



Universidade Federal do ABC
Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas
Programa de Graduação em Engenharia de Instrumentação, Automação e
Robótica

Um estudo envolvendo os fundamentos da cinemática e suas aplicações diretas em um protótipo robótico humanoide

**Bruno Costa da Silva
Humberto Yago da Silva Pinto
Maurício Lura Segura**

Número de Ordem : XXXX

Santo André - SP, Dezembro de 2022

Bruno Costa da Silva
Humberto Yago da Silva Pinto
Maurício Lura Segura

**Um estudo envolvendo os fundamentos da cinemática e
suas aplicações diretas em um protótipo robótico
humanoide**

Dissertação de Graduação apresentada
ao Programa de Graduação em Engenharia
de Instrumentação, Automação e Robótica,
como parte dos requisitos necessários para
a obtenção do Título de Bacharel em
Engenharia de Instrumentação, Automação e
Robótica.

Universidade Federal do ABC – UFABC
Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas
Programa de Graduação em Engenharia de Instrumentação, Automação e Robótica

Orientador: Luiz Antônio Celiberto Júnior

Santo André - SP
Dezembro de 2022

Bruno Costa da Silva
Humberto Yago da Silva Pinto
Maurício Lura Segura

Um estudo envolvendo os fundamentos da cinemática e suas aplicações diretas em um protótipo robótico humanoide/
Bruno Costa da Silva
Humberto Yago da Silva Pinto
Maurício Lura Segura. – Santo André - SP, Dezembro de 2022-
Orientador: Luiz Antônio Celiberto Júnior

Dissertação (Graduação) – Universidade Federal do ABC – UFABC
Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas
Programa de Graduação em Engenharia de Instrumentação, Automação e Robótica,
Dezembro de 2022.
I. modelagem cinemática. II. reconhecimento de imagens. III. Luiz Celiberto.
IV. Universidade Federal do ABC. V. Um estudo envolvendo os fundamentos da
cinemática e suas aplicações diretas em um protótipo robótico humanoide

CDU
XX.XXX.XXX.XXX

Bruno Costa da Silva
Humberto Yago da Silva Pinto
Maurício Lura Segura

**Um estudo envolvendo os fundamentos da cinemática e
suas aplicações diretas em um protótipo robótico
humanoide**

Dissertação de Graduação apresentada
ao Programa de Graduação em Engenharia
de Instrumentação, Automação e Robótica,
como parte dos requisitos necessários para
a obtenção do Título de Bacharel em En-
genharia de Instrumentação, Automação e
Robótica.

Trabalho aprovado. Santo André - SP, 01 de dezembro de 2022:

Luiz Antônio Celiberto Júnior
Orientador

Elvira Rafikova
Avaliadora

Santo André - SP
Dezembro de 2022

Para nossas famílias, nossos amigos e para todos aqueles que nos apoiaram.

Agradecimentos

Agradecemos ao nosso orientador, Luiz Celiberto, pela confiança ao longo de todo o tempo durante a execução das disciplinas de TG I, II e III.

Aos nossos amigos pelo suporte, as boas piadas e alguns dos melhores momentos que tivemos durante todo o período acadêmico.

Aos nossos professores que atuaram ao longo de toda a construção de nossa jornada dentro da universidade, assim como seus conselhos, o suporte na tomada de decisões e o apoio em momentos difíceis.

Agradecemos também a Universidade Federal do ABC pela disponibilização de infraestrutura para execução do projeto, assim como TA's e professores envolvidos no desenvolvimento do projeto.

*“Ou nós encontramos um caminho,
ou abrimos um”*

*Original do latim: “Aut viam inveniam, aut faciam”
(Provérbio atribuído ao General Cartaginês Aníbal(247 a.C. - 183 a.C.))*

Resumo

Por meio do estudo de modelagem cinemática e o uso de ferramentas de reconhecimento de imagem desenvolveu-se um protótipo com base em um projeto Open Source InMoov. Integra-se ao projeto então fundamentos de aprendizado de máquina e fundamentos de manufatura aditiva. A proposta do trabalho de graduação é compreender, analisar e aplicar os principais aspectos de cinemática e dinâmica de mecanismos ligados a robótica, por meio de um estudo teórico e sua aplicação atrelada a um modelo de braço humanoide, pautando-se em reconhecimento de padrões para gerar as instruções de acionamento do equipamento. Para tornar este protótipo funcional, revisitou-se tópicos apresentados em disciplinas de aprendizado de máquina, sobretudo redes neurais e sua combinação com dispositivos microcontroladores.

Palavras-chaves: cinemática. reconhecimento de imagem. InMoov. aprendizado de máquina. manufatura aditiva.

Abstract

Through the study of kinematic modeling and the use of image recognition tools, a prototype was developed based on an Open Source InMoov project. It then integrates machine learning fundamentals and additive manufacturing fundamentals into the project. The proposal of the graduation project is to understand, analyze and apply the main aspects of kinematics and dynamics of mechanisms linked to robotics, through a theoretical study and its application linked to a humanoid arm model, based on pattern recognition to generate the equipment activation instructions. To make this prototype functional, topics presented in machine learning disciplines were revisited, especially neural networks and their combination with microcontroller devices.

Keywords: kinematic.image recognition. InMoov. machine learning. additive manufacturing.

Lista de ilustrações

Figura 1 – Identificação de Ossos e Articulações da Mão	4
Figura 2 – Identificação de Músculos da Mão	4
Figura 3 – Descrição de parâmetros de DH	5
Figura 4 – Descrição de parâmetros de DH	5
Figura 5 – Modelagem do dedo polegar	8
Figura 6 – Modelagem do dedo polegar	10
Figura 7 – Modelagem do dedo indicador e médio	12
Figura 8 – Projeto InMoov	14
Figura 9 – Fases do Processo de Manufatura Aditiva	18
Figura 10 – Modelagem de Superfície B-Spline	19
Figura 11 – Modelagem de Superfície STL	20
Figura 12 – Interface de software fatiador para impressão 3D	21
Figura 13 – Principais parâmetros envolvidos no processo de impressão 3D	22
Figura 14 – Principais perfis de preenchimento	23
Figura 15 – Estimativas de gastos (tempo e filamento) fornecidas pelo fatiador	23
Figura 16 – Interface de pré-visualização do processo de impressão	24
Figura 17 – Comando para contração total da articulação	25
Figura 18 – Comando para aproximadamente metade da expansão máxima da articulação	25
Figura 19 – Comando para máxima expansão da articulação	25
Figura 20 – Primeira Arquitetura Proposta	26
Figura 21 – Arquitetura Final	27
Figura 22 – Exemplo de saídas na imagem feitas com OpenCV	28
Figura 23 – Exemplo de Identificação das Juntas	29
Figura 24 – Exemplo de Identificação das Juntas	29
Figura 25 – Perceptron	30
Figura 26 – Exemplo de Rede Neural Perceptron Multicamadas	31
Figura 27 – Sinal PWM de controle do servo	32
Figura 28 – Conjunto de peças impressas	36
Figura 29 – Fixação necessárias nos dedos	36
Figura 30 – Fixação necessárias nos dedos	37
Figura 31 – Fixação necessárias no braço	37
Figura 32 – Estrutura da mão	37
Figura 33 – Juntas dos dedos	38
Figura 34 – Montagem da mão completa	38
Figura 35 – Montagem do pulso	39

Figura 36 – Passagem de fios pelos dedos	39
Figura 37 – Arremate dos fios na ponta do dedo	40
Figura 38 – Passagem de fios pela mão	40
Figura 39 – Esquema de passagem de fios	41
Figura 40 – Engrenagens do pulso e passagem de fios pelo pulso	42
Figura 41 – Fixação do pulso no braço	42
Figura 42 – Suporte servos	43
Figura 43 – Servos fixados	44
Figura 44 – Servos com polias	44
Figura 45 – Fios conectados as polias	45
Figura 46 – Fios tensionados	45
Figura 47 – Montagem final	46
Figura 48 – Montagem final	46
Figura 49 – Montagem final	47
Figura 50 – Montagem final	47
Figura 51 – Primeira Etapa Arquitetura	48
Figura 52 – Pré Processamento	48
Figura 53 – Parte da Matriz BGR gerada no passo 3 do pré processamento	48
Figura 54 – Segunda Etapa Arquitetura	49
Figura 55 – Processamento MediaPipe	49
Figura 56 – Exemplo de saída para imagem do passo 4	49
Figura 57 – Terceira Etapa Arquitetura	50
Figura 58 – Exemplo entrada e saída da rede neural	50
Figura 59 – Exemplo de Entrada Sem Processamento	50
Figura 60 – Exemplo de Coordenada Relativa	51
Figura 61 – Exemplo de Normalização	51
Figura 62 – Exemplo de execução do código do apêndice A	52
Figura 63 – Exemplo de Dado Salvo	52
Figura 64 – Matriz de Confusão Obtida Após o Treinamento do Modelo	54
Figura 65 – Estrutura do Modelo	55
Figura 66 – Quarta Etapa Arquitetura	55
Figura 67 – Cálculo Distância Polegar Indicador	56
Figura 68 – Percentual de Expansão	56
Figura 69 – Parâmetros para cada gesto	57
Figura 70 – Teste de Integração do Polegar	59
Figura 71 – Teste de Integração do Indicador	60
Figura 72 – Identificação Pulso	61
Figura 73 – Identificação Polegar	61
Figura 74 – Identificação Indicador	61

Figura 75 – Identificação Médio	62
Figura 76 – Identificação Anelar	62
Figura 77 – Identificação Mindinho	62
Figura 78 – Diversas Distâncias Entre os Dedos	63

Lista de tabelas

Tabela 1 – Notação de Juntas para o modelo	7
Tabela 2 – Notações para links no modelo	7
Tabela 3 – Parâmetros DH para o polegar	9
Tabela 4 – Parâmetros DH para o anelar e dedo mínimo	10
Tabela 5 – Parâmetros DH para o polegar	12
Tabela 6 – Lista de componentes para impressão do braço direito	15
Tabela 7 – Listagem de partes de composição dos dedos	15
Tabela 8 – Gestos	24
Tabela 9 – Tabela de componentes impressos	35
Tabela 10 – Rótulos dos Gestos	52

Listas de abreviaturas e siglas

DH	Notação de Denavit-Hartenberg
STL	Standard Triangle Language (em inglês:)
CAD	Desenho Assistido por Computador (em inglês: <i>Computer Aided Design</i>)
CAM	Manufatura Assistida por Computador (em inglês: <i>Computer Aided Manufacturing</i>)
DOF	Graus de Liberdade (em inglês: <i>Degress of freedom</i>)
PLA	Poliácido Lático
PETG	Polietileno Tereftalato Glicol
ABS	Acrilonitrila Butadieno Estireno (em inglês: <i>Acrylonitrile Butadiene Styrene</i>)
AM	Manufatura Aditiva (em inglês: <i>Additive Manufacturing</i>)
AMF	Formato de Manufatura Aditiva (em inglês: additive manufacturing format)
STEP	Padrão para a troca de dados do modelo de produto (em inglês: Standard for the Exchange of Product Model data)
CMC	Carpometacarpal
MCF	Metacarpofalângica
IFP	Interfalângica Proximal
IFD	Interfalângica Distal
PWM	Pulse Width Modulation

Sumário

1	INTRODUÇÃO	1
Introdução		1
1.1	Objetivos	2
Objetivos		2
1.1.1	Objetivos Específicos	2
Objetivos Específicos		2
2	TÓPICOS TEÓRICOS RELEVANTES AO PROJETO	3
2.1	Aspecto biológico/estrutural da mão humana	3
2.2	Análise Cinemática	5
2.2.1	Cinemática de Sistemas Mecânicos	5
2.3	Modelagem Cinemática	6
2.3.1	Cinemática do polegar	8
2.3.2	Cinemática do anelar e dedo mínimo	9
2.3.3	Cinemática do indicador e dedo médio	11
2.4	Projeto InMoov	14
3	MATERIAIS E MÉTODOS	17
3.1	Manufatura Aditiva	17
3.1.1	Softwares de Modelagem 3D	18
3.1.2	Processo de Geração de Modelo Mosaico	19
3.1.3	Softwares Fatiadores	20
3.2	Impressão e montagem mecânica	23
3.3	Ferramentas de Visão Computacional	24
3.3.1	Finalidade	24
3.3.2	Arquitetura da Solução Desenvolvida	25
3.3.3	OpenCV	28
3.3.4	MediaPipe	28
3.3.5	Redes Neurais Perceptron Multicamadas	30
3.4	Ferramentas para Controle de Microcontroladores	31
3.4.1	Arduíno	32
3.4.2	Esquema elétrico	32
3.4.3	Servo motores	33
3.4.4	Python	33
4	ESTRUTURA DO PROJETO	35

4.1	Impressão de componentes	35
4.2	Estrutura desenvolvida	36
4.3	Computação Visual	41
4.3.1	Identificação das Coordenadas das Juntas das Mão	42
4.3.2	Reconhecimento de Gestos	43
4.3.2.1	Coleta de Dados de Treino	44
4.3.2.2	Treinamento do Modelo	53
4.3.3	Controle da Expansão e Contração da Articulação	55
4.4	Processo de Conexão entre Interfaces	57
4.4.1	Interface entre algoritmo de reconhecimento e braço robótico	57
5	RESULTADOS E DISCUSSÃO	59
5.1	Consolidação sistêmica	59
5.2	Integração do sistema com dispositivo eletromecânico	60
5.3	Computação Visual	60
5.3.1	Reconhecimento de Gestos	60
5.3.2	Controle da Expansão e Contração da Articulação	63
6	CONCLUSÕES E TRABALHOS FUTUROS	65
Conclusão e Trabalhos Futuros		65
6.1	Conclusões	65
6.2	Trabalhos Futuros	65
REFERÊNCIAS		67
APÊNDICE A – SCRIPT PARA COLETA DE DADOS DE TREINO		69
APÊNDICE B – SCRIPT PARA TREINAMENTO DO MODELO		79
APÊNDICE C – SCRIPT PARA EXECUÇÃO DO PROJETO		81
ANEXOS		95
ANEXO A – ESQUEMA ELÉTRICO		97
ANEXO B – DATASHEET MG995 - TOWER PRO		99

1 Introdução

Ao avaliar os principais fundamentos de modelagem cinemática, autores usualmente definem estes passos como introdutórios para a evolução dos fundamentos sobre o funcionamento de dispositivos robóticos (HARTENBERG, 1955) e (CRAIG, 2004). Ao tratar sobre dispositivos baseados em robótica humanoide, algumas considerações adicionais se mostram necessárias a considerar: as limitações entre uma estrutura real e uma estrutura simulada, tendo como corpo de estudo uma mão humana, compara-se a estrutura real (SOBOTTA, 2000) e sua versão simulada (STOPPA, 2017).

A comparação por sua vez enriquece o espaço das ideias, favorecendo ainda mais a discussão e o questionamento sobre oportunidades para aprimoramento entre estruturas, visando assim aprimorar os modelos de representação bioinspirados.

Por sua vez, a manufatura aditiva ocupa papel primordial no funcionamento deste projeto, se dando majoritariamente pela confecção direta de componentes mecânicos (CUNICO, 2015) considerando os fundamentos de impressão 3D conforme indicado por literatura dedicada (PORTELA, 2019) e (VALLE, 2016).

Somando-se os aspectos teóricos e práticos estruturais, contando com uma base de projeto *Open Source*, a proposta de estudo neste trabalho baseia-se em modelagens desenvolvidas inicialmente pelo artista plástico (LANGEVIN, 2012) em seu projeto InMoov.

Uma das possíveis aplicações do projeto consiste na utilização de um algoritmo de reconhecimento de imagem, com auxílio de bibliotecas como OpenCV (SENYAEY, 2022) e MediaPipe (MEDIAPIPE,) para coleta de imagens, conversão dessas imagens em modelos que podem ser identificados pelo computador e reconhecimento de padrões que permitem a posterior classificação dos mesmos

O processamento das imagens por sua vez, conta com um modelo de rede neural para classificação dos gestos (SCIKIT-LEARN, 2022), e a partir dessa classificação movimenta o braço robótico de forma responsiva, através do protocolo Firmata (DOCS, 2022).

1.1 Objetivos

O objetivo do projeto é majoritariamente pelo estudo da modelagem cinemática aplicado a uma mão robótica humanoide, somando-se ao reconhecimento de imagens para desenvolvimento de um protótipo funcional, garantindo o funcionamento de um dispositivo eficaz baseando-se em aspectos de manufatura aditiva para concepção do equipamento e o uso de redes neurais artificiais para o reconhecimento de padrões.

1.1.1 Objetivos Específicos

Pode-se destacar como principais objetivos específicos do projeto:

- Por meio dos fundamentos de manufatura aditiva desenvolver um protótipo funcional de mão humanoide com base em ferramentas Open Source;
- Utilizar as principais bibliotecas de aprendizado de máquina para fomentar o funcionamento do dispositivo como reconhecedor de padrões;
- Desenvolver uma boa arquitetura eletromecânica para o movimento do dispositivo;
- Basear o acionamento dos trechos da mão robótica por meio dos fundamentos de modelagem cinemática.

2 TÓPICOS TEÓRICOS RELEVANTES AO PROJETO

2.1 Aspecto biológico/estrutural da mão humana

A proposta de realização de um estudo dos fundamentos de cinemática aplicados a um protótipo robótico humanoide estão diretamente atrelados a um processo de réplica do mecanismo natural, deste modo aborda-se através de ma síntese dos fundamentos biológicos agregados a estrutura do membro superior estudado (mão), assim como sua estrutura auxiliar (antebraço).

A mão por sua vez, é identificada como a composição final da estrutura de membro superior do corpo humano. Sua formação é composta por (SOBOTTA, 2000):

- 27 ossos
- 17 articulações
- 19 músculos

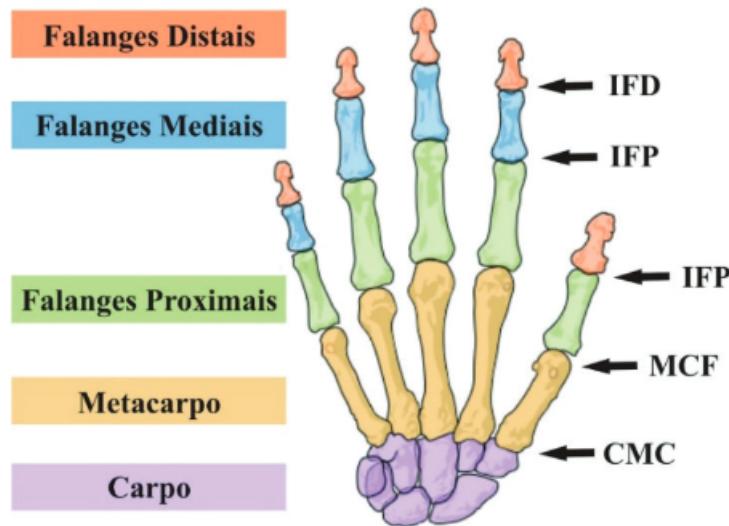
A estrutura em questão garante ao membro um total de 23 graus de liberdade (DOF),

O conjunto dos 23 ossos é categorizado (SOBOTTA, 2000) em três grupos. São eles: Carpo (Capitato, Escafóide, Piramidal, Pisiforme, Trapézio e Trapezóide), Falange (Distal, Medial e Proximal) - destaca-se que o polegar não possui falange medial e Metacarpo (1 por dedo). A figura 1 sintetiza o vínculo direto e a localização destes três grupos.

As articulações sinalizadas na figura 1 possuem sua nomenclatura por sua vez é fornecida por meio de sua localização estrutural, sendo listadas de modo geral como:

- Carpometacarpal (CMC)
- Metacarpofalângica (MCF)
- Interfalângica Proximal (IFP)
- Interfalângica Distal (IFD)

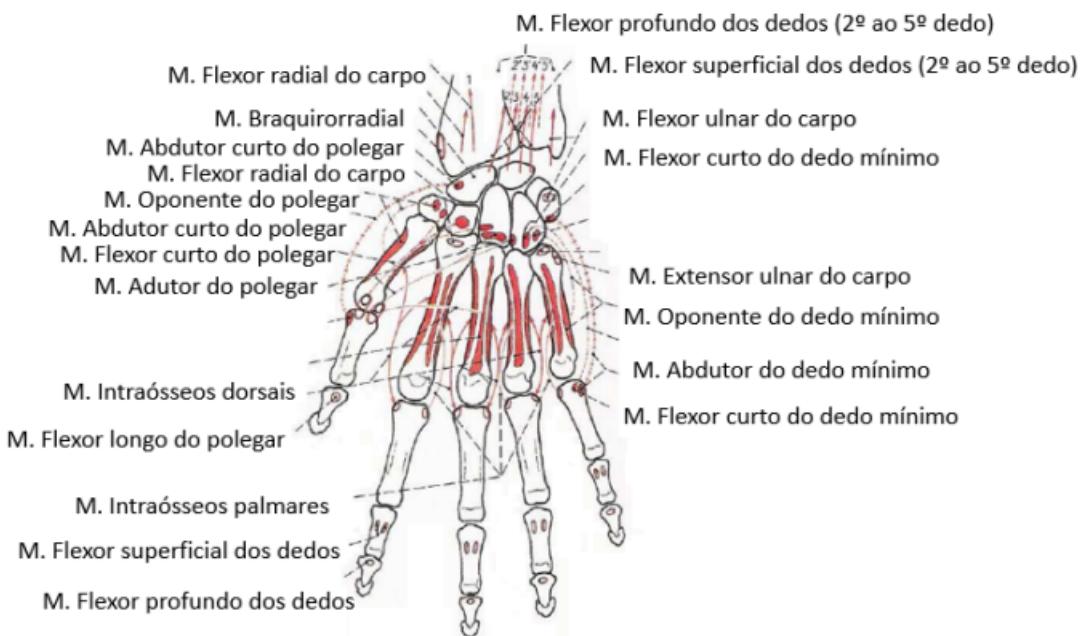
Figura 1 – Identificação de Ossos e Articulações da Mão



Fonte: ([STOPPA, 2017](#))

O conjunto de ossos por sua vez está vinculado diretamente ao agrupamento de músculos e articulações, sendo este conjunto responsável pela variedade de movimentos que podem ser realizados pela mão. Os músculos por sua vez podem ser identificados conforme a figura 2 e as articulações podem ser visualizadas por meio da figura 1.

Figura 2 – Identificação de Músculos da Mão



Fonte: ([CRUZ, 2017](#))

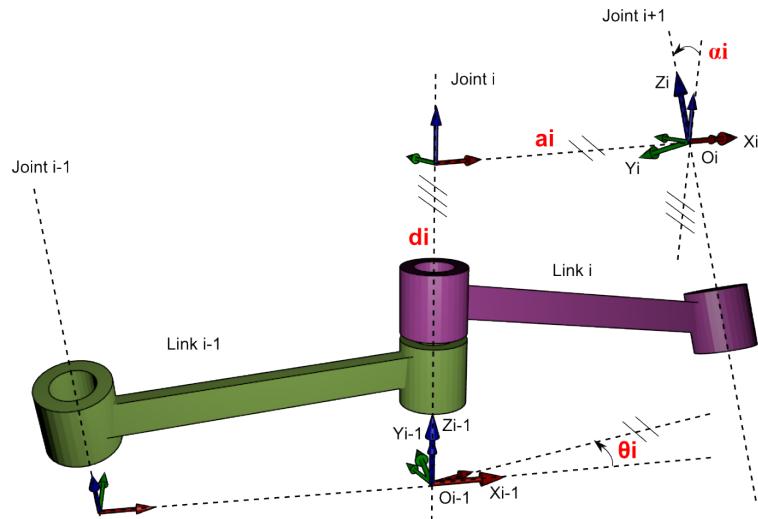
2.2 Análise Cinemática

2.2.1 Cinemática de Sistemas Mecânicos

Avaliando os aspectos agregados aos parâmetros cinemáticos de movimentação dentro de um espaço tridimensional, um dos primeiros fundamentos a se considerar é a notação de Denavit-Hartenberg (DH) como uma ferramenta utilizada para metodizar a descrição cinemática de sistemas mecânicos articulados com n graus de liberdade (HARTENBERG, 1955).

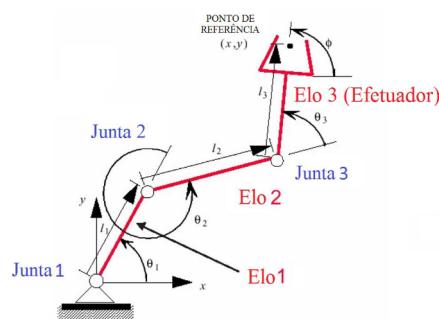
Por sua vez, a representação DH para uma estrutura rígida está diretamente vinculada a quatro aspectos essenciais para descrição do desempenho do dispositivo mecânico. As figuras 3 e 4 ilustram os parâmetros definidos através desta representação.

Figura 3 – Descrição de parâmetros de DH



Fonte: Wikimedia Commons

Figura 4 – Descrição de parâmetros de DH



Fonte: (VALLE, 2016)

Ademais, pode-se definir os parâmetros ilustrados na figura 3 como (ROSÁRIO, 2005):

- θ_i como ângulo de junta obtido tendo como referencial os eixos X (X_{i-1} e X_i) aplicados ao eixo Z_{i-1} - Vale destacar que um dos métodos para obtenção desta relação é através da regra da mão direita.
- d_i é a distância demarcada através da origem em $(i - 1)$ -ésimo composto do sistema de coordenadas até sua interseção no eixo Z_{i-1} com o eixo X_i ao longo do eixo $Z_i - 1$.
- a_i é a distância de *off-set* determinado entre a interseção do eixo Z_{i-1} com eixo X_i , até a origem do i-ésimo sistema de referência ao longo do eixo X_i - Vale destacar que outra alternativa de referência pode se dar através da menor distância entre os eixos Z_{i-1} e Z_i .
- α_i é o ângulo de *off-set* obtido através da medição em X_i da diferença entre Z_{i-1} e Z_i - De modo análogo ao θ_i também pode ser obtido através da regra da mão direita.

Uma vez definidos os parâmetros considerando as figuras 3 e 4 pode-se avaliar que ao tratar-se de uma junta de caráter rotacional, as variáveis: α_i , d_i e a_i serão os parâmetros desta junta. Ademais, seus valores estão atrelados a relação entre a rotação do elo i e seu vínculo com $i - 1$. Em paralelo, ao avaliar-se um perfil de uma junta prismática, são definidos: α_i , θ_i e a_i como parâmetros da junta, por sua vez, d_i é apontada como a variável de junta em um cenário de deslocamento linear.

Uma vez que o conjunto que compõe o sistema de coordenadas DH foi estabelecido, torna-se possível gerar a matriz para transformação homogênea. A composição dessa matriz está atrelada aos frames: i-ésimo ao $(i - 1)$ do conjunto de coordenadas. A matriz abaixo demonstra sua composição.

$$\begin{bmatrix} \cos \theta_i & -\cos \alpha_i \cdot \sin \theta_i & \sin \alpha_i \cdot \sin \theta_i & a_i \cdot \cos \theta_i \\ \sin \theta_i & \cos \alpha_i \cdot \sin \theta_i & -\sin \alpha_i \cdot \cos \theta_i & a_i \cdot \sin \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Uma vez desenvolvida a matriz homogênea de transformação, torna-se possível realizar todo processo de modelagem cinemática.

2.3 Modelagem Cinemática

Tendo indicado o conjunto de articulações conforme a figura 1, pode-se desenvolver o conjunto de notações atrelados a essas juntas para cada um dedos.

Tabela 1 – Notação de Juntas para o modelo

Juntas	Dedos				
	Mínimo (L)	Anelar (R)	Médio (M)	Indicador (I)	Polegar(T)
Interfalângica Distal (DIP)	$\theta_{L,DIP}$	$\theta_{R,DIP}$	$\theta_{M,DIP}$	$\theta_{I,DIP}$	-
Interfalângica Proximal (PIP)	$\theta_{L,PIP}$	$\theta_{R,PIP}$	$\theta_{M,PIP}$	$\theta_{I,PIP}$	-
Metacarpofalângica (MCP)	$\theta_{L,MCP}$	$\theta_{R,MCP}$	$\theta_{M,MCP}$	$\theta_{I,MCP}$	$\theta_{T,MCP}$
Carpometacarpal (CMC)	$\theta_{L,CMC}$	$\theta_{R,CMC}$	-	-	-
Interfalângica (IP)	-	-	-	-	$\theta_{T,IP}$
Trapézio metacarpal (TMC)	-	-	-	-	$\theta_{T,TMC}$

Tabela 2 – Notações para links no modelo

Links	Dedos				
	Mínimo (L)	Anelar (R)	Médio (M)	Indicador (I)	Polegar(T)
Metacarpal (MC)	$\ell_{L,MC}$	$\ell_{R,MC}$	-	-	$\ell_{T,MC}$
Proximal (P)	$\ell_{L,P}$	$\ell_{R,P}$	$\ell_{M,P}$	$\ell_{I,P}$	$\ell_{T,P}$
Média (M)	$\ell_{L,M}$	$\ell_{R,M}$	$\ell_{M,M}$	$\ell_{I,M}$	-
Distal (D)	$\ell_{L,D}$	$\ell_{R,D}$	$\ell_{M,D}$	$\ell_{I,D}$	$\ell_{T,D}$

Tendo sido desenvolvido o conjunto de notações para as juntas da mão conforme as tabelas 1 e 2, torna-se possível prosseguir baseando-se na notação de Denavit-Hartenberg (DH).

Os procedimentos para realizar a análise consistem em: ([STOPPA, 2017](#))

1. Identificar o conjunto de eixos L_i em cada uma das juntas;
2. Realizar a escolha do pé de perpendicular comum (O_i) aos eixos do conjunto de juntas L_i e L_{i+1} , situados sobre L_i . Ademais, em situações em que ocorra o paralelismo ou a coincidência entre eixos, pode-se realizar uma escolha arbitrária de perpendicular comum. Tal ação pode ser realizada devido ao fato das considerações de simetria e simplicidade;
3. Sendo X_i é um vetor unitário dessa perpendicular comum, pautando-se na orientação de L_i na direção da articulação L_{i+1} . De forma análoga ao pé perpendicular, se os eixos forem concorrentes ou coincidentes, a orientação será arbitrária.
4. Z_i é um vetor unitário referente ao eixo L_i , sendo este orientado arbitrariamente;
5. Y_i por sua vez é definido pelo produto vetorial positivo.

As soluções para cada dedo do ponto de vista da cinemática direta se dá por meio do conjunto de matrizes homogêneas, este conjunto por sua vez fornece ao dispositivo a possibilidade de terminar a posição e a orientação das pontas dos dedos relativas ao sistema de coordenação inercial ([KHALIL W.; KLEINFINGER, 1986](#))).

Pautando-se no conjunto de parâmetros de DH, assumi-se o conjunto de 3 configurações, com os agrupamentos de:

- Dedo Polegar;
- Dedo indicador e médio;
- Dedo anelar e mínimo.

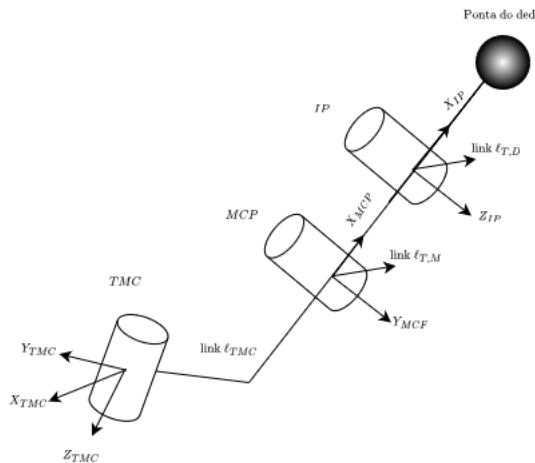
Devido as limitações da estrutura do projeto InMoov ([LANGEVIN, 2012](#)), avaliou-se apenas movimentos de flexão e três posições cinemáticas para cada um dos dedos.

2.3.1 Cinemática do polegar

Uma vez estabelecido o conjunto de princípios para a modelagem cinemática, os fundamentos baseados em DH e a estrutura da mão humana, pode-se dar início ao desenvolvimento do estudo de cinemática.

Utilizando os fundamentos descritos no capítulo anterior, a modelagem do polegar pode ser representada por meio da figura 5.

Figura 5 – Modelagem do dedo polegar



Fonte: Autores (com base em ([STOPPA, 2017](#)))

O polegar por sua vez pode ser descrito como ([CRAIG, 2004](#)):

$$P_T =_0^{\omega} T_T(r_T, \omega)_1^0 T_T(\theta_{T,TMC})_2^1 T_T(\omega_{T,MCP})_3^2 T_T(\omega_{T,IP})_4^3 T_T(\theta_{T,D})$$

Onde P_T representa a matriz que compõe a posição e a orientação da ponta do polegar em relação ao punho. Já T_T são matrizes que representam as contribuições de translação ou rotação de cada junta, gerando a matriz: ()

$$k^{-1}T_T(\theta_{i,j}) = \begin{bmatrix} \cos \theta_{i,j} & -\sin \theta_{i,j} & 0 & d_{i,j} \\ \cos \alpha_{i,j} \sin \theta_{i,j} & \cos \alpha_{i,j} \cos \theta_{i,j} & -\sin \alpha_{i,j} & -r_{i,j} \sin \alpha_{i,j} \\ \sin \alpha_{i,j} \sin \theta_{i,j} & \sin \alpha_{i,j} \cos \theta_{i,j} & -\cos \alpha_{i,j} & -r_{i,j} \cos \alpha_{i,j} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Tabela 3 – Parâmetros DH para o polegar

junta	θ_μ	d_μ	r_μ	α_μ
W	0	0	$r_{T,w}$	0
1	$\theta_{T,TMC}$	0	$r_{T,MC}$	$\alpha_{T,MC}$
2	$\theta_{T,MCP}$	0	$r_{T,P}$	$\alpha_{T,P}$
3	$\theta_{T,IP}$	$d_{T,P}$	0	0
4	0	$d_{T,D}$	0	0

De forma geral, as matrizes T_T são dadas por:

$$\begin{aligned} {}^0\omega T_T(r_{T,\omega}) &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & r_{T,\omega} \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ {}^s s^{-1}T_T(\theta_{T,j}) &= \begin{bmatrix} \cos \theta_{T,j} & -\sin \theta_{T,j} & 0 & d_{i,j} \\ \cos \alpha_{T,\nu} \sin \theta_{T,j} & \cos \alpha_{T,\nu} \cos \theta_{T,j} & -\sin \alpha_{T,\nu} & -r_{T,\nu} \sin \alpha_{T,\nu} \\ \sin \alpha_{T,\nu} \sin \theta_{T,j} & \sin \alpha_{T,\nu} \cos \theta_{T,j} & -\cos \alpha_{T,\nu} & -r_{T,\nu} \cos \alpha_{T,\nu} \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

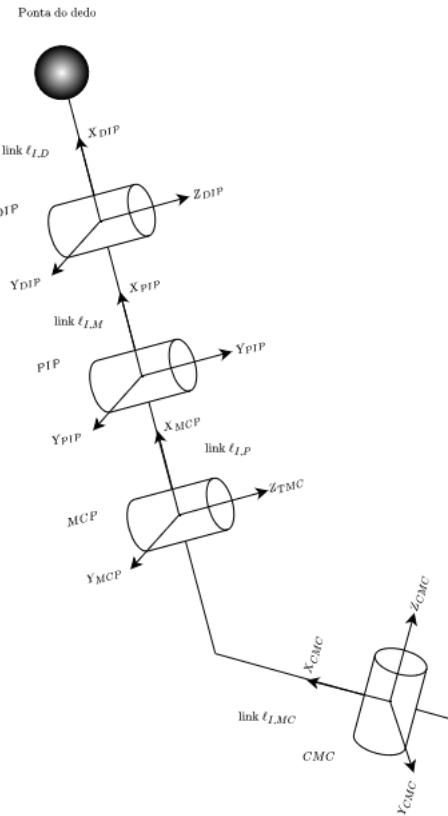
onde ($s = 1, 2$), ($\nu = MC, P$) e ($j = TMC, MCP$), nessa ordem.

$$\begin{aligned} {}^2T_T(\theta_{T,IP}) &= \begin{bmatrix} \cos \theta_{T,IP} & -\sin \theta_{T,IP} & 0 & d_{T,P} \\ \sin \theta_{T,IP} & \cos \theta_{T,IP} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ {}^3T_T(\theta_{T,D}) &= \begin{bmatrix} 0 & 0 & 0 & d_{T,D} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

2.3.2 Cinemática do anelar e dedo mínimo

Considerando os fundamentos aplicados no processo de análise de modelagem no polegar, torna-se possível desenvolver a modelagem do dedo anelar e mínimo pode ser representada por meio da figura 7.

Figura 6 – Modelagem do dedo polegar



Fonte: Autores (com base em ([STOPPA, 2017](#)))

Os dedos anelares e mínimos por sua vez podem ser descritos utilizando os fundamentos aplicados ao polegar:

Tabela 4 – Parâmetros DH para o anelar e dedo mínimo

junta	θ_μ	d_μ	r_μ	α_μ
W	0	0	$r_{\gamma,w}$	0
1	$\theta_{\gamma,CMC}$	0	$r_{\gamma,MC}$	$\alpha_{\gamma,MC}$
2	$\theta_{\gamma,MCP}$	0	$r_{\gamma,P}$	$\alpha_{\gamma,P}$
3	$\theta_{\gamma,PIP}$	$d_{\gamma,P}$	0	0
4	$\theta_{\gamma,DIP}$	$d_{\gamma,D}$	0	0
5	θ	$d_{\gamma,D}$	0	0

$$P_\gamma = {}^\omega_0 T_\gamma(r_\gamma, \omega)_1^0 T_\gamma(\theta_{\gamma,CMC})_2^1 T_\gamma(\omega_{\gamma,MCP})_3^2 T_\gamma(\omega_{\gamma,DIP})_4^3 T_\gamma(\theta_{\gamma,PIP})_5^4 T_\gamma(\theta_{\gamma,DIP})$$

Vale ressaltar que P_γ representa o conjunto de matrizes que fornecem a posição e orientação das pontas dos dedos com base no centro do punho. Já T_γ compõe as matrizes indicadas abaixo:

$$\begin{aligned} {}_0^{\omega}T_{\gamma}(r_{\gamma,\omega}) &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & r_{\gamma,\omega} \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ {}_s^{-1}T_{\gamma}(\theta_{\gamma,j}) &= \begin{bmatrix} \cos \theta_{\gamma,j} & -\sin \theta_{\gamma,j} & 0 & 0 \\ \cos \alpha_{\gamma,\nu} \sin \theta_{\gamma,j} & \cos \alpha_{\gamma,\nu} \cos \theta_{\gamma,j} & -\sin \alpha_{\gamma,\nu} & -r_{\gamma,\nu} \sin \alpha_{\gamma,\nu} \\ \sin \alpha_{\gamma,\nu} \sin \theta_{\gamma,j} & \sin \alpha_{\gamma,\nu} \cos \theta_{\gamma,j} & \cos \alpha_{\gamma,\nu} & -r_{\gamma,\nu} \cos \alpha_{\gamma,\nu} \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

onde ($s = 1, 2$), ($\nu = MC, P$) e ($j = CMC, MCP$), nessa ordem.

$$\begin{aligned} {}_h^{-1}T_{\gamma}(\theta_{\gamma,k}) &= \\ &\begin{bmatrix} \cos \theta_{\gamma,k} & -\sin \theta_{\gamma,k} & 0 & d_{\gamma,\omega} \\ \sin \theta_{\gamma,k} & \cos \theta_{\gamma,k} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

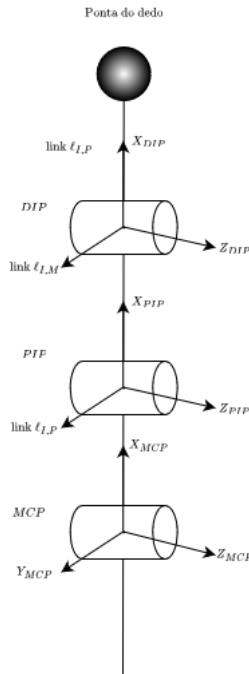
onde ($h = 1, 2$), ($w = P, M$) e ($k = PIP, DIP$), nessa ordem.

$${}^4T_{\gamma}(\theta_{\gamma,D}) = \begin{bmatrix} 0 & 0 & 0 & d_{\gamma,D} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.3.3 Cinemática do indicador e dedo médio

De forma análoga ao estudo de fundamentos de modelagem descritos para os outros dedos, considerando o algoritmo de DH e aspectos da morfologia humana, os dedos indicadores e médios por sua vez podem ser descritos como:

Figura 7 – Modelagem do dedo indicador e médio



Fonte: Autores (com base em ([STOPPA, 2017](#)))

De forma análoga ao polegar, anelar e mínimo, pode-se modelar o dedo indicador e médio.

Tabela 5 – Parâmetros DH para o polegar

junta	θ_μ	d_μ	r_μ	α_μ
W	0	0	$r_{b,w}$	0
1	$\theta_{\beta,MCP}$	0	$r_{b,MC}$	$\alpha_{b,MC}$
2	$\theta_{\beta,PIP}$	$d_{\beta,P}$	0	0
3	$\theta_{\beta,DIP}$	$d_{\beta,M}$	0	0
4	0	$d_{\beta,D}$	0	0

$$P_T = {}^0_0T_\beta(r_\beta, \omega)^0_1T_\beta(\theta_{\beta,MCP})^1_2T_\beta(\omega_{\beta,PIP})^2_3T_\beta(\omega_{T,DIP})^2_4T_\beta(\theta_{\beta,D})$$

onde P_β representa o conjunto de matrizes que fornecem a posição e orientação das pontas de dedos em relação ao punho. Ademais, T_β são matrizes semelhantes as aplicadas aos polegares,

$${}^0_1T_\beta(\theta_{\beta,MCP}) = \begin{bmatrix} -\sin \theta_{\beta,MCP} & -\sin \theta_{T,MCP} & 0 & 0 \\ \cos \alpha_{\beta,MCP} \sin \theta_{\beta,MCP} & \cos \alpha_{\beta,MCP} \cos \theta_{\beta,MCP} & -\sin \alpha_{\beta,MCP} & -r_{\beta,MCP} \sin \alpha_{\beta,MCP} \\ \sin \alpha_{\beta,MCP} \sin \theta_{\beta,MCP} & \sin \alpha_{\beta,MCP} \cos \theta_{\beta,MCP} & \cos \alpha_{\beta,MCP} & -r_{\beta,MCP} \cos \alpha_{\beta,MCP} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}_{s^{-1}}T_{\beta}(\theta_{\beta,j}) = \begin{bmatrix} \cos \theta_{\beta,j} & -\sin \theta_{\beta,j} & 0 & d_{\beta,\nu} \\ \sin \theta_{\beta,j} & \cos \theta_{\beta,j} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

onde ($s = 1, 2$), ($\nu = P, M$) e ($j = PIP, DIP$), nessa ordem.

$${}_{4^{-1}}T_{\beta}(\theta_{\beta,D}) = \begin{bmatrix} 1 & 0 & 0 & d_{\beta,D} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Uma vez encontrados os parâmetros de modelagem cinemática para simulações computacionais, pode-se dar continuidade ao projeto, modelando os 5 dedos.

2.4 Projeto InMoov

O Projeto InMoov trata-se de uma infraestrutura em hardware e software *open source* desenvolvida pelo escultor e designer francês Gael Langevin. A iniciativa teve início em 2012 com o desenvolvimento de uma prótese de membro superior (mão) ([LANGEVIN, 2012](#)). A figura 8 ilustra sua estrutura parcialmente concluída pelo criador do projeto. A estrutura em questão é majoritariamente composta por componentes plásticos que podem ser impressos através de manufatura aditiva, ademais, também existem parafusos, motores e linhas de *nylon* na composição da estrutura.

O projeto InMoov por sua vez foi escolhido devido a fácil disponibilidade de arquivos e orientações gerais para impressão 3D, somadas ao conjunto de informações coletadas ao longo do desenvolvimento de referências bibliográficas, torna-se possível utilizar a estrutura de mão e antebraço para aplicação de algoritmos de visão computacional.

Figura 8 – Projeto InMoov



Fonte: ([LANGEVIN, 2012](#))

O conjunto de peças que compõe o braço direito do InMoov (figura 8) é descrito através das tabelas 7 e 6.

Tabela 6 – Lista de componentes para impressão do braço direito

Componente	Quantidade
Thumb*	1
Index*	1
Majeure*	1
RingFinger*	1
Auriculaire* (Pinky)	1
BoltEntretoise	1
Wristlarge	1
Wristsmall	1
Topsurface	1
Coverfinger	1
Robcap	1
Robpart2	1
Robpart3	1
Robpart4	1
Robpart5	1
ElbowShaftGear	1
rotawrist1	1
rotawrist2	1
rotawrist3	1
WristGears	1
CableHolderWrist	1

Fonte: ([LANGEVIN, 2012](#))

Ademais, ao avaliar a tabela 6, destaca-se que cada um dos dedos (identificados com *) possui um subconjunto de componentes conforme a tabela 7.

Tabela 7 – Listagem de partes de composição dos dedos

Nome do Conjunto	Quantidade de Peças
Auriculaire3	6
Index3	6
Majeure3	6
Ringfinger3	6
Thumb5	6
Coverfinger1	5

Fonte: ([LANGEVIN, 2012](#))

3 Materiais e Métodos

3.1 Manufatura Aditiva

A manufatura aditiva assume papel primordial no processo de desenvolvimento mecânico do protótipo, conforme descrito anteriormente, o projeto InMoov por sua vez já possuí todos os componentes mecânicos essenciais para o desenvolvimento do braço robótico humanoide. Uma vez que o autor já disponibiliza as principais instruções de como efetuar os processos de impressão 3D e fixação de componentes mecânicos.

Os processos de fabricação tradicionalmente se baseiam em ações de modelagem e moldagem de um material, até a década de 80 entre as principais ações para moldagem de material pode-se destacar ([VOLPATO, 2017](#)):

- Fusão - Por meio do uso de moldes por injeção de plástico, metalurgia do pó (como a sinterização), modelagem de dispositivos em fibra, e etc;
- Remoção - Utilizando-se de ações para subtração de material até obtenção do formato desejado (processos como: torneamento, fresamento, furação, eletroerosão, e etc);
- Conformação - Método que baseia-se na consolidação da geometria da peça através da deformação plástica da matéria-prima (por exemplo, forjamento, extrusão, laminiação, e etc);
- União de componentes - Por meio de ações de soldagem, brasagem, colagem, entre outros;
- Divisão de componentes - Utilizando-se de atividades de serragem e corte.

Contudo, a partir da década de 1990, surge um novo procedimento para fabricação baseado na disposição de material, denominado atualmente de AM (do inglês *additive manufacturing*) - *Manufatura aditiva* ou impressão 3D ([VOLPATO, 2017](#)).

Este procedimento por sua vez se define como um conjunto de ações de fabricação através da disposição de material em camadas subsequentes, tais camadas possuem um conjunto de coordenadas específicas e pré-determinadas através do suporte de *softwares* de fatiamento de arquivos. A linha do tempo de um processo de MA pode ser identificada por meio da figura 9.

As fases indicadas na figura 9 podem ser descritas como:

Figura 9 – Fases do Processo de Manufatura Aditiva



Fonte: Autores

- Modelagem da Peça - Ação baseada na geração de um modelo tridimensional do artefato que deseja-se imprimir. Comumente utilizam ferramentas de sistema CAD para composição dos objetos;
- Geração de Modelo Mosaico - Por meio do modelo geométrico desenvolvido no software de CAD (*Computer Aided Design* - Desenho Assistido por Computador), gera-se um novo formato de arquivo baseado em uma malha de triângulos em um padrão que possa ser reconhecido por ferramentas de fatiamento, os formatos mais comuns são: STL - *Standard Triangle Language* e AMF - *Additive Manufacturing Format*;
- Fatiamento de peça - O processo é realizado com base no conjunto de triângulos gerados pelo arquivo de modelo mosaico e soma-se ao conjunto de planejamento de ações para melhor processo de disposição de camadas para consolidação da peça;
- Impressão 3D - Fabricação do objeto considerando os aspectos definidos ao longo do fatiamento da peça, assim como seu modelo desenvolvido anteriormente em CAD.

3.1.1 Softwares de Modelagem 3D

Apesar de não serem diretamente utilizados neste projeto pois as peças impressas já foram previamente modeladas pelo artista plástico responsável pelo projeto, opta-se por uma breve apresentação dos principais recursos no que tange a atividades de modelagem 3D. Selecionou-se um grupo de 5 principais ferramentas no cenário de modelagem, destaca-se

que a maior dos softwares citados abaixo possuem licenças pagas, mas também contam com versões estudantis.

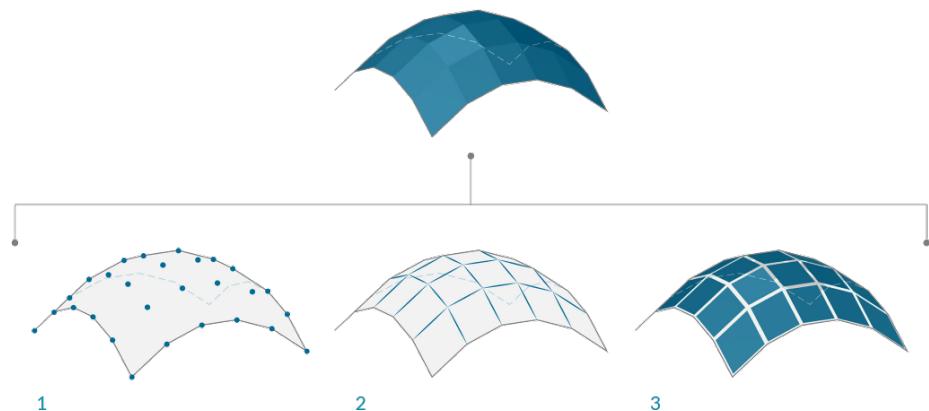
- Inventor
- Blender
- Maya
- SolidWorks
- Fusion360

3.1.2 Processo de Geração de Modelo Mosaico

Uma vez concluída a modelagem de peças, conforme ilustrado pela figura 9, torna-se possível dar início ao processo de geração de modelo mosaico (BUSTAMANTE, 2019). Este processo por sua vez é realizado de forma simplificada, uma vez que a maior parte das ferramentas de modelagem 3D já possui em sua estrutura a liberdade para geração de arquivos em formatos de modelo mosaico (CRUZEIRO, 2019).

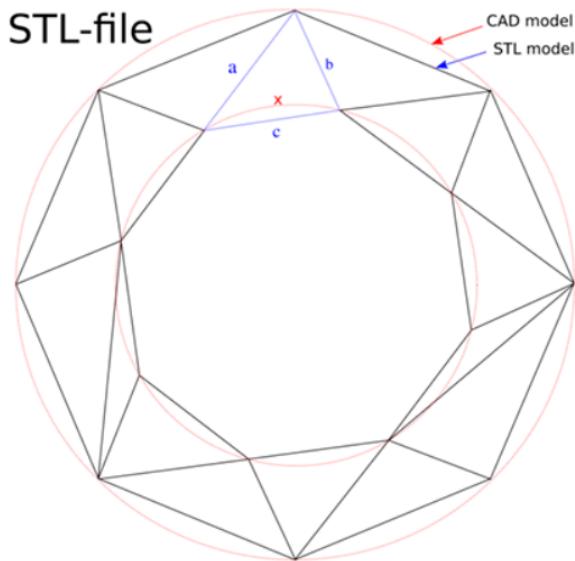
Os principais modelos de geração de mosaicos são: STL, AMF e STEP (*Standard for the Exchange of Product Model data*) e por meio destes, a representação do modelo com ou sem a simplificação do modelo CAD original, sempre se baseando na geração de superfícies biquadráticas, *sweeping*, *B-Spline* (conforme a figura 10) e malhas de triângulos (conforme a figura 11) (BUSTAMANTE, 2019).

Figura 10 – Modelagem de Superfície B-Spline



Fonte: pngwing.com

Figura 11 – Modelagem de Superfície STL



Fonte: 3D Lab

A principal justificativa para conversão dos arquivos desenvolvidos está atrelada a versatilidade que o novo arquivo possui, pois em seu formato de modelo mosaico existem infinitas partes menores para serem processadas, ação essa facilitada, pois em softwares fatiadores torna-se mais eficaz a atuação de divisão em camadas e o carregamento da peça como um todo (PORTELA, 2019).

Uma vez concluído o processo de geração de modelo mosaico, pode-se dar início ao fatiamento desta peça, ação essa facilitada, uma vez que o conjunto de partículas que compõe a peça é facilmente interpretado e integrado a grande parte das ferramentas fatiadoras no mercado de AM.

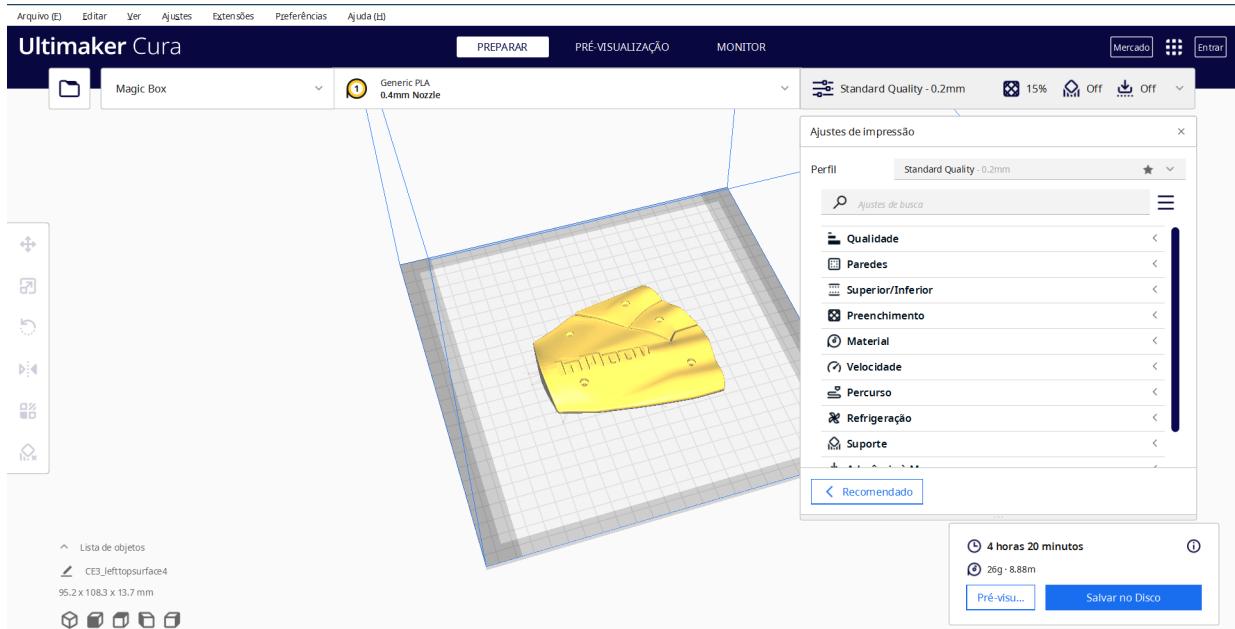
3.1.3 Softwares Fatiadores

Fatiadores são ferramentas fundamentais para o processo de transformação de uma peça 3D em um arquivo G-Code para impressão(REPRAP, 2010). É através destes softwares que torna-se possível configurar uma série de parâmetros que modificam tanto questões estéticas (como acabamento), quanto estrutural (considerando aspectos de distribuição de tensão entre pontos).

A interface de impressão 3D é apresentada na figura 12, assim como os principais recursos utilizados pelo software fatiador escolhido pelo grupo - Cura são indicados nas figuras 13, 15 e 16.

Através da figura 12 pode-se destacar os 3 principais blocos da interface para o processo de fatiamento de peças 3D, são eles:

Figura 12 – Interface de software fatiador para impressão 3D



Fonte: Autores

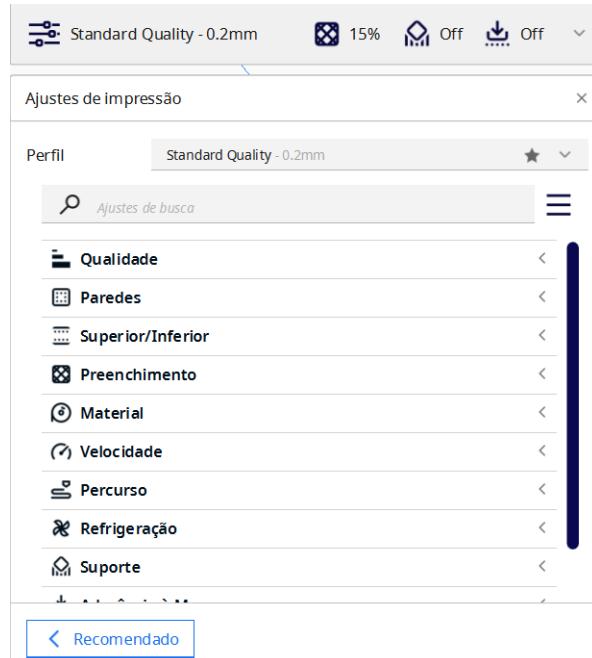
- Parâmetros de impressão (figura 13).
- Dimensões e estimativas de impressão (figura 15).
- Seção de pré-visualização (figura 16).

Conforme ilustrado pela figura 13, através da interação com os principais parâmetros de impressão 3D torna-se possível adaptar o processo conforme a destinação específica pretendida ao modelo.

Dentre os principais parâmetros considerados no processo de impressão como podem ser vistos na figura 13, pode-se listar os seguintes pontos:

- **Altura de camada:** Parâmetro de destaque no processo de acabamento e também rigidez das peças. A altura de camada determina a quantidade de "fatias" que a peça terá, influenciando assim diretamente no tempo necessário para impressão.
- **Espessura de paredes:** De forma similar ao parâmetro anterior, a espessura da parede está intimamente ligado a resistência do objeto, uma vez que determina a quantidade de material que será depositada por parede em razão das dimensões absolutas da peça.
- **Densidade de preenchimento:** Parâmetro utilizado para determinar a taxa de preenchimento que a peça possuirá ao longo de sua extensão. Peças com maiores densidades de preenchimento possuem maior resistência mecânica e consequentemente, levam mais tempo para serem impressas.

Figura 13 – Principais parâmetros envolvidos no processo de impressão 3D

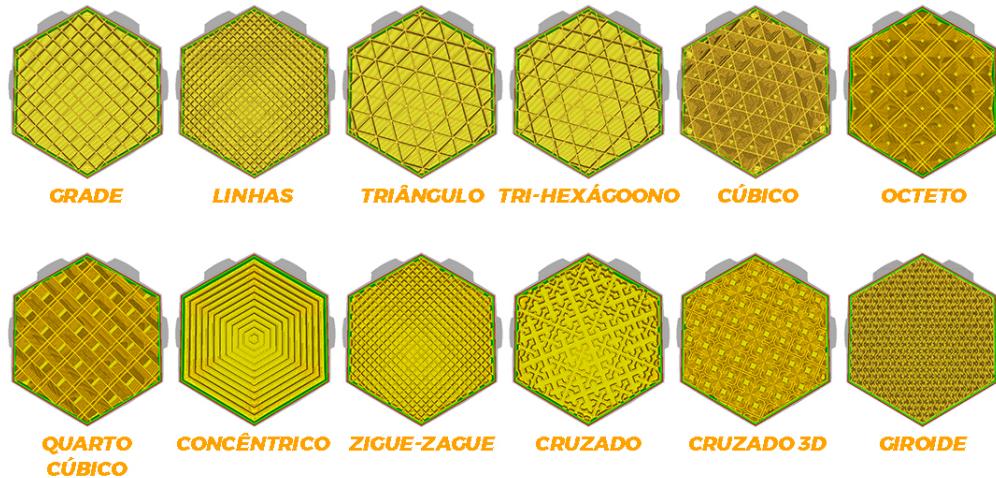


Fonte: Autores

- **Padrão de preenchimento:** Parâmetro utilizado para definir o formato das camadas se serão dispostas internamente na peça, alguns dos principais padrões são ilustrados na figura 14.
- **Temperatura de Extrusora:** Parâmetro utilizado para controlar a temperatura do bico extrusor visando garantir que o material que será depositado na mesa estará com boa viscosidade para adesão a mesa. Vale destacar que a temperatura tende a oscilar conforme o tipo de polímero utilizado.
- **Temperatura de Mesa:** Parâmetro utilizado para controle mesa de base onde será depositado. É essencial utilizar valores específicos considerando o polímero utilizado, tendo como objetivo garantir uma boa fixação das camadas na mesa de impressão.
- **Velocidade de impressão:** Parâmetro operacional responsável por definir a velocidade de trajetória da extrusora, através deste parâmetro pode-se controlar e avaliar as melhores opções de custo-benefício considerando tempo para o processo de impressão 3D.
- **Utilização de suportes:** Comando opcional para situações em que a peça possua seções com pontos inclinados em um ângulo superior a 45°.

Uma vez que os parâmetros mencionados acima foram considerados, torna-se possível prosseguir com o processo de impressão 3D, conforme a figura 15 a estimativa de massa e comprimento do filamento gasto é fornecida, assim como o tempo previsto.

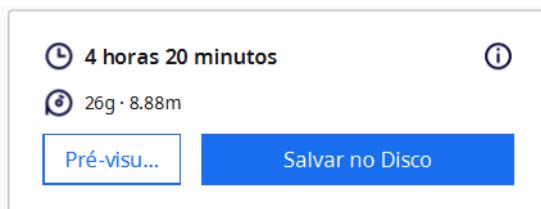
Figura 14 – Principais perfis de preenchimento



Fonte: 3D Lab

Ademais, na terceira interface do fatiador é possível acompanhar o processo de separação em camadas e a trajetória que será executada pela extrusora nos limites da mesa de impressão.

Figura 15 – Estimativas de gastos (tempo e filamento) fornecidas pelo fatiador



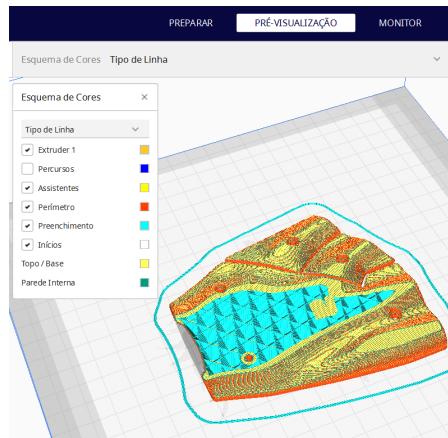
Fonte: Autores

A interface de pré-visualização pode ser visualizada através da figura 16, o fatiador por sua vez fornece uma distinção entre o perfil de cada tipo de linha que será executado pela extrusora. Podemos avaliar o preenchimento e sua disposição na peça, conforme ilustrado anteriormente na figura 14 dispõe-se de uma variedade de topologias para preenchimento que por sua vez podem ser alteradas conforme a justificativa para impressão da peça.

3.2 Impressão e montagem mecânica

O conjunto total de peças pertencentes ao bíceps e a mão são apresentados na tabela 6, assim como a quantidade necessária de cada uma peças plásticas

Figura 16 – Interface de pré-visualização do processo de impressão



Fonte: Autores

3.3 Ferramentas de Visão Computacional

3.3.1 Finalidade

O objetivo da computação visual neste projeto é realizar o controle das articulações do braço robótico por meio da captura dinâmica de imagens das duas mãos de um operador.

A mão esquerda é responsável por indicar qual articulação será movimentada por meio de um gesto pré configurado enquanto que a mão direita é responsável por controlar a expansão e contração da articulação.

Será possível realizar o controle de seis articulações de acordo com os gestos abaixo:

Gesto	Articulação Acionada
	Pulso
	Polegar
	Indicador
	Médio
	Anelar
	Mindinho

Tabela 8 – Gestos

A expansão e contração da articulação selecionada pela mão esquerda será controlada pela distância entre o dedo polegar e indicador da mão direita, sendo que quanto

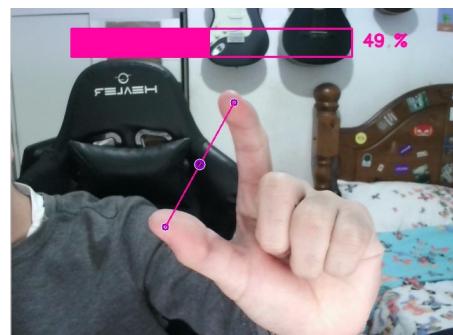
maior a distância entre esses dois dedos, maior será a expansão da articulação até que ocorra a expansão máxima ou contração máxima, conforme exemplos abaixo:

Figura 17 – Comando para contração total da articulação



Fonte: Autores

Figura 18 – Comando para aproximadamente metade da expansão máxima da articulação



Fonte: Autores

Figura 19 – Comando para máxima expansão da articulação



Fonte: Autores

3.3.2 Arquitetura da Solução Desenvolvida

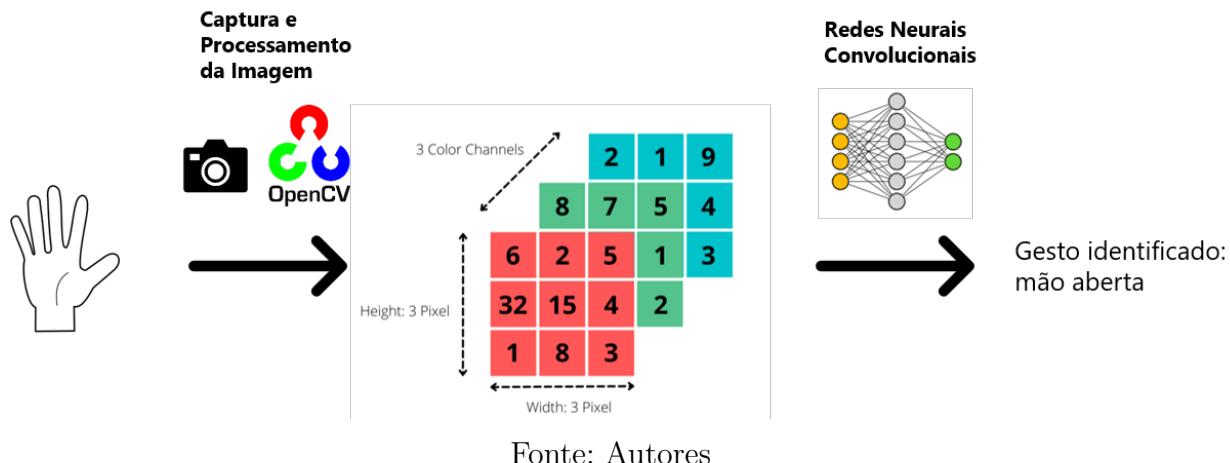
A visão computacional e suas ferramentas são o conceito chave para o desenvolvimento de uma solução que atenda a finalidade proposta acima. Dito isto, para que seja possível realizar o controle do braço robótico é necessário identificar diferentes gestos de mãos nas imagens que serão processadas por uma câmera ou webcam.

Para realizar o processamento de imagens há uma biblioteca de computação visual chamada OpenCV que torna possível a captação e manipulação de imagens. Por meio dela é possível capturar uma imagem como um vetor ou matriz de pixels, sendo que cada pixel contém informações sobre os valores das cores vermelho, verde e azul (de 0 a 255 para cada um deles) para o caso de uma imagem colorida em um sistema RGB, ou informações sobre os valores de escuridão e brancura (de 0 a 255 para ambos) para imagens em preto e branco.

Uma vez obtida a matriz da imagem é necessário processá-la com algum método computacional de forma que através de padrões nas informações dos pixels seja identificado gestos de mãos. Um forte candidato para este tipo de processamento são os algoritmos de aprendizado de máquina que vieram do estudo de reconhecimento de padrões e inteligência artificial. Dentro dos tipos de algoritmos de aprendizado de máquina, o ideal para o problema em questão são os algoritmos de aprendizado supervisionado, em que são apresentadas ao computador exemplos de entradas e saídas desejadas e o algoritmo por sua vez cria uma regra que relaciona às entradas e saídas de forma que quando exposto a uma nova entrada, ele consiga obter uma saída desejada que não foi explicitamente programada anteriormente. Dentre os algoritmos de aprendizado de máquina supervisionados um forte candidato para identificação de padrões em imagens são as redes neurais convolucionais, pois sua camada convolucional integrada reduz a alta dimensionalidade das imagens sem perder suas informações e por isso que elas são especialmente adequadas para esse caso de uso.

Com isso em vista, a primeira abordagem para solução do problema seria o desenvolvimento de um modelo de rede neural convolucional para receber como entrada uma matriz RGB da imagem obtida pelo OpenCV, processá-la e entregar como saída uma classificação do gesto feito pela mão, conforme o diagrama de exemplo abaixo:

Figura 20 – Primeira Arquitetura Proposta

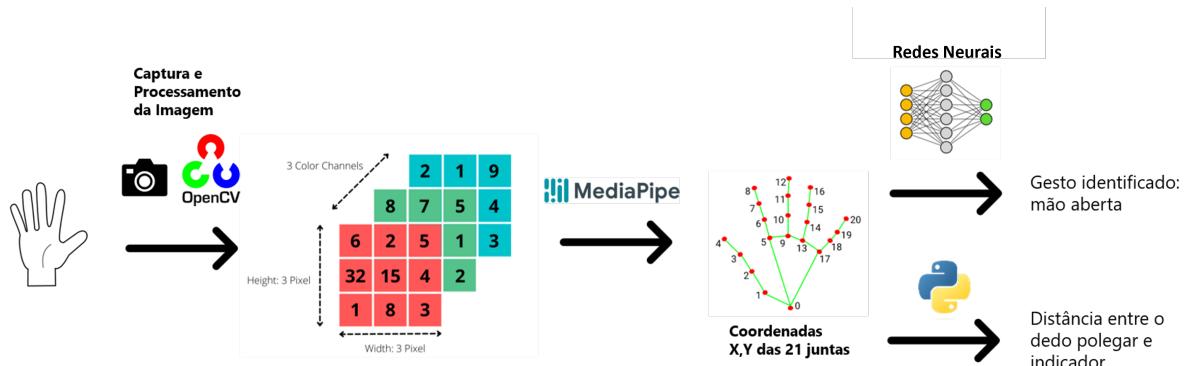


Ao tentar pôr em prática a arquitetura acima foi encontrado uma série de dificul-

dades, a primeira delas é que é um processo extremamente complexo realizar o treino de um modelo de redes neurais convolucionais para a identificação de mãos que seja genérico para quaisquer pessoa e formato de mão, pois seria necessário uma extensa base de dados catalogada com diversos formatos de mãos, tons de pele, ângulos, gestos entre outros, e além disso há a complexidade do fundo da imagem em questão, pois alguns modelos performavam bem em fundos simples mas para fundos com muitos objetos a identificação já se tornava menos acurada. Além de treinar por conta própria um modelo de rede convolucional, também foi tentado utilizar um serviço de Machine Learning para treinar modelos pré configurados chamado Teachable Machine, porém a falta de acurácia para identificação das mãos em fundos ou mãos que não as treinadas pelo modelo levaram à inviabilidade de seu uso.

Tendo em vista que o reconhecimento de mãos é um problema comum de visão computacional a abordagem final utilizada pelo grupo consiste na utilização de uma solução de Machine Learning pronta para o uso e personalizável para o reconhecimento das mãos. Optou-se por utilizar o MediaPipe, um framework multiplataforma que possui em uma de suas soluções o reconhecimento de mãos. A saída desse framework seria basicamente as coordenadas das juntas das mãos identificadas na imagem, que serviriam como insumo para um outro modelo de redes neurais perceptron multicamadas desenvolvido especificamente para identificar gestos à partir das coordenadas de entrada, removendo toda a complexidade de identificação das informações dos pixels, conforme arquitetura abaixo:

Figura 21 – Arquitetura Final



Fonte: Autores

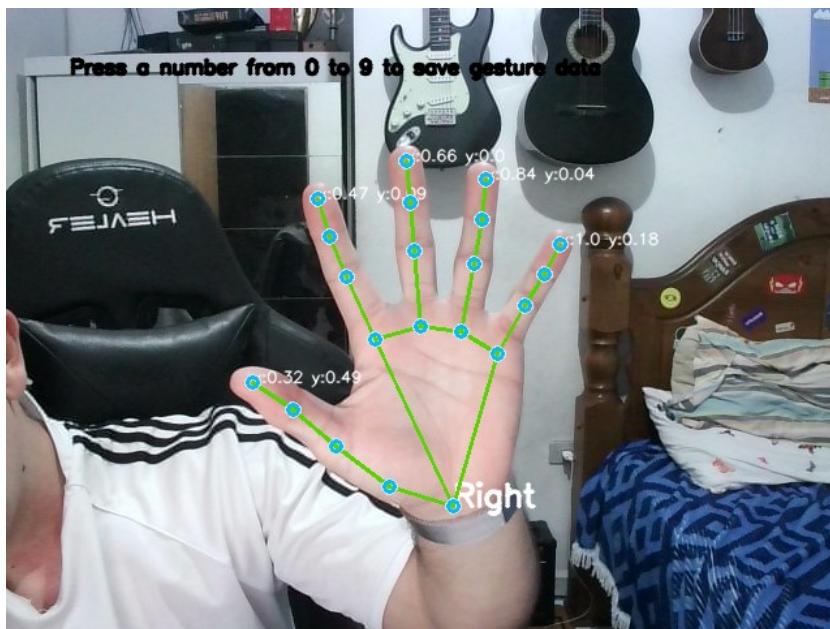
Posteriormente a esse processamento o gesto identificado é utilizado para indicar qual servo motor será acionado (cada articulação possui um servo) e a distância entre o dedo polegar e indicador é utilizado para indicar quantos graus o servo rotacionará, expandindo ou contraindo a junta.

3.3.3 OpenCV

O primeiro passo da arquitetura utilizada consiste na utilização do OpenCV para captura e processamento inicial da imagem, sendo uma biblioteca de computação visual open source, multiplataforma e totalmente livre ao uso acadêmico e comercial. É uma biblioteca desenvolvida nas linguagens de programação C e C++ porém da suporte para utilização também nas linguagens Java, Python e Visual Basic. O grupo optou pela utilização do Python como linguagem de programação para desenvolvimento do projeto e consequentemente foi utilizado a sua API com suporte para o Python.

O uso do OpenCV na arquitetura da solução consiste em extrair imagens em tempo real do dispositivo de vídeo conectado ao computador que o executa, seja uma webcam ou celular, converter essa imagem em uma matriz BGR e fornecer ferramentas para manipulação dessa matriz. Para que seja possível realizar o processamento pelo MediaPipe é necessário converter a imagem em RGB e invertê-la horizontalmente, passos que podem ser realizados com a utilização do OpenCV. Além disso ele permite escrever e desenhar em cima da imagem, podendo exibir as articulações identificadas na imagem e suas coordenadas, conforme exemplo abaixo:

Figura 22 – Exemplo de saídas na imagem feitas com OpenCV



Fonte: Autores

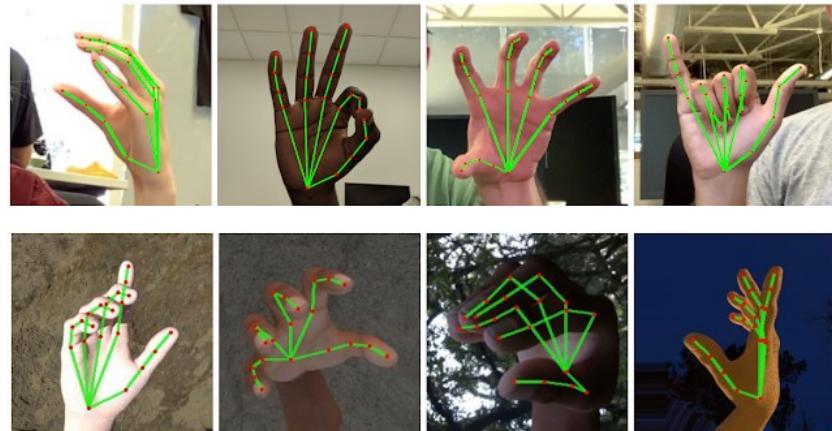
3.3.4 MediaPipe

O segundo passo da arquitetura utilizada consiste na utilização do MediaPipe para processar a matriz RGB obtida no passo anterior pelo OpenCV.

O MediaPipe é uma solução desenvolvida pelo Google de aprendizado de máquina customizável open source com soluções prontas para o uso, como por exemplo detecção de rostos, iris, corpo, mãos, cabelo, objetos entre outros.

A solução à ser utilizada no projeto é a MediaPipe Hands, que consiste no rastreamento de mãos e dedos inferindo as coordenadas de 21 juntas a partir de uma imagem, conforme exemplos abaixo:

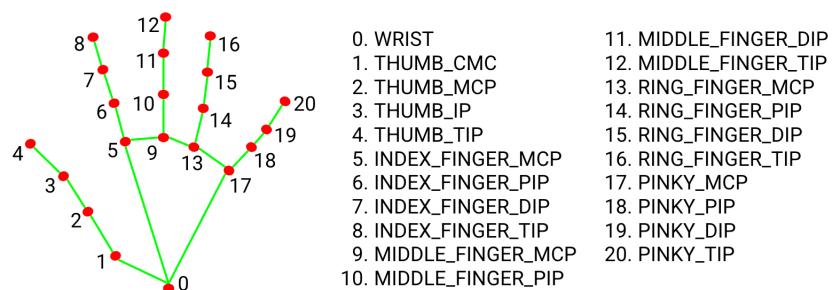
Figura 23 – Exemplo de Identificação das Juntas



Fonte: MediaPipe

Das soluções disponíveis dentro do MediaPipe Hands é utilizado no projeto o inferimento da destreza da mão, uma vez que é importante identificar qual mão é a esquerda para identificar os gestos e qual é a direita para controlar a extensão e contração da articulação, e o inferimento das coordenadas x e y das 21 juntas observadas na imagem abaixo:

Figura 24 – Exemplo de Identificação das Juntas



Fonte: MediaPipe

Essas coordenadas por sua vez são pré processadas para servir como entrada para o modelo de redes neurais responsável por identificar o gesto feito pela mão.

O controle da extensão e contração da articulação também faz uso dessas coordenadas, uma vez que utiliza como base a distância entre a junta 4 (THUMB_TIP) e a junta

8 (INDEX_FINGER_TIP).

A grande vantagem de utilizar o MediaPipe Hands é que é um modelo desenvolvido com alta acurácia para os mais diversos tipos e tons de mãos e fundos de imagens pois o modelo é treinado em imagens com iluminação variada e diversas condições de ruído e movimento, além de utilizar imagens capturadas de pessoas de 14 diferentes regiões do mundo para garantir diversidade nos tipos de mãos utilizadas para o treino do modelo.

3.3.5 Redes Neurais Perceptron Multicamadas

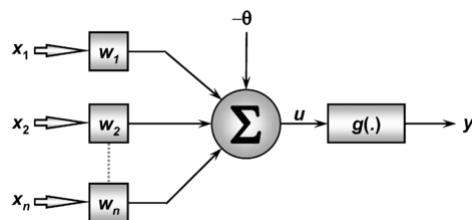
A última etapa da arquitetura consiste no desenvolvimento de um modelo de redes neurais perceptron multicamadas para, com base nas coordenadas x e y das 21 juntas processadas pelo MediaPipe, identificar qual foi o gesto realizado pela mão.

O modelo de rede neural perceptron de multicamadas, também conhecido em sua sigla em inglês como MLP, é um modelo de redes neurais que consiste na utilização de camadas de neurônios de entrada, saída e ocultas. O propósito dessa rede neural é modelar uma rede de neurônios similar à do nosso cérebro, em que a rede é composta por camadas de neurônios ligadas entre si por sinapses de peso.

Temos que um perceptron, também chamado de neurônio artificial, recebe valores de entradas, pesos para cada valor de entrada, um bias e uma função de ativação. A ideia é que a sua saída seja a combinação linear da soma da multiplicação dos sinais de entrada por seus pesos, somada do bias e então passada por uma função de ativação que é responsável por definir se o neurônio será ativo e propagará a sua informação para demais neurônios ou não:

$$y = g\left(\left(\sum_{i=1}^m W_i * x_i\right) + \theta\right)$$

Figura 25 – Perceptron

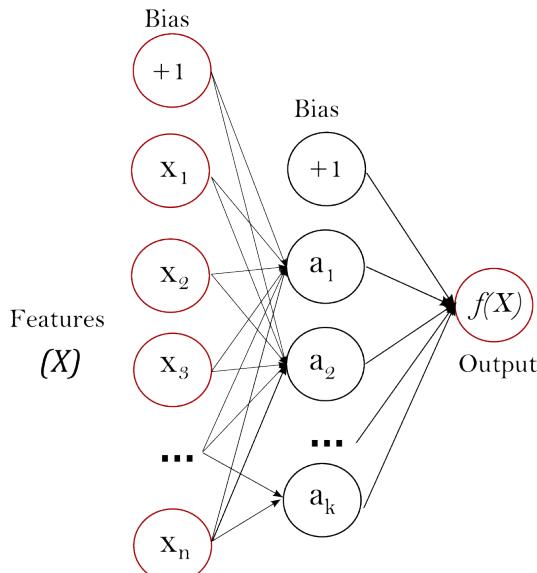


Fonte: Wikipedia

A rede neural perceptron multicamadas consiste em três ou mais camadas de perceptrons totalmente conectada em que cada perceptron se conecta com um perceptron da camada seguinte. Essencialmente o que diferencia um perceptron de outro são os pesos e bias associados na conexão, de forma que o treinamento de um modelo deste tipo consiste

na definição dos pesos e bias de cada conexão de perceptrons para que sejam obtidas as saídas desejadas dados as entradas definidas.

Figura 26 – Exemplo de Rede Neural Perceptron Multicamadas



Fonte: Scikit Learn

Para o projeto em questão a camada de entrada consistirá de 42 neurônios, sendo os valores x e y de cada uma das juntas, e dado as combinações lineares e conexões de cada perceptron nas camadas ocultas se obterá na camada final a ativação de um dos seis neurônios, indicando qual dos seis gestos treinados foi identificado para aquela combinação de coordenadas das juntas.

Um outro algoritmo que performa bem com dados tabulares como é o caso do projeto é o XGBoost, um algoritmo de aprendizado de máquina baseado em árvore de decisão e que utiliza uma estrutura de gradient boosting. Todavia foi optado por ser feito o uso de uma rede neural pois os dados de entrada são homogêneos, isto é, todos são do mesmo tipo (coordenadas), enquanto que o XGBoost usualmente performa melhor para dados heterogêneos, como por exemplo um modelo que receba como entrada idade, altura, localidade e outras variáveis de tipos distintos.

3.4 Ferramentas para Controle de Microcontroladores

O sistema para controle do braço robótico é composto por um conjunto de servos motores, uma câmera e um Arduíno para controle dos servos. Devido a maior documentação e facilidade de implementação da biblioteca para o reconhecimento de imagem na linguagem python, decidiu-se pela programação do Arduino via python também.

3.4.1 Arduíno

Nesse projeto o Arduíno é responsável apenas pela movimentação dos servos motores que fazem o deslocamento do pulso e dos dedos, controlando os mesmos de 0 até 180°, a partir da modulação de ciclo ativo de uma onda quadrada, como pode ser visto na figura 27

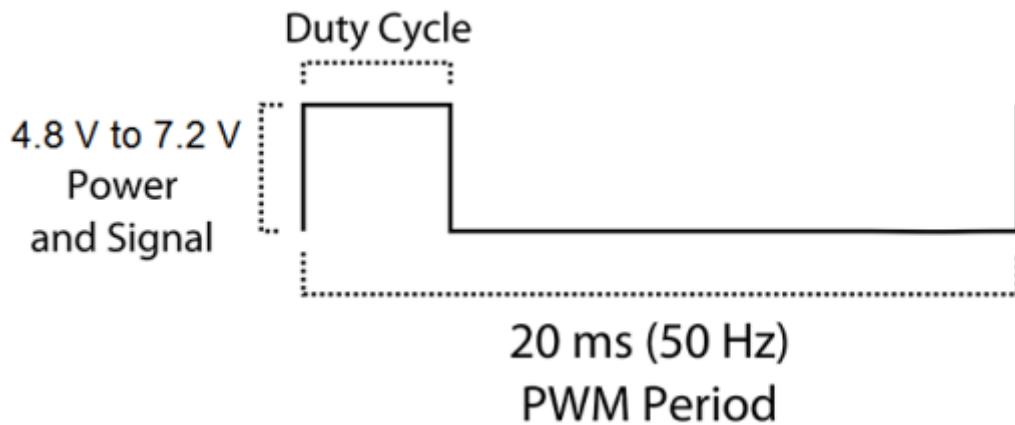


Figura 27 – Sinal PWM de controle do servo

Além disso, utilizou-se um protocolo de comunicação chamado Firmata no Arduíno, mais especificamente o Pyfirmata que é uma interface de comunicação Python com o microcontrolador, permitindo que um algoritmo em python realize comandos no microcontrolador. Possibilitando a integração do algoritmo de reconhecimento de imagem, que é totalmente desenvolvido em python, com os servos conectados ao Arduíno.

3.4.2 Esquema elétrico

O esquema elétrico A desse projeto separa a parte de maior consumo, os servos motores, de seu sistema de acionamento, as saídas digitais do Arduíno.

Cada servo possui uma corrente nominal de cerca de 350 mA, medida em bancada devido a falta de documentação específica desse componente, totalizando cerca de 2,1A de consumo nominal em caso de acionamento de todos os servos ao mesmo tempo. fazendo necessária uma fonte de alimentação de 5 V com pelo menos 3 A de corrente para funcionamento adequado de todos os servos. Foi descartado o uso do regulador de tensão do Arduíno, o NCP1117, que suporta no máximo 1A de corrente dependendo sua tensão de entrada.

O Arduíno é alimentado pela porta USB de programação que é conectado ao computador que roda o algoritmo de reconhecimento de imagem, essa conexão é suficiente para o funcionamento e acionamento dos servos, sendo necessário apenas a equalização de potencial do GND do Arduíno e da fonte.

3.4.3 Servo motores

Os servos motores utilizados no projeto são os MG995, um servo de alta velocidade com engrenagens metálicas e que permite a rotação entre 0 e 180°.

Os 5 dedos são controlados pelos servos motores, que atuam comprimindo ou estendendo as linhas de nylon, que atuam como tendões da mão, e assim realizam o movimento de abertura ou fechamento de cada dedo de forma independente. Além disso, o sexto servo motor faz a rotação do pulso, a partir de um sistema de engrenagens.

3.4.4 Python

A utilização do protocolo firmata e sua interface Pyfirmata para programação do microcontrolador, permitiu que toda a programação do projeto seja desenvolvida em Python, facilitando a integração do algoritmo de reconhecimento e os parâmetros de controle dos servo motores, além de permitir a utilização de diversas bibliotecas disponíveis para o python, diversificando ainda mais as funcionalidades do projeto.

4 Estrutura do Projeto

Neste capítulo é descrito todo o processo de desenvolvimento do projeto, uma vez que todo o embasamento bibliográfico foi realizado anteriormente, torna-se possível iniciar as atividades de integração e os testes de funcionalidade individuais e em conjunto atrelado ao projeto.

4.1 Impressão de componentes

A impressão de componentes do Projeto InMoov é realizada conforme orientações de modelagem 3D fornecidas pelo artista plástico responsável pelo projeto ([LANGEVIN, 2012](#)), conforme pré-estabelecido no início do projeto, a estrutura escolhida contempla a mão, pulso e antebraço.

A tabela 9 indica os gastos de tempo e gramas em material para o desenvolvimento do projeto. Destaca-se que ao todo foram gastas cerca de 116 horas em tempo de impressão, assim como uma estimativa de 750 gramas em material impresso. O material utilizado em questão é o PLA, devido a sua resistência mecânica e fácil manuseio para impressão 3D.

Tabela 9 – Tabela de componentes impressos

Qtd	Arquivo	Tipo	Preenchimento	Espessura (mm)	Suportes	Brim	Raft	Material (gramas)	Tempo (minutos)
1	Auriculaire3.stl	1	30%	1.5	N	N	N	8	100
1	Bolt_entreoise7.stl	5	30%	2	S	S	N	14	145
1	coverfinger1.stl	4	30%	2	S	N	N	14	143
1	Index3.stl	1	30%	1.5	N	N	N	13	145
1	Majeure3.stl	1	30%	1.5	N	N	N	16	172
1	ringfinger3.stl	1	30%	1.5	N	N	N	11	129
1	robcap3V2.stl	3	30%	2	N	N	N	31	249
1	robpart2V4.stl	2	30%	2	N	S	N	57	565
1	robpart3V4.stl	2	30%	2	N	S	N	44	395
1	robpart4V4.stl	2	30%	2	N	S	N	74	672
1	robpart5V4.stl	2	30%	2	N	S	N	93	873
1	thumb5.stl	3	30%	2	N	N	N	28	270
1	topsurfaceUP6.stl	4	30%	2	S	N	N	53	475
1	WristlargeV4.stl	3	30%	2	N	N	N	66	607
1	WristsmallV4.stl	3	30%	2	N	N	N	28	252
1	RobCableBackV3.stl	3	30%	2	N	N	N	8	69
1	RobCableFrontV3.stl	3	30%	2	N	N	N	16	131
1	RobServoBedV6.stl	3	30%	2	N	N	N	54	406
1	Servo-pulleyX5.stl	3	30%	2	N	N	N	11	111
1	CableHolderWristV5.stl	3	30%	2	N	N	N	6	59
1	RotaWrist1V4.stl	3	30%	2	N	N	N	49	467
1	RotaWrist2V3.stl	3	30%	2	N	N	N	25	221
1	RotaWrist3V3.stl	3	30%	2	N	N	N	12	107
1	WristGearsV5.stl	3	30%	2	N	N	N	11	135
Total								742	6898
Total em tempo de Impressão (horas)									116h 38 min

Fonte: Autores

Considerando os aspectos rotulados na tabela 9, destaca-se que em determinadas situações mostrou-se necessário o uso de elementos para fixação das peças na mesa de

impressão. Nestes casos, optou-se pelo uso alternado entre técnicas de *brim* e *raft*, conforme o perfil da peça e suas formas de configuração em relação a peça.

Figura 28 – Conjunto de peças impressas



Fonte: Autores

4.2 Estrutura desenvolvida

Partindo da impressão do conjunto de peças e posteriormente a separação para cada etapa da montagem, se iniciou o processo de fixação do grupo de componentes do braço, que por sua vez requeriram um processo de impressão separado por limitações de espaço e/ou de dimensões da mesa da impressora por conta do formato dos suportes necessários.

Figura 29 – Fixação necessárias nos dedos



Fonte: Projeto InMoov

Figura 30 – Fixação necessárias nos dedos



Fonte: [Projeto InMoov](#)

Figura 31 – Fixação necessárias no braço



Fonte: Autores

Com todas as peças devidamente fixadas, iniciou-se o processo de montagem da mão, sua forma inicial (ainda sem a adição dos dedos), pode ser identificada pela figura 32:

Figura 32 – Estrutura da mão



Fonte: Autores

Logo após isso foram feitas as juntas dos dedos, que consistem em pedaços de filamento de 1,75mm, com acabamento feito com o auxilio de um ferro de solda, que por sua vez podem ser vistos na figura 33:

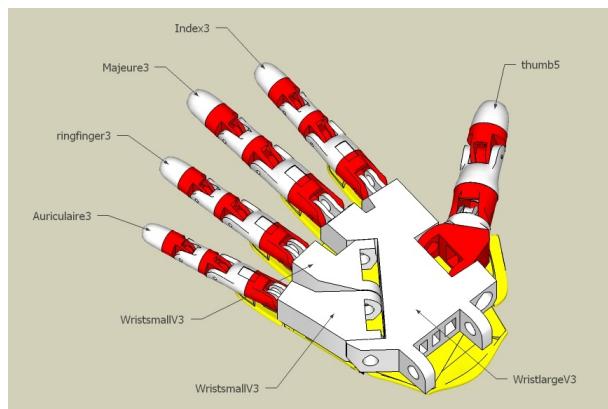
Figura 33 – Juntas dos dedos



Fonte: Autores

Com a junção de todos os dedos, a mão fica basicamente completa, faltando apenas a montagem do pulso e as ligações dos dedos nos servos, conforme a figura 34:

Figura 34 – Montagem da mão completa



Fonte: [Projeto InMoov](#)

Para a ligação do pulso é necessário um parafuso para travar na engrenagem com passagem dos fios e um pino para realizar a ligação ao restante da mão, como pode ser visto na figura 35.

Cada dedo é composto por um par de linhas de pesca de 0,35mm, um responsável pela compressão do dedo e um pela extensão. Antes do inicio da passagem dos fios é necessário realizar a medida estimada do tamanho necessário para cada dedo, considerando as posições finais e o suficiente para amarras na polia do servo motor (figura 36).

Os dois fios, de extensão e compressão, são amarrados nas pontas dos dedos conforme a figura 37.

Todas as peças que ligam os servos motores aos dedos possuem furos (figuras 38 e 39) de passagem dos cabos.

A movimentação do pulso é realizada por um par de engrenagens, que pode ser

Figura 35 – Montagem do pulso



Fonte: Autores

Figura 36 – Passagem de fios pelos dedos



Fonte: Autores

visto na figura 40, junto com a passagem dos cabos de controle dos dedos.

Após a passagem de todos os fios do pulso, foi feita a fixação do pulso conforme a figura 41, no restante da estrutura, seria possível ter feito essa fixação antes, mas a passagem dos cabos seria mais difícil.

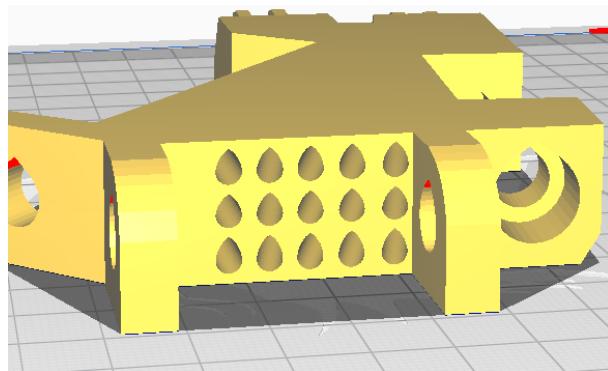
Os servos possuem um suporte para fixação no braço, como pode ser visto na figura 42 e figura 43.

Figura 37 – Arremate dos fios na ponta do dedo



Fonte: Autores

Figura 38 – Passagem de fios pela mão



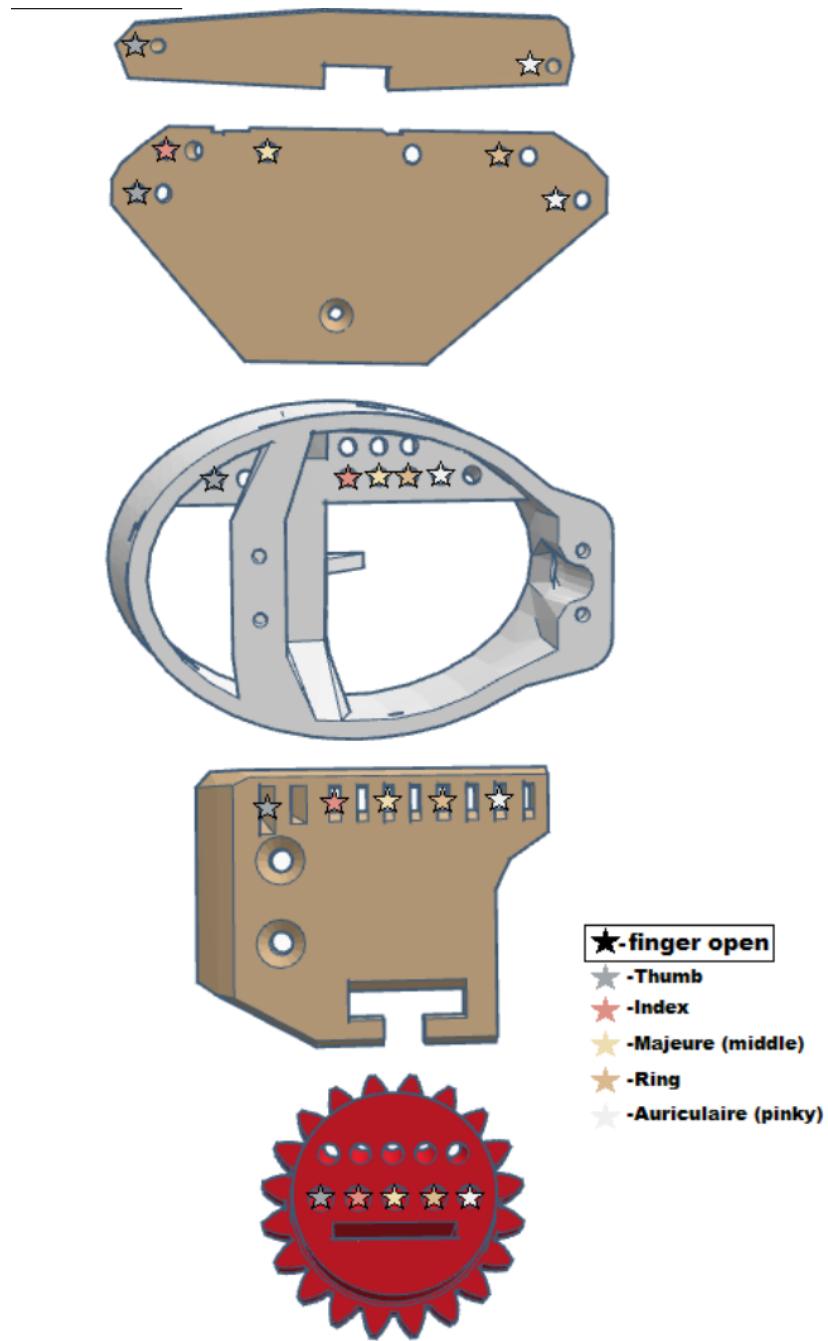
Fonte: Autores

Cada servo tem sua polia onde serão amarrados os fios de controle dos dedos, em cada polia (conforme a figura 45) vai o par de fios de um determinado dedo, a posição inicial do servo (figura 44) é com o fio que estende o dedo tensionado, ao rodar 90º no sentido anti-horário, o fio de compressão do dedo é tensionado enquanto o fio de extensão perde a tensão (figura 46), fazendo a atuação de cada dedo.

Após a conexão de todos os fios a montagem já está funcional, faltando apenas os acabamentos para fechar o braço.

E por fim, após a colocação dos acabamentos, como o fechamento do braço, pontas dos dedos e coberturas da mão, o conjunto de braço e mão finalizado pode ser visto nas figuras 47, 48, 49 e 50.

Figura 39 – Esquema de passagem de fios

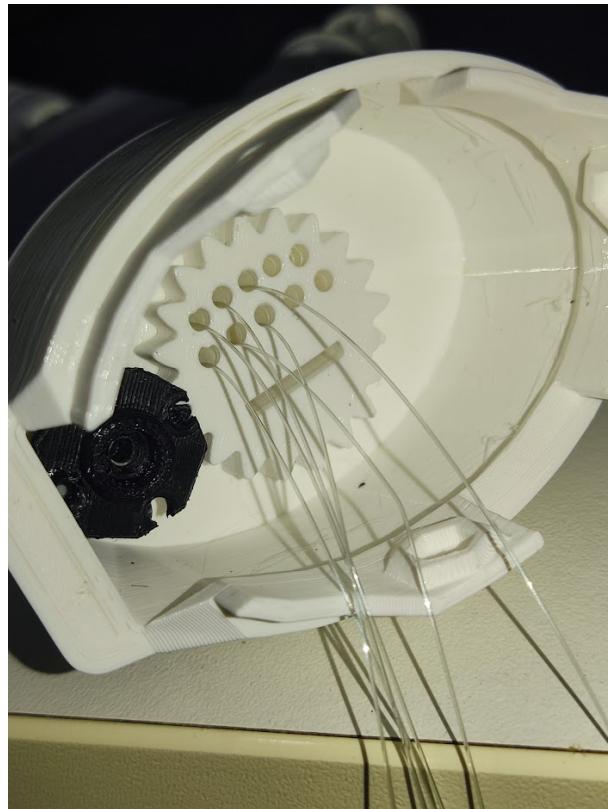


Fonte: [Projeto InMoov](#)

4.3 Computação Visual

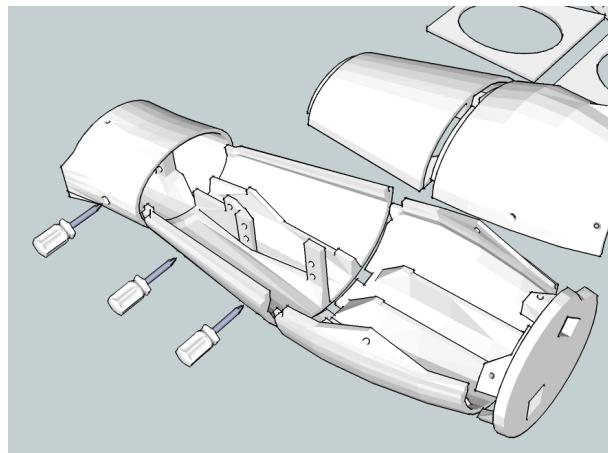
Como discutido no capítulo de ferramentas, a computação visual utiliza a arquitetura da figura 21 e consistem em três grandes etapas: identificação das coordenadas das juntas das mãos, reconhecimento de gestos e o controle da expansão e contração das articulações.

Figura 40 – Engrenagens do pulso e passagem de fios pelo pulso



Fonte: Autores

Figura 41 – Fixação do pulso no braço



Fonte: [Projeto InMoov](#)

4.3.1 Identificação das Coordenadas das Juntas das Mão

O primeiro passo para a identificação da coordenadas das juntas é o pré processamento da imagem:

Para que seja possível utilizar o MediaPipe é necessário primeiro capturar a imagem por meio de uma matriz RGB, converter essa matriz RGB em BGR e realizar uma inversão horizontal. Esse pré processamento é feito utilizando métodos do OpenCV e pode ser

Figura 42 – Suporte servos



Fonte: Autores

observado na imagem abaixo:

Os dados que serão interpretados pelo MediaPipe, por sua vez, é uma matriz como a observada abaixo:

O segundo passo é alimentar o modelo de rede neural convolucional do MediaPipe com a matriz acima:

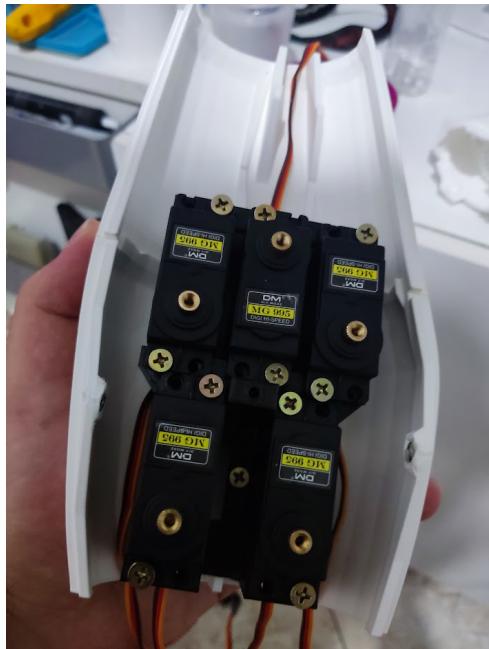
O MediaPipe por sua vez interpretará a imagem e devolverá duas respostas: as coordenadas das 21 juntas junto da destreza para cada mão identificada na imagem:

Essas coordenadas, por sua vez, possuem valores de 0 até 1 e representam o percentual da distância máxima daquele eixo que a coordenada possui. Isso torna as coordenadas agnósticas à resolução da câmera à ser utilizada, uma vez que para obter a coordenada real em pixels basta multiplicar o valor do eixo X pela largura do vídeo e o valor do eixo Y pela altura do vídeo.

4.3.2 Reconhecimento de Gestos

O reconhecimento de gestos por meio de um modelo de redes neurais consiste na terceira etapa da arquitetura da solução proposta:

Figura 43 – Servos fixados



Fonte: Autores

Figura 44 – Servos com polias



Fonte: Autores

4.3.2.1 Coleta de Dados de Treino

A primeira etapa para que seja possível o desenvolvimento de um modelo de redes neurais para o reconhecimento de gestos é a coleta de dados de treinos categorizados, isto é, uma base de dados com as entradas e a saída esperada para aquelas entradas, de forma que o modelo possa ser treinado para prever a saída de entradas novas.

Figura 45 – Fios conectados as polias



Fonte: Autores

Figura 46 – Fios tensionados



Fonte: Autores

A saída que o MediaPipe fornece são as coordenadas em largura, altura e profundidade de cada junta, entretanto para o reconhecimento de gestos a profundidade é irrelevante pois um gesto será o mesmo independente da profundidade avaliada, com isso os dados de entrada do modelo serão as coordenadas x e y de cada uma das 21 juntas, totalizando 42 entradas para resultar na classificação de um dos seis gestos possíveis, conforme exemplo abaixo:

Figura 47 – Montagem final



Fonte: Autores

Figura 48 – Montagem final



Fonte: Autores

Entretanto não se pode utilizar as coordenadas cruas como entrada do modelo pois a posição da mão na imagem alteraria as coordenadas ainda que se mantenha o mesmo gesto, tornando praticamente inviável o treinamento de um modelo para todos os casos possíveis de coordenadas para cada gesto. Um exemplo pode ser observado abaixo, em que as coordenadas da ponta do dedo médio variam muito em uma mudança de posição utilizando o mesmo gesto de mão aberta. No exemplo já foi feita a multiplicação das

Figura 49 – Montagem final



Fonte: Autores

Figura 50 – Montagem final

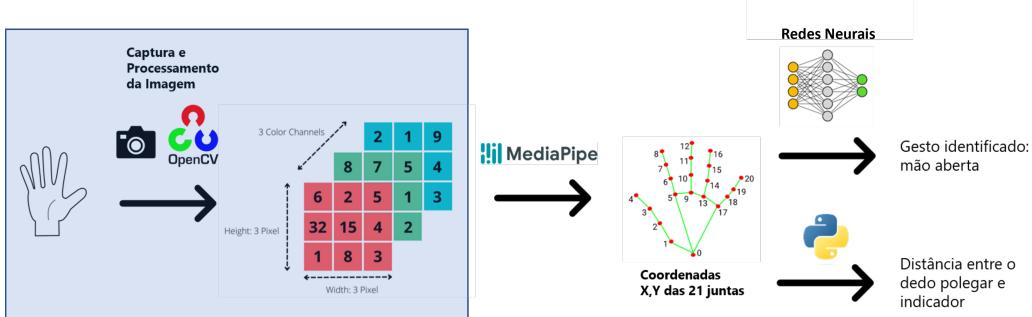


Fonte: Autores

coordenadas relativas do MediaPipe pela largura e altura do vídeo, para que sejam obtidas as coordenadas reais em pixels:

O tipo de dado de entrada que se deseja para treinar o modelo é uma coordenada que varie apenas quando ocorra uma mudança de gesto e uma forma de se conseguir isso é utilizar coordenadas relativas. Inspirado pela solução desenvolvida em ([KINIVI](#),)

Figura 51 – Primeira Etapa Arquitetura



Fonte: Autores

Figura 52 – Pré Processamento



Fonte: Autores

Figura 53 – Parte da Matriz BGR gerada no passo 3 do pré processamento

```
array([[129, 132, 123],
       [138, 141, 133],
       [134, 136, 131],
       ...,
       [160, 144, 124],
       [160, 145, 123],
       [160, 145, 123]],

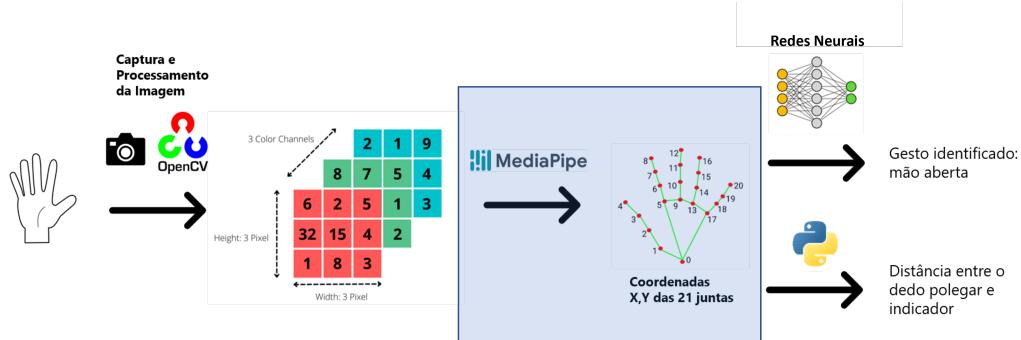
      [[132, 135, 126],
       [138, 140, 133],
       [132, 133, 129],
       ...,
       [161, 145, 126],
       [160, 144, 124],
       [160, 145, 124]],
```

Fonte: Autores

a estratégia utilizada para processar as coordenadas é realizar a subtração de todas as coordenadas pelas coordenadas do pulso, tornando a coordenada do pulso a origem (0,0) e todas as demais coordenadas em relação à ela, conforme exemplo abaixo:

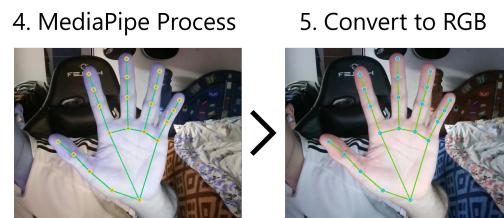
Após a obtenção das coordenadas relativas, uma boa prática para que o modelo performe melhor é realizar a normalização dos dados de entrada. O objetivo da normalização é alterar os valores das colunas numéricas para uma escala comum sem causar a distorção nos intervalos dos valores. Isso é útil para o modelo de redes neurais perceptron em questão, pois ele utiliza combinações lineares das entradas associando pesos às mesmas, e como as coordenadas podem ter diversos intervalos distintos em seus valores, como por exemplo a coordenada do dedo do meio ter valores na faixa de 0 a 100 para a coordenada Y e o dedo polegar de 300 a 400, temos que a coordenada do dedo polegar pode influenciar

Figura 54 – Segunda Etapa Arquitetura



Fonte: Autores

Figura 55 – Processamento MediaPipe



Fonte: Autores

Figura 56 – Exemplo de saída para imagem do passo 4

```

|: results.multi_handedness
executed in 14ms, finished 16.05.06 2022-11-04
|: [classification {
  index: 1
  score: 0.9415285587310791
  label: "Right"
}]

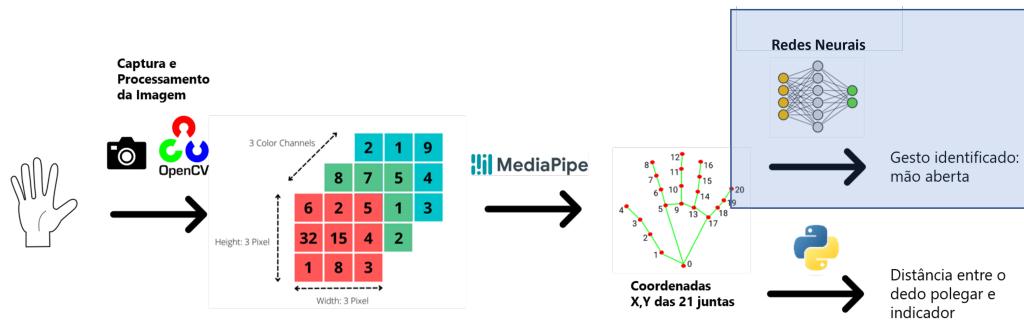
|: results.multi_hand_landmarks
executed in 17ms, finished 16.04.57 2022-11-04
|: [landmark {
  x: 0.538591206073761
  y: 0.9306017160415649
  z: 9.418806143912661e-07
}
landmark {
  x: 0.41028764843940735
  y: 0.8931456804275513
  z: -0.06407913565635681
}
landmark {
  x: 0.31494826078414917
  y: 0.7892177104949951
  z: -0.1003132164478302
]
  
```

Fonte: Autores

bastante o resultado devido aos seus valores maiores e não necessariamente porque ela é mais importante como um preditor.

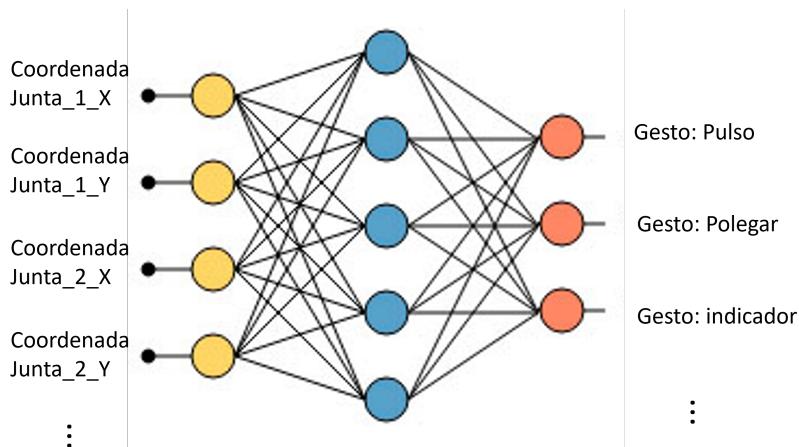
Dito isto, o método utilizado para realizar a normalização das coordenadas é a normalização mínima-máxima, em que eu coloco todas as variáveis em valores entre 0 e 1 a partir da seguinte fórmula:

Figura 57 – Terceira Etapa Arquitetura



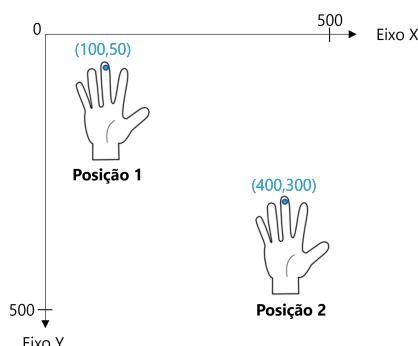
Fonte: Autores

Figura 58 – Exemplo entrada e saída da rede neural



Fonte: Autores

Figura 59 – Exemplo de Entrada Sem Processamento

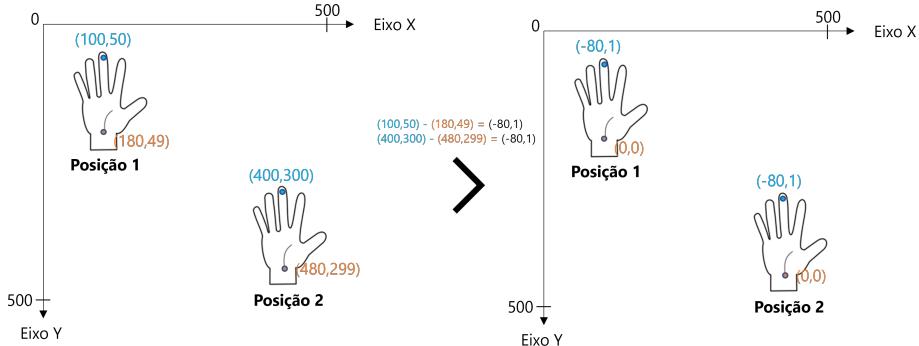


Fonte: Autores

$$\text{coordenada}_{\text{escalada}} = \frac{\text{coordenada} - \text{coordenada}_{\text{mínima}}}{\text{coordenada}_{\text{máxima}} - \text{coordenada}_{\text{mínima}}}$$

Antes de aplicar a normalização, transformamos o vetor anterior de 21 colunas contendo um array interno com as coordenadas x e y em um único vetor com 42 colunas, de forma que a normalização utilize a coordenada máxima e mínima de todo o conjunto de

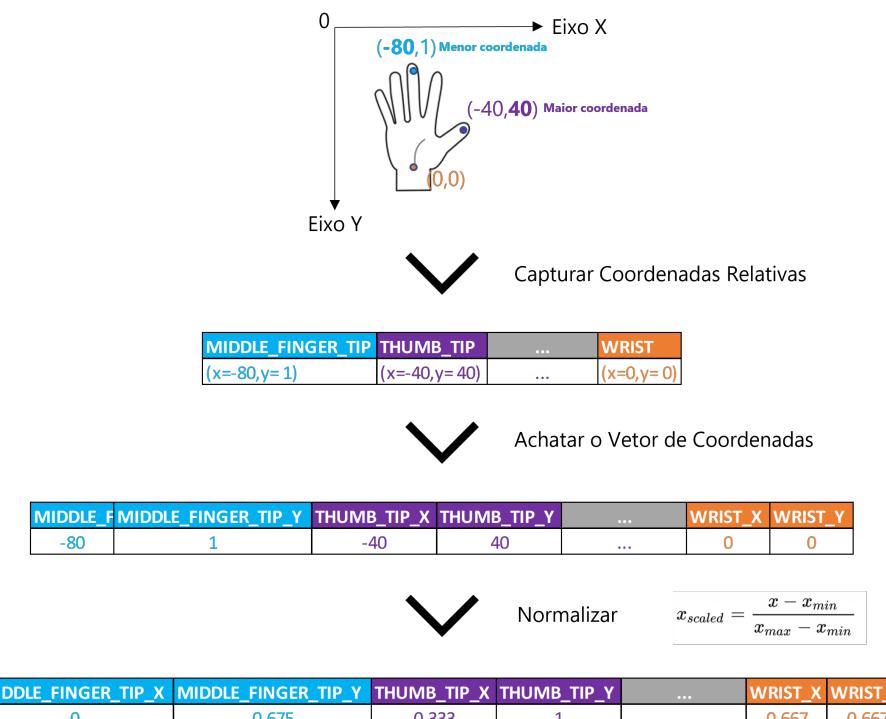
Figura 60 – Exemplo de Coordenada Relativa



Fonte: Autores

coordenadas. Com isso, teremos como dados de entrada as coordenadas da seguinte forma:

Figura 61 – Exemplo de Normalização



Fonte: Autores

Com este pré processamento definido, para realizar a captura das 42 coordenadas e associá-las à um rótulo, foi utilizada à seguinte tabela:

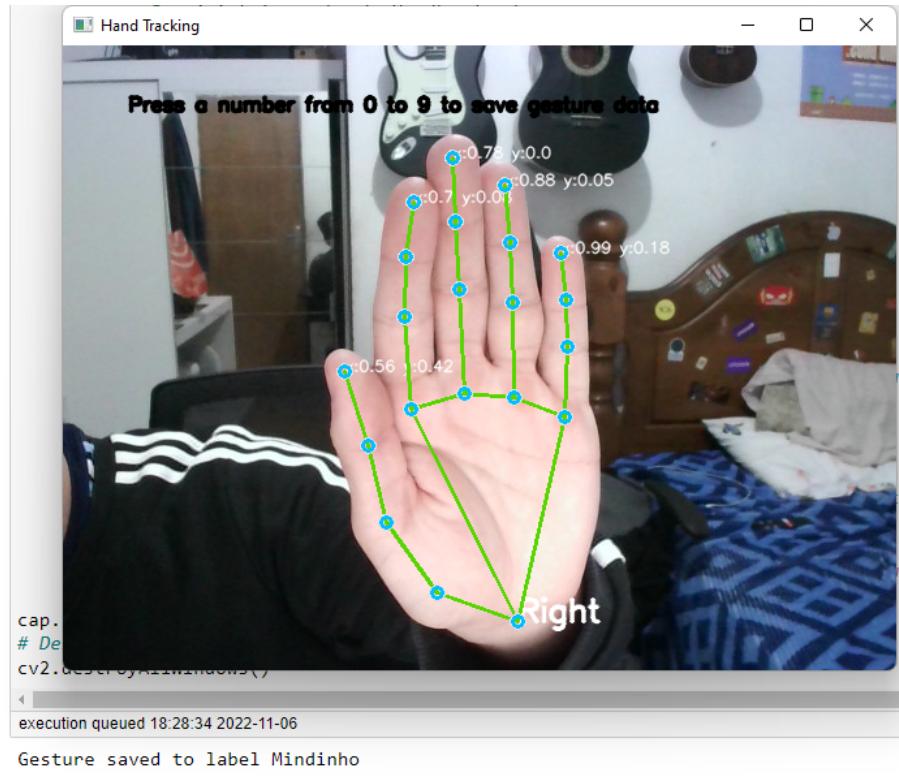
E com a utilização do MediaPipe, OpenCV e Python no script do apêndice A, foi desenvolvido um código que abre a câmera de vídeo do computador, desenha na imagem as juntas das mãos e as coordenadas relativas das pontas dos dedos (apenas como um exemplo dos dados que serão capturados) e instrui o usuário à apertar um dígito de 0 a 9. Ao realizar isso, o código salvará dentro de um arquivo csv as 42 coordenadas das juntas associadas ao gesto no feito no momento em que o dígito foi pressionado e o

Rótulo	Gesto
0	Pulso
1	Polegar
2	Indicador
3	Medio
4	Anelar
5	Mindinho

Tabela 10 – Rótulos dos Gestos

associará ao número pressionado, de forma que para capturar os gestos que representam a movimentação da articulação do pulso, por exemplo, é necessário fazer o gesto equivalente ao pulso na tabela 10 e pressionar o número equivalente ao pulso na tabela 8.

Figura 62 – Exemplo de execução do código do apêndice A



Fonte: Autores

Figura 63 – Exemplo de Dado Salvo

gesture_id	wrist_x	wrist_y	thumb_cmc_x	thumb_cmc_y	pinky_tip_x	pinky_tip_y
5.0	0,895	0,895	0,743	0,848	0,996	0,191

Fonte: Autores

Um ponto importante à ser observado é que uma vez que o modelo é treinado com os dados pré processados, esse mesmo pré processamento ocorre a cada captura de imagem no script do apêndice C quando se utiliza o modelo para identificar o gesto realizado.

4.3.2.2 Treinamento do Modelo

Para o treinamento do modelo foi utilizado a biblioteca do Python scikit-learn, que possui diversas ferramentas para aprendizado de máquina. Dentro delas foi utilizado o modelo MLPClassifier, que implementa o modelo de redes neurais perceptron de múltiplas camadas. Ao utilizar-se esse modelo foi definido os seguintes parâmetros:

- Tamanho das camadas escondidas: se refere à quantidade de neurônios artificiais que serão utilizados em cada uma das camadas escondidas. Para o modelo treinado foi utilizado cinco camadas de tamanhos 20, 15, 13, 10 e 8.
- Função de ativação: é a função de ativação à ser utilizada para definir se um neurônio será ativo ou não. Para o modelo treinado foi utilizado a função "relu", que basicamente zera valores negativos e tem como resultado o próprio valor de entrada caso ele seja positivo.
- Resolvedor: se refere ao algoritmo que será utilizado para encontrar os pesos e bias dos neurônios com base nos dados de treino. Para o modelo treinado foi utilizado o método de gradiente descendente. Esse parâmetro é importante pois dependendo da quantidade de dados de entrada utilizar um resolvedor diferente pode otimizar o tempo de treino.
- Taxa de aprendizado: se refere ao quanto o modelo atualizará os parâmetros a cada iteração de treino. Para o modelo treinado foi utilizado uma taxa de aprendizado constante de 0.001.
- Número máximo de iterações: indica quantas iterações serão feitas durante o treino. A cada iteração o modelo procura se enquadrar melhor aos dados de entrada, conforme altera seus parâmetros de pesos e bias. Para o modelo treinado foi utilizado 10000 iterações.

O código com a definição dos parâmetros acima e treinamento do modelo está disponível no apêndice B.

A base de treinamento consiste em 651 registros de coordenadas, sendo divididos em:

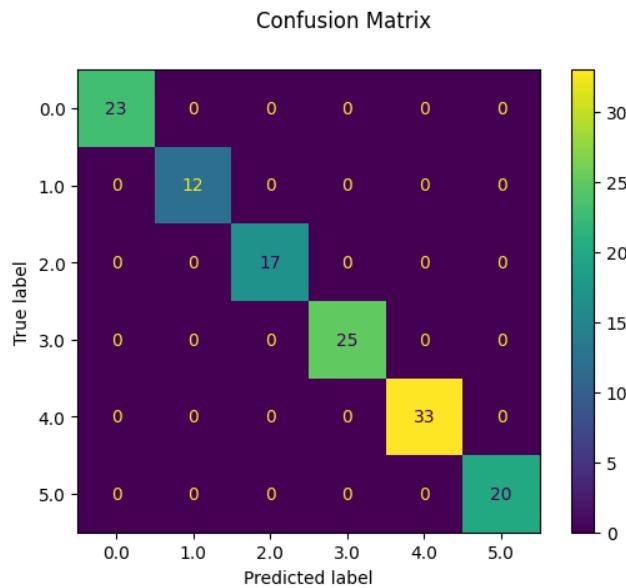
- 106 registros para o gesto "Pulso"
- 65 registros para o gesto "Polegar"
- 96 registros para o gesto "Indicador"
- 109 registros para o gesto "Médio"

- 155 registros para o gesto "Anelar"
- 120 registros para o gesto "Mindinho"

A captura de dados inclui a realização dos gestos em distintas profundidades, inclinações e pequenas rotações, de forma que seja mais fácil para o operador ter o gesto desejado identificado.

Para o treinamento do modelo foi utilizado 80% dos dados da base de treinamento para o treino (521 registros) e 20% para teste (130 registros). Isso serve para verificar se o modelo não sofrerá "overfitting", isto é, ficará tão ajustado aos dados em que foi treinado que não conseguirá fazer previsões adequadas para dados que nunca viu. Ao testar o modelo nestes 130 registros rotulados não utilizados no treino, observá-se que ele previu adequadamente o gesto realizado para todos os casos, conforme a seguinte matriz de confusão:

Figura 64 – Matriz de Confusão Obtida Após o Treinamento do Modelo



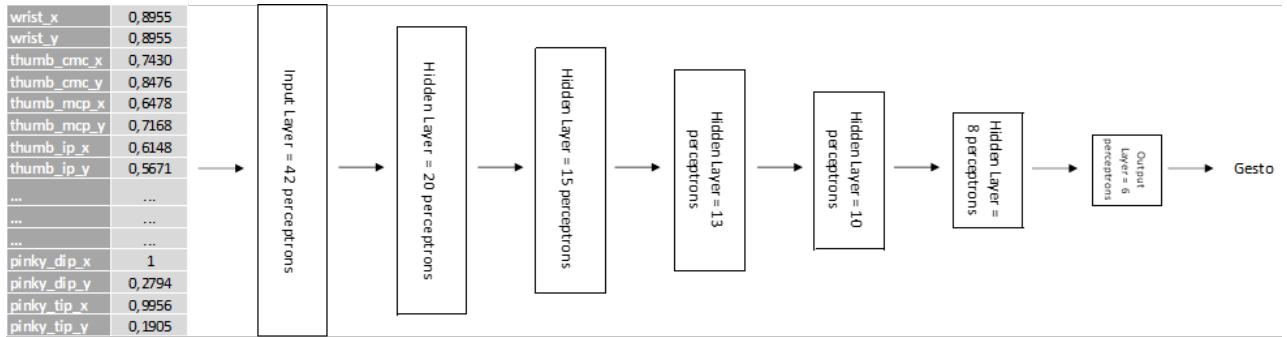
Fonte: Autores

O modelo treinado ficou portanto com a seguinte estrutura:

O último passo após o treinamento do modelo é salvá-lo de forma que seja possível exportá-lo e utilizá-lo no código final para identificar os gestos com base nas imagens em tempo real.

Na prática o modelo é um objeto em Python que obteve seus parâmetros configurados com base nos dados e treino e que, para utilizá-lo, basta chamar o método "predict" com novos dados de entrada. Uma forma de exportá-lo é através de arquivos pickle, que por sua vez servem para converter os objetos em memória durante a execução de um código para um fluxo de bytes que pode ser armazenado na memória em disco do computador, de

Figura 65 – Estrutura do Modelo



Fonte: Autores

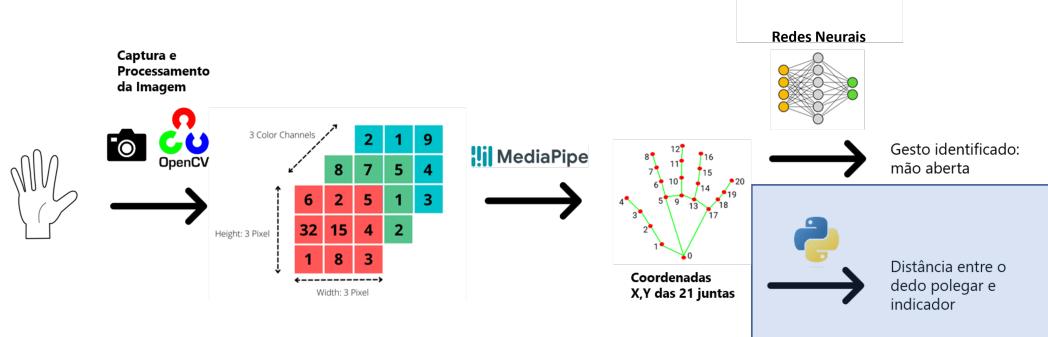
forma que ao carregar esse fluxo de bytes novamente no código ele possa ser desserializado de volta para um objeto em Python.

Para exportação do modelo treinado em um arquivo pickle foi utilizado a biblioteca joblib que oferece tanto o método para salvá-lo quanto para lê-lo novamente.

4.3.3 Controle da Expansão e Contração da Articulação

O cálculo da distância entre o dedo polegar e o indicador consiste na quarta e última etapa da arquitetura da solução proposta:

Figura 66 – Quarta Etapa Arquitetura

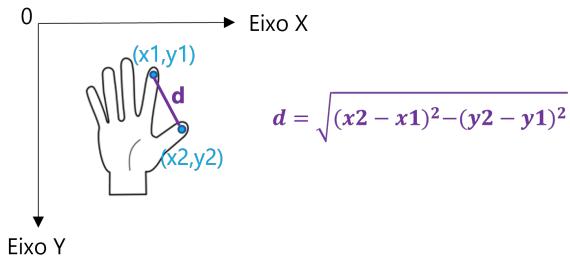


Fonte: Autores

Essa distância é responsável por indicar com o auxílio do gesto identificado o quanto uma articulação irá se expandir ou contrair. A expansão da articulação é feita com base no ângulo de rotação do servo motor, sendo a ideia na parte da computação visual obter um percentual entre 0% e 100% indicando o quanto da expansão máxima se deseja obter, sendo 100% a saída para quando se deseja a articulação completamente esticada e 0% quando se deseja completamente contraída. Esse percentual por sua vez é utilizado para definir o ângulo em que o servo motor será rotacionado de forma a transmitir essa expansão para a articulação do braço robótico, como se pode observar no código disponível no apêndice C.

A distância entre as pontas dos dedos polegar e indicador pode ser calculada utilizando as coordenadas desses dois pontos obtidos através do MediaPipe utilizando-se o teorema de Pitágoras:

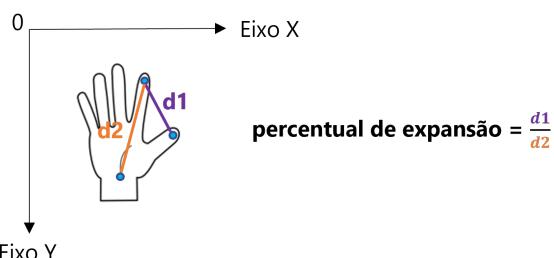
Figura 67 – Cálculo Distância Polegar Indicador



Fonte: Autores

Temos que essa distância é dada como um número escalar em pixels. Para torná-la um percentual de 0% a 100%, sendo 0% quando os dedos estiverem encostados em 100% algo próximo da distância máxima que o usuário consiga fazer entre eles foi calculado também a distância entre o pulso e a ponta do dedo do dedo indicador e então feito o quociente de ambas:

Figura 68 – Percentual de Expansão



Fonte: Autores

Para os casos em que a distância entre a ponta do dedo polegar e indicador superarem a distância entre o pulso e a ponta do dedo indicador é forçado como saída o valor de 100%. A ideia se utilizar essa razão como saída é motivada por deixar um percentual genérico à qualquer mão que for utilizada, uma vez que são distâncias relativas à mão do operador, e também por conta de não existir uma distância máxima entre a ponta do dedo polegar e indicador, variando de pessoa para pessoa o tanto que é possível distanciá-los. Com essa abordagem foi possível atingir diversos graus de expansão das articulações, conforme pode ser observado nos exemplos das figuras 17, 18 e 19 do capítulo 3.3.

4.4 Processo de Conexão entre Interfaces

4.4.1 Interface entre algoritmo de reconhecimento e braço robótico

A interface de controle do microcontrolador para atuação do braço e o algoritmo que faz o reconhecimento dos gestos são desenvolvidos na mesma linguagem e podem rodar em conjunto, o que simplifica a conexão entre os mesmos. Cada gesto representa a atuação de um servo, para cada servo temos as seguinte informações:

Figura 69 – Parâmetros para cada gesto

```
# Dictionary containing
articulation_dict = {
    'Mindinho' : {
        'pin': 2,
        'fator': 1.3
    },
    'Anelar' : {
        'pin': 3,
        'fator': 1
    },
    'Medio' : {
        'pin': 4,
        'fator': 1.4
    }
}
```

Onde *pin* é o pino de controle do servo motor e fator é o um multiplicador para correção do quanto a mais o servo deve rodar para compensar os fios menos tensionados, cada servo deve rodar cerca de 90º para realizar o movimento completo, entretanto o servo tem liberdade para rodar até 180º e isso é utilizado para compensar o tensionamento das cordas.

A atuação dos motores é feita a partir das informações que podem ser vistas na figura 69 e da multiplicação do escalar explicado no capítulo 4.3.3, onde para os dedos, 0% representa o servo motor a 90º e o dedo totalmente contraído e 100% o servo a 0º e o dedo totalmente estendido, para o pulso a escala representa apenas uma leve rotação no pulso, mas o comportamento do servo é mantido.

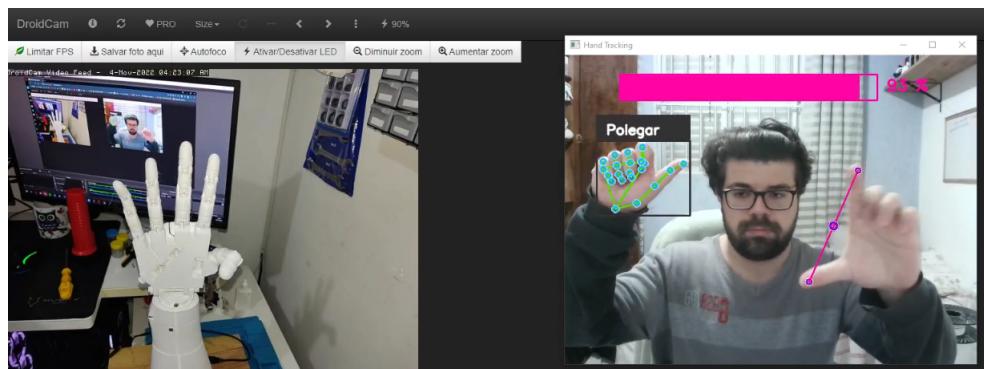
5 Resultados e Discussão

5.1 Consolidação sistêmica

Uma vez consolidado o conjunto de montagem estrutural (mecânico e elétrico) e também os aspectos sistêmicos da operação, pôde-se realizar o conjunto de testes finais de consolidação do protótipo.

A figura 70 ilustra de forma funcional o resultado de consolidação em que o braço mecânico possui a capacidade de assimilar em tempo real o processo de gesticulação da mão humana. O fator de captura e reconhecimento de imagem se dá com base no conjunto anteriormente treinado pela rede neural, garantindo assim o movimento dos dedos.

Figura 70 – Teste de Integração do Polegar

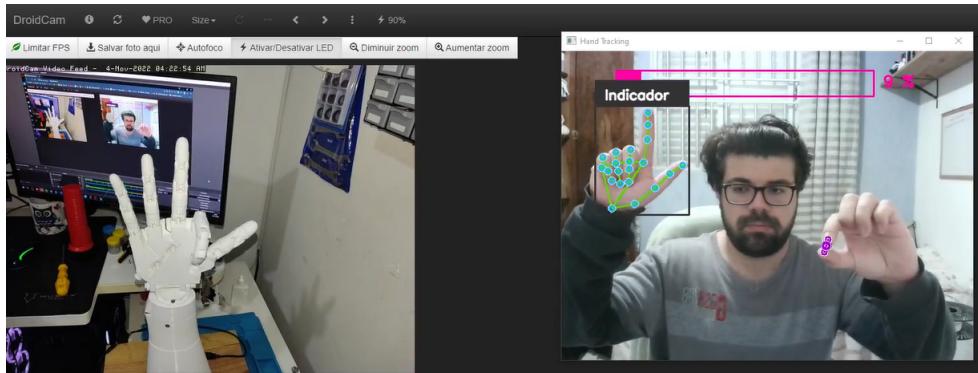


Fonte: Autores

Uma vez efetuado o movimento, a mão permanece aguardando novas instruções, baseando-se continuamente em interpretação do posicionamento dos dedos e comparação com a mão que fora modelada.

A figura 71 sintetiza de forma análoga ao item anterior a capacidade de resposta do sistema, assim como o conjunto de referências que a rede neural toma para assumir o novo posicionamento dos dedos.

Figura 71 – Teste de Integração do Indicador



Fonte: Autores

Vale destacar que junto ao processo de integração do dispositivo, todo o processo eletromecânico, i.e., o controle de motores é realizado simultaneamente por meio dos comandos elétricos fornecidos pelo microcontrolador aplicado aos servo motores.

5.2 Integração do sistema com dispositivo eletromecânico

Em relação a atuação dos servo motores para movimentação de cada dedo, pode-se destacar que a montagem e ajuste de cada conjunto é personalizado, uma vez que a faixa de tensão para atuação de cada dispositivo acaba sendo diferente conforme detalhes de cada montagem.

Tendo isso em vista, tornou-se necessário o uso de um fator de multiplicação para atuação de cada dedo, resultando em um ângulo de rotação um pouco maior do que fora inicialmente previsto. No entanto, tal resultado apenas garantiu o tensionamento do fio de modo a completar o movimento do dedo previsto com base na teoria.

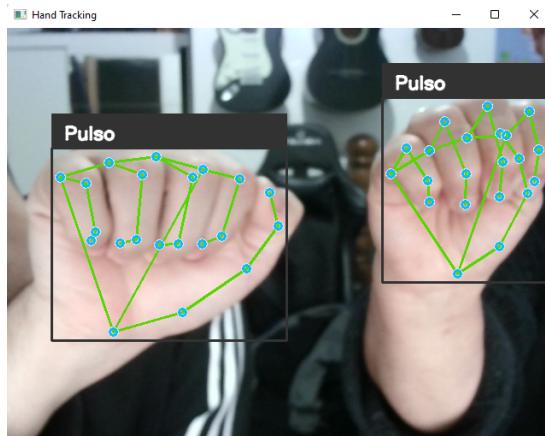
Ademais, houve uma certa preocupação com o tempo de resposta dos motores em relação aos comandos fornecidos pelo algoritmo, entretanto, por ser tratarem de dispositivos de alta velocidade, ao longo dos procedimentos experimentais descartou-se tal preocupação, devido ao desempenho dos componentes eletrônicos e sua ação conjunta nos movimentos da mão.

5.3 Computação Visual

5.3.1 Reconhecimento de Gestos

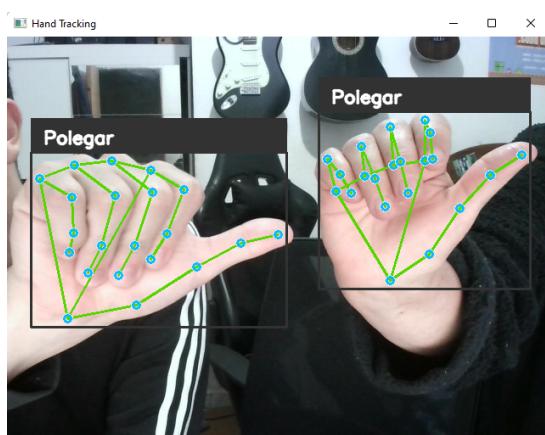
Com relação ao reconhecimento de gestos da tabela 8 o modelo performou bem mesmo em mãos que não foi treinado, conforme observado nas imagens abaixo:

Figura 72 – Identificação Pulso



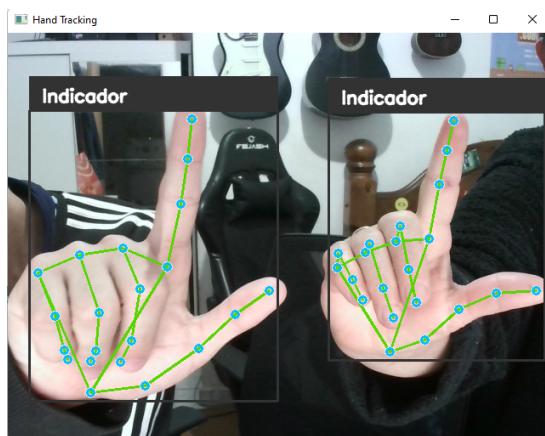
Fonte: Autores

Figura 73 – Identificação Polegar



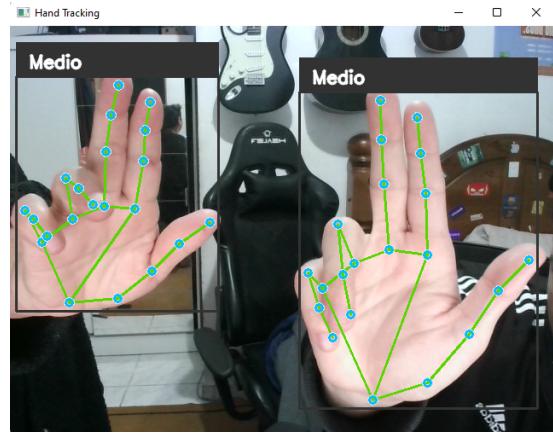
Fonte: Autores

Figura 74 – Identificação Indicador



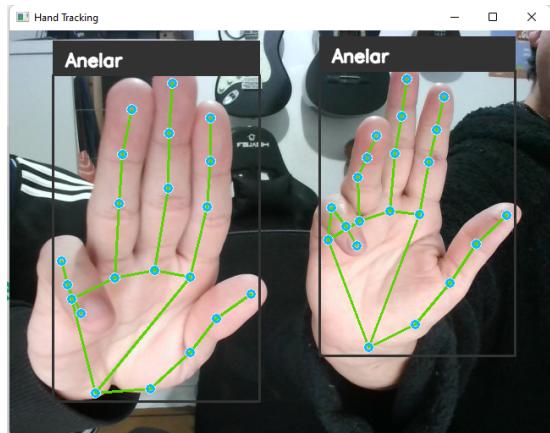
Fonte: Autores

Figura 75 – Identificação Médio



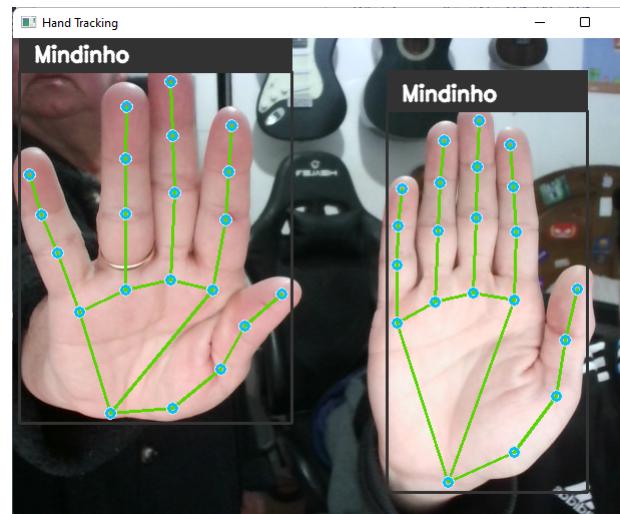
Fonte: Autores

Figura 76 – Identificação Anelar



Fonte: Autores

Figura 77 – Identificação Mindinho



Fonte: Autores

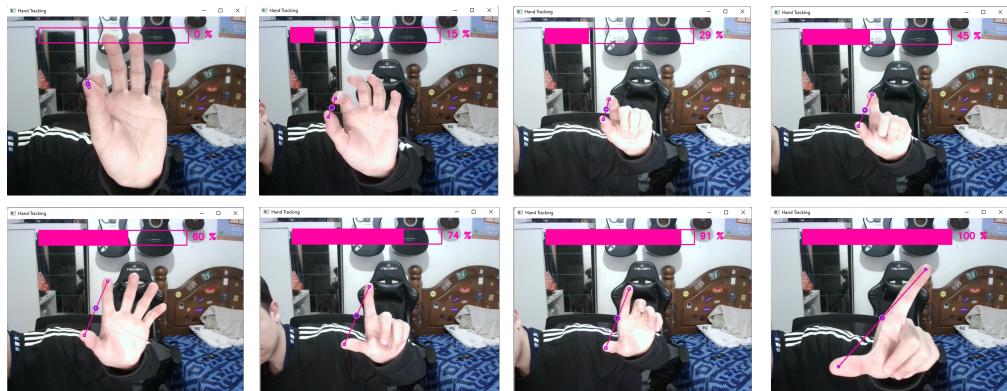
Há uma flexibilidade no gesto à ser realizado com a mão, como pode ser observado na figura 77 em que uma mão possui os dedos mais encostados e a outra um pouco mais espaçados, além de admitir algumas rotações com relação ao ângulo de observação da câmera como pode ser observado na figura 72 em que uma das mãos não está totalmente alinhada à câmera.

Dentre os gestos utilizados o modelo obteve um pouco de dificuldade em distinguir o pulso e polegar pois há apenas uma sutil diferença na coordenada x da ponta do dedo polegar entre ambos os gestos. O modelo acaba falhando para gestos intermediários entre ambos, como por exemplo deixar o dedo polegar um pouco mais distante da mão ao realizar o gesto para o pulso, entretanto não é um problema para o operador pois a realização do gesto de forma precisa o torna inteiramente identificável pelo modelo.

5.3.2 Controle da Expansão e Contração da Articulação

Para expansão e contração da articulação, pode-se observar que a abordagem utilizada captura bem diversos graus de expansão, conforme observado nas imagens abaixo:

Figura 78 – Diversas Distâncias Entre os Dedos



Fonte: Autores

Uma dificuldade encontrada é com relação à contração total, isto é, atingir o 0%, pois sempre há uma distância por menor que seja entre os dedos. Entretanto uma solução para isso é dar uma leve inclinação na palma da mão para cima, como pode ser visto na primeira figura do canto superior esquerdo da figura 78, pois dessa forma a distância entre o pulso e o indicador aumenta e ajuda a zerar o quociente. De resto, é possível controlar com facilidade os diversos graus entre 0% a 100% com passos de aproximadamente 5%.

6 Conclusões e Trabalhos Futuros

6.1 Conclusões

Avaliando o conjunto de objetivos estabelecidos na premissa deste projeto, destaca-se que estes foram alcançados, uma vez que com a junção de parâmetros eletromecânicos, os fundamentos de manufatura aditiva e os recursos de inteligência computacional empregados fornecem ao dispositivo a cognição necessária para atuar como reconhecedor de padrões. O funcionamento e o controle dos dedos da mão apresentaram-se de forma fluida e funcional conforme as necessidades de projeto.

Vale observar que ao longo de toda a evolução no que tange ao desenvolvimento computacional do projeto, os aspectos funcionais da mão se comunicaram extremamente bem com seu microcontrolador agregado, favorece uma gesticulação de fácil compreensão e tempo de resposta curto, conforme evidenciado pelo conjunto de vídeos fornecidos ao time de professores avaliadores deste projeto.

Ademais, destaca-se também a variedade de oportunidades que podem ser seguidas pela continuidade deste projeto, uma vez que foi dado o start com uma parte da estrutura principal, conforme dito na subseção posterior, forma-se uma série de caminhos que podem ser tomados com base no modelo aqui desenvolvido, uma vez que o projeto Open Source InMoov continua em execução e os fundamentos de aprendizado de máquina continuam sendo aprimorados ao redor do mundo.

Por fim, avaliando o conjunto de técnicas aqui empregados, o projeto em questão apresenta-se como um bom catalizador de alguns dos principais fundamentos vistos ao longo dos cursos de Instrumentação, Automação e Robótica, sobretudo aos eixos de robótica e controle.

6.2 Trabalhos Futuros

Avaliando as oportunidades não apenas de aprimoramento técnico, mas também de funcionamento do protótipo, sugere-se uma série de oportunidades para continuidade no desenvolvimento de projetos que se assemelham em fundamentação teórica, mas que por sua vez podem apresentar resultados distintos aos aqui indicados, sobretudo:

- Utilização da infraestrutura completa do projeto InMoov para desenvolvimento de um robô humanoide bípede
- Utilização de recursos do torço e cabeça do projeto InMoov para efetuar o reco-

nhecimento de padrões por meio de câmeras diretamente agregadas a estrutura do projeto

- Incremento no sistema de representação de sinais efetuado pelo dispositivo, garantindo uma tradução efetiva, assim como a ampliação de suas funções cognitivas no que tange ao reconhecimento de padrões
- Ampliação da biblioteca, assim como os modelos de representação de sinais, com adição de novos modelos de mãos realizando gestos similares aos aqui efetuados, dando assim maiores capacidades cognitivas ao dispositivo, lhe dando assim maiores características de um "robô"

Outras perspectivas podem ser agregadas ao projeto, tendo em vista a gama de oportunidades no setor de manufatura aditiva, podendo garantir não apenas diferentes designs ao projeto, mas outras funcionalidades estruturais ao projeto, com a adição de membros para o suporte das ações motoras, ou mesmo para facilitar seu deslocamento.

Referências

- BUSTAMANTE, R. M. *Fatiador AMF COM SUBDIVISÃO DE TRIÂNGULOS CURVOS E AJUSTE DE CURVAS*. Dissertação (Mestrado) — UDESC - Universidade do Estado de Santa Catarina, feb 2019. Citado na página 19.
- CRAIG, J. *Introduction to robotics: mechanics and control*. third. [S.l.]: Prentice Hall, 2004. Citado 2 vezes nas páginas 1 e 8.
- CRUZ, G. G. F. *Classificação dos movimentos da mão baseados na aquisição não Invasiva de sinais mioelétricos provenientes dos músculos do antebraço através de redes neurais artificiais*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina (UFSC), 2017. Citado na página 4.
- CRUZEIRO, A. C. *Explorando a impressão 3D - Conceito, etapas e futuro*. 2019. Disponível em: <<https://via.ufsc.br/explorando-a-impressao-3d-conceito-etapas-e-futuro/>>. Citado na página 19.
- CUNICO, M. *Impressoras 3D: O novo Meio Produtivo*. [S.l.]: Concep3d Pesquisas Científicas, 2015. Citado na página 1.
- DOCS, A. *Firmata Library*. 2022. Disponível em: <<https://docs.arduino.cc/hacking/software/FirmataLibrary>>. Citado na página 1.
- HARTENBERG, J. D. R. A kinematic notation for lower-pair mechanisms based on matrices. *Trans ASME J. Appl*, v. 23, p. 215–221, 1955. Citado 2 vezes nas páginas 1 e 5.
- KHALIL W.; KLEINFINGER, J. *A new geometric rotation for open closed-loop robots*. first. [S.l.]: IEEE International Conference on Robotics and Automation, 1986. v1. (3, v1). Citado na página 7.
- KINIVI. *hand-gesture-recognition-using-mediasuite*. Disponível em: <<https://github.com/kinivi/hand-gesture-recognition-mediasuite>>. Citado na página 47.
- LANGEVIN, G. *InMoov - Open Source 3D Printed Life-Size Robot*. [s.n.], 2012. Disponível em: <<http://inmoov.fr/>>. Citado 5 vezes nas páginas 1, 8, 14, 15 e 35.
- MEDIAPIPE. *Media Pipe - Solutions: Hands*. Disponível em: <<https://google.github.io/mediapipe/solutions/hands.html>>. Citado na página 1.
- PORTELA, S. *Afinal o que é um arquivo STL*. 2019. Disponível em: <<https://3dlab.com.br/tag/formato-stl/>>. Citado 2 vezes nas páginas 1 e 20.
- REPRAP. *G-Code*. [s.n.], 2010. Disponível em: <<https://reprap.org/wiki/G-code>>. Citado na página 20.
- ROSÁRIO, J. *Princípios de Mecatrônica*. first. [S.l.]: Pearson Prentice Hall, 2005. Citado na página 5.
- SCIKIT-LEARN. *Neural Network Models (Supervised)*. 2022. Disponível em: <https://scikit-learn.org/stable/modules/neural_networks_supervised.html>. Citado na página 1.

SENYAEY, A. S. A. *Open CV - Python 4.6.0.66*. 2022. Disponível em: <<https://pypi.org/project/opencv-python/>>. Citado na página 1.

SOBOTTA, J. *Atlas da Anatomia Humana*. twenty-first. Rio de Janeiro: Gunabara Koogan, 2000. Citado 2 vezes nas páginas 1 e 3.

STOPPA, J. P. M. *Tecnologias em Pesquisa: Engenharias*. first. [S.l.]: Blucher Open Access, 2017. v. 1. Citado 6 vezes nas páginas 1, 4, 7, 8, 10 e 12.

VALLE, C. M. C. O. *Controle por Torque Computado de um Robô Bípede Simulado com Locomoção via Aprendizado por Reforço*. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro, apr 2016. Citado 2 vezes nas páginas 1 e 5.

VOLPATO, N. *Manufatura Aditiva: tecnologias e aplicações da impressão 3D*. [S.l.]: Blucher, 2017. Citado na página 17.

APÊNDICE A – Script para Coleta de Dados de Treino

```

import mediapipe as mp
import cv2
import numpy as np
from itertools import chain
import pandas as pd
import csv
import os
import joblib
from time import time

def get_coordinates(joint, hand_index, results, video_width,
                     video_height):
    """
    Params:
        joint: the joint name (WRIST, THUMB_CMC, INDEX_FINGER_MCP, etc)
        hand_index = the positional index of the hand identified in the
                    results.multi_hand_landmarks list. If two hands were detected
                    for example, the hand in the second position of the array will
                    have index 1, and the first index 0
        results = the output of mp.solutions.hands(...).process(
                    image)
        video_width = the width of the video output. Usually gotten from
                      cap.get(cv2.CAP_PROP_FRAME_WIDTH)
        video_height = the height of the video output. Usually gotten
                      from
                      cap.get(cv2.CAP_PROP_FRAME_HEIGHT)

    Outputs:
        (x,y)
        - x -> x axis coordinate in pixel
        - y -> y axis coordinate in pixel
    """
    normalized_coordinates = results.multi_hand_landmarks[hand_index].landmark[mp_hands.HandLandmark[joint]]
    coordinates = tuple(
        np.multiply(
            [normalized_coordinates.x,
             normalized_coordinates.y],
            [video_width, video_height])
    )
    return coordinates

```

```

        [video_width, video_height]
    ).astype(int)
)
return coordinates

def get_handedness(hand_index, results):
    """
    Params:
        hand_index = the positional index of the hand identified in the
        results.multi_hand_landmarks list. If two hands were detected
        for example, the hand in the second position of the array will
        have index 1, and the first index 0
        results = the output of mp.solutions.hands.Hands(...).process(
            image)
        video_width = the width of the video output. Usually gotten from
        cap.get(cv2.CAP_PROP_FRAME_WIDTH)
        video_height = the height of the video output. Usually gotten
            from
        cap.get(cv2.CAP_PROP_FRAME_HEIGHT)

    Outputs:
        - handedness = if the hand is 'left' or 'right'

    Observation: the results.multi_hand_landmarks is an array in which
                    each
                    element represents one hand, and each hand will have an array with
                        21
                    coordinates of the hand landmarks. The results.multi_handedness is
                        similar,
                    is an array in which each element represents one hand and each hand
                        will have
                    a label with the handedness and the score of the classification.
                    The relationship between both is based on the position of the hand
                        in the array,
                    for example the results.multi_hand_landmarks[0] will have the
                        landmarks of the
                    same hand in the results.multi_handedness[0]
    """
output = None

    # Getting handedness label
label = results.multi_handedness[hand_index].classification[0].label
#score = round(results.multi_handedness[hand_index].classification[0
                ].score,2)
output = f'{label}'

return output

```

```

def draw_normalized_coordinates(image, coordinates,
                                 normalized_coordinates):
    """
    Params:
        image = the image to draw the normalized coordinates
        coordinates = list of tuples containing the original coordinates
                      in
        which the normalized coordinates will be drawn
        normalized_coordinates = list of tuples with the coordinates
                                  normalized

    Outputs:
        - handedness = if the hand is 'left' or 'right'
    """
    for coordinate, normalized_coordinate in zip(coordinates,
                                                   normalized_coordinates):
        x, y = coordinate
        norm_x, norm_y = normalized_coordinate
        image = cv2.putText(
            img = image,
            text = f'x:{norm_x} y:{norm_y}',
            org = coordinate,
            fontFace = cv2.FONT_HERSHEY_SIMPLEX,
            fontScale = 0.4,
            color = (255, 255, 255),
            thickness = 1,
            lineType = cv2.LINE_AA
        )
    return image

def pre_process_hand_landmarks(hand_index, results, video_width,
                               video_height):
    """
    Params:
        hand_index = the positional index of the hand identified in the
                    results.multi_hand_landmarks list. If two hands were detected
                    for example, the hand in the second position of the array will
                    have index 1, and the first index 0
        results = the output of mp.solutions.hands.Hands(...).process(
                    image)
        video_width = the width of the video output. Usually gotten from
                      cap.get(cv2.CAP_PROP_FRAME_WIDTH)
        video_height = the height of the video output. Usually gotten
                       from
                      cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
    Output:

```

```

processed_hand_landmarks: flatten, normalized and traslated
coordinates
that will be used as input for the MLP model. Is an array with
42 points,
ordered by the joint index, in the following way -> [wrist.x,
wrist.y,
thumb_cmc.x, thumb_cmc.y, ..., pinky_dip.x, pinky_dip.y,
pinky_dip.x,
pinky_dip.y].
"""

# Extract the x and y coordinates from the results
hand_landmarks_coordinates = []
for coordinates in results.multi_hand_landmarks[hand_index].landmark
    :
    x_coord = coordinates.x * video_width
    y_coord = coordinates.y * video_height
    hand_landmarks_coordinates.append((x_coord, y_coord))

# Translate the coordinates making the wrist coordinates the origin
hand_landmarks_translated_coordinates = []
for idx, coordinate_pair in enumerate(hand_landmarks_coordinates):
    if idx==0:
        base_coordinate_pair = coordinate_pair
        hand_landmarks_translated_coordinates.append((0,0))
    else:
        translated_coordinate_pair = np.subtract(coordinate_pair,
                                                base_coordinate_pair)
        hand_landmarks_translated_coordinates.append(tuple(
            translated_coordinate_pair
        ))

# Flat array and normalize all coordinates using the min-max
normalization
flattened_hand_landmarks_translated_coordinates = list(chain.
from_iterable(
    hand_landmarks_translated_coordinates
))
array = np.array(flattened_hand_landmarks_translated_coordinates)
processed_hand_landmarks = (array-array.min())/(array.max()-array.
min())

return processed_hand_landmarks

# Object that let us draw landmarks in our image
mp_drawing = mp.solutions.drawing_utils

# Object with all hand tracking methods of mediapipe

```

```

mp_hands = mp.solutions.hands

# Object that reads from the webcam
cap = cv2.VideoCapture(0)

# Get the width and height so we can draw on the image using opencv
video_width = cap.get(cv2.CAP_PROP_FRAME_WIDTH)
video_height = cap.get(cv2.CAP_PROP_FRAME_HEIGHT)

# Hand gesture label map
gesture_label = pd.read_csv('gesture-label.csv', encoding = 'latin1')

with mp_hands.Hands(
    static_image_mode=False,
    max_num_hands=2,
    model_complexity=1,
    min_detection_confidence=0.8,
    min_tracking_confidence=0.5
) as hands:
    while cap.isOpened():
        # cap.read() return two variables, the 'results' which is a
        # boolean
        # identifying if the image was read and the frame that is a cv2
        # image
        # object of the frame captured
        ret, frame = cap.read()

        # Before processing our image with mediapipe is necessary to
        # convert it
        # from BGR to RGB, because mediapipe works with RGB and opencv
        # with BGR
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Flip on horizontal so the lib detects correct handness
        image = cv2.flip(image, 1)

        # Setting the writable flag to false before process with
        # mediapipe leads
        # to improvement in the performance
        image.flags.writeable = False

        # Do the actual processing with the mediapipe lib
        results = hands.process(image)

        # Setting back the flag of writable so we can draw in the image
        image.flags.writeable = True

```

```
# Writing that is waiting for the key to save the coordinates
image = cv2.putText(
    img = image,
    text = 'Press a number from 0 to 9 to save gesture data',
    org = (50,50), #coordinates
    fontFace = cv2.FONT_HERSHEY_SIMPLEX,
    fontScale = 0.5,
    color = (0, 0, 0), #RGB
    thickness = 2,
    lineType = cv2.LINE_AA
)

# Drawing landmarks to the image
## If any hand was detected
if results.multi_hand_landmarks:
    # The enumerate is used to multi hand detection, the hand
    # variable
    # is basically all the landmarks from one hand
    for hand_index, hand_landmarks in enumerate(results.
                                                multi_hand_landmarks):

        # Get handedness label
        handedness_label = get_handedness(
            hand_index = hand_index,
            results = results
        )

        # Drawing handedness on the wrist
        image = cv2.putText(
            img = image,
            text = handedness_label,
            org = get_coordinates('WRIST', hand_index,
                                  results,
                                  video_width
                                  ,
                                  video_height
                                  ), #
                                  coordinates

            fontFace = cv2.FONT_HERSHEY_SIMPLEX,
            fontScale = 0.8,
            color = (255, 255, 255), #RGB
            thickness = 2,
            lineType = cv2.LINE_AA
        )
```

```

# Pre process the hand landmarks coordinates
processed_hand_landmarks = pre_process_hand_landmarks(
    hand_index, results,
    video_width,
    video_height)

# Draw the pre-processed coordinates according to the
# joint list
joint_list = ['INDEX_FINGER_TIP', 'THUMB_TIP', ,
              'MIDDLE_FINGER_TIP', ,
              'RING_FINGER_TIP', ,
              'PINKY_TIP']

coordinates = []
normalized_coordinates = []
for joint in joint_list:
    coordinates.append(
        get_coordinates(
            joint,
            hand_index,
            results,
            video_width,
            video_height
        )
    )
    joint_index = mp_hands.HandLandmark[joint].numerator
    normalized_coordinates.append(
        tuple(processed_hand_landmarks[(joint_index*2):(
            (
                joint_index*2)+2)])
    )

normalized_coordinates = np.around(
    normalized_coordinates,
    2)

image = draw_normalized_coordinates(
    image = image,
    coordinates = coordinates,
    normalized_coordinates = normalized_coordinates
)

# Utility used to draw the image based on the landmark
# values
mp_drawing.draw_landmarks(
    image = image,
    landmark_list = hand_landmarks,

```

```

        connections = mp_hands.HAND_CONNECTIONS,
        landmark_drawing_spec = mp_drawing.DrawingSpec(
            color = (1, 190, 255),
            thickness = 2,
            circle_radius = 4
        ),
        connection_drawing_spec = mp_drawing.DrawingSpec(
            color = (86, 213, 0),
            thickness = 2,
            circle_radius = 2
        )
    )

# Converting it back to BGR so we can display using opencv
image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

# Display the output frame of the webcam
cv2.imshow('Hand Tracking', image)

# - If 'q' is pressed the window is closed;
# - If 'e' is pressed all training data will be erased;
# - If a number from 0 to 9 is pressed then the coordinates
#   will be saved in the training sv file with those
#   coordinates
#   associated to the number label pressed.
key = cv2.waitKey(10)
if key==ord('q'):
    break
if key==ord('e'):
    os.remove('training-data.csv')
if key>=ord('0') and key<=ord('9'):
    # Create the headers of the csv with the joint landmark
    # names
    headers = ['gesture_id']
    for joint in mp_hands.HandLandmark:
        headers.append(joint.name.lower()+'_x')
        headers.append(joint.name.lower()+'_y')

    with open('training-data.csv','a',newline='') as file:
        writer = csv.writer(file)
        row = np.append(int(chr(key)),processed_hand_landmarks)
        # If file was created from scratch, append header. If
        # not,
        # only append row
        if file.tell()==0:
            writer.writerow(headers)
        writer.writerow(row)

```

```
    else:
        writer.writerow(row)

    # Erase the coordinates after saving it, to avoid write
    # duplicate data
    # when no hand be captured by the video
    processed_hand_landmarks = None

    try:
        label = gesture_label.loc[gesture_label[gesture_label.
                                         Label==int(chr(key))].
                                         ].index[0], 'Gesture'
        print(f'Gesture saved to label {label}')
    except:
        print(f'No label detected. Update the "gesture-label.csv"
              " file with the
              label of "{chr(key)}"
              ".')
    cap.release()
    # Destroy all the windows
    cv2.destroyAllWindows()
```


APÊNDICE B – Script Para Treinamento do Modelo

```

#!/usr/bin/env python
# coding: utf-8

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import plot_confusion_matrix
import joblib

df = pd.read_csv('training-data.csv')

features_list = df.columns.to_list()
features_list.pop(0)
y = df['gesture_id']
X = df[features_list]

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=2,
                                                    test_size=0.2)

clf = MLPClassifier(
    hidden_layer_sizes = (20,15,13,10,8),
    activation ="relu",
    solver = 'sgd',
    learning_rate = 'constant',
    max_iter = 10000,
    tol = -1,
    verbose = True
)

clf.fit(X_train, y_train)

fig=plot_confusion_matrix(clf, X_test, y_test)
fig.figure_.suptitle("Confusion Matrix")

joblib.dump(clf, 'clf.pkl', compress=9)

```


APÊNDICE C – Script Para Execução do Projeto

```

#!/usr/bin/env python
# coding: utf-8

import mediapipe as mp
import cv2
import numpy as np
from itertools import chain
import pandas as pd
import os
import joblib
import pyfirmata
from pyfirmata import Arduino, SERVO, util
import time

def get_handedness(hand_index, results):
    """
    Params:
        hand_index = the positional index of the hand identified in the
                    results.multi_hand_landmarks list. If two hands were detected
                    for example, the hand in the second position of the array will
                    have index 1, and the first index 0
        results = the output of mp.solutions.hands(...).process(
                    image)
        video_width = the width of the video output. Usually gotten from
                    cap.get(cv2.CAP_PROP_FRAME_WIDTH)
        video_height = the height of the video output. Usually gotten
                    from
                    cap.get(cv2.CAP_PROP_FRAME_HEIGHT)

    Outputs:
        - handedness = if the hand is 'left' or 'right'

    Observation: the results.multi_hand_landmarks is an array in which
                  each
                  element represents one hand, and each hand will have an array with
                  21
                  coordinates of the hand landmarks. The results.multi_handedness is
                  similar,
                  is an array in which each element represents one hand and each hand
                  will have
  
```

```

    a label with the handedness and the score of the classification.
    The relationship between both is based on the position of the hand
        in the array,
    for example the results.multi_hand_landmarks[0] will have the
                    landmarks of the
    same hand in the results.multi_handedness[0]
    """
    output = None

    # Getting handedness label
    label = results.multi_handedness[hand_index].classification[0].label
    #score = round(results.multi_handedness[hand_index].classification[0
    #                ].score,2)
    output = f'{label}'

    return output

def pre_process_hand_landmarks(hand_index, results, video_width,
                               video_height):
    """
    Params:
        hand_index = the positional index of the hand identified in the
            results.multi_hand_landmarks list. If two hands were detected
            for example, the hand in the second position of the array will
            have index 1, and the first index 0
        results = the output of mp.solutions.hands(...).process(
            image)
        video_width = the width of the video output. Usually gotten from
            cap.get(cv2.CAP_PROP_FRAME_WIDTH)
        video_height = the height of the video output. Usually gotten
            from
            cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
    Output:
        processed_hand_landmarks: flatten, normalized and traslated
            coordinates
        that will be used as input for the MLP model. Is an array with
            42 points,
        ordered by the joint index, in the following way -> [wrist.x,
            wrist.y,
            thumb_cmc.x, thumb_cmc.y, ..., pinky_dip.x, pinky_dip.y,
            pinky_dip.x,
            pinky_dip.y].
    """
    # Extract the x and y coordinates from the results
    hand_landmarks_coordinates = []
    for coordinates in results.multi_hand_landmarks[hand_index].landmark
        :

```

```

        x_coord = coordinates.x * video_width
        y_coord = coordinates.y * video_height
        hand_landmarks_coordinates.append((x_coord, y_coord))

    # Translate the coordinates making the wrist coordinates the origin
    hand_landmarks_translated_coordinates = []
    for idx, coordinate_pair in enumerate(hand_landmarks_coordinates):
        if idx==0:
            base_coordinate_pair = coordinate_pair
            hand_landmarks_translated_coordinates.append((0,0))
        else:
            translated_coordinate_pair = np.subtract(coordinate_pair,
                                                       base_coordinate_pair)
            hand_landmarks_translated_coordinates.append(tuple(
                translated_coordinate_pair
            ))

    # Flat array and normalize all coordinates using the min-max
    # normalization
    flattened_hand_landmarks_translated_coordinates = list(chain.
        from_iterable(
            hand_landmarks_translated_coordinates
        ))
    array = np.array(flattened_hand_landmarks_translated_coordinates)
    processed_hand_landmarks = (array-array.min())/(array.max()-array.
        min())

    return processed_hand_landmarks

# Object that let us draw landmarks in our image
mp_drawing = mp.solutions.drawing_utils

# Object with all hand tracking methods of mediapipe
mp_hands = mp.solutions.hands

# Loading gesture recognition model
gesture_classifier = joblib.load('clf.pkl')

# Variable used to control the time in which the servo motor
# will be triggered
ref = None

# Initializing variable that will contain the gesture recognized
gesture = None
# Initializing variable that will contain the module in which the
# servo will be moved
module = None

```

```
# Dictionary containing the pins for each articulation
articulation_dict = {
    'Mindinho' : {
        'pin': 2,
        'fator': 1.3
    },
    'Anelar' : {
        'pin': 3,
        'fator': 1
    },
    'Medio' : {
        'pin': 4,
        'fator': 1.4
    },
    'Indicador' : {
        'pin': 5,
        'fator': 1.1
    },
    'Polegar' : {
        'pin': 6,
        'fator': 1
    },
    'Pulso' : {
        'pin': 7,
        'fator': 1
    },
}
}

port = 'COM3'
board = pyfirmata.Arduino(port)

# Setting Arduino's pins
for info in articulation_dict.values():
    board.digital[info['pin']].mode = SERVO

# Object that reads from the webcam
cap = cv2.VideoCapture(0)

# Get the width and height so we can draw on the image using opencv
video_width = cap.get(cv2.CAP_PROP_FRAME_WIDTH)
video_height = cap.get(cv2.CAP_PROP_FRAME_HEIGHT)

# Hand gesture label map
gesture_label = pd.read_csv('gesture-label.csv', encoding = 'latin1')

with mp_hands.Hands(
```

```

    static_image_mode=False,
    max_num_hands=2,
    model_complexity=1,
    min_detection_confidence=0.8,
    min_tracking_confidence=0.5
) as hands:
    while cap.isOpened():
        # cap.read() return two variables, the 'results' which is a
        # boolean
        # identifying if the image was read and the frame that is a cv2
        # image
        # object of the frame captured
        ret, frame = cap.read()

        # Before processing our image with mediapipe is necessary to
        # convert it
        # from BGR to RGB, because mediapipe works with RGB and opencv
        # with BGR
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Flip on horizontal so the lib detects correct handness
        image = cv2.flip(image, 1)

        # Setting the writable flag to false before process with
        # mediapipe leads
        # to improvement in the performance
        image.flags.writeable = False

        # Do the actual processing with the mediapipe lib
        results = hands.process(image)

        # Setting back the flag of writable so we can draw in the image
        image.flags.writeable = True

        handedness_detected = []

        # Drawing landmarks to the image
        ## If any hand was detected
        if results.multi_hand_landmarks:
            # The enumerate is used to multi hand detection, the hand
            # variable
            # is basically all the landmarks from one hand
            for hand_index, hand_landmarks in enumerate(results.
                multi_hand_landmarks):

                # Get handedness label
                handedness_label = get_handedness(

```

```

        hand_index = hand_index,
        results = results
    )

handedness_detected.append(handedness_label)

# Draw bounding box only to the left hand
if handedness_label=='Left':
    x_max = 0
    y_max = 0
    x_min = video_width
    y_min = video_height
    for coordinates in hand_landmarks.landmark:
        x, y = int(coordinates.x * video_width), int(
            coordinates.
            y *
            video_height
        )
        if x > x_max:
            x_max = x
        if x < x_min:
            x_min = x
        if y > y_max:
            y_max = y
        if y < y_min:
            y_min = y
    cv2.rectangle(image, (x_min-10, y_min-10), (x_max+10
                                                , y_max+10), (50
                                                , 50, 50), 2)
    cv2.rectangle(image, (x_min-10, y_min-10), (x_max+10
                                                , y_min-50), (50
                                                , 50, 50), -1)

# Pre process the hand landmarks coordinates
processed_hand_landmarks =
    pre_process_hand_landmarks
    (hand_index,
     results,
     video_width,
     video_height)

# Predicting the gesture label
label_predicted = int(gesture_classifier.predict(
    processed_hand_landmarks
    .reshape(1,-1))[
    0])

```

```

text_label_predicted = gesture_label[gesture_label.
                                    Label ==
                                    label_predicted]
                                    .Gesture.to_list
() [0]

# Updating the gesture with the text_label_predicted
gesture = text_label_predicted

# Writing the predicted label to the frame
image = cv2.putText(
    img = image,
    text = text_label_predicted,
    org = (x_min+5,y_min-20), #coordinates
    fontFace = cv2.FONT_HERSHEY_SIMPLEX,
    fontScale = 0.7,
    color = (255, 255, 255), #RGB
    thickness = 2,
    lineType = cv2.LINE_AA
)

# Utility used to draw the image based on the
# landmark values
mp_drawing.draw_landmarks(
    image = image,
    landmark_list = hand_landmarks,
    connections = mp_hands.HAND_CONNECTIONS,
    landmark_drawing_spec = mp_drawing.DrawingSpec(
        color = (1, 190, 255

            thickness = 2,
            circle_radius = 4
        ),
    connection_drawing_spec = mp_drawing.DrawingSpec(
        (
            color = (86, 213, 0)

            thickness = 2,
            circle_radius = 2
        )
    )
elif handedness_label=='Right':
    # Drawing a circle in the joints of the THUMB_TIP
    # and

```

```

    INDEX_FINGER_TIP
    and

# storing its coordinates
joints = ['THUMB_TIP', 'INDEX_FINGER_TIP']
joints_coordinates = {}
for joint in joints:
    coordinates = {}
    coordinates['x'] = int(results.
                           multi_hand_landmarks
                           [hand_index]
                           .landmark[
                           mp_hands.
                           HandLandmark
                           [joint]].x*
                           video_width)

    coordinates['y'] = int(results.
                           multi_hand_landmarks
                           [hand_index]
                           .landmark[
                           mp_hands.
                           HandLandmark
                           [joint]].y*
                           video_height
                           )

    joints_coordinates[joint] = coordinates
# Drawing circle
image = cv2.circle(
    img = image,
    center = (coordinates['x'], coordinates['y'])

    ,
    radius = 4,
    color = (148, 0, 211),
    thickness = 2
)
# Drawing white border for the circle
image = cv2.circle(
    img = image,
    center = (coordinates['x'], coordinates['y'])

    ,
    radius = 6,
    color = (255, 255, 255),
    thickness = 1
)

# Draw line between the thumb_tip and
# index_finger_tip
image = cv2.line(

```

```

        img = image,
        pt1 = (joints_coordinates['THUMB_TIP']['x'],
                joints_coordinates
                ['THUMB_TIP'
                 ]['y']),
        pt2 = (joints_coordinates['INDEX_FINGER_TIP']['x'
                ],
                joints_coordinates
                [
                INDEX_FINGER_TIP
                ]['y']),
        color = (255,4,163),
        thickness = 2
    )

    # Drawing circle in the center of the line
    min_x = min(joints_coordinates['THUMB_TIP']['x'],
                 joints_coordinates
                 [
                 INDEX_FINGER_TIP
                 ]['x'])
    min_y = min(joints_coordinates['THUMB_TIP']['y'],
                 joints_coordinates
                 [
                 INDEX_FINGER_TIP
                 ]['y'])
    diff_x = int(abs(joints_coordinates['THUMB_TIP']['x'
            ] -
                     joints_coordinates
                     [
                     INDEX_FINGER_TIP
                     ]['x'])/2)
    diff_y = int(abs(joints_coordinates['THUMB_TIP']['y'
            ] -
                     joints_coordinates
                     [
                     INDEX_FINGER_TIP
                     ]['y'])/2)
    image = cv2.circle(
        img = image,
        center = (min_x + diff_x, min_y + diff_y),
        radius = 6,
        color = (148, 0, 211),
        thickness = 3
    )

```

```
# Drawing white border for the circle in the center
# of the line

image = cv2.circle(
    img = image,
    center = (min_x + diff_x, min_y + diff_y),
    radius = 8,
    color = (255, 255, 255),
    thickness = 1
)

# Draw the rectangle border that will contain the
# relative
# distance of the
# line

# between the thumb_tip and index_finger_tip

beginning_x_coordinate_rect = 85
end_x_coordinate_rect = 485

image = cv2.rectangle(
    img = image,
    pt1 = (beginning_x_coordinate_rect, 30),
    pt2 = (end_x_coordinate_rect, 70),
    color = (255, 4, 163),
    thickness = 2
)

# The relative distance will be the line between the
# thumb tip and
# index_finger_tip
# divided by the line between the wrist and
# index_finger_dip

wrist_coordinates = (results.multi_hand_landmarks[
    hand_index].landmark[
        mp_hands.HandLandmark['WRIST']].x *
        video_width,
    results.multi_hand_landmarks[
        hand_index].landmark[
        mp_hands.HandLandmark['WRIST']].y *
        video_height)
```

```

distance_wrist_index = np.sqrt(((wrists_coordinates[0]
]-
joints_coordinates[',
INDEX_FINGER_TIP
'] [,x'])**2) +
((wrists_coordinates[1]-
joints_coordinates[',
INDEX_FINGER_TIP
'] [,y'])**2))

distance_thumb_index = np.sqrt(((joints_coordinates['THUMB_TIP'] [,x']
]-joints_coordinates[',
INDEX_FINGER_TIP
'] [,x'])**2) +
((joints_coordinates['THUMB_TIP'] [,y']
]-joints_coordinates[',
INDEX_FINGER_TIP
'] [,y'])**2))

relative_distance_thumb_index = distance_thumb_index /
distance_wrist_index

#Offset
relative_distance_thumb_index =
relative_distance_thumb_index -
0.08

if relative_distance_thumb_index > 1:
    relative_distance_thumb_index = 1
elif relative_distance_thumb_index < 0:
    relative_distance_thumb_index = 0

linear_interpolation = int(
beginning_x_coordinate_rect +
((end_x_coordinate_rect -

```

```

beginning_x_coordinate_rect
)*((
relative_distance_thumb_index
)))))

# Draw the rectangle that will fill the base
rectangle
according to the
distance

# between the thumb_tip index_finger_tip
image = cv2.rectangle(
    img = image,
    pt1 = (beginning_x_coordinate_rect, 30),
    pt2 = (linear_interpolation, 70),
    color = (255,4,163),
    thickness = -1
)

# Writing relative distance
image = cv2.putText(
    img = image,
    text = str(int(relative_distance_thumb_index*100
                    ))+' %',
    org = (end_x_coordinate_rect+15,55), #
coordinates
    fontFace = cv2.FONT_HERSHEY_SIMPLEX,
    fontScale = 0.8,
    color = (255,4,163), #RGB
    thickness = 2,
    lineType = cv2.LINE_AA
)

# The servo will contract the closest to 1 and
expand the
closest to
# 0, so we will get the complementary of 1 to the
module
module = 1 - relative_distance_thumb_index

# Converting it back to BGR so we can display using opencv
image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

# Display the output frame of the webcam
cv2.imshow('Hand Tracking', image)

# Condition to not write old angles to the servo in case we don'
t

```

```
# have hands in the screen
if 'Right' not in handedness_detected:
    module = None
if 'Left' not in handedness_detected:
    gesture = None

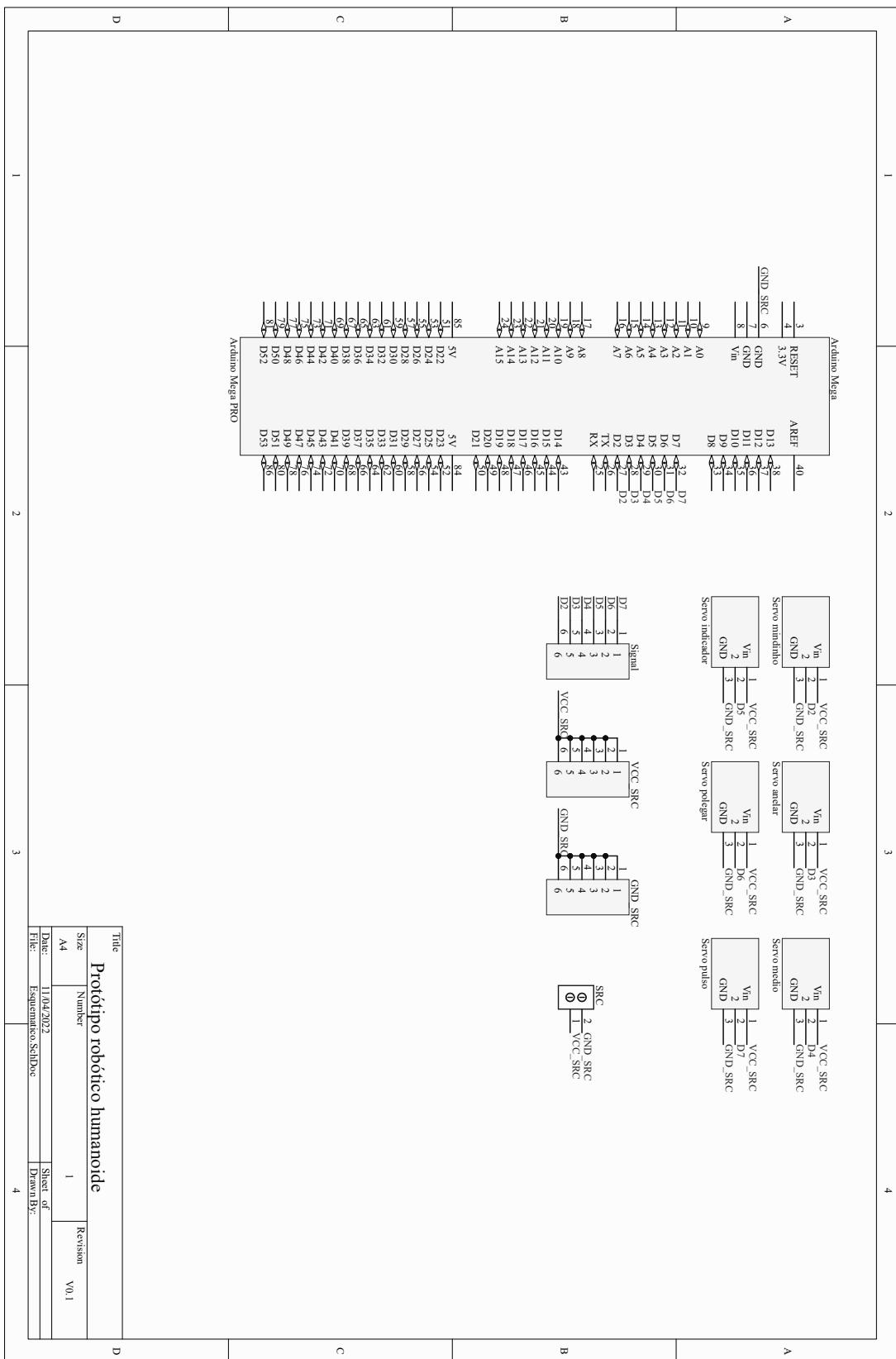
# Triggering the servo motor
if (gesture != None and module != None):
    board.digital[articulation_dict[gesture]['pin']].write(
        module*90*
        articulation_dict[
            gesture]['fator'])

# - If 'q' is pressed the window is closed;
key = cv2.waitKey(10)
if key==ord('q'):
    break

cap.release()
# Destroy all the windows
cv2.destroyAllWindows()
```

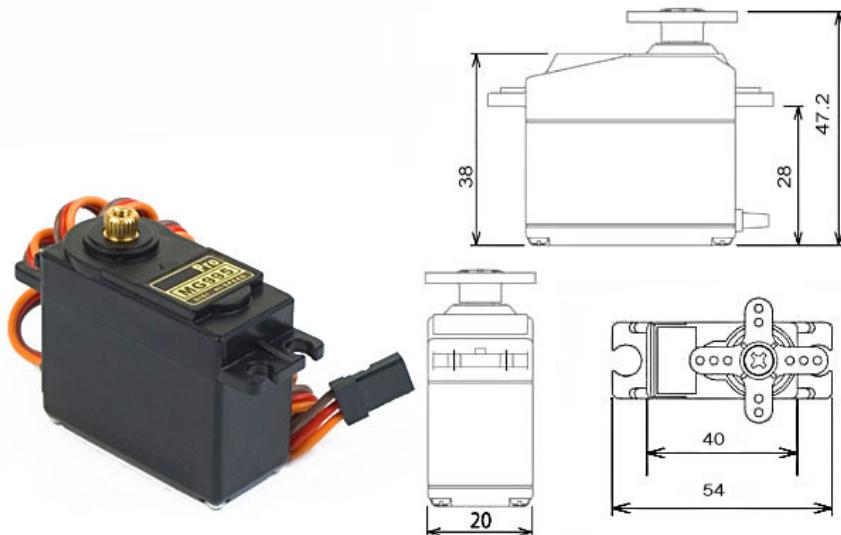

Anexos

ANEXO A – Esquema Elétrico



ANEXO B – Datasheet MG995 - Tower Pro

MG995 High Speed Metal Gear Dual Ball Bearing Servo



The unit comes complete with 30cm wire and 3 pin 'S' type female header connector that fits most receivers, including Futaba, JR, GWS, Cirrus, Blue Bird, Blue Arrow, Corona, Berg, Spektrum and Hitec.

This high-speed standard servo can rotate approximately 120 degrees (60 in each direction). You can use any servo code, hardware or library to control these servos, so it's great for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. The MG995 Metal Gear Servo also comes with a selection of arms and hardware to get you set up nice and fast!

Specifications

- Weight: 55 g
- Dimension: 40.7 x 19.7 x 42.9 mm approx.
- Stall torque: 8.5 kgf·cm (4.8 V), 10 kgf·cm (6 V)
- Operating speed: 0.2 s/60° (4.8 V), 0.16 s/60° (6 V)
- Operating voltage: 4.8 V a 7.2 V
- Dead band width: 5 µs
- Stable and shock proof double ball bearing design
- Temperature range: 0 °C – 55 °C