UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS
DEPARTAMENTO DE ENGENHARIA MECÂNICA

LUCAS ARNAUD DE ARAÚJO

DESENVOLVIMENTO DE UM ROBÔ SOLUCIONADOR DO CUBO DE RUBIK COM USO DE MOTORES DE PASSO E SENSORES DE COR

Recife

2024

LUCAS ARNAUD DE ARAÚJO

DESENVOLVIMENTO DE UM ROBÔ SOLUCIONADOR DO CUBO DE RUBIK COM USO DE MOTORES DE PASSO E SENSORES DE COR

Trabalho de Conclusão de Curso submetido ao Departamento de Engenharia Mecânica da Universidade Federal de Pernambuco como requisito parcial para obtenção do título de Bacharel em Engenharia Mecânica.

Orientador: Prof. Dr. José Rodrigues de Oliveira Neto

Recife

2024

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Araújo, Lucas Arnaud de.
Desenvolvimento de um robô solucionador do cubo de Rubik com uso de motores de passo e sensores de cor / Lucas Arnaud de Araújo. - Recife, 2024.
101 p. : il., tab.

Orientador(a): José Rodrigues de Oliveira Neto
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, Engenharia Mecânica - Bacharelado, 2024.
Inclui referências, apêndices.

1. Robótica. 2. Cubo de Rubik. 3. Motores de passo. 4. Sensores de cor. 5. Algoritmos. I. Neto, José Rodrigues de Oliveira. (Orientação). II. Título.

620 CDD (22.ed.)

LUCAS ARNAUD DE ARAÚJO

DESENVOLVIMENTO DE UM ROBÔ SOLUCIONADOR DO CUBO DE RUBIK COM USO DE MOTORES DE PASSO E SENSORES DE COR

Trabalho de Conclusão de Curso submetido ao Departamento de Engenharia Mecânica da Universidade Federal de Pernambuco como requisito parcial para obtenção do título de Bacharel em Engenharia Mecânica.

Aprovado em: 15 de Outubro de 2024.

BANCA EXAMINADORA

_____
Prof. José Rodrigues de Oliveira Neto (Orientador)
Universidade Federal de Pernambuco

_____
Prof. Adrien Joan Sylvain Durand Petiteville (Examinador Interno)
Universidade Federal de Pernambuco

_____
Prof. Vítor de Andrade Coutinho (Examinador Externo)
Universidade Federal Rural de Pernambuco

AGRADECIMENTOS

Ao concluir este trabalho, o sentimento é de profunda gratidão. Em primeiro lugar, meu coração se volta a Deus, pois, sem Ele, nada posso fazer. Tudo é graça de Deus e tudo aqui realizado foi feito pela Sua benção, no seu perfeito amor e infinita misericórdia, que iluminou o caminho e me alicerçou com as melhores pessoas para superar os desafios. Agradeço aos meus pais, João e Liana, em que devo tudo que sou hoje, e a minha irmã gêmea, Luana, que é minha grande parceira. Eles três são minha base, minha fonte de apoio e acompanharam de perto todos os desafios e as renúncias para realização desse projeto.

Lembro também, com gratidão, do meu irmão Anderson, pelas suas palavras de encorajamento e suas orientações em relação à impressão 3D, e do meu irmão Arthur, que sempre me incentivou neste projeto. Da mesma forma, manifesto meu eterno agradecimento a meu amigo Thiago, que, num gesto de enorme generosidade, me emprestou sua impressora 3D por mais de seis meses, para que este projeto pudesse acontecer. Igualmente, expresso meu profundo obrigado a meus amigos Rebeca e Heitor, cuja parceria e amizade foram fundamentais para superar as dificuldades, de provas e trabalhos, ao longo de todas as cadeiras do curso. Da mesma forma, agradeço a todos os meus amigos do Bonde CDU e do Kayrós, que por várias vezes, me ouviram falar das dificuldades do projeto, me incentivaram e torceram para que tudo desse certo. Ainda que não seja possível nomear a todos, expresso minha gratidão por cada amigo e familiar, que me apoiaram nessa trajetória. Além disso, agradeço aos colegas que construí na EIXO Consultoria e na Stellantis, que foram fundamentais para o meu desenvolvimento pessoal e profissional.

Faço também um agradecimento especial ao meu professor orientador, Dr. José Rodrigues de Oliveira Neto, que acompanhou toda a trajetória até aqui. Desde a escolha do tema, passando pelo pré-projeto de TCC, depois, a primeira versão construída deste robô, que foi bastante difícil, até chegarmos agora, no final, com o projeto consolidado. Muito obrigado por todas as suas orientações, suporte constante e paciência comigo.

Por fim, digo que levo deste trabalho, não apenas o conhecimento adquirido, mas também os ensinamentos de amizade que encontrei neste caminho. Este trabalho é resultado de uma caminhada de muitas lutas, mas acima de tudo, de muito aprendizado e de crescimento. Com ele, aprendi mais do que sobre robôs, aprendi principalmente que, embora escolhamos grandes desafios na vida, em cada passo, Deus nos ilumina e nos oferece as melhores pessoas para vencê-los.

Obrigado a todos.

RESUMO

Este trabalho apresenta o desenvolvimento de um robô solucionador do cubo de Rubik, focando em velocidade de resolução e simplicidade da leitura das cores do cubo. Sendo assim, o robô foi projetado com o objetivo de identificar as cores das peças do cubo usando sensores de cor e, com base nesses dados, aplicar um algoritmo eficiente para calcular e executar a sequência de movimentos necessária, com uso de motores de passo, para solucionar o quebra-cabeça. Para esse fim, foram abordados aspectos específicos de projeto mecânico, eletrônica, programação e algoritmos. Com isso, o robô desenvolvido utiliza seis motores de passo conectados aos eixos centrais do cubo, permitindo a rotação ordenada das faces, para realização da leitura e da resolução do cubo. Por outro lado, a leitura das cores é realizada com dois sensores de cor, a partir de um método próprio que realiza a leitura dos 48 quadrados com 60 movimentos das faces. Além disso, se implementou o algoritmo k-ésimo vizinho mais próximo (KNN, do inglês *k-nearest neighbors*) para classificar em cores os valores capturados pelos sensores e também um código para calcular a precisão das leituras automaticamente a partir de um embaralhamento aleatório empregado pelo robô no cubo. Ademais, utilizou-se uma implementação em código do algoritmo de Kociemba para encontrar a solução do cubo com no máximo 20 movimentos. Além da construção do robô, foram explorados conceitos importantes no uso de microcontroladores e de transferências de dados para integrar os sensores e os motores com um computador e viabilizar a leitura e a resolução do cubo. O projeto foi validado em testes para diferentes padrões do cubo e demonstrou sua capacidade de operar de forma eficaz. Nessas avaliações, o robô alcançou uma mediana de 39 leituras corretas de 48 e acertou, ao todo, 1375 das 1776 leituras (77,42%) de cores. Acerca da velocidade, foi alcançado um tempo mínimo de resolução de 3,3 segundos e um tempo médio de 3,6 segundos, um desempenho comparável a projetos de referência na área.

Palavras-chave: Robótica. Cubo de Rubik. Motores de passo. Sensores de cor. Algoritmos.

ABSTRACT

This work presents the development of a Rubik's cube-solving robot, focusing on solving speed and simplicity in color detection. The robot was designed to identify the colors of the cube pieces using color sensors and, based on this data, apply an efficient algorithm to calculate and execute the necessary sequence of moves using stepper motors to solve the puzzle. To this end, specific aspects of mechanical design, electronics, programming and algorithms were adressed. The developed robot uses six stepper motors connected to the cube's central axes, allowing orderly face rotations for reading and solving. Color detection is performed with two color sensors using a custom method that reads all 48 squares with 60 face movements. Alternatively, the k-nearest neighbors (KNN) algorithm was implemented to classify the colors detected by the sensors, along with code to calculate reading accuracy based on a random shuffle performed by the robot. Additionally, the Kociemba algorithm was implemented in code to find the solution with a maximum of 20 moves. Alongside the construction of the robot, important concepts were explored in the use of microcontrollers and data transfers to integrate sensors and motors with a computer, enabling the reading and solving of the cube. The project was validated through tests with different cube patterns and demonstrated its effectiveness. It achieved a median of 39 correct readings out of 48 and accurately identified 1375 of 1776 color readings (77.42%). Regarding speed, the robot achieved a minimum solving time of 3.3 seconds and an average time of 3.6 seconds, a performance comparable to reference projects in the field.

Keywords: Robotics. Rubik's cube. Stepper motors. Color sensors. Algorithms.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

IoT	Internet das coisas (do inglês, *Internet of things*)
AI	Inteligência artificial (do inglês, *Artificial intelligence*)
KNN	K-ésimo vizinho mais próximo (do inglês, *k-nearest neighbor*)
IDA*	Algoritmo de aprofundamento iterativo A* com função heurística (do inglês, *Iterative Deeping A**
MIT	Intituto de Tecnologia de Massachusetts (do inglês, *Massachusetts Intitute of Techlogy*)
AC	Corrente alternada (do inglês, *Alternating Current*)
DC	Corrente contínua (do inglês, *Direct Current*)
FPGA	Arranjo de porta programável em campo (do inglês, *Field Programmable Gate Array*)
LDR	Resistor dependente de luz (do inglês, *Light dependent resitor*)
RGB	Vermelho, verde e azul (do inglês, *Red, Green and Blue*)
LED	Diodo emissor de luz (do inglês, *Light-Emiting Diode*)
PWM	Modulação por Largura de Pulso (do inglês, *Pulse Width Modulation*)
IDE	Ambiente de desenvolvimento integrado (do inglês, *Integrated Development Environment*)
VDD	Tensão em corrente contínua
GND	Malha de terra
GPLv3	Licença Pública Geral versão 3 (do inglês, *Genereal Public License version 3*
CNC	Controle Numérico Computadorizado

LISTA DE SÍMBOLOS

H	Subconjunto H, que corresponde a reunião de 19,5 bilhões de possíveis posições do cubo de Rubik que permitem a resolução dele com no máximo 18 movimentos e, ao mesmo tempo, possui todas as posições alcançáveis com no máximo 12 movimentos a partir de qualquer posição do cubo
A_{10}	Conjunto dos 10 movimentos que mantém o cubo de Rubik no subconjunto H e que são úteis para resolver o cubo
k	Número de vizinhos mais próximos do algoritmo KNN
A_p	Ângulo de passo
N_p	Número de passos

SUMÁRIO

1 INTRODUÇÃO

A robótica é uma das áreas mais dinâmicas e revolucionárias da engenharia moderna, ocupando um papel crucial em diversas indústrias no contexto da 4ª revolução industrial (LEE; ENRIQUEZ; LEE, 2022). A criação dos primeiros robôs industriais voltados para a manufatura se iniciou na década de 1960. Desde então, o campo cresceu rapidamente (MATARIĆ, 2014). Com isso, a demanda e as aplicações dos robôs se tornaram cada vez mais complexas e abrangentes, atingindo setores como a saúde (MORGAN et al., 2022), o transporte (LOZANO-PEREZ, 2012), a exploração espacial(YOSHIDA; WILCOX, 2008) e o entretenimento (BOGUE, 2022).

Entre as muitas aplicações da robótica, os robôs manipuladores se destacam por sua capacidade de realizar tarefas complexas com alta precisão, como montagem de componentes, manipulação de materiais e até mesmo intervenções médicas (WANG; TAO; LIU, 2018). Com o desenvolvimento rápido da indústria robótica, em escopo e em escala, surgem cada vez mais os "robôs inteligentes" que fazem o uso de tecnologias como inteligência artificial (AI, do inglês *artifical itelligence*) e Internet das Coisas (IoT, do inglês *Internet of things*) (LEE; ENRIQUEZ; LEE, 2022). Além disso, esses robôs, equipados com sensores e sistemas de controle sofisticados, são essenciais para automatizar processos que exigem movimentos repetitivos e precisos, representando um avanço significativo em produtividade e segurança (KEMP; EDSINGER; TORRES-JARA, 2007).

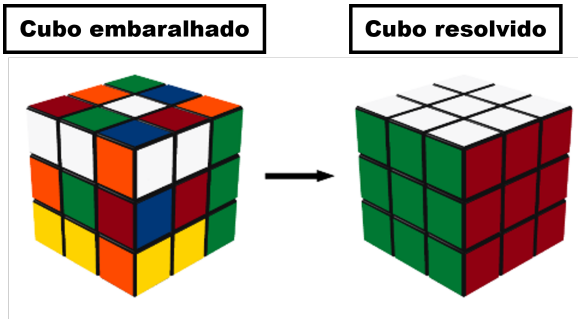
Dentro desse contexto, os robôs solucionadores do cubo de Rubik emergem como uma aplicação de destaque no seguimento dos robôs manipuladores (DAN; HARJA; NAŞCU, 2021), combinando o uso de ferramentas de projeto mecânico, eletrônica e programação (WEN; GAIESKI, 2020). Esses robôs são projetados para resolver o cubo de Rubik, um dos quebra-cabeças tridimensionais mais desafiadores e famosos de todos os tempos (TOSHNIWAL; GOLHAR, 2019). Para isso, eles alteram o cubo de um estado embaralhado aleatoriamente ao seu estado original e solucionado, em que cada lado possui uma única cor (SOUZA, 2011), conforme mostrado na Figura 1.

Nesse sentido, de forma particular, os robôs solucionadores do cubo de Rubik promovem a integração das atividades de: construção de design mecânico, coleta e transferência de dados, manipulação de atuadores e controle automático. Especificamente, cada robô pode utilizar de diferentes tecnologias para desenvolver tais domínios, conforme apresentado na Figura 2. Mesmo assim, em geral, eles se propõem a cumprir os mesmos objetivos. Isto é, inicialmente, identificar o estado do cubo embaralhado, e, em seguida, a partir de algoritmos computacionais avançados, definir e implementar a sequência de movimentos das faces do cubo para solucionar o quebra cabeça $3 \times 3 \times 3$ (WEN; GAIESKI, 2020).

Sendo assim, esses robôs, para a etapa de resolução, requerem quase todas as habilidades que são essenciais em um robô de serviço, como: controle de força, manipulação eficiente e elaboração de um plano de movimentos (KASPRZAK; SZYNKIEWICZ; CZAJKA, 2006). Além

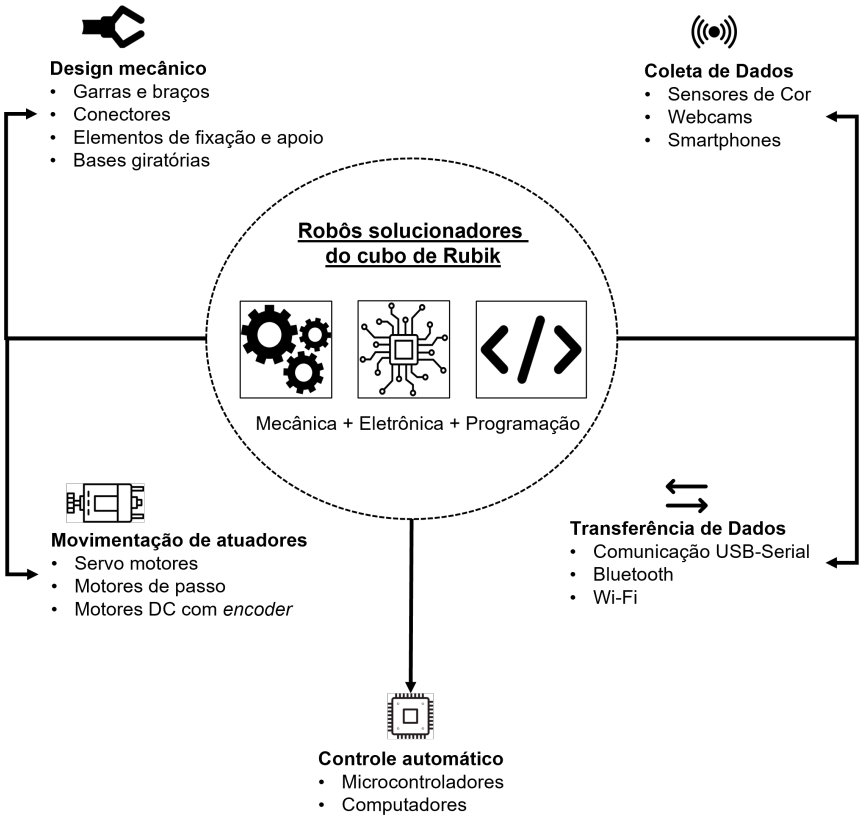
disso, no que se refere a coleta de dados (ou etapa de leitura do cubo), os robôs que utilizam de sensores de identificação de cores, especificamente, implementam uma tecnologia que possui aplicabilidades importantes no ambiente industrial. Como por exemplo, no aspecto de verificação de qualidade, para inspecionar as cores e tonalidades dos produtos fabricados (ANUPAMA et al., 2012).

Figura 1 – Comparação do Cubo embaralhado x Cubo resolvido.



Fonte: Adaptado de Souza (2011).

Figura 2 – Exemplos de tecnologias utilizadas em cada frente para arquitetura dos robôs solucionadores do cubo de Rubik.



Fonte: Adaptado de Wen e Gaieski (2020).

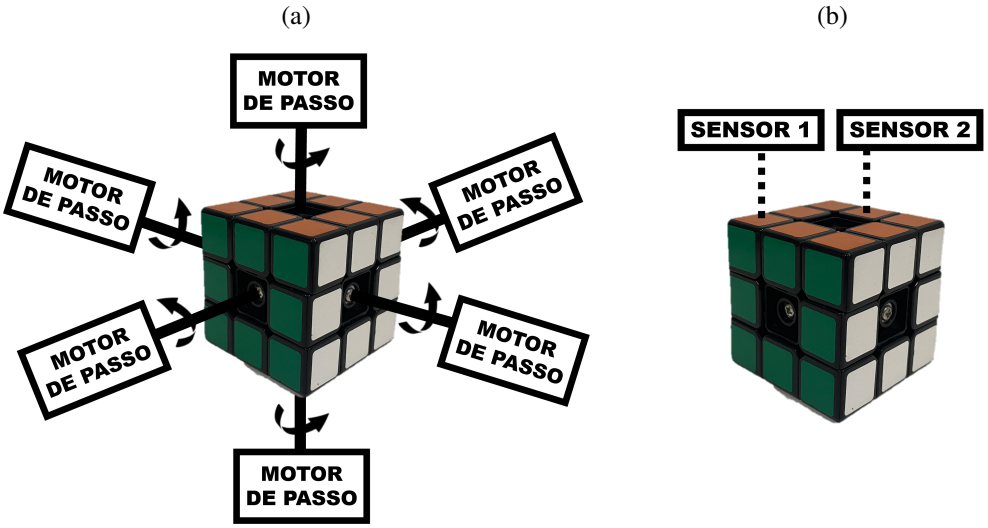
A partir desse cenário, este trabalho propõe o desenvolvimento de um robô solucionador do cubo de Rubik que utiliza tecnologias específicas selecionadas da Figura 2. Para isso, são defi-

nidas premissas que impõem restrições conceituais ao robô e, assim, definem suas características. As premissas são:

1. Para a resolução, o robô deve utilizar de um conceito que promova menor número de movimentos, isto é, sem movimentos auxiliares de giro no cubo, realizando apenas os giros de 90° no sentido horário e anti-horário e de 180° de todas as seis faces do cubo. E para tal, deve fazer uso de motores de passo com seus eixos acoplados nos eixos centrais do cubo, conforme a Figura 3a;

2. Para a leitura, o robô deve utilizar de um conceito que busque mais simplicidade técnica, isto é, sem dispor de processamento de imagem e webcams, mas sim, do uso de sensores de cor para as medições pela face superior do cubo, conforme a Figura 3b.

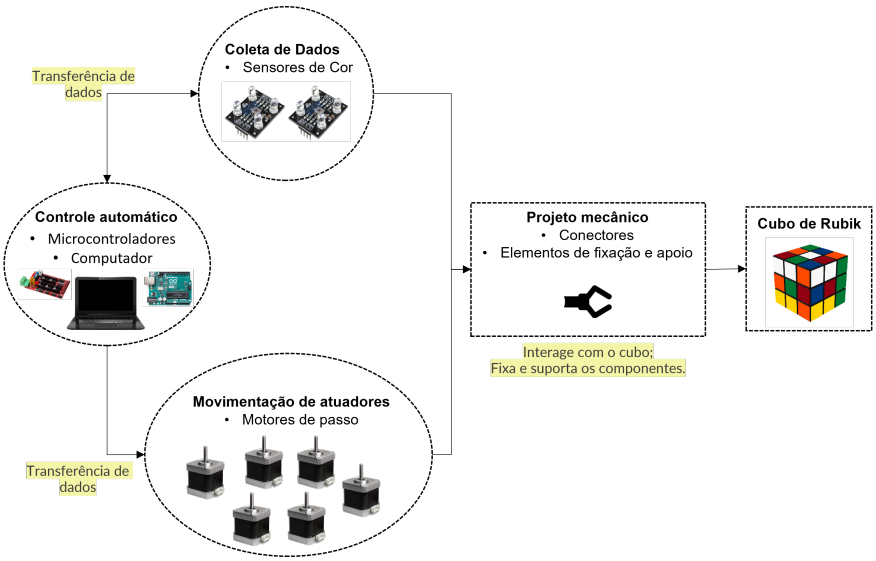
Figura 3 – Visualizações das premissas construtivas para o robô: (a) premissa 1, (b) premissa 2.



Fonte: Autoria própria.

Diante das duas premissas, é realizada a concepção do robô envolvendo: no design mecânico, o uso de estruturas de suporte, sustentação, fixação e conectores; no campo dos atuadores, o emprego de motores de passo e, na coleta de dados, o uso de sensores de cor. Além disso, o controle automático é feito a partir de um computador, que utiliza um código escrito na linguagem de programação Python e um algoritmo solucionador, a fim de realizar as transferências de dados com microcontroladores conectados aos dispositivos para controlar o acionamento dos atuadores e a leitura dos sensores de forma ordenada, possibilitando assim, as execuções do embaralhamento, leitura, cálculo da assertividade da leitura e resolução do cubo. A Figura 4 apresenta o diagrama dos componentes do robô e suas interações.

Figura 4 – Diagrama dos componentes do robô e suas interações.



Fonte: Autoria própria.

1.1 OBJETIVOS

O objetivo geral deste trabalho é desenvolver um robô solucionador do cubo de Rubik, visando a melhor combinação dos aspectos de velocidade de resolução e simplicidade na leitura, que realize a identificação das cores das peças e, em seguida, com uso de um algoritmo eficiente, encontre a solução de movimentos e os implemente no cubo para resolvê-lo.

Os objetivos específicos desta proposta são:

a) Projetar, fabricar e montar o projeto mecânico do robô, incluindo as estruturas de suporte, sustentação, fixação e conectores;

b) Projetar e levantar as especificações dos componentes apropriados ao *hardware* do robô, partindo da utilização de motores de passo como atuadores e sensores de cor como coletores de dados;

c) Selecionar os microcontroladores, plataformas e ferramentas capazes de realizar o controle dos atuadores e do sensores;

d) Selecionar a implementação em código do algoritmo solucionador a ser utilizada no robô capaz de fornecer a sequência de movimentos solucionadores para um estado embaralhado do cubo;

e) Implementar os códigos para controle dos atuadores e sensores, que promovam a transferência de dados entre os dispositivos, os microcontroladores e o computador;

f) Implementar o código capaz de medir a assertividade da leitura das cores das peças do cubo de forma automática, após um embaralhamento arbitrário;

g) Implementar o código final que, com uso do *hardware* selecionado e do projeto mecânico desenvolvido, coordene integradamente a realização das etapas de recebimento de uma sequência de embaralhamento, embaralhamento do cubo, leitura, cálculo da assertividade da leitura e resolução do cubo.

1.2 METODOLOGIA

A fim de alcançar os objetivos mencionados na Seção 1.1, o roteiro de execução do trabalho se divide nas etapas sumarizadas a seguir.

- ETAPA 1- *Projeto mecânico do robô:* será desenvolvido o conceito mecânico do robô, com bases nas premissas do projeto, se fará a modelagem na plataforma de nuvem Onshape, a fabricação das peças utilizando impressora 3D e montagem das estruturas com uso de bases e caixa de acrílico.
- ETAPA 2 - *Seleção do hardware:* nesta etapa, serão selecionados todos os dispositivos eletrônicos que atuarão em conjunto para funcionamento do robô em cada uma de suas tarefas.
- ETAPA 3 - *Montagem do hardware:* será feita a conexão dos componentes e a montagem deles na estrutura mecânica do robô.
- ETAPA 4 - *Projeto do firmware e software do robô:* será feita a escolha e a construção dos códigos que serão utilizados para promover o funcionamento do robô utilizando da plataforma Arduino, placas de desenvolvimento Arduino e shields compatíveis.
- ETAPA 5 - *Testes e validações:* serão realizados testes para validar o funcionamento do sistema completo do robô, incluindo a coleta e análise dos dados obtidos ao longo dos testes. Além disso, se buscará avaliar o robô e identificar as melhorias possíveis.

1.3 ESTRUTURA DO DOCUMENTO E CONTRIBUIÇÕES

O documento está estruturado da seguinte forma:

- No Capítulo 1 é apresentado a introdução ao tema, os objetivos geral e específicos deste projeto, a metodologia aplicada no desenvolvimento e a estrutura deste documento.
- No Capítulo 2 são discutidos temas importantes para construção deste projeto. Isto é, aspectos sobre a teoria do cubo de Rubik e do algoritmo solucionador de Kociemba e dos robôs solucionadores do cubo de Rubik que são referências a este trabalho. Em seguida, são apresentados conceitos acerca do algoritmo de classificação k-ésimo vizinho mais próximo (KNN, do inglês k - *nearest neighbors*), os quais são úteis neste projeto.

- No Capítulo 3 é descrito o projeto mecânico do robô. Desde a modelagem tridimensional das peças até a fabricação delas utilizando impressão 3D e as suas montagens para desenvolver a estrutura completa do robô.
- No Capítulo 4 é detalhada a seleção dos componentes do *hardware* do robô, com base em outros projetos de referência, que serão utilizados no circuito eletrônico.
- No Capítulo 5 são descritas as etapas para conexão dos dispositivos eletrônicos e também a montagem deles na estrutura mecânica do robô.
- No Capítulo 6 são apresentados os códigos utilizados e construídos para o desenvolvimento do *firmware* e *software* do robô. Assim, são descritos os fluxogramas desenvolvidos para implementar em código as tarefas do robô.
- No Capítulo 7 são descritos os resultados do projeto, incluindo o detalhamento em dados do nível de assertividade da leitura do cubo desenvolvido pelo robô e também sua assertividade e velocidade para resolução do cubo.
- No Capítulo 8 são revisitadas de forma sumária as principais contribuições do trabalho, elencados possíveis desdobramentos do projeto em trabalhos futuros.
- No Apêndice A são expostas as fichas técnicas das peças desenvolvidas neste projeto.
- No Apêndice B são apresentados os dez padrões de cubo utilizados para realização dos testes e validações do robô.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, é apresentada a fundamentação teórica de temas relevantes para este trabalho. Inicialmente, são abordados os aspectos gerais do cubo de Rubik, sua história, tipos de peças e notações dos movimentos, assim como sobre o funcionamento teórico do algoritmo solucionador utilizado: o algoritmo de Kociemba (ROKICKI, 2008; KOCIEMBA, s.d; KAUR, 2015). Em seguida, são mostradas referências de robôs solucionadores do cubo de Rubik, alguns comerciais e outros encontrados na literatura (WEN; GAIESKI, 2020; SWIEZY; KNOPF, 2017; WANGYUYYT, 2022; TSOY, 2016; KATZ, 2018; CUTHBERTSON, 2024; INDUSTRIES, s.d; RCR3D, s.d), que são norteadores técnicos e conceituais deste projeto. Além disso, são apresentados conceitos do algoritmo KNN (PACHECO, 2017; FUKUNAGA; NARENDRA, 1975; SACRAMENTO, 2023).

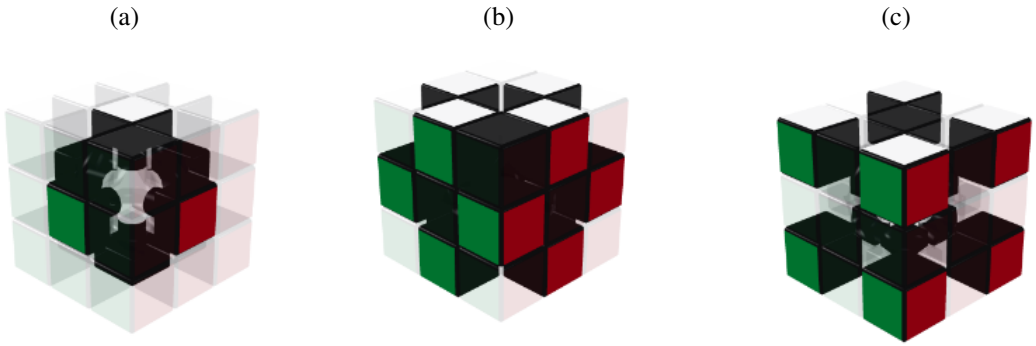
2.1 O CUBO DE RUBIK E O ALGORITMO DE KOCIEMBA

O cubo de Rubik foi inventado em 1974, pelo professor e arquiteto húngaro Erno Rubik. Ele é um quebra cabeça combinacional que possui uma simplicidade e, ao mesmo tempo, uma alta dificuldade que despertam a atração e o fascínio de diversas pessoas no mundo (SOUZA, 2011). Numa configuração $3 \times 3 \times 3$, em que cada lado possui uma única cor, o quebra cabeça tem como objetivo alterar o cubo de um estado embaralhado aleatoriamente ao seu estado solucionado (SOUZA, 2011), como mostra a Figura 1.

2.1.1 Tipos de peças e notações dos movimentos

De forma mais detalhada, o cubo possui 6 faces de cores diferentes divididas em 9 partes, completando 27 peças, 26 sendo cubos menores e o 27º o mecanismo central que sustenta todos (CEZARIO; MIURA, 2021). Além disso, o cubo mágico, como é conhecido o cubo de Rubik no Brasil, possui três tipos de peças: centros, meios e cantos. Aquelas com apenas uma cor, são os centros; com duas cores, são os meios e com três cores, são os cantos. Assim, as 26 peças cúbicas distribuem-se em: seis centros, doze meios e oito cantos (SILVA, 2015), conforme é mostrado na Figura 5.

Figura 5 – Tipos de peças no cubo de Rubik: (a) peças de centro, (b) de peças de meio e (c) peças de canto.



Fonte: Adaptado de Souza (2011).

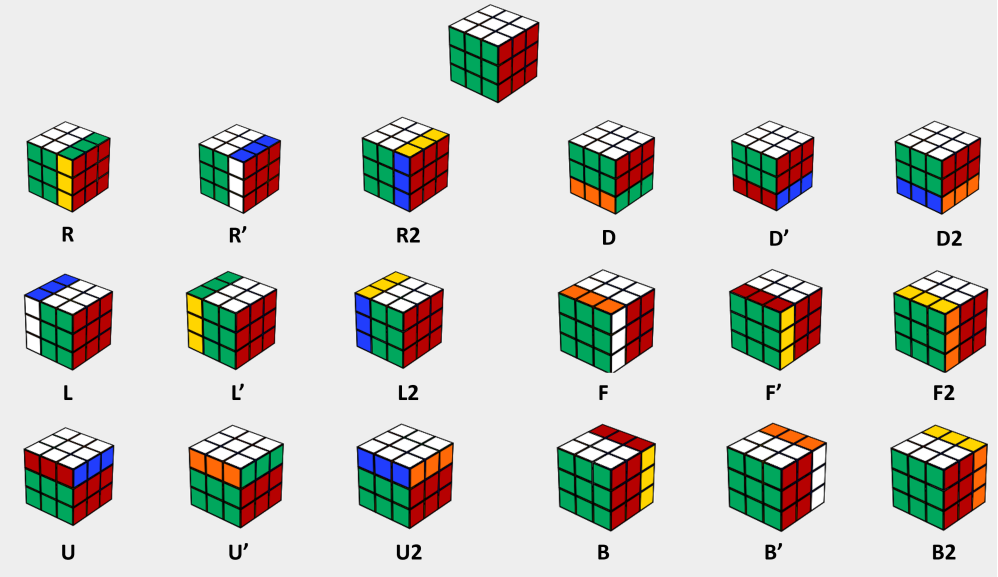
No seu formato tradicional, o cubo possui as faces nas cores: branca, amarela, verde, azul, vermelha e laranja. E, dado que os centros são parafusados, as faces adquirem pares de cores que são opostas e sempre fixas, conforme a Figura 6, no formato de: branca x amarela, azul x verde e vermelha x laranja (SILVA, 2015).

Nos aspectos de movimentação, há variadas possibilidades, desde a rotação de duas camadas simultaneamente até a rotação do cubo em seus eixos. No entanto, apenas os movimentos básicos recebem notações específicas e são considerados nos algoritmos solucionadores do cubo de Rubik. Tais movimentos básicos são aqueles realizados por cada face no sentido horário, anti-horário ou de giro duplo ($180°$), o que totaliza 18 movimentos. Para as notações deles, se utilizam os símbolos das letras: U, D, R, L, F e B, para as faces superior, inferior, direita, esquerda, frontal e traseira, respectivamente. As letras sozinhas representam os movimentos no sentido horário e, se adicionadas do apóstrofo (') ou do número dois (2), representa que o movimento é no sentido anti-horário ou de $180°$, respectivamente (CEZARIO; MIURA, 2021). Nesse sentido, a representação dos movimentos básicos se traduz conforme a Figura 7.

Figura 6 – Cores das faces e suas respectivas faces opostas.

Fonte: Adaptado de Silva (2015).

Figura 7 – Notações para os 18 movimentos básicos do cubo de Rubik.



Fonte: Adaptado de Silva (2015).

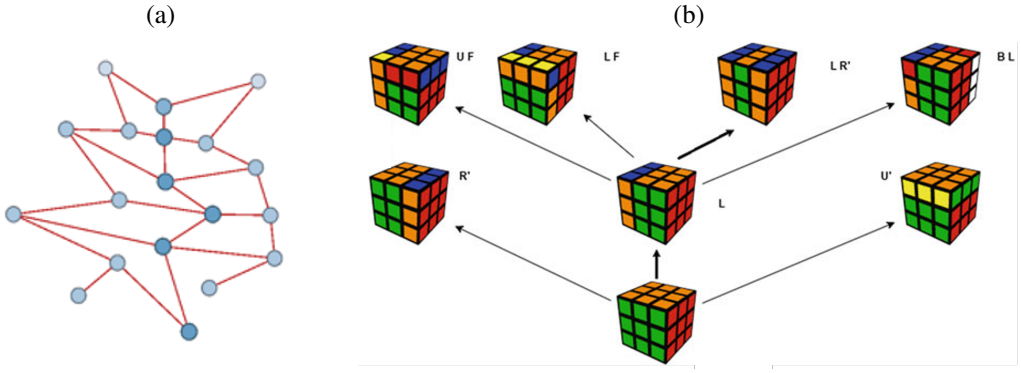
O cubo de Rubik possui 43.252.003.274.489.856.000 (quarenta e três quintilhões) possibilidades de combinações distintas. Esse valor é obtido a partir de algumas considerações de análise combinatória no cubo (KORF, 1997). Mesmo com esse número consideravelmente grande de combinações, diversos algoritmos foram criados para solucionar o cubo de Rubik, alguns mais simples e outros capazes de serem executados apenas por máquinas (KAUR, 2015).

Em 2010, Tomas Rokicki, Herbert Kociemba, Morley Davidson e John Dethridge por meio de uma elevada potência computacional, em parceria com a Google, obtiveram a prova de que toda e qualquer combinação do cubo pode ser resolvida com no máximo 20 movimentos básicos (ROKICKI et al., 2014), tal constatação ficou conhecido como “número de Deus”. Para a descoberta, ao longo dos anos, se utilizaram de variados algoritmos de busca relevantes (CHEN, 2022), especificamente, o algoritmo de Kociemba, que é apresentado a seguir.

2.1.2 Algoritmo de Kociemba

De forma particular, é possível entender que o cubo de Rubik pode ser modelado como uma problemática de grafo em que os nós representam um conjunto de todas as configurações possíveis e as arestas representam o conjunto de movimentos de uma configuração a outra (KHEMANI et al., 2018), conforme mostrado na Figura 8.

Figura 8 – Estados do cubo de Rubik em modelagem de grafos: (a) Modelo de um grafo, com nós e arestas , (b) cubo de Rubik numa visualização de grafos.



Fonte: (a) adaptado de Bitencourt (2016) e (b) adaptado de Khemani et al. (2018).

Utilizando a representação em um grafo de profundidade, é possível visualizar o número de nós (estados do cubo) na árvore de busca como uma função de profundidade, até a profundidade 18 (KORF, 1997), conforme a Tabela 1. Percebe-se nela, que com a execução de apenas 1 movimento (profundidade 1), existem 18 diferentes estados possíveis do cubo, os quais são os mesmos que a Figura 7 apresenta. Com a execução de dois movimentos (profundidade 2), existem 243 possíveis estados para o cubo, e assim sucessivamente (KORF, 1997).

Tabela 1 – Nós em uma árvore de busca como função da profundidade.

Depth	**Nodes**
1	18
2	243
3	3,240
4	43,254
5	577,368
6	7,706,988
7	102,876,480
8	1,373,243,544
9	18,330,699,168
10	244,686,773,808
11	3,266,193,870,720
12	43,598,688,377,184
13	581,975,750,199,168
14	7,768,485,393,179,328
15	103,697,388,221,736,960
16	1,384,201,395,738,071,424
17	18,476,969,736,848,122,368
18	246,639,261,965,462,754,048

Fonte: (KORF, 1997).

Dessa forma, dada a quantidade expressiva de nós, é pertinente o uso de um algoritmo de busca que forneça o caminho mais curto do nó de origem (cubo embaralhado) até o nó de objetivo (cubo resolvido) (CHEN, 2022).

Um dos mais apropriados é o algoritmo de Kociemba, desenvolvido pelo alemão Herbert Kociemba (ROKICKI, 2008), o qual representa uma versão aprimorada do algoritmo de Thistlethwaite. No seu algoritmo, Kociemba aplica uma implementação avançada com uso da busca por profundidade iterativa A* com uma função heurística, denominada IDA* (do inglês, *Iterative Deeping A**), que é capaz de, dividindo o problema em duas fases, encontrar a solução para qualquer estado do cubo (KAUR, 2015).

Para desenvolver o seu método, Kociemba considerou a seguinte situação: girando as faces de um cubo resolvido apenas com os movimentos U, D, R2, L2, F2 e B2 é possível criar um subconjunto de estados do cubo (KOCIEMBA, s.d). Este subconjunto é conhecido como subconjunto H e corresponde a reunião de 19,5 bilhões de possíveis posições do cubo que permitem a resolução dele com no máximo 18 movimentos (REID, 1995). Ao mesmo tempo, uma dessas posições possíveis podem ser alcançadas com no máximo 12 movimentos a partir de qualquer posição do cubo (ROKICKI, 2008). De forma complementar, o subconjunto H pode ser mantido pelo conjunto A_{10} de movimentos: U, U2, U', D, D2, D', R2, L2, F2, B2. E, com eles, é possível alcançar o cubo resolvido (ROKICKI, 2008).

Sendo assim, o algoritmo de Kociemba propõe resolver o cubo em duas fases: a fase 1, que tem como objetivo encontrar uma sequência de movimentos que leve o cubo de uma posição arbitrária até alguma posição pertencente ao subconjunto H. E a fase 2, que é responsável por definir a sequência de movimentos que leve o cubo de uma posição de H até a posição do cubo completamente resolvido utilizando o conjunto de movimentos A_{10} (ROKICKI, 2008).

2.2 ROBÔS SOLUCIONADORES DO CUBO DE RUBIK

De acordo com Wen e Gaieski (2020), a variedade de robôs solucionadores do cubo de Rubik pode ser categorizada em três grupos, conforme os seus objetivos:

1. Quebrar recordes: com alta velocidade de resolução;
2. Acessibilidade: que possuem facilidade de reprodução;
3. Produtos comerciais: com baixo custo, priorizando a estética e preço acessível.

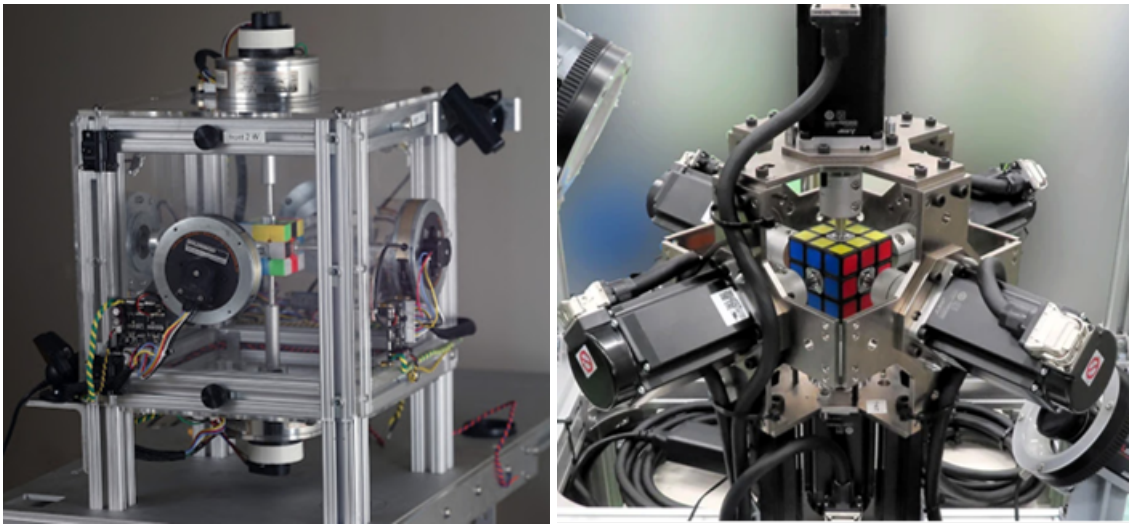
Com o foco no desempenho de velocidade de resolução, existe o robô desenvolvido por estudantes do MIT (do inglês, *Massachusetts Institute of Technology*), apresentado na Figura 9a, com o tempo de resolução de 0,380 segundos. Ele utiliza seis motores Kollmorgen ServoDisc série U9/N9[1] (com um *encoder* acoplado) para movimentar as faces, e realiza a leitura das cores com uso de duas câmeras PlayStation Eye[2] e um *software* de processamento de imagem (KATZ, 2018). No entanto, o atual recordista mundial é o robô TOKUFASTbot, desenvolvido

[1] <https://www.ebay.com/itm/141192172754>
[2] <https://en.wikipedia.org/wiki/PlayStation_Eye>

pela empresa Mitsubishi Electric, apresentado na Figura 9b, com o recorde de 0,305 segundos (SENDA, 2024). A equipe de engenharia da empresa utilizou motores avançados que permitem o giro de uma das seis faces do cubo 90° em 0,009 segundo. Além de reconhecer as cores com uso de câmeras com inteligência artificial (CUTHBERTSON, 2024).

Figura 9 – Exemplos de robôs solucionadores do cubo de Rubik construídos para quebrar recordes: (a) robô desenvolvido pelo MIT, antigo recordista mundial, com 0,380 segundos de tempo de resolução, (b) robô desenvolvido pela empresa Mitsubishi Eletric, atual recordista mundial, com 0,305 segundos de tempo de resolução.

(a) (b)



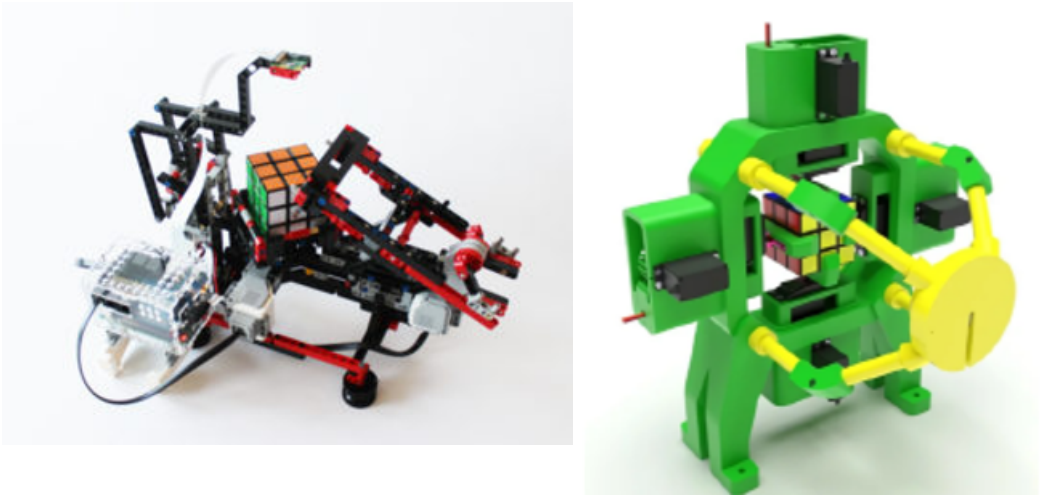
Fonte: (a) (KATZ, 2018) e (b) (CUTHBERTSON, 2024).

Pelo aspecto da acessibilidade, se destaca o projeto BricKuber, mostrado na Figura 10a, desenvolvido pela empresa Dexter Industries. Para essa solução, o robô utiliza apenas dois motores para todas as movimentações no cubo, junto com um kit LEGO Mindstorms EV3. Assim, o controle dos motores, é feito com uma Raspberry Pi e um kit BrickPi. Além disso, faz uso de uma câmera Raspberry Pi para tirar uma foto do cubo embaralho (INDUSTRIES, s.d).

De forma complementar, na questão comercial, se tem o robô Otvinta, apresentado na Figura 10b. Feito em estrutura de impressão 3D, o robô atua com oito servomotores tal que quatro deles rotacionam as faces laterais e os outros garantem o movimento linear dos efetuadores que se acoplam ao cubo (RCR3D, s.d).

Figura 10 – Exemplos de robôs solucionadores do cubo de Rubik construídos visando acessibilidade ou ser um produto comercial: (a) Robô Brickuber, exemplo de robô visando acessibilidade e (b) Robô Otvinta, exemplo de robô que visa ser um produto comercial.

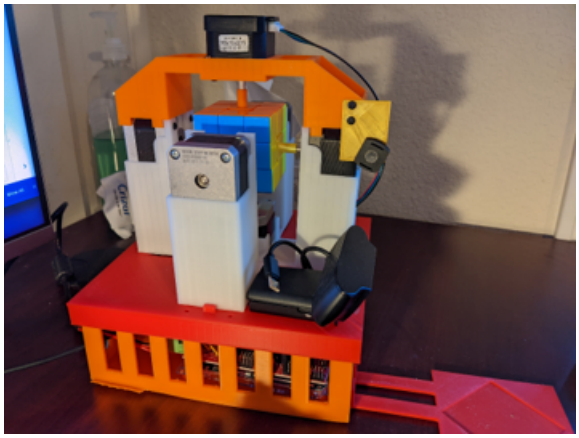
(a) (b)



Fonte: (a) (INDUSTRIES, s.d) e (b) (RCR3D, s.d).

Existem robôs que buscam conciliar os três fatores, isto é, possuem facilidade de reprodução, preços acessíveis nos componentes e velocidade de resolução interessante, é o caso da solução desenvolvida por Wangyuyyt (2022), apresentada na Figura 11. O robô é constituído de uma estrutura de peças impressas em impressora 3D e utiliza uma *webcam* e um código de processamento de imagem para realizar a leitura das faces do cubo. Para encontrar a solução do cubo embaralhado, ele faz uso da implementação em código do algoritmo de Kociemba desenvolvida por Tsoy (2022). E, para resolução, ele promove a integração dos componentes listados na Tabela 2, para, movimentando coordenadamente seis motores de passo, solucionar o cubo.

Figura 11 – Robô desenvolvido por Wangyuyyt (2022).



Fonte: (WANGYUYYT, 2022).

Tabela 2 – Lista de componentes do hardware do robô desenvolvido por Wangyuyyt (2022).

Componente	Quantidade
Arduino MEGA	1
Shield RAMPS 1.4	1
Motor de passo	6
Driver de motor de passo	6
Módulo de expansão para *driver* de motor de passo	1
Conversor 12V AC DC	1
Cabo DC	1

Fonte: Adaptado de Wangyuyyt (2022).

Todos os robôs elencados, utilizam câmeras para processamento de imagem e identificação do estado do cubo. No entanto, existem algumas soluções que realizam essa tarefa com uso de sensores de cor. É o caso do robô desenvolvido por Jacob Swiezy e Nathaniel Knopf, estudantes do MIT, mostrado na Figura 12. Nesta solução, eles utilizam seis motores de passo acoplados em cada uma das faces do cubo e que são controlados por uma placa FPGA (do inglês, *Field Programmable Gate Array*) denominada Nexys 4. Dessa forma, a placa realiza o controle dos motores para movimentar as faces para que cada um dos quadrados do cubo de Rubik possam ser lidos sequencialmente por dois sensores de cor (um que que realiza a leitura das peças de meio e o outro as peças de canto). Com isso, é identificado o estado do cubo embaralhado, e assim, com uso do algoritmo solucionador de Kociemba, é gerada a sequência de movimentos para a solução, que ao ser implementada pelos motores, resolve o quebra-cabeça (SWIEZY; KNOPF, 2017).

Figura 12 – Robô desenvolvido por Jacob Swiezy e Nathaniel Knopf no MIT.

Fonte: (SWIEZY; KNOPF, 2017).

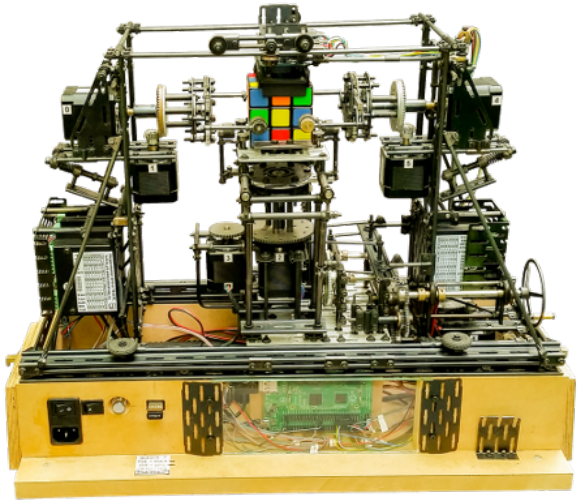
De acordo com Swiezy e Knopf (2017), a utilização de sensores de cor apresenta desafios para determinação do estado do cubo, principalmente por fatores como presença de sombras,

manchas nas faces do cubo e contaminação da luz pelas cores das peças adjacentes. Tais razões afetam a confiabilidade do robô para determinação das cores de forma consistente.

Por outro lado, mesmo com uso de sensores, especificamente fotoresistores LDR (do inglês, *Light dependent resistor*), o robô desenvolvido por Tsoy (2016), apresentado na Figura 13, consegue superar esses desafios para identificação das cores. Para isso, são utilizadas algumas técnicas, das quais se destacam a eliminação da luz ambiente com uso de proteções e a utilização de um algoritmo de clusterização das cores pelos dados coletados.

Nessa solução, se utiliza de três LDR's que realizam as leituras das peças do cubo. Sendo assim, é realizada uma movimentação do cubo e dos próprios LDR's de forma que os sensores percorrer pelas faces do cubo e mapear as cores de todas as peças. Para isso, se desenvolve uma estrutura composta por mecanismos metálicos de garras no formato de pinças que, movimentadas por motores de passo, promovem a rotação do cubo e de suas faces.

Figura 13 – Robô desenvolvido por Tsoy (2016).



Fonte: (TSOY, 2016).

2.3 O ALGORITMO DE CLASSIFICAÇÃO *K-NEAREST NEIGHBORS* (KNN)

No contexto de aprendizagem de máquina, existem duas abordagens de destaque para identificar padrões e fazer previsões a partir de um conjunto de dados: a clusterização e a classificação. A clusterização, ou agrupamento, é uma técnica não supervisionada que agrupa dados em grupos com base em suas semelhanças, sem a necessidade de fornecimento de rótulos prévios dos dados. Já a classificação, é uma técnica supervisionada que categoriza novos dados em classes predefinidas com base em exemplos rotulados. Dessa forma, se tem a clusterização como uma técnica descritiva, que faz agrupamentos dos dados pela avaliação da proximidade dos registros em termos de distância. Enquanto isso, a classificação é uma abordagem preditiva, que classifica novos registros que serão passados com base num treinamento desenvolvido em cima de dados previamente fornecidos (SACRAMENTO, 2023).

De forma mais específica no aspecto da classificação, um dos algoritmos de destaque é o k-*nearest neighbors* (KNN), desenvolvido por Fukunaga e Narendra em 1975 (FUKUNAGA; NARENDRA, 1975). Ele opera identificando os k vizinhos mais próximos de uma determinada instância de dados e, com base em suas classes (rótulos) decide a classe da instância desconhecida por meio de votação majoritária. A precisão do algoritmo depende de três fatores: a escolha de k, a métrica de distância utilizada (como distância Euclidiana ou Manhattan) e a qualidade dos dados (PACHECO, 2017). Assim, o método KNN possui características como: ser um algoritmo de aprendizado lento (em inglês, *lazy learning*), em que os dados de treinamento são memorizados; facilidade de implementação e inserção de novos dados na série de maneira instantânea (VAZ, 2021). No aspecto de implementação, o Algoritmo 1 apresenta os passos para execução do método.

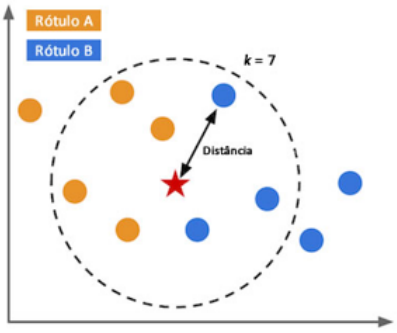
Algorithm 1 Classificação KNN

1: **inicialização:**
2: Preparar conjunto de dados de entrada e saída
3: Informar o valor de k
4: **for** cada nova amostra **do**
5: Calcular distância para todas as amostras
6: Determinar o conjunto das k's distâncias mais próximas
7: O rótulo com mais representantes no conjunto dos k's vizinhos será o escolhido
8: **end for**
9: **retornar:** conjunto de rótulos de classificação

Fonte: Adaptado de Pacheco (2017).

A Figura 14 apresenta um exemplo de classificação KNN com dois rótulos de classe e com k=7. Assim, são calculadas as distâncias de uma nova amostra (estrela vermelha), às demais amostras, representadas pelas bolinhas azuis e amarelas. Como base no valor de k, se verificam as sete amostras mais próximas da nova amostra, sendo 4 são do rótulo A e 3 do rótulo B. Com isso, dado que existem mais vizinhos do rótulo A, se define A que o rótulo da nova amostra (PACHECO, 2017).

Figura 14 – Exemplo de classificação via algoritmo KNN.



Fonte: (PACHECO, 2017).

3 PROJETO MECÂNICO DO ROBÔ

Este capítulo aborda, inicialmente, com base nas premissas adotadas para o robô, a modelagem tridimensional desenvolvida e, em sequência, as impressões 3D e montagens das peças de cada conjunto estrutural do robô. Isto inclui: a estrutura de movimentação das faces laterais, a estrutura de movimentação da face inferior, a estrutura de movimentação da face superior e de fixação dos sensores de cor e a estrutura de sustentação.

3.1 MODELAGEM TRIDIMENSIONAL

Visando o alcance das duas premissas estabelecidas no Capítulo 1, este trabalho utiliza como principais referências conceituais o robô desenvolvido por Wangyuyyt (2022), o qual se utiliza exatamente de seis motores de passo, um acoplado em cada face do cubo (cumprindo a Premissa 1), e o robô desenvolvido por Swiezy e Knopf (2017), que utiliza dois sensores de cor para a realização da leitura (cumprindo a Premissa 2).

A proposta de unir esses dois conceitos tem como principal objetivo, por meio de projetos acessíveis, desenvolver um robô que combine potencial para alta velocidade de resolução com simplicidade na leitura, que é objetivo central do projeto. Entretanto, como apresentado por Swiezy e Knopf (2017), o uso de sensores de cor impõe um desafio em relação a assertividade da leitura, a qual passa a necessitar de estratégias específicas para um bom desempenho, como evidenciado por Tsoy (2016). Vale ressaltar que embora o projeto de Tsoy (2016) seja importante neste trabalho, ele não é utilizado como referência ao projeto mecânico dado que não cumpre as duas premissas com exatidão, pois, além de não utilizar seis motores acoplados em cada face, também utiliza três LDR's que se movimentam para realizar a leitura.

Sendo assim, o projeto mecânico desenvolvido neste trabalho realiza uma combinação entre as duas soluções e implementa modificações para cada grupo da estrutura de movimentação, que são compreendidas em quatro, sendo elas:

a. Estrutura de movimentação das faces laterais (frontal, traseira, esquerda e direita);

b. Estrutura de movimentação da face inferior;

c. Estrutura de movimentação da face superior e de fixação dos sensores de cor;

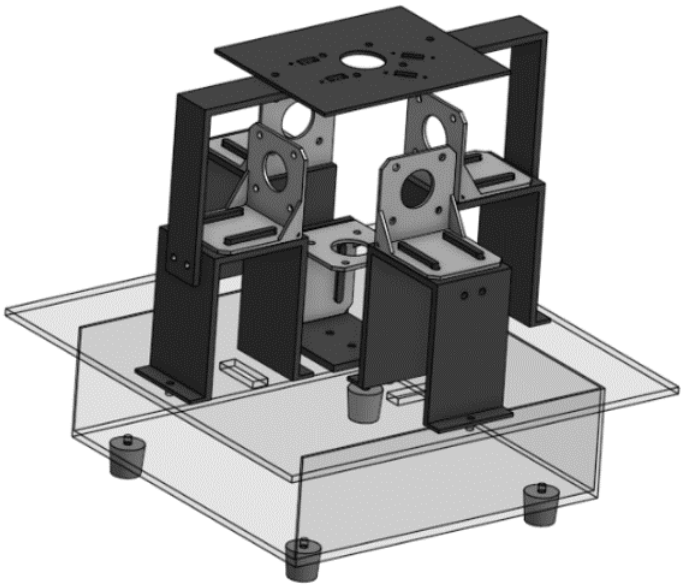
d. Estrutura de sustentação.

De forma mais detalhada, a Figura 15 apresenta a modelagem tridimensional construída do projeto mecânico do robô, especificando cada uma das estruturas, desenvolvidas utilizando a plataforma de nuvem Onshape[3].

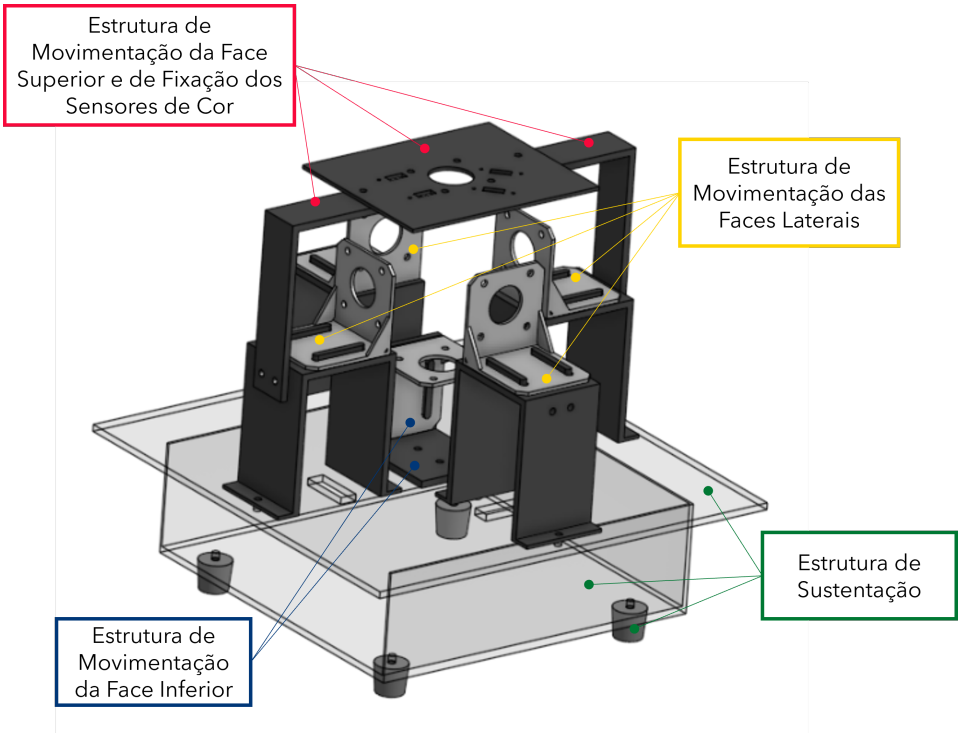
[3] <https://www.onshape.com/en/>

Figura 15 – Modelagem tridimensional do projeto mecânico do robô projetada na plataforma de nuvem Onshape: (a) modelagem tridimensional e (b) especificação de cada estrutura na modelagem.

(a)



(b)



Fonte: Autoria própria.

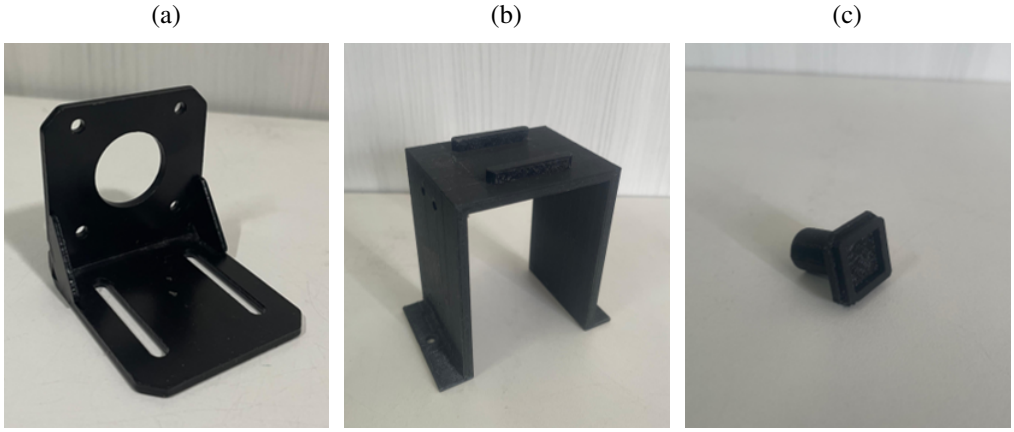
3.2 ESTRUTURA DE MOVIMENTAÇÃO DAS FACES LATERAIS

A estrutura de movimentação das faces laterais (frontal, traseira, esquerda e direita) do cubo utiliza de quatro conjuntos iguais, um para cada face, de três peças, sendo elas: um suporte metálico do motor de passo, um suporte de apoio e um conector eixo-cubo.

A Figura 16 apresenta as peças que compõem o conjunto para movimentação das faces laterais. Na Figura 16a se tem o suporte metálico do motor de passo [4], que é adquirido junto com o motor de passo. Já na Figura 16b e na Figura 16c se tem, respectivamente, o suporte de apoio e o conector eixo-cubo [5], impressos em impressora 3D. A modelagem tridimensional e as dimensões do Suporte de Apoio podem ser encontradas na Figura 71 no Apêndice A.

De forma específica, neste conjunto, a peça do conector eixo-cubo é utilizada para realizar a ligação entre cada eixo do motor de passo e o respectivo eixo central do cubo. Assim, a peça garante o giro de cada face do cubo em sincronismo com o giro dos motores de passo. Além disso, o suporte metálico em união com o suporte de apoio tem a função de garantir a conexão do motor de passo com o cubo na altura correta para movimentação da face.

Figura 16 – Conjunto de peças para movimentação das faces laterais do cubo: (a) suporte metálico do motor de passo, (b) suporte de apoio e (c) conector eixo-cubo.



Fonte: Autoria própria.

3.3 ESTRUTURA DE MOVIMENTAÇÃO DA FACE INFERIOR

Para a estrutura de movimentação da face inferior do cubo, se utiliza de um conjunto de três peças, sendo elas: um suporte metálico do motor de passo, um suporte de apoio reorientado e um conector eixo-cubo.

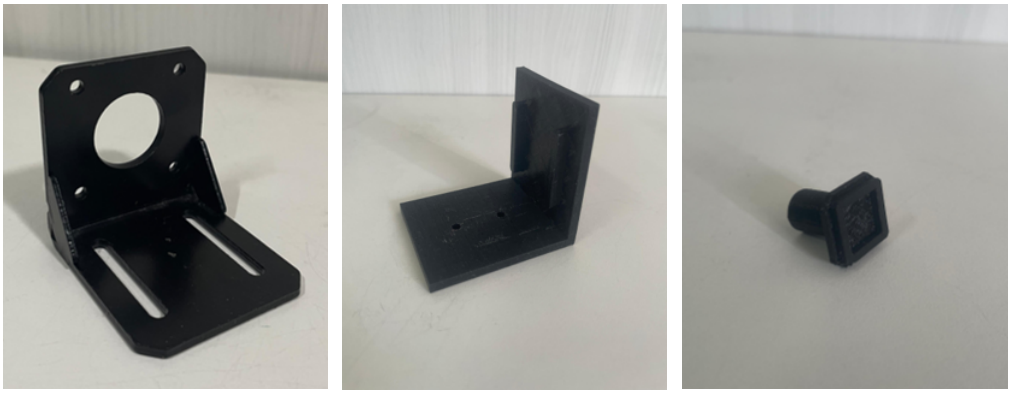
Este conjunto também possui o suporte metálico do motor de passo e o conector-eixo cubo, Figura 17a e Figura 17c, respectivamente. No entanto, o suporte de apoio é reorientado e com diferentes dimensões em relação ao suporte de apoio das faces laterais. Essas adaptações são

[4] <https://loja.forsetisolucoes.com.br/suporte-em-aco-p-motor-de-passo-nema-17-e-23>
[5] <https://www.thingiverse.com/thing:2999309>

feitas para que todo o conjunto possa ser acoplado embaixo do cubo. A Figura 17b apresenta a peça do suporte de apoio reorientado impressa em impressora 3D. A modelagem tridimensional e as dimensões do Suporte de Apoio Reorientado estão na Figura 72 apresentada no Apêndice A.

Figura 17 – Conjunto de peças para movimentação da face inferior: (a) suporte metálico do motor de passo, (b) suporte de apoio reorientado e (c) conector eixo-cubo.

(a) (b) (c)



Fonte: Autoria própria.

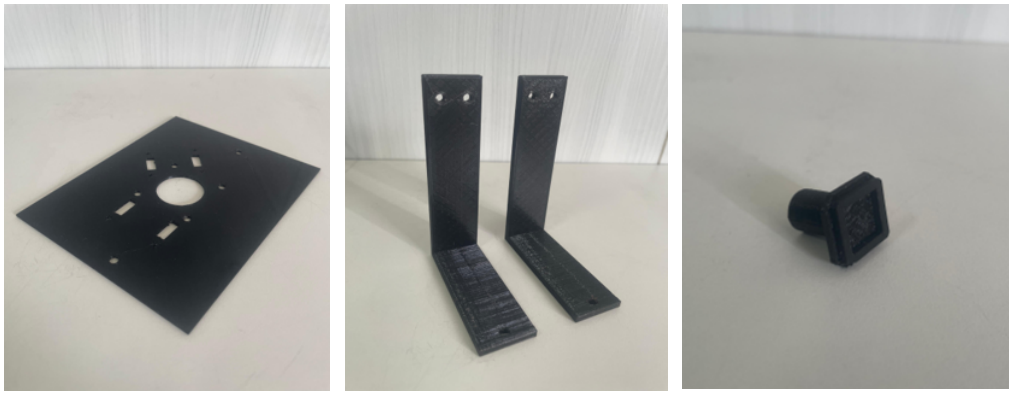
3.4 ESTRUTURA DE MOVIMENTAÇÃO DA FACE SUPERIOR E DE FIXAÇÃO DOS SENSORES DE COR

A estrutura de movimentação da face superior e de fixação dos sensores de cor utiliza de um conjunto de quatro peças, sendo elas: dois suportes superiores em L, uma base superior e um conector eixo-cubo.

Neste conjunto, o conector eixo-cubo segue o mesmo desenho dos demais, conforme a Figura 18c. No entanto, é acrescentada a base superior em conjunto com os suporte superiores em L que atuam na fixação do motor de passo superior no cubo mantendo os dois sensores de cor próximos da face superior. A Figura 18a e a Figura 18b apresentam, respectivamente a base superior e os suportes superiores em L impressos em impressora 3D. A modelagem tridimensional e as dimensões das mesmas podem ser encontradas nas Figuras 74 e 73 no Apêndice A.

Figura 18 – Conjunto de peças para movimentação da face superior e fixação dos sensores de cor: (a) base superior, (b) suportes em L e (c) conector eixo-cubo.

(a) (b) (c)



Fonte: Autoria própria.

3.5 ESTRUTURA DE SUSTENTAÇÃO

Para a estrutura de sustentação do robô se utiliza de um conjunto de seis peças, sendo elas: uma base de sustentação geral, uma caixa inferior e quatro pezinhos de base. A Figura 19 apresenta a base de sustentação geral em acrílico cortada a laser, juntamente com a caixa inferior. A Figura 19c apresenta os pezinhos da caixa inferior, impressos em impressora 3D. As modelagens tridimensionais e as dimensões de cada peça do conjunto de sustentação podem ser encontradas nas Figuras 75, 76 e 77 no Apêndice A.

Figura 19 – Conjunto de peças da estrutura de sustentação: (a) base de sustentação geral, (b) caixa inferior e (c) quatro pezinhos de base.

(a) (b) (c)

Fonte: Autoria própria.

3.6 MONTAGEM E FIXAÇÃO DAS ESTRUTURAS

A montagem das estruturas e fixação é feita em partes. Primeiramente, são fixados os quatro pezinhos de base na caixa inferior. Para isso, se utiliza de quatro parafusos Philips Panela M3 Rosca Máquina de 16 mm e quatro porcas M3, conforme mostrado na Figura 20.

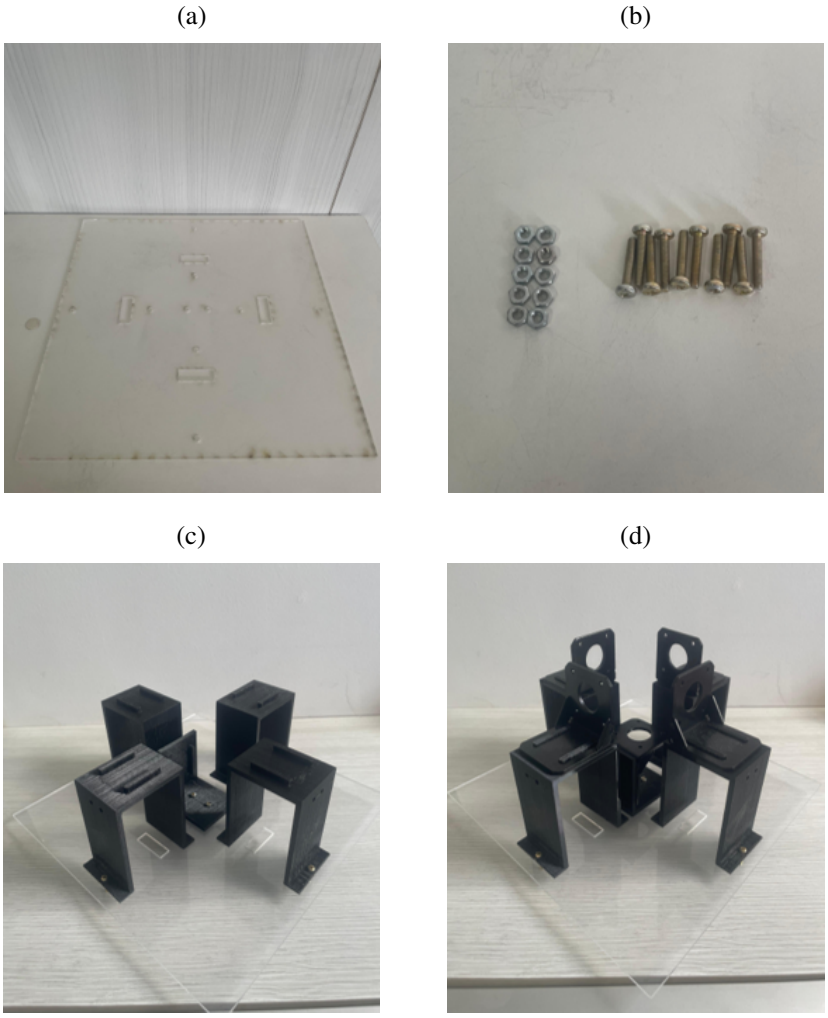
Figura 20 – Fixação das peças do conjunto de sustentação: (a) parafusos Philips Panela M3 Rosca Máquina de 16 mm e porcas M3, (b) caixa inferior com pezinhos de base fixados.

(a) (b)

Fonte: Autoria própria.

Posteriormente, são fixados os quatro suportes de apoio e o suporte de apoio reorientado na base de sustentação geral, utilizando de dez parafusos Philips Panela M3 Rosca Máquina de 16 mm e dez porcas M3, conforme a Figura 21. De forma complementar, são encaixados os suporte metálicos dos motores de passo nos suporte de apoio, conforme a Figura 21d.

Figura 21 – Fixação das peças do conjunto de movimentação das faces laterais e da face inferior: (a) base de sustentação, (b) parafusos Philips Panela M3 Rosca Máquina de 16 mm e porcas M3, (c) suportes de apoio fixados na base de sustentação e (d) suportes metálicos dos motores de passo encaixados na estrutura.



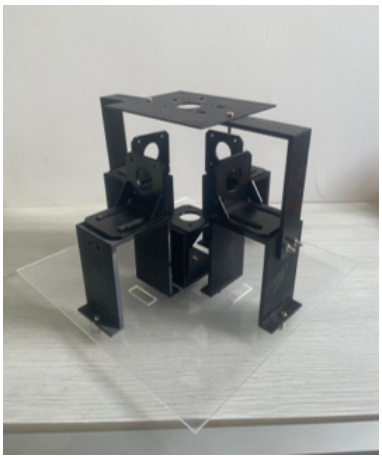
Fonte: Autoria própria.

Adicionalmente, para montagem do conjunto superior, são fixados os suportes em L, junto com a base superior, nos suportes de apoio manuseando seis parafusos Philips Panela M3 Rosca Máquina de 16 mm e seis porcas M3, Figura 22a. Assim, a estrutura mecânica do robô é inteiramente montada e fixada, como pode ser visto na Figura 22c. Vale destacar que os conectores eixo-cubo são montados na estrutura em conjunto com os motores de passo, como é apresentado no Capítulo 5.

Figura 22 – Montagem das peças do conjunto de movimentação da face superior e de fixação dos sensores cor: (a) parafusos Philips Panela M3 Rosca Máquina de 16 mm e porcas M3, (b) suportes superiores em L e base superior fixados na estrutura (c) montagem completa com a caixa inferior.

(a)

(b)



(c)

Fonte: Autoria própria.

4 PROJETO DO HARDWARE

Neste capítulo, é apresentada a seleção dos componentes apropriados ao hardware do robô. Primeiramente, são definidos os requisitos funcionais do robô e, em seguida, a seleção é dividida em três etapas baseadas nas atividades apresentadas no diagrama da Figura 2:

1. Coleta de dados;
2. Movimentação dos atuadores;
3. Transferências de dados.

4.1 REQUISITOS FUNCIONAIS DO ROBÔ

De forma inicial, a partir das premissas adotadas no Capítulo 1, se tem que o robô possui como requisitos funcionais: a movimentação do cubo com uso dos motores de passo e a coleta de dados via sensores de cor. Sendo assim, são levantados os componentes específicos para implementar cada uma dessas tarefas para, em seguida, integrá-las.

4.2 SELEÇÃO DOS COMPONENTES PARA COLETA DE DADOS

Os componentes selecionados para coleta de dados são: dois sensores de cor TCS3200 (TAOS, 2009), um Arduino UNO R3 (ARDUINO, 2024) e uma *protoboard* de 170 pontos[6]. De forma particular, os sensores TCS3200 foram escolhidos em razão da disponibilidade do item nos laboratórios do Departamento de Engenharia Mecânica (DEMEC).

4.2.1 Sensor de cor TCS3200

O robô deste trabalho se propõe a coletar os dados das cores das peças do cubo a partir da utilização do sistema de cores RGB (do inglês, *Red, Green and Blue*). Esse sistema, admite a representação aditiva de cor em que todas as cores podem ser representadas como combinações das cores vermelho, verde e azul (OLIVEIRA, 2022). Conforme a Figura 23a, a representação aditiva de cores é caracterizada pela mistura entre elas. De forma que as três cores produzem juntas o branco e a ausência de cores gera o preto (RAMBAUSKE, 1985). Além disso, o modelo RGB pode ser interpretado como um sistema de coordenadas cartesianas que produzem um cubo em que os três vértices representam as respectivas cores (FILHO; NETO, 1999), conforme a Figura 23b.

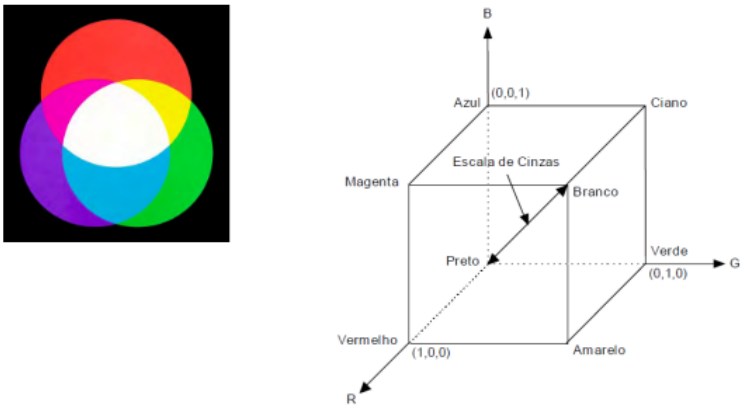
Assim, pela Figura 23b, se tem que o espaço tridimensional é definido nas três direções (R, G e B), variando de 0 a 1. Segundo Lopes (2013), de forma geral, as implementações do sistema RGB adotam valores entre 0 e 255 para representar o nível de intensidade de cada

[6] <https://www.eletrogate.com/mini-protoboard-170-pontos>

componente. Essa mudança do formato normalizado (de 0 a 1) para discretizar em 256 números inteiros, de 0 a 255, decorre do fato de valores inteiros promoverem maior velocidade de processamento nos sistemas gráficos (LOPES, 2013).

Figura 23 – Sistema RGB: (a) Mistura Aditiva de Cores e (b) Modelo RGB em coordenadas cartesianas (cubo).

(a) (b)

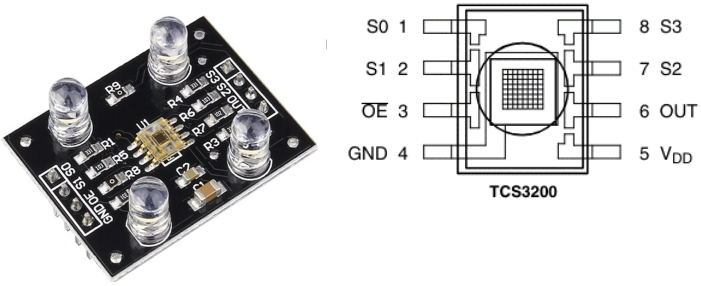


Fonte: (a) (RAMBAUSKE, 1985) e (b) (FILHO; NETO, 1999).

Sendo assim, empregando o sistema RGB, é definida a utilização do sensor TCS3200, Figura 24a, que possui uma matriz de 64 fotodiodos (16 para cada uma das três cores e 16 sem filtro) além de quatro LED's brancos para iluminar o local de leitura e oito pinos (TAOS, 2009), conforme Figura 24. A partir dessa configuração, o sensor consegue converter a luz a partir dos sinais recebidos dos fotodiodos e enviar essa informação em numa frequência de saída que pode ser lida por um microcontrolador (WARDANA; INDAHWATI; FITRIYAH, 2018).

Figura 24 – Sensor TCS3200: (a) exemplo físico e (b) pinos do sensor TCS3200.

(a) (b)



Fonte: (TAOS, 2009).

Com isso, o microcontrolador define quais fotodiodos serão ativados seguindo os sinais nos pinos S2 e S3, conforme a Tabela 3. E, em seguida, a luz incidente nos fotodiodos selecionados e gera uma corrente proporcional à sua intensidade, que é então convertida em uma

frequência por um conversor interno. Com isso, o sinal de frequência é enviado para o pino OUT, onde o microcontrolador pode medir a frequência e, com base na escala ajustada, determinar a intensidade da luz da cor selecionada. De forma complementar, nessa etapa, os pinos de controle S0 e S1 são usados para selecionar a escala de frequência de saída (reduzida a 2%, 20% ou 100% da frequência original), permitindo a adaptação a diferentes níveis de intensidade de luz, conforme a Tabela 4. Dessa forma, reproduzindo essas etapas para cada cor (vermelho, verde, azul e transparente) é possível obter uma leitura completa das componentes de cor do objeto (TAOS, 2009).

Tabela 3 – Tabela de seleção do fotodiodo pelos pinos S2 e S3.

S2	S3	TIPO DE FOTODIODO
L	L	Red
L	H	Blue
H	L	Clear (no filter)
H	H	Green

Tabela 4 – Tabela para seleção da escala de frequência pelos pinos S0 e S1.

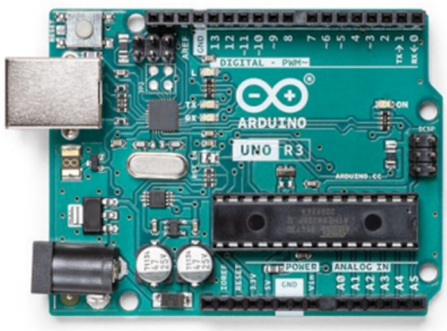
S0	S1	ESCALA DE FREQUÊNCIA DE SAÍDA (f_o)
L	L	Power down
L	H	2%
H	L	20%
H	H	100%

Fonte: Adaptado de TAOS (2009).

4.2.2 Arduino UNO

Para recebimento e interpretação dos dados coletados com os sensores, é utilizada a plataforma de prototipagem eletrônica de código aberto Arduino, desenvolvida por Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino e David Mellis em 2005 (NAYYAR; PURI, 2016). Especificamente, é usada a placa de desenvolvimento Arduino UNO na versão R3, mostrada na Figura 25. Esta placa, utiliza o microcontrolador ATmega328P, possui 14 pinos de entrada e saída digitais, dos quais 6 podem ser usados como saídas PWM (do inglês, *Pulse Width Modulation*), além de 6 entradas analógicas. Além disso, a placa opera com uma tensão de 5V e pode ser alimentada via conexão USB ou com uma fonte externa de 7 a 12V. Sua programação é feita através da Arduino IDE (do inglês, *Integrated Development Environment*), que utiliza uma linguagem de programação baseada em Wiring, uma simplificação do C++, e o seu software é compatível com os sistemas operacionais Windows, Linux e MAC (ARDUINO, 2024).

Figura 25 – Arduino UNO.



Fonte: (ARDUINO, 2024).

4.3 SELEÇÃO DOS COMPONENTES PARA MOVIMENTAÇÃO DO ROBÔ

Os componentes selecionados para movimentação do robô, tendo a Tabela 2 do robô de Wangyuyyt (2022) como referência, são: seis motores de passo 17HS4401 (TECHNOLOGY, s.d), seis drivers A4988 (RYNDACK, s.d), um Arduino MEGA 2560 R3 (ARDUINO, 2023), uma *shield* RAMPS 1.4 (RERAP, 2024), um módulo SM3D (USINAINFO, s.d), uma fonte DC 12V 1A e um conector plug P4 fêmea com borne (BEAGABREW, 2014).

4.3.1 Motores de passo 17HS4401

Segundo Condit e Jones (2004), os motores de passos são dispositivos eletromecânicos que convertem pulsos digitais em movimentos mecânicos incrementais. Nesse sentido, eles funcionam com base nos princípios de eletromagnetismo, em que a passagem de corrente por bobinas produz campos magnéticos que interagem com ímãs permanentes no rotor, causando seu movimento em passos discretos, o que promove um controle preciso do movimento. Esta característica, faz com que os motores de passo sejam amplamente utilizados em aplicações que requerem posicionamento de precisão, como em impressoras 3D, robótica e máquinas de controle numérico computadorizado (CONDIT; JONES, 2004). Por isso, eles são uma alternativa eficaz em robôs solucionadores do cubo de Rubik.

Sendo assim, este projeto utiliza de seis motores de passo do modelo NEMA 17 17HS4401 (CNC, s.d), exemplo mostrado na Figura 26, que se caracterizam por serem motores híbridos de 4 fios, com ângulo de passo de 1,8° e torque de retenção de 43Ncm (TECHNOLOGY, s.d). Sendo híbridos, eles combinam características dos motores de passo de relutância variável e dos motores de ímã permanente, possuindo ímãs permanentes tanto nos estatores com enrolamentos, quanto nos rotores, o que lhes garante alta precisão (CONDIT; JONES, 2004). Além disso, o passo de 1,8° é suficiente para proporcionar um movimento suave, dado os seus 200 passos por revolução. Obtidos seguindo (4.1), em que ângulo de passo é (A_P) e o número de passos por revolução é (N_P) (CONSTANDINOU, 2013).

Figura 26 – Motor NEMA 17HS4401.

Fonte: (CNC, s.d).

$$A_P = \frac{360^\circ}{N_P} \tag{4.1}$$

4.3.2 Driver A4988

Para o controle dos motores de passo, se utiliza de seis *drivers* A4988 (RYNDACK, s.d), que são circuitos integrados que permitem o controle preciso da posição, velocidade e torque do motor de passo. Especificamente, o *driver* A4988 opera através de um conjunto de pinos com funções determinadas. Isto é, possui os pinos STEP e DIR, responsáveis por determinar o avanço dos passos e a direção do movimento, respectivamente. Tem também os pinos MS1, MS2 e MS3 que permitem configurar o modo de subdivisão de passos, proporcionando maior resolução no controle do motor, e os pinos de alimentação (VDD, GND) e de controle de corrente, como VREF e SLEEP, que permitem ajustar a corrente máxima fornecida ao motor, com auxílio de um potenciômetro embutido para ajustes manuais (ALLEGRO, s.d).

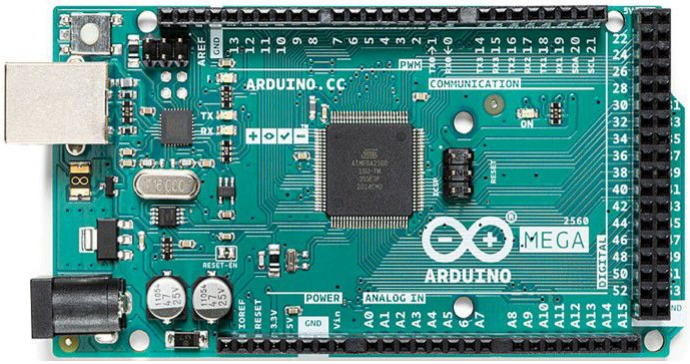
Figura 27 – Driver A4988.

Fonte: (RYNDACK, s.d).

4.3.3 Arduino MEGA

Ainda para o controle dos motores, é utilizada a placa de desenvolvimento Arduino MEGA (ARDUINO, 2023), apresentada na Figura 28, que é equipada com o microcontrolador ATmega2560 e que ofere 54 pinos de entrada e saída digitais, dos quais 15 podem ser usados como saídas PWM, e 16 como entradas analógicas. A placa também inclui 4 portas de comunicação serial. Além da maior capacidade de entrada e saída, o Arduino MEGA é compatível com a maioria das *shields* desenvolvidas para o Arduino UNO e a programação é igualmente realizada através da Arduino IDE (ARDUINO, 2023).

Figura 28 – Arduino MEGA.

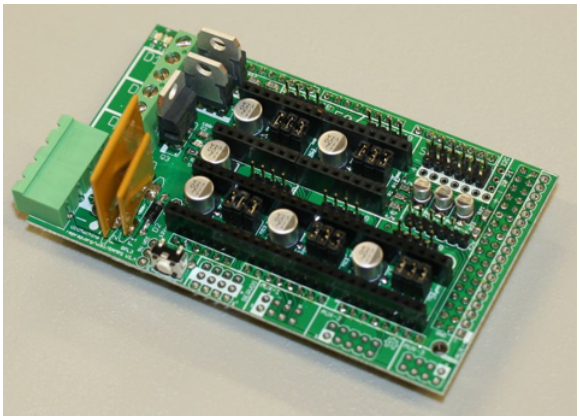


Fonte: (ARDUINO, 2023).

4.3.4 RAMPS 1.4

Tendo como referência Wangyuyyt (2022), é utilizada no projeto, a *shield* RAMPS (RepRap Arduino Mega Pololu Shield) 1.4, mostrada na Figura 29, que é uma placa de expansão utilizada principalmente em impressoras 3D (RERAP, 2024). Com ela, é possível a conexão e o controle de cinco motores de passo e outros componentes. Além disso, a RAMPS 1.4 é equipada com entradas para *drivers* de motores de passo, servo motores e uma ampla gama de interfaces de conexão (RERAP, 2024).

Figura 29 – RAMPS 1.4.



Fonte: (RERAP, 2024).

Nesse projeto, a RAMPS 1.4 é empregada para facilitar a conexão dos *drivers* e para permitir o controle preciso das movimentações dos motores com uso do *firmware* Marlin, que é gratuito, de código aberto, amplamente utilizado em impressoras 3D e licenciado sob a GPLv3 (do inglês, *General Public License version 3*). Ademais, o Marlin é compatível com a plataforma Arduino e especialmente otimizado para funcionar com a RAMPS 1.4 (JBRAZIO, 2021).

Outro aspecto do Marlin, útil para este trabalho, é a sua capacidade de ser personalizado a partir do seu arquivo de configuração, em que é possível ajustar diversos parâmetros como

velocidade e quantidade dos motores, e o fato de utilizar um vocabulário completo da linguagem G-gode com mais de 150 comandos (JBRAZIO, 2021).

O uso da linguagem de programação G-code, é atrativo por se tratar de uma linguagem muito útil para controle de máquinas CNC, incluindo as impressoras 3D. Ela consiste em uma série de instruções que definem movimentos, velocidades, trajetórias e outras operações da máquina. Em impressoras 3D, o código em G-code é gerado por *softwares* de fatiamento, que convertem modelos 3D em camadas de instruções que os dispositivos devem executar sequencialmente para o construir o objeto final (SHIN; SUH; STROUD, 2007). Essas instruções ou comandos G-code permitem ajustes finos na operação da impressora 3D, como modificações na velocidade de impressão, ajustes de temperatura em tempo real e correções de trajetória, conforme exemplos na Tabela 5 (SHIN; SUH; STROUD, 2007), e que são úteis para a robô conseguir movimentar os motores.

Tabela 5 – Exemplos de comandos G-code utilizados no Marlin e suas funções.

Comando	Função
G0 Z0.250 F50	Movimento de Z para 0.250 mm (absoluto ou relativo) com taxa de avanço de 50.
M302 P1	Desabilita a checagem de extrusão a frio
T1	Seleciona a extrusora 1
G91	Habilita o modo de posição relativa
M203 E5	Define taxa de avanço máxima para as extrusoras como 5

Fonte: Adaptado de JBRAZIO (2021).

Sendo assim, o uso do Arduino MEGA com a *shield* RAMPS 1.4 no robô, facilita a conexão dos componentes (motores e *drivers*) de forma integrada (RERAP, 2024). E com o emprego do *firmware* Marlin, se torna possível manipular, via comandos G-code, as movimentações dos motores de passo de forma precisa e coordenada.

4.3.5 Módulo SM3D

Dado que a *shield* RAMPS 1.4 suporta apenas a conexão de cinco *drivers* de motores de passo (RERAP, 2024), é então utilizado o módulo SM3D, mostrado na Figura 30, para possibilitar o acréscimo de mais um *driver* A4988 para o sexto motor. Com esse módulo, é possível o encaixe do *driver* diretamente em sua barra de pinos, evitando a necessidade de utilização de *jumpers* e de uma *protoboard* (USINAINFO, s.d).

Figura 30 – Módulo SM3D.

Fonte: (USINAINFO, s.d).

Dessa forma, o módulo é conectado ao conjunto Arduino MEGA com a *shield* RAMPS 1.4 e o Marlin é instalado com alterações para essa adaptação, conforme é descrito nos Capítulos 5 e 6, respectivamente.

4.3.6 Fonte de energia

Neste projeto, especificamente para alimentação dos motores e *drivers*, é utilizada uma fonte de alimentação positivo interno e negativo externo com entrada de corrente alternada (110V/220V) e saída de corrente contínua (12V) com capacidade de corrente de até 1A. Além disso, na saída da fonte se utiliza um plug conector P4 fêmea, apresentado na Figura 31.

Figura 31 – Fonte de energia e conector plug P4.

Fonte: Adaptado de BEAGABREW (2014).

4.4 SELEÇÃO DOS COMPONENTES PARA TRANSFERÊNCIA DE DADOS

Para a transferência de dados entre os dispositivos do robô, este projeto utiliza três componentes: um computador e dois cabos de comunicação USB-serial.

4.4.1 Computador

Se faz uso de um computador com o intuito de realizar o controle integrado, via código escrito na linguagem Python, entre os motores, sensores e algoritmo solucionador. Outrossim, o computador contribui como fonte de energia (5V) para o Arduino UNO e para sensores TCS3200 pela alimentação USB.

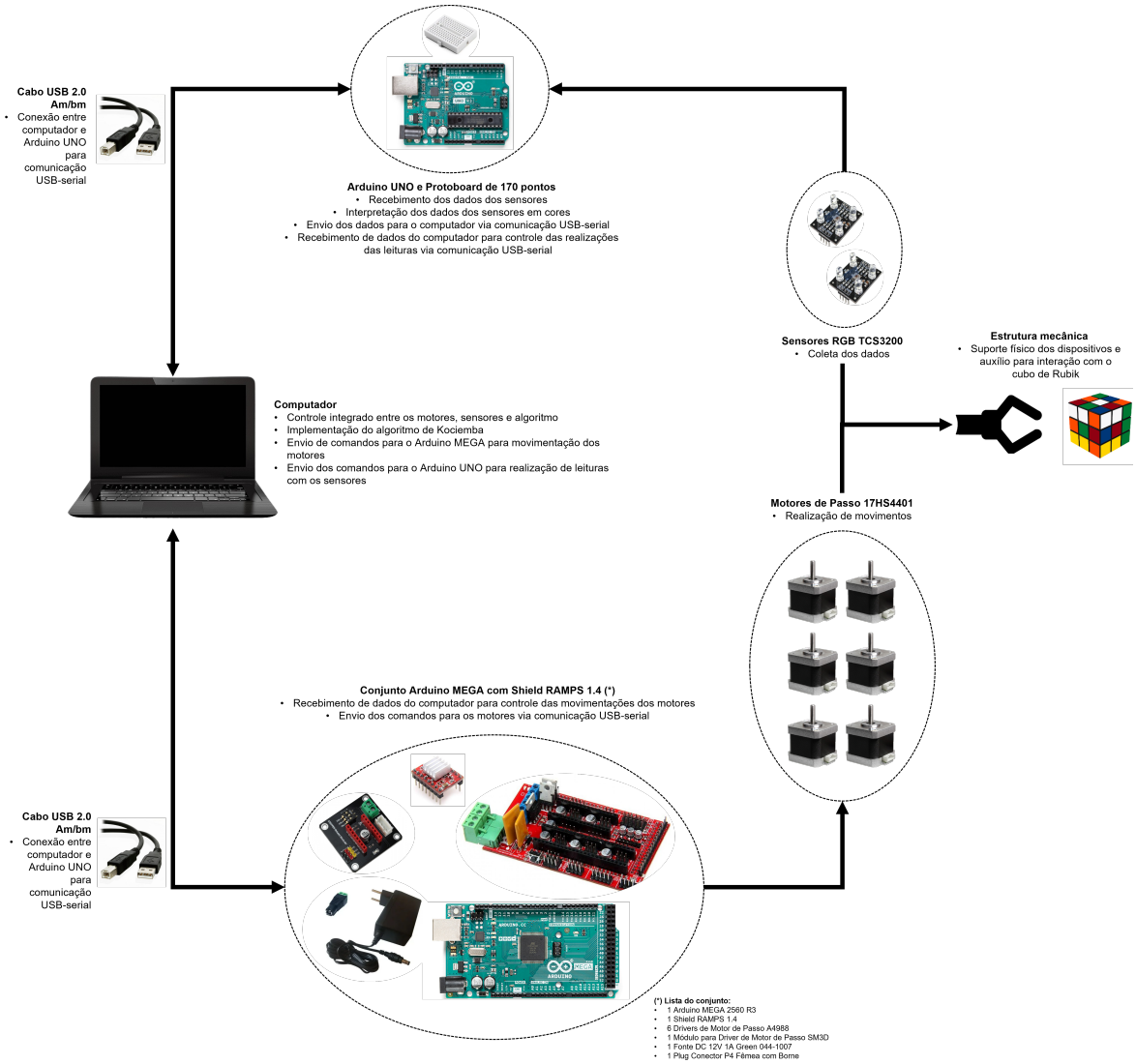
4.4.2 Cabo de comunicação USB-serial

Também, são utilizados, no projeto, dois cabos de comunicação USB-serial Am/bm de 1,8 metros do modelo PC-USB1801[7], para viabilizar a transferência de dados entre as duas plataformas Arduino e o computador.

4.5 DIAGRAMA GERAL DO ROBÔ E COMPONENTES

Com todos os componentes selecionados, é possível compreender o funcionamento geral do robô seguindo o diagrama geral apresentado na Figura 32.

Figura 32 – Diagrama geral de funcionamento do robô com seus componentes.



Fonte: Autoria própria.

[7] <https://www.pluscable.com.br/cabo-usb-20-para-impressora-am-bm-18m-pc-usb1801-pluscable/p>

5 MONTAGEM DO HARDWARE DO ROBÔ

Neste capítulo, inicialmente, é apresentada a lista de todos os componentes eletrônicos utilizados pelo robô. Além disso, é detalhada a sequência de passos para realizar as conexões entre os motores de passo e seus *drivers* no Arduino MEGA junto com a *shield* RAMPS 1.4, seguindo Wangyuyyt (2022), e dos sensores de cor TCS3200 no Arduino UNO. Em seguida, é descrita a montagem e fixação de todos os componentes na estrutura do robô.

5.1 COMPONENTES DO HARWARE DO ROBÔ

De forma geral, o robô utiliza onze diferentes tipos de componentes eletrônicos, sendo 23 ao todo, conforme apresentado na Tabela 6.

Tabela 6 – Lista de componentes eletrônicos utilizados no robô.

Item	Nome	Modelo	Quantidade
1	Motor de Passo	17HS4401	6
2	Sensor de cor	TCS3200	2
3	*Driver* de motor de passo	A4988	6
4	Módulo para *driver* de motor de passo	SM3D	1
5	Arduino UNO	R3	1
6	Arduino MEGA	2560 R3	1
7	*Shield* RAMPS	1.4	1
8	Fonte DC 12V 1A	Green 044-1007	1
9	Conector plug P4 fêmea com borne	Storm PGPQ0002	1
10	Protoboard de 170 pinos	SYB-170	1
11	Cabo USB 2.0 Am/bm de 1,8 metros	PC-USB1801	2

Fonte: Autoria própria.

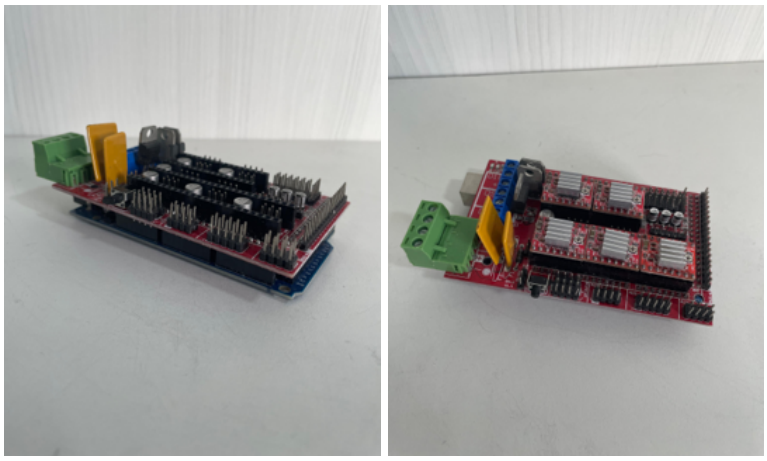
5.2 CONEXÃO DOS MOTORES DE PASSO E *DRIVERS* NO ARDUINO MEGA 2560 COM A *SHIELD* RAMPS 1.4

O Arduino MEGA 2560 é responsável pelo controle do sistema motor do robô em conjunto com a *shield* RAMPS 1.4, a qual suporta cinco *drivers* A4988 e possui conexões para cinco motores de passo.

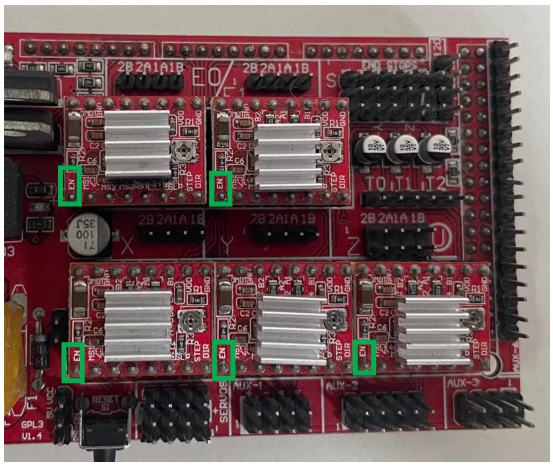
Assim, é conectada a *shield* RAMPS 1.4 ao Arduino MEGA conforme a Figura 33a. Também são acoplados os cinco *drivers* A4988, junto com seus dissipadores de calor, na placa e com a orientação correta (com os pinos ENABLE na posição adequada), conforme mostra a Figura 33b e a Figura 33c.

Figura 33 – Montagem entre Arduino MEGA, RAMPS 1.4 e *drivers* A4988: (a) conexão da *shield* RAMPS 1.4 no Arduino MEGA, (b) encaixe de cinco *drivers* A4988 na RAMPS 1.4 com seus dissipadores de calor, (c) detalhamento da posição dos pinos Enable dos *drivers* A4988 para orientação correta.

(a) (b)



(c)



Fonte: Autoria própria.

Depois, são conectados os cinco motores de passo com seus fios na RAMPS 1.4, conforme mostrado na Figura 34a e na Figura 34b, seguindo a Tabela 7. Ademais, não são inseridos *jumpers* nos pinos inferiores aos *drivers*, visando manter a configuração de passo completo (sem subdivisões no passo ou *microstepping*), conforme descrito em RERAP (2024) e justificado no Capítulo 6.

De forma complementar, são conectados as entradas de alimentação provenientes da fonte 12V 1A com a RAMPS 1.4, com uso do plug conector P4 fêmea com borne, conforme a Figura 34c.

Figura 34 – Montagem dos motores de passo e fonte 12V 1A na RAMPS 1.4: (a) conexão dos fios dos cinco motores de passo na RAMPS 1.4, (b) os cinco motores conectados na RAMPS 1.4 e (c) conexões da fonte 12V 1A, com uso do plug conector P4 fêmea com borne, na RAMPS 1.4.

(a) (b) (c)

Fonte: Autoria própria.

5.3 ADIÇÃO DO SEXTO MOTOR DE PASSO NA RAMPS 1.4 COM USO DO MÓDULO SM3D

Conforme apresentado no Capítulo 4, é necessária a adição de um *driver* a mais para o sexto motor de passo e esse processo é feito com o uso do módulo SM3D.

Nesse sentido, conforme mostrado na Figura 35, são conectados cinco pinos da *shield* RAMPS 1.4 no módulo SM3D, conforme as orientações de Wangyuyyt (2022):

a. os pinos GND e 5V da RAMPS 1.4 com os pinos GND e 5V do módulo SM3D;

b. o pino D40 da RAMPS 1.4 com o pino STEP do módulo SM3D;

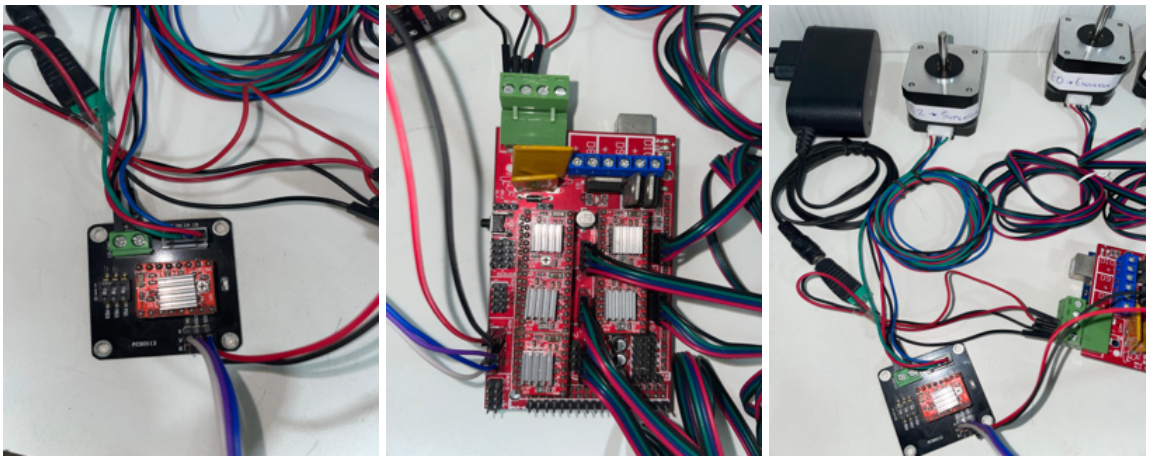
c. o pino D42 da RAMPS 1.4 com o pino DIR do módulo SM3D;

d. o pino D44 da RAMPS 1.4 com o pino EN do módulo SM3D.

De forma complementar são conectadas no módulo SM3D as entradas de alimentação 12V, o *driver* A4988 e os fios do sexto motor, definido como o motor superior e conectado a entrada para a extrusora E2, conforme apresentado na Figura 35.

Figura 35 – Adição do sexto motor de passo na RAMPS 1.4: (a) conexões no módulo SM3D, dos pinos de 5V, GND, STEP, DIR e EN, encaixe do sexto *driver* A4988 no módulo SM3D e dos fios do sexto motor juntamente com conexão das entradas de alimentação 12V e GND, (b) conexões nos pinos de 5V, GND, D40, D42 e D44 na RAMPS 1.4, (c) sexto motor conectado fisicamente a RAMPS 1.4 como uso do módulo SM3D.

(a) (b) (c)



Fonte: Autoria própria.

Com essas conexões realizadas se tem os seis motores de passo conectados na RAMPS 1.4, seguindo o padrão da Tabela 7, como é mostrado na Figura 36.

Tabela 7 – Padrão para conexão entre as entradas para motores de passo na RAMPS 1.4 e cinco motores do robô.

Entrada de pinos na RAMPS 1.4	Motor de Passo conectado
Eixo X	Motor Inferior
Eixo Y	Motor Frontal
Eixo Z	Motor Direito
Extrusora E0	Motor Esquerdo
Extrusora E1	Motor Traseiro
Extrusora E2	Motor Superior

Fonte: Autoria própria.

Figura 36 – Seis motores conectados a RAMPS 1.4

Fonte: Autoria própria.

Por Wangyuyyt (2022), as conexões feitas nos pinos da RAMPS 1.4 implicam em alterações nas configurações do *firmware* Marlin, de forma que é necessário informar no programa a adição de mais um motor de passo. Além disso, o padrão mostrado na Tabela 7 é útil para definição dos comandos G-code que movimentam os motores. Esses dois pontos são abordados no Capítulo 6.

5.4 CONEXÃO DOS SENSORES TCS3200 NO ARDUINO UNO

Para auxílio das conexões entre os dois sensores TCS3200 e o Arduino UNO é utilizada uma *protoboard* de 170 pinos e, conforme mostrado na Figura 37b, são feitas as seguintes conexões:

a. os pinos de 5V e GND do Arduino UNO são conectados no barramento da *protoboard*;

b. os pinos VCC e GND dos dois sensores são conectados no barramento da *protoboard*;

c. os pinos S0 e S1 dos dois sensores são conectados na linha de 5V do barramento da *protoboard*;

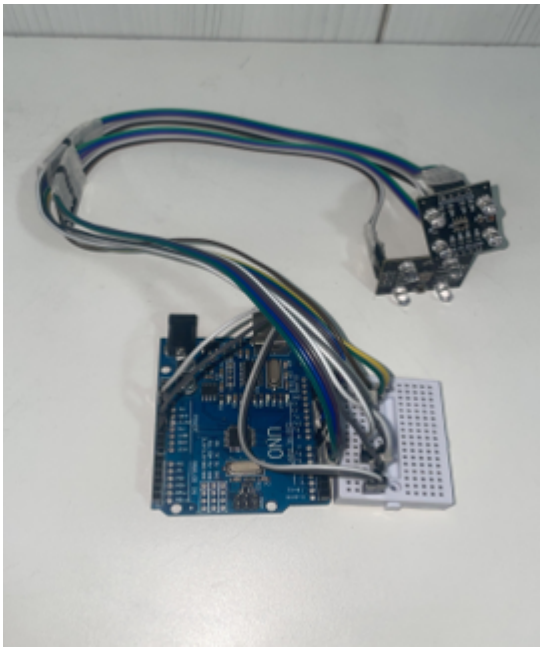
d. os pinos S3, S2 e OUT do Sensor 1 são conectados, respectivamente, nas portas 3,4 e 5 do Arduino UNO;

e. os pinos S3, S2 e OUT do Sensor 2 são conectados, respectivamente, nas portas 8,9 e 10 do Arduino UNO;

Figura 37 – Montagem dos dois sensores TCS3200 conectados no Arduino UNO com uso de uma *protoboard* de 170 pinos: (a) os dois sensores TCS3200 e (b) sensores conectados no Arduino UNO com uso da *protoboard* de 170 pinos.

(a)

(b)



Fonte: Autoria própria.

5.5 CIRCUITO ELETRÔNICO

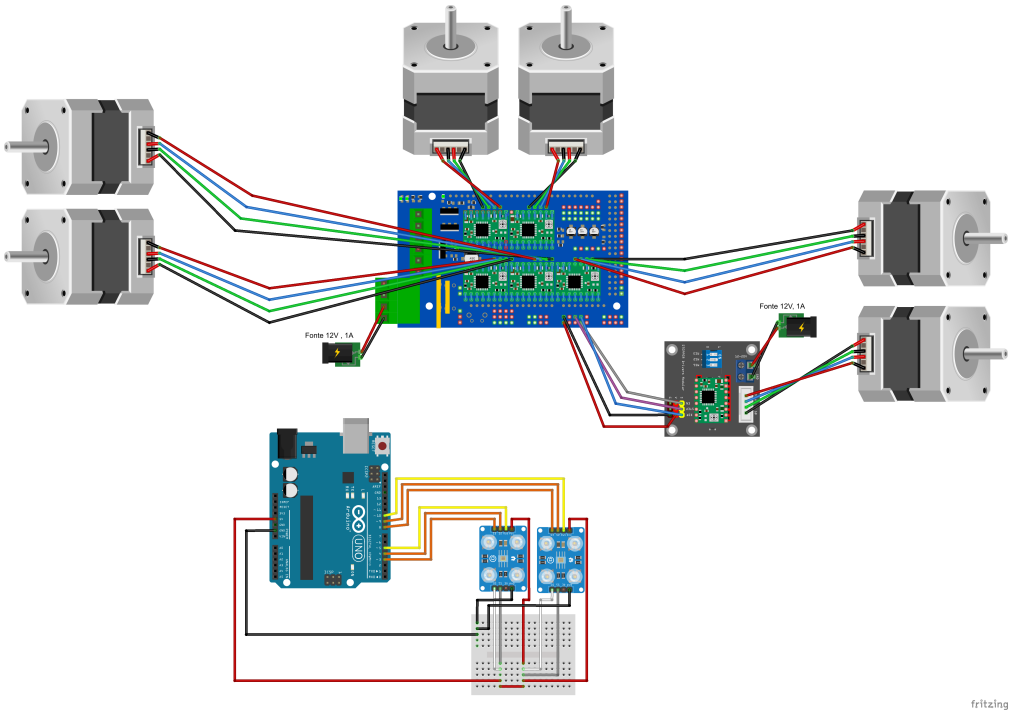
Com todas as conexões realizadas, o circuito completo montado pode ser visto na Figura 38a e também representado conforme esquemático detalhado na Figura 38b, criado com o auxílio

do software Fritzing[8].

Figura 38 – Circuito eletrônico do robô: (a) Todos os componentes eletrônicos do robô conectados e (b) esquemático do circuito eletrônico do robô.

(a)

(b)



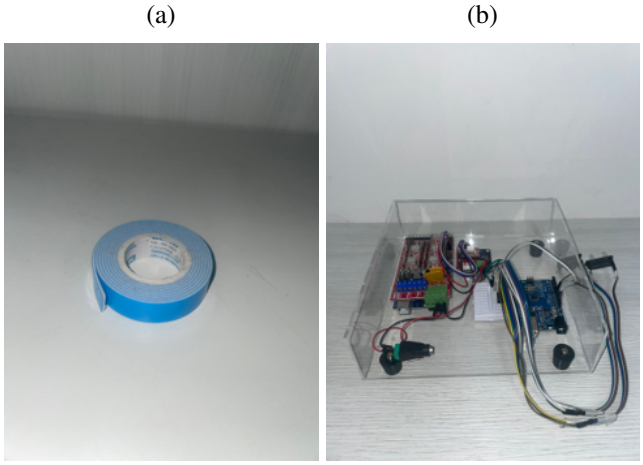
Fonte: Autoria própria.

[8] <https://fritzing.org/>

5.6 MONTAGEM DOS COMPONENTES NA ESTRUTURA DO ROBÔ

Os componentes são montados na estrutura do robô em etapas. Inicialmente, os dispositivos eletrônicos são fixados na base da caixa inferior utilizando fita espuma, conforme a Figura 39.

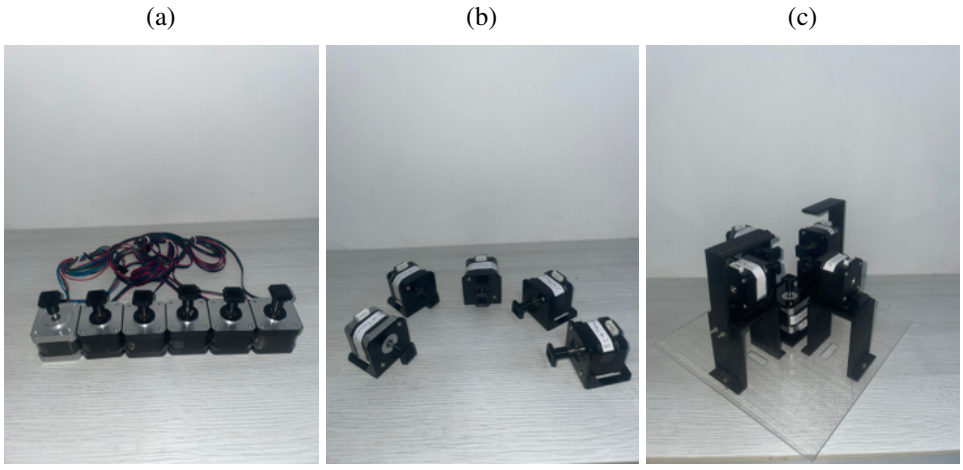
Figura 39 – Fixação dos dispositivos eletrônicos na base da caixa inferior: (a) fita espuma utilizada e (b) dispositivos eletrônicos fixados na base da caixa inferior.



Fonte: Autoria própria.

Em seguida, são acoplados nos seis motores de passo os conectores eixo-cubo, como apresenta a Figura 40a. Assim, os motores laterais e o motor inferior são fixados nos suportes metálicos com uso de parafusos Philips Panela M3 Rosca Soberba de 5 mm, como na Figura 40b, e também são acoplados na estrutura do robô sem a base superior, tal como a Figura 40c.

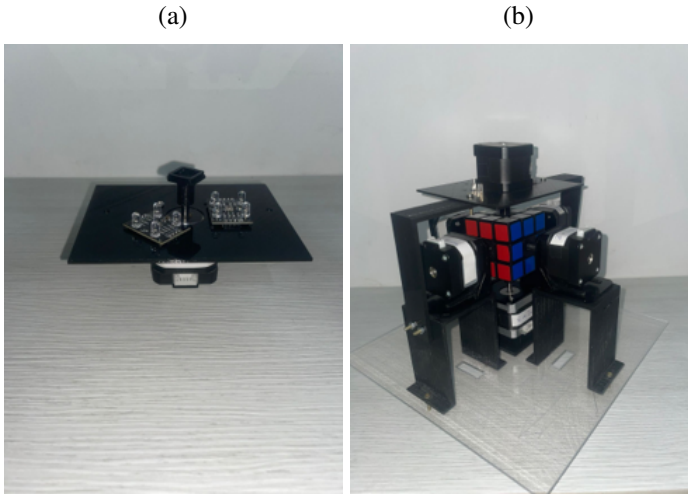
Figura 40 – Montagem dos motores laterais e motor inferior na estrutura: (a) motores acoplados nos conectores eixo-cubo, (b) motores fixados nos suportes metálicos com os parafusos e (c) os cinco motores montados na estrutura do robô.



Fonte: Autoria própria.

Numa etapa complementar, são fixados os componentes na base superior. A Figura 41a apresenta que os sensores são fixados com uso de fita espuma e o motor superior com parafusos Philips Panela M3 Rosca Soberba de 5 mm. Nesse processo, a base superior é desconectada dos suportes em L e, em seguida, é reconectada na estrutura, acoplando também o cubo (sem as peças centrais) nos conectores eixo-cubo, conforme a Figura 41b.

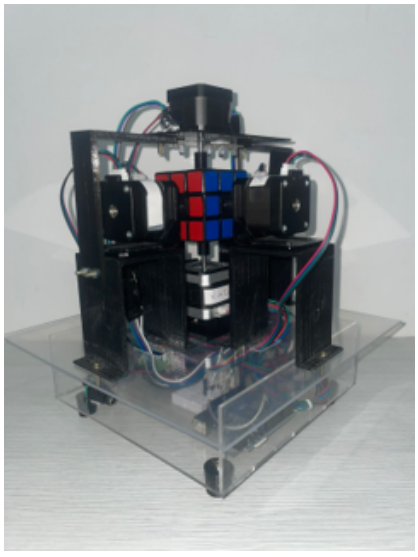
Figura 41 – Montagem do motor superior e dos sensores na estrutura do robô: (a) fixação do motor e sensores na base superior desconectada da estrutura e (b) fixação do motor e sensores na base superior reconectada na estrutura.



Fonte: Autoria própria.

Na etapa final, todos os fios são conectados aos dispositivos respectivos e são fixados próximos a estrutura do robô, em conjunto com a caixa inferior, garantindo a conclusão da montagem do robô com estrutura mecânica e *hardware*, tal como a Figura 42.

Figura 42 – Montagem final do robô com o cubo acoplado.



Fonte: Autoria própria.

6 PROJETO DO FIRMWARE E SOFTWARE DO ROBÔ

Neste capítulo, são apresentados os códigos utilizados e desenvolvidos para o *firmware* e *software* do robô. De forma inicial, é apresentado o fluxograma resumido do funcionamento do robô. Em seguida, são detalhados em cada seção todos os códigos necessários para a implementação do fluxograma resumido em um código final.

6.1 FLUXOGRAMA DE FUNCIONAMENTO DO ROBÔ

De forma geral, conforme apresentado no Capítulo 1, o robô desenvolvido neste trabalho tem, entre os seus objetivos específicos, para solucionar o cubo de Rubik, a aplicação de três blocos de tarefas:

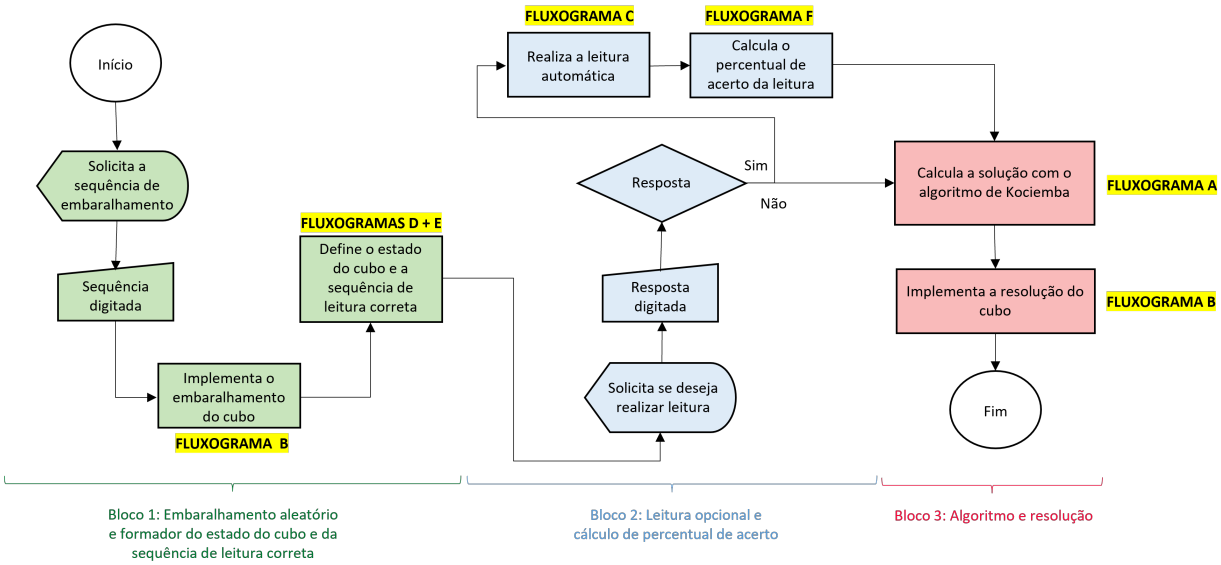
a. realização de um embaralhamento aleatório (bloco 1);

b. execução opcional da leitura e cálculo do percentual de acerto (bloco 2);

c. implementação da resolução do cubo (bloco 3).

Numa visão mais aprofundada, a forma como o robô se propõe conectar e realizar os três blocos de tarefas está apresentada no fluxograma resumido da Figura 43. Nele, se tem que cada processo representa um fluxograma em específico.

Figura 43 – Fluxograma resumido do funcionamento do robô.



Fonte: Autoria própria.

As etapas de implementação em código do Fluxograma resumido segue a sequência dos Fluxogramas A, B, C, D, E e F representados na Figura 43.

6.2 CÓDIGO DO ALGORITMO SOLUCIONADOR DE KOCIEMBA

Visando dispor do algoritmo de Kociemba, este trabalho utiliza do código desenvolvido por Tsoy (2022). Ele é utilizado em outros robôs solucionadores do cubo de Rubik com um resultado muito eficaz, de acordo com Tsoy (2022).

De forma prática, é importado o pacote python denominado *kociemba*, utilizando dele a sua função única *solve()*, a qual recebe uma *string* de estado do cubo para retornar a *string* da sequência de movimentos para a solução (TSOY, 2022), conforme a Figura 44.

Assim, a *string* de 54 caracteres tem a sequência das faces para a construção da *string* de U, R, F, D, L e B, utilizando a notação da Figura 44c, seguindo a sequência: U1, U2, U3, U4, U5, U6, U7, U8, U9, R1, R2, R3, R4, R5, R6, R7, R8, R9, F1, F2, F3, F4, F5, F6, F7, F8, F9, D1, D2, D3, D4, D5, D6, D7, D8, D9, L1, L2, L3, L4, L5, L6, L7, L8, L9, B1, B2, B3, B4, B5, B6, B7, B8, B9 (TSOY, 2022).

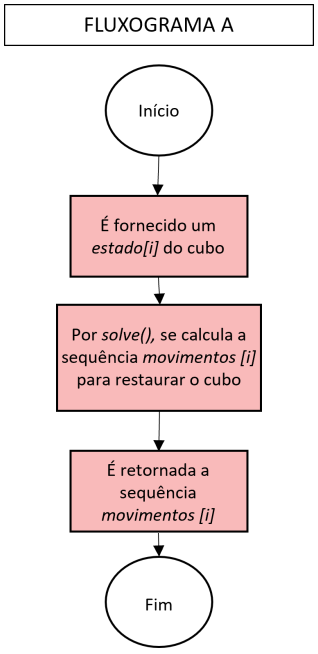
Figura 44 – Funcionamento do código do algoritmo de Kociemba: (a) exemplo de funcionamento do código de Kociemba, (b) Fluxograma A e (c) notação para definição da *string* de estado do cubo.

(a)

```
>>> import kociemba

>>> kociemba.solve('DRLUUBFBRBLURRLRUBLRDDFDLFUFUFFDBRDUBRUFLLFDDBFLUBLRBD')
u"D2 R' D' F2 B D R2 D2 R' F2 D' F2 U' B2 L2 U2 D R2 U"
```

(b)



(c)

```
            |************|
            |*U1**U2**U3*|
            |************|
            |*U4**U5**U6*|
            |************|
            |*U7**U8**U9*|
            |************|
************|************|************|************
*L1**L2**L3*|*F1**F2**F3*|*R1**R2**R3*|*B1**B2**B3*
************|************|************|************
*L4**L5**L6*|*F4**F5**F6*|*R4**R5**R6*|*B4**B5**B6*
************|************|************|************
*L7**L8**L9*|*F7**F8**F9*|*R7**R8**R9*|*B7**B8**B9*
************|************|************|************
            |************|
            |*D1**D2**D3*|
            |************|
            |*D4**D5**D6*|
            |************|
            |*D7**D8**D9*|
            |************|
```

Fonte: Adaptado de Tsoy (2022).

Seguindo o Fluxograma A, implementado pela solução de Tsoy (2022), para executá-lo interagindo fisicamente com o cubo, se torna necessário que o robô tenha implementado, portanto: um código que forneça para função *solve()* a *string* de 54 caracteres do cubo embaralhado e outro código capaz de movimentar os motores a partir da sequência fornecida por *solve()*.

6.3 CÓDIGO PARA MOVIMENTAÇÃO DOS MOTORES

A movimentação dos motores é útil para os três blocos de tarefas do robô, isto é, embaralhamento, leitura e resolução. E para promover as movimentações, é utilizado o Arduino MEGA com a *shield* RAMPS 1.4, conforme apresenta o Capítulo 4. Sendo assim, inicialmente, se realiza a instalação do *firmware* Marlin no Arduino MEGA.

No entanto, como há a adição do sexto motor de passo (Seção 5.3), o *firmware* é instalado com as seguintes adaptações orientadas por Wangyuyyt (2022):

a. no arquivo *Configuration.h*, na linha 230, é alterado o número de extrusoras para 3, conforme a Figura 45a;

b. no arquivo *Marlin/pins/ramps/pins_RAMPS.h*, a partir da linha 200, são acrescentadas as informações dos pinos para a extrusora 2. Em que os pinos de STEP, DIR e ENABLE são, respectivamente, 40, 42 e 44, conforme a Figura 45b.

Figura 45 – Adaptações no Marlin para inserção do sexto motor Wangyuyyt (2022): (a) alteração do número de extrusoras para 3 e (b) inserção dos pinos STEP, DIR e ENABLE da extrudora 2.

(a)

```
225
226   // @section extruder
227
228   // This defines the number of extruders
229   // :[0, 1, 2, 3, 4, 5, 6, 7, 8]
230   #define EXTRUDERS 3
231
```

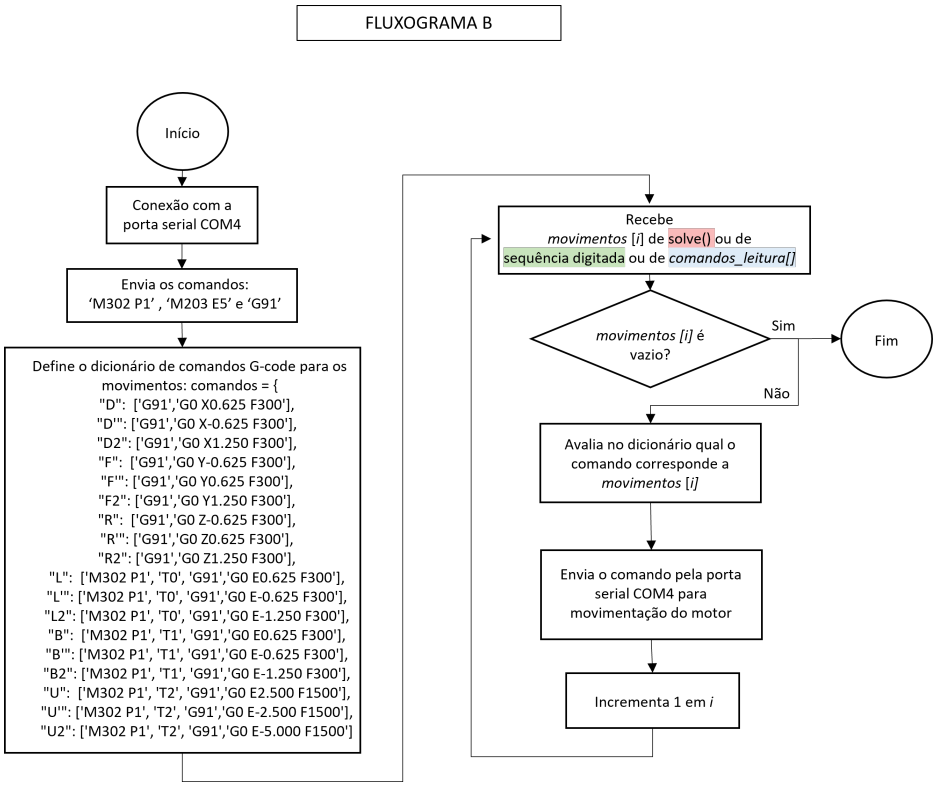
(b)

```
200
201   #ifndef E2_STEP_PIN
202     #define E2_STEP_PIN          40
203   #endif
204   #ifndef E2_DIR_PIN
205     #define E2_DIR_PIN           42
206   #endif
207   #ifndef E2_ENABLE_PIN
208     #define E2_ENABLE_PIN        44
209   #endif
```

Fonte: Autoria própria.

Além disso, para que a sequência de movimentos da solução, da leitura ou do embaralhamento chegue aos motores como comandos que eles devem executar, é desenvolvido um código escrito na linguagem de programação Python que segue o Fluxograma B, apresentado na Figura 46. O código da implementação desse fluxograma é disponibilizado por Araujo (2024) e faz uso das funções *enviar_comando__motores()* e *movimentar_motores()*, junto com a conexão serial com a porta COM4 com 250000 de taxa de transmissão (em inglês, *baud rate*).

Figura 46 – Fluxograma B: para implementação do código de movimentação dos motores seja na resolução, no embaralhamento ou na leitura.



Fonte: Autoria própria.

Pelo Fluxograma B, se verifica que é definido um dicionário de comandos G-code que são responsáveis por implementar cada um dos 18 movimentos nos motores. As funções dos comandos G-code utilizados estão descritas na Tabela 5 e a divisão dos motores por face segue a Tabela 7. Para a definição dos valores utilizados no dicionário, além de se realizar testes, para avaliar em cada motor, os melhores valores que empregam os giros corretos e precisos de cada movimento, se faz uso das seguintes considerações:

1. a configuração de passo completo, definida pela ausência de *jumpers* na RAMPS 1.4, conforme apresentado na Seção 5.2. Dessa forma, os motores passam a se mover por passos inteiros, sem subdivisões, a cada comando, isto é, é mantido neles os 200 passos completos por revolução, estabelecidos por CNC (s.d), e o ângulo de passo de 1,8°, definido por (4.1). Tal decisão, é arbitrada buscando a possibilidade de melhorar a velocidade dos motores, em razão deles passarem a promover passos maiores, do que em *microstepping*, e também na intenção de se manter o máximo de torque possível em cada passo, o que pode ser útil para evitar a perda de passos e superar possíveis atritos nas peças do cubo.

2. a configuração da velocidade máxima e da aceleração máxima dos motores, pela definição dos valores nos campos *DEFAULT_MAX_FEEDRATE* e *DEFAULT_MAX_ACCELERATION* no arquivo *Configuration.h* do Marlin. Por padrão, os motores X e Y possuem os valores

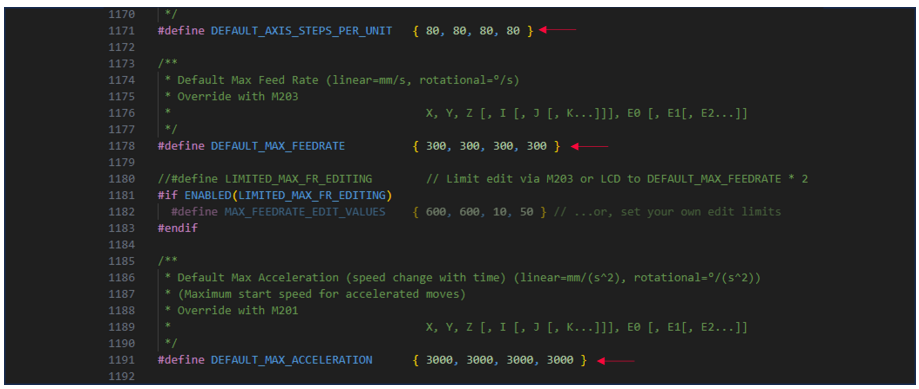
de 300 mm/s e 3000 mm/s^2, para velocidade e aceleração máximas, respectivamente. Logo, para garantir a uniformidade entre os motores, esses valores são expandidos para os demais, conforme apresenta a Figura 47. No caso da velocidade, o valor de 300 mm/s é utilizado em cada um dos motores no campo “F” dos comandos, conforme apresenta a Figura 46.

3. a configuração dos passos por milímetro, pela definição dos valores no campo *DEFAULT_AXIS_STEPS_PER_UNIT* no arquivo *Configuration.h* do Marlin. Por padrão, o Marlin define o valor de 80 para os motores X e Y. Conforme o item anterior, para manter o padrão entre os motores, o valor é replicado aos demais, como mostra a Figura 47.

4. considerando que se utiliza o ângulo de passo de 1,8°, são definidos os valores de posições de destino dos comandos “G0” do dicionário. Isto é, para os motores X, Y, Z, E0 e E1 se utiliza 0,625 (para os movimentos de 90°, variando apenas o sinal conforme o sentido de giro) e 1,250 (para os movimentos de 180°). O valor é calculado considerando que, por exemplo, para 90°, são necessários 50 passos, e, como se estabelece 80 passos por milímetro, no item anterior, se tem que a distância movimentada pelos motores será a razão entre os números, isto é, 0,625 mm. Logo, para 180°, o valor é dobrado. A interpretação de distância movimentada só é possível, levando em consideração a configuração de posição relativa, estabelecida pelo comando “G91”, anterior ao comando “G0”, conforme explica a Tabela 5.

5. para os motores E0, E1 e E2 (extrusoras), se utilizam comandos extras como “T”, para seleção da extrusora e “M302 P1” para habilitar a movimentação de cada extrusora a frio. Para o motor E2, em razão de ser uma extrusora adicionada, os valores de distância relativa e de velocidade são definidos por testes realizados com o motor, de forma que se obter os valores para um comportamento equivalente aos outros motores.

Figura 47 – Configurações no Marlin para calibração dos motores e definição do dicionário de comandos.



Fonte: Autoria própria.

De forma complementar, pela Figura 46, se verifica que a sequência de movimentos a ser executada advém de origens diferentes em cada aplicação. Isto é, na resolução vem da função *solve()*, no embaralhamento a sequência é digitada e na leitura segue os comandos da leitura.

6.4 IMPLEMENTAÇÃO DA LEITURA COM OS SENSORES DE COR

A realização da leitura, se refere ao Fluxograma C, do Fluxograma resumido. Neste trabalho, tendo como referência Swiezy e Knopf (2017), se utiliza de dois sensores TCS3200 para a execução da leitura, conforme apresentado no Capítulo 5, sendo um para leitura dos cantos e o outro para leitura dos meios. No entanto, é utilizado um método desenvolvido neste trabalho para realizar leitura das peças do cubo.

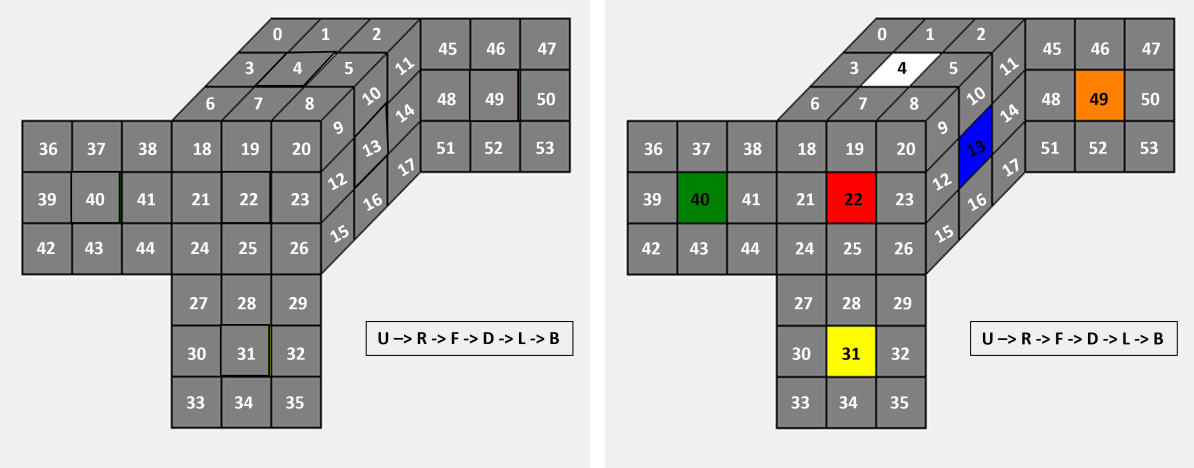
6.4.1 Método construído para realização da leitura do cubo

Como apresentado na Seção 6.2, o algoritmo de Kociemba por Tsoy (2022) necessita de uma *string* de estado a qual é formada pelos caracteres com as letras das faces do cubo (na sequência das faces U, R, F, D, L e B). Para isto, é necessário arbitrar a posição que o cubo terá no robô, dado que cada cor tem que corresponder a uma face específica para ser possível formar a *string* de estado a partir da leitura.

Sendo assim, se mapeia cada posição da *string* de 54 caracteres com números (pela sequência de Tsoy de U, R, F, D, L e B), conforme a Figura 48a. Após isso, se define arbitrariamente que a posição das faces do cubo no robô é tal que: a face frontal é a vermelha e a superior é a branca. Assim, como as peças centrais 4, 13, 22, 31, 40 e 49 são fixas, com a condição arbitrada, então, tem-se que elas tem as cores branca, azul, vermelho, amarela, verde e laranja, respectivamente, conforme a Figura 48b.

Figura 48 – Mapeamento de cada posição do cubo com números: (a) mapeamento de cada posição do cubo com números seguindo Tsoy (2022) e (b) numeração das posições do cubo com a posição arbitrada de face superior como branca e face frontal como vermelha.

(a) (b)

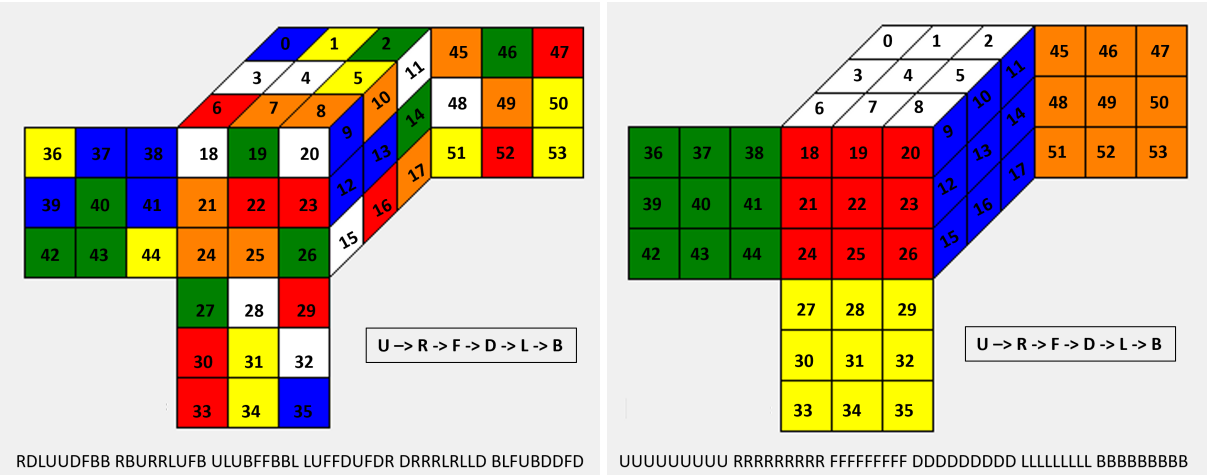


Fonte: Autoria própria.

Assim, como a sequência das faces para a construção da *string* é U, R, F, D, L e B, é possível se obter as *strings* de um cubo embaralhado aleatoriamente, Figura 49a, e do cubo resolvido, Figura 49b, apenas ordenando os caracteres das cores das faces na sequência de 0 a 53.

Figura 49 – Exemplos de *strings* de estado do cubo seguindo o mapeamento de cada posição do cubo em números: (a) numeração das posições com um cubo embaralhado aleatoriamente com sua *string* de estado e (b) numeração das posições com um cubo resolvido na posição arbitrada com sua *string* de estado.

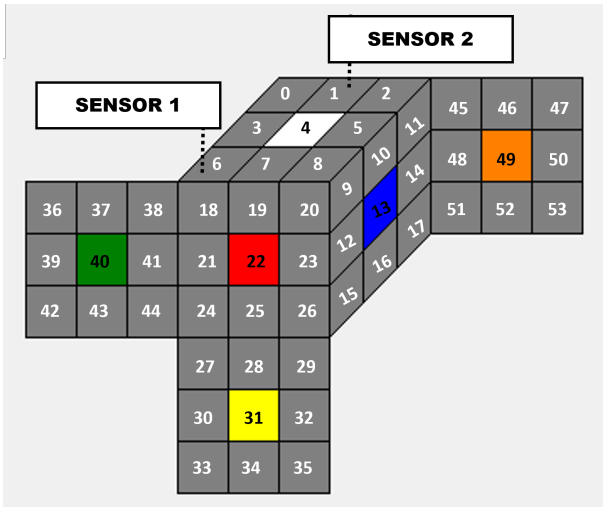
(a) (b)



Fonte: Autoria própria.

Com o mapeamento das posições em números realizado, se estabelece as posições dos sensores em 1 e em 6, conforme Figura 50. Sendo assim, se constrói uma sequência de movimentos necessários às faces para que todos os cantos passem pela posição 6 e todos os meios pela posição 1, a fim de que realizar as leituras com os sensores. Com esse objetivo, são encontradas diferentes alternativas para realizar a leitura sem comprometer o embaralhamento, mas com um número elevado de movimentos. Isto é, por busca manual, primeiro se encontra uma sequência de 82 movimentos para leitura dos cantos pelo Sensor 1 na posição 6 e também outra sequência para leitura dos meios pelo Sensor 2 na posição 1 com 108 movimentos, o que totalizaria uma sequência completa de 190 movimentos. Entretanto, realizando simplificações e retirando as redundâncias dos passos que feitos para um sensor já possibilitam as leituras de peças novas com o outro sensor, se define uma sequência reduzida capaz de, com 60 movimentos, realizar a leitura das 48 posições necessárias, dado que das 54 posições, 6 são as peças fixas com cores já conhecidas.

Figura 50 – Posições em que os sensores lêem as peças do cubo: Sensor 1 em 6 e Sensor 2 em 1.



Fonte: Autoria própria.

Essa sequência de movimentos desenvolvida é descrita no passo a passo apresentado na Figura 51 e sua visualização aplicada no cubo está apresentada em vídeo[9].

[9] <https://youtu.be/ZXycYXFnmuQ>

Figura 51 – Sequência de passos para execução da leitura das 48 posições com 60 movimentos.

Passo 1: Lê o 6; Lê o 1 = 0 movimentos (2 lidos)

Passo 2: R, L', Lê o 24; U; Lê o 26; Lê o 21; U; Lê o 20; Lê o 7; U; Lê o 18; Lê o 23; U = 6 movimentos (7 lidos)

Passo 3: F', B, Lê o 10; Lê o 15; U; Lê o 17; U; Lê o 11; Lê o 16; U; Lê o 9; U = 6 movimentos (6 lidos)

Passo 4: F2, B2, Lê o 42; Lê o 37; U; Lê o 44; U; Lê o 38; Lê o 43; U; Lê o 36; U = 6 movimentos (6 lidos)

Passo 5: R, L', Lê o 35; U; Lê o 29; Lê o 25; U; Lê o 27; U; Lê o 33; Lê o 19; U = 6 movimentos (6 lidos)

Passo 6: F, B', Lê o 45; Lê o 41; U; Lê o 51; U; Lê o 53; Lê o 39; U; Lê o 47; U = 6 movimentos (6 lidos)

Passo 7: R, L', F, B'; Lê o 28; U; Lê o 8; Lê o 32; U; Lê o 2; Lê o 34; U; Lê o 0; Lê o 30; U = 8 movimentos (7 lidos)

Passo 8: R, L', F, B', U; Lê o 48; U2; Lê o 50; U = 7 movimentos (2 lidos)

Passo 9: R, L', F, B'; Lê o 14; U; Lê o 46; U; Lê o 12; U; Lê o 52; U = 8 movimentos (4 lidos)

Passo 10: R, L', U, Lê o 3; U2; Lê o 5; U, F, B' = 7 movimentos (2 lidos)

TOTAL = 60 movimentos e 48 quadrados lidos

Fonte: Autoria própria.

Pela Figura 51, se verifica que a leitura ocorre com a interação de todos os dispositivos, isto é, tanto com os motores, para implementação dos movimentos, quanto com os sensores, para leitura das cores de cada posição. Além disso, o passo a passo da Figura 51, para ser implementado no robô, exige que os comandos de leitura e movimentação sejam enviados coordenadamente e de forma automática.

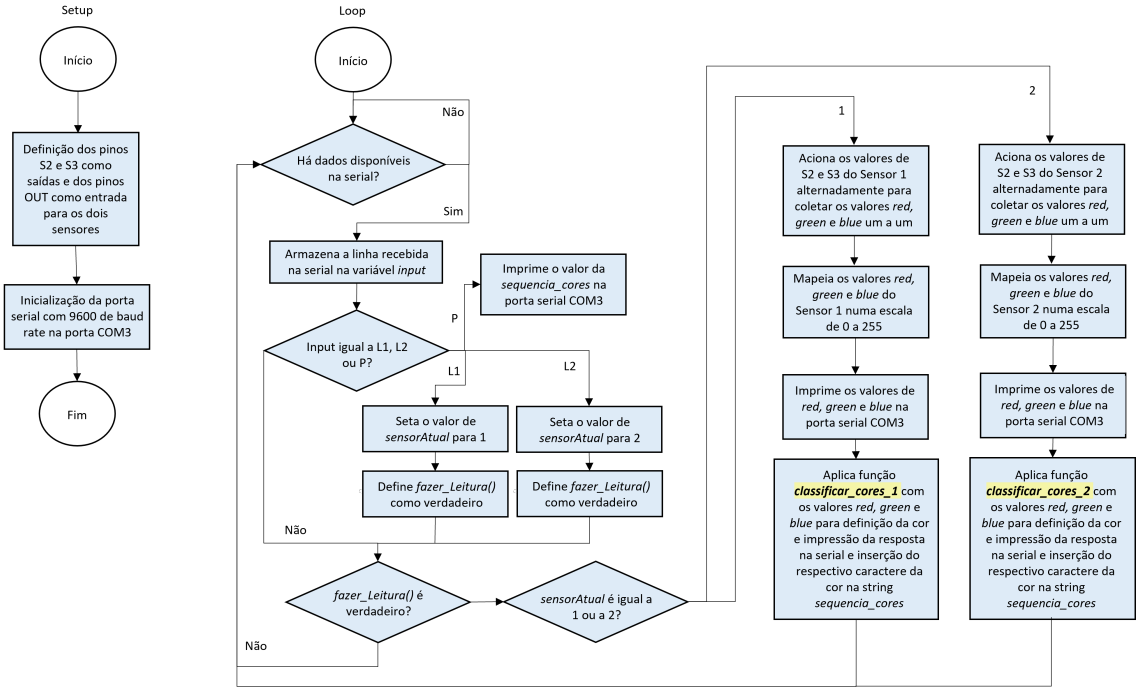
Sendo assim, para a execução dos movimentos, se tem o uso do Fluxograma B apresentado na Seção 6.3. E, para as leituras, se utiliza do fluxograma para execução de leituras e classificação de cores, detalhado na Seção 6.4.2 e do fluxograma para realização da leitura automática, o qual interage com os dois anteriores, a partir do fluxograma apresentado na Seção 6.4.3.

6.4.2 Código para execução de leituras e classificação dos valores RGB em cores

A Figura 52 apresenta o fluxograma do código que realiza a leitura e a classificação dos valores RGB em cores com uso de dois sensores TCS3200 e um Arduino UNO. Para isso, inicialmente, no *setup*, se configura os pinos digitais (apresentados na Seção 5.4), para comunicação com os sensores e se inicializa a serial com 9600 de taxa de transmissão. Em seguida, no *loop*, são definidos dois comandos que, caso sejam enviados pela porta serial, definida como COM3, iniciam uma leitura com a função *fazer_Leitura()*. Especificamente, se tem "L1" para leitura com o sensor 1 (define variável *sensorAtual* como 1) e "L2" para leitura com o sensor 2 (define variável *sensorAtual* como 2). Em caso de recebimento de um dos dois comandos, é definida a função *fazer_Leitura()* como verdadeira e, em sequencia, ela é executada. A partir dela,

em função do valor da variável *sensorAtual*, são intercalados os valores digitais dos pinos S2 e S3, com base na Tabela 3, do sensor em questão, para se coletar os pulsos de vermelho, verde e azul e mapeá-los numa escala RGB (de 0 a 255). Vale ressaltar que se utiliza a função *pulseIn()* pelo Arduino para contar quanto tempo os pinos de saída dos sensores (OUT) permanecem em nível lógico alto ou baixo. Após cada leitura, os valores RGB mapeados são impressos no monitor serial, e uma função é chamada para classificar a cor detectada. Após a classificação da cor, o caractere da cor definida (relacionado a face do cubo) é armazenado em uma *string*, concatenando a sequência de cores lidas. De forma complementar, também é estabelecida a condição de impressão da sequência de cores classificadas através do envio do comando "P" na serial.

Figura 52 – Fluxograma do código de execução da leitura com os dois sensores TCS3200.



Fonte: Autoria própria.

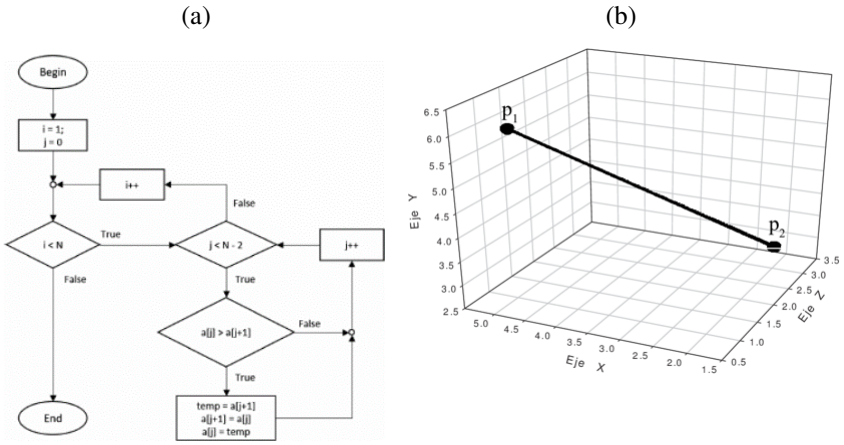
De forma particular, a classificação assertiva dos valores RGB em cores representa um desafio, como apresentado por Swiezy e Knopf (2017). Sendo assim, inspirado nas recomendações de Tsoy (2016), apresentadas na Seção 2.3, a função de *classificar_cores()* é desenvolvida com um algoritmo de classificação: o KNN. A escolha desse método se baseia nas razões apresentadas por Pacheco (2017) e que foram tomadas como importantes para este trabalho: a facilidade de implementação, pela não necessidade de treinamento prévio, e, a condição de inserção de novos dados na série de maneira instantânea.

Sendo assim, é construída a função *classificar_cores()* com base no pseudocódigo apresentado no Algoritmo 1. Além disso, para realizar a ordenação das distâncias dentro do código se utilizou o algoritmo Bubble Sort. Este que é um algoritmo de ordenação simples que

compara pares de elementos adjacentes em uma lista e os troca de posição se estiverem fora de ordem. Esse processo é repetido várias vezes até que toda a lista esteja ordenada (GRIGSBY, 2018), conforme descrito no fluxograma básico do Bubble Sort apresentado na Figura 53a.

Sendo assim, o código da função é construído seguindo o Fluxograma descrito na Figura 54. Vale ressaltar que são construídas duas delas, dado que cada uma terá os dados do seu respectivo sensor. Inicialmente, a função *classificar_cor()* recebe os valores RGB mapeados, define uma matriz com as 60 amostras de referência (10 de cada cor) e também um vetor com os nomes das cores. Em seguida, são calculadas as distância euclidianas, apresentada na Figura 53b, entre as cores detectadas (*red, green, blue*) e todas as amostras de referência. Esse processo é feito dentro de um *for* e, para cada uma das 60 amostras, a função extrai os valores de vermelho, verde e azul e os resultados vãos sendo armazenado em cada iteração num vetor chamado *distancias[]*. Por esse mesmo *for*, a função classifica a amostra de acordo com a cor correspondente (branco, amarelo, vermelho, laranja, verde ou azul), que é identificada com base no índice da amostra e guardada em um vetor *indices_cor[]*, de 0 a 5.

Figura 53 – Métodos usados na função KNN: (a) bubble sort e (b) distância euclidiana em 3D.



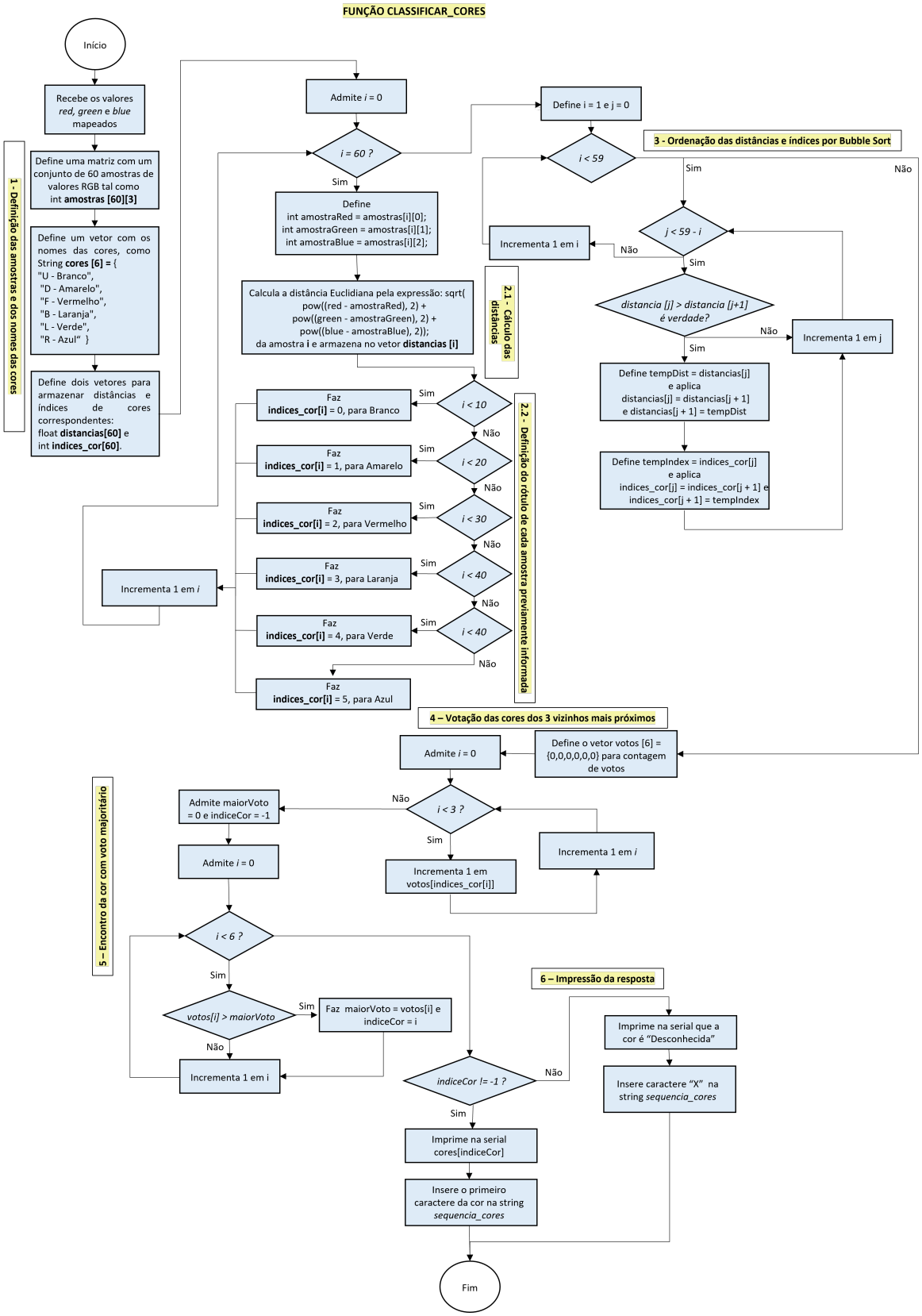
Fonte: (a) (GRIGSBY, 2018) e (b) (NÚÑEZ-COLÍN et al., 2004)

Após calcular as distâncias, um segundo *loop for* implementa o algoritmo de ordenação Bubble Sort, reordenando as distâncias de forma crescente, com base no fluxograma da Figura 53a. Assim, para cada troca de posição entre distâncias, o índice de cor correspondente também é trocado, garantindo que a relação entre distância e cor seja mantida.

Num quarto bloco, a função realiza uma votação usando o conceito dos k-ésimos vizinhos mais próximos (KNN). Assim, um terceiro loop *for* soma os votos para as três amostras mais próximas ($k = 3$). Depois, o quinto bloco determina qual cor recebeu mais votos. Finalmente, com isso, a cor com o maior número de votos é identificada como a cor correspondente e é exibida no monitor serial, no bloco 6. Caso a cor não seja identificada, o sistema marca a cor como desconhecida e registra uma letra “X” para manter o controle da sequência de cores

identificadas.

Figura 54 – Fluxograma da função para classificar os valores em RGB em cores pelo algoritmo KNN.

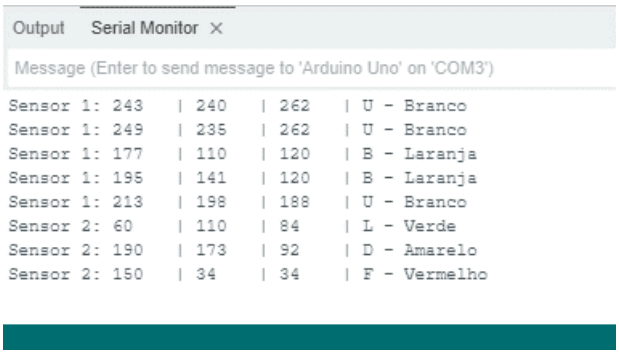


Fonte: Autoria própria.

De forma mais aprofundada os valores de $k = 3$ e de 60 amostras em cada conjunto são arbitrados visando conciliar a celeridade e assertividade na resposta.

O código que implementa os fluxogramas das Figuras 52 e 54 de forma integrada está no arquivo *.ino* denominado “leitura_KNN” disponibilizado por Araujo (2024). Além disso, a Figura 55 apresenta um exemplo de funcionamento da classificação dos valores em RGB em cores pelo algoritmo KNN impressos no monitor serial.

Figura 55 – Exemplo de funcionamento da classificação dos valores em RGB em cores pelo algoritmo KNN impressos no monitor serial.



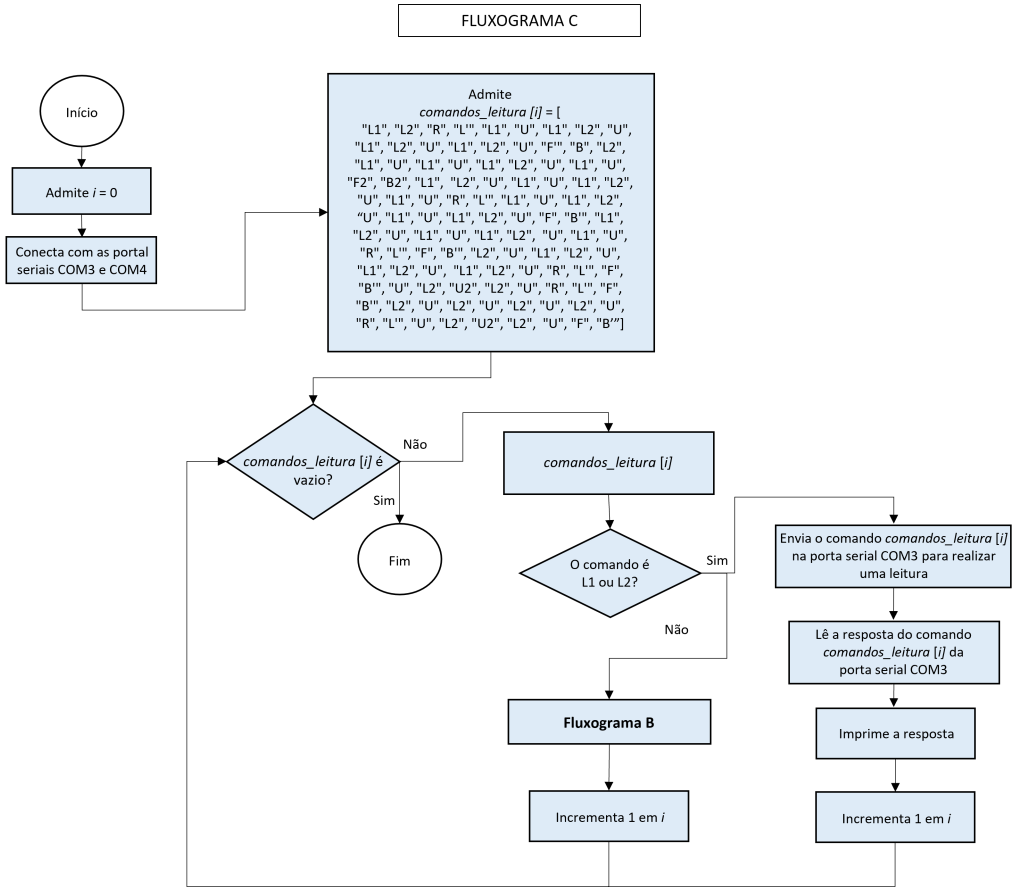
Fonte: Autoria própria.

6.4.3 Código para execução da leitura automática

A implementação da leitura automática utiliza a movimentação dos motores e a execução de leituras, de forma que interage com as duas portas seriais, COM3 (9600 de *baud rate*) e COM4 (250000 de *baud rate*), seguindo a ordem dos comandos da Figura 51. Essa interação é detalhada no Fluxograma C, descrito na Figura 56. Nela, se verifica que a variável *comandos_leitura[]* segue os passos da leitura da Figura 51.

O código que implementa esse fluxograma está na função *leitura_automatica()* descrito por Araujo (2024) no arquivo *.py* denomimado “Código_final”. Ela utiliza de funções como *enviar_comando_sensores()* e *ler_linha_serial_sensores()* para se comunicar com os sensores no Arduino UNO pela porta serial COM3 e a função *movimentar_motores()* para se comunicar com os motores.

Figura 56 – Fluxograma C: para implementação do código de execução da leitura automática.



Fonte: Autoria própria.

6.5 CÓDIGO DEFINIDOR DO ESTADO DO CUBO PELO EMBARALHAMENTO

Como mostrado, o algoritmo de Kociemba desenvolvido por Tsoy (2022), consegue, a partir da informação do estado do cubo, fornecer a solução para restaurá-lo. Entretanto, o código de Tsoy (2022) não consegue informar o estado do cubo após a execução dos movimentos no cubo resolvido.

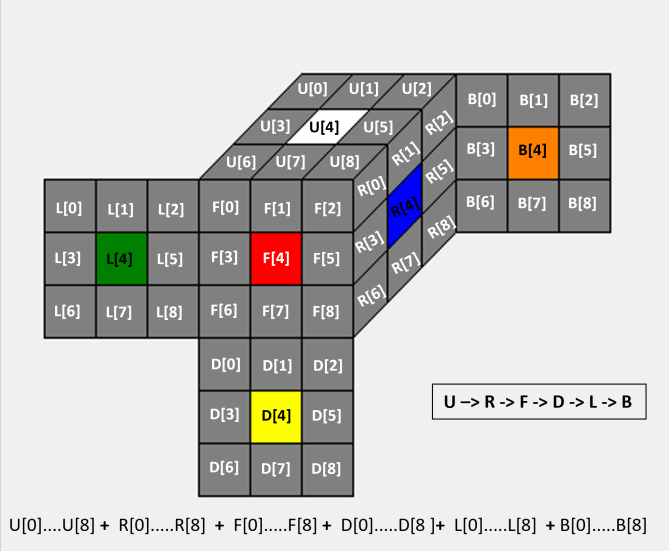
Diante dessa situação, é desenvolvido o código capaz de definir o estado do cubo pelo embaralhamento realizado no cubo resolvido. Essa definição do estado é importante para conseguir avaliar a assertividade da leitura. Sendo assim, para implementação desse código, é necessário compreender os efeitos de cada movimento na string de estado do cubo.

6.5.1 Efeitos de cada movimento na *string* de estado do cubo

Dado um embaralhamento aplicado no cubo resolvido, se tem que as peças são reorientadas, e, assim, se forma uma nova *string* de estado, isto é, os caracteres são re-sequenciados. Devido essa situação, para conseguir obter a nova *string* de estado do cubo, passa a ser necessário tratar o estado do cubo como uma lista.

De forma mais precisa, o estado do cubo pode ser, então, interpretado como uma junção de seis listas (uma de cada face), Figura 57, de maneira que a partir de cada movimento, os itens das listas trocam de posição.

Figura 57 – Estado do cubo definido como uma junção de seis listas, uma de cada face.

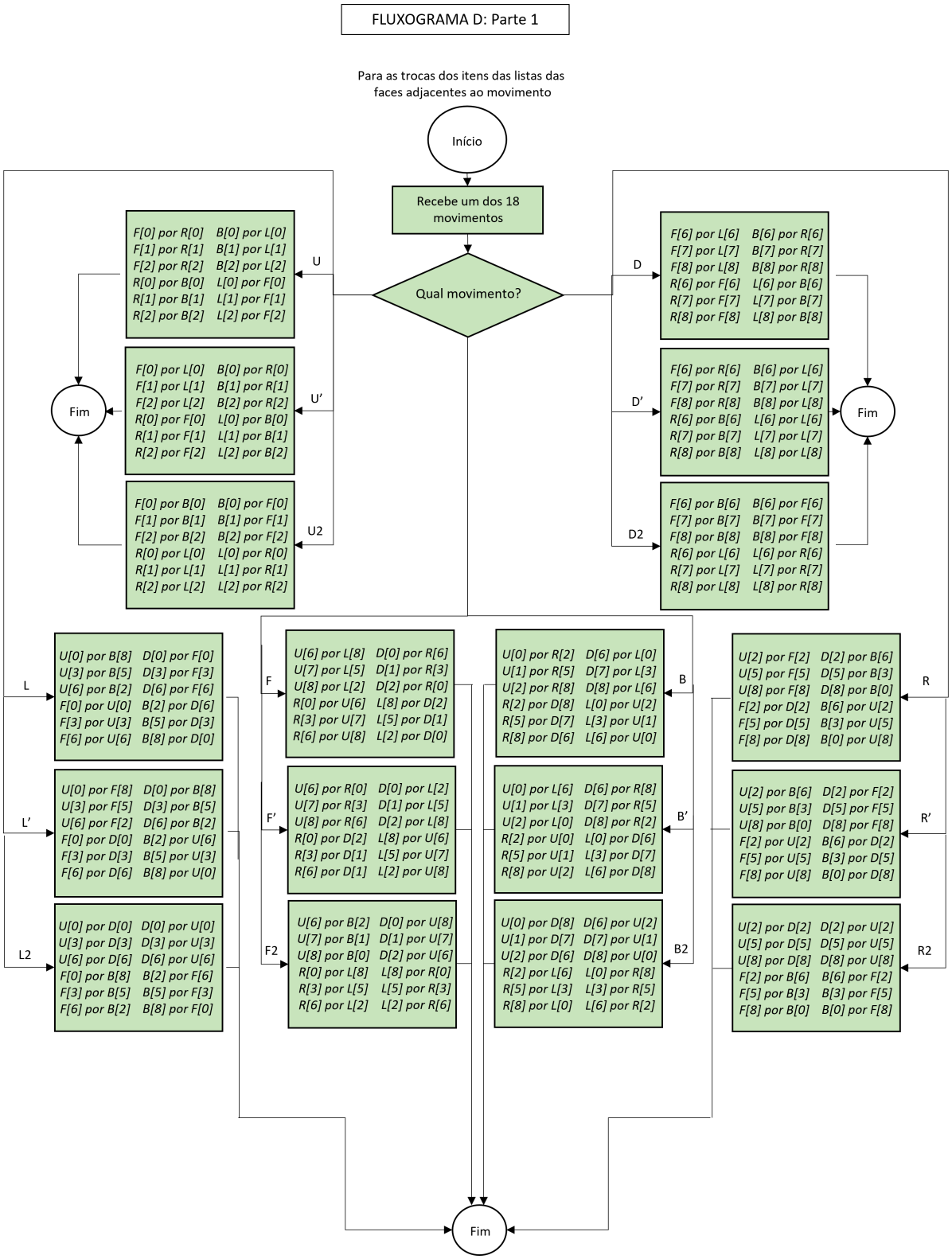


Fonte: Autoria própria.

Essas trocas de posições dos itens podem ser divididas em duas categorias: as trocas dos itens das listas das faces adjacentes ao movimento, que segue o fluxograma da Figura 58 e as trocas na lista da própria face do movimento, seguindo o fluxograma da Figura 59. A visualização do Fluxograma D (Parte 1 e 2) no cubo está apresentada em vídeo[10].

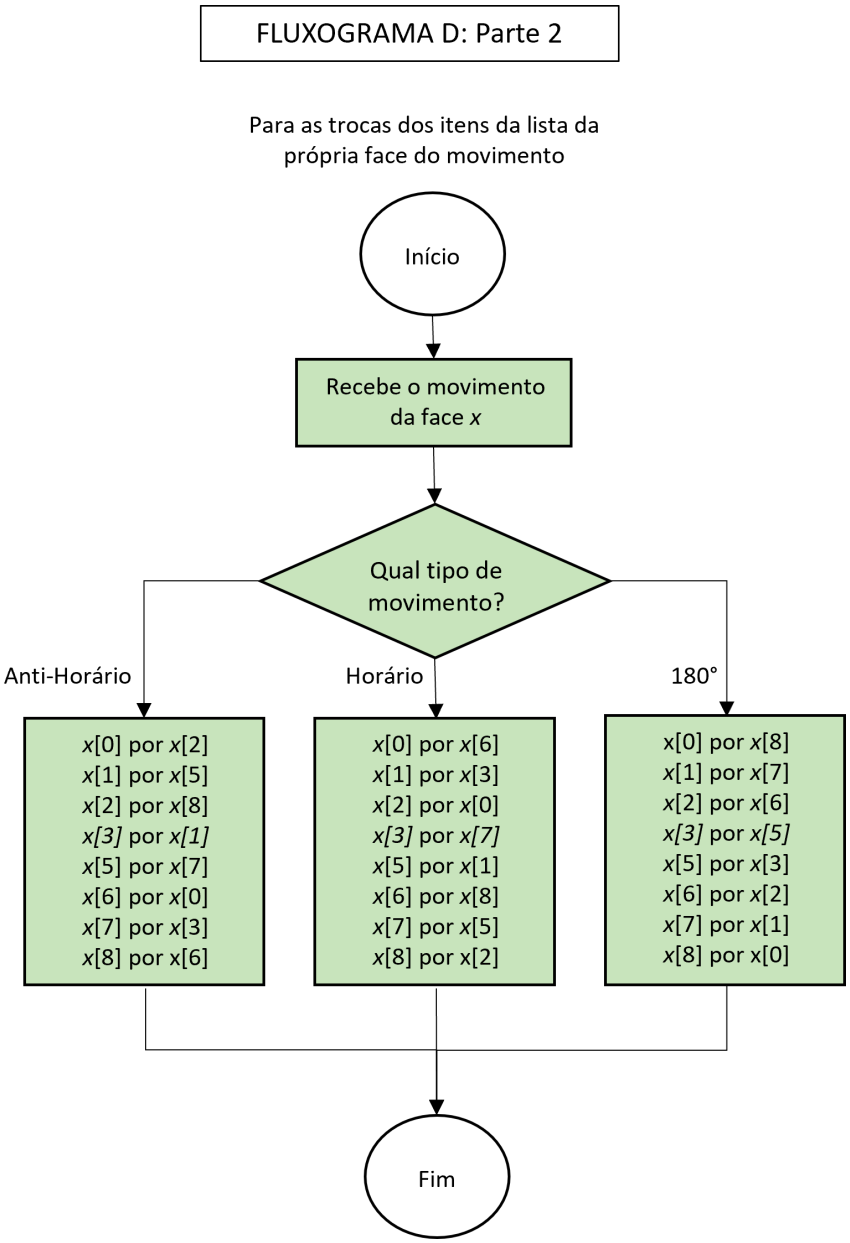
[10] <https://youtu.be/JzfnOCuDKhc>

Figura 58 – Fluxograma D - Parte 1: para construção do código definidor da *string* de estado do cubo: para as trocas dos itens das listas das faces adjacentes.



Fonte: Autoria própria.

Figura 59 – Fluxograma D - Parte 2: para construção do código definidor da *string* de estado do cubo: para os itens da lista da face do movimento.



Fonte: Autoria própria.

Ademais, a união das duas partes do Fluxograma D são implementadas no arquivo *.py*, "Código_final", detalhado por Araujo (2024), por meio das funções: *rotacionar_face_sentido_horario()*, *rotacionar_face_sentido_antihorario()*, *rotacionar_face_180()* e *aplicar_movimento()*.

6.5.2 Definição da sequência esperada de leitura após o embaralhamento

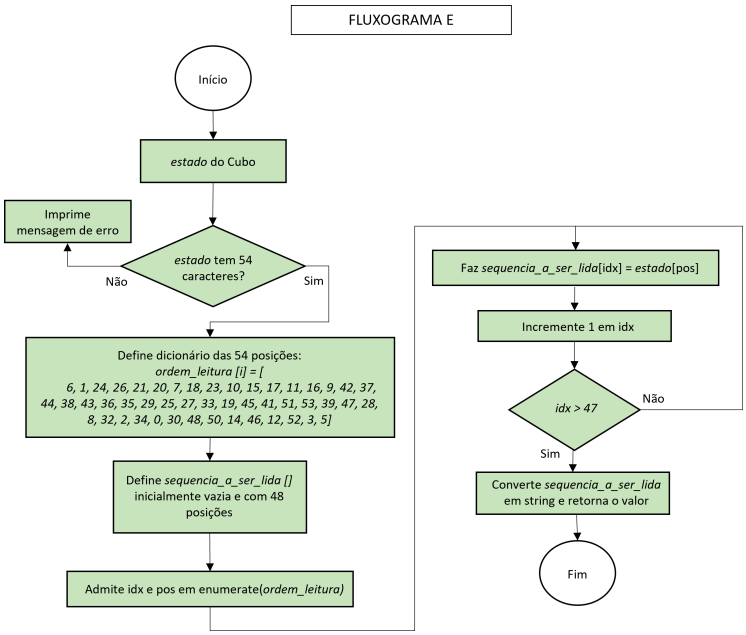
De forma complementar, após obter o código definidor do estado do cubo embaralhado, é necessário desenvolver o código que informe a sequência de leitura esperada para esse estado do cubo. Com intuito de que seja possível, após execução da leitura automática, obter o percentual

de acerto da leitura.

Dessa forma, se implementa o código para a partir da *string* de estado de 54 caracteres se obter a sequência esperada de 48. Para isso, se utiliza, como mostrado na Figura 51, que a sequência lida das posições é 6, 1, 24, 26, 21, 20, 7, 18, 23, 10, 15, 17, 11, 16, 9, 42, 37, 44, 38, 43, 36, 35, 29, 25, 27, 33, 19, 45, 41, 51, 53, 39, 47, 28, 8, 32, 2, 34, 0, 30, 48, 50, 14, 46, 12, 52, 3, 5. E, assim, se faz a organização dos caracteres do estado para a ordem da leitura, seguindo o Fluxograma E, apresentado na Figura 60a, que tem sua implementação na função *estado_para_leitura()* disponível por Araujo (2024), no arquivo *.py*, "Código_final". A Figura 60b apresenta um exemplo de funcionamento da reorganização do cubo embaralhado na *string* de sequência de leitura.

Figura 60 – Fluxograma E: (a) definição da sequência a ser lida para o estado do cubo e (b) exemplo de funcionamento.

(a)



(b)

```
Cubo inicial:              UUUUUUUUURRRRRRRRRFFFFFFFFFDDDDDDDDDLLLLLLLLLBBBBBBBBB
Sequência de embaralhamento: F2 U2 R2 U F2 U2 L' D B2 D2 R2 B' L' B' F' R B D' R
Realizando embaralhamento.....
Embaralhamento finalizado
Cubo embaralhado (54):     UBUDUUUFD FBRFRUDUL RDRRFRRLL BFBRDLFLF LLFRLULDD BDBFBBUBD
Sequência de leitura (48):  UBRLRRFRRBDLRUFLLDFDLFBLBFDBUUDRBFDLULURFBUDFBDU
```

Fonte: Autoria própria.

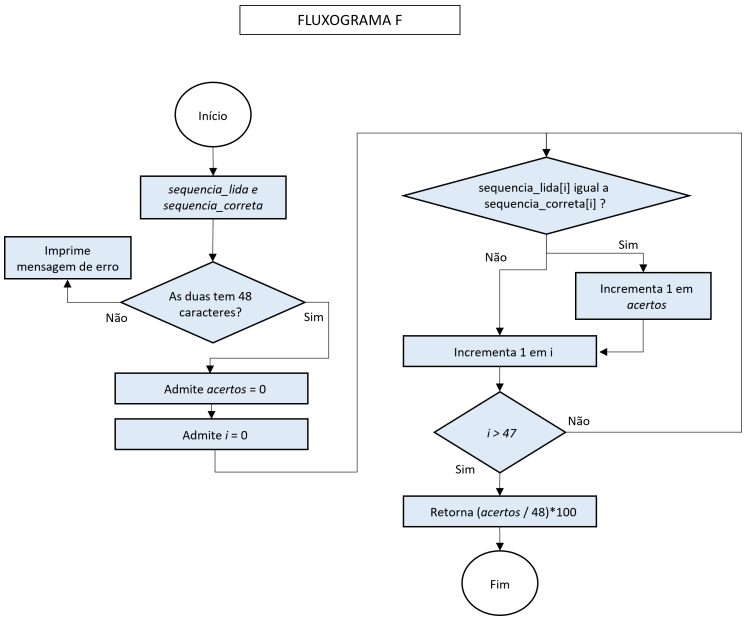
6.5.3 Cálculo do percentual de acerto da leitura

O cálculo do percentual de acerto da leitura é feito comparando os caracteres um a um das duas *strings*: a *string* da sequência lida e a *string* da leitura correta. Sendo assim, é construída a função *calcular_percentual_acerto()*, disponível por Araujo (2024), seguindo o Fluxograma

F, da Figura 61a, o qual possui como entradas as duas sequências de comparação e retorna o percentual de igualdade entre elas. A Figura 61b, apresenta um exemplo desse funcionamento.

Figura 61 – Fluxograma F: (a) fluxograma para cálculo do percentual de acerto da leitura e (b) exemplo de funcionamento.

(a)



(b)

```
Sensor 2: 158   | 47    | 55    | F - Vermelho
Sensor 2: 193   | 110   | 80    | B - Laranja
Sensor 2: 196   | 179   | 126   | D - Amarelo
Sensor 2: 215   | 222   | 218   | U - Branco
Leitura do cubo finalizada.
Essa é a sequência lida (48): UBRLRRFRRBDLRUFLLDFDLFBLBFDBUUBRBFDLULURFUUDFBDU
Essa é a sequência correta (48): UBRLRRFRRBDLRUFLLDFDLFBLBFDBUUDRBFDLULURFBUDFBDU
Número de acertos: 46.00000000000001 de 48
Percentual de acerto: 95.83%
```

Fonte: Autoria própria.

6.6 CÓDIGO FINAL DO ROBÔ

Sendo assim, após as implementações em código de todos os fluxogramas, se obtém o código final do robô realizando a união dos códigos, conforme apresentado no arquivo *.py*, "Código_final" por Araujo (2024). Ele segue o Fluxograma Detalhado, Figura 62a, e, inicialmente, são feitas as definições de cada uma das funções necessárias aos fluxogramas, conforme Tabela 8 e em seguida é executada a função *menu_principal()* que reúne todas as tarefas do robô e é detalhada no fluxograma da Figura 62b.

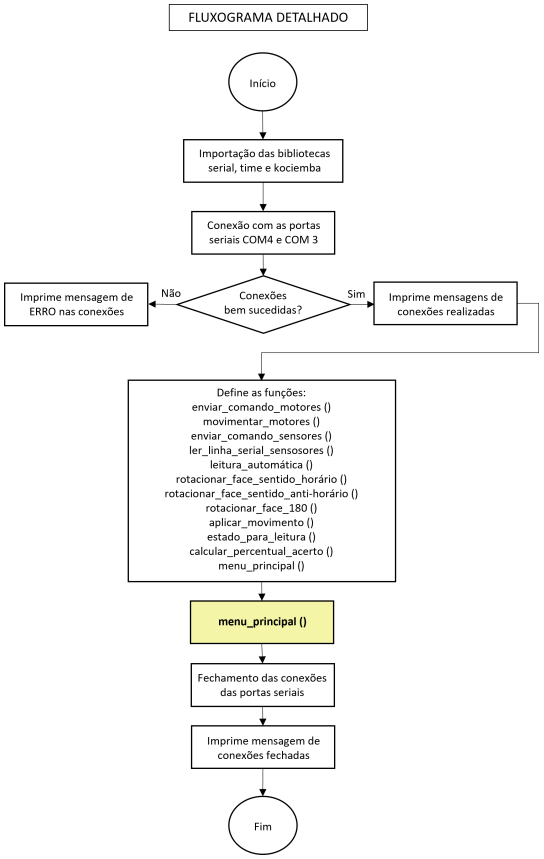
Tabela 8 – Funções do código final do robô e os fluxogramas em que são aplicadas.

Função	Aplicação
enviar_comando_motores()	Fluxogramas A, B e C
movimentar_motores()	Fluxogramas A, B e C
enviar_comando_sensores()	Fluxograma C
ler_linha_serial_sensosores()	Fluxograma C
leitura_automatica()	Fluxograma C
rotacionar_face_sentido_horario()	Fluxograma D
rotacionar_face_sentido_antihorario()	Fluxograma D
rotacionar_face_180()	Fluxograma D
aplicar_movimento()	Fluxograma D
estado_para_leitura()	Fluxograma E
calcular_percentual_acerto()	Fluxograma F
menu_principal()	Fluxograma Detalhado

Fonte: Autoria própria.

Figura 62 – Fluxograma Detalhado do funcionamento do robô: (a) Fluxogarama Detalhado agrupando a função *menu_principal()* e (b) fluxograma da função *menu_principal()*.

(a)



(b)

Fonte: Autoria própria.

7 RESULTADOS

Com a conclusão do código final, foi possível submeter o robô a testes em cada uma de suas funções. Para isso, visando avaliar a consistência das capacidades do robô em diferentes estados do cubo, foram selecionados 10 padrões aleatórios do cubo a partir da geração randômica do Cube Explorer[11], detalhados no Apêndice B. Eles são os modelos para as verificações dos resultados. Sendo assim, cada um deles possui definida: a sua sequência de movimentos geradores do padrão (embaralhamento), uma sequência de movimentos para resolução (solução), a *string* de 54 caracteres (estado do cubo) e a *string* de 48 caracteres (sequência de leitura do cubo).

Dessa forma, os testes são divididos em duas partes:

1. verificação da assertividade da leitura do cubo;
2. verificação da assertividade e do tempo de resolução.

7.1 ASSERTIVIDADE DA LEITURA DO CUBO

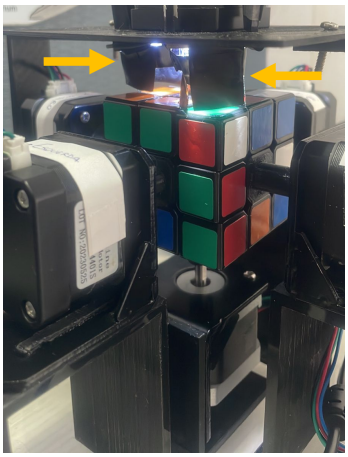
De forma preliminar aos testes, é adicionado, ao redor dos dois sensores, proteções pretas com fita isolante, conforme Figura 63a, visando isolar os sensores da luz ambiente e diminuir a interferência das peças adjacentes, conforme orienta Tsoy (2016). Em seguida, entre os dias 06/09/2024 e 09/09/2024 são coletados valores RGB com cada um dos sensores. Para isso, com a estrutura montada, se movimenta as faces do cubo manualmente e se utiliza o código do Fluxograma C (sensores TCS3200 e o Arduino UNO) para coletar 360 amostras em cada sensor, sendo 60 de cada cor. Tais valores podem ser verificadas no arquivo *.xlsx* denominado "Valores_RGB" disponível em Araujo (2024). A visualização clusterizada dos valores dessas medições das cores com os dois sensores podem ser vistas nas Figuras 63b e 63c.

Diante delas, se verifica que os valores de cores diferentes são bem próximos, principalmente para o Laranja e o Vermelho nos dois sensores. Além disso, se percebe a existência de valores bem distantes do padrão de sua cor, que podem ser considerados como *outliers*. Tal cenário também foi evidenciado por Swiezy e Knopf (2017), e revela o desafio para definição de critérios ou limites dos valores RGB para conseguir classificá-los em cores. E, portanto, reforça e valida a importância do uso de um algoritmo de classificação, como recomendado por Tsoy (2016). No caso deste trabalho, o algoritmo KNN.

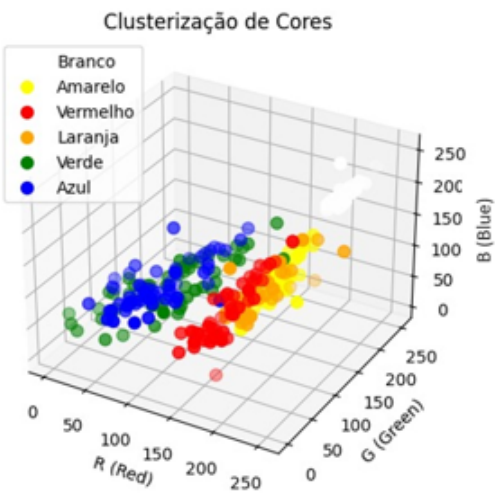
[11] Disponível em: <https://github.com/hkociemba/CubeExplorer/tree/master>

Figura 63 – Medições dos valores RGB antes de iniciar os testes: (a) proteções pretas adicionadas as redor dos sensores e visualização das 720 amostras RGB em *clusters* das cores: (b) para o Sensor 1 e (c) para o Sensor 2.

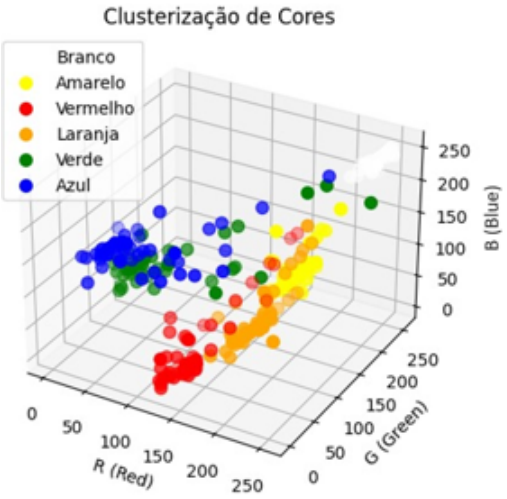
(a)



(b)



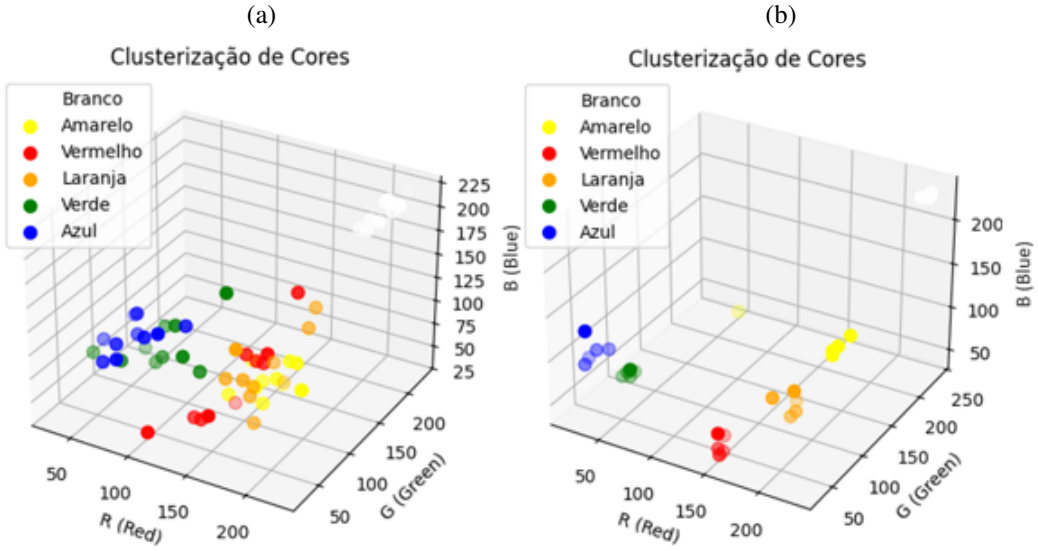
(c)



Fonte: Autoria própria.

Com essa consideração constatada, as 720 amostras são inseridas como vetores nas funções *classificar_cores_1* e *classificar_cores_2*, como determinado na Seção 6.4.2, para se iniciar os testes. No entanto, ao executar o código, o monitor serial apresentou problemas com essa quantidade de amostras. Sendo assim, as matrizes de amostras dos sensores são reduzidas, ficando com as 60 primeiras do Sensor 1 as 30 primeiras do Sensor 2 de cada um das seis cores. A decisão por manter um número de amostras maior no Sensor 1 decorre da tentativa de melhorar a possibilidade de acertos do KNN, dado que os dados do Sensor 1 estão menos agrupados que o Sensor 2. Sendo assim, a nova visualização das amostras dos dois sensores podem ser visualizadas nas Figuras 64a e 64b.

Figura 64 – Redução das amostras: (a) nova visualização das amostras para o Sensor 1 e (b) nova visualização das amostras para o Sensor 2.



Fonte: Autoria própria.

Após a redução das amostras, entre os dias 11/09/2024 e 12/09/2024 são executados quatro testes de funcionamento do robô em cada um dos 10 padrões, totalizando 40 testes.

Neles, são executados o Fluxograma detalhado do robô e, em cada um, é enviada a devida sequência de embaralhamento do padrão, quando solicitado, e na opção de escolha pela leitura é enviado "S" (Sim), garantindo que a leitura aconteça. Dessa forma, são coletados os dados de quantidade de acertos em cada teste, bem como os valores RGB identificados em cada cor. Os valores identificados podem ser verificadas no arquivo *.xlsx* denominado "Valores_RGB_TESTES" disponível por Araujo (2024). De forma mais aprofundada, nesse arquivo é apresentado um relatório detalhado da quantidade de acertos e percentual de acertos em cada teste e padrão, por sensor e por cores. A Tabela 9 apresenta um relatório resumido do desempenho das leituras por sensores e cores. A visualização do robô executando um exemplo de leitura pode ser encontrada em vídeo [12].

Tabela 9 – Relatório resumido dos acertos da leitura nos 40 testes realizados com o robô.

-	VERMELHO	LARANJA	BRANCO	AMARELO	VERDE	AZUL	TOTAL
SENSOR 1	82 de 148	85 de 148	148 de 148	89 de 148	118 de 148	109 de 148	**631 de 888**
SENSOR 2	111 de 148	108 de 148	143 de 148	129 de 148	119 de 148	134 de 148	**744 de 888**
TOTAL GERAL	193 de 296	193 de 296	291 de 296	218 de 296	237 de 296	243 de 296	**1.375 de 1.776**
%	65,20%	65,20%	98,31%	73,65%	80,07%	82,09%	**77,42%**

Analisando os dados do relatório detalhado descrito no arquivo *.xlsx*, "Valores_RGB_TESTES", disponível por Araujo (2024), é possível realizar as seguintes constatações:

[12] <https://youtu.be/cCYAtc2piww>

1. O robô obteve a melhor média de percentual de acerto no Padrão 10, no valor de 92,32%, e a pior média no Padrão 6, no valor de 48,18%, conforme apresenta da Figura 65a;
2. O percentual geral de acerto considerando todos os 37 testes concluídos foi de 77,42%, com 1.375 acertos na cor de 1.776 peças, conforme apresenta a Figura 65b.
3. O melhor resultado foi de 46 acertos no Teste 3 do Padrão 10, e o pior foi de 20 acertos no Teste 3 do Padrão 6, como mostrado na Figura 65c;
4. O Sensor 2 apresentou uma assertividade de 83,78%, consideravelmente melhor que o Sensor 1, de 71,06%, como apresentado na Figura 65d;
5. O ranking de assertividade por cores no Sensor 1 foi: Branco (100%), Verde (80%), Azul (74%), Amarelo (60%), Laranja (57%) e Vermelho (55%), conforme a Figura 65e;
6. O ranking de assertividade por cores no Sensor 2 foi: Branco (97%), Azul (91%), Amarelo (87%), Verde (80%), Vermelho (75%) e Laranja (73%), conforme a Figura 65f;
7. No Padrão 9, apenas um teste foi concluído, pois nos outros três aconteceram erros na movimentação do cubo.
8. A mediana de acertos considerando todos os 37 testes concluídos foi de 39 acertos.

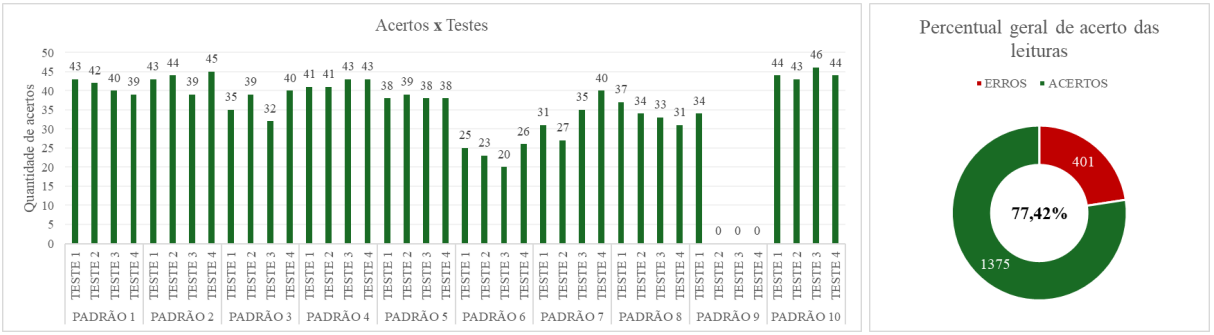
Com tais análises, também fica evidente que, mesmo com uma redução maior das amostras na função *classificar_cores_2*, o Sensor 2 ainda obteve um desempenho superior ao Sensor 1. Essa situação levanta a hipótese de que caso tivesse mais amostras no código, o percentual de acerto poderia ser melhor.

Por outro lado, o Sensor 1, especificamente para as cores Vermelho e Laranja, tiveram um nível de acerto muito oscilante e foram decisivas para obter um percentual geral inferior aos 80%. Em geral, os erros das duas se concentraram na troca de classificação entre elas mesmas.

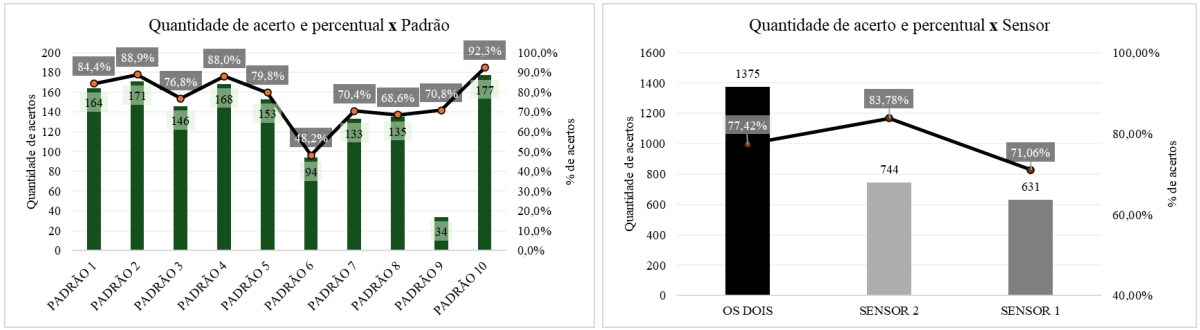
Como explicação para esse desempenho, se pode verificar a Figura 66, com as visualizações em 3D dos dados coletados nos testes, retirados no arquivo *.xlsx* "Valores_RGB_TESTES" de Araujo (2024). Por essas visualizações, se percebe que os valores do Sensor 1 estiveram de fato muito próximos, especialmente nas cores Vermelho e Laranja. Já para o Sensor 2, há uma distribuição melhor dos grupos, isto é, apesar de alguns pontos *outliers*, é possível dizer que os valores se concentram melhor em cada uma das cores.

Figura 65 – Gráficos dos resultados das leituras realizadas nos 40 testes: (a) gráfico da quantidade de acertos de cada teste dos padrões, (b) gráfico do percentual geral de acertos, (c) gráfico da quantidade e percentual de acertos em cada padrão, (d) gráfico do quantidade e percentual de acertos em cada sensor, (e) gráfico da quantidade e percentual de acertos em cada cor no Sensor 1 e (f) gráfico dos da quantidade e percentual de acertos em cada cor no Sensor 2.

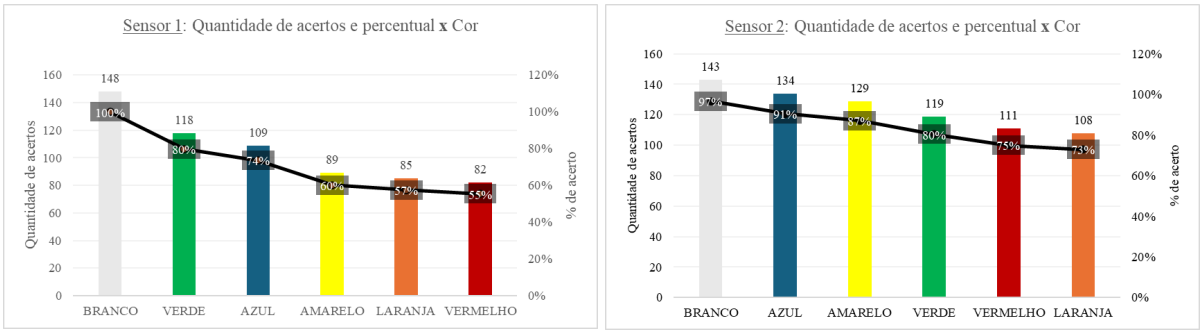
(a) (b)



(c) (d)

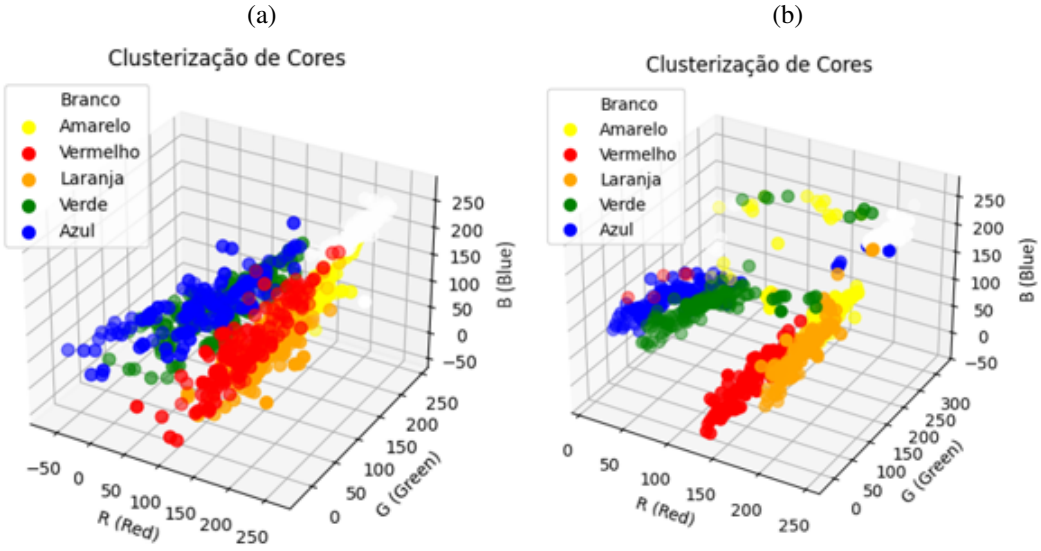


(e) (f)



Fonte: Autoria própria.

Figura 66 – Visualização dos valores coletados nos 1776 testes: (a) 888 amostras do Sensor 1 e (b) 888 amostras do Sensor 2.



Fonte: Autoria própria.

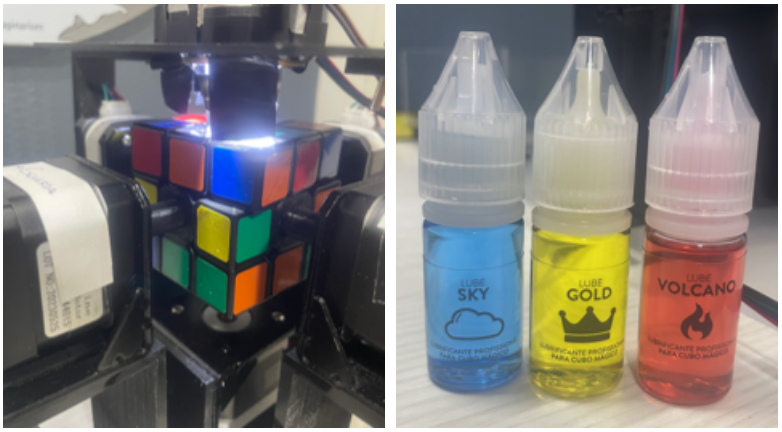
De forma complementar, no aspecto da movimentação para realização das leituras, o robô obteve sucesso completo em 37 dos 40 testes, o que resulta num percentual de 92,5%. Num aspecto mais aprofundado, os três erros nos testes foram no Padrão 9 e aconteceram durante execução do movimento U, em que o motor superior (Extrusora 2) não conseguiu completar os passos necessários, o que impediu os movimentos posteriores para conclusão da leitura. A principal causa levantada para esse tipo de problema foi o fato de que alguns movimentos não são completados com precisão, em razão do atrito entre as peças. O que resulta numa situação de ter uma face ligeiramente inclinada, conforme o exemplo da Figura 67a, o que obstaculiza o movimento e impede o motor de completar os passos do comando, e, portanto, impossibilita os movimentos posteriores.

Para prevenir esse tipo de erro, no início dos testes, foram aplicados lubrificantes profissionais[13] do cubo de Rubik, mostrados na Figura 67b, que contribuem para superar situações como essas, deixando o cubo mais ágil. No entanto, mesmo o cubo de Rubik absorvendo esse tipo de situação na maioria dos testes, no caso do Padrão 9, não foi possível.

[13] <https://cuberbrasil.mercadoshops.com.br/MLB-1740947724-lubrificante-para-cubo-magico-cuber-pro-sky-gold-volcano-_JM>

Figura 67 – Inclinações nas faces do cubo: (a) exemplo e (b) lubrificantes utilizados para tornar o cubo mais ágil.

(a) (b)



Fonte: Autoria própria.

7.2 ASSERTIVIDADE E VELOCIDADE DA RESOLUÇÃO DO CUBO

A avaliação da assertividade e da velocidade do robô para realizar a resolução do cubo, em parte, já pôde ser feita junto com os testes da leitura. Pois a resolução foi realizada com sucesso em todos 37 testes que chegaram nessa etapa.

A média de tempo de execução dos testes foi de 3 minutos e 30 segundos, considerando as etapas de embaralhamento, leitura e resolução. Em cada um dos testes, são executados, aproximadamente, cerca de 100 movimentos no cubo. Sendo 20 para o embaralhamento, 60 para leitura e 20 (na maioria dos casos) para resolução. Além disso, cada movimento estava com um intervalo de envio dos comandos dos movimentos de 0,500 segundo.

Visando avaliar estritamente a tarefa de resolução, são realizados mais testes, apenas dessa etapa, a partir do código “TESTE_DE_VELOCIDADE” disponível em Araujo (2024). Com o objetivo de encontrar o limite máximo de velocidade que o robô consegue resolver o cubo. Para isso, se utilizam novamente dos 10 padrões descritos no Apêndice B, em quatro ciclos de testes. Cada ciclo propõe um intervalo entre os comandos, sendo eles de: 0,500 segundo, 0,100 segundo, 0,0500 segundo e 0,0450 segundo. A Tabela 10 apresenta um relatório dos resultados dos testes em cada padrão e exemplos do robô realizando resoluções podem ser visualizados em vídeo [14].

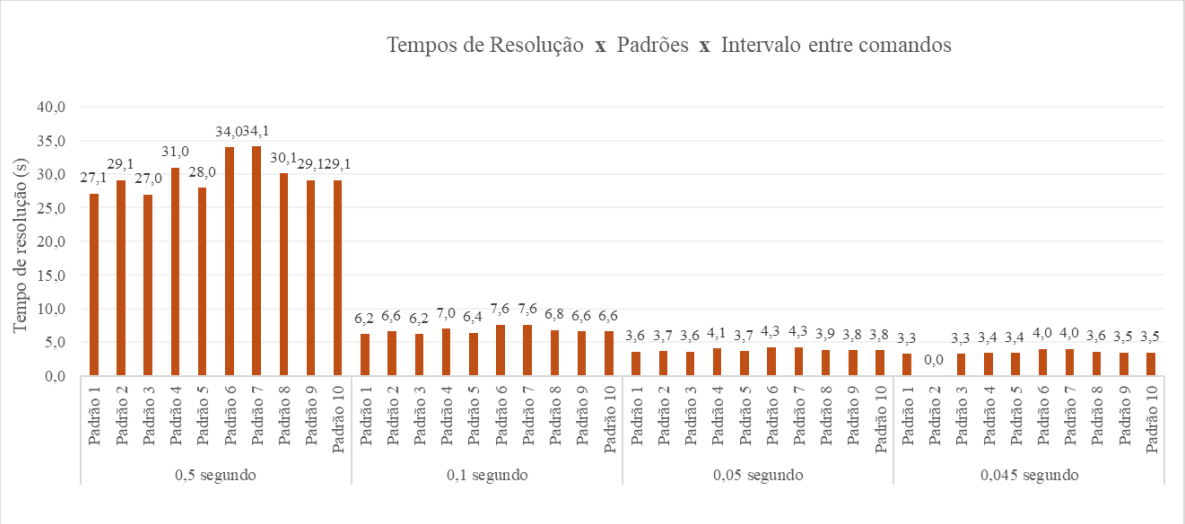
A partir desses dados, é possível verificar uma acentuada diminuição no tempo de resolução, a medida que o intervalo entre os comandos é reduzido, como apresenta o gráfico da Figura 68. Assim, a média de tempo é reduzida de 29,86 segundos para 3,56 segundos.

[14] <https://youtu.be/HBaynMq1-yQ>

Tabela 10 – Relatório dos 40 testes realizados com o robô visando avaliar velocidade máxima de resolução.

PADRÃO	Intervalo de 0,5 segundo		Intervalo de 0,1 segundo		Intervalo de 0,050 segundo		Intervalo de 0,045 segundo	
	Execução	Tempo (s)	Execução	Tempo (s)	Execução	Tempo (s)	Execução	Tempo (s)
Padrão 1	OK	27,1 s	OK	6,2 s	OK	3,6 s	OK	3,3 s
Padrão 2	OK	29,1 s	OK	6,6 s	OK	3,7 s	KO	-
Padrão 3	OK	27,0 s	OK	6,2 s	OK	3,6 s	OK (2ª tentativa)	3,3
Padrão 4	OK	31,0 s	OK	7,0 s	OK	4,1 s	2ª tentativa	3,4 s
Padrão 5	OK	28,0 s	OK	6,4 s	OK	3,7 s	OK	3,4 s
Padrão 6	OK	34,0 s	OK	7,6 s	OK	4,3 s	OK	4,0 s
Padrão 7	OK	34,1 s	OK	7,6 s	OK (2ª tentativa)	4,3 s	OK (2ª tentativa)	4,0 s
Padrão 8	OK	30,1 s	OK	6,8 s	OK	3,9 s	OK	3,6 s
Padrão 9	OK	29,1 s	OK	6,6 s	OK	3,8 s	OK	3,5 s
Padrão 10	OK	29,1 s	OK	6,6 s	OK	3,8 s	OK	3,5 s
Média	100%	29,86 s	100%	6,76 s	90%	3,88 s	60%	3,56 s

Figura 68 – Tempos de resolução por resolução por padrão e por intervalos entre os comandos.

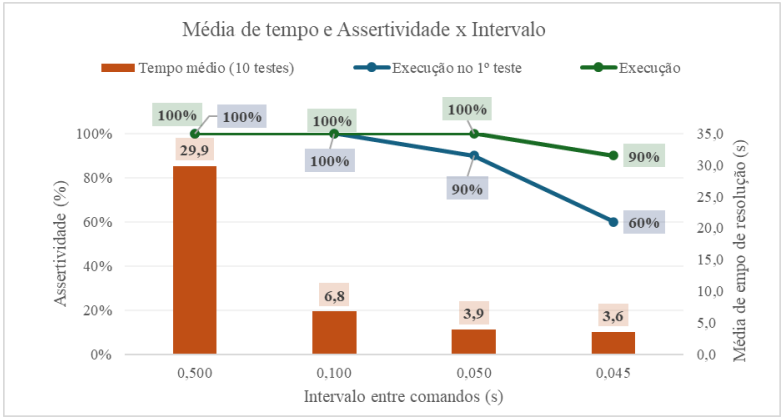


Fonte: Autoria própria.

É válido destacar que, como o intervalo é entre comandos, os motores que necessitam de mais comandos levam mais tempo para completar seus movimentos. Em particular, os motores definidos como extrusoras (E0, E1 e E2), que correspondem, respectivamente, aos motores Esquerdo, Traseiro e Superior, resultam em tempos de movimento mais longos. Devido a esse fato, pode-se afirmar que soluções que envolvem menos movimentos dessas faces resultam em tempos de resolução menores.

Por outro aspecto, a medida que a velocidade de resolução aumenta, a confiabilidade de executar a resolução reduz. De forma que, com os intervalos de 0,500 segundo e 0,100 segundo todos os testes foram executados sem erros. Mas, com um intervalo de 0,050 segundos, 90% deles que funcionaram na 1ª execução e, com um intervalo de 0,045 segundo, o número reduz para 60%, como apresenta o gráfico da Figura 69.

Figura 69 – Relação entre média de tempo de resolução e assertividade por intervalo entre os comandos.

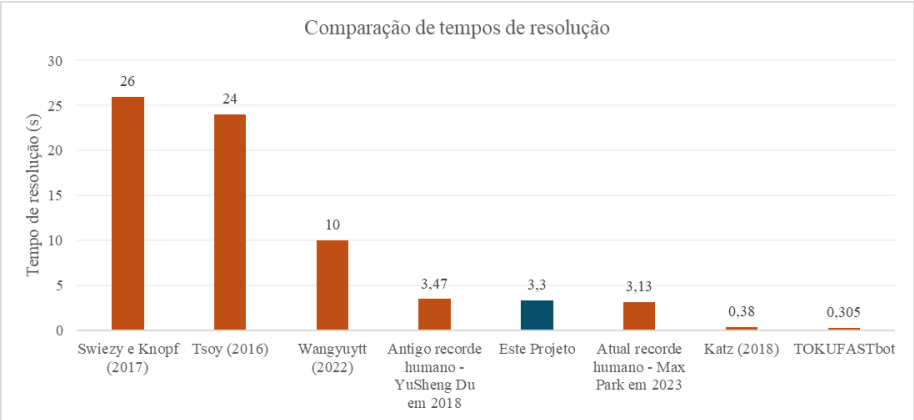


Fonte: Autoria própria.

De toda forma, esse comportamento é justificável, uma vez que intervalos extremamente curtos entre envio dos comandos levam a uma maior simultaneidade no giro dos motores e das faces do cubo, o que, devido ao atrito entre as peças, pode resultar no travamento do mecanismo. O Padrão 2 com intervalo de 0,045 segundo não foi concluído por ocorrência dessa situação.

Além disso, se verifica que o melhor resultado de tempo de resolução do robô foi de 3,3 segundos, em dois casos: para os Padrões 1 e 3 com intervalo de 0,045 segundo. Num aspecto comparativo, pela Figura 70, é possível verificar como este resultado se posiciona a outros tempos de resolução do cubo de Rubik. Entre eles, os trabalhos referências na área e os recordes humanos de resolução, o antigo de 3,47s e o atual, de 3,13s [15].

Figura 70 – Comparação entre tempos de resolução entre robôs.



Fonte: Autoria própria.

15 <https://www.guinnessworldrecords.com/world-records/72863-fastest-time-to-solve-a-rubiks-cube>

8 CONCLUSÕES

Este trabalho teve como objetivo principal desenvolver um robô solucionador do cubo de Rubik, visando a melhor combinação dos aspectos de velocidade de resolução e simplicidade na leitura, que realize a identificação das cores das peças e, em seguida, com uso de um algoritmo eficiente, encontre a solução de movimentos e os implemente no cubo para resolvê-lo. Ao longo do desenvolvimento, foram abordados temas importantes como projeto mecânico, seleção, conexão e montagem de *hardware*, assim como implementações de algoritmos e desenvolvimento de *firmware* e *software*. A partir das implementações, testes e análises realizadas, foi possível obter um robô que consegue ter uma mediana de 39 acertos para leitura das peças do cubo e resolvê-lo, no tempo mínimo de 3,3 segundos.

As contribuições deste trabalho são relevantes e variadas. Inicialmente, se tem a construção de um conceito novo para o estrutura mecânica de um robô solucionador do cubo de Rubik. Em seguida, o desenvolvimento de um projeto de *hardware* que, com a conexão entre um computador e microcontroladores, consegue utilizar sensores de cor para coletar dados do cubo e, com motores de passo, movimentar suas faces de forma ordenada e inteligente. Além disso, há a estruturação de um método próprio para realização da leitura, com dois sensores, dos 48 quadrados do cubo em apenas 60 movimentos.

Outrossim, se tem implementações em código que, a partir de um embaralhamento aleatório fornecido, consegue definir o estado do cubo e, assim, automatizar o cálculo do percentual de acerto da leitura. De forma complementar, existe a implementação do algoritmo KNN no Arduino como método de classificação dos valores RGB dos sensores em cores.

Ademais, acerca dos resultados, o projeto obteve um desempenho muito interessante no tempo de resolução e até mesmo superior aos projetos que são referências deste trabalho. De forma similar, a assertividade da leitura das cores, mesmo não conseguindo alcançar o nível de 48 acertos de interesse, oferece números muito positivos para conseguir identificar o estado do cubo, como uma mediana de 39 acertos. Pois, pela teoria do cubo, são necessárias apenas 40 leituras corretas para mapear o cubo, dado que 8 cores de peças podem ser previstas por exclusão. Dado que, nos casos dos 8 cantos, com duas cores conhecidas, a terceira de cada um já se pode definir.

Numa outra frente, o desenvolvimento deste robô também representa uma contribuição no campo da automação e robótica, destacando-se pela simplicidade na implementação e pela vivência de um tema de destaque no seguimento dos robôs manipuladores, com potencial de aplicações educacionais.

8.1 TRABALHOS FUTUROS

Por fim, cabe destacar que este projeto pode ser aperfeiçoado com melhorias em trabalhos futuros em três categorias de complexidade:

1. **Melhorias mais simples e acessíveis:**
    - Inserção do processo de classificação dos valores RGB em cores dentro do computador. O qual, devido sua maior capacidade de memória, com o aumento do número de amostras dos valores de cada cor, promoverá um melhor acerto do algoritmo KNN;
    - Adaptação da estrutura mecânica para um conceito *plug and play* que melhore o acesso aos dispositivos do robô mesmo com a estrutura montada;
    - Desenvolvimento de um conceito mais eficaz para proteção dos sensores para reduzir a influência das cores das peças adjacentes.
2. **Melhorias de complexidade intermediária:**
    - Otimização do método construído para leitura, reduzindo etapas com a utilização de mais sensores;
    - Desenvolvimento de um processo de calibração automática, em que as amostras dos valores RGB de cada cor sejam coletadas de forma autônoma e, posteriormente, utilizadas para realização da classificação;
    - Implementação de um algoritmo de busca para que o robô não dependa das 48 leituras, mas que consiga, considerando as exclusões, com apenas 40 leituras, definir o estado do cubo;
    - Utilização de um mini-computador de placa única multiplataforma que consiga realizar as funções do computador, a fim de tornar o projeto mais independente e com um *hardware* de menor tamanho.
3. **Melhorias avançadas e desafiadoras:**
    - Uso de tecnologias como *Wi-Fi* e *Bluetooth* para transferir os dados, permitindo a interação com o robô via aplicativo ou interface *web*;
    - Desenvolvimento de uma interface de usuário amigável para visualização em tempo real do progresso da leitura do cubo embaralhado, oferecendo um *feedback* visual claro e interativo sobre cada leitura de cor que é realizada;
    - Utilização de motores de passo com *encoders* acoplados, visando obter uma malha fechada no controle dos movimentos do robô, com informação de *feedback* da rotação do eixo, para melhorar a precisão de giro dos motores e a velocidade de resolução.

REFERÊNCIAS

ALLEGRO. **DMOS Microstepping Driver with Translator And Overcurrent Protection**. s.d. Disponível em: <https://www.pololu.com/file/0J450/a4988_DMOS_microstepping_driver_with_translator.pdf>. Acessado em: 15 agosto 2024. Citado na página 41.

ANUPAMA, P. et al. An intelligent reflective colour sensor system for paper and textile industries. In: IEEE. **2012 Sixth International Conference on Sensing Technology (ICST)**. [S.l.], 2012. p. 481–485. Citado na página 14.

ARAUJO, L. A. d. **Robô Solucionador do Cubo de Rubik com uso de motores de passo e sensores de cor**. 2024. Disponível em: <https://github.com/Lucas-Arnaud/TCC_Rob-_Cubo_Rubik/tree/main>. Acessado em: 24 setembro 2024. Citado 9 vezes nas páginas 57, 67, 71, 72, 73, 76, 78, 79 e 82.

ARDUINO. **Arduino**. 2023. Disponível em: <https://www.arduino.cc/>. Acessado em: 30 outubro 2023. Citado 3 vezes nas páginas 40, 41 e 42.

ARDUINO. **Arduino UNO R3**. 2024. Disponível em: <https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf>. Acessado em: 31 maio 2024. Citado 2 vezes nas páginas 37 e 39.

BEAGABREW. **Fonte Eletrônica 12v 5a Bivolt + Plug P4 Femea**. 2014. Disponível em: <https://www.bhbrew.com.br/fonte-eletronica-12v-5a-bivolt-plug-p4-femea>. Acessado em: 20 agosto 2024. Citado 2 vezes nas páginas 40 e 44.

BITENCOURT, L. **Teoria dos Grafos – Uma introdução**. 2016. Disponível em: <https://luizbitencourt.wordpress.com/2016/08/05/teoria-dos-grafos-uma-introducao/>. Acessado em: 15 maio 2024. Citado na página 22.

BOGUE, R. The role of robots in entertainment. **Industrial Robot: the international journal of robotics research and application**, Emerald Publishing Limited, v. 49, n. 4, p. 667–671, 2022. Citado na página 13.

CEZARIO, A. D. P.; MIURA, S. M. D. N. Visão computacional e microcontroladores aplicação na resolução do cubo de rubik. **REVISTA DE ENGENHARIA E TECNOLOGIA**, v. 13, n. 4, 2021. Citado 2 vezes nas páginas 19 e 20.

CHEN, J. Different algorithms to solve a rubik's cube. In: IOP PUBLISHING. **Journal of Physics: Conference Series**. [S.l.], 2022. v. 2386, n. 1, p. 012018. Citado 2 vezes nas páginas 21 e 22.

CNC, I. **Motor de Passo Nema 17 4,2 kgf.cm 17hs4401S**. s.d. Disponível em: <https://www.impactocnc.com/motor-de-passo-nema-17-hs-4401-4.2kgf?srsltid=AfmBOorjP-aIFvXJVSMXfCe5GETvRTMJeVl5A_tGNBMxbHlC6vIA7GP_>. Acessado em: 30 outubro 2023. Citado 2 vezes nas páginas 40 e 58.

CONDIT, R.; JONES, D. W. Stepping motors fundamentals. **Microchip Inc. Publication AN907**, p. 1–22, 2004. Citado na página 40.

CONSTANDINOU, T. G. Stepper motors uncovered (1). **Elektor Electronics**, v. 11, p. 36–40, 2013. Citado na página 40.

CUTHBERTSON, A. **Robot smashes Rubik's Cube world record after solving puzzle in literally the blink of an eye.** 2024. Disponível em: <https://www.independent.co.uk/tech/robot-rubiks-cube-world-record-time-b2551994.html>. Acessado em: 31 maio 2024. Citado 2 vezes nas páginas 19 e 24.

DAN, V.; HARJA, G.; NAŞCU, I. Advanced rubik's cube algorithmic solver. In: IEEE. **2021 7th International Conference on Automation, Robotics and Applications (ICARA)**. [S.l.], 2021. p. 90–94. Citado na página 13.

FILHO, O. M.; NETO, H. V. **Processamento digital de imagens**. [S.l.]: Brasport, 1999. Citado 2 vezes nas páginas 37 e 38.

FUKUNAGA, K.; NARENDRA, P. M. A branch and bound algorithm for computing k-nearest neighbors. **IEEE transactions on computers**, IEEE, v. 100, n. 7, p. 750–753, 1975. Citado 2 vezes nas páginas 19 e 28.

GRIGSBY, S. S. Artificial intelligence for advanced human-machine symbiosis. In: SPRINGER. **Augmented Cognition: Intelligent Technologies: 12th International Conference, AC 2018, Held as Part of HCI International 2018, Las Vegas, NV, USA, July 15-20, 2018, Proceedings, Part I**. [S.l.], 2018. p. 255–266. Citado na página 65.

INDUSTRIES, D. **BricKuber Project – A Raspberry Pi Rubiks Cube Solving Robot**. s.d. Disponível em: <https://www.dexterindustries.com/projects/brickuber-project-raspberry-pi-rubiks-cube-solving-robot-project/>. Acessado em: 31 maio 2024. Citado 3 vezes nas páginas 19, 24 e 25.

JBRAZIO. **What is Marlin?** 2021. Disponível em: <https://marlinfw.org/docs/basics/introduction.html>. Acessado em: 20 julho 2024. Citado 2 vezes nas páginas 42 e 43.

KASPRZAK, W.; SZYNKIEWICZ, W.; CZAJKA, Ł. Rubik's cube reconstruction from single view for service robots. **Machine Graphics & Vision International Journal**, Polish Academy of Sciences Warsaw, Poland, Poland, v. 15, n. 3, p. 451–459, 2006. Citado na página 13.

KATZ, B. **The Rubik's Contraption**. 2018. Disponível em: <https://build-its-inprogress.blogspot.com/2018/03/the-rubiks-contraption.html>. Acessado em: 20 maio 2024. Citado 3 vezes nas páginas 19, 23 e 24.

KAUR, H. **Algorithms for solving the Rubik's cube: A study of how to solve the Rubik's cube using two famous approaches: The Thistlewaite's algorithm and IDA* algorithm.** 2015. Citado 3 vezes nas páginas 19, 21 e 23.

KEMP, C. C.; EDSINGER, A.; TORRES-JARA, E. Challenges for robot manipulation in human environments [grand challenges of robotics]. **IEEE Robotics & Automation Magazine**, IEEE, v. 14, n. 1, p. 20–29, 2007. Citado na página 13.

KHEMANI, C. et al. Solving rubik's cube using graph theory. In: **Computational Intelligence: Theories, Applications and Future Directions-Volume I: ICCI-2017**. [S.l.]: Springer, 2018. p. 301–317. Citado 2 vezes nas páginas 21 e 22.

KOCIEMBA, H. **The Subgroup H and its cosets**. s.d. Disponível em: <https://kociemba.org/cube.htm#_ts1722861853693>. Acessado em: 20 maio 2024. Citado 2 vezes nas páginas 19 e 23.

KORF, R. E. Finding optimal solutions to rubik's cube using pattern databases. In: **AAAI/IAAI**. [S.l.: s.n.], 1997. p. 700–705. Citado 2 vezes nas páginas 21 e 22.

LEE, H.; ENRIQUEZ, J. L.; LEE, G. Robotics 4.0: Challenges and opportunities in the 4th industrial revolution. **Journal of Internet Services and Information Security (JISIS)**, v. 12, n. 4, p. 39–55, 2022. Citado na página 13.

LOPES, J. M. B. Cor e luz. **Texto elaborado para a disciplina de Computação Gráfica**, p. 34, 2013. Citado 2 vezes nas páginas 37 e 38.

LOZANO-PEREZ, T. **Autonomous robot vehicles**. [S.l.]: Springer Science & Business Media, 2012. Citado na página 13.

MATARIĆ, M. J. **Introdução à robótica**. [S.l.]: Editora Blucher, 2014. Citado na página 13.

MORGAN, A. A. et al. Robots in healthcare: a scoping review. **Current robotics reports**, Springer, v. 3, n. 4, p. 271–280, 2022. Citado na página 13.

NAYYAR, A.; PURI, V. A review of arduino board's, lilypad's & arduino shields. In: IEEE. **2016 3rd international conference on computing for sustainable global development (INDIACom)**. [S.l.], 2016. p. 1485–1492. Citado na página 39.

NÚÑEZ-COLÍN, C. et al. Construcción de dendrogramas de taxonomía numérica mediante el coeficiente de distancia $\chi 2$: una revisión. **Revista Chapingo Serie Horticultura**, v. 10, n. 2, p. 229–237, 2004. Citado na página 65.

OLIVEIRA, G. d. C. **Sensor de cores RGB para determinações colorimétricas: avaliação e análise quantitativa de soluções coloridas**. 74 f. Monografia (Doutorado) — Instituto de Química, Programa de Pós-Graduação em Química, Universidade Federal de Uberlândia, 2022. Citado na página 37.

PACHECO, A. **K vizinhos mais próximos - KNN**. 2017. Disponível em: <https://computacaointeligente.com.br/algoritmos/k-vizinhos-mais-proximos/>. Acessado em: 08 setembro 2024. Citado 3 vezes nas páginas 19, 28 e 64.

RAMBAUSKE, A. M. Decoração e design de interiores: Teoria da cor. **Instituto de Artes (IAR) da Unicamp. Campinas**, 1985. Citado 2 vezes nas páginas 37 e 38.

RCR3D. **INTRODUCTION**. s.d. Disponível em: <https://www.rcr3d.com/intro.html#intro_01>. Acessado em: 31 maio 2024. Citado 3 vezes nas páginas 19, 24 e 25.

REID, M. New upper bounds. **Cube Lovers**, v. 7, 1995. Citado na página 23.

RERAP. **RAMPS 1.4**. 2024. Disponível em: <https://reprap.org/wiki/RAMPS_1.4>. Acessado em: 20 julho 2024. Citado 4 vezes nas páginas 40, 42, 43 e 47.

ROKICKI, T. Twenty-five moves suffice for rubik's cube. **arXiv preprint arXiv:0803.3435**, 2008. Citado 2 vezes nas páginas 19 e 23.

ROKICKI, T. et al. The diameter of the rubik's cube group is twenty. **SIAM Review**, v. 56, n. 4, p. 645–670, 2014. Disponível em: <https://doi.org/10.1137/140973499>. Citado na página 21.

RYNDACK. **Placa Driver Motor de Passo A4988 com Dissipador**. s.d. Disponível em: <https://www.ryndackcomponentes.com.br/placa-driver-motor-de-passo-a4988-com-dissipador.html>. Acessado em: 15 agosto 2024. Citado 2 vezes nas páginas 40 e 41.

SACRAMENTO, G. **Clusterização de dados: entenda o conceito e formas de uso**. 2023. Disponível em: <https://blog.somostera.com/data-science/clusterizacao-de-dados>. Acessado em: 08 setembro 2024. Citado 2 vezes nas páginas 19 e 27.

SENDA, M. **Japanese company's robot solves a puzzle cube in 0.305 seconds to break record**. 2024. Disponível em: <https://www.guinnessworldrecords.com/news/commercial/2024/5/japanese-companys-robot-solves-a-puzzle-cube-in-0-305-seconds-to-break-record>. Acessado em: 31 maio 2024. Citado na página 24.

SHIN, S.-J.; SUH, S.-H.; STROUD, I. Reincarnation of g-code based part programs into step-nc for turning applications. **Computer-Aided Design**, Elsevier, v. 39, n. 1, p. 1–16, 2007. Citado na página 43.

SILVA, J. V. d. N. **Uma proposta de aprendizagem usando o cubo mágico em Malta–PB**. 71 f. Monografia (Mestrado) — Programa de Mestrado Profissional em Matemática em Rede Nacional, Universidade Estadual da Paraíba Centro de Ciências e Tecnologias, 2015. Citado 3 vezes nas páginas 19, 20 e 21.

SOUZA, W. P. R. d. **Soluções Eficientes para o Cubo Mágico.** 24 f. Monografia (Especialização) — Curso de Matemática, Departamento de Ciência da Computação, Universidade de São Paulo, 2011. Citado 4 vezes nas páginas 13, 14, 19 e 20.

SWIEZY, J.; KNOPF, N. **Rubik's Cube Solving Robot.** 2017. Disponível em: <https://web.mit.edu/6.111/www/f2017/projects/jeswiezy_Project_Final_Report.pdf#_ts1722956605720>. Acessado em: 10 novembro 2023. Citado 6 vezes nas páginas 19, 26, 29, 60, 64 e 76.

TAOS. **TCS3200, TCS3210, PROGRAMMABLE COLOR LIGHT-TO-FREQUENCY CONVERTER**. 2009. Disponível em: <https://www.mouser.com/catalog/specsheets/tcs3200-e11.pdf>. Acessado em: 20 maio 2024. Citado 3 vezes nas páginas 37, 38 e 39.

TECHNOLOGY, H. **17HS4401S 1.7A Torque:43N.cm Stepper Motor**. s.d. Disponível em: <https://www.handsontec.com/dataspecs/17HS4401S.pdf>. Acessado em: 30 agosto 2024. Citado na página 40.

TOSHNIWAL, E. S.; GOLHAR, Y. Rubik's cube solver: A review. In: IEEE. **2019 9th International Conference on Emerging Trends in Engineering and Technology-Signal and Information Processing (ICETET-SIP-19)**. [S.l.], 2019. p. 1–5. Citado na página 13.

TSOY, M. **FAC system Rubik's Cube solver.** 2016. Disponível em: <https://blog.zok.pw/hacking/2015/08/18/fac-rubik-solver/#1-color-ranging>. Acessado em: 10 novembro 2023. Citado 6 vezes nas páginas 5, 19, 27, 29, 64 e 76.

TSOY, M. **A pure Python and pure C ports of Kociemba's algorithm for solving Rubik's cube.** 2022. Disponível em: <https://github.com/muodov/kociemba>. Acessado em: 10 novembro 2023. Citado 5 vezes nas páginas 25, 56, 57, 60 e 68.

USINAINFO. **Shield Stepper Motor Arduino SM3D para Driver A4988 e DRV8825 para Impressora 3D**. s.d. Disponível em: <https://www.usinainfo.com.br/driver-para-motor/shield-stepper-motor-arduino-sm3d-para-driver-a4988-e-drv8825-para-impressora-3d-5451.html>. Acessado em: 20 agosto 2024. Citado 3 vezes nas páginas 40, 43 e 44.

VAZ, A. L. **KNN — K-vizinho mais próximo, o que é?** 2021. Disponível em: <https://medium.com/data-hackers/knn-k-nearest-neighbor-o-que-Ãl'-aeebe0f833eb>. Acessado em: 08 setembro 2024. Citado na página 28.

WANG, T.-M.; TAO, Y.; LIU, H. Current researches and future development trend of intelligent robot: A review. **International Journal of Automation and Computing**, Springer, v. 15, n. 5, p. 525–546, 2018. Citado na página 13.

WANGYUYYT. **rubik_cube_solver**. 2022. Disponível em: <https://github.com/wangyuyyt/rubik_cube_solver/tree/main>. Acessado em: 10 novembro 2023. Citado 13 vezes nas páginas 5, 6, 8, 19, 25, 26, 29, 40, 42, 46, 48, 50 e 57.

WARDANA, H. K.; INDAHWATI, E.; FITRIYAH, L. A. Measurement of non-invasive blood glucose level based sensor color tcs3200 and arduino. **IOP Conference Series: Materials Science and Engineering**, IOP Publishing, v. 336, n. 1, p. 012019, apr 2018. Disponível em: <https://dx.doi.org/10.1088/1757-899X/336/1/012019>. Citado na página 38.

WEN, C.; GAIESKI, F. K. **Robô Para Solução do Cubo de Rubik**. 74 f. Monografia (Trabalho de Conclusão de Curso) — Departamento de Engenharia Mecatrônica e Sistemas Mecânicos - PMR, Escola Politécnica da Universidade de São Paulo - POLI USP, São Paulo, 2020. Citado 4 vezes nas páginas 13, 14, 19 e 23.

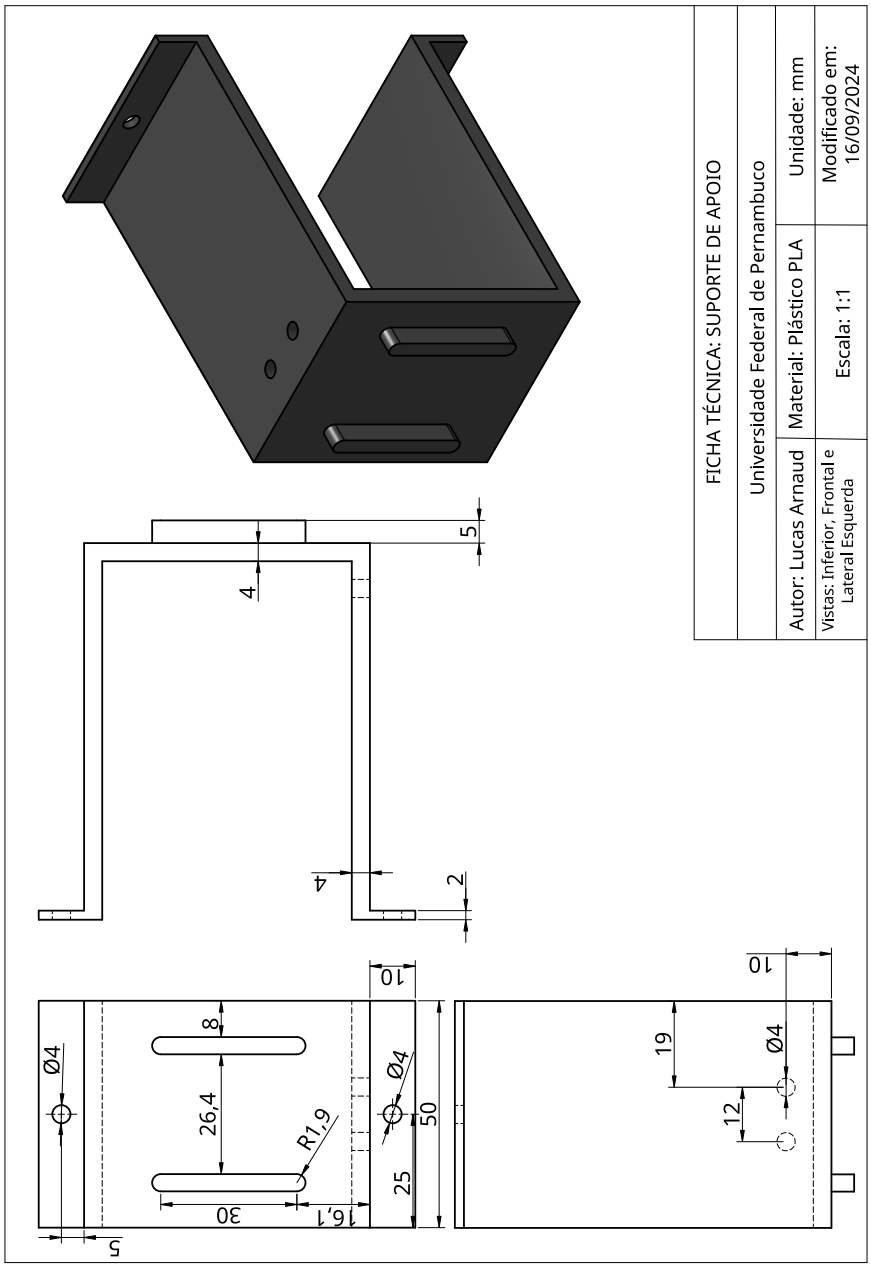
YOSHIDA, K.; WILCOX, B. Space robots. **Springer handbook of robotics**, Springer Berlin, p. 1031–1063, 2008. Citado na página 13.

APÊNDICE A – FICHAS TÉCNICAS DAS PEÇAS DO ROBÔ

DIMENSÕES DAS PEÇAS DA ESTRUTURA DO ROBÔ

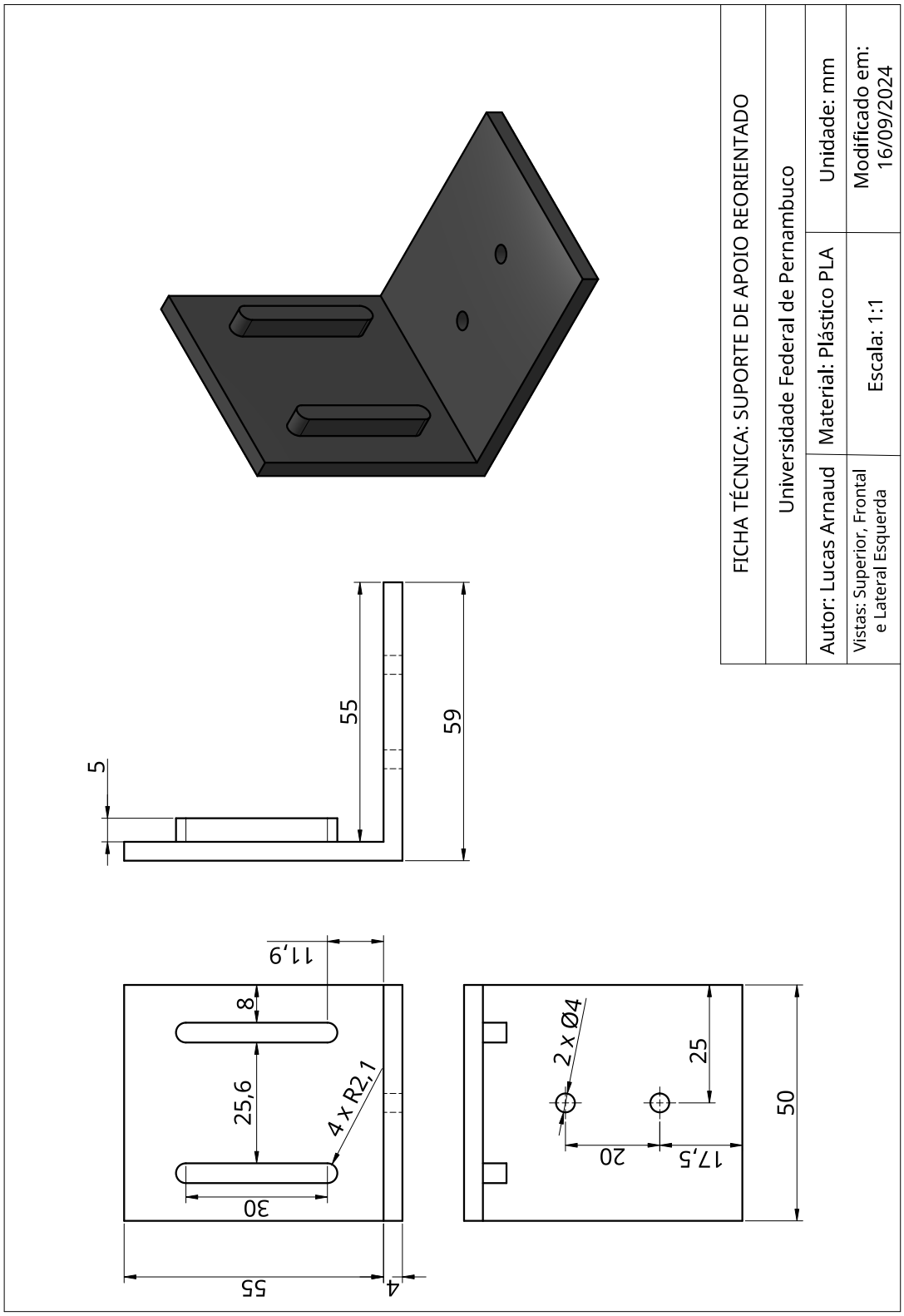
Neste apêndice são apresentadas as fichas técnicas completa com as medidas de cada peça utilizada na estrutura do robô destinada à fabricação, impressão 3D e montagem, conforme descrito no Capítulo 3.

Figura 71 – Ficha técnica do Suporte de Apoio.



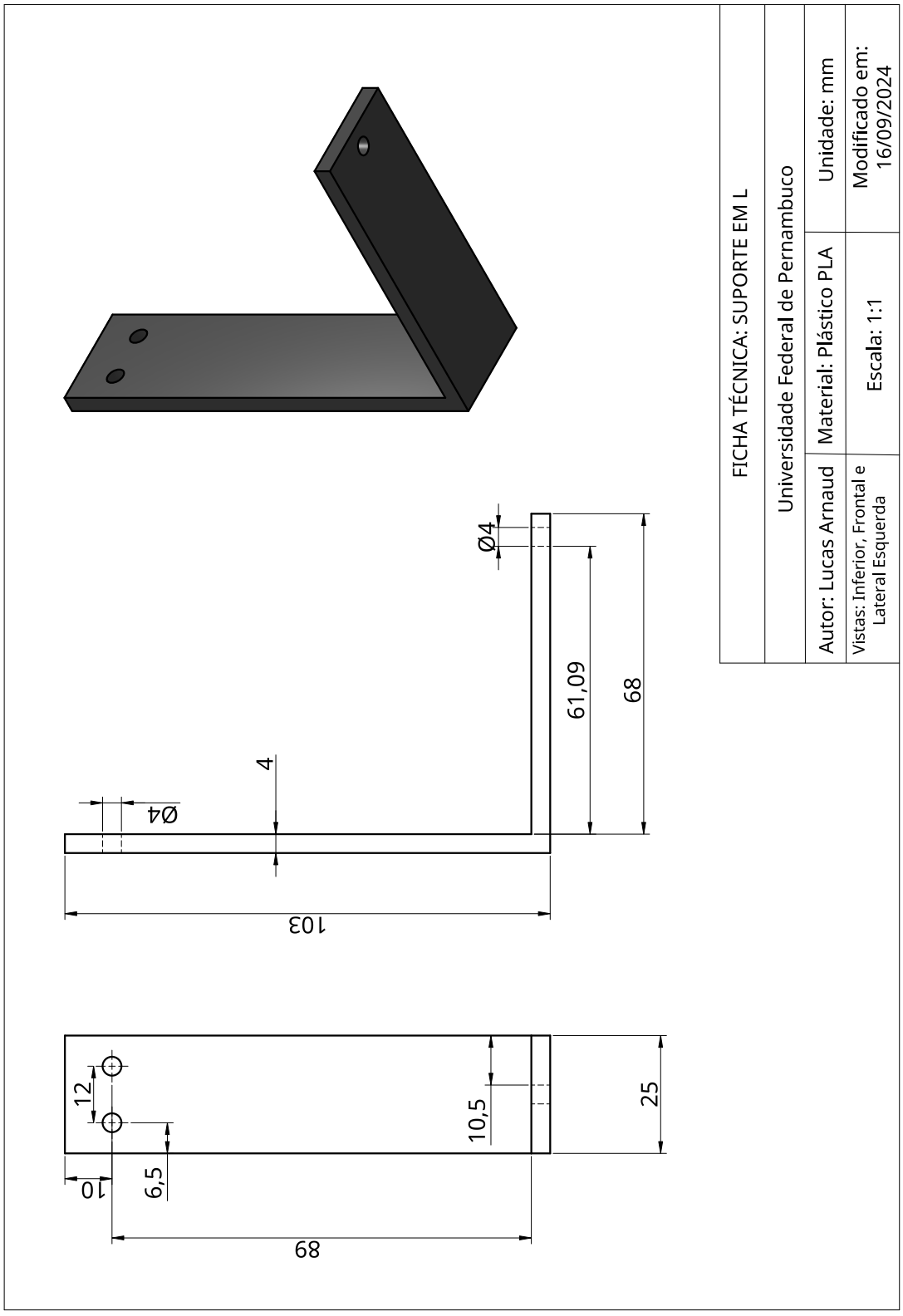
Fonte: Autoria própria.

Figura 72 – Ficha técnica do Suporte de Apoio Reorientado.



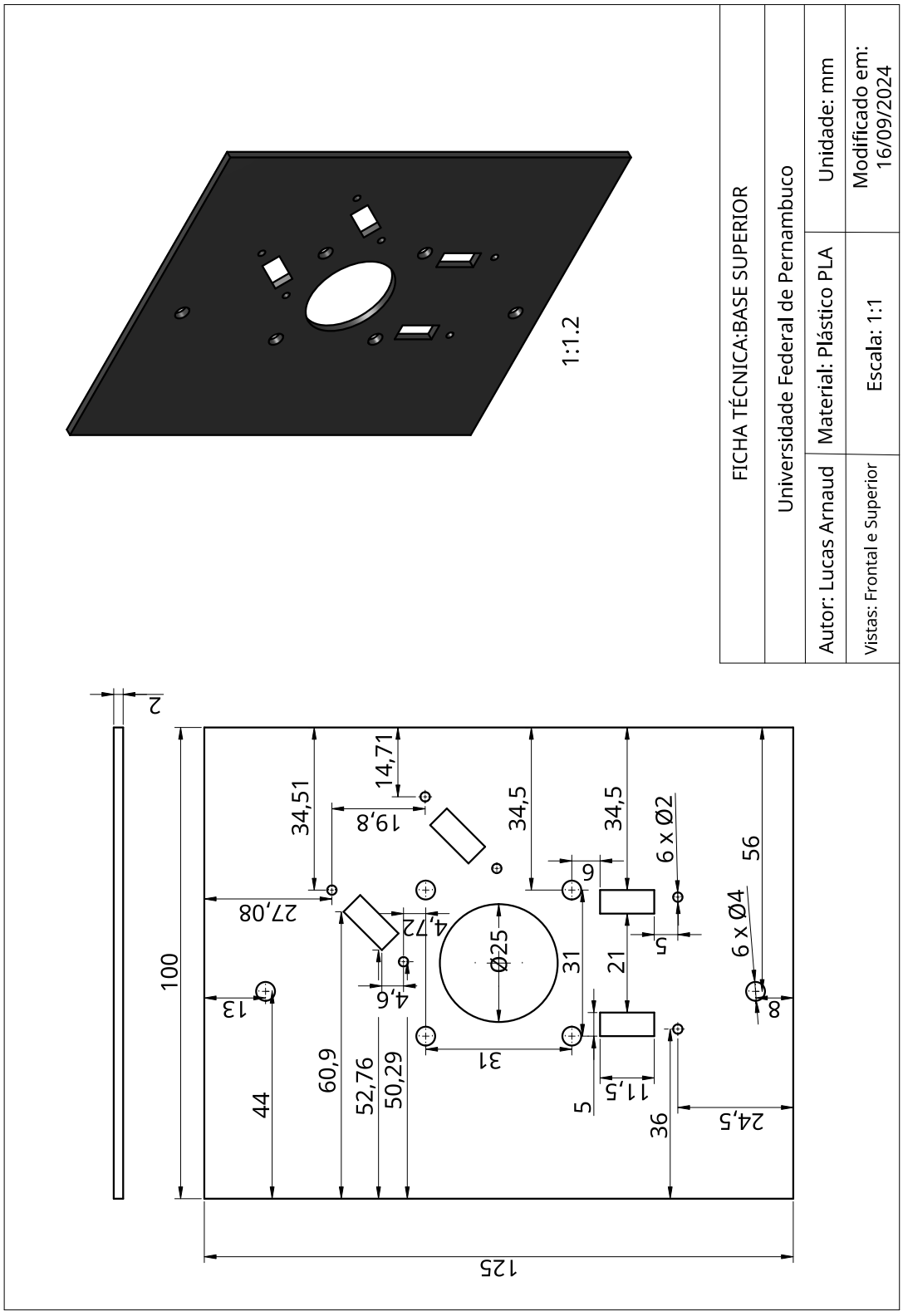
Fonte: Autoria própria.

Figura 73 – Ficha técnica do Suporte em L.



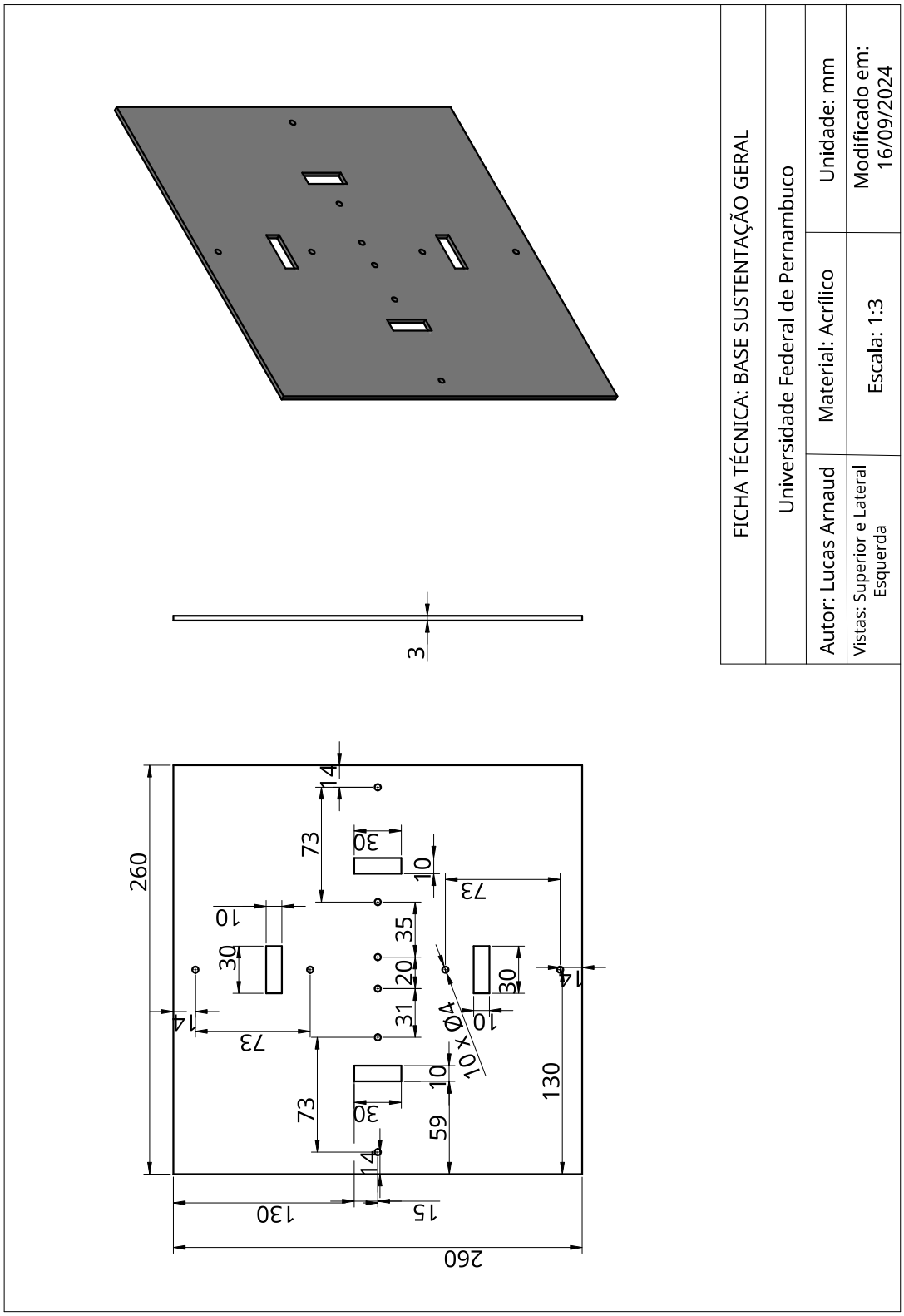
Fonte: Autoria própria.

Figura 74 – Ficha técnica da Base Superior.



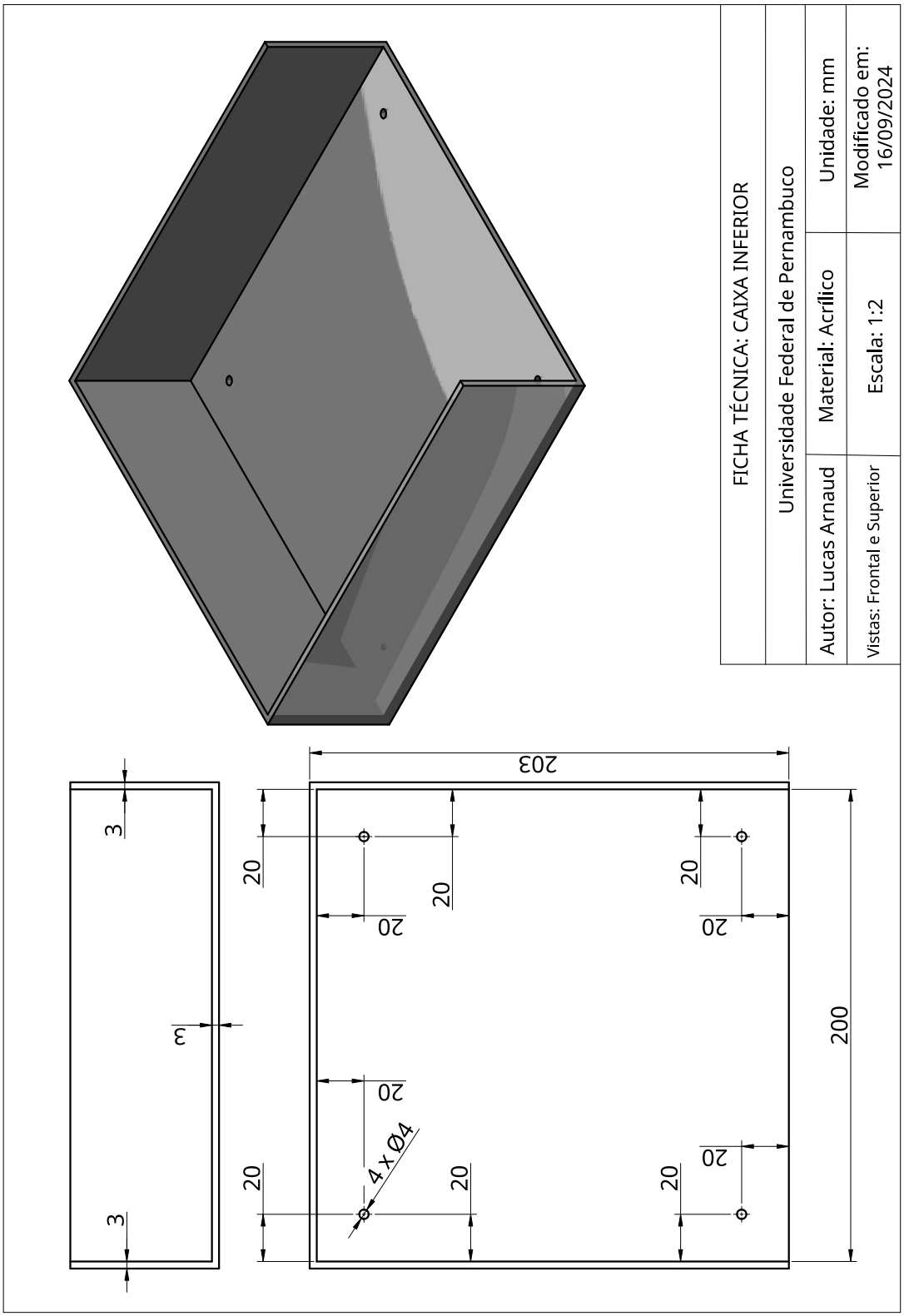
Fonte: Autoria própria.

Figura 75 – Ficha técnica da Base de sustentação geral.



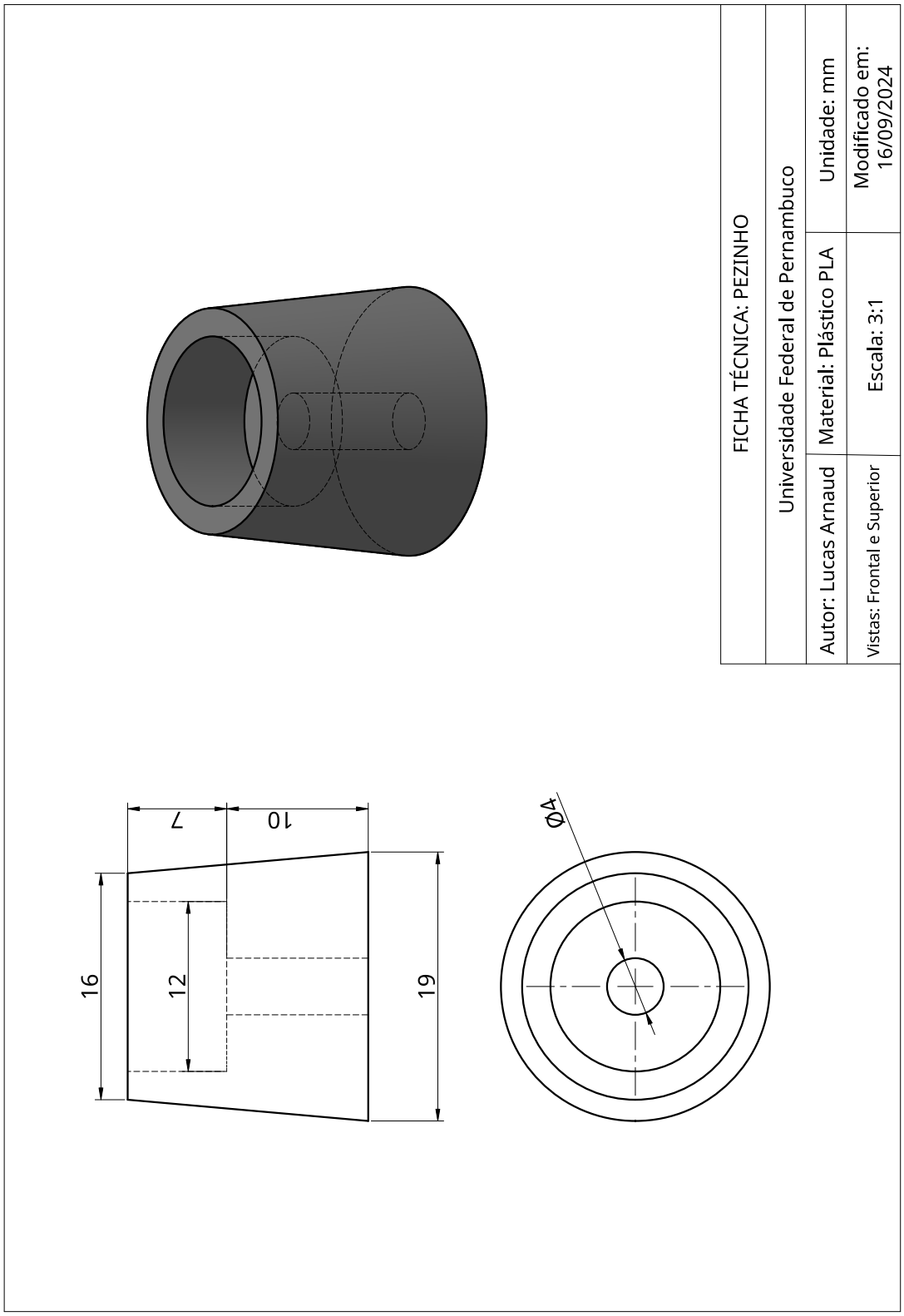
Fonte: Autoria própria.

Figura 76 – Ficha técnica da Caixa Inferior.



Fonte: Autoria própria.

Figura 77 – Ficha técnica do Pezinho.



Fonte: Autoria própria.

APÊNDICE B – PADRÕES DO CUBO DE RUBIK UTILIZADOS NOS TESTES

Neste apêndice é apresentado o detalhamento dos padrões do cubo de Rubik utilizados nos testes do robô do Capítulo 7. Incluindo, para cada um, a sequência de movimentos geradores do padrão (embaralhamento), uma sequência de movimentos para resolução, a *string* de 54 caracteres (estado do cubo) e a *string* de 48 caracteres (sequência de leitura do cubo).

1. **PADRÃO 1:**

   Embaralhamento: F2 U2 L2 D R2 D' B2 F2 R2 D F2 U2 R F2 D' L' R' U' F2 R.

   Exemplo de solução: R' F2 U R L D F2 R' U2 F2 D' R2 F2 B2 D R2 D' L2 U2 F2.

   Estado do cubo (54): RDLUUDFBBRBURRLUFBULUBFFBBLLUFFDUFDRDRR-RLRLLDBLFUBDDFD.

   Sequência de leitura (48): FDBLBUBUFBUBUFRLRDRLDRFBLFLBRDDRFUBUL-DRFUDLLRFUD.

2. **PADRÃO 2:**

   Embaralhamento: F2 U2 R2 F2 D' R2 U' F2 R2 F2 U2 B2 U' R' F2 D L B2 U' F2.

   Exemplo de solução: F2 U B2 L' D' F2 R U B2 U2 F2 R2 F2 U R2 D F2 R2 U2 F2.

   Estado do cubo (54): LBUFUULUDLLFLRRRRUBBFFFDRRBBBUDDDRDLDLU-FLRFBDRLBUBUFFD.

   Sequência de leitura (48): LBRBFFUBDLRUFRLFLDUBDLURBRBRRFDFBBDDU-DLDUURLLFFU.

3. **PADRÃO 3:**

   Embaralhamento: B2 D2 F2 R2 F2 L2 F L2 R' U' R2 D B2 D2 F2 D2 R' F D' U2.

   Exemplo de solução: U2 D F' R D2 F2 D2 B2 D' R2 U R L2 F' L2 F2 R2 F2 D2 B2.

   Estado do cubo (54): DRBBUDRRFLBLFRFDLDFBURFLBLFUBLUDULDRFUULLDBF-DUURRBDBFR.

   Sequência de leitura (48): RRBFRURFLBDDLLLBUDUFFRLLULBUDBRLRBFUBD-DURDFUFFBD.

4. **PADRÃO 4:**

   Embaralhamento: U' L2 R2 U' F2 D R2 D' L2 U B F D' L' U F2 D2 B2 F2 L2.

   Exemplo de solução: L2 F2 B2 D2 F2 U' L D F' B' U' L2 D R2 D' F2 U R2 L2 U.

   Estado do cubo (54): DDFRULRULBBLDRFRURURDBFLUFUFUFLDBRLDBBFFL-DUFLDRBDBRLUB.

Sequência de leitura (48): RDUUBDUULBRRLUBUBLFFBDFFFRRDDLBFBULBFL-DLDRFRDURL.

5. **PADRÃO 5:**

   Embaralhamento: R2 D' L2 D F2 U2 F R' B2 R B L F2 R2 D' F' R' B2 F2 U'.

   Exemplo de solução: U F2 B2 R F D R2 F2 L' B' R' B2 R F' U2 F2 D' L2 D R2.

   Estado do cubo (54): RDDFUUFBDRBULRLRDUBDLFFBLLUBUBRDFDRFDU-FULLBBRLRFDBRUFL.

   Sequência de leitura (48): FDLUFLBBBBRUUDRBURFBDFBLBDDLLULUFUDF-DRRRDRLRLFFU.

6. **PADRÃO 6:**

   Embaralhamento: U' B2 U' R2 U R2 B' U' L D R B2 D L B L2 F' L F U'.

   Exemplo de solução: U F' L' F L2 B' L' D' B2 R' D' L' U B R2 U' R2 U B2 U.

   Estado do cubo (54): DFBLUBLRRRRFFRBLLLBUFFFDUBDFLFBDFRLUBDUR-LULUUDRBDBDRUD.

   Sequência de leitura (48): LFUDFFRBDRLLFLRLDUUUBUFBFRUDURDRBLRF-BLDBDDBRFULB.

7. **PADRÃO 7:**

   Embaralhamento: D' U' B2 F2 U' F2 L2 U' B U2 R2 B L R' B' D2 L' D2 U L'.

   Exemplo de solução: U' D2 L D2 B R L' B' R2 U2 B' U L2 F2 U F2 B2 U D.

   Estado do cubo (54): FULRULBFLRDBBRRFDURLFBFDLBDDUDFDRRFBDUU-BLRBDRULFFBLLUU.

   Sequência de leitura (48): BULDBFFRDDFUBDRBURUDDBDBDRLURLUBFULRLFFF-FLRLBURL.

8. **PADRÃO 8:**

   Embaralhamento: D' F2 R2 D2 L2 D L2 U2 R2 F R' B D2 U' B' F' U' F L B.

   Exemplo de solução: B' L' F' U F B U D2 B' R F' R2 U2 L2 D' L2 D2 R2 F2 D.

   Estado do cubo (54): LBLFULDDLFDBFRDRRDUBFDFURBFDLBLDULUFRLURLR-BUDBRRFBFUBU.

   Sequência de leitura (48): DBRFDFDUUDRDBRFBLDUURFBBDLBBRUURRLLU-LULLFFDRFBFL.

9. **PADRÃO 9:**

   Embaralhamento: U2 F2 L2 D' R2 F2 U2 R' U2 F R2 B' R D2 B' U' B R' F2 R.

Exemplo de solução: R’ F2 R B’ U B D2 R’ B R2 F’ U2 R U2 F2 R2 D L2 F2 U2.

Estado do cubo (54): DRFUUFBDRFDFBRLLBDFRURFDDLDBFUFDURLBLRRULBRULUFBBBLUDL.

Sequência de leitura (48): BRDDRUDFDDLDFBFRRLRULBULBRRUBULUBFRUFLDFBLLFBDUF.

10. **PADRÃO 10:**

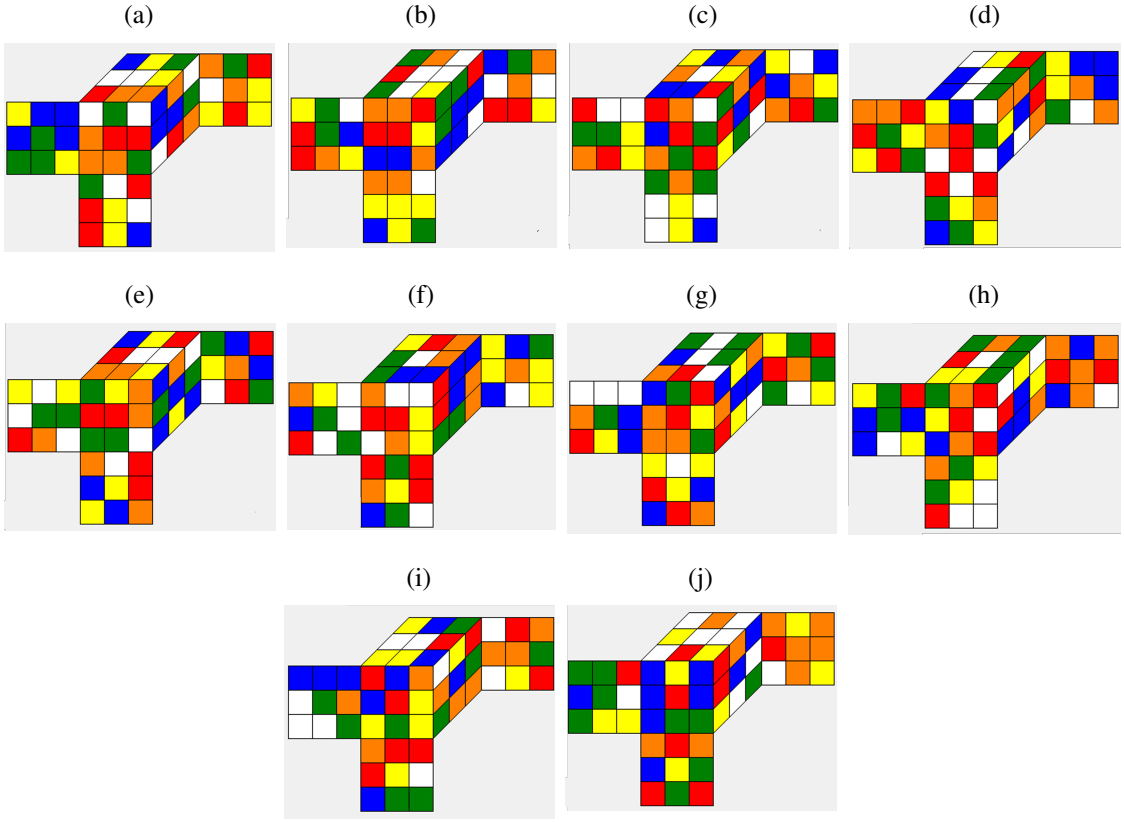
Embaralhamento: F2 U2 R2 U F2 U2 L’ D B2 D2 R2 B’ L’ B’ F’ R B D’ R.

Exemplo de solução: R’ D B’ R’ F B L B R2 D2 B2 D’ L U2 F2 U’ R2 U2 F2.

Estado do cubo (54): RBFDUUUFLFBUFRURUDRDBRFRDLLBFLRDLULBFLFRLULDDRDBFBBUBD.

Sequência de leitura (48): UBDLRBFRRBRDUUFLLDFDFBLLBUDRUUDRBFLLFLRRFBUDFBDU.

Figura 78 – Padrões do cubo de Rubik utilizados nos testes considerando a face frontal como Vermelha e a face superior como Branco: (a) Padrão 1, (a) Padrão 1, (b) Padrão 2, (c) Padrão 3, (d) Padrão 4, (e) Padrão 5, (f) Padrão 6, (g) Padrão 7, (h) Padrão 8, (i) Padrão 9 e (j) Padrão 10.



Fonte: Autoria própria.