UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Análise Comparativa de Técnicas para Controle de um Manipulador Robótico Utilizando o Sensor Kinect

Luiz Arthur Silva Moura

Itajubá, 6 de setembro de 2019

UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Luiz Arthur Silva Moura

Análise Comparativa de Técnicas para Controle de um Manipulador Robótico Utilizando o Sensor Kinect

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica como parte dos requisitos para obtenção do Título de Mestre em Ciências em Engenharia Elétrica.

Área de Concentração: Automação e Sistemas Elétricos Industriais

Orientador: Prof. Dr. Guilherme Sousa Bastos

6 de setembro de 2019
Itajubá

UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Análise Comparativa de Técnicas para Controle de um Manipulador Robótico Utilizando o Sensor Kinect

Luiz Arthur Silva Moura

Dissertação aprovada por banca examinadora em 12 de Julho de 2019, conferindo ao autor o título de **Mestre em Ciências em Engenharia Elétrica.**

Banca Examinadora:
Prof. Dr. Guilherme Sousa Bastos (Orientador)
Prof. Dr. Leonardo Rocha Olivi
Prof. Dr. Luiz Edival de Souza

Itajubá
2019

Luiz Arthur Silva Moura

Análise Comparativa de Técnicas para Controle de um Manipulador Robótico Utilizando o Sensor Kinect/ Luiz Arthur Silva Moura. – Itajubá, 6 de setembro de 2019-

115 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Guilherme Sousa Bastos

Dissertação (Mestrado)
Universidade Federal de Itajubá
Programa de Pós-Graduação em Engenharia Elétrica, 6 de setembro de 2019.

1. Palavra-chave1. 2. Palavra-chave2. I. Orientador. II. Universidade xxx. III. Faculdade de xxx. IV. Título

CDU 07:181:009.3

Luiz Arthur Silva Moura

Análise Comparativa de Técnicas para Controle de um Manipulador Robótico Utilizando o Sensor Kinect

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica como parte dos requisitos para obtenção do Título de Mestre em Ciências em Engenharia Elétrica.

Trabalho aprovado. Itajubá, 12 de Julho de 2019:

Prof. Dr. Guilherme Sousa Bastos
Orientador

Prof. Dr. Leonardo Rocha Olivi

Prof. Dr. Luiz Edival de Souza

Itajubá
6 de setembro de 2019

Agradecimentos

Primeiramente eu gostaria de agradecer a Deus. Pela vida e pela a oportunidade de estar aqui hoje. Gostaria de agradecer aos meus pais, que sempre me motivaram a estudar para ser alguém na vida. Gostaria também de agradecer ao meu orientador, por ter acreditado no trabalho apresentado e também no meu potencial; a Unifei pelas instalações e material de estudo. Por fim, agradecer a todos os companheiros do LRO, que sempre me apoiaram tirando dúvidas e dando sugestões, e aos amigos, em especial ao Fábio Teruo Higa pelo grande ajuda prestada na área da programação, e pela Ariane Andrade Prado, por todo apoio emocional e motivacional.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

"Quem diz que não pode ser feito, nunca deve interromper aquele que está fazendo"
(Monkey D. Luffy)

Resumo

Esse trabalho busca controlar um manipulador robótico utilizando o sensor Kinect utilizando diferentes métodos aplicados, como cinemática direta e inversa, controle por lógica *fuzzy* e equações dinâmicas com um controlador do tipo proporcional-derivativo. Além disso, este trabalho se propõe em realizar uma análise comparativa dos métodos supracitados. O sensor Kinect foi utilizado para coletar dados de posição e rotação do braço humano. Enquanto o ROS envia essas informações para o simulador V-REP, que possui os modelos dos manipuladores utilizados na simulação. Os resultados deste trabalho validam os métodos aplicados e sugerem uma melhor performance nos métodos que utilizam os dados de posição da mão.

Palavras-chaves: Kinect, Manipulador Robótico, ROS, V-Rrep, Cinemática, Fuzzy, Dinâmica

Abstract

This paper aims to control a robot arm using Microsoft Kinect with different methods applied, such as forward and inverse kinematics, fuzzy logic control, and dynamics model with a proportional-derivative controller. Moreover, this paper proposes to perform a comparative analysis of these methods. The Kinect sensor will be used to collect position and rotation data of the human arm. ROS sends the information received from the human arms to V-rep simulator, which robot models are created based on the ED-7220C robot arm. The results validate the methods applied and suggests high performance on methods that uses hand position data.

Key-words: Kinect, robot arm, ROS, V-Rep, Kinematics, Fuzzy, Dynamics

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

fps	*frames per second*
KF	*Kalman Filter*
LRO	Laboratório de Robótica
MA	*Moving Average*
RIA	*Robotic Industries Association*
ROS	*Robot Operating System*
RViz	*ROS Vizualization*
SCARA	*Selective Compliance Assembly Robot Arm*
TF	*Transform*
UNIFEI	Universidade Federal de Itajubá
V-REP	*Virtual Robot Experimentation Platform*

Sumário

1 Introdução

Com a necessidade de aumentar sua produtividade e também pela uniformidade da qualidade para com seus produtos, as indústrias estão cada vez mais investindo na automação de suas tarefas. Aliado à grande competitividade do mercado na atualidade e devido à inflexibilidade das máquinas acompanhada com custo elevado das mesmas, levaram as empresas a buscar por um ambiente de trabalho mais flexível e por um menor custo de produção. A partir disso, as empresas optaram pela utilização de robôs, que segundo a Robotic Industries Association RIA (2008-2019), sua definição é dada como[1]: "Um robô é um manipulador reprogramável e multifuncional, designado para movimentar materiais, peças, ferramentas ou dispositivos especiais, através de movimentos variáveis programado para a realização de diversas tarefas".

Aliado a este cenário, há um aumento de ferramentas para o ambiente de realidade virtual e realidade aumentada, como o sensor Kinect, óculos RIFT, Wiimote e Wiimote *plus*, entre outros. Segundo Kirner e Siscouto (2007), a realidade aumentada pode ser definida de várias maneiras, uma delas diz que ela é o enriquecimento do ambiente real com objetos virtuais, utilizando um dispositivo tecnológico, funcionando em tempo real.

Estas ferramentas acabam sendo combinadas para o ambiente da Robótica Colaborativa. Pois estas podem expandir e diversificar o controle dos robôs. Michalos et al. (2015) e Beaupre (2014-2019) apresentam que a interação dos humanos com os robôs vem aumentando gradativamente. Uma vez que essa cooperação trazem diversos benefícios, tendo em vista que o robô se sobressai em realizar atividades simples, enquanto o ser humano possui habilidades cognitivas incríveis. Por sua vez, Michalos et al. (2015) mostra que um dos primeiros tipos de implementação de robótica colaborativa é o controle remoto do robô. Rozo et al. (2016) e Cherubini et al. (2016) propõem criar um método de aprendizagem de máquinas para que o ser humano e o robô possam conviver e cooperar no mesmo espaço de trabalho. Para isso a segurança do ser humano precisa ser assegurada, preocupação esta que também pode ser vista em Zanchettin et al. (2015).

Ciente deste cenário, este projeto visa seguir os passos dos trabalhos propostos por Pedro (2013), Cavalcante (2012) Grushko e Bobovsky (2016),Nguyen et al. (2017), Pagi e Padmajothi (2017), Silva et al. (2012), Ligutan et al. (2017), Espiau e Boulie (1998), entre outros tratados na bibliografia deste trabalho, com o objetivo de aplicar a realidade aumentada proporcionada pelo Kinect em um manipulador robótico, para movimentar em tempo real de acordo com os movimentos exercidos por um ser humano frente ao sensor. Para isso foi utilizado a plataforma *Robot Operating System* (ROS) e o simulador *Virtual*

[1] Traduzido do seguinte trecho:"a programmable, mechanical device used in place of a person to perform dangerous or repetitive tasks with a high degree of accuracy"

Robot Experimentation Platform (V-REP) como ferramentas de comunicação e aplicação dos resultados.

Para a realização deste trabalho, faz-se necessário um conhecimento sobre a cinemática, dinâmica e controle de manipuladores robóticos. Este conhecimento pode ser visto nos trabalhos de Fu, Gonzalez e Lee (1987), Nilsson (2009) DeWolf (2013), Espiau e Boulie (1998) e Abdallah, Bouteraa e Rekik (2016). Muito importante também foi verificar métodos para melhorar o sinal resultante da cinemática aplicada. Para isso foi verificado técnicas de filtros Digitais para eliminar ruído do sinal, bem como problemas de *outliers*. Técnicas estas que podem ser vistas nos trabalhos de Smith (1999), Haykin (2001), Faragher (2012), Thrun (2011) e Ting, Theodorou e Schaal (2007). Juntamente verificado técnicas de lógica fuzzy, como forma de ampliar as opções de controle do sistema.Como encontradas em Junior et al. (2016), Zadeh (1965), Leekwijck e Kerre (1999) e Zhang (2010).

Este trabalho pode-se considerar importante pelo fato de apresentar diversas técnicas presentes no ambiente de manipuladores, também trazendo diversas aplicações para área da robótica, sendo possível utilizar a realidade aumentada para a exploração de regiões de risco, tele cirurgia, treinamento de máquinas, aplicações livres, entre outras aplicações relacionadas à robótica cooperativa.

1.1 Objetivo Geral

Este trabalho tem como objetivo geral, controlar um manipulador robótico reproduzindo os movimentos do corpo humano e enviando como comandos através do sensor Kinect utilizando diferentes técnicas, sendo elas cinemática direta, cinemática inversa, lógica *fuzzy* e dinâmica de manipuladores. Realizando também uma análise comparativa entre estas técnicas.

1.2 Objetivos Específicos

Os itens abaixo servirão como engrenagens importantes para a realização do objetivo principal deste trabalho:

- Criar um uma ponte para transmissão de dados do Kinect para o V-REP;
- Criar conjunto de dados padronizados para serem utilizados em análise;
- Criar um modelo de um manipulador no simulador V-REP;
- Implementar um algoritmo de controle do manipulador, mediante a cinemática direta e inversa, dinâmica e lógica *fuzzy*;

- Aplicar e comparar técnicas quanto a sua capacidade de seguir os movimentos;
- Reproduzir o algoritmo criado no modelo construído.

1.3 Organização do trabalho

O trabalho em questão apresenta inicialmente a fundamentação teórica utilizada no capítulo 2. Composta pelas teorias de Robótica de Manipuladores, Sensor Kinect, ROS, Filtros Digitais, Controlador *fuzzy* e sobre o simulador V-REP. Em seguida é mostrado no capítulo 3 a metodologia implementada, juntamente com os trabalhos relacionados. Terminando com os Resultados e Análise de dados no capítulo 4 e com a Conclusão no capítulo 5.

2 Fundamentação Teórica

Neste Capítulo será mostrado toda a teoria utilizada para a realização deste trabalho. Iniciando pela teoria sobre manipuladores, presente na seção 2.1. Em seguida será abordado a plataforma ROS em 2.2. Na seção 2.3 será apresentada características sobre o dispositivo Kinect, enquanto na seção 2.4 são apresentados alguns filtros digitais. Adiante é tratado o método *fuzzy* presente na seção 2.5. Finalizando com algumas característica do simulador V-REP na seção 2.6.

2.1 Robôs Fixos - Manipuladores

Para Fu, Gonzalez e Lee (1987), um manipulador industrial consiste em diversos elos conectados por juntas, podendo ser juntas de revolução ou prismáticas. Este também precisa possuir inteligência, ser reprogramado, possuir sensores para medição e controle, e atuadores para efetuar a transformação de energia, sendo ela elétrica, hidráulica ou pneumática, para a energia mecânica.

As classificações dos manipuladores podem ser realizadas de diversas formas, porém o método mais usual para classificação é com relação ao tipo de junta, mais precisamente as três primeiras juntas próximas à base do manipulador. O primeiro caso é o manipulador do tipo cartesiano, tendo este nome por utilizar três juntas prismáticas orientadas em uma direção diferente, se assemelhando ao plano cartesiano. Como pode ser visto na figura 1.

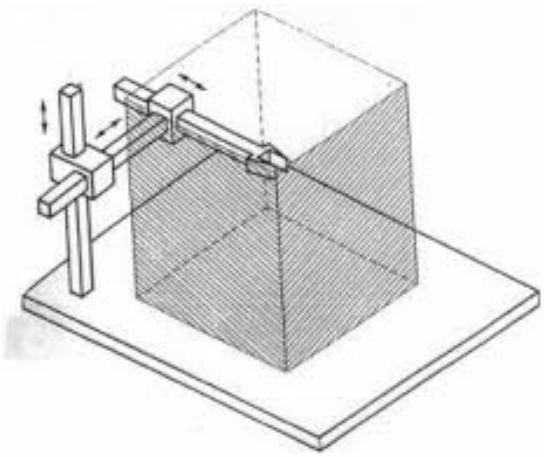

Figura 1 – Robô Cartesiano e seu volume de trabalho.
Fonte: Scopel e Barcelos (2017).

O segundo caso são os manipuladores cilíndricos, utilizando uma junta prismática, uma junta de revolução e outra junta prismática. Como pode ser visualizado na figura 2.

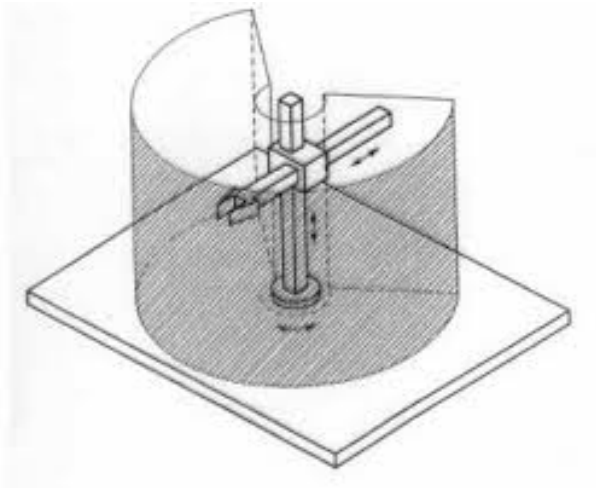

Figura 2 – Robô Cilíndrico e seu volume de trabalho.
Fonte: Scopel e Barcelos (2017).

O terceiro tipo de manipulador são os manipuladores esféricos, que possuem 2 juntas de revolução e a terceira junta prismática. Como pode ser visto na figura 3.

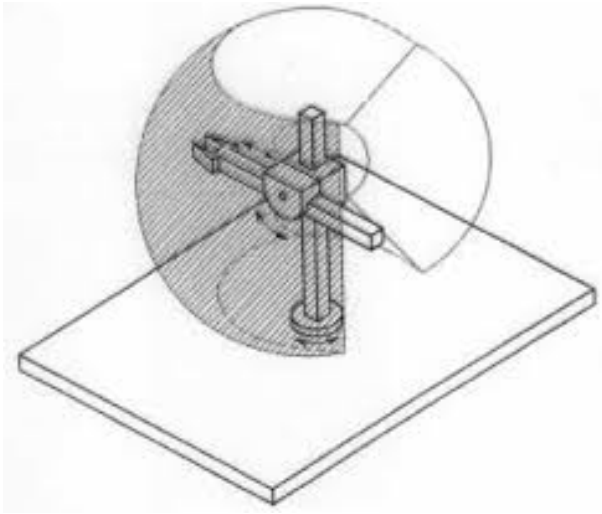

Figura 3 – Robô Esférico e seu volume de trabalho.
Fonte: Scopel e Barcelos (2017).

Há também os robôs SCARA, que assim como os esféricos possuem duas juntas de rotação e um prismática. Porém a sua disposição das juntas ocorrem de maneira diferente. Como pode ser visto na figura 4.

E o último tipo de manipulador é o manipulador antropomórfico ou manipulador de revolução. Que utiliza somente juntas de revolução. Como pode ser visto na figura 5.

Essa classificação está intimamente relacionada ao seu Volume de Trabalho, que representa o volume espacial em que a extremidade livre[1] possa operar. Se o manipulador

[1] Extremidade livre é a região do manipulador em que se coloca as ferramentas para poder realizar as tarefas.

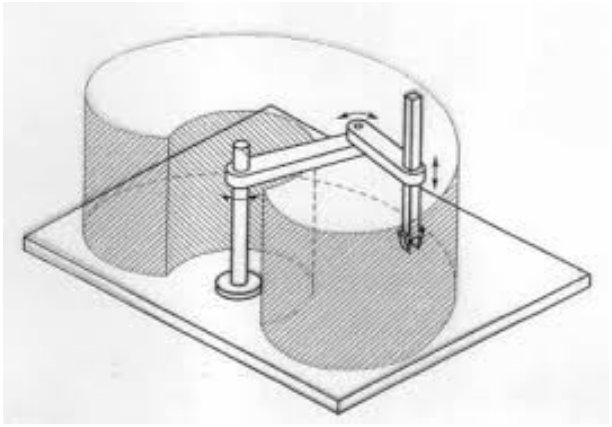

Figura 4 – Robô SCARA e seu volume de trabalho.
Fonte: Scopel e Barcelos (2017).

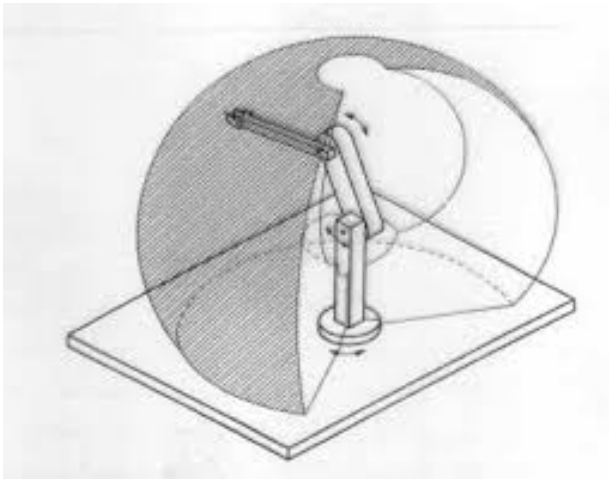

Figura 5 – Robô de Revolução e seu volume de trabalho.
Fonte: Scopel e Barcelos (2017).

precisar realizar alguma tarefa fora dessa região espacial, ele não conseguirá alcançar e, portanto, não poderá cumprir a tarefa requisitada.

2.1.1 O problema da Cinemática

A cinemática de um manipulador robótico lida com o estudo da geometria de movimento, mediante uma referência fixa sem relacionar torque e momentos de inércia que influenciam no movimento. Além disso a cinemática se preocupa com a descrição analítica do deslocamento do robô em função do tempo, em particular a relação entre as juntas, e a postura da extremidade do manipulador. (FU; GONZALEZ; LEE, 1987)

Dessa forma duas perguntas podem ser feitas em relação a este problema:

- **1)** Dado um manipulador, cuja as variáveis das juntas é dado por q(t)=(q1(t), q2(t),...,qn(t)), sendo n o número de graus de liberdade, e também de acordo com

os parâmetros dos elos do manipulador. Qual é a posição da extremidade do manipulador?

- **2)** De acordo com uma posição e orientação desejada para a extremidade do manipulador, e também dado os parâmetros dos elos e das juntas, é possível o manipulador alcançar a posição e a orientação desejada? Se sim, quantas configurações diferentes existem do manipulador para satisfazer a condição?

A pergunta 1 refere-se à cinemática direta, enquanto a 2 se refere à cinemática inversa.

Tratando inicialmente o problema da cinemática direta, na visão de Fu, Gonzalez e Lee (1987), este pode ser reduzido a encontrar a matriz de Transformação que relaciona a coordenada da base do manipulador com a coordenada de referência. Uma matriz 3x3 de rotação é utilizada para descrever as operações rotacionais do manipulador, as coordenadas homogêneas são utilizadas para representar os vetores de posição em um espaço tridimensional, dessa forma a matriz de rotação será ampliada para uma matriz de transformação homogênea 4x4, para incluir as operações de translação. Essa representação matricial foi utilizada inicialmente por Denavit e Hartenberg em 1955. A equação 2.1 representa a configuração da matriz de Transformação.

$$T = \begin{bmatrix} R_{3\times3} & p_{3\times1} \\ f_{1\times3} & 1 \end{bmatrix} = \begin{bmatrix} Matriz\_de\_Rotação & Vetor\_de\_Posição \\ Perspectiva\_de\_Transformação & Escala \end{bmatrix} \quad (2.1)$$

Para explicar a equação 2.1, além do vetor de posição **p** e da perspectiva de transformação **f**, é necessário apresentar o conceito da matriz de rotação. Ela será responsável por operar em um vetor de posição em um espaço Euclidiano tridimensional, e mapear suas coordenadas em um sistema de coordenadas rotacionais OUVW (na parte fixa do manipulador), até a coordenada de referência OXYZ (na extremidade da ferramenta de trabalho do manipulador). Na figura 6 é possível notar estes dois sistemas de coordenadas. Enquanto o sistema OXYZ é um sistema fixo no espaço tridimensional, o sistema de coordenadas OUVW é rotacionado para que este alcance a posição de referência.

Assumindo que o vetor de coordenadas do sistema OXYZ e do sistema OUVW sejam respectivamente (i_x, j_y, k_z) e (i_u, j_v, k_w), um ponto **p** no espaço também pode ser representados por estas coordenadas com suas respectivas sistemas de coordenadas, ou seja:

$$p_{uvw} = (p_u, p_v, p_w)^T \quad e \quad p_{xwz} = (p_x, p_y, p_z)^T \quad (2.2)$$

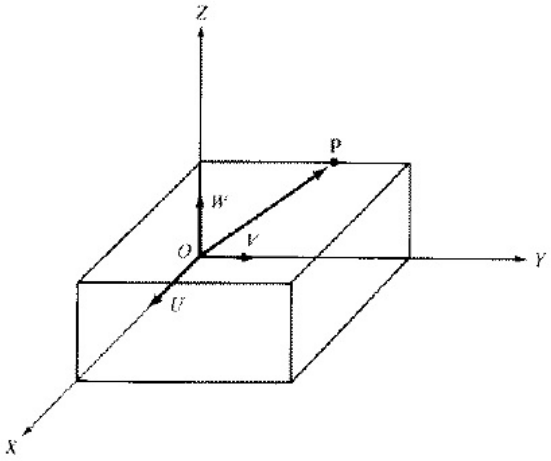

Figura 6 – Eixo de Coordenadas OXYZ e OUVW.
Fonte: Fu, Gonzalez e Lee (1987).

Assim, no problema da cinemática direta, necessita-se encontrar uma matriz de rotação **R** que transforme as coordenadas do ponto p_{uvw} no ponto p_{xyz}, ou seja:

$$p_{xwz} = Rp_{uvw} \tag{2.3}$$

Lembrando que os componentes de um vetor podem ser expressas como:

$$p_{uvw} = p_u i_u + p_v j_v + p_w k_w \tag{2.4}$$

Ao utilizar a definição de produto escalar pode ser obtido a seguinte relação entre p_{xyz} e p_{uvw}:

$$\begin{aligned} p_x &= i_x \cdot p = i_x \cdot i_u p_u + i_x \cdot j_v p_v + i_x \cdot k_w p_w \\ p_y &= i_y \cdot p = j_y \cdot i_u p_u + j_y \cdot j_v p_v + j_y \cdot k_w p_w \\ p_z &= i_z \cdot p = k_z \cdot i_u p_u + k_z \cdot j_v p_v + k_z \cdot k_w p_w \end{aligned} \tag{2.5}$$

e isso na forma matricial pode ser visto como:

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} i_x \cdot i_u & i_x \cdot j_v & i_x \cdot k_w \\ j_y \cdot i_u & j_y \cdot j_v & j_y \cdot k_w \\ k_z \cdot i_u & k_z \cdot j_v & k_z \cdot k_w \end{bmatrix} \begin{bmatrix} p_u \\ p_v \\ p_w \end{bmatrix} \tag{2.6}$$

De maneira similar, é possível encontrar a rotação inversa também. Ou seja:

$$p_{uvw} = Qp_{xyz} \tag{2.7}$$

Possuindo a seguinte forma matricial:

$$\begin{bmatrix} p_u \\ p_v \\ p_w \end{bmatrix} = \begin{bmatrix} i_u \cdot i_x & i_u \cdot j_y & i_u \cdot k_z \\ j_v \cdot i_x & j_v \cdot j_y & j_v \cdot k_z \\ k_w \cdot i_x & k_w \cdot j_y & k_w \cdot k_z \end{bmatrix} \begin{bmatrix} p_u \\ p_v \\ p_w \end{bmatrix} \tag{2.8}$$

Como uma das propriedades do produto escalar é eles serem comutativos, isso mostra que:

$$\begin{aligned} Q = R^{-1} = R^T \\ QR = R^T R = R^{-1} R = I \end{aligned} \tag{2.9}$$

Sendo I a matriz identidade.

A partir disso, Denavit e Hartenberg propuseram um método para descrever as relações de rotação e translação das juntas de maneira sistemática, fazendo com que cada conjunto de junta/elo seja relacionado com o conjunto anterior. Consequentemente, é possível representar um sistema cartesiano de coordenadas para cada junta individualmente, adotando o número de juntas como sendo o número de graus de liberdade do manipulador. O sistema de coordenadas de cada junta ($OXYZ_i$) são determinados a partir destas três regras (FU; GONZALEZ; LEE, 1987).

- 1) O eixo z_{i-1} se encontra em direção ao sentido de movimento da junta i;
- 2) O eixo x_i é normal ao eixo z_{i-1} e aponta para longe do eixo z_{i-1};
- 3) O eixo y_i é definido por meio da regra da mão direita.

Dessa forma é possível definir os eixos de coordenadas de todas as juntas. Para a coordenada de origem ($x_0, y_0, z_0 \quad e \quad O_0$), este pode ser posicionado em qualquer lugar da base do manipulador, mas ainda assim é necessário que as regras supracitadas sejam seguidas corretamente. Nessa representação, quatro parâmetros são utilizados para descrever completamente qualquer junta, seja ela prismática ou de revolução. E estes parâmetros são:

- θ_i o ângulo entre os eixos x_{i-1} e x_i medido em torno de z_{i-1}, é variável quando a junta é rotativa;
- d_i distância ao longo de z_{i-1} de O_{i-1} até a interseção dos eixos x_i e z_{i-1}. Se a junta for prismática d_i é uma variável;
- a_i distância na direção de x_i entre a interseção dos eixos x_i e z_{i-1} até O_i;

- α_i o ângulo entre z_{i-1} e z_i em torno de x_i.

A figura 7 mostra a aplicação das regras para encontrar cada parâmetro de DH.

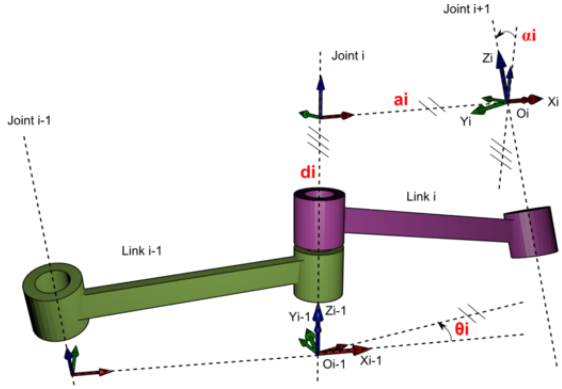

Figura 7 – Aplicação das regras para a obtenção dos parâmetros de DH.
Fonte: **??**).

A partir destas orientações e da figura 7, é possível encontrar as matrizes de transformação Homogênea H vista na equação 2.10.

$$H_{i-1}^{i} = Rot_{z_{i-1},\theta} T_{z_{i-1},d_i} T_{x_i,a_i} Rot_{x_i,\alpha_i} \tag{2.10}$$

A equação 2.10 expandindo para forma matricial é vista como:

$$\begin{bmatrix} c_{\theta_i} & s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_{\theta_i} & -c_{\alpha_i} s_{\theta_i} & s_{\alpha_i} s_{\theta_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\alpha_i} c_{\theta_i} & -s_{\alpha_i} c_{\theta_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.11}$$

Levando em conta que $c_{\theta_i} = cos(\theta_i)$ e $s_{\alpha_i} = sen(\alpha_i)$. Assim, a matriz de Transformação resultante é o produtório das matrizes de transformação homogêneas de cada junta, ou seja:

$$T_0^n = H_0^1 \cdot H_1^2 ... \cdot H_{n-1}^n \tag{2.12}$$

Encontrando a matriz homogênea está sendo encontrando a relação de todas as juntas para obter a posição global da extremidade do manipulador.

Como a maioria dos manipuladores robóticos são operados de tal forma a utilizar coordenadas globais, o controle de posição e orientação da extremidade livre do robô se torna mais interessante fazendo uso da cinemática inversa. De acordo com Fu, Gonzalez e Lee (1987) e Nilsson (2009), a cinemática inversa é utilizada quando se deseja encontrar

os parâmetros das juntas de um manipulador, quando se possui uma posição absoluta de referência para a extremidade livre do manipulador. Ou seja, precisa-se encontrar os valores correspondentes dos ângulos das juntas q_1 até q_n que satisfaça a posição dada.

Para resolver esse problema, Nilsson (2009) apresenta duas categorias de solução. A solução analítica e a solução iterativa. Na solução analítica procura-se encontrar todas as possíveis soluções para este sistema. Se o sistema possuir número de graus de liberdade suficientes para cobrir o espaço de trabalho, haverá soluções finitas para o sistema. Caso tenha poucos graus de liberdade, o sistema pode não ter solução. E se caso exista mais do que o suficiente, o sistema possui infinitas soluções, sendo comumente categorizado como robô redundante.

Na solução iterativa, uma das suas soluções populares e utilizar a matriz Jacobiana, que é uma aproximação linear de uma função diferenciável próxima ao ponto desejado. Neste trabalho será abordado somente a solução iterativa, por se tratar do método utilizado no mesmo. De maneira semelhante à solução analítica, na solução iterativa existe uma função que representa cada junta, porém ao invés dessas funções representarem a posição relativa, estas equações só dizem respeito ao movimento da extremidade livre do manipulador. Porém, pelo fato das funções serem não lineares, faz-se necessário uma aproximação linear, derivada da posição relativa, para descrever um movimento parcial da extremidade livre. A partir dessas aproximações, a matriz Jacobiana pode ser expressada como visto em 2.13:

$$J(q) = \frac{\delta p}{\delta q} \tag{2.13}$$

Cada coluna de J descreve uma mudança de posição aproximada da extremidade livre do manipulador, tanto em questões lineares quanto angulares, quando alterado o valor escalar da junta. Assim, se extrapolar essa situação para o conjunto escalar para todas as juntas $\Delta\theta$ e multiplicar pela matriz Jacobiana, é possível encontrar uma mudança aproximada da posição δP da extremidade livre, ou seja:

$$\delta p = J(q)\delta q \tag{2.14}$$

Porém, para encontrar os valores das juntas, basta isolar δq da equação 2.14, tendo assim a seguinte equação:

$$\delta q = J(q)^{-1}\delta p \tag{2.15}$$

Assim, a equação 2.15 diz que a partir de um manipulador robótico com n graus de liberdade q e uma garra P, uma matriz Jacobiana pode ser criada a partir da equação 2.14 e os valores aproximados $\delta\theta$ podem ser encontrados através da equação 2.15.

A partir deste conceito, diversos métodos foram criados para resolver este problema, um deles sendo o mínimos quadrados amortecido, do inglês, *Damped Least Squares* (DLS). Tendo o foco em eliminar e reduzir algumas singularidades da matriz de Jacobiana, além de encontrar um valor que estabilize o δq, o método também chamado de algoritmo de Levenberg-Marquardt busca encontrar o menor vetor de δq que minimize a equação 2.16:

$$||J\delta\theta - \vec{e}||^2 - \lambda^2||\delta\theta||^2 \tag{2.16}$$

em que $\vec{e}$ é o vetor da posição, quando este está além do alcance do manipulador e λ é um valor real constante e diferente de 0. Ao reescrever a equação 2.16 encontra-se a seguinte fórmula:

$$|| \begin{pmatrix} J^T \\ \lambda I \end{pmatrix} \delta q - \begin{pmatrix} \vec{e} \\ 0 \end{pmatrix} || \tag{2.17}$$

Expandindo a equação 2.17 tem-se:

$$\begin{pmatrix} J \\ \lambda I \end{pmatrix}^T \begin{pmatrix} J \\ \lambda I \end{pmatrix} \delta q = \begin{pmatrix} J \\ \lambda I \end{pmatrix}^T \begin{pmatrix} \vec{e} \\ 0 \end{pmatrix} \tag{2.18}$$

Dessa forma, ao isolar δq é obtido a seguinte equação:

$$\delta q = J^T(J^T + (\lambda I)^2)^{-1}\vec{e} \tag{2.19}$$

A partir da equação 2.19, pode-se observar em primeira instância a importância do conjunto identidade λI, pois este termo garante que a inversa exista, tendo a ressalva deste não dar a matriz exata e sim uma aproximação dela.

Além disso, é importante ressaltar a escolha do valor da constante λ, uma vez que se ele for um valor muito pequeno, acarretará em movimentos indesejados de maneira semelhante ao método da pseudo-inversa, também abordado por Nilsson (2009). Porém caso o valor seja muito alto, este apresentará um resultado de δq menor, garantindo que o movimento seja executado de maneira suave, porém isso demandará um maior número de iterações. Portanto, é importante encontrar um valor de λ que garanta uma rápida solução e que não produza movimentos indesejados.

2.1.2 Dinâmica de Manipuladores

Quando se fala da dinâmica de manipuladores robóticos, está sendo levado em consideração a influência da força (ou torque) aplicada à estrutura mecânica de todo o

sistema. Bem como os efeitos da inércia, da gravidade e de forças como a centrífuga e a *coriolis*. Assim, de acordo com Fu, Gonzalez e Lee (1987), Abdallah, Bouteraa e Rekik (2016), Espiau e Boulie (1998) e DeWolf (2013) a expressão de *Euler-Lagrange* para a representação da dinâmica de um manipulador é dado como:

$$\tau(t) = M(q(t))\ddot{q}(t) + h(q(t), \dot{q}(t)) + G(q(t)) \tag{2.20}$$

Sendo que:

- τ(t) é o vetor n×1 de torque aplicado nas juntas;
- q(t) é o vetor n×1 de juntas do manipulador;
- $\dot{q}$(t) é o vetor n×1 velocidade das juntas;
- $\ddot{q}$(t) é o vetor n×1 aceleração das juntas do manipulador;
- M(q(t)) é a matriz n×n que relaciona a inércia do sistema com a aceleração;
- h(q(t),$\dot{q}$(t)) é uma matriz n×1 não linear que representa a força centrífuga e a força inercial de coriolis;
- G(q(t)) é o vetor n×1 que representa a força da gravidade no sistema.

Para transformar a equação 2.20 de modo a ser aplicável em programação e também ser possível controlar o sistema, é necessário levar em consideração os conceitos de inércia, energia cinética e energia potencial gravitacional.

Iniciando pela inércia, a ideia do sistema possuir massa, faz com que o sistema tenha dificuldades em se movimentar para um determinado ponto, pois a inércia cria uma involuntariedade ao manipulador em responder prontamente a força aplicada. Dessa forma, faz-se necessário levar em consideração essa força, para isso deve-se considerar o centro de massa de cada elo do manipulador, conforme a figura 8.

O centro de massa de cada elo, se este estiver com a densidade uniforme, seu centro de massa está no centroide. No caso da figura 8, com juntas rotativas no eixo X-Z é dado por:

$$cdm_0 = \begin{pmatrix} \frac{Lcos(q0)}{2} \\ 0 \\ \frac{Lsen(q0)}{2} \end{pmatrix} \quad e \quad cdm_1 = \begin{pmatrix} \frac{Lcos(q1)}{2} \\ 0 \\ \frac{Lsen(q1)}{2} \end{pmatrix} \tag{2.21}$$

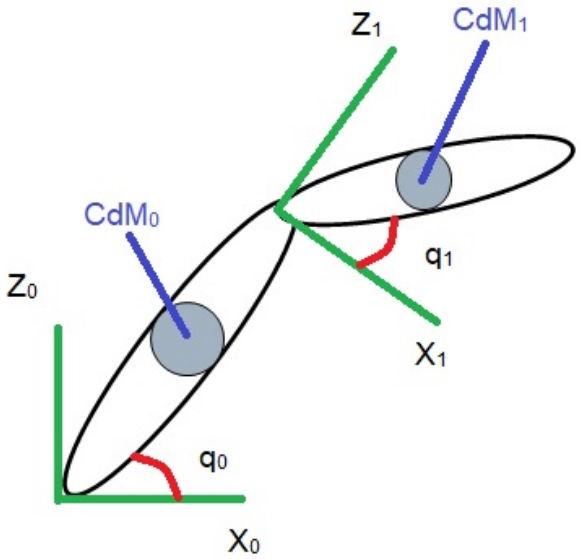

Figura 8 – Modelo de manipulador robótico com duas juntas e seus centros de massa.
Fonte: DeWolf (2013).

Vale ressaltar que essa matriz se altera caso a rotação é feita em eixos diferentes. A partir disso é possível encontrar o Jacobiano do centro de massa, conforme pode ser visto em DeWolf (2013) e em Espiau e Boulie (1998).

$$J_{cdm\_i} = \frac{\delta p_i}{\delta q} \tag{2.22}$$

Sendo que:

$$\dot{p}_i = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\omega}x \\ \dot{\omega}y \\ \dot{\omega}z \end{pmatrix} \tag{2.23}$$

Onde os termos $\dot{x}$, $\dot{y}$ e $\dot{z}$ são encontrados pelo produto existente na equação 2.24. Já os termos $\dot{\omega}x$, $\dot{\omega}y$ e $\dot{\omega}z$ são dados de acordo com o eixo em que determinada junta está se rotacionando. No caso da exemplo dado na figura 8, todas as juntas estão rotacionando no eixo Y, consequentemente os valores de $\dot{\omega}x$, $\dot{\omega}y$ e $\dot{\omega}z$ serão respectivamente 0,1,0.

$$p_i = T_0^i \begin{pmatrix} cdm_i \\ 1 \end{pmatrix} \tag{2.24}$$

Para i=0,1,2,...,n-1 e $T_0^0 = \mathrm{I}$

Seguindo para a energia cinética (EC), é possível calculá-la individualmente para cada junta, e o resultado final pode ser calculado como sendo a soma das energias. Dessa forma, a equação da energia cinética é dada pela equação 2.25. (FU; GONZALEZ; LEE, 1987), (DEWOLF, 2013), (ESPIAU; BOULIE, 1998),(ABDALLAH; BOUTERAA; REKIK, 2016)

$$EC = \frac{1}{2}mv^2 \tag{2.25}$$

Transformando essa equação para o caso da robótica de manipuladores, é dada a seguinte equação 2.26.

$$EC = \frac{1}{2}\dot{x}^T M_m(q)\dot{x} \tag{2.26}$$

Onde:

$$M_m(q) = \begin{pmatrix} m_i & 0 & 0 & 0 & 0 & 0 \\ 0 & m_i & 0 & 0 & 0 & 0 \\ 0 & 0 & m_i & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{xx} & I_{xy} & I_{xz} \\ 0 & 0 & 0 & I_{yx} & I_{yy} & I_{yz} \\ 0 & 0 & 0 & I_{xz} & I_{yz} & I_{zz} \end{pmatrix} \tag{2.27}$$

Na equação 2.27 m_i representa a massa de determinado elo, e I_{ij} representam as forças inerciais. Assim, ao isolar $\dot{x}$ na equação 2.22 e fazer o somatório de todas as juntas na equação 2.26.

$$EC = \frac{1}{2}\dot{q}^T \sum_{i=0}^{n}(J_i^T M_{mi}(q) J_i)\dot{q} \tag{2.28}$$

Como $\dot{q}$ é uma relação de todas as juntas, este pode ir para fora do somatório. Dessa forma, pode-se definir que:

$$M(q) = \sum_{i=0}^{n}(J_i^T M_{mi}(q) J_i) \tag{2.29}$$

Em que,

$$EC = \frac{1}{2}\dot{q}^T M(q)\dot{q} \tag{2.30}$$

Por fim, há de se levar em consideração os efeitos da gravidade no sistema do manipulador robótico. Para isso, foi utilizado o conceito de energia potencial gravitacional visto em DeWolf (2013), e em Espiau e Boulie (1998) assim como em Fu, Gonzalez e Lee

(1987) e em Abdallah, Bouteraa e Rekik (2016). Assim, a equação 2.31 representa a energia gravitacional do sistema.

$$E_p = \sum_{i=0}^{n} F_{gi}^T \dot{p}_i \tag{2.31}$$

ou

$$E_p = F_q^T \dot{q} \tag{2.32}$$

A equação 2.32 leva em consideração o sistemas de juntas. Por causa da conservação de energia essa equivalência é válida. Além disso, é dado que:

$$F_{gi}^T = m_i g^T \quad e \quad g = \begin{pmatrix} 0 \\ 0 \\ -9.81 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{2.33}$$

Isolando o termo $\dot{q}$ presente na equação 2.22 e aplicando na equação 2.31:

$$E_p = \sum_{i=0}^{n} F_{gi}^T J_i \dot{q}_i \tag{2.34}$$

Substituindo 2.32 em 2.35 há o seguinte:

$$F_q^T \dot{q} = \sum_{i=0}^{n} F_{gi}^T J_i \dot{q}_i \tag{2.35}$$

Por fim, anulando o elemento $\dot{q}$ e removendo a transposta de F_q^T é dado a seguinte equação 2.36:

$$F_q = \sum_{i=0}^{n} F_{gi}^T J_i = G(q(t)) \tag{2.36}$$

Tendo todos os elementos dinâmicos considerados no sistema de manipuladores, será feito agora o controle do sistema. A teoria de controle foi criada com a intenção de verificar os sinais do sistema, e corrigi-los para que o valor de saída seja de acordo com seu valor de referência. Atualmente existem diversos métodos para realizar o controle de um sistema. O mais comum e mais popular é o controlador Proporcional, Integrador e/ou Derivativo, ou comumente chamado de PID, sendo este o método utilizado neste trabalho. Segundo o Ogata (2010) este método é o mais útil quando não se conhece o

modelo matemático do sistema. A figura 9 representa o diagrama de blocos de um sistema com um controlador PID.

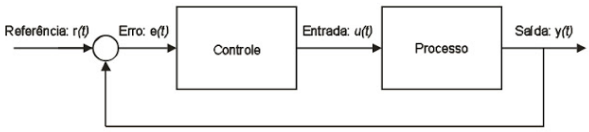

Figura 9 – Diagrama de blocos do sistema em malha fechada com um controlador.

Os parâmetros do controlador são 3: ganho proporcional, ganho derivativo e ganho integrador. Estes por sua vez tem a seguinte relação no tempo, expressada pela equação 2.38 (OGATA, 2010), (SANTOS et al., 2017).

$$u(t) = K_p \cdot e(t) + K_i \int_0^t e(t) \cdot dt + K_d \cdot \frac{de(t)}{dt} \tag{2.37}$$

Estes valores K_p, K_i e K_d são estimados de tal forma a fazer com que e(t) tenda a 0. Uma vez que:

$$e(t) = r(t) - y(t) = \text{Valor Desejado} - \text{Valor Medido} \tag{2.38}$$

Para a dinâmica de manipuladores, foi utilizado um controlador Proporcional e Derivativo (PD) com a intenção de controlar o torque, conforme é visto em Fu, Gonzalez e Lee (1987), Espiau e Boulie (1998) e também DeWolf (2013). Fu, Gonzalez e Lee (1987) também menciona que utilizar todos os termos da equação 2.20 pode ser ineficiente e as vezes impossível, quando realizado o controle de torque em um sistema com múltiplas juntas. Dessa forma, este sugere simplificações. DeWolf (2013) e Espiau e Boulie (1998) aconselham eliminar os elementos de força centrífuga e força de *coriolis*, deixando somente os elementos da gravidade e da inércia, pois este necessita de momentos de inércia muito precisos, caso contrário o sistema entraria em instabilidade. Fazendo isso, é encontrado em 2.39 a seguinte equação de controle.

$$\tau = M(q)\ddot{q}_{des} + G(q) \tag{2.39}$$

Sendo que,

$$\ddot{q} = K_p(q_{des} - q) + K_d(\dot{q}_{des} - \dot{q}) \tag{2.40}$$

Em que K_p e K_d são respectivamente os ganhos proporcionais e derivativos do sistema, q_{des} é a posição desejada e $\dot{q}_{des}$ é a velocidade desejada para o sistema. Substituindo a equação 2.40 em 2.39 é dada a equação 2.41.

$$\tau = M(q)(K_p(q_{des} - q) + K_d(\dot{q}_{des} - \dot{q})) + G(q) \tag{2.41}$$

Assim, a equação 2.41 foi a equação utilizada para o método da dinâmica de manipuladores.

2.2 Robot Operating System (ROS)

ROS é uma plataforma *open-source* voltado para robótica, em que suas características permitem realizar tarefas de diferentes segmentos da área, para uma diversidade de modelos de robôs, sendo eles fixos ou móveis. Uma dessas características, é o fato do ROS ser modular, podendo criar ferramentas que possibilitem desenvolver um projeto de um robô de forma fragmentada em pacotes, fazendo com que diferentes robôs possam utilizar do mesmo pacote criado. Estes pacotes podem ser desenvolvidos atualmente em *python* e em C++, dando também uma flexibilidade para a plataforma. Não obstante, a comunidade para esta plataforma permite discutir e compartilhar ferramentas criadas por desenvolvedores em todo o mundo (QUIGLEY et al., 2009).

O princípio básico de comunicação do ROS trabalha com tópicos, que são canais onde circulam mensagens entre os nós presentes no computador. Como pode ser visto na figura 10.

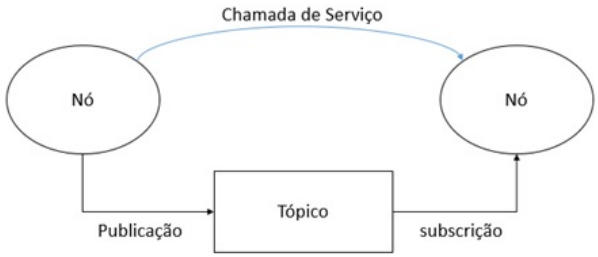

Figura 10 – Princípio de funcionamento do ROS.
Fonte: ROS (2007).

Para isso, faz-se necessário entender alguns conceitos mostrados no ROS (2007) [2]:

- **Nós (*Nodes*)**: Processos que executam a programação. Estes podem conter ações, serviços, e também depender de outras bibliotecas advindas de outros pacotes. Usualmente o sistema de controle de um robô utiliza diversos nós. Por exemplo, um robô utiliza um nó para controlar o sensor laser, um nó para controlar os motores das rodas, um nó para localização e assim por diante. Um nó pode ser escrito na linguagem de programação *python*, C++ e em Lisp;
- **Mensagens (*Messages*)**: Meio de comunicação entre os nós. A mensagem é uma estrutura de dados, podendo conter diferente tipos de dados, desde simples variáveis dos diversos tipos (*float*, int, *bool*, *string*), até vetores e matrizes;

[2] http://wiki.ros.org

- **Tópicos (*Topics*)**: As mensagens são levadas ao um sistema de transporte que utiliza a semântica de *publish/subscribe*. Um nó envia uma mensagem que o publica em um determinado tópico. O nome do tópico serve para identificar o conteúdo da mensagem. Dessa forma caso algum nó se interessa no dado publicado, basta ele "assinar" aquele tópico para ter acesso às informações. Vale ressaltar que podem ter vários publicadores e assinantes utilizando um único tópico;

- **Serviços (*Services*)**: os serviços são arquivos de extensão do tipo *.srv*, localizados dentro da pasta *srv* de um pacote. Sua principal função é transportar mensagens que necessitam uma condição de pedido e retorno, dando uma via de mão dupla para as mensagens, diferentemente dos tópicos.

2.2.1 Transform Library (TF)

O pacote tf tem como objetivo, manter um rastreamento constante dos *frames*[3] de coordenadas e também de dados de transformadas de forma individual, porém mantendo suas relações com todo o sistema, sendo estas relacionada com o ambiente, com um outro *frames* ou também com um *frames* pai. Dessa forma ele garante de forma simplificada a observação de diversas partes de um robô sem a necessidade do conhecimento de todas as partes, visualizando somente a que o usuário deseja.

Segundo Foote (2013), o pacote em questão se assemelha muito com o conceito de *scene graphs*, que pode ser traduzido literalmente como gráficos de cena. Um gráfico de cena nada mais é do que um tipo de estrutura de dados utilizado para representar uma cena em 3D para renderização. Estes consistem de uma árvore de objetos para ser renderizados. Cada objeto é anexado a um objeto pai com uma posição e uma outra informação. Dependendo da aplicação essa outra informação pode ser usado para atualizar regras e propriedades inerciais para simuladores. No caso do tf é possível observar a árvore de transformadas, como é o caso da figura 11:

Assim como o caso dos gráficos de cena, essa biblioteca também utiliza o método de árvore de objetos, no caso árvore de transformadas. Segundo Foote (2013), este método se mostra eficaz pois permite uma rápida busca para as relações feitas entre as transformadas. Além do que evita que seja criado múltiplos caminhos, evitando assim a ambiguidade. Todavia, a diferença que este pacote tem em relação aos gráficos de cenas é que a árvore de transformadas foi projetada para entregar os valores dos dados de forma assíncrona, enquanto os gráficos de cenas enviam seus dados periodicamente. A figura 12 mostra a representação de uma árvore de transformadas.

Para a utilização e distribuição dessas transformadas, Foote (2013) utilizou um sistema análogo aos tópicos do tipo *publisher* e *subscriber*, este sistema análogo foi cha-

[3] Um *frame* é uma estrutura de armazenamento de dados

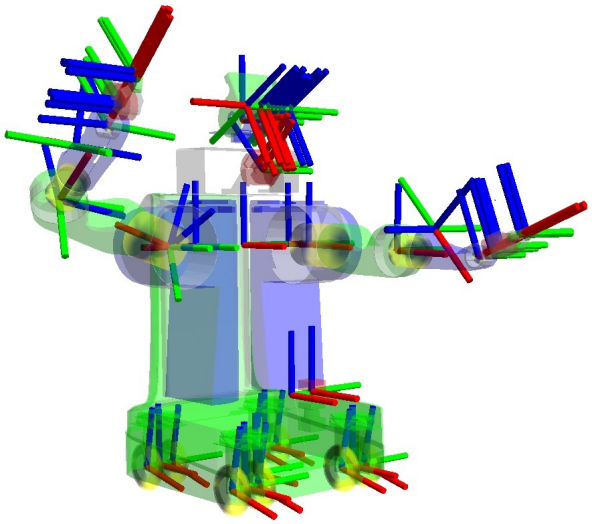

Figura 11 – Árvore de tf's de um robô.
Fonte: Foote (2013).

Figura 12 – Modelo simplificado de uma árvore de tf.
Fonte: Foote (2013).

mado de *Transform Broadcaster* e *Transform Listener*. O caso do *Broadcaster* funciona de forma bem simples, tudo que ele faz é transmitir (*broadcast*) as mensagens das transformadas a todo momento, independente se houve ou não mudança em seus valores, dado uma frequência mínima de transmissão. Já o *Listener* coleta estes dados em uma lista classificada, e quando é requisitado ele realiza uma interpolação dos 2 dados mais próximos no tempo. Em contrapartida ao *Broadcaster* que envia os dados regularmente, o *Listener* nem sempre pode assumir que haverá um dado futuro. Assim sendo, a frequência do *broadcaster* precisa ser alta o suficiente para garantir que a interpolação linear esférica, do inglês *spherical linear interpolation (SLERP)*, das duas amostras se aproxime do movimento da junta. Uma maior frequência garante maior precisão, porém isso requer maiores requisitos de processamento. Essa ferramenta de interpolação garante com que os publicadores sejam assíncronos, e que também operem em diferentes frequências. Aliado a isso, a interpolação cria uma robustez no sistema contra perda de pacotes.

Para computar a transformada de dois *frames*, a e c, com um *frame* b entre eles,

é realizada a seguinte relação:

$$T_{ac} = T_{ab} \dots T_{bc} \tag{2.42}$$

2.2.2 Rosbag

Segundo Quigley, Gerkey e Smart (2015), *rosbag* é uma ferramenta de gravação de mensagens e que podem ser reproduzidos quando quiser. Sendo ela uma ferramenta muito útil para identificar problemas, fazer análises e comparar resultados sem a necessidade de estar utilizando o robô em todas essas situações. Para gravar as mensagens, faz-se necessário o uso da função *record*, listando também os nomes dos tópicos. Caso queira é possível realizar a gravação de todos os tópicos.

- user@hostname **rosbag record scan tf**

Este comando irá salvar todas as mensagens e dará o nome de acordo com o tempo exato do computador (levando em consideração ano, mês, dia, hora, minuto e segundo respectivamente). Garantindo que o nome seja único, ao menos que você faça duas gravações em menos de um segundo. Ao usar a *flag* -a, você estará gravando todos os tópicos que estão publicando no momento. Como pode ser observado:

- user@hostname **rosbag record -a**

Para dar nomes personalizados é utilizado a *flag* -0 ou -o, a segunda opção adiciona a informação do horário atual. Como é demonstrado em seguida

- user@hostname **rosbag record -0 robo1.bag scan tf**
- user@hostname **rosbag record -o robo2.bag scan tf**

E caso queira dar informações específicas sobre quais tópicos você gostaria de gravar, é necessário dar a informação deles após o nome do arquivo, como pode ser visto abaixo:

- user@hostname **rosbag record -0 robo1.bag /left_shoulder_joint left_elbow_joint left_hand_joint**

Assim o usuário está gravando todos os dados do tópico publicador /left_shoulder_joint, /left_elbow_joint e /left_hand_joint. Também é possível modificar o tamanho do *buffer*, bem como ajustar o tempo de gravação. Essas informações podem ser vistas no site[4] do Wiki ROS.

[4] http://wiki.ros.org/rosbag/Commandline

2.2.3 Actionlib

Este pacote do ROS tem uma proposta semelhante ao ROS *services*, de enviar um pedido para um determinado nó para realizar uma tarefa. Porém, diferente do ROS *services*, o actionlib tem mecanismos para interromper as atividades do nó, ou receber constantemente uma atualização da situação da tarefa, fazendo com que seja muito utilizado em tarefas de longa duração (ROS, 2011).

Este necessita de dois nós que irão se comunicar por meio do *ROS Action Protocol*. O primeiro nó, chamado de *Action Client* é responsável por mandar os objetivos a serem realizados e de receber uma atualização dos resultados, se este terminou ou houve algum erro. Já o *Action Server* é responsável por realizar a tarefa com o objetivo recebido. E atualizar o andamento da tarefa para o *Client*. A figura 13 representa um diagrama de comunicação entre os dois nós.

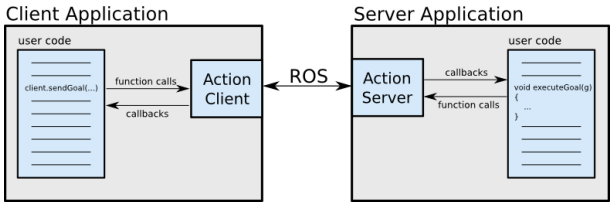

Figura 13 – Interação entre *Action Client* e *Action Server*.
Fonte: ROS (2011).

Para que essa comunicação seja efetivada, o *actionlib* utiliza-se de mensagens chamadas de *action*. Nessas mensagens são definidas variáveis para o Objetivo, *Feedback* e *Result* (ROS, 2011).

2.3 Sensor Kinect

Inicialmente desenvolvido pela *Prime Sense*, depois pela Microsoft®, o dispositivo Kinect foi lançado em novembro de 2010 para o *video game* X-box 360 e tinha como objetivo uma maior imersão do jogador para os jogos, podendo controlar o personagem de um jogo através dos movimentos do seu corpo. Embora o dispositivo não tenha gerado um sucesso entre o seu público-alvo, gerando assim descontinuidade do dispositivo em 2017. O seu uso em pesquisa tem sido abrangente nas mais diversas área de estudo por ser um sensor de boa qualidade e baixo custo (ANDÚJAR, 2012).

Segundo Andújar (2012), o dispositivo Kinect consiste em uma câmera RGB, e um conjunto de sensores para medir a profundidade, sendo composto por um sensor infravermelho e um sensor monocromático C-MOS. Além disso ele possui microfone embutido para comando de voz e uma alavanca motorizada para inclinar o dispositivo. A figura 14 mostra o dispositivo Kinect com os componentes supracitados destacados.

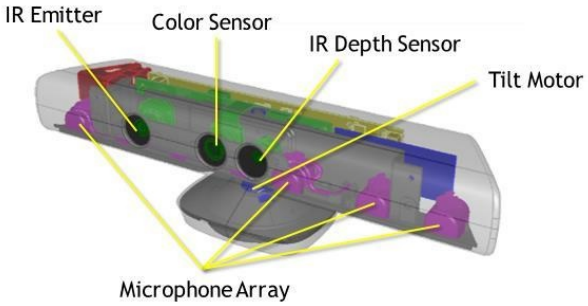

Figura 14 – Sensor Kinect do X-box 360.
Fonte: Andújar (2012).

Tabela 1 – Especificações das câmeras do sensor Kinect- X-box 360

Resolução Câmera RGB	640 x 480 e 30fps do sensor e de saída
Resolução Câmera Infra-vermelho	1280x1024 do sensor, 640x480 e 30fps de saída
Alcance de Operação	0.5m até 5m
Campo de Visão do sensor de profundidade	58º Horizontal, 45º Vertical e 70º Diagonal
Resolução Espacial (2m de distância)	2mm
Resolução de Profundidade (2m de distância)	6mm

Fonte: Andújar (2012) e Nguyen, Izadi e Lovell (2012).

Dessa forma, as imagens são capturadas e processadas no chip PS1080-A2 e armazenadas na memória flash de 64 Mb DDR2. As especificações da imagem são mostradas na tabela 1.

Para a obtenção da profundidade na imagem, (ANDÚJAR, 2012) mostra que diversos pontos são liberados pelo sensor infravermelho em um padrão irregular e variando a intensidade destes pontos. Depois ele reconstrói a imagem levando em consideração a distorção destes pontos. A figura 15 mostra o fluxograma, e a figura 16 mostra um padrão de pontos feito pelo sensor infravermelho, que servem de orientação. Mais detalhes sobre o padrão do sensor infravermelho pode ser visto em (KONOLIGE; MIHELICH, 2012).

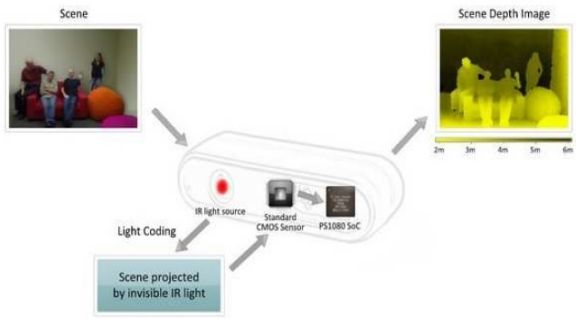

Figura 15 – Funcionamento do sensor infravermelho.
Fonte: Andújar (2012).

Vale ressaltar também que, segundo Nguyen, Izadi e Lovell (2012) o erro da medida do sensor de profundidade aumenta quadraticamente de acordo com a distância,

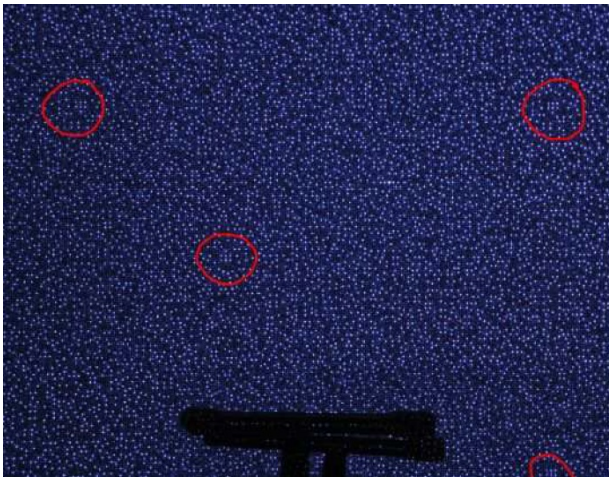

Figura 16 – Padrão de emissão de luz infravermelho.
Fonte: Andújar (2012).

alcançando um valor de 14mm de erro para a distância de 3 metros. Dessa forma é recomendado uma distância de 1 a 2 metros em relação ao aparelho para aplicações de mapeamento e rastreamento.

2.4 Filtros Digitais

Quando se lida com sistemas que necessitam realizar medições através de sensores, muitas vezes os valores medidos não demonstram o valor real, isso se dá pelo erro inerente do sensor, bem como os ruídos ali presentes. Na situação de ruídos, os filtros digitais podem fazer a diminuição e até mesmo sua eliminação, bem como muitas vezes fornecer um valor mais fidedigno do que o valor medido. Assim, neste capítulo será abordado três técnicas de filtros digitais, o método da Média Móvel, o filtro de Kalman convencional, e uma variação deste modelo, chamado de filtro de Kalman Robusto.

2.4.1 Filtro Média Móvel

O filtro Média Móvel, segundo Smith (1999), é o filtro mais conhecido por ser de fácil entendimento e de fácil implementação, este também se mostra eficiente quando o objetivo é a redução de ruído branco[5] no domínio do tempo discreto, Porém este é considerado um dos piores filtros para se utilizar no domínio da frequência.

Assim como o nome sugere, o filtro Média Móvel trabalha através do cálculo de uma média de um determinado número de pontos (também chamada de janela de dados), em que este se atualiza à medida que novas amostras aparecem. A equação 2.43 apresenta

[5] Ruído branco segundo Smith (1999) é quando você possui variáveis independentes e identicamente distribuídas

a ideia deste método.

$$y[i] = \frac{1}{M} \sum_{j=0}^{j=M-1} x[i-j] \tag{2.43}$$

Dessa forma, supondo uma janela de cinco amostras e o valor do resultado de número quarenta é dado que:

$$y[40] = \frac{x[40] + x[39] + x[38] + x[37] + x[36]}{5} \tag{2.44}$$

Este intervalo de j no somatório pode ser substituído com intenções de fazer o número i como sendo a amostra central, ou seja, de maneira simétrica assim j seria igual a $\frac{-M-1}{2}$ até $\frac{M-1}{2}$. Lembrando que para essa condição, o valor de M precisa ser ímpar. Porém, na questão da programação é preferível utilizar a opção apresentada pela equação 2.44. Embora nesta situação é criado um pequeno deslocamento entre o sinal de saída e o sinal de entrada. como pode ser observado pela figura 17.

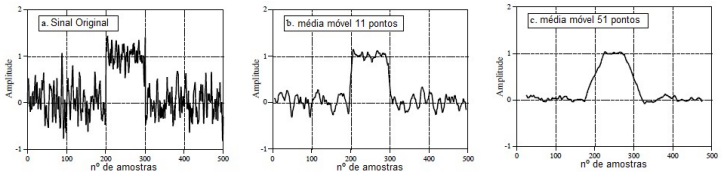

Figura 17 – Gráfico de um filtro Média Móvel para diferentes janelas.
Fonte: Smith (1999).

Apesar deste atraso, que pode ser maior se o valor da janela aumentar, Smith (1999) argumenta que o Média Móvel é o filtro que apresenta a melhor redução de ruído branco, dentre todas as opções de filtros lineares, dada uma determinada banda de corte. Isso se dá pelo fato do ruído branco ser aleatório, dessa forma não existem pontos importantes, ou seja, todos os pontos possuem a mesma importância. Dessa forma, o menor ruído será quando todos os valores de entrada possuem a mesma importância, que é exatamente o que o filtro Média Móvel se propõe a fazer.

2.4.2 Filtro de Kalman

O filtro de Kalman é muito utilizado em algoritmos de estimação de dados. Criado em 1960 por Rudolf Kalman, esta técnica possui solução recursiva ótima para problemas lineares de filtragem. Essa solução recursiva é atualizada em cada estado, fazendo com que ele precise somente do estado anterior para ser efetivo em sua técnica, isso faz com que não seja necessário o armazenamentos de dados de todo o passado, tornando-o mais eficiente computacionalmente falando do que outros métodos que precisam de todos os dados para criar uma estimação de valores. O conceito de estado é muito importante para o entendimento desta técnica de filtragem, assim, Segundo Haykin (2001):

> "O vetor de estados ou simplesmente estado, representado por x_k, é definido como um conjunto mínimo de dados que permite descrever de maneira única, o comportamento de uma dinâmica não forçada de um sistema. [6]

Na maioria dos casos, o estado x_k é desconhecido, porém é possível estimá-lo através de um conjunto de dados obtidos por sensoriamento, que pode ser denotado como y_k. A partir disso, a figura 18 representa o modelo de sistema em que o método filtro de kalman se aplica.

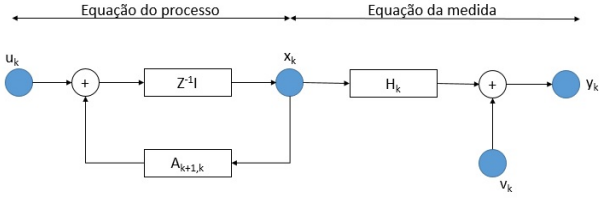

Figura 18 – Diagrama de blocos do filtro de Kalman.
Fonte: Haykin (2001).

Dessa forma, tem-se que a equação que define o processo é dado pela equação 2.45.

$$x_{k+1} = A_{k+1}x_k + u_k \tag{2.45}$$

Onde A_{k+1} representa a matriz de transição do estado x_k para o estado x_{k+1}, e u_k representam o ruído do processo, sendo ele um ruído branco. Já a equação que representa o processo de medição é dada pela equação 2.46.

$$y_k = H_k x_k + v_K \tag{2.46}$$

Em que H_k representa a matriz do sensor e v_k o ruído do sensor. Dessa forma, ao tomar as equações 2.45 e 2.46, é possível através de uma medida atual de y_k, estimar um valor de x_k. Pelo fato do método ser recursivo, a estimação atual depende da estimação anterior, que pode ser representada como sendo $\hat{x}_k^-$, e a estimação posterior pode ser representada por $\hat{x}_k$. Assim, segue a equação:

$$\hat{x}_k = \hat{x}_k^- + K_k(y_k - H_k\hat{x}_k^-) \tag{2.47}$$

De acordo com a equação 2.47, o elemento K_k é chamado de ganho de Kalman. Este é o principal responsável por escolher qual resultado terá o maior peso na estimação, se a estimação anterior, ou a medida. Para isso encontrar o valor do ganho, será utilizado

[6] traduzido do seguinte texto: "The *state vector* or simply *state*, denoted by x_k, is defined as the minimal set of that that is sufficient to uniquely describe the unforced dynamical behavior of the system"(HAYKIN, 2001).

a matriz de covariância estimada do sistema, denotada por P_k^-, e também a covariância do sensor, representado por R_k como pode ser observado na equação 2.48.

$$K_k = P_k^- H_k^T [H_k P_k^- H_k^T + R_k]^{-1} \tag{2.48}$$

Dessa forma, para estimar a covariância seguinte, é dado pela equação 2.49.

$$P_k^- = A_{k,k-1} P_{k-1} A_{k,k-1}^T + Q_{k-1} \tag{2.49}$$

Pode-se observar na equação 2.49 que o valor da covariância estimada depende do seu valor estimado anteriormente, a mesma matriz de transição de estado, e a covariância da equação. Para a estimação do próximo estado é possível observar que este depende da estimação anterior, como pode ser visto na equação 2.50:

$$\hat{x}_k = A_{k,k-1} \hat{x}_{k-1} \tag{2.50}$$

E por fim, para atualizar o valor de covariância é dado a seguinte equação 2.51:

$$P_k = (I - K_k H_k) P_k^- \tag{2.51}$$

Para a execução do filtro de Kalman, é necessário também os valores iniciais da covariância e do valor do estado inicial. Dessa forma, para estes valores, utiliza-se o conceito do erro em mínimos quadrados.

$$\hat{x}_0 = E[x_0] \tag{2.52}$$
$$P_0 = E[(x_0 - E[x_0])(x_0 - E[x_0])^T] \tag{2.53}$$

Este conceito da média ponderada, visto na equação 2.47 é a principal ferramenta que faz o filtro de Kalman garantir um resultado melhor que o valor encontrado pelo sensor e também melhor que o estimado, se analisados separadamente. Segundo Faragher (2012) e Thrun (2011), pode representar tanto a medida quanto a estimação por Gaussianas, contendo média e variância. Porém, o que o filtro de Kalman faz é uma média ponderada entre estes dois dados. Ao fazer isso, está sendo encontrando uma Gaussiana, sendo um produto das gaussianas da medida e da estimação. Obtendo um resultado com uma variância menor e com a média mais próxima do valor verdadeiro.

2.4.3 Uma Variação do Filtro de Kalman: Filtro de Kalman Robusto

Apesar do filtro de Kalman ser muito eficiente para problemas lineares, este filtro ainda assim é sensível para problemas de *outliers*, que segundo Ting, Theodorou e Schaal

(2007), é definido como uma observação que se encontra fora de um padrão de distribuição geral. Estes *outliers* podem surgir por conta de ruído, falhas no sensor ou até distúrbios do ambiente que não foram previstos. Para perceber se essa observação é de fato um *outlier* ou não, algum conhecimento dos dados observados precisa ser conhecido. Todavia, armazenar um conjunto inteiro de dados para uma situação em tempo real se torna inviável, uma vez que a frequência de amostragem do sensor necessita ser alta e o sistema muitas vezes não possui capacidade de processamento alta o suficiente para processar e identificar o *outlier* em um curto espaço de tempo. Para este cenário, Ting, Theodorou e Schaal (2007) propõe a modificação do filtro de Kalman, para que este seja insensível às ocorrências de *outliers*, uma vez que o Filtro de Kalman convencional é excelente para aplicação de sistemas em tempo real, utilizando apenas o dado observado no tempo atual. Assim, para que seja superado este problema, um ganho escalar w_k será atribuído em cada observação y_k. Ganhos estes dados e conhecidos. Também será feita uma distribuição Gamma, priorizando que estes ganhos sempre sejam positivos. Além disso, será aprendido a dinâmica do sistema para cada nova amostra. Assim, as principais distribuições deste modelo serão mostradas nas equações 2.54, 2.55 e 2.56:

$$y_k|x_k, w_k \sim Normal(Hx_k, R/w_k) \tag{2.54}$$

$$x_k|x_{k-1} \sim Normal(Ax_{k-1}, Q) \tag{2.55}$$

$$w_k \sim Gamma(a_{w_k}, b_{w_k}) \tag{2.56}$$

O objetivo destas alterações é tratar este sistema como um problema de aprendizado por minimização de expectativa e maximização da probabilidade logarítmica $\log p(x_1 : k, y_1 : k, w_1 : k)$. Caso fosse possível analisar a probabilidade de toda a amostra, seria possível encontrar a verdadeira distribuição posterior de todas as variáveis ocultas de $Q(w, x)$. Porém, como está sendo utilizado sistema em tempo real e este não permite analisar toda a distribuição, foi utilizada a seguinte técnica de aproximação fatorial para encontrar a verdadeira distribuição posterior:

$$Q(w, x) = \prod_{i=1}^{N} Q(w_i) \prod_{i=1}^{N} Q(x_i|x_{i-1})Q(x_0) \tag{2.57}$$

A equação 2.57 de fatoração de x conserva a propriedade de Markov presente no filtro de Kalman, mesmo com as alterações proposta pelo autor. Esta aproximação foi escolhida propositalmente, para que $\mathrm{Q}(w_k)$ seja independente de $\mathrm{Q}(x_k)$. Dessa forma,

as equações 2.58, 2.59, 2.60, 2.61 e 2.62 representam as equações do filtro de Kalman Robusto.

$$\hat{x}_k^- = A_{k,k-1}\hat{x}_{k-1} \tag{2.58}$$

$$P_k^- = Q_k \tag{2.59}$$

$$K_k = P_k^- H_k^T [H_k P_k^- H_k^T + (1/w_k)R_k]^{-1} \tag{2.60}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(y_k - H_k\hat{x}_k^-) \tag{2.61}$$

$$P_k = (I - K_k H_k)P_k^- \tag{2.62}$$

Sendo que:

$$w_k = \frac{a_{w_{k,0}} + 1/2}{b_{w_{k,0}} + (y_k - H_k\hat{x}_k^-)^T R_k^{-1}(y_k - H_k\hat{x}_k^-)} \tag{2.63}$$

Percebe-se que houveram apenas duas alterações em relação ao filtro de Kalman convencional, estes se encontram na equação 2.59 e 2.60. Em 2.59, P_k^- não depende mais da propagação da covariância no estado anterior. Além disso, na equação 2.60 é observado que R_k possui agora um peso sobre ele de $\frac{1}{w_k}$. A presença deste peso revela que se a predição do erro em y_k for muito grande ao ponto de influenciar o denominador, isso acarretará em um peso muito pequeno em w_k. Isso acarretará em um ganho pequeno de Kalman K_k. Dessa forma, a influência do valor medido y_k será diminuído quando for obtido $\hat{x}_k$.

Para os valores iniciais, Ting, Theodorou e Schaal (2007) sugere que $a_{w_{k,0}}$=$b_{w_{k,0}}$ = 1, isso garante com que a média seja $a_{w_{k,0}}/b_{w_{k,0}} = 1$ e a variância $(a_{w_{k,0}}/b_{w_{k,0}})^2 = 1$. Segundo o autor, estes valores são geralmente válidos para qualquer conjunto de dados ou aplicações e que não precisa ser modificado, a menos que o usuário tenha uma razão para fazer isso. Além disso, você pode fazer com que a matriz **A** e **H** sejam iguais a matriz identidade **I**, uma vez que o algoritmo é insensível em seu começo e no fim todos os valores convergirão para o mesmo resultado no final, independente destes valores. Os valores de Q e R podem ser atribuídos como uma fração da matriz identidade. O autor sugeriu fazer **Q**=**R**=0,01**I** para sinais com muito ruídos. E **Q**=**R**=10^{-4}**I** para sinais pouco ruidosos.

2.5 Controle Nebuloso: Lógica *Fuzzy*

O conceito de lógica *fuzzy*, também conhecida de lógica nebulosa, foi inicialmente apresentado em 1965 por Lofti Zadeh. Neste novo conceito, seria operado de maneira

diferente da lógica booleana, propondo criar um conjunto de classes que possuem uma amplitude em um intervalo de 0 a 1 no conjunto dos números reais. Permitindo assim criar descrições linguísticas, como por exemplo definir uma faixa de altura como sendo uma pessoa alta, média ou baixa, ou criar margens para um resultado ótimo, normal e ruim. Estas classes, juntamente com algumas regras de inferência, podem definir a ação que o mecanismo irá tomar. A figura 19 representa o diagrama de blocos do funcionamento da lógica *fuzzy*.

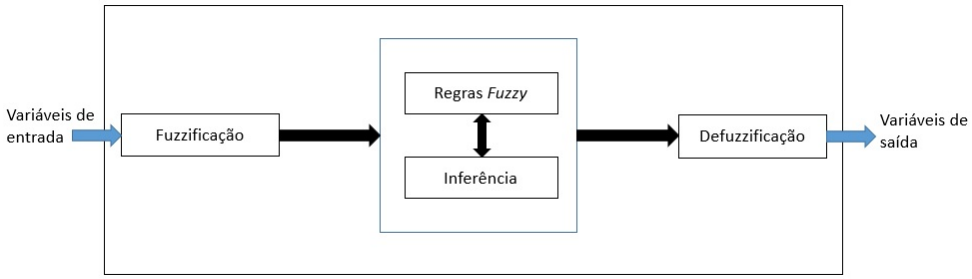

Figura 19 – Diagrama de blocos da lógica *Fuzzy*.
Fonte: Ligutan et al. (2017).

Este consiste em receber os dados do sistema, e adequar eles dentro das classes na fuzzificação, para que os dados numéricos sejam transcrevidos para as descrições linguísticas presente nas classes *fuzzy*. Estes conjuntos podem ser do tipo triangular, trapezoidal e gaussiana. A figura 20 mostra um exemplo de classe para fuzzificação (LIGUTAN et al., 2017) e (ZHANG, 2010).

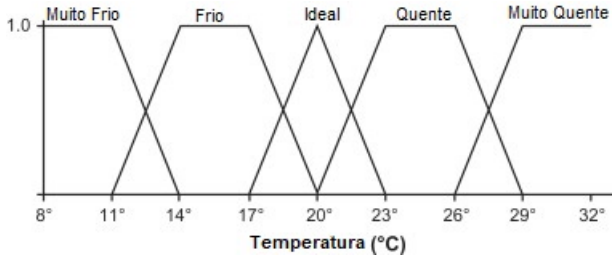

Figura 20 – Exemplo de fuzzificação de uma classe de temperaturas.
Fonte: Zhang (2010).

Percebe-se que na figura 20 existem 5 conjuntos para a classe temperatura. Sendo elas: "Muito Frio","Frio", "Ideal", "Quente"e "Muito Quente". Dessa forma, Zhang (2010) mostra que para uma temperatura de 18°C, é dado em sua fuzzificação que essa variável é "Frio"em um grau de 0.75 e "Ideal"em um grau de 0.25.

Estes valores seguirão para a tabela de regras, para verificar qual será a resposta do sistema para a entrada obtida. E as regras servem para indicar qual será a resposta à variável de entrada. No exemplo da figura 20, supondo que se trata da temperatura de uma sala, e há um ar condicionado nessa sala que controla a temperatura do ambiente. As regras na lógica *fuzzy* são dadas como situações de Se/Então, ou *If / Then*. Por exemplo,

é possível fazer como saída a velocidade do motor presente no ar condicionado e criar regras como:

- Se temperatura = Muito Frio, então velocidade do motor = Desligado;
- Se temperatura = Frio, então velocidade do motor = Lento;
- Se temperatura = Ideal, então velocidade do motor = Médio;
- Se temperatura = Quente, então velocidade do motor = Rápido;
- Se temperatura = Muito Quente, então velocidade do motor = Muito Rápido;

Perceba que tanto as regras quanto as delimitações dos conjuntos vão de acordo com a opinião e expertise dos projetistas. Vale ressaltar também que as regras na lógica *fuzzy* convencional precisam abordar todas as possíveis soluções. Porém há casos como no *fuzzy* hierárquico sugerido em Junior et al. (2016) que essas regras podem ser reduzidas.

Por fim, na defuzzificação há novamente a transcrição da variável (dessa vez de saída) que está na forma linguística descritiva para o seu valor real. Para isso existem métodos como o Centro de Área (ou Centro de Gravidade), Centro de Máximo e Média de Máxima para realizar essa transcrição, cada um tendo suas peculiaridades. Leekwijck e Kerre (1999) aborda melhor sobre as características de cada método (ZHANG, 2010).

2.6 Virtual Robot Experimental Platform (V-REP)

Desenvolvido pela Coppelia Robotics, Virtual Robot Experimental Platform, ou V-REP, é um simulador com uma arquitetura muito versátil, e também possui diversas funcionalidades que podem ser ativadas ou desativadas de maneira independente. Uma delas em questão são as funções para a comunicação com o ROS. Estas funcionalidades operam de maneira sincronizada, fazendo com que uma coopere com a outra, como pode ser observado no exemplo dado por Freese et al. (2010), em que um robô humanoide, utiliza-se da cinemática para calcular o movimento das pernas, para uma posição e orientação desejada dos pés, enquanto se calcula a posição das juntas para serem usadas como posições desejadas pelo módulo dinâmico, com a finalidade de atingir o resultado esperado. A figura 21 mostra a tela inicial do simulador, onde é possivel verificar algumas das ferramentas utilizadas, bem como a hierarquia dos objetos utilizados no cenário.

Com estas ferramentas, é possível criar um cenário de diversas formas, como também criar robôs através de ferramentas denominados *scene objects*, e até mesmo criar terrenos. Semelhante ao pacote do tf do ROS, este simulador utiliza-se da hierarquia baseada em árvores. Dessa forma, é possível citar as ferramentas, de acordo com Freese et al. (2010) e Rohmer, Singh e Freese (2013), como sendo:

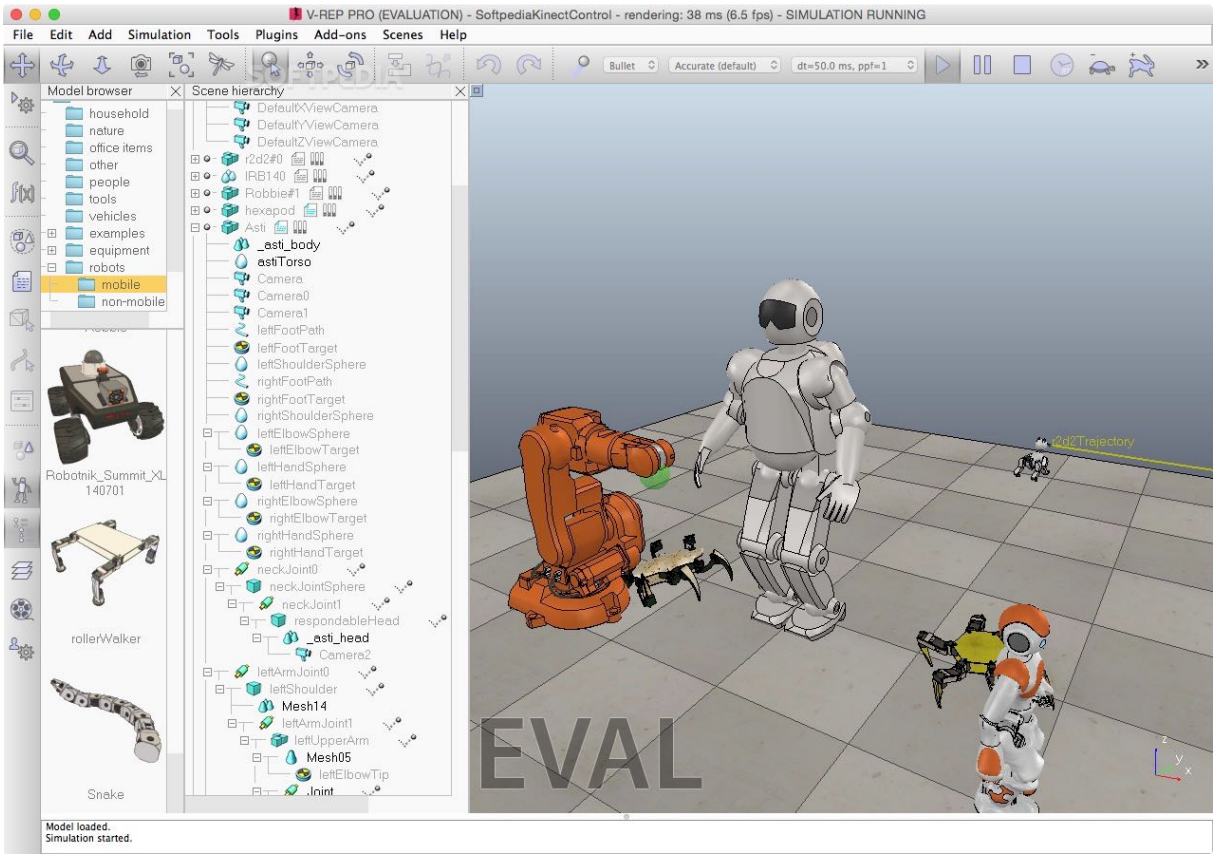

Figura 21 – Tela principal do simulador V-REP.
Fonte: Robotics (2010).

- **Juntas**: elementos que ligam e movimentam dois ou mais objetos com um, dois ou até três graus de liberdade. Os tipos de juntas são juntas prismática, juntas de revolução, juntas esférica, entre outras;
- **Trajetórias**: permite a criação de uma trajetória com movimentos complexos no espaço, podendo ter uma sucessão de combinações de translação, rotação e até pausa;
- **Objetos Sólidos**: forma sólida criada por malhas triangulares, com objetivo de criar a visualização de um corpo rígido;
- **Luz e Câmera**: Utilizado somente para a visualização dos cenários, embora as luzes podem influenciar diretamente o funcionamento de um dos tipos de sensor;
- ***Dummies***: os *dummies* são pontos com orientação, utilizados como referência ou objetivos para diversas tarefas;
- **Sensores**: Podendo ser sensores de proximidade (detecção de obstáculo), renderização (extrair informação de imagens complexas) e sensores de força (gravar informação de torque e força aplicado em um corpo rígido);
- **Laminador**: essa ferramenta tem como objetivo realizar cortes em corpos sólidos;

- **Gráficos**: essa ferramenta permite gravar uma grande variedade de dados em 1 dimensão. E reproduzi-los diretamente (o dado em relação ao tempo), ou combinar diferentes dados para reproduzir um gráfico 2d (X/Y) ou até mesmo curvas em 3D.

Além destas ferramentas, o simulador conta com diversos mecanismos para calcular todas as interações realizadas dentro do V-REP. Dentre estes mecanismos é possível citar:

- **Módulo de Cinemática Direta e Inversa**: Calcula a cinemática para qualquer tipo de mecanismo. Ele se baseia no cálculo dos mínimos quadrados amortecidos e da pseudo inversa;
- **Módulos Dinâmicos ou Físicos**: Calcula a dinâmica de um corpo rígido (resposta de colisão, agarramento, entre outros), onde se é utilizado o pacote *Bullet Physics*;
- **Módulos de planejamento de trajetória**: Permite o planejamento de trajetórias de tarefas holonômicas e não holonômicas através de um algoritmo baseado em aproximação de derivadas chamado *Rapidly-exploring Random Tree*;
- **Módulo de detecção de colisão**: Verifica qualquer interferência entre um corpo em relação aos demais corpos presentes no cenário, é possível também calcular o contorno destes objetos. Para isso, este módulo utiliza em uma estrutura de dados baseada em um conjunto binário dos limites de uma caixa orientada *Oriented Bounding Boxes*, além de utilizar uma técnica de otimização chamada de *temporal coherency caching*;
- **Módulo do cálculo de mínima distância**: Calcula a mínima distância entre dois ou mais objetos no cenário. Este utiliza a mesma estrutura de dados vista do módulo de detecção de colisão, bem como a mesma técnica de otimização.

Ele também permite uma flexibilização da forma que deseja simular. Podendo criar uma interface com o ROS através do *ROS Interface*, ou ter o controle de toda a simulação através do *RemoteAPI*, criando um *plugin*, um *AddOn* e até trabalhando com o *Blue zero* através do *Blue Zero Node*, que pode ser observado no *site* [7].

O V-REP não é o único simulador de robótica, dentre os simuladores do gênero pode-se citar outros como o Gazebo, Open RHP e até o Webots. Pelo fato do Gazebo ser nativo ao ROS, enquanto o V-rep foi considerado o que teve maior taxa de satisfação segundo Ivaldi, Padois e Nori (2014). Nogueira (2014) comparou estes dois simuladores nos seguintes pontos: integração com o ROS, Modelagem de cenário, Modificação nos modelos dos robôs, controle programático e utilização da CPU. A partir dessa comparação, é notável que o V-REP apresentou maior eficiência nos diversos tópicos em relação ao

[7] http://www.coppeliarobotics.com/helpFiles/index.html

Gazebo. A única desvantagem foi na integração do ROS, pois o Gazebo é um simulador projetado para trabalhar nativamente com o ROS. Todavia as funções do V-REP de comunicação com o ROS são suficientes para garantir uma boa integração. Essa comparação foi importante para escolher o simulador utilizado neste trabalho, também para justificar a escolha do mesmo.

3 Procedimentos Experimentais

Neste capítulo será abordado o cenário do estado da arte, mostrando diferentes autores que realizaram a aplicação de controle de um manipulador robótico com diferentes técnicas encontradas, bem como apresentar a metodologia utilizada em nosso trabalho.

3.1 Estado da Arte

A utilização do Kinect na robótica permitiu pesquisas nas áreas de reconhecimento de gestos, rastreamento de esqueleto e controle de manipuladores e robôs móveis.

Pedro (2013), por exemplo, fez um trabalho para controle de um manipulador utilizando o Kinect para verificar o ambiente e detectar colisões com o manipulador, de maneira a tratar estas colisões de tal forma a serem evitadas, ou incentivar a colisão caso seja benéfica para concretização das tarefas. Pode-se observar em Cavalcante (2012) e Grushko e Bobovsky (2016) utilizaram o dispositivo Kinect para detectar movimentos humanos para encontrar os ângulos e controlar um robô humanoide, Grushko e Bobovsky (2016) utilizou o simulador V-REP em seus experimentos. Nguyen et al. (2017) usou o Kinect para controlar um robô humanoide, mas a diferença é que este utilizou a versão atual do sensor, e fez o controle por meio da cinemática inversa, além de comparar a diferença entre os dados coletador pelo Kinect com o resultado do movimento do robô. Silva et al. (2012) controlou um manipulador através do Kinect, e também criou um sistema supervisório utilizando o Arduino para mostrar para o usuário o movimento realizado pelo manipulador. Pagi e Padmajothi (2017) controlou um manipulador robótico utilizando o Kinect para visão computacional, onde se identificava padrões de gestos com as mãos para determinar qual ação o manipulador iria executar. Há de se citar também a utilização da cinemática inversa para controlar o Robô NAO presente em Li et al. (2016) e para agarrar um objeto, usando o Kinect como a visão do robô em Ali et al. (2015). Por fim, na parte da cinemática é possível citar Barakat, Gouda e Bozed (2016) que faz uma análise da cinemática utilizando o MATLAB.

Na área da dinâmica de manipuladores pode-se verificar em Abdallah, Bouteraa e Rekik (2016) a utilização das equações dinâmicas para controlar um manipulador utilizando *Labview*, enquanto Espiau e Boulie (1998) atenta seu foco na importância do cálculo do centro de massa de robôs articulados. Já em Liang et al. (2016) utiliza-se a segunda versão do Kinect para controlar o manipulador de maneira mimetizada.

Ligutan et al. (2017) operou o manipulador utilizando lógica *fuzzy*, assim como Junior et al. (2016) que incorporou a lógica *fuzzy* em um controlador proporcional. Enquanto

Santos et al. (2017) utiliza-se da ferramenta do ROS actionlib para controlar um manipulador de duas maneiras, utilizando lógica *fuzzy* e controle PID. O controle PID também foi visto nos artigos de Albán e Adorno (2017) utilizando um controle híbrido de posição e força para controlar o movimento do manipulador e a força de contato quando este toca uma superfície. E também foi visto por Afthoni, Rizal e Susanto (2013) utilizando um controlador PD para controlar um manipulador.

Por fim, foi visto em Zhao et al. (2016) um método semelhante ao trabalho. Onde ele calcula o ângulo das juntas e envia diretamente para o manipulador, adicionando um filtro média móvel para suavizar o efeito do ruído.

3.2 Metodologia do Trabalho

Juntamente com os trabalhos supracitados, este trabalho propõe controlar em simulação um manipulador robótico com o Kinect e também utilizar diversas técnicas, sendo elas: a cinemática direta e inversa, controle por lógica *fuzzy* e dinâmica de manipuladores com um controlador proporcional-derivativo. Além de traçar um comparativo entre elas, utilizando erro quadrático médio entre o sinal de saída e o sinal de entrada, a variância do erro e análises visuais do próprio gráfico e dos vídeos criados. O objetivo de utilizar o erro quadrático médio é de verificar o quanto o sinal de saída se deslocou do sinal de entrada em amplitude, já a variância do erro é de verificar se o erro se mantém constante frente as alterações do sinal de entrada. Caso isso ocorra significa que a saída está acompanhando o sinal da entrada, independente se houver um *offset* do sinal.

Para obter os resultados esperados, a metodologia de pesquisa abordada neste trabalho foi a pesquisa experimental. Este tipo de metodologia foi escolhida, pois segundo Rodrigues (201-), "consiste essencialmente em determinar o objeto de estudo, selecionar as variáveis capazes de influenciá-lo, e definir formas de controle e de observação dos efeitos que a variável produz no objeto".

Dessa forma o trabalho consiste em adquirir as informações do sensor Kinect, enviar para o computador utilizando o OpenNI, e coletar as informações para obter as tf's de cada junta utilizando o programa *openni_tracker*, converter as tf's em mensagens do tipo geometry_msgs/Vector3 no programa *openni_listener*. E por fim passar essas informações para o simulador que possui o modelo dos manipuladores utilizados e também para os métodos encontrados na literatura para movimentação e controle do manipulador. A figura 22 representa o fluxograma do projeto.

A primeira parte do projeto consistiu em adquirir os dados que foram utilizados nas técnicas sugeridas. Sendo assim, foi utilizado o *software* OpenNI para fazer o rastreamento do esqueleto e coletar os dados das juntas do corpo humano. O algoritmo *openni_tracker*, disponibilizado pela OpenNI foi responsável em pegar as informações das 15 partes do

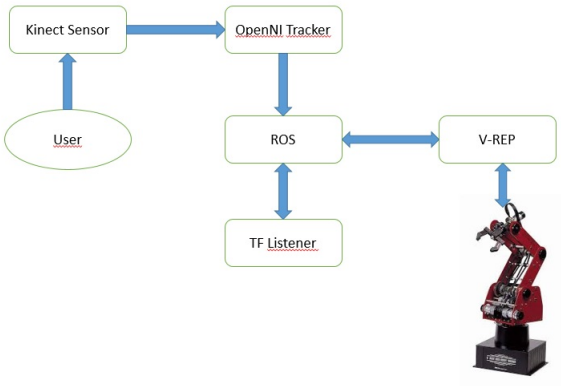

Figura 22 – Fluxograma proposto para o trabalho.
Fonte: Autor deste trabalho.

corpo e transformar em arquivos do tipo tf. A figura 23 representa o rastreamento sendo bem sucedido e mostrado no simulador RViz. Um vídeo sobre este teste pode ser visto no *link*[1] em questão.

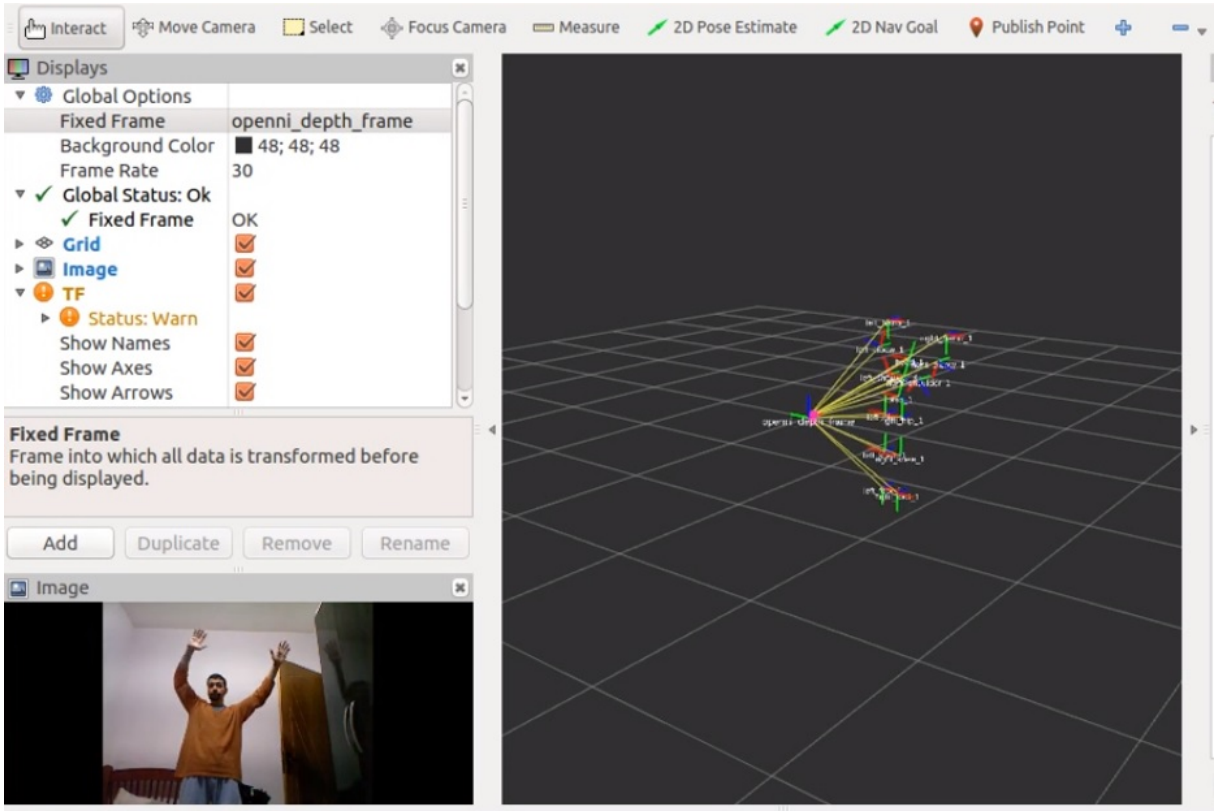

Figura 23 – TF das juntas do corpo, rastreado pelo Kinect.
Fonte: Autor deste trabalho.

Como é possível observar na figura 23, também no vídeo de aquisição de dados o algoritmo conseguiu rastrear o corpo com eficiência, além disso foi possível verificar alguns ruídos e pequenos erros, como havia sido mencionado na teoria presente na seção 2.3, porém foi observado também que este encontrou problemas de instabilidade no rotacional

[1] Aquisição de dados: https://youtu.be/HhwGE2LE5RQ

da região dos ombros e também no cotovelo, quando alguns movimentos foram realizados (principalmente quando as mãos cruzam o tronco), este problema também foi observado por Cavalcante (2012), e pode ser um problema do OpenNI ou da própria versão do Kinect. Com o objetivo de reparar este problema, foi aplicado filtros digitais para eliminar ou suavizar essa oscilação.

O passo seguinte foi criar o algoritmo *transform listener* para ler estes dados enviados pelo sensor Kinect. Os dados enviados pelo Kinect são de posição das juntas e também a rotação na forma de Quaternion das mesmas. Dessa forma, para as técnicas que utilizam o ângulo de cada junta, foi realizado a conversão do Quaternion para ângulos de Euler em radianos. As juntas utilizadas foram a do ombro e a do cotovelo direito. Não foi utilizada a rotação da junta da mão pois o OpenNI para o X-box 360 não informa a rotação desta junta.

Tendo então criado o algoritmo, foi criado um arquivo *.bag* de nome *subset.bag* contendo as informações de ombro e cotovelo, que podem ser representados pelas figuras 24 e 25 respectivamente.

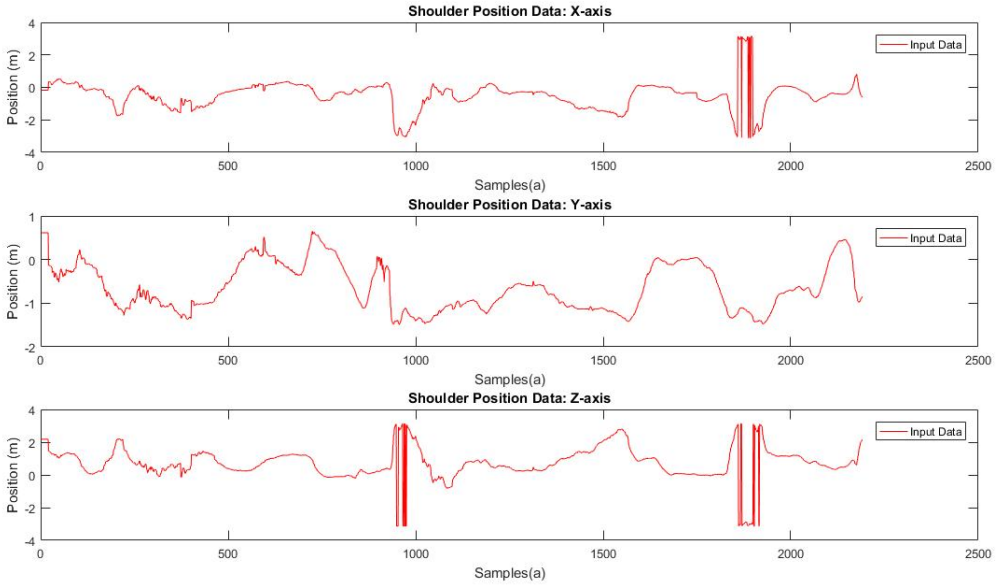

Figura 24 – Dados do rotacional do ombro, gravados em um arquivo *bag*.
Fonte: Autor deste trabalho.

As mesmas oscilações encontradas por Cavalcante (2012) também foram observadas na figura 24 no eixo X e Z e também nos eixos X e Z da figura 25. Todavia, dos eixos que apresentaram essa oscilação será utilizado apenas o eixo Z do ombro, presente na figura 24. No caso das técnicas que necessitam apenas da posição da mão, foi utilizada as informações de posição da mão direita no plano cartesiano, isso em relação à posição do ombro direito, criando uma relação de distância entre estes dois pontos. A figura 26 mostra o conjunto de dados da posição da mão gravados em um segundo arquivo do tipo

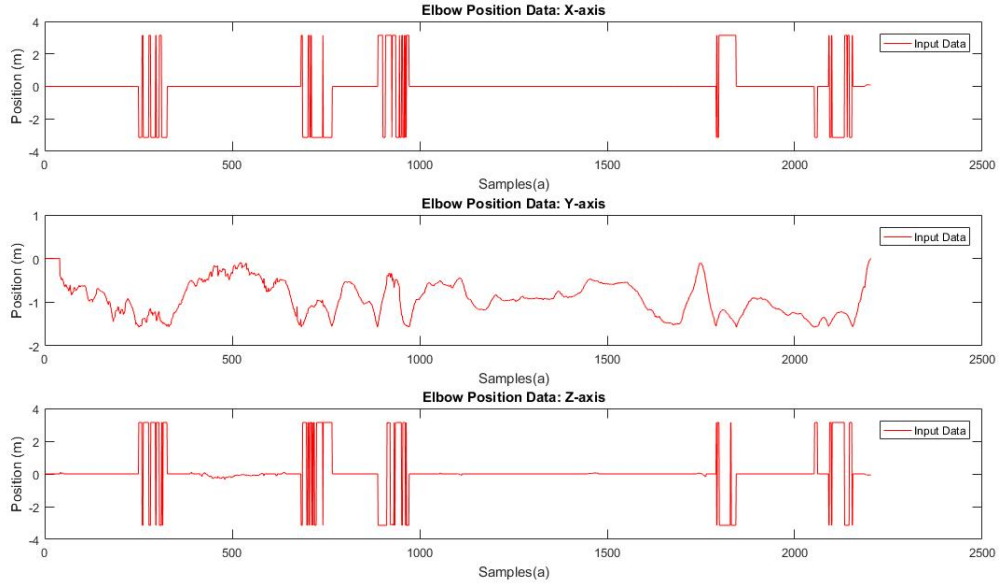

Figura 25 – Dados do rotacional do cotovelo, gravados em um arquivo *bag*.
Fonte: Autor deste trabalho.

bag de nome *subset2.bag*.

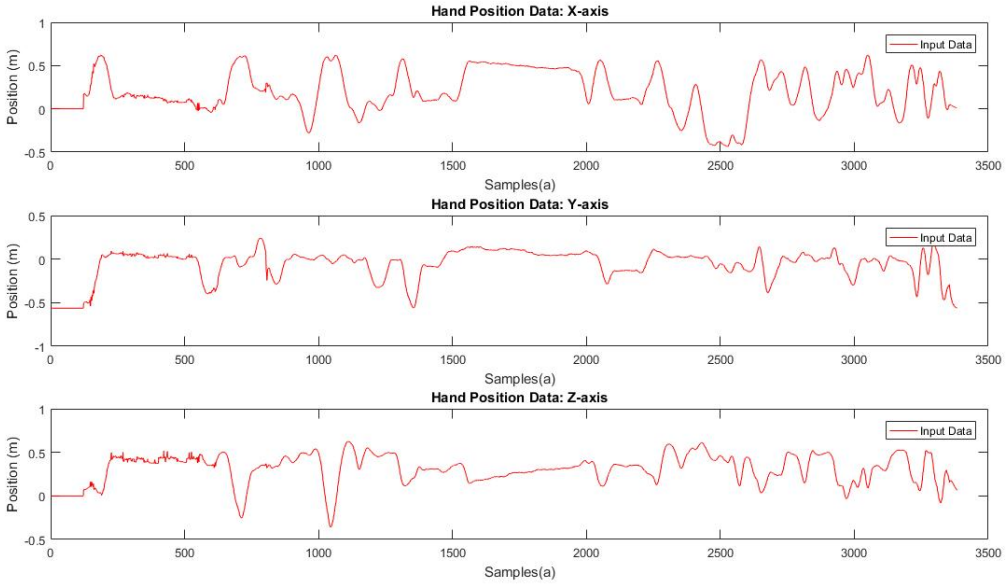

Figura 26 – Dados da posição da mão, gravados em um arquivo *bag*.
Fonte: Autor deste trabalho.

As informações presentes nos arquivos *.bags* foram enviadas ao ROS. Vale ressaltar que a orientação das juntas no Kinect são diferentes em relação à orientação convencional. O eixo X do Kinect corresponde ao eixo y convencional, o eixo Z do Kinect representa o eixo X e o eixo Y do Kinect equivale ao eixo Z. Essa informação foi importante para utilizar corretamente os dados na hora da simulação.

Tabela 2 – Especificações dos ângulos das juntas do manipulador robótico ED-7220C

Junta da Base	310º
Junta do Ombro	+130º/-35º
Junta do Cotovelo	± 130º
Junta do Punho	360º
Abertura da Garra	55mm

Fonte: System ().

Tabela 3 – Parâmetros de DH do manipulador ED-7220C.

Junta(i)	θ_i	d_i	$a_{i,i+1}$	$\alpha_{i,i+1}$
1	θ_1	d_1=370mm	a_1=12mm	90º
2	θ_2	d_2=0	a_2=220mm	0
3	θ_3	d_3=0	a_3=220mm	0
4	θ_4	d_4=0	a_4=0	90º
5	θ_5	d_5=150mm	a_5=0	0

Fonte: Autor deste trabalho.

O passo seguinte foi criar o ambiente de simulação. O modelo utilizado foi do manipulador ED7220-C. Dessa forma foi criada uma representação deste manipulador no simulador V-REP com formas simples, de forma que estes atendam as configurações presentes na tabela 2 e também na tabela 3.

Os valores da tabela 3 também foram confirmados nos trabalhos Oualid e Oussama (2016) e Maneetham e Leng (2018). Juntamente com os valores da matriz Homogênea. Dessa forma, as seguintes matrizes de transformação são dadas pelas equações 3.1-3.5:

$$T_0^1 = \begin{bmatrix} C_1 & 0 & S_1 & a_1C_1 \\ S_1 & 0 & -C_1 & a_1S_1 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

$$T_1^2 = \begin{bmatrix} C_2 & -S_2 & 0 & a_2C_2 \\ S_2 & C_2 & 0 & a_2S_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

$$T_2^3 = \begin{bmatrix} C_3 & -S_3 & 0 & a_3C_3 \\ S_3 & C_3 & 0 & a_3S_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

$$T_3^4 = \begin{bmatrix} C_4 & 0 & S_4 & 0 \\ S_4 & 0 & -C_4 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

$$T_4^5 = \begin{bmatrix} C_5 & -S_5 & 0 & 0 \\ S_5 & C_5 & 0 & 0 \\ 0 & 0 & 1 & d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

Além disso, internamente no simulador V-REP, foi possível atribuir um algoritmo em LUA para controlar e manipular dados no projeto criado. A figura **??** representa o modelo criado. Este modelo foi utilizado em todos os métodos sugeridos neste trabalho, com exceção das equações dinâmicas.

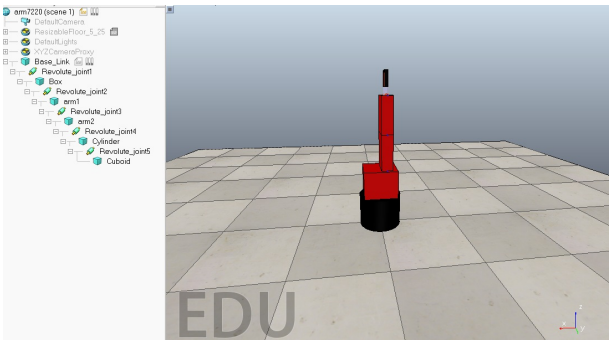

Figura 27 – Modelo criado do manipulador ED-7220C.
Fonte: Autor deste trabalho.

3.2.1 Metodologia: Cinemática

O primeiro método a ser analisado foi a cinemática direta. Tendo o objetivo de utilizar as informações dos rotacionais das juntas do ombro e do cotovelo vistas nas figuras 24 e 25, e passar para uma junta específica do modelo do manipulador. Dessa forma foi necessário relacionar os dados do eixo y do ombro a junta azimutal do manipulador, comumente chamada de junta de revolução 1; o eixo z da junta do ombro foi atribuído à junta 2, e o eixo y do cotovelo foi responsável pela terceira junta de revolução. Esta por ser a primeira técnica foi o principal método para comparar diversas variações dos filtros de Kalman e média móvel, para verificar qual teve o melhor desempenho em eliminar as oscilações vistas nas figuras 24 e 25. Assim, foi aplicado em primeira instância o filtro média móvel com diferentes janelas (6, 10 e 14). O pseudocódigo presente no apendice 6 sintetiza a execução deste método com o filtro média móvel.

Para o método do filtro de Kalman, algumas variáveis precisaram ser estabelecidas. A primeira delas diz respeito a covariância do sensor, onde na teoria é representada como

sendo R_k. Este valor como pode ser visto em Nguyen, Izadi e Lovell (2012), é apontado como o erro no eixo x. Para uma distância de 2.0m, o valor do erro foi de $\pm 0.006m$ enquanto erro em y e z para a mesma distância é dada como sendo $\pm 0.002m$, estes valores de erro foram atribuídos a variável do ruído do sensor v_k . Já o segundo fator foi a atribuir à matriz H o valor de 1 em todas as análises. E por fim, foi tratado cada variável de maneira separada, tornando as matrizes utilizadas no filtro como variáveis escalares, facilitando a implementação do filtro em código. Dessa forma, na primeira análise foi fixada a covariância do modelo Q_k em 0.8, 0.08 e 0.008, e também utilizada um modelo baseado na derivada, ou seja, $\hat{x}_k - \hat{x}_{k-1}$. Isso garante um modelo linear e torna um sistema flexível para qualquer conjunto de dados com diferentes valores de covariância do modelo. O algoritmo criado foi semelhante. O algoritmo proposto se assemelha ao demonstrado para o filtro média móvel, modificando apenas as funções para aplicar o filtro de Kalman, e consequentemente a inicialização das variáveis, como pode ser observado no pseudocódigo encontrado no apendice 7

Para o filtro de Kalman robusto, essa função foi alterada para abordar as equações conforme a seção 2.4.3.

Diferente da cinemática direta, os dados fornecidos pelo *openni_listener* ou pelo rosbag para o método da cinemática inversa foram atribuídos para um objeto esférico. Dessa forma foi criado uma relação deste objeto com a extremidade livre do manipulador. A partir dessa relação criada, foi possível aplicar o método D.L.S presente em *Tool> calculation Modules* no simulador, para realizar o cálculo dos ângulo das juntas necessárias para que a extremidade livre alcance o objeto esférico, conforme é apresentado na figura **??**.

Com isso, a função *simHandleIKGroup* pôde ser utilizada.

3.2.2 Metodologia: Lógica *fuzzy*

Para o método da lógica *Fuzzy*, foi tomado como base o trabalho de Santos et al. (2017), que utilizou a ferramenta actionlib do ROS para controlar a posição das juntas do motor. Assim, foi criado um arquivo do tipo *Client* e *Server* para utilizar as funções do actionlib, além disso também foi utilizado a biblioteca fuzzylite[2], uma biblioteca em C++ para utilizar a lógica *Fuzzy*. Também foi feita uma modificação no código fornecido pela referência, uma vez que este controlava apenas a posição de um motor, e neste trabalho foi necessário controlar três juntas.

Além disso, foi necessário enviar os dados do rosbag ou do *openni_listener* para o V-REP, e de lá enviar os dados para o *FuzzyClient*. Com o intuito de organizar os dados do ombro e do cotovelo para uma única mensagem para que este seja representado como

[2] https://fuzzylite.com/

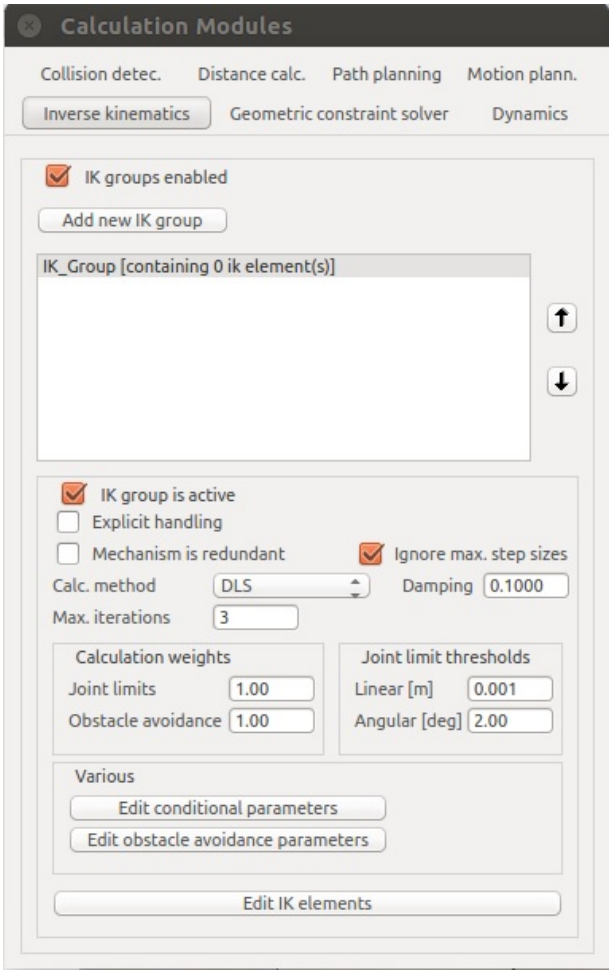

Figura 28 – *Layout* do V-REP para a implementação do método D.L.S.

o *goal* para os programas do ROS. O *FuzzyClient* enviam estes dados como objetivo para o *FuzzyServer* que é responsável por realizar os cálculos da lógica *fuzzy* , enviando para o arquivo LUA presente no V-REP, que serviu apenas para receber e enviar os dados para o ROS por meio do ROSInterface do V-REP. A figura 29 representa a estrutura que os códigos estavam enviando suas mensagens.

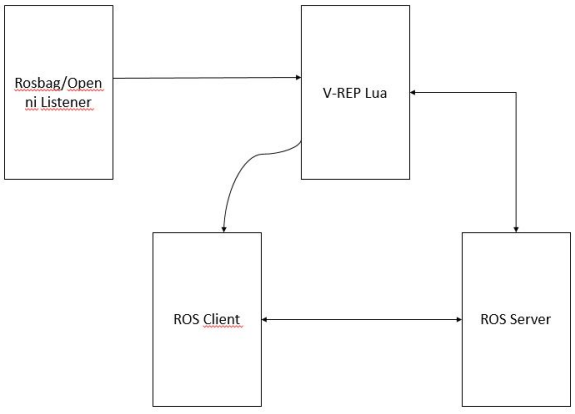

Figura 29 – Estrutura utilizada para o controle com lógica *Fuzzy*.
Fonte: Autor deste trabalho.

Para a fuzzificação, foram utilizadas as entradas de erro e derivada do erro para cada junta como entrada. Sendo que o erro foi obtido através da subtração das posições da junta atual e anterior, e a derivada do erro é a subtração do erro atual e do anterior. Os valores de entrada foram mantidos de acordo com a referência, mas os valores de saída foram modificados para se adequarem ao projeto proposto, criando uma saída para cada junta e essa sendo limitada para o alcance de cada junta conforme a tabela 2. Dessa forma, as figuras 30 e 31 representam os parâmetros de entrada, enquanto as figuras 32, 34 e 34 representam os parâmetros de saída na lógica *Fuzzy*.

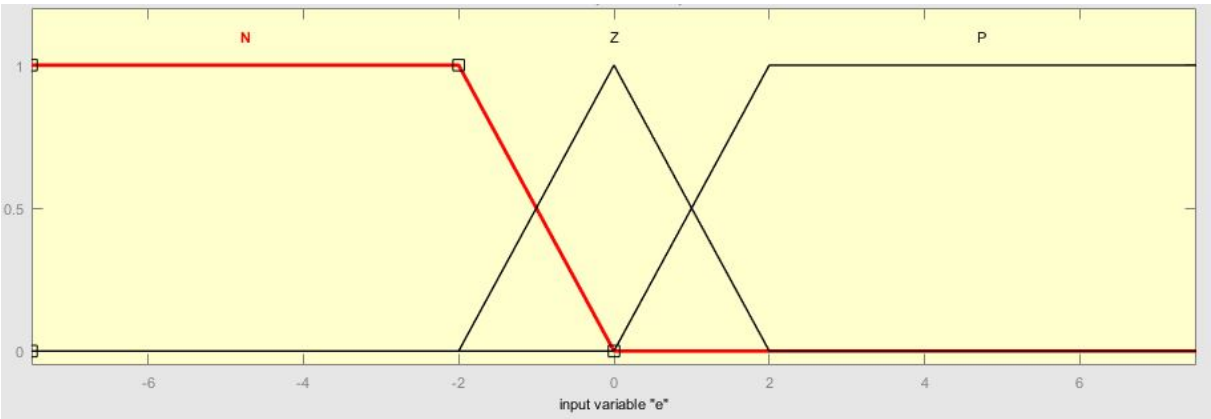

Figura 30 – Parâmetro de entrada da fuzzificação: Erro.
Fonte: Santos et al. (2017).

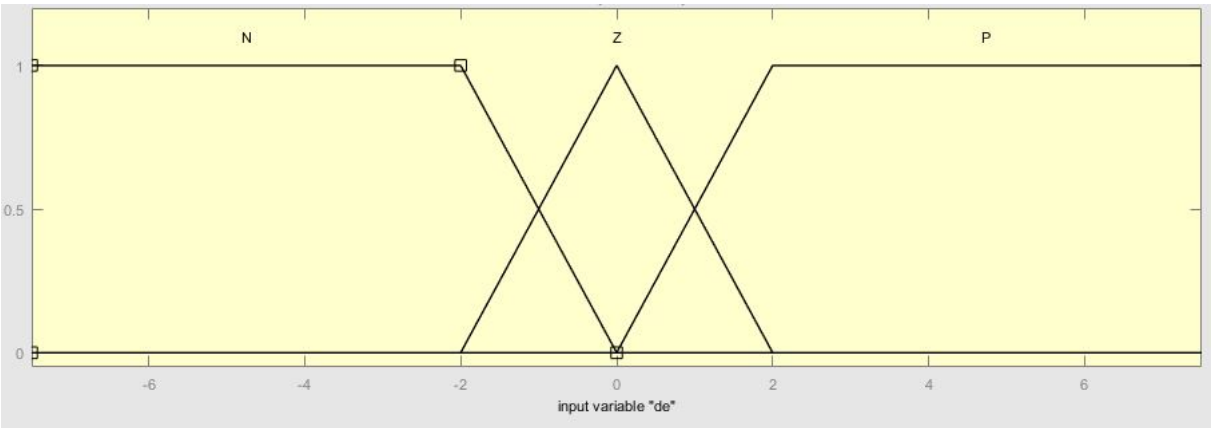

Figura 31 – Parâmetro de entrada da fuzzificação: dErro.
Fonte: Santos et al. (2017).

Como observado nas figuras 30, 31, os valores de entrada erro e derivada do erro foram caracterizados como negativo (N), zero (Z) e positivo (P). Enquanto as valores de saída das juntas presentes nas figuras 32, 33 e 34 foram categorizados como negativo-grande (NG), negativo-pequeno (NP), zero (Z), positivo-pequeno (PP) e positivo-grande (PG). AS regras utilizadas para a tomada de decisões não sofreram alterações em relação à referência. Dessa forma, a tabela 4 apresentam as regras utilizadas para o sistema de inferências.

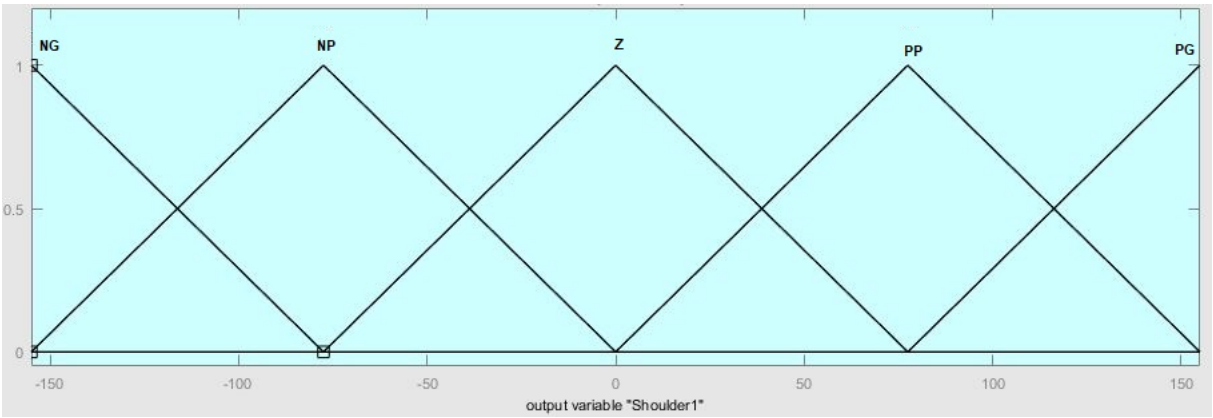

Figura 32 – Parâmetro de saída da fuzzificação: Junta1.
Fonte: Santos et al. (2017).

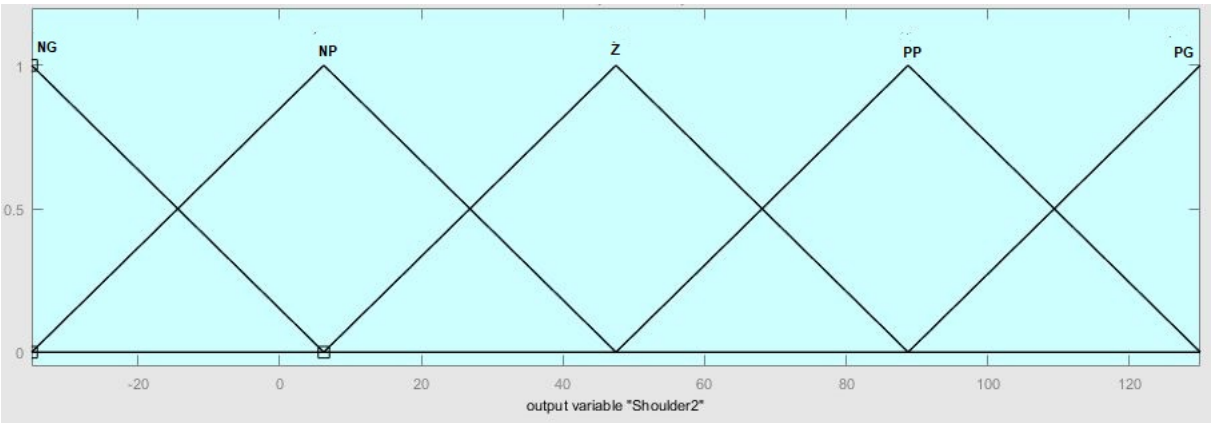

Figura 33 – Parâmetro de saída da fuzzificação: Junta2.
Fonte: Santos et al. (2017).

Tabela 4 – Regras utilizadas no controle por lógica *Fuzzy*.

e / de	N	Z	P
N	NG	NP	Z
Z	NP	Z	PP
P	Z	PP	PG

Fonte: Santos et al. (2017).

Dessa forma, o pseudocódigo "fuzzy_vrep"mostra como o algoritmo em LUA faz a comunicação com o *Fuzzy client* e o *Fuzzy server*.Já pseudocódigo "fuzzy_client"mostra a construção do arquivo *client* do actionlib. E por fim, a parte *server* do actionlib é apresentado no pseudocódigo "fuzzy_server", todos os pseudocódigos presentes no apendice 8.

3.2.3 Metodologia: Dinâmica de manipuladores

Na configuração da dinâmica de manipuladores, algumas mudanças foram realizadas para a execução deste método. A primeira modificação foi o modelo do manipulador

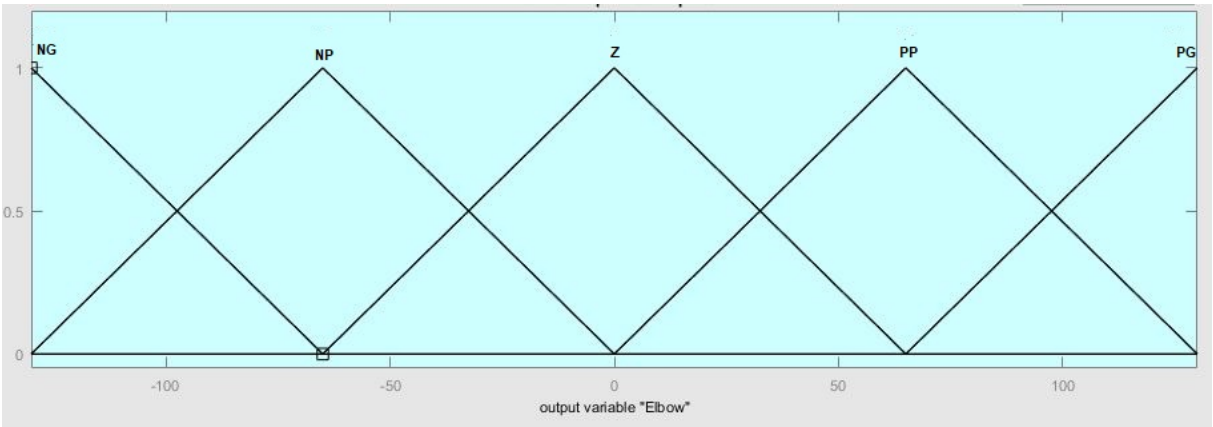

Figura 34 – Parâmetro de saída da fuzzificação: Junta3.
Fonte: Santos et al. (2017).

utilizado nas demais técnicas, que não pôde ser utilizado na dinâmica devido à colisão dos objetos. Optou-se inicialmente por utilizar um dos modelos existentes, porém o modelo encontrado mais semelhante ao ED-7220C sofria problemas de colisão interna, fazendo as peças se soltarem ao longo da simulação. Dessa forma foi criado um modelo simples com 3 juntas, sendo essas dispostas da mesma forma que no manipulador ED-7220C. A figura 35 mostra o modelo criado.

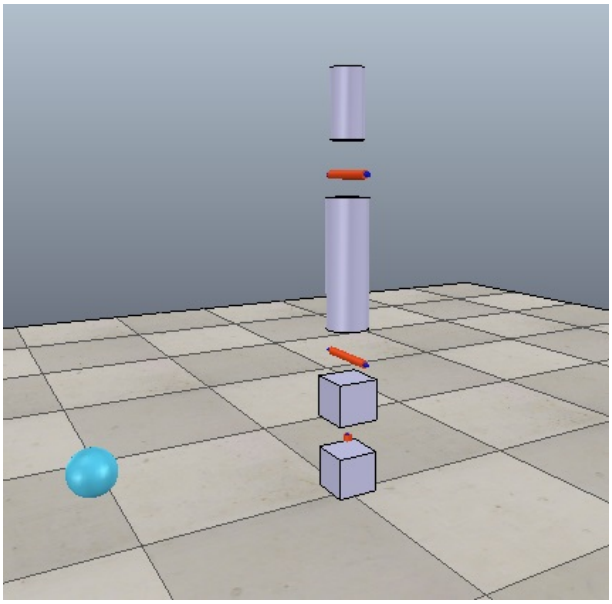

Figura 35 – Modelo do manipulador utilizado para a dinâmica.
Fonte: Autor deste trabalho.

Outra modificação foi a utilização de 3 juntas. Essa modificação foi feita porque além de diminuir o esforço computacional, este também satisfaz todas as juntas que o Kinect consegue rotacionar. Em contra partida, utilizar apenas 3 juntas faz o manipulador ter diversos pontos em que ele não consegue alcançar. A matriz DH deste modelo é representado pela tabela 5.

Aplicando os dados presentes na tabela 5 e encontrando a matriz de transformação

Tabela 5 – Parâmetros de DH do manipulador utilizado na dinâmica.

Junta(i)	θ_i	d_i	$a_{i,i+1}$	$\alpha_{i,i+1}$
1	$\theta_1 = \pm 175$	d_1=140mm	a_1=0	90º
2	$\theta_2 = \pm 160$	d_2=0	a_2=420mm	0
3	$\theta_3 = \pm 170$	d_3=0	a_3=225mm	0

Fonte: Autor deste trabalho.

baseada em 3.1, 3.2 e 3.3, foi encontrada assim a seguinte matriz Jacobiana, presente na equação 3.6.

$$J = \begin{bmatrix} -s_1(a_3c_{23} + a_2c_2) & -c_1(a_3s_{23} + a_2s_2) & -a_3s_{23}c_1 \\ c_1(a_3c_{23} + a_2c_t2) & -s_1(a_3s_{23} + a_2s_2 & -a_3s_{23}s_1 \\ 0 & a_3c_{23} + a_2c_2 & a_3c_{23} \\ 0 & s_1 & s_1 \\ 0 & -c_1 & -c_1 \\ 1 & 0 & 0 \end{bmatrix}_{6x3} \tag{3.6}$$

Ressaltando que $s_1 = sin(\theta_1)$, $c_{23} = cos(\theta_2 + \theta_3)$.

Também foi encontrado o Jacobiano de cada junta levando em consideração o centro de massa. Dessa forma, a equação 3.10, 3.11 e 3.12 representam os parâmetros x,y e z do centro de massa de cada junta.

$$com_1 = \begin{bmatrix} \frac{a_1c_1}{2} \\ \frac{a_1s_1}{2} \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.7}$$

$$com_2 = \begin{bmatrix} \frac{a_2c_2}{2} \\ 0 \\ \frac{a_2s_2}{2} \end{bmatrix} \tag{3.8}$$

$$com3 = \begin{bmatrix} \frac{a_3c_3}{2} \\ 0 \\ \frac{a_3s_3}{2} \end{bmatrix} \tag{3.9}$$

Dessa forma, multiplicando cada junta pela sua matriz de transformação, levando em consideração que $T_1^1 = \mathrm{I}$, são dadas as seguintes equações:

$$Tcom_1 = T_0^1 \cdot com_1 = \begin{bmatrix} \frac{a_1c_1}{2} \\ \frac{a_1s_1}{2} \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.10}$$

$$Tcom_2 = \begin{bmatrix} \frac{a_2c_1c_2+a_2s_1s_2}{2} \\ \frac{a_2c_1s_2+a_2c_2s_1}{2} \\ d1 \end{bmatrix} \tag{3.11}$$

$$Tcom3 = \begin{bmatrix} a_2c_1c_2 + \frac{a_3s_1s_3+a_3c_1c_2c_3}{2} \\ a_2c_2s_1 - \frac{a_3c_1s_3+a_3s_1c_2c_3}{2} \\ d_1 + a_2s_2 + \frac{a_3c_3s_2}{2} \end{bmatrix} \tag{3.12}$$

Para que a multiplicação das equações 3.7, 3.8 e 3.9 acontecesse, foi necessário adicionar uma linha com o valor 1 nas equações 3.10, 3.10 e 3.10. Com essas equações foi possível encontrar as equações do Jacobiano para o centro de massa. Sendo eles as equações 3.13, 3.14, 3.15.

$$J_{com1} = \begin{bmatrix} \frac{-a_1s_1}{2} & 0 & 0 \\ \frac{a_1c_1}{2} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{3.13}$$

$$J_{com2} = \begin{bmatrix} \frac{a_2c_1s_2-a_2c_2s_1}{2} & \frac{a_2c_2s_1-a_2c_1s_2}{2} & 0 \\ \frac{a_2c_2c_1+a_2s_1s_2}{2} & \frac{-a_2c_2c_1-a_2s_1s_2}{2} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.14}$$

$$J_{com3} = \begin{bmatrix} -a_2c_2s_1 + \frac{a_3c_1s_3-a_3c_2c_3s_1}{2} & -a_2c_1s_2 - \frac{a_3c_1c_3s_2}{2} & \frac{a_3c_3s_1-a_3c_1c_2s_3}{2} \\ a_2c_1c_2 + \frac{a_3s_1s_3+a_3c_1c_2c_3}{2} & -a_2s_1s_2 - \frac{a_3c_3s_1s_2}{2} & \frac{-a_3c_1c_3-a_3c_2s_1s_3}{2} \\ 0 & a_2c_2 + \frac{a_3c_2c_3}{2} & \frac{-a_3s_2s_3}{2} \\ 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.15}$$

As equações da posição da junta foram dadas pela última coluna da matriz homogênea, presente na equação 3.16.

$$EE_{xyz} = \begin{bmatrix} a_2c_1c_2 + a_3c_1c_2c_3 - a_3c_1s_2s_3 \\ a_2c_2s_1 + a_3c_2c_3s_1 - a_3s_1s_2s_3 \\ d_1 + a_2s_2 + a_3c_2s_3 + a_3c_3s_2 \end{bmatrix} \tag{3.16}$$

Ao definir as equações, foi possível realizar o controle do manipulador. Os valores de massa e de inércia foram obtidos nas propriedades dos elos dentro do simulador, sendo que a massa do elo 1 (m1) foi de 0.15Kg, m2 = 1.171Kg e m3= 0.229Kg. Enquanto os valores da Inércia do elo 1 foram: $I_{xx} = I_{yy} = I_{zz} = 1.7e^{-3}$; o valor da inércia para o elo 2 foram: $I_{xx} = I_{yy} = 8.1e^{-3} \quad e \quad I_{zz} = 1.3e^{-3}$; e os valores para o elo 3 foram: $I_{xx} = I_{yy} = 5.6e^{-4} \quad e \quad I_{zz} = 1.8e^{-4}$.

Tendo o modelo definido, duas abordagens foram realizadas. A primeira abordagem se assemelhou com o que foi feito na lógica *Fuzzy*. Para isso foi utilizada uma biblioteca em C++ para os cálculos matriciais chamada Armadillo[3]. Para a configuração dinâmica foi utilizado um protótipo de um manipulador com 3 juntas rotativas, representando todas as juntas que o kinect do Xbox 360 consegue rotacionar. A figura 36 mostra a configuração deste protótipo.

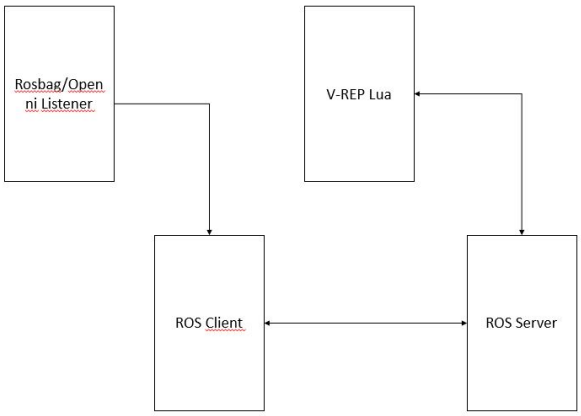

Figura 36 – Estrutura utilizada para a técnica da dinâmica: Actionlib.
Fonte: Autor deste trabalho.

Assim como na lógica *fuzzy*, os algoritmos *client* e *server* são muito parecidos, alterando apenas a função "ExecutarCalculos"presente no *Server*, em que as equações da dinâmica e do controlador proporcional-derivativo foram adicionadas como pode ser observado no fragmento do pseudocódigo presente no apendice 9. Além das declarações das variáveis utilizadas.

Além dos algoritmos baseado no actionlib, foi criado uma nova abordagem para essa técnica. Criou-se um *python script* do ROS para realizar os cálculos e controlar a simulação, fazendo uso do *remoteAPI* do V-REP, não alterando as equações utilizadas. Dessa forma o tráfego de dados se limitará apenas as informações do rosbag ou do *openni_listener* para o arquivo ROS *python*, como pode ser observado na figura 37.

Dessa forma, o pseudocódigo *"pythondym"* também presente no apendice 9 diz respeito à abordagem utilizando *python script*.

[3] http://arma.sourceforge.net/

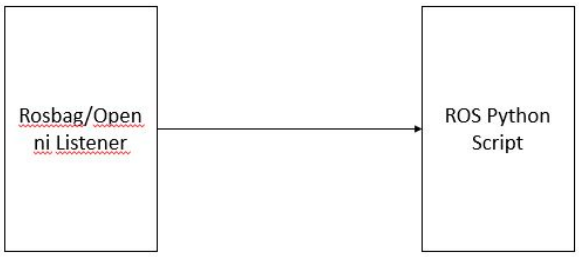

Figura 37 – Estrutura utilizada para a técnica da dinâmica: *Python Script*.
Fonte: Autor deste trabalho.

Para a realização deste trabalho foi utilizado um computador com um processador Intel Core 2 duo de 2.93 GHz, 4GB RAM DDR2 e uma placa de vídeo NVidia Geforce GTX 760. A versão utilizada do ROS foi o *Indigo*, disponível para o sistema operacional Linux 14.04, a versão utilizada do V-REP foi 3.3.2-*Educational* e o dispositivo Kinect utilizado foi a versão do X-box 360.

4 Resultados e Análises dos Dados

Neste Capítulo se encontram os resultados obtidos neste trabalho. Na seção 4.1 temos os resultados obtidos utilizando a cinemática direta, também se encontram as comparações de filtros utilizados, sendo eles o filtro Média Móvel, filtro de Kalman e filtro de Kalman robusto. Já na seção 4.2, estão os resultados encontrados na cinemática inversa. Em seguida há os resultados do controle por lógica *Fuzzy* na seção 4.3. Finalizando com os resultados da Dinâmica de Manipuladores, presentes na seção 4.4.

4.1 Resultados: Cinemática Direta

4.1.1 Cinemática direta: Filtro Média Móvel

Como este foi o primeiro método aplicado e também por conta do problema de pico de sinal encontrado na região do ombro, observado também por Cavalcante (2012), este foi o modelo utilizado para a comparação dos filtros digitais, que foram utilizados para minimizar ou até eliminar este problema. Neste cenário foram utilizados três tipos de filtros digitais. O primeiro foi o de Média Móvel, pois além de ser um filtro simples, ele é um excelente filtro para tratar ruído branco. Assim foi projetado um filtro de Média Móvel variando a janela entre 6 , 10 e 16 amostras. O resultado de cada junta foi criado para as 3 janelas propostas e as simulações tiveram um tempo de amostragem de 50 ms. As figuras 38, 39 e 40 mostram o resultado para a janela de 6, 10 e 14 amostras respectivamente.

A primeira observação que pode ser feita é com relação ao atraso do sinal de saída em relação ao sinal de entrada. Este atraso não foi possível identificar o motivo de sua existência nem mensurar o quanto, pois este atraso aumentou com o tempo de simulação. Nota-se também que a diferença entre as janelas não implicaram em uma diferença drástica do ruído no sinal de saída. A única diferença que notável foi verificando os vídeos criados da simulação, onde se viu o filtro com a janela de 6 amostras[1] tendo maior problema de tremulação que o filtro de 10 amostras[2], o filtro de 14 amostras[3] apresentou um *delay* maior, porém a mesma redução nas tremulações que a janela de 10 amostras. Os valores do erro quadrático médio (EQM) e também da variância do erro de todas as configurações do filtro Média Móvel se encontra na tabela 6.

Ao analisar o relação de erro e variância, percebe-se um pequeno aumento de ambos os valores para uma maior número de janelas. O que condiz com a teoria, pois as

[1] Cinemática direta: Média Móvel(janela de 6 amostras) - https://youtu.be/HEZps5cnnr4
[2] Cinemática direta: Média Móvel (janela de 10 amostras) - https://youtu.be/UM7e5DO53IE
[3] Cinemática direta: Média Móvel (janela de 14 amostras) - https://youtu.be/79kGrNJ91pE

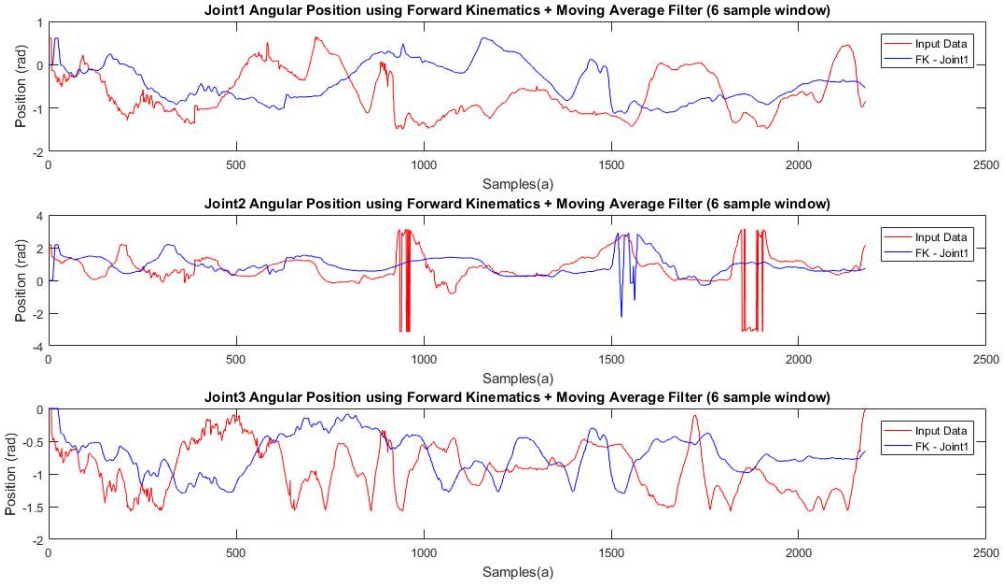

Figura 38 – Cinemática direta: filtro Média Móvel (janela de 6 amostras).
Fonte: Autor deste trabalho.

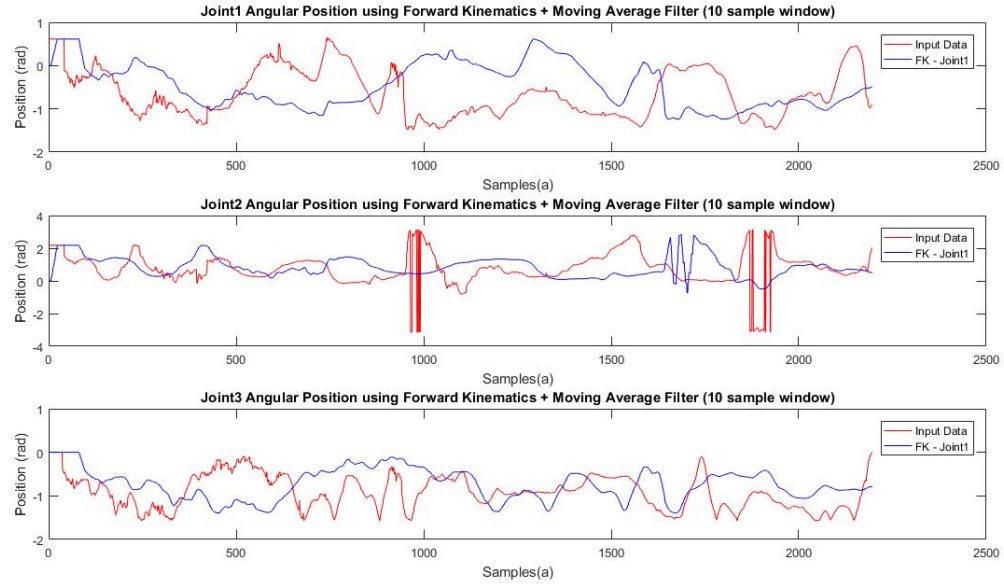

Figura 39 – Cinemática direta: filtro Média Móvel (janela de 10 amostras).
Fonte: Autor deste trabalho.

janelas além de diminuir o sinal ruidoso (aumentando o erro em relação ao sinal ruidoso), também cria um deslocamento no sinal, fazendo com que a variância aumente. Dessa forma a configuração com 10 amostras acabou sendo a escolhida como opção de filtro para as possíveis aplicações. os valores de EQM e variância foram próximos, este visualmente apresentou o resultado com a melhor relação de redução de ruído em relação ao atraso acarretado pela janela de amostras. O caso da instabilidade presente na segunda junta não foi resolvido por esta técnica de filtro, porém isso já era esperado devido à simplicidade

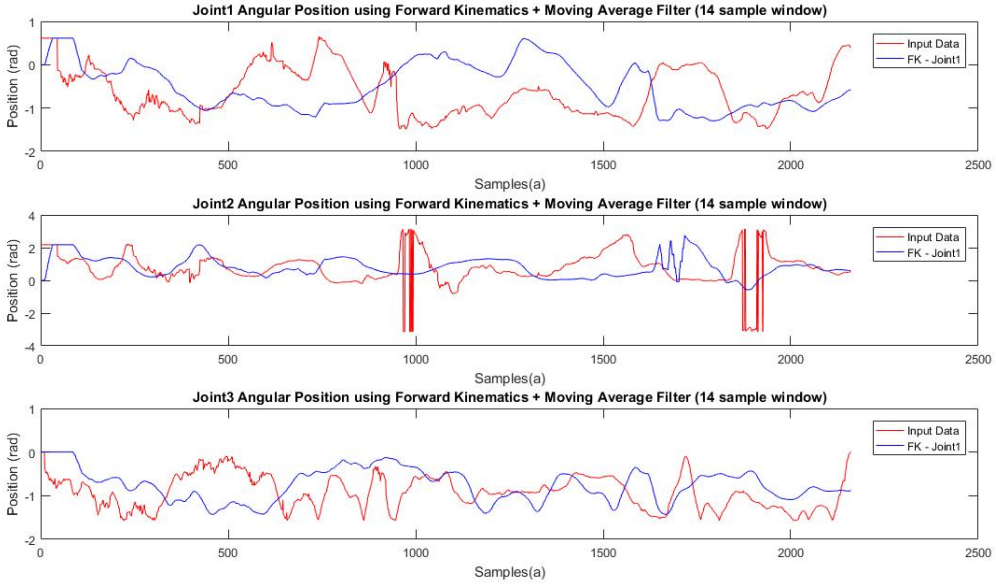

Figura 40 – Cinemática direta: filtro Média Móvel (janela de 14 amostras).
Fonte: Autor deste trabalho.

Tabela 6 – Erro quadrático médio e variância da cinemática direta - filtro Média Móvel.

Método	EQM	Variância(E)
Cinemática Direta: Média Móvel (6 amostras)	0.6678	0.1865
Cinemática Direta: Média Móvel (10 amostras)	0.72	0.19
Cinemática Direta: Média Móvel (14 amostras)	0.7372	0.1955

Fonte: Autor deste trabalho.

que este filtro apresenta.

4.1.2 Cinemática direta: Filtro de Kalman

Em seguida foi feito a mesma análise para o filtro de Kalman. Este por sua vez foi modificado apenas a covariância do modelo Q_k, variando ele entre 0.8, 0.08 e 0.008. A simulação também foi executada com um tempo de amostragem de 50 ms e os resultados dessas variações podem ser observados nas figuras 41,42 e 43 respectivamente.

É possível observar que a utilização do Filtro de Kalman teve alguns resultados diferentes entre as opções apresentadas. Para Q=0.8[4], este não teve uma suavização do sinal tão grande quanto o resultado de Q=0.08[5]. O resultado da covariância de 0.008[6] apresentou diversas oscilações, ao ponto de ser uma opção descartável. Segundo Ting, Theodorou e Schaal (2007), mostra em seu trabalho que o filtro de kalman convencional não consegue lidar bem com *outliers*, isso justifica o porquê do Filtro de Kalman em sua melhor configuração não conseguir eliminar tal problema. Por isso foi utilizado uma varia-

[4] Cinemática direta: filtro de Kalman: Q=0.8 - https://youtu.be/ivSMz3HFIj0
[5] Cinemática direta: filtro de Kalman: Q=0.08 - https://youtu.be/uN2iCZFomRY
[6] Filtro de Kalman: Q=0.008 - https://youtu.be/Q_3q-ls-KWE

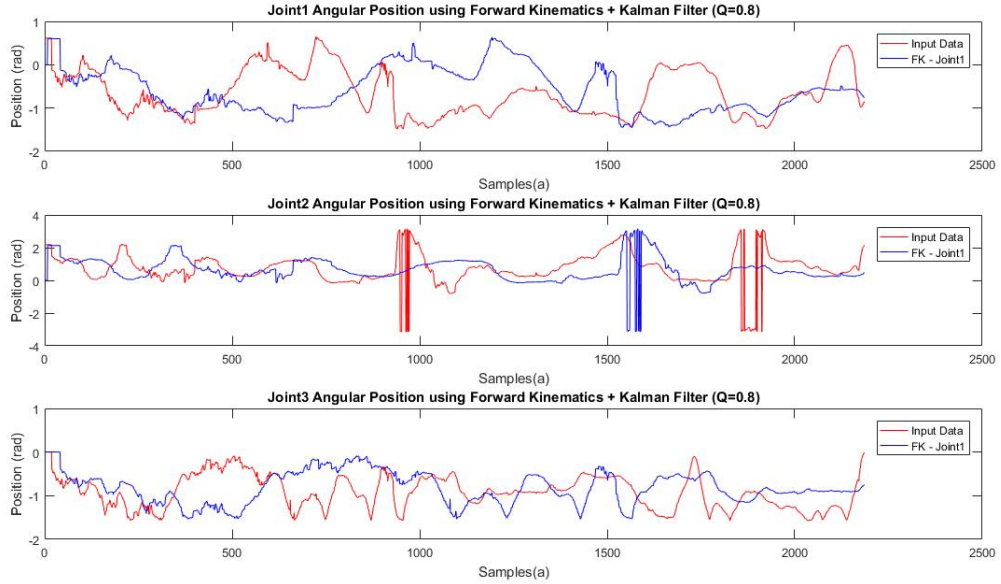

Figura 41 – Cinemática direta: filtro de Kalman (covariância Q=0.8).
Fonte: Autor deste trabalho.

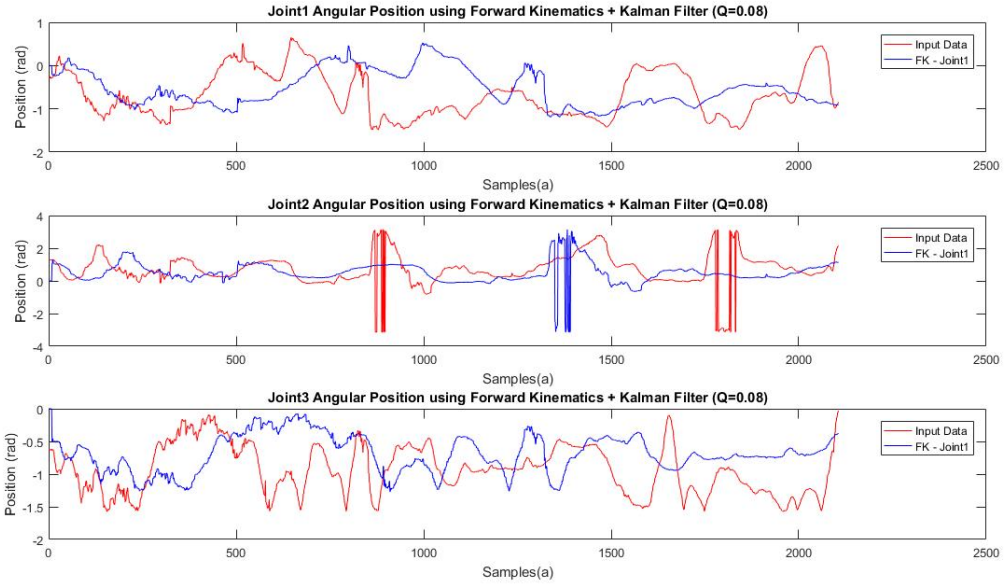

Figura 42 – Cinemática direta: filtro de Kalman (covariância Q=0.08).
Fonte: Autor deste trabalho.

ção do filtro de Kalman, que é próprio para este tipo de ocasião, tornando o método ainda mais robusto e também flexível, permitindo sua utilização em sistemas em tempo real. Assim, como mostrado na referência supracitada, foi atribuído que **A=H=1**, e **Q=0.01**, $a_{w_{k0}} = b_{w_{k0}} = 1$. A figura 44 mostra o resultado deste método apresentado.

A partir destas imagens, pode-se perceber que o modelo se tornou insensível às variações do sinal de entrada. Este também resultou em uma completa redução do ruído, além de reduzir o impacto do *outlier* presente no ombro. A tabela 7 mostra os valores de

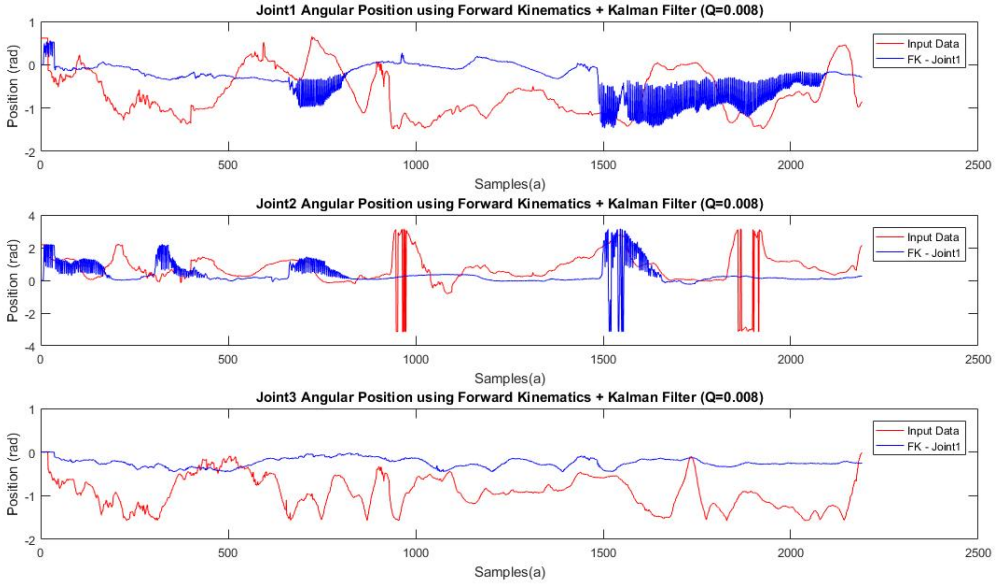

Figura 43 – Cinemática direta: filtro de Kalman (covariância Q=0.008)
Fonte: Autor deste trabalho.

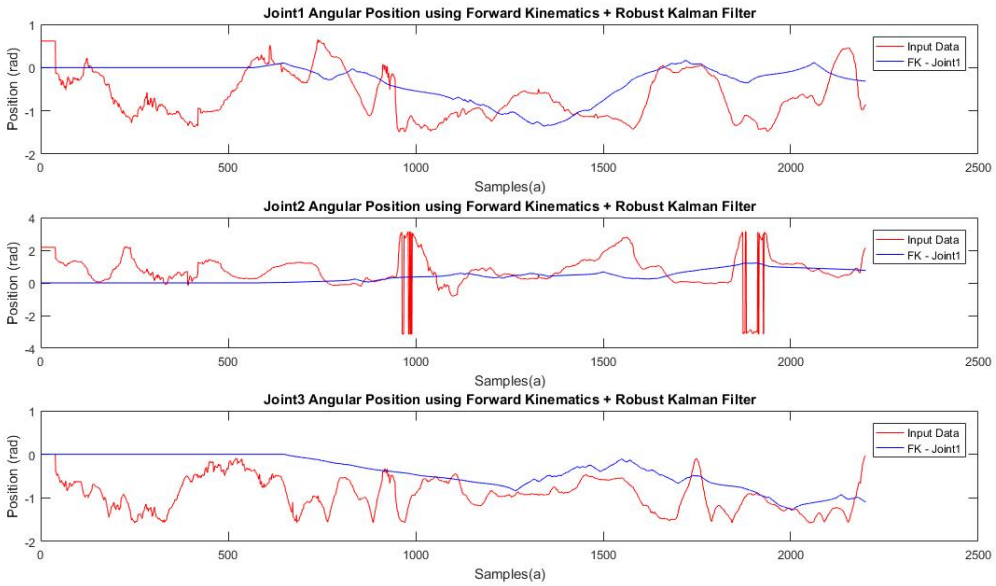

Figura 44 – Cinemática direta: filtro de Kalman robusto.
Fonte: Autor deste trabalho.

EQM e variância do erro para os métodos do filtro de Kalman utilizados.

De acordo com os valores da tabela 7, percebe-se que a variância do filtro de Kalman de Q=0.8 foi o maior resultado, em contra partida o filtro de Kalman robusto apresentou a menor variância, também tendo o menor valor de EQM não só em relação ao filtro de Kalman, mas também em relação a todos os métodos utilizados na cinemática direta. Sendo o Filtro de Kalman para Q=0.08 o segundo melhor resultado nestes parâmetros. Apesar do sinal de saída do filtro de Kalman robusto ter uma resposta mais lenta e

Tabela 7 – Erro quadrático médio e variância da cinemática direta - filtro de Kalman.

Método	EQM	Variância(E)
Cinemática direta: Filtro de Kalman(Q=0.8)	0.6606	0.2404
Cinemática direta: Filtro de Kalman(Q=0.08)	0.5875	0.1730
Cinemática direta: Filtro de Kalman(Q=0.008)	0.6577	0.1488
Cinemática direta: Filtro de Kalman Robusto	0.535	0.1272

Fonte: Autor deste trabalho.

Tabela 8 – Erro quadrático médio e variância da cinemática inversa.

Método	EQM	Variância(E)
Cinemática inversa: Média Móvel(janela de 10 amostras)	0.1866	0.0373
Cinemática inversa: Filtro de Kalman robusto	0.2534	0.0320

Fonte: Autor deste trabalho.

totalmente descaracterizada em relação ao sinal de entrada, este atinge os seus resultados sem criar instabilidades no manipulador, como é observado no vídeo [7] referente à esta técnica. Por fim, foi gravado um vídeo[8] mostrando a movimentação livre do manipulador utilizando a cinemática Direta com o Filtro de Kalman Robusto

4.2 Resultados: Cinemática Inversa

Fazendo a mesma análise da cinemática direta, agora para a cinemática inversa. Foi utilizado apenas a posição da mão, por esta ser a variável de interesse para a reprodução da cinemática inversa. Vale ressaltar também que o método utilizado para o cálculo da cinemática inversa foi o Levenberg-Marquardt, também conhecido como D.L.S. Dessa forma, as figuras 45 e 46 representam os sinais da posição da mão utilizando o filtro Média Móvel e o filtro de Kalman robusto respectivamente. Estes métodos foram os únicos utilizados, após feita a análise na cinemática direta, e o tempo de amostragem também foi de 50ms.

É observado que nestes dois casos, ambos os filtros se comportaram de maneira semelhante. E como neste caso não existe problemas de *outliers* ambos poderiam ser utilizados. A tabela 8 mostra os resultados de EQM e variância dos métodos utilizados para a cinemática inversa.

Analisando também o os dados de erro e variância, os dois modelos apresentaram valores muito próximos. Tendo o método da Média Móvel[9] menor EQM, enquanto o filtro de Kalman robusto [10] apresentou uma variância levemente menor. Vale ressaltar que neste trabalho não foi levada em consideração a orientação da extremidade livre, uma vez que

[7] Cinemática direta: filtro de Kalman robusto - https://youtu.be/HHbIUt2ZsbQ
[8] Movimentação livre: cinemática direta - https://youtu.be/K1zXSJo05z0
[9] Cinemática inversa: Média Móvel - https://youtu.be/F0d16CUTYWE
[10] Cinemática inversa: filtro de Kalman robusto - https://youtu.be/hmCvfuCFoEU

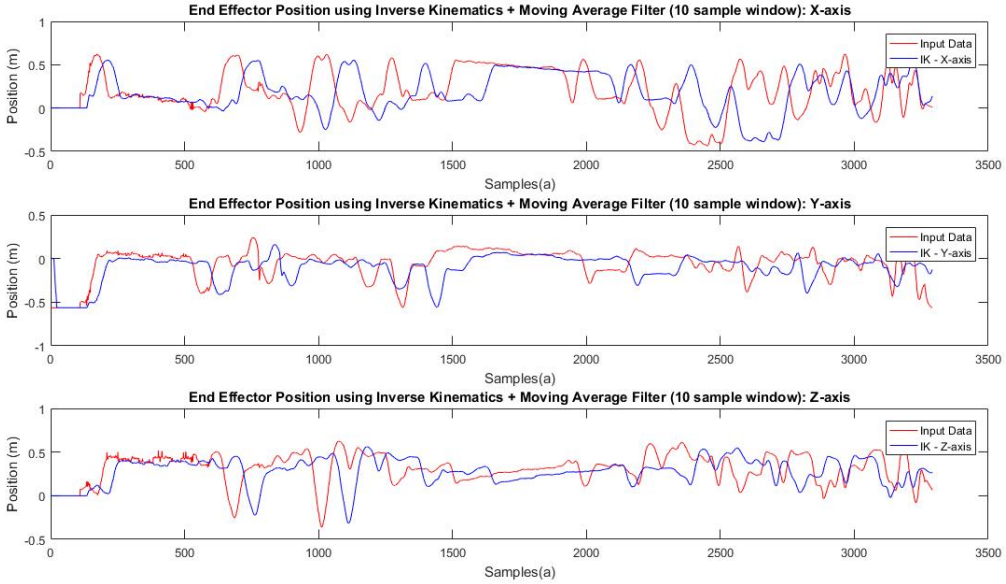

Figura 45 – Cinemática inversa: filtro Média Móvel (janela de 10 amostras).
Fonte: Autor deste trabalho.

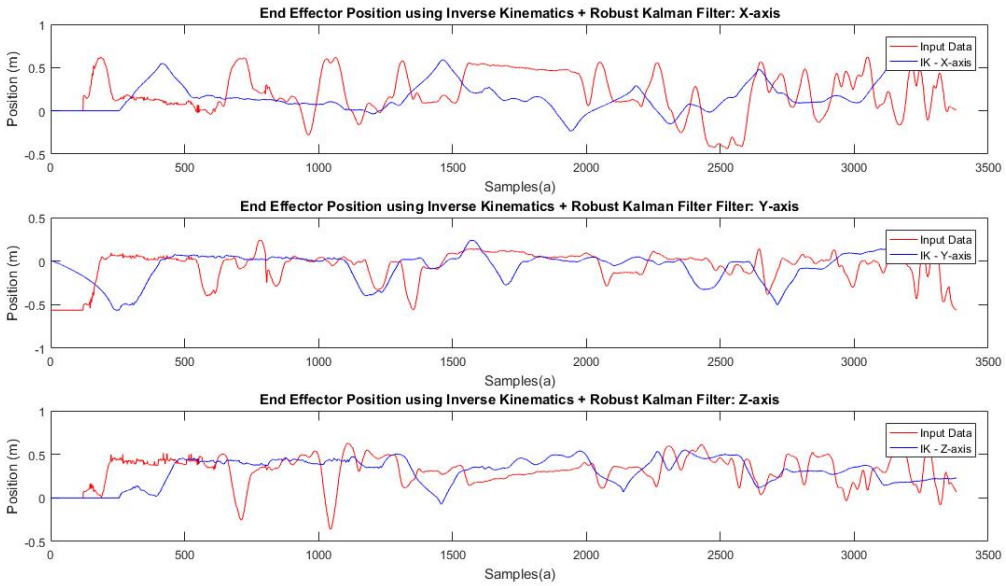

Figura 46 – Cinemática inversa: filtro de Kalman robusto.
Fonte: Autor deste trabalho.

o sensor Kinect não fornece a informação de rotação do punho. Um vídeo[11] foi gravada apresentando a movimentação livre utilizando o filtro Média Móvel com a cinemática inversa.

[11] Movimentação livre: cinemática inversa - https://youtu.be/HmVzNwVUTQw

Tabela 9 – Erro quadrático médio e variância do controle por lógica *fuzzy*.

Método	EQM	Variância(E)
Fuzzy s/ filtro	0.5650	0.1
Fuzzy c/ filtro	0.4630	0.0645

Fonte: Autor deste trabalho.

4.3 Resultados: Lógica *fuzzy*

Para os resultados da lógica *fuzzy*, primeiro foram gerados os gráficos sem a utilização de filtro. Assim, a figura 47 representa o gráfico de cada junta.

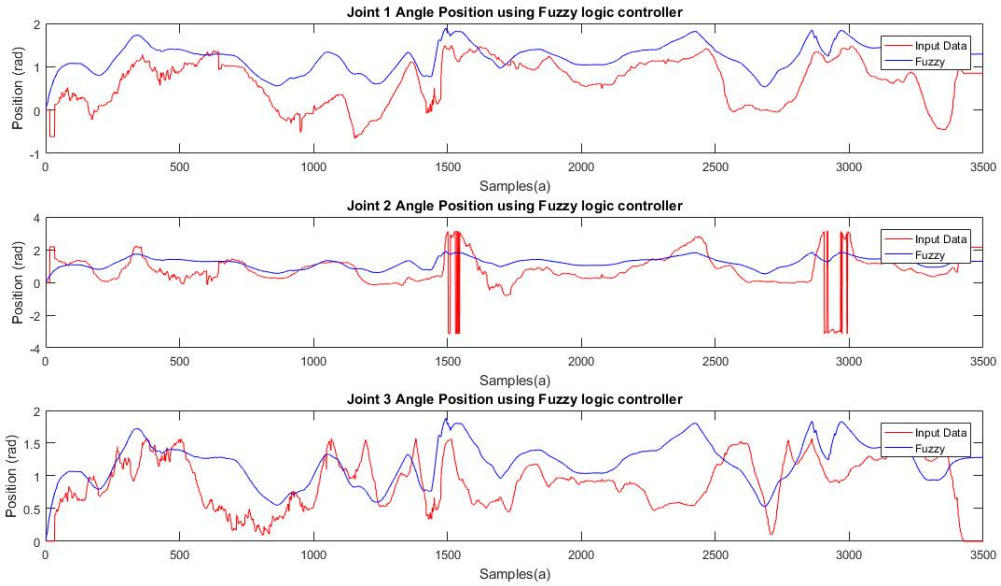

Figura 47 – Ângulo das juntas para um controle baseado em lógica *fuzzy*: sem filtro.
Fonte: Autor deste trabalho.

Em seguida fora adicionado o filtro de Kalman robusto, para verificar se haveria uma diferença no sinal de saída. Dessa forma, a figura 48 representa o sinal de saída com o filtro.

Analisando as figuras 47 e 48, percebe-se que em ambos os caso as oscilações foram filtradas, dispensando a utilização do filtro. Por mais que os resultados não apresentaram deslocamento em relação ao sinal de entrada, houve um *offset* considerável entre o valor de entrada e o valor de saída nos dois casos. Este resultado pode ser relacionado ao número de funções de pertinência das variáveis entrada ocuparem um grande espaço no intervalo. A tabela 9 indica os valores da variância e do EQM para este método.

Ao analisar os dados de erro e variância, é notável que o método *fuzzy* com filtro apresentou melhores resultados em relação ao mesmo método sem filtro. Outro agravante surgiu ao analisar os conteúdos do vídeo. Na lógica *fuzzy* sem filtro [12] foi mostrado uma

[12] Lógica *Fuzzy*: sem filtro - https://youtu.be/0MgyEQwWDh8

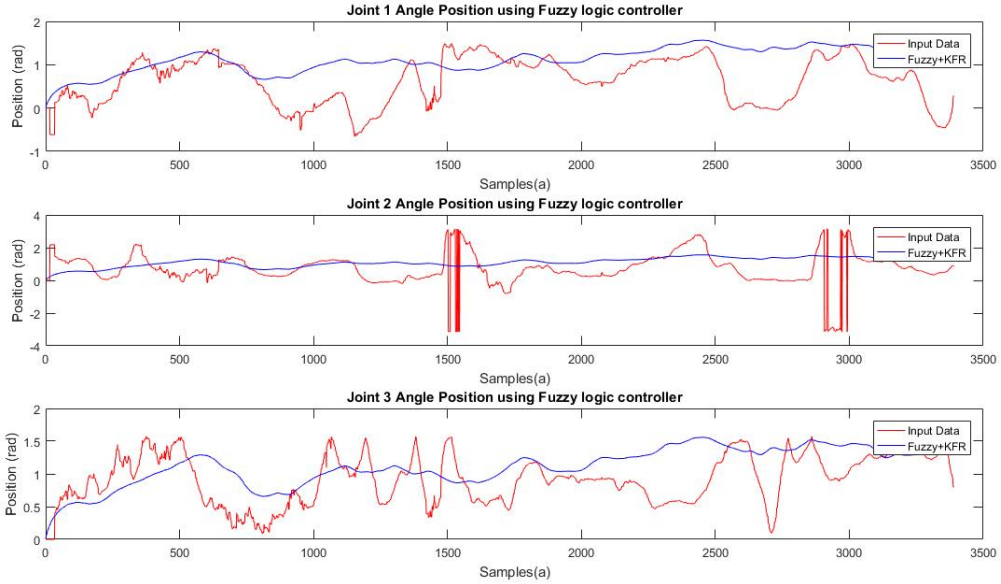

Figura 48 – Ângulo das juntas para um controle baseado em lógica *fuzzy*: filtro de Kalman robusto.

Fonte: Autor deste trabalho.

interdependência entre as juntas. Fazendo com que todas se movimentassem em conjunto. O mesmo foi verificado na lógica *fuzzy* com filtro [13]. Essa interdependência compromete a utilização deste método caso uma solução não seja feita. Um vídeo [14] foi criado para utilizar a lógica nebulosa de maneira livre.

4.4 Resultados: Dinâmica de Manipuladores

Por fim, para as equações dinâmicas, o primeiro caso foi a utilização do actionlib[15]. Dessa forma, a figura 49 representa a posição da extremidade do manipulador em relação à posição desejada para este método. Para este método a simulação teve um tempo de amostragem de 1 ms.

Ao analisar a figura 49 foi perceptível que o controle não funcionou. Pois mesmo com o objeto estando parado este fica sobre movimento oscilatório. E isso se persiste para qualquer valor de K_p e K_d utilizados. Ao utilizar um *debugger* para verificar o código, constatou-se que todos os cálculos estavam coerentes. Dessa forma foi creditado ao problema a ausência de sincronismo dos códigos. Pois, como há o transporte de mensagem do arquivo LUA para o arquivo *Server* e também existe troca de mensagens entre o *Client* e o *Server* por conta do actionlib. Muitas vezes os dados chegavam em seu destino com atraso, fazendo com que o manipulador executasse uma ação sendo que ela não era própria

[13] Lógica *Fuzzy*: com filtro - https://youtu.be/GaRZdJ_6_A4
[14] Movimentação livre: lógica *Fuzzy* - https://youtu.be/_xoVPLm7A6Q
[15] Dinâmica de manipuladores: Actionlib - https://youtu.be/-kQuuxIDbSg

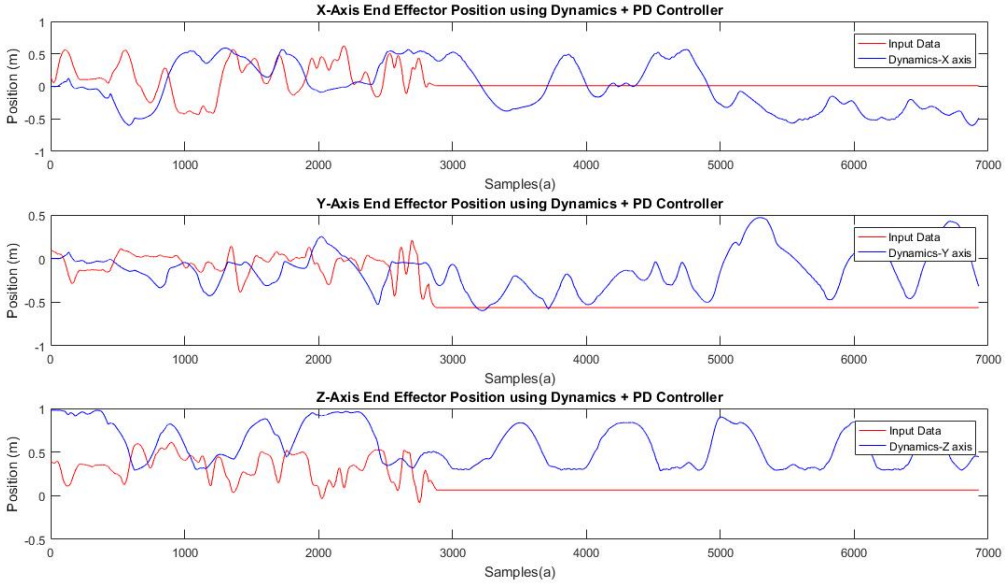

Figura 49 – Posição do *end-effector* para um controle PD utilizando as equações dinâmicas:actionlib.

Fonte: Autor deste trabalho.

para aquele determinado momento. Nos outros métodos este empecilho não surgiu pela simplicidade de cálculo, em relação aos cálculos dinâmicos.

Por conta deste percalço, foi sugerido a abordagem de calcular e controlar o passo da simulação em um único algoritmo. Esta segunda abordagem foi criada utilizando a linguagem *python*[16]. Sendo assim, a figura 50 representa o resultado da posição obtido nesta nova abordagem, mantendo também o tempo de amostragem de 1ms para a simulação.

Já nesta configuração, é possível perceber pela figura 50 que o sistema consegue atingir seu objetivo com uma pequeno *offset* no sinal quando este atinge a estabilidade. É perceptível também que há uma discrepância no sinal perto da amostra 500, por motivos do manipulador atingir sua restrição na posição de uma das juntas, fazendo necessário um replanejamento de rota. Perceba também que a saída teve um sinal sem ruídos, diferentemente do sinal de entrada.

Este resultado comprova que a mudança de estrutura dos códigos foi o principal motivo para o sucesso desta técnica, uma vez que não teve alteração nas equações. Porém fazer dessa forma demandou muito esforço computacional para conseguir gerar estes dados. Pois 1 milissegundo da simulação demorou entre 1 e 2 segundos no tempo real. Além disso o computador congelava a tela caso eu tentasse controlar livremente o manipulador, tornando ainda mais útil a utilização do arquivo rosbag. Isso pode ser explicado pelas configurações do computador utilizado na simulação e também com o que foi apresentado por Fu, Gonzalez e Lee (1987), em que menciona a complexidade computacional da equa-

[16] Dinâmica de manipuladores: *python script* - https://youtu.be/oyDI-G5GeFU

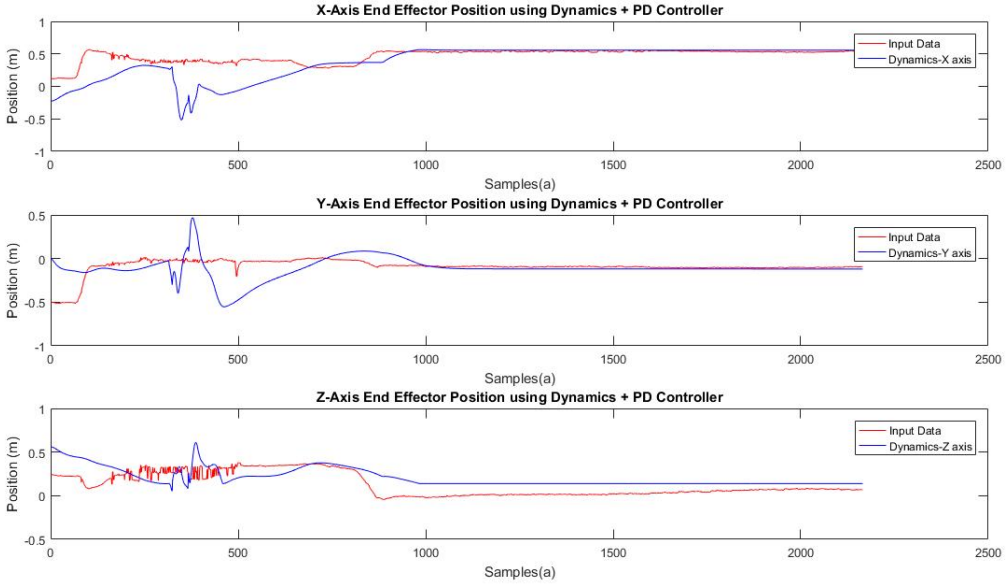

Figura 50 – Posição do *end-effector* para um controle PD utilizando as equações dinâmicas: *Python script*.

Fonte: Autor deste trabalho.

Tabela 10 – Erro quadrático médio e variância da dinâmica de manipuladores.

Método	EQM	Variância(E)
Dinâmica: Actionlib	0.3422	0.0475
Dinâmica: *Python script*	0.1229	0.0305

Fonte: Autor deste trabalho.

ção de Euler-Lagrange é de $O(n)^4$, sendo n o número de juntas. A tabela 10 apresenta os valores de EQM e variância do método em suas duas abordagens.

Pela análise de erro percebe-se a eficácia do método da Dinâmica, este pela segunda abordagem apresentou os melhores valores de EQM e também de variância. Apesar do método utilizando actionlib não ter funcionado, este entregou resultados melhores que alguns métodos utilizando os valores de rotação do ombro e cotovelo, porém foi o pior resultado para os métodos que utilizaram a posição da mão como os dados de controle.

Para concluir esta análise, a tabela 11 apresentam todos os dados de erro quadrático médio e da variância apresentados neste capítulo. Este por sua vez foi ordenado em ordem crescente em relação a variância do erro do sistema, pois essa variável representa melhor a eficácia do método utilizado do que o erro quadrático médio. Uma vez que esta variável se preocupa em retratar o quão constante um erro pode ser, frente às alterações do sinal de entrada e saída. Vale ressaltar também que o deslocamento presente nos resultados das cinemáticas direta e inversa contribuíram para um aumento dos valores destes dois parâmetros. Porém não foi possível mensurar quanto, pois este atraso aumentava com o tempo.

A partir da tabela 11, nota-se que os melhores resultados se encontram nos métodos

Tabela 11 – Erro quadrático médio e variância de todos os métodos.

Método	EQM	Variância(E)
Dinâmica: *Python script*	0.1229	0.0305
Cinemática inversa: Filtro de Kalman robusto	0.2534	0.0320
Cinemática inversa: Filtro Média Móvel	0.1866	0.0373
Dinâmica: Actionlib	0.3422	0.0475
Lógica *Fuzzy* s/ filtro	0.5650	0.1
Cinemática direta: Filtro de Kalman robusto	0.535	0.1272
Cinemática direta: Filtro de Kalman(Q=0.008)	0.6577	0.1488
Cinemática direta: Filtro de Kalman(Q=0.08)	0.5875	0.1730
Cinemática direta: Filtro Média Móvel(6 amostras)	0.6678	0.1865
Cinemática direta: Filtro Média Móvel(10 amostras)	0.72	0.19
Cinemática direta: Filtro Média Móvel(14 amostras)	0.7372	0.1955
Lógica *Fuzzy* c/ filtro	0.5430	0.1970
Cinemática direta: Filtro de Kalman(Q=0.8)	0.6606	0.2404

Fonte: Autor deste trabalho.

que utilizaram a informação da posição da mão, se assemelhando com a cinemática inversa.

5 Conclusão

A partir deste trabalho, foi possível observar o quão útil pode ser o dispositivo Kinect para o controle de manipuladores robóticos, se tornando um forte aliado para o treinamento de manipuladores em nível industrial, podendo também ser aplicado em situações de riscos ou tele operáveis e até servindo como suporte para aprimoramento da acessibilidade para pessoas com alguma limitação física. A interação do ser humano com o manipulador desenvolvida neste trabalho também reforça a utilização da robótica colaborativa para as diversas atividades supracitadas. Sua associação com o Kinect, ROS e também com o V-REP produziu soluções simples e de fácil implementação para a movimentação do manipulador, ao serem mostradas as técnicas de cinemática inversa, direta e lógica *fuzzy*. As ferramentas do V-REP de criação e de controle auxiliaram para uma construção do algoritmo de diversas maneiras. Sua comunicação com o ROS e a ferramente do *remoteAPI* para o recebimento dos dados do dispositivo foi imprescindível para o sucesso deste trabalho. A utilização do ROS permitiu padronizar os dados de entrada através dos arquivos *.bags*, facilitou a manipulação dos dados fornecidos pelo sensor Kinect e pelo OpenNI. E também permite a comunicação do *software* com o *hardware*, em caso de implementação.

Observou-se que o rastreamento do Kinect apresentou resultados satisfatórios, apresentando pouco ruído no sinal e um pequeno erro de medida, baseado na distância do usuário até o sensor. Todavia este apresentou problemas de instabilidade rotacional na região dos ombros, para uma determinada configuração de posição dos membros superiores. Este problema foi reparado através do filtro de Kalman robusto, apresentando como mais uma solução, e pela técnica de controle por lógica *fuzzy*, face à solução encontrada por Cavalcante (2012). Ao tratar essa instabilidade como um *outlier*, este filtro foi o suficiente para lidar com o problema, suavizando o sinal e diminuindo o ruído do mesmo. O mesmo resultado foi visto no controle por lógica *fuzzy*. Todavia este método apresentou um problema de interdependência entre as juntas. O Filtro média móvel também se mostrou eficaz para a diminuição de ruído, este foi recomendado para uso na cinemática inversa devido a sua simplicidade, mesmo o filtro de Kalman robusto apresentando uma variância menor. O método Levenberg-Marquardt apresentou resultados surpreendentes na cinemática inversa, principalmente quando a esfera de referência se mostrava fora do volume de trabalho do modelo utilizado, pois o método fazia a extremidade livre ir para o ponto mais próximo da esfera, dentro dos limites do volume de trabalho do manipulador.

Na questão da dinâmica, o esforço computacional exercido pelas equações matriciais fizeram desta técnica um grande obstáculo. Um vez que o modelo utilizando o actionlib apresentou atrasos nos envios das mensagens, não obtendo assim o resultado esperado.

Este cenário motivou a simplificar a estrutura dos códigos, utilizando *remoteAPI* com o ROS *python script* para sincronizar os cálculos com a simulação. A abordagem se comportou de maneira coerente ao projetado, porém este trouxe uma demanda de processamento muito maior.

Em suma, as técnicas que utilizavam as posições da mão, ou seja, faziam do uso direto ou indireto da cinemática inversa foram os que produziram resultados melhores. Podendo este ser responsável pelos problemas encontrados ao utilizar os dados rotacionais do Kinect. Dessa forma, os melhores resultados foram obtidos através da cinemática inversa e da dinâmica com o *script* em *python*.

Concluindo, pode-se afirmar que o objetivo proposto foi alcançado. Todavia, por mais que tenha sido encontrada uma solução neste trabalho, muito ainda pode ser explorado sobre este tema. Assim, sugere-se a implementação física deste trabalho, verificando os possíveis problemas que podem existir, que não foram percebidos em um cenário de simulação. Além disso, é possível realizar verificação e comparação de outras soluções para o problema de outlier encontrado, com o objetivo de encontrar maior eficiência na movimentação. Bem como propor uma modificação para o método da lógica *fuzzy* para remover a interdependência das juntas. E também utilizar métodos que além de capturar os movimentos pensem também na segurança do usuário. E por fim, poder utilizar outras ferramentas, como a atualização do sensor Kinect, presente no vídeo-game X-Box One e também uma atualização de *framework*, verificando se este apresenta melhores resultados do que os que foram encontrados neste trabalho.

6 Apêndice A - Pseudocódigo: cinemática direta - média móvel

```
algoritmo "media movel: cinematica direta"

funcao ombro_direito : real

i<-i_ombro

ombro_x[i]<-dados_ombro.x
ombro_y[i]<-dados_ombro.y
ombro_z[i]=dados_ombro.z

se (i == 10) entao
    i_ombro<-1

fim-se

ombro<- vetor [(ombro_x[1]+ombro_x[2]+ombro_x[3]+ombro_x[4]+ombro_x[5]+om

(ombro_y[1]+ombro_y[2]+ombro_y[3]+ombro_y[4]+ombro_y[5]+ombro_y[6]+ombro_

(ombro_z[1]+ombro_z[2]+ombro_z[3]+ombro_z[4]+ombro_z[5]+ombro_z[6]+ombro_

fimfuncao

funcao cotovelo_direito : real

i<-i_cotovelo

cotovelo_x[i]<-dados_cotovelo.x
cotovelo_y[i]<-dados_cotovelo.y
cotovelo_z[i]=dados_cotovelo.z

se (i == 10) entao
    i_cotovelo<-1
```

```
fim-se

cotovelo<- vetor [(cotovelo_x[1]+cotovelo_x[2]+cotovelo_x[3]+cotovelo_x[4

(cotovelo_y[1]+cotovelo_y[2]+cotovelo_y[3]+cotovelo_y[4]+cotovelo_y[5]+co

(cotovelo_z[1]+cotovelo_z[2]+cotovelo_z[3]+cotovelo_z[4]+cotovelo_z[5]+co

fimfuncao

se (sim_call_type==sim_childscriptcall_initialization) entao

var i_ombro, i_cotovelo : inteiro
var ombro_x, ombro_y, ombro_z : vetor [1-10] real
var cotovelo_x, cotovelo_y, cotovelo_z : vetor [1-10] real
var junta_1, junta_2, junta_3 : real

i_ombro <-1
i_cotovelo <-1
ombro_x <- vetor[0,0,0,0,0,0,0,0,0,0]  {numeros alterados de
ombro_y <- vetor[0,0,0,0,0,0,0,0,0,0] de acordo com o numero da
ombro_z <- vetor[0,0,0,0,0,0,0,0,0,0]  janela da media movel}
cotovelo_x <- vetor[0,0,0,0,0,0,0,0,0,0]
cotovelo_y <- vetor[0,0,0,0,0,0,0,0,0,0]
cotovelo_z <- vetor[0,0,0,0,0,0,0,0,0,0]

    {Atribuindo variaveis as juntas de revolucao}
jointHandles : vetor real
jointHandles[1]<-simGetObjectHandle('Revolute_joint1')
jointHandles[2]<-simGetObjectHandle('Revolute_joint2')
jointHandles[3]<-simGetObjectHandle('Revolute_joint3')

    {Relacionando o V-REP com o ROS}

sub1<-simExtRosInterface_subscribe('/dados_ombro_direito', 'geometry_msgs

sub2<-simExtRosInterface_subscribe('/dados_cotovelo_direito', 'geometry_m
```

```
fim-se

se (sim_call_type==sim_childscriptcall_sensing) entao

junta1 <- pi/2-ombro[2]*1.33 {necessario para ajustar os valores de dados
junta2 <- ombro[3]
junta3 <-  cotovelo[2]*(-1)

result1<-simSetJointPosition(jointHandles[1],junta1)
result2=simSetJointPosition(jointHandles[2],junta2)
result3=simSetJointPosition(jointHandles[3],junta3)

fim-se

se (sim_call_type==sim_childscriptcall_actuation) entao
        {estrutura pre-montada do V-REP, porem nao utilizada}
fim- se

se (sim_call_type==sim_childscriptcall_cleanup) entao

    simExtRosInterface_shutdownSubscriber(sub1)
    simExtRosInterface_shutdownSubscriber(sub2)
    simExtRosInterface_shutdownSubscriber(sub3)
fim-se
fim algoritmo
```

7 Apêndice B - Pseudocódigo: Cinemática direta - filtro de Kalman

```
algoritmo "filtro de Kalman: cinematica direta"

funcao ombro_direito : real

Ax<-0
Ay<-0
Az<-0


ombro_x<-dados_ombro_direito.x
ombro_y<-dados_ombro_direito.y
ombro_z<-dados_ombro_direito.z


Ax <- ombro_final_x-ombro_anterior_x
Ay <- ombro_final_y-ombro_anterior_y
Az <- ombro_final_z-ombro_anterior_z

ombro_anterior_x <- ombro_final_x
ombro_anterior_y <- ombro_final_y
ombro_anterior_z <- ombro_final_z

x <- Ax*ombro_final_x + (2* random-1)/100
y <- Ay*ombro_final_y + (2* random-1)/100
z <- Az*ombro_final_z + (2* random-1)/100

Px <- Ax*P_final_ombro_x*Ax+0.8
Py <- Ay*P_final_ombro_y*Ay+0.8
Pz <- Az*P_final_ombro_z*Az+0.8

K_final_x <- Px/(Px+Rx)
K_final_y <- Py/(Py+Ry)
K_final_z <- Pz/(Pz+Rz)
```

```
ombro_final_x <-  x + K_final_x * (ombro_x - x);
ombro_final_y <-  y + K_final_y * (ombro_y - y);
ombro_final_z <-  z + K_final_z * (ombro_z - z);

P__final_ombro_x <- (1-K_final_x)*Px
P__final_ombro_y <- (1-K_final_y)*Py
P__final_ombro_z <- (1-K_final_z)*Pz

ombro<- vetor [ombro_final_x,ombro_final_y,ombro_final_z] real

fimfuncao


funcao cotovelo_direito : real

Ax<-0
Ay<-0
Az<-0

cotovelo_x<-dados_cotovelo_direito.x
cotovelo_y<-dados_cotovelo_direito.y
cotovelo_z<-dados_cotovelo_direito.z

Ax <- cotovelo_final_x-cotovelo_anterior_x
Ay <- cotovelo_final_y-cotovelo_anterior_y
Az <- cotovelo_final_z-cotovelo_anterior_z

cotovelo_anterior_x <- cotovelo_final_x
cotovelo_anterior_y <- cotovelo_final_y
cotovelo_anterior_z <- cotovelo_final_z

x <- Ax*cotovelo_final_x+ (2* random-1)/100
y <- Ax*cotovelo_final_y+ (2* random-1)/100
z <- Ax*cotovelo_final_z+ (2* random-1)/100

Px <- Ax*P_final_cotovelo_x*Ax+0.8 (0.08 ou 0.008)
Py <- Ay*P_final_cotovelo_y*Ay+0.8 (0.08 ou 0.008)
Pz <- Az*P_final_cotovelo_z*Az+0.8 (0.08 ou 0.008)
```

```
K_final_x <- Px/(Px+Rx)
K_final_y <- Py/(Py+Ry)
K_final_z <- Pz/(Pz+Rz)

cotovelo_final_x <-  x + K_final_x * (cotovelo_x - x);
cotovelo_final_y <-  y + K_final_y * (cotovelo_y - y);
cotovelo_final_z <-  z + K_final_z * (cotovelo_z - z);

P__final_cotovelo_x <- (1-K_final_x)*Px
P__final_cotovelo_y <- (1-K_final_y)*Py
P__final_cotovelo_z <- (1-K_final_z)*Pz

cotovelo<- vetor [cotovelo__final_x,cotovelo__final_y,cotovelo__final_z]

fimfuncao

se (sim_call_type==sim_childscriptcall_initialization) entao

var ombro_final_x, ombro_final_y, ombro_final_z : real
var cotovelo_final_x, cotovelo_final_y, cotovelo_final_z : real
var ombro_anterior_x, ombro_anterior_y, ombro_anterior_z : real
var cotovelo_anterior_x, cotovelo_anterior_y, cotovelo_anterior_z : real
var P_final_ombro_x, P_final_ombro_y, P_final_ombro_z : real
var P_final_cotovelo_x, P_final_cotovelo_y, P_final_cotovelo_z : real
var Rx, Ry, Rz : real

var cotovelo : vetor [1-3] real
var ombro : vetor [1-3] real

ombro_final_x<-0.0000001
ombro_final_y<-0.0000001
ombro_final_z<-0.0000001



P_final_ombro_x<-1
P_final_ombro_y<-1
P_final_ombro_z<-1

cotovelo_final_x<-0.0000001
```

```
cotovelo_final_y <-0.0000001
cotovelo_final_z <-0.0000001

P_final_cotovelo_x<-1
P_final_cotovelo_y<-1
P_final_cotovelo_z<-1

ombro_anterior_x<-0
ombro_anterior_y<-0
ombro_anterior_z<-0

cotovelo_anterior_x<-0
cotovelo_anterior_y<-0
cotovelo_anterior_z<-0

Rx<-0.06
Ry<-0.02
Rz<-0.02


 {Atribuindo variaveis as juntas de revolucao}
var jointHandles : vetor[1-3] real
jointHandles[1]<-simGetObjectHandle('Revolute_joint1')
jointHandles[2]<-simGetObjectHandle('Revolute_joint2')
jointHandles[3]<-simGetObjectHandle('Revolute_joint3')

    {Relacionando o V-REP com o ROS}

    sub1<-simExtRosInterface_subscribe('/dados_ombro_direito', 'geometry_
    sub2<-simExtRosInterface_subscribe('/dados_cotovelo_direito', 'geomet

fim-se
...
fim algoritmo
```

8 Apêndice C - Pseudocódigo: Lógica *fuzzy*

8.1 pseudocódigo do arquivo em LUA utilizado dentro do simulador V-REP

```
algoritmo "vrep_fuzzy"

funcao dados_fuzzy_server : void

    junta_1 <-saida_posicao.x
    junta_2 <-saida_posicao.y
    junta_3 <-saida_posicao.z

fimfuncao

funcao ombro_direito : void

    ombro_y <- msg.y
    ombro_z <- -msg.z

fimfuncao

funcao cotovelo_direito : void


    cotovelo_y <- -msg.y

fimfuncao


 se (sim_call_type==sim_childscriptcall_initialization) entao

var junta_1, junta_2, junta_3 : real
var ombro_y, ombro_z, cotovelo_y : real
var openshoulderpos, azishoulderpos, elbowpos : real

junta_1 <- 0
```

```
junta_2 <- 0
junta_3 <- 0

ombro_y <- 0
ombro_z <- 0
cotovelo_y <- 0

openshoulderpos <- 0
azishoulderpos <- 0
elbowpos <- 0

var jointHandles : vetor [1-3] real
jointHandles[1] <- simGetObjectHandle('Revolute_joint1')
jointHandles[2] <- simGetObjectHandle('Revolute_joint2')
jointHandles[3] <- simGetObjectHandle('Revolute_joint3')


{Relacionando o V-REP com o ROS}

    sub1<-simExtRosInterface_subscribe('/saida_posicao', 'geometry_msgs/V
    sub2<-simExtRosInterface_subscribe('/dados_ombro', 'geometry_msgs/Vec
    sub3<-simExtRosInterface_subscribe('/dados_cotovelo', 'geometry_msgs/

    pub1<-simExtRosInterface_advertise('/dados_junta','geometry_msgs/Vect
    simExtRosInterface_publisherTreatUInt8ArrayAsString(pub1)

    pub2<-simExtRosInterface_advertise('/posicao_junta','geometry_msgs/Ve
    simExtRosInterface_publisherTreatUInt8ArrayAsString(pub2)


fim-se

se (sim_call_type==sim_childscriptcall_sensing) entao

elbowpos <- junta_3

openshoulderpos <- junta_2

azishoulderpos <- junta_1
```

```
result1 <- simSetJointPosition(jointHandles[1],azishoulderpos)
result2 <- simSetJointPosition(jointHandles[2],openshoulderpos)
result3 <- simSetJointPosition(jointHandles[3],elbowpos)

    d <- vetor {}
    d['x'] <- ombro_y
    d['y'] <- ombro_z
    d['z'] <- cotovelo_y
    simExtRosInterface_publish(pub1,d)

    d2 <- vetor {}
    d2['x'] <- azishoulderpos
    d2['y'] <- openshoulderpos
    d2['z'] <- elbowpos
    simExtRosInterface_publish(pub2,d2)

fim-se


se (sim_call_type==sim_childscriptcall_actuation) entao

fim-se


se (sim_call_type==sim_childscriptcall_cleanup) entao

        simExtRosInterface_shutdownSubscriber(sub1)
    simExtRosInterface_shutdownSubscriber(sub2)
    simExtRosInterface_shutdownSubscriber(sub3)
    simExtRosInterface_shutdownPublisher(pub1)
    simExtRosInterface_shutdownPublisher(pub2)

fim-se
fim algoritmo
```

8.2 Pseudocódigo do arquivo *FuzzyClient* utilizado no ROS actionlib

```
algoritmo "fuzzyclient"
classe FuzzyClient

  publico:

    FuzzyClient : client
      ac(nome_topico, true)

            {armazena nome}
      nome_acao<-nome_topico

              {Conectar com o Fuzzy Server}
      ROS_INFO("Esperando Servidor", nome_acao)

              {Espera a conexao ser realizada}
      ac.waitForServer()

      ROS_INFO("%s Encontrou um servidor...", nome_acao)

             {Recebendo os dados e enviando para a funcao GoalCallback}
      goalsub<- ROS_subscribe


                {Funcao para dizer se o servidor alcancou ou nao o objeti
    funcao doneCb : void

        ROS_INFO("termininou no estado...", state.toString)

        ROS_INFO("Resultado: %i", result->ok)

fimfuncao

    {Chama a funcao para quando receber um novo objetivo}
funcao activeCb : void

        ROS_INFO("Objetivo ativado")
fimfuncao

  {Verifica frequentemente a situacao do robo, para saber se atingiu o ob
```

```
funcao feedbackCb : const tutorial2_controller::FuzzyControlFeedbackConst

        ROS_INFO("Feedback rumo ao objetivo: posicao x y z", feedback->po
fimfuncao

funcao GoalCallback : const geometry_msgs::Vector3& msg {a mensagem varia

        goal.position.x <- dados_junta.x
        goal.position.y <- dados_junta.y
        goal.position.z <- dados_junta.z
                {Enviar o objetivo para o Server}
        ac.sendGoal
fimfuncao

privado:
        SimpleActionClient : ac
        string : nome_acao
        FuzzyControlGoal : goal
        Subscriber goalsub
        NodeHandle n
fimclasse

inicio
        {Iniciando o node FuzzyClient}
        ros::init

        FuzzyClient client(ros::this_node::getName(),"fuzzycontrol1","/da


        ros::spin()

fim algoritmo
```

8.3 Pseudocódigo do arquivo *FuzzyServer* utilizado no ROS actionlib

```
algoritmo "fuzzy_server"
```

```
Defina PI <- 3.1415

classe FuzzyServer
publico:

    FuzzyServer : server

    nome_acao = name_topico

    as.registerPreemptCallback

    as.start

    posicao_junta <- n2.subscribe

        {publicador da saida da logica fuzzy}
    saida_posicao <- n2.advertise

       {limitando a saida maxima do controlador}
    max <- PI
    min <- -PI

       {Iniciando o sistema}
    Initialize(min, max)

fim FuzzyServer


funcao  preemptCB : void

    ROS_INFO("Foi Interrompido", nome_acao)
    result.ok <- 0
    as.setPreempted(result, "Eu fui interrompido")
fimfuncao


funcao ExecutarCalculos : void

    Se (!as.isActive() OU as.isPreemptRequested()) entao
```

```
return
fim-se

    {Criando taxa de frequencia}
ros::Rate rate(100)

bool successo <- true

 {Look do Controlador}
Enquanto (1) faca

  Vector3 : msg_pos

   {Controlador Fuzzy}
  msg_pos.x <- FuzzyController
  msg_pos.y <- FuzzyController2
  msg_pos.z <- FuzzyController3

    {publicando o resultado}
  saida_posicao.publish(msg_pos)

    {cada ponto do feedback corresponde a uma junta especifica}
  feedback.position.x <- posicao_junta.x
  feedback.position.y <- posicao_junta.y
  feedback.position.z <- posicao_junta.z

   {publicar o feedback para o client}
  as.publishFeedback(feedback)

    {verificar se a comunicacao com o ROS morreu}
  se (!ros::ok()) entao
    success <- false;
    ROS_INFO(" Desligando ", nome_acao)
    Interromper(break)
  fim-se

   {Se o programa foi interormpido, parar o processo}
  se (!as.isActive() || as.isPreemptRequested())  entao
   return
```

```
    rate.sleep
    fim-se

   {enviar para o cliente o estado quando o programa nao foi interrompi
  se (sucesso) entao
    result.ok <- 1
    as.setSucceeded(result)
  fim-se
  senao
    result.ok <- 0
    as.setAborted(result, "Falhei");
  fim-senao
fim enquanto

fim funcao

funcao Initialize : void

  setOutputLimits(min, max)
  lastError1 <- 0
  lastError2 <- 0
  lastError3 <- 0
fimfuncao

funcao setOutputLimits : void

  if (min > max) return

  minLimit = min
  maxLimit = max
fim funcao

      {As tres funcoes sao iguais, houve a separacao para poder separar
FuzzyController,  FuzzyController2,  FuzzyController3 : float

  var error, dErr : float

  error <- setpoint - PV
```

```
dErr <- error - lastError1

//Create fuzzy
fl::Engine* engine = new fl::Engine
engine->setName("")

 {criar a entrada fuzzy : erro}
 {Os termos em ingles sao funcoes proprias do fuzzylite}
fl::InputVariable* variavel1 <- new fl::InputVariable
variavel1 -> setEnabled(true)
variavel1 -> setName("e")
variavel1 -> setRange(-7.000, 7.000)
variavel1 -> addTerm(new fl::Trapezoid("N", -7.000, -7.000, -1.750, 0
variavel1 -> addTerm(new fl::Triangle("Z", -1.750, 0.000, 1.750))
variavel1 -> addTerm(new fl::Trapezoid("P", 0.000, 1.750, 7.000, 7.00
engine->addInputVariable(variavel1 )

{criar a entrada fuzzy : dErro}
fl::InputVariable* variavel2 <- new fl::InputVariable
variavel2 -> setEnabled(true)
variavel2 -> setName("e")
variavel2 -> setRange(-7.000, 7.000)
variavel2 -> addTerm(new fl::Trapezoid("N", -7.000, -7.000, -1.750, 0
variavel2 -> addTerm(new fl::Triangle("Z", -1.750, 0.000, 1.750))
variavel2 -> addTerm(new fl::Trapezoid("P", 0.000, 1.750, 7.000, 7.00
engine->addInputVariable(variavel2)

{criar a saida fuzzy : saida}
fl::OutputVariable* saida <- new fl::OutputVariable
saida -> setEnabled(true)
saida -> setName("out")
saida -> setRange(-3.500, 3.500)
saida -> fuzzyOutput()->setAccumulation(new fl::Maximum)
saida -> setDefuzzifier(new fl::Centroid(200))
saida -> setDefaultValue(0.000)
saida -> setLockValidOutput(true)
saida -> setLockOutputRange(true)
saida -> addTerm(new fl::Triangle("NG", -3.500, -3.500, -1.750))
saida -> addTerm(new fl::Triangle("NP", -3.500, -1.750, 0.000))
```

```
saida -> addTerm(new fl::Triangle("Z", -1.750, 0.000, 1.750))
saida -> addTerm(new fl::Triangle("PP", 0.000, 1.750, 3.500))
saida -> addTerm(new fl::Triangle("PG", 1.750, 3.500, 3.500))
engine-> addOutputVariable(saida)

 {Regras}
fl::RuleBlock* conjunto de regras = new fl::RuleBlock
regras->setEnabled(true)
regras->setName("")
regras -> setConjunction(new fl::Minimum)
regras -> setDisjunction(NULL)
regras -> setActivation(new fl::Minimum)
regras -> addRule(fl::Rule::parse("if e is N and de is N then out is
regras -> addRule(fl::Rule::parse("if e is N and de is Z then out is
regras -> addRule(fl::Rule::parse("if e is N and de is P then out is
regras -> addRule(fl::Rule::parse("if e is Z and de is N then out is
regras -> addRule(fl::Rule::parse("if e is Z and de is Z then out is
regras -> addRule(fl::Rule::parse("if e is Z and de is P then out is
regras -> addRule(fl::Rule::parse("if e is P and de is N then out is
regras -> addRule(fl::Rule::parse("if e is P and de is Z then out is
regras -> addRule(fl::Rule::parse("if e is P and de is P then out is
engine-> addRuleBlock(regras)

 {Entrada das variaveis}
variavel1 -> setInputValue(error)
variavel2 -> setInputValue(dErr)

 {Iniciar Fuzzy}
engine->process()

 {Defuzzificacao}
fl::scalar saida <- outputVariable->defuzzify();


valor_saida <- (valor_saida + saida) / 90

 {limitar a saida}
valor_saida <- min
valor_saida <- max
```

```
    //Required values for next round */
    lastError1 <- error

    returna valor_saida
  fim funcao

 funcao SensorCallback : void

    variavel_junta.x <- dados_junta.x
    variavel_junta.y <- dados_junta.y
    variavel_junta.z <- dados_junta.z
 fimfuncao

protegido:
  ros::NodeHandle n
  ros::NodeHandle n2

        {Variavel de subscricao}
  ros::Subscriber posicao_junta

        {Variavel publicador}
  ros::Publisher saida_posicao



        {Variaveis Actionlib}
  SimpleActionServer : as
  FuzzyControlFeedback : feedback
  tutorial2_controller::FuzzyControlResult result
  string nome_acao

   {Variaveis de Controle}
  Vector3 variavel_junta
var lastError1, lastError2, lastError3, minLimit : real
var maxLimit output : real

fim classe

Inicio
```

```
  ros::init("fuzzy_server")

  se (argc != 1) entao

    ROS_INFO("Utilizando fuzzy_server")
    retorna 1
  fim-se

  FuzzyServer server(ros::this_node::getName(), "fuzzycontrol1", "/feedba

  ros::spin()

fim algoritmo
```

9 Apêndice D - Pseudocódigo: Dinâmica de manipuladores

9.1 Fragmento do pseudocódigo utilizado no arquivo *DymServer* para o ROS actionlib

```
algoritmo "dym_server"
...
funcao ExecutarCalculos : void

 Enquanto (1) faca

      {Recebendo a posicao vinda do GoalCallback do ROS client}
    targetxyz(0) <- goal->PositionGoal.x
    targetxyz(1) <- goal->PositionGoal.y
    targetxyz(2) <- goal->PositionGoal.z

    goalpub.publish(goal->PositionGoal)



    joint_target_velocities(0) <- 10
    joint_target_velocities(1) <- 10
    joint_target_velocities(2) <- 10

      {Lock criado para garantir que sejam recebidos os dados}
    lock_.lock()

    q[0] <- m_info.joint_pos1
    q[1] <- m_info.joint_pos2
    q[2] <- m_info.joint_pos3

    dq(0) <- m_info.joint_vel1
    dq(1) <- m_info.joint_vel2
    dq(2) <- m_info.joint_vel3

    torque(0) <- m_info.jointfor1
```

```
torque(1) <- m_info.jointfor2
torque(2) <- m_info.jointfor3
lock_.unlock()

L <- [0, .42, .225]

        {Encontrar matematicamente a posicao da extremidade atraves d
xyzz[0] <- L[0]*cos(q[0])+L[1]*cos(q[0])*cos(q[1])+L[2]*cos(q[0])*cos

xyzz[1] <- L[0]*cos(q[0])+L[1]*cos(q[1])*sin(q[0])+L[2]*sin(q[0])*cos

xyzz[2] <- L[1]*sin(q[1])+L[2]*(cos(q[1])*sin(q[2])+cos(q[2])*sin(q[1

    {passar essas informacoes para o objeto matriz do Armadillo}
para i = 0 ate 2 faca
  xyz(i) <-xyzz[i]
fim para

    {Calculo da Matriz Jacobiana do sistema}

jee[0][2] <- -L[2] * sin(q[1]+q[2])*cos(q[0]);
jee[1][2] <- -L[2] * sin(q[1]+q[2])*sin(q[0]);
jee[2][2] <- L[2]*cos(q[1]+q[2]);
jee[0][1] <- -cos(q[0])*(L[2]*sin(q[1]+q[2])+L[1]*sin(q[1]));
jee[1][1] <- -sin(q[0])*(L[2]*sin(q[1]+q[2])+L[1]*sin(q[1]));
jee[2][1] <- L[2] * cos(q[1]+q[2]) + L[1]*cos(q[1]);
jee[0][0] <- -sin(q[0])*(L[0]+jee[2][1]);
jee[1][0] <- cos(q[0])*(L[0]+jee[2][1]);

para i=0 ate 2 faca
  para j=0 ate 2 faca
    JEE(i,j) <- jee[i][j]
  fim para
fim para

    {Calculo da matriz Jacobiana levando em contra o centro de massa}

jcom1[0][0] <- L[0] * -sin(q[0])/2
jcom1[1][0] <- L[0] * cos(q[0])/2
```

```
jcom1[5][0] <- 1.0


jcom2[0][1] <- L[1] * (cos(q[1]) * sin(q[0]) - cos(q[0]) * sin(q[1]))

jcom2[1][1] <- L[1] * (-cos(q[1]) * cos(q[0]) - sin(q[1]) * sin(q[0])

jcom2[4][1] <- 1.0

jcom2[0][0] <- L[1]*(cos(q[0])*sin(q[1])-cos(q[1])*sin(q[0]))/2-L[0]*s

jcom2[1][0] <- L[1] * (sin(q[0])*sin(q[1]) + cos(q[0])*cos(q[1]))/2 +

jcom2[4][0] <- 1.0


jcom3[0][2] <- L[2]*(cos(q[2])*sin(q[0]) - cos(q[0])*cos(q[1])*sin(q[

jcom3[1][2] <- L[2]*(-cos(q[2])*cos(q[0]) - cos(q[1])*sin(q[0])*sin(q

jcom3[2][2] <- -L[2]*sin(q[1])*sin(q[2])/2

jcom3[4][2] <- 1.0

jcom3[0][1] <- L[2] * -cos(q[0])*cos(q[2])*sin(q[1])/2 - L[1]*cos(q[0

jcom3[1][1] <- L[2] * -cos(q[2])*sin(q[0])*sin(q[1])/2 - L[1]*sin(q[0

jcom3[2][1] <- (L[2]*cos(q[1])*cos(q[2])/2)+L[1]*cos(q[1])

jcom3[4][1] <- 1.0
jcom3[0][0] <- L[2]*(cos(q[0])*sin(q[2]) - cos(q[1])*cos(q[2])*sin(q[

jcom3[1][0] <- L[2]*(sin(q[0])*sin(q[2]) + cos(q[0])*cos(q[1])*cos(q[

jcom3[4][0] <- 1.0

para i=0 ate 5 faca
   para j=0 ate 2 faca
```

```
      JCOM1(i,j) <- jcom1[i][j]
      JCOM2(i,j) <- jcom2[i][j]
      JCOM3(i,j) <- jcom3[i][j]
    fim para
fim para

      {Criando a matriz de inercia de cada junta}
M11[0][0] <- m1
M11[1][1] <- m1
M11[2][2] <- m1
M11[3][3] <- 0.001667
M11[4][4] <- 0.001667
M11[5][5] <- 0.001667

M22[0][0] <- m2
M22[1][1] <- m2
M22[2][2] <- m2
M22[3][3] <- 0.008121
M22[4][4] <- 0.008121
M22[4][4] <- 0.008121
M22[5][5] <- 0.001242

M33[0][0] <- m3
M33[1][1] <- m3
M33[2][2] <- m3
M33[3][3] <- 0.0005561
M33[4][4] <- 0.0005561
M33[5][5] <- 0.0001747

para i=0 ate 5 faca

  para j=0 ate 5 faca

    se (i==j) entao

      M1(i,j) <- M11[i][j]
      M2(i,j) <- M22[i][j]
      M3(i,j) <- M33[i][j]
```

```
    fim se
  fim para
fim para

    {JCOM.t() e a matriz transposta de JCOM}

Mq <- JCOM1.t() * M1 *JCOM1 + JCOM2.t() * M2 *JCOM2 + JCOM3.t() * M3

para j=0 ate 5 faca

  g(j) <- gravity[j];
fim para

    {Encontrando a matriz que relaciona a inercia do sistema}
Mq_g <- JCOM1.t() * M1 *g + JCOM2.t() * M2 *g + JCOM3.t() * M3 *g

Mx_inv <- JEE * inv(Mq) * JEE.t()

svd(Mu,ms,Mv,Mx_inv)

para i=0 ate ms.n_elem faca {numero de elementos de ms}

  se(ms(i)<1e-5) entao
    ms(i) <- 0
  fim-se
  senao
    ms(i)= 1/ms.at(i)
  fim-senao

Mx <- Mv.t()*diagmat(ms)*Mu.t()

{Definindo os parametros do controlador}

var kp, kv :real
kp <- 100
kv <- 2

u_xyz <- Mx*(kp*(targetxyz-xyz))
```

```
{calculando o torque para cada junta}

u <- JEE.t()*u_xyz - Mq*(kv*dq) - Mq_g;

u <- u*-1;

{Verificando o sentido da velocidade de acordo com o torque do manipu

para i=0 ate 2 faca
   se (torque.at(i) * u.at(i) <0) entao

    joint_target_velocities(i)<- joint_target_velocities.at(i)*-1
  fim-se
fim para

{preparando as variaveis de posicao, velocidade e forca para enviar p

robot_info.jointpos1 <- 0
robot_info.jointpos2 <- 0
robot_info.jointpos3 <- 0

robot_info.jointfor1 <- abs(u.at(0))
robot_info.jointfor2 <- abs(u.at(1))
robot_info.jointfor3 <- abs(u.at(2))

robot_info.jointvel1 <- joint_target_velocities.at(0)
robot_info.jointvel2 <- joint_target_velocities.at(1)
robot_info.jointvel3 <- joint_target_velocities.at(2)

pubinfo.publish(robot_info)

    {enviando o feedback para o arquivo client}
feedback.PositionFeedback.x <- xyz[0]
feedback.PositionFeedback.y <- xyz[1]
feedback.PositionFeedback.z <- xyz[2]


as.publishFeedback(feedback)
```

```
    //Check for ROS kill
    if (!ros::ok())
    {
      success = false;
      ROS_INFO("%s Shutting Down", action_name.c_str());
      break;

    }

    {enviar para o cliente o estado quando o programa nao foi interrompid
    se (sucesso) entao
      result.ok <- 1
      as.setSucceeded(result)
    fim-se
    senao
      result.ok <- 0
      as.setAborted(result, "Falhei");
    fim-senao
  fim enquanto

  fim funcao
  }
...
```

9.2 Pseudocódigo do *python script* utilizado na dinâmica de manipuladores

```
algoritmo "pythondym"

var count : inteiro
var  dt, track_hand, track_target : real
var nome_juntas, joint_handles : vetor char

cont<-0
dt <- 0.001
mao_r <- []
track_target <- []
nome_juntas <- ['Revolute_joint','shoulder', 'elbow']
```

```
joint_handles <- []
target_handle <-1
_ <- 0
juntas_vel <- vetor_de_zeros(dimensao de nome_juntas)

funcao callback : void


se (clientID != -1 E cont== 0) entao
    Escreva ("conectado com o simulador")

    vrep.simxSynchronous(clientID,True)


    juntas_vel <- vetor_de_1 (dimensao de nome_juntas) * 10000.0


    para i=0 ate dimensao de nome_juntas faca
        joint_handles(i) <- [vrep.simxGetObjectHandle(clientID,
       nome_juntas(i), vrep.simx_opmode_blocking)

    fim para

     _, target_handle <- vrep.simxGetObjectHandle(clientID,
          'target', vrep.simx_opmode_blocking)


    vrep.simxSetFloatingParameter(clientID,
        vrep.sim_floatparam_simulation_time_step,
        dt, vrep.simx_opmode_oneshot)


                {Inicia a simulacao}
    vrep.simxStartSimulation(clientID,
                vrep.simx_opmode_blocking)
            print("Starting Simulation")


    e (clientID!=-1 E count >= 0 E count<6)
```

```
target_xyz <- [dados_mao.x, dados_mao.y, dados_mao.z]

    {Enviando a posicao para o objeto esferico}

vrep.simxSetObjectPosition(clientID, target_handle, -1, target_xy
se _ !=0 entao
    Escreva ("Deu ruim parceiro")
fim-se



q[0] <- m_info.joint_pos1
q[1] <- m_info.joint_pos2
q[2] <- m_info.joint_pos3

dq[0] <- m_info.joint_vel1
dq[1] <- m_info.joint_vel2
dq[2] <- m_info.joint_vel3

torque(0) <- m_info.jointfor1
torque(1) <- m_info.jointfor2
torque(2) <- m_info.jointfor3
lock_.unlock()

L <- [0, .42, .225]

    {Encontrar matematicamente a posicao da extremidade atraves d
xyzz[0] <- L[0]*cos(q[0])+L[1]*cos(q[0])*cos(q[1])+L[2]*cos(q[0])

xyzz[1] <- L[0]*cos(q[0])+L[1]*cos(q[1])*sin(q[0])+L[2]*sin(q[0])

xyzz[2] <- L[1]*sin(q[1])+L[2]*(cos(q[1])*sin(q[2])+cos(q[2])*sin

{Calculo da Matriz Jacobiana do sistema}

jee[0][2] <- -L[2] * sin(q[1]+q[2])*cos(q[0]);
jee[1][2] <- -L[2] * sin(q[1]+q[2])*sin(q[0]);
jee[2][2] <- L[2]*cos(q[1]+q[2]);
```

```
jee[0][1] <- -cos(q[0])*(L[2]*sin(q[1]+q[2])+L[1]*sin(q[1]));
jee[1][1] <- -sin(q[0])*(L[2]*sin(q[1]+q[2])+L[1]*sin(q[1]));
jee[2][1] <- L[2] * cos(q[1]+q[2]) + L[1]*cos(q[1]);
jee[0][0] <- -sin(q[0])*(L[0]+jee[2][1]);
jee[1][0] <- cos(q[0])*(L[0]+jee[2][1]);

{Calculo da matriz Jacobiana levando em contra o centro de massa}

jcom1[0][0] <- L[0] * -sin(q[0])/2
jcom1[1][0] <- L[0] * cos(q[0])/2
jcom1[5][0] <- 1.0


jcom2[0][1] <- L[1] * (cos(q[1]) * sin(q[0]) - cos(q[0]) * sin(q[

jcom2[1][1] <- L[1] * (-cos(q[1]) * cos(q[0]) - sin(q[1]) * sin(q

jcom2[4][1] <- 1.0

jcom2[0][0]<- L[1]*(cos(q[0])*sin(q[1])-cos(q[1])*sin(q[0]))/2-L[

jcom2[1][0] <- L[1] * (sin(q[0])*sin(q[1]) + cos(q[0])*cos(q[1]))

jcom2[4][0] <- 1.0


jcom3[0][2] <- L[2]*(cos(q[2])*sin(q[0]) - cos(q[0])*cos(q[1])*si

jcom3[1][2] <- L[2]*(-cos(q[2])*cos(q[0]) - cos(q[1])*sin(q[0])*s

jcom3[2][2] <- -L[2]*sin(q[1])*sin(q[2])/2

jcom3[4][2] <- 1.0

jcom3[0][1] <- L[2] * -cos(q[0])*cos(q[2])*sin(q[1])/2 - L[1]*cos

jcom3[1][1] <- L[2] * -cos(q[2])*sin(q[0])*sin(q[1])/2 - L[1]*sin

jcom3[2][1] <- (L[2]*cos(q[1])*cos(q[2])/2)+L[1]*cos(q[1])
```

```
        jcom3[4][1] <- 1.0
        jcom3[0][0] <- L[2]*(cos(q[0])*sin(q[2]) - cos(q[1])*cos(q[2])*si

        jcom3[1][0] <- L[2]*(sin(q[0])*sin(q[2]) + cos(q[0])*cos(q[1])*co

        jcom3[4][0] <- 1.0

    {Criando a matriz de inercia de cada junta}
        M11[0][0] <- m1
        M11[1][1] <- m1
        M11[2][2] <- m1
        M11[3][3] <- 0.001667
        M11[4][4] <- 0.001667
        M11[5][5] <- 0.001667

        M22[0][0] <- m2
        M22[1][1] <- m2
         M22[2][2] <- m2
         M22[3][3] <- 0.008121
        M22[4][4] <- 0.008121
        M22[4][4] <- 0.008121
        M22[5][5] <- 0.001242

        M33[0][0] <- m3
        M33[1][1] <- m3
        M33[2][2] <- m3
        M33[3][3] <- 0.0005561
        M33[4][4] <- 0.0005561
        M33[5][5] <- 0.0001747

        {JCOM.t() e a matriz transposta de JCOM}

        Mq <- JCOM1.t() * M1 *JCOM1 + JCOM2.t() * M2 *JCOM2 + JCOM3.t() *


        g(j) <- [0,0,-9.81,0,0,0];

    {Encontrando a matriz que relaciona a inercia do sistema}
```

```
    Mq_g <- JCOM1.t() * M1 *g + JCOM2.t() * M2 *g + JCOM3.t() * M3 *g

    Mx_inv <- JEE * inv(Mq) * JEE.t()

    svd(Mu,ms,Mv,Mx_inv)

para i=0 ate ms.n_elem faca {numero de elementos de ms}

  se(ms(i)<1e-5) entao
    ms(i) <- 0
  fim-se
  senao
    ms(i)= 1/ms(i)
  fim-senao

Mx <- Mv.t()*diagmat(ms)*Mu.t()

{Definindo os parametros do controlador}

var kp, kv :real
kp <- 100
kv <- 2

u_xyz <- Mx*(kp*(targetxyz-xyz))

{calculando o torque para cada junta}

u <- JEE.t()*u_xyz - Mq*(kv*dq) - Mq_g;

u <- u*-1;

para i=0 ate dimensao nome_juntas faca

   var torque: vetor real
   torque[i] <- vrep.simxGetJointForce(clientID,
                              joint_handle,
                              vrep.simx_opmode_blocking)
   se (torque(i) * u(i) <0) entao
```

```
        joint_target_velocities(i)<- joint_target_velocities.at(i)*-1
      fim-se
    fim para

    vrep.simxSetJointForce(clientID,joint_handle, abs(u[ii]),
            vrep.simx_opmode_blocking)

    vrep.simxSynchronousTrigger(clientID)
    count += dt
    fim-se
    senao se (clientID != -1 and cont>=6) entao
                vrep.simxStopSimulation(clientID, vrep.simx_opmode_blocki
        vrep.simxGetPingTime(clientID)

        vrep.simxFinish(clientID)
    fim-senao-se
        senao
                raise Exception('Conexao falhou')
        fim-senao

        retorna cont

inicio

        var clientID : inteiro

        rospy.init_node
        vrep.simxFinish(-1)

        se cont==0 entao
                clientID <- vrep.simxStart('127.0.0.1', 19997, True, True
        fim-se
        sub = rospy.Subscriber("dados_mao", Vector3, callback)
        rospy.spin()
        retorna clientID

fim algoritmo
```

Referências

ABDALLAH, I. B.; BOUTERAA, Y.; REKIK, C. Kinect-based sliding mode control for lynxmotion robotic arm. 2016. 17, 28, 30, 31, 50

AFTHONI, R.; RIZAL, A.; SUSANTO, E. Proportional derivative control based robot arm system using microsoft kinect. *2013 International Conference on Robotics, Biometics, Intelligent Computational Systems (ROBIONETICS)*, 2013. 51

ALBÁN, D. S.; ADORNO, B. V. External hybrid force/pose controller for manipulator robots using dual quaternion algebra. *2017 Latin American Robotics Symposium (LARS)*, 2017. 51

ALI, M. M. et al. Arm grasping for mobile robot transportation using kinect sensor and kinematic analysis. In: IEEE. *2015 IEEE International Instrumentation and Measurement Technology Conference (I2MTC) Proceedings*. [S.l.], 2015. p. 516–521. 50

ANDÚJAR, C. *Kinect*. 2012. Disponível em: <https://www.goodreads.com/book/show/36330754-kinect>. Acesso em: 31 outubro 2018. 37, 38, 39

BARAKAT, A. N.; GOUDA, K. A.; BOZED, K. A. Kinematics analysis and simulation of a robotic arm using matlab. In: IEEE. *2016 4th International Conference on Control Engineering & Information Technology (CEIT)*. [S.l.], 2016. p. 1–5. 50

BEAUPRE, M. *Collaborative Robot Technology and Applications*. RIA, 2014–2019. Disponível em: <https://www.robotics.org/userAssets/riaUploads/file/4-KUKA_Beaupre.pdf>. Acesso em: 11 junho 2019. 16

CAVALCANTE, F. Z. M. S. *Reconhecimento de Movimentos Humanos para Imitação e Conrole de um Robô Humanoide*. 2012. 16, 50, 53, 66, 78

CHERUBINI, A. et al. Collaborative manufacturing with physical human–robot interaction. *Robotics and Computer-Integrated Manufacturing*, Elsevier, v. 40, p. 1–13, 2016. 16

DEWOLF, T. *Robot Control Part 3: Accounting for Mass and Gravity*. 2013. Disponível em: <https://studywolf.wordpress.com/2013/09/07/robot-control-3-accounting-for-mass-and-gravity/>. Acesso em: 24 maio 2019. 17, 28, 29, 30, 32

ESPIAU, B.; BOULIE, R. *On the Computation and Control of the Mass Center of Articulated Chains*. Tese (Doutorado) — INRIA, 1998. 16, 17, 28, 29, 30, 32, 50

FARAGHER, R. Understanding the basis of the kalman filter via a simple and intuitive derivation. In: IEEE. *IEEE SIGNAL PROCESSING MAGAZINE*. [S.l.], 2012. p. 128–132. 17, 42

FOOTE, T. tf: The transform library. *Technologies for Practical Robot Applications (TePRA)*, 2013. 34, 35

FREESE, M. et al. Virtual robot experimentation platform v-rep: A versatile 3d robot simulator. In: *Simulation, Modeling, and Programming for Autonomous Robots*. [S.l.]: Springer - Verlag Berlin Heidelberg, 2010. p. 51–62. 46

FU, K. S.; GONZALEZ, R. C.; LEE, C. S. G. *Robotics: Control, Sensing, Vision and Intelligence*. [S.l.]: McGraw-Hill Book Company, 1987. 12–75;82–102;220–221 p. 17, 19, 21, 22, 23, 24, 25, 28, 30, 31, 32, 75

GRUSHKO, S.; BOBOVSKY, Z. Teleoperated humanoid robot. 2016. 16, 50

HAYKIN, S. *Kalman Filtering and Neural Networks*. [S.l.]: Wiley-Interscience, 2001. v. 1. 1–10 p. 17, 40, 41

IVALDI, S.; PADOIS, V.; NORI, F. Tools for dynamics simulation of robots: a survey based on user feedback. 2014. 48

JUNIOR, W. L. R. et al. Aplicação de controladores fuzzy e proporcional para um robô seguidor de parede autônomo em ambiente estático. *Revista de Sistemas e Computação-RSC*, v. 6, n. 1, 2016. 17, 46, 50

KIRNER, C.; SISCOUTO, R. *Realidade Virtual e Aumentada: Conceitos, Projeto e Aplicações*. [S.l.]: Editora SBC - Sociedade Brasileira de Computação, 2007. 10 p. 16

KONOLIGE, K.; MIHELICH, P. *Technical description of Kinect calibration*. 2012. Disponível em: <http://wiki.ros.org/kinect_calibration/technical>. Acesso em: 31 outubro 2018. 38

LEEKWIJCK, W. V.; KERRE, E. E. Defuzzification: criteria and classification. *Fuzzy sets and systems*, Elsevier, v. 108, n. 2, p. 159–178, 1999. 17, 46

LI, C. et al. Development of kinect based teleoperation of nao robot. In: IEEE. *2016 International Conference on Advanced Robotics and Mechatronics (ICARM)*. [S.l.], 2016. p. 133–138. 50

LIANG, P. et al. An augmented discrete-time approach for human-robot collaboration. *Discrete Dynamics in Nature and Society*, Hindawi, v. 2016, 2016. 50

LIGUTAN, D. D. et al. Design and implementation of a fuzzy logic-based joint controller on a 6-dof robot arm with machine vision feedback. *Computing Conference 2017*, IEEE, p. 249–257, 2017. 16, 45, 50

MANEETHAM, D.; LENG, S. Pc-based 5dof industrial robotic arm with object color sorting by imageprocessing. In: *SNRU Jour nal of Science and Technology 10*. [S.l.: s.n.], 2018. p. 145–155. 55

MICHALOS, G. et al. Design considerations for safe human-robot collaborative workplaces. *Procedia CIrP*, Elsevier, v. 37, p. 248–253, 2015. 16

NGUYEN, C. V.; IZADI, S.; LOVELL, D. Modeling kinect sensor noise for improved 3d reconstruction and tracking. In: IEEE. *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012 Second International Conference*. [S.l.], 2012. p. 524–530. 38, 57

NGUYEN, H. X. et al. Performance evaluation of an inverse kinematic based control system of a humanoid robot arm using ms kinect. In: IEEE. *2017 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. [S.l.], 2017. p. 469–474. 16, 50

NILSSON, R. *Inverse Kinematics*. 2009. 17, 25, 26, 27

NOGUEIRA, L. Comparative analysis between gazebo and v-rep robotic simulators. In: *Seminário Interno de Cognição Artificial - SICA*. [S.l.: s.n.], 2014. p. 5–9. 48

OGATA, K. *Engenharia de Controle Moderno*. [S.l.]: Pearson, 2010. 2;521–525 p. 31, 32

OUALID, B.; OUSSAMA, C. *Design of a decoupling position control for the ED-7220C 5-axes robot*. 2016. 55

PAGI, T.; PADMAJOTHI, V. Gesture based robotic arm control using hand movements. 2017. 16, 50

PEDRO, L. M. *Uma Proposta de Sistema Robótico para Manipulação e Interação Física Segura em Ambientes não Estruturado*. Tese (Doutorado) — Universidade de São Paulo, 2013. 16, 50

QUIGLEY, M. et al. Ros: an open-source robot operating system. In: KOBE. *ICRA workshop on open source software*. [S.l.], 2009. v. 3, p. 5. 33

QUIGLEY, M.; GERKEY, B.; SMART, W. D. *Programming Robots with ROS: a practical introduction to the Robot Operating System*. [S.l.]: O'Reilly, 2015. 138–140 p. 36

RIA. *Defining The Industrial Robot Industry and All It Entails*. 2008–2019. Disponível em: <https://www.robotics.org/robotics/industrial-robot-industry-and-all-it-entails>. Acesso em: 31 outubro 2018. 16

ROBOTICS, C. *Writing code in and around V-REP*. 2010. Disponível em: <http://www.coppeliarobotics.com/helpFiles/index.html>. Acesso em: 28 maio 2019. 47

RODRIGUES, J. F. *Pesquisa Experimental*. 201–. Disponível em: <http://www.escritacientifica.sc.usp.br/wp-content/uploads/MPCC_5_DataAnalysis06-PesquisaExperimental.pdf>. Acesso em: 31 outubro 2018. 51

ROHMER, E.; SINGH, S. P. N.; FREESE, M. V-rep: a versatile and scalable robot simulation framework. In: IEEE. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. [S.l.], 2013. p. 1321–1326. 46

ROS. *Wiki ROS*. 2007. Disponível em: <http://wiki.ros.org/>. Acesso em: 31 Outubro 2018. 33

ROS. *Wiki Ros: actionlib*. 2011. Disponível em: <http://wiki.ros.org/actionlib>. Acesso em: 27 Maio 2019. 37

ROZO, L. et al. Learning physical collaborative robot behaviors from human demonstrations. *IEEE Transactions on Robotics*, IEEE, v. 32, n. 3, p. 513–527, 2016. 16

SANTOS, H. B. et al. Control of mobile robots using actionlib. In: SPRINGER. *Robot Operating System (ROS)*. [S.l.], 2017. p. 161–189. 32, 51, 57, 59, 60, 61

SCOPEL, A. d. R.; BARCELOS, P. C. Desenvolvimento de modelo icônico em estação pick & place. 2017. 19, 20, 21

SILVA, L. O. T. d. et al. Interação natural para manipulação de um braço robótico em aplicação da realidade aumentada por meio do reconhecimento de gestos usando o kinect. 2012. 16, 50

SMITH, S. W. *The Scientist and Engineer's Guide to Digital Signal Processing*. [S.l.]: California Technical Publishing, 1999. v. 2. 277–279 p. 17, 39, 40

SYSTEM, I. R. . *Arm Robot Trainer: ED-7220C*. Disponível em: <http://www.adinstruments.es/WebRoot/StoreLES/Shops/62688782/4C61/2F15/726A/B301/6188/C0A8/28BB/86B9/ED_7220C.pdf>. Acesso em: 31 outubro 2018. 55

THRUN, S. *Artificial Intelligence for Robotics*. 2011. Disponível em: <https://www.udacity.com/course/viewer#!/c-cs373/l-48704330/e-48748083/m-48665991>. Acesso em: 31 outubro 2018. 17, 42

TING, J.-A.; THEODOROU, E.; SCHAAL, S. Learning an outlier-robust kalman filter. In: ECML. *Machine Learning:ECML 2007*. [S.l.]: Springer Berlin Heidelberg, 2007. p. 748–756. 17, 43, 44, 68

ZADEH, L. A. Fuzzy sets. *Information and control*, Elsevier, v. 8, n. 3, p. 338–353, 1965. 17

ZANCHETTIN, A. M. et al. Safety in human-robot collaborative manufacturing environments: Metrics and control. *IEEE Transactions on Automation Science and Engineering*, IEEE, v. 13, n. 2, p. 882–893, 2015. 16

ZHANG, P. *Advanced Industrial Control Technology*. [S.l.]: William Andrew, 2010. 291–295 p. 17, 45, 46

ZHAO, J. et al. Interactive mechanical arm control system based on kinect. In: IEEE. *2016 35th Chinese Control Conference (CCC)*. [S.l.], 2016. p. 5976–5981. 51