UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS GRADUAÇÃO EM INFORMÁTICA

BRUNO GABRIEL CAVALCANTE LIMA

Controle Natural Humano-Robô orientado a usuário com Thin-Plate Splines e LRCN

Maceió-AL

Maio de 2021

BRUNO GABRIEL CAVALCANTE LIMA

Controle Natural Humano-Robô orientado a usuário com Thin-Plate Splines e LRCN

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-Graduação em Informática do Instituto de Computação da Universidade Federal de Alagoas.

Orientador: Prof. Dr. Tiago Figueiredo Vieira

Maceió-AL

Maio de 2021

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecário: Jone Sidney A. de Oliveira – CRB-4 – 1485

L732c Lima, Bruno Gabriel Cavalcante.
Controle natural humano-robô orientado a usuário com thin-Plate splines e lrcn. / Bruno Gabriel Cavalcante Lima, - 2021.
104 f. : il. col.

Orientador: Prof. Dr. Tiago Figueiredo Vieira.
Dissertação (Mestrado em Informática) - Universidade Federal de Alagoas. Instituto de Computação. Maceió, 2021.

Bibliografia: f. 94 - 98.
Anexo: f. 99 – 104.

1. Interação Homem-Robô. 2. Teleoperação, 3. Mapeamento Cinemático. 4. Inteligência Artificial. I. Título.

CDU: 004.89

AGRADECIMENTOS

A Deus, por sempre me proporcionar força e serenidade.

À minha família, pela educação e suporte.

Ao meu orientador Tiago Vieira, pelo suporte nos momentos mais difíceis, sempre com ótimos conselhos e orientações.

Aos colegas de laboratório que contribuíram para a realização deste trabalho.

Aos técnicos da secretaria do IC e do PPGI, pela disposição e boa vontade.

A todos os professores que me fizeram crescer acadêmica, pessoal e profissionalmente.

À FAPEAL, edital 05/2018, pelo suporte financeiro para realização deste trabalho de pesquisa.

À UFAL e ao IC como um todo, pela infra-estrutura e pelo ambiente acolhedor.

“Nunca se afaste dos seus sonhos.
Se eles se forem, continuarás vivendo...
mas terás deixado de existir.”
(Charles Chaplin)

RESUMO

Este trabalho propõe uma nova abordagem no ramo da teleoperação de braços robóticos baseada em visão computacional, Thin-Plate Splines e Redes Recorrentes. Utilizando uma única câmera de profundidade como sensor de entrada, tal abordagem isenta o usuário da necessidade de utilizar quaisquer dispositivos vestíveis. Através da aplicação de uma interface natural de usuário, esta abordagem inovadora facilita o ajuste fino paramétrico outrora necessário para a calibração de usuário para o controle robótico de posição, transformando o processo de calibração numa direta captura de poses do corpo humano. Utilizando a mão do usuário como forma de controle, a abordagem proposta é constituída por duas partes principais. A primeira é um mapeamento de posição customizável com componentes lineares e não-lineares, baseado em Thin-Plate Splines (TPS), para transferir diretamente o movimento do braço humano para o movimento do braço robótico. Tal mapeamento permite a correspondência de corpos dissimilares com diferentes restrições cinemáticas e diferentes formatos de espaço de trabalho. A segunda é um classificador dinâmico do estado da mão do usuário, baseado em Redes Recorrentes Convolucionais de Longo-Prazo (LRCN), que explora a coerência temporal dos dados de profundidade adquiridos. Ao fim, é realizada uma validação e avaliação da abordagem proposta. Para o classificador da mão, é realizada uma validação cruzada comparando a abordagem proposta com um *baseline*. Resultados revelam uma elevação na acurácia do classificador ao se explorar as relações temporais entre as imagens de profundidade. Para o mapeamento de movimento, é realizado um estudo com usuários envolvendo variantes da tarefa de *pick-and-place* em um cenário simplificado de manufatura. Para esse estudo, um ambiente de validação foi desenvolvido utilizando o Robot Operaing System (ROS) como *framework* principal. Também comparado a um baseline, a abordagem utilizando TPS revelou maior conforto e precisão no controle dos usuários em regiões próximas dos limites do espaço de trabalho do robô, nas quais a abordagem convencional se mostrava prejudicada. Além disso, resultados sugerem que a nova abordagem não apresentou aumento na dificuldade das tarefas.

Palavras-chaves: Interação Homem-Robô, Teleoperação, Interface Natural de Usuário, Mapeamento Cinemático, Redes Convolucionais Recorrentes de Longo Prazo.

ABSTRACT

This work proposes a novel vision-based robotic-arm teleoperation approach. By using a single depth-based camera, such an approach exempts the user from using any wearable devices. Through applying a natural user interface, such an approach also leverages the conventional fine-tuning process of the robotic position control calibration, turning the process into a direct capture of the human body. The proposed approach consists of two main parts. The first is a nonlinear customizable movement mapping based on Thin-Plate Splines (TPS), to directly transfer human body motion to robotic arm motion. Such mapping allows for matching dissimilar bodies, with different kinematics constraints and different workspace shapes. The second is a Deep Neural Network hand-state classifier based on Long-term Recurrent Convolutional Networks (LRCN), which exploits the temporal coherence of the acquired depth data. In the end, validation and evaluation of the proposed approach are performed. For the hand-state classifier, a cross-validation experiment comparing the current approach with a baseline is performed. Results reveal an increase in the classifier accuracy through exploring the temporal coherence present in sequential depth data. For the movement mapping, a user study is performed over a set of practical experiments involving variants of pick-and-place tasks in a simplified manufacturing environment. For this study, we developed a validation environment using Robot Operating System (ROS) as the main framework. Also compared to a baseline, the position mapping approach using TPS revealed better comfort and precision of user control in regions near to robot workspace boundaries, where the baseline approach showed a poor performance. Moreover, results suggested that the new approach did not present an increase in the experiment's task difficulty.

Keywords: Human-Robot Interaction, Teleoperation, Natural User Interface, Kinematics Mapping, Thin Plate Spline, Long-Term Recurrent Convolutional Networks.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

HRI	Human-Robot Interaction
DoF	Degrees of Freedom
RGB	Red, Green and Blue
RGBD	RGB plus Depth
AI	Artificial Inteligence
ML	Machine Learning
DL	Deep Learning
ANN	Artificial Neural Network
CNN	Convolutional Neural Networks
LCRN	Long-term Recurrent Convolutional Networks
LSTM	Long Short-term Memory Networks
BPTT	Backpropagation Through Time
NUI	Natural User Interface
SDK	Software Development Kit
ToF	Time of Flight
IR	Infra-Red
ROS	Robot-Operating System
FK	Forward Kinematics
IK	Inverse Kinematics
DH	Denavit-Hartenberg
PID	Proportional, Integral Derivative (control)
TCP	Transmission Control Protocol
URDF	Universal Robot Description Format
ROS	Robot Operating System

VR	Virtual Reality
VRCR	Virtual Reality Control Room
GUI	Graphical User Interface
CLI	Command Line Interface

SUMÁRIO

1 INTRODUÇÃO

Este capítulo contextualiza o desenvolvimento do presente trabalho, apresentando o problema abordado e suas características, bem como a abordagem proposta. Ao fim desta seção, são destacadas as principais contribuições desta pesquisa.

1.1 Motivação

A área da robótica tem se tornado cada vez mais presente em nosso cotidiano. Podemos exemplificar aplicações robóticas voltadas a diminuir o esforço físico humano, por exemplo, através da delegação de tarefas onerosas para as máquinas. Isso é confirmado desde a etimologia da palavra robô, do tcheco *robota*, que significa trabalho forçado, laborioso, repetitivo. O robô seria a máquina capaz de realizar trabalhos do homem, porém desprovido de sentimentos ou fadiga. Não há dúvidas quanto ao contínuo avanço da robótica nos últimos anos. Podemos citar exemplos que vão do ambiente residencial, com os robôs de serviço (IROBOT CORPORATION, LTD., 2021; WILLOW GARAGE, LTD., 2021; BOSTON DYNAMICS, LTD., 2021), à exploração espacial, com os robôs de missão específica (ARTEMIS, 2021; PERSEVERANCE, 2021). Em sua maioria, está presente a Interação Homem-Robô (do inglês *Human-Robot Interaction*, ou HRI), que constitui o campo da robótica onde homens e robôs se comunicam local ou remotamente para atingir um objetivo bem definido (GOODRICH; SCHULTZ, 2008).

A interação entre homens e robôs pode ser próxima ou distante, quando avaliados os aspectos temporal e espacial. Com relação ao aspecto espacial, nos casos em que um robô é comandado à distância, utiliza-se o termo Teleoperação (BARROS; LINDEMAN, 2009). Quando há um efetuador físico envolvido, que permite a manipulação de objetos próximos ao robô, utilizamos uma variante deste termo, chamada Tele-manipulação.

Em Nourbakhsh e Siegwart (2004), são citados exemplos históricos de uso da tele-manipulação. Um deles é o robô Pioneer, utilizado para inspecionar o sarcófago de Chernobyl, onde índices de radiação eram nocivos à saúde humana. Inclusive, segundo Spong et al. (2006), o conceito de tele-manipulação surgiu durante a Segunda Guerra Mundial, com o desenvolvimento de alternativas para a manipulação de materiais radioativos.

Caso a tele-manipulação ocorra dentro de um cenário de proximidade temporal, ou seja, onde a comunicação entre homem e robô ocorra sem atrasos perceptíveis, uma das formas de tratar o problema é através do controle direto do robô, onde um usuário controlador age baseando-se no retorno dos sensores disponíveis, como câmeras, microfones e sensores de distância. Tal forma de controle vai de contra ao paradigma do comportamento autônomo, sendo geralmente preferível no caso de situações críticas, em cenários não-controlados e imprevisíveis, onde uma rápida e complexa resposta se faz necessária (GOODRICH; SCHULTZ, 2008). Assim, é possível aproveitar o melhor dentre os dois mundos: (1) a precisão dos sensores e a capacidade

de interação do robô com o ambiente; e (2) o controle direto sobre este mesmo robô pelo usuário controlador, aproveitando da habilidade motora, percepção espacial e capacidade cognitiva humanas. Cenários que se beneficiam da tele-manipulação direta incluem: inspeção de obras; inspeção industrial; busca de sobreviventes em desabamentos; desarmamento de explosivos.

Dado o cenário discutido, faz-se necessário o desenvolvimento de interfaces práticas e eficientes que viabilizem tal abordagem. Os controles e interfaces convencionais, baseados em botões e *switches*, deixam de suprir tais necessidades à medida que o número de graus de liberdade (*degrees of freedom*, ou ainda DOF) presentes no robô se torna elevado (JR et al., 2017). Por conseguinte, é indispensável o desenvolvimento de alternativas de controle manual eficientes para casos de tele-manipulação em ambientes não-controlados com proximidade temporal.

1.2 Justificativa

Uma abordagem que satisfaz os requerimentos de praticidade e eficiência para a tele-manipulação em proximidade temporal são as Interfaces Natural de Usuário (*Natural User Interfaces*, ou ainda NUIs). Segundo Jr et al. (2017), NUIs são formas eficientes, intuitivas, ergonômicas e não-intrusivas de interação com sistemas controlados.

Na literatura, há exemplos comuns de NUIs que utilizam o movimento do corpo humano capturado por dispositivos vestíveis, como luvas (HOKAYEM; SPONG, 2006; XIAO et al., 2014; LIPTON; FAY; RUS, 2017; LI; WANG; LIU, 2021) e sensores de eletromiografia (Gowtham et al., 2020; LEE; PARK; PARK, 2018), também chamados sensores EMG. Em ambas as variantes, é necessário atrelar dispositivos de leitura ao corpo do usuário controlador, o que peca em ergonomia e não-intrusividade, além de dificultar o processo de troca de usuário.

Uma alternativa aos dispositivos vestíveis para a tele-manipulação robótica é a captura e o uso do movimento do corpo humano através de câmeras de profundidade como forma de controle, metodologia abordada neste trabalho. As câmeras de profundidade tem se mostrado promissoras no reconhecimento de gestos (SHARP et al., 2015; OYEDOTUN; KHASHMAN, 2017; LI, 2012; AMARAL et al., 2018; CHENG et al., 2019). No ramo da tele-manipulação robótica, elas também tem apresentado bons resultados (LIPTON; FAY; RUS, 2017; LIMA et al., 2019; CHEN et al., 2020), embora seja necessário uma rotina inicial de calibração geralmente custosa para que a interface se adapte a um usuário específico. Outrossim, diferentes usuários podem apresentar diferentes limitações cinemáticas, além de potencial assimetria em seu espaço de trabalho. Tais características geram a necessidade de um estudo de modelo de controle que atenda às necessidades específicas dos usuários, mantendo a eficiência, intuitividade, ergonomia e não-intrusividade, ao passo que a mobilidade do dispositivo robótico seja preservada.

Para atender às necessidades discutidas, o presente trabalho sugere um novo método de mapeamento de movimento humano-robô baseado no mapeamento de poses-chave (*key-poses*) através de modelos não-lineares que utilizam *Thin-Plate Splines* (BOOKSTEIN, 1989). A não-

linearidade inerente ao modelo permite o mapeamento entre diferentes usuários com diferentes limitações cinemáticas para o mesmo robô, sem que haja perda de mobilidade. Além disso, como o processo de calibração é diferenciado numa captura de poses, é possível mapear os mesmos pontos capturados do usuário para diferentes robôs (com diferentes formas de espaço de trabalho).

Além do mapeamento de movimento, o presente trabalho também avalia 2 (duas) técnicas de classificação da mão do usuário controlador, ambas baseadas em Redes Neurais Convolucionais (*Convolutional Neural Networks*, ou CNNs) que utilizam os mesmos dados captados pela câmera profundidade. A diferença entre elas é que uma é estática, considerando apenas a imagem de profundidade atual na entrada, enquanto a outra é dinâmica, considerando uma sequência de imagens como entrada.

1.3 Objetivo

O objetivo geral deste trabalho é analisar a viabilidade de um sistema de tele-manipulação robótica baseado em Computação Visual e Thin-Plate Splines de forma a aumentar o conforto e reduzir a influência de limitações cinemáticas por parte do usuário sobre o processo. Uma arquitetura geral do sistema analisado é ilustrado na Figura 1.

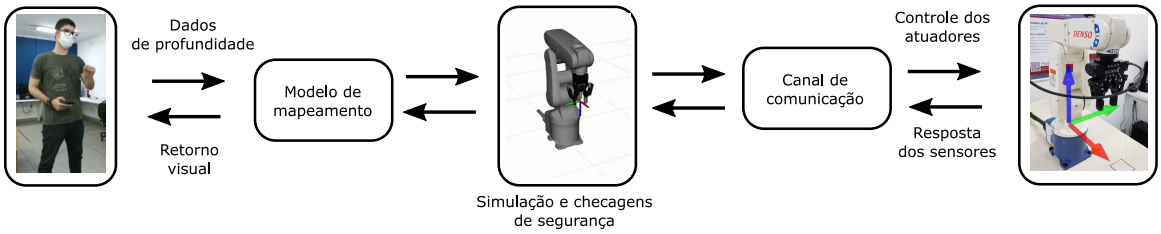

Figura 1 – Esquema visual simplificado do sistema de tele-manipulação proposto. O controle robótico de posição será realizado apenas através do movimento do corpo, sem o uso de joysticks ou dispositivos vestíveis. A informação visual do ambiente é utilizada como *feedback*.

Os objetivos específicos são:

- Desenvolver uma aplicação de comunicação, simulação e testes do braço robótico utilizando o ROS (Robot Operating System). Tal aplicação será utilizada para a realização de experimentos com o usuário;
- Realizar o estudo comparativo entre dois modelos mapeamento de movimento do braço humano para um braço robótico (*Naive-Affine* e *Thin-Plate Spline*);
- Realizar o estudo de dois modelos de mapeamento do estado da mão do usuário pare o efetuador robótico (Redes Convolucionais puras e Redes Recorrentes de Longo-prazo) para complementar o sistema de Tele-manipulação.

Dados os objetivos citados, o presente trabalho visa responder às seguintes perguntas de pesquisa:

QP1: *É possível diminuir a influência das limitações cinemáticas próprias de cada usuário sobre o espaço de trabalho alcançável pelo robô, no contexto da tele-manipulação robótica baseada em captura de movimento?*

QP2: *A relação sequencial entre os frames das imagens de profundidade contribui para aumentar a precisão do classificador neural de gestos de mão, no contexto da tele-manipulação robótica?*

1.4 Contribuição da pesquisa

As principais contribuições deste trabalho são as seguintes:

- Propomos o uso de uma forma inovadora de mapeamento humano-robô através de modelos não-lineares baseados em *Thin-Plate Splines*.
- Propomos a exploração das relações temporais na classificação da mão do usuário, através de Redes Neurais Recorrentes de Longo-prazo.
- Propomos e implementamos uma aplicação distribuída em tempo-real para simulação e testes com tele-manipulação robótica;

1.5 Estrutura da dissertação

O Capítulo 2 apresenta uma abordagem geral dos conceitos teóricos utilizados no decorrer deste trabalho, incluindo as principais ferramentas de software. O Capítulo 3 apresenta trabalhos que abordaram a tele-manipulação robótica ou que utilizaram métodos similares. O Capítulo 4 trata da solução proposta e implementada, bem como suas principais características. Por fim, o Capítulo 5 trata dos resultados obtidos ao aplicar a metodologia ao problema, bem como a discussão dos mesmos. O Capítulo 6 finaliza o trabalho apresentando sintetizando o que foi alcançado, discutindo sobre as perguntas de pesquisa e sugerindo potenciais trabalhos futuros.

2 FUNDAMENTAÇÃO

Este capítulo discute o ferramental teórico utilizado no desenvolvimento do presente trabalho. A Seção 2.1 aborda conceitos básicos sobre vetores no espaço e suas operações. A Seção 2.2 aborda os conceitos relacionados à câmera de profundidade, incluindo a captura das poses do esqueleto do usuário e das imagens de profundidade. A Seção 2.3 discute os aspectos envolvendo a robótica em si, bem como as principais características do ambiente de simulação utilizado. Por fim, a Seção 2.4 discute sobre os modelos de mapeamento e classificação baseados em Inteligência Artificial, constituindo o cerne deste trabalho.

2.1 Geometria Espacial e suas operações

Esta seção discorre sobre o ferramental matemático necessário para operações no espaço 3D. Serão tratados conceitos como vetores, espaços vetoriais, sistemas de coordenadas e transformações homogêneas.

2.1.1 Geometria Euclidiana

O sistema de coordenadas Cartesiano é uma representação matemática do espaço físico real, composta por n eixos principais ortogonais (por exemplo, $n = 2$, $n = 3$). Nessa representação, é possível representar posições, velocidades, acelerações, bem como outras derivadas da posição de maior ordem (SPONG et al., 2006).

Por espaço Euclidiano, entende-se um conjunto ordenado de vetores linearmente independentes, denominado base vetorial, sobre a qual se define pelo menos o produto interno. Duas operações são bastante utilizadas frequentemente quando nos referimos a vetores do espaço Euclidiano: o produto interno (ou produto escalar, que resulta num número), e o produto vetorial (resulta num vetor). Sejam os vetores $(\vec{u} = u_x, u_y, u_z)$ e $\vec{v} = (v_x, v_y, v_z)$, o produto interno e o produto vetorial desses vetores são definidos pelas Equações 2.1 e 2.2, respectivamente. A Equação 2.3 apresenta uma forma simplificada para o cálculo da norma do produto vetorial. Tais operações serão necessárias para a definição de sistemas de coordenadas, discutidas em seguida.

$$\vec{u} \cdot \vec{v} = \|\vec{u}\| \, \|\vec{v}\| \, \cos(\theta) \tag{2.1}$$

$$\vec{u} \times \vec{v} = \begin{Vmatrix} \hat{i} & \hat{j} & \hat{k} \\ u_x & u_y & u_z \\ v_x & v_y & v_z \end{Vmatrix} \tag{2.2}$$

$$\|\vec{u} \times \vec{v}\| = \|\vec{u}\| \, \|\vec{v}\| \, \sin(\theta) \tag{2.3}$$

O espaço Cartesiano é um exemplo de sistema de coordenadas. Um sistema i de coordenadas ortogonal pode ser definido fora da origem do espaço Cartesiano. Dois vetores são definidos como ortonormais entre si quando ambos são unitários (possuem norma 1) e ortogonais (formam 90 graus entre si). Nesse caso, i descreve uma orientação e uma posição, sendo definido por um conjunto ordenado de vetores ortonormais (x_i, y_i, z_i) que representam uma orientação bem definida com origem em um ponto O_i (SICILIANO; KHATIB, 2016). Ao conjunto formado por orientação e posição, damos o nome de frame ou pose. Assim, partindo da Equação 2.1, para quaisquer dois vetores u_i e v_i que fazem parte do sistema de coordenadas ortonormal i, valem as propriedades da Equação 2.4.

$$\vec{u_i} \cdot \vec{v_i} = \begin{cases} 1, \text{ caso } u_i = v_i \\ 0, \text{ caso } u_i \neq v_i \end{cases} \tag{2.4}$$

É importante frisar que qualquer grandeza vetorial precisa de um sistema de coordenadas de referência. Caso não seja especificado, assume-se o sistema de coordenadas global, cuja origem coincide com a origem do espaço Cartesiano, representada pelo ponto $p^0 = [0, 0, 0]$. Deste parágrafo em diante, o número supraescrito indicará sempre o sistema de coordenadas ortogonal sobre o qual o ponto está definido (0 significa origem global), não importando se ele venha antes ou depois do referido vetor.

Geralmente, na robótica, atrelamos um frame a um corpo rígido no espaço. Dessa forma, o movimento de um corpo rígido é suficientemente representado pelo movimento relativo entre dois frames, um deles denominado "frame móvel"enquanto o outro pode ser denominado "frame fixo"(sobre o qual se localiza um observador). Perceba que o conceito de frame fixo está diretamente relacionado com o observador, mas nada impede de também ele estar em movimento.

2.1.2 Translação e Rotação

O processo de translação de um corpo no espaço físico pode ser representado pelo deslocamento de um frame atrelado a ele, indo de uma posição inicial para uma posição final. Assim, qualquer representação de uma posição pode ser usada para criar a representação de um deslocamento, e vice-e-versa.

Por outro lado, a rotação de um corpo no espaço físico se configura como uma forma de movimento em torno de um eixo bem definido, de forma que as distâncias iniciais de todos os pontos do corpo rígido em relação a um certo eixo continuam a mesma antes, durante e após o movimento (SICILIANO; KHATIB, 2016). De forma similar, qualquer representação de orientação pode ser utilizada para criar a representação de uma rotação, e vice-e-versa.

Uma das formas de representar a orientação de um frame i relativamente a um frame j é através da matriz de rotação ${}^jR_i$. Ela pode ser formada através da projeção dos vetores $(\hat{x}_i, \hat{y}_i, \hat{z}_i)$

sobre os vetores $(\hat{x}_j, \hat{y}_j, \hat{z}_j)$, como mostra a Equação 2.5.

$$ {}^{j}R_i = \begin{pmatrix} \hat{x}^i \cdot \hat{x}^j & \hat{y}^i \cdot \hat{x}^j & \hat{z}^i \cdot \hat{x}^j \\ \hat{x}^i \cdot \hat{y}^j & \hat{y}^i \cdot \hat{y}^j & \hat{z}^i \cdot \hat{y}^j \\ \hat{x}^i \cdot \hat{z}^j & \hat{y}^i \cdot \hat{z}^j & \hat{z}^i \cdot \hat{z}^j \end{pmatrix} \tag{2.5} $$

As colunas da matriz ${}^{j}R_i$, que são formadas a partir do produto interno de vetores de bases ortonormais, também são ortonormais. Uma matriz formada por vetores mutualmente ortonormais é conhecido como uma matriz ortogonal, possuindo a propriedade de que sua inversa é igual à sua transposta.

Tomemos, agora, como exemplo, a rotação sobre os vetores que formam o sistema de coordenadas global. Utilizando as Equações 2.1 e 2.5, uma vez que os vetores do frame global são unitários, as seguintes rotações elementares sobre os eixos $\hat{x}$, $\hat{y}$ e $\hat{z}$ são derivadas geometricamente:

$$ R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & cos\theta \end{bmatrix} R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & cos\theta \end{bmatrix} R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6} $$

As matrizes da Equação 2.6 podem ser multiplicadas sequencialmente para formar rotações compostas de forma associativa, exemplificadas pela Equação 2.7.

$$ {}^{k}R_i = {}^{k}R_j \, {}^{j}R_i \tag{2.7} $$

Uma matriz de rotação contém 9 elementos, embora seja possível representar minimamente a orientação de um frame utilizando apenas 3 elementos (por exemplo, os ângulos em relação a cada um dos eixos elementares, conhecidos como ângulos de Euler). Tal fato resulta em 6 relações auxiliares existentes entre os elementos da matriz. Três dessas relações dizem respeito às colunas serem ortonormais, enquanto as outras três dizem respeito às colunas serem vetores unitários. Craig (2012) sintetiza as propriedades de uma matriz de rotação R_i^j como seguem:

- é ortogonal, pois as colunas são ortonormais duas a duas;
- transforma um vetor definido em coordenadas do frame móvel i para um vetor definido em coordenadas do frame fixo j, assumindo ambos numa origem em comum;
- pode ser entendida como a rotação necessária ao frame fixo j para alinhar seus eixos com o frame móvel i, assumindo ambos numa origem em comum;
- pertence ao grupo especial ortogonal de dimensão 3. Isso significa que o produto de quaisquer duas matrizes que pertençam ao grupo também pertencerá ao grupo;

- seu determinante $\det \|^j R_i\|$ possui sempre valor igual a $+1$, resultando no fato do comprimento de um vetor permanecer inalterado após a operação de rotação;
- o inverso da matriz é igual à sua transposta, ou seja, $R^{-1} = R^T$

2.1.3 Matrizes Homogêneas

Através das matrizes de transformação homogênea, é possível combinar posição e orientação numa mesma representação. As regras de associatividade das matrizes de rotação também continuam valendo para as transformações homogêneas (SICILIANO; KHATIB, 2016).

Seja o vetor ${}^i v$ expresso em relação ao frame i. Tal vetor pode ser expresso em relação a um outro frame j, desde que a posição e orientação de i sejam conhecidas em relação a j. Assuma o vetor posição definido por ${}^j p_i = ({}^j x_i, {}^j y_i, {}^j z_i)^T$. A orientação do frame i em relação a j é definida pela matriz de rotação ${}^j R_i$. Assim, temos que:

$$ {}^j v = {}^j R_i \cdot {}^i v + {}^j p_i \tag{2.8} $$

A Equação 2.8 pode ser re-escrita em sua forma matricial, como segue:

$$ \begin{pmatrix} {}^j v \\ 1 \end{pmatrix} = \begin{pmatrix} {}^j R_i & {}^j p_i \\ 0_{1x3} & 1 \end{pmatrix} \begin{pmatrix} {}^i v \\ 1 \end{pmatrix} \tag{2.9} $$

A matriz responsável pela aplicação da transformação possui formato $4x4$, localizada à esquerda da matriz coluna que contém v_i. Tal matriz é denominada matriz de transformação homogênea, definida formalmente na Equação 2.10.

$$ {}^j T_i = \begin{pmatrix} {}^j R_i & {}^j p_i \\ 0_{1x3} & 1 \end{pmatrix} \tag{2.10} $$

É possível partir da Equação 2.8 e das propriedades da matriz de rotação discutidas para realizar a transformação inversa. Nesse caso, temos que:

$$ -{}^j R_i \cdot {}^i v = {}^j p_i - {}^j v \tag{2.11} $$

$$ {}^j R_i \cdot {}^i v = {}^j v - {}^j p_i \tag{2.12} $$

$$ {}^j R_i^T \cdot {}^j R_i \cdot {}^i v = {}^j R_i^T \cdot ({}^j v - {}^j p_i) \tag{2.13} $$

$$ {}^i v = {}^j R_i^T \cdot ({}^j v - {}^j p_i) \tag{2.14} $$

Escrevendo a equação 2.14 matricialmente, temos:

$$\begin{pmatrix} {}^{i}v \\ 1 \end{pmatrix} = \begin{pmatrix} {}^{j}R_i^T & -{}^{j}R_i{}^T {}^{j}p_i \\ 0_{1x3} & 1 \end{pmatrix} \begin{pmatrix} {}^{j}v \\ 1 \end{pmatrix} \tag{2.15}$$

A matriz inversa da matriz ${}^{j}T_i$, denotada por ${}^{j}T_i^{-1}$, transforma vetores do frame j ao frame i. Formalmente, tal matriz é definida como:

$${}^{j}T_i^{-1} = {}^{i}T_j = \begin{pmatrix} {}^{j}R_i^T & -{}^{j}R_i^T {}^{j}p_i \\ 0_{1x3} & 1 \end{pmatrix} \tag{2.16}$$

2.2 Captura dos Dados de Entrada

Esta seção será responsável por tratar dos dados de entrada utilizados por ambos os modelos de mapeamento e classificação. Serão discutidos conceitos como o processo de captura dos dados de profundidade e das posições das juntas do usuário.

2.2.1 Câmera de Profundidade

O Microsoft Kinect v1 foi lançado em 2010, sendo um dos mais populares controladores de jogo já lançados, vendendo 24 milhões de unidades até 2013 (LUN; ZHAO, 2015). Sua proposta inovadora era permitir a interação natural com um computador ou console de jogo através de gestos e/ou comandos de voz. Com tal popularidade, cientistas passaram a utilizar o sensor para aplicações em outras áreas, a citar: saúde, educação, realidade virtual, reconhecimento de sinais e robótica.

A câmera de profundidade utilizada neste trabalho foi o Microsoft Kinect v2 (MICROSOFT, 2014), lançado em 2014. Entre as suas especificações estão uma câmera RGB com resolução 1920x1080 e uma câmera de profundidade com resolução 512x424. As versões 1 e 2 do sensor estão ilustradas na Figura 2. As principais diferenças entre o Kinect V2 e seu predecessor consistem na resolução da câmera, no número de pontos do esqueleto captados e no padrão USB adotado[1]. Uma lista completa das diferenças está presente na Tabela 1. A produção do Kinect v2 foi encerrada em 2017[2] pela Microsoft, que agora volta esforços para o Kinect Azure[3]. Essa nova versão conta com a mais alta resolução da câmera de profundidade (1024x1024) dentre a família Kinect, agora voltada especificamente para aplicações de IA na indústria.

De acordo com Wasenmüller e Stricker (2016), o Kinect v1 realiza a inferência da profundidade com o princípio de Projeção de Padrões, onde um padrão bem-definido de luz estruturada infra-vermelha (do inglês *infra-red*, ou IR) é projetado na cena. A profundidade é,

[1] <https://channel9.msdn.com/coding4fun/kinect/Kinect-1-vs-Kinect-2-a-side-by-side-reference>
[2] <https://finance.yahoo.com/news/downfall-kinect-why-microsoft-gave-183900710.html>
[3] <https://azure.microsoft.com/pt-br/services/kinect-dk/>

Figura 2 – Ilustração visual do Microsoft Kinect v1 (à esquerda) lançado em 2010, e do Kinect v2 (à direita) lançado em 2014.

então, computada de acordo com a distorção desse padrão presente no retorno captado pelo receptor. A tecnologia foi desenvolvida pela PrimeSense (ZALEVSKY et al., 2013). Uma vez que é necessário haver certa distância entre cada dois pontos adjacentes projetados no ambiente, apenas 1 a cada 20 pixels projetados possuem a profundidade real da cena, com os demais sendo interpolados. Dessa forma, a resolução efetiva da câmera de profundidade do Kinect v1 é significantemente menor que a resolução nominal de 640x480. Além disso, os padrões projetados são fortemente prejudicados pela presença de forte iluminação e/ou calor no local. A Figura 3 demonstra o processo de projeção dos pontos além de exemplificar a interferência de reflexos luminosos no processo.

Por outro lado, o Kinect V2 se baseia no Tempo-de-Vôo (do inglês Time-of-Flight, ou ToF) de pulsos IR, determinando a profundidade dos objetos de acordo com o tempo que o pulso levou para refletir no objeto e retornar à câmera. São utilizados diodos lasers IR para projetar luz infra-vermelha modulada sobre a cena. Um gerador de pulsos é utilizado para sincronizar as ações de ambos, transmissor e receptor. O tempo de vôo é inferido de acordo com a defasagem entre o sinal emitido e o retornado.

Segundo Lun e Zhao (2015), uma novidade no Kinect v2 diz respeito ao emissor IR ser periodicamente ligado e desligado, como mostra a Figura 4. Dessa forma é possível reduzir os impactos da iluminação ambiente através da subtração do seu sinal (captado quando o emissor

Característica	Kinect v1	Kinect v2
Câmera RGB	640 x 480 @30 fps	1920 x 1080 @30 fps
Câmera de Profundidade	320 x 240	512 x 424
Máxima profundidade captada	$\sim$4.5 m	8 m
Mínima profundidade captada	40 cm	50 cm
FOV horizontal da profundidade	57 graus	70 graus
FOV vertical da profundidade	43 graus	60 graus
Motor de inclinação	sim	não
Número de Juntas no Esqueleto	20 juntas	25 juntas
Número de esqueletos simultâneos	2	6
Padrão USB	2.0	3.0

Tabela 1 – Principais diferenças entre o Kinect v1 e o Kinect v2. Fonte: Microsoft.

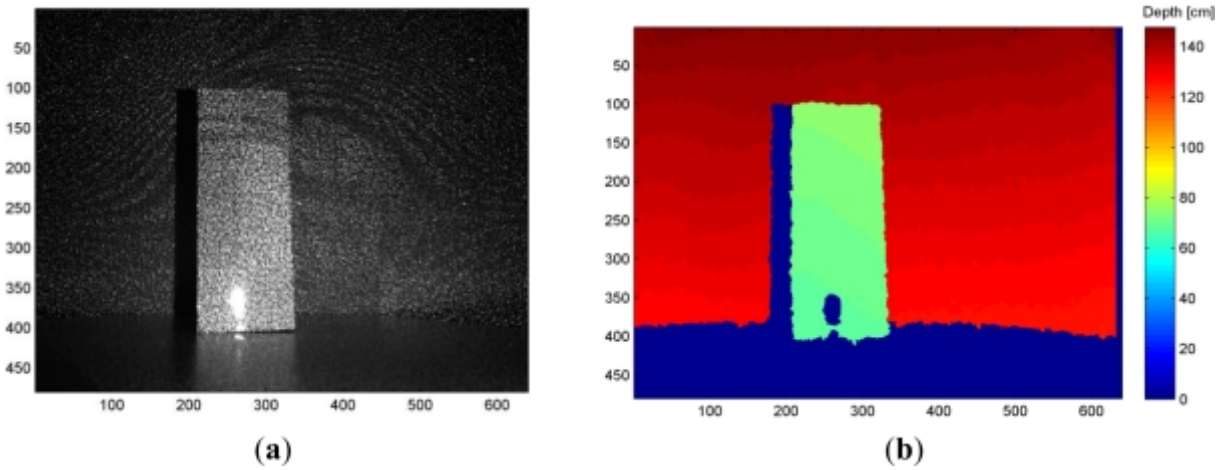

Figura 3 – Utilizando a projeção de padrões de pontos infra-vermelhos sobre o ambiente (a) é possível estimar a profundidade desses pontos por triangulação (b), uma vez que cada dois pontos adjacentes possuem configuração única. Além disso, reflexos luminosos de luz infravermelha deterioram o reconhecimento dos padrões. Figura retirada de Khoshelham e Elberink (2012), pg 2.

IR está desligado) sobre o sinal captado quando o emissor está ligado, sobrando apenas a luz modulada que fora emitida e refletida. No decorrer deste trabalho, ao utilizarmos o termo câmera de profundidade, assume-se que estamos falando do Kinect v2.

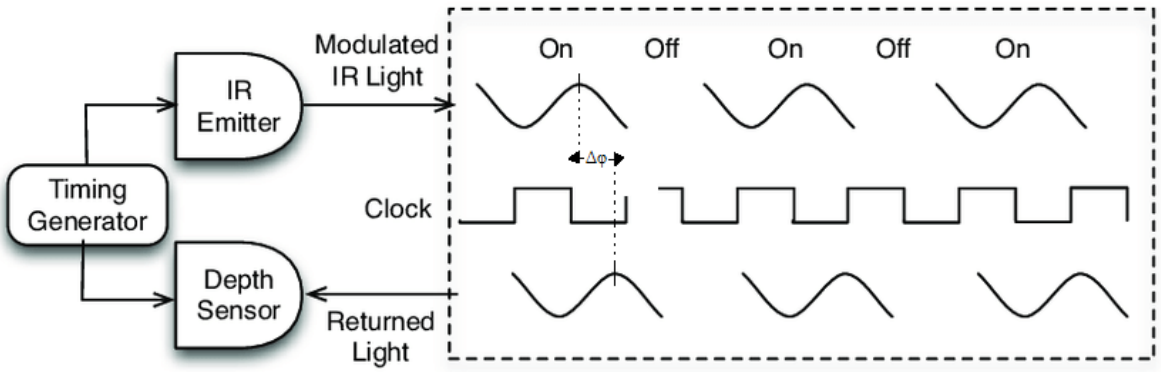

Figura 4 – Demonstração do processo de cálculo de distâncias do Kinect V2. Um pulso IR modulado é enviado ao ambiente para cada pixel da imagem, numa ação periódica. Ao retornar ao receptor, a defasagem $\Delta\phi$ entre o sinal emitido e capturado é calculada para inferir o tempo de vôo do pulso IR. Fonte: Lun e Zhao (2015), pg. 7.

2.2.2 Interface e leitura do usuário

Para poder utilizar os dados captados pela câmera, é necessário Kit de Desenvolvimento de Software v2 (SDK v2, utilizado no presente trabalho) disponibilizado pela Microsoft, ou outro conjunto de bibliotecas produzidas por terceiros, como o OpenNI[4]. É valido ressaltar que o

[4] <https://structure.io/openni>

Kinect v2 permite a conexão de apenas uma câmera ao computador, com leitura possivelmente compartilhada entre diferentes aplicações. Tal funcionalidade não estava presente no Kinect v1, que exigia a conexão de uma câmera exclusiva para cada aplicação.

O uso mais comum do Kinect v2 se baseia da leitura de 3 fluxos de dados de diferentes naturezas, a citar: quadros de imagem 2D; nuvem de pontos 3D; e esqueletos dos usuários. Como enfatizado por Lun e Zhao (2015), a disponibilidade das informações de esqueleto estimadas das imagens de profundidade acelera o desenvolvimento da aplicação uma vez que os desenvolvedores possuem um modelo de localização e pose humanas pronta para uso. Entretanto, nada impede o desenvolvedor de utilizar as imagens 2D e/ou 3D para criar seu modelo próprio de estimação de pose através do SDK. No tocante ao assunto, o Kinect v2 SDK já possui um classificador do estado da mão do usuário, com 3 classes distintas (mão fechada, mão aberta, e apontando). Nesse sentido, o presente trabalho visa oferecer uma alternativa customizada de classificação, com o potencial de expandir para mais classes a depender das necessidades do usuário.

É válido mencionar que a documentação do Microsoft Kinect v2 está disponível também no Github[5]. As interfaces de leitura para imagens RGB, de profundidade e frame do usuário estão presentes nas classes *ColorFrameReader*, *DepthFrameReader* e *BodyFrameReader*, respectivamente. Todas as classes citadas percentem ao namespace *WindowsPreview.Kinect*.

O modelo de estimação de pose de usuário padrão do Kinect v2, representado pelo *BodyFrameFreader*, retorna um objeto do tipo *Body* contendo as posições das juntas (como pulsos, cotovelos, ombros) baseado nas imagens captadas em relação ao frame de origem da câmera. A completa descrição dessa classe está presente na Tabela 2. De posse das juntas oferecidas pelo modelo, é possível calcular os vetores entre elas (e.x.: braço, antebraço).

No Kinect V2, o esqueleto é formado por 25 juntas distintas, que são demonstradas na Figura 5. Note que o Kinect fornece as juntas numa visão espelhada. Dessa forma, o lado esquerdo da imagem representa as juntas dos membros esquerdos do usuário, e vice-e-versa (Fonte: SeaLeft Studios[6]). Na seção Enumerations -> JointType da documentação do Kinect v2, é possível encontrar a referência (na forma de *enums*) e os índices para cada junta presente na Figura 5.

2.2.3 Projeção do Esqueleto sobre a imagem 2D

Em determinadas aplicações, faz-se desejável juntar a informação de diferentes fluxos de dados que possuem diferentes resoluções e que foram obtidos de diferentes sensores, em diferentes posições, numa mesma imagem. Por exemplo, pode ser necessário mapear a informação das juntas do usuário, as quais são disponibilizadas no espaço 3D com origem no frame da câmera, para a imagem RGB 2D, a fim de realizar uma análise visual do comportamento do usuário.

[5] <https://github.com/Kinect/Docs/blob/master/Kinect4Windows2.0/k4w2/Reference/C++_Reference.md>

[6] Adaptado de <https://www.sealeftstudios.com/blog/blog20160701.php>.

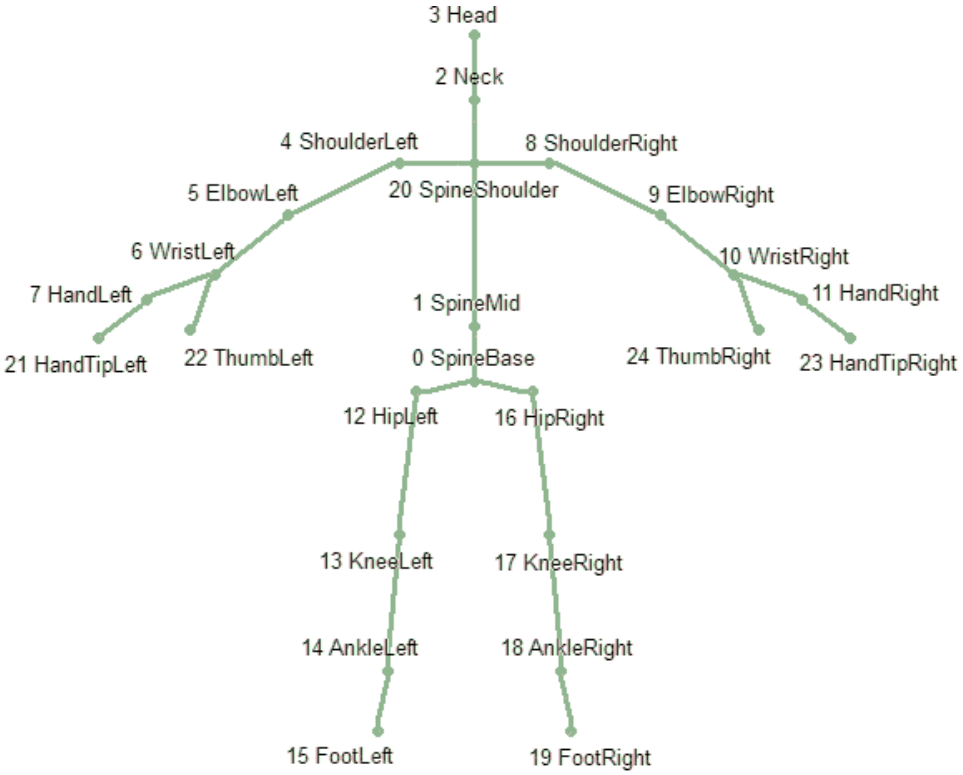

Figura 5 – Ilustração das juntas do usuário captadas pelo Kinect V2 SDK. O Kinect fornece as juntas numa visão espelhada do usuário. Fonte: SeaLeft Studios.

Função	**Descrição**
HandLeftConfidence	Gets the confidence of the body's left hand state.
HandLeftState	Gets the status of the body's left hand state.
HandRightConfidence	Gets the confidence of the body's right hand state.
HandRightState	Gets the status of the body's right hand state.
IsRestricted	Gets whether or not the body is restricted.
IsTracked	Gets whether or not the body is tracked.
JointCount	Gets the number of joints in a body.
JointOrientations	Gets the joint orientations of the body.
Joints	Gets the joint positions of the body.
Lean	Gets the lean vector of the body.
LeanTrackingState	Gets the tracking state for the body lean.
TrackingId	Gets the tracking ID for the body.

Tabela 2 – Excerto da classe WindowsPreview.Kinect.Body, contendo atributos levantes ao presente trabalho. Fonte: Kinect SDK Github.

Nesse caso, a classe *CoordinateMapper*, também presente no namespace *Windows.Kinect*, fornece soluções práticas para tal necessidade. A vantagem desta classe é a abstração de fatores como o alinhamento das câmeras e seus parâmetros intrínsecos, outrora necessários para a correta projeção entre diferentes espaços de coordenadas. Na Tabela 3 são mostradas algumas funções disponibilizadas pelo SDK para tal necessidade.

Note que, além de mapear pontos entre diferentes espaços, é possível também mapear frames (imagens formadas por pixels). Por exemplo, através da função *MapColorFrameToCameraSpace* é possível adicionar informações de profundidade à imagem RGB capturada, enxergando os pixels RGB no espaço 3D da câmera. A operação de junção de informações de diferentes espaços é exemplificada na Figura 6. Nela são mostradas exemplos de leitores de fluxo disponibilizados pelo Kinect V2 SDK, além de demonstrar o emprego do Nesse exemplo, vemos o emprego dos leitores de fluxos de dados para imagens IR e para as juntas do usuário, além de ver o emprego de funções do *CoordinateMapper*.

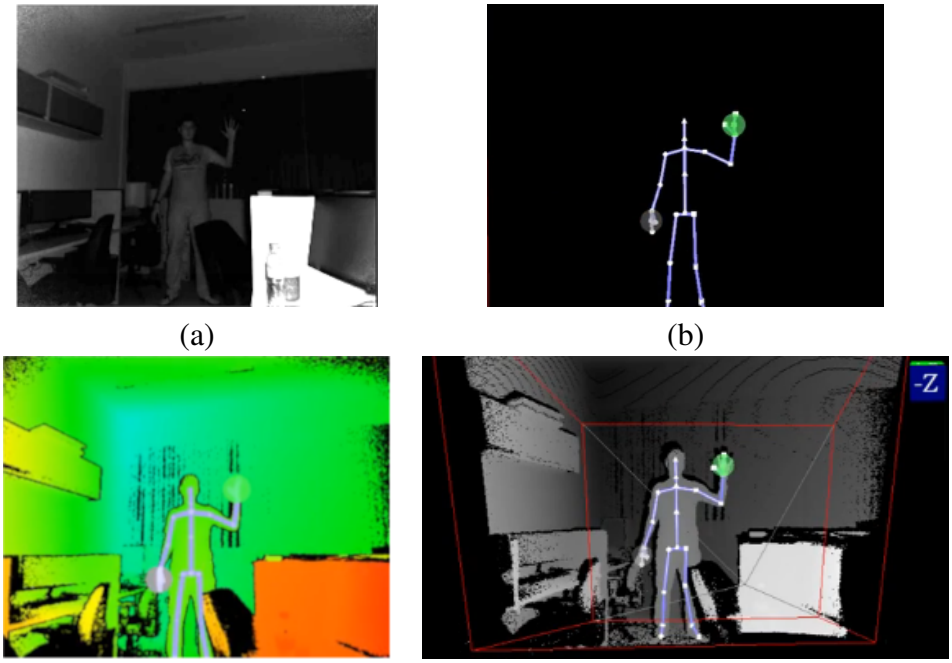

Figura 6 – Exemplo de mapeamento de coordenadas entre diferenças espaços. Em (a) vemos a imagem IR captada pelo *DepthFrameReader*. Em (b) vemos as juntas em 3D, junto aos vetores formados entre elas, captadas pelo *BodyFrameReader*. Em (c) vemos a projeção das juntas sobre a imagem de profundidade (colorizada), através da função *MapCameraPointToDepthSpace* disponível pelo *CoordinateMapper*. Em (d) vemos o mapeamento da image IR sobre o espaço 3D com origem no frame da câmera, através da função *MapDepthFrameToCameraSpace*, também presente no *CoordinateMapper*.

2.2.4 Transformação do Sistema de Coordenadas

Até o presente momento, discutimos as posições das juntas do usuário em relação ao sistema de coordenadas da câmera. Para determinadas aplicações, entretanto, pode ser desejável

Função	Descrição
GetDepthCameraIntrinsics	Returns the calibration data for the depth camera.
MapCameraPointToColorSpace	Maps a point from camera space to color space.
MapCameraPointToDepthSpace	Maps a point from camera space to depth space.
MapColorFrameToCameraSpace	Maps a frame from color space to camera space.
MapColorFrameToDepthSpace	Maps a frame from color space to depth space.
MapDepthFrameToCameraSpace	Maps a frame from depth space to camera space.
MapDepthFrameToColorSpace	Maps a frame from depth space to color space.
MapDepthPointToCameraSpace	Maps a point/depth from depth space to camera space.
MapDepthPointToColorSpace	Maps a point/depth from depth space to color space.

Tabela 3 – Excerto da descrição da classe WindowsPreview.Kinect.CoordinateMapper. Fonte: Microsoft.

ter em mãos a posição das juntas em relação a um frame localizado no próprio usuário, de forma que tais posições sejam invariantes à localização e orientação (em relação ao eixo Z) do usuário.

O primeiro passo é a definição do frame atrelado ao corpo do usuário. De posse das juntas, que são pontos no espaço 3D, é necessário formar os 3 vetores ortonormais (em nosso caso, aproximadamente ortonormais) que servirão de base para o novo sistema de coordenadas como 2.1.1. Uma possibilidade é a definição de um vetor vertical e um horizontal, com o terceiro vetor (de profundidade) sendo definido a partir do produto vetorial entre os dois primeiros.

Seja 0 o frame na origem da câmera, e seja u o frame atrelado ao corpo do usuário. De acordo com o discutido na Seção 2.2.4, para cada junta p_i, são válidas as seguinte relações:

$$
\begin{aligned}
{}^0p_i &= {}^0T_u\ {}^up_i && (2.17)\\
{}^0T_u^T\ {}^0p_i &= ({}^0T_u^T\ {}^0T_u)\,{}^up_i && (2.18)\\
{}^up_i &= {}^0T_u^T\ {}^0p_i && (2.19)\\
&= \begin{pmatrix} {}^0R_u^T & -{}^0R_u^T\ {}^0p_u \\ 0_{1x3} & 1 \end{pmatrix}\ {}^0p_i && (2.20)
\end{aligned}
$$

As equações anteriores demonstram a transformação homogênea responsável por transformar uma junta (ponto 3D) do espaço da câmera para o espaço do usuário, assumindo que ${}^0R_u$ e ${}^0p_u$ foram pré-estabelecidos. O vetor ${}^0p_u$ pode ser extraído diretamente das informações fornecidas pelo Kinect SDK v2 através da escolha de uma das juntas como origem do novo sistema de coordenadas. Já a matriz ${}^0R_u$ pode ser diretamente obtida a partir da Equação 2.5.

2.2.5 Mapeamento Naive

O mapeamento de posição utilizado como baseline neste trabalho foi denominado Mapeamento de Posição Ingênuo (do inglês *Naive Position Mapping*, ou NA-PM), sendo introduzido por Lima et al. (2019). O trabalho descreve um modelo de mapeamento entre a posição da mão

do usuário no sistema de referência do ombro para a posição do centro do efetuador no sistema de referência do da base do robô. Tal abordagem era invariante à posição e orientação do usuário no espaço da câmera.

O sistema de mapeamento ponto-a-ponto pode ser descrito como segue. Seja $\{rs\}$ um frame de referência localizado no ombro direito do usuário. Podemos representar o vetor posição da mão direita como ${}^{rs}\vec{u} = ({}^{rs}u_x, {}^{rs}u_y, {}^{rs}u_z)$. Assume-se que a posição da mão é normalizada com relação ao alcance máximo L do braço direito do usuário, o que leva a $\|{}^{rs}\vec{u}\| \in [-1, 1]$. Seja $\{b\}$ o frame de referência localizado na base do robô. Podemos representa o vetor posição do centro do efetuador como ${}^{b}\vec{P}_G = ({}^{b}P_{Gx}, {}^{b}P_{Gy}, {}^{b}P_{Gz})$.

O mapeamento NA-PM proposto por por Lima et al. (2019) é constituído de uma transformação afim composta por uma rotação, uma escala e um offset em cada eixo. Tal mapeamento é generalizado como segue:

$$ {}^{b}\vec{P}_G = \begin{bmatrix} \eta \cdot \omega_x \cdot {}^{rs}u_z + \Delta_x \\ \eta \cdot \omega_y \cdot {}^{rs}u_x + \Delta_y \\ \omega_y \cdot {}^{rs}u_y + \Delta_z \end{bmatrix} \tag{2.21} $$

onde:

- $\eta \in \{1, -1\}$ é uma constante de espelhamento;
- $\omega_x, \omega_y, \omega_z$ são constantes multiplicativas;
- $\Delta_x, \Delta_y, \Delta_z$ são constantes de offset no espaço do robô;

Perceba que tal mapeamento é parametrizado por 8 constantes, contando com o tamanho do braço do usuário L. Além disso, esse mapeamento assume um espaço de trabalho simétrico da parte do usuário (por exemplo, assume que o usuário possui o mesmo alcance para a direita e para a esquerda).

2.2.6 Mapeamento com Thin-plate Splines

Comecemos definindo o que são splines: também conhecidas como aproximação polinomial por partes de pontos $(x, f(x))$, são uma forma de interpolação que utilizam uma função de aproximação contínua e com derivadas contínuas até certa ordem (ÁVILA, 2019), além de obrigatoriamente englobar todos os pontos do conjunto de entrada (aproximação exata). As splines consideradas naturais devem também possuir valor da segunda derivada igual zero nos pontos da extremidade de cada intervalo, sendo assim consideradas uma forma suave de aproximação. Uma interpolação por splines de primeira ordem implica em obter retas entre cada par de pontos. Uma spline de segunda ordem busca equações quadráticas entre cada par de pontos, com a garantia da existência de sua derivada de primeira ordem. A spline de terceira ordem ou cúbica produz um

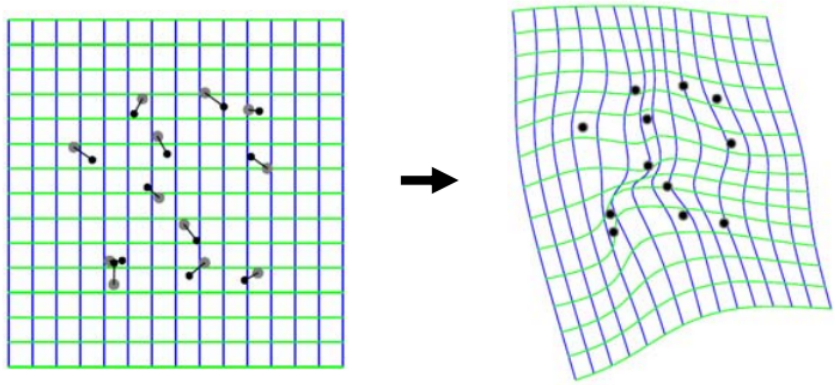

Figura 7 – Representação visual do processo de mapeamento entre superfícies. A movimentação de um ponto de controle faz com que a superfície se deforme mantendo os demais pontos de controle fixos, ao passo que tenta minimizar sua energia de deformação. Fonte: Sprengel et al, EMBS (1996).

polinômio diferençável em primeira e segunda ordem. Como representação intuitiva das splines cúbicas 1D, pode-se imaginar uma haste de metal fina, de movimento restringido nos pontos de entrada x_i.

O conceito das splines se aplica também a dados bidimensionais (EBERLY, 1996). As chamadas *Thin-Plate Spline*, ou TPS, surgiram com o intuito da interpolação 2D voltada para conjuntos de dados do tipo $(x_i, y_i, f(x_i, y_i))$, sendo inicialmente apresentada por Duchon (1977). Elas são uma generalização das splines cúbicas naturais. Nesse caso, a superfície da spline representa uma folha de metal fina de movimento restringido nos pontos de entrada (x_i, y_i). Como apontado por Júnior (2012), sua formulação garante restrições de que a superfície interpolante apresente mínima energia de deformação e que seja suave. Sua construção é baseada na escolha da função que minimiza a integral energia de distorção (do inglês, *bending energy*) da superfície. Uma representação visual do conceito de deformação está disponível na Figura 7.

A TPS pode ser entendida como o resultado de um problema de minimização (JU, 1996). Ela é obtida após a minimização de duas funções: (1) a distância entre pontos transformados e os pontos target desejados; (2) a energia de distorção geral do espaço. Podemos descrever esse problema de minimização definido sobre pontos amostrados como segue:

$$E = E_f + \lambda E_d \tag{2.22}$$

$$E_f = \sum_{i=1}^{m} \|f(x_i) - y_i\|^2 \tag{2.23}$$

$$E_d = \int_{\mathbb{R}^n} \|D^2 f\|^2 dX \tag{2.24}$$

onde

- E_f mede o quão distante está o ponto transformado do seu alvo, de forma linear;

- E_d mede o quão distorcido está o espaço. Caso a transformação seja afim (translação, rotação, escala), o valor dessa energia é zero.;
- λ controle o quanto de deformação não-rígida é permitida;
- $D^2 f$ é a matriz de derivadas parciais de segunda ordem de f;
- $\|D^2 f\|^2$ é a soma dos quadrados dos elementos da matriz.
- $dX = dx_1 \cdots dx_n$ é o elemento infinitesimal do hiper-volume, onde x_i são as componentes de x.

Um caso especial da TPS é o mapeamento suave $f : \mathbb{R}^3 \mapsto \mathbb{R}^3$ entre volumes 3D, onde $f(x_i, y_i, z_i) = (x_o, y_o, z_o)$. Nesse caso, é possível escolher uma série de m pontos de controle e m pontos alvo (do inglês *target*), definidos por $\{p_i, q_i\}$, com $i = 1 \cdots m$. No cenário descrito, a função de energia de deformação definida sobre os pontos amostrados resulta em:

$$E_d = \int_{\mathbb{R}^3} \Big[\Big(\frac{\partial^2 f}{\partial x^2}\Big)^2 + \Big(\frac{\partial^2 f}{\partial y^2}\Big)^2 + \Big(\frac{\partial^2 f}{\partial z^2}\Big)^2 + 2\Big(\frac{\partial^2 f}{\partial xy}\Big)^2 + 2\Big(\frac{\partial^2 f}{\partial xz}\Big)^2 + 2\Big(\frac{\partial^2 f}{\partial yz}\Big)^2 \Big] dxdydz \tag{2.25}$$

É provado (BOOKSTEIN, 1989) que a minimização da Equação 2.23 existe unicamente, e pode ser decomposta em uma componente afim e outra não-afim como segue:

$$f(p) = M \cdot p + \sum_{i=1}^{m} U(\|p - p_i\|)\ w_i \tag{2.26}$$

onde

- M é uma matriz de transformação afim;
- p é o ponto p em coordenadas homogêneas, da forma $(1, p_x, p_y, p_z)$;
- w é uma matriz dos coeficientes de deformação do espaço;
- $U(r)$ é o kernel radial da forma $U(r) = r^2\ \ln r$;
- $\sum_{i=1}^{m} U(\|p - p_i\|)\ w_i$ é a componente de deformação.

Para utilizar o mapeamento através da TPS, é preciso encontrar as matrizes M e w. Seja P a matriz $m \times n$ composta pelos pontos de controle em representação homogênea, onde m é o número de pontos de referência e $n = 4$ é a dimensão das coordenadas homogêneas. Seja Q a matriz dos pontos target em sua representação original. Como demonstrado em Bookstein (1989), é válida a relação $P^T w = 0$, resultando na seguinte relação matricial:

$$\begin{bmatrix} w_{m\times 3} \\ A_{4\times 3} \end{bmatrix} = \begin{bmatrix} K_{m\times m} & P_{m\times 4} \\ P^T_{4\times m} & 0_{4\times 4} \end{bmatrix}^{-1} \begin{bmatrix} Q_{m\times 3} \\ 0_{4\times 3} \end{bmatrix} \tag{2.27}$$

$$H = L^{-1} \cdot B \tag{2.28}$$

onde A é a matriz dos coeficientes da transformação afim, e os números subscritos indicam as dimensões das submatrizes envolvidas. Perceba que a forma apresentada é definida como um sistema de equações lineares, sendo H a matriz de variáveis representando os parâmetros do modelo TPS que se deseja encontrar, L a matriz de constantes conhecida e B a matriz de variáveis independentes conhecida.

A matriz de parâmetros H retornada pela TPS possuirá tanto os coeficientes da transformação de deformação quando os coeficientes da transformação rígida. Especificamente, ela terá o seguinte formato:

$$H = \begin{bmatrix} w_1^x & w_1^y & w_1^z \\ & \vdots & \\ w_m^x & w_m^y & w_m^z \\ a_1^x & a_1^y & a_1^z \\ a_x^x & a_x^y & a_x^z \\ a_y^x & a_y^y & a_y^z \\ a_z^x & a_z^y & a_z^z \end{bmatrix} \tag{2.29}$$

onde o supraescrito indica o eixo sobre o qual se está computado os dados, definido pelo número da coluna. As primeiras m linhas são responsáveis pelos coeficientes de deformação, enquanto as últimas 4 linhas indicam os coeficientes da transformação afim.

De posse da matriz H, podemos mapear pontos utilizando o modelo definido. Em específico, o mapeamento dependerá dos pontos alvo e de controle escolhidos, bem como a matriz de parâmetros e as coordenadas iniciais do ponto $p = (p_x, p_y, p_z)$ desejado. Seja $p' = (p'_x, p'_y, p'_z)$ o ponto transformado, uma forma simples de definir esse mapeamento é calcular cada coordenada em separado, como segue:

$$\begin{aligned} p'_x &= a_1^x + a_x^x p_x + a_y^x p_y + a_z^x p_z + \sum_{i=1}^{m} w_i^x U(p, p_i) \\ p'_y &= a_1^y + a_x^y p_x + a_y^y p_y + a_z^y p_z + \sum_{i=1}^{m} w_i^y U(p, p_i) \\ p'_z &= a_1^z + a_x^z p_x + a_y^z p_y + a_z^z p_z + \sum_{i=1}^{m} w_i^z U(p, p_i) \end{aligned} \tag{2.30}$$

2.2.7 GLM

A biblioteca OpenGL Mathematics (GLM)[7] fornece ferramentas práticas para trabalhar com vetores no espaço tridimensional de forma nativa na linguagem em C++. Para utilizar suas funções, basta importar os cabeçalhos prontos desejados, dispensando quaisquer tipos de instalação. Na GLM, há suporte para operações com vetores (1x3), vetores na forma homogênea (1x4), matrizes de rotação (3x3) e matrizes de transformação (4x4) ou frames.

A GLM é baseada nas especificações da OpenGL Shading Language (GLSL)[8], provendo classes e funções desenvolvidas com a mesma nomenclatura. Entretanto, o projeto vai além das funções presentes na GLSL, provendo adicionais como quatérnions, encapsulamento de dados, números aleatórios, ruído, dentre outros.

2.3 Robótica

Esta seção discorre sobre as bases teóricas necessárias para a descrição matemática formal do braço robótico Denso VP6242, bem como do efetuador Robotiq 2F-85, ambos utilizados no presente trabalho. Ao final do capítulo será discutido sobre o Matlab e sobre o ROS.

2.3.1 Elos, Juntas e Manipuladores

A cinemática estuda o movimento dos corpos sem se preocupar com a sua causa, ou com as forças envolvidas. Na robótica, uma das formas de se descrever um robô é através de uma cadeia cinemática equivalente, ou seja, através da descrição das partes móveis do robô no espaço 3D (SICILIANO; KHATIB, 2016). Vimos na Seção 2.2.4 que um corpo rígido pode ser completamente descrito no espaço através de um frame (ou sistema de coordenadas) atrelado ao mesmo. Portanto, um robô pode ser descrito matematicamente através de uma sequência de frames sucessivos.

Numa cadeia cinemática existem elos (do inglês, links), que são os corpos rígidos, e as juntas, que realizam a ligação física entre elos. Um corpo rígido é um modelo matemático que assume uma poção limitada do espaço e é indeformável. Na mundo real, todos os corpos são não-rígidos, ou seja, se deformam mediante aplicação de uma determinada força que varia a depender do tipo do material considerado.

Uma junta permite que um elo se mova relativamente ao seu elo pai (elo anterior). Há 2 tipos de juntas primárias: prismáticas e rotacionais. As rotacionais, ou de revolução, permitem a rotação relativa entre elos sobre um determinado eixo. As prismáticas ou lineares, permitem o deslocamento linear entre elos sobre um determinado eixo. A representação das juntas prismáticas e rotacionais é mostrada na Figura 8. Outros tipos podem ser derivados das juntas primárias,

[7] <https://github.com/g-truc/glm>
[8] <https://www.opengl.org/registry/doc/GLSLangSpec.4.50.diff.pdf>

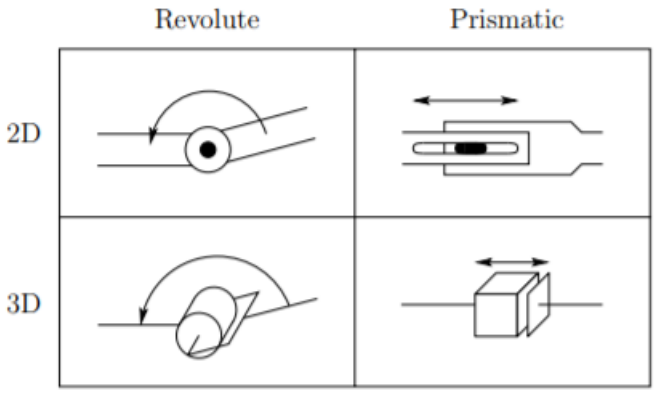

Figura 8 – Representação simbólica das juntos de revolução e prismática. Fonte:(SPONG et al., 2006), p.4

como as universais (2 rotacionais), as esféricas (3 rotacionais), as cilíndricas (1 rotacional e 1 linear) e as planares (2 rotacionais).

Qualquer junta representa sempre uma ligação entre dois links. É possível descrever robôs em função do tipo de movimento de suas juntas. Um robô com 3 juntas rotacionais é comumente chamado RRR, ou 3R. Um robô com duas juntas rotacionais e uma prismática é chamado RRL.

Um manipulador robótico é um uma cadeia de elos e juntas atrelada a uma base (que pode ser fixa ou móvel) em um dos extremos, e a um efetuador robótico no outro (SICILIANO; KHATIB, 2016). O efetuador robótico é o responsável por interações com objetos do ambiente.

Quando nós temos os valores atuais de cada junta, e queremos saber a pose final do efetuador, dizemos que estamos realizando uma cinemática direta (do inglês, *forward kinematics*, ou FK). No caso oposto, quando nós temos a pose desejada do efetuador e queremos saber os valores necessários das juntas, estamos realizando uma cinemática inversa (do inglês, inverse kinematics, ou FK).

Um objeto é dito ter n graus de liberdade (degrees-of-freedom, ou DOF) se sua configuração pode ser minimamente especificada por n parâmetros. Um objeto no espaço 3D possui 6 graus de liberdade, pois sua descrição mínima assume 3 valores de posição e três valores de orientação para ser completamente descrito.

No caso de um manipulador robótico, seu DOF é igual ao número de juntas básicas não-redundantes presente em sua cadeia cinemática. Dessa forma, para que um manipulador possa mover e rotacionar livremente um objeto no espaço, em qualquer eixo, é necessário que ele possua no mínimo 6 DOF independentes.

A maior parte dos manipuladores robóticos de hoje possuem 6 DOF ou menos (SPONG et al., 2006). Eles geralmente são classificados com base nas suas 3 primeiras juntas, fazendo referência ao "braço"do robô. As demais juntas são a representação do "pulso e mão", sendo

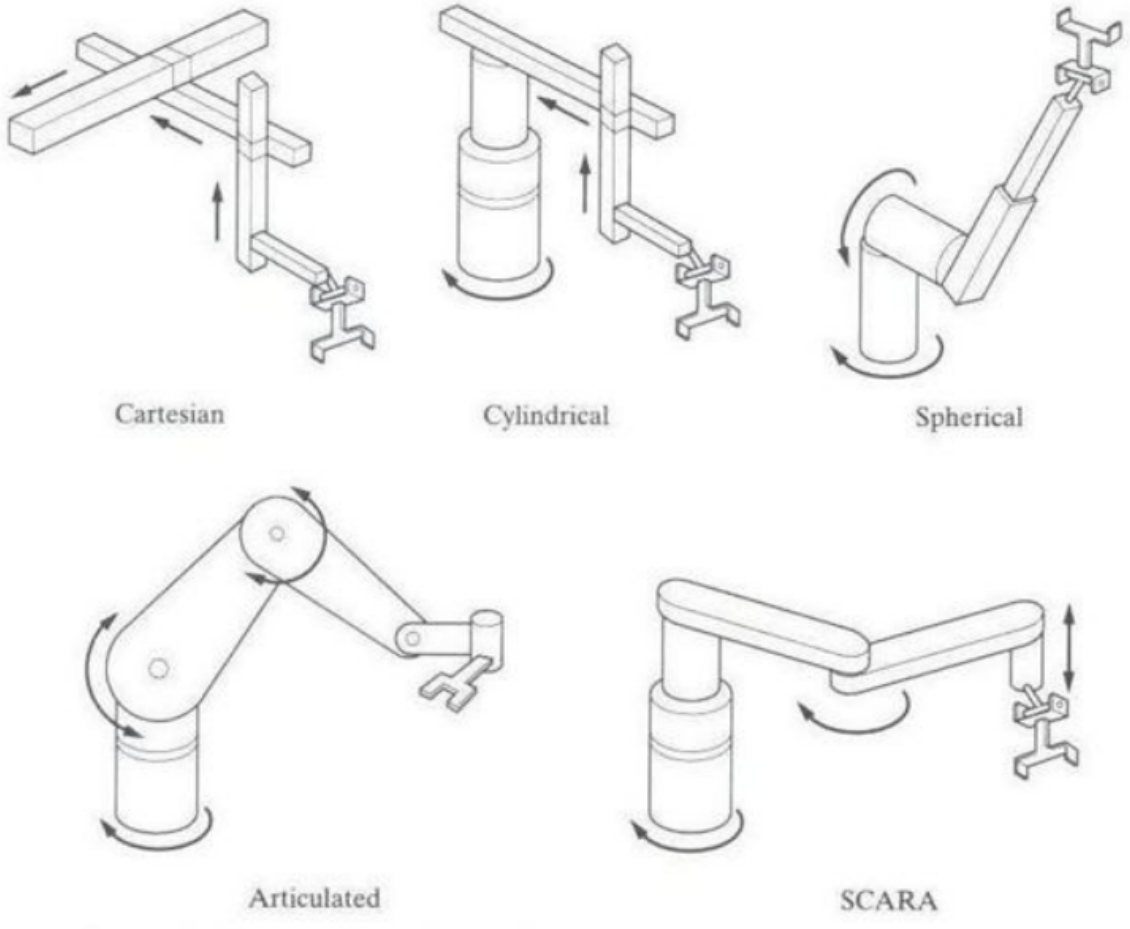

Figura 9 – Nomenclatura das configurações mais comuns de robôs presentes na literatura. Fonte: (EVANS, 2014)

descritas separadamente.

As configurações de manipulador mais comuns pertencem a um dos 5 grupos seguintes: Cartesiano (PPP), cilíndrico (RPP), esférico (RRP), articulado (RRR) e SCARA (RRP). Tais configurações são visualizadas na Figura 9.

2.3.2 Modelagem de Robôs e Cinemática Direta

Seja o frame b composto pelos eixos x_b, y_b, z_b e atrelado à origem da base do manipulador, a qual assumimos ser fixa. A esse frame damos o nome de frame da base, ou frame $\{0\}$. Cada elo $i = \{1 \cdots n\}$ consecutivo será atrelado um novo frame $i = (x_i, y_i, z_i)$ que se moverá por ação da junta i. O frame $\{e\}$, atrelado ao efetuador, é escolhido convenientemente de acordo com sua geometria em particular. Se o efetuador é uma garra (como a do presente trabalho), a origem do efetuador é localizada no centro da garra, como mostrado na Figura 10. A escolha dos frames é realizada da seguinte forma:

- o vetor unitário a_e é definido na direção de aproximação do objeto;

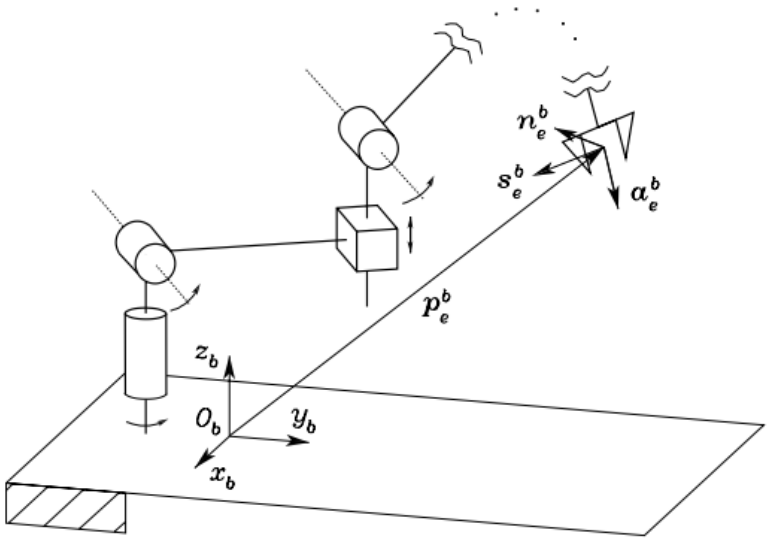

Figura 10 – Demonstração do processo de cinemática direta. Fonte: (SICILIANO; KHATIB, 2016), pg 59.

- o vetor unitário s_e é definido ortogonal a a_e e na direção do plano deslizante dos dedos das garra;
- e o vetor unitário n_e é definido como ortogonal aos outros dois de forma que a tupla $\{n_e, s_e, a_e\}$ siga a regra da mão direita.

A função da cinemática direta é responsável por definir o frame do efetuador levando em consideração os valores das juntas q, bem como o posicionamento relativo inicial entre elas. Seguindo o exemplo da Figura 10, a cinemática direta pode ser descrita formalmente como a seguinte transformação homogênea:

$$
{}^{b}T_e(q) = \begin{bmatrix} n_e^b(q) & n_s^b(q) & n_a^b(q) & p_e^b(q) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.31}
$$

$$
= {}^{b}A_1(q1)\ {}^{1}A_2(q2)\ \cdots {}^{n-1}A_n(q_n)\ {}^{n}A_e \tag{2.32}
$$

2.3.3 Convenção Denavit-Hartenberg

Para que possamos computar recursivamente a Equação 2.32, faz-se necessário um método sistemático para para definir as posições e orientações relativas entre dois elos consecutivos. Em outras palavras, faz-se necessário definir uma convenção de acoplamento de frames aos elos. A convenção Denavit-Hartenberg (DH) é uma delas.

Seja o eixo i o eixo da junta que conecta o elo $i-1$ ao elo i. De acordo com Siciliano e Khatib (2016), a convenção DH define o frame i da seguinte forma:

- Escolha o eixo z_i ao longo do eixo da junta $i+1$

- Localize a origem O_i na interseção do eixo z_i com a normal comum aos eixos z_i e z_{i-1}. Também localize O_{i-1} na interseção entre z_{i-1} e a normal comum.
- Escolha o eixo x_i ao longo da normal comum entre os eixos z_{i-1} e z_i, com direção da junta i à junta $i+1$.
- Escolha o exiso y_i obedecendo à regra da mão direita.

Tal representação é ilustrada na Figura 11. É importante frisar que a convenção DH permite diferentes soluções de acoplamento de frames em casos específicos. Nesses casos, é possível utilizar das indeterminações arbitrariamente para simplificar o modelo. Há indeterminação de frames nos seguintes casos:

- Para o frame $\{0\}$, apenas a direção do eixo z_0 é especificada. Assim, O_0 e x_0 podem ser escolhidos arbitrariamente;
- Para o último frame n, uma vez que não há junta $n+1$, z_n não é unicamente definido enquanto x_n precisa ser normal ao eixo z_{n-1}. Geralmente, a junta n é de revolução, e então o eixo z_n deve ser alinhado ao eixo z_{n-1}.
- Quando dois eixos consecutivos são paralelos, a normal comum entre eles não é unicamente definida.
- Quando dois eixos consecutivos se interceptam, a definição de x_i é arbitrária.
- Quando a junta i é prismática, a direção de z_{i-1} é arbitrária.

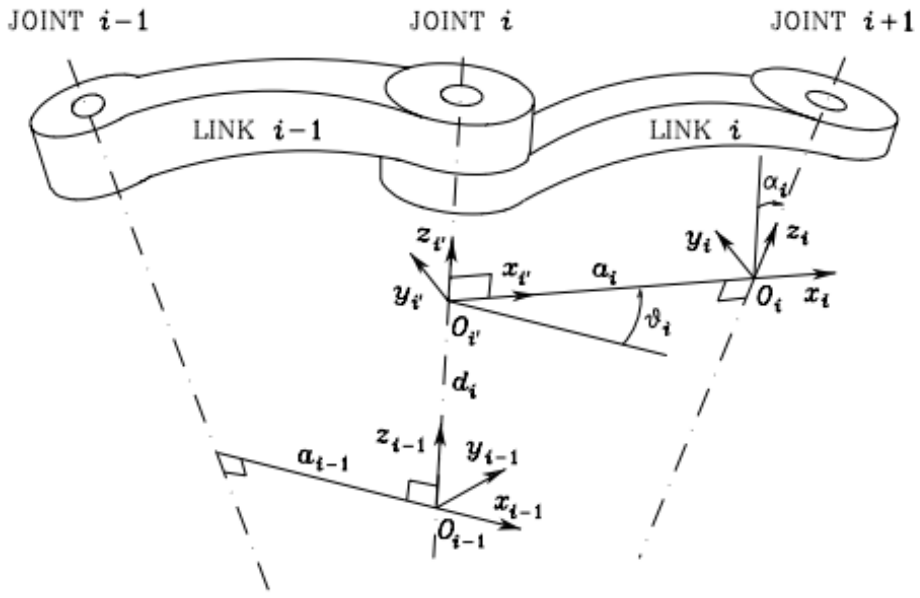

Figura 11 – Representação visual da Convenção DH.

Após todos os frames haverem sidos propriamente acoplados, a posição e a orientação do frame i com relação ao frame $i-1$ são completamente especificadas por 4 parâmetros:

- a_i (comprimento de elo) é a distância entre O_i e $O_{i'}$;
- d_i (offset de junta) é a coordenada de $O_{i'}$ sobre z_{i-1}.
- α_i (torção de elo) é o ângulo entre os eixos z_{i-1} e z_i em torno do eixo x_i, positivo no sentido anti-horário;
- θ_i (ângulo de junta) é o ângulo entre os eixos x_{i-1} e x_i em torno do eixo z_{i-1}, positivo no sentido anti-horário.

Os parâmetros a_i e α_i são sempre constantes, dependendo apenas da geometria de conexões entre juntas consecutivas que são estabelecidas através do elo i. Quanto aos dois parâmetros restantes, apenas um será variável a depender do tipo de junta. Em juntas rotacionais, θ é variável. Em juntas prismáticas, d_i é constante.

Dessa forma, é possível expressar as transformações entre os frames i e $i-1$ de acordo com os passos seguintes:

1. Suponha um frame móvel alinhado com o frame $i-1$
2. Desloque o frame móvel de uma medida d_i sobre o eixo z_{i-1} e rotacione este frame de θ graus em torno do eixo z_{i-1}; esta sequência alinha o frame móvel com o frame i', sendo descrito através da seguinte transformação homogênea:

$$A_i'^{i-1} = \begin{pmatrix} c\theta & -s\theta & 0 & 0 \\ s\theta & c\theta & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.33}$$

3. Translade o frame móvel alinhado com i' de um valor a_i sobre o eixo x_i' e rotacione por α_i sobre o eixo x_i'; Esta sequência alinha o frame móvel com o frame i, e é descrito através da seguinte transformação homogênea:

$$A_i'^{i} = \begin{pmatrix} 1 & 0 & 0 & a_i \\ 0 & c\alpha & -s\alpha & 0 \\ 0 & s\alpha & c\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.34}$$

A transformação resultante entre os frames $i-1$ e i pode ser resumida numa única transformação equivalente da forma:

$$A_i^{i-1} = A_i'^{i-1}\, A_i'^{i} = \begin{pmatrix} c\theta & -s\theta_i c\alpha_i & s\theta_i s\alpha_i & a_i c\theta_i \\ s\theta_i & c\theta_i c\alpha_i & -c\theta_i s\alpha_i & a_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.35}$$

Note que a transformação A_{i-1}^{i}, embora se constitua numa matriz $4x4$, é função de apenas 4 parâmetros (com apenas 1 parâmetro variável). Dessa forma, uma descrição mínima de uma cadeia cinemática pode é constituída por uma tabela contendo n linhas (uma para cada elo) e 4 colunas (uma para cada parâmetro).

Perceba também que, segundo a convenção DH, a junta i se localiza na região proximal do elo i. Já o frame i se localiza na região distal do elo i. Dessa forma, quando a junta i varia o seu valor θ_i, ambos o elo i e o frame i se movem juntos.

2.3.4 Sobre o manipulador VP6242

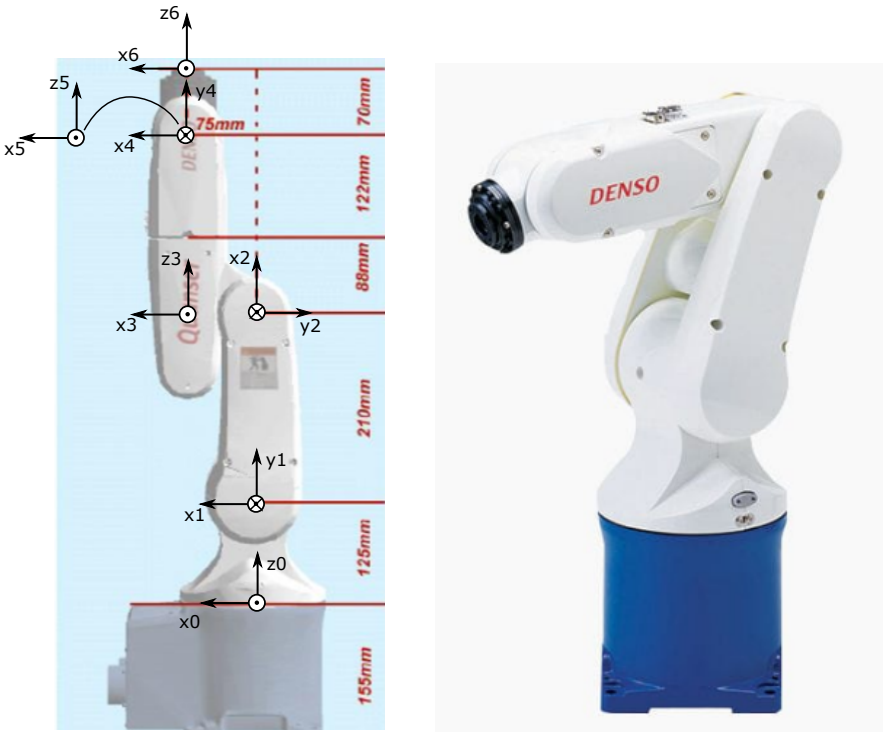

Figura 12 – À esquerda, vemos o acoplamento dos frames do segundo os parâmetros DH fornecidos pelo fabricante. Fonte: Imagem adaptada de Denso.

O modelo de manipulador robótico utilizado no presente trabalho é o Denso VP6242G (DENSO, 2021), um robô de 6 DOF com todas as suas juntas rotacionais, e que segue a configuração articulada ou antropomórfica. Os controladores de cada motor operam a uma taxa

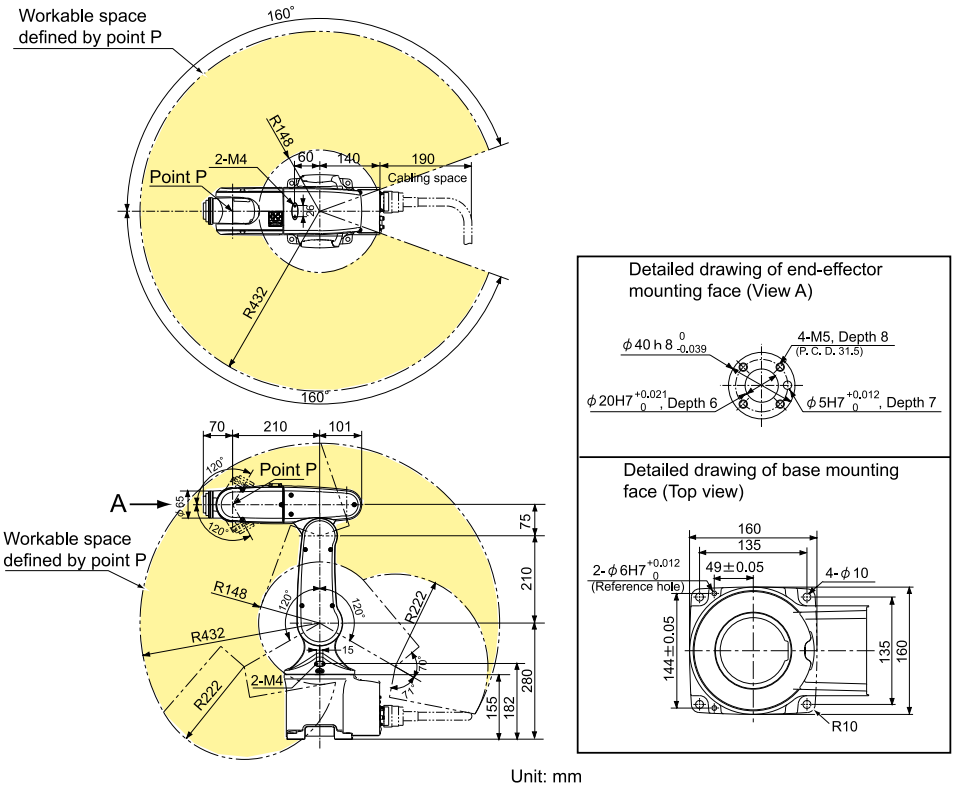

Figura 13 – Dimensões do robô e forma de seu espaço de trabalho. Fonte: Denso.

de 4 KHz, possuindo um *encoder* de posição por motor. O peso total do presente manipulador é de aproximadamente 15 Kg. Seu alcance máximo é de 432 mm para o ponto P localizado no frame acoplado ao fim do último elo. Sua estrutura junto às suas dimensões são mostradas na Figura 13.

O prefixo VP indica a linha de manipuladores robóticos considerados de pequena escala. O código 6242 abriga informações sobre a capacidade de peso e alcance do robô:

- 6 : número de eixos do robô (DOF);
- 2 : capacidade de carga padrão de 2 kg. A capacidade máxima é 2,5 kg;
- 42 : comprimento total do braço robótico de 420 mm.

Os parâmetros DH fornecidos pelo fabricante do Denso estão disponíveis na Tabela 4.

O acoplamento dos frames pode ser visualizado na Figura 12. Os valores mínimos e máximo de cada uma das 6 juntas estão disponíveis na Tabela 5.

A comunicação com o braço robótico é realizada pelo seu controlador através da comunicação TCP com o controlador da Quanser. Ao energizar o controlador, um processo de *boot* é iniciado, seguido de um processo de *self-test*. Durante esse tempo, a luz do controlador ficará acesa o tempo inteiro. Ao finalizar o processo de inicialização e testes, a luz passará para o estado intermitente, indicação de que o controlador está pronto para conexão.

Elo i	a_i	α_i	d_i	θ_i
1	0	$\pi/2$	0.125	θ_1
2	0.210	0	0	$\theta_2 + \pi/2$
3	-0.075	$-\pi/2$	0	$\theta_3 - \pi/2$
4	0	$\pi/2$	210	θ_4
5	0	$-\pi/2$	0	θ_5
6	0	0	0.07	θ_6

Tabela 4 – Modelagem do robô VP6242 seguindo a convenção DH original (DENSO, 2021).

Junta	Liberdade (graus)	Razão de Engrenagem do Motor	Calibração do Encoder (count/deg)	Constante de Torque (N.m/ amp)
J1	±160	120:1	43690.666670	0.38
J2	±120	160:1	58254.222220	0.38
J3	+160, +19	120:1	43690.666670	0.22
J4	±160	100:1	36408.888890	0.21
J5	±120	100:1	36408.888890	0.21
J6	±360	100:1	36408.888890	0.21

Tabela 5 – Valores referentes às juntas do manipulador Denso VP6242G.

2.3.5 Sobre o efetuador Robotiq 2F-85

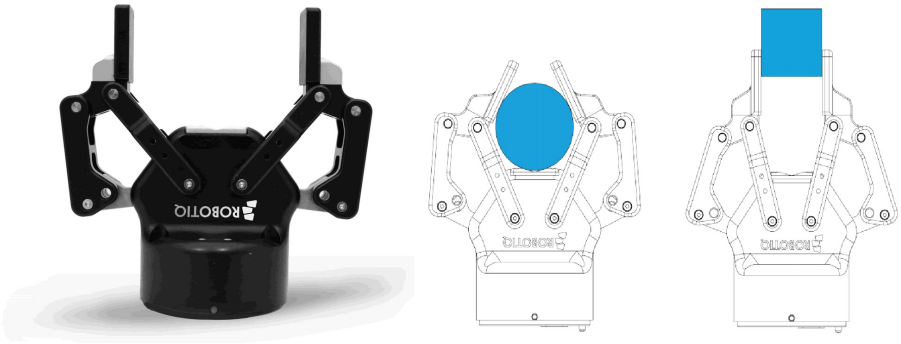

Figura 14 – Ilustração do modelo Robotiq 2F-85 e suas posturas envoltória (ao centro) e paralela (à direita). Fonte: Robotiq 2f-85 Manual, pg 10.

O efetuador utilizado junto ao manipulador robótico é fabricado pela empresa Robotiq, com o modelo 2F-85 (ROBOTIQ, 2021). A nomenclatura informa que há dois dedos no manipulador (do inglês 2 *fingers*, ou 2F) com uma abertura máxima de 85 mm entre os dedos. É possível visualizar o modelo na Figura 14. Suas dimensões são especificadas na Figura 15.

A força exercida pelo efetuador varia entre 20 e 235 N. Com uma massa total de 1 kg, o efetuador pode exercer uma força de levante de um objeto com até 5 kg. A posição dos dedos possui uma resolução de 0.4 mm, enquanto sua velocidade de abertura e fechamento é de 150 mm/s. Vale ressaltar que a posição, a velocidade e a força aplicada pelo efetuador são configuráveis. Além disso, há rotinas de inicialização e auto-calibração que precisam ser realizadas uma vez após sua energização.

O efetuador robótico 2F-85 tem um único atuador, cujo movimento é transmitido mecanicamente para ambos os dedos. Ele é classificado como garra adaptativa, uma vez que os dedos dispostos em uma cadeia cinemática fechada se adaptam mecanicamente à forma do objeto manipulado. Dependendo da forma do objeto e da distância do mesmo à garra, os dedos podem assumir diferentes posturas.

Para assumir a postura envoltória, é necessário que os dedos pressionem os objetos com sua região proximal (parte mais interna). Para assumir a postura paralela, é necessário que os dedos pressionem os objetos com sua região distal (mais afastadas). Tais configurações são demonstradas na Figura 14.

O controle do efetuador Robotiq 2F-85 é, também, através da comunicação TCP, nesse caso entre o Simulink e o *Universal Controller* da Robotiq (ROBOTIQ, 2021). É possível ver a montagem do robô (manipulador e efetuador) e seus controladores na Figura 16.

2.3.6 Matlab

O Matlab (MATLAB, 2021) é uma plataforma de programação e computação numérica multi-funções que oferece soluções para facilitar tarefas como modelagem, controle e análise de dados. Através dele, é possível utilizar tanto a programação textual convencional, quanto a programação em blocos e simulação dos dados, através da ferramenta Simulink (SIMULINK, 2021).

A ferramenta Simulink suporta o uso das chamadas *toolboxes*, que são ferramentas auto-contidas, geralmente com interface gráfica de usuário. Por exemplo, há *toolboxes* na área de Controle, Física, Aprendizagem de Máquina e Robótica. Um exemplo é o *Robotic Toolbox* (CORKE, 2021), *toolbox* que foi gerado para dar suporte às atividades práticas discutidas no

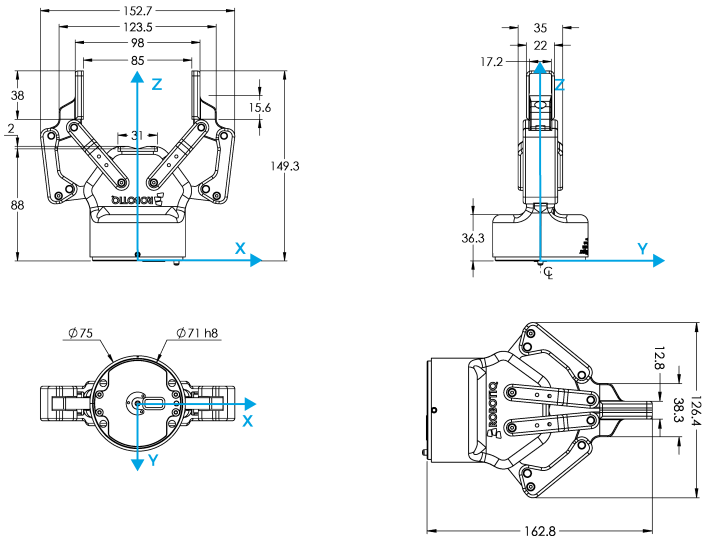

Figura 15 – Dimensões dos elos presentes no manipulador Robotiq 2f-85. Fonte: Manual Robotiq 2f-85, pgs 87 e 88.

Figura 16 – Ilustração da montagem completa do Denso VP6242 junto ao Robotiq 2F-85, incluindo uma visão detalhada de seus respectivos controladores. Fonte: Adaptada de Denso Open Architecture User Manual e Robotiq Universal Controller User Manual.

livro do mesmo autor, denominado *Robotics, Vision and Control* (CORKE, 2017).

No presente trabalho, o Simulink é utilizado para realizar a tarefa de comunicação e controle de ambos os dispositivos robóticos: manipulador e efetuador. Em nosso laboratório de robótica, há um computador exclusivo para a comunicação com os controladores, realizada através de uma conexão TCP direta exclusiva para cada controlador.

Com relação ao *toolbox* do manipulador robótico, a fabricante oferece os blocos de controle que fazem uso da biblioteca Quarc (QUANSER, 2021). Essa biblioteca possui funções para o controle em tempo real de dispositivos de hardware externos, incluindo funcionalidades como compilação, envio, início e interrupção de códigos com tempo real, além do gerenciamento desses códigos no controlador remoto. Também há diretrizes para leitura de sensores externos.

Através dos blocos de controle do VP6242, é possível visualizar os valores de posição e corrente elétrica nas juntas do manipulador, seus limites de máximo e mínimo setados por software (*soft limits*), bem como o os parâmetros do controle PID das mesmas.

Já o toolbox responsável pelo controle do efetuador robótico conta com a rotina de inicialização, bem como o controle do modo de operação atual. Há quatro modos de controle oferecidos pela ferramenta, associados aos índices de 1 a 4. Tais índices se constituem em:

1. Modo de inicialização e calibração inicial. Deve ser o modo atual na primeira comunicação do Simulink com o controlador do efetuador (do contrário, a garra não responderá aos demais modos). Durante esse modo, a garra executará a rotina de *self-test*, através do

completo fechamento seguido da completa abertura;

2. Modo de abertura com valor pré-definido. Ignora o valor de junta presente na entrada;
3. Modo de fechamento com valor pré-definido. Ignora o valor de junta presente na entrada;
4. Modo de abertura direta de acordo com o valor de junta.

É importante ressaltar que o programa de controle do manipulador permite a inter-comunicação com outros sistemas utilizando um servidor TCP. Tal funcionalidade foi utilizada para realizar a comunicação desacoplada com módulos do *framework* ROS, que será discutido na seção a seguir.

2.3.7 ROS

O Sistema Operacional para Robôs (do inglês Robot Operating System, ou ROS) é um *framework* flexível de aplicações robóticas que segue uma arquitetura baseada em *publishers* e *subscribers* na qual nós conversam entre entre si através de tópicos (ROS, 2021). Dessa forma, é possível criar arquiteturas distribuídas escaláveis, incluindo a comunicação entre nós rodando em diferentes computadores através da rede TCP. O fato do ROS possuir código aberto permite que pessoas de todo o mundo possam ajudar na solução de possíveis problemas, promovendo a ideia do desenvolvimento colaborativo.

O framework começou a ser desenvolvido em 2007, quando dois alunos da Universidade de Stanford, Eric Berger e Keenan Wyrobek, tiveram a ideia de desenvolver o que denominaram *linux dos robôs* (Fonte: TheConstructSim[9]). A motivação para o seu desenvolvimento foi a observação de que os estudantes precisavam refazer toda a arquitetura de comunicação (tanto

[9] <https://www.theconstructsim.com/history-ros/>

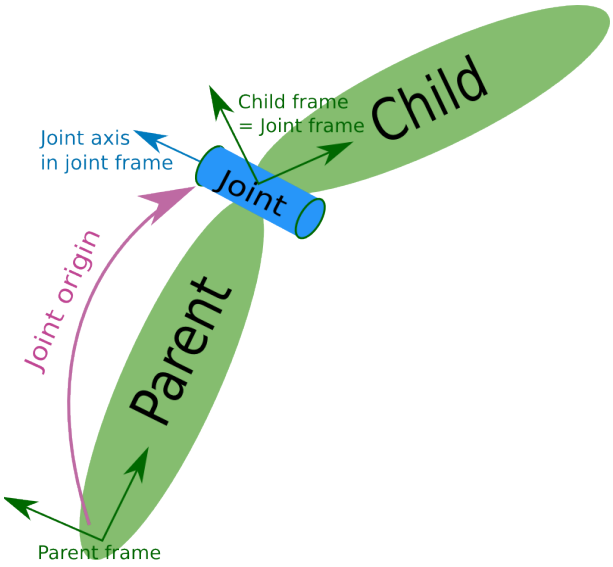

Figura 17 – Representação de robôs segundo o formato URDF. Fonte: ROS Wiki.

entre módulos, quanto com sensores/atuadores) quando iniciavam um novo projeto. Esse processo poderia demorar mais tempo que a resolução do problema principal. Foram então contratados pela empresa Willow Garage, cujo chefe Scott Hassa compartilhava da mesma visão. Nesse cenário, o ROS surge em sua primeira versão em 2008 como o framework de comunicação principal utilizado no robô Personal Robot 2 (ou PR2) da empresa Willow Garage. Hoje em dia, o software é mantido pela Open Source Robotics Foundation (OSRF, 2021).

O ROS utiliza o URDF (Universal Robot Description Format) como forma de descrição padrão de robôs. Seguindo a notação XML, é possível descrever as características visuais, inerciais e de colisão dos elos, além de limites e constantes físicas das juntas. Sua forma de representação é diferente da convenção DH estudada na Seção 2.3.3.

Com o URDF, é possível especificar o eixo de movimento para uma determinada junta. Além disso, o frame do elo i é o mesmo frame da junta i e está localizado na região proximal do elo. Por outro lado, na convenção DH, o eixo de movimento da junta é sempre o eixo Z, além do frame do elo i se localizar sobre a junta $i + 1$. A representação dos robôs no formato URDF está disponível na Figura 17.

Além do URDF, o ROS também suporta o formato XACRO[10], que permite a adição de propriedades, expressões matemáticas, macros, blocos condicionais, além do uso de múltiplas instâncias do URDF num arquivo único. O intuito é utilizar o reuso de código para criar descrições robóticas mais sucintas e de melhor legibilidade.

A simulador físico padrão do ROS é o Gazebo (GAZEBO, 2021), também mantido pela OSRF. Ele possui suporte a diferentes motores físicos, como ODE, Bullet, Simbody e DART. Sua interface gráfica de usuário permite a interação de forma prática com robôs e objetos da simulação. É possível simular dados de sensores, opcionalmente com ruídos, com suporte a câmeras 2D/3D, sensores de distância a laser, câmeras de profundidade IR como o Kinect, sensores de contato, sensores de torque, dentre outros. Seu motor gráfico é o OGRE.

Uma das ferramentas disponibilizadas pelo framework ROS é a biblioteca rosbag[11]. Sua função é oferecer ao usuário a possibilidade gravar tópicos selecionados para poderem ser reproduzidos novamente num momento futuro. A forma como foi desenvolvida a biblioteca permite que o processo ocorra em alta performance além de evitar serialização e desserialização desnecessária das mensagens. Também é disponibilizada uma interface de linha de comando (do inglês *Command Line Interface*, ou CLI) para a execução da ferramenta. Utilizamos o rosbag para gravação testes envolvendo o mapeamento de posição.

A versão do ROS utilizada no presente trabalho foi o ROS 1 Melodic[12] para Windows disponibilizada pelo gerenciador de pacotes Chocolatey[13]. Todo o processo de instalação da

[10] <http://wiki.ros.org/xacro>
[11] <http://wiki.ros.org/rosbag>
[12] ROS 1 Melodic build 20201012.0.0.2010230001, instalação em <http://wiki.ros.org/Installation/Windows>
[13] <https://chocolatey.org/>

versão nativa para Windows está descrita na própria página da Open Robotics.

2.4 Inteligência Artificial

Esta seção discorre sobre as bases teóricas necessárias para a compreensão e aplicação dos modelos neurais utilizados no trabalho. Serão discutidas as variantes de Rede Neural utilizadas, bem como o modelo de mapeamento baseado em TPS.

2.4.1 Aprendizagem de Máquina

De acordo com Chollet (2017), o campo da Inteligência Artificial (IA) pode ser definido como "o esforço de automatizar tarefas intelectuais normalmente executadas por humanos". Dessa forma, o campo da IA engloba os campos da Aprendizagem de Máquina e da Aprendizagem Profunda, mas também inclui outros campos que não envolvem qualquer tipo de aprendizado. Por um longo tempo, estudiosos acreditavam que a inteligência artificial poderia ser atingida através da programação manual de um conjunto suficientemente grande de regras explícitas de manipulação do conhecimento, abordagem conhecida como IA simbólica. Entretanto, tal abordagem se mostrava inviável para tarefas complexas como classificação de imagem, reconhecimento de voz e tradução de idiomas. Nesse cenário, surge uma nova abordagem para tomar o lugar da IA simbólica: a aprendizagem de máquina.

O conceito de Aprendizagem de Máquina (do inglês *Machine Learning*, ou ML) foi utilizado pela primeira vez por Alan Turing em 1850 em seu paper seminal *Computing Machinery and Intelligence* que, dentre outros fatos, introduziu o Teste de Turing. Ao discorrer sobre a Máquina Analítica inventada por Ada Lovelace e Charles Babbage nos anos de 1840, Turing apresentou a seguinte questão: poderia, um computador, aprender por si próprio como executar uma determinada tarefa? Ou ainda: ao invés de programadores informarem regras de processamento de dados manualmente, é possível que um computador aprenda tais regras apenas com a experiência? A resposta era que sim, era possível, e a máquina inventada por Lovelace e Babbage era capaz de realizar tal feito.

Surgia então o conceito de a aprendizagem de máquina, onde humanos fornecem dados de entrada junto às respostas esperadas dos mesmos, enquanto o computador aprende as regras de associação/inferência por si próprio. Diz-se então que um sistema baseado em ML é treinado, ao invés de explicitamente programado. Embora o campo tenha se desenvolvido mais nos anos de 1990 (CHOLLET, 2017), ele rapidamente se tornou o mais mais popular bem sucedido da IA, tendência liderada pelo surgimento de hardwares mais potentes e mais rápidos, além de datasets substancialmente maiores.

2.4.2 Redes Neurais Artificiais como Classificadores

O sistema nervoso é um sistema de processamento de informações altamente complexo, não-linear e paralelo (HAYKIN; NETWORK, 2004). Ele tem a capacidade de adaptar suas unidades constituintes de forma a otimizar diferentes computações (como percepção, reconhecimento e atividade motora). Tal organização é possibilitada pela propriedade conhecida como plasticidade cerebral, que permite ao cérebro aprender as regras do ambiente em que vive através da experiência. Uma Rede Neural Artificial (do inglês *Artificial Neural Network*, ou ANN) é uma forma de aprendizagem de máquina supervisionada baseada na representação matemática altamente simplificada da arquitetura presente no cérebro humano. Por supervisionada, entende-se que é fornecido ao modelo o conjunto de dados de entrada junto à saída esperada.

O cérebro humano pode ser descrito como a conexão de inúmeras unidades computacionais simples, denominadas neurônios (AGGARWAL et al., 2018). Os neurônios se conectam através dos axônios (responsáveis por enviar pulsos elétricos) e dos dendritos (responsáveis por receber esses pulsos de outros neurônios). A região de conexão entre tais estruturas é conhecida como sinapse. A força das conexões sinápticas pode mudar, enfraquecendo ou fortalecendo, a depender do estímulo externo. Essa mudança nas conexões sinápticas é o processo pelo qual organismos vivos aprendem.

Uma ANN em sua forma mais simples é constituída por um único neurônio artificial, sendo inicialmente proposto como o *perceptron* de Rosenblatt (ROSENBLATT, 1958). Um perceptron, representado na Figura 18, é uma estrutura que recebe um vetor de entradas $\vec{x} = x_1, ..., x_m$ e realiza seu produto interno com o vetor de pesos sinápticos $\vec{w} = w_1, ..., w_m$, produzindo a saída $z = \vec{x} \cdot \vec{w}$. Em determinados casos, pode ser desejado incorporar como parcela extra do produto interno o termo *bias*, uma constante b com peso sináptico $w_b = 1$. Após devidamente calculado, o produto interno é injetado numa função de ativação f, produzindo a saída $y = f(z)$. No perceptron original, a função de ativação utilizada foi a Heaviside ou função degrau, definida por $f \mapsto \{0, 1\}$, apresentando saída nula para entradas negativas, e saída $f(x) = 1$ caso contrário. Há diversas possibilidades para a função de ativação, que é responsável pela não-linearidade inerente aos perceptrons.

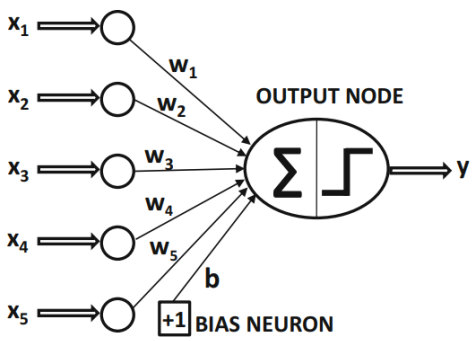

Figura 18 – Fonte: Representação visual do perceptron, a unidade constituinte das Redes Neurais Artificiais. Fonte: (AGGARWAL et al., 2018), pg 19

As ANNs básicas são formadas por camadas sucessivas de neurônios, onde cada neurônio da camada atual se conecta a todos os neurônios da camada seguinte. Essa configuração é conhecida como camadas densas, ou totalmente conectadas (do inglês, *fully-connected*). Cada camada pode variar em quesitos como tamanho da camada (número de neurônios por camada), função de ativação, dentre outros. Durante o processo de treinamento, o vetor de pesos w é atualizado automaticamente durante o aprendizado da rede. A todos os demais parâmetros que são setados antes do treinamento e se mantém fixo durante o aprendizado, damos o nome de hiper-parâmetros.

Para que se possa otimizar uma ANN, é necessário que haja uma métrica de distância entre a sua saída atual e a saída esperada. Essa métrica é retornada pela função de perda da rede, também conhecida como função objetiva. Seja $\hat{y}$ a saída estimada pelo modelo, e seja y a saída real esperada para um determinado exemplo. A função de perda $L(y, \hat{y})$ computa o quão bem se comportou a rede neural para esse exemplo específico. É possível, entretanto, estimar a perda média sobre um conjunto de dados de treinamento. A ideia fundamental das ANNs é usar a métrica produzida pela função de perda como um *feedback*, ajustando o valor dos pesos das camadas na direção que diminua a distância entre saída atual e a esperada. Esse ajuste é papel do otimizador.

Um problema bastante discutido ao se utilizar ANNs é o sobre-ajuste (do inglês, *over-fitting*). Ele se dá quando o modelo aprende a reproduzir a saída esperada sobre o conjunto de treinamento, mas não possui generalização o suficiente para se comportar bem com outros conjuntos de dados. Um sinal prático desse efeito é quando o valor de erro sobre o conjunto de testes começa a subir, embora o erro de treinamento continue decrescendo. Um conceito paralelo a esse é o efeito de subajuste (do inglês *underfitting*), quando o modelo não possui nem a capacidade de apresentar a saída esperada no conjunto de treinamento, necessitando de maior complexidade. A Figura 19 demonstra um exemplo de sub-ajuste e o sobre-ajuste para uma função de aproximação unidimensional.

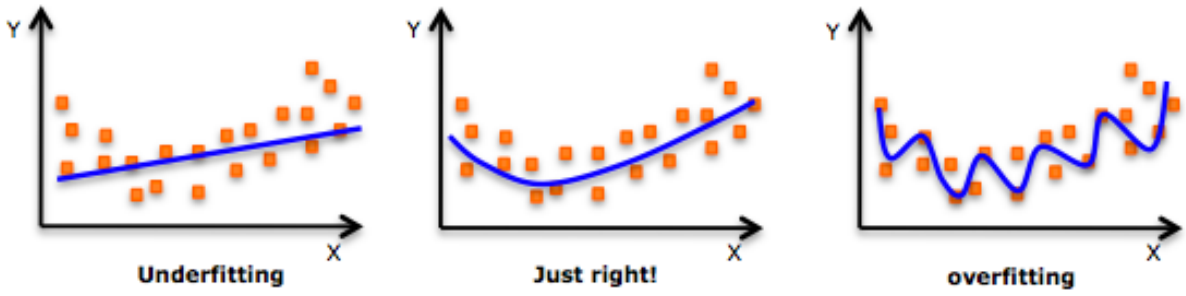

Figura 19 – Exemplo de sub-ajuste e sobre-ajuste de função. À esquerda, falta complexidade no modelo. À direita, vemos que a complexidade do modelo é maior que a necessário pecando em generalização. O modelo ao centro seria o ideal. Fonte: EpicalSoft Blog, Machine Learning.

Nas redes neurais, realiza-se a propagação à frente (do inglês *forward-propagation*)

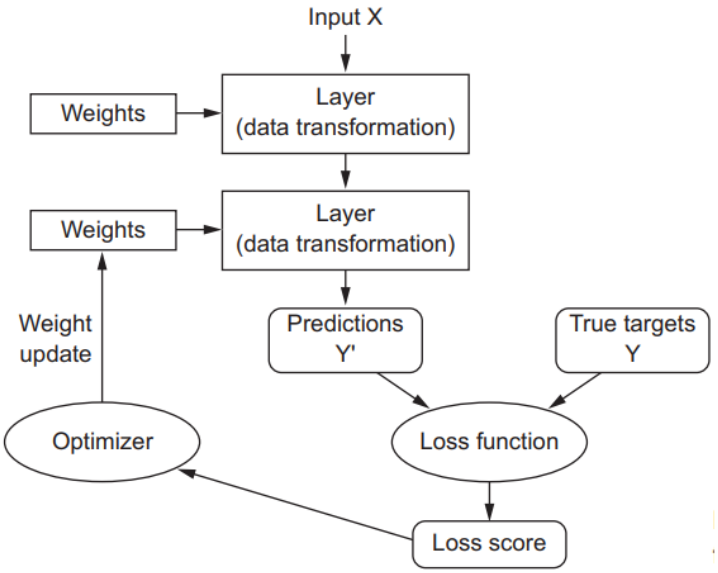

Figura 20 – Processo de otimização de uma Rede Neural Artificial. O valor de perda é utilizado como sinal de *feedback* para ajustar os pesos. Fonte: (CHOLLET, 2017), pg. 11

para obter a saída do seu modelo. Com a saída, é possível checar o quão distante ela está da saída esperada, através do erro retornado pela função de perda. A retro-propagação (do inglês *backpropagation*) consiste no movimento reverso que consiste em encontrar as derivadas parciais do erro com relação aos pesos. Em outras palavras, queremos encontrar o gradiente do erro com relação aos pesos, que pode ser entendido como um vetor que indica a direção no qual o erro cresce. Ao subtrair esse vetor gradiente do vetor de pesos, estamos atualizando o vetor de pesos numa operação conhecida como Gradiente Descendente (do inglês gradient descent). Tal operação faz com que o erro diminua iterativamente, ajustando os pesos para mais ou para menos a depender do gradiente. A Figura 20 demonstra o fluxo do processo discutido nesta subseção para uma ANN com duas camadas.

As ANNs tendem a sofrer com dois tipos de problema no processo de retro-propagação: (1) a explosão dos gradientes (do inglês, *exploding gradients*); e (2) o desaparecimento dos gradients (do inglês, *vanishing gradients*) (BUILTIN, 2021). Tais problemas se referem ao tamanho do gradiente, ou seja, da derivada parcial dos pesos em relação ao erro.

A explosão do gradiente ocorre quando o algoritmo de otimização atribui um valor cada vez mais alto para os pesos, levando o modelo a se distanciar cada vez mais do ponto de erro mínimo. Além disso, modelos com pesos demasiadamente altos tendem a apresentar maior tendência de sobre-ajuste. Esse problema pode ser atenuado através do truncamento dos gradientes, ou de métodos de regularização.

Já o desaparecimento do gradiente ocorre quando os seus valores são muito pequenos, tendendo a zero ao retro-propagar pelas camadas. Tal fato faz com que o modelo pare de aprender ou demore um tempo demasiadamente longo ou impraticável de se esperar. Esse problema pode ser atenuado reduzindo o número de camadas densas no modelo.

Segundo Chollet (2017), os métodos de regularização se baseaiam no Princípio de Occam para a resolução de problemas: dados duas explicações para algo, a explicação mais simples é a mais provável de ser correta. Aplicando esse conceito às ANNs, dado um conjunto de dados de entrada, há diversos modelos que podem explicar os dados. Esses modelos incluem uma arquitetura e os seus pesos. Nesse cenário, os modelos simples são menos prováveis de apresentar sobre-ajuste quando comparados a modelos complexos.

Modelos simples são aqueles com poucos parâmetros ou modelos onde a distribuição dos valores dos parâmetros é homogênea, com menos entropia. Métodos de regularização tentam atingir justamente isso: adicionar limitações na complexidade de uma rede, forçando os pesos a assumir valores pequenos para gerar uma distribuição de pesos mais regular. Essa técnica consiste em adicionar à função de perda um custo associado com o valor dos pesos: na regularização L1, o custo é proporcional ao valor absoluto; na regularização L2, o custo é proporcional ao quadrado do valor do peso, e é também chamada de decaimento do peso.

Uma outra forma importante de regularização é o *dropout*, uma das formas mais efetivas e comumente utilizadas para Redes Neurais. Quando aplicada a uma camada, o *dropout* consiste em tornar zero o valor de certos pesos escolhidos aleatoriamente durante o processo de treinamento. A taxa de *dropout* é a fração dos pesos daquela camada que serão zerados, geralmente escolhidos entre 0,2 e 0,5. Na fase testes, o *dropout* é desativado e não influência no resultado final - de fato, durante os testes, a saída das camadas que possuem *dropout* são diminuídas pela mesma taxa para compensar o fato de mais neurônios estarem ativos.

No caso de uma tarefa de classificação, a última camada dentre as densas deve conter o número de neurônios igual ao número de classes existentes (possivelmente com uma função de ativação *softmax*). A *softmax* garante que o fato de uma classe ser mais provável implica nas demais classes serem menos prováveis (indicado nos casos onde apenas uma classe é considerada correta por amostra). A função de perda mais utilizada para n classes é a entropia cruzada categórica (CHOLLET, 2017). Para casos onde há apenas 2 classes, a entropia cruzada binária é mais indicada.

Algumas técnicas podem auxiliar na divisão do conjunto de dados entre treinamento e testes, de forma a diminuir a influência da escola aleatória sobre os resultados. Duas dessas técnicas são comumente utilizadas no treinamento das ANNs: estratificação e validação cruzada. A estratificação consiste em aproximar ao máximo a distribuição de classes entre diferentes conjuntos. A validação cruzada consiste em dividir os dados disponíveis em K partições (do inglês, $folds$), geralmente 4 ou 5, instanciando K diferentes modelos e treinando cada modelo com $K-1$ partições enquanto a última serve para validação. O score final de validação é tomado então como a média dos K scores de validação obtidos em cada instância. A Figura 21 demonstra visualmente o conceito de validação cruzada.

2.4.3 Redes Neurais Convolucionais

As Redes Convolucionais são uma variante das ANNs que apresentam ótimos resultados quando os dados de entrada compreendem estruturas com padrões locais, como áudio, imagens e vídeos. Podemos definir uma imagem como um conjunto de pontos (pixels) organizados em forma de uma matriz. De forma análoga, um vídeo é uma sequência de imagens dispostas em sequência. Uma imagem em escala de cinza é uma matriz de pontos unidimensionais, onde cada ponto representa a intensidade de um pixel. Já uma imagem colorida é representada por uma matriz de pontos 3D, onde cada ponto representa os três valores de cor presentes no pixel: vermelho, verde e azul. Independentemente do seu tipo, as imagens podem ser utilizadas como dados de entrada para as ANNs, bastando que sejam devidamente estruturadas e organizadas.

As Redes Neurais Convolucionais (do inglês *Convolutional Neural Networks*, ou CNNs, ou ainda convnets) são um ramo das ANNs que utilizam filtros convolucionais para a extração de características locais hierárquicas. A motivação inicial para as Redes Convolucionais foi obtida através do entendimento de Hubel e Wiesel (2020) sobre o funcionamento do córtex visual dos gatos, no qual porções específicas do campo visual excitam grupos específicos de neurônios em particular.

A diferença fundamental entre uma ANN comum e uma CNN é a seguinte: camadas densas aprendem representações globais sobre o seu espaço de entrada (padrões que envolvam todos os pixels), enquanto camadas convolucionais aprendem representações locais (no caso de imagens, padrões internos a uma pequena janela 2D denominada campo receptivo). Isso é possível devido à aplicação de filtros convolucionais, que varrem a imagem em busca de determinados padrões. A Figura 22 ilustra uma visão simplificada do funcionamento de uma CNN, na qual uma imagem pode ser entendida como a composição hierárquica de padrões locais mais simples.

Com relação às imagens, na prática, uma CNN é composta por duas etapas: a etapa convolucional e a etapa densa (CHOLLET, 2017). A etapa convolucional é formada por um

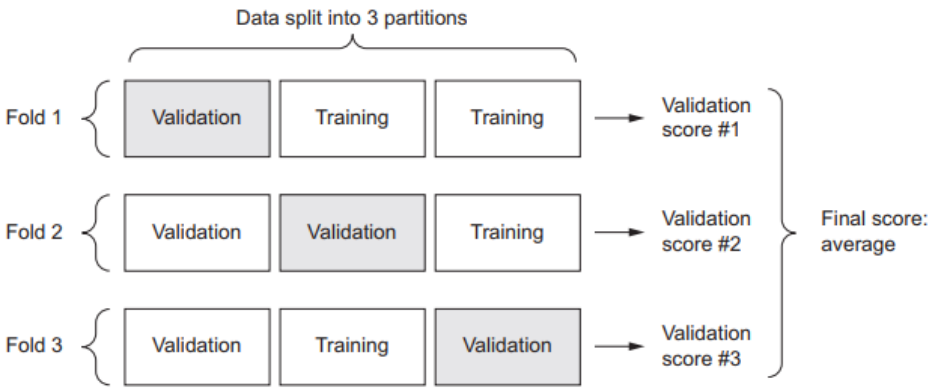

Figura 21 – Exemplo de validação cruzada utilizado $K = 3$ partições.

empilhamento intercalado entre dois tipos de camadas: convolucionais e de *pooling*.

Uma camada convolucional é formada por um ou mais filtros convolucionais, onde cada filtro possui um núcleo (do inglês *kernel*) de dimensão constante e é responsável por localizar determinada característica (do inglês *feature*) relevante para o modelo durante sua varredura. A varredura dos filtros é realizada a um passo (ou *stride*) comum e constante dentro de uma mesma camada, assim como é constante a sua dimensão.

Outro hiper-parâmetro que precisa ser escolhido é o preenchimento (do inglês *padding*). Ele realiza a função de determinar se a saída da camada convolucional diminuirá sua largura/altura (processando só os pixels que cabem no filtro) ou se manterá inalterada, através do preenchimento das bordas com zero. A Figura 23 representa o início de uma varredura convolucional sobre uma imagem RGB, utilizando os seguintes hiper-parâmetros: filtro 3x3, com passo 1 e preenchimento das bordas com zeros. Nesse exemplo, ambas a imagem de entrada e de saída possuem dimensão 5x5 graças ao preenchimento das bordas durante o processo de varredura.

Uma camada de *pooling* serve para diminuir a complexidade de saída da camada anterior, reduzindo sua dimensão e extraindo as informações de valor mais relevante. Assim como na convolução, é necessário especificar a dimensão da janela de varredura, seu passo e o tipo de preenchimento de bordas como hiper-parâmetros. Durante o processo de *pooling*, apenas um valor dentro da janela receptora é escolhido para ser passado adiante, mediante um determinado critério. Os critérios mais comum são o valor máximo e a média. Vale ressaltar que camadas de pooling não possuem pesos.

Segundo Chollet (2017), é possível sintetizar as características das camadas convolucionais nos seguintes tópicos:

- As features aprendidas são invariantes à posição na imagem. Após aprender um certo padrão em determinada região da imagem, a rede convolucional é capaz de localizá-la

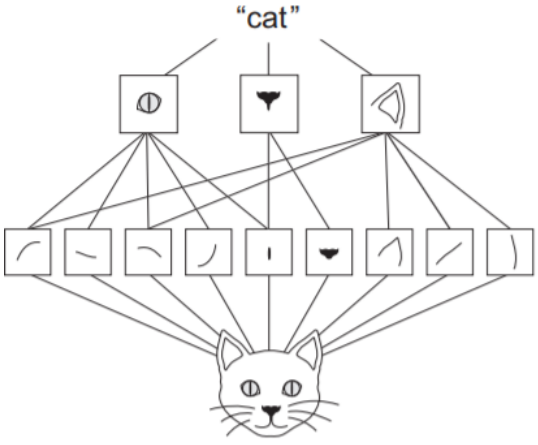

Figura 22 – Ilustração simplificada do processo hierárquico das Camadas Convolucionais. Imagens podem ser entendidas como a composição de padrões locais tais como bordas e texturas. Fonte: (CHOLLET, 2017), p.123

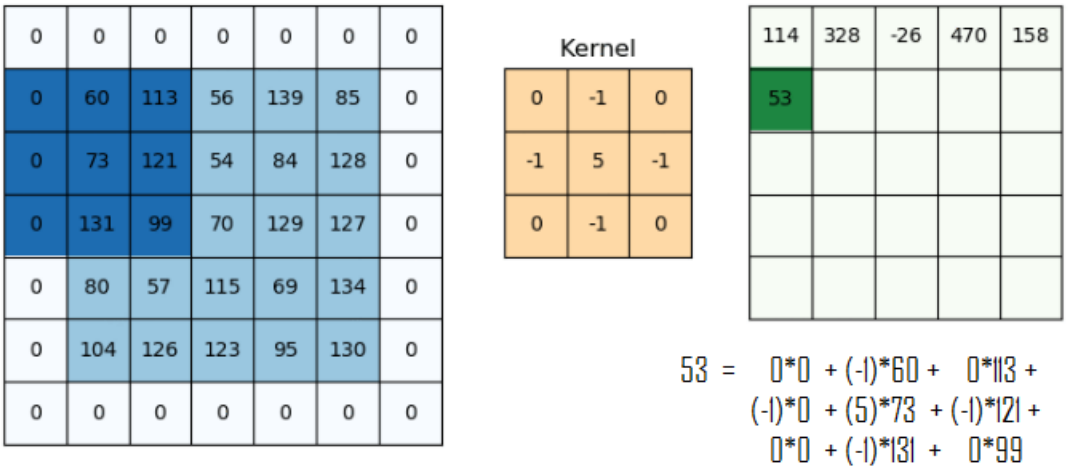

Figura 23 – Exemplo de início de varredura de um filtro convolucional utilizando o zero padding. Fonte: Imagem adaptada de PyImageSearch, Keras Conv2D and Convolutional Layers.

em qualquer posição. Uma rede densa comum teria que reaprender o padrão, caso ele aparecesse em outro local diferente do aprendido. Nesse quesito, CNNs apresentam uma melhor generalização para imagens não controladas.

- São aprendidas hierarquias espaciais de padrões. Uma primeira camada convolucional aprenderá padrões locais simples, como arestas. Uma segunda camada aprenderá padrões feitos das das primeiras camadas, e assim por diante. Tal fato leva ao aprendizado eficiente de conceitos visuais incrementalmente complexos e abstratos.

Por fim, após a etapa convolucional em uma CNN, as features extraídas pela última camada da etapa convolucional passam por uma etapa de enfileiramento de dimensões (do inglês *flattening*), sendo transformada num vetor unidimensional antes de entrar nas camadas densas.

2.4.4 Redes Neurais Recorrentes

Uma característica importante sobre as ANNs comuns é o fato delas não possuírem memória. A inteligência biológica difere nesse sentido. Ela processa informação incrementalmente enquanto mantém um modelo interno do que está sendo processado, construído através de experiências passadas e sendo constantemente atualizado a cada nova informação. Este é o conceito utilizado como base para as Redes Neurais Recorrentes (do inglês *Recurrent Neural Networks*, ou RNNs). As RNNs mantém um estado contendo informações relativas ao que foi visto até então. Comparada à ANN, cada amostra da RNN é um conjunto de valores em forma de sequência. A arquitetura possui um loop que itera sobre os elementos da sequência de entrada, como mostrado na Figura 24.

O estado (ou saída) de uma RNN no tempo t é calculado através do seu último estado em $t - 1$ e sua entrada atual no tempo t. É válido ressaltar que todas as conexões na RNN

possuem peso e bias, incluindo as conexões do loop interno. Outro fator importante é conceito de compartilhamento de pesos no tempo. Isso indica que a mesma célula recorrente (com os mesmos pesos) é utilizada para elementos consecutivos em uma amostra.

Assim como há a retro-propagação para as ANNs *feed-forward* de amostra de entrada unitárias, nas RNNs existe o conceito retroprogação no tempo (do inglês *backpropagation through time*, ou BPTT) para amostras em forma de sequência. O BPTT difere do método tradicional uma vez que soma os erros em cada passo de tempo, já que os pesos são compartilhados através do tempo. Para facilitar o entendimento de tal conceito, é possível utilizar o conceito de desenrolamento (do inglês, *unrolling*), onde elementos em diferentes instantes de tempo são representados em posições distintas. A versão desenrolada de uma RNN é disposta nas Figuras 25 e 26. Perceba que a entrada x_t é concatenada com o estado interno anterior h_{t-1} antes de passar pela camada densa com função de ativação tangente hiperbólica. No processo do BPTT o erro é propagado do último passo de tempo para o primeiro. Note que o processo se torna computacionalmente custoso à medida que os passos de tempo aumentam.

Os problemas da explosão e do desaparecimento do gradiente discutidos na Seção 2.4.2 estão presentes em todas as derivações das ANNs, embora nas RNNs seus efeitos sejam acentuados pelo BPTT. Outra característica importante sobre o seu uso, é que a RNN comum descrita nesta seção possui uma memória curta (do inglês *short-term memory*). Isso significa que ela é capaz de lembrar apenas sobre um passado recente, imediato, definido pela quantidade de passos de tempo considerada. Para tratar dessas características, surge o conceito de *gates* e da arquitetura conhecida como *Long Short-Term Memory*, que será discutido na seção a seguir.

Um último fato importante sobre as RNNs é o formato dos dados de entrada. Como elas esperam sequências ao invés de amostras únicas na entrada, é necessário realizar o processo conhecido como janelamento, ou janela deslizante (do inglês *sliding window*) de forma a concatenar amostras de entrada em sequências de tamanho fixo. Por exemplo, seja $X = [1, 2, 3, 4, 5]$, de tamanho $N = 5$, o nosso conjunto de dados originais. Após o janelamento com um tamanho

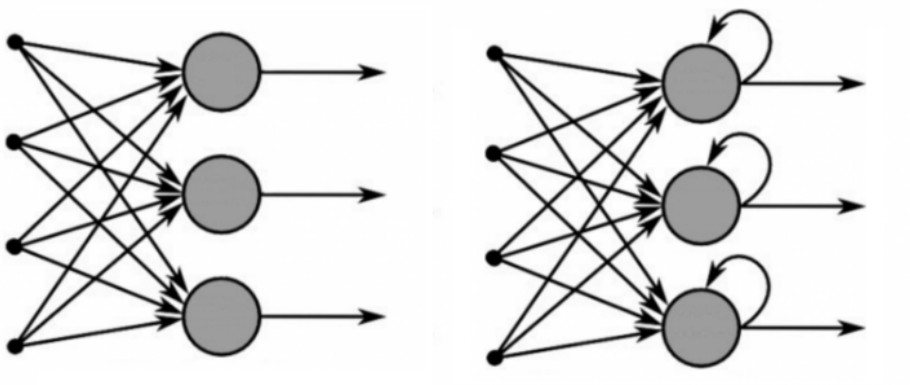

Figura 24 – Comparação entre as arquiteturas *feed-forward* (à esquerda) e recorrente (à direita). Na *feed-forward*, os dados transitam em apenas uma direção, enquanto na recorrente há loops na arquitetura. Fonte: imagem adaptada de Builtin DataScience

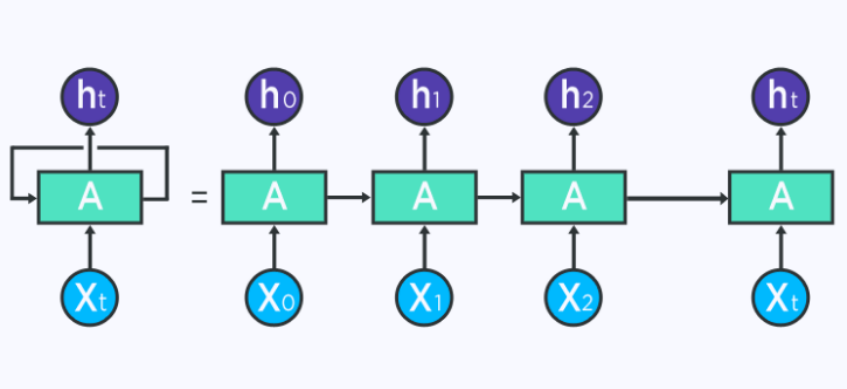

Figura 25 – Versão desenrolada de uma RNN sobre o tempo. Fonte: Builtin DataScience.

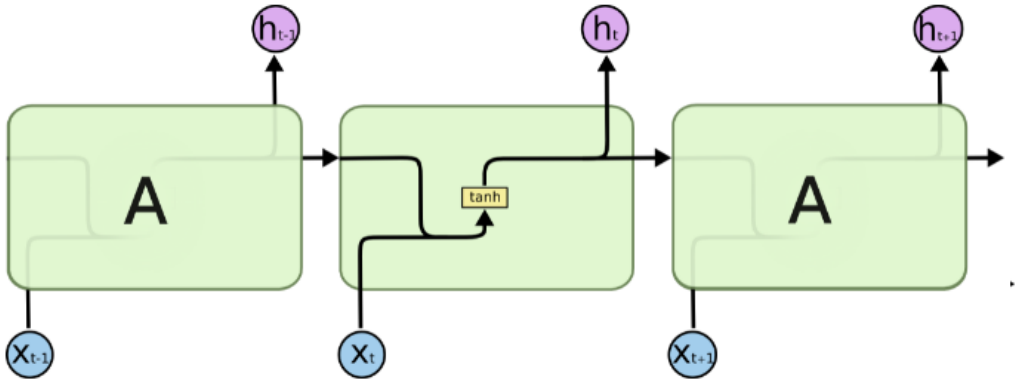

Figura 26 – Visão interna de uma célular RNN simples. Fonte: Christopher Olah's blog.

fixo $S = 3$ obtemos o conjunto $X' = [[1, 2, 3], [2, 3, 4], [3, 4, 5]]$. Perceba que o tamanho total do conjunto após o janelamento é dado por $N - K + 1$.

2.4.5 Redes Long Short-Term Memory

As redes Long Short-Term Memory estendem o conceito de memória das RNNs comuns, adicionando a capacidade do ajuste fino em sua memória interna. Se tornam, então, apropriadas para situações onde seja necessário uma memória seletiva que inclua experiências, não apenas de um passado recente, mas também temporalmente distantes. A arquitetura foi proposta por Hochreiter e Schmidhuber (1997) ao tratar do problema da dependência de curto prazo fazendo uma analogia às memórias dos computadores.

Da mesma forma que é possível ler, deletar e escrever na memória de um computador, as células LSTM foram propostas seguindo uma arquitetura baseada em três válvulas (ou *gates*), as quais: (1) *gate* de esquecimento, que decide deletar a informação quando ela não é importante; (2) *gate* de entrada, que decide qual informação é relevante para ser adicionada ao passo atual; (3) *gate* de saída, que determina qual o próximo estado será.

A Figura 27 demonstra a arquitetura desenrolada da LSTM. Perceba que a entrada atual x_t também é concatenada com o estado (ou saída) anterior h_{t-1}, assim como na célula RNN

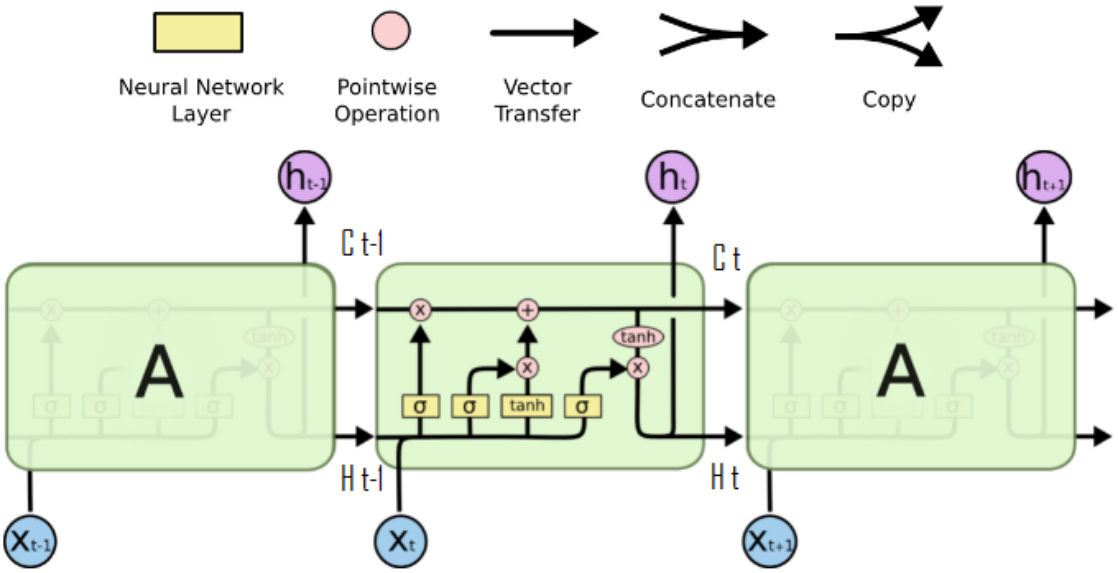

Figura 27 – Arquitetura desenrolada da LSTM. Fonte: Adaptada do blog de Christopher Olah.

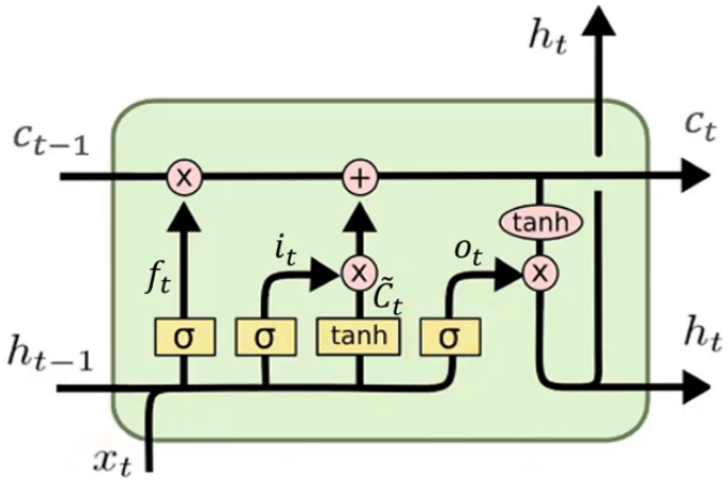

Figura 28 – Visão interna detalhada das células LSTM. Fonte: Adaptada do blog de Christopher Olah.

comum. Aqui, entretanto, vemos a presença de camadas neurais com duas ativações diferentes: a sigmoide (ou função logística), representada pela letra grega sigma, possuindo saída entre 0 e 1; e a tangente hiperbólica, representada pelo acrônimo tanh, com saída entre -1 e 1.

A Figura 28 demonstra a visão interna detalhada de uma célula LSTM. O ramo horizontal superior representa o fluxo do estado interno da célula, que atualiza c_{t-1} para formar c_t. O gate do esquecimento é o responsável por gerar f_t. O gate de entrada gera $i_t * \tilde{C}_t$. O gate de saída gera h_t. Todo o processo pode ser descrito nas seguintes equações:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.36}$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.37}$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{2.38}$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.39}$$
$$c_t = f_t * c_{t-1} + i_t * \tilde{C}_t \tag{2.40}$$
$$h_t = o_t * \tanh(c_t) \tag{2.41}$$

2.4.6 OpenCV

O OpenCV é uma biblioteca de código aberto que inclui centenas de algoritmos voltados para a Computação Visual (BRADSKI, 2000). Sua licença é a BSD3, o que permite que a biblioteca seja utilizada tanto para pesquisa quanto para fins comerciais de forma gratuita. É otimizada para aplicações em tempo real, estando disponível em C++, Python and Java, além de oferecer suporte para Linux, MacOS, Windows, iOS, e Android. No presente trabalho, toda a parte de procesamento de imagens é realizada através do OpenCV. A versão utilizada nesse trabalho foi a versão padrão do ROS Melodic, que é a 3.2.

2.4.7 ScikitLearn

Scikit-learn (PEDREGOSA et al., 2011) é um módulo em Python de código aberto para Aprendizagem de Máquina construída sobre a biblioteca ScriPy, também distribuída sobre a licença BSD3. O projeto iniciou em 2007 como um projeto do programa Google Summer of Code por David Cournapeau, tendo recebido contribuições de vários voluntários desde então. Em nosso trabalho, utilizamos o scikit-learn para operações envolvendo datasets, tais como divisão entre conjunto de treino/testes utilizando de *folds* estratificados.

2.4.8 TensorFlow

O TensorFlow (ABADI et al., 2016) é uma plataforma de código aberto para Aprendizagem de Máquina bastante conhecida por suas otimizações na área de Redes Neurais, sob a licença Apache2. Foi originalmente desenvolvido por pesquisadores e engenheiros trabalhando no time da Google Brain. Uma das suas vantagens é o seu suporte ao uso das placas de vídeo (através do CUDA) para a aceleração de operações como a retro-propagação e o gradiente descendente. Há suporte a linguagens de programação como C++, JavaScript e Python, além de suporte pra Windows, Linux, MacOS, iOS e Android. A versão do TensorFlow utilizada nesse trabalho foi a 1.10.

2.4.9 Keras

O Keras (CHOLLET et al., 2015) é uma bibioteca de código aberto que provê uma interface em Python para Redes Neurais Artificiais, funcionando como um *wrapper* para facilitar a prototipação. Até a versão 2.3, o Keras suportava múltiplos backends, incluindo TensorFlow e Theano. A partir da versão 2.4, apenas o TensorFlow passou a ser suportado. Hoje, o Keras é embutido no Tensorflow. Seu foco é a praticidade, extensibilidade e a modularização. Seu principal autor e mantenedor é o François Chollet, um engenheiro da Google a autor de um dos livros de referência utilizados nesse trabalho (CHOLLET, 2017). A versão do Keras utilizada nesse trabalho foi a 2.2.

3 TRABALHOS RELACIONADOS

Neste capítulo, é feita uma revisão da literatura com os principais trabalhos relacionados ao tema. Estão incluídos aqueles que usam técnicas similares, mesmo que aplicados a temas distintos.

3.1 Tele-manipulação Robótica

O problema da tele-manipulação robótica tem sido abordado por diferentes técnicas ao longo da última década. Em termos de nível de ação, Barros e Lindeman (2009) discutem o tema abordando duas subdivisões: mapeamento direto, quando o robô é comandado a descrever o movimento do corpo do usuário; e mapeamento indireto, quando o robô recebe informações em mais alto nível, decodificando e transformando num movimento real. O trabalho até exemplifica casos de tais formas de mapeamento quando cita a seguinte frase: "Um exemplo de mapeamento direto seria usar o movimento do braço do usuário de um operador para controlar um braço robótico. Um exemplo de mapeamento indireto é utilizar um joystick pra controlar o quão rápido as rodas de um robô estão girando". Quanto à tele-manipulação baseada em movimentos corporais, a principal vantagem reside exatamente no controle direto, dispensando o uso de componentes de hardware adicionais como joysticks ou dispositivos vestíveis. Por fim, o survey faz uma extensa discussão sobre aspectos técnicos da teleoperação em geral, tais quais a definição de sensores de entrada e saída mais comuns, técnicas de *feedback* visual e táctil e métricas utilizadas para a validação de modelos HRI. Dentre as métricas discutidas, as seguintes são apropriadas para o presente trabalho: tempo para completar cada tarefa, confiança do usuário, engajamento do usuário, esforço/conforto do usuário, acurácia de reconhecimento na percepção visual.

O survey realizado por Lichiardopol (2007) é uma cobertura mais direta quanto aos aspectos computacionais do tema. O trabalho apresenta definições, bem como um histórico do desenvolvimento da tele-operação. Segundo eles, o conceito de teleoperação é dividido em três classes:

- *Controle de malha fechada.* Nesse caso, o operador controla os atuadores através de sinais diretos, recebendo *feedback* do robô em tempo real. Tal característica só é possível quando os *delays* no loop de controle são mínimos. Nesses casos, não há loop interno no robô, e toda a tarefa de controle é realizada pelo operador. Uma exemplo desse tipo de controle são os carros rádio-controlados: caso a comunicação com o controle de rádio cesse, o carro para, pois não mantém estado;
- *Teleoperação coordenada.* Aqui, o loop interno da malha está presente no robô, e o papel do operador é apenas definir o *set-point* desejado. Não há, entretanto, autonomia da parte

do robô. Nosso trabalho está inserido nesse ramo: as juntas do robô são transmitidas via rede, mas o robô possui um controlador próprio responsável pelo controle PID de seus valores;

- *Controle supervisório.* Nesse caso, a maior parte do controle e do comportamento do robô são realizados automaticamente. O usuário basicamente monitora o seu funcionamento transmite alguns comandos de alto nível. Um exemplo são os robôs de patrulha, que realizam trajetórias e desvio de obstáculos por si próprio, avisando ao usuário caso algo fora do comum seja detectado.

A segurança é uma das principais temáticas abordadas quando o assunto é interação entre homem e robô. Por exemplo, Lasota et al. (2017) discorrem sobre as implicações da presença robótica no espaço de trabalho humano e como ela pode ser realizada de modo a garantir a segurança de ambos homem e robô. O survey discorre sobre a necessidade de abordar os aspectos físicos, psicológicos e sociais sobre os indivíduos que se tornam expostos à percepção, cognição e ação robóticas. O trabalho também demonstra que houve um aumento em dispositivos sensoriais RGB-D a partir de 2010, causando um notável interesse por sistemas de segurança baseados em percepção visual. Como dados de entrada, esses sistemas utilizam informação de profundidade sobre o ambiente, humanos e outros robôs. Segundo eles, a maior vantagem desses sistemas é uma alta invariância à cor e condições de iluminação, embora alguns deles sejam fortemente prejudicados em ambientes ao ar livro devido à luz do sol. Uma implicação direta é o uso de dados de profundidade para o reconhecimento de esqueleto de usuário e inferência de pose humana. Segundo o survey, sistemas de localização e reconhecimento de usuário baseados em visão-computacional abrem a possibilidade de interações em alto-nível, como detecção de ações e comandos baseados em movimento.

Na área específica do controle de manipuladores robóticos via movimentos corporais, o survey mais recente foi realizado por Li, Wang e Liu (2021), o qual discorre sobre as vantagens e desafios envolvendo a captura do movimento das mãos do usuário. O trabalho destaca as vantagens deste tipo de controle em três motivações principais: (1) é possível executar as ações de propósito geral num tempo reduzido; (2) remove da máquina a necessidade de algoritmos de detecção de objetos, planejamento de trajetória e detecção de posição de pegada para o efetuador; (3) permite a expansão do leque de ações do robô, antes condicionadas pela capacidade de programação e de inteligência artificial.

Ainda segundo Li, Wang e Liu (2021), há duas abordagens principais quanto à teleoperação baseada no movimento da mão do usuário, que são os dispositivos hápticos e os métodos baseados em visão computacional. Métodos baseados em visão requerem apenas uma câmera para a captura do movimento humano. Os métodos de mapeamento hápticos, por outro lado, são baseados principalmente em dispositivos intrusivos, tais como luvas, para estimar a pose da mão e fornecer um retorno tátil ao usuário. Ambas as abordagens podem utilizar diferentes técnicas

de estimação de pose (detecção da posição das juntas e orientação dos elos) e de reconhecimento de gestos. A Figura 29 ilustra essa subdivisão. Para o presente trabalho, é válido ressaltar a distinção entre mapeamento junta-a-junta e ponto-a-ponto: na primeira variante, é feita uma correspondência no espaço de juntas, enquanto na variação ponto-a-ponto o mapeamento é realizado no espaço cartesiano (espacial). Embora o survey não mencione, há ainda uma terceira classe composta pelos dispositivos vestíveis que não fornecem retorno tátil, como sensores de eletromiografia e acelerômetros. A proposta do nosso trabalho atual se encaixa em duas vertentes destacadas no diagrama: mapeamento ponto-a-ponto, uma vez que mapeamos o centro da mão do usuário para o centro do efetuador do robô; e mapeamento de pose dinâmica, uma vez que utilizamos RNNs para detectar o estado da mão do usuário.

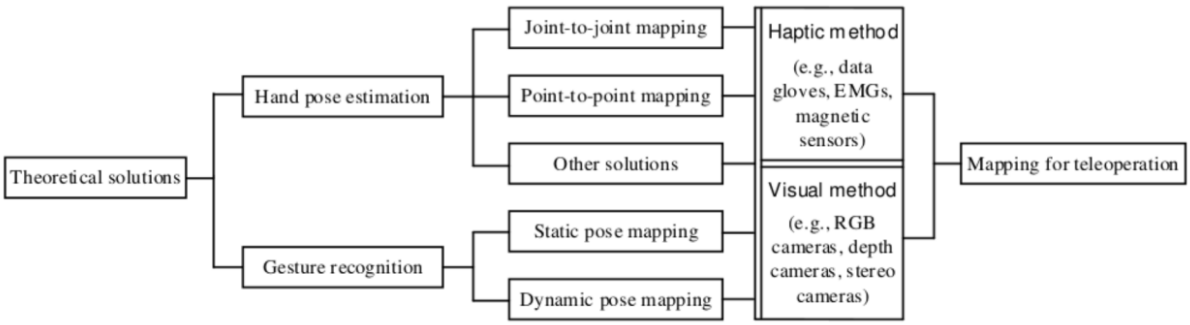

Figura 29 – Subdivisão das abordagens de mapeamento mão-robô. Fonte: Li, Wang e Liu (2021), pg 3.

Há trabalhos importantes quanto à teleoperação robótica de forma geral. Nuño, Basañez e Ortega (2011) introduziram arquiteturas de controle baseadas em passividade para operação bilateral, discutindo sobre propriedades de estabilidade. Por controle bilateral, entende-se aquele onde o robô transmite retorno tátil ao mestre. Sobre o tema, Hokayem e Spong (2006) apresentaram um survey com uma clara disposição histórica, além de propor tratamentos teóricos sobre técnicas de controle, problemas de atraso e problemas de perda de informação. Nosso trabalho é baseado unicamente no retorno visual obtido pelo usuário, não possuindo retorno tátil. Portanto não pode ser classificado como bilateral.

Uma outra variante do controle robótico através de movimentos corporais consiste no uso de sensores de eletromiografia (EGM) que fornecem um valor de tensão de acordo com os impulsos elétricos emitidos pela atividade muscular. Essa abordagem é utilizada por Gowtham et al. (2020), que realizaram a medição da atividade muscular na região do cotovelo para comandar um braço antropomórfico de 5 DOF, realizando a correspondência direta dos ângulos das juntas do braço para os ângulos das juntas do robô (mapeamento junta-a-junta). No processo de captura do movimento do usuário, eles utilizaram dois acelerômetros e um sensor EMG. Como o mapeamento proposto por eles foi realizado no espaço de juntas, a completa orientação do braço de usuário era replicada no robô. Em nosso caso, o efetuador do robô se mantém numa orientação fixa, voltada para baixo, própria para facilitar tarefas de *pick-and-place*, enquanto é realizado o

mapeamento ponto-a-ponto.

Em Lipton, Fay e Rus (2017), o tema da teleoperação é abordado num cenário de manufatura através de técnicas de realidade virtual (do inglês *virtual reality*, ou VR). Eles introduzem o conceito de Sala de Controle em Realidade Virtual (do inglês *Virtual Reality Control Room*, ou VRCR), onde o estado do usuário é mapeado para o VRCR, e então o VRCR é mapeado para o sistema robótico, desacoplando o retorno sensorial (que é proporcionado pelo ambiente virtual) da comunicação com o robô. Eles testaram o sistema junto ao Óculos Rift com controles *Touch*, utilizando o robô Baxter. O Baxter provia feedback visual para o usuário de duas fontes: uma câmera estéreo localizada na cabeça do robô, e uma câmera localizada em cada efetuador. Eles analisaram os efeitos da comunicação numa rede de alta-latência, simulando como a teleoperação se comportaria num sistema de conectividade limitada. Nosso trabalho, nesse primeiro momento, não realizou uma análise detalhada da influência da teleoperação sobre a rede, mas garantiu que o sistema funcionasse a 30 FPS numa rede local, que é a velocidade máxima de captação da nossa câmera de profundidade. Similarmente a nós, eles também testaram o sistema para tarefas de *pick-and-place* e de montagem, medindo o tempo decorrido. A metodologia deles, assim como a nossa, consistia em dividir as tarefas em unidades menores chamadas *sub-tasks*. Essas *sub-tasks* envolviam a interação com cubos, que podiam ser classificadas como tarefas de aproximação, alinhamento, pegada ou montagem (empilhar um cubo sobre outro). A tentativa era considerada uma falha quando o usuário executava um erro irrecuperável, alinhava os blocos erroneamente, ou demorava mais que 6 minutos para completar a tarefa. Nosso trabalho utilizou um entendimento similar para o entendimento de falhas, possibilitando até 3 falhas antes da tarefa ser considerada não-cumprida. Além disso, não foi estipulado um prazo máximo, embora todos os usuários realizassem suas tentativas em menos de 2 minutos. Diferentemente de nós, eles mapearam ambos os braços do usuário para o Baxter. Similarmente a eles, nós também utilizamos um ambiente virtual como intermediário entre usuário e robô, embora este tenha sido usado apenas para diretrizes de segurança.

Em Lee, Park e Park (2018), o problema é abordado do ponto de vista de um enfermeiro robótico. O foco deles foi na colaboração homem-robô em ambientes médicos, realizando um estudo sobre a aceitabilidade de um sistema robótico tele-assistivo entre profissionais da saúde. A metodologia consistia em testar 3 diferentes modos de controle para a tarefa de manipulação: controle háptico, controle por gestos e comandos de voz. O método de controle por gestos utilizou um Kinect para a captura do da parte superior do esqueleto do usuário e de sua pose. Também foram utilizados dois sensores EMG (na forma de pulseira) para realizar o papel de reconhecimento de gestos na mão do usuário (abrir/fechar mão). As pulseiras também proviam informação de giroscópio para medir a rotação relativa da mão do usuário, mapeando essa informação para o movimento da base do robô. Os experimentos de validação utilizaram 11 estudantes de medicina e engenharia, avaliando dois parâmetros principais na forma de uma pesquisa na escala de Likert: dificuldade no controle e segurança. Diferentemente da abordagem deles, em nosso trabalho a base do robô era fixa. Além disso, ambas a pose do

usuário e o classificador da mão do usuário eram inferidos diretamente dos dados RGB-D. Nossa pesquisa pós-experimento também utilizou a escala Likert, mas com diferentes aspectos medidos (suavidade do movimento e dificuldade próximo aos limites do workspace).

Chen et al. (2020) discutem o problema do mapeamento cinemático enquanto propõe um critério de similaridade baseado em juntas virtuais para a avaliação da performance do mapeamento. Duas plataformas foram implementadas para verificar experimentalmente a flexibilidade e a eficácia da métrica proposta, ambas usando a plaraforma MATLAB. Especificamente, eles usaram o *Robotic Toolbox* para o modelo de visualização e o *Optimization Toolbox* para o mapeamento cinemático. Dois robôs com 6 DOF foram utilizados: o UR5 e o IRB2400. Similar ao nosso projeto, um mapeamento cinemático de um dos braços do usuário para o braço robótico foi o objetivo do trabalho, sendo dividido em três passos: captura da pose do usuário, transformação cinemática e, por último, transferência para o robô. Diferentemente deles, nossa métrica de mapeamento está implícita ao uso da TPS, que se baseia no conceito da menor energia de deformação possível entre os mapeamentos. Em nosso caso, o mapeamento é realizado através da especificação de pontos entre os espaços de trabalho do controlador e do robô controlado. Por fim, o trabalho deles não abordou o mapeamento entre o estado da mão do usuário e o efetuador do robô.

Lima et al. (2019) abordaram o problema da teleoperação através de uma câmera de profundidade para rastrear e mimicar os movimentos do teleoperador através de uma interface natural de usuário. Neste trabalho é realizado um mapeamento ponto-a-ponto da posição da mão do usuário, em relação a um sistema de coordenadas em seu ombro, para a posição do efetuador, com relação a um sistema de coordenadas na base do robô. A abordagem proposta era invariante à posição e orientação do usuário no espaço da câmera. Entretanto, o mapeamento proposto era um simples mapeamento linear, o qual que deixa a desejar pelo fato do operador e do imitador possuírem espaços de trabalho com diferentes formatos, além de cada operador possuir diferentes limitações cinemáticas e conceito de conforto. O manipulador consistia no Denso VP6242 junto à Robotiq 2F-85. Em nosso trabalho, por outro lado, é realizada a proposição de um mapeamento que envolve componentes lineares e não-lineares, com um modelo baseado no uso da TPS que é customizado para cada usuário. Tal modelo resolve o problema das limitações cinemáticas específicas em que haja perda de mobilidade por parte do braço robótico.

3.2 Reconhecimento de Gestos da Mão por Visão Computacional

Nos últimos anos, modelos de aprendizagem profunda tem mostrado ótimos resultados na área de classificação de mãos, em especial aqueles baseados em visão computacional. O survey apresentado por Rautaray e Agrawal (2015) trata exatamente de modelos baseados em visão para reconhecimento de gestos da mão, sendo um dos mais recentes na área. O autor enfatiza a busca por modelos de interação homem-máquina (IHM) que sejam tão naturais quanto a interação entre humanos. São citadas as principais aplicações, dentre as quais está inserida a robótica,

como mostra a Figura 30. Segundo o trabalho, o controle robótico pode ser beneficiado através do reconhecimento de gestos, com aplicações na robótica móvel, na robótica com múltiplos DOF e na programação por demonstração (quando o robô aprende uma tarefa através da observação de um especialista que a executa).

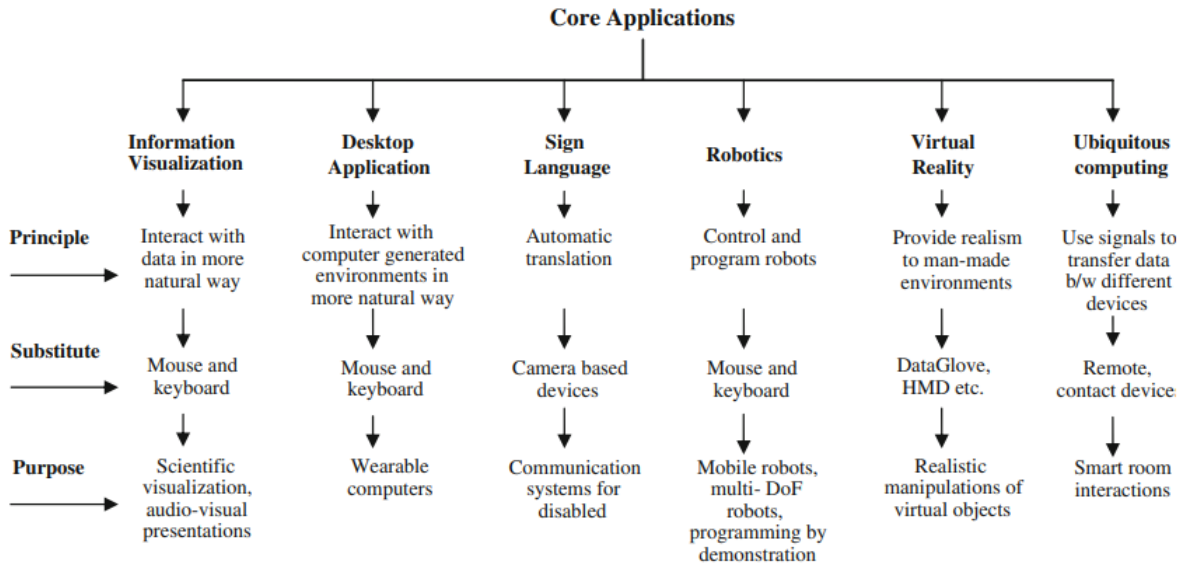

Figura 30 – Aplicações do reconhecimento de gestos de mão baseado em visão computacional. Fonte: Rautaray e Agrawal (2015), pg 23.

Ainda segundo Rautaray e Agrawal (2015), a preferência por modelos baseados em imagens de profundidade se dá pela maior resiliência quanto a variações nas condições de iluminação e quanto à aparência do background. Uma das abordagens de reconhecimento citadas pelo trabalho é o uso das Redes Neurais de Atraso de Tempo (do inglês, *Time Delay Neural Networks*, ou TDNN), que é apenas um outro nome para as RNNs. Em nosso trabalho, o conceito é utilizado no classificador de gestos através das LSTMs.

Sharp et al. (2015) utilizam dados do Kinect V1 para propor um modelo em tempo real para o reconhecimento de gestos. É proposto um sistema híbrido, que combina um modelo discriminativo com um modelo gerativo. O sistema implementa um reinicializador que garante a recuperação em caso de perda de localização, que funciona em paralelo com um reconhecimento temporal baseado em modelo. O reinicializador prediz uma distribuição hierárquica sobre as poses da mão, tendo como entrada as imagens de profundidade. O treinamento desse preditor é realizado sobre imagens sintéticas. O modelo temporal se baseia em algoritmo genético e otimização de enxame, levando em consideração uma nova métrica denominada *energia de ouro* que minimiza o erro de reconstrução entre a malha 3D da mão e a imagem de profundidade observada. Similarmente a nós, o trabalho utiliza imagens de uma única câmera de profundidade como dados de entrada, segmentando os pixels do background. Diferentemente de nós, o trabalho utiliza uma malha 3D da mão além de um modelo estocástico hierárquico (que não utiliza redes neurais).

Li (2012) desenvolve o trabalho na área do Reconhecimento de Gestos de Mão (do inglês, *Hand Gesture Recognition*, ou HGR) utilizando imagens de profundidade do Kinect V1. O sistema compara os gestos do usuário aos sinais de um conjunto desejado, mostrando o significado inferido e a imagem referente ao mesmo. Dois conjuntos são apresentados: o conjunto de gestos populares, contendo 9 gestos; e o conjunto de números, também com 9 gestos. A precisão do do sistemas para os gestos varia entre 84% e 99% utilizando apenas uma mão, e entre 90% e 100% utilizando duas mãos com o mesmo gesto. Toda a detecção de gestos foi realizada utilizando algoritmos de processamento de imagem, sem o uso de redes neurais. O processamento consiste na identificação do fecho convexo contendo a mão, seguida da detecção das pontas dos dedos, para então identificar o gesto baseado no vetores dos dedos e nas angulações relativas entre eles. Diferentemente do trabalho citado, nós utilizamos o Kinect V2, além de um classificador baseado em RNNs.

Cheng et al. (2019) também propõe o reconhecimento estático de gestos da mão com imagens do Kinect V1. O autor propõe um classificador que usa ambas as imagens RGB e de profundidade, utilizando CNN e *Restricted Boltzmann machine* (RBM). Diferentemente deles, nosso trabalho utiliza apenas imagens de profundidade e não utiliza RBMs.

Oyedotun e Khashman (2017) apresentam modelos de classificação estática de imagens de profundidade, utilizados para o reconhecimento de gestos de mão através de CNN e *Stacked Denoising AutoEncoder* (SDAE). O trabalho realiza o reconhecimento de todos os 24 sinais da Língua Americana de Sinais (ASL). É demonstrado que as CNNs e os SDAEs são capazes de aprender a classificar gestos complexos com baixas taxas de erro. Também é realizada a comparação com trabalhos anteriores que utilizavam subconjuntos menores do banco de imagens. Similarmente a nós, o trabalho utiliza imagens de profundidade e redes neurais para classificação. Por outro lado, nós não utilizamos SDAEs, além de usarmos apenas 2 classes de saída.

Amaral et al. (2018) propõem o uso de reconhecimento dinâmico de gestos da mão aplicados à Língua Brasileira de Sinais (LiBraS) através de imagens de profundidade. O autor utiliza a câmera Intel RealSense como sensor de profundidade. Quanto ao classificador, é feita uma comparação entre CNNs 3D e LRCNs. Uma janela de tamanho fixo de 40 frames nos dados de entrada é utilizada. Este o primeiro trabalho dentre os relacionados a utilizar LRCNs, assim como o nosso, embora a arquitetura e os hiperparâmetros sejam diferente em alguns aspectos. Por exemplo, nosso trabalho realiza o teste da influência do *dropout* sobre os resultados, além de utilizar uma janela de 15 frames na entrada. O tamanho reduzido da janela se dá pelo fato dos gestos serem consideravelmente mais simples que os presentes na LiBraS.

Lima et al. (2019) utilizam imagens de profundidade captadas pelo Kinect V1 para desenvolver um classificador estático do estado da mão do operador, utilizando CNNs. O trabalho apresentava oscilações na saída, sendo necessário o uso de um filtro temporal passa-baixa para reduzir os efeitos das oscilações. Diferentemente do trabalho citado, nós utilizamos o Kinect V2 e propomos um classificador baseado em Redes Neurais Recorrentes de Longo-Prazo (do inglês

Long-Term Recurrent Convolutional Networks, ou LRCN). Em nossos testes, não foi necessário utilizar filtros temporais, pois o reconhecimento dinâmico é mais resiliente a oscilações.

3.3 Mapeamento com Thin-Plate Spline

Whitbeck e Guo (2006) implementam um algoritmo de deformação de imagens 2D utilizando a TPS através de múltiplos pontos de controle ao mesmo tempo. Quando publicado, o algoritmo era uma evolução do conhecido AlexWarp, de mesma função, mas que só permitia a deformação com um ponto de controle por vez. Diferentemente de nós, o trabalho utiliza a TPS em 2D, embora aborde definições em comum como energia de deformação, relações matriciais e decomposição entre componentes afins e não-rígidas.

Sinthanayothin e Bholsithi (2009) propõem uma nova forma de análise pré-operatória de cirurgias ortognáticas, realizadas para corrigir assimetrias e deformidades faciais na região maxilar. Primeiro, uma malha 3D de alta densidade do rosto do paciente é montada. Então, uma malha simplificada é montada próximo à região cirúrgica utilizando splines cúbicas, cujos vértices servirão como pontos de controle para a TPS. Após definidos os pontos de controle, a TPS é aplicada sobre a malha original. Dessa forma, através do movimento dos pontos de controle, é possível se obter uma simulação do resultado desejado antes da cirurgia ser realizada.

Hu, Zhou e Wu (2009) utilizam a TPS no alinhamento de pontos característicos entre rostos 3D. A abordagem permite, por exemplo, gerar a animação de metamorfose no tempo, onde a malha 3D passa do estado inicial para o estado final. Chen et al. (2017) abordam o mesmo problema, agora de forma mais técnica, denominado registro craniofacial. O trabalho propõe uma transformação não-rígida, utilizando a TPS, para estabelecer uma correspondência ponto-a-ponto entre modelos craniofaciais em um sistema de coordenadas unificado.

O ponto em comum entre o presente trabalho e os demais citados nessa subseção é o mapeamento através da TPS entre duas superfícies. Em nosso caso, é como se o espaço de trabalho do usuário fosse a superfície inicial, que é transformada através dos pontos de controle (as poses da mão escolhidas) no espaço de trabalho do robô que representa a superfície final. Essa transformação envolve componentes rígidas e não-rígidas e, após calculada, pode ser utilizada para transformar quaisquer outros pontos do espaço de trabalho de origem (usuário) para o espaço de trabalho alvo (robô).

4 MATERIAIS E MÉTODOS

Nesta seção, são discutidos como aplicamos as bases teóricas na busca pela solução das perguntas de pesquisa. A metodologia do presente trabalho é composta por duas partes principais:

- a proposição, implementação e análise de um modelo de classificação da mão do usuário baseado nas relações dinâmicas entre as imagens de profundidade;
- a proposição, implementação e análise de um modelo de mapeamento ponto-a-ponto entre a mão do usuário e o efetuador de um robô.

Iniciaremos especificando a implementação dos classificadores da mão do usuário na Seção 4.1, para então tratarmos dos detalhes do mapeamento de posição na Seção 4.2. Por fim, trataremos da arquitetura geral do sistema de teleoperação desenvolvido como um todo, na Seção 4.3.

4.1 Reconhecimento de Gestos

A metodologia utilizada para o reconhecimento de gestos foi baseada nos trabalhos de Amaral et al. (2018) e Lima et al. (2019). O primeiro aborda o uso das LRCNs para classificação dinâmica de sinais de Libras, mas não avalia a influência do dropout sobre o resultado final, além de utilizar uma câmera Intel RealSense. O último aborda a classificação estática do estado da mão (fechada/aberta) para a teleoperação, utilizando apenas CNNs com imagens obtidas de um Kinect V1.

Nesse cenário, o presente trabalho utiliza LRCNs para a classificação dinâmica da mão (aberta/fechada) voltada para a teleoperação, além de incluir a avaliação sobre a influência do dropout e do tamanho da sequência de entrada. Além disso, utilizamos como sensor de profundidade o Kinect V2. Como baseline, também optamos por implementar um classificador estático baseado em CNN a fins de comparação. O processo de implementação dos modelos segue as seguintes etapas:

1. Construir a base de dados apropriada, levando em consideração o tipo de modelo utilizado;
2. Definir um modelo parametrizado, com base nos trabalhos similares;
3. Definir o espaço de busca dos parâmetros e otimizá-los através de uma busca em grade;
4. Fazer o registro e a análise dos dados de treinamento;

4.1.1 Construção da Base de Dados

As imagens de profundidade captadas pelo Kinect V2 passam por uma etapa de pré-processamento antes de serem utilizadas. O primeiro passo é extrair a região de interesse contendo a mão direita do usuário. Então, a imagem é filtrada baseando-se na informação de profundidade para que os elementos do background sejam removidos. Por fim, a imagem passa por um processo de *threshold* para que ela se torne preto-e-branco (binária).

O pré-processamento discutido é utilizado tanto na aquisição do dataset de treinamento, quanto nos experimentos de validação. Sua execução baseia-se nas seguintes informações disponibilizadas pelo Kinect SDK:

- a imagem de profundidade original definida por $d(x, y) : \mathbb{R}^2 \mapsto \mathbb{R}$;
- as posições 3D das juntas do usuário, com relação ao sistema de coordenadas da câmera, definidas por $j_i \in \mathbb{R}^3$ onde i é o índice da junta;
- as projeções das juntas sobre o plano da imagem 2D de profundidade, em pixels, definidas por $p_i \in U$, onde i é o índice da junta.

Seja rh o índice da junta representando o centro da mão do usuário. No Kinect SDK V2, o índice da junta é definido como $rh = 11$. A imagem de profundidade bidimensional d é recortada na região quadrada $\mathcal{H}$, cujo centro coincide com p_{rh}. O tamanho do lado dessa região quadrada, em pixels, é aproximado de forma linear, sendo inversamente proporcional à distância ${}^z j_{rh}$ da mão do usuário até a câmera. É possível, então, definirmos o tamanho de retângulo b em pixels como segue:

$$b(j_{rh}) = K \cdot \frac{1}{{}^z j_{rh}} \tag{4.1}$$

onde ${}^z j_{rh}$ é a componente z da posição da junta j_{rh}, e K é uma constante de proporcionalidade que pode ser calculada empiricamente. Em nosso trabalho, definimos um retângulo de 50×50 pixels quando o usuário se encontra a 1.2 metros do Kinect V2, portanto $K = 50 \cdot 1.2 = 60$. Vale ressaltar que a região extraída deve estar dentro dos limites da imagem, portanto deve-se respeitar os limites de máximo e mínimo através da seguinte relação:

$$\begin{aligned}
\mathcal{H}_{x_{\text{min}}} &= \max(0, \mathcal{H}_{x_{\text{max}}}) \\
\mathcal{H}_{x_{\text{max}}} &= \min(d_{\text{width}}, \mathcal{H}_{x_{\text{max}}}) \\
\mathcal{H}_{y_{\text{min}}} &= \max(0, \mathcal{H}_{y_{\text{min}}}) \\
\mathcal{H}_{y_{\text{max}}} &= \min(d_{\text{height}}, \mathcal{H}_{y_{\text{max}}})
\end{aligned} \tag{4.2}$$

Os casos de extrapolação, as Equações 4.2 garantem a estabilidade do sistema. Entretanto, não é possível garantir a boa qualidade da região extraída caso a mão do usuário se aproxima dos limites laterais da visão da câmera, ou chegue muito perto. Nesses casos, o melhor a fazer é descartar a imagem deteriorada.

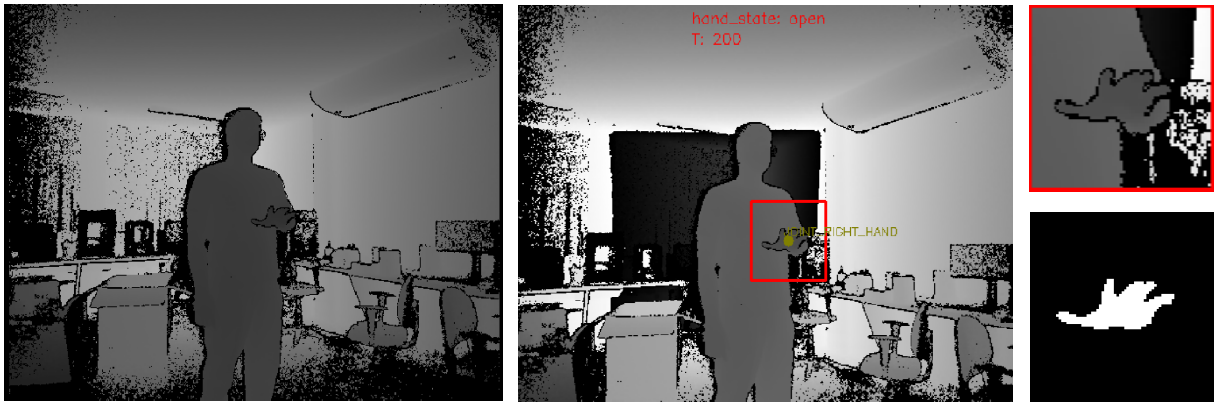

Figura 31 – Etapa de pré-processamento das imagens de profundidade para a construção do dataset de treinamento dos classificadores. À esquerda, vemos a imagem d. Ao centro, vamos a demarcação da região $\mathcal{H}$. À direita, vemos a imagem da mão extraída d', na primeira linha, e sua binarização d'' da segunda linha. Fonte: Autor.

Após extraída região $\mathcal{H}$ da imagem de profundidade d, temos o recorte $d' : \mathcal{H} \mapsto \mathbb{R}$, possivelmente contendo ruídos e partes do background inclusas da extração. Assim, é gerada uma imagem binária d'' através de um filtro limiar definido como:

$$d''(x,y) = \begin{cases} 1, & d'(x,y) \leq D_{\text{min}} + T \\ 0, & d'(x,y) > D_{\text{min}} + T \end{cases} \tag{4.3}$$

onde T é um valor limiar definido empiricamente como 0.2 metros, e $D_{\text{min}} = \min d'(x,y)$ é ponto mais próximo da câmera assumindo que o usuário está com a mão apontada para frente. Após aplicada a binarização, a imagem é escalada para assumir o tamanho de entrada dos classificadores que é de $50x50$ pixels. O resultado deste processo de pré-processamento pode ser visualizado na Figura 31.

Após definir a etapa do pré-processamento, o próximo passo consistiu em definir as diretrizes de captura. Para isso, foi desenvolvida um software que contava com uma Interface Gráfica de Usuário (do inglês *Graphical User Interface*, ou GUI) para auxiliar no processo de captura e anotação, utilizando PyQt5. Durante o processo, os próprios voluntários eram responsáveis pela anotação das imagens, processo que fora parcialmente automatizado. Enquanto a mão direita do usuário era gravada, a mão esquerda segurava um controle remoto (ex.: um mouse wireless). O software de captura monitorava a ação do controle que, ao ser pressionado, alternava o estado da mão associado à imagem capturada no momento. Enquanto não fosse pressionado o controle, o estado associado à captura atual era o mesmo da última captura.

Essa abordagem de anotação semi-automática traz uma certa heterogeneidade ao processo, pois cada usuário tem sua própria noção de quando o estado da mão deve ser alternado. Nossa hipótese era a de que a heterogeneidade poderia contribuir para uma maior robustez da rede. A única diretriz que era repassada para os voluntários antes da captura era a seguinte: "pressione o controle durante a transição entre os estados". Uma imagem sintética da mão de um

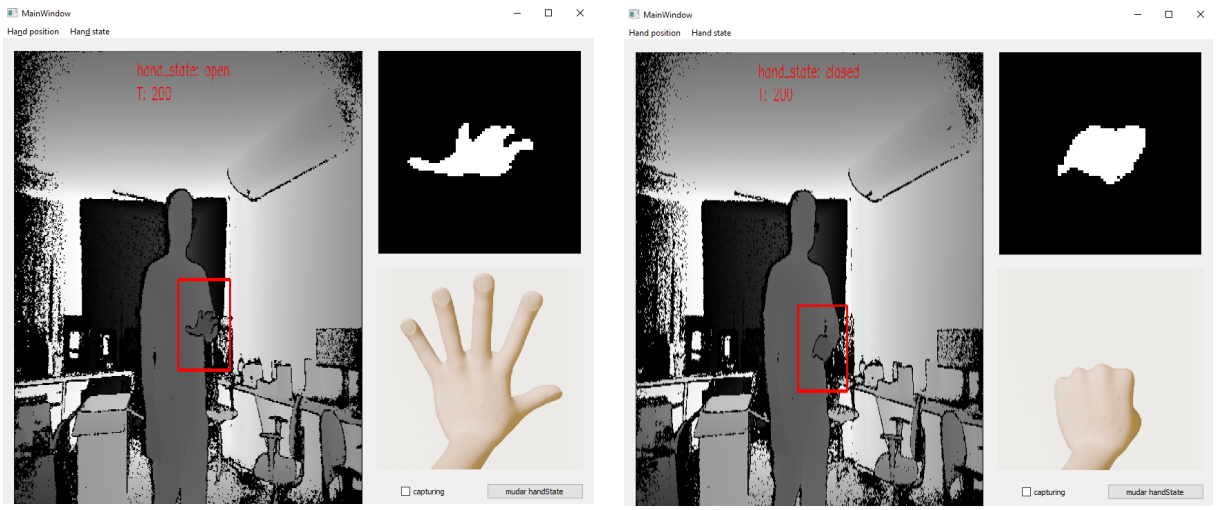

Figura 32 – GUI de captura do banco de dados implementada para o trabalho. Tanto o voluntário sendo gravado quanto o pesquisador podem alternar o estado da mão do usuário. A numeração dos episódios e das imagens de profundidade é realizada de forma automática. Fonte: Autor.

avatar era apresentada na GUI para que o voluntário pudesse reapidamente checar qual o estado atual da anotação.

A captura das imagens era realizada por episódios. Cada episódio consistia na repetição de uma sequência de movimentos bem definidos. Durante a execução dos movimentos, uma sequência de imagens ordenadas que iniciam com o índice zero eram salvas no disco, com o padrão de nomeclatura "im_${img_index}_${img_annotation}.pgm"Ṫal padrão continha o índice da imagem internamente ao episódio, de modo que sua anotação fosse de fácil acesso. O formato PGM tem um lado bom que é a leveza e a rapidez no salvamento. O lado ruim é a falta de suporte nativo no Windows, decorrendo também na falta de pré-visualização pelo gerenciador de arquivos.

A definição dos movimentos requisitados aos voluntários foi realizada visando refletir os movimentos das tarefas do experimento final. O intuito era que a distribuição de imagens de profundidade fosse similar à dos experimentos. Com isso em mente, a seguinte sequência de ações era solicitada durante um episódio:

1. Fechar a mão;
2. Realizar o movimento de um lado a outro com a mão fechada;
3. Abrir a mão;
4. Realizar o movimento de volta ao lado inicial com a mão aberta.

Cada usuário executava dois episódios, alternando o lado inicial entre eles. Embora tal simetria pudesse ser alcançada via software, achamos que o modo escolhido pudesse trazer

nuanças físicas próprias aos usuários, além de uma maior heterogeneidade para a base. A Figura 32 demonstra a GUI do software auxiliar de captura das imagens de profundidade. Tanto o voluntário quando o pesquisador são capazes de setar o estado atual da mão do usuário, bem como iniciar e finalizar o processo de captura de um episódio. A numeração das imagens e dos episódios é realizada de forma automática: na inicialização, o software realiza uma busca na pasta de saída para procurar episódios antigos, definindo automaticamente o índice atual.

4.1.2 Desenvolvimento dos Classificadores

No presente trabalho, foram implementados dois classificadores para o dataset discutido na seção anterior, sendo um classificador baseado em CNN e outro baseado em LRCN. A ideia foi se basear nos trabalhos de Lima et al. (2019) e Amaral et al. (2018), dessa vez explorando a relação temporal entre as imagens de profundidade captadas da mão dos usuários. Nessa seção discutiremos os pormenores da implementação, que foi realizada em Python utilizando a biblioteca Keras. Ambos os classificadores são binários, apresentando uma camada densa de saída com 2 neurônios e ativação softmax representando a probabilidade da mão estar aberta ou fechada.

Para a divisão da base de dados entre treinamento e validação, utilizamos a validação cruzada estratificada com 2 partições. Para tomar proveito do treinamento na placa de vídeo, utilizamos *batches* de tamanho 128, com o número máximo de épocas igual a 300. Foi utilizado um processo de salvamento dos pesos da rede sempre que a acurácia de validação melhorava. Além disso, caso o modelo não apresentasse melhora no erro de validação durante as últimas 30 épocas, o treinamento era encerrado prematuramente antes das 300 épocas máximas.

Em ambos os classificadores foi realizada uma busca em grade para encontrar os hiper-parâmetros que melhor se adequariam ao problema. A busca em grade consiste instanciar o modelo várias vezes, cada uma delas com um conjunto de hiper-parâmetros diferentes pertencentes a um espaço de busca bem definido. A função de perda utilizada para o treinamento de ambos foi a entropia cruzada categórica e o otimizador foi o Adam. Os hiper-parâmetros específicos de cada modelo serão discutidos nos parágrafos a seguir.

O primeiro classificador tinha o objetivo de servir como um baseline. Era um classificador estático, baseado em CNN, que recebia imagens de profundidade unitárias como entrada. Como não há necessidade de relação sequencial no conjunto de entrada, bastou fazer a varredura da da pasta contendo os episódios para formar o vetor de dados de entrada, carregando as imagens encontradas e suas classes. O algoritmo detalhado de carregamento está presente no Anexo A. Os hiper-parâmetros utilizados na busca em grade foram:

- o tamanho $F \in \{3 \times 3, 5 \times 5\}$ dos kernels dos filtros convolucionais da entrada, que podia ser $3x3$ ou $5x5$;
- O número $S \in \{8, 16, 32, 64, 128\}$ de filtros convolucionais em cada camada;

- O tamanho $D \in [d_1, d_2]$ onde $d_1 \in \{80, 120, 160\}$ e $d_2 \in \{0, 80, 120, 160\}$ das duas camadas escondidas. $d_2 = 0$ indica que há apenas uma camada escondida d_1.
- A probabilidade $P \in \{0, 0.5\}$ de dropout em ambas as camadas convolucionais e escondidas.

O segundo classificador representa parte da inovação do presente trabalho. Ele constitui um classificador dinâmico baseado em LSTM, que recebe uma sequência de imagens como dados de entrada. O carregamento do dados consiste na varredura dos episódios e no janelamento de tamanho fixo das imagens pertencentes a cada episódio para a formação das sequências. O algoritmo detalhado do carregamento da base e do janelamento para a LRCN está presente no Anexo B. É importante ressaltar que, para cada sequência formada pelo janelamento, a classe associada à sequência é a classe do último elemento desta sequência. As camadas iniciais da rede LRCN correspondem às mesmas da rede CNN, com a diferença de agora serem distribuídas no tempo. Outra mudança é a troca das camadas escondidas pelas camadas LSTM. Considerando que o sistema de classificação é executado a 30 frames por segundo, escolhemos um tamanho de janelamento de $K = 15$ para a formação do vetor de entrada de modo a permitir uma relação temporal de meio segundo internamente à sequência. Os hiper-parâmetros utilizados na busca em grade para a arquitetura LRCN foram:

- o tamanho $F \in \{3 \times 3, 5 \times 5\}$ dos kernels dos filtros convolucionais da entrada, que podia ser $3x3$ ou $5x5$;
- O número $S \in \{8, 16, 32, 64, 128\}$ de filtros convolucionais em cada camada;
- O número de neurônios $D \in \{50, 100, 250, 500\}$ da camada única LSTM.
- A probabilidade $P \in \{0, 0.5\}$ de dropout em ambas as camadas convolucionais e escondida.

Após treinada a LSTM, testamos a influência da alteração do tamanho do janelamento utilizando $K = \{5, 10, 15\}$ para a melhor arquitetura encontrada. Os resultados dos treinamentos, bem como a melhor arquitetura para a CNN e para a LSTM estão presentes na Seção 5.

4.2 Mapeamento de Posição

4.2.1 Construção do Modelo TPS

Nesta seção discutiremos sobre o mapeamento ponto-a-ponto não-linear de posição entre a mão do usuário e o efetuador do robô, baseado nas Thin-Plate Splines. A proposta visa tratar dos seguintes problemas inerentes a cada usuário no processo de teleoperação: (1) diferentes limites cinemáticos; (2) possível falta de simetria em relação ao eixo vertical; (3) adequação a diferentes espaços de trabalho dos robôs; (4) conceito de conforto. Portanto este trabalho é

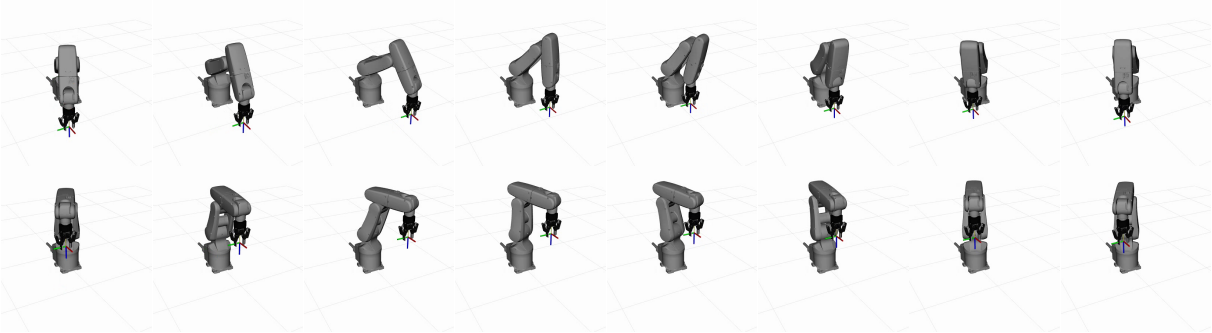

Figura 33 – Conjunto das 16 posições do efetuador no modelo espelhado, utilizadas como pontos alvo para a TPS. Os pontos foram organizados em dois níveis de altura, sendo escolhido de modo a formar um polígono representando os limites do espaço de trabalho do robô para as tarefas experimentais. Fonte: Autor.

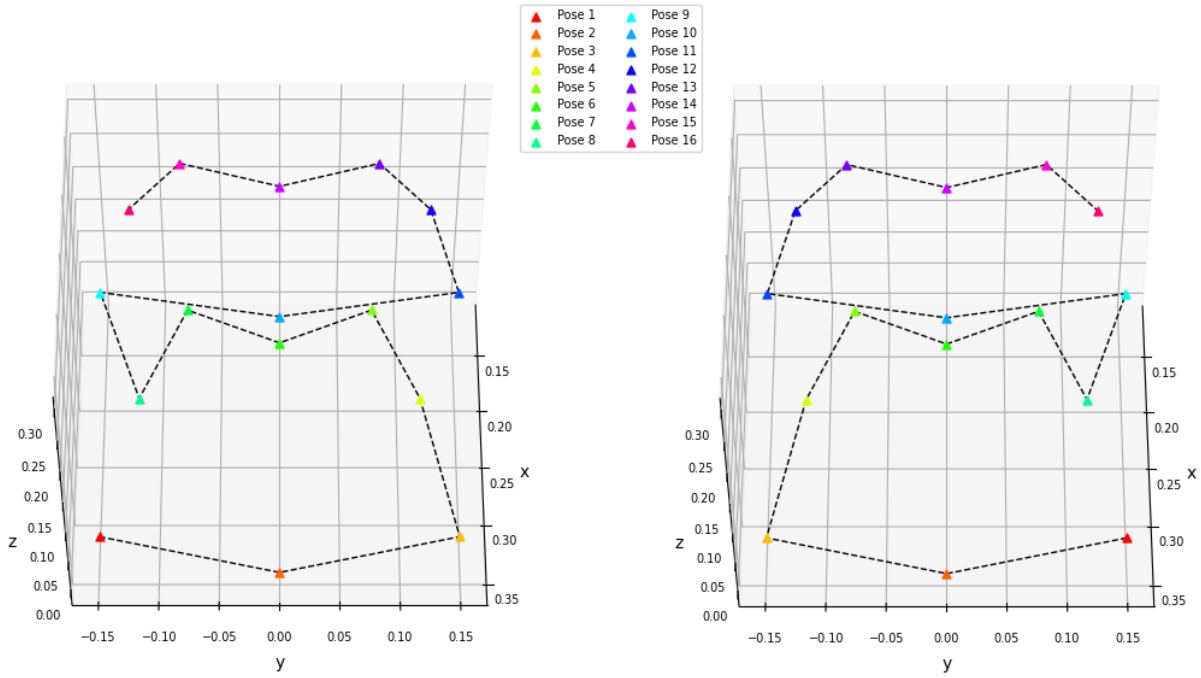

Figura 34 – Conjunto dos 16 pontos escolhidos nos limites do espaço de trabalho do manipulador, em dois níveis de altura diferentes. Do lado esquerdo, temos a versão espelhada dos pontos coletados (que foi o modelo utilizado durante os experimentos). Do lado direito, temos a versão normal dos pontos coletados. Fonte: Autor.

uma extensão de Lima et al. (2019), cuja solução apresentada não tratava dos problemas aqui discutidos. Outro ponto importante é que o mapeamento proposto dispensa a parametrização manual do modelo, transformando-a num processo prático de captura de poses.

Nós propomos a computação da função de mapeamento da TPS baseando-se na correspondência 1-para-1 entre poses do usuário e poses do robô. De acordo com a teoria discutida na Seção 2.2.6, tal mapeamento apresenta duas componentes: uma componente linear representada por uma função afim e uma componente não-linear que se baseia na minimização da energia de deformação. Tal fato gera uma função de transformação exata, mas suave, entre os espaços

	espelhada			direta		
índice	x	y	z	x	y	z
1	0.32	-0.15	0	0.32	0.15	0
2	0.35	0	0	0.35	0	0
3	0.32	0.15	0	0.32	-0.15	0
4	0.2	0.12	0	0.2	-0.12	0
5	0.12	0.08	0	0.12	-0.08	0
6	0.15	0	0	0.15	0	0
7	0.12	-0.08	0	0.12	0.08	0
8	0.2	-0.12	0	0.2	0.12	0
9	0.27	-0.14	0.3	0.27	0.14	0.3
10	0.29	0	0.3	0.29	0	0.3
11	0.27	0.14	0.3	0.27	-0.14	0.3
12	0.2	0.12	0.3	0.2	-0.12	0.3
13	0.16	0.08	0.3	0.16	-0.08	0.3
14	0.18	0	0.3	0.18	0	0.3
15	0.16	-0.08	0.3	0.16	0.08	0.3
16	0.2	-0.12	0.3	0.2	0.12	0.3

Tabela 6 – Valores dos 16 pontos na versão espelhada e normal do modelo.

de trabalho do usuário e do manipulador robótico. Nossa hipótese é que, através da escolha de alguns pontos chaves relevantes nos limites do espaço de trabalho, as demais poses seriam suavemente e intuitivamente mapeadas através das propriedades de interpolação da TPS.

Nós definimos empiricamente um conjunto de 16 posições Cartesianas no espaço do manipulador robótico para serem mapeadas para suas respectivas posições da mão humana. Para evitar extrapolação, tais posições-chave foram escolhidas nos limites do espaço de trabalho do robô para as tarefas definidas, em dois níveis de altura diferentes (8 posições por nível). Para implementar o modo de controle espelhado descrito em Lima et al. (2019), utilizamos outro paradigma. Como nossa abordagem consiste num mapeamento direto entre posições-chave, foi necessário definir dois conjuntos ordenados de pontos-alvo diferentes: um convencional (quando o operador está no ponto de visão do robô) e um espelhado (quando o operador está de frente para o robô, imaginando um espelho). Embora neste caso específico haja apenas uma inversão de sinal no eixo y, a forma utilizada de mudar os pontos alvo foi feita propositalmente para evidenciar a possibilidade de gerar diversos modelos utilizando apenas uma captura do usuário. A listagem detalhada dos dois conjuntos está presente na Tabela 6. A representação visual dos mesmos está disponível nas Figuras 34 e 33.

De posse dos pontos alvo da TPS, o próximo passo envolve a captura dos pontos de referência sobre o espaço de trabalho do usuário. Primeiro, foi realizada uma transformação de coordenadas, mapeando do frame da câmera para um frame localizado no ombro direito do usuário. As seguintes juntas providas pelo Kinect SDK 2.0 (apresentadas na Figura 5) do esqueleto humano foram utilizadas para formar os vetores ortonormais da nova base: ombro direito $rs = 8$, cotovelo direito $re = 9$, mão direita $rh = 7$, ombro esquerdo $ls = 4$, centro entre

os ombros $sc = 20$, e centro da espinha $sp = 1$.

A base ortonormal no ombro do usuário é formada por três vetores relacionados ao plano do torso. Dois deles representam as direções vertical e horizontal. Mais especificamente, nós computamos os vetores unitários $\hat{v}$, $\hat{h}$ e $\hat{n}$ como segue:

$$\hat{v} = \frac{j_{sc} - j_{sp}}{\|j_{sc} - j_{sp}\|} \tag{4.4}$$

$$\hat{h} = \frac{j_{ls} - j_{rs}}{\|j_{ls} - j_{rs}\|} \tag{4.5}$$

$$\hat{n} = \frac{\hat{h} \times \hat{v}}{\|\hat{h} \times \hat{v}\|} \tag{4.6}$$

O vetor u que representa a posição da mão direita em relação ao sistema de coordenadas da câmera é:

$$\vec{u} = \vec{j_{rh}} - \vec{j_{rs}} \tag{4.7}$$

Por fim, o vetor $\vec{u_{rs}}$ em relação ao sistema de coordenadas do ombro e, portanto, invariante à posição e orientação do usuário, é definido por:

$$\vec{u_{rs}} = \vec{u} \cdot \left[\hat{h}, \hat{v}, \hat{n}\right] = [u_x, u_y, u_z] \cdot \begin{bmatrix} h_x & v_x & n_x \\ h_y & v_y & n_y \\ h_z & v_z & n_z \end{bmatrix} \tag{4.8}$$

O vetor $\vec{u_{rs}}$, representando a posição da mão direita do usuário com relação ao seu ombro direito, é a entrada do modelo TPS. A saída do modelo é representada pela posição $\vec{P_G}$. Os vetores de entrada e saída da TPS podem ser visualizados na Figura 35.

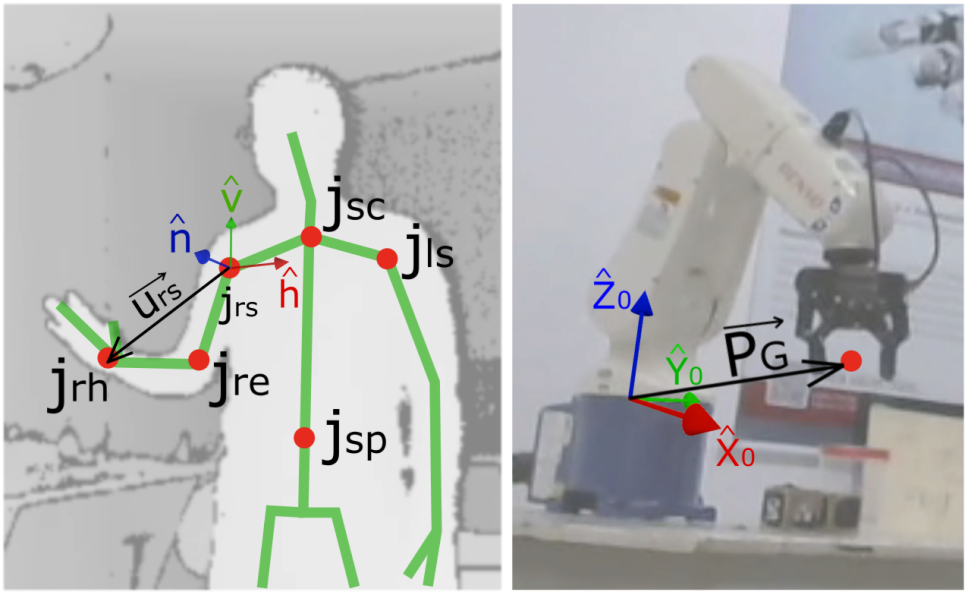

Figura 35 – Vetor $\vec{u_{rs}}$ à direita representa a entrada do modelo TPS. O vetor $\vec{P_G}$ à esquerda representa a saída do modelo. Fonte: Autor.

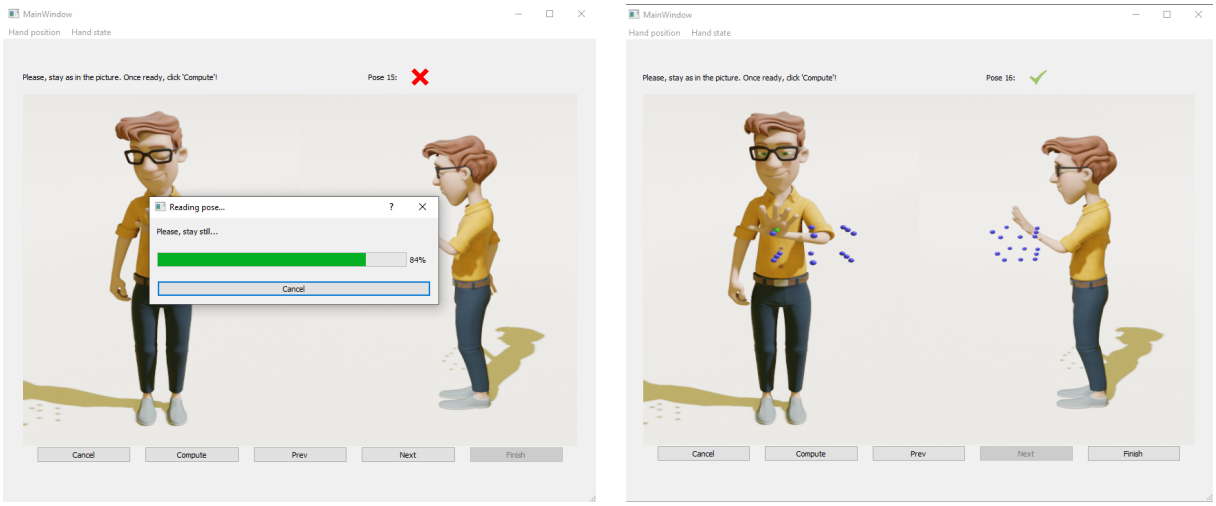

Figura 36 – Ilustração da GUI de treinamento desenvolvida para auxiliar na geração dos modelos TPS.

Figura 37 – Todas as 16 poses utilizadas como referência para o treinamento do modelo TPS. Para cada pose do avatar na colagem, uma pose correspondente do usuário é mostrada abaixo. Ambos os braços do avatar e do usuário estão espelhados. Um conjunto de 16 esferas com as posições ideais da mão do usuário é utilizado para a referência da mão do avatar. Esfera verde significa posição atual, esfera azul significa as demais posições.

Para auxiliar na captura dos 16 valores de $\vec{u_{rs}}$, uma ferramenta de treinamento de modelo é desenvolvida no presente trabalho, utilizando o ROS, Kinect SDK 2.0 e o PyQt. Foi realizada uma implementação em C++ que utilizava o Kinect SDK V2, fazia as transformações nos sistemas de coordenadas e publicava os resultados em um tópico do ROS. Também foi utilizado um avatar 3D de nome Vincent[1] para demonstração das poses espelhadas aproximadas que o usuário deveria assumir em cada um dos 16 pontos. Visando reduzir o efeito do ruído sobre os dados captados, um número $n = 30$ de amostras de $\vec{u_{rs}}$ era capturada por um intervalo $t = 2$ segundos para cada pose, período após o qual a mediana entre essas amostras era armazenada.

[1] Vincent, o avatar utilizado como referência no Blender3D: <https://cloud.blender.org/p/characters/5718a967c379cf04929a4247>

Após as 16 medianas serem captadas, era possível visualizar a distribuição espacial dos pontos captados. Caso algum dos pontos apresentasse uma localização imprópria, era possível captar apenas o ponto problemático novamente (conservando os demais já existentes).

Ao constatar que todos os pontos estavam bem distribuídos, os dois modelos (direto e espelhado) eram gerados automaticamente pelo sistema a partir dos pontos de referência do usuário P e dos pontos alvo do manipulador robótico Q. Tais modelos eram gerados resolvendo o sistema linear da Equação 2.28 para encontrar a matriz de pesos descrita na Equação 2.29. Essas matrizes eram persistidas no disco para poderem ser utilizadas quando desejado. É possível visualizar o GUI de treinamento do modelo tps na Figura 36. A equivalência entre as 16 poses do usuário e a representação do avatar pode ser visualizada na Figura 37. Uma análise estatística sobre o tempo necessário para o treinamento de um modelo e para a transformação de um ponto (após o modelo treinado) foi realizado sobre 1000 (mil) iterações para cada análise. Os resultados estatísticos da análise estão presentes na Tabela 7.

	median (ms)	mean (ms)	std (ms)	min (ms)	max (ms)	iters./segundo
treinamento	2	2,32	0,54	1	5	431
mapeamento	0	0,34	0,47	0	1	2941

Tabela 7 – Análise estatística dos tempos de treinamento e mapeamento para o modelo TPS. Em determinadas ocasiões, o tempo de mapeamento se mostrou irrisório.

Durante as poses, era pedido que o usuário imaginasse uma gaiola à sua frente, com o centro da gaiola à frente do seu ombro direito. A representação visual da gaiola era mostrada através do avatar, onde a esfera verde indicava a posição atual que a mão deveria assumir. Note que essa gaiola serve apenas para referência relativa entre as posições, pois cada usuário define a região mais confortável para manter a mão durante os experimentos práticos. Uma solicitação era que a gaiola imaginária ficasse não muito próxima do corpo para que a região de trabalho não prejudicasse os classificadores na hora da segmentação da mão.

4.2.2 Validação do Mapeamento TPS

Para validar o modelo de mapeamento proposto, nós executamos testes de usabilidade com relação à experiência do usuário. Ambos os modelos de mapeamento foram analisados, de forma que o experimento se dividiu em duas seções: uma seção utilizando o modelo linear *naive* (NA-PM) proposto por Lima et al. (2019); e outra seção utilizando o modelo não-linear baseado em Thin-Plate Spline (TPS-PM). Nesse cenário, definimos um conjunto de 5 tarefas no âmbito manipulação dos objetos com o braço robótico para serem executadas pelo usuário. Os seguintes aspectos foram avaliados para cada abordagem:

- *Realização do objetivo.* Os usuários foram capazes de realizar o objetivo das tarefas?
- *Conforto*. Os usuários acharam alguma abordagem mais confortável em determinada tarefa?

- *Curva de Aprendizagem*. Os usuários foram capazes de de lidar com a dificuldade incremental apresentada pelas tarefas?

Ao início de cada seção, a calibração dos usuários era realizada. Para a seção NA-PM, a calibração consistia no usuário estendendo seu braço direito para que seu comprimento fosse medido. Para a seção TPS-PM era necessário capturar as poses discutidas na Seção 4.2.1. Um supervisor examinava visualmente os resultados das 16 poses coletadas para checar se havia alguma distribuição espacial com inconformidades. Exemplos de inconformidades são dois vértices muito juntos ou em posição relativa não esperada. Nesses casos, era necessário captar o ponto novamente ou reiniciar a calibração como um todo.

As seções envolviam a interação com os seguintes elementos:

- dois cubos de comprimento igual a 5 cm;
- uma plataforma cilíndrica com um fundo aberto marcado de vermelho com 10 cm de diâmetro e uma altura de 12 cm. O topo da plataforma possui 7,5 cm de diâmetro. Quando a plataforma era posicionada de cabeça para baixo, ele virava uma cesta.

De forma a quantificar o tempo de cada tarefa, nós utilizamos o conceito de movimentos atômicos. Um movimento atômico é um deslocamento contínuo do efetuador do robô, com tempos de início e fim bem definidos. Alterações no estado do efetuador (movimento de abertura/fechamento, também chamado agarrar/soltar) também são considerados movimentos atômicos. Essa abordagem é similar ao conceito de *sub-tasks* definido por Lipton, Fay e Rus (2017). Além disso, o espaço de trabalho das tarefas foi dividido em 4 regiões, nomeadas de A a D. Nesse cenário, as especificação das tarefas é como segue:

- *Levar um cubo da região B para a região A*. É uma tarefa inicial para testar a habilidade do usuário em mover com sucesso um cubo para outra região. Pode ser alcançada utilizando um mínimo de 4 movimentos atômicos: *home*-para-B, agarrar, B-para-A, soltar.
- *Alternar cubos entre as regiões B e A, usando a região C como auxiliar*. A proposta desta tarefa é medir o incremento nas habilidades do usuário desde a primeira tarefa. Pode ser realizada em um mínimo de 12 movimentos atômicos, todos eles com a mesma dificuldade da Tarefa 1, os quais: *home*-para-A, agarrar, A-para-C, soltar, C-para-B, agarrar, B-para-A, A-para-C, agarrar, C-para-B, soltar.
- *Trazer os cubos das regiões A e B para a cesta localizada em D*. Esta tarefa introduz a necessidade de movimento em maiores alturas. Os usuários também precisam levar em consideração o desvio de obstáculos representado pela cesta durante o movimento de subida da garra. Habilidades avaliadas nessa tarefa incluem a facilidade de uso próximo aos limites do espaço de trabalho do robô, uma vez que a região D é a mais afastada do centro.

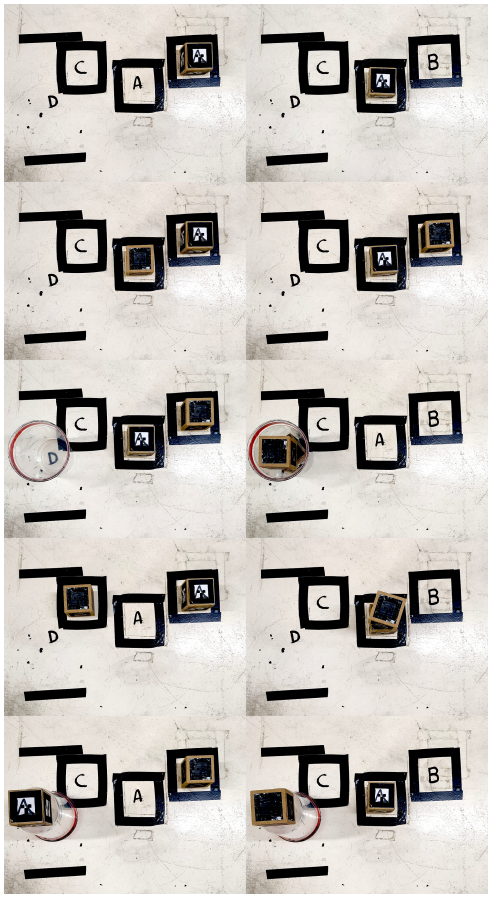

Figura 38 – Configurações iniciais (coluna esquerda) e finais (coluna direita) para cara tarefa executada pelo usuário. Cada linha representa uma tarefa, variando de 1 a 5, respectivamente.

Também pode ser executado em um mínimo de 8 movimentos atômicos: *home*-para-A, agarrar, A-para-cesta, soltar, cesta-para-B, agarrar, B-para-cesta, soltar.

- *Empilhar cubos na região A*. Esta tarefa introduz um novo componente na avaliação, que é a habilidade de ajuste fino na posição do cubo. As tarefas anteriores tinham uma aceitação menos restrita com relação a onde os cubos poderiam ser colocados. Também pode ser realizado em 8 movimentos atômicos: *home*-para-C, agarrar, C-para-A, soltar, A-para-B, agarrar, B-para-A, soltar.

- *Mover um cubo da plataforma para A, e de B para a plataforma*. Habilidades requeridas são controle próximos aos limites do espaço de trabalho e a habilidade de ajuste fino na posição do cubo sobre a plataforma. Também pode ser executado em um mínimo de 8 movimentos atômicos: *home*-para-plataforma, agarrar, plataforma-para-A, soltar, A-para-B, agarrar, B-para-plataforma, soltar.

Ambas as configurações iniciais e fiais de cada uma das tarefas discutidas estão presentes na Figura 38. De forma a medir o tempo total para a finalização de cada tarefa, as seguintes condições foram definidas:

1. o cronômetro inicia tão logo quanto o controle do usuário é ativado;
2. o cronômetro para assim que o objetivo final é atingido e os objetos de interação permanecem imóveis;
3. Um sucesso ocorre quando a configuração final é atingida e os objetos permanecem dentro das áreas estabelecidas (não tocando a região fora dos marcadores de cor preta);
4. Uma falha ocorre quando o experimento entra num estado em que não é possível atingir a configuração final sem intervenção humana. Por exemplo, quando um cubo sai do espaço de trabalho do robô.

Antes de iniciar cada sessão de experimentos, era dado um período de *playground* ou *sand-box* para os usuários (2 minutos) de modo que eles obtivessem certa familiaridade com cada modelo. Após finalizado o último experimento de cada sessão, os usuários eram solicitados a responder duas perguntas baseadas no método Likert (LIKERT, 1932), onde um valor inteiro entre 1 e 5 era solicitado sobre o entendimento dos usuários a respeito da experiência. As perguntas foram:

- *(Q1) O quão suave foi o movimento do braço robótico durante o controle?* Esta questão avalia o quão confortável estavam os usuários com relação ao movimento do braço robótico. É possível ter um entendimento matemático sobre a qeustão. Seja $\frac{dq}{dp}$ a derivada do movimento do efetuador com relação ao movimento da mão do usuário. Por movimento suave, entende-se que $\frac{dq}{dp} \approx 1$. Por movimento brusco, entende-se que $\frac{dq}{dp} \gg 1$.
- *(Q2) O quão difícil foi o movimento próximo dos limites do espaço de trabalho?* Essa questão avalia a qualidade do mapeamento da posição quando o efetuador do robô está próximo dos limites do espaço de trabalho do experimento. Um bom mapeamento apresentaria uma dificuldade consistentemente baixa, independentemente da região de operação, com o esforço geral do usuário sendo o mínimo possível.

Após ambas as sessões com cada modelo de mapeamento serem finalizadas e suas respectivas questões *Likert* serem respondidas, os usuários eram submetidos a um conjunto de 3 questões discursivas a fim de fazer uma comparação entre as sessões:

- *(Q3) Em sua opinião, quais foram as vantagens de cada mapeamento?*
- *(Q4) Quais foram os desafios encontrados durante os experimentos?*
- *(Q5) Você teria alguma sugestão de melhoria para o sistema? Em caso afirmativo, qual seria?*

Todas as respostas dos participantes foram coletadas através de um formulário online que incluía o consentimento de uso das imagens e das respostas dos participantes para fins exclusivos de pesquisa científica, sem quaisquer fins lucrativos. Além dos tempos coletados para ambas as abordagens de mapeamento, foi também realizado um Teste T de Student (STUDENT, 1908) utilizando como variável testada a razão $\frac{T}{n_a}$, onde T é o tempo total e n_a é o número de movimentos atômicos.

4.3 Sistema de Tele-manipulação Robótica

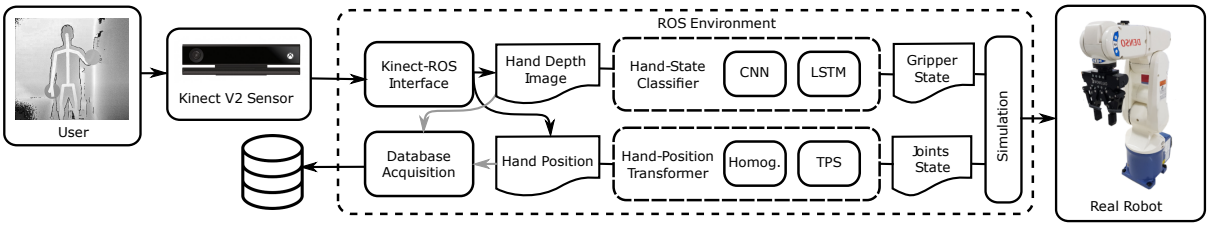

Figura 39 – Diagrama da arquitetura do sistema. O Kinect V2 é utilizado como sensor de profundidade. Os módulos rodam de maneira distribuída, utilizado o ROS como framework de comunicação. Classificação e mapeamento de posição rodam de forma independente. A comunicação com robô real é realizada de forma desacoplada na qual as juntas passam por duas camadas de visualização e checagem de limites (uma no ROS e outra no Simulink) antes de chegarem ao robô real.

Esta seção descreve os detalhes sobre o ambiente de tele-manipulação como um todo desenvolvido com o ROS e o Matlab. Todos os módulos foram desenvolvidos utilizando o Windows 10, Intel Core i5, 16 GB RAM DD3 e uma Nvidia GeForce GTX 2080 TI 12 GB. Para os experimentos, foi utilizado o manipulador robótico Denso VP6242 junto ao efetuador Robotiq 2F-85.

Um diagrama geral dos módulos discutidos está presente na Figura 39. O usuário junto e sensor de captura de dados são representados à esquerda. Ao centro temos a arquitetura central de processamento no ROS. O robô real junto aos seus módulos de controle estão representados à direita. Os dados são passados da esquerda para a direita, exceto quando a persistência dos dados é desejada. Todo o sistema é capaz de rodar a uma taxa de 30 FPS.

Todo o sistema foi desenvolvido utilizando dois computadores. Um dos computadores era o responsável exclusivamente pela comunicação direta com os controladores do braço robótico e do manipulador através do Matlab 2015. Nele estavam presentes diretrizes de controle tais como o controle PID malha-fechada utilizado para o controle das juntas do robô. O outro computador era responsável pelos módulos de captura de dados, classificação e simulação. A comunicação entre os computadores era realizada através do protocolo TCP, contando como um roteador exclusivo para a rede interna entre os dois. Essa foi a melhor alternativa encontrada para a comunicação, pois permitia o máximo desacoplamento entre ambos.

Os módulos do nosso sistema se comunicam de maneira distribuída através da arquitetura de *publisher/subscriber* presente no ROS. Um modelo visual do manipulador robótico foi desenvolvido utilizando as extensões URDF e XACRO discutidas em 2.3.7. De posse do modelo, o MoveIT foi utilizado para o serviço de cinemática inversa, necessário para computação do vetor de juntas a ser enviado pro Matlab por TCP. Dentre os vários algoritmos prontos de cinemática oferecidos pelo MoveIT, o presente trabalho utilizou o plugin padrão KDL, um plugin numérico desenvolvido pelo grupo Orocos[2]. A resolução utilizada foi de 0.5 (meio) milímetro.

Vale ressaltar que também implementamos o modelo de mapeamento de posição proposto por (LIMA et al., 2019) que utiliza transformações parametrizadas unicamente lineares. Tal modelo foi aqui denominado *naive* (ou NA-PM, ou simplesmente NA). Assim, desenvolvemos uma GUI que servia de página principal para que o usuário pudesse escolher entre os modos de mapeamento de posição (TPS ou NA), além de executar a chamada da função de coleta de dataset ou de escolha do modelo de classificação (CNN ou LSTM). De modo geral, é possível representar o sistema como um todo através dos seguintes módulos:

- *Módulo de interfaceamento com o Kinect V2 para ROS*. Desenvolvido em C++, é o responsável por fazer a leitura dos fluxos de imagem RGB, imagem de profundidade e posições das juntas do usuário. Também responsável por fazer a transformação de coordenadas da mão do usuário do frame da câmera para o frame do ombro. Os dados discutidos são publicados em seus respectivos tópicos no framework ROS.

- *Módulo de aquisição do banco de dados das imagens de profundidade binarizadas*. Responsável por fazer a captura do banco de dados de imagens de profundidade, realizando a extração da mão do usuário discutida na Seção 4.1.1.

- *Módulo de classificação da mão do usuário*. Responsável por carregar um dos classificadores treinados de acordo com o discutido na Seção 4.1.2. Apenas um modelo pode estar ativo por vez, sendo ele o CNN ou o LRCN.

- *Módulo de mapeamento da posição da mão do usuário*. Responsável por carregar um dos modelos de mapeamento ponto-a-ponto discutidos na Seção 4.2. Apenas um modelo pode estar ativo por vez, sendo ele o NA ou o TPS.

- *Módulo de simulação visual e cinemática inversa*. Responsável por deixar os parâmetros do robô online, por subir os plugins de cinemática inversa e por mostrar o robô no ambiente de visualização RViz[3]. Permite testes em simulação sem a necessidade do módulo do Simulink (de comunicação com o robô real) estar rodando.

- *Módulo de interface TCP com o Matlab*. É uma interface de comunicação entre o ROS e o Simulink. Esse módulo é o responsável pelo envio dos ângulos de junta presentes nos

[2] <https://www.orocos.org/kdl.html>
[3] <http://wiki.ros.org/rviz/UserGuide>

parâmetros do ROS e mostrados no RViz para o servidor TCP localizado no módulo do Simulink.

- *Módulo de controle do robô real*. Desenvolvido no Matlab 2015, através do Simulink. É quem sem comunica diretamente com os controladores do braço e do efetuador robótico, e onde os parâmetros PID das juntas são configuradas. Também possui um servidor TCP que espera receber os valores desejados das juntas (6 juntas do braço + 1 junta do efetuador). Possui diretrizes internas de segurança tais como uma chave seletora para chavear entre a posição padrão do braço e a posição recebida via TCP. Após parar de receber um valor de junta, ele mantém o último valor recebido como valor desejado. Além disso, também conta com um simulador visual interno próprio que mostra o o estado desejado do braço robótico.

5 RESULTADOS E DISCUSSÕES

5.1 Reconhecimento de Gestos

Tabela 8 – Resultados ordenados por acurácia de testes dos 10 melhores classificadores estáticos baseados em CNN.

F	S	D	P	weights	accuracy	loss
[[5, 5], [5, 5]]	32	[120, 80]	[0.5, 0.5]	347,466	96.5349	0.1294
[[5, 5], [5, 5]]	64	[160, 160]	[0.5, 0.5]	959,810	96.5246	0.1308
[[5, 5], [5, 5]]	16	[160, 80]	[0.5, 0.5]	227,394	96.5246	0.1210
[[5, 5], [5, 5]]	64	[120, 80]	[0.5, 0.5]	736,170	96.5142	0.1280
[[5, 5], [5, 5]]	32	[160, 80]	[0.5, 0.5]	454,386	96.5142	0.1241
[[5, 5], [3, 3]]	64	[160, 0]	[0.5, 0.5]	1,063,074	96.4418	0.1327
[[5, 5], [3, 3]]	64	[160, 80]	[0.5, 0.5]	1,075,794	96.4315	0.1350
[[5, 5], [3, 3]]	16	[160, 80]	[0.5, 0.5]	271,938	96.4315	0.1136
[[5, 5], [3, 3]]	64	[160, 80]	[0, 0]	1,075,794	96.4108	0.3058
[[5, 5], [5, 5]]	128	[160, 80]	[0.5, 0.5]	2,085,138	96.4004	0.1502

Nós coletamos um banco de dados de imagens binárias em formato .PGM, composto por 9668 imagens anotadas de 7 pessoas diferentes e organizado em 14 episódios. O banco possui cerca de 4 MB e está publicamente disponível[1]. Para cada pessoa, 2 episódios foram gravados a uma taxa de 30 FPS, enquanto uma tarefa imaginária de pegar-e-soltar era executada de uma lado a outro, como descrito na Seção 4.1.1. A partir de agora, discutiremos sobre a busca em grade realizada para a otimização dos hiper-parâmetros de cada modelo de classificação, levando em consideração as acurácias nos conjuntos de teste. As melhores arquiteturas encontradas estão presentes na Figura 40.

[1] <https://drive.google.com/file/d/1bOQfOFtDxgGfp_b9HELAK3WNchBUxGdX>

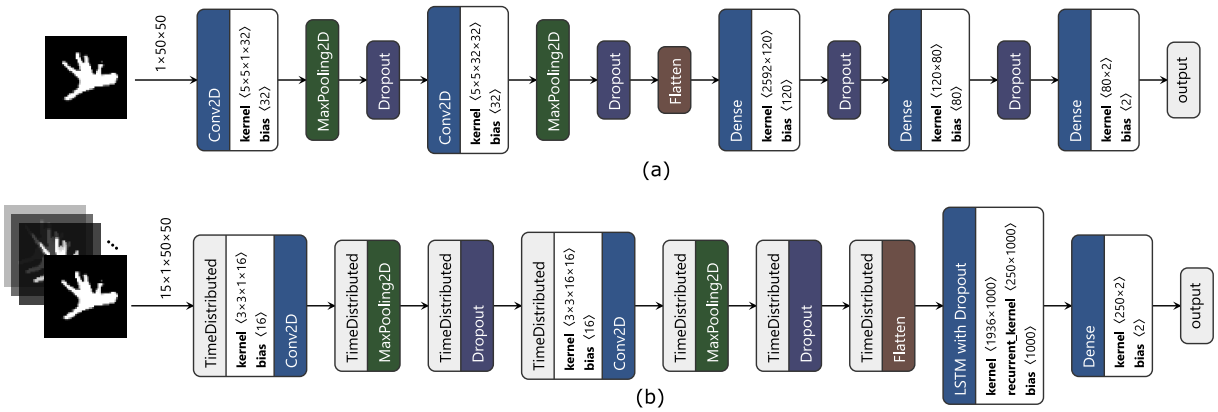

Figura 40 – Melhores classificadores baseados em CNN e LRCN encontrados nos experimentos descritos na Seção 4.1.2. Imagens foram geradas com o auxílio do software Netronapp (ROEDER, 2021).

Tabela 9 – Resultados ordenados por acurácia de testes dos 10 melhores classificadores dinâmicos baseados em LRCN.

F	S	D	P	weights	accuracy	loss
[[3, 3], [3, 3]]	16	250	[0.5, 0.5]	2,189,982	98.7225	0.0494
[[3, 3], [5, 5]]	128	50	[0, 0]	2,981,310	98.7225	0.0413
[[5, 5], [3, 3]]	64	250	[0.5, 0.5]	6,690,094	98.7119	0.0541
[[3, 3], [5, 5]]	32	250	[0.5, 0.5]	3,477,454	98.7119	0.0503
[[3, 3], [5, 5]]	32	50	[0, 0]	676,254	98.6908	0.0438
[[5, 5], [5, 5]]	128	250	[0, 0]	11,032,558	98.6908	0.0428
[[5, 5], [3, 3]]	32	100	[0.5, 0.5]	1,330,682	98.6908	0.0497
[[3, 3], [5, 5]]	128	100	[0, 0]	5,571,610	98.6908	0.0433
[[3, 3], [5, 5]]	16	500	[0.5, 0.5]	4,209,578	98.6803	0.0564
[[5, 5], [3, 3]]	64	50	[0.5, 0.5]	1,328,894	98.6803	0.0461

Tabela 10 – Acurácia de testes ao fixar os hiperparâmetros do melhor modelo LRCN encontrado, variando-se apenas o tamanho da janela de entrada.

sequence len.	accuracy	loss
15	98.7225	0.0494
10	98.3337	0.0575
5	98.1377	0.0699

A Tabela 8 mostra os resultados das 10 melhores arquiteturas para a CNN. O uso de *dropout* com probabilidades de 0,5 em ambas as camadas convolucionais e escondidas demonstrou contribuir para uma melhor generalização das redes. Além disso, o filtros de entrada de kernel $[5 \times 5]$ na primeira camada convolucional se mostraram um fator unânime entre todos os modelos no top 10. A quantidade de filtros, entretanto, apresentou alta variância dentre os modelos, não sendo possível determinar influência direta sobre o resultado. Os modelos em primeiro e terceiro lugar são preferíveis, pois demonstram uma boa capacidade de generalização, além de possuir menor quantidade de pesos quando comparados aos demais.

A Tabela 9 mostra os resultados das 10 melhores arquiteturas para a LRCN. Para a classificação dinâmica, o *dropout* mostrou menor influência sobre a acurácia. Os parâmetros que se mostraram mais presentes no top 10 foram o as dimensões do kernel $F = [[3, 3], [5, 5]]$ e o número de neurônios $D = 250$ na camada LSTM única. Os resultados sugerem que a relação temporal entre as imagens de profundidade binárias da mão do usuário contribuem para aumentar a acurácia de validação.

Para melhor investigar os efeitos do tamanho da sequência, nós experimentamos a influência da variação do tamanho da janela de entrada sobre a melhor arquitetura de LRCN encontrada. Os resultados são mostrados na Tabela 10, sugerindo uma maior acurácia para maiores sequências.

5.2 Mapeamento de Posição

Ao todo, 5 participantes estavam presentes durante os experimentos com os mapeamentos. Após finalizar o procedimento de customização de usuário, cada indivíduo executou a sessão completa com o Mapeamento de Posição *naive* (NA-PM) primeiro. Em seguida, executou a sessão completa com o Mapeamento de Posição baseado em TPS (TPS-PM). A Tabela 11 mostra os tempos dos participantes para completar cada uma das tarefas. É válido ressaltar que nós não pedíamos para os participantes completarem as tarefas no menor tempo possível, permitindo que eles tomassem o tempo necessário sem pressa. Todas as tarefas foram completadas em 3 tentativas ou menos. Um exemplo de cada tarefa pode ser visto num vídeo disponível publicamente [2]. Os tempos coletados durante os experimentos estão disponíveis da Tabela 11. Uma imagem de demonstração do experimento pode ser visualizada na Figura 41.

Figura 41 – Imagem demonstrativa de um dos experimentos realizados.

Analisando os intervalos de tempo da Tabela 11, é possível inferir que a mudança no modo de controle não teve influência substancial sobre a duração total de cada tarefa. Para melhor investigar esse fato, nós conduzimos um teste-T *two-sample* sobre ambas as classes de mapeamento para a variável *mean/atomics*, assumindo que as classes NA-PM e TPS-PM eram

[2] <https://youtu.be/Rk3iS_KnaWc>

Tabela 11 – Tempos de cada tarefa executada pelos voluntários. F significa falha na atual tentativa, com cada tarefa possuindo um total de 3 tentativas. Os tempos são representados em segundos. A forma de contagem está descrita na Seção 4.2.2.

	duration (secs) for NA-PM					duration (secs) for TPS-PM				
participant/experiment	1	2	3	4	5	1	2	3	4	5
p1	13.27	29.21	29.08	23.01	38.19	11.05	43.10	25.25	21.22	F, 26.19
p2	14.12	33.14	F, 30.05	34.09	25.12	11.24	32.10	36.04	22.09	28.17
p3	12.21	43.06	F, 36.13	23.19	32.23	19.15	76.01	51.10	F, F, 31.03	F, 35.05
p4	20.13	47.25	28.13	34.18	28.24	17.27	54.05	29.15	59.29	35.00
p5	08.12	21.04	15.10	13.23	20.11	09.03	F, 15.02	23.03	16.16	22.22
mean	13.27	33.14	29.08	23.19	28.24	11.24	43.10	29.15	22.09	28.17
std	4.33	10.57	7.70	8.82	6.88	4.39	22.96	11.30	17.25	5.63
atomics	4	12	8	8	8	4	12	8	8	8
mean/atomic	3.32	2.76	3.64	2.90	3.53	2.81	3.59	3.64	2.76	3.52

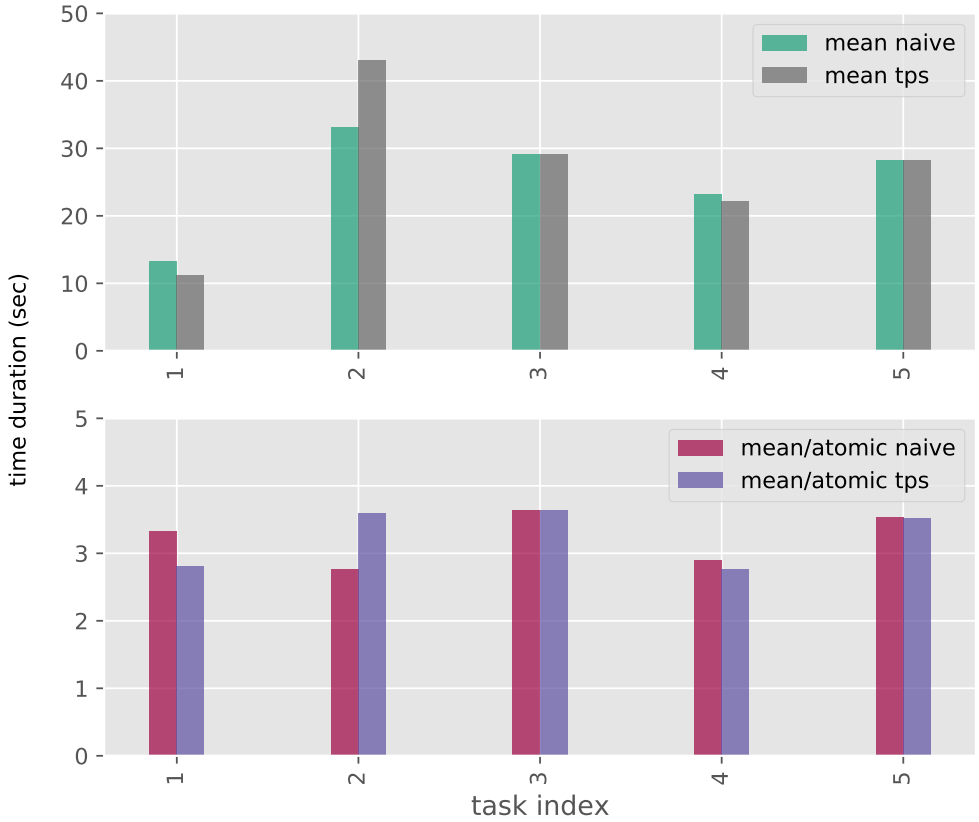

Figura 42 – Comparação entre as durações de ambos os modos de mapeamento (NA-PM e TPS-PM). Nesse gráfico são mostradas as variáveis duração média e duração média sobre o número de movimentos atômicos. Os gráficos não mostram alteração substancial nos valores das durações, o que sugere que o modelo proposto TPS-PM não acrescenta dificuldade na execução das tarefas.

independentes. O teste-T resultou em um valor p de $0.899 > 0.05$, o que leva a concluir que a hipótese nula, na qual ambos os mapeamentos (NA-PM e TPS-PM) tem a mesma esperança para a variável analisada, não pode ser descartada. Tal fato sugere que ambos os modos tem uma curva de aprendizado similar, embora possuam suas próprias peculiaridades. Esse fato pode ser confirmado visualmente na Figura 42.

Certos participantes foram mais apressados para completar as tarefas. Outros apresentaram uma postura mais cuidadosa e curiosa, tentando experimentar as nuances do movimento do robô, como o participante p3. Essa heterogeneidade no comportamento dos participantes contribuiu para uma maior variância no resultado final das durações.

As Tarefas 1, 2, e 4, as quais não possuíam operações em elevadas altitudes ou interação com a plataforma/cesta, apresentaram os menores valores de duração/atômicos. As Tarefas 1 e 2 foram completadas na primeira tentativa com todos os participantes, exceto com um. As falhas das Tarefas 3 e 5 foram devido à colisão do efetuador do robô com a cesta/plataforma, causando um deslocamento na mesma e levando a um estado não recuperável (necessitando assim reiniciar a tarefa). Falhas na Tarefa 4 foram devidas a um deslocamento específico no centro da mão do usuário que é fornecido pelo Kinect SDK 2.0 quando o estado da mão alterna entre fechado e

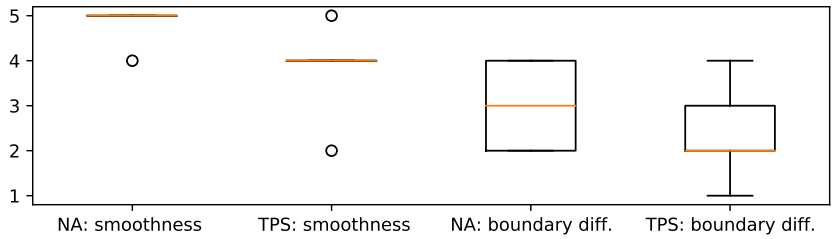

Figura 43 – Comparação entre os modos de controle (NA-PM e TPS-PM). Ambas as questões Q1 e Q2 foram respondidas numa numa escala de 1 a 5. O mapeamento *naive* foi considerado melhor com respeito à suavidade, de acordo com os usuários. A performance do mapeamento TPS com relação à dificuldade foi melhor próximo dos limites do espaço de trabalho do experimento.

aberto: quando a mão fecha, seu centro é deslocado para mais próximo do punho. O usuário p3, devido ao seu perfil de explorador, tentou entender e explorar esse comportamento durante o experimento, resultando num tempo maior para completar a tarefa.

As respostas dos participantes para as questões Likert são apresentadas na Figura 43. A partir destes *boxplots* apenas, é sugerido que os participantes concordaram mais em relação à suavidade do movimento do braço robótico. Por outro lado, apresentaram uma maior variância com relação ao controle do braço robótico próximo aos limites do espaço de trabalho, embora também haja uma tendência nessas regiões.

Em geral, os usuários consideraram o primeiro modo de controle (NA-PM) como sendo mais suave em relação à velocidade do robô. Também consideraram o segundo modo de controle (TPS-PM) como sendo melhor nas proximidades dos limites de trabalho. Ambos os fenômenos são explicados pelo fato da TPS-PM permitir que os usuários definam seu espaço de trabalho de forma customizada e não-linear. Como os usuários definem um espaço de trabalho pessoal muito menor, eles estranham a maior variação no movimento do braço robótico para pouco movimento de seu braço. Assim, embora essa customização forneça um mapeamento mais preciso nos limites do espaço de trabalho do robô (uma vez que o usuário não precisa esticar o braço tanto, gerando ruídos), sua não-linearidade também implica em variações na magnitude do gradiente da função de mapeamento, afetando o senso de suavidade.

Como as questões Q3 a Q5 eram abertas, discutiremos aqui os comentários mais relevantes dentre os participantes. A Questão Q3 foi relacionada com as vantagens de cada mapeamento. De acordo com os usuários, as vantagens do NA-PM são a intuitividade e a suavidade. Um participante reportou que "A vantagem do NA-PM é o deslocamento do braço casando com o deslocamento do robô, ou seja, o movimento relativo é quase o mesmo. A vantagem do TPS-PM é o fato do movimento nas bordas não degenerar".

A Questão Q4 era relacionada aos desafios encontrados durante a execução das tarefas, onde houve bastante diversidade nas responsas. Um participante comentou sobre dificuldades relacionadas ao classificador de mão. Após analisar seus comentários, nós concluímos que

essa dificuldade estava diretamente relacionada com o tamanho do quadrado de extração da mão, definido na Equação 4.1. Da forma que foi feita, o quadrado não leva em consideração o tamanho da mão do usuário, mas apenas sua distância até a câmera. Assim, usuários com mãos excessivamente maiores ou menores, que não estavam presentes na base de dados de treinamento, enfrentam um comportamento deteriorado do classificador. A solução mais direta pra esse problema é utilizar um aumento na base de dados gerando escalas aleatórias das imagens das mãos dentro de um intervalo bem definido.

Outro desafio importante citado por outro usuário foi sobre a estrutura do experimento: os voluntários precisavam aprender duas formas de mapeamento em sessões consecutivas. Este fato pode gerar uma certa confusão no aprendizado e, portanto, a uma performance não-ótima. Uma solução seria separar modos de controle diferentes em dias diferentes, para garantir que o aprendizado de um não influencie no outro.

Por fim, a Questão Q5 falava sobre sugestões de melhoria. Uma sugestão foi ter um classificador que funcionasse com qualquer orientação de mão. No presente trabalho, para que o classificador fosse mais preciso, os participantes eram solicitados a manter suas mãos em uma orientação específica (com a palma da mão de frente para a câmera de profundidade). Esse problema pode ser resolvido através do usuário variando a orientação de sua mão durante o processo de gravação. Outra sugestão válida foi criar uma forma mais intuitiva de calibração da TPS que incluísse um *feedback* visual e dinâmico ao invés de um avatar estático, reduzindo a necessidade de recalibração em alguns pontos.

Analisando os experimentos do ponto de vista de replicabilidade, nota-se como vantagem do mapeamento de posição baseado em Thin-Plate Spline (TPS-PM) o fato de uma captura de poses do usuário servir para diferentes robôs, bastando trocar o conjunto de pontos alvos desejado. Outra vantagem da TPS-PM é o fato de melhor explorar o espaço de trabalho do robô, que pode assumir diferentes formas, e do usuário, que pode definir sua região de trabalho da forma que melhor achar confortável.

Comparando o processo de customização de usuário para ambos os modelos de mapeamento, nota-se que o NA-PM é muito mais rápido já que é necessário apenas estender o braço direito. Em contraste, a calibração TPS-PM requer considerável percepção espacial por parte dos usuários para que eles possam visualizar a grade de posições ideal à frente do seu ombro direito. Parte do problema é devido à inexistência de um retorno visual: todo o processo se baseava no usuário inferindo onde sua mão deveria estar de acordo com a pose do avatar que lhe era mostrado. Além da componente de percepção espacial por parte dos usuários, em determinados casos, o ruído referente ao sensor e a oclusão do ombro por parte da câmera poderiam contribuir para a má captura dos pontos. Por isso era necessário um supervisor para checar se os pontos eram distribuídos uniformemente. A Figura 44 mostra exemplos de uma boa e de uma má distribuição de pontos de referência para o modelo TPS.

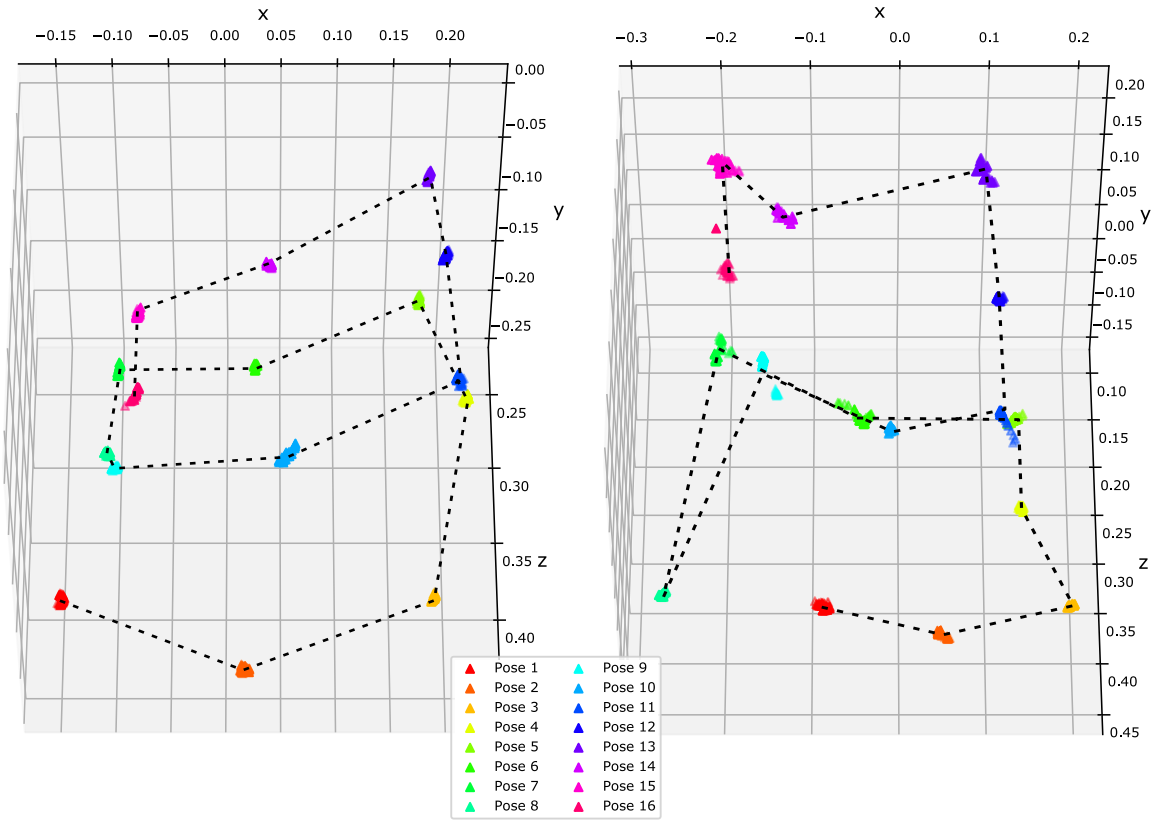

Figura 44 – Visualização tridimensional de uma customização TPS-PM de sucesso (à esquerda) e rejeitada (à direita). Há uma notável diferença nas posições relativas entre poses consecutivas. Uma calibração ruim não segue o deslocamento relativo esperado entre uma pose e outra, gerando um modelo mal formado.

6 CONCLUSÃO

Neste trabalho nós propomos, implementamos e validamos modelo inovador de Tele-manipulação robótica com Thin Plate Splines (TPS) e Redes Convolucionais Recorrentes de Longo-Prazo (LRCNs) que utiliza apenas o movimento do corpo humano como forma de controle. Tal sistema visa atender às necessidades motoras específicas dos usuários, enquanto preserva a eficiência, intuitividade, ergonomia e não-intrusividade desejáveis em um sistema de tele-manipulação baseada em visão computacional, sem deteriorar a mobilidade do robô. As limitações pessoais inerentes aos usuários que foram tratadas são: (1) diferentes limites cinemáticos; (2) possível falta de simetria em relação ao eixo vertical; (3) adequação a diferentes espaços de trabalho dos robôs; (4) conceito de conforto.

O sistema implementado funciona de forma distribuída com o ROS (Robot Operating System) como principal framework de comunicação. A validação foi realizada através de experimentos de usuário, seguidos de uma avaliação através de questionário, contendo questões abertas e questões de análise de sentimento baseadas na escala Likert. Nesse cenário, retornamos às questões de pesquisa abordadas inicialmente, agora com a experiência necessária para respondê-las:

QP1: *É possível diminuir a influência das limitações cinemáticas próprias de cada usuário sobre o espaço de trabalho alcançável pelo robô, no contexto da tele-manipulação robótica baseada em captura de movimento?* **Resposta:** Sim. Uma das formas é através da customização pessoal de cada usuário das poses-chave referentes ao seu espaço de trabalho, levando em consideração suas limitações físicas e o seu conforto. O TPS-PM se configura numa solução viável para o mapeamento dessas posições sobre o espaço de trabalho do robô, uma vez que sua interpolação é exata e possui importantes propriedades de suavidade.

QP2: *A relação sequencial entre os frames das imagens de profundidade contribui para aumentar a precisão do classificador neural de gestos de mão, no contexto da tele-manipulação robótica?* **Resposta:** Sim. Nossos testes envolvendo a validação cruzada demonstraram um aumento da acurácia média de validação de $96,5\%$ para $98,7\%$ quando comparada a classificação estática com a classificação dinâmica. Também foi realizado um estudo mostrando o aumento da acurácia com relação ao tamanho da sequência.

Os experimentos com os usuários demonstraram a adaptação com sucesso ao novo modelo de mapeamento proposto. Embora as durações de tempo utilizando o novo modelo sejam similares ao mapeamento linear *naive*, experimentos sugerem que a TPS facilita operações próximas aos limites de operação do manipulador robótico.

6.1 Trabalhos futuros

Como contribuições futuras, o presente trabalho oferece as seguintes sugestões:

- A gravação do conjunto de imagens de profundidade a partir de diferentes angulações da mão humana, para que o reconhecimento se comporte bem independentemente da sua orientação;
- O aumento na base de dados adicionando operações de escala aleatórias para aumentar a cobertura de tamanhos de mão dos usuários;
- A ampliação do conjunto de gestos reconhecidos. O presente trabalho fez uma prova de conceito com apenas dois gestos: abrir e fechar a mão. Uma sugestão seria um gesto para alternar o robô entre a posição *home* e o estado de tele-manipulação;
- Controlar a orientação da garra robótica. Atualmente, a orientação do efetuador é fixa, voltada para baixo. Há casos onde outras orientações podem ser desejadas, como a abertura de maçanetas de portas ou montagem de componentes;
- Testar diferentes funções de distância e a sua influência sobre o mapeamento TPS;
- Processamento de ambas as mãos durante o reconhecimento de gestos. Atualmente apenas a mão direita é utilizada. Entretanto, é possível utilizar a outra mão, por exemplo, para mudar a orientação do efetuador;
- *Feedback* de torque (ou corrente elétrica) nas juntas. Por questões de segurança para o próprio robô e para o ambiente ao seu redor, é importante saber o esforço que está sendo aplicado nas juntas de forma a prevenir situações de colisão desapercebidas, parando o movimento automaticamente.
- *Feedback* visual de uma câmera acoplada ao efetuador. Para que a tele-manipulação seja completa, é necessário que possamos ver o mundo aos olhos do robô. Isso pode ser realizado acoplando uma câmera numa posição fixa em relação ao centro do efetuador. Tarefas de aprendizagem por reforço também podem se beneficiar desta configuração. Embora o Kinect V2 tenha sido descontinuado, a Microsoft agora lança o Azure Kinect[1], uma câmera de profundidade portátil que serve exclusivamente para aplicações de I.A.
- Um estudo detalhado sobre o consumo de rede e sobre a possibilidade de implantação via internet (por exemplo, 5G). O foco do trabalho atual foi o teste de uma forma inovadora de mapeamento, embora mais estudos sejam necessários para um caso de tele-operação online em campo aberto, por exemplo.

[1] <https://azure.microsoft.com/pt-br/services/kinect-dk/>

REFERÊNCIAS

ABADI, M. et al. Tensorflow: A system for large-scale machine learning. In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. [S.l.: s.n.], 2016. p. 265–283. Citado na página 58.

AGGARWAL, C. C. et al. Neural networks and deep learning. *Springer*, Springer, v. 10, p. 978–3, 2018. Citado 2 vezes nas páginas 7 e 48.

AMARAL, L. et al. Evaluating deep models for dynamic brazilian sign language recognition. In: SPRINGER. *Iberoamerican Congress on Pattern Recognition*. [S.l.], 2018. p. 930–937. Citado 4 vezes nas páginas 16, 66, 68 e 72.

ARTEMIS. 2021. <https://www.nasa.gov/specials/artemis/>. NASA's Artemis Lunar Project. Citado na página 15.

BARROS, P. G. D.; LINDEMAN, R. W. A survey of user interfaces for robot teleoperation. *Worcester Polytechnic Institute*, 2009. Citado 2 vezes nas páginas 15 e 60.

BOOKSTEIN, F. L. Principal warps: Thin-plate splines and the decomposition of deformations. *IEEE Transactions on pattern analysis and machine intelligence*, IEEE, v. 11, n. 6, p. 567–585, 1989. Citado 2 vezes nas páginas 16 e 32.

BOSTON DYNAMICS, LTD. *Boston Dynamics*. 2021. <https://www.bostondynamics.com/atlas>. Citado na página 15.

BRADSKI, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. Citado na página 58.

BUILTIN. *RNNs and LSTMs*. 2021. <https://builtin.com/data-science/recurrent-neural-networks-and-lstm>. Online; Acessado em 22 apr. 2021. Citado na página 50.

CHEN, Y. et al. 3d craniofacial registration using thin-plate spline transform and cylindrical surface projection. *PloS one*, Public Library of Science San Francisco, CA USA, v. 12, n. 10, p. e0185567, 2017. Citado na página 67.

CHEN, Z. et al. Virtual-joint based motion similarity criteria for human–robot kinematics mapping. *Robotics and Autonomous Systems*, Elsevier, v. 125, p. 103412, 2020. Citado 2 vezes nas páginas 16 e 64.

CHENG, W. et al. Jointly network: a network based on cnn and rbm for gesture recognition. *Neural Computing and Applications*, Springer, v. 31, n. 1, p. 309–323, 2019. Citado 2 vezes nas páginas 16 e 66.

CHOLLET, F. *Deep learning with python*. [S.l.]: Manning Publications Co., 2017. Citado 7 vezes nas páginas 7, 47, 50, 51, 52, 53 e 59.

CHOLLET, F. et al. *Keras*. GitHub, 2015. Disponível em: <https://github.com/fchollet/keras>. Citado na página 59.

CORKE, P. *Robotics, Vision and Control: Fundamental Algorithms In MATLAB® Second, Completely Revised*. [S.l.]: Springer, 2017. v. 118. Citado na página 44.

CORKE, P. *Robotics Toolbox*. PeterCorke, 2021. Disponível em: <https://petercorke.com/toolboxes/robotics-toolbox/>. Citado na página 43.

CRAIG, J. J. *Robótica. 3ª edição*. [S.l.]: São Paulo: Editora Pearson, 2012. Citado na página 21.

DENSO. *VP6242 Specs*. 2021. <https://densorobotics.com/content/user_manuals/19/005929.html>. Online; Acessado em 26 out. 2018. Citado 3 vezes nas páginas 10, 40 e 42.

DUCHON, J. Splines minimizing rotation-invariant semi-norms in sobolev spaces. In: *Constructive theory of functions of several variables*. [S.l.]: Springer, 1977. p. 85–100. Citado na página 31.

EBERLY, D. *Thin-Plate Splines*. 1996. <https://www.geometrictools.com/Documentation/ThinPlateSplines.pdf>. Online; Acessado em 27 apr. 2021. Citado na página 31.

EVANS, H. *Introduction to Robotics*. 2014. <https://slideplayer.com/slide/5718308/19>. Online; Acessado em 04 apr. 2021. Citado 2 vezes nas páginas 6 e 36.

GAZEBO. *Gazebo*. 2021. Disponível em: <http://gazebosim.org/>. Citado na página 46.

GOODRICH, M. A.; SCHULTZ, A. C. *Human-robot interaction: a survey*. [S.l.]: Now Publishers Inc, 2008. Citado na página 15.

Gowtham, S. et al. Emg-based control of a 5 dof robotic manipulator. In: *2020 International Conference on Wireless Communications Signal Processing and Networking (WiSPNET)*. [S.l.: s.n.], 2020. p. 52–57. Citado 2 vezes nas páginas 16 e 62.

HAYKIN, S.; NETWORK, N. A comprehensive foundation. *Neural networks*, v. 2, n. 2004, p. 41, 2004. Citado na página 48.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural computation*, MIT Press, v. 9, n. 8, p. 1735–1780, 1997. Citado na página 56.

HOKAYEM, P. F.; SPONG, M. W. Bilateral teleoperation: An historical survey. *Automatica*, Elsevier, v. 42, n. 12, p. 2035–2057, 2006. Citado 2 vezes nas páginas 16 e 62.

HU, Y.; ZHOU, M.; WU, Z. A dense point-to-point alignment method for realistic 3d face morphing and animation. *International Journal of Computer Games Technology*, Hindawi, v. 2009, 2009. Citado na página 67.

HUBEL, D. H.; WIESEL, T. N. 8. receptive fields of single neurones in the cat's striate cortex. In: *Brain Physiology and Psychology*. [S.l.]: University of California Press, 2020. p. 129–150. Citado na página 52.

IROBOT CORPORATION, LTD. *iRobot*. 2021. <https://www.irobot.com.br/About-iRobot/Company-information/History>. Citado na página 15.

JR, J. J. L. et al. *3D user interfaces: theory and practice*. [S.l.]: Addison-Wesley Professional, 2017. Citado na página 16.

JU, T. *CSE 554 Lecture 7*. 1996. <https://www.cse.wustl.edu/~taoju/cse554/lectures/lect07_Deformation2.pdf>. Online; Acessado em 27 apr. 2021. Citado na página 31.

JÚNIOR, J. P. M. O uso de thin-plate splines na transformação de coordenadas com modelagem de distorções entre realizações de referenciais geodésicos. Universidade Estadual Paulista (UNESP), 2012. Citado na página 31.

KHOSHELHAM, K.; ELBERINK, S. O. Accuracy and resolution of kinect depth data for indoor mapping applications. *Sensors*, Molecular Diversity Preservation International, v. 12, n. 2, p. 1437–1454, 2012. Citado 2 vezes nas páginas 6 e 25.

LASOTA, P. A. et al. *A survey of methods for safe human-robot interaction*. [S.l.]: Now Publishers, 2017. Citado na página 61.

LEE, W.; PARK, J.; PARK, C. H. Acceptability of tele-assistive robotic nurse for human-robot collaboration in medical environment. In: . New York, NY, USA: Association for Computing Machinery, 2018. (HRI '18), p. 171–172. ISBN 9781450356152. Disponível em: <https://doi.org/10.1145/3173386.3177084>. Citado 2 vezes nas páginas 16 e 63.

LI, R.; WANG, H.; LIU, Z. Survey on mapping human hand motion to robotic hands for teleoperation. *IEEE Transactions on Circuits and Systems for Video Technology*, IEEE, 2021. Citado 4 vezes nas páginas 8, 16, 61 e 62.

LI, Y. Hand gesture recognition using kinect. In: *2012 IEEE International Conference on Computer Science and Automation Engineering*. [S.l.: s.n.], 2012. p. 196–199. Citado 2 vezes nas páginas 16 e 66.

LICHIARDOPOL, S. A survey on teleoperation. *Technische Universitat Eindhoven, DCT report*, Citeseer, v. 20, p. 40–60, 2007. Citado na página 60.

LIKERT, R. A technique for the measurement of attitudes. *Archives of psychology*, 1932. Citado na página 81.

LIMA, B. et al. Real-time hand pose tracking and classification for natural human-robot control. In: INSTICC. *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 5: VISAPP,*. [S.l.]: SciTePress, 2019. p. 832–839. ISBN 978-989-758-354-4. Citado 11 vezes nas páginas 16, 29, 30, 64, 66, 68, 72, 74, 75, 78 e 83.

LIPTON, J. I.; FAY, A. J.; RUS, D. Baxter's homunculus: Virtual reality spaces for teleoperation in manufacturing. *IEEE Robotics and Automation Letters*, IEEE, v. 3, n. 1, p. 179–186, 2017. Citado 3 vezes nas páginas 16, 63 e 79.

LUN, R.; ZHAO, W. A survey of applications and human motion recognition with microsoft kinect. *International Journal of Pattern Recognition and Artificial Intelligence*, World Scientific, v. 29, n. 05, p. 1555008, 2015. Citado 5 vezes nas páginas 6, 23, 24, 25 e 26.

MATLAB. *What is MATLAB?* MATLAB, 2021. Disponível em: <https://www.mathworks.com/discovery/what-is-matlab.html>. Citado na página 43.

MICROSOFT. *Kinect v2 for Windows SDK*. 2014. <https://developer.microsoft.com/pt-br/windows/kinect/>. Online; Acessado em 04 apr. 2021. Citado na página 23.

NOURBAKHSH, I. R.; SIEGWART, R. Introduction to autonomous mobile robots. *The MIT Press, Cambridge, Massachusetts, England, ISBN 0*, v. 262, n. 19502, p. 142–150, 2004. Citado na página 15.

NUÑO, E.; BASAÑEZ, L.; ORTEGA, R. Passivity-based control for bilateral teleoperation: A tutorial. *Automatica*, Elsevier, v. 47, n. 3, p. 485–495, 2011. Citado na página 62.

OSRF. *OSRF*. 2021. Disponível em: <https://www.openrobotics.org/>. Citado na página 46.

OYEDOTUN, O. K.; KHASHMAN, A. Deep learning in vision-based static hand gesture recognition. *Neural Computing and Applications*, Springer, v. 28, n. 12, p. 3941–3951, 2017. Citado 2 vezes nas páginas 16 e 66.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011. Citado na página 58.

PERSEVERANCE. 2021. <https://www.bbc.com/portuguese/geral-56255570>. NASA's Perseverance Rover. Citado na página 15.

QUANSER. *QUARC*. Quanser, 2021. Disponível em: <https://www.quanser.com/products/quarc-real-time-control-software/>. Citado na página 44.

RAUTARAY, S. S.; AGRAWAL, A. Vision based hand gesture recognition for human computer interaction: a survey. *Artificial intelligence review*, Springer, v. 43, n. 1, p. 1–54, 2015. Citado 3 vezes nas páginas 8, 64 e 65.

ROBOTIQ. *2F-85 Specs*. 2021. <https://robotiq.com/products/2f85-140-adaptive-robot-gripper>. Online; Acessado em 26 out. 2018. Citado 2 vezes nas páginas 42 e 43.

ROEDER, L. *Netron*. [S.l.]: GitHub, 2021. <https://github.com/lutzroeder/netron>. Citado 2 vezes nas páginas 9 e 85.

ROS. *About ROS*. 2021. Disponível em: <https://www.ros.org/about-ros/>. Citado na página 45.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958. Citado na página 48.

SHARP, T. et al. Accurate, robust, and flexible real-time hand tracking. In: ACM. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. [S.l.], 2015. p. 3633–3642. Citado 2 vezes nas páginas 16 e 65.

SICILIANO, B.; KHATIB, O. *Springer handbook of robotics*. [S.l.]: springer, 2016. Citado 6 vezes nas páginas 6, 20, 22, 34, 35 e 37.

SIMULINK. *What is Simulink?* MATLAB, 2021. Disponível em: <https://www.mathworks.com/products/simulink.html>. Citado na página 43.

SINTHANAYOTHIN, C.; BHOLSITHI, W. 3d facial deformable using cubic spline and thin plate spline. In: IEEE. *2009 6th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*. [S.l.], 2009. v. 2, p. 668–671. Citado na página 67.

SPONG, M. W. et al. *Robot modeling and control*. [S.l.]: Wiley New York, 2006. v. 3. Citado 4 vezes nas páginas 6, 15, 19 e 35.

STUDENT. The probable error of a mean. *Biometrika*, JSTOR, p. 1–25, 1908. Citado na página 82.

WASENMÜLLER, O.; STRICKER, D. Comparison of kinect v1 and v2 depth images in terms of accuracy and precision. In: SPRINGER. *Asian Conference on Computer Vision*. [S.l.], 2016. p. 34–45. Citado na página 23.

WHITBECK, M.; GUO, H. Multiple landmark warping using thin-plate splines. *IPCV*, v. 6, p. 256–263, 2006. Citado na página 67.

WILLOW GARAGE, LTD. *Willow Garage*. 2021. <https://www.willowgarage.com>. Citado na página 15.

XIAO, Y. et al. Human–robot interaction by understanding upper body gestures. *Presence: teleoperators and virtual environments*, MIT Press, v. 23, n. 2, p. 133–154, 2014. Citado na página 16.

ZALEVSKY, Z. et al. *Method and system for object reconstruction*. [S.l.]: Google Patents, 2013. US Patent 8,400,494. Citado na página 24.

ÁVILA, S. L. *Cálculo numérico aplicado à engenharia elétrica com MATLAB*. [S.l.]: Publicação do IFSC, 2019. Citado na página 30.

A ANEXO: CARREGAMENTO DO DATASET CNN

```
import cv2
import os
import numpy as np
# Sort files by the number after character '_'
def sortby( filename ):
    parts = str.split(filename, '_')
    if len(parts) < 2:
        return 0
    return int( parts[1] )

def load_dataset(episodes_path):
    # Initiate inputs and labels
    X, y = [], []
    # Get all images
    for current_path, folders, files in os.walk(episodes_path):
        folders.sort(key=sortby)
        files.sort(key=sortby)
        for filename in files:
            # Only analyze .pgm files
            if '.pgm' in filename:
                file_path = os.path.join( current_path, filename )
                try:
                    cv2_image = cv2.imread( file_path, cv2.IMREAD_GRAYSCALE
                                                                    )
                    np_image = np.array( cv2_image, dtype=float ).reshape(1
                                                                , 50, 50)
                    np_image = np_image * 1./255
                    X.append( np_image )

                    if 'closed' in filename:
                        y.append(1)
                    else:
                        y.append(0)
                except Exception as e:
                    print('[ERROR} {}'.format(e))
    X = np.array(X)
    y_categorical = np_utils.to_categorical(y)
    return (X, y, y_categorical)
```

B ANEXO: CARREGAMENTO DO DATASET LSTM

```
import os
import cv2
import numpy as np
# Sort files by the number after character '_'
def sortby( filename ):
    parts = str.split(filename, '_')
    if len(parts) < 2:
        return 0
    return int( parts[1] )

def load_dataset(episodes_path, images_per_sequence=15):
    # Initiate inputs and labels
    X, y = [], []
    # Get all images, iterating over folders
    for current_path, folders, files in os.walk(episodes_path):
        folders.sort(key=sortby)
        files.sort(key=sortby)
        ppgm_files = []
        ppgm_labels = []
        # Search for specific extension
        for filename in files:
            # Only analyze .pgm files
            if '.pgm' in filename:
                file_path = os.path.join( current_path, filename )
                try:
                    cv2_image = cv2.imread( file_path, cv2.IMREAD_GRAYSCALE
                                                                    )
                    np_image = np.array( cv2_image, dtype=float ).reshape(1
                                                                , 50, 50)
                    np_image = np_image * 1./255
                    ppgm_files.append( np_image )

                    if 'closed' in filename:
                        ppgm_labels.append(1)
                    else:
                        ppgm_labels.append(0)
                except Exception as e:
                    print('[ERROR} {}'.format(e))

        # Create LSTM sequences within each episode (different folder)
        total_images = len(ppgm_files)
        for image_index in range(images_per_sequence, total_images + 1):
            sample = ppgm_files[image_index - images_per_sequence :
                                                    image_index]
```

```
            label = ppgm_labels[ image_index - 1 ]

            X.append(sample)
            y.append(label)

    X = np.array(X)
    y_categorical = np_utils.to_categorical(y)
    return X, y, y_categorical
```

C ANEXO: IMPLEMENTAÇÃO DO MAPEAMENTO TPS

```
"""
Esta implementacao tomou como base a versao 2D da TPS implpementada pelo
                                    OpenCV e discutida por Khanh Ha em:
https://khanhha.github.io/posts/Thin-Plate-Splines-Warping/
"""
import numpy as np
import pickle
np.set_printoptions(precision=4, suppress=True)

class TPS3D:
    def distance(self, p1, p2):
        """
        Distance calculation between p1 and P2.
        TPS uses radial basis distance of form r^2 * log (r).
        x1 and x2 are 3d points.
        """
        diff = np.array(p1) - np.array(p2)
        norm = np.linalg.norm(diff)
        return norm * norm * np.log(norm) if norm > 0 else 0

    def fit(self, c_points, t_points):
        """
        Parameters are control and target points of type numpy_array, both
                                            with dimension [m x 3].
        """
        assert c_points.shape[0] == t_points.shape[0], "Control and target
                                            array points must have same
                                            dimension."
        m = c_points.shape[0]
        self.c_points = c_points
        self.t_points = t_points

        """
        K matrix contains the spatial relation between control points,
                                            calculating their distance 2-
                                            by-2.
        P matrix contains the control_points in homogeneous form (1, c_x,
                                            c_y, c_z)
        """
        self.K = np.zeros((m, m))
        self.P = np.zeros((m, 4))

        for i in range(m):
            for j in range(m):
```

```
                self.K[i, j] = self.distance(c_points[i], c_points[j])
            self.P[i, 0] = 1
            self.P[i, 1:] = c_points[i]

        """
        L matrix aggregate information about K, P and P.T blocks
        """
        self.L = np.zeros((m+4, m+4))
        self.L[:m, :m] = self.K
        self.L[:m, m:] = self.P
        self.L[m:, :m] = self.P.T

        # Building the right hand side B = [v 0].T.
        self.B = np.zeros((m+4, 3))
        self.B[:m, :] = t_points

        # Solve for TPS parameters
        # Find vector  = [w a]^T
        self.tps_params = np.linalg.solve(self.L, self.B)

    def transform(self, p):
        """
        Apply a tps warp into a 3d numpy array point. Returns the point
                                                after transformation.
        """
        output_point = np.zeros((3, 1))
        for i in range(3):
            # Calculate affine term
            a1 = self.tps_params[-4, i]
            ax = self.tps_params[-3, i]
            ay = self.tps_params[-2, i]
            az = self.tps_params[-1, i]

            affine = a1 + ax*p[0] + ay*p[1] + az*p[2]

            # Calculate non-rigid deformation.
            # Sum up contribuitions for all kernels, given the kernel
                                                    weights
            nonrigid = 0
            for j in range(self.c_points.shape[0]):
                nonrigid += self.tps_params[j, i] * self.distance(self.
                                                        c_points[j], p)

            output_point[i] = affine + nonrigid
        return output_point

    def assessModel(self):
```

```
        # check errors between target and transformed points
        total_error = 0
        for (index, input_point) in enumerate(self.c_points):
            output_point = self.transform(input_point).squeeze()
            target_point = self.t_points[index].squeeze()
            error_vector = np.around(target_point - output_point, decimals=
                                                    5)
            total_error += np.linalg.norm(error_vector)
        return total_error

    def save(self, filename='tps_model.pkl'):
        with open(filename, 'wb') as output_file:
            pickle.dump(self, output_file, pickle.HIGHEST_PROTOCOL)

    def load(self, filename='tps_model.pkl'):
        with open(filename, 'rb') as input_file:
            loaded_model = pickle.load(input_file)
            self.__class__ = loaded_model.__class__
            self.__dict__ = loaded_model.__dict__

if __name__ == '__main__':
    """
    Testing the TPS model with sample points.
    """
    # Control points
    xs = [-411.34, -276.03, -98.33, 40.22, 136.59]
    ys = [-143.36, -153.81, -133.12, -137.78, -113.75]
    zs = [-371.41, -391.57, -364.37, -290.95, -158.73]
    c_points = np.vstack([xs, ys, zs]).T

    # Target points
    xt = [0.32 , 0.35, 0.32,  0.2, 0.12]
    yt = [-0.15,    0, 0.15, 0.12, 0.08]
    zt = [0.1  ,  0.2,  0.3,  0.4,  0.5]
    t_points = np.vstack([xt, yt, zt]).T

    # Instantiating TPS model
    tps_model = TPS3D()
    tps_model.fit(c_points, t_points)

    p = np.array([-411.34, -143.36, -371.41])
    print("Output for {} is {}: ".format(p, tps_model.transform(p).T.
                                        squeeze() ) )

    total_error = tps_model.assessModel()
    print("Total absolute error for control and target points is: {}".
                                        format(total_error))
```