UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO
GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

# SOLUÇÃO AUTOMÁTICA DO CUBO DE RUBIK USANDO VISÃO COMPUTACIONAL

Gustavo Vinícius Taveira Lima

BRASIL

2024

UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO

# TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO NO REPOSITÓRIO INSTITUCIONAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio do Repositório Institucional (RI/UFG), regulamentado pela Resolução CEPEC no 1240/2014, sem ressarcimento dos direitos autorais, de acordo com a Lei no 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo dos Trabalhos de Conclusão dos Cursos de Graduação disponibilizado no RI/UFG é de responsabilidade exclusiva dos autores. Ao encaminhar(em) o produto final, o(s) autor(a)(es)(as) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

## 1. Identificação do Trabalho de Conclusão de Curso de Graduação (TCCG)

Nome completo do autor: Gustavo Vinícius Taveira Lima

Título do trabalho: Solução Automática do Cubo de Rubik usando Visão Computacional

## 2. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador) Concorda com a liberação total do documento [ X ] SIM [ ] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante: a) consulta ao(à)(s) autor(a)(es)(as) e ao(à) orientador(a); b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo do TCCG. O documento não será disponibilizado durante o período de embargo.

**Casos de embargo:**

- Solicitação de registro de patente;

- Submissão de artigo em revista científica;

- Publicação como capítulo de livro.

**Obs.: Este termo deve ser assinado no SEI pelo orientador e pelo autor.**

Documento assinado eletronicamente por **Adriano Cesar Santana**, **Professor do Magistério Superior**, em 12/12/2024, às 14:52, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Gustavo Vinicius Taveira Lima**, **Discente**, em 12/12/2024, às 15:00, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Gustavo Vinícius Taveira Lima

# SOLUÇÃO AUTOMÁTICA DO CUBO DE RUBIK USANDO VISÃO COMPUTACIONAL

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia Elétrica, Mecânica e de Computação da Universidade Federal de Goiás como requisito parcial à obtenção do título de Bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Adriano César Santana

GOIÂNIA

2024

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Lima, Gustavo Vinícius Taveira
SOLUÇÃO AUTOMÁTICA DO CUBO DE RUBIK USANDO VISÃO COMPUTACIONAL [manuscrito] / Gustavo Vinícius Taveira Lima. - 2024.
f.: il.

Orientador: Prof. Dr. Adriano César Santana.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Goiás, Escola de Engenharia Elétrica, Mecânica e de Computação (EMC), Engenharia da Computação, Goiânia, 2024.
Bibliografia. Apêndice.
Inclui siglas, fotografias, tabelas, lista de figuras, lista de tabelas.

1. Arduino. 2. Cubo de Rubik. 3. ESP32-CAM. 4. Visão Computacional. 5. Python. I. Santana, Adriano César, orient. II. Título.

CDU 621

UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO

# ATA DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO

Ao(s) doze dia(s) do mês de dezembro do ano de 2024 iniciou-se a sessão pública de defesa do Trabalho de Conclusão de Curso (TCC) intitulado "**Solução Automática do Cubo de Rubik usando Visão Computacional**", de autoria de **Gustavo Vinícius Taveira Lima**, do curso de Engenharia de Computação, do(a) EMC da UFG. Os trabalhos foram instalados pelo(a) Prof. Dr. Adriano César Santana com a participação dos demais membros da Banca Examinadora: Prof. Dr. Marcelo Stehling de Castro e Prof. Dr. José Wilson Lima Nerys. Após a apresentação, a banca examinadora realizou a arguição do(a) estudante. Posteriormente, de forma reservada, a Banca Examinadora atribuiu a nota final de (**10,0**) , tendo sido o TCC considerado **aprovado**.

Proclamados os resultados, os trabalhos foram encerrados e, para constar, lavrou-se a presente ata que segue assinada pelos Membros da Banca Examinadora.

Documento assinado eletronicamente por **Adriano Cesar Santana**, **Professor do Magistério Superior**, em 12/12/2024, às 14:51, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Jose Wilson Lima Nerys**, **Professor do Magistério Superior**, em 12/12/2024, às 14:52, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Marcelo Stehling De Castro**, **Professor do Magistério Superior**, em 12/12/2024, às 14:52, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **5029115** e o código CRC **C4D84274**.

**Referência:** Processo nº 23070.044791/2024-37 SEI nº 5029115

UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO

# DECLARAÇÃO

FREQUÊNCIA A SER PREENCHIDA PELO ORIENTADOR(A)

## FREQUÊNCIA DOS DISCENTES EM PFC

| Nome do Discente | Frequência (%) |
|---|---|
| Gustavo Vinícius Taveira Lima | 100 |

Conforme artigo Art. 29 da resolução 02/2023 da EMC/UFG:

"A aprovação na disciplina Projeto Final de Curso fica condicionada à nota final (NF) e frequências mínimas serem maiores ou iguais às estabelecidas no RGCG na UFG no semestre letivo em curso."

Documento assinado eletronicamente por **Adriano Cesar Santana**, **Professor do Magistério Superior**, em 12/12/2024, às 14:52, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **5029156** e o código CRC **6EB92484**.

**Referência:** Processo nº 23070.044791/2024-37 SEI nº 5029156

*Dedico este trabalho à minha mãe, Aparecida Taveira de Souza, por estar sempre ao meu lado e ser meu porto seguro ao longo de toda a minha vida e jornada acadêmica, oferecendo apoio, conselhos e grande sabedoria. Aos meus familiares, pelo acolhimento, incentivo e apoio constantes, que me permitiram chegar até aqui.*

# Agradecimentos

Agradeço primeiramente a Deus e à minha família, que formam a base sobre a qual me apoiei para ter saúde, força e incentivo ao longo de todos os meus anos de estudo.

Agradeço ao meu orientador, Prof. Dr. Adriano César Santana, pelo apoio e atenção ímpares, além dos valiosos esclarecimentos fornecidos em cada etapa do desenvolvimento deste trabalho.

Por fim, agradeço a toda a equipe da Escola de Engenharia Elétrica, Mecânica e de Computação da UFG e a todos os professores com quem tive a honra de aprender, por proporcionarem um ambiente fértil para a construção do conhecimento e um ensino de excelência, que transformou minha realidade. Digo que essa experiência foi muito mais do que apenas aprendizado, foi uma verdadeira lição de vida. Meus sinceros agradecimentos.

# Resumo

T. Lima, Gustavo Vinícius. **Solução Automática do Cubo de Rubik usando Visão Computacional.** 2024. Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Computação) - Universidade Federal de Goiás, Goiânia, 2024.

O presente trabalho tem como objetivo desenvolver um sistema automatizado para resolver o cubo de Rubik utilizando visão computacional. Para a execução deste projeto, foi realizada uma pesquisa teórica preliminar, seguida pela aquisição dos componentes necessários e pela montagem do protótipo, que utiliza os microcontroladores Arduino Uno e ESP32-CAM. O algoritmo de visão computacional foi implementado em Python, utilizando a biblioteca OpenCV, enquanto a programação dos microcontroladores foi realizada em C++. Para a resolução do cubo, foi adotado o algoritmo de Kociemba. O sistema desenvolvido é capaz de reconhecer automaticamente as cores das faces do cubo e executar os movimentos necessários para sua resolução, alcançando uma solução eficiente, com tempo de resolução aproximado de quatro minutos, além de oferecer uma interface intuitiva e de fácil utilização.

**Palavras-chave**: Arduino. Cubo de Rubik. ESP32-CAM. Visão Computacional. Python.

# Abstract

T. Lima, Gustavo Vinícius. **Automatic Solving of the Rubik's Cube using Computer Vision.** 2024. Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Computação) - Universidade Federal de Goiás. Goiânia, 2024.

The present work aims to develop an automated system to solve the Rubik's Cube using computer vision. To execute this project, preliminary theoretical research was conducted, followed by the acquisition of the necessary components and the assembly of the prototype, which uses the Arduino Uno and ESP32-CAM microcontrollers. The computer vision algorithm was implemented in Python, using the OpenCV library, while the microcontroller programming was carried out in C++. To solve the cube, the Kociemba algorithm was adopted. The developed system is capable of automatically recognizing the colors of the cube's faces and executing the necessary moves to solve it, achieving an efficient solution, with a resolution time of approximately four minutes, in addition to offering an intuitive and easy-to-use interface.

**Keywords**: Arduino. Computer Vision. ESP32-CAM. Python. Rubik's Cube.

# Lista de ilustrações

# Lista de tabelas

# Sumário

# 1 Introdução

A visão computacional é um campo de estudo que visa conceder às máquinas a capacidade de extrair informações de imagens, assim como a visão humana. Utilizando um conjunto de algoritmos que recebem como entrada uma ou mais imagens, a visão computacional, conforme discutido por (SZELISKI, 2022), busca entender o mundo tridimensional representado no plano bidimensional da imagem, permitindo a extração de informações relevantes para apoiar a tomada de decisões. Dessa forma, é possível aplicar técnicas e algoritmos de visão computacional para possibilitar o reconhecimento automático das cores do cubo de Rubik, um quebra-cabeça criado em 1974 pelo professor de arquitetura húngaro Ernő Rubik, que, até os dias atuais, conta com uma comunidade bastante forte e ativa.

Este trabalho visa o aprimoramento de um projeto já existente, no qual o método de entrada de dados da abordagem anterior consistia na digitação manual das cores, um processo demorado e propenso a erros, já que era necessário informar todas as 54 cores presentes nas seis faces do cubo. Como alternativa, este trabalho se propõe a obter um mecanismo que utiliza a visão computacional como método de entrada de dados, automatizando a identificação das cores.

Para alcançar esse resultado, o algoritmo utilizado foi dividido em três etapas: inicialmente, detecta-se a face ou *grid* do cubo na cena da imagem. Em seguida, realiza-se o rastreamento conforme o cubo se movimenta, evitando que a detecção precise ser reiniciada a cada novo *frame* do vídeo. Por fim, o processamento das cores é a última etapa, na qual as cores de cada face apresentada à câmera são identificadas e processadas em conjunto, sendo então convertidas em uma representação matricial, possibilitando, posteriormente, a resolução computacional do cubo por meio do algoritmo Kociemba.

Ainda, por meio deste trabalho, busca-se promover soluções baseadas no uso de visão computacional e demonstrar, por meio de um exemplo prático, a aplicação dessa poderosa funcionalidade, que pode ser implementada em máquinas para torná-las mais inteligentes, além de possibilitar a criação de soluções inovadoras.

## 1.1 Objetivo geral

O objetivo deste trabalho é obter um mecanismo que soluciona automaticamente o cubo de Rubik, usando visão computacional como método de entrada de dados, permitindo que o usuário informe o estado do cubo de forma mais rápida e prática, apenas apresentando as faces à câmera do sistema.

## 1.2 Objetivos específicos

Os objetivos específicos deste trabalho são:

- Obter um protótipo de robô que realiza a resolução do cubo de Rubik corretamente.
- Desenvolver um programa em Python, com o uso da biblioteca OpenCV, que utiliza algoritmos de visão computacional para a correta identificação de todas as cores presentes nas seis faces do cubo de Rubik.
- Promover o uso de visão computacional em sistemas de automação, demonstrando sua viabilidade e vantagens com um exemplo prático.

# 2 Pesquisa bibliográfica

Inicialmente, foi realizada uma pesquisa bibliográfica para compreender todos os aspectos teóricos necessários à execução correta do projeto. O primeiro tema abordado foi o cubo de Rubik, suas características e métodos de resolução. Essa etapa é importante, pois define o estado-alvo em que o cubo é considerado resolvido, os movimentos possíveis e os algoritmos disponíveis para resolução, informações essenciais para a programação de uma máquina capaz de resolver o cubo.

Em seguida, foram pesquisados trabalhos existentes relacionados aos objetivos propostos por este trabalho. Três estudos foram analisados quanto às suas vantagens e desvantagens. Como base deste projeto, foi escolhido um algoritmo que utiliza técnicas de detecção baseadas em bordas, operando sobre os valores de intensidade dos pixels no plano da imagem. Através de testes práticos realizados no protótipo desenvolvido, essa abordagem demonstrou ser suficientemente adequada para a correta detecção do cubo de Rubik na imagem.

Na etapa seguinte, foram analisadas as tecnologias de microcontroladores. O Arduino Uno foi escolhido para o controle dos atuadores, enquanto o ESP32-CAM foi selecionado para realizar a aquisição de imagens, que serão transmitidas ao programa de visão computacional. Essas tecnologias foram escolhidas devido à familiaridade do autor com ambas e ao custo-benefício, que permite desenvolver projetos de automação com menor custo e alta qualidade. Para a execução do projeto, foram utilizados três programas: um em Python, para o algoritmo de visão computacional, outro em C++ para o Arduino Uno, programado com os movimentos necessários para resolver o cubo, e por fim, o plugin Arduino-ESP32, conforme (Espressif Systems, 2024), para possibilitar a comunicação do ESP32-CAM com os demais programas, por meio da transmissão das imagens capturadas.

Para a comunicação entre os programas, foram utilizadas conexões físicas por Wi-Fi e USB (*Universal Serial Bus*). A conexão Wi-Fi foi estabelecida via protocolo HTTP (*HyperText Transfer Protocol*), no qual o ESP32-CAM atua como emissor das imagens, e o programa de visão computacional em Python, como receptor, acessando as imagens por meio de uma URL (Uniform Resource Locator) para posterior processamento. Após obter as cores de cada face do cubo, essas informações são enviadas ao algoritmo de resolução de Kociemba. A solução, representada por uma cadeia de caracteres, é então enviada via USB, através do protocolo de comunicação serial, ao computador executando a aplicação Python. A seguir, são detalhados os principais aspectos teóricos necessários para o desenvolvimento deste trabalho.

## 2.1 Cubo de Rubik

O cubo de Rubik, também conhecido popularmente como cubo mágico, é um quebra-cabeça mundialmente conhecido, desenvolvido em 1974 pelo professor de arquitetura húngaro Ernő Rubik. O cubo original 3x3x3 possui seis faces, sendo que cada uma possui uma cor distinta quando se encontra em seu estado inicial, denominado estado resolvido. As cores possíveis são: verde, vermelho, amarelo, azul, branco e laranja.

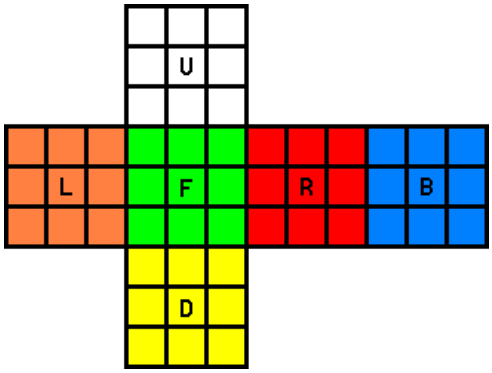
Figura 1 – Estado inicial do cubo de Rubik: cada face possui uma cor distinta.


Fonte: Autor.

O cubo é composto por nove pequenas subdivisões em cada face, organizadas em um formato de *grid* (isto é, uma disposição em grade), onde cada subdivisão pode assumir uma cor diferente, dependendo do estado do cubo. Essas subdivisões são chamadas de peças de quina, peças de meio e peça central (ou centro). Algumas características dessas peças são: as quinas sempre possuem três adesivos de cores vizinhas distintas, enquanto os meios têm apenas duas cores vizinhas. Além disso, é importante observar que as cores de faces opostas do cubo nunca estão juntas em uma mesma peça de quina ou meio. Por último, temos a peça central ou centro, que possui uma característica fundamental: sua cor nunca se altera ao realizar os movimentos do cubo. Assim, essas peças são fixas e determinam a cor de cada face do cubo. Os movimentos do cubo podem ser realizados no sentido horário ou anti-horário, sendo indicados pela ausência ou presença de aspas simples após a notação do movimento, respectivamente. Cada face movimentada recebe uma letra como indicador do tipo de movimento realizado. Mantendo o cubo com uma face voltada para o jogador, esta representa a face frontal (F - *FRONT*). As faces laterais são as faces da esquerda (L - *LEFT*) e da direita (R - *RIGHT*), e as faces verticais são as faces de cima (U - *UPPER*) e de baixo (D - *DOWN*). A face oposta à face frontal é a face traseira (B - *BACK*). Além dos movimentos simples, existem os movimentos duplos, onde a cada movimento é acrescentado o número ’2’, indicando que o movimento deve ser executado duas vezes. Para a resolução do cubo, foi adotado o padrão de cores ocidental, por se tratar da forma de organização de cores mais comum para o Cubo de Rubik (Speedsolving Wiki, 2024), e

que está em conformidade com as regras da *World Cube Association* (World Cube Association, 2024).

Figura 2 – Esquema de cores ocidental do cubo de Rubik.




Fonte: (Speedsolving Wiki, 2024).

O desafio do cubo de Rubik começa quando realizamos diversos movimentos aleatórios no cubo, de forma a alterar o seu estado resolvido para algum outro estado possível. Em seguida, devemos tentar retorná-lo ao estado resolvido. A quantidade de estados possíveis para o cubo 3x3x3 é superior a 43 quintilhões (MIT, 2011), e, para solucioná-lo adequadamente, é necessário utilizar um algoritmo de resolução. Neste trabalho, será utilizado o algoritmo Kociemba.

## 2.2 Kociemba

O algoritmo Kociemba é um método computacional para a resolução do cubo de Rubik. Este método se apoia nos conceitos da teoria de grupos da álgebra abstrata, onde um grupo é definido como um conjunto de sequências de movimentos possíveis para o cubo, a partir de seu estado resolvido. Cada sequência representa um estado possível do cubo. Este algoritmo, também conhecido como algoritmo de duas fases, pois alcança a solução do cubo em duas etapas. A primeira etapa consiste em executar movimentos no cubo, partindo de um estado aleatório, para levá-lo a um grupo de estados que possui restrições quanto aos movimentos possíveis. O objetivo desta etapa é reduzir a complexidade para se alcançar a solução, restringindo os estados do cubo a um número limitado de movimentos. Uma vez nesse grupo, a segunda etapa permite que o algoritmo busque o estado resolvido em um conjunto reduzido de possibilidades, devido a restrições de movimentos impostas.

Os movimentos possíveis em cada grupo são listados abaixo:

- G0: é o grupo que permite todos os movimentos possíveis e não possui nenhuma restrição ( U, U', D, D', R, R', L, L' F, F', B, B'). Cada estado é representado por uma sequência de movimentos a partir do estado resolvido.

- G1: é um subgrupo de G0 que possui restrições em seus movimentos possíveis (U, U', D, D', R2, L2, F2, B2). Os estados desse subgrupo têm como principal característica levar ao estado resolvido do cubo, utilizando apenas sequências de movimentos pertencentes a G1.

A fase 1 inicia-se a partir de um estado qualquer do espaço de estados total do cubo de Rubik representado por G0, e leva a um estado definido por G1. Na fase 2, a partir do estado de G1, é encontrada uma sequência de movimentos que leva até ao estado resolvido do cubo. Pode-se notar que essa abordagem utiliza de um princípio de engenharia muito importante, que consiste em tornar a solução de problemas complexos em uma combinação de problemas menores, que podem ser mais facilmente gerenciados, tornando-os mais simples de serem resolvidos.

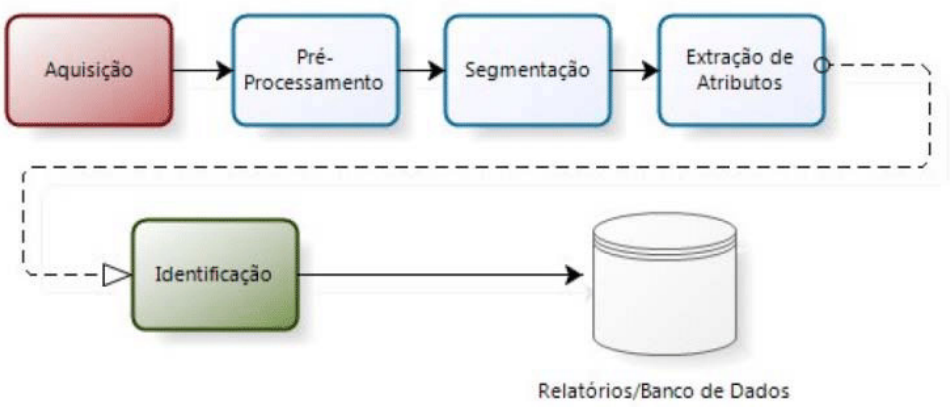
Uma característica interessante desses grupos é que, a partir de G0, são necessários no máximo 12 movimentos para se atingir G1, e, partindo de G1, são necessários no máximo 18 movimentos para atingir o estado resolvido (ROKICKI, 2008). Portanto, o algoritmo de Kociemba resolve o cubo de Rubik para qualquer posição em no máximo 30 movimentos.

## 2.3 Visão computacional

A visão computacional é um campo de estudo que tem como objetivo conceder às máquinas a funcionalidade de visão, de forma similar à capacidade da visão humana, ou até mesmo, em algumas situações, superando-a. Segundo (NAYAR, 2024) o objetivo da visão computacional é construir máquinas que consigam ver. Para atingir esse objetivo, o principal objeto de estudo são as imagens. Uma imagem pode ser modelada como uma função bidimensional contínua $f(x,y)$ onde cada posição é representada por um nível de intensidade de luz. Ao realizar combinações dessas intensidades de luz em diferentes bandas espectrais na faixa do visível, obtemos as cores. Um sistema de visão computacional deve ser capaz de extrair informações de imagens, que podem ser usadas posteriormente para apoio à tomada de decisões. A figura 3 ilustra uma sequência básica de etapas que são normalmente realizadas por um sistema de visão computacional.

A aquisição de imagens é a primeira etapa e consiste em obter imagens em formato digital para posterior processamento. As imagens podem ser adquiridas por meio de câmeras que se baseiam fundamentalmente no modelo da câmera escura, que consiste em uma caixa com uma pequena abertura em uma de suas faces. Por essa abertura, passam os raios de luz do ambiente externo, que são projetados de forma a gerar uma imagem na face interna da caixa. Essa abordagem, porém, apresenta uma grande desvantagem de capturar pouca luz, gerando imagens de baixa qualidade e tempos prolongados de exposição. Por isso, as câmeras modernas utilizam lentes como mecanismo da etapa de aquisição de imagens, permitindo capturar imagens com tempos curtos de exposição e com alta qualidade. Algumas câmeras oferecem a possibilidade de calibração, o que proporciona ao usuário maior controle sobre como as imagens devem ser

Figura 3 – Etapas básicas de um sistema de visão computacional.




Fonte: (NETO et al., 2015).

capturadas. É possível configurar parâmetros como a abertura da lente, que controla a quantidade de luz e a profundidade de campo, a sensibilidade à luz do sensor de imagem e o tempo de exposição.

Após a etapa de aquisição de imagens, a imagem é convertida em sua representação digital por meio de sensores internos da câmera. Atualmente, as tecnologias CCD (*Charge Coupled Device*) e CMOS (*Complementary Metal Oxide Semiconductor*) são as mais utilizadas como sensores para as câmeras digitais modernas. Esses sensores transformam a luz capturada em sinais elétricos, que são posteriormente convertidos para uma representação digital da imagem capturada. Uma imagem digital pode ser representada como uma matriz de pixels. Um pixel, ou *picture element*, é o menor elemento de uma imagem digital. Trata-se de um número de 8 bits que possui uma faixa de intensidades luminosas variando de 0 a 255. Uma imagem RGB (*Red, Green, Blue*) possui três canais de cores, cada um variando de 0 a 255, de modo que essas três representações combinadas formam o espaço de cores RGB. Essa abordagem permite a criação de imagens coloridas em sua representação digital.

A etapa de pré-processamento envolve a melhoria da qualidade da imagem para torná-la mais adequada às etapas posteriores. Algumas das principais técnicas dessa etapa incluem o processamento de histograma, que consiste em corrigir brilho e contraste na imagem, e a redução de ruídos por meio do uso de filtros de convolução. Esses filtros são aplicados iterativamente sobre cada pixel da imagem, promovendo efeitos como desfoque ou aguçamento da imagem, o que permite reduzir ou aumentar variações de intensidade dos pixels, respectivamente. Dessa forma, é possível controlar o nível de detalhes da imagem antes de sua entrega às etapas seguintes do sistema de visão computacional.

A próxima etapa após o pré-processamento é a segmentação, que consiste em dividir a imagem em regiões ou objetos de interesse, separando grupos de pixels com base nas características de intensidade de brilho e cor. Algumas das técnicas utilizadas nessa etapa

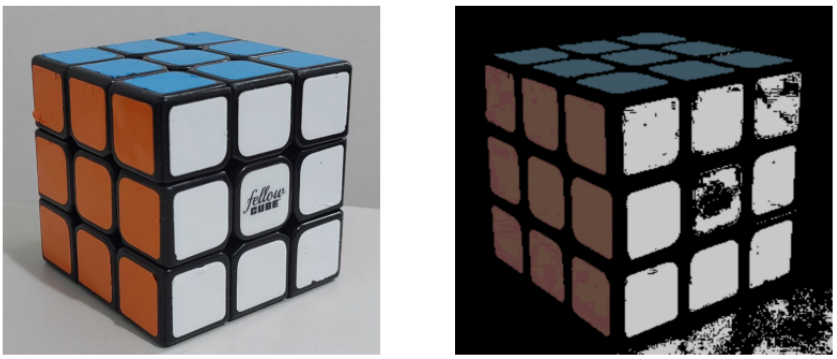
Figura 4 – Exemplo de filtro de suavização gaussiano. O filtro itera sobre todos os pixels da imagem e gera um efeito de desfoque.

$$\frac{1}{16} \times \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Fonte: (GONZALEZ; WOODS, 2010).

incluem a limiarização e a segmentação baseada em cores. A limiarização é aplicada a uma imagem em escala de cinza, transformando-a em uma imagem binária, também chamada de máscara, com base em um valor limite predefinido. Pixels com intensidade abaixo desse valor são convertidos para pixels escuros (nível 0), enquanto pixels acima desse valor tornam-se pixels claros (nível 255). Essa máscara pode ser aplicada à imagem original, para separar as regiões de interesse do restante da imagem. Por outro lado, a segmentação de cores consiste em estabelecer intervalos de valores de cor para identificar pixels com cores semelhantes e aplicar a limiarização para cada intervalo de cores definido. Assim, são obtidas diversas máscaras que podem ser aplicadas na imagem original e podemos, assim, segmentar regiões com base nos valores obtidos por essas máscaras. Essa abordagem, entretanto, possui uma desvantagem significativa: se as condições de iluminação variarem de intensidade ou o plano de fundo da imagem for muito complexo e detalhado, a detecção pode se tornar difícil e vulnerável a falhas, comprometendo a robustez da solução. Em um ambiente industrial, onde é possível controlar os parâmetros de

Figura 5 – Técnica de segmentação baseada em cores aplicada ao cubo de Rubik.



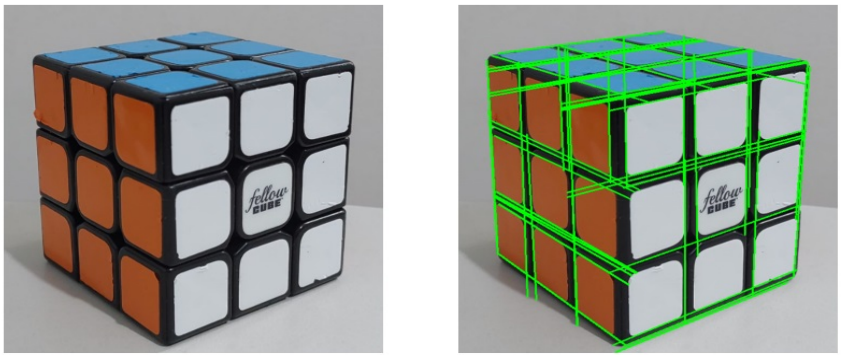
Fonte: Autor.

iluminação e a complexidade do plano de fundo da imagem, essa abordagem pode se tornar

muito satisfatória devido à simplicidade de implementação e aos ótimos resultados que podem ser alcançados.

Após a segmentação, a etapa seguinte é a extração de atributos ou características da imagem. Nessa etapa, informações como bordas, cantos, cores, retas, texturas e outras propriedades relevantes da imagem são obtidas e utilizadas para a análise dos elementos da imagem. Isso possibilita que etapas futuras, como o reconhecimento, a detecção e o rastreamento de objetos, possam utilizar essas informações para concluir suas tarefas. Aqui há uma ampla gama de técnicas que podem ser utilizadas para extração de características, como, por exemplo, o filtro laplaciano, que consiste em um filtro de convolução, que tem como objetivo encontrar bordas em uma imagem. As bordas são variações na intensidade dos pixels, nas duas dimensões, de altura e largura, que constituem a imagem. Outra técnica útil que pode ser aplicada após a detecção de bordas é a transformada de Hough, que tem como objetivo principal encontrar segmentos de reta na imagem. Essa técnica é particularmente útil para um algoritmo que reconheça o cubo de Rubik, pois, devido a sua simetria regular, a detecção de segmentos de reta evidencia a presença do *grid* do cubo no plano da imagem.

Figura 6 – Extração de segmentos de reta utilizando a transformada de Hough.



Fonte: Autor.

Por fim, as etapas de identificação e armazenamento em banco de dados são as últimas etapas de um sistema de visão computacional. A identificação consiste em utilizar as informações obtidas na etapa anterior, de extração de atributos, para tomar decisões sobre os elementos da imagem, detectar e reconhecer objetos, bem como possibilitar o rastreamento em ambientes dinâmicos de imagem, como vídeos e gravações em tempo real. A visualização, o armazenamento e a eventual transmissão das informações extraídas formam a etapa final do sistema de visão computacional. Aqui, relatórios podem ser gerados, os dados obtidos podem abastecer sistemas de informação para apoiar a tomada de decisões e também podem ser transmitidos por redes de computadores ou outras tecnologias de comunicação, permitindo o compartilhamento das informações obtidas com partes interessadas e outros sistemas externos. Um exemplo de algoritmo utilizado para o rastreamento de objetos em tempo real é o algoritmo de fluxo óptico Lucas-Kanade. Esse algoritmo recebe como entrada um conjunto de pontos a serem rastreados a

partir de uma imagem, e, em seguida, compara essa imagem com a próxima em uma sequência de imagens de uma cena. Para cada ponto, a posição para a qual ele se moveu em relação a imagem anterior é identificada na imagem atual, e, então, é calculado o vetor deslocamento $\vec{v}$ que vai da posição original até a nova posição detectada. Quanto mais rápido for o movimento na imagem, maior será a distância entre os dois pontos. Assim, pode-se quantificar o fluxo óptico, ou seja, o movimento do objeto, pelo tamanho em módulo do vetor deslocamento $\vec{v}$, calculado para cada ponto passado como entrada, possibilitando, assim, o rastreamento do objeto.

Figura 7 – Rastreamento utilizando o algoritmo de fluxo óptico Lucas-Kanade.

Fonte: (OpenCV, 2024).

## 2.4 Trabalhos anteriores

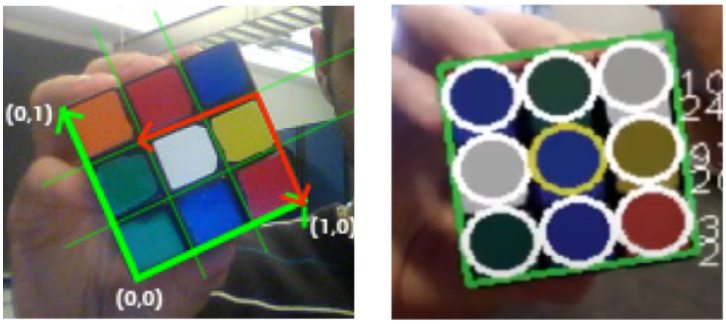
Nesta seção, serão analisados três trabalhos anteriores que foram relevantes para a elaboração deste trabalho. A seguir, serão abordadas as metodologias, tecnologias e limitações associadas a cada solução. Existem diversos projetos similares relacionados à solução automática do cubo de Rubik, mas, com o avanço na compreensão de técnicas de visão computacional, novas possibilidades têm surgido para aprimorar a identificação e reconhecimento das cores do cubo. Esses avanços permitem que a solução tenha maior velocidade e precisão na entrada de dados, além de tornar o processo ainda mais automatizado.

### 2.4.1 Extraindo cores do cubo de Rubik

Esse trabalho foi desenvolvido com o objetivo de obter um algoritmo eficaz para a detecção e localização do cubo de Rubik em um *stream* de vídeo, em tempo real. A abordagem utilizada realiza a extração automática das cores das faces, com mínima intervenção do usuário. O algoritmo desenvolvido foi implementado em Python com auxílio da biblioteca OpenCV, utilizando uma abordagem baseada em bordas que é invariável a diferentes configurações de câmera, e não depende de um esquema de cores específico para o cubo. Por meio da transformada de Hough, o projeto consegue detectar bordas paralelas fortes, que são características específicas

do cubo de Rubik. Com base nas bordas detectadas, o algoritmo utiliza técnicas de geometria computacional (MARK et al., 2008) para encontrar pontos de interseção entre segmentos de reta. Isso possibilita a construção de um sistema de coordenadas, por meio de uma matriz de transformações geométricas, de forma que as posições das cores, nas faces do cubo, possam ser localizadas. É utilizado o algoritmo de rastreamento óptico Lucas-Kanade para suavizar

Figura 8 – Detecção do cubo de Rubik utilizando Python e a biblioteca OpenCV.




Fonte: (KARPATHY, 2009).

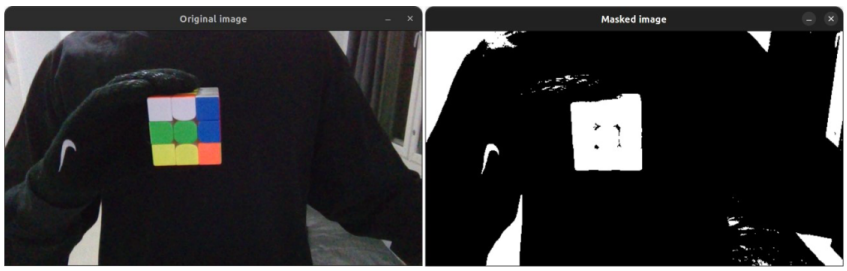
a detecção, de modo que não seja necessário reconhecer o cubo a cada novo *frame* enviado pela câmera, prevenindo o efeito de cintilação ou *flickering*, causado pela latência inerente ao processamento. O algoritmo proporciona uma leitura rápida e confiável do estado do cubo em 5 a 10 segundos (KARPATHY, 2009), superando métodos manuais em velocidade e precisão. Com base nos resultados apresentados, o algoritmo é capaz de reconhecer todas as cores das seis faces do cubo de Rubik, permitindo seu uso por sistemas automatizados que realizam a solução do quebra-cabeça.

### 2.4.2 Segmentação baseada em cores para o cubo de Rubik

Muitas das abordagens iniciais utilizadas para a detecção das cores do cubo de Rubik baseiam-se na segmentação de cores. A presença das diferentes cores em cada face do cubo nos traz a intuição de que essa abordagem pode produzir bons resultados. Nesse trabalho, foi utilizada a abordagem de segmentação baseada em cores utilizando a biblioteca OpenCV e Python. No entanto, essa abordagem é muito sensível a condições de iluminação e à complexidade de elementos presentes no plano de fundo da imagem. Para contornar esses desafios, o projeto impõe uma série de restrições que permitem que o algoritmo consiga realizar a detecção corretamente. A primeira restrição é que o fundo da imagem deve ser o mais escuro possível, permitindo que o cubo seja identificado mais facilmente. Outra restrição é que o cubo precisa estar a uma distância específica da câmera, pois o parâmetro de área de contorno utilizado para a identificação é fixo, e, se o cubo for muito pequeno ou grande demais na imagem, o mesmo não será detectado. Uma terceira restrição é que o cubo precisa estar próximo ao centro da imagem, de forma que fundos complexos nos cantos da imagem possam ser ignorados por tratativa em software, considerando apenas uma pequena região central da imagem como adequada para a detecção. Além disso, o

cubo deve ser apresentado sem estar rotacionado, e essa restrição visa evitar que as cores de cada face sejam detectadas em posições incorretas, levando a erros na detecção.

Figura 9 – Detecção do cubo de Rubik utilizando segmentação.



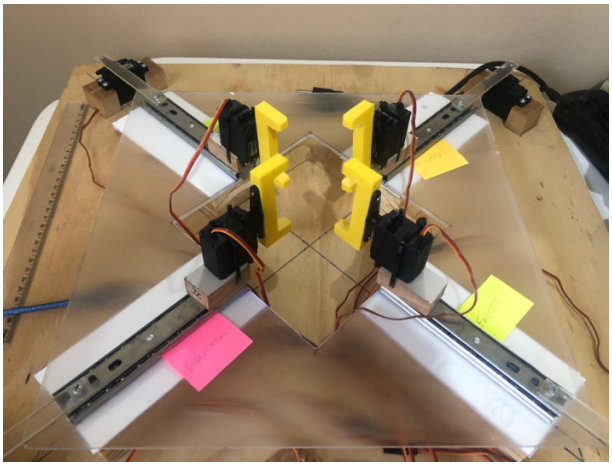

Fonte: (JOKINEN, 2024).

O cubo é resolvido utilizando o algoritmo Kociemba, e com base em orientações informadas em tela, o usuário deve executar os movimentos necessários, até atingir o estado resolvido do cubo de Rubik. Devido às inúmeras restrições mencionadas, é possível perceber que a segmentação baseada em cores pode ser muito bem utilizada em ambientes controlados. Porém, quando não se pode controlar o ambiente de detecção, devem ser consideradas outras abordagens e técnicas de visão computacional, que possam lidar melhor com possíveis restrições, como por exemplo a abordagem baseada em bordas.

### 2.4.3 Mecanismo automatizado para a resolução do cubo de Rubik

Nesse trabalho, foi desenvolvido um protótipo funcional para a resolução automática do cubo de Rubik. Foram utilizados oito servomotores, sendo quatro deles responsáveis pela movimentação horizontal, que é possibilitada por guias lineares abaixo dos servos, e os outros quatro servomotores são responsáveis pelo movimento de rotação, que executa os movimentos no cubo. Os servomotores foram controlados por meio do Arduino Mega 2560. As garras fixadas aos servomotores de rotação foram impressas em 3D, e os componentes de suporte do mecanismo foram fabricados em acrílico e madeira. Para a resolução do cubo, foi utilizado o método BLD (*Blindfold*), que consiste em resolver o cubo de olhos vendados. A vantagem desse projeto está no seu baixo custo, fácil implementação e robustez quanto aos movimentos necessários para realizar a resolução. A desvantagem do projeto é que o mecanismo de entrada de dados consiste em informar as cores manualmente, por meio de comunicação serial com o computador, o que pode tornar o processo de identificação das cores do cubo demorado e suscetível a erros. Embora o projeto não utilize visão computacional, ele demonstra um mecanismo simples, porém eficaz, para a resolução automática do cubo de Rubik.

Figura 10 – Mecanismo automatizado para a resolução do cubo de Rubik.



Fonte: (CARDEAL, 2022).

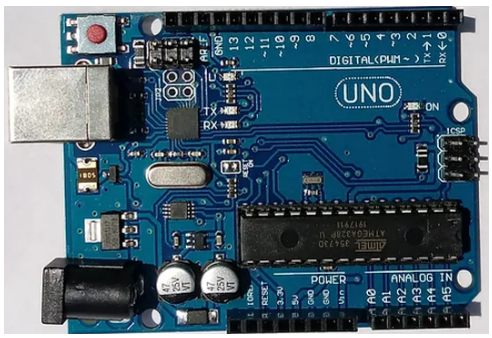
# 3 Materiais e métodos

Esta seção tem como objetivo detalhar todas as etapas do desenvolvimento do protótipo e os principais componentes utilizados em sua construção.

## 3.1 Arduino

O Arduino é uma plataforma de prototipagem eletrônica de código aberto, criada com o objetivo de ser uma ferramenta fácil de usar e destinada ao aprendizado de eletrônica e lógica de programação. Devido à sua facilidade de uso, a plataforma Arduino é amplamente adotada em diversos projetos de automação, robótica e internet das coisas (IoT). Com o Arduino, é possível criar modelos de projetos eletrônicos que podem ter diversas funções, como controle de motores, leitura de dados de sensores, controle de LEDs e transmissão ou recepção de dados de sistemas remotos. Para a programação do Arduino, é necessário instalar o Arduino IDE (*Integrated Development Environment*), um ambiente de desenvolvimento integrado que permite a criação de códigos utilizando a linguagem de programação C++, que podem ser carregados diretamente para o dispositivo via conexão USB. Tendo em vista as possibilidades oferecidas por essa plataforma e a familiaridade do autor com essa tecnologia, neste projeto foi escolhido o Arduino Uno R3 como a placa microcontroladora para o controle dos atuadores que realizam os movimentos de resolução do cubo de Rubik. O Arduino Uno R3 é uma placa microcontroladora baseada no microcontrolador de chip único ATmega328P. Esse modelo possui 14 entradas e saídas digitais, sendo que seis saídas podem ser usadas como PWM (*Pulse Width Modulation*), muito utilizadas no controle de servomotores, e possui também seis entradas analógicas, um cristal oscilador de 16 MHz, conexão USB, uma entrada para fonte, soquetes para o protocolo de comunicação ICSP (*In-Circuit Serial Programming*) e um botão de reset.

Figura 11 – Arduino Uno R3 compatível com ATmega328P.

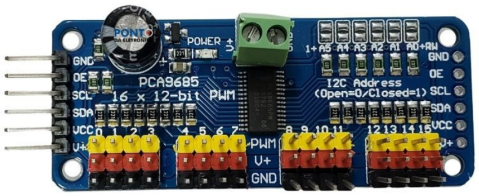



Fonte: (Just4Fun Electronics, 2024).

Devido à presença de apenas seis saídas PWM no Arduino, foi necessário adicionar um

módulo eletrônico auxiliar para que fosse possível controlar os oito servo motores necessários para realizar os movimentos de resolução do cubo de Rubik. O módulo escolhido foi o controlador PWM PCA9685, que possui 16 canais PWM de 12 bits e é muito utilizado em projetos que precisam controlar múltiplos servomotores e LEDs em aplicações de robótica e automação (Circuit Digest, 2024). Esse módulo permite que possamos expandir os canais PWM do Arduino para 16 canais, o que atende aos requisitos deste trabalho.

Figura 12 – Módulo controlador PWM 16 canais 12 bits PCA9685.




Fonte: (Ponto da Eletrônica, 2024).

O módulo controlador PWM PCA9685 utiliza o protocolo I2C (*Inter-Integrated Circuit*) para a comunicação com o Arduino. Essa conexão permite que múltiplos dispositivos sejam conectados ao mesmo barramento, tornando o módulo versátil para controlar vários componentes, com uma quantidade mínima de fiação. Para utilizar o PCA9685, é preciso acessar o Arduino IDE e realizar a instalação da biblioteca *Adafruit PWM Servo Driver Library*, que possui todas as funcionalidades necessárias para realizar o controle dos servomotores de forma simplificada.

Foram utilizados oito servomotores MG995 para realizar os movimentos de translação e rotação necessários à resolução do cubo de Rubik. Cada servomotor possui uma faixa de rotação de 180° e uma tensão de operação que varia entre 4,8 V e 7,2 V. Com uma tensão de 4,8 V, o torque é de 9,4 kg.cm, enquanto a 6 V o torque é aumentado para 11,0 kg.cm (Tower Pro, 2024).

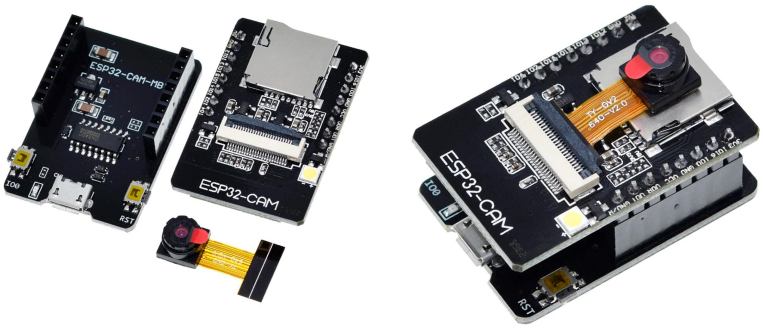
Figura 13 – Servomotor MG995 Tower Pro.


Fonte: (Maker Hero, 2024).

## 3.2 ESP32-CAM

Como dispositivo de aquisição de imagens utilizado neste trabalho, foi empregado o módulo ESP32-CAM, que integra as funcionalidades do microcontrolador ESP32 com suporte a câmera OV2640 de 2 MP. Além disso, o modelo utilizado acompanha um conversor *Shield USB Serial* CH340, que facilita a conexão do ESP32-CAM ao computador via USB, eliminando a necessidade de fios ou circuitos eletrônicos auxiliares. A câmera do ESP32-CAM permite a transmissão de imagens capturadas em tempo real por meio de um servidor web executando diretamente no dispositivo. Para habilitar essa funcionalidade foi necessário instalar o *plugin* fornecido pelo fabricante (Espressif Systems, 2024) no ambiente Arduino IDE. Após a instalação, foi necessário acessar o menu disponibilizado, verificar a listagem de exemplos e selecionar o programa *CameraWebServer*. Feita a seleção, o código-fonte do programa em C++ aparecerá no editor, e os parâmetros de acesso à rede à qual o ESP32-CAM irá se conectar devem ser alterados, informando o nome da rede e a senha. Deve-se também escolher o modelo de ESP32-CAM que está sendo utilizado, dentre os disponíveis presentes no código, na forma de comentários. Com o código configurado, o programa foi carregado no ESP32-CAM, e o acesso à câmera foi realizado digitando, em um navegador web, o IP (*Internet Protocol*) atribuído pelo roteador da rede ao módulo, por meio de um *notebook* conectado à mesma rede, via conexão Wi-Fi utilizando o protocolo HTTP. Ao acessar o programa *CameraWebServer* no navegador, é possível configurar diversos parâmetros da câmera como brilho, saturação e contraste, processo esse chamado de calibração. Testes realizados durante a montagem do protótipo mostraram que a detecção do cubo e a captura das cores tornam-se mais robustas quando os parâmetros de saturação, contraste e brilho são aumentados, e que a redução dos parâmetros de qualidade da imagem, como a densidade de pixels, torna a transmissão das imagens do ESP32-CAM via Wi-Fi mais rápida.

Figura 14 – Modelo de ESP32-CAM utilizado e seus componentes constituintes.



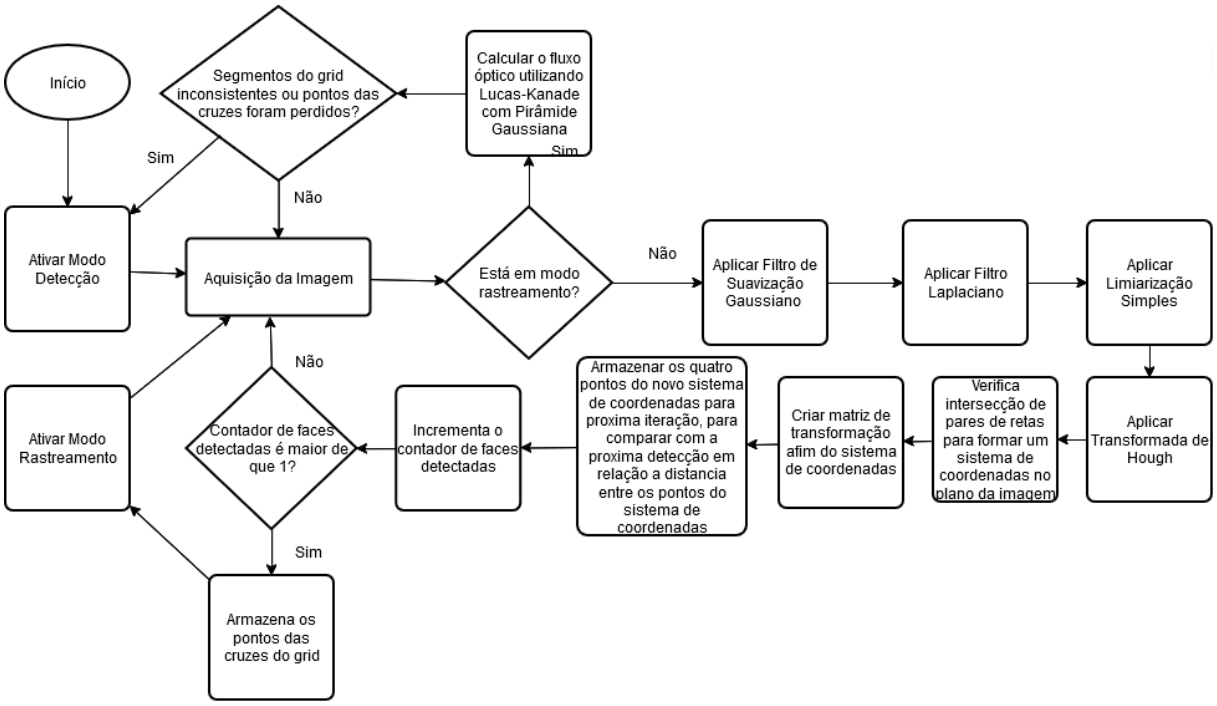

Fonte: (Casa da Robótica, 2024).

## 3.3 Algoritmo de visão computacional

Nesta seção, serão apresentadas as etapas realizadas pelo algoritmo de visão computacional utilizado neste trabalho para realizar a detecção e rastreamento do cubo de Rubik, bem como o processamento e a identificação das cores. O algoritmo utilizado teve como referência o trabalho anterior desenvolvido por (KARPATHY, 2009) e foi implementado na linguagem de programação Python, em conjunto com a biblioteca OpenCV, que possui diversos algoritmos de visão computacional pré-implementados e fornece uma interface de programação amigável para o desenvolvimento de aplicações que trabalham com extração de informações de imagens.

### 3.3.1 Detecção e rastreamento

Figura 15 – Fluxograma de execução da detecção e rastreamento do cubo de Rubik na imagem.




Fonte: Autor.

O processo de detecção e rastreamento do cubo de Rubik começa com a aquisição da imagem por meio do ESP32-CAM. Inicialmente, o programa Python realiza uma requisição ao servidor web do ESP32-CAM e estabelece uma conexão HTTP via Wi-Fi para a aquisição das imagens. Esse processo é gerenciado pela biblioteca OpenCV, por meio da funcionalidade de *VideoCapture*. Em seguida, o usuário deve apresentar as seis faces do cubo em uma ordem predefinida no programa. Predefinimos a ordem de apresentação das faces, pois isso facilita a identificação das cores pelo algoritmo. Em cada posição dessa ordenação, assume-se que a face apresentada corresponde a uma cor específica. Como a cor central define a cor da face do cubo, ela pode ser identificada diretamente, o que auxiliará no processamento das cores posteriormente.

#### 3.3.1.1 Detecção

Cada face apresentada passa por um processo de detecção, no qual é aplicado um pré-processamento por meio da abordagem baseada em bordas. Inicialmente, a imagem é suavizada para remoção de ruídos e variações muito bruscas de intensidade dos pixels, utilizando o filtro gaussiano. Em seguida, é aplicado um filtro laplaciano, que realiza a extração de características de bordas da imagem. Após isso, a limiarização simples é aplicada para realçar as bordas identificadas, binarizando a imagem em pixels claros, onde há grande variação de intensidade (bordas) e pixels escuros, onde não há variação. A seguir, outra característica é extraída por meio da transformada de Hough que, ao final, retorna a posição dos segmentos de reta que pertencem à face do cubo. Alguns critérios são adotados para a seleção dos segmentos de reta que são candidatos a formar um sistema de coordenadas do cubo:

- Os segmentos devem ter tamanhos próximos ou similares. O que reflete a aparência original das bordas do cubo.
- Algumas das extremidades de um segmento devem estar próximas de alguma das extremidades do outro segmento. Isso indica que possam ser segmentos que se intersectem em suas extremidades.
- Caso não existam segmentos de retas que se intersectem em suas extremidades, verifica-se se eles se intersectam em algum ponto no intervalo entre seus extremos utilizando a técnica descrita em (SAALFELD, 1987). Em seguida, compara-se os parâmetros obtidos (equações paramétricas das retas) com os valores de um terço e dois terços, pois estes significam que os segmentos de reta se intersectam como os segmentos internos do cubo de Rubik. Em seguida, com base nos pontos finais e iniciais dos segmentos de retas que se intersectam, são obtidos três pontos do sistema de coordenadas, que vão de zero a um nas direções horizontais e verticais partindo da origem, manipulando-se os pontos dos segmentos de reta, utilizando operações da álgebra vetorial. Esses pontos são armazenados para validações posteriores.

Com base nos segmentos de reta que atendam a essas validações, os pontos do sistema de coordenadas obtidos são testados quanto ao seu maior ângulo. Isso é feito para garantir uma consistência no sistema de coordenadas obtido. Quando dois segmentos de reta são perpendiculares, o maior ângulo formado por eles é 90° e quando estes são paralelos, o maior ângulo formado é 180°. Portanto, subtrai-se os ângulos de cada segmento em relação ao eixo horizontal positivo, em módulo, de forma a obter o ângulo formado entre eles e em seguida, subtrai-se o valor obtido por 90°, em módulo. Se o valor obtido após esses cálculos for maior do que 0°, significa que os segmentos têm uma tendência a serem paralelos, quanto maior for o resultado obtido. Logo, é necessário definir um valor máximo para que o algoritmo considere os segmentos de reta como perpendiculares. No algoritmo utilizado, foi configurado que esse valor deve ser menor do que 30°.

Após a obtenção dos pontos do sistema de coordenadas, é criada uma matriz de transformação afim $T$, onde as colunas representam os vetores que formam a base do novo sistema de coordenadas, e são utilizadas coordenadas homogêneas para transformar a matriz para um formato 3x3, onde a terceira coluna contém as coordenadas da origem. Podemos representá-la da seguinte forma:

$$T = \begin{bmatrix} x_1 - x_0 & x_2 - x_0 & x_0 \\ y_1 - y_0 & y_2 - y_0 & y_0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Essa matriz deve ser invertida para permitir a conversão dos pontos no plano da imagem para o plano do novo sistema de coordenadas, localizado na face detectada do cubo de Rubik. Em seguida, é realizada a verificação de quantos segmentos de reta se alinham ao novo sistema de coordenadas e que podem representar os segmentos de reta internos do cubo. Para isso, é analisado o critério do maior ângulo para cada segmento de reta detectado, em relação aos segmentos que correspondem aos eixos do sistema de coordenadas. Se o segmento de reta detectado for perpendicular a pelo menos um dos dois eixos, ele será considerado como pertencente ao *grid* do cubo. Na implementação, foi utilizado o valor de até 6°, como critério para o maior ângulo formado, para que o segmento de reta possa ser considerado como perpendicular aos eixos considerados.

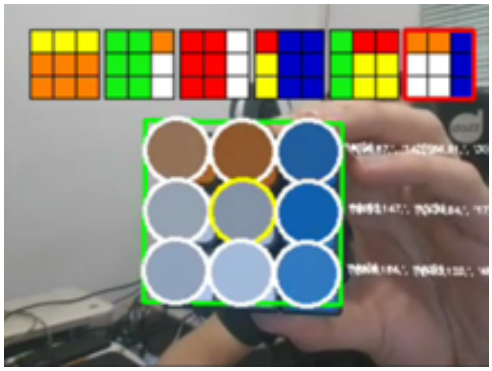
Cada ponto, que representa a extremidade de um segmento de reta detectado, é projetado no novo sistema de coordenadas por meio da transformação linear definida pela matriz afim inversa e, então, são realizadas as seguintes validações de consistência:

- A posição do ponto projetado no novo sistema de coordenadas deve estar entre zero e um para que pertença ao *grid* do cubo de Rubik.
- O ponto deve estar a uma distância de um terço ou dois terços dos eixos do sistema de coordenadas, pois isso indica que ele pertence aos segmentos internos da face detectada.

Quando as duas extremidades do segmento de reta detectado atenderem a essas validações, o segmento, então, é considerado como pertencente ao *grid* interno do cubo. A quantidade de segmentos de reta em conformidade com essas validações, associada a cada sistema de coordenadas obtido, é armazenada. No algoritmo utilizado, dois critérios devem ser atendidos para que a detecção seja considerada bem-sucedida: ao menos três linhas internas válidas devem ser detectadas para um sistema de coordenadas específico e a média das distâncias euclidianas mínimas deve ser menor do que 10 pixels. Essa média é calculada somando-se as menores distâncias dos pontos do sistema de coordenadas atual em relação a cada ponto da última detecção válida e dividindo o resultado pelo número de pontos. Caso o novo sistema de coordenadas atenda a esses critérios, seus quatro pontos, representando os vértices da face detectada, são armazenados em uma lista. Após a obtenção desses quatro pontos, a detecção é finalizada. No entanto, há um parâmetro configurado no algoritmo que define a quantidade de

detecções válidas necessárias para que a etapa do rastreamento seja iniciada. Esse parâmetro está configurado para realizar duas detecções bem-sucedidas da face do cubo, o que garante maior robustez ao processo e permite que detecções incorretas sejam desconsideradas. Após duas detecções realizadas com sucesso, a etapa de rastreamento é iniciada.

Figura 16 – Detecção do cubo de Rubik realizada com sucesso.



Fonte: Autor.

#### 3.3.1.2 Rastreamento

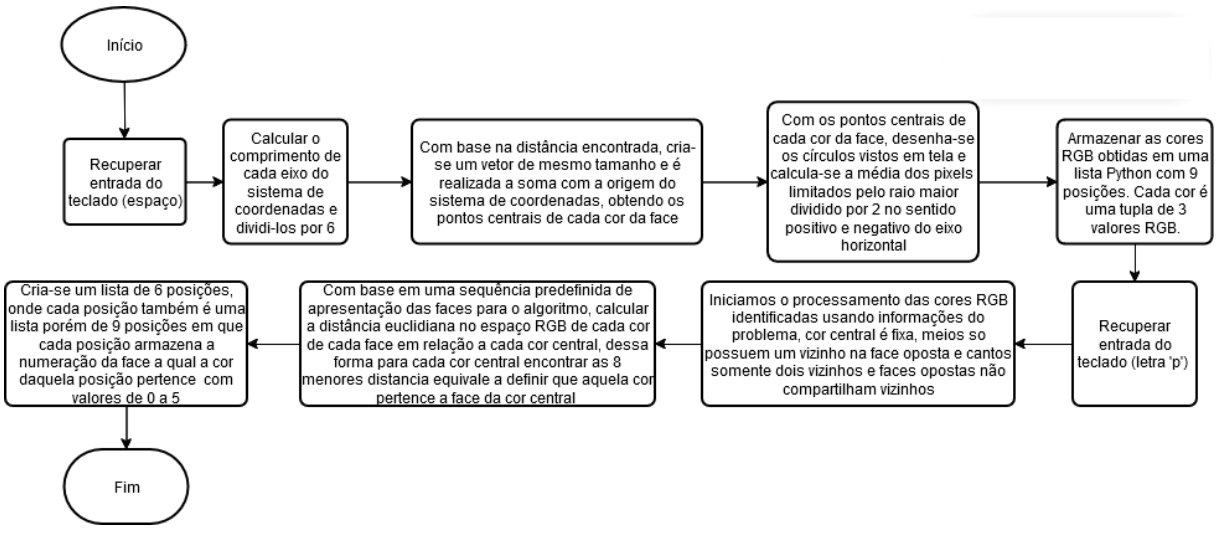
Após a detecção de faces suficientes para que possamos considerar uma detecção correta da face do cubo, a etapa seguinte consiste no rastreamento. Por meio do sistema de coordenadas obtido, os pontos correspondentes às cruzes do cubo (trechos de interseção dos segmentos de reta internos do cubo), são armazenados em uma lista. Essa lista é então passada para o algoritmo de rastreamento Lucas-Kanade, responsável pelo cálculo do fluxo óptico entre os pontos do sistema de coordenadas do *frame* atual em relação ao próximo *frame* transmitido pelo ESP32-CAM. Esses pontos foram escolhidos por se tratar de características muito comuns às faces do cubo de Rubik, o que facilita sua localização por algoritmos de rastreamento. Pontos com essas características são conhecidos como pontos-chave ou *keypoints*. Utilizando a biblioteca OpenCV, os pontos das cruzes do cubo, juntamente com as imagens anterior e atual, são passados como parâmetro para o método *calcOpticalFlowPyrLK* (OpenCV, 2024). Esse método realiza o cálculo do fluxo óptico dos pontos para a nova imagem e retorna suas posições atualizadas na nova imagem. A técnica é associada ao uso de pirâmides gaussianas, técnica muito utilizada na visão computacional para reduzir a resolução da imagem em escalas múltiplas de dois. Com isso, é possível obter imagens com menos detalhes, devido a menor resolução, priorizando os elementos com maior presença na imagem, como grandes objetos. Além disso, o método permite uma melhor detecção de movimentos rápidos, evidenciados por vetores de fluxo óptico com deslocamentos maiores entre *frames* consecutivos em um *streaming* de vídeo. Como método de verificação de consistência para os pontos detectados pelo algoritmo de rastreamento, são realizadas as seguintes validações:

- É verificado se o comprimento das linhas horizontais mais acima e mais abaixo entre as cruzes do cubo de Rubik são próximos ou similares. No algoritmo utilizado, o parâmetro adotado estabelece que a diferença entre o comprimento dessas linhas horizontais não deve exceder 40%. O mesmo critério aplica-se às linhas verticais.
- As linhas entre as cruzes do cubo, sejam verticais ou horizontais, não devem ser muito curtas. No algoritmo, um valor mínimo de 10 pixels é utilizado como critério nesta validação. Linhas com comprimentos inferiores a esse valor são consideradas muito curtas.

Em caso de não conformidade com essas validações, o algoritmo considera que o rastreamento falhou. Logo, será necessária uma nova aquisição de imagem e uma nova detecção para que o rastreamento seja reiniciado. O rastreamento permite que o algoritmo seja mais eficiente na identificação do cubo, pois a cada novo *frame* recebido, não é necessário realizar uma nova detecção do zero. Os novos pontos do sistema de coordenadas são calculados com base na localização dos pontos rastreados das cruzes do cubo. Como esses pontos estão dispostos de maneira simétrica em relação as bordas esquerda e inferior do cubo, é possível, por meio de soma vetorial, ajustá-los novamente para as bordas do cubo na nova imagem, preservando a detecção realizada na imagem anterior.

### 3.3.2 Processamento das cores

Figura 17 – Fluxograma do processamento das cores do cubo de Rubik.



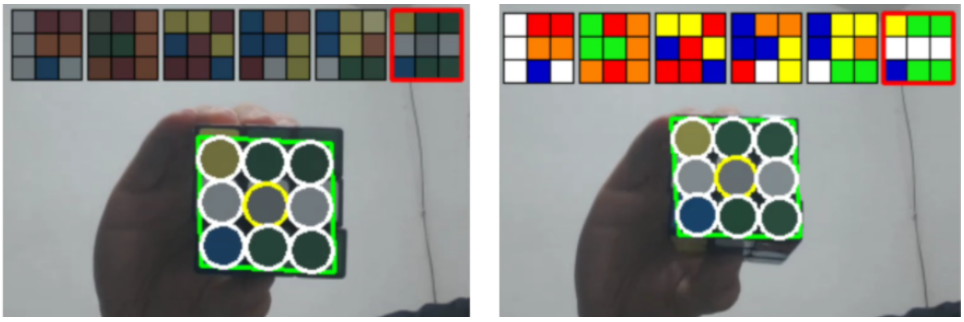

Fonte: Autor.

Para cada face do cubo de Rubik apresentada, o processo de detecção e rastreamento é executado repetidamente. As linhas e círculos que aparecem na região em que a face foi detectada são desenhadas com o auxílio de métodos geométricos da biblioteca OpenCV, como *line* e *circle*. Para a detecção da cor dentro de cada círculo, os vetores que formam o sistema de coordenadas são divididos por seis, permitindo acessar o centro de cada posição de cor da face, a

partir de múltiplos dessa divisão. Por exemplo, para acessar o centro da cor localizada na posição mais inferior à esquerda, multiplica-se o vetor resultante (da divisão) por um em cada direção do sistema de coordenadas. Já para acessar a cor de posição central da face, o vetor resultante deve ser multiplicado por três em cada direção. Com o pixel central identificado, calcula-se a média dos pixels em sua vizinhança, e esse valor médio obtido representará a cor correspondente naquela posição da face. Quando o usuário deseja armazenar as cores da face, que estão sendo atualmente detectadas, basta pressionar a tecla de espaço no teclado. A média das cores é, então, armazenada e exibida na parte superior da tela, juntamente com os valores das cores detectadas. Esse processo se repete para as seis faces do cubo, que devem ser apresentadas para o sistema de visão computacional em uma ordem predefinida no código. Essa abordagem garante que as cores centrais do cubo sejam corretamente identificadas.

Ao pressionar ’R’ no teclado, os dados de cor são descartados, permitindo uma nova detecção. Com as teclas ’M’ e ’N’, é possível navegar entre as detecções realizadas, avançando ou retrocedendo. Caso a tecla de espaço seja pressionada em uma detecção já salva, ela será sobrescrita pela detecção mais recente. Ao pressionar ’P’ após a detecção das cores das seis faces,

Figura 18 – Detecção das cores do cubo de Rubik e processamento das cores realizada.



Fonte: Autor.

é iniciado o processamento das cores, ou seja, será definida a qual face cada cor está associada. Inicialmente, o algoritmo itera sobre cada cor de cada face do cubo de Rubik. Para cada cor, calcula-se a distância euclidiana no espaço RGB entre a cor específica e cada uma das faces do cubo. A face que corresponder à menor distância é armazenada em uma matriz, representando a cor por meio de um valor inteiro de zero a cinco, e a posição dessa cor no cubo detectado é registrada como o índice de linha e coluna da matriz. Se algum vizinho da cor atual já tiver o mesmo índice de face da cor em análise, esse vizinho deverá ter sua face removida da matriz. Isso se deve às características do cubo, nas quais as quinas e os meios possuem cores vizinhas em faces diferentes e não podem pertencer à mesma face, inclusive as cores de faces opostas. Para cada face, é mantida uma contagem de ocorrências de quantas cores já foram atribuídas a ela. Essa contagem é usada na verificação da distância euclidiana, de modo que, se a face já estiver totalmente preenchida, a cor analisada será direcionada para a face com a próxima menor distância.

Esse processo de identificação se repete para todas as peças coloridas de cada face do cubo, exceto a peça central, até que a matriz de cores seja montada corretamente. Nessa matriz, as linhas representam as faces, e as colunas representam as cores identificadas. Cada cor é armazenada como o índice da face à qual pertence, e considerando sua posição, obedecendo à ordem de cima para baixo da esquerda para direita, em relação a face do cubo. Após a identificação, as cores são exibidas na tela com seus valores fixados para o usuário. Caso sejam corretamente identificadas, elas são enviadas automaticamente ao algoritmo de resolução Kociemba. A solução gerada é então transmitida ao Arduino, que, por meio dos atuadores, executa os movimentos necessários para resolver o cubo de forma automática.

## 3.4 Biblioteca Python: Kociemba

Figura 19 – Exemplo de uso da biblioteca Kociemba para um estado qualquer do cubo e para o estado resolvido.

```
import kociemba
solution = kociemba.solve('DRLUUBFBRBLURRLRUBLRDDFDLFUFUFFDBRDUBRUFLLFDDBFLUBLRBD');
print(solution)
```

```
D2 R' D' F2 B D R2 D2 R' F2 D' F2 U' B2 L2 U2 D R2 U
```

```
solvedCube = 'UUUUUUUUURRRRRRRRRFFFFFFFFFDDDDDDDDDLLLLLLLLLBBBBBBBBB'
solution = kociemba.solve(solvedCube)
print(solution)
```

```
R L U2 R L' B2 U2 R2 F2 L2 D2 L2 F2
```

Fonte: Autor.

Após o processamento das cores detectadas, o próximo passo é enviar a matriz de cores obtida para o algoritmo de resolução, que ficará responsável por definir os movimentos necessários para a resolução do cubo de Rubik. Para atingir esse objetivo, foi utilizada a biblioteca Kociemba, implementada na linguagem Python. Essa biblioteca fornece uma interface de programação amigável e intuitiva, simplificando o uso do algoritmo Kociemba, uma vez que este já está implementado internamente. Após a instalação e importação da biblioteca, foi utilizado o método *solve* que recebe uma sequência de caracteres representando a posição das cores do cubo de Rubik. Nessa etapa, foi necessário realizar um mapeamento entre os valores numéricos das faces, obtidos pelo algoritmo de visão computacional, e os caracteres alfabéticos necessários para a execução correta do algoritmo Kociemba. O esquema de cores ocidental do cubo foi adotado, e a cadeia de caracteres gerada segue o padrão definido pela biblioteca, na qual a sequência de faces deve ser informada na seguinte ordem: U (branca), seguida por R (vermelha), F (verde), D (amarela), L (laranja) e B (azul). Cada letra na cadeia de caracteres representa uma cor, indicando a face à qual essa cor pertence. As nove cores de cada face devem ser apresentadas

em sequência, seguindo a ordem de cima para baixo, da esquerda para direita (Python Package Index, 2024). Realizando testes na biblioteca, foi verificado que, ao informar o estado resolvido do cubo, a biblioteca executa alguns movimentos seguindo o algoritmo Kociemba. No entanto, como o objetivo é solucionar o cubo, os movimentos se tornam redundantes, pois o cubo retornará ao seu estado resolvido após a execução.

## 3.5 Comunicação com o Arduino

A comunicação entre o Arduino e a aplicação Python, responsável pelos algoritmos de visão computacional e de resolução do cubo de Rubik, foi implementada por meio de comunicação serial via cabo USB. Para isso, foi necessário o uso da biblioteca *PySerial* que possibilita a comunicação de aplicações Python e outros dispositivos por meio de comunicação serial, oferecendo funcionalidades básicas de recebimento e transmissão de dados. Inicialmente foi implementada a função *serialEvent()* no código do Arduino (Arduino Docs, 2024). Essa função sempre é executada ao final da função *loop()* e verifica se há dados disponíveis na entrada serial por meio da função *Serial.available()*. Se houver dados disponíveis, ela realiza a leitura por meio da função *Serial.read()* e armazena a cadeia de caracteres recebida, contendo a sequência de movimentos de resolução, que é armazenada numa variável do tipo *string*. Se o recebimento da cadeia de caracteres for realizado com sucesso, uma variável booleana de sucesso é configurada como verdadeira e na próxima execução da função *loop()*, o programa Arduino iniciará a resolução do cubo. Para cada movimento detectado na cadeia de caracteres de resolução, foi implementada uma função que realiza o movimento descrito. Uma estrutura de repetição foi implementada para iterar sobre cada caractere e, ao detectar um movimento, seu respectivo comando é enviado para o controle dos atuadores, que executam os movimentos diretamente no cubo. Ao final das iterações o cubo estará resolvido e, então, é liberado. Em seguida, todo o processo de detecção das cores, resolução e envio para o Arduino pode ser reiniciado, permitindo que novas resoluções possam ser executadas para estados diferentes do cubo.

## 3.6 Montagem do protótipo

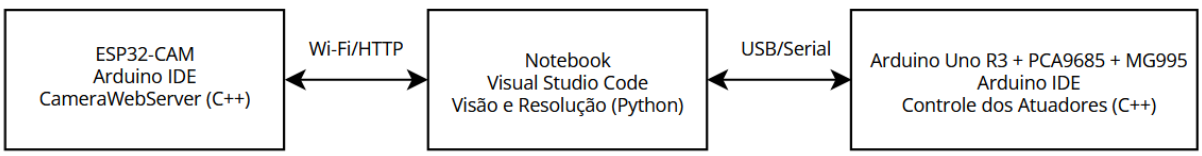
A etapa de montagem do protótipo iniciou-se com a aquisição das peças necessárias, baseando-se na análise dos trabalhos anteriores relatados. O modelo de robô e o formato das peças utilizadas foram inspirados no trabalho anterior visto em (CARDEAL, 2022). Posteriormente, foi realizada a montagem e foram adquiridos itens auxiliares adicionais, que se mostraram necessários para a correta execução da montagem, como parafusos e jumpers extras para as conexões mecânicas e elétricas do protótipo. Foi utilizado o auxílio de impressão 3D, que se mostrou bastante útil para imprimir algumas peças do protótipo que, de outra forma, seriam mais difíceis de se produzir ou mais caras de se adquirir.

Em seguida, iniciou-se a etapa de programação e testes dos microcontroladores, utilizando a linguagem de programação C++, e do algoritmo de visão computacional e resolução do cubo, desenvolvido em Python. Essa última etapa foi importante pois também incluiu a calibração do protótipo. A calibração consistiu em sucessivas execuções do processo completo de resolução automática do cubo de Rubik usando visão computacional, com a identificação dos erros que ocorreram durante esse percurso. Entre os erros identificados, destacaram-se travamentos em certos movimentos dos atuadores, que impediam que a garras de rotação concluíssem seus movimentos, comprometendo, assim, a conclusão da resolução. Além disso, outro problema foi identificado, pois, devido à estrutura utilizada, não seria possível rotacionar as faces superior (U) e inferior (D) sem alterar a posição inicial do cubo. Portanto, foi necessário programar um movimento de giro no eixo horizontal do cubo, envolvendo os atuadores das faces L e R. Esse movimento exigiu uma programação de movimentos especiais, para evitar que o cubo caísse da estrutura. Enquanto um servomotor mantém sua posição original na vertical, o da face oposta se posiciona na horizontal, dessa forma o cubo pode ser rotacionado corretamente no eixo horizontal. Durante a montagem, também foi definida a posição inicial do cubo: a face branca voltada para cima, e a face verde orientada para o servomotor de rotação da face frontal (F) que se localiza a direita da posição do ESP32-CAM, com sua câmera voltada para fora da estrutura e apontada para o usuário.

Além disso, foram realizadas calibrações no ESP32-CAM, por meio da sua interface gráfica que possibilitaram uma melhor captura das cores do cubo de Rubik, por meio do aumento dos valores de brilho, contraste e saturação. Os códigos do Arduino e Python passaram por refatorações e comentários adicionais para melhorar sua legibilidade e facilitar a compreensão.

A principal técnica utilizada durante os testes foi a de tentativa e erro, sendo o critério de qualidade definido como a capacidade do protótipo de realizar uma execução automática completa da resolução do cubo, sem erros que comprometessem sua execução. Após sucessivos testes executados, a versão final que atendeu a esse critério foi obtida com sucesso, e a montagem do protótipo foi finalizada. A arquitetura geral da versão final do protótipo está representada na figura abaixo.

Figura 20 – Arquitetura geral da versão final do protótipo.

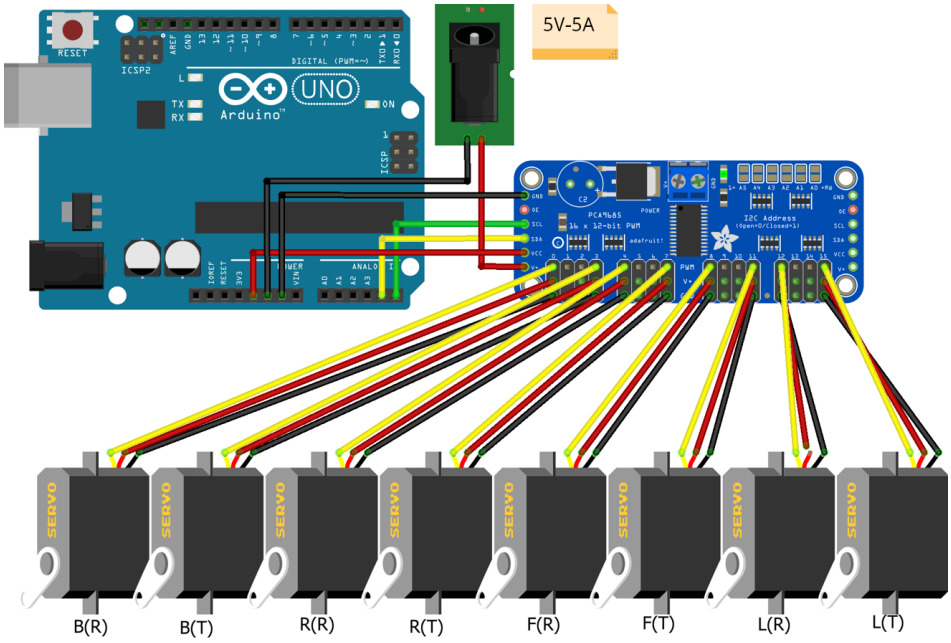



Fonte: Autor.

### 3.6.1 Diagrama do circuito

O diagrama do circuito utilizado para a conexão entre Arduino, PCA9685 e atuadores MG995 é exibido na figura 21. Cada servomotor é identificado por uma descrição com duas letras em maiúsculo. A primeira letra indica a face do cubo de Rubik à qual o servomotor está associado, e a segunda, entre parênteses, especifica o tipo de movimento possibilitado pelo servomotor, sendo eles, os movimentos de rotação e translação.

Figura 21 – Diagrama do circuito conectando Arduino, PCA9685 e servomotores MG995.




Fonte: Autor.

### 3.6.2 Componentes utilizados

Os seguintes componentes foram utilizados na montagem do protótipo:

- Componentes eletrônicos: incluem-se nessa categoria o Arduino Uno R3, PCA9685, ESP32-CAM e MG995, além dos cabos USB e micro-USB dedicados a comunicação e fornecimento de energia para o Arduino e ESP32-CAM, respectivamente.
- Componentes estruturais: esta categoria abrange a base de madeira de 40x40x1,5 cm, os pés redondos cromados, que garantiram o suporte físico da estrutura e a estabilidade necessária para a operação correta do projeto. Outro componente utilizado foi a guia linear que permitiu a inclusão de movimentos horizontais de translação no protótipo. Adicionalmente, foram utilizados fixadores de fio circulares para organizar os jumpers posicionados na parte inferior da base de madeira, melhorando a disposição da fiação. Também foram adquiridos parafusos diversos para conexões mecânicas entre os componentes. Para a fixação do *case*

do ESP32-CAM ao seu suporte, foi utilizado um parafuso em combinação com uma porca sextavada. Os critérios para a seleção dos parafusos foram estabelecidos a partir de testes realizados no momento da compra, com o objetivo de determinar o tamanho adequado para a fixação de cada componente. Portanto, não foram adotados critérios de dimensionamento técnico preciso para os parafusos utilizados neste trabalho. Outro critério considerado

Tabela 1 – Tabela com a quantidade de parafusos utilizados para cada tipo de fixação de componentes.

| Fixação de componentes | Parafusos Utilizados |
|---|---|
| Pés redondos cromados e base de madeira | 16 |
| Servomotores de translação e base de madeira | 16 |
| Guia linear e base de madeira | 8 |
| Suportes do servomotores de rotação e guias lineares | 8 |
| Servomotores de rotação e seus suportes | 8 |
| Servomotores de rotação e garras | 8 |
| Caixa e base de madeira | 4 |
| Caixa e arduino | 2 |
| Caixa e tampa | 4 |
| Suporte ESP32-CAM e base de madeira | 4 |
| Suporte ESP32-CAM e *case* | 1 |
| Suportes dos servomotores de rotação e haste | 4 |
| Total | 83 |

Fonte: Autor.

foi que os parafusos fixados à base de madeira não poderiam atravessá-la, ou provocar rachaduras, tanto na madeira quanto em outros componentes. Ao todo, foram utilizados 83 parafusos conforme apresentado na tabela 1. Outros componentes pertencentes a esta categoria incluem o arame número 20 (0,8 mm), utilizado para conectar os servomotores de translação ao eixo horizontal, o qual está conectado ao suporte do servomotor de rotação. Como o furo dessa conexão era muito estreito, não foram encontrados parafusos com encaixe adequado e que permitissem o movimento livre entre a haste e o braço do servo motor. Por isso, o arame foi utilizado como solução paliativa. Também foi utilizada cola quente para fixar o extensor DC P4 fêmea em uma abertura feita na caixa impressa em 3D que abriga o Arduino, PCA9685 e fiações, sendo a cola quente usada exclusivamente para essa função.

- Componentes elétricos: esta categoria incluiu a aquisição de jumpers para a conexão entre o Arduino, o PCA9685 e os servomotores, além de uma fonte chaveada 5V 5A, necessária para fornecer energia aos servomotores. Um extensor DC P4 fêmea foi utilizado para realizar a interface entre a fonte chaveada 5V 5A e os jumpers conectados aos componentes eletrônicos. Fita isolante foi empregada para garantir a fixação das conexões

entre os jumpers, além de reforçar as emendas elétricas realizadas entre o extensor e os jumpers utilizados.

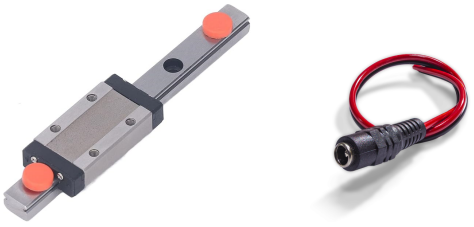
Tabela 2 – Tabelas de componentes utilizados e custo total para o ano de 2024.

| Item | Quantidade | Valor (R$) | Total (R$) |
|---|---|---|---|
| Arduino Uno R3 + cabo USB | 1 | 66,40 | 66,40 |
| PCA9685 | 1 | 44,99 | 44,99 |
| ESP32-CAM | 1 | 59,99 | 59,99 |
| MG995 | 8 | 49,90 | 399,20 |
| Guia linear 10 cm | 4 | 67,89 | 271,56 |
| Cubo de Rubik 3x3x3 Fellow Cube tradicional | 1 | 69,90 | 69,90 |
| Base de madeira 40x40x1,5 cm | 1 | 49,30 | 49,30 |
| Fonte chaveada 5 V 5 A | 1 | 49,90 | 49,90 |
| Extensor DC P4 fêmea | 1 | 2,90 | 2,90 |
| PLA 1 kg 1,75 mm | 1 | 102,90 | 102,90 |
| Pé redondo cromado | 4 | 8,20 | 32,80 |
| Cabo micro-USB Altomex | 1 | 25,00 | 25,00 |
| Arame galvanizado nº 20 1 kg | 1 | 17,10 | 17,10 |
| Fita isolante | 1 | 2,29 | 2,29 |
| Parafusos | 83 | 0,15 | 12,45 |
| Porca sextavada M3 | 1 | 0,10 | 0,10 |
| Bastão de cola quente 0,7x30 cm | 1 | 1,59 | 1,59 |
| Kit 10 jumpers (macho x macho) | 2 | 3,50 | 7,00 |
| Kit 10 jumpers (fêmea x fêmea) | 2 | 3,50 | 7,00 |
| Kit 10 jumpers (macho x fêmea) | 2 | 3,50 | 7,00 |
| Fixador de fio circular 6,0 mm² (pct. 50 un.) | 1 | 6,90 | 6,90 |
| Total (R$) | | | 1.236,27 |

Fonte: Autor.

Os componentes utilizados, bem como a quantidade e os preços associados a cada um deles, podem ser consultados na tabela 2. Os valores listados correspondem aos preços praticados pelo mercado no ano de 2024 e podem variar, dependendo dos anos subsequentes. Ao final, é apresentada a soma total dos valores, obtendo assim o custo estimado do projeto.

Figura 22 – Guia linear e extensor DC P4 fêmea utilizados na montagem do protótipo.

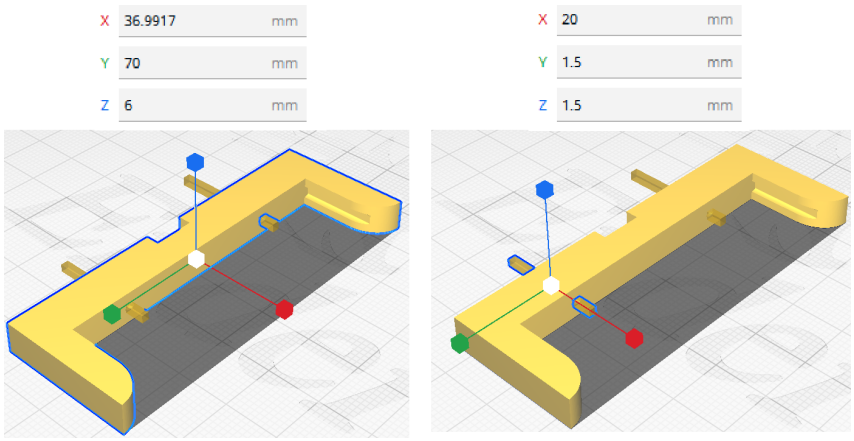


Fonte: (Amazon, 2024) e (Robocore, 2024).

### 3.6.3 Impressão 3D

A impressão 3D foi usada como método para aquisição de algumas peças específicas do projeto. As peças foram fabricadas com a impressora 3D Creality Ender 3, utilizando material PLA (*Polylactic Acid*), adequado para trabalhos de impressão. Os modelos 3D foram obtidos por meio de pesquisa em sites web da comunidade de prototipagem (Instructables, 2024a). O software Ultimaker Cura foi utilizado para a geração e configuração do arquivo de impressão executável *.gcode*, reconhecido pela impressora, permitindo a execução da impressão. Durante os testes do protótipo, identificou-se que o modelo da garra apresentava espessura excessiva e que os braços horizontais, presentes nas garras utilizadas para segurar e rotacionar o cubo de Rubik, estavam muito curtos. Isso ocasionava travamentos frequentes e, em alguns casos, fazia com que o cubo caísse da estrutura, pois a garra não conseguia segurá-lo adequadamente. Para solucionar esses problemas, foram realizadas mudanças nas dimensões das garras por meio do software Ultimaker Cura, ajustando-as para reduzir a espessura e aumentar o comprimento dos braços horizontais. Essas mudanças eliminaram os travamentos e garantiram que o cubo permanecesse estável durante todo o processo de resolução.

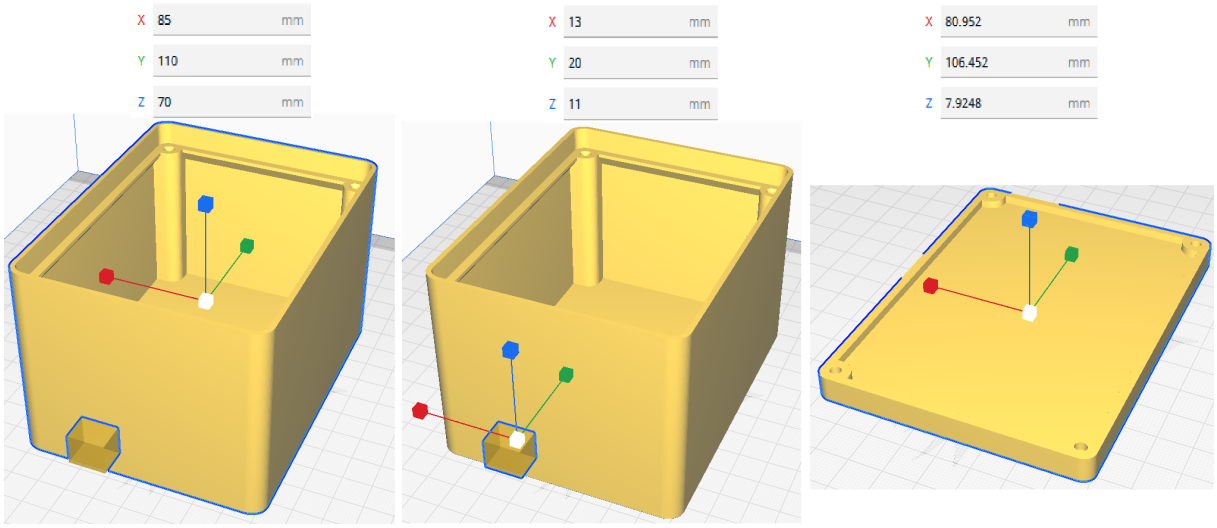
Figura 23 – Modelo 3D e dimensões da garra e furos no Ultimaker Cura.




Fonte: Autor.

Outro modelo 3D que foi obtido por meio da comunidade, mas que teve suas medidas alteradas, foi o modelo da caixa utilizada para abrigar as fiações e os componentes eletrônicos responsáveis pelo controle dos atuadores. O modelo obtido trata-se de uma caixa de uso geral, que pode ser redimensionada conforme a necessidade do usuário. Assim, o modelo foi redimensionado para ocupar um espaço adequado sobre a base de madeira do protótipo, sem interferir na movimentação dos servomotores. Além disso, foi configurado um *support blocker* com preenchimento zero de material PLA. Essa funcionalidade do Ultimaker Cura permite abrir furos no modelo 3D e dessa forma possibilitou a criação de uma abertura na parede da caixa. Posteriormente, foi realizado o encaixe da entrada do cabo USB do Arduino, viabilizando a comunicação com o dispositivo, que foi posicionado no interior da caixa.

Figura 24 – Modelo 3D e dimensões da caixa usada para os componentes eletrônicos.



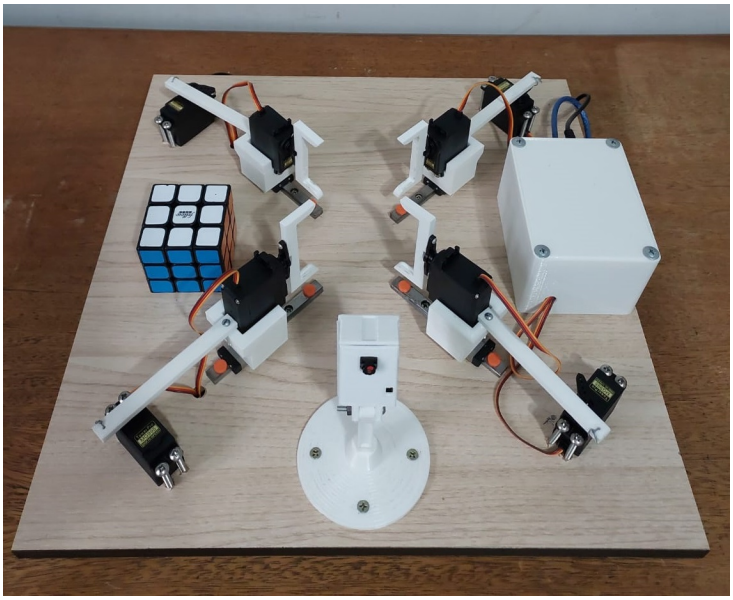

Fonte: Autor.

Por fim, para que o ESP32-CAM pudesse se posicionar de forma que a câmera ficasse na vertical quando conectada ao notebook por meio do cabo micro-USB, foi impresso um modelo 3D de suporte conforme disponibilizado em (Cults3D, 2024). O suporte foi parafusado na base de madeira, de forma a manter a estabilidade da câmera durante a etapa de aquisição de imagens do processo de resolução automática do cubo de Rubik.

# 4 Resultados

Como resultado deste trabalho, foi obtido um protótipo funcional capaz de solucionar automaticamente o cubo de Rubik, utilizando visão computacional como mecanismo de entrada de dados. Além disso, foi obtido um programa de visão computacional implementado em Python, com o uso da biblioteca OpenCV, que é capaz de detectar o cubo por meio de imagens capturadas por uma câmera presente no módulo ESP32-CAM. Esse programa identifica as cores presentes nas seis faces do cubo e armazena essas informações para apoiar a tomada de decisões do algoritmo de resolução Kociemba, responsável por definir os movimentos necessários para solucionar o cubo. A sequência de movimentos gerada é então enviada ao Arduino, que controla os atuadores responsáveis por executar, de forma sequencial, os movimentos que alteram o estado do cubo até que ele seja solucionado.

Figura 25 – Protótipo com a montagem finalizada.



Fonte: Autor.

O mecanismo executa os movimentos de forma ininterrupta até que a sequência de movimentos seja concluída, momento esse em que o cubo de Rubik estará em seu estado resolvido. Ao atingir esse estado, o cubo é liberado, permitindo que seja embaralhado novamente para execução de novas resoluções. Observou-se que a taxa de sucesso durante as resoluções é de 100%: independentemente do estado inicial informado, o protótipo é capaz de retornar o cubo ao seu estado resolvido. Esse desempenho foi alcançado graças às calibrações realizadas no protótipo, que evitaram quedas do cubo da estrutura e eliminaram travamentos que poderiam comprometer a conclusão da resolução.

Com a estrutura construída e as peças utilizadas, o protótipo apresenta um tempo médio de resolução de 4 minutos e 3 segundos. Para essa estimativa, foram realizadas 10 medições e então foi calculada a média dos tempos obtidos, conforme apresentado na tabela 3.

Tabela 3 – Tempo de resolução do protótipo para diferentes estados do cubo de Rubik.

| Descrição | Tempo de Resolução (min.) |
|---|---|
| Estado 1 | 04:40 |
| Estado 2 | 03:15 |
| Estado 3 | 04:01 |
| Estado 4 | 03:53 |
| Estado 5 | 04:12 |
| Estado 6 | 04:00 |
| Estado 7 | 04:35 |
| Estado 8 | 03:56 |
| Estado 9 | 03:43 |
| Estado 10 | 04:23 |
| Tempo médio | 04:03 |

Fonte: Autor

Outro objetivo atingido com sucesso neste trabalho foi demonstrar, por meio de um exemplo prático, a viabilidade do uso da visão computacional em sistemas de automação. Embora a aplicação deste trabalho seja voltada ao cubo de Rubik, um quebra-cabeças mundialmente conhecido, os conceitos fundamentais envolvidos podem ser aplicados em diversas áreas. Por exemplo, em ambientes industriais, onde robôs guiados por visão computacional podem auxiliar na tomada de decisões, controle de qualidade e identificação de peças defeituosas em linhas de montagem. Outras áreas incluem a saúde, com robôs utilizados em cirurgias e diagnóstico de doenças, métodos esses apoiados por processos automatizados, em conjunto com técnicas de visão computacional. Na área de segurança, a visão computacional pode ser aplicada na identificação de pessoas por meio de características físicas, como face, íris ocular e digitais, além de ser utilizada na detecção de invasões ou no monitoramento do uso de EPIs (Equipamentos de Proteção Individual) por colaboradores em zonas de construção ou de risco.

Como limitações conhecidas do protótipo, identificou-se, durante a exposição do trabalho no 21º CONPEEX (Congresso de Pesquisa, Ensino e Extensão), que o algoritmo de visão computacional utilizado apresentou dificuldades na detecção das cores do cubo de Rubik, principalmente devido à variação de iluminação em certos momentos do evento. Isso resultava em reconhecimento incorreto das cores, o que exigia o reinício do processo de detecção e identificação das cores do cubo. Ainda durante o CONPEEX, foi identificado que o modelo de comunicação sem fio utilizado para a aquisição de imagens do ESP32-CAM, via roteamento móvel, foi bastante afetado por interferências de outros sinais de comunicação, especialmente durante momentos de maior movimentação no evento. Isso impactou negativamente o desempenho da detecção do cubo na imagem, sendo necessário recalibrar a qualidade da imagem do

Figura 26 – Exposição do protótipo no 21º Congresso de Pesquisa, Ensino e Extensão.

Fonte: Autor.

ESP32-CAM, reduzindo a densidade de pixels nas imagens obtidas para diminuir a demanda de desempenho da rede. Outra limitação refere-se à estrutura utilizada, que realiza movimentos simples apenas nas faces do cubo orientadas na horizontal, o que impede a realização de rotações diretamente nas faces superior e inferior. Isso implicou na necessidade de mais programação para ajustar o cubo em uma posição adequada, de modo que esses movimentos pudessem ser realizados e, com isso, houve impactos no tempo de resolução. Além disso, o protótipo é compatível apenas com cubos de tamanho similar ao utilizado no projeto. Esses devem possuir suavidade em seus movimentos, assim como é observado em cubos profissionais, ou seja, devem apresentar pouca resistência aos movimentos realizados e devem ter certa tolerância ao desalinhamento do cubo para concluir algum movimento. Cubos de Rubik do mesmo tamanho, mas sem essas características descritas, tendem a causar travamentos no mecanismo, impossibilitando a conclusão da resolução.

Durante a etapa final deste trabalho, os resultados obtidos e o protótipo construído foram comparados com projetos similares, conforme a tabela 4. Os valores de custo exibidos são estimativas baseadas nos preços de 2024 para os materiais utilizados, mencionados em cada projeto, uma vez que os valores exatos não foram explicitamente informados. Para o robô Gan, o valor refere-se ao seu preço de mercado em 2024, por ser um produto comercial. Todos os projetos comparados utilizam o algoritmo Kociemba como método de resolução, exceto o projeto apresentado em (Instructables, 2024b), que adota o método baseado em camadas. Essa diferença explica o tempo significativamente maior de resolução desse projeto em relação aos demais. Observa-se, também, que, embora o presente trabalho apresente um tempo de resolução superior em comparação aos outros projetos, ele se destaca pela robustez. O protótipo possui uma precisão de 100% para cada resolução iniciada, e o algoritmo de visão computacional empregado não apresenta restrições em relação ao plano de fundo. Essa característica contrasta com os projetos de (WEN; GAIESKI, 2020), (Instructables, 2024b) e (Raspberry Pi, 2024), que utilizam segmentação e geração de máscaras aplicadas às imagens para identificar regiões de interesse

(ROI), facilitando a captura das cores do cubo. Contudo, essas abordagens dependem de um plano de fundo simples. A presença de elementos complexos no plano de fundo da imagem pode comprometer a detecção correta das cores e afetar a funcionalidade do sistema.

Tabela 4 – Comparação do protótipo construído com projetos similares que solucionam o cubo de Rubik.

| Projeto | Tempo de Resolução (min.) | Precisão (%) | Custo (R$) |
|---|---|---|---|
| (LIMA, 2024) | 04:03 | 100 | 1.236,27 |
| (GANCUBE, 2024) | 00:15 | 100 | 799,90 |
| (WEN; GAIESKI, 2020) | 01:08 | 90 | 395,82 |
| (Instructables, 2024b) | 20:00 | 100 | 285,00 |
| (BECKER et al., 2015) | 01:30 | 70 | 5.499,00 |
| (Raspberry Pi, 2024) | 01:10 | 100 | 634,15 |

Fonte: Autor

Como proposta de aprimoramento para trabalhos futuros, sugere-se comparar a abordagem de visão computacional utilizada neste trabalho com técnicas de aprendizado de máquina e aprendizado profundo, a fim de aperfeiçoar a detecção do cubo de Rubik na imagem e o reconhecimento das cores em diferentes condições de iluminação. Outra proposta seria adequar o mecanismo implementado para suportar cubos de diferentes tamanhos ou aceitar variantes, como cubos 4x4 e 5x5. Recomenda-se também estudar a viabilidade de implementar mecanismos no protótipo que permitam a redução do tempo de resolução, seja por meio da substituição dos atuadores por modelos mais rápidos e precisos, como, por exemplo, motores de passo, seja alterando a estrutura para permitir que giros nas faces inferior e superior do cubo sejam realizados diretamente por atuadores, sem a necessidade de ajustar o cubo em uma posição específica.

# 5 Conclusão

A visão computacional é uma área com grande potencial. Assim como a visão humana é um importante sentido utilizado pelas pessoas para a aquisição de informações, a visão computacional pode ser utilizada para aprimorar equipamentos e máquinas, concedendo a estes equipamentos a funcionalidade de extrair informações de imagens, funcionalidade essa poderosa e que está relacionada aos avanços que observamos no dia a dia no contexto de mecanismos e sistemas inteligentes.

Este trabalho consistiu na construção do mecanismo de solução automática do cubo de Rubik e no aprimoramento do mecanismo de entrada de dados, que recebe as cores do cubo utilizando visão computacional. O objetivo foi atingido com sucesso. Como mecanismos de automação, foram utilizados os microcontroladores Arduino e ESP32-CAM, responsáveis pelo controle dos atuadores e pela aquisição de imagens, respectivamente.

É importante destacar que a abordagem utilizada se baseou nos algoritmos clássicos de visão computacional, em contraste com a tendência atual de se utilizar modelos de aprendizado de máquina e aprendizado profundo para resolver problemas similares. Isso se deve ao fato de que o objeto a ser detectado neste trabalho, o cubo de Rubik, possui muita simetria e regularidade, o que possibilitou a combinação de técnicas de geometria computacional e de visão computacional e, dessa forma, foi possível implementar um algoritmo sem a necessidade de *frameworks* ou modelos de aprendizado de máquina.

Para projetos que possuam bastante simetria e regularidade, ou que as condições de iluminação e plano de fundo possam ser controladas, as abordagens clássicas podem levar a soluções eficazes e de grande qualidade. No entanto, caso o problema não apresente padrões e similaridades facilmente reconhecidas, a abordagem utilizando modelos de aprendizado de máquina pode resultar em soluções mais eficazes e robustas.

Ainda, por meio deste trabalho, espera-se inspirar estudantes e profissionais da área de tecnologia e engenharia, bem como qualquer pessoa interessada, a desenvolver soluções inovadoras de automação auxiliadas por técnicas de visão computacional. Projetos nesta área não apenas abordam desafios modernos na automação, mas também exigem uma abordagem interdisciplinar, como ilustrado neste trabalho, que integrou conceitos de engenharia como eletrônica, impressão 3D, microcontroladores, programação e redes de computadores, além de princípios científicos como mecânica, eletricidade e óptica. Com a construção do protótipo, demonstrou-se que esse conjunto de conhecimentos só pode ser plenamente explorado e compreendido por meio da prática, momento esse em que desafios reais surgem, a criatividade humana é posta à prova e uma compreensão mais sólida dos conceitos se torna possível.

# 6 Referências


Amazon. *Produto: Trilho de Guia Linear Mgn9h Guia Linear Rolamento Guia Transporte Guia Deslizante Estendido Guias de Movimento Linear de Aço*. 2024. Acesso em: 18 nov. 2024. Disponível em: <https://www.amazon.com.br/dp/B0D7S86PT8>. Citado na página 41.

Arduino Docs. *Arduino Documentation: SerialEvent Example*. Arduino, 2024. Acesso em: 15 nov. 2024. Disponível em: <https://docs.arduino.cc/built-in-examples/communication/SerialEvent/>. Citado na página 37.

BECKER, R. F.; RODRIGUES, L. F. A.; MOREIRA, H. R. Estudo e desenvolvimento de um robô para resolução do cubo de rubik. *Jornada 2015*, Muzambinho, MG, 2015. Citado na página 47.

CARDEAL, T. L. D. Solução automática do cubo de rubik (cubo mágico) usando o arduino. Goiás, Brasil, 2022. Citado 2 vezes nas páginas 26 e 37.

Casa da Robótica. *Módulo ESP32-CAM com Câmera OV2640 e Conversor USB-Serial CH340*. 2024. Acesso em: 09 nov. 2024. Disponível em: <https://www.casadarobotica.com/placas-embarcadas/esp/placas/modulo-esp32-cam-com-camera-ov2640-conversor-usb-serial-ch340>. Citado na página 29.

Circuit Digest. *PCA9685 Multiple Servo Control Using Arduino*. 2024. Acesso em: 5 nov. 2024. Disponível em: <https://circuitdigest.com/microcontroller-projects/pca9685-multiple-servo-control-using-arduino>. Citado na página 28.

Cults3D. *Housing for ESP32-CAM*. 2024. Acesso em: 16 nov. 2024. Disponível em: <https://cults3d.com/en/3d-model/gadget/housing-for-esp32-cam>. Citado na página 43.

Espressif Systems. *Arduino-ESP32: Installing*. [S.l.], 2024. Acesso em: 30 out. 2024. Disponível em: <https://docs.espressif.com/projects/arduino-esp32/en/latest/installing.html>. Citado 3 vezes nas páginas 16, 29 e 54.

GANCUBE. *GAN Speed Cube Robot*. 2024. Acesso em: 1 dez. 2024. Disponível em: <https://www.gancube.com/products/gan-speed-cube-robot>. Citado na página 47.

GONZALEZ, R. C.; WOODS, R. E. *Processamento Digital de Imagens*. 3. ed. [S.l.]: Pearson and Artmed, 2010. ISBN 9788576054016. Citado na página 21.

Instructables. *Rubik Cube Solver Robot*. 2024. Acesso em: 16 nov. 2024. Disponível em: <https://www.instructables.com/Rubik-Cube-Solver-Robot/>. Citado na página 42.

Instructables. *Rubik's Cube Solver*. 2024. Acesso em: 1 dez. 2024. Disponível em: <https://www.instructables.com/Rubiks-Cube-Solver/>. Citado 2 vezes nas páginas 46 e 47.

JOKINEN, L. Solving a rubik's cube with computer vision. 2024. Citado na página 25.

Just4Fun Electronics. *Arduino Uno Compatible ATmega328P Development Board*. 2024. Acesso em: 05 nov. 2024. Disponível em: <https://www.just4funelectronics.com/product-page/arduino-uno-compatible-atmega328p-development-board>. Citado na página 27.

KARPATHY, A. *Extracting sticker colors on Rubik's Cube*. [S.l.], 2009. Acesso em: 02 nov. 2024. Disponível em: <https://github.com/zhangyuan324/rubiks-cube-tracker-Andrej-Karpathy/blob/master/525report.pdf>. Citado 2 vezes nas páginas 24 e 30.

LIMA, G. V. T. *Solução Automática do Cubo de Rubik usando Visão Computacional*. Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Computação) — Universidade Federal de Goiás, Goiânia, Brasil, 2024. Orientador: Prof. Dr. Adriano César Santana. Citado na página 47.

Maker Hero. *Servo TowerPro MG995 Metálico*. 2024. Acesso em: 5 nov. 2024. Disponível em: <https://www.makerhero.com/produto/servo-towerpro-mg995-metalico/>. Citado na página 28.

MARK, d. B. et al. *Computational geometry algorithms and applications*. [S.l.]: Spinger, 2008. Citado na página 24.

MIT. *The Mathematics of the Rubik's Cube*. 2011. Acesso em: 30 out. 2024. Disponível em: <https://web.mit.edu/sp.268/www/rubik.pdf>. Citado na página 18.

NAYAR, S. *First Principles of Computer Vision*. 2024. Acesso em: 01 nov. 2024. Disponível em: <https://fpcv.cs.columbia.edu/>. Citado na página 19.

NETO, E. C. et al. Development control parking access using techniques digital image processing and applied computational intelligence. *IEEE Latin America Transactions*, IEEE, v. 13, n. 1, p. 272–276, 2015. Citado na página 20.

OpenCV. *Optical Flow*. 2024. Acesso em: 02 nov. 2024. Disponível em: <https://docs.opencv.org/4.x/d4/dee/tutorial_optical_flow.html>. Citado 2 vezes nas páginas 23 e 33.

Ponto da Eletrônica. *Adafruit 16 Canais PWM Driver Servo de 12Bits Interface I2C PCA9685*. 2024. Acesso em: 5 nov. 2024. Disponível em: <https://www.pontodaeletronica.com.br/adafruit-16-canais-pwm-driver-servo-de-12bits-interface-i2c-pca9685.html>. Citado na página 28.

Python Package Index. *Kociemba: Python Package for Solving Rubik's Cube Using Two-Phase Algorithm*. 2024. Acesso em: 15 nov. 2024. Disponível em: <https://pypi.org/project/kociemba/>. Citado na página 37.

Raspberry Pi. *Cubotino: The Rubik's Cube-Solving Robot*. 2024. Citado 2 vezes nas páginas 46 e 47.

Robocore. *Cabo Extensor DC P4 Fêmea*. 2024. Acesso em: 18 nov. 2024. Disponível em: <https://www.robocore.net/conector/cabo-extensor-dc-p4-femea/>. Citado na página 41.

ROKICKI, T. Twenty-five moves suffice for rubik's cube. *arXiv preprint arXiv:0803.3435*, 2008. Disponível em: <https://arxiv.org/abs/0803.3435>. Citado na página 19.

SAALFELD, A. It doesn't make me nearly as cross: Some advantages of the point-vector representation of line segments in automated cartography. *International Journal of Geographical Information Systems*, Taylor & Francis, v. 1, n. 4, p. 379–386, 1987. Citado na página 31.

Speedsolving Wiki. *Western Color Scheme*. 2024. Acesso em: 30 out. 2024. Disponível em: <https://speedsolving.fandom.com/wiki/Western_Color_Scheme>. Citado 2 vezes nas páginas 17 e 18.

SZELISKI, R. *Computer vision: algorithms and applications*. [S.l.]: Springer Nature, 2022. Citado na página 14.

Tower Pro. *MG995 Servo*. 2024. <https://towerpro.com.tw/product/mg995/>. Acesso em: 18 nov. 2024. Citado na página 28.

WEN, C.; GAIESKI, F. K. *Robô Para Solução do Cubo de Rubik*. Brasil, 2020. Trabalho de Conclusão de Curso, Orientador: Jun Okamoto Jr. Citado 2 vezes nas páginas 46 e 47.

World Cube Association. *Regulamentos (tradução para o português brasileiro)*. 2024. Acesso em: 30 out. 2024. Disponível em: <https://www.worldcubeassociation.org/regulations/translations/portuguese-brazilian/>. Citado na página 18.

# Apêndices

# APÊNDICE A – Código utilizado

O código utilizado para a programação dos microcontroladores, implementação do algoritmo de visão computacional, resolução do cubo utilizando a biblioteca Kociemba e comunicação serial com o Arduino está disponível na plataforma GitHub, no seguinte link: <https://github.com/gustavotaveira/cube-solver>.

## A.1 Aplicação Python: bibliotecas e variáveis principais

Analisando detalhadamente o código-fonte em Python que implementa o algoritmo de visão computacional, observamos que ele faz uso de um conjunto de bibliotecas essenciais para a captura, detecção e processamento das imagens do cubo de Rubik, além de interagir com o Arduino. A biblioteca *cv2*, por exemplo, é responsável pela captura de frames enviados pelo ESP32-CAM e pela aplicação de algoritmos como a transformada de Hough para detecção de linhas e o fluxo óptico Lucas-Kanade para rastreamento de pontos na imagem. Já a biblioteca *math* é usada para cálculos geométricos necessários para análise dos contornos do cubo, enquanto o numpy facilita a manipulação matricial das imagens. O algoritmo de resolução do cubo é implementado pela biblioteca Kociemba, e as operações de tempo e execução assíncrona são gerenciadas pelas bibliotecas *time* e *threading*, respectivamente. A comunicação serial entre o programa e o Arduino é estabelecida através da biblioteca serial, garantindo uma troca eficiente de dados com uma taxa de transmissão de 115200 bits/s.

A função *main()* dá início à execução do programa, configurando variáveis essenciais para o seu correto funcionamento. A variável *capture* gerencia a entrada contínua de frames da câmera, enquanto as variáveis *W* e *H* são usadas para redimensionar as imagens, otimizando o processamento. A detecção das faces do cubo é controlada por parâmetros como o *THR*, que define o número mínimo de interseções no espaço de Hough para que um segmento de reta seja detectado pela Transformada de Hough, e *dects*, que determina o número ideal de segmentos detectados na imagem. O rastreamento contínuo é monitorado pelas variáveis *succ* e *tracking*, que indicam o sucesso das detecções e o modo de rastreamento do programa, respectivamente. O processamento das cores é gerenciado pelas variáveis *colors* e *assigned*, enquanto a comunicação serial com o Arduino é mantida pelo objeto *SerialArduino*, que permanece ativo durante toda a execução do programa, garantindo a interação contínua entre a aplicação Python e o Arduino. Essas variáveis são fundamentais para o funcionamento do programa e atuam como parâmetros ajustáveis que determinam o comportamento da solução de visão computacional. O IDE Visual Studio Code foi utilizado durante a implementação do programa, por ser uma ferramenta gratuita e que atendeu as necessidades de programação do projeto, proporcionando um ambiente organizado para edição e execução do código Python, e também a visualização de logs.

## A.2 Aplicações C++: microcontroladores

A programação dos microcontroladores Arduino e ESP32-CAM foi realizada no ambiente Arduino IDE utilizando a linguagem de programação C++. O código para o ESP32-CAM foi obtido por meio de um exemplo chamado *CameraWebServer*, fornecido pelo plugin da fabricante (Espressif Systems, 2024), o qual pode ser instalado no Arduino IDE. Ao acessar o código, duas configurações precisam ser feitas. A primeira refere-se à escolha do modelo de câmera. No caso do ESP32-CAM utilizado neste projeto, o modelo a ser selecionado é o *CAMERA_MODEL_AI_THINKER*. Deve-se remover o comentário da linha de código referente a esse modelo, em relação aos outros disponíveis, sendo ele definido por meio da diretiva *#define*. A segunda configuração necessária é a atribuição das credenciais da rede Wi-Fi à qual o ESP32-CAM irá se conectar, permitindo que ele atue como um servidor web e forneça imagens para clientes via streaming, utilizando o protocolo HTTP. As variáveis a serem alteradas são *ssid* e *password*. Na variável *ssid*, deve-se incluir o nome da rede, e na variável *password*, a senha correspondente. Após compilar e carregar o programa no ESP32-CAM, é possível acessá-lo por meio de um navegador, digitando o IP fornecido na barra de endereços. Esse IP pode ser visualizado no monitor serial, porém nesse caso, é necessário resetar o ESP32-CAM, pressionando o botão de reset no microcontrolador e, em seguida, acessar o monitor serial. Dessa forma, o IP será exibido e o acesso poderá ser realizado.

O código utilizado para o Arduino Uno foi implementado para possibilitar o envio de comandos aos atuadores, em conjunto com a placa PWM PCA9685. O uso da biblioteca *Adafruit_PWMServoDriver*, por meio da variável *board1*, permite o controle dos servos conectados ao PCA9685 utilizando o protocolo I2C, ajustando os ângulos de movimento por meio de PWM. São declaradas variáveis que representam os servomotores e seus movimentos de translação ou rotação em cada face lateral do cubo de Rubik. Essas variáveis recebem um valor inteiro que indica o conjunto de pinos do PCA9685 ao qual o servomotor está conectado. As variáveis *SERVOMIN* e *SERVOMAX* definem os limites do movimento dos servos, enquanto funções como *frontHorario()* e *rightHorario()* são responsáveis pela execução dos movimentos do cubo, simulando as rotações necessárias para sua resolução.

Além disso, a função *executaResolucao()* é responsável pela execução dos comandos de resolução do cubo, recebendo a cadeia de caracteres com os movimentos necessários para a solução do cubo por meio da variável *inputString*. A resolução ocorre por meio de uma estrutura de repetição que itera sobre todos os movimentos recebidos. A cada iteração, o movimento identificado é armazenado na variável *avaliar*, e é verificado para saber se corresponde a algum dos movimentos implementados. Isso é feito por meio das estruturas condicionais do tipo *if* e *else if*, que são exemplos de processos básicos de tomada de decisão das máquinas. Uma vez que o movimento tenha sido identificado, o comando previamente implementado por meio de uma função é executado, e o servomotor realiza o movimento no cubo de Rubik. Esse processo é repetido a cada iteração até que não reste mais nenhum movimento a ser executado, momento

em que o cubo estará resolvido. Nesse ponto, a função *soltarCubo()* é executada, liberando o cubo para novas tentativas de resolução.

A função *serialEvent()* é predefinida no ambiente de código Arduino IDE e foi sobrescrita para funcionar por meio de comunicação assíncrona, ouvindo a porta serial e identificando eventos de recebimento de dados. Quando isso ocorre, significa que a aplicação Python enviou os movimentos de resolução do cubo. Assim, a variável *inputString* é preenchida, e a variável *stringComplete* é marcada como verdadeira, indicando que os movimentos foram recebidos com sucesso. Na próxima execução da função *loop()*, a execução da resolução pode ser iniciada. O método *agarrarCubo()* é então executado para capturar o cubo, que deve ser posicionado adequadamente no protótipo pelo usuário (com a face verde voltada para o servomotor de rotação da face F e a face branca do cubo para cima). Em seguida, o processo de resolução é iniciado por meio da função *executaResolucao()*. Originalmente, a função *serialEvent()* é chamada automaticamente antes da função *loop()* durante a execução do programa no Arduino. No entanto, ela não possui implementação definida por padrão, ficando a cargo do programador declarar e implementar essa função em seu código para tirar proveito da funcionalidade de comunicação assíncrona, baseada em eventos de recebimento de dados na porta serial.