em torno de representar um ponto final em um sistema de coordenadas base; ou seja, fazer multiplicações sucessivas de transformações afins. Em um manipulador, cada junta é representada por uma matriz de rotação e cada segmento por uma matriz de translação e, com isso, o ponto final é definido por uma sequência de multiplicações de matrizes de rotação e translação; uma matriz para cada junta e uma matriz de translação do segmento com essa junta. Matematicamente:

$$H = R_1(T_1)R_2(T_2)...R_n(T_n) \tag{2.3}$$

Sendo H a matriz de transformação do ponto final para a base e n o número de juntas. Essas matrizes de translação e rotação em um sistema de três dimensões, por

exemplo, são definidas na forma das Equações 2.4, 2.5, 2.6 e 2.7.

$$R_z = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

$$R_y = \begin{bmatrix} c_{\theta_i} & 0 & -s_{\theta_i} & 0 \\ 0 & 1 & 0 & 0 \\ s_{\theta_i} & 0 & c_{\theta_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\theta_i} & -s_{\theta_i} & 0 \\ 0 & s_{\theta_i} & c_{\theta_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

$$T = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

Sendo dx, dy e dz a translação nas respectivas dimensões cartesianas e c e s, seno e cosseno respectivamente. Esse processo é suficiente para resolver o problema da FK; determinar as matrizes R e T de cada junta e segmento e multiplicá-las.

Para um melhor entendimento desses conceitos e para futuras aplicações nesse trabalho, a FK utilizando matrizes de transformação homogênea está resolvida na sequência para o estudo de caso do braço tridimensional, que envolve um braço tridimensional de três juntas. O intuito é encontrar as equações genéricas que descrevem x, y e z para que estas possam ser utilizadas para encontrar a JM. O braço desse estudo de caso está na Figura 2.1 e, como pode ser observado, sua FK é determinada através da multiplicação das transformações afins da Equação 2.8.

$$H = R_z(\theta 1)T_z(L_1)R_y(\theta 2)T_z(L_2)R_y(\theta 3)T_z(L_3) = A_1A_2A_3 \tag{2.8}$$

Ou seja, a primeira junta possui uma rotação no eixo z, a segunda e a terceira junta estão transladadas no eixo z, rotacionando em y. L_1, L_2 e L_3 é o comprimento do segmento entre as juntas, transladados em z; ou seja: dz da primeira translação é L_1, dz da segunda translação é L_2 e assim por diante. Devido à configuração inicial na qual foi

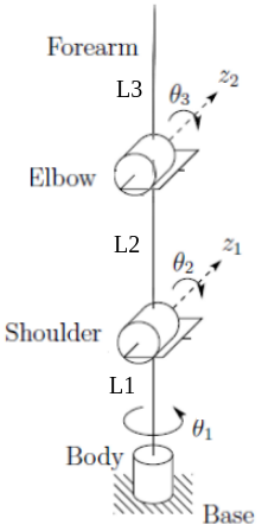


Figura 2.1 – Braço robótico tridimensional de 3 DoFs.

realizada a cinemática, dx e dy são nulos em todas as translações.

Como base nas matrizes de transformação afins previamente exemplificadas 2.4, 2.5, 2.6 e 2.7, pode-se definir A_1, A_2 e A_3, vide Equação 2.9.

$$A_1 = \begin{bmatrix} c_{\theta_1} & -s_{\theta_1} & 0 & 0 \\ s_{\theta_1} & c_{\theta_1} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

$$A_2 = \begin{bmatrix} c_{\theta_2} & 0 & -s_{\theta_2} & 0 \\ 0 & 1 & 0 & 0 \\ s_{\theta_2} & 0 & c_{\theta_2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & L_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.10}$$

$$A_3 = \begin{bmatrix} c_{\theta_3} & 0 & -s_{\theta_3} & 0 \\ 0 & 1 & 0 & 0 \\ s_{\theta_3} & 0 & c_{\theta_3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & L_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.11}$$

Por último, multiplicando as matrizes de transformação A_1, A_2 e A_3, encontra-se a

matriz de transformação homogênea simplificada na origem, vide Equação 2.12.

$$H = \begin{bmatrix} c_{\theta_1}c_{\theta_2}c_{\theta_3} - s_{\theta_3}c_{\theta_1}s_{\theta_2} & -s_{\theta_1} & c_{\theta_1}c_{\theta_2}s_{\theta_3} + s_{\theta_2}c_{\theta_1}c_{\theta_3} & c_{\theta_1}c_{\theta_2}s_{\theta_3}L_3 + s_{\theta_2}c_{\theta_1}c_{\theta_3}L_3 + c_{\theta_1}s_{\theta_2}L_3 \\ s_{\theta_1}c_{\theta_2}c_{\theta_3} - s_{\theta_3}s_{\theta_1}s_{\theta_2} & c_{\theta_1} & s_{\theta_1}c_{\theta_2}s_{\theta_3} + s_{\theta_1}s_{\theta_2}c_{\theta_3} & s_{\theta_1}c_{\theta_2}s_{\theta_3}L_3 + s_{\theta_1}s_{\theta_2}c_{\theta_3}L_3 + s_{\theta_1}s_{\theta_2}L_2 \\ -s_{\theta_2}c_{\theta_3} - c_{\theta_2}s_{\theta_3} & 0 & -s_{\theta_2}s_{\theta_3} + c_{\theta_2}c_{\theta_3} & -s_{\theta_2}s_{\theta_3}L_3 + c_{\theta_2}c_{\theta_3}L_3 + c_{\theta_2}L_1L_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.12}$$

Onde as posições cartesianas x, y e z são, respectivamente, os elementos de translação $a03$, $a13$ e $a23$ da matriz de transformação na origem.

$$x = c_{\theta_1}c_{\theta_2}s_{\theta_3}L_3 + s_{\theta_2}c_{\theta_1}c_{\theta_3}L_3 + c_{\theta_1}s_{\theta_2}L_3 \tag{2.13}$$

$$y = s_{\theta_1}c_{\theta_2}s_{\theta_3}L_3 + s_{\theta_1}s_{\theta_2}c_{\theta_3}L_3 + s_{\theta_1}s_{\theta_2}L_2 \tag{2.14}$$

$$z = -s_{\theta_2}s_{\theta_3}L_3 + c_{\theta_2}c_{\theta_3}L_3 + c_{\theta_2}L_1L_2 \tag{2.15}$$

Se aplicado também ao braço planar de 3 DoFs da Figura 2.2 (objeto de estudo do primeiro caso), pode-se obter a relação de matrizes, vide 2.16, 2.17, 2.18 e 2.19.

$$A_1 = \begin{bmatrix} c_1 & -s_1 & L_1c_1 \\ s_1 & c_1 & L_1s_1 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.16}$$

$$A_2 = \begin{bmatrix} c_2 & -s_2 & L_2c_2 \\ s_2 & c_2 & L_2s_2 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.17}$$

$$A_3 = \begin{bmatrix} c_3 & -s_3 & L_3c_3 \\ s_3 & c_3 & L_3s_3 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.18}$$

$$T_3^0 = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ 0 & 0 & 1 \end{bmatrix} \tag{2.19}$$

Sendo os coeficientes da Equação 2.20.

$$\begin{aligned}
a_{00} &= (-s_{12} + c_{12})c_3 + (-s_1c_2 - s_2c_1) * s_3, \\
a_{01} &= (-s_{12} + c_{12})s_3 + (-s_1c_2 - s_2c_1) * c_3, \\
a_{02} &= c_1L_1 - s_{12}L_2 + c_{12}L_2 + L_3(-s_{12} + c_{12})c_3 + L_3(-s_1c_1 - s2c_1)s_3, \\
a_{10} &= (-s_{12} + c_{12})s_3 + (-s_1c_2 - s_2c_1) * c_3, \\
a_{11} &= (-s_{12} + c_{12})c_3 + (-s_1c_2 - s_2c_1) * s_3, \\
a_{12} &= s_1L_1 + s_1c_2L_2 + c_1s_2L_2 + L_3(-s_{12} + c_{12})s_3 + L_3(s_1c_2 + s2c_1)c_3
\end{aligned} \tag{2.20}$$

Diferentemente da matriz 2.12, a matriz de tranformação 2.19 possui uma dimensão a menos por se tratar de um braço planar. Além disso, pode-se perceber que a complexidade dos elementos da matriz de transformação é maior mesmo se tratando de um exemplo mais simples, em uma dimensão a menos. Isso se deve ao fato de que a matriz 2.19 não possui os seus elementos simplificados, o que ocorre em 2.12. Por último, de 2.19 a os elementos da posição cartesiana podem ser inferidos, vide Equações 2.21 e 2.22.

$$x = c_1L_1 - s_{12}L_2 + c_{12}L_2 + L_3(-s_{12} + c_{12})c_3) + L_3(-s_1c_1 - s2c_1)s_3) \tag{2.21}$$

$$y = s_1L_1 + s_1c_2L_2 + c_1s_2L_2 + L_3(-s_{12} + c_{12})s_3) + L_3(s_1c_2 + s2c_1)c_3) \tag{2.22}$$

O processo realizado acima é genérico para realizar a FK de qualquer braço ou membro de um robô. Entretanto, a análise da cinemática de um membro com muitos DoFs pode se tornar complexa, como mencionado em Spong et al. (2006). É possível facilitar e otimizar esse processo utilizando a convenção Denavit-Hartenberg (DHC) (SPONG et al., 2006) e outros métodos. As equações discutidas nesta Seção de FK para os dois primeiros estudos de caso são utilizadas nas demais seções de revisão sobre cinemática e aplicadas na metodologia de seus respectivos estudos de caso.

2.1.2 Cinemática Inversa

O problema da IK busca encontrar o valor para as juntas $q_1, q_2, ..., q_n$ dado uma posição e orientação cartesiana. Como pode ser observado no exemplo 3.7 de Spong et al. (2006), em muitos casos é inviável resolver a IK diretamente na forma fechada devido a complexidade do sistema de equações trigonométricas pode ter. Enquanto que a FK sempre resulta em somente uma solução, a IK pode ou não pode ter solução e, se existir,

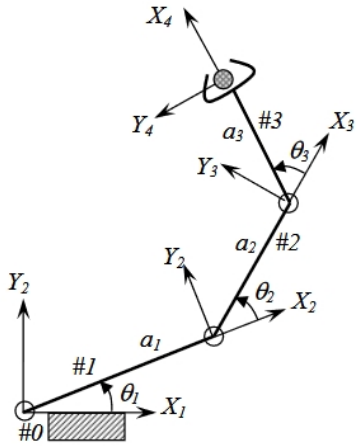


Figura 2.2 – Braço robótico planar de 3 Dofs.

Fonte: Spong et al. (2006).

pode ser única ou não. Dessa forma é necessário utilizar técnicas s sistemáticas que explorem a estrutura cinemática do manipulador.

O problema da IK é geralmente abordado utilizando um método geométrico (FEATHERSTONE, 1983), algébrico (DUFFY, 1980) ou iterativo (BUSS, 2004). Como discutido em Spong et al. (2006), os métodos iterativos resolvem o problema da IK incrementando as juntas dos ângulos passo a passo buscando minimizar a diferença entre a posição e a orientação atual do orgão terminal. Um método iterativo bastante usado se baseia em DK e será discutido na sequência.

2.1.2.1 Cinemática Diferencial - Jacobiano

A DK, também chamada de cinemática de velocidade, relaciona a variação cartesiana do orgão terminal com a variação das juntas. Em outras palavras, é a relação de quando uma mudança no espaço cartesiano acarreta na mudança do espaço de juntas. Essa relação de velocidade é determinada pelo Jacobiano desta função.

$$\dot{p}_e = J_P(q)\dot{q} \tag{2.23}$$

$$\dot{\omega}_e = J_O(q)\dot{q} \tag{2.24}$$

As equações 2.23 e 2.24 expressam a variação cartesiana $\dot{p}_e$ e variação angular $\dot{\omega}_e$ do orgão terminal como funções da variação angular $\dot{q}$, onde J_P e J_O são as JMs que relacionam a variação angular $\dot{q}$ com as variações descritas. A JM é uma das ferramentas de controle de movimentos mais utilizadas. Está presente em muitos aspectos da

manipulação de robôs, desde planejamento e execução de trajetórias à determinação de singularidades. É importante notar que J_P e J_O são funções da atual posição das juntas q. Ou seja, cada configuração diferente de q resulta em uma JM diferente.

Para o estudo de caso do braço planar, por exemplo, a JM é definida da forma da Equação 2.25:

$$J_P = \begin{bmatrix} \frac{dx}{d\theta 1} & \frac{dx}{d\theta 2} & \frac{dx}{d\theta 3} \\ \frac{dy}{d\theta 1} & \frac{dy}{d\theta 2} & \frac{dy}{d\theta 3} \end{bmatrix} \tag{2.25}$$

Tomando a derivada de 2.21 e 2.22, pode-se definir a JM J_P; vide Equação 2.26.

$$J_P = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \tag{2.26}$$

Sendo os coeficientes da Equação 2.27.

$$\begin{aligned} a_{00} = -a_2 sin(q_1 + q_2) - a_1 sin(q_1) - a_3 sin(q_1 + q_2, +q_3), \\ a_{01} = -a_2 sin(q_1 + q_2) - a_3 sin(q_1 + q_2 + q_3), \\ a_{02} = -a_3 sin(q_1 + q_2 + q_3), \\ a_{10} = a_2 cos(q_1 + q_2) + a_1 cos(q_1) + a_3 cos(q_1 + q_2, +q_3), \\ a_{11} = a_2 cos(q_1 + q_2) + a_3 cos(q_1 + q_2 + q_3), \\ a_{12} = a_3 cos(q_1 + q_2 + q_3) \end{aligned} \tag{2.27}$$

As equações descritas são referentes ao problema da FK, onde a posição das juntas é utilizada para calcular a posição do orgão terminal. Para resolver o problema da IK com a abordagem do jacobiano, a JM precisa ser invertida.

$$\dot{q} = J_P(q)^{-1} \dot{p}_e \tag{2.28}$$

$$\dot{q} = J_O(q)^{-1} \dot{\omega}_e \tag{2.29}$$

Ou seja, as equações 2.28 e 2.29 descrevem como usar a JM para determinar a variação angular $\dot{q}$ dada uma variação cartesiana do orgão terminal.

Como mencionado anteriormente, a JM descreve o comportamento do sistema somente próximo à configuração atual das juntas. Variações muito grandes para o orgão terminal geram imprecisão. Para evitar imprecisão, é necessário utilizar pequenas variações. Como consequência, o processo de utilizar a JM inversa precisa ser repetido para cada pequeno passo.

A outra complicação é que a inversão de matriz pode ser computacionalmente custosa e matrizes não quadradas não são invertíveis. As JMs raramente são quadráticas,

o que ocorre somente quando o número de juntas e coordenadas são o mesmo. Por exemplo, para a JM de uma variação cartesiana de um orgão terminal em um espaço tridimensional necessitaria que o robô tivesse seis juntas, se for levado em consideração a posição e a orientação.

Dois dos estudos de caso do presente trabalho possuem JMs não quadráticas e, portanto, não inversível. Para o braço planar de 3 juntas a matriz é 2X3 e para o braço do robô Thormang3, 6X7. Matrizes não inversíveis podem ser resolvidas através de aproximações, apesar de introduzirem ainda mais imprecisão aos resultados.

O método utilizado neste trabalho para calcular a JM pseudo-inversa é o método Monroe-Penrose. Como discutido in MacAusland (2014), esse método generaliza a forma de encontrar uma matriz inversa, permitindo que uma matriz inversa aproximada A^+ seja encontrada para qualquer matriz. A matriz aproximada A^+ pode ser vista na Equação 2.30.

$$A^+ = [R^{-1}\sigma]Q \tag{2.30}$$

Onde Q é um matriz unitária, R é uma matriz triangular superior e σ é uma matriz de zeros. Seja o exemplo da Equação 2.31.

$$SejaA = \begin{bmatrix} 1 & -1 & 4 \\ 1 & 4 & -2 \\ 1 & 4 & 2 \\ 1 & -1 & 0 \end{bmatrix} \tag{2.31}$$

As matrizes decomposição são Q e R, como pode ser observados nas Matrizes 2.32.

$$Q = \begin{bmatrix} 1/2 & -1/2 & 1/2 \\ 1/2 & 1/2 & -1/2 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & -1/2 & 1/2 \end{bmatrix} R = \begin{bmatrix} 2 & 3 & 2 \\ 0 & 5 & -2 \\ 0 & 0 & 4 \end{bmatrix} \tag{2.32}$$

O próximo passo é inverter R, resultando na Matriz 2.33.

$$R^{-1} = \begin{bmatrix} 1/2 & -3/10 & -2/5 \\ 0 & 1/5 & 1/10 \\ 0 & 0 & 1/4 \end{bmatrix} \tag{2.33}$$

A matriz aproximada A^+ para esse caso pode ser definida então, vide Equação 2.34.

$$A^+ = \begin{bmatrix} 1/2 & -3/10 & -2/5 & 0 \\ 0 & 1/5 & 1/10 & 0 \\ 0 & 0 & 1/4 & 0 \end{bmatrix} \begin{bmatrix} 1/2 & -1/2 & 1/2 \\ 1/2 & 1/2 & -1/2 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & -1/2 & 1/2 \end{bmatrix} = \begin{bmatrix} 1/5 & 3/10 & -1/10 & 3/5 \\ -1/20 & 1/20 & 3/20 & -3/20 \\ 1/8 & -1/8 & 1/8 & -1/8 \end{bmatrix} \tag{2.34}$$

Se o processo para teste for realizado, o resultado pode ser observado na Equação 2.35.

$$A^+A = \begin{bmatrix} 1/5 & 3/10 & -1/10 & 3/5 \\ -1/20 & 1/20 & 3/20 & -3/20 \\ 1/8 & -1/8 & 1/8 & -1/8 \end{bmatrix} \begin{bmatrix} 1 & -1 & 4 \\ 1 & 4 & -2 \\ 1 & 4 & 2 \\ 1 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.35}$$

Esse método está implementado por bibliotecas de muitas linguagens de programação, incluindo Python, e foi, portanto, utilizado dessa forma para o cálculo da JM inversa quando não quadrada.

2.2 ROS - ROBOTICS OPERATING SYSTEM

Escrever um *software* para robôs pode ser uma tarefa árdua, principalmente considerando que o escopo que envolve robótica é grande e continua crescendo. O reuso de código entre robôs não é uma tarefa trivial devido à grande variação de *hardware* existente entre eles. Além disso, a semelhança entre os códigos geralmente é muito pequena. Um mesmo *software* pode ser responsável por somente uma aplicação ou por uma série de aplicações de variados escopos, da camada de dispositivos até controle e inteligência. Ou seja, as arquiteturas de *softwares* para robôs precisam dar suporte a integração de uma vasta escala heterogênea de aplicações.

Para isso, muitos *frameworks* para o desenvolvimento de *software* foram desenvolvidos na área da robótica. Apesar de buscarem gerenciar a complexidade e facilitar a prototipagem, esses *frameworks*, de modo geral, foram desenvolvidos para propósitos específicos, talvez devido a ineficiência de outros *frameworks*, ou para enfatizar aspectos mais relevantes no processo de desenvolvimento que um determinado robô pode demandar. Partindo dessa situação que o ROS (QUIGLEY et al., 2009) foi projetado. ROS é

um *framework* que busca uma integração em larga escala e tem como algumas de suas filosofias ser modular, aberto, gratuito e multilingue.

ROS não é um sistema operacional tradicional se analisado do ponto de vista de gerência de projetos e escalonamento. Ele proporciona uma camada de comunicação estruturada sobre outro sistema operacional, sendo categorizado, portanto, como um pseudo-sistema operacional. Um sistema baseado em ROS consiste, essencialmente, de um número de processos conectados através de uma topologia ponto a ponto (`peer-to-peer`), que é uma arquitetura de redes de computadores onde cada um dos pontos ou nós da rede funciona tanto como cliente quanto como servidor, permitindo compartilhamentos de serviços e dados sem a necessidade de um servidor central. Essa característica de não possuir um servidor de dados central evita problemas de conexão, desde que a maioria dos robôs é composto por muitos computadores e microcontroladores que realizam tarefas homogêneas e que precisam compartilhar dados via rede.

Entretanto, para tornar a topologia *peer-to-peer* válida é necessário que exista um mecânismo que permita que os processos possam se achar em tempo real. Esse trabalho é realizado pelo nó principal, do inglês *ROS Master*. Todas as aplicações do sistema são associadas ao nó principal e este precisa estar ativo para que o sistema funcione.

ROS possui suporte a muitas línguas de programação, sendo C++ e Python as mais comumente utilizadas. Possui um micronúcleo com muitas ferramentas para compilar e rodar componentes e reúsa código de muitos projetos abertos, como os de visão computacional de OpenCV (BRADSKI; KAEHLER, 2000), planejamento de OpenRAVE (DIANKOV; KUFFNER, 2008), navegação de Player and Stage Project (GERKEY; VAUGHAN; HOWARD, 2003), (GERKEY et al., 2001), (VAUGHAN; GERKEY; HOWARD, 2003). É distribuído sobre os termos da licença BSD, o que permite o desenvolvimento de projetos comerciais e não comerciais. Os conceitos fundamentais de ROS serão discutidos na sequência.

2.2.1 Conceitos Fundamentais

Os conceitos fundamentais da implementação do ROS são nós, mensagens, tópicos e serviços.

Nós são os processos que realizam a computação. Um nó pode ser interpretado como o módulo de um *software* que realiza o execução de um determinado processo. Um robô por exemplo, pode ter um nó que realiza a visão computacional e outro responsável por controlar os servo-motores do robô. Um sistema robótico, entretanto, tipicamente é distribuído em vários nós.

Para o exemplo citado acima com somente dois nós, é possível que o robô realize movimento e evitar colisão. Para isso é necessário que o nó de visão passe informações

para o nó de controle dos servo-motores. Essa comunicação é feita através mensagens. Uma mensagem é geralmente um número inteiro ou flutuante, uma *String*, um valor booleano, uma estrutura, etc. Vetores também são suportados e uma mensagem pode ser decomposta em outras mensagens.

Para que o nó de controle dos servo-motores receba a mensagem do nó de visão é necessário que o nó de visão publique em um tópico esse dado e que o nó de controle esteja inscrito nesse tópico. Um nó que está interessado em um determinado dado se inscreverá em um tópico apropriado, que forneça esse dado. O nó de visão pode, por exemplo, publicar em um tópico uma matriz de inteiros representando um *frame* atual da imagem desde que esse tópico aceite como mensagem uma matriz de inteiros. Da mesma forma o nó de controle pode se inscrever nesse tópico e receber uma matriz quando outro nó publicar nele. Podem existir vários nós inscritos ou subscritos em um mesmo tópico, e um nó pode se inscrever e/ou publicar em vários tópicos.

É conveniente renderizar a comunicação através de um grafo. Para o exemplo descrito, essa representação pode ser vista na Figura 2.3. Apesar desse exemplo simples, os grafos são geralmente muito complexos devido ao número de nós do sistema.

Essa forma de comunicação, entretanto, não é apropriada para trocas de mensagens que precisam sincronização. Para resolver esse problema existem serviços. A ideia de serviços no ROS é analoga a ideia de *web*, onde a comunicação é uma par de mensagens, sendo uma de requisição e outra de resposta. Diferentemente dos tópicos, um serviços pode ser utilizado somente por um nó ao mesmo tempo e cada serviço tem um nome único.

2.2.2 Desenvolvimento Colaborativo e Composição de Funcionalidade

Em Quigley et al. (2009) está detalhado as vantagens que ROS oferece, tais como a possibilidade de debugação e desenvolvimento paralelo a execução, monitoramento, transformações, etc. As duas principais vantagens são o desenvolvimento colaborativo e a composição de funcionalidade.

Devido ao vasto escopo da robótica e da inteligência artificial, colaboração entre pesquisadores é fundamental para o desenvolvimento de sistemas de grande complexidade. Para dar suporte ao desenvolvimento colaborativo, o sistema de *software* do ROS é organizado em pacotes, do inglês *packages*. Um pacote ROS é um simples diretório com um arquivo em linguagem XML (BRAY et al., 1997) listando as suas dependências. O sistema baseado em pacotes permite que as aplicações possam ser particionadas pequenos *softwares*, que podem ser desenvolvidos e mantidos pelo próprio time de desenvolvedores. Tipicamente, um pacote ROS possui dependências e códigos fonte em linguagem C++ e *scripts* em linguagem Python, além da descrição XML. Um pacote ROS é compilado utili-

Figura 2.3 – Exemplo de comunicação entre dois nós no ROS.

zando *catkin* (JOSEPH, 2018), que é uma coleção de macros *CMake* e associados códigos desenvolvidos para ROS.

Com ROS é possível inicializar um agrupamento de nós (*cluster*), uma vez que este é descrito em um arquivo XML. Entretanto, muitas vezes é necessário instaciar várias vezes o mesmo *cluster*. ROS possibilita que isso seja feito utilizando o comando *roslaunch*, desde que a inexistência de colisões de nomes seja assegurada. Ou seja, com *roslaunch* é possível instânciar um conjunto de dados (aplicações, modelos, etc) múltiplas vezes desde que não exista conflito de nomes.

Essas duas características são a base do desenvolvimento desse trabalho. A primeira vantagem foi explorada no desenvolvimento de um pacote ROS, que é explorada com mais detalhes no Capítulo 3 assim como a composição de funcionalidade, que também é abordado com mais detalhes ainda nesta neste Capítulo 2.

2.2.3 Simulador Gazebo

Simuladores têm sido uma peça fundamental em pesquisas sobre robótica como ferramentas para testes rápidos e eficientes de novos conceitos, estratégias e algorítimos. O simulador de robótica Gazebo (KOENIG; HOWARD, 2004) foi desenvolvido idealizando a criação de um ambiente tridimensional dinâmico que possibilite a interação de múltiplos robôs e que seja capaz de recriar a complexidade do mundo que envolve a robótica.

Para criar esse ambiente realístico, todos os objetos simulados no Gazebo possuem massa, velocidade, fricção, e inúmeras outras propriedades físicas. Ele permite a junção de múltiplos objetos rígidos conectados via juntas, modelando uma estrutura dinâmica, onde forças lineares e angulares podem ser aplicadas para gerar locomoção e interação com o ambiente. A soma de todas as estruturas dinâmicas cria o mundo no simulador. A maioria das propriedades físicas são controláveis, das condições de luz a inércia de objetos.

Gazebo segue os princípios estabelecidos pelo Player and Stage Projects (GERKEY; VAUGHAN; HOWARD, 2003) sendo parte integrante do projeto e estando, portanto,

presente no ROS. É uma ferramenta escalável, eficiente e simples que tem sido responsável por abrir o campo da robótica para uma comunidade muito grande.

2.2.3.1 Arquitetura e Ambiente

Como pode ser observado na Figura 2.4, o mundo no Gazebo representa o conjunto de todos os modelos e propriedades do ambiente tais como gravidade e luminosidade, onde cada modelo é composto por ao menos um corpo e um número de juntas e sensores. Para que o modelo e essas propriedades existam, um conjunto de bibliotecas independentes realizam uma interface para o funcionamento do simulador no nível mais baixo, prevenindo a dependência de ferramentas específicas que possam sofrer alterações e permitindo a possibilidade do emprego de melhores bibliotecas caso surjam. Por último, comandos do cliente são recebidos através de uma interface com memória compartilhada.

Gazebo utiliza, por exemplo, a biblioteca *Open Dynamics Engine* (SMITH, 2001) como uma interface para simular a dinâmica e a cinemática associada a corpos rígidos articulados. Esta *engine* possui várias características tais como detecção de colisão, funções de massa e rotação, etc; é uma das mais utilizadas para a simulação física dentre as de código aberto.

A interface para o usuário é outro exemplo. Essa interface precisa ser sofisticada e rápida ao mesmo tempo que não pode prejudicar o processamento da capacidade de simular a dinâmica do simulador, que é a essência do Gazebo e demanda uma carga de processamento significante. Buscando amenizar ao máximo essa condição, o simulador utiliza *OpenGL Utility Toolkit* (SHREINER; GROUP et al., 2009). OpenGL é uma biblioteca padrão para a criação de aplicações interativas em duas e três dimensões, sendo estável, escalável e permitindo a execução em GPUs. O conjunto de ferramentas de utilidades (*OpenGL Utility Toolkit*) fornece mecanismos para interação do usuário com o Gazebo através de dispositivos de entrada, tais como teclado.

Um modelo é um objeto que possui uma representação física composta por pelo menos um corpo rígido, sensores, juntas e interfaces para facilitar o fluxo de dados. Um corpo é a estrutura física mais básica, como um cubo, cilindro, esfera, etc; que possui propriedades físicas que são conectados por juntas, que dessa forma, permitem a formação de relações cinemáticas e dinâmicas.

Uma junta pode ser, por exemplo, de revolução (engrenagem) de uma ou duas dimensões ou prismática, que é o deslocamento em uma direção do DoF. Além de conectar dois corpos, uma junta também pode atuar como motor quando um movimento é gerado pela fricção entre os corpos resultante da força aplicada na junta. Cuidados especiais precisam ser tomados quando um modelo possui muitas juntas para evitar que a simulação perca estabilidade quando sobre a atuação de parâmetros incorretos, como é descrito no

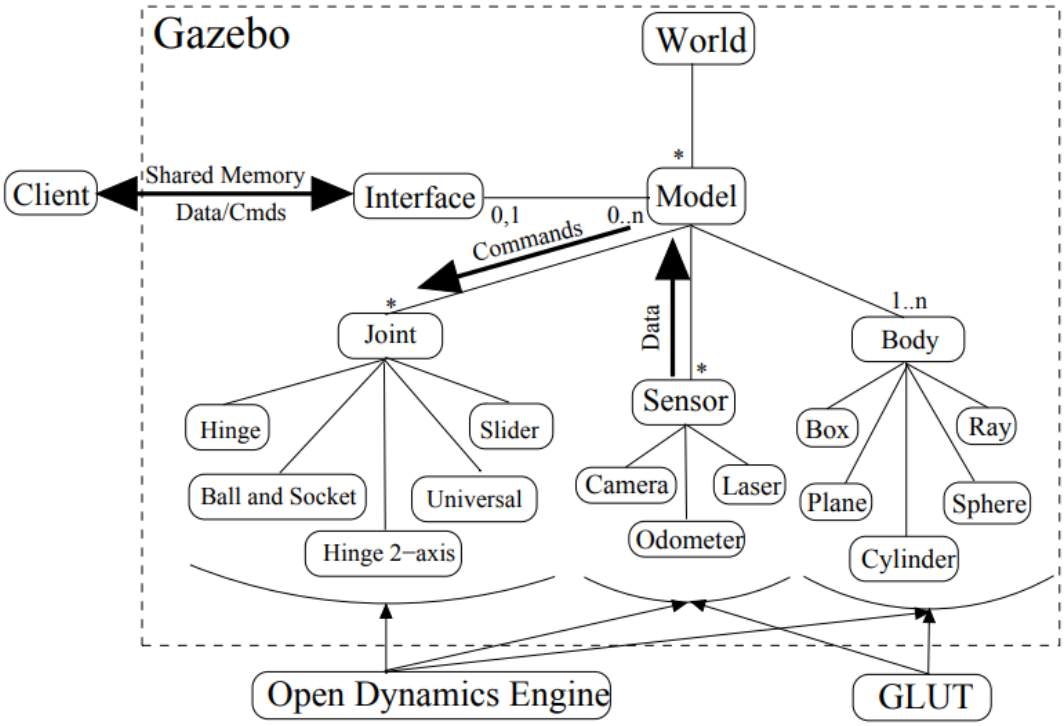


Figura 2.4 – Arquitetura e Organização do Gazebo.

Fonte: Koenig e Howard (2004).

Capítulo 3 dos estudos de caso do braço tridimensional e do Thormang3.

Para executar tarefas úteis um robô precisa de sensores. No Gazebo um sensor é um dispositivo abstrato, sem representação física, que ganha um encorpamento somente quando descrito em um modelo. Ou seja, um sensor pode ser reusado por qualquer modelo. Por exemplo, um robô pode ter inúmeras câmeras desde que essas câmeras estejam descritas em modelos que compõem o modelo final deste robô.

A implementação de um modelo no Gazebo é feita utilizando a linguagem XML. Um modelo é implementado utilizando dois arquivos, sendo um de configuração (*modelo.config*) e outro com a descrição (*modelo.sdf*). O arquivo de configuração possui informações sobre o modelo tais como nome (*<name>*), versão (*<version>*) e autor (*<author>*) enquanto que no arquivo de descrição estão presentes características do modelo tais como os corpos ((*<link>*)) que compõem esse modelo com suas propriedades físicas: visual (*<visual>*), (*<collision>*), massa (*<mass>*), etc. Dois corpos (*<links>*) são conectados utilizando uma junta (*<joint>*) onde é definido um hierarquia entre esse dois corpos, pai (*<parent>*) e filho (*<child>*). Para robôs que possuem uma descrição muito grande, como é o caso do Thormang3, se utiliza XML Macro (Xacro) (MACROXML, 2018) para organizar a descrição em arquivos XML menores e mais legíveis.

Um exemplo de um robô descrito no Gazebo pode ser visualizado na Figura 2.5 onde o robô Pioneer2AT é criado através da junção de quatro cilindros e um chassis com

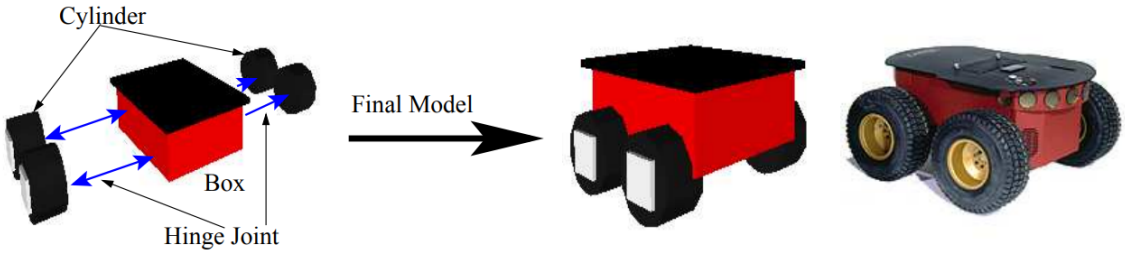


Figura 2.5 – Comparação dp robô Pionner2AT descrito no Gazebo e real

Fonte: Koenig e Howard (2004).

quatro juntas de revolução.

Resumidamente, um modelo de robô no Gazebo é simplesmente uma junção entre vários corpos. Entretanto, robô com um elevado número de juntas e segmentos pode possuir muitos nós e tópicos. Além disso, o mundo aonde esse robô será instânciado, do inglês *spawn*, pode necessitar possuir outras instâncias do mesmo modelo ou de modelos afins. Para efetivamente colocar o mundo de um robô no ambiente do Gazebo é utilizado o comando *roslaunch*. Com *roslaunch*, um conjunto de nós e modelos são instânciados ao menos uma vez, inicializando definitivamente o ambiente.

2.3 THORMANG3

Thormang3 (THORMANG3, 2018) é um robô humanoide que possui alta capacidade de processamento e de carga, sensoriamento sofisticado e habilidade de movimento que proporcionam uma plataforma extensa para pesquisas e atividades de educação.

Como pode ser observado na Figura 2.6, ele possui 1,375 m de altura, 0,22 m de profundidade, 0,569 m de largura e 1,645 m de envergadura; pesando 42 quilogramas no total. Possui 29 atuadores controlados por dois computadores e vários sensores, além de baterias e um roteador *wireless*.

Cada braço possui sete DoFs e nove atuadores, com os dois da garra. O ombro (*shoulder*) possui dois graus de inclinação (*arm_shp1*, *arm_sh_p2*) e um grau de rolamento (*arm_sh_r*), o cotovelo possui giro no eixo vertical (*arm_el_y*) e o pulso possui giro nos três eixos (*arm_wr_r*, *arm_wr_y*, *arm_wr_p*). Na Tabela 2.1 podem ser analisados os alcances em graus de cada atuador do braço.

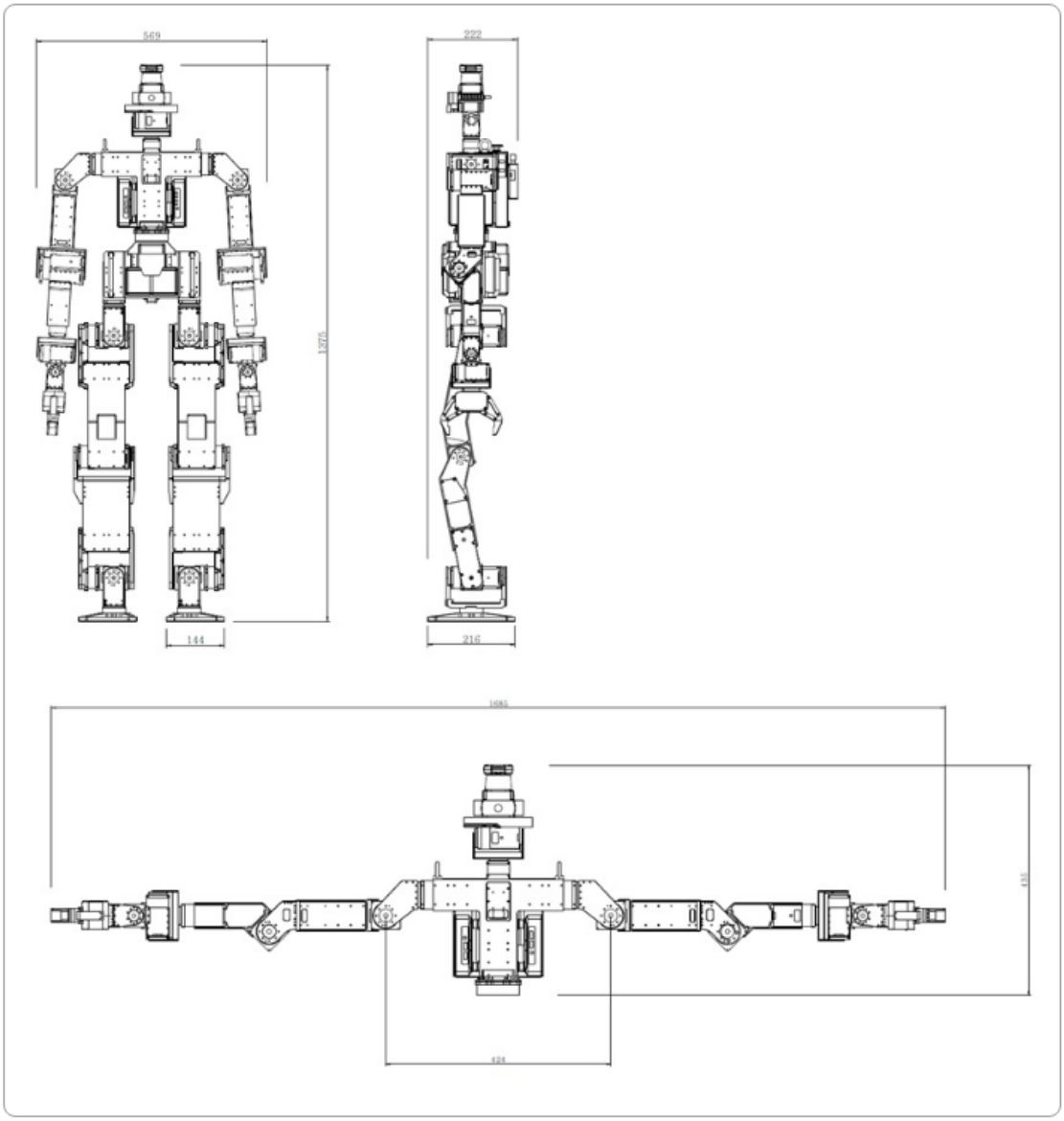


Figura 2.6 – Dimensões do Thormang3.

Fonte: Thormang3 (2018).

Junta	Ângulo mínimo	Ângulo máximo
Shoulder Pitch 1	$-90°$	$90°$
Shoulder Roll	$-90°$	$90°$
Shoulder Pitch 2	$90°$	$90°$
Elbow Yaw	$-72°$	$72°$
Wrist Roll	$-162°$	$162°$
Wrist Yaw	$-81°$	$81°$
Wrist Pitch	$-81°$	$81°$
Gripper 1	$0°$	$65°$
Gripper 2	$0°$	$65°$

Tabela 2.1 – Alcance de ângulos para cada atuador do braço

2.3.1 Pacotes ROS e Simulação no Gazebo

Em termos de *software*, Thormang3 utiliza o sistema operacional Ubuntu LTS 16.04 64-bit e é compatível com versão Kinetic do ROS. A parte de *software* é feita via ROS e todas as funcionalidades que o robô possui podem ser utilizadas através de comandos de ROS, ou em código com ROScpp[2] e ROSpy[3]. Desse modo, Thormang3 possui vários pacotes, sendo utilizados para esse trabalho os pacotes da simulação no Gazebo e o de movimentos, juntamente com suas dependências.

O pacote de movimentos do Thormang3[4] possui, dentre vários módulos, o módulo de cinemática. Nesse módulo está descrita toda a cinemática do Thormang3, incluindo os dois pés e braços, tronco e cabeça; bem como funções para utilizá-la. A composição do robô é feita por um vetor de estruturas onde cada posição é uma estrutura que possui todas as características de uma junta (ângulo máximo e mínimo, massa, inércia, eixo da junta e centro de massa), e sua posição em relação as demais juntas parentes (parentesco e posição relativa). Por exemplo, a composição da junta *left_arm_wrist_yaw* pode ser visualizada na Figura 2.7.

Neste caso, a junta *left_arm_wrist_yaw* é a junta 12 do vetor, possuindo a junta 10 como junta pai e a junta 14 como filho. Possui um alcance de $-81°$ à $81°$ e massa de 768 gramas, girando no sentido anti-horário do eixo Z; além do centro de massa, inércia e posição relativa.

Essas características permitem que a FK, IK, JM, etc, possam ser calculadas através de funções. As funções de cinemática utilizadas neste trabalho estão listadas abaixo:

- $std :: vector < int > KinematicsDynamics :: findRoute(intfrom, intto)$
- $voidKinematicsDynamics :: calcForwardKinematics(intjoint_id)$
- $Eigen :: MatrixXdKinematicsDynamics :: calcJacobian(std :: vector < int >)$

[2] ROScpp é a implementação de ROS em C++
[3] ROSpy é a implementação de ROS em Python
[4] https://github.com/ROBOTIS-GIT/ROBOTIS-THORMANG-MPC

```
// left arm wrist yaw
thormang3_link_data_[12]->name_              =  "l_arm_wr_y";
thormang3_link_data_[12]->parent_            =  10;
thormang3_link_data_[12]->sibling_           =  -1;
thormang3_link_data_[12]->child_             =  14;
thormang3_link_data_[12]->mass_              =  0.768;
thormang3_link_data_[12]->relative_position_ =  robotis_framework::getTransitionXYZ( 0.039 , 0.000 , 0.045 );
thormang3_link_data_[12]->joint_axis_        =  robotis_framework::getTransitionXYZ( 0.0 , 0.0 , -1.0 );
thormang3_link_data_[12]->center_of_mass_    =  robotis_framework::getTransitionXYZ( 0.023 , 0.001 , -0.046 );
thormang3_link_data_[12]->joint_limit_max_   =  0.45 * M_PI;
thormang3_link_data_[12]->joint_limit_min_   =  -0.45 * M_PI;
thormang3_link_data_[12]->inertia_           =  robotis_framework::getInertiaXYZ( 0.00059 , -0.00002 , -0.00002 , 0.00078
```

Figura 2.7 – Exemplo da descrição cinemática de uma junta em código.

A função *findRoute()* retorna uma lista de inteiros que representam as juntas entre as duas juntas passadas por parâmetro. Por exemplo, o ID da primeira junta do braço esquerdo é 2 (*l_arm_sh_p1*), enquanto que o ID do orgão terminal é 34. Logo, passando 2 e 34 como parâmetro, a função retorna o vetor de IDs de todo o braço esquerdo: (2, 4, 6, 8, 10, 12, 14, 34). Essa função é utilizada para facilitar a determinação da lista de juntas para o cálculo da JM.

A função *calcForwardKinematics()* realiza o cálculo da FK partindo de uma junta inicial. Por exemplo, utilizando a junta 2 como parâmetro, a FK do braço esquerdo é calculada e a posição e orientação do orgão terminal definida. Em termos práticos, o cálculo da FK é feito de forma recursiva utilizando as funções de matemática da Robotis, fabricante do Thormang3.

A última função utilizada é a função *calcJacobian()*. Como o próprio nome sugere, essa função calcula a JM de uma lista de juntas em baseado na posição atual delas. Ela retorna uma matriz de 6XN, sendo N o número de juntas da lista passada por argumento. Ou seja, para o braço a função retorna uma matriz 6X7. Essa função precisa ser utilizada após o cálculo da FK, uma vez que a JM utiliza a posição e orientação cartesiana atual. Assim com o função de FK, essa função utiliza funções de matemática da Robotis.

O outro pacote utilizado [5] permite utilizar o Thormang3 no ambiente de simulação do Gazebo. Esse pacote possui a descrição do robô em XML e os códigos para controle na interface com ROS [6]. Na inicialização da simulação, por definição e neste trabalho, é utilizado o comando:

roslaunch thormang3_gazebo robotis_world.launch

O arquivo *robotis_world.launch*, quando executado, inicia os nós, tópicos e serviços para o controle do Thormang3 e instancia o mundo do gazebo, que contém o robô e possíveis outros objetos.

Após a inicialização, é possível, por exemplo, definir valores para uma junta qualquer publicando um número em ponto flutuante em um respectivo tópico dessa junta. Por

[5] https://github.com/ROBOTIS-GIT/ROBOTIS-THORMANG-Common

[6] Essa interface com ROS é a mesma utilizada para controle do Thormang3 real, sendo necessário ativar o modo de simulação nos seus arquivos de controle desse pacote para que a simulação funcione

exemplo, para se definir um ângulo de 180 graus para a junta do braço direito de rolamento no pulso, usa-se o comando:

rostopic pub -1 */thormang3/r_arm_wr_r_position/command std_msgs/Float64 "data:* 3.1415*"*

O mesmo pode ser feito utilizando um *publisher* com ROSpy ou ROScpp, como será feito neste trabalho.

De modo geral, esses conceitos são suficientes para compreender a metodologia do estudo de caso do Thormang3. Como está descrito nas seções mais à frente, neste trabalho é desenvolvido um pacote que utiliza o módulo de cinemática do Thormang3 e suas funções, onde os valores calculados são publicados em tópicos nas juntas do Thormang3.

2.4 REDES NEURAIS ARTIFICIAIS

O cérebro humano é composto por neurônios que recebem pulsos elétricos de outros neurônios como uma entrada, gerando uma excitação ou inibição em sua saída que são utilizados como entrada por outros neurônios. Quando a excitação de um neurônio atinge um certo nível, o neurônio atinge seu potencial de ação, do inglês (*firing*), e o processo é repetido. A eficácia de um neurônio em excitar outro neurônio não é constante e melhora conforme a experiência que eles compartilham. Desse modo, a habilidade de aprender é gerada por uma mudança na força das conexões entre os neurônios. Em uma rede de neurônios, a saída também é calculada como sendo uma função da entrada e a relação entre a entrada e a saída da rede também pode ser alterada por experiência. Portanto é a conexão entre os neurônios que determina o comportamento da rede e como esse comportamento varia. Essa é a ideia básica de como o aprendizado funciona em sistemas biológicos, como descrito em Hebb (2002), e é baseado nela que as redes neurais artificiais, do inglês *artificial neural networks*(ANNs), foram desenvolvidas.

Uma ANN utiliza técnicas baseadas em estatística para aprender com um conjunto de dado, sem ser programado explicitamente; ou seja, é uma abordagem de aprendizado de máquina (NASRABADI, 2007). Desde o desenvolvimento dos primeiros modelos matemáticos de neurônios conectados em uma rede para o cálculo booleano de funções lógicas básicas (MCCULLOCH; PITTS, 1943) até as atuais e complexas ANNs de aprendizado profundo (LECUN; BENGIO; HINTON, 2015) muitos passos foram necessários, sendo o desenvolvimento das ANNs do tipo perceptron e do algoritmo de aprendizado *backpropagation* alguns dos mais importantes. Tais passos serão discutidos com mais detalhes na sequência.

2.4.1 Perceptron

O modelo de ANN *perceptron* (ROSENBLATT, 1958) foi baseado na ideia de que o cérebro é um associador de aprendizado que computa classificações em resposta a estimulos. Com isso, a ideia de cognição em uma ANN *perceptrion* é baseado em separabilidade estatística ao invés de uma função de lógica simbólica, que era a ideia de uma ANN até então. Um *perceptron* é um modelo simplificado de um neurônio biológico, produzindo um comportamento semelhante apesar da complexidade de um neurônio biológico. A representação gráfica de uma célula pode ser vista na Figura 2.8.

Uma célula perceptron C mapeia suas entradas x em um valor único de saída a. Para isso cada entrada x_i de C é multiplicada por um peso w_i de C, representando a força de conexão entre duas células ou, comparativamente, a influência potencial de ação da saída de um neurônio biológico conectado a outro neurônio. Para que o modelo convirja, também é somado um viés b_i, do inglês (*bias*), aleatório para deslocar no sentido vertical a função que representa e evitar que função fique estagnada na origem; ou seja, evitar que C assuma o valor zero. O valor numérico que a célula C assume é a soma de todos os pesos w de C; em termos biológicos, é o quão forte é a conexão de uma célula com todos os neurônios que enviam sinais elétricos a ela. Por último, uma função de ativação, que será discutido com mais detalhes mais à frente, é aplicada sobre essa soma para efetivamente mapear essa soma; o que, comparativamente, é uma representação abstrata da taxa de ação de potencial de um neurônio biológico.

Uma ANN perceptron, em sua forma mais simples como pode ser visto na Figura 2.9, possui três camadas. Cada camada possui um número de neurônios, onde o conjunto de neurônios da primeira camada representa os dados de entrada e o conjunto de neurônios da saída é a resposta da ANN. Todos os neurônios da camada de entrada são conectadas com os neurônios da camada intermediária, que por sua vez são conectados aos neurônios da camada de saída da mesma forma.

Além de representar a resposta da ANN, o conjunto de neurônios da saída também inibem uns aos outros e aos neurônios de associação, dos quais eles não recebem entrada. Dessa forma, quando um padrão de entradas é apresentado à ANN, vários neurônios de associação são ativados, e estes, por sua vez, ativam alguns neurônios de saída. Essa capacidade de generalizar e aprender de uma ANN perceptron é mostrada com mais detalhes e comprovada em Rosenblatt (1958).

2.4.2 Função de ativação

Como descrito anteriormente, uma função de ativação é uma abstração que representa a taxa de ação de potencial de um neurônio biológico. Na forma mais simples, uma função de ativação é binária (função de Heaviside), representando se o neurônio está ativo

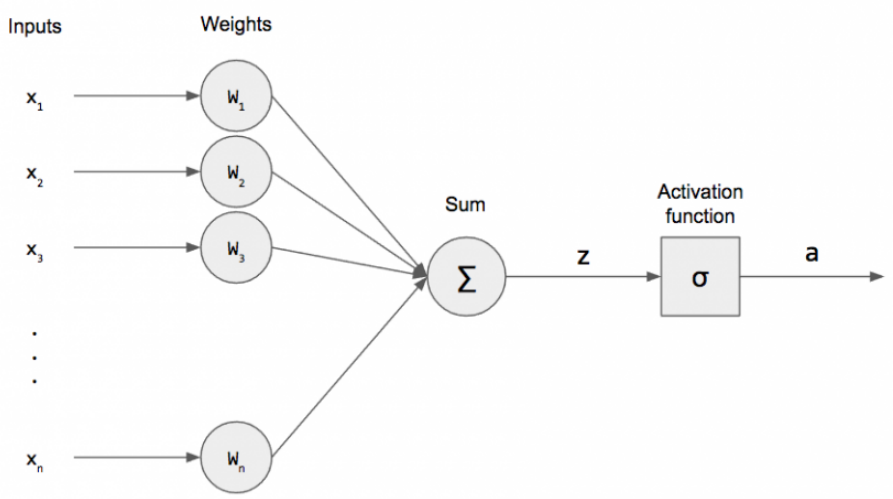


Figura 2.8 – Representação gŕafica de uma célula perceptron.

Fonte: Learning (2018).

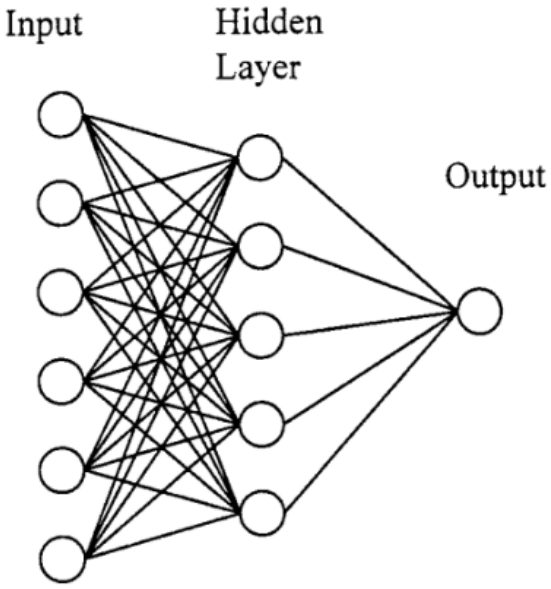


Figura 2.9 – Representação gŕafica de uma ANN perceptron múlti-camada de uma camada escondida.

Fonte: Rosenblatt (1958).

ou não. Entretanto, a simplicidade dessa função de ativação gera uma a necessidade do uso de muitos neurônios para que a ANN produza bons resultados.

Isso pode ser amenizado utilizando uma função de ativação *sigmoid* normalizável, onde o neurônio permanece em zero até a chegada de uma entrada que aumenta rápidamente no início e graduamente atinge um ápice. Esse comportamento é semelhante ao de um neurônio biológico. Entretanto essa função de ativação não é diferenciável, fator necessário para o cálculo do algoritmo de aprendizado *backpropagation* que será visto mais à frente.

Uma das melhores soluções é utilizar uma função de ativação sigmoidal na forma de tangente hiperbólica. Também conhecida como função logística, essa função de ativação é muito aplicada em ANNs por ser aplicável e em muitos outros campos da ciência. A função possui a forma apresentada na Equação 2.36, assumindo valores entre 0 e 1, e está mostrada na Figura 2.10.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.36}$$

Apesar de ser biológicamente semelhante, um neurônio que utiliza a função logística possui um desempenho inferior a outras funções de ativação mais biológicamente discrepante, tais como a função de ativação tangente hiperbólica; principalmente quando aplicada à ANNs com múltiplas camadas. Como está detalhadamente discutido em Glorot, Bordes e Bengio (2011), neurônios com a função de ativação ReLU, do inglês (*Rectified Linear Units*), possuem um desempenho similar ou superior ao de outras funções de ativação, além de representar um modelo de neurônio biológico ainda melhor. A função ReLU ou função rampa é a função de ativação mais popular atualmente em ANNs com múltiplas camadas, como é o caso da arquitetura de ANN usada neste trabalho. A função ReLU (2.37) possui o comportamento que pode ser visto na Figura 2.11.

$$f(x) = log(1 + exp(x)) \tag{2.37}$$

As duas funções de ativação abordadas acima foram utilizadas na arquitetura de ANN utilizada neste trabalho. Como está discutido com mais detalhes no Capítulo 3, a função ReLU foi utilizada nos neurônios das camadas de entrada e intemediárias, enquanto que a função logística foi utilizado na camada de saída.

2.4.3 Treinamento Supervisionado

Treinamento supervisionado em aprendizado de máquina é o procedimento que busca encontrar uma função que melhor mapeie uma entrada para uma saída baseado em pares entrada-saída de dados (RUSSELL; NORVIG, 2016). Supondo, por exemplo, que

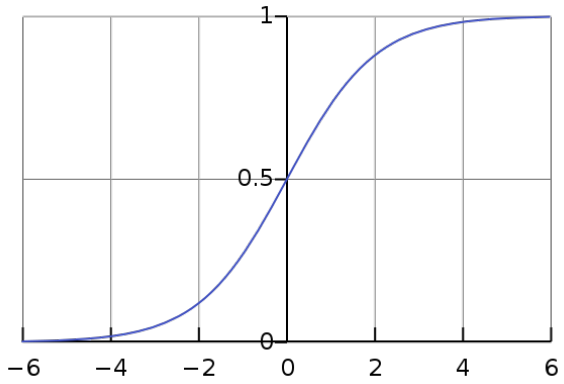


Figura 2.10 – Função de ativação sigmoid na forma de tangente hiperbólica.

Fonte: Wikipedia (2018).

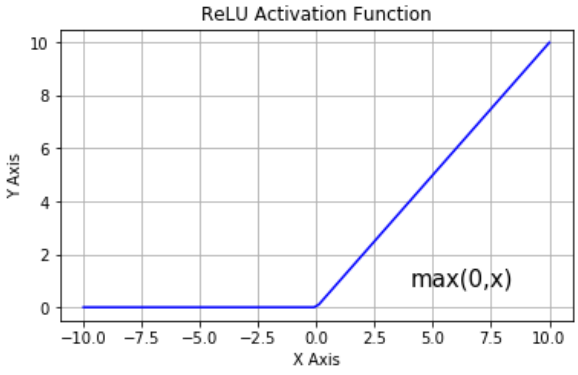


Figura 2.11 – Função de ativação ReLU.

Fonte: LearnOpenCV (2018).

uma concessionária de carros queira utilizar o histórico de vendas de carros para saber o valor estimado de um carro levando em consideração somente a sua idade; nesse caso o treinamento supervisionado irá utilizar o histórico de vendas para encontrar uma função que estime o valor do carro usando a idade dele como entrada.

Para problemas de regressão como esse, também é definido uma função de custo (*loss function*) que mede o erro (distância) entre o valor estimado pela ANN e o valor desejado; para o exemplo da concessionária seria a diferença do valor que foi estimado pela ANN e o valor real da venda de um dos carros do histórico de vendas. Então, os pesos de cada neurônio dessa ANN são otimizados para reduzir esse erro usando um algoritmo de aprendizado, tal como o algoritmo *backpropagation*.

Neste trabalho foi utilizado treinamento supervisionado aplicado a um problema de regressão. Para que os neurônios de saída da ANN sejam capazes de predizer a variação angular aproximada baseado na posição angular e na variação cartesiana, foi utilizado a função de custo erro médio quadrático (MSE) juntamente com o algoritmo de aprendizado *backpropagation* otimizado com o otimizador Adam.

2.4.3.1 Função de Custo Erro Médio Quadrático

A função de custo serve para avaliar o quão bem uma ANN modela o *dataset*. Se a predição da ANN estiver muito errada, a função de custo terá um valor elevado enquanto que um valor baixo será o resultado da função de custo de uma ANN bem treinada.

O custo para um algoritmo de aprendizado parte da suposição de que pode ser definido como sendo a média do custo de cada amostra, uma vez que a otimização do algoritmo de aprendizado é feita em cima de um exemplo de treino ou de um pequeno conjunto que precisa ser generalizado para todas as amostras. Além disso, também parte-se do princípio que a função de custo pode ser definida como uma função dos pesos da saída da ANN.

Dentre todas as funções de custo a mais utilizada é função MSE, por ser fácil de entender e implementar e possuir boa performance. Para calcular o MSE, basta encontrar a diferença entre a predição e a saída esperada, elevar ao quadrado e realizar uma média desse processo para cada amostra do conjunto de dado. Isso pode ser observado na Equação 2.38.

$$E = \frac{1}{n}\sum_{i=1}^{n}(Y_i - Y_i^{'})^2 \tag{2.38}$$

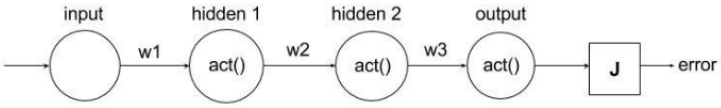


Figura 2.12 – Exemplo de ANN para *backpropagation*.

2.4.3.2 Backpropagation

O algoritmo de aprendizado *backpropagation* (WERBOS, 1990) é considerado um dos algoritmos mais importantes já desenvolvidos para ANNs. Ele é base para o treinamente das ANNs modernas, realizando papel determinante para que ANNs complexas possam ser treinadas.

O algoritmo consiste na aplicação do método da descida do gradiente à função de custo para ajustar os pesos das sinapses de forma a minimizar o custo. O algoritmo calcula as derivadas parciais do custo da saída em relação aos parâmetros, que são as sinapses e os vieses. Começa na saída, e volta propagando essa derivada para as camadas anteriores, de forma a ajustar os pesos das conexões na direção que minimiza o custo.

Na primeira parte do algoritmo, *forward pass*, as entradas são propagadas para a saída da ANN através de multiplicações de vetores e de ativações. Por exemplo, considerando a ANN da Figura 2.12 o saída do neurônio da primeira camada escondidada (*hidden1*) é definida na Equação 2.39.

$$out1 = act(w1 * input + b1 * input) \tag{2.39}$$

Sendo $w1$ o peso e $b1$ o viés do neurônio da primeira camada escondida e $act()$ a função de ativação da camada. O mesmo pode ser definido para a segunda camada escondida (*hidden2*) 2.40 e para a camada de saída (*output*) 2.41.

$$out2 = act(w2 * out1 + b2 * out1) \tag{2.40}$$

$$output = act(w3 * out2 + b3 * out2) \tag{2.41}$$

Para esse exemplo, o erro de saída utilizando a função de custo MSE pode ser visto na Equação 2.42.

$$loss = (output - desiredoutput)^2 \tag{2.42}$$

Na segunda parte do algoritmo, *backward pass*, os pesos são atualizados de forma a diminuir o erro entre o saída esperada e a obtida. Para realizar essa atualização dos pesos é necessário utilizar um algoritmo de otimização. Muitos otimizadores se baseiam no algoritmo de gradiente descendente, onde cada peso é atualizado por alguma redução

escalar negativa da derivada do erro em relação a este peso. Em outras palavras, a alteração do peso do neurônio é feita de acordo com a influência no erro do seu valor atual em relação a todos os pesos. Essa derivada parcial se baseia na regra da cadeia, que é uma fórmula para calcular a derivada da composição de duas ou mais funções. Ela está definida na Equação 2.43.

$$\frac{\partial E}{\partial w_{i,j}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{i,j}} \tag{2.43}$$

Onde E é o erro da camada de saída,$w_{i,j}$ o peso, o_j a saída da ANN e net_j as saídas das camadas entre i e j da regra da cadeia. Exemplificando, o cálculo da derivada para o peso $w1$ do exemplo 2.12 pode ser visualizado na Equação 2.44.

$$\frac{\partial error}{\partial w1} = \frac{\partial error}{\partial output}\frac{\partial output}{\partial out2}\frac{\partial out2}{\partial out1}\frac{\partial out1}{\partial w1} \tag{2.44}$$

Sabendo o quanto o peso $w1$ influencia no erro, é possível definir o novo valor para o peso usando um otimizador. Um otimizador é um método para atualização de pesos, sendo o mais simples o otimizador regra delta. Baseado nesse otimizador, o novo valor para o peso $w1$ pode ser definido, vide Equação 2.45.

$$w1 = w1 - lr\frac{\partial error}{\partial w1} \tag{2.45}$$

Sendo lr a taxa de aprendizagem (LR) do algoritmo de aprendizado, que é uma constante(geralmente pequena) para atualizar o peso de forma suave e lenta, evitando passos muito grande e caóticos.

Com a atualização de todos os pesos da ANN, o processo de *backpropagation* é finalizado. Como os pesos são atualizados em um pequeno passo de cada vez, é necessário aplicar o algorimo de aprendizado várias vezes (várias iterações). O número de iterações depende de inúmeros fatores: LR da ANN, números de camadas, otimizador, inicialização dos pesos e qualidade do *dataset*.

Dentre todos os otimizadores utilizado com *backpropagation*, o otimizador gradiente descendente estocástico (SGD) (BOTTOU, 2010) é bastante associado. SGD realiza uma aproximação estocástica da otimização do gradiente descendente. Em outras palavras, SGD utiliza o custo somente de uma amostra ou de um número específico de amostras (*batch*) a cada iteração ao invés do custo de todas as amostras do *dataset*. Matematicamente, o peso $w1$ da Equação 2.45 leva em consideração o custo E de somente de uma amostra ou de *batch* de amostras.

SGD é considerado o otimizador padrão para *backpropagation* e serviu como base para o desenvolvimento de outros otimizadores, inclusive do otimizador utilizado nesse trabalho.

2.4.3.3 Otimizador Adam

Adaptive Moment Estimation - Adam (KINGMA; BA, 2014) é um otimizador de funções estocásticas[7] de primeira ordem baseadas em gradiente. Adam é simples de implementar, computacionalmente eficiente e demanda pouca memória. Além disso, ele é muito eficiente para problemas que demandem muitos parâmetros e dados, como é o caso do presente trabalho.

Ele é essencialmente uma variação do otimizador SGD, se diferenciando basicamente no fato de usar um LR adaptativo ao invés de fixo. Com Adam um LR diferente é adaptado para cada peso da ANN através da estimação do primeiro e do segundo momento dos gradientes.

Foi desenvolvido buscando combinar as vantagens de outros dois otimizadores baseados em SGD: AdaGrad (DUCHI; HAZAN; SINGER, 2011) e RMSProp (TIELEMAN; HINTON, 2012). Adam mantém um LR adaptativo da mesma forma que o otimizador AdaGrad, tendo, dessa forma, boa performance com problemas de visão computacional, linguagem natural e outros problemas de gradiente esparso. Ao mesmo tempo que possui um LR adaptativo, Adam define o LR baseado na média das magnitudes recentes do gradiente do peso; ou seja, o quão rápido ele está mudando. Essa característica, incorporado de RMSProp, faz com que ele tenha boa performance em problemas não estacionários, como o problema do ruído por exemplo.

Complementarmente a ideia de RMSProp, Adam também faz o uso do segundo momento do gradiente (variância não centralizada). Especificamente, o algoritmo calcula uma média móvel exponencial do gradiente e do gradiente quadrático, tendo o decair, do inglês *decay*, controlado pelos parâmetros β_1 e β_2.

Matematicamente, os pesos com o otimizador Adam são ajustados de acordo com a Equação 2.46.

$$w_{t+1} = w_t - lr\frac{Mw}{\sqrt{Vw} + C} \tag{2.46}$$

Sendo t a iteração atual, w o peso e C uma pequena constante para evitar a divisão por zero. Mw Vw representam o primeiro e segundo momento, respectivamente, e são definidos pelas Equações 2.47 e 2.48.

$$Mw = \frac{\beta_1 Mw_t + (1-\beta_1)\nabla wL_t}{1-(\beta_1)_{t+1}} \tag{2.47}$$

$$Vw = \frac{\beta_2 Vw_t + (1-\beta_2)(\nabla wL_t)^2}{1-(\beta_2)_{t+1}} \tag{2.48}$$

Onde β_1 e β_2 são os parâmetros controláveis, ∇_w o gradiente e L_t a função de custo no instante atual.

[7] Funções estocásticas são funções determinadas por um conjunto de dados aleatórios

No otimizador Adam da ANN deste trabalho, foram usados valores padrão para os parâmetros β_1, β_2, C , enquanto que o parâmetro lr foi modificado, como pode ser visto no Capítulo 3.

2.4.3.4 Regularização Dropout

Dropout (SRIVASTAVA et al., 2014) é uma técnica de regularização onde alguns neurônios são ignorados de forma aleatória durante o treinamento. Com essa técnica um percentual aleatório de neurônios da rede não possui os pesos atualizados no *backpropagation* de cada época de treinamento. Essa característica faz com que uma quantidade de neurônios escolhidos aleatoriamente em cada época não "aprendam", tendendo a diminuir o *overfitting* que a ANN pode sofrer.

A técnica é de simples aplicação prática e é utilizada em todos os estudos de caso do trabalho com um percentual específico em cada estudo.

3 METODOLOGIA

A abordagem deste trabalho é substituir o uso da JM inversa por uma ANN para resolver o problema da IK de forma iterativa. O foco principal é treinar uma ANN que funcione bem em situações onde a inversa da JM não apresenta boa performance.

Neste capítulo é apresentado como as tecnologias da fundamentação teórica foram empregadas para alcançar esse objetivo. Como está mencionado nas seções anteriores, foram realizados três estudos de caso para buscar validar essa abordagem: um braço planar de três juntas de revolução, um braço tridimensional de três juntas de revolução e um braço tridimensional de sete juntas de revolução. Apesar de possuírem solução fechada conhecida, os dois primeiros estudos de caso foram desenvolvidos para clarificar a essência da abordagem e servir de referência para o desenvolvimento do último estudo de caso, que, além de não possuir solução fechada, melhor exemplifica uma braço robótico típico.

Nas primeiras subseções abaixo são apresentadas as características comuns para as três abordagens e, na sequência, as especificidades de cada estudo de caso.

3.1 ARQUITETURA DA ANN

Resumidamente, os algoritmos de IK, geralmente, recebem uma posição e orientação absoluta desejada para o orgão terminal e computam os ângulos das juntas que representam uma pose que coloca o orgão terminal naquela posição. No caso da solução usando DK, os ângulos de cada junta são aplicados nas equações de cada elemento da JM, gerando a JM da pose atual. Após ser invertida, a JM é multiplicada por uma variação cartesiana, gerando a variação angular. A variação de cada junta é, então, somada com o ângulo atual da junta. A FK é realizada para encontrar a posição e orientação cartesiana do orgão terminal dada a pose mais incremento. Se a posição e orientação calculada for a posição e orientação desejada, dentro de uma margem de erro, o processo é finalizado. Se não o processo é repetido até que a posição desejada para o orgão terminal seja alcançada, dentro da margem de erro.

Dessa forma, a arquitetura da ANN possui como entrada o conjunto de valores dos ângulos que representam a pose na qual o robô está no momento e o conjunto de valores da variação cartesiana do orgão terminal desejada, enquanto que a saída é o conjunto de variação das juntas.

Como pode ser observado na Figura 3.1, o primeiro neurônio de entrada da ANN(ou o mais acima) recebe o primeiro angulo do braço, enquanto que o último neurônio(ou mais abaixo) recebe a última variação do orgão terminal; que é o delta da posição em y para o

braço planar, delta da posição em z para o braço tridimensional ou delta da orientação em z para o estudo do Thormang3. Na saída, o primeiro neurônio(ou mais acima) representa a a variação cartesiana da primeira junta e o último(ou mais abaixo), a variação cartesiana da última junta. Para cada estudo de caso o número de entradas e saídas varia, mas segue essa estrutura.

Além disso, a arquitetura da ANN, para ambos estudos, é do tipo *perceptron* com *backpropagation* de múltiplas camadas, com o número de camadas variando para cada experimento. A função de custo utilizada é a função erro quadrático médio, uma vez que, como busca-se o ângulo de uma junta que um valor específico absoluto, é um problema de regressão.

O otimizador utilizado na arquitetura da ANN para todos os casos foi o otimizador Adam. Em todos os casos ele foi definido com os seguintes parâmetros:

- lr = 0,001
- β_1 = 0,9
- β_2 = 0,999
- epsilon = None
- decay = 0
- amsgrad = False

A taxa de aprendizado decrementa durante o treinamento é modificada a cada época, seguindo a seguinte ordem de descendência da Equação 3.1.

$$lr = lr - \frac{lr}{1000} \tag{3.1}$$

Ou seja, a cada época de treinamento o otimizador utilizado é redefinido, mudando somente a taxa de aprendizado.

As funções de ativação utilizadas foram as funções *sigmoid* na camada de saída e ReLU nas camadas escondidas. Uma melhor percepção da arquitetura da ANN utilizada pode ser vista na Figura 3.1.

O número de parâmetros, camadas variou para cada estudo. As combinações utilizadas foram encontradas após o treinamento e teste de vários modelos, desde pequenos(poucos camdas e parâmetros) à grandes(várias camadas e parâmetros). Para essa verificação possível, foi utilizado o Google Colaboratory[1] como ambiente de desenvolvimento.

[1] https://colab.research.google.com/

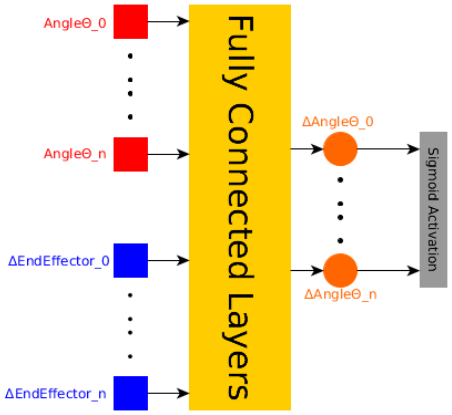


Figura 3.1 – Arquitetura de ANN utilizada.

3.2 GERAÇÃO E NORMALIZAÇÃO DOS DADOS

O primeiro passo para gerar uma amostra foi gerar um conjunto de ângulos θ válidos para as juntas, definindo assim uma posição válida para a posição e orientação do orgão terminal. A pose aleatória segue uma distribuição uniforme nos ângulos e a forma com que esta posição aleatória é definida como válida é específica de cada estudo de caso.

O segundo passo é gerar uma pequena e também aleatória variação incremental Δ_θ para cada ângulo das juntas do braço. Essa variação, entretando, foi gerada seguindo uma distribuição normal. A soma da pose aleatória com essa pequena variação incremental em cada junta define uma nova, porém próxima, posição para o orgão terminal do braço. Após calcular a posição e orientação para a pose aleatória e para a pose aleatória com incremento utilizando a FK, é obtida a variação cartesiana Δ_p do *end-effector* para essa amostra. Em outras palavras, essa variação Δ_p é um vetor de distância entre os dois pontos no espaço, onde o primeiro valor desse vetor, por exemplo, representa a variação Δ_x da posição cartesiana.

Com isso, a pose inicial (gerada amostrando uma distribuição uniforme nos ângulos das juntas), a pose acrescida do incremento (incremento gerado amostrando uma distribuição normal), e o vetor da diferença no espaço cartesiano são armazenados como uma amostra no *dataset*. Para treinar a ANN, a amostra é lida do *dataset* e um vetor com os dados que representam a pose aleatória e a variação cartesiana é criado e usado como entrada. Na saída, um vetor com os dados que representa a variação angular aleatória é utilizado.

Cada ângulo da pose aleatória é gerado entre um valor mínimo e máximo que cada junta pode assumir; $-\pi$ e π por exemplo. O mesmo é feito para o incremento dos ângulos, onde é definido um limite de Δ_θ; entre $-5°$ e $5°$ para um limite de $5°$, por exemplo. Como os ângulos são gerados de forma uniforme entre seus limites, nenhum dado é descartado. Para o incremento dos ângulos, entretanto, isso não ocorre uma vez que, como os dados gerados seguem um forma normal, eles eventualmente excedem esses limites. Dessa

forma, se houver um ângulo que exceda o limite inferior ou superior no vetor de incrementos para todas as juntas, este vetor é descartado e um novo vetor é gerado.

Para a variação cartesiana também é definido uma variação máxima que cada posição e orientação pode ter. Por exemplo, se o limite for 0,1 metros para um dos campos da posição e 0,1 radianos para um dos campos da orientação e em ao menos dos dos campos da variação cartesiana do orgão terminal existe um valor maior que esse limite, esse dado é descartado.

Esse descarte de dados foi realizado para permitir um alcance fixo e válido para a geração do *dataset*, uma vez que os incrementos dos ângulos seguem uma distribuição normal e valores muito grandes e distantes dos limites podem ser gerados. Como a ANN busca resolver um problema de regressão, quanto maior o alcance dos dados, maior é escopo que a ANN precisa aprender e menor é a precisão que a ANN vai apresentar. Além disso, isso facilita a normalização dos dados, uma vez que os limites de normalização estão definidos e fixos para cada amostra de antemão, não precisando analisar todas amostras para encontrar os limites.

A normalização, portanto, é feita entre limites definidos na geração e descarte de dados e, como são utilizadas as funções de ativação ReLU e *sigmoid* na ANN, os dados são normalizados entre 0 e 1. No estudo de caso do Thormang3 e do braço tridimensional, entretanto, a normalização foi feita entre 0,1 e 0,9, uma vez que, como pode ser observado nos resultados do primeiro estudo de caso, a ANN deste estudo de caso possuiu baixa eficiência próximo aos extremos da normalização devido a características da função de ativação *sigmoid*.

3.3 AMBIENTE DE TESTE

Para verificar a validade da abordagem, foi desenvolvido um ambiente de teste para cada estudo de caso. Nesse ambiente de teste, tarefas foram criadas para comparar a precisão da ANN com o método da inversa da JM.

Todas as tarefas buscam, iterativamente, seguir uma trajetória no espaço, seja ela circular, espiral ou por pontos. Primeiramente é definido a pose inicial para o braço e depois a trajetória é realizada, individualmente, utilizando a ANN e a JM inversa.

Na primeira iteração da trajetória circular e da trajetória espiral é, primeiramente, calculada a posição do orgão terminal das juntas utilizando a FK. Depois é definido um ponto de destino dentro da trajetória à uma distância do menor tamanho de passo que a ANN foi treinada em relação a posição cartesiana atual do orgão terminal. Por exemplo, se a ANN foi treinada com amostras de tamanho de passo entre 20 mm e 50 mm, 20 mm seria essa distância. Após calcular a variação das juntas usando a ANN ou a JM inversa, a variação angular é somada aos ângulos das juntas, formando a nova pose. Essa nova pose

é utilizada na FK para calcular a nova posição no espaço do orgão terminal. A distância entre a posição cartesiana do ponto de destino desejado e a nova posição cartesiana do orgão terminal que foi calculada é salva como sendo o erro da iteração; isso ocorre tanto para a ANN quanto para a JM inversa. Após isso um novo ponto de destino dentro da trajetória é definido. Esse ponto também é definido a uma distância do menor tamanho de passo que a ANN foi treinada em relação a posição cartesiana atual do orgão terminal. O processo é repetido até um limite máximo de iterações. Após esse processo ser finalizado para o menor passo que a ANN foi treinada, o passo é incrementado e o processo repetido até que o tamanho máximo de passo que a ANN conhece seja alcançado.

Na trajetória por pontos o processo é o mesmo, diferenciando-se somente na condição de parada. Nessa trajetória, a condição de parada, ao invés de ser somente baseada em um número máximo de iterações, também é baseada na chegada a pontos de verificação. Se atingir o último ponto de verificação, a trajetória é finalizada mesmo se o número máximo de iterações não é alcançado.

As particularidades das tarefas no ambiente de teste cada estudo de caso será abordado na sequência, assim como as demais especificidades.

3.4 BRAÇO PLANAR DE TRÊS JUNTAS

Este primeiro estudo de caso foi desenvolvido como uma prova de conceito para validar a abordagem proposta. Foi utilizada uma variação de um simulador planar[2] desenvolvido na linguagem de programação Python com a biblioteca PyGame. Esse simulador, originalmente, foi desenvolvido pelo aluno de mestrado da UFSM Fabrício Montenegro. Fabrício e o autor de presente trabalho desenvolveram juntos o estudo sobre este estudo de caso, o que acabou sendo contemplado um artigo (MONTENEGRO et al., 2018).

Ele possui duas juntas rotacionais em um espaço planar e com 600 unidades de comprimento, representando um braço de 60 cm, tamanho próximo ao de uma braço robótico típico. Na versão original o braço também é dividido em dois segmentos de 30 cm cada.

Dessa forma, o simulador foi adaptado para possuir três juntas rotacionais, também em um espaço planar. Cada segmento passou a ter 20 cm para manter os 60 cm de comprimento de um braço típico. Na Figura 3.2 pode-se observar o braço planar já posicionado na posição inicial do ambiente de teste.

[2]Simulador planar original: https://github.com/SplinterFM/JacobianNNPlanar

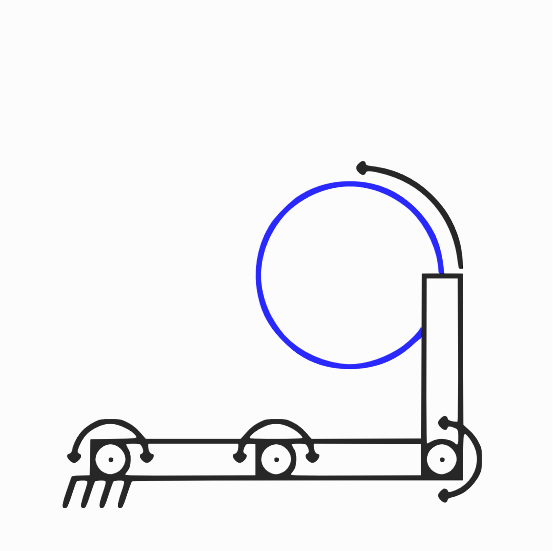


Figura 3.2 – Pose inicial do braço no simulador. As linhas pretas representam os segmentos dos braços e o circulo azul representa a trajetória.

3.4.1 ANN

Como entrada para da ANN foram utilizados os ângulos $\theta 1$, $\theta 2$ e $\theta 3$, representando a posição da primeira, segunda e terceira junta do braço, respectivamente; a outra entrada foi a variação da posição cartesiana de x e y, Δx e Δy. Para a saída foram utilizados as variações angulares, $\Delta\theta_1$, $\Delta\theta_2$ e $\Delta\theta_3$. Estas informações estão sumarizadas na Tabela 3.1.

Entradas	θ_1, θ_2, θ_3, Δx, Δy
Saídas	$\Delta\theta_1$, $\Delta\theta_2$, $\Delta\theta_3$

Tabela 3.1 – Entradas e saídas da ANN para o braço planar

Para esse estudo de caso foi utilizado uma ANN de cinco camadas, com 640, 320 e 160 neurônios nas camadas escondidas, totalizado 260.002 parâmetros no total. Devido ao elevado número de parâmetros, a ANN, nas primeiras versões, apresentou *over-fitting*, problema que foi resolvido na versão final através da utilização da técnica de regularização *Dropout* na segunda e na terceira camada escondida. O sumário da ANN pode ser observado na Figura 3.3. O treinamento da ANN foi feito durante 100 épocas com um tamanho de *batch* de 1024 amostras.

3.4.2 Geração de dados

Para treinar a ANN foram gerados *datasets* grandes, com dois milhões de amostras cada. Para gerar cada amostra foram utilizadas as equações 2.21 e 2.22 da FK desse

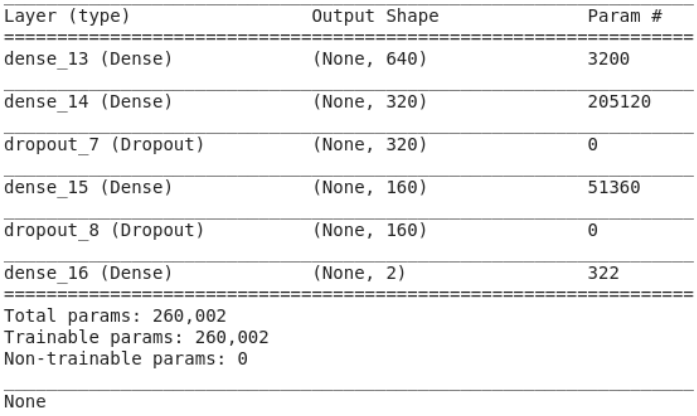


Figura 3.3 – Sumário da ANN utilizada para o braço planar

estudo.

Ao total foram gerados quatro *datasets*, sendo treinada uma versão da ANN para cada um dos *datasets*. Os limites de cada *dataset* podem ser visualizados na Tabela 3.2.

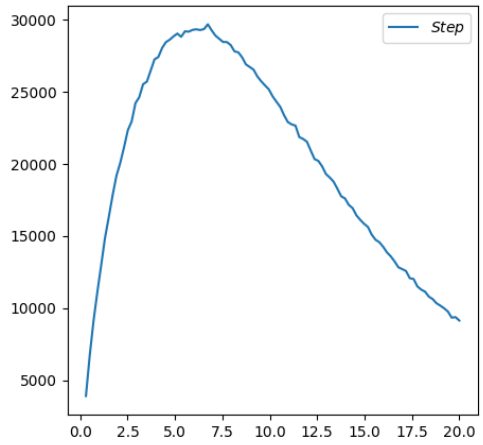
Ângulo máximo de Δ_θ	Tamanho de passo mínimo	Tamanho de Passo máximo
$5°$	0,1 mm	20 mm
$15°$	20 mm	50 mm
$20°$	50 mm	100 mm
$20°$	0,1 mm	100 mm

Tabela 3.2 – Limites dos *datasets* do braço planar

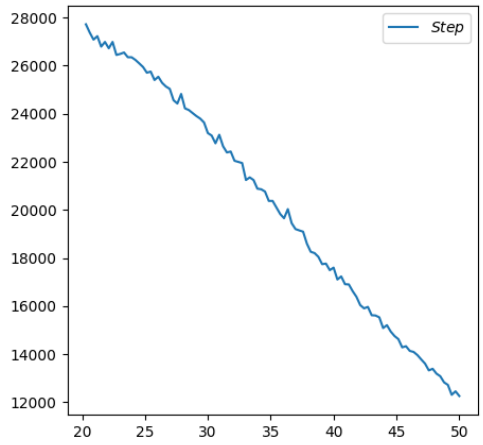
As poses aleatórias foram geradas entre o ângulo mínimo que cada uma das juntas pode assumir, $-\pi$, e o ângulo máximo, π. A variação angular foi gerada utilizando uma distribuição com média de 0 e desvio padrão de 0,33. Valores maiores que 1 e menores que -1 foram descartados e os valores restantes foram dimensionados para o limite definido para a variação dos ângulos para cada *dataset*; $-5°$ e $5°$ para um limite de $5°$, por exemplo. Analogamente, quando o vetor Δx e Δy resultante da FK era maior ou menor que o máximo e mínimo tamanho de passo definido, respectivamente, esse dado era descartado; caso que ocorreu, por exemplo, no primeiro *dataset* se o vetor da variação cartesiana for maior que 20 mm e menor que 0,1 mm.

A distribuição das amostras para cada *dataset* pode ser observado nas Figuras 3.4. A forma do histograma das amostras possui um viés de distribuição normal, o que demonstra a influência das não linearidades da cinemática do braço.

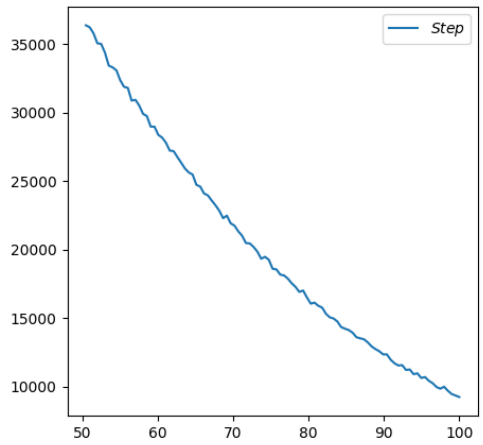
Nas Figuras 3.4a e 3.4d pode-se observar a curva de distribuição normal do número de amostras. Como esperado, há a presença de um número muito elevado de amostras de pequeno e médio tamanho de passo. Como está discutido nas Seções 4 e 5, é muito difícil superar a JM inversa para passos pequenos devido à grande precisão que que ela possui. Por isso o foco da metodologia está em tamanhos de passo grande, onde o método tradicional não funciona muito bem. Assim, no segundo e terceiro *datasets*, das Figuras



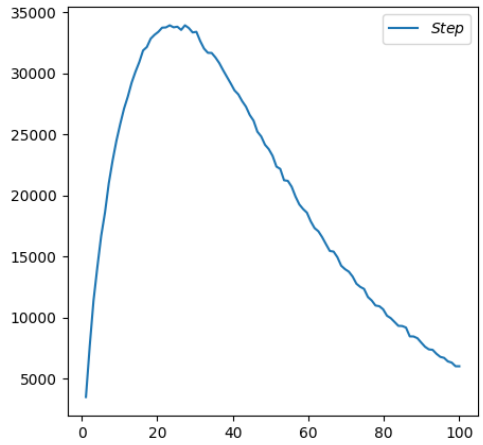
(a) Passo de $0,1mm$ até $20mm$



(b) Passo de $20mm$ até $50mm$



(c) Passo de $50mm$ até $100mm$



(d) Passo de $0,1mm$ até $100mm$

Figura 3.4 – Histograma mostrando o número de amostras para cada tamanho de passo dos *datasets* do braço planar

3.4b e 3.4c, amostras com pequeno tamanho de passo foram descartadas. Nestes dois casos a distribuição normal acabou sendo truncada devido ao descarte das amostras com tamanho de passo menor que 20 mm e 50 mm, respectivamente, produzindo uma curva descendente no histograma de ambos os casos.

Durante o treinamento de cada ANN para cada *dataset*, os dados foram divididos em dados de treinamentos e dados de validação, sendo 90% e 10% dos dados para cada um, respectivamente.

3.4.3 Ambiente de Teste

Após o treinamento da ANN foi desenvolvida uma tarefa para comparar o desempenho com a JM inversa. A tarefa consistiu em controlar o braço para seguir uma trajetória circular de 100 mm de raio, como pode ser observado, em azul, na Figura 3.2. A trajetória foi centralizada nas coordenadas (300,200), sendo seguida pelo braço no sentido anti-horário de movimento.

Portanto, como esse estudo de caso segue uma forma circular de validação, foi definido um limite máximo de 1000 iterações para completar a tarefa para cada tamanho de passo. O incremento utilizado para o aumento do tamanho de passo foi de 0,01 mm; ou seja, para o primeiro conjuto de dados, com dados entre 0,1 mm e 20 mm, 1000 iterações foram feitas utilizando um tamanho de passo de 0,1 mm, 1000 iterações usando um tamanho de passo de 0,11 mm e assim por diante.

Para a ANN, os dados referentes à posição atual θ do braço durante a trajetória e o incremento Δ_p foram normalizados dentro de seus respectivos limites e os dados de saída da ANN foram desnormalizados, também dentro de seus limites definidos, e somados aos ângulos da pose atual. A JM inversa, por sua vez, foi calculada a cada iteração, conforme a sua definição 2.26.

3.5 BRAÇO TRIDIMENSIONAL DE TRÊS JUNTAS

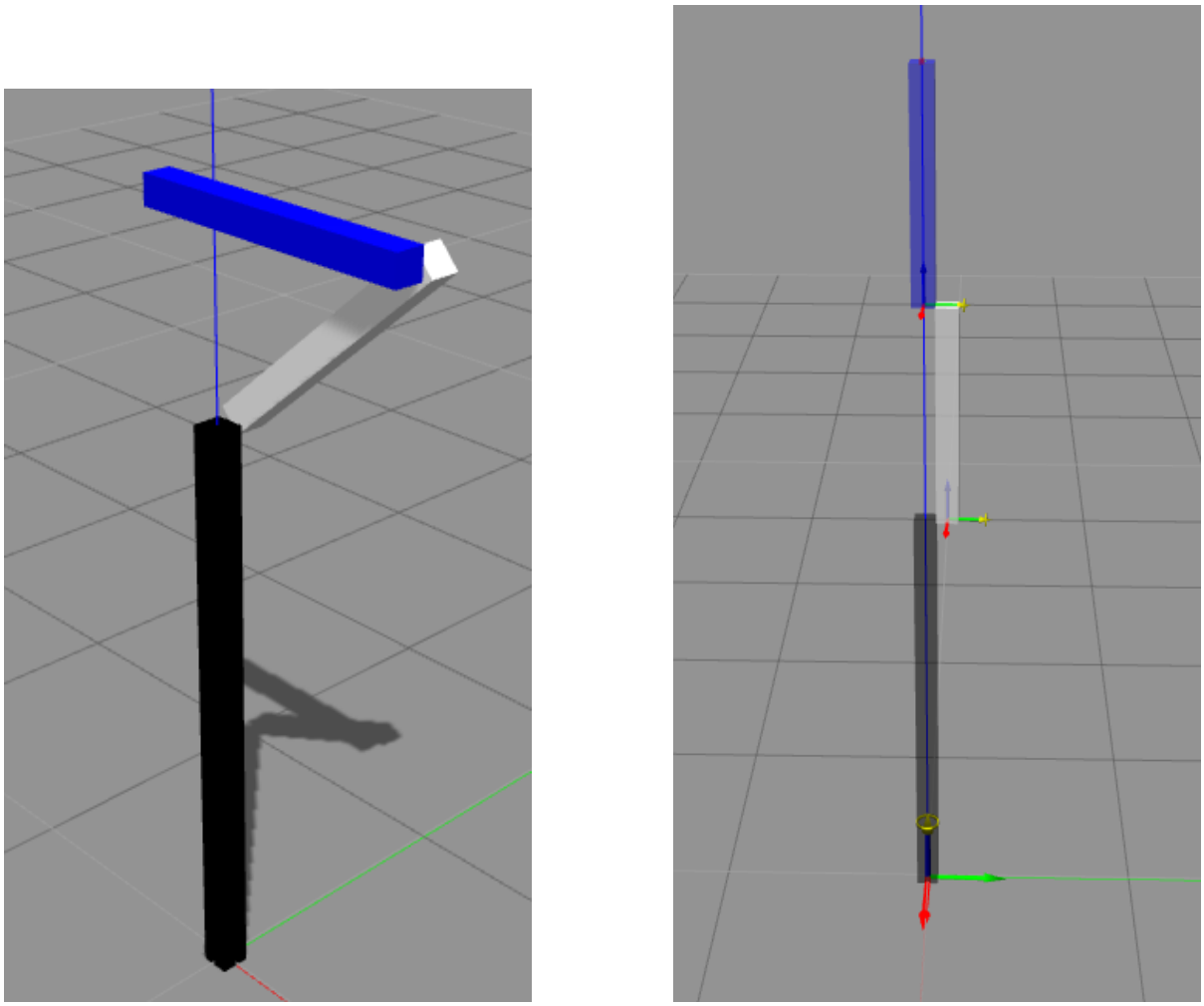
O segundo estudo de caso foi desenvolvido mais para ser uma prova de conceito para a abordagem proposta, desta vez em um espaço tridimensional. Para esse estudo de caso foi desenvolvido um pacote[3] de ROS para criá-lo e controlá-lo no simulador Gazebo. Foi feita a descrição do braço, juntamente com o controle com interface com ROS e um *plugin* para controlá-lo.

Na descrição[4], os três segmentos foram descritos com massa, largura e profundidade idênticas, de 1 unidade de massa, 0,1 unidade de comprimento e 1 unidade de massa, respectivamente. As alturas dos segmentos 1,2 3, a partir da base, foram definidas com 2 unidades de comprimento, 1 unidade de comprimento e 1 unidade de comprimento, respectivamente. A altura refere-se a dimensão z, a largura à y e a profundidade à x. Foi considerado 1 unidade de comprimento representando 1 m. Além disso, foram utilizados materiais e texturas padrões do Gazebo durante a descrição. O mundo[5] possui apenas o braço e ambos podem ser observados na Figura 3.5a, já com o braço posicionado na posição inicial.

[3] Braço de 3 juntas no simulador Gazebo: https://github.com/ricardoGrando/Three-DoF-Arm

[4] https://github.com/ricardoGrando/Three-DoF-Arm/blob/master/rrbot_description/urdf/three_dof_arm.xacro

[5] https://github.com/ricardoGrando/Three-DoF-Arm/blob/master/three_dof_arm_gazebo/worlds/three_dof_arm.world



(a) Posição inicial do Braço (b) Juntas do Braço

Figura 3.5 – Braço tridimensional de três tuntas no simulador Gazebo

A inércia de cada segmento foi definida da seguinte forma:

$$\begin{aligned} ixx &= m/12,0 * (l * l + a * a) \\ ixy &= 0,0 \\ ixz &= 0,0 \\ iyy &= m/12,0 * (a * a + p * p) \\ iyz &= 0,0 \\ izz &= m/12,0 * (l * l + p * p) \end{aligned} \tag{3.2}$$

O primeiro segmento do braço está ligado ao mundo através de uma junta de revolução em z, enquanto que a segunda e a terceira junta, ligadas entre os segmentos 1 e 2 e 2 e 3, respectivamente, possui rotação em y. A segunda junta e o segundo segmento foram deslocados 0,1 unidades de comprimento para evitar colisões entre segmentos próximos. O amortecimento dinâmico utilizado foi de 0,7 unidades de amortecimento para todas as

juntas, enquanto que o coeficiente de Coulomb foi de 0,2 em ambas direções para todos os segmentos.

O controle interfaciado[6] com ROS se baseou no *plugin* padrão para controle do Gazebo com ROS, *libgazebo_ros_control.so*. As juntas foram definidas como do tipo *JointPositionController*, que é o tipo de junta de revolução no *libgazebo_ros_control*

A inicialização da descrição[7] e controle[8] estão listadas abaixo:

- $roslaunch\ three\_dof\_arm\_gazebo\ three\_dof\_arm\_world.launch$
- $roslaunch\ three\_dof\_arm\_control\ three\_dof\_arm\_control.launch$

Para dar vida ao pacote desenvolvido, foi criado um *plugin*. Esse *plugin* possui uma classe principal[9], uma classe[10] para realizar a interface entre a classe principal e os serviços de ROS do pacote e uma classe[11] para realizar a interface entre a classe principal e os tópicos de ROS do pacote, além das funções auxiliares da classe principal; tudo desenvolvido na linguagem de programação Python com Rospy. Os tipos de mensagem e de serviços utilizados são padrão do ROS e do Gazebo.

A classe que comunica com os tópicos é uma *thread* cuja função é definir um valor para um tópico específico de uma junta. Essa classe possui um controle de exclusão mútua entre a classe principal que à instancia e as demais *threads*; ou seja, as *threads* e a classe principal não rodam ao mesmo tempo, pois compartilham variáveis. Nessa classe, um *publisher* é criado com Rospy para um tópico, fica rodando enquanto ROS estiver funcionando. O valor publicado é do tipo *Float64*, advindo da classe principal.

A classe resposável por interfacear com os serviços apenas realiza a instanciação de um objeto do mundo do Gazebo. Essa classe foi utilizada para instanciar um objeto na posição do orgão terminal durante a trajetória para melhor visualizá-la. Para fazer isso, um *ServiceProxy* do serviço *SpawnModel* do Gazebo foi criado, instanciando o objeto quando chamado.

A classe principal possui várias funções para aplicar a abordagem no pacote. Possui a função de define a cinemática do braço, conforme sua descrição 2.13, 2.14 e 2.15. Nessa mesma função essas funções da FK são derivadas e os coeficientes da JM inversa genérica (sem os ângulo de uma pose aplicados) são definidos. Essa classe, além de usar as outras classes para realizar a interface com ROS e Gazebo, também possui as funções para testar o desempenho da JM inversa e da ANN, gerar dados e amostrar resultados, que estão abordadas nas sub seções seguintes.

[6] https://github.com/ricardoGrando/Three-DoF-Arm/blob/master/three_dof_arm_control/config/three_dof_arm_control.ya
[7] https://github.com/ricardoGrando/Three-DoF-Arm/blob/master/three_dof_arm_gazebo/launch/three_dof_arm_world.lau
[8] https://github.com/ricardoGrando/Three-DoF-Arm/blob/master/three_dof_arm_control/launch/three_dof_arm_control.la
[9] https://github.com/ricardoGrando/Three-DoF-Arm/blob/master/kinematics/kinematics.py
[10] https://github.com/ricardoGrando/Three-DoF-Arm/blob/master/kinematics/serviceCartesianState.py
[11] https://github.com/ricardoGrando/Three-DoF-Arm/blob/master/kinematics/topicCartesianState.py

3.5.1 ANN

Para este estudo de caso também foram utilizados $\theta 1$, $\theta 2$ e $\theta 3$ para representar os ângulos da primeira, segunda e terceira junta do braço, respectivamente. Além de Δx e Δy, foi utilizada também a variação da posição cartesiana em z (Δz), diferentemente do primeiro estudo de caso onde, por se tratar de um braço planar, somente Δx e Δy existiam. Para a saída foram definidos as variações angulares, $\Delta\theta_1$, $\Delta\theta_2$ e $\Delta\theta_3$. Estas informações estão sumarizadas na Tabela 3.3.

Entradas	θ_1, θ_2, θ_3, Δx, Δy, Δz
Saídas	$\Delta\theta_1$, $\Delta\theta_2$, $\Delta\theta_3$

Tabela 3.3 – Entradas e saídas da ANN para o braço tridimensional de três juntas

Nesse estudo de caso também foi utilizada uma ANN de cinco camadas, com 640, 320 e 160 neurônios nas camadas escondidas, totalizado 261,443 parâmetros no total. Também foi utilizado *dropout* na segunda e terceira camada escondida para evitar *overfitting*. O sumário da ANN pode ser observado na Figura 3.6.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_17 (Dense)             (None, 640)               4480
_________________________________________________________________
dense_18 (Dense)             (None, 320)               205120
_________________________________________________________________
dropout_9 (Dropout)          (None, 320)               0
_________________________________________________________________
dense_19 (Dense)             (None, 160)               51360
_________________________________________________________________
dropout_10 (Dropout)         (None, 160)               0
_________________________________________________________________
dense_20 (Dense)             (None, 3)                 483
=================================================================
Total params: 261,443
Trainable params: 261,443
Non-trainable params: 0
_________________________________________________________________
None
```

Figura 3.6 – Sumário da ANN utilizada para o braço tridimensional

O treinamento da ANN foi feito durante 100 épocas com um tamanho de *batch* de 4096 amostras.

3.5.2 Geração de dados

A geração de dados para este estudo de caso foi feita baseada nas equações do Capítulo 2: 2.13 para x, 2.14 para y e 2.15 para z.

Foram gerados dois conjuntos de *datasets*, totalizando oito *datasets* ao total com 6.000.000 de amostras cada. O primeiro conjunto de *datasets* possui as mesmos limites utilizados para os *datasets* do primeiro estudo de caso, vide Tabela 3.2.

O segundo conjunto de *datasets* foi gerado levando em consideração o maior comprimento do braço desse estudo de caso quando comparado com o comprimento do braço do estudo anterior. O braço tridimensional de três juntas possui uma proporção sete vezes maior e, levando isso em consideração, foram gerados datasets com tamanho de passo mínimo e máximo sete vezes maiores. Os limites para esse conjunto de *datasets* podem ser visualizados na Tabela 3.4. Da mesma forma que no estudo anterior, uma versão da ANN foi treinada para cada *dataset*, totalizando oito modelos, portanto.

Ângulo máximo de Δ_θ	Tamanho de passo mínimo	Tamanho de passo máximo
$5°$	0,7 mm	140 mm
$15°$	140 mm	350 mm
$20°$	350 mm	700 mm
$20°$	0,7 mm	700 mm

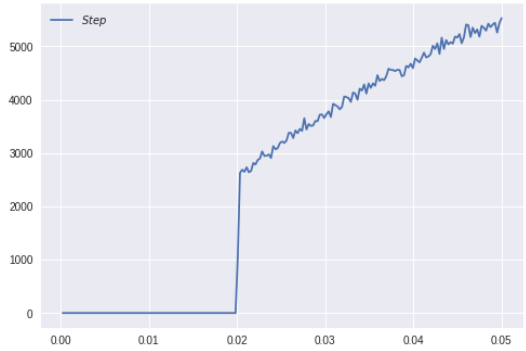
Tabela 3.4 – Limites do segundo conjunto de *datasets* do braço tridimensional

O ângulo de cada junta foi gerado de forma aleatória seguindo uma distribuição uniforme entre o ângulo mínimo e máximo que cada junta pode assumir; - π e π respectivamente para todas as juntas. A variação angular foi gerada utilizando distribuição normal com média de 0 e desvio padrão de 0,33. Os valores foram redimensionados para o limite definido para a variação dos ângulos de cada *dataset* e valores fora do limite de ângulo máximo foram descartados, assim como no estudo anterior. Da mesma forma, foram descartados os dados cuja norma do vetor da variação cartesiana estivessem fora do alcance compreendido entre o tamanho de passo mínimo e o tamanho de passo máximo. A distribuição das amostras pode ser visualizado nas imagens 3.7 e 3.8. Foi utilizado 500.000 amostras de cada *dataset* para criar o histograma

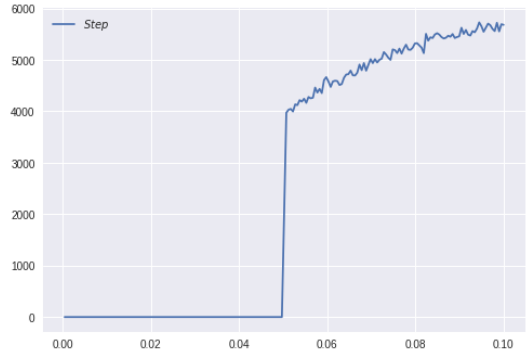
De modo geral, o mesmo viés de distribuição normal observado nos *datasets* do estudo anterior pode ser observado nos histogramas 3.8 e 3.7, assim como o truncamento das amostras. Além disso, pode ser observado que os histogramas do primeiro conjunto de *datasets* não possuem uma curva normal completa nos *datasets* que não possuem truncamento, diferentemente do conjunto de *datasets* do estudo anterior. Isso evidencia que o tamanho de passo é muito pequeno, dada as dimensões do braço. No histograma do segundo conjunto de *datasets*, onde há uma proporcionalidade entre os braços planar e tridimensional, a distribuição normal foi melhor representada e os histogramas se assemelharam aos histogramas do primeiro estudo de caso.

Durante o treinamento de cada ANN para cada *dataset*, os dados foram divididos em dados de treinamento e dados de validação, sendo 80% e 20% dos dados para cada um, respectivamente.

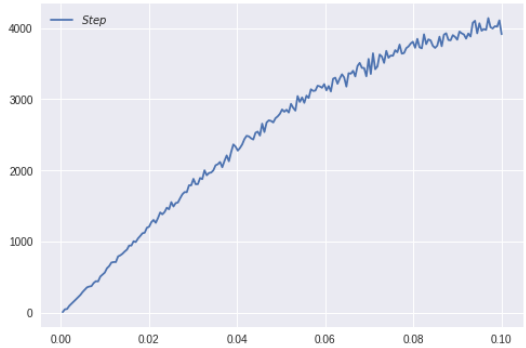
(a) Passo de $0,1mm$ até $20mm$



(b) Passo de $20mm$ até $50mm$



(c) Passo de $50mm$ até $100mm$



(d) Passo de $0,1mm$ até $100mm$

Figura 3.7 – Histograma mostrando o número de amostras para cada tamanho de passo do primeiro conjunto de *datasets* do braço tridimensional

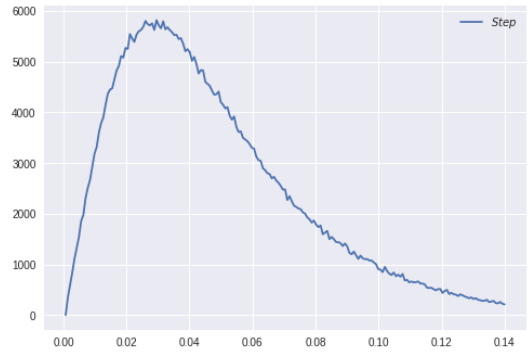
3.5.3 Ambiente de Teste

A tarefa utilizada para comparar o desempenho da ANN com a JM inversa consistiu em controlar o braço para seguir uma trajetória espiral de 700 mm de raio. A posição inicial do orgão terminal foi definida em $x = -0,2928$, $y = 0$ e $z = 2,7071$, o que equivale a um conjunto de ângulos com $\theta_1 = 0$, $\theta_2 = \frac{\pi}{4}$ e $\theta_2 = -\frac{3\pi}{4}$. A trajetória foi realizada no sentido anti-horário. Na Figura 3.5a pode ser observada a posição inicial e na Figura 3.9 pode ser observado um exemplo de trajetória sendo executado com a JM com um tamanho de passo de 50 mm.

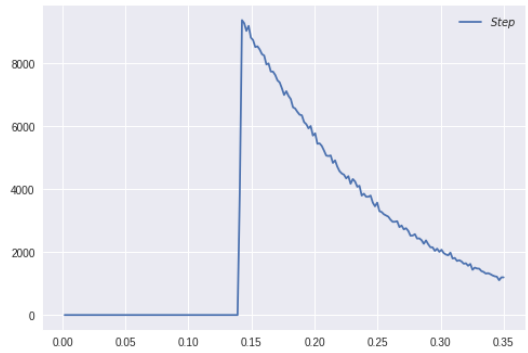
Nesse estudo foi definido um limite de 1000 iterações para completar a tarefa para cada tamanho de passo. O incremento utilizado no tamanho de passo foi de 1 mm.

3.6 THORMANG3 - GAZEBO

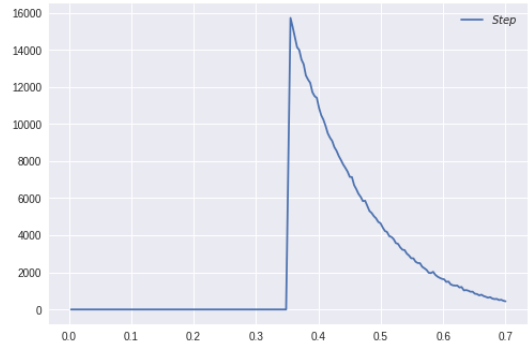
Os dois primeiros estudos de caso foram desenvolvidos focando em apresentar a ideia que o trabalho propõe, mostrando os conceitos que à fundamentam em exemplos simples; com soluções fechadas. Este último estudo de caso foi feito para validar a abor-



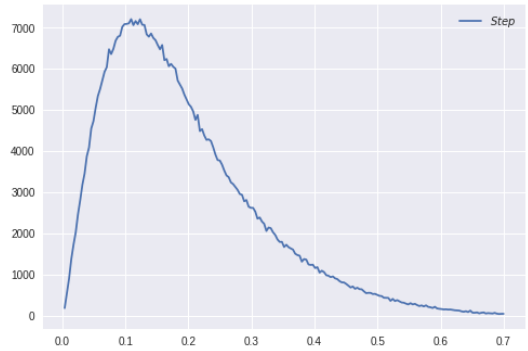
(a) Passo de $0,7mm$ até $140mm$



(b) Passo de $140mm$ até $350mm$



(c) Passo de $350mm$ até $700mm$



(d) Passo de $0,7mm$ até $700mm$

Figura 3.8 – Histograma mostrando o número de amostras para cada tamanho de passo do primeiro conjunto de *datasets* do braço tridimensional

dagem em um braço robô mais complexo, mais próximo do que é tipicamente encontrado em robôs humanoides e em braços manipuladores industriais.

O braço do Thormang3 é consideravelmente mais complexo do que os demais braços discutidos neste trabalho. Criar uma interface em ROS ou em qualquer outra linguagem partindo do zero, como foi feito para os outros dois braços, resultaria em bastante trabalho. Entretando, por ser um robô bastante usado, ele possui uma interface em ROS e no Gazebo, como está mencionado no Capítulo 2. Dessa forma, para aplicar a abordagem nesse estudo de caso, foi criado um pacote [12] em ROS que utiliza a interface do Thormang3 já pronta.

Para poder utilizar a descrição cinemática do Thormang3, foi desenvolvido um servidor[13] na linguagem C++ que realiza os cálculos da FK para um determinado conjunto de juntas, no caso as juntas do braço esquerdo, e calcula a JM desse conjunto, dado a posição e orientação do orgão terminal daquela pose. Dessa forma, para utilizar esse serviço, os ângulos das juntas são enviados e a posição e orientação cartesiana juntamente com a JM é retornada para a função requisitora.

No algoritmo, o servidor requisita a função de cinemática do Thormang3, definindo

[12] Link do pacote: https://github.com/ricardoGrando/hybrid_control_api

[13] https://github.com/ricardoGrando/hybrid_control_api/blob/master/src/hybrid_control_api/fk_and_jacobian_server.cpp

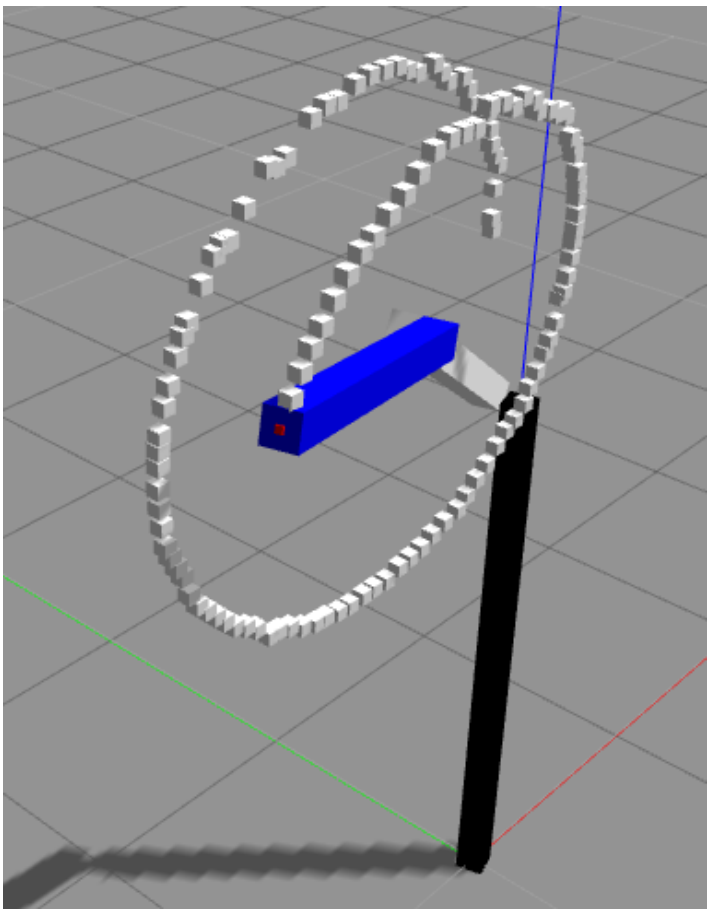


Figura 3.9 – Exemplo de trajetória no ambiente de teste.

as respectivas juntas no braço esquerdo. Os endereços para as juntas do braço esquerdo na biblioteca da cinemática são os números pares entre 2 e 14, sendo 34 o endereço para o orgão terminal. Após definir as juntas, a função da biblioteca de cinemática *calcForwardKinematics()* é chamada utilizando o endereço da primeira junta do braço esquerdo, *Shoulder Pitch 1*. Para a JM, uma lista contendo os endereços das 7 juntas, de 2 a 14, é utilizada na função *calcJacobian()*, retornana a matriz com os coeficientes calculados. Portanto, o processo para cálculo da IK, que ainda envolve a inversão da matriz e multiplicações iterativas, é realizado em um cliente, fora do escopo do servidor, apesar da biblioteca possuir funções prontas para o calculo da IK. Isso foi feito para tornar o servidor utilizável na geração de dados, nos testes da IK com a JM e nos teste com a ANN. Ou seja, um só servidor realiza todas as demandas cinemáticas que o pacote necessita.

Além de necessitar uma interface com as bibliotecas de cinemáticas do Thormang3, o pacote possui uma interface com o Gazebo, permitindo assim a visualização, mesmo que em simulação, de movimentos que possam ser calculados com a cinemática. Essa interface, assim como na ideia apresentada para o braço tridimensional de três juntas, possui uma classe[14] que realiza a interface com o Thormang3 no Gazebo. Essa classe possui um função (*runOnThormang()*) que, quando solicitada, realiza o controle de *threads*

[14]https://github.com/ricardoGrando/hybrid_control_api/blob/master/scripts/runOnThormang.py

para que essas publiquem um determinado valor para o tópico de uma determinada junta no Gazebo por um determinado número de vezes. A classe da thread[15] que publica no tópico é a mesma do pacote do braço tridimensional de três juntas. O dado publicado é um número em ponto flutuante.

O pacote também possui aplicações para gerar dados[16], realizar testes[17] e funções auxiliares[18] para normalização e desnormalização de dados e para conversão de ângulos.

3.6.1 ANN

A complexidade desse estudo de caso, se comparado com os outros dois, fica melhor evidente quando analisadas as entradas e as saídas na arquitetura da ANN. Os ângulos de todas as juntas do braço, sumarizados na Tabela 2.1, foram utilizados como entrada, com exceção das juntas da garra. Em complemento a esses, na variação cartesiana neste estudo também foi utilizado a orientação, uma vez que possui 7 juntas. Ao total, são 14 entradas, sendo 3 para a variação cartesiana da posição e 4 para a variação cartesiana da orientação, uma vez que a última foi utilizada em forma de *quaternion*. Estas informações estão sumarizadas na Tabela 3.5.

Entradas	θ_{sh1}, θ_{shr}, θ_{sh2}, θ_{ely}, θ_{wrr}, θ_{wry}, θ_{wrp}, Δx, Δy, Δz, Δw_x, Δw_y, Δw_z, Δw_w
Saídas	$\Delta_{\theta_{sh1}}$, $\Delta_{\theta_{shr}}$, $\Delta_{\theta_{sh2}}$, $\Delta_{\theta_{ely}}$, $\Delta_{\theta_{wrr}}$, $\Delta_{\theta_{wry}}$, $\Delta_{\theta_{wrp}}$

Tabela 3.5 – Entradas e saídas da ANN para o estudo no Thormang3

Seguindo a ideia dos demais estudos, foi utilizada uma arquitetura de ANN com um número de camadas escondidas proporcional ao número de juntas, totalizando, assim, sete camadas escondidas e 9 camadas no total.

O número de neurônios nas camadas escondidas foi de 150, 120, 90, 75, 65, e 55, partindo da camada escondida mais próxima da entrada para a mais próxima a saída. O número total de parâmetros foi de 47.047 parâmetros. O sumário da ANN pode ser observado na Figura 3.10.

O treinamento da ANN foi feito durante 340 épocas com um tamanho de *batch* de 2000 amostras.

[15] https://github.com/ricardoGrando/hybrid_control_api/blob/master/scripts/topicCartesianState.py
[16] https://github.com/ricardoGrando/hybrid_control_api/blob/master/scripts/createDataset.py
[17] https://github.com/ricardoGrando/hybrid_control_api/blob/master/scripts/realJacobian.py
[18] https://github.com/ricardoGrando/hybrid_control_api/blob/master/scripts/utils.py

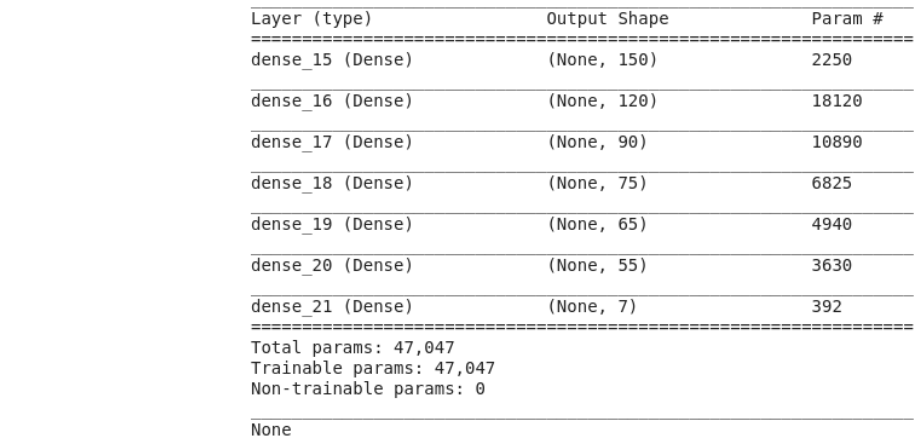


Figura 3.10 – Sumário da ANN utilizada para o estudo de caso do Thormang3.

3.6.2 Geração de dados

Nos estudos anteriores a geração aleatória e uniforme do conjunto de ângulos para as juntas era feita tendo como fator limitante somente o menor e o maior ângulo que a junta poderia ter. Essa condição era a única necessária para definir que um conjunto de ângulos para as juntas resultava em uma posição válida.

Para esse último estudo, além de levar em consideração os limites de ângulos que cada uma das juntas pode assumir, vide a Tabela 2.1, a definição de uma posição válida precisou levar em consideração também auto-colisões com o restante do corpo que um conjunto de ângulos das juntas do braço pode causar. Como não há nenhuma biblioteca para Thormang3 que possibilite identificar um conjunto de juntas que resulte em auto-colisão, foi desenvolvido um código para utilizar a interface do Thormang3 no Gazebo e verificar a validade de uma pose.

Primeiramente, foi modificada a descrição XML do Thormang3 no Gazebo [19] e adicionada uma condição para fixar a sua base no mundo. Esse código permitiu que o robô ficasse estático no mundo, não sofrendo o efeito de um movimento brusco do braço, que poderia resultar em uma queda. Dessa forma, uma posição aleatória poderia ser definida instantaneamente para o braço no simulador, sem precisar realizar uma interpolação para chegar lá para evitar uma queda. O modo que o Thormang3 foi fixado no mundo e sua posição inicial pode ser visualizado na Figura 3.11.

O próximo passo foi desativar as colisões do corpo do Thormang3 e colocar caixas cujas as dimensões envolvessem as partes do corpo aonde o braço poderia colidir. Isso foi necessário fazer porque o simulador Gazebo também não possui uma biblioteca que possibilite detectar colisões entre partes de um mesmo modelo, somente de modelos diferentes. O Thormang3 com as caixas para verificar as colisões pode ser visualizado na Figura 3.12.

Com isso, conseguindo detectar auto-colisões e podendo definir instantaneamente

[19] https://github.com/ROBOTIS-GIT/ROBOTIS-THORMANG-Common/blob/master/thormang3_description

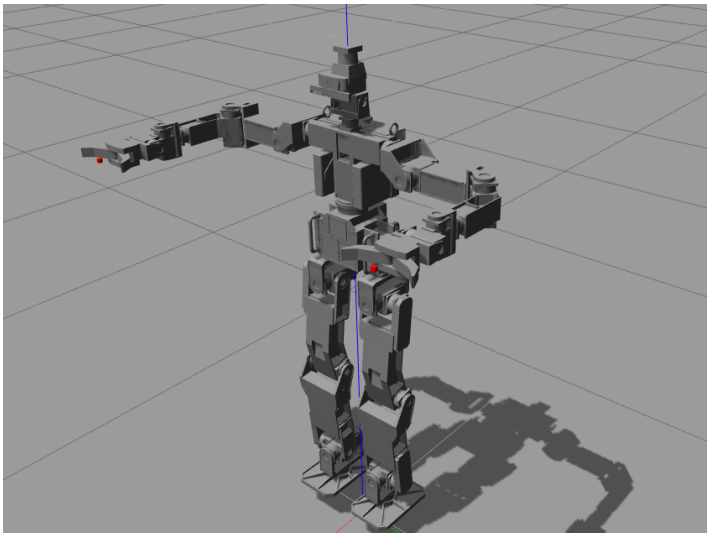


Figura 3.11 – Posição inicial do Thormang3 fixado no mundo

uma pose para o braço sem se preocupar com uma queda do robô, foi desenvolvido um pacote para gerar poses aleatórias e uniformes e enviá-las para o simulador. Um outro código também foi desenvolvido para verificar se houve colisão entre o braço e alguma das caixas e enviar essa informação de validade para o primeiro pacote. Não havendo colisão, a pose era salva em um *dataset* de posições válidas.

Esse processo é custoso computacionalmente e muito lento para gerar uma quantidade razoável de posições válidas, mesmo desativando a interface gráfica do Gazebo. Devido a essa condição, dessa forma se tornou inviável gerar uma quantidade suficiente de posições válidas para gerar dados para a ANN. Ao total somente aproximadamente 30.000 amostras de posições válidas foram geradas.

Dado esse outro fator limitante, além de gerar um *dataset* de posições válidas, também foi gerado um *dataset* de posições inválidas; esse com aproximadamente 7000 amostras. Utilizando esses dois *datasets* novas amostras foram geradas e sua validade foi testada levando em consideração a distância entre cada uma delas e cada uma das amostras dos dois *datasets*. Se a amostra gerada estivesse mais próximo a uma amostra do *dataset* de amostra válida; ela era dada como válida; da mesma forma que se estivesse mais próximo a uma amostra do *dataset* de amostras inválidas ela era dada com inválida. Apesar não ser possível ter absoluta certeza que a amostra gerada resultaria em colisão ou não, dessa forma foi possível otimizar o processo de geração de poses válidas.

Ao total, foram geradas aproximadamente 1.000.000 de poses válidas dessa forma. Apesar do processo para gerar amostras ter ficado mais rápido, ele ainda foi lento pois para cada amostra era necessário verificar a distância dela para cada uma das aproximadamente 37.000 amostras do outros dois *datasets*.

Após a geração do *dataset* de poses válidas, foi realizada a geração de dados para

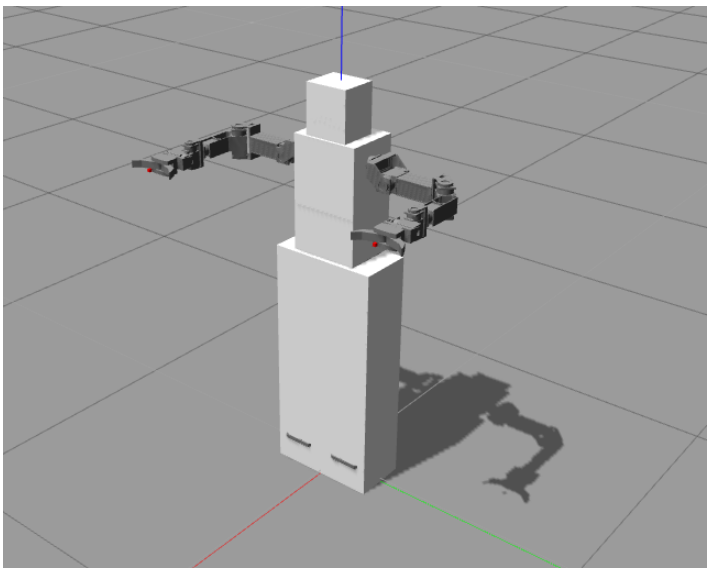


Figura 3.12 – Thormang3 com as caixas para verificar auto-colisões

treinar a ANN utilizando as bibliotecas de cinemática[20] do Thormang3. Para esse estudo de caso, levando em consideração as condições para gerar poses válidas, somente um *dataset* para a ANN foi gerado e, portanto, somente um modelo da ANN foi treinado.

A variação dos ângulos foi gerada utilizando uma distribuição normal com média de 0 e desvio padrão de 0,33 com variação máxima de 5 graus. O valor para cada junta foi redimensionado para o ângulo máximo e mínimo que ela suporta, sendo descartados os dados fora do limite. Dados com uma variação cartesiana resultante da posição em x, y e z maior que 100 mm foram descartados; o mesmo aconteceu para dados com uma variação cartesiana da orientação maior que 0,1 unidades de *quaternion* em x, y, z e w.

Durante o treinamento das primeiras versões da ANN, foi constatado que ela apresentava um desempenho ruim com poucos dados, como era esperado. Entretanto, mesmo utilizando o *dataset* de 1.000.000 de amostras o desempenho não melhorou como esperado. Uma medida tomada para tentar melhorar o desempenho foi gerar mais amostras, mas sem criar novas poses válidas. Foram gerados quatro novos *datasets* de 2.500.000 de amostras em cada um deles, totalizando 10.000.000 de amostras. Assim, cada pose válida foi utilizada para gerar 10 amostras. Apesar de aparentemente haver uma tendência para ocorrer *over-fitting*, esse problema aparentemente não ocorreu, como pose ser visto na Seção 4. Na Figura 3.13 pode ser observado o histograma gerado sobre 500.000 amostras de um dos quatro *datasets* gerados.

Durante o treinamento da ANN foi utilizado um *generator* para pegar aleatoriamente uma amostra por vez e treinar na ANN devido a quantidade de amostras. Os dados foram divididos em dados de treinamento e dados de validação, sendo 90% e 10% dos dados para cada um, respectivamente.

[20] https://github.com/ROBOTIS-GIT/ROBOTIS-THORMANG-MPC/blob/master/thormang3_kinematics_dynamics

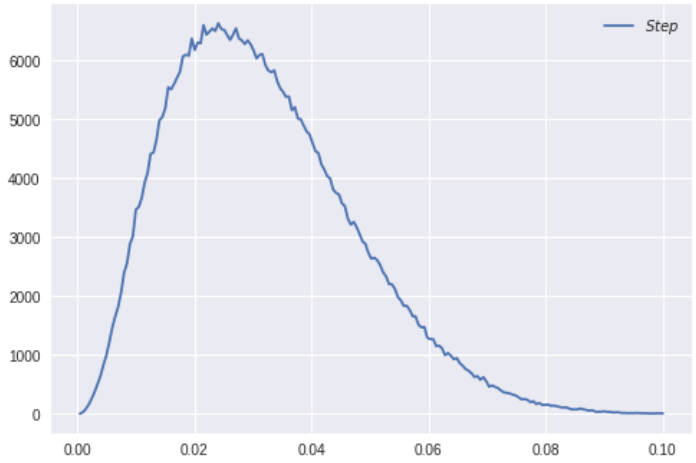


Figura 3.13 – Histograma de um dos *datasets* do Thormang3.

3.6.3 Ambiente de Teste

A tarefa para comparar o desempenho da ANN com a JM inversa consistiu em controlar o braço para ir de um ponto inicial ao um ponto final em uma trajetória por pontos. A posição inicial do orgão terminal braço foi definida na posição $p_x = 0,43$, $p_y = 0,462$ e $p_z = 0,121$ com orientação em *quaternion* de $\theta_x = 1$, $\theta_y = 0$, $\theta_z = 0$ e $\theta_w = 0$. Os pontos da trajetória, que constituem a trajetória por pontos, podem ser analisados na Tabela 3.6.

Pontos da trajetória	Vetor da posição e orientação
Ponto Inicial	(0,43, 0,462, 0,121, 1, 0, 0, 0)
Ponto A	(0,53, 0,462, 0,121, 1, 0, 0, 0)
Ponto B	(0,53, 0,362, 0,121, 1, 0, 0, 0)
Ponto C	(0,53, 0,362, 0,021, 1, 0, 0, 0)
Ponto D	(0,43, 0,362, 0,021, 1, 0, 0, 0)
Ponto E	(0,43, 0,362, 0,121, 1, 0, 0, 0)
Ponto Final	(0,43, 0,462, 0,121, 1, 0, 0, 0)

Tabela 3.6 – Pontos no espaço da trajetória por pontos do ambiente de teste do Thormang3.

Foi utilizado um limite de 10.000 iterações para completar a trajetória e um raio de chegada de 1 mm. O tamanho do passo inicial utilizado foi de 0,1 mm, mesma grandeza utilizada para o incremento do passo. Na Figura 3.14 pode ser observado um exemplo com os pontos da trajetória e a trajetória executada pela ANN com um tamanho de passo de 25 mm.

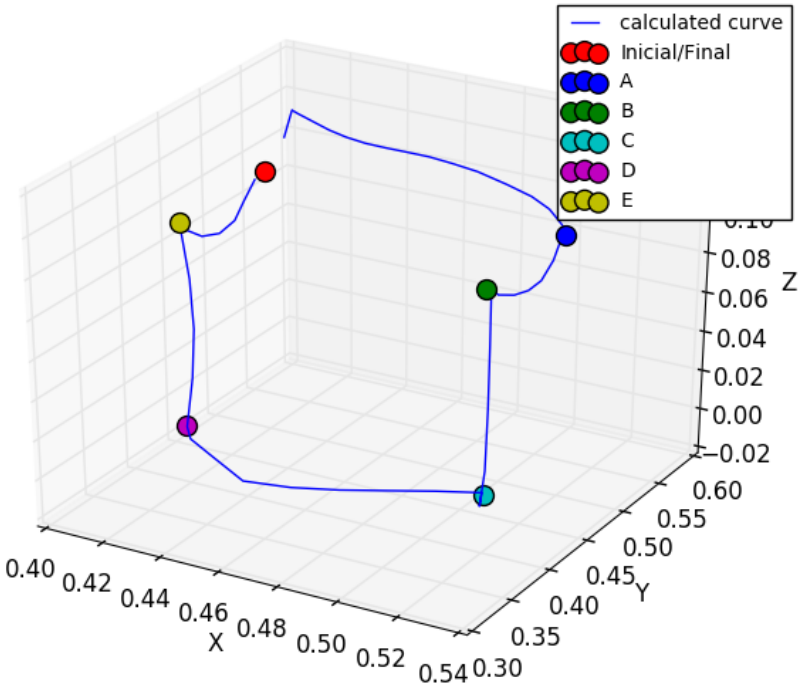


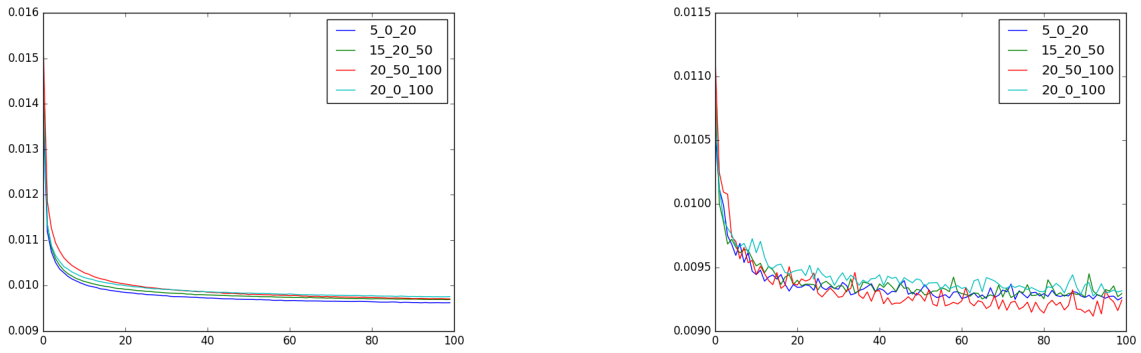
Figura 3.14 – Exemplo de realização da trajetória.

4 RESULTADOS

Ao total foram treinados 17 modelos de ANN em todos os estudos, com um *dataset* específico para cada modelo. Os resultados variaram para cada estudo e estão apresentados na sequência.

4.1 BRAÇO PLANAR DE TRÊS JUNTAS

Nas Figuras 4.1 pode-se observar a *loss* de treinamento e validação para cada um dos quatro modelos de ANN treinados.



(a) *Loss* de treinamento das ANNs.

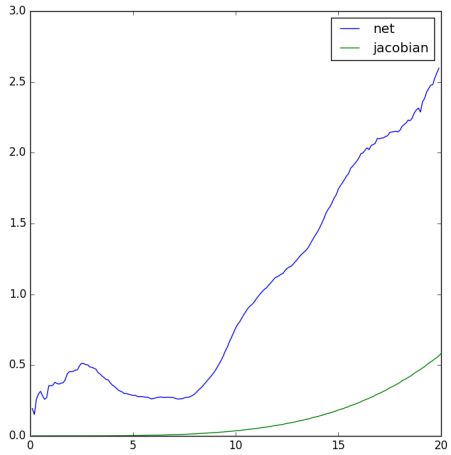
(b) *Loss* de validação das ANNs.

Figura 4.1 – *Loss* de treinamento e validação dos modelos de ANN treinados.

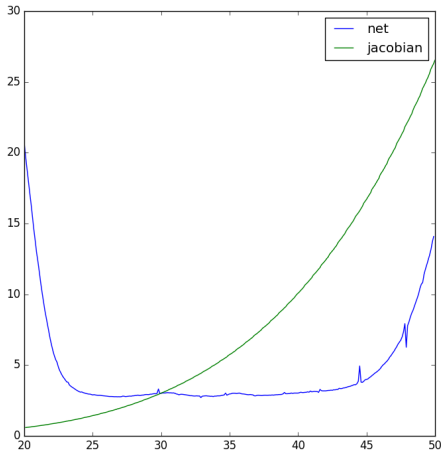
De modo geral, os testes realizados mostraram que a ANN é mais estável que a JM inversa para realizar a IK, possuindo um MSE mais baixo para grandes tamanhos de passo. Os resultados dos estudos realizados podem ser observados nas Figuras 4.2, onde é amostrado a MSE para cada tamanho de passo.

Como esperado, a ANN possui um desempenho pior que a JM inversa para pequenos tamanhos de passo. Isso fica claro na comparação do primeiro *dataset* 4.2a, onde a ANN foi mais ineficiente que a JM inversa para todo o alcance analisado. Entretanto, a ANN apresentou um desempenho superior a JM inversa para grandes tamanhos de passo, superando a JM inversa a partir de um tamanho de passo de aproximadamente 30 mm, como pode ser observado na Figura 4.2b. Conforme o tamanho do passo aumenta, o MSE tanto da ANN quanto da JM inversa aumenta exponencialmente, mas a uma taxa menor para a ANN.

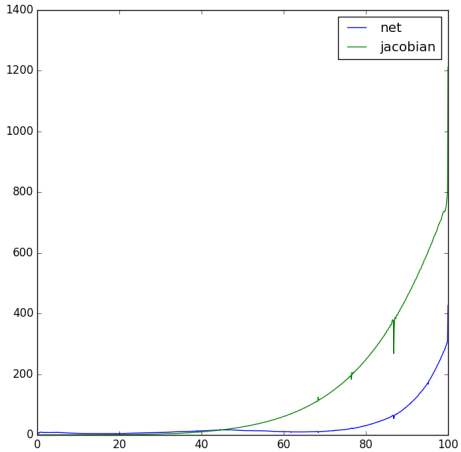
Como um teste complementar, foi comparado o comportamento de ambos os métodos quando próximos a uma singularidade. O teste consistiu em realizar uma trajetória



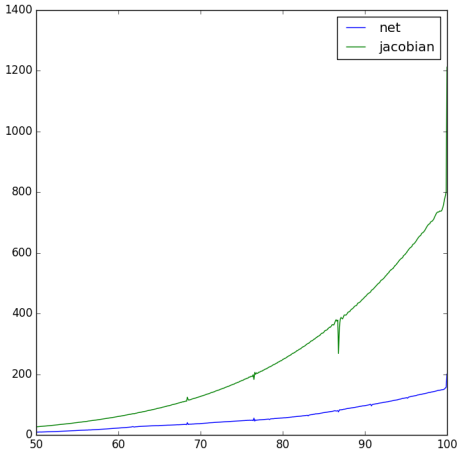
(a) Alcance de 0 a 20 mm com 5° de delta.



(b) Alcance de 20 a 50 mm com 15° de delta.



(c) Alcance de 0 a 100 mm com 20° de delta.

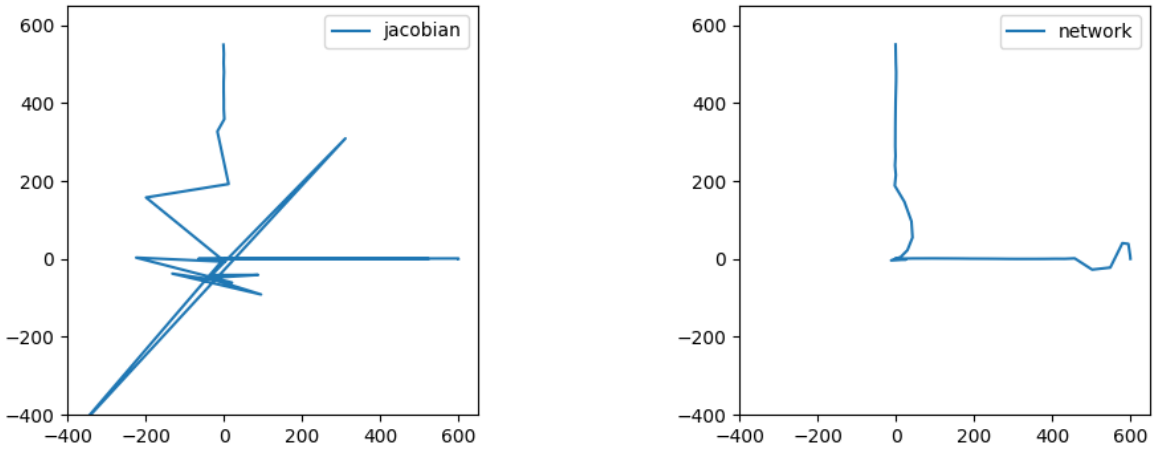


(d) Alcance de 50 a 100 mm com 20° de delta.

Figura 4.2 – Resultados obtidos para o braço planar de três juntas.

com o orgão terminal do braço saindo da posição inicial, passando pela origem e subindo em *y*, criando uma trajetória em L. Os resultados obtidos podem ser observados na Figura 4.3.

A trajetória mais suave realizada pela ANN mostra a maior estabilidade que a ANN possuiu também nas proximidades de singularidades. Matematicamente, enquanto que a JM inversa sofre com as divisões por zero na JM, a ANN consegue aproximar com maior perfeição a trajetória graças às não-linearidades das várias camadas que possui.



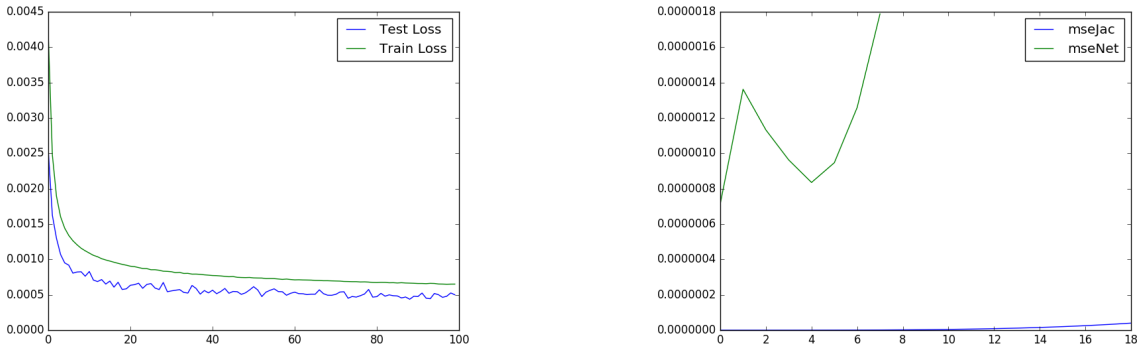
(a) IK com JM Inversa

(b) IK com ANN

Figura 4.3 – Comparação do desempenho entre a ANN e a JM inversa quando próximo a uma singularidade.

4.2 BRAÇO TRIDIMENSIONAL DE TRÊS JUNTAS

De modo geral, os testes realizados neste estudo de caso mostraram que a ANN não foi mais estável que a JM. Apesar de possuir um MSE muito baixo e conseguir realizar a trajetória proposta com eficiência, os modelos de ANN treinados não conseguiram apresentar desempenho superior ao da JM inversa. Nas Figuras 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 4.10 e 4.11 pode-se observar os resultados obtidos para cada um dos oito modelos treinados e testados com a JM inversa.

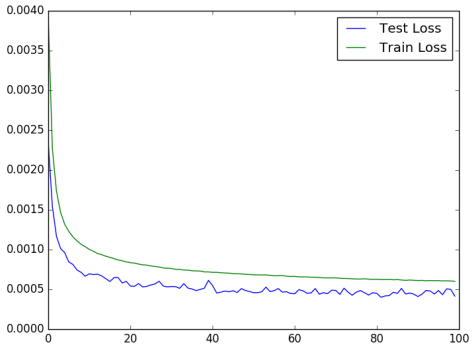


(a) *Loss* de treinamento e validação da ANN.

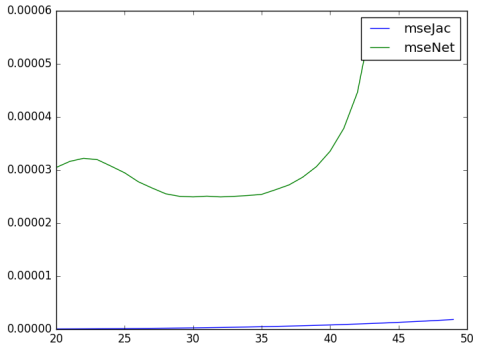
(b) Comparação da ANN com a JM Inversa.

Figura 4.4 – *Loss* da ANN e comparação com JM inversa no alcance de 0 mm a 20 mm com 5° de delta.

No primeiro conjunto de *datasets* pode-se constatar que o desempenho da ANN foi muito bom. Em 4.7 por exemplo, o MSE foi na ordem de 0,075 mm para um passo de 50

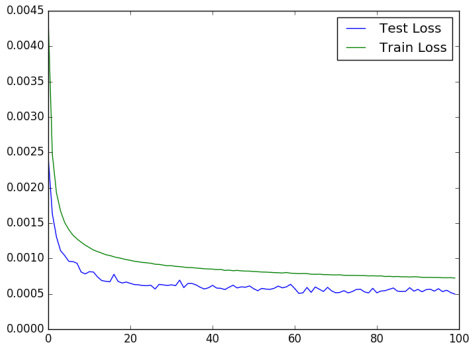


(a) *Loss* de treinamento e validação da ANN.

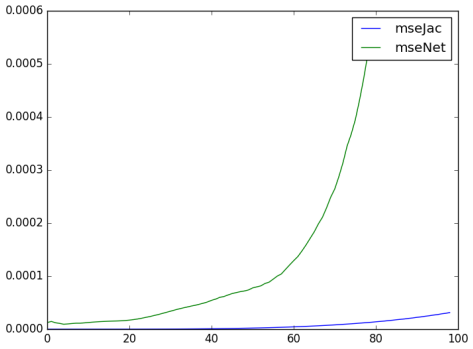


(b) Comparação da ANN com a JM Inversa.

Figura 4.5 – *Loss* da ANN e comparação com JM inversa no alcance de 20 mm a 50 mm com $15°$ de delta.

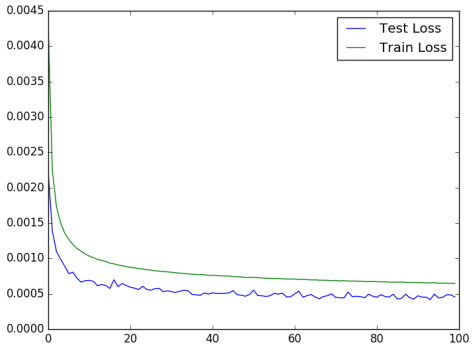


(a) *Loss* de treinamento e validação da ANN.

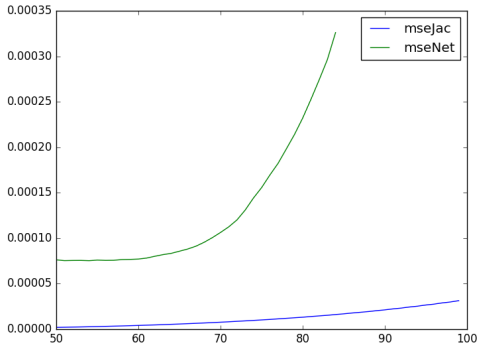


(b) Comparação da ANN com a JM Inversa.

Figura 4.6 – *Loss* da ANN e comparação com JM inversa no alcance de 0 mm a 100 mm com $20°$ de delta.

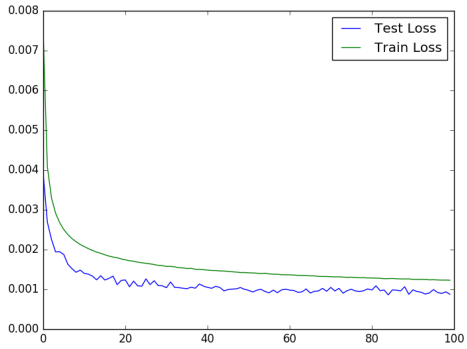


(a) *Loss* de treinamento e validação da ANN.

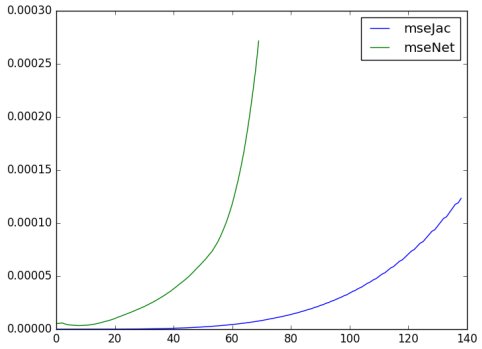


(b) Comparação da ANN com a JM Inversa.

Figura 4.7 – *Loss* da ANN e comparação com JM inversa no alcance de 50 mm a 100 mm com $20°$ de delta.

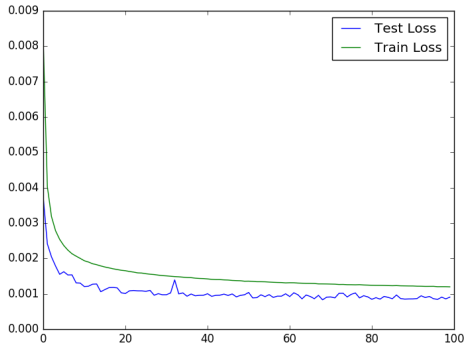


(a) *Loss* de treinamento e validação da ANN.

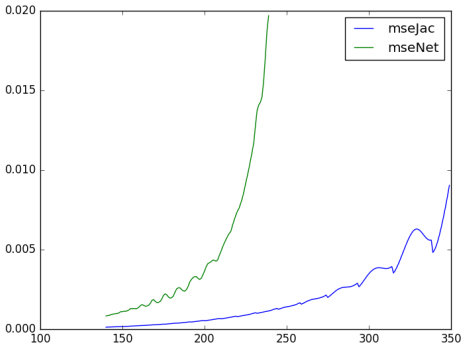


(b) Comparação da ANN com a JM Inversa.

Figura 4.8 – *Loss* da ANN e comparação com JM inversa no alcance de 0 mm a 140 mm com $5°$ de delta.

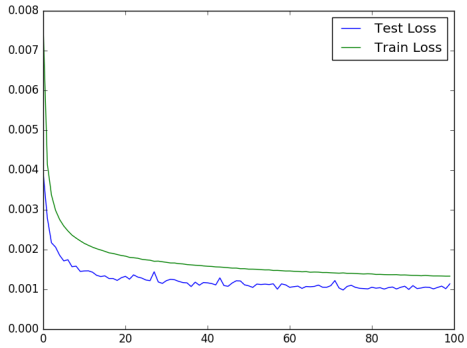


(a) *Loss* de treinamento e validação da ANN.

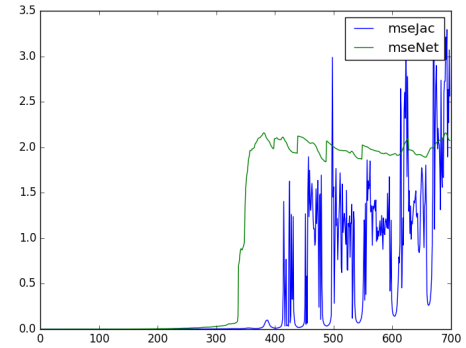


(b) Comparação da ANN com a JM Inversa.

Figura 4.9 – *Loss* da ANN e comparação com JM inversa no alcance de 140 mm a 350 mm com $15°$ de delta.

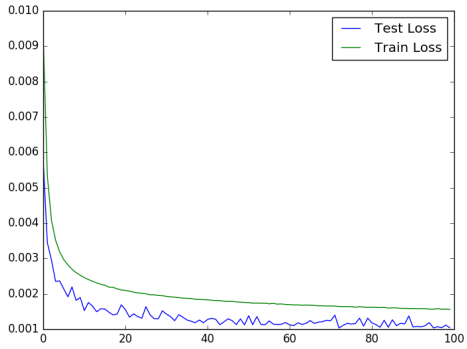


(a) *Loss* de treinamento e validação da ANN.

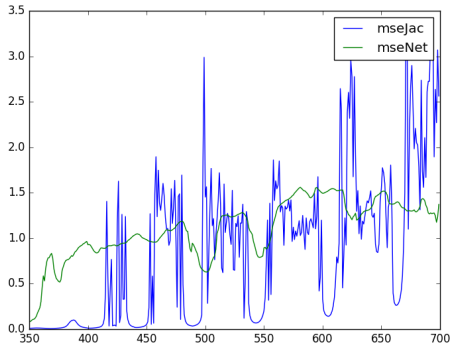


(b) Comparação da ANN com a JM Inversa.

Figura 4.10 – *Loss* da ANN e comparação com JM inversa no alcance de 0 mm a 700 mm com $20°$ de delta.



(a) *Loss* de treinamento e validação da ANN.



(b) Comparação da ANN com a JM Inversa.

Figura 4.11 – *Loss* da ANN e comparação com JM inversa no alcance de 350 mm a 700 mm com 20° de delta.

mm, enquanto que o da JM inversa foi quase zero. O MSE em todas as comparações foi similar para o menor tamanho de passo que a ANN foi treinada, mas, diferentemente do estudo anterior, a taxa de crescimento do erro conforme o aumento do tamanho do passo desta vez cresceu muito mais rapidamente para a ANN do que a JM inversa.

ANN só foi capaz de superar a JM inversa com um tamanho de passo muito grande aonde ambos apresentaram muita instabilidade, vide Figura 4.10 e 4.11. O MSE nesse caso foi tão elevado para ambos que não pode-se considerar esse alcance de tamanho de passo como um espaço que valide a proposta nesse estudo.

4.3 BRAÇO DO THORMANG3

Na Figura 4.12 pose-se observar a *loss* de treinamento e validação durante o treinamento da ANN.

Nesse estudo de caso, os resultados se aproximaram dos resultados esperados na proposta do trabalho, se assemelhando dessa forma ao primeiro estudo de caso e se afastando dos resultados do segundo estudo de caso. Os testes realizados mostraram que a ANN foi capaz de apresentar melhor desempenho que a JM inversa mesmo para pequenos tamanhos de passo e, principalmente, apresentou um aumento do MSE próximo a uma taxa linear, não exponencial como a JM apresenta. Nas Figuras 4.13, 4.14, 4.15, 4.16, 4.17 e 4.18 pode-se observar a comparação da trajetória realizada para alguns tamanhos de passo.

Na Figura 4.13 a trajetória realizada pela MJ inversa com um tamanho de passo de 1 mm é praticamente perfeita, passando por todos os pontos em um trajeto retilíneo. A ANN, em contraste, realizou uma trajetória errada e não conseguir completar a tarefa de passar por todos os pontos.

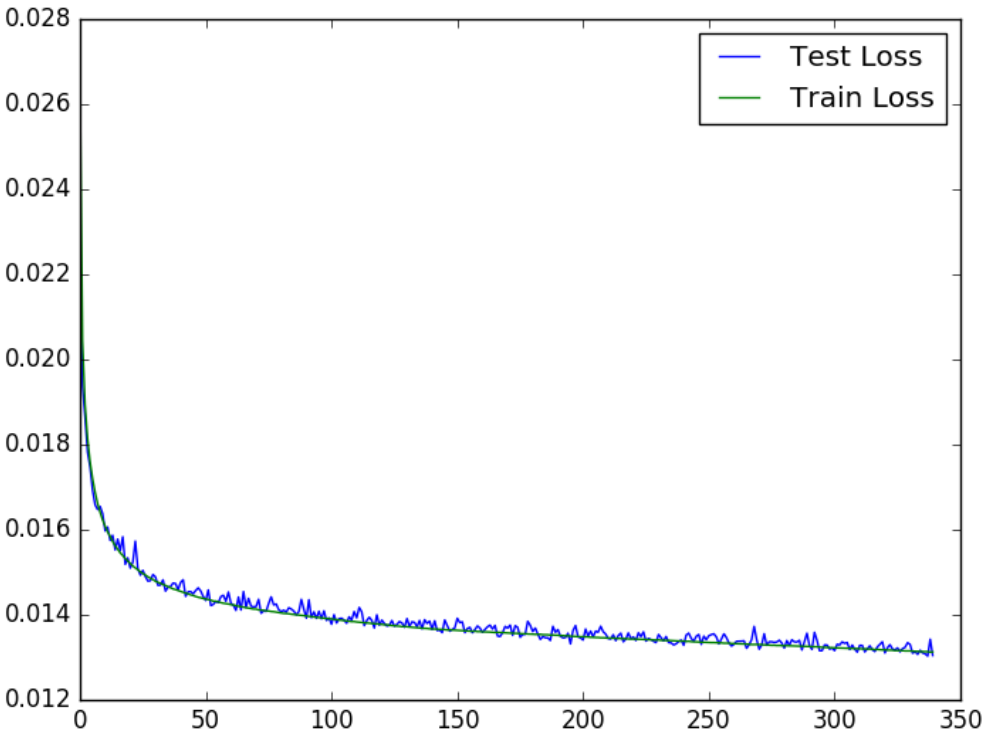
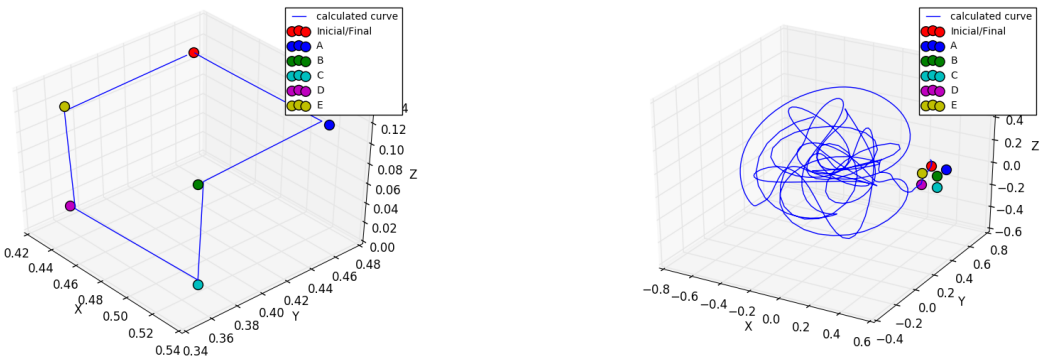


Figura 4.12 – *Loss* da ANN durante o treinamento.

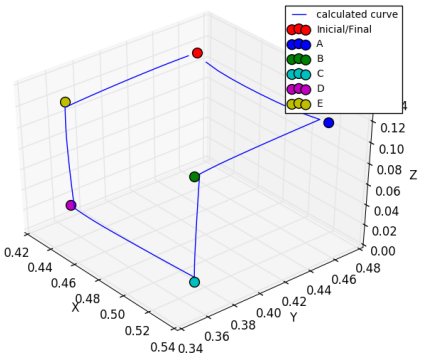


(a) Trajetória realizada com a JM inversa.

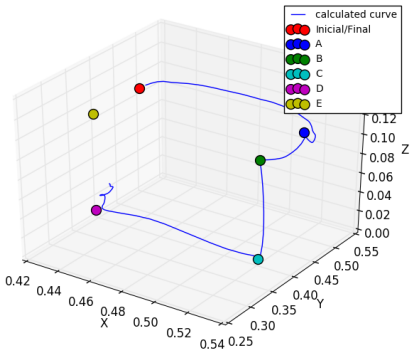
(b) Trajetória realizada com a ANN.

Figura 4.13 – Comparação da trajetória realizada com um tamanho de passo de 1 mm

Na Figura 4.14 a JM inversa começou a apresentar imperfeições na trajetória de um ponto ao outro, mas também conseguiu passar por todos os pontos e finalizar o trajeto. A ANN, por outro lado, conseguiu passar pelos pontos A, B, C e D mas não conseguiu passar pelo ponto E e chegar ao ponto final. Entretanto, já pode-se observar um aumento considerável do desempenho da ANN se comparado com o teste com 1 mm. O mesmo pode-se observar na trajetória realizada com um tamanho de passo de 25 mm. Isso pode

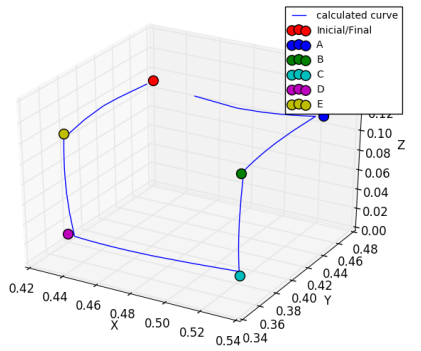


(a) Trajetória realizada com a JM inversa.

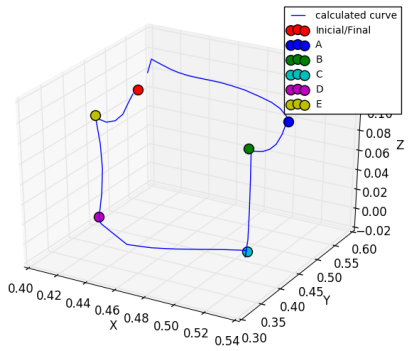


(b) Trajetória realizada com a ANN.

Figura 4.14 – Comparação da trajetória realizada com um tamanho de passo de 10 mm

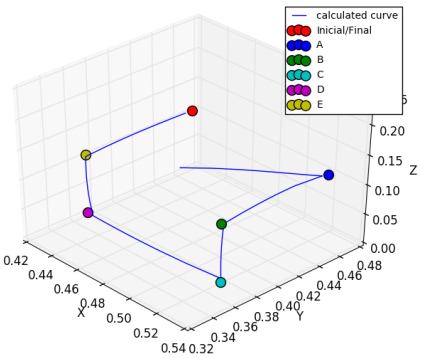


(a) Trajetória realizada com a JM inversa.

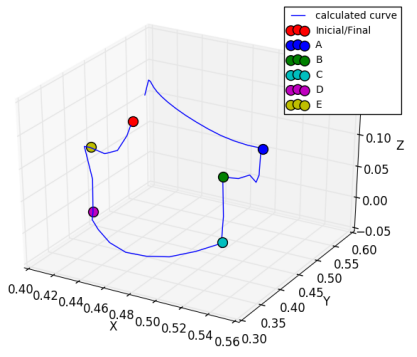


(b) Trajetória realizada com a ANN.

Figura 4.15 – Comparação da trajetória realizada com um tamanho de passo de 50 mm



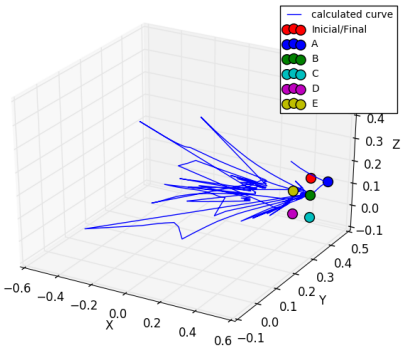
(a) Trajetória realizada com a JM inversa.



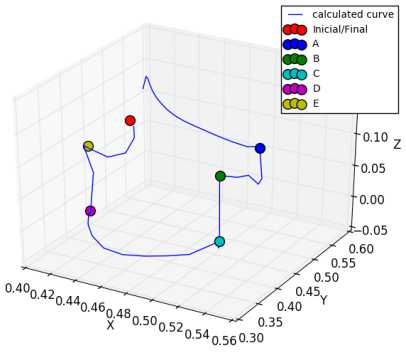
(b) Trajetória realizada com a ANN.

Figura 4.16 – Comparação da trajetória realizada com um tamanho de passo de 50 mm

ser observado na Figura 4.15.

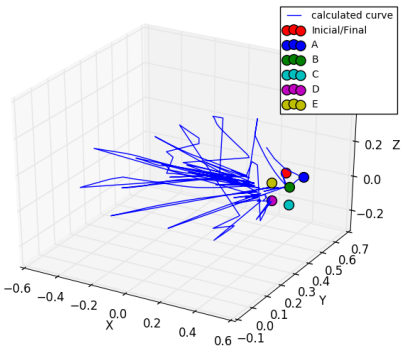


(a) Trajetória realizada com a JM inversa.

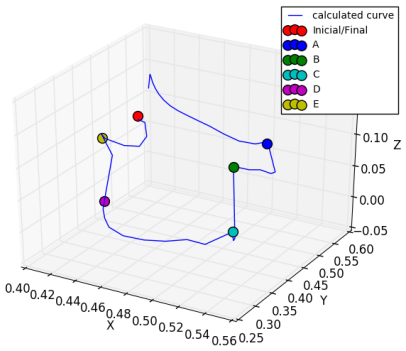


(b) Trajetória realizada com a ANN.

Figura 4.17 – Comparação da trajetória realizada com um tamanho de passo de 75 mm



(a) Trajetória realizada com a JM inversa.



(b) Trajetória realizada com a ANN.

Figura 4.18 – Comparação da trajetória realizada com um tamanho de passo de 90 mm

Na Figura 4.16 ambos conseguiram realizar a trajetória. Pode-se observar que em ambos houve uma diminuição da precisão para ir de um ponto ao outro, apesar de terem completado a trajetória. A ANN entre os pontos inicial e B e os pontos C e D apresentou uma trajetória menos retilínea que a realizada entre esses mesmos pontos para 25 mm.

Na comparação das Figuras 4.17 e 4.18 pode-se observar que o a trajetória não foi completa utilizando a JM Inversa, apresentando um desempenho pior para 90 mm que 75 mm. O desempenho da ANN se manteve praticamente idêntico para 75 mm e 90 mm, até mesmo quando comparado com o desempenho da ANN para 50 mm.

O resultado das trajetórias realizadas fica mais claro quando analisado a comparação do MSE para cada tamanho de passo que a ANN foi treinada. Todo o alcance testado pode ser observado na Figura 4.19 enquanto que um intervalo que melhor demonstra a proposta do trabalho pode ser visualizado na Figura 4.20.

Da Figura 4.19 observa-se o desempenho ruim e a incapacidade que a ANN apresentou para completar a trajetória para um tamanho de passo muito pequeno, até próximo

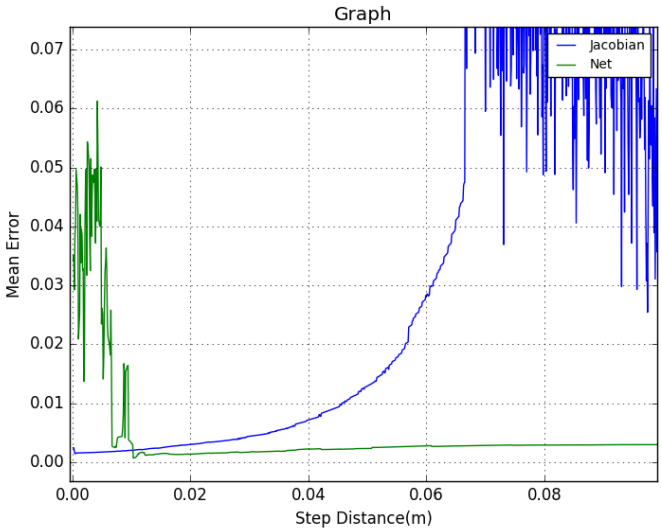


Figura 4.19 – Comparação do MSE para todo alcance de tamanhos de passo treinados.

à 10 mm. Após 10 mm, pode-se observar que o MSE da JM cresceu exponencialemente até que o tamanho do passo tornou a trajetória inviável de ser realizada, enquanto que o MSE da ANN cresceu a uma taxa muito inferior, se aproximando a uma taxa linear.

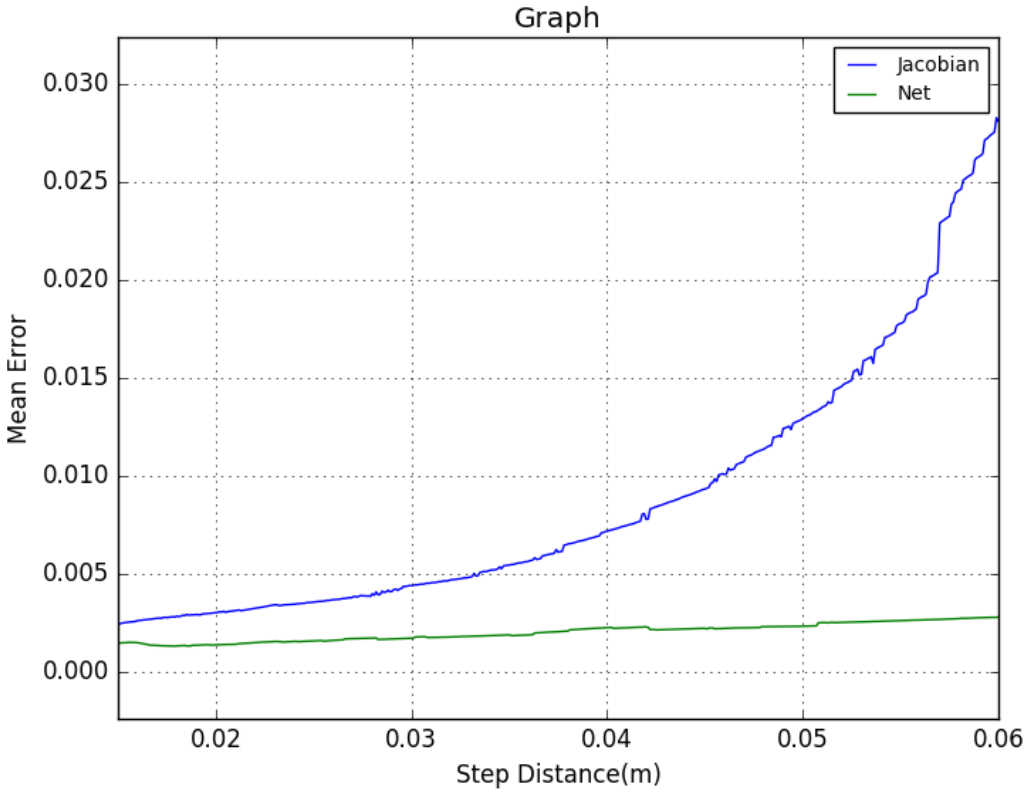


Figura 4.20 – Comparação do MSE com tamanho de passo entre 15 mm e 60 mm.

5 CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho foi realizado um estudo sobre o uso de ANNs como alternativa para ao método da JM inversa para realizar a IK. Foram realizados três estudos de caso para comparar o desempenho do método proposto com o método de DK clássico.

Vários modelos e arquiteturas de ANNs foram treinadas sobre com vários *datasets* de variados alcances para a DK. Ao total 17 *datasets* e modelos de ANNs foram criados utilizando um princípio comum e características específicas, limitadas por cada estudo. Após várias épocas de treinamento, 66.000.000 de amostras alimentadas aos modelos ao total e várias horas de desenvolvimento de código e ambientes de teste, conclui-se que a abordagem apresentou resultados promissores para dois dos três estudo de caso abordados.

Para o estudo de caso do braço tridimensional, que não apresentou bom desempenho baseado no erro médio quadrático, as características do braço podem ter influenciado. A JM quadrada que a composição do braço apresenta pode ter sido um fator determinante, já que nos outros dois estudos a composição dos braços geraram JMs não quadradas aonde faz-se o uso de algoritmos para a realização de uma pseudo inversão. Pode-se concluir ao menos que, até o momento, ANNs são capazes de desempenhar melhores resultados que o método clássico do jacobiano para resolver iterativamente o problema da IK para manipuladores cuja a composição não gera uma JM quadrada, que é a situação mais comum na prática. Essa suposição precisa ser avaliada em futuros trabalhos.

Baseado nos resultado obtidos, propõe-se o uso de ANNs não como uma alternativa de substituição ao método clássico, mas sim como um complemento. A utilização de um método híbrido entre ANN e JM inversa poderia apresentar algumas vantagens práticas.

Em um modelo híbrido a ANN poderia ser utilizada para para mover incrementalmente o braço ou manipulador com tamanhos de passo grandes, enquanto distantes do ponto de destino. Conforme ocorre a aproximação com o destino, o método convencional poderia ser utilizado para tamanhos de passo menores para uma maior precisão na chegada ao ponto de destino. Dessa forma, o cálculo da IK levaria menos iterações para ser concluído, podendo ser mais rápido ou não dependendo da configuração do robô. O uso de ANNs também proporciona a opção de usar GPU para rodá-la, possibilitando uma opção para diminuir a carga de trabalho que a complexidade computacional do método clássico demanda. Além disso, os resultados obtidos no estudo de caso do braço planar em relação à singularidades também proporciona uma maior estabilidade e eficiência para ir de um ponto no espaço à outro, uma vez que a IK com o método clássico pode não ser concluída com sucesso caso a trajetória entre os pontos passe por um ponto próximo à singularidade.

Essa proposta utilizando um sistema híbrido está sendo desenvolvido como um

trabalho futuro. Aplicado mais especificamente ao objeto de estudo de caso do Thormang3, está sendo desenvolvido um sistema de controle híbrido para realizar tarefas, tanto no ambiente de simulação quanto real. Uma primeira versão do sistema já foi desenvolvida para realizar uma tarefa no ambiente de simulação do Gazebo, mais especificamente a tarefa de resolver o jogo Torre de Hanói[1], e atualmente está sendo desenvolvida uma aplicação do sistema para o robô Thormang3 real, em parceria com o Educational Robotics Center, do Department of Electrical Engineering da National Taiwan Normal University.

[1] https://www.youtube.com/watch?v=fLpN4AC5rEA&t=212s

REFERÊNCIAS BIBLIOGRÁFICAS

AGGARWAL, L.; AGGARWAL, K.; URBANIC, R. J. Use of artificial neural networks for the development of an inverse kinematic solution and visual identification of singularity zone (s). **Procedia Cirp**, Elsevier, v. 17, p. 812–817, 2014.

ALMUSAWI, A. R.; DÜLGER, L. C.; KAPUCU, S. A new artificial neural network approach in solving inverse kinematics of robotic arm (denso vp6242). **Computational intelligence and neuroscience**, Hindawi, v. 2016, 2016.

ALOM, M. Z. et al. Handwritten bangla digit recognition using deep learning. **CoRR**, abs/1705.02680, 2017. Disponível em: <http://arxiv.org/abs/1705.02680>.

BINGUL, Z.; ERTUNC, H.; OYSU, C. Comparison of inverse kinematics solutions using neural network for 6r robot manipulator with offset. In: IEEE. **Computational Intelligence Methods and Applications, 2005 ICSC Congress on**. [S.l.], 2005. p. 5–pp.

BOTTOU, L. Large-scale machine learning with stochastic gradient descent. In: **Proceedings of COMPSTAT'2010**. [S.l.]: Springer, 2010. p. 177–186.

BRADSKI, G.; KAEHLER, A. Opencv. **Dr. Dobb's journal of software tools**, 2000.

BRAY, T. et al. Extensible markup language (xml). **World Wide Web Journal**, v. 2, n. 4, p. 27–66, 1997.

BUSS, S. R. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. **IEEE Journal of Robotics and Automation**, v. 17, n. 1-19, p. 16, 2004.

CHAPELLE, F.; BIDAUD, P. Closed form solutions for inverse kinematics approximation of general 6r manipulators. **Mechanism and machine theory**, Elsevier, v. 39, n. 3, p. 323–338, 2004.

CHIU, C. et al. State-of-the-art speech recognition with sequence-to-sequence models. **CoRR**, abs/1712.01769, 2017. Disponível em: <http://arxiv.org/abs/1712.01769>.

CRAIG, J. J. **Introduction to robotics: mechanics and control**. [S.l.]: Pearson/Prentice Hall Upper Saddle River, NJ, USA:, 2005. v. 3.

DIANKOV, R.; KUFFNER, J. Openrave: A planning architecture for autonomous robotics. **Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34**, v. 79, 2008.

DUCHI, J.; HAZAN, E.; SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. **Journal of Machine Learning Research**, v. 12, n. Jul, p. 2121–2159, 2011.

DUFFY, J. **Analysis of mechanisms and robot manipulators**. [S.l.]: John Wiley & Sons, Inc., 1980.

DUKA, A.-V. Neural network based inverse kinematics solution for trajectory tracking of a robotic arm. **Procedia Technology**, Elsevier, v. 12, p. 20–27, 2014.

Eppe, M. et al. Deep Neural Object Analysis by Interactive Auditory Exploration with a Humanoid Robot. **ArXiv e-prints**, jul. 2018.

FEATHERSTONE, R. Position and velocity transformations between robot end-effector coordinates and joint angles. **The International Journal of Robotics Research**, Sage Publications Sage CA: Thousand Oaks, CA, v. 2, n. 2, p. 35–45, 1983.

GERKEY, B.; VAUGHAN, R. T.; HOWARD, A. The player/stage project: Tools for multi-robot and distributed sensor systems. In: **Proceedings of the 11th international conference on advanced robotics**. [S.l.: s.n.], 2003. v. 1, p. 317–323.

GERKEY, B. P. et al. Most valuable player: A robot device server for distributed control. In: IEEE. **Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on**. [S.l.], 2001. v. 3, p. 1226–1231.

GLOROT, X.; BORDES, A.; BENGIO, Y. Deep sparse rectifier neural networks. In: **Proceedings of the fourteenth international conference on artificial intelligence and statistics**. [S.l.: s.n.], 2011. p. 315–323.

GOLDENBERG, A.; BENHABIB, B.; FENTON, R. A complete generalized solution to the inverse kinematics of robots. **IEEE Journal on Robotics and Automation**, IEEE, v. 1, n. 1, p. 14–20, 1985.

HA, D.; SCHMIDHUBER, J. World models. **CoRR**, abs/1803.10122, 2018. Disponível em: <http://arxiv.org/abs/1803.10122>.

HASAN, A. T. et al. Artificial neural network-based kinematics jacobian solution for serial manipulator passing through singular configurations. **Advances in Engineering Software**, Elsevier, v. 41, n. 2, p. 359–367, 2010.

HEBB, D. The organization of behavior. 1949. **New York Wiely**, v. 2, n. 7, p. 8, 2002.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural computation**, MIT Press, v. 9, n. 8, p. 1735–1780, 1997.

JOSEPH, L. Programming with ros. In: **Robot Operating System for Absolute Beginners**. [S.l.]: Springer, 2018. p. 171–236.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. **arXiv preprint arXiv:1412.6980**, 2014.

KOENIG, N. P.; HOWARD, A. Design and use paradigms for gazebo, an open-source multi-robot simulator. In: CITESEER. **IROS**. [S.l.], 2004. v. 4, p. 2149–2154.

KÖKER, R. A genetic algorithm approach to a neural-network-based inverse kinematics solution of robotic manipulators based on error minimization. **Information Sciences**, Elsevier, v. 222, p. 528–543, 2013.

KÖKER, R. et al. A study of neural network based inverse kinematics solution for a three-joint robot. **Robotics and autonomous systems**, Elsevier, v. 49, n. 3-4, p. 227–234, 2004.

LEARNING, P. M. **Python Machine Learning**. 2018. <https://pythonmachinelearning.pro/perceptrons-the-first-neural-networks/>. [Online; accessed 16-August-2018].

LEARNOPENCV. **Understanding Activation Functions in Deep Learning**. 2018. <https://www.learnopencv.com/understanding-activation-functions-in-deep-learning/>. [Online; accessed 16-August-2018].

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **nature**, Nature Publishing Group, v. 521, n. 7553, p. 436, 2015.

LIPPMANN, R. P. et al. Speech recognition by machines and humans. **Speech communication**, v. 22, n. 1, p. 1–15, 1997.

MACAUSLAND, R. The moore-penrose inverse and least squares. **Math 420: Advanced Topics in Linear Algebra**, p. 1–10, 2014.

MACROXML. **Macro XML**. 2018. <http://wiki.ros.org/xacro>. [Online; accessed 16-August-2018].

Martins, P. H.; Custódio, L.; Ventura, R. A deep learning approach for understanding natural language commands for mobile service robots. **ArXiv e-prints**, jul. 2018.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, Springer, v. 5, n. 4, p. 115–133, 1943.

MONTENEGRO, F. J. C. et al. Neural network as an alternative to the jacobian for iterative solution to inverse kinematics. In: IEEE. **2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)**. [S.l.], 2018. p. 333–338.

NASRABADI, N. M. Pattern recognition and machine learning. **Journal of electronic imaging**, International Society for Optics and Photonics, v. 16, n. 4, p. 049901, 2007.

Quadrana, M. et al. Modeling Musical Taste Evolution with Recurrent Neural Networks. **ArXiv e-prints**, jun. 2018.

QUIGLEY, M. et al. Ros: an open-source robot operating system. In: KOBE, JAPAN. **ICRA workshop on open source software**. [S.l.], 2009. v. 3, n. 3.2, p. 5.

Rabiul Islam, S. A Deep Learning Based Illegal Insider-Trading Detection and Prediction Technique in Stock Market. **ArXiv e-prints**, jul. 2018.

REDMON, J. et al. You only look once: Unified, real-time object detection. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2016. p. 779–788.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. **Psychological review**, American Psychological Association, v. 65, n. 6, p. 386, 1958.

RUSSELL, S. J.; NORVIG, P. **Artificial intelligence: a modern approach**. [S.l.]: Malaysia; Pearson Education Limited„ 2016.

SCHROFF, F.; KALENICHENKO, D.; PHILBIN, J. Facenet: A unified embedding for face recognition and clustering. **CoRR**, abs/1503.03832, 2015. Disponível em: <http://arxiv.org/abs/1503.03832>.

SCIAVICCO, L.; SICILIANO, B. **Modelling and control of robot manipulators**. [S.l.]: Springer Science & Business Media, 2012.

SHREINER, D.; GROUP, B. T. K. O. A. W. et al. **OpenGL programming guide: the official guide to learning OpenGL, versions 3.0 and 3.1**. [S.l.]: Pearson Education, 2009.

SMITH, R. Open dynamics engine. **http://www. ode. org/**, 2001.

SPONG, M. W. et al. **Robot modeling and control**. [S.l.]: Wiley New York, 2006. v. 3.

SRIVASTAVA, N. et al. Dropout: a simple way to prevent neural networks from overfitting. **The Journal of Machine Learning Research**, JMLR. org, v. 15, n. 1, p. 1929–1958, 2014.

Tang, G.; Michmizos, K. P. Gridbot: An autonomous robot controlled by a Spiking Neural Network mimicking the brain's navigational system. **ArXiv e-prints**, jul. 2018.

TEJOMURTULA, S.; KAK, S. Inverse kinematics in robotics using neural networks. **Information sciences**, Elsevier, v. 116, n. 2-4, p. 147–164, 1999.

THORMANG3, R. **Thormang3 E-Manual**. 2018. <http://emanual.robotis.com/docs/en/platform/thormang3/introduction/>. [Online; accessed 16-August-2018].

TIELEMAN, T.; HINTON, G. Lecture 6.5-rmsprop, coursera: Neural networks for machine learning. **University of Toronto, Technical Report**, 2012.

VAUGHAN, R. T.; GERKEY, B. P.; HOWARD, A. On device abstractions for portable, reusable robot code. In: IEEE. **Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on**. [S.l.], 2003. v. 3, p. 2421–2427.

WERBOS, P. J. Backpropagation through time: what it does and how to do it. **Proceedings of the IEEE**, IEEE, v. 78, n. 10, p. 1550–1560, 1990.

WIKIPEDIA. **Sigmoid Function**. 2018. <https://en.wikipedia.org/wiki/Sigmoid_function>. [Online; accessed 16-August-2018].

Zhou, T.; Wachs, J. P. Spiking Neural Networks for Early Prediction in Human Robot Collaboration. **ArXiv e-prints**, jul. 2018.

ZIKOPOULOS, P.; EATON, C. et al. **Understanding big data: Analytics for enterprise class hadoop and streaming data**. [S.l.]: McGraw-Hill Osborne Media, 2011.