ESCOLA SUPERIOR **NÁUTICA** INFANTE D. HENRIQUE

Projeto Integrado

Professor Camilo Christo

Braço Robótico com 2 graus de liberdade e mão robótica com 5 graus de liberdade

Feito por

Henrique Abrantes (15196@enautica.pt)

Christian Rodrigues (15202@enautica.pt)

Rodrigo Maria (15217@enautica.pt)

TeSP Robótica e Inteligência Artificial

Índice

Índice de Figuras

Índice de Tabelas

1 Introdução

O presente projeto surge no âmbito da Unidade Curricular de Projeto Integrado, na qual foram propostos diversos temas para desenvolvimento. Após a análise das propostas iniciais, optou-se pela implementação de um projeto alternativo [1], focado no desenvolvimento de um braço robótico humanoide controlado por sistemas de visão computacional.

O objetivo central do trabalho consiste na conceção e assemblagem de um braço robótico com 2 graus de liberdade (DoF), permitindo a rotação da base (*Yaw*) e a elevação do cotovelo (*Pitch*). A este braço foi acoplada uma mão robótica antropomórfica com 5 dedos independentes, conferindo mais 5 DoF ao sistema (totalizando 7 DoF no conjunto).

A estratégia de controlo baseia-se no processamento de imagem em tempo real, onde o sistema interpreta os movimentos humanos e os replica nos servomotores. Relativamente à construção mecânica, toda a estrutura estrutural foi produzida integralmente através de processos de manufatura aditiva (impressão 3D), recorrendo aos equipamentos e recursos disponibilizados pela instituição de ensino.

2 Descrição do Projeto

O desenvolvimento deste sistema robótico antropomórfico resulta da integração multidisciplinar entre engenharia mecânica, eletrónica de potência e desenvolvimento de *software*. O presente capítulo detalha as etapas construtivas e as decisões de projeto que permitiram transformar o modelo teórico num protótipo funcional, capaz de mimetizar os movimentos do membro superior humano.

A descrição técnica encontra-se estruturada de forma sequencial, abordando as três camadas fundamentais do sistema:

1. **Estrutura Mecânica:** Descrição dos processos de manufatura aditiva (Impressão 3D), escolha de materiais e assemblagem das articulações baseadas no projeto *open-source* InMoov.
2. **Hardware e Eletrónica:** Análise da arquitetura de circuitos, distribuição de energia e seleção de atuadores (servomotores e motores de passo) e controladores (Arduinos).
3. **Lógica de Controlo:** Explicação dos algoritmos de *firmware* responsáveis pela gestão cinemática e comunicação entre os diversos módulos do sistema.

Ao longo das secções seguintes, são justificados os componentes selecionados e descritas as adaptações realizadas para cumprir os requisitos de estabilidade, torque e tempo de resposta do robô.

2.1 Manufatura Aditiva

Os modelos tridimensionais dos componentes estruturais do braço robótico tiveram como base um projeto já existente [2], devidamente citado nas referências.

Para o processo de fabrico via manufatura aditiva (Impressão 3D), recorreu-se aos recursos laboratoriais disponibilizados pela instituição de ensino. O equipamento selecionado para a produção integral das peças foi a impressora 'Bambu Lab A1'. Esta escolha justificou-se pela sua superior velocidade de deposição e elevada precisão dimensional em comparação com as restantes alternativas disponíveis no laboratório.

Relativamente à matéria-prima, optou-se pela utilização de filamento PLA (Ácido Polilático), devido às suas características mecânicas adequadas e facilidade de impressão.

Figura 1: Impressora Bambu Lab A1 durante o processo de impressão.

Durante a fase inicial de fabrico, identificou-se um desafio técnico: após a deposição das primeiras camadas, observou-se o levantamento de uma das extremidades da peça, fenómeno tecnicamente designado por *warping*. Em consequência desta anomalia, foi necessário interromper o processo de impressão.

Numa primeira tentativa de mitigação, procedeu-se à elevação da temperatura da base aquecida (*bed*) de 65 ºC para 70 ºC, com o intuito de potenciar a aderência do material. Simultaneamente, reduziu-se a velocidade de impressão para o modo 'Silent' (50%). Contudo, estas medidas não se revelaram eficazes, persistindo o problema de descolamento.

Como solução definitiva, recorreu-se à implementação da técnica de 'Brim' (aba), que consiste na geração de uma estrutura periférica adicional em torno da peça. Esta técnica permitiu ampliar a área de contacto com a superfície de impressão, maximizando a aderência. Nesta configuração final, manteve-se a velocidade no modo 'Standard' (100%) e a temperatura da base nos 65 ºC, obtendo-se resultados satisfatórios.

As principais configurações da impressora para a impressão foram as seguintes:

Definições de impressão	
Temperatura do extursor	220º.C
Temperatura da mesa	65º.C
Velocidade de Impressão	"Standart" 100%
"Wall Loops" por peça	2
Tamanho da camada	0.20mm
Densidade do "Infill"	30%

Tabela 1: Configurações da impressora

2.2 Base do Braço e Estrutura

De forma a garantir a estabilidade mecânica do sistema e fornecer um grau de liberdade adicional (*Yaw*) ao conjunto, foi desenvolvido um suporte estrutural personalizado. Ao contrário da configuração padrão do projeto InMoov, que contempla um torso completo, este projeto exigiu a conceção de uma base compacta capaz de alojar os microcontroladores e suportar as cargas axiais e radiais exercidas pelo movimento do braço.

O design mecânico foi modelado em ambiente CAD (SolidWorks), priorizando a modularidade e a facilidade de montagem através de conexões parafusadas, conforme ilustrado na **Figura 4**.

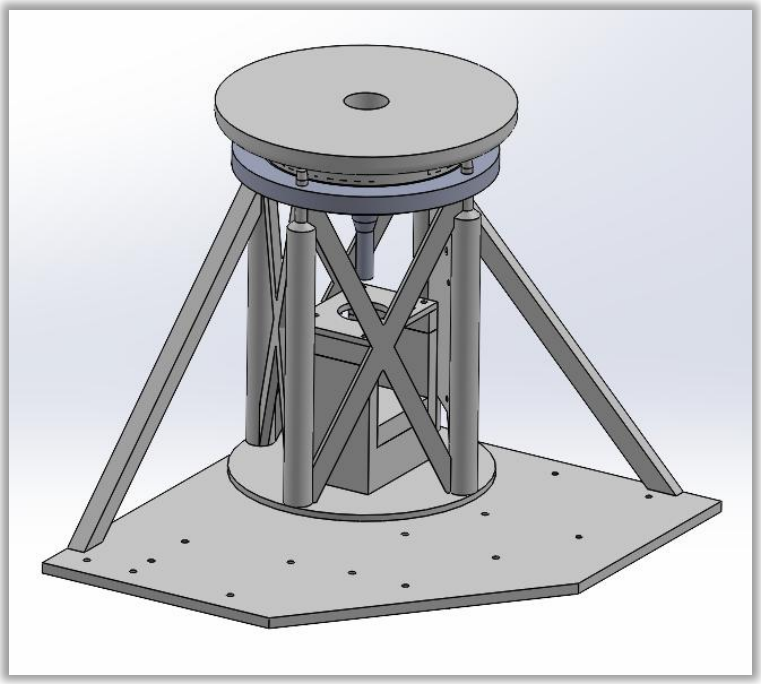

Figura 2: Modelo CAD da base do braço (montagem em Solidworks).

2.2.1 Componentes da Base:

- **Estrutura principal da base (Fig. 5):** Componente estrutural que aloja o Motor de passo, o Raspberry Pi, o Arduino MEGA e o Arduino Uno. Desempenha ainda a função crítica de suportar a carga axial (peso) do braço robótico, isolando o veio do motor de passo desse esforço mecânico.

- **Espaçadores do Motor de passo (Fig. 6 e 7):** Peças de extensão desenhadas para garantir o alinhamento vertical correto, permitindo que o veio do motor alcance o Acoplamento Rotativo (Fig. 10).

- **Interface de Acoplamento do Braço (Fig. 8):** Devido à área de contacto reduzida do rolamento (Fig. 12), foi necessário desenvolver uma interface intermédia. Esta estrutura é composta por dois segmentos: a **Base do Braço Inferior** (Fig. 8) e a **Base do Braço Superior** (Fig. 9). A base inferior acopla-se diretamente ao rolamento, enquanto a base superior se fixa à inferior, garantindo uma superfície estável para o braço.

- **Acoplamento Rotativo (Fig. 10):** Elemento de transmissão mecânica responsável pela união solidária entre o rolamento e o veio do motor de passo.

- **Suporte de Fixação do Motor (Fig. 11):** Componente que assegura a fixação rígida do motor à estrutura principal da base.

- **Rolamento (Fig. 12):** Elemento de ligação mecânica que conecta a estrutura estática da base à parte móvel (Base do braço inferior). Trata-se de um componente comercial (hardware não impresso).

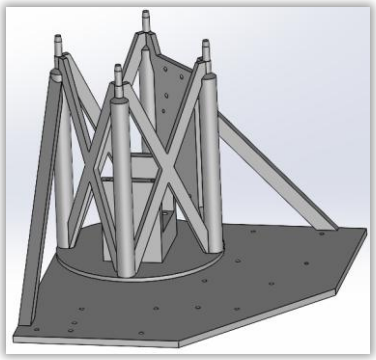

Figura 3

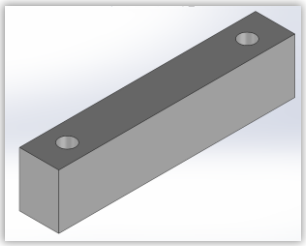

Figura 4

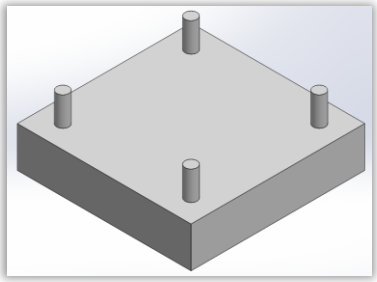

Figura 5

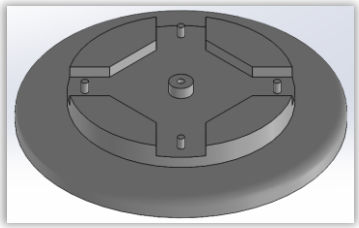

Figura 6

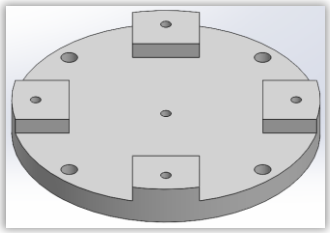

Figura 7

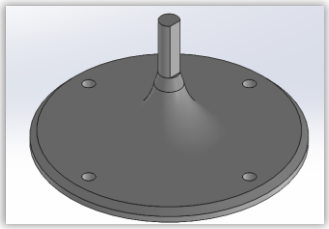

Figura 8

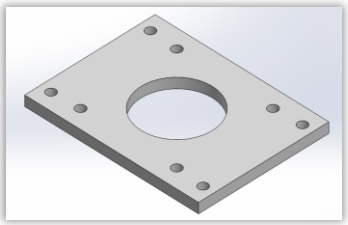

Figura 9

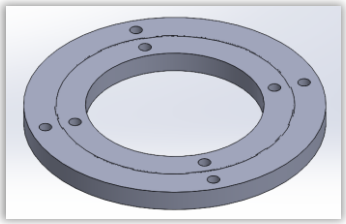

Figura 10

2.3 Atuadores e Cinemática

O processo de seleção dos servomotores baseou-se no cálculo do binário (*torque*) necessário para realizar os movimentos críticos do braço, garantindo a capacidade de suporte de carga e a estabilidade do sistema.

2.3.1 Motor do Cotovelo (*Pitch*)

Para esta articulação, o cálculo focou-se no binário estático necessário para vencer a força gravítica e sustentar o peso do antebraço e da mão. A equação fundamental do binário é dada por:

$$\tau = F \cdot l$$

Onde:

- τ: Binário ($N \cdot m$)
- F: Força Peso (N), calculada por $F = m \cdot g$
- l: Comprimento do braço de alavanca (distância ao ponto de rotação)

Considerando os seguintes parâmetros do projeto:

- Comprimento do braço de alavanca (l): 0.25m (25cm)
- Massa estimada do conjunto (m): $\sim 1\ Kg$
- Aceleração gravítica (g): $9.80665 m/s^2$

Obtém-se o binário nominal:

$$F_{base} = m \cdot g$$

Combinando as fórmulas, temos:

$$\tau = m \cdot g \cdot l$$

Substituindo a fórmula:

$$\tau_{base} = mgl \rightarrow 1 * 9.80665 * 0.25 = 2.45 N \cdot m$$

Para garantir a robustez do sistema face a acelerações dinâmicas, vibrações e peso de cablagens não contabilizado, aplicou-se um **Fator de Segurança (FS)** de 1.1.

$$\tau_{final} = 2.45 * 1.1 = 2.7 N \cdot m$$

Convertendo para a unidade comercial de servomotores ($1\ N \cdot m \sim 10.197\ kgf\ \cdot cm$), o requisito mínimo é de aproximadamente **27.5 kgf · cm**

Seleção: A escolha recaiu sobre o modelo **DS5160**, um servo digital com capacidade de **60 kgf · cm**. Embora sobredimensionado face ao valor calculado, esta opção foi ditada pela disponibilidade de *stock* na instituição, oferecendo em contrapartida uma excelente margem de segurança e estabilidade térmica.

2.3.2 Atuadores do Efeito Final (Dedos)

Para dimensionar os motores da mão, considerou-se a força de atrito necessária para a preensão segura de um objeto de massa conhecida (ex: bola de ténis), tendo em conta as propriedades do material dos dedos (PLA).

Parâmetros de Cálculo:

- Número de dedos (n): 5
- Massa do objeto (m): 0,06 kg
- Coeficiente de atrito estático (μ): 0,2 (caso desfavorável) a 0,5 (caso favorável)
- Comprimento do braço de momento (Ltotal): ≈ 0,18 m
- Fator de Segurança (FS): 2

Cálculo das Forças:

O peso do objeto (W) é calculado como:

$$W = m \cdot g = 0.06 \cdot 9.80665 = 0.59N$$

Força normal necessária em cada dedo (sem fator de segurança):

$$N_{\text{cada}} = \frac{W}{\mu \cdot n}$$

Para $\mu = 0.2$:

$$N_{\text{cada}} = \frac{0.59}{0.2 \cdot 5} = 0.59N$$

Aplicando o fator de segurança:

$$N_{\text{req}} = SF \cdot N_{\text{cada}} = 2 \cdot 0.59 = 1.18N$$

Binário necessário no atuador (caso desfavorável, μ = 0.2):

$$\tau_{\text{motor}} = N_{\text{req}} \cdot L_{\text{total}} = 1.18 \cdot 0.18 = 0.22N \cdot m \ (\sim 2.24kgf \cdot cm)$$

Caso favorável (μ = 0.5):

$$N_{\text{cada}} = \frac{0.59}{0.5 \cdot 5} = 0.24N \cdot m$$
$$N_{\text{req}} = 2 \cdot 0.24 = 0.47N \cdot m$$
$$\tau_{\text{motor}} = 0.47 \cdot 0.18 = 0.085N \cdot m \ (\sim 0.87kgf \cdot cm)$$

Análise Crítica e Seleção:

Embora o cálculo teórico aponte para um requisito máximo de 2,24 kgf.cm, este modelo idealizado não contempla perdas mecânicas significativas do sistema real, tais como:

1. Atrito nos tendões e nas articulações impressas em 3D;
2. Elasticidade e deformação do material (PLA);
3. Distribuição desigual de carga entre os dedos;
4. Resistência mecânica oferecida pelas molas de retorno.

Estima-se que estas ineficiências possam elevar o requisito real de binário entre 5 e 10 vezes o valor teórico. Consequentemente, optou-se pela utilização de servomotores MG996R de engrenagens metálicas, com binário entre 9 e 11 kgf.cm. Esta escolha garante a força necessária para uma preensão firme, compensando todas as perdas por atrito e ineficiências mecânicas do sistema de cabos.

Para a base utilizou-se um motor de passo pela sua precisão.

Motores utilizados:

REFERÊNCIA	FORÇA	QUANTIDADE	UTILIDADE
SERVO DS5160	60kgf/cm	1	Cotovelo
SERVO MG996R	9-11kgf/cm	6	Dedos e Pulso
STEP-MOTOR 17HS4401S	4.3kgf/cm	1	Base

Tabela 2: Motores utilizados

2.4 Arquitetura de Hardware e Eletrónica

A arquitetura do sistema foi estruturada com base numa abordagem distribuída, compreendendo três unidades de processamento distintas: um **Raspberry Pi 5** (RPi), responsável pelas tarefas de visão computacional de alto nível; um **Arduino MEGA 2560**, encarregue do controlo lógico de baixo nível e coordenação geral; e um **Arduino Uno**, dedicado exclusivamente ao acionamento e controlo do motor de passo da base.

- **Raspberry Pi 5 (Visão Computacional)** O Raspberry Pi 5 consiste num *Single Board Computer* (SBC) de elevado desempenho, operando sob o sistema operativo Linux. Distingue-se dos microcontroladores convencionais pela integração de um processador Quad-core Arm Cortex-A76 e memória RAM dedicada.

- **Função no Projeto:** Este dispositivo é responsável pela aquisição e processamento de vídeo em tempo real. A sua seleção deveu-se à capacidade de processamento necessária para executar tarefas computacionalmente exigentes, inviáveis para um microcontrolador padrão, tais como a execução de algoritmos de Inteligência Artificial e o cálculo da cinemática do braço. O *software* de visão foi implementado através de *scripts* na linguagem Python.

- **Arduino MEGA 2560 (Controlo Central)** O Arduino MEGA é uma placa de desenvolvimento baseada no microcontrolador ATmega2560, atuando neste sistema como a unidade central de coordenação e controlo.

    - **Função no Projeto:** A opção pelo modelo MEGA, em detrimento do Uno para esta função específica, justifica-se pela disponibilidade de quatro portas de comunicação série por *hardware* (UART - Serial1, Serial2, etc.). Esta característica permite estabelecer comunicação de alta velocidade com o Raspberry Pi numa porta dedicada e, simultaneamente, transmitir comandos para o Arduino Uno numa segunda porta, evitando o uso de emulação por *software* (*SoftwareSerial*) e a consequente latência. Programado em C/C++, este módulo recebe os comandos angulares do RPi e gera os sinais de controlo precisos para os servomotores.

- **Arduino Uno (Controlo do Motor de Passo)** Esta unidade foi alocada exclusivamente ao controlo do motor de passo da base. Programado em C/C++, o Arduino Uno recebe as instruções de posicionamento provenientes do Arduino MEGA e converte-as nos sinais elétricos necessários para o acionamento preciso do motor.

2.4.1 Diagrama de Blocos do Hardware

Mostra como os cabos ligam as placas.

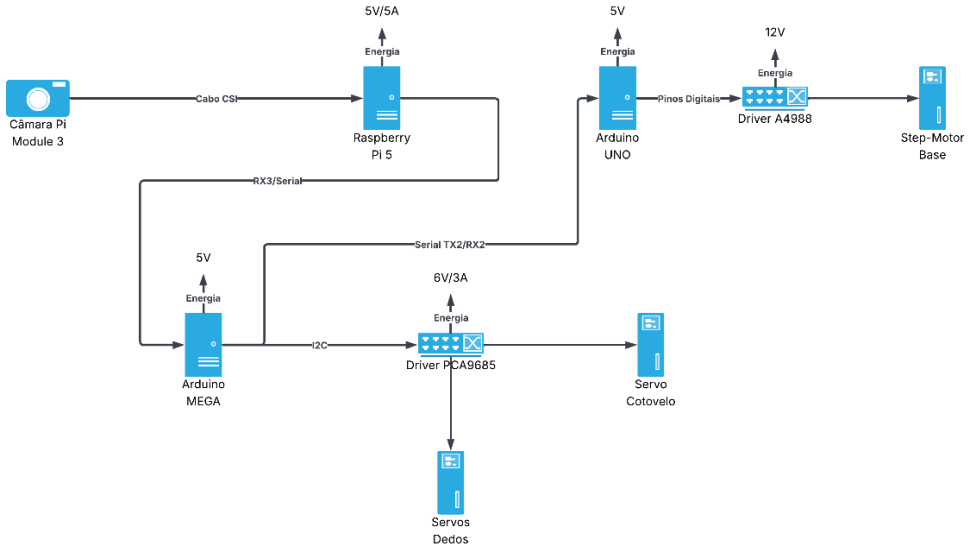

Figura 11: Diagrama de Blocos

2.4.2 Eletrónica e Alimentação

Devido ao elevado consumo energético dos atuadores mecânicos, optou-se pela utilização de duas fontes de alimentação de bancada independentes para isolar os circuitos de potência: uma fonte configurada a **7V/3A** dedicada aos servomotores e uma segunda fonte a **13V/2A** exclusiva para o motor de passo.

Relativamente aos microcontroladores, a alimentação do Arduino MEGA e do Arduino Uno é assegurada através da conexão USB. Por sua vez, o Raspberry Pi 5 recebe energia através de uma fonte de alimentação dedicada (transformador) ou baterias oficiais, garantindo a estabilidade de corrente necessária para o processamento.

- **Servomotores (MG996R e DS5160):** O conjunto de 7 servomotores (responsáveis pela actuação dos 5 dedos, do pulso e do cotovelo) recebe os sinais de controlo e a alimentação de potência através do módulo *driver* PWM (PCA9685), cujo funcionamento será detalhado na secção seguinte.
- **Motor de Passo (17HS4401S):** O controlo da base é efetuado por intermédio de um *driver* **A4988**. Este componente interpreta os sinais lógicos de *STEP* (passo) e *DIR* (direção) enviados pelo Arduino, sendo a alimentação de potência (13V, 2A) fornecida pela fonte externa mencionada anteriormente.

2.4.2.1 Eletrónica de Controlo (PCA9685)

Um desafio comum em projetos de robótica é o facto de o microcontrolador (Arduino) não possuir pinos PWM em número suficiente ou com corrente estável para controlar múltiplos motores em simultâneo. Por esse motivo, implementámos o controlador **PCA9685**. Este módulo permite controlar até 16 servomotores utilizando apenas **dois pinos do Arduino (barramento I2C)**. Além disso, permite ligar uma fonte de alimentação externa dedicada aos motores, o que evita que o Arduino sofra quedas de tensão ou danos por picos de consumo de corrente

2.5 Esquema Elétrico e Interconexões

A integridade do sistema robótico depende de uma arquitetura elétrica robusta, capaz de interligar componentes com diferentes níveis de tensão lógica (3.3V no Raspberry Pi e 5V nos Arduinos) e isolar os circuitos de alta corrente (motores) dos circuitos de processamento.

A **Figura 12** ilustra o diagrama esquemático completo desenvolvido na plataforma *Cirkit Designer*. Para uma análise detalhada de cada conexão e pinout, recomenda-se a consulta do diagrama digital interativo disponível nas referências **[14]**.

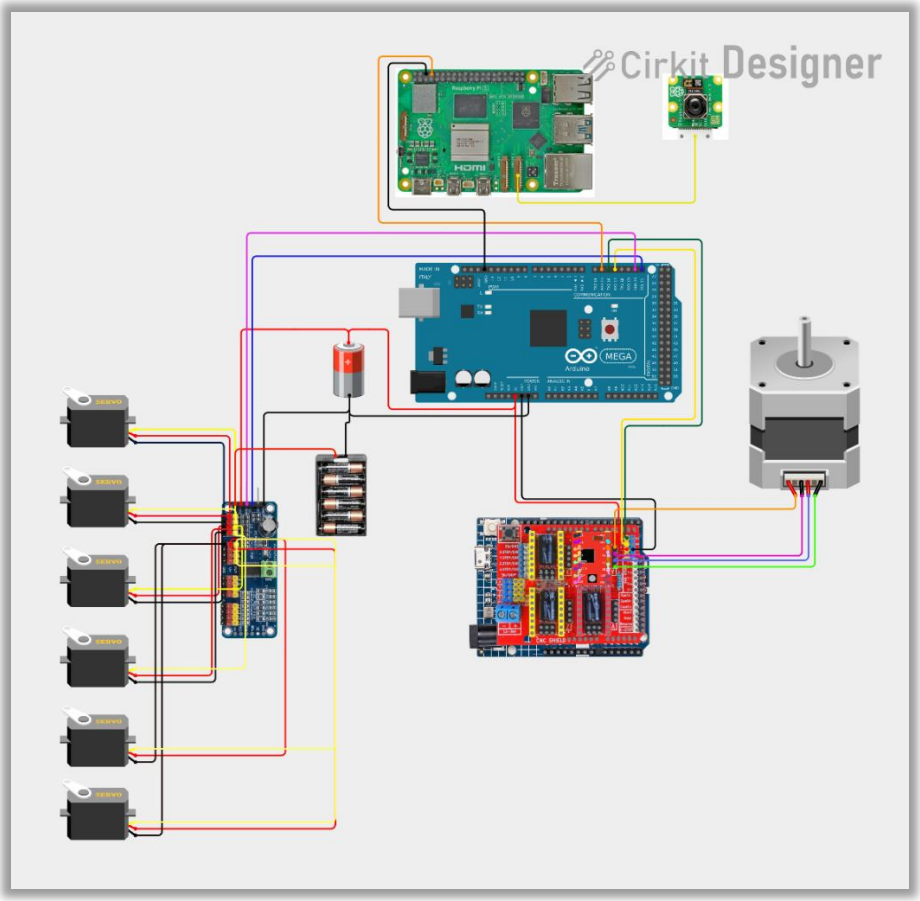

*Figura 12: Diagrama de ligações elétricas e fluxo de sinais (Gerado em Cirkit Designer). Versão interativa disponível em **[14]**.*

2.5.1 Matriz de Interconexões

Para facilitar a leitura técnica e a replicação do projeto, a **Tabela 3** resume as interfaces físicas estabelecidas entre os principais módulos do sistema.

ORIGEM (MESTRE /SINAL)	DESTINO (ESCRAVO/RECEPTOR)	TIPO DE CONEXÃO	PINOS ORIGEM	PINOS DESTINO	DESCRIÇÃO DA FUNÇÃO
RASPBERRY PI 5	**Arduino MEGA**	UART (Série)	GPIO 14 (TX)	Pin 19 (RX1)	Envio de coordenadas de visão (Mão/Corpo).
ARDUINO MEGA	**Arduino Uno**	UART (Série)	Pin 18 (TX1)	Pin 0 (RX)	Envio de comandos p/ base ('0', '1', '2').
ARDUINO MEGA	**Driver PCA9685**	I2C	SDA (20) / SCL (21)	SDA / SCL	Controlo PWM dos 6 Servomotores.
ARDUINO UNO	**CNC Shield V3**	*Shield* (Direto)	Pinos 2, 5, 8	X-Step, X-Dir, En	Controlo do Motor de Passo NEMA 17.
FONTE 1 (LOGIC)	**Arduinos / RPi**	USB / 5V	-	VCC / 5V	Alimentação dos microcontroladores.
FONTE 2 (POWER)	**Atuadores**	Cabo AWG	-	Terminais Ext.	Alimentação de carga (Servos e NEMA 17).

Tabela 3: Mapeamento de Pinos e Protocolos de Comunicação.

2.5.2 Gestão de Energia e Isolamento

Um dos pontos críticos do projeto foi a separação dos barramentos de alimentação para evitar que o ruído elétrico dos motores interferisse no processamento lógico ou causasse *brownouts* (quedas de tensão) nos Arduinos.

A. Driver de Servos (PCA9685):

O módulo PCA9685 possui dois terminais de alimentação distintos, configurados da seguinte forma:

- **VCC (Lógica):** Alimentado com **5V** provenientes do Arduino MEGA. Este pino alimenta apenas o *chip* controlador do driver e a comunicação I2C.
- **V+ (Potência):** Conectado diretamente a uma fonte externa de **6V** de alta amperagem. Esta entrada fornece a corrente exclusiva para o movimento dos servomotores, garantindo que o Arduino não suporta a carga elétrica dos atuadores.

B. CNC Shield V3 (Motor de Passo):

O *shield* acoplado ao Arduino Uno recebe uma alimentação externa de **12V** nos bornes de potência, necessária para garantir o binário do motor NEMA 17. A lógica de controlo (STEP, DIR, ENABLE) opera a 5V, partilhando apenas o referencial de massa (**Common Ground**) com o restante sistema.

3 2.5.3. Configuração do Arduino Uno (Controlador da Base)

O Arduino Uno atua como um escravo dedicado, executando um *firmware* específico para a gestão do *driver* A4988. Conforme o código implementado, as ligações físicas no CNC Shield correspondem às definições padrão do Eixo X:

- **Pino 2 (STEP):** Recebe os pulsos de passo (onda quadrada) gerados pela função moveMotor().
- **Pino 5 (DIR):** Define o sentido de rotação (Horário/Anti-horário).
- **Pino 8 (ENABLE):** Ativo a nível baixo (LOW). É utilizado para energizar as bobinas do motor, garantindo o binário de retenção ("travão") quando o braço está parado.

Esta arquitetura hierárquica (Raspberry Pi -> Arduino MEGA -> Arduino Uno) assegura que o processamento pesado de imagem não introduz latência na geração dos sinais críticos de controlo dos motores.

3.1 Controlo de Baixo Nível (Firmware)

O subsistema de controlo de baixo nível constitui a camada de abstração entre o processamento de imagem (alto nível) e a atuação mecânica. Dada a complexidade cinemática do robô e a necessidade de gerir múltiplos sinais em tempo real, optou-se por uma **arquitetura de processamento distribuído**.

Ao contrário de uma abordagem centralizada, onde um único processador gere a visão e os motores, este projeto delega as tarefas críticas de temporização (*hard real-time*) a microcontroladores dedicados (ATmega2560 e ATmega328P). Esta estratégia oferece duas vantagens fundamentais:

1. **Estabilidade do Sinal:** Garante que a geração de sinais PWM (para os servos) e de pulsos de passo (para a base) não sofre interrupções ou latência devido ao processamento pesado da visão computacional no Raspberry Pi.
2. **Modularidade:** Permite testar e depurar cada articulação do robô individualmente.

O *firmware* foi desenvolvido em C++ (ambiente Arduino IDE), implementando uma topologia **Mestre-Escravo** (*Master-Slave*), conforme detalhado nas secções seguintes. O Arduino MEGA atua como o controlador central (Mestre), recebendo coordenadas da visão e distribuindo tarefas, enquanto o Arduino Uno atua como um controlador auxiliar (Escravo) dedicado exclusivamente à gestão precisa do motor de passo da base.

3.1.1 Subsistema de Controlo da Base (Arduino Uno 'Slave')

Para assegurar o posicionamento preciso do eixo de rotação da base (*Yaw*), optou-se por uma arquitetura distribuída onde um Arduino Uno opera como unidade secundária (*slave*). Esta abordagem isola o Arduino MEGA da tarefa de gerar sinais de alta frequência necessários para o controlo do motor de passo.

A. Implementação de Hardware e Potência:

A atuação mecânica é realizada por um motor de passo bipolar (modelo **17HS4401S**), gerido através de uma placa de expansão **CNC Shield V3**. A escolha deste *shield* oferece vantagens críticas ao projeto:

- **Isolamento de Potência:** Permite a conexão de uma fonte de alimentação externa dedicada ao motor (12V), separando-a da alimentação lógica do microcontrolador (5V).
- **Modularidade e Filtragem:** Integra *sockets* para drivers (utilizou-se o modelo **A4988**) e condensadores eletrolíticos para estabilização de tensão e supressão do ruído elétrico gerado pela comutação das bobinas.

Interconexões do Sistema: As ligações físicas na placa CNC Shield foram estabelecidas da seguinte forma:

- **Eixo X (Motor):** O motor de passo conecta-se diretamente à barreira de 4 pinos do driver X.
- **Interface de Comunicação:** Estabeleceu-se uma ponte UART cruzada, ligando o pino RX do Shield ao TX do Arduino MEGA (e vice-versa), permitindo a receção de comandos.
- **Referência Comum (Common Ground):** Os pinos GND de ambos os microcontroladores foram interligados para garantir um nível de tensão de referência zero comum, essencial para a estabilidade da comunicação lógica.

B. Lógica de Controlo e Firmware:

O *firmware* implementado é responsável pela gestão de estados da posição angular (0º, +90º, -90º). O ciclo de funcionamento divide-se em três etapas fundamentais:

1. Configuração de Pinos (Setup) Durante a inicialização, os pinos de controlo do driver A4988 são definidos como saídas digitais, operando com a seguinte lógica:

- **ENABLE (Ativação):** Opera com lógica inversa. É mantido em estado BAIXO (LOW) para energizar as bobinas e garantir o binário de retenção (*holding torque*). O estado ALTO (HIGH) desativaria o motor, libertando o eixo.
- **DIR (Direção):** Sinal binário que determina o sentido de rotação. A alternância entre HIGH e LOW inverte a sequência de energização das fases.
- **STEP (Passo):** Responsável pela execução do movimento discreto. Cada transição ascendente (de LOW para HIGH) resulta no avanço de um passo mecânico (1.8º).

2. Processamento e Decisão O sistema monitoriza continuamente o *buffer* da porta série. Ao detetar um comando válido ('0', '1' ou '2'), o algoritmo compara o valor recebido com a variável global posicaoAtual. Se houver divergência, calcula o diferencial de passos e define o estado do pino DIR para a rotação correta.

3. Atuação (Geração de Sinal) A função moveMotor() converte a decisão lógica em movimento físico, gerando uma onda quadrada no pino STEP. A velocidade angular é controlada com precisão através de um atraso de 500 microssegundos (delayMicroseconds) entre as transições de estado, garantindo um movimento suave e sem perda de passos.

(O código-fonte completo deste módulo encontra-se disponível no Anexo A).

3.1.2 Lógica de Controlo do Braço (Arduino MEGA 2560)

Arduino MEGA 2560 assume o papel de coordenador central do movimento robótico. Diferindo do Raspberry Pi (focado na visão computacional) e do Arduino Uno (dedicado ao motor de passo), este microcontrolador é responsável pela gestão simultânea de múltiplos protocolos de comunicação. O seu ciclo de funcionamento baseia-se em três etapas fundamentais:

1. Gestão do *Buffer* Série O processador monitoriza a porta Série 1 (*HardwareSerial*) aguardando a receção do marcador de início ($). Após a deteção, os *bytes* subsequentes são armazenados num *buffer* de memória até ser encontrado o marcador de fim (\n). Este mecanismo de validação previne a execução de comandos incompletos ou corrompidos, assegurando a integridade da operação.

2. *Parsing* (Interpretação de Dados) A cadeia de caracteres recebida (ex: "1,0,180,0,180...") é fragmentada utilizando a vírgula como delimitador. Os valores textuais extraídos são convertidos em números inteiros, representando os ângulos ou estados lógicos desejados para cada atuador.

3. Mapeamento e Atuação I2C Os valores angulares (0º a 180º) não são enviados diretamente para os servomotores. O Arduino converte estes valores em largura de pulso PWM (tipicamente mapeados entre 150 e 600 unidades discretas para o contador de 12-bits). Estes dados são posteriormente transmitidos via barramento I2C para o módulo controlador PCA9685.

O código-fonte completo desenvolvido para o Arduino MEGA encontra-se disponível no **Anexo C**.

Controlador PWM/Servo (PCA9685)

Para o controlo dos servomotores, recorreu-se à utilização do módulo **PCA9685**. Este *driver* de 16 canais permite a alimentação externa dos servos (isolando a carga de potência do microcontrolador) e utiliza uma interface I2C para o comando preciso dos sinais PWM. O esquema de pinagem (*pinout*) do PCA9685 encontra-se disponível no **Anexo 2**.

Interconexões do Sistema: As ligações efetuadas entre o módulo e o Arduino MEGA foram as seguintes:

- **VCC:** Ligado a 5V (Alimentação lógica).
- **GND:** Ligado ao Ground (Terra comum).
- **SDA:** Ligado ao pino SDA (Pino 20 no MEGA).
- **SCL:** Ligado ao pino SCL (Pino 21 no MEGA).

3.1.3 Protocolo de Comunicação UART

O estabelecimento da comunicação entre a unidade de processamento de imagem (Raspberry Pi) e o controlador central (Arduino MEGA) é concretizado através do protocolo **UART** (*Universal Asynchronous Receiver-Transmitter*), mediante a interligação física dos pinos de transmissão (TX) e receção (RX).

Neste processo, é transmitida continuamente uma cadeia de caracteres (*string*) formatada, contendo as instruções de controlo. De forma a garantir a sincronização e integridade dos dados, cada mensagem é encapsulada por um marcador de início ($) e um terminador de fim de linha (\n). A trama de dados é composta por 8 parâmetros separados por vírgulas.

Estrutura da Trama de Dados:

$<Orientação>,<Flexão>,<D1>,<D2>,<D3>,<D4>,<D5>,<Rotação>\n

Legenda dos Parâmetros:

- **Orientação (Base):** Valor inteiro (0, 1 ou 2) para controlo do motor de passo.
- **Flexão (Cotovelo):** Estado binário (0 ou 1) para extensão ou flexão.
- **D1 a D5:** Estado binário (0 ou 1) correspondente aos dedos (Indicador, Médio, Anelar, Mínimo e Polegar).
- **Rotação:** Valor analógico (0 a 180) correspondente ao ângulo do servo de rotação do pulso.

Exemplo de Comando:

- $1,0,1,1,1,1,1,90\n *(Interpretação: Base ao centro, Cotovelo estendido, Todos os dedos fechados, Pulso rodado a 90 graus).*

Processamento de Dados (Parsing): Do lado do recetor (Arduino MEGA), foi implementado um *parser* (analisador sintático) que monitoriza o *buffer* de entrada. O algoritmo aguarda a deteção do marcador $, procedendo à leitura da *string* até encontrar o terminador \n. Subsequentemente, os valores são segmentados com base no delimitador (vírgula) para distribuição pelos respetivos atuadores. Esta abordagem assegura que apenas pacotes de dados completos são processados, incrementando a robustez do sistema.

4 Sistema de Visão Computacional

O subsistema de visão computacional constitui a camada de inteligência do projeto, atuando como o elemento primário de interação homem-máquina (HMI). Ao substituir controladores físicos tradicionais (como *joysticks* ou botões) por algoritmos de reconhecimento gestual, este módulo permite uma operação intuitiva e sem contacto (*contactless*).

O presente capítulo descreve a arquitetura de *software* desenvolvida para o Raspberry Pi 5, fundamentando a seleção da *framework* MediaPipe para a extração de esqueleto humano. Serão detalhados os desafios de processamento em tempo real, a fundamentação matemática para o cálculo de ângulos articulares e a lógica de conversão destes dados em comandos de atuação mecânica.

4.1 Plataforma de Processamento

O núcleo computacional do sistema de visão artificial baseia-se no *Single Board Computer* (SBC) **Raspberry Pi 5**. A seleção desta plataforma foi determinante, pois a execução de modelos de *Deep Learning* em tempo real exige uma capacidade de processamento vetorial significativamente superior à dos microcontroladores convencionais.

Hardware Utilizado: Optou-se pela versão de **8GB de RAM** do Raspberry Pi 5, equipada com o processador Broadcom BCM2712 (Arm Cortex-A76 *quad-core* a 2.4GHz). Esta especificação garante a fluidez necessária (15-20 FPS) para a inferência simultânea dos modelos de mãos e postura corporal.

Acesso Remoto e Telemetria (Headless): Para garantir a portabilidade do robô e eliminar a necessidade de monitores ou periféricos físicos acoplados à estrutura, o sistema opera em modo *headless*. A interface de visualização e *debug* é acedida remotamente através do serviço **Raspberry Pi Connect**. A conexão de rede é estabelecida via **Hotspot Móvel 5G**, uma escolha estratégica para minimizar a latência de vídeo e garantir uma largura de banda estável, contornando as flutuações e restrições de segurança típicas de redes Wi-Fi locais partilhadas.

Interface de Comunicação (Hardware UART): Diferenciando-se das abordagens convencionais que utilizam cabos USB para comunicação série, este projeto implementa uma ligação direta ao nível do barramento. A transmissão de dados para o controlador utiliza a porta **UART (Universal Asynchronous Receiver-Transmitter)** dos pinos GPIO:

- **Ligação Física:** O pino **TX (GPIO 14)** do Raspberry Pi conecta-se diretamente ao pino **RX** do Arduino MEGA.
- **Vantagem Técnica:** Esta topologia elimina a *overhead* do protocolo USB e a necessidade de conversores USB-Série externos, reduzindo a latência de envio dos pacotes de comando e simplificando a cablagem interna do robô.

Ambiente de Software: O sistema corre sobre o **Raspberry Pi OS (Bookworm) 64-bit**, utilizando **Python 3** e as bibliotecas **OpenCV** e **MediaPipe** para o processamento visual, e **PySerial** para a gestão do fluxo de dados na porta UART.

4.2 Fluxograma do Software

Mostra como o código pensa.

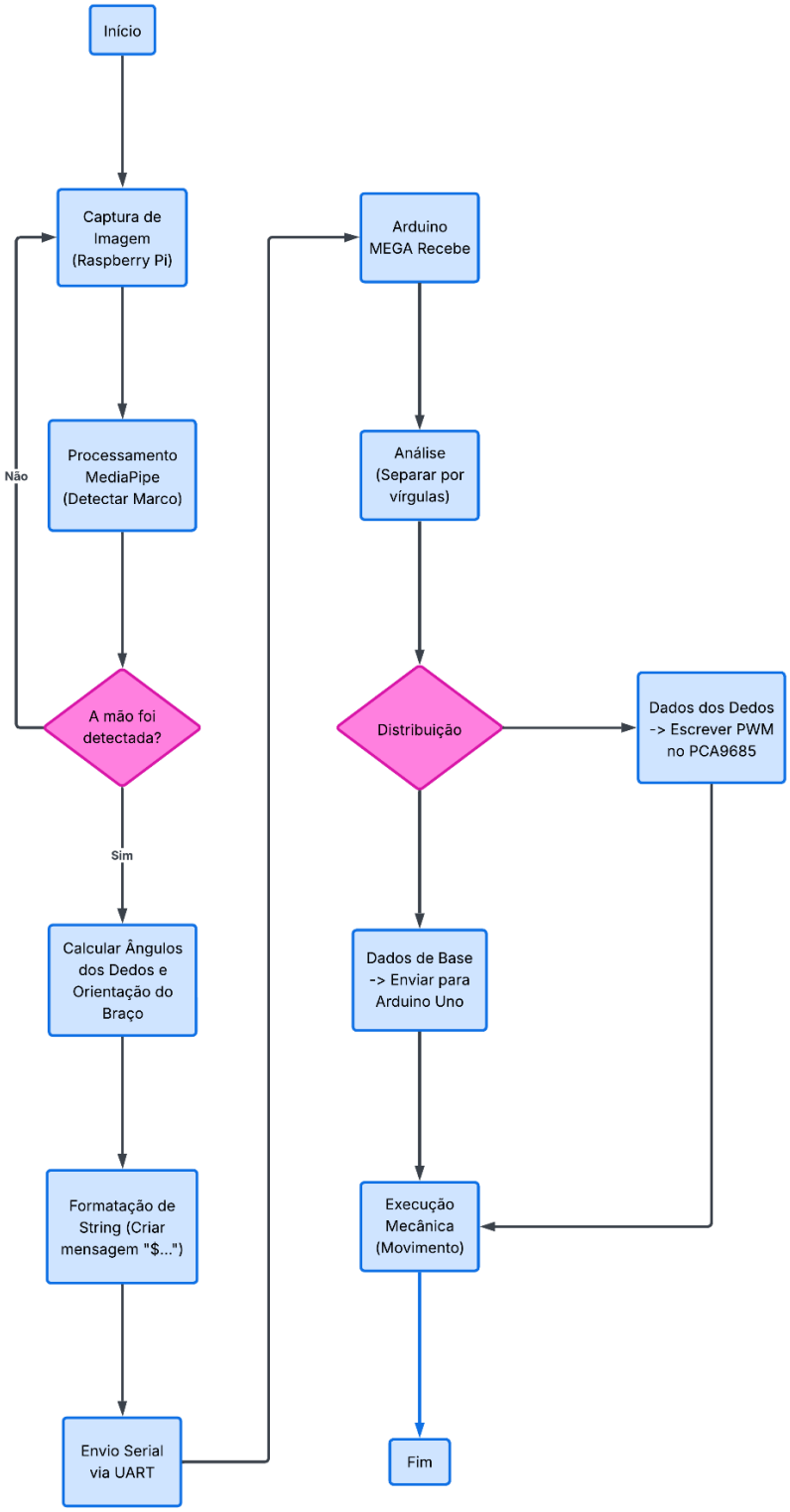

Figura 13: Fluxograma

4.3 Seleção da Ferramenta: Framework MediaPipe

Para a tarefa de deteção e rastreamento da mão e postura corporal, optou-se pela utilização do **MediaPipe**, uma *framework* de código aberto desenvolvida pela Google.

A escolha do MediaPipe, em detrimento de redes neuronais convolucionais (CNNs) tradicionais (como YOLO ou SSD) treinadas para deteção de objetos, justifica-se pela sua arquitetura otimizada para inferência em *hardware* com recursos limitados (CPU). Enquanto uma CNN típica processa toda a imagem a cada *frame* para encontrar uma "caixa delimitadora" (bounding box) à volta da mão, o MediaPipe utiliza uma abordagem em duas etapas:

1. **Deteção de Palma:** Um modelo leve localiza a mão na imagem inicial.
2. **Regressão de Landmarks:** Uma vez localizada a mão, um segundo modelo mais complexo estima a posição de **21 pontos esqueléticos 3D** (designados por *landmarks*) dentro dessa região, sem necessidade de reprocessar a imagem inteira.

Figura 14: Topologia dos 21 landmarks detetados pelo modelo MediaPipe Hands.

Esta abordagem permite atingir taxas de atualização elevadas (aproximadamente 20 FPS no Raspberry Pi 5), essenciais para um controlo robótico fluido e em tempo real. Foram utilizados dois módulos específicos:

- **MediaPipe Hands:** Para a deteção detalhada dos 21 pontos das articulações dos dedos e pulso (Figura 15).
- **MediaPipe Pose:** Para a deteção da postura superior do corpo, especificamente os pontos dos ombros, utilizado para controlar a rotação da base (Figura 16).

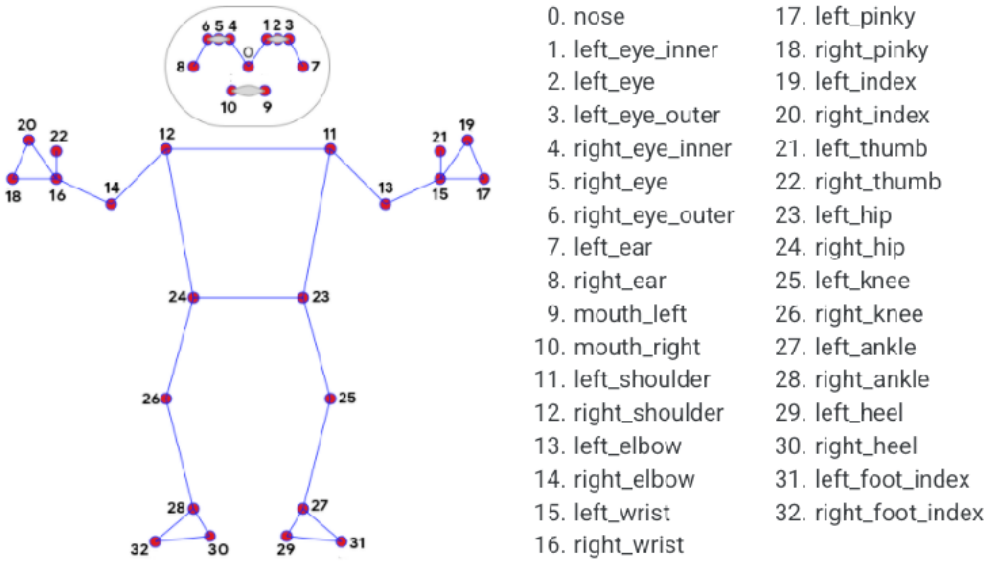

Figura 15: Mapeamento dos landmarks corporais do modelo MediaPipe Pose.

A visualização destes pontos pode ser observada na **Figura 16**. As linhas azuis representam as conexões esqueléticas da mão detetada pelo módulo *Hands*, enquanto a linha verde horizontal no topo representa a conexão entre os ombros detetada pelo módulo *Pose*.

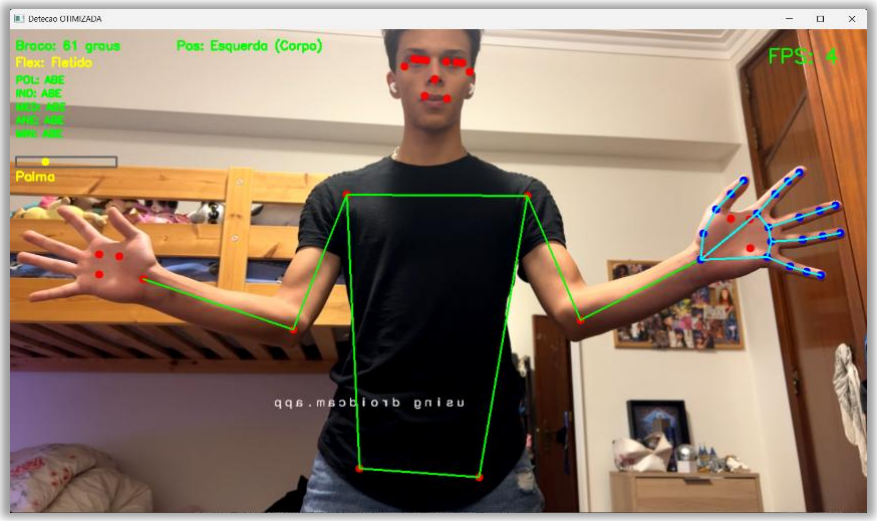

Figura 16: Visualização em tempo real das landmarks detetadas.

4.4 Metodologia de Desenvolvimento (PC vs. Raspberry Pi)

Devido à complexidade dos cálculos geométricos e à necessidade de calibração fina dos limiares (*thresholds*) de deteção, o desenvolvimento do *software* seguiu uma abordagem faseada.

Inicialmente, o algoritmo foi integralmente desenvolvido e testado num computador pessoal (PC) com maior poder de processamento. Foi criado um *script* de depuração dedicado (VisionDebugger_PC.py) que permite visualizar os ângulos calculados em tempo real no ecrã, facilitando o ajuste dos parâmetros sem a latência inerente ao *hardware* embarcado. Apenas após a validação robusta dos algoritmos, o código foi migrado e otimizado para o ambiente Linux do Raspberry Pi 5, garantindo uma transição suave do ambiente de desenvolvimento para o de produção.

4.5 Aquisição de Imagem e Multithreading

Para maximizar o desempenho, a captura de vídeo foi dissociada do processamento principal. Implementou-se uma classe dedicada (FluxoCamera no código) que utiliza *multithreading*. Um processo independente é responsável por ler os *frames* da câmara (utilizando a biblioteca OpenCV) e colocá-los num *buffer*. O ciclo principal do programa lê sempre o *frame* mais recente disponível neste *buffer*.

Esta técnica evita que o tempo de processamento da IA e dos cálculos matemáticos bloqueie a leitura da câmara, reduzindo significativamente a latência total do sistema.

4.6 Lógica de Controlo e Utilização de Landmarks

A tradução dos movimentos humanos para o robô baseia-se na análise geométrica dos *landmarks* fornecidos pelo MediaPipe. O sistema processa dois fluxos de dados distintos, visualizados na interface de depuração (**Figura 17**): as linhas de conexão **verdes** correspondem ao esqueleto corporal (*MediaPipe Pose*) e as linhas **azuis** correspondem à estrutura da mão (*MediaPipe Hands*).

Embora o MediaPipe forneça 21 pontos por mão, não é necessário utilizar todos para determinar o estado básico de preensão. Foram selecionados pontos estratégicos para reduzir a carga computacional.

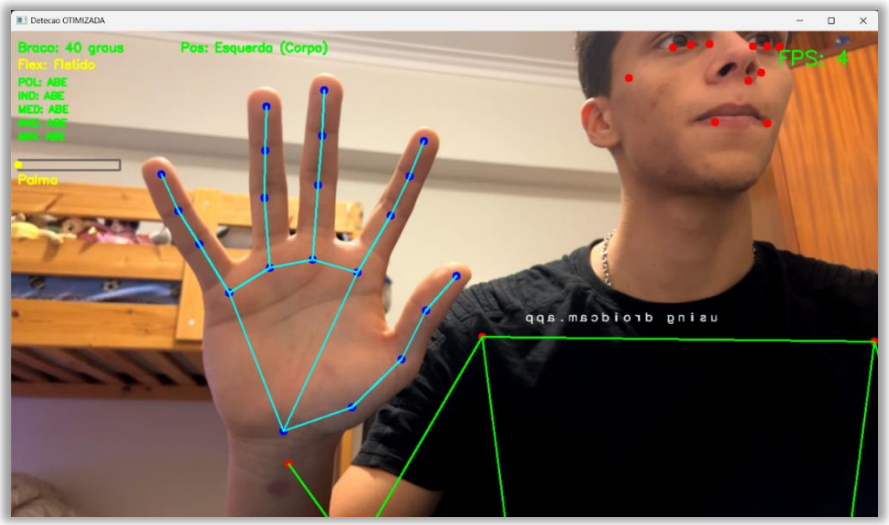

Figura 17: Interface de depuração em funcionamento

4.6.1 Fundamentação Matemática: Cálculo de Ângulos Articulares

Uma das decisões críticas do projeto foi não basear o estado dos dedos ("aberto" ou "fechado") na simples distância euclidiana entre a ponta do dedo e o pulso. Este método é falível, pois varia consoante a proximidade da mão à câmara (escala).

Em vez disso, optou-se por uma abordagem baseada na **geometria vetorial**, calculando o ângulo formado pelas articulações. Este método é invariante à escala e à rotação da mão. Para determinar o estado de um dedo, calcula-se o ângulo θ no vértice formado por três pontos-chave: a base do dedo (A), a articulação intermédia (B) e a ponta do dedo (C).

A função angulo_3pts, implementada no código, utiliza o produto escalar entre dois vetores (BA e BC) e a função arco-cosseno para determinar este ângulo:

$$\theta = arccos(\frac{BA * BC}{|BA||BC|})$$

O seguinte trecho de código ilustra a implementação desta função utilizando a biblioteca Math para eficiência vetorial:

```python
def calcular_angulo_3pts(a, b, c):
    """
    Matemática: Calcula o ângulo entre 3 pontos.
    (Exemplo: Ombro -> Cotovelo -> Pulso para saber se o braço está esticado)
    """
    ax, ay = a[0]-b[0], a[1]-b[1]
    cx, cy = c[0]-b[0], c[1]-b[1]
    num = ax*cx + ay*cy
    den = math.hypot(ax, ay) * math.hypot(cx, cy) + 1e-8
    return math.degrees(math.acos(max(-1.0, min(1.0, num/den))))
```

4.6.2 Estado dos Dedos (Aberto/Fechado)

Para determinar se um dedo se encontra fletido ou estendido, não se utiliza a distância entre pontos (que varia com a profundidade), mas sim o cálculo vetorial do ângulo na articulação intermédia.

Tomando como exemplo o dedo indicador, o algoritmo seleciona três pontos chave definidos no dicionário FINGERS do código:

- **Ponto 5 (MCP):** Articulação da base (referência proximal).
- **Ponto 6 (PIP):** Articulação intermédia (o vértice do ângulo).
- **Ponto 8 (TIP):** Ponta do dedo.

A função angulo_3pts calcula o ângulo "θ" (teta) no vértice (Ponto 6) através do produto escalar dos vetores formados pelos segmentos de reta entre os pontos.

- **Lógica:** Se o ângulo θ for superior a TH_INDICADOR (ex: 160º), o dedo considera-se **Aberto**. Caso contrário, considera-se **Fechado**. Esta lógica replica-se para os cinco dedos, garantindo robustez independentemente da rotação da mão.

Esta configuração repete-se para os restantes dedos, conforme o dicionário implementado no *software*:

```python
# 4. MAPA DOS DEDOS (Pontos do MediaPipe)
# Diz ao programa quais os "pontos" do esqueleto que formam cada dedo
DEDOS = {
    "polegar": [1, 2, 4],
    "indicador": [5, 6, 8],
    "medio": [9, 10, 12],
    "anelar": [13, 14, 16],
    "mindinho": [17, 18, 20]
}
```

4.6.3 Controlo da Base e Cotovelo (Direita / Centro / Esquerda)

Para a determinação da **Orientação da Base** (controlar se o robô roda para a esquerda, direita ou fica ao centro), o algoritmo não segue a posição X da mão na imagem (o que seria instável), mas sim a **abertura do braço** (ângulo do cotovelo).

Conforme o código desenvolvido (VisionDebugger_PC_PT.py) na função *determinar_orientacao_braco* e a topologia do MediaPipe Pose (Figura 16), os pontos utilizados são:

Se for detetado o lado Direito do utilizador (padrão):

- **Ponto 12 (Ombro Esquerdo/Direito na imagem):** Ponto de âncora proximal.
- **Ponto 14 (Cotovelo):** O **Vértice** onde é calculado o ângulo.
- **Ponto 16 (Pulso):** O ponto móvel que define a extensão.

(Nota: Se o sistema detetar o outro braço, usa os pontos simétricos: 11, 13 e 15).

Como funciona o cálculo: O software calcula o ângulo formado no **Cotovelo (Ponto 14)** entre o segmento do Ombro e o segmento do Pulso.

- **Braço Fechado (< 70º):** O sistema interpreta como comando "Esquerda".
- **Braço Aberto (> 130º):** O sistema interpreta como comando "Direita".
- **Intermédio:** O sistema mantém-se no "Centro".

Isto permite controlar a base apenas esticando ou recolhendo o braço, sem precisar de deslocar o corpo pela sala.

```python
# 3. LIMIARES DO BRAÇO
TH_BRACO_ESQUERDA = 70    # Graus para considerar virado à esquerda
TH_BRACO_CENTRO_MAX = 130 # Até quantos graus é centro?
TH_FLEXAO_Y = 0.1         # Quão levantado o braço precisa estar para ser "fletido"

def determinar_orientacao_braco(angulo):
    """Diz para onde o braço aponta baseado no ângulo do ombro."""
    if angulo <= TH_BRACO_ESQUERDA:
        return "Esquerda (Corpo)"
    elif angulo <= TH_BRACO_CENTRO_MAX:
        return "Centro (Frente)"
    else:
        return "Direita (Fora)"
```

4.6.4 Flexão do Cotovelo (Cima / Baixo)

Para a **Flexão do Cotovelo (Movimento Cima / Baixo - Pitch)**, o algoritmo não utiliza ângulos, mas sim a diferença de altura (coordenada Y) entre dois pontos específicos para determinar a intenção do operador.

Conforme a implementação na função *determinar_flexao_braco* no ficheiro VisionDebugger_PC_PT.py e o mapa corporal na **Figura 16**, os pontos utilizados são:

- **Ponto 11 ou 12 (Ombro):** Serve como ponto de referência estático (altura base).
- **Ponto 15 ou 16 (Pulso):** Serve como ponto móvel.

Lógica de Funcionamento:

O sistema calcula a distância vertical absoluta entre o ombro e o pulso:

$$\Delta Y = |Y_{ombro} - Y_{pulso}|$$

- Ao contrário da rotação da base (que usa 3 pontos para um ângulo), aqui compara-se apenas a **posição relativa vertical** do pulso em relação ao ombro.

- Se o pulso estiver muito abaixo do ombro (grande diferença em Y), o braço está "em baixo".
- Se o pulso subir e se alinhar com o ombro (pequena diferença em Y), o sistema deteta a elevação.

```python
# 3. LIMIARES DO BRAÇO
TH_BRACO_ESQUERDA = 70    # Graus para considerar virado à esquerda
TH_BRACO_CENTRO_MAX = 130 # Até quantos graus é centro?
TH_FLEXAO_Y = 0.1         # Quão levantado o braço precisa estar para ser "fletido"

def determinar_flexao_braco(ombro_y, pulso_y):
    """
    Diz se o braço está levantado (fletido).
    Compara a altura (Y) do ombro e do pulso.
    """
    diferenca_y = abs(ombro_y - pulso_y)
    if diferenca_y > TH_FLEXAO_Y:
        return "Fletido", diferenca_y
    else:
        return "Estendido", diferenca_y
```

4.6.5 Rotação do Pulso (Roll)

Visto que o MediaPipe não fornece nativamente o ângulo de *Roll* (rotação da palma), foi desenvolvido um algoritmo heurístico na função *calcular_rotacao_mao*. Este método compara a posição relativa da ponta do **Polegar (Ponto 4)** e do **Dedo Mínimo (Ponto 20)**.

A inovação deste algoritmo reside na **troca dinâmica de eixos**:

- **Caso A (Braço Fletido):** A rotação é calculada pela diferença no **Eixo X** (horizontal) entre o polegar e o mínimo.
- **Caso B (Braço Estendido):** A rotação é calculada pela diferença no **Eixo Y** (vertical).

O valor diferencial resultante é normalizado num intervalo de -1 a 1 e mapeado para o intervalo de 0º a 180º, permitindo que o servo da rotação copie o movimento natural da mão do operador, quer este esteja com o braço levantado ou em baixo.

```python
def calcular_rotacao_mao(marcos_mao, status_flexao, orientacao_braco, lado="Direita"):
    """
    Calcula se a mão está de Palma ou de Costas (Dorso).
    Mede a distância entre o Polegar e o Mindinho.
    Se a distância for positiva é palma, negativa é dorso (ou vice-versa).
    """
    polegar = marcos_mao[4]  # Ponto na ponta do polegar
    mindinho = marcos_mao[20] # Ponto na ponta do mindinho

    diferenca = 0

    # A lógica muda dependendo se o braço está dobrado ou esticado.
    esta_fletido = (status_flexao == "Fletido")

    if esta_fletido:
        # CASO 1: Braço Fletido (Dobrado)
        # Usamos diferença no eixo X (Horizontal)
        val_p = polegar.x
        val_m = mindinho.x

        # LÓGICA CORRETA PARA IMAGEM ESPELHADA (Modo Selfie):
        if lado == "Direita":
            # Mão Direita (no ecrã Direita): Palma significa Polegar(Esq) < Mindinho(Dir) -> Diferença é Negativa
            diferenca = -(val_p - val_m)
        else:
            # Mão Esquerda (no ecrã Esquerda): Palma significa Polegar(Dir) > Mindinho(Esq) -> Diferença é Positiva -> Inverter
            diferenca = (val_p - val_m)

        eixo_usado = "X (Fletido)"

    else:
        # Braço Estendido
        if "Direita" in orientacao_braco:
            # CASO 3: Estendido para Direita -> Usar eixo Y (Vertical)
            val_p = polegar.y
            val_m = mindinho.y
            diferenca = (val_p - val_m)
            eixo_usado = "Y (Est-Dir)"

        else:
            # CASO 2: Estendido para Centro/Corpo -> Usar eixo Y Invertido
            val_p = polegar.y
            val_m = mindinho.y
            diferenca = -(val_p - val_m)
            eixo_usado = "Y-Inv (Est-Corpo)"

    # Normalização: Converte a diferença bruta de pixels num valor padrão entre -1 e 1
    DIF_MAXIMA = 0.22
    valor_rotacao = max(-1.0, min(1.0, diferenca / DIF_MAXIMA))

    # Classificar por texto para Depuração (Debugging)
    orientacao = "Indefinido"
    if valor_rotacao < -0.3:
        orientacao = "Palma"
    elif valor_rotacao > 0.3:
        orientacao = "Dorso"
    else:
        orientacao = "A Rodar / Canto"

    return diferenca, valor_rotacao, orientacao, eixo_usado
```

5 Resultados

A fase final do projeto consistiu na integração dos subsistemas de visão computacional, controlo lógico e mecânica. Os testes realizados validaram a arquitetura do sistema, embora tenham sido identificados desvios face ao desenho teórico inicial.

5.1 Resultados da Construção

Após a conclusão dos ciclos de impressão, procedeu-se à remoção das estruturas de suporte e à limpeza de todos os componentes (rebarbamento). Nesta fase, foi realizada uma inspeção visual rigorosa para validar a integridade estrutural das peças e a aderência entre camadas, garantindo que não existiam deformações críticas que comprometessem a montagem mecânica. A **Figura 2** ilustra o conjunto completo de peças preparadas para a fase de assemblagem, dispostas para inventariação.

Figura 18: Conjunto de componentes estruturais do braço produzidos em PLA.

Seguidamente, iniciou-se a pré-montagem estrutural do braço e da mão, ainda sem a integração dos atuadores (servomotores) e da cablagem. Esta etapa intermédia foi fundamental para verificar as tolerâncias dimensionais e o correto acoplamento entre as juntas rotativas e as peças estáticas.

Como se pode observar na **Figura 3**, o modelo InMoov demonstrou uma boa estabilidade mecânica apenas com as conexões estruturais, validando a qualidade da impressão 3D em PLA. Nesta configuração, confirmou-se a liberdade de movimento dos dedos e do pulso antes da fixação definitiva dos componentes eletrónicos.

Figura 19: Vista geral do sistema robótico parcialmente montado.

A montagem física do robô, baseada na manufatura aditiva (Impressão 3D em PLA), revelou desafios de tolerância e ajuste que exigiram intervenções corretivas durante a fase de construção:

- **Incompatibilidades Geométricas:** Na montagem da estrutura de suporte, verificou-se uma interferência entre a cabeça dos parafusos de fixação e a superfície de contacto entre a **Base do Braço Superior** (Fig. 9) e a **Base do Braço Inferior** (Fig. 8). Foi necessário realizar um pós-processamento manual (desbaste de material) para garantir o nivelamento e o acoplamento correto das peças.
- **Resistência Estrutural e Tolerâncias:** Durante a inserção do rolamento mecânico na **Estrutura Principal** (Fig. 5), observou-se uma fratura nas extremidades do componente impresso. Esta falha resultou de um ajuste de interferência excessivo para a rigidez do PLA, agravado pelas tensões internas do material arrefecido. A peça foi recuperada através de reforço estrutural localizado, permitindo a continuidade dos ensaios.

Figura 20: Componente da base superior do braço sujeito a alterações pós-impressão.

5.2 Validação da Visão Computacional

A fase final do projeto consistiu na integração total dos subsistemas de visão computacional, processamento lógico e atuação mecânica. Os testes realizados em ambiente real demonstraram o sucesso da arquitetura distribuída (Raspberry Pi + Arduinos), validando a capacidade do braço robótico em mimetizar os movimentos humanos com elevada fidelidade.

Conforme ilustrado na **Figura 19**, o sistema operou de forma autónoma, traduzindo a detecção dos *landmarks* faciais e manuais em comandos mecânicos precisos.

Análise de Desempenho:

1. **Tempo de Resposta (Latência):** O sistema de visão, a correr no Raspberry Pi 5, manteve uma taxa de atualização estável (entre 15 a 20 FPS). A latência entre o gesto do operador e o movimento do servomotor foi percetível, mas reduzida o suficiente para ser considerada "tempo real" para aplicações de telepresença.
2. **Precisão da Mimetização:**
    - **Dedos:** O algoritmo vetorial (baseado em ângulos) provou ser robusto, ignorando falsos positivos causados pela rotação da mão ou variação da distância à câmara.
    - **Base e Cotovelo:** A lógica de controlo gestual (baseada na extensão do braço e altura do pulso) permitiu controlar os motores de maior carga de forma intuitiva, sem exigir que o operador se deslocasse fisicamente da frente da câmara.
3. **Estabilidade Mecânica:** O protocolo de comunicação UART implementado garantiu que não existissem perdas de pacotes de dados, resultando num movimento fluido, embora limitado pelas restrições mecânicas dos próprios servomotores (velocidade de rotação e binário).

Em suma, a demonstração visual (**Figura 19**) comprova que o sistema atingiu os objetivos propostos: uma interface natural "humano-máquina" onde o corpo do operador funciona como o próprio comando do robô.

Figura 21: Demonstração funcional do sistema integrado

O desenvolvimento do módulo de visão computacional representou o desafio lógico mais complexo do projeto, constituindo a camada de abstração que permite a interação natural entre o operador humano e o sistema robótico.

A opção pela *framework* **MediaPipe** revelou-se acertada para a arquitetura de *hardware* limitada do Raspberry Pi 5. Ao contrário de redes neuronais de deteção de objetos convencionais (como YOLO), a abordagem baseada em *landmarks* esqueléticos permitiu manter uma taxa de processamento elevada (aproximadamente 20 FPS), garantindo uma latência mínima crítica para aplicações de telepresença.

Do ponto de vista algorítmico, conclui-se que a transição de uma lógica baseada em distâncias euclidianas para uma baseada em **geometria vetorial** foi determinante para a robustez do sistema. O cálculo de ângulos articulares tornou o controlo invariante à escala, permitindo que o sistema funcione corretamente e independentemente da distância do utilizador à câmara. Adicionalmente, a implementação de heurísticas dinâmicas — como a troca de eixos para o cálculo da rotação do pulso e a máquina de estados para o controlo da base — permitiu contornar as limitações de um sistema de câmara monocular (2D), inferindo comportamentos tridimensionais complexos.

Em suma, o *software* desenvolvido não se limita a detetar a mão, mas atua como um intérprete de intenções, convertendo dados visuais brutos em comandos determinísticos prontos a serem transmitidos ao controlador de baixo nível (Arduino).

5.3 Validação Funcional (Mimetização)

O sistema atingiu o seu objetivo primário: a tradução de gestos humanos em movimentos robóticos. Conforme demonstrado na **Figura X** (registo vídeo dos testes), o braço robótico foi capaz de replicar a postura do operador em tempo real.

Os resultados da visão computacional indicaram uma taxa de atualização estável (aprox. 20 FPS no Raspberry Pi), permitindo uma latência "movimento-ação" suficientemente baixa para criar uma sensação de controlo direto. Os algoritmos vetoriais de deteção dos dedos mostraram-se robustos, acionando os servomotores corretamente mesmo com variações na iluminação ambiente.

Figura 22: Sequência de teste funcional em tempo real.

5.4 Desvios ao Projeto Inicial

Embora a funcionalidade principal tenha sido assegurada, o protótipo final apresenta limitações face à especificação idealizada:

- **Graus de Liberdade (DoF):** Não foi concretizada a implementação do servomotor adicional responsável pela inclinação frontal do ombro (*Forward Pitch*). Desta forma, a cinemática do braço ficou restrita à rotação da base, elevação lateral do ombro, rotação do pulso e actuação dos dedos.
- **Estabilidade da Instalação:** A fixação da base robótica à bancada de testes foi realizada através de métodos provisórios (adesivos temporários), o que resultou numa menor estabilidade face a movimentos de inércia elevada. Adicionalmente, a câmara de visão não possui um suporte solidário à estrutura do robô, tornando o sistema suscetível a desalinhamentos caso ocorram choques acidentais na mesa de trabalho, exigindo recalibrações da posição do operador.

Figura 23: Mecanismo de acoplamento da base superior.

5.5 Lista de Material Final

Nome	Referência/Especificação	Quantidade	Preço
Molas	3/16″x1-3/4 or 4.8mm x 44.5mm	5 molas (total)	7.49€
Tubos	Teflon pipes ID1.5MM X OD2.5MM	1595mm	2€
Servo motores (dedos)	MG996R	6	8.90€
Servo motores (braço)	DS5160	1	emStock
Motor de passo (base)	17HS4401S	1	emStock
Tendões	braided fishing line 0.8mm 200LB	5x50cm	8.12
Cabo Eletronico		16x40cm	em stock
Ecoflex	RTV silicone Ecoflex™ 00-10	1	35.00€
Câmera	Raspberry Pi Module 3	1	emStock
Raspberry PI 5	Raspberry PI 5	1	emStock
Arduino MEGA	Arduino MEGA	1	emStock
Arduino Uno	Arduino Uno	1	emStock
CNC Shield	CNC Shield V3	1	emStock
Manga Termo Retráctil		1.2m ou 800pz	3.51€
Íman	Ø 2.5 mm, altura 1 mm	5	4.6
Fluxo RMA	096-0207	1	1.99
Fita em poliamida	095-0793	1	5.03
Cabo Unifilar			4.49€

Tabela 4: Lista de Material

"Numa primeira abordagem, privilegiou-se a adesão à lista de materiais especificada no guia de referência [2]. No entanto, constatou-se que diversos componentes se encontravam descontinuados comercialmente e que as referências originais (*links*) estavam obsoletas.

Face a esta limitação, procedeu-se à elaboração de uma nova lista de materiais [Tabela 3], ajustada aos requisitos específicos do projeto. Relativamente aos componentes não disponíveis no *stock* da instituição, foi realizado um estudo de mercado com o objetivo

6 Conclusões e Trabalho Futuro

A realização deste projeto culminou no desenvolvimento de um sistema robótico antropomórfico funcional, capaz de validar a integração complexa entre fabrico aditivo, eletrónica de controlo distribuída e visão computacional. O protótipo final atingiu o objetivo primordial de mimetização, demonstrando que é possível criar interfaces humano-máquina intuitivas e de baixo custo utilizando tecnologias acessíveis como o Raspberry Pi e o MediaPipe.

No entanto, a análise crítica dos resultados experimentais permitiu identificar áreas onde a implementação atual, embora funcional, pode evoluir para um sistema de maior fidelidade e robustez.

6.1 Propostas de Trabalho Futuro

Com o intuito de aproximar o protótipo de uma aplicação profissional de telepresença, sugerem-se as seguintes linhas de investigação e desenvolvimento:

1. Implementação de Controlo Proporcional (Mimetização Analógica) A limitação mais significativa do *software* atual reside na natureza discreta (binária ou por estados) dos movimentos. O sistema deteta apenas se um dedo está "Aberto" ou "Fechado", ou se o braço está na "Esquerda" ou "Direita".

- **Melhoria Proposta:** Evoluir os algoritmos de visão para um **mapeamento proporcional direto**. Em vez de comparar ângulos com um limiar fixo, o sistema deverá converter o ângulo de flexão do dedo do operador (ex: 45º) diretamente num valor PWM correspondente para o servo (ex: servo.write(90)). Esta lógica deve ser estendida à rotação da base e à flexão do cotovelo, permitindo que o robô copie a velocidade e a posição exata do operador em tempo real, e não apenas salte para posições pré-definidas.

2. Evolução da Estrutura Mecânica e Suporte da Câmara A estabilidade física do sistema carece de otimização. A fixação provisória da base e a independência da câmara revelaram-se pontos fracos durante os testes.

- **Melhoria Proposta:** Redesenhar a base do robô para incluir um **suporte integrado e ajustável para a câmara**. Isto garantiria que o referencial de visão fosse sempre solidário ao robô, eliminando a necessidade de recalibração caso o sistema seja movido. Adicionalmente, deve ser desenvolvido um sistema de fixação à bancada por grampos (em vez de adesivos) para suportar a inércia de movimentos rápidos.

3. Expansão dos Graus de Liberdade (Cinemática Completa) O modelo atual carece do movimento de inclinação frontal do ombro (*Forward Pitch*), o que restringe a área de trabalho manipulável.

- **Melhoria Proposta:** Concretizar a instalação do servomotor em falta na articulação do ombro e integrar o respetivo controlo na cinemática inversa. Isto permitiria ao braço realizar movimentos de alcance para a frente e para trás, essenciais para agarrar objetos dispostos em diferentes profundidades na mesa.

4. Feedback Sensorial (Malha Fechada) Atualmente, o operador não recebe qualquer retorno físico sobre o sucesso da preensão.

- **Melhoria Proposta:** Instalar sensores de pressão (FSR) nas pontas dos dedos e motores de vibração numa luva háptica para o operador. Desta forma, quando o robô tocasse num objeto, o utilizador sentiria uma vibração correspondente, fechando o ciclo de controlo e permitindo a manipulação de objetos frágeis com maior segurança.

Em suma, este projeto estabeleceu uma base sólida para o desenvolvimento de robótica assistida, provando que a visão computacional moderna pode substituir controladores físicos complexos, tornando a tecnologia mais natural e acessível ao ser humano.

7 Referências

[1] H. Abrantes, C. Rodrigues e R. Maria, "Proposta de Projeto Integrado: Braço Robótico Controlado por Visão", Escola Náutica Infante D. Henrique, Paço de Arcos, 2025.

[2] G. Langevin, "InMoov Hand I2", InMoov Open-Source 3D Printed Life-Size Robot. [Online]. Disponível: https://inmoov.fr/hand-and-forarm. [Acedido: 28-Nov-2025].

[3] G. Langevin, "InMoov Hand and Forearm", InMoov Open-Source 3D Printed Life-Size Robot. [Online]. Disponível: https://inmoov.fr/hand-i2. [Acedido: 28-Nov-2025].

[4] Dsservo, "DS5160 Digital Servo Datasheet", 2023. [Online]. Disponível: https://m.media-amazon.com/images/I/81EFGw8qkhL.pdf.

[5] Handson Technology, "17HS4401S Bipolar Stepper Motor Datasheet". [Online]. Disponível: https://www.alldatasheet.com/datasheet-pdf/view/1492290/ETC/17HS4401S.html.

[6] Arduino Forum, "Using CNC Shield V3 directly with Arduino IDE", 2022. [Online]. Disponível: https://forum.arduino.cc/t/using-cnc-shield-v3-directly-with-arduino-ide/1022999/2.

[7] Google Developers, "MediaPipe Hands," MediaPipe Solutions. [Online]. Disponível: https://ai.google.dev/edge/mediapipe/solutions/vision/pose_landmarker.

[8] Google Developers, "MediaPipe Pose," MediaPipe Solutions. [Online]. Disponível: https://developers.google.com/mediapipe/solutions/vision/pose_landmarker.

[9] OpenCV Team, "OpenCV: Open-Source Computer Vision Library," 2024. [Online]. Disponível: https://opencv.org/.

[10] Raspberry Pi Foundation, "Raspberry Pi 5 – Documentation and Hardware Specifications," 2024. [Online]. Disponível: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html.

[11] Arduino, "Arduino Mega 2560 Rev3 Datasheet & Pinout". [Online]. Disponível: https://docs.arduino.cc/hardware/mega-2560.

[12] Protoneer, "Arduino CNC Shield V3.00 – Schematics and Pinout," 2022. [Online]. Disponível: https://www.zyltech.com/arduino-cnc-shield-instructions/.

[13] Grupo RIA-G7, "Repositório de Código Fonte: Humanoid Robotic Arm Vision Control," GitHub, 2025. [Online]. Disponível: https://github.com/Bolofofopt/HumanoidRoboticArmVision.

[14] Grupo RIA-G7, "Esquema Elétrico Completo e Interativo - Braço Robótico," *Cirkit Designer Project*, 2025. [Online]. Disponível: https://app.cirkitdesigner.com/project/.