UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
CURSO DE ENGENHARIA MECATRÔNICA

MATHEUS WILGEN GONÇALVES

LOCKERSYS: SISTEMA DE AUTENTICAÇÃO FACIAL COMO FORMA DE FACILITAR O USO DE ARMÁRIOS

Joinville
2022

MATHEUS WILGEN GONÇALVES

LOCKERSYS: SISTEMA DE AUTENTICAÇÃO FACIAL COMO FORMA DE FACILITAR O USO DE ARMÁRIOS

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica no curso de Engenharia Mecatrônica, da Universidade Federal de Santa Catarina, Centro Tecnológico de Joinville.

Orientador: Prof. Dr. Giovani Gracioli

Joinville
2022

Dedico esse trabalho aos meus pais Marcelo Gonçalves e Marli Wilgen Gonçalves
que sempre me apoiaram nos meus estudos.

AGRADECIMENTOS

Aos meus Pais, Marcelo e Marli, por todo o apoio durante o projeto e na graduação.
Ao Professor Giovani, por suas orientações ao longo do trabalho.
A Universidade Federal de Santa Catarina UFSC, que foi essencial no meu processo de formação profissional.

RESUMO

Este trabalho teve como objetivo desenvolver um software chamado LockerSys, que emprega a autenticação facial como chave de acesso para portas de um armário. Para isso foi utilizado um modelo treinado de rede neural convolucional. Algumas das etapas realizadas no desenvolvimento do trabalho foram, primeiramente, especificar os requisitos que o projeto deve alcançar e criar diagramas UML. Na implementação do sistema para a parte de interface gráfica e de visão computacional, foi utilizado bibliotecas para a Linguagem de programação Python: OpenCV, Kivy e FaceRecognition. Para validar o software foi desenvolvido um armário com vinte e oito portas, e instalado um microcomputador Raspberry Pi 3B em conjunto com uma tela touchscreen. O sistema foi testado, avaliando a precisão e acurácia com matrizes de confusão. Outros atributos analisados foram o tempo de processamento e custos do projeto. Os resultados mostram uma alta precisão e acurácia na autenticação facial, em um intervalo de tempo satisfatório para o processamento. Algumas limitações foram observadas em relação à câmera e mecanismos anti-fraude quando comparados a produtos comerciais semelhantes ao projeto desse trabalho. Enfim, por meio do projeto realizado e o estudo dos resultados foi possível confirmar que o software LockerSys executando em uma Raspberry Pi 3B funciona de maneira precisa e estável.

Palavras-chave: Visão Computacional. Python. Raspberry Pi. Autenticação Facial.

ABSTRACT

This work aimed to develop a software called LockerSys, which uses facial authentication as a key to access locker doors, for which a trained model of a convolutional neural network was used. Some steps carried out in the development of the work were, firstly, specify the requirements that the project must achieve and create the UML diagrams, in the implementation of the system for the graphical user interface and computer vision part, libraries for the Python programming language were used: OpenCV, Kivy and FaceRecognition. To validate the software, a locker with twenty-eight doors was developed, and a Raspberry Pi 3B microcomputer was installed together with a touchscreen. The system was tested evaluating the precision and accuracy with confusion matrices, other attributes analyzed were the processing time and project costs. The results show a high precision and accuracy in face authentication, and a satisfactory processing time interval. Some limitations were observed in relation to the camera and anti-fraud mechanisms when compared to commercial products similar to the project of this work. Finally, through the project carried out and the study of the results, it was possible to confirm that the LockerSys software running on a Raspberry Pi 3B works accurately and stably.

Keywords: Computer Vision. Python. Raspberry Pi. Face Authentication.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

BLE	*Bluetooth Low Energy*
CAD	*Computer Aided Design*
CV	*Computer Vision*
FN	*False Negative*
FP	*False Positive*
GLCM	*Grey Level Co-Occurrence Matrix*
HOG	*Histogram of Oriented Gradients*
IOT	*Internet of Things*
LSB	*Least Significant Bit*
MSB	*Most Significant Bit*
OTP	*One Time Password*
PCA	*Principal Component Analysis*
SMS	*Short Message Service*
SLAM	*Simultaneous Localisation And Mapping*
TP	*True Positive*
TN	*True Negative*
UML	*Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

Os seres humanos estão acostumados a detectar rostos diariamente, mas, para os computadores essa não é uma tarefa trivial, entretanto, o uso da autenticação facial vem crescendo a cada ano na área da biometria, inclusive nos celulares. O desenvolvimento de novas abordagens, algoritmos e aplicações são importantes para agregar aprimoramentos nessa área.

De forma geral, a autenticação facial é utilizada com o intuito de facilitar a autenticação biométrica em relação à impressão digital e reconhecimento de íris. Com isso, algoritmos como o Principal Component Analysis (PCA), Viola e Jones e Histogram of Oriented Gradients (HOG), se popularizaram nas aplicações de detecção e reconhecimento facial (SZELISKI, 2011). Esse método de autenticação, em que se desbloqueia o acesso a algum recurso por meio do rosto, tem aplicações potenciais, como em armários.

O uso de armários em lojas, aeroportos, academias é uma escolha adequada quando é necessário guardar algum objeto. Geralmente, o acesso a cada uma das portas desses armários é realizado com uma chave de metal, como consequência dessa característica, as chaves são gradualmente perdidas, quebradas ou copiadas indevidamente. Outra forma utilizada é o uso pessoal de cadeados, entretanto, alguns usuários acabam por esquecer o cadeado e a respectiva porta fica inutilizada por tempo indeterminado.

Existe um método onde em cada porta têm um teclado numérico, é necessário então cadastrar uma senha para tranca-la, e a mesma senha é digitada para destravar a porta. Essa última forma não é muito segura, já que alguém com más intenções pode ver a senha digitada, outro problema é o fato de esquecer a senha, que causa um transtorno para retirar o pertence. Dessa maneira, buscou-se identificar algoritmos de autenticação facial, visando responder ao seguinte problema: Qual é o melhor algoritmo para autenticação facial que apresente desempenho para ser aplicado em um protótipo de armário, em que o microcomputador é o Raspberry pi 3B?

Este trabalho tem como objetivo aplicar uma rede neural convolucional para realizar a autenticação facial e aplicar em um armário. Uma das etapas importantes é conceituar detecção facial e visão computacional que envolvem o desenvolvimento do projeto (JAIN; ROSS; NANDAKUMAR, 2011; SZELISKI, 2011). O último passo é analisar se a solução proposta nesse trabalho consegue resolver problemas relacionados aos outros métodos que utilizam chaves, códigos numéricos e cadeados.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Desenvolver um armário inteligente utilizando a autenticação facial para facilitar o uso e proteger as portas de acesso.

1.1.2 Objetivos Específicos

1. Situar a área de autenticação facial no campo da visão computacional;
2. Identificar o algoritmo adequado de detecção facial, afim de otimizar computacionalmente, atingindo uma alta taxa de precisão;
3. Apresentar o desenvolvimento do hardware e software, que constituem o armário inteligente;
4. Validar o protótipo de armário com uma aplicação prática.

2 FUNDAMENTAÇÃO TEÓRICA

A detecção facial é um ramo de pesquisa que envolve análise de imagens para detectar regiões com rostos. Esse capítulo tem o intuito de explorar e compreender a relação entre visão computacional e os algoritmos utilizados na tarefa de detecção facial.

2.1 VISÃO COMPUTACIONAL

Para os seres humanos o processo de enxergar o mundo e identificar profundidade, formas, pessoas e cores, como também atribuir sentido ao que foi visualizado são tarefas fáceis. Em relação aos computadores o desafio é maior, a visão computacional é um processo realizado com o uso de técnicas matemáticas que permite extrair de imagens informações, como forma, iluminação e distribuição de cores (SZELISKI, 2011).Ver é uma tarefa essencialmente baseada em processamento, tanto no sentido biológico quanto no sentido computacional, ou seja, não é apenas capturar os reflexos de luz, mas sim extrair dados no processo (ZHANG; LIN, 2013).

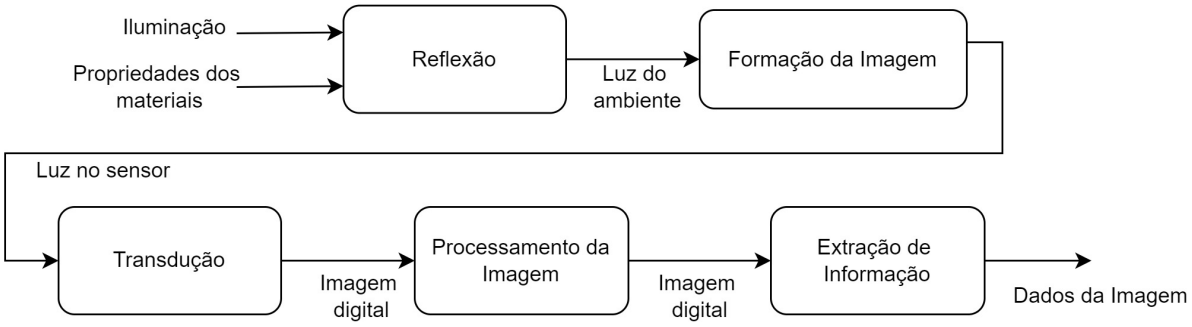
Segundo Corke (2011) visão computacional é composta por um conjunto de processos que envolve a luz capturada por uma lente, transformada em uma imagem digital e processada por vários algoritmos visando extrair informações desejadas. Como bem nos assegura Leavers (1992), visão computacional é um meio para que a automação alcance todo o seu potencial. Muitos processos industriais requerem inspeção visual, com o intuito de identificar falhas de fabricação. A aplicação de visão computacional a esses casos pode aumentar a qualidade e produtividade dos produtos.

Pode-se perceber na Figura 1 que o processo de visão computacional é composto por diferentes áreas, primeiramente os efeitos óticos da lente da câmera focam a iluminação para que o sensor capte uma imagem nítida, logo após o sinal analógico da imagem é transformado em digital e por fim os dados binários resultantes podem ser processados pelo computador. Para Kakani et al. (2020) visão computacional facilita o reconhecimento de objetos, navegação autônoma de veículos, reconhecimento facial, reconhecimento de impressão digital e navegação robótica.

2.2 REDES NEURAIS ARTIFICIAIS

Inspirado na neurociência, as redes neurais artificiais são compostas por neurônios artificiais. Um único neurônio é basicamente um modelo matemático em

Figura 1 – Processo da visão computacional.

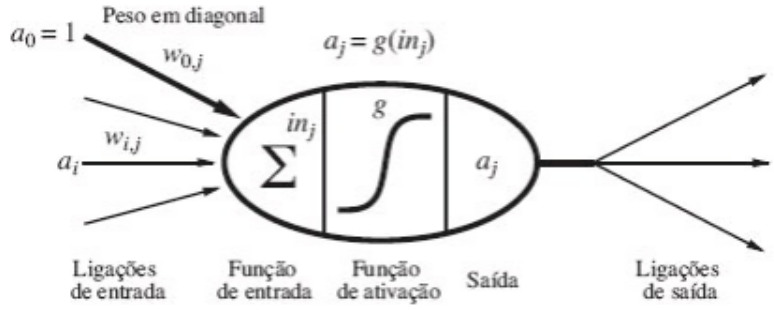


Fonte: Adaptado de Corke (2011, p. 222).

que o objetivo é indicar ou não uma ativação. Na Figura 2 é representado um neurônio artificial, pode-se ver que a função de ativação a_j tem como argumento a combinação linear de suas entradas, como mostra a Equação 1, que considera os pesos de entrada multiplicados pelas ativações dos neurônios anteriores ou entradas de uma rede neural (RUSSEL; NORVIG, 2013).

$$a_j = g\left(\sum_{i=0}^{n}(w_{i,j} \cdot a_i)\right) \tag{1}$$

Figura 2 – Neurônio artificial, a saída a_j representa a ativação.

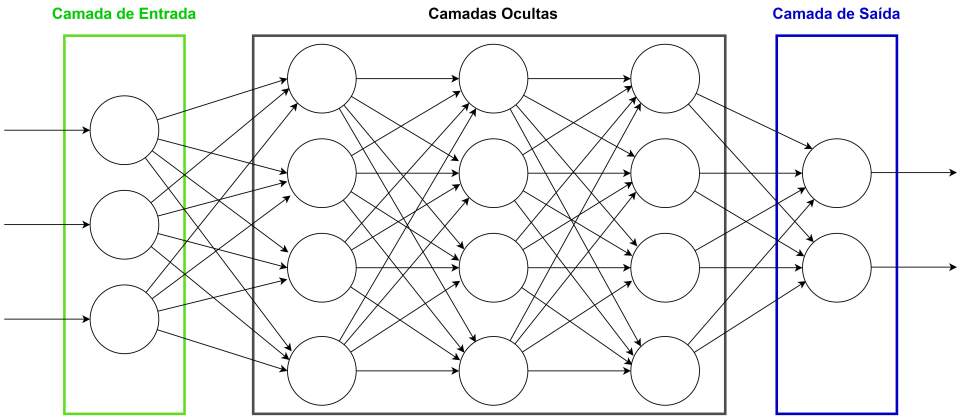


Fonte: (RUSSEL; NORVIG, 2013).

Uma rede neural é composta por vários neurônios organizados em camadas. Entre as redes mais utilizadas para reconhecimento de imagens estão as redes convolucionais com a alimentação para frente (TAIGMAN et al., 2014; HE et al., 2015). Na Figura 3 é mostrado um exemplo dessa configuração de rede, pode-se ver que há três neurônios na camada de entrada, três camadas ocultas contendo quatro neurônios em cada camada e na saída, dois neurônios.

Partindo do exemplo apresentado na Figura 3 pode-se ampliar o conceito a redes maiores como a apresentada na Figura 4, nesse modelo criado pelo Facebook foi utilizado uma camada de entrada 3D alinhada de três canais (RGB) com tamanho 152x152x3, após a extração do rosto filtrado em RGB é iniciado o processamento nas

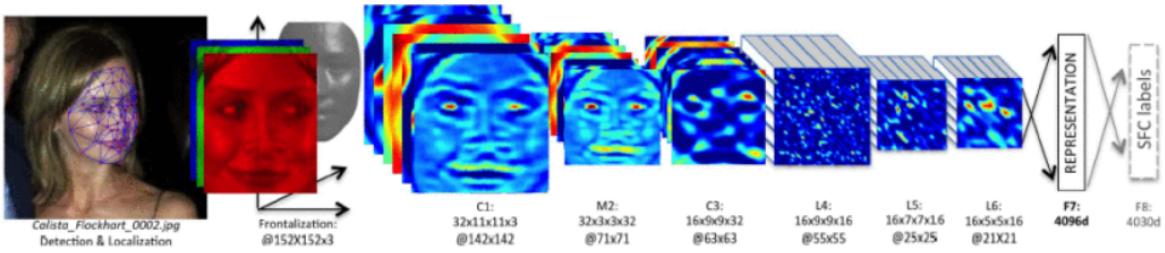
Figura 3 – Rede neural convolucional, três neurônios de entrada e dois na saída.



Fonte: Autor.

camadas ocultas, a primeira é a C1 com 32 filtros do tamanho 11x11x3 denotados 32x11x11x3. O resultado com os 32 mapas de características da camada C1 é alimentado na camada M2 que realiza a máxima junção dos vizinhos espaciais, a próxima camada C3 é empregada para filtrar texturas e bordas, para isso foi implementado 16 filtros com tamanhos de 9x9x16 (TAIGMAN et al., 2014).

Figura 4 – Arquitetura do DeepFace.



Fonte: (TAIGMAN et al., 2014).

As próximas camadas L4, L5 e L6 da Figura 4 são filtros para diferentes características do rosto. Por último as camadas de saída F7 e F8 conseguem correlacionar entre características das imagens dos rostos, como exemplo, a forma e a posição da boca e a forma e posição dos olhos (TAIGMAN et al., 2014).

As redes neurais convolucionais demonstram excelente desempenho, quando utilizadas para classificação de dígitos escritos a mão e detecção facial (ZEILER; FERGUS, 2013). No treinamento de redes para detecção e reconhecimento de objetos é necessário um banco de dados com imagens rotuladas e diversificadas, dessa maneira obtém-se precisão nos resultados de classificação ou extração de características, um dos bancos mais conhecidos é o ImageNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

2.3 ALGORITMO DE AUTENTICAÇÃO FACIAL - DISTÂNCIA EUCLIDIANA

Após extrair os vetores com as características desejadas das imagens, é possível analisar a semelhança entre esses dados para um dado valor de limiar. Uma maneira de aplicar essa análise e verificar a similaridade entre dois vetores n-dimensionais obtidos das imagens, é utilizando o cálculo da distância euclidiana, como mostra a Equação 3 (SZELISKI, 2011).

$$D = \sqrt{(v[1] - p[1])^2 + (v[2] - p[2])^2 + ... + (v[i] - p[i])^2)} \tag{2}$$

$$D = \sqrt{\sum_{i=1}^{n} (v[i] - p[i])^2} \tag{3}$$

O uso da distância euclidiana pode ser melhor compreendido no seguinte exemplo: dado dois vetores, v[4] = [1, 4, 2, 6] e p[4] = [1.2, 4, 2.5, 6.6] é utilizada a Equação 3, que resulta no seguinte valor: 0.65. Considerando um valor de 0.7 para limiar de similaridade, conclui-se que os dois vetores são similares, pois $0.65 < 0.7$. Na Figura 5 é apresentado o algoritmo de distância euclidiana em pseudocódigo.

Figura 5 – Pseudocódigo distância euclidiana.

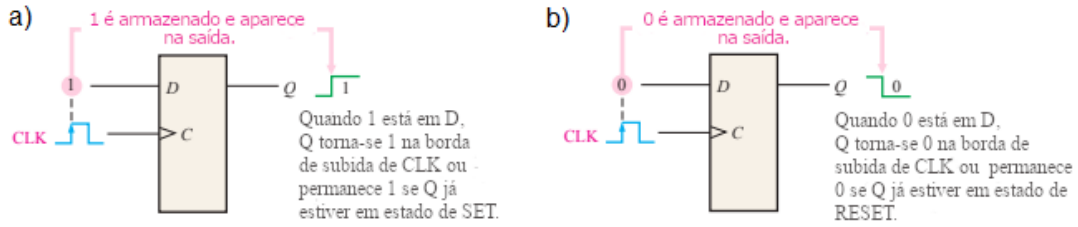
```
1 Distancia_Euclidiana(Vetor1[n],Vetor2[n])
2     i <- 0
3     soma <- 0
4     Enquanto i < n
5         soma = soma + (Vetor1[i] - Vetor2[i])^2
6         i <- i + 1
7     Fim Enquanto
8     resultado <- raiz_quadrada(soma)
9 retorna resultado
```

Fonte: Autor.

2.4 REGISTRADORES DE DESLOCAMENTO

Um registrador tem a finalidade de armazenar e movimentar dados binários. São constituídos por arranjos de flip-flops, na qual são utilizados em série no objetivo de manipular os dados binários (FLOYD, 2015). A Figura 6 mostra como um flip-flop do tipo D pode ser utilizado para armazenar um dado binário. Na Figura 6a pode-se ver que a borda de subida do sinal CLK, resulta no carregamento do bit 1 na saída Q, na Figura 6b o processo é o mesmo, mas agora é carregado o bit 0. A associação de flip-flops em diferentes configurações resulta em registradores de deslocamento específicos.

Figura 6 – Flip Flop do tipo D armazenando um dado binário.



Fonte: Adaptado de Floyd (2015)

.

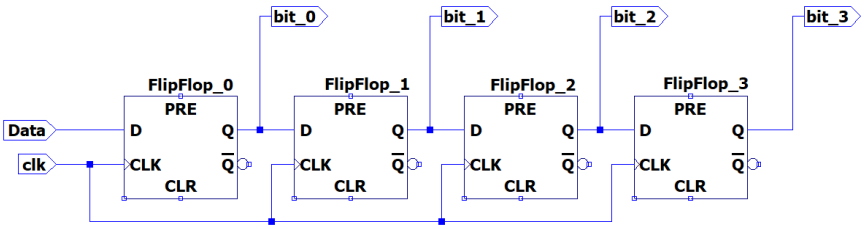
As configurações são classificadas pelo modo em que é realizado o fluxo de dados, podem ter entradas seriais ou paralelas e saídas seriais ou paralelas. Na Figura 7a é mostrado um registrador com quatro flip-flops em cascata, nesse caso a entrada está em série e a saída é realizada de forma paralela. Na Figura 7b é demonstrado uma simulação do registrador, o dado inserido via serial é 1010, é importante notar que o dado foi enviado partindo do LSB (Least Significant Bit) ao MSB (Most Significant Bit), após quatro sinais de clock o dado está configurado na saída.

2.5 TRABALHOS RELACIONADOS

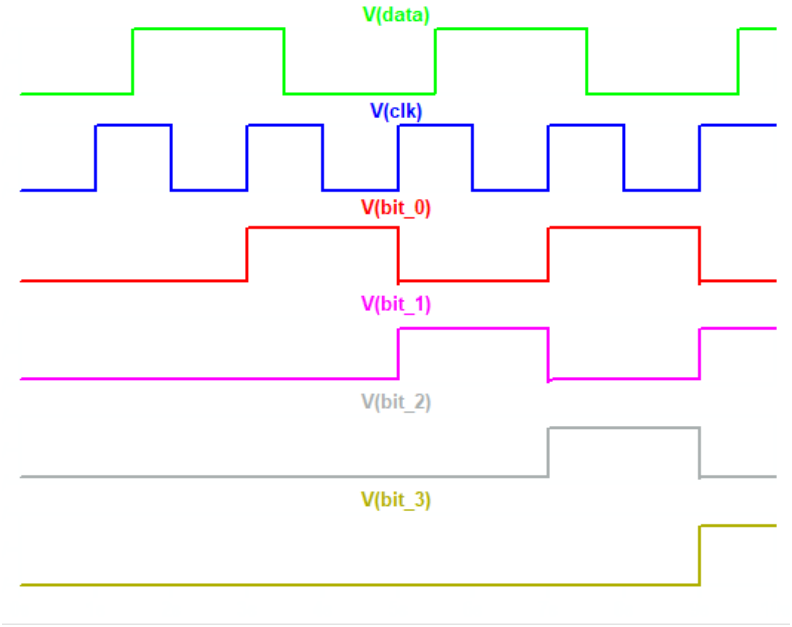
Em Kassem et al. (2016) é proposto um sistema de fechaduras inteligentes com base no uso dos celulares smartphones. Na aplicação as senhas são armazenadas no celular quando utilizado o aplicativo desenvolvido, essas senhas ficam ligadas a um usuário único. Por motivos de segurança o sistema opera apenas em redes locais e utiliza criptografia para as chaves de abertura. Na Figura 8 pode-se ver como ocorre a interação dos dispositivos no sistema, os comandos são enviados para a central de comando, por meio do Wifi até o roteador e o roteador comunica por ethernet com a central de controle.

Em (MLADENOVA; VALOVA; VALOV, 2021) é explorado a biometria como forma de controlar as fechaduras inteligentes. O algoritmo utilizado para extrair as características dos rostos é o PCA (Principal Component Analysis), o software foi embarcado em uma Raspberry Pi 3B+, por razões computacionais associadas ao reconhecimento facial. Na Figura 9a pode-se ver o diagrama de blocos em que o sistema foi baseado, existem dois bancos de dados, um na nuvem e outro local na raspberry. Na Figura 9b é apresentado o sistema de acionamento, realizado por meio de uma porta GPIO.

Figura 7 – Registrador com 4 flip-flops, saída paralela.



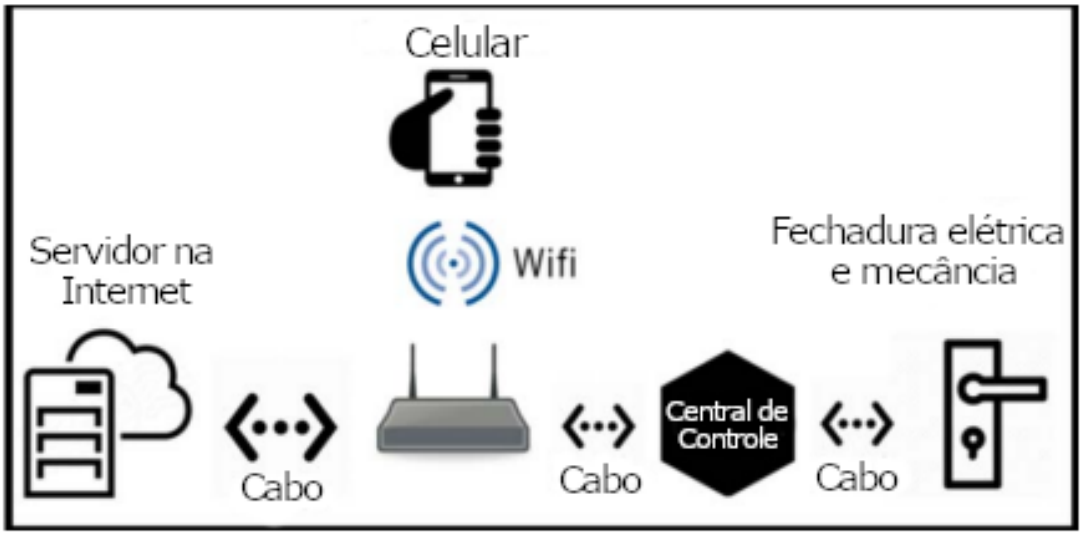
(a) Quatro flip flops do tipo D associados em cascata.



(b) Sinais de teste do registrador com quatro flip flops.

Fonte: Autor.

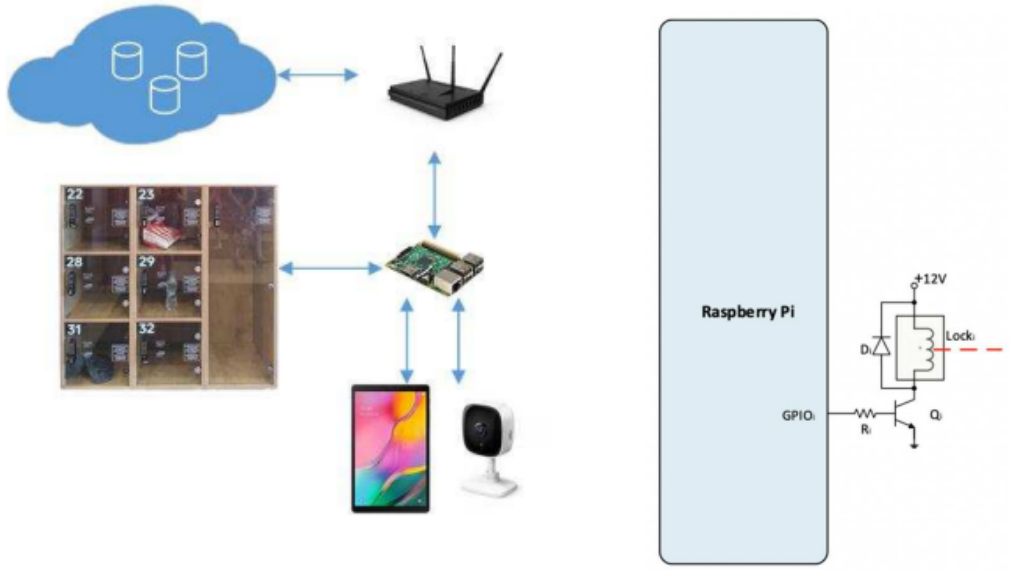
Figura 8 – Arquitetura do sistema SLS.



Fonte: Adaptado de Kassem et al. (2016)

.

Figura 9 – Diagrama de blocos e acionamento para utilização em armários, solução proposta por (MLADENOVA; VALOVA; VALOV, 2021)



(a) Diagrama de blocos, mostrando os componentes presentes no sistema.

(b) Acionamento das portas.

Fonte: (MLADENOVA; VALOVA; VALOV, 2021).

No trabalho de Anusha, Sai e Srikar (2017) foi utilizado o algoritmo de Viola-Jones para extrair da imagem a localização do rosto. Os detalhes do rosto são extraídos usando Filtro de Gabor, GLCM (Grey Level Co-Occurrence Matrix) e HOG. Na arquitetura do software, é proposto a criação de um perfil com usuário e senha e informações de número de telefone e e-mail.

Sempre que um usuário deseja usar uma porta, é necessário inserir a senha cadastrada, dada a confirmação da senha a câmera no armário tira uma foto para autenticar na base de dados. Com o sucesso na autenticação é enviado uma senha de uso único (OTP) por SMS ao celular da pessoa. A atividade de retirar o objeto previamente armazenado, é realizada utilizando apenas a senha OTP e uma foto capturada pelo armário.

Nesse trabalho será feito diferente do primeiro trabalho ao não usar um smartphone na utilização do sistema e realizar a computação localmente. Com relação ao segundo, a diferença está no método de autenticação facial, nesse é utilizado uma rede neural convolucional e não é utilizado banco de dados externo (na nuvem). Em relação ao último trabalho, a diferença está no método de autenticação facial e não é utilizado uma senha OTP.

3 DESENVOLVIMENTO DO PROJETO

A linha central desse capítulo é descrever sobre como foi utilizado a especificação de requisitos para guiar o desenvolvimento do trabalho. Nesse sentido são apresentados os materiais, métodos e ferramentas utilizadas no projeto das partes mecânicas, eletrônicas, e do software LockerSys.

3.1 ESPECIFICAÇÃO DE REQUISITOS DO SISTEMA

Essa seção do trabalho propõe-se a descrever o comportamento do sistema que será desenvolvido. Será dividido em quatro subseções, iniciando-se por uma descrição sobre o que é esperado do sistema, após estabelecer o comportamento do sistema é necessário caracterizar os requisitos esperados. Outra etapa importante é a construção de diagramas de blocos do hardware e diagramas do projeto do software usando a linguagem de modelagem UML (diagrama de casos de uso, atividades, classes, máquina de estados e sequência).

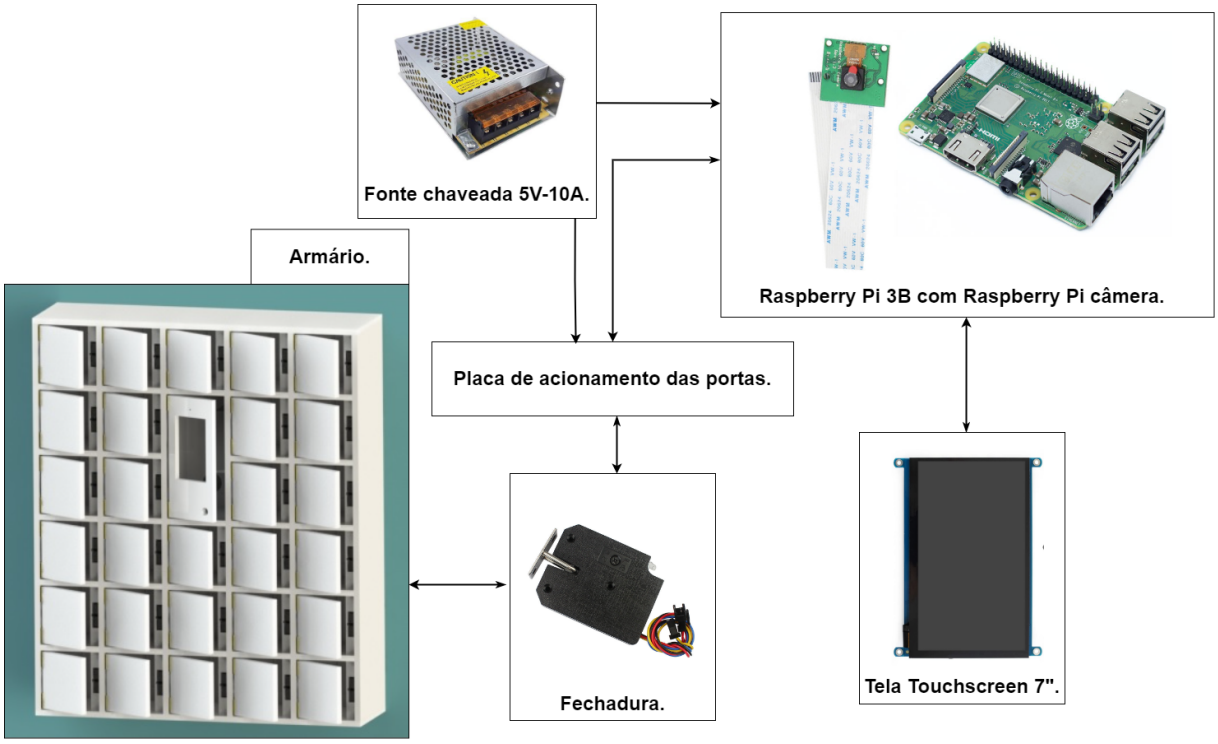
3.1.1 Descrição Geral

O software LockerSys é responsável pela administração do sistema de armário inteligente, e para isso deve realizar as seguintes tarefas: permitir o registro de uma imagem contendo o rosto de um usuário, permitir a escolha de uma porta desejada, por meio de uma interface gráfica e realizar a autenticação facial garantindo a segurança do objeto armazenado. O desenvolvimento do software para o armário, deve considerar a administração das portas, isto é, assegurar que seja utilizado apenas uma porta por vez de maneira segura.

Na Figura 10 pode-se ver a visão geral do sistema, em virtude da interface gráfica foi utilizado uma tela touchscreen de sete polegadas que captura os eventos de toque para captura de fotos e escolha das portas. A Raspeberry pi 3B foi selecionada como recurso computacional, principalmente por conta do software LockerSys que necessita utilizar autenticação facial. Para alimentar a Raspberry Pi 3B e a placa de acionamento das portas é utilizado uma fonte de 5 Volts e 10 Amperes.

O sistema será utilizado por todas as pessoas que desejarem armazenar um objeto pessoal. O comportamento esperado é facilidade no cadastro da face e escolha de uma porta disponível no armário. Para a retirada do objeto, é necessário que o armário abra a porta anteriormente escolhida de maneira automática.

Figura 10 – Visão geral do sistema proposto.



Fonte: Autor.

3.1.2 Levantamento de Requisitos

Após estabelecer o conceito geral do projeto, o próximo passo é especificar os requisitos que devem ser satisfeitos, no intuito de ao final do desenvolvimento do trabalho, o resultado corresponda ao produto desejado. Os tópicos abaixo descrevem sobre os requisitos funcionais e não funcionais, que direcionam o projeto.

3.1.2.1 Requisitos Funcionais

Os requisitos funcionais descrevem as funções obrigatórias que o sistema deve realizar para ser funcional, esses requisitos foram divididos em dois tópicos, software e hardware:

Requisitos para o software:

RF01 Permitir escolher uma porta disponível para guardar o objeto.
RF02 Proteger a abertura da porta por meio da autenticação facial.
RF03 Permitir captar imagens.
RF04 Comunicar-se com o hardware via serial.

Requisitos para o hardware:

RF05 Receber comandos de acionamento via serial para abertura das portas.
RF06 Responder com os estados das portas, ou seja, se estão abertas ou fechadas.
RF07 Conter espaços para armazenamento de objetos.

RF08 Ter uma câmera para captar as imagens a serem processadas pelo software LockerSys.

3.1.2.2 Regras de Negócio:

As regras de negócio descrevem como e de qual maneira o sistema deverá realizar as suas funções:

RN01 Serão 28 portas por armário.
RN02 Só poderá ser escolhida uma porta por vez.
RN03 Um usuário por vez pode operar no armário.
RN04 Deve haver um meio de informar que uma porta não está disponível para o uso.
RN05 Uma pessoa não pode conseguir abrir uma porta diferente da escolhida no processo de guardar o objeto.
RN06 Cada porta deve ter uma fechadura elétrica que abra quando alimentada com 5V.
RN07 O hardware utilizado será uma Raspberry pi 3 modelo B.
RN08 Utilizar o sistema operacional Raspbian Buster.
RN09 Será utilizado a biblioteca Kivy para implementar a interface gráfica.
RN10 Será utilizado a biblioteca PySerial comunicação serial entre o LockerSys e o hardware.

3.1.2.3 Requisitos Não Funcionais

Os requisitos não funcionais estão relacionados ao uso do software LockerSys. Esses requisitos foram divididos em: Usabilidade, Confiabilidade, Desempenho, Manutenibilidade e Segurança.

Externos:

RNF01 Em cada porta deve haver um sensor que indique o estado da mesma, ou seja, aberta ou fechada.

Desempenho:

RNF02 O tempo de resposta para autenticar e abrir a porta deve ser de até 6 segundos.

Manutenibilidade:

RNF03 Linguagem de programação em Python para a Raspberry pi.
RNF04 O armário é alimentado por uma fonte de 5V.

3.1.3 Diagramas

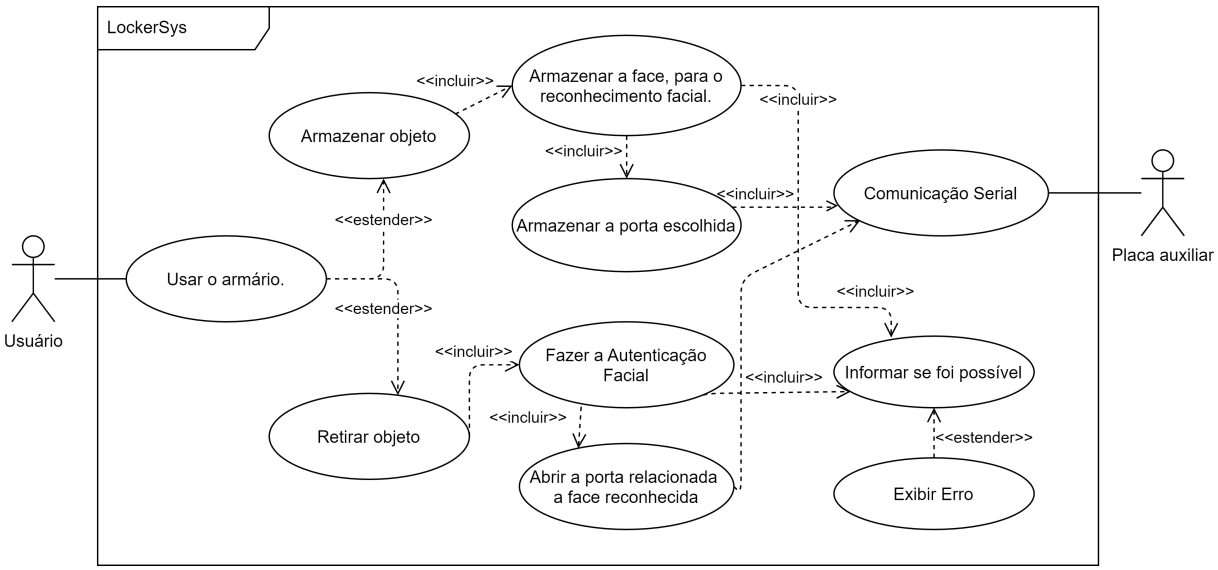
Com os requisitos estabelecidos, a próxima etapa é elaborar os diagramas que representem de maneira gráfica o sistema. Os diagramas UML (Unified Modeling Language) foram utilizados, pois permitem modelar diferentes partes de um software.

Cada diagrama tem um objetivo na modelagem de um sistema orientado a objetos, para apresentar os utilizados no trabalho foi reservado subseções com descrições a respeito do diagrama específico e das funcionalidades do software, sendo elas: Casos de Uso, Atividades, Classes, Máquina de estados e Sequência.

3.1.3.1 Casos de Uso

O diagrama de casos de uso apresenta o que o sistema faz no ponto de vista do usuário, permite visualizar como os atores interagem com o sistema, ou seja, é possível entender quem realiza a ação no sistema e como o software reage a mesma. Na Figura 11 pode-se ver a interação do ator usuário com as funcionalidades do sistema, o principal caso de uso é "Usar o armário", a partir desse caso o LockerSys identifica se o usuário está armazenando ou retirando o objeto.

Figura 11 – Diagrama de casos de uso do armário inteligente.



Fonte: Autor.

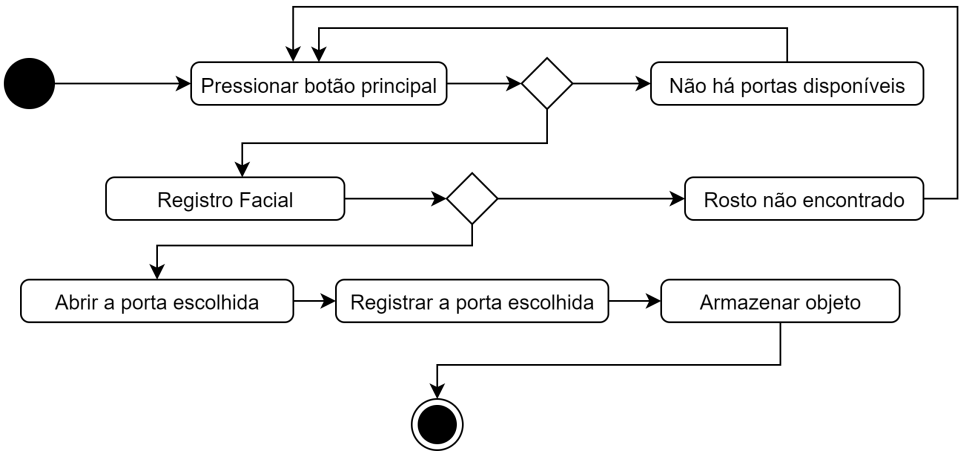
Se o usuário estiver armazenando um objeto, será registrada uma foto do seu rosto e as portas disponíveis serão apresentadas para escolha. Após a escolha da porta, um sinal de abertura é enviado para a placa auxiliar. O fluxo para retirar o objeto armazenado envolve realizar a autenticação facial e enviar o sinal de abertura para placa auxiliar. Dessa forma, a porta relacionada ao usuário é aberta.

3.1.3.2 Atividades

Os diagramas de atividades são utilizados para demonstrar e ilustrar a lógica de um algoritmo, dessa maneira facilita-se a tradução de um processo em Linguagem de programação. Na Figura 12 pode-se ver o fluxo das atividades a serem desenvolvidas no processo de armazenar um objeto no armário, primeiramente

utiliza-se um botão principal sinalizando a intenção de utilizar o armário. Se houver portas disponíveis é realizado a aquisição do rosto do usuário, caso o rosto não seja encontrado é retornado a primeira atividade. As últimas três atividades referem-se a abrir a porta escolhida pelo usuário, para permitir o seu uso.

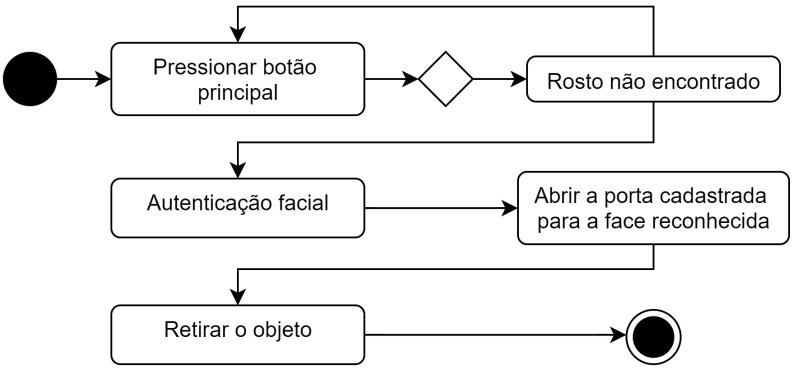
Figura 12 – Diagrama de atividades para o procedimento de guardar um objeto.



Fonte: Autor.

A Figura 13 representa os passos para realizar a retirada de um objeto do armário, nota-se que é utilizado o botão principal, e se o rosto for encontrado é realizado a autenticação facial, resultando na abertura da porta associada ao rosto com a comparação bem-sucedida.

Figura 13 – Diagrama de atividades para o procedimento de retirar um objeto.



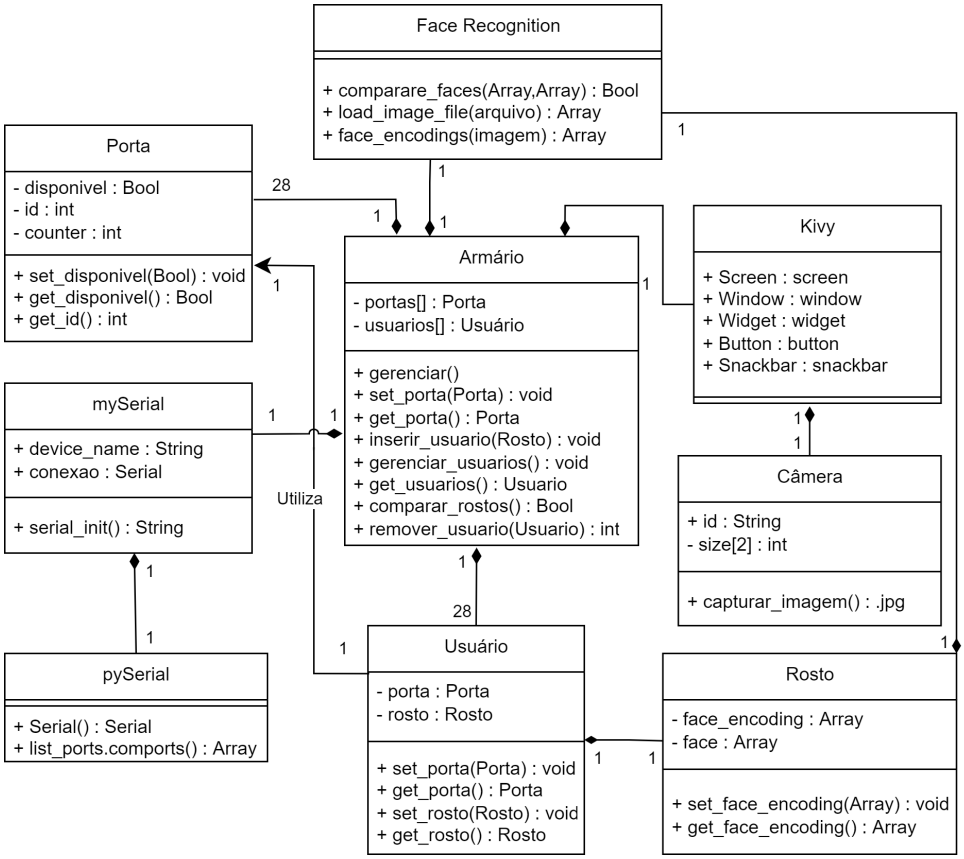
Fonte: Autor.

3.1.3.3 Classes

Um diagrama de classes é fundamental para a modelagem de um sistema, nele é possível estabelecer e identificar os componentes necessários ao software. No paradigma de programação orientada a objetos, os diagramas de classes podem ser

traduzidos em classes implementadas no software em desenvolvimento.

Figura 14 – Diagrama de classes.



Fonte: Autor.

A Figura 14 mostra o diagrama de classes para o sistema Lockersys, nele pode-se ver nove diferentes classes que abstraem o software. A principal classe é a do Armário, pois será responsável de administrar as atividades no sistema. A classe Usuário modela os atributos necessários para os usuários, ela é utilizada para armazenar a porta escolhida e o rosto.

A interface gráfica foi modelada para utilizar a biblioteca chamada Kivy (ver tópico 3.2.2), pode-se ver no diagrama que é a partir dela que o sistema tem acesso a câmera e a recursos como screen, window, widget, button e snackbar. Outra biblioteca que foi representada no diagrama de classe foi a Face Recognition (ver tópico 3.2.1) o armário utiliza os métodos relacionados carregamento de imagens, codificação das características de um rosto e comparação de rostos.

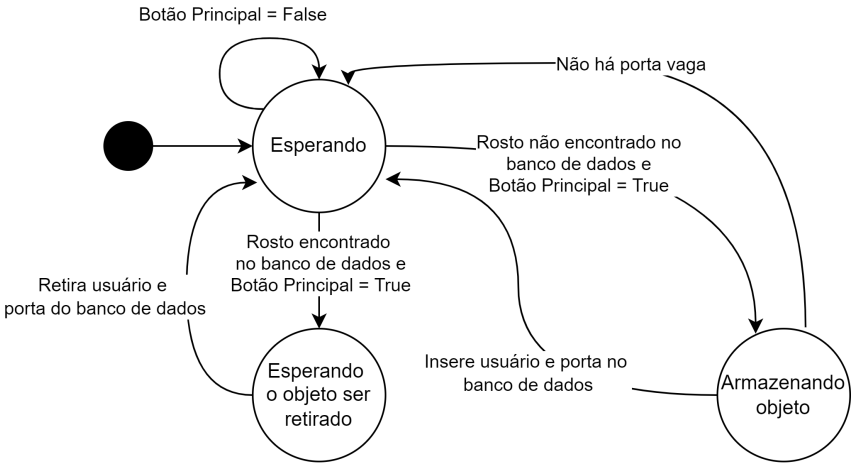
Considerando a regra de negócio RN01, foi modelado a possibilidade de o LockerSys fazer a comunicação via serial da porta que ele deseja abrir. Para essa parte do sistema foi considerado a utilização da biblioteca pySerial (ver tópico 3.2.3), entretanto foi realizado um encapsulamento por meio da classe mySerial. Em relação aos relacionamentos entre as classes foi considerado a utilização da composição entre

as classes, principalmente em relação à classe Armário, pois se esse contêiner for destruído, as outras classes são destruídas também.

3.1.3.4 Máquina de estados finita

A Figura 15 mostra, por meio de uma máquina de estados os eventos do sistema, sendo eles: Esperando, Esperando o objeto ser retirado e Armazenando objeto. O fluxo de eventos ocorre partindo do estado Esperando, caso o botão de iniciar tenha sido pressionado e o rosto não encontrado no banco de dados, ocorre então a mudança para o estado Armazenando objeto.

Figura 15 – Diagrama máquina de estados.



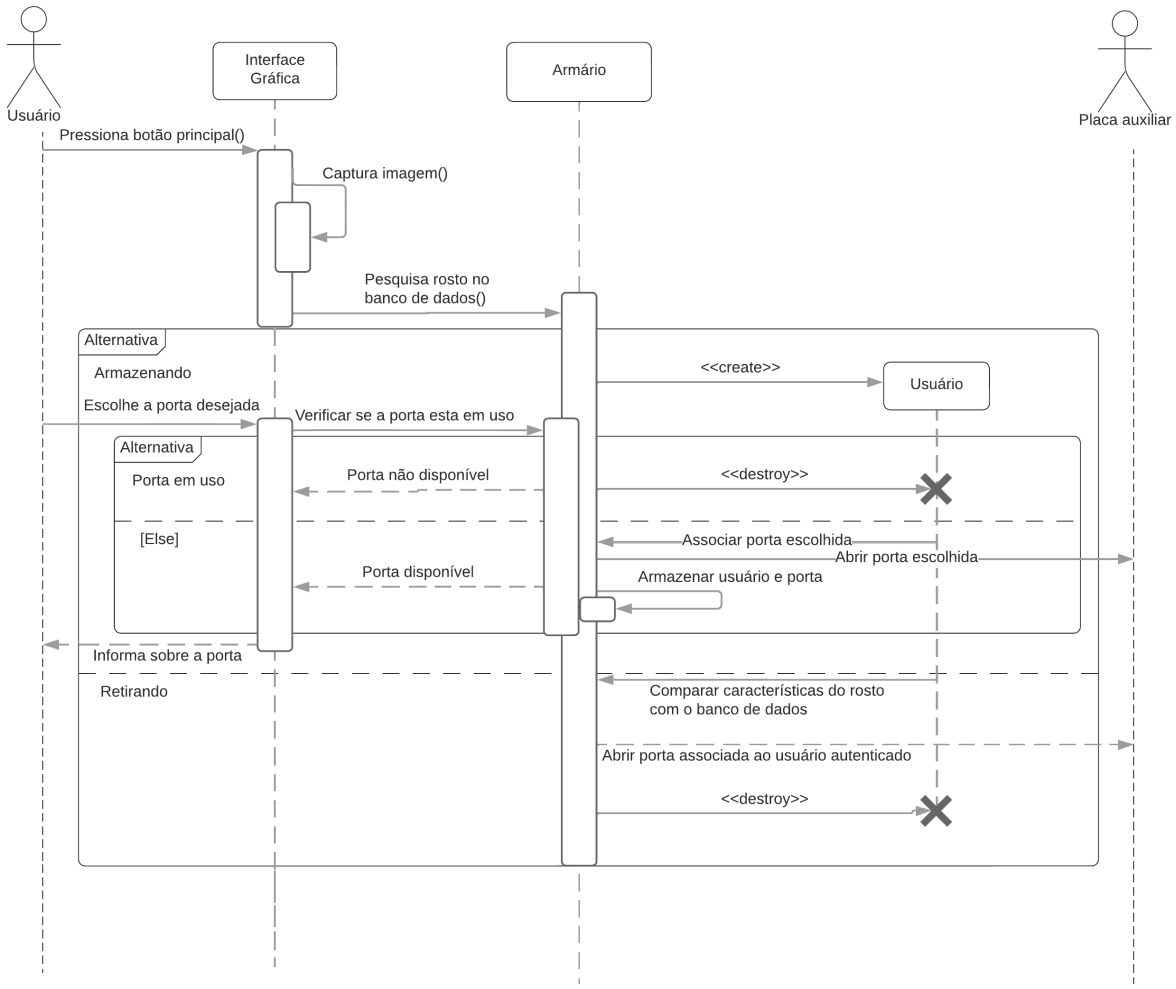
Fonte: Autor.

O estado Armazenando objeto é alterado, apenas se não houver portas vagas ou o objeto ter sido armazenado. Caso o rosto tenha sido identificado e o botão tenha sido pressionado é alterado para Esperando o objeto ser retirado, após a porta ser fechada é retornado para o estado Esperando.

3.1.3.5 Sequência

A Figura 16 mostra o diagrama de sequência para o sistema, o objetivo é apresentar de forma gráfica como as mensagens são trocadas entre os objetos no LockerSys. Pode-se ver que o usuário interage com o armário, por meio de um botão principal. A partir dessa interação o primeiro passo é realizar a captura da imagem e verificar se o rosto correspondente já está no banco de dados, se estiver, a alternativa será armazenando, o contrário será retirando.

Figura 16 – Diagrama de sequência.



Fonte: Autor.

Prosseguindo no diagrama é possível notar que internamente na alternativa Armazenando é realizado o processo de verificação se a porta está ou não disponível para o uso. Se o processo de escolha da porta obteve sucesso, é enviado um sinal para que a placa auxiliar abra a porta escolhida. A alternativa Retirando é voltada a autenticação facial, que basicamente compara as características do rosto com as presentes no banco de dados, ao encontrar o rosto correspondente é realizado a abertura da porta relacionada ao usuário encontrado.

3.2 IMPLEMENTAÇÃO DO SOFTWARE LOCKERSYS

Partindo dos requisitos e da modelagem do sistema, a próxima etapa é a implementação do software. Nessa seção são apresentadas as bibliotecas utilizadas em conjunto com exemplos das funcionalidades das classes e interação com a interface gráfica. A explicação foi separada em tópicos referentes a função no sistema,

o primeiro é Face Recognition e Dlib, o segundo é OpenCV e Kivy, por último é apresentado a pySerial.

3.2.1 Bibliotecas Utilizadas - Face Recognition e Dlib

Visando desenvolver o software LockerSys foi utilizado a Linguagem de programação Python atendendo o requisisto não funcional RNF03. A biblioteca escolhida para reconhecimento e manipulação de imagens de rostos, foi a Face Recognition (GEITGEY, 2020), essa biblioteca foi criada utilizando recursos de outra biblioteca chamada Dlib, devido ao interesse em utilizar a rede neural convolucional chamada *dlib_face_recognition_resnet_model_v1.dat.bz2*, esse modelo foi baseado no trabalho de (HE et al., 2015) e criado por Davis King na biblioteca Dlib (KING, 2017).

A rede neural *dlib_face_recognition_resnet_model_v1.dat.bz2* retorna um vetor com 128 posições, que descreve as características do rosto presente na imagem, esse processo chama-se codificação do rosto (Em inglês: face encoding). Esse vetor de codificação do rosto é importante no processo de autenticação facial, pois permite a utilização do algoritmo de distância euclidiana (Ver tópico 2.3), a biblioteca Face Recognition tem um método para utilizar esse algoritmo, dessa maneira é possível verificar se dois vetores, originados de duas imagens independentes, contêm características semelhantes o suficiente para afirmar que pertencem a mesma pessoa.

Os métodos citados anteriormente satisfazem o requisito funcional RF02 e foram utilizados na classe Rosto e na classe Armário, como mostra o diagrama de classe na Figura 14. Na classe Rosto é realizado a codificação do rosto do usuário, na Figura 17 no construtor da classe Rosto, pode-se ver que na linha 5 é carregado a imagem que será codificada, na linha 6 é feito a codificação em um vetor de 128 posições, no fim do construtor, linha 7 é realizado uma verificação se o processo retornou as características do rosto.

Na classe armário foi utilizado o método de comparação dos rostos, na Figura 18 pode-se ver como é realizado a comparação entre dois vetores com características dos rostos, na linha 2 a variável resultado recebe um valor verdadeiro (True) se a distância euclidiana resultante entre face_encoding_1 em relação ao vetor face_encoding_2 for menor que 0.6, ou falso (False) caso contrário.

A interação da classe rosto com o método comparar_rostos pode ser visualizada na Figura 19, na linha 5 é realizado a comparação, verificando se o vetor de usuários está vazio, caso esteja, significa que é o primeiro usuário utilizando o armário e basta inserir o rosto criado na linha 3 no banco de dados, caso não seja o primeiro usuário, é realizado uma comparação no banco de dados verificando se esse rosto já utilizou o armário (linha 10).

Figura 17 – Implementação da classe Rosto.

```
1  import face_recognition
2
3  class Rosto:
4      def __init__(self):
5          self._face = face_recognition.load_image_file('Imagens/
               foto.jpg')
6          self._face_encoding = face_recognition.face_encodings(
               self._face)
7          if(len(self._face_encoding) != 0):
8              self._face_encoding = self._face_encoding[0]
9
10     def set_face_encoding(self,face_encoding):
11         self._face_encoding = face_encoding
12
13     def get_face_encoding(self):
14         return self._face_encoding
```

Fonte: Autor.

Figura 18 – Método comparar_rostos da classe Armário.

```
1  def comparar_rostos(self, face_encoding_1, face_encoding_2):
2          resultado = face_recognition.compare_faces([
               face_encoding_1], face_encoding_2)
3
4          if resultado[0]:
5              return True
6          else:
7              return False
```

Fonte: Autor.

Caso seja localizado o rosto, é realizado a remoção do usuário entregando o objeto armazenado (linhas 11 - 13), esse comportamento está conforme o diagrama da Figura 15 e atende as regras de negócios RN02 e RN05, se o rosto não for encontrado, significa que é um usuário novo armazenando um objeto, então é inserido o rosto criado no banco de dados (linha 3).

3.2.2 Bibliotecas Utilizadas - OpenCV e Kivy

A biblioteca OpenCV (Open Source Computer Vision Library) é utilizada para projetos em visão computacional e aprendizado de máquina (OPENCV, 2022). Utilizada por várias empresas, como exemplo, Google, Microsoft, Intel, IBM e Sony. A OpenCV oferece mais de 2500 algoritmos, empregados em aplicação diversas, como exemplo, reconhecimento e detecção de objetos e faces, extração de modelos 3D de

Figura 19 – Método gerenciar_usuarios() da classe Armário.

```
1  def gerenciar_usuarios(self):
2          print("gerenciar_usuarios()")
3          rosto = Rosto()
4          if len(rosto.get_face_encoding()) != 0:
5              if(len(self.get_usuarios()) == 0):
6                      print("Primeiro usuario")
7                      self.inserir_usuario(rosto)
8              else:
9                  for usr in self.get_usuarios():
10                 if(self.comparar_rostos(rosto.
                       get_face_encoding(), usr.get_rosto().
                       get_face_encoding())):
11                     print("Comparacao bem sucedida")
12                     aux = self.remover_usuario(usr)
13                     return aux
14             else:
15                 print("Demais usuarios")
16                 self.inserir_usuario(rosto)
17
18             print(self.get_usuarios())
19             return "Armazenar objeto"
20         else:
21             return "Rosto nao encontrado"
```

Fonte: Autor.

objetos e permite o acesso a câmera do computador.

Um exemplo da utilização da biblioteca OpenCV para utilizar a câmera do computador é apresentado na Figura 20, na primeira linha é importado a biblioteca, seguindo no código é criado uma classe chamada Camera, em que no construtor é configurado atributos para resolução. No método capturar_imagem() é realizado o acesso a câmera na linha 9 e configurado a resolução nas linhas 10 e 11, para certificar que a câmera está acessível e foi possível capturar uma imagem são feitas condições na linha 13 e na linha 16, respectivamente, então caso passe nessas condições é gravado a imagem em uma pasta, no exemplo é usado o caminho 'Imagens/foto.jpg' (linha 17), caso contrário ocorre uma exceção (linha 19).

Uma biblioteca que pode ser utilizada para criar interfaces gráficas na Lingugem de programação Python é a Kivy, os benefícios de se aplicar essa biblioteca são: possibilidade de multiplataforma, ou seja, é possível criar uma aplicação para Windows, Linux, MacOS, Android, IOS e Raspberry Pi, outro ponto forte é a licença MIT, que permite o uso em produtos comerciais de maneira gratuita (KIVY, 2021).

Em virtude da regra de negócio RN09, foi escolhido implementar a interface gráfica com Kivy, a biblioteca Kivy utiliza internamente a OpenCV para fazer captura

Figura 20 – Exemplo de utilização da biblioteca OpenCV para acessar a câmera do computador.

```
1  import cv2
2
3  class Camera:
4      def __init__(self):
5          self._largura: int = 640
6          self._altura: int = 480
7
8      def capturar_imagem(self):
9          camera = cv2.VideoCapture(0) # escolhe a primeira webcam
               do computador.
10         camera.set(3,self._largura) # Configura a resolucao da
               camera
11         camera.set(4,self._altura)
12
13         if camera.isOpened():
14             _,frame = camera.read()
15             camera.release()
16             if _ and frame is not None:
17                 cv2.imwrite('Imagens/foto.jpg', frame)
18         else:
19             raise IOError("Nao pode abrir a webcam")
```

Fonte: Autor.

de imagens semelhante ao exemplo da Figura 20, e esse recurso foi utilizado na tela inicial da interface gráfica, durante o desenvolvimento do software LockerSys.

No desenvolvimento utilizando o framework Kivy é construído uma estrutura, utilizando uma Linguagem chamada KvLang em que se pode declarar Widgets a serem utilizados na tela. Associado a uma estrutura de tela está uma classe desenvolvida em Python que contém as funcionalidades dos Widgets, um exemplo desse conceito é apresentado na Figura 21 e na Figura 22.

Na Figura 22 é criado a estrutura para a tela chamada CameraTela, na linha 5 é especificado que será utilizado um MDBoxLayout com orientação vertical (linha 6), o próximo widget é o MDToolbar configurado com a cor cinza (linha 8), no meio da tela foi especificado para a câmera (linhas 12-16), por último é configurado o widget Button, esse componente é o botão principal na modelagem, pois indica a utilização do armário, na linha 24 foi descrito o evento de on_touch_down: que chama uma função chamada capture().

Na Figura 21 pode-se ver a implementação da classe CameraTela, o principal método é chamado capture() (linha 7), pois realiza o início da troca de mensagens no sistema de acordo com o diagrama de sequência na Figura 16, dessa forma foi

Figura 21 – Implementação da tela inicial chamada CameraTela, código em Python.

```
class CameraTela(MDScreen):
    def __init__(self,sm, **kw):
        super().__init__(**kw)
        self.ids.camera.play = True
        self.screen_m = sm

    def capture(self):
        camera = self.ids['camera']
        camera.export_to_png('Imagens/foto.jpg')
        print("Captured")
        aux = armario_obj.gerenciar_usuarios()
        if aux == "Armazenar objeto":
            self.screen_m.transition.direction = 'left'
            self.screen_m.current = 'PortasTela'
        elif aux == "Rosto nao encontrado":
            self._snackbar_erro()
        else:
            try:
                con_serial.conexao.write((str(aux) + '\n').encode
                    ('utf-8'))
                var = con_serial.conexao.read_until()
                print(var)
            except:
                print("Falha na comunicação")


    def _snackbar_erro(self):
        snackbar = Snackbar(
            text="Não foi possível localizar o rosto, tente
                novamente",
            bg_color=(1,0,0,1),
            snackbar_x="10dp",
            snackbar_y="10dp"
        )
        snackbar.size_hint_x = (
                Window.width - (snackbar.snackbar_x * 2)
            ) / Window.width

        snackbar.open()
```

Fonte: Autor.

Figura 22 – Implementação da tela inicial chamada CameraTela, código em KvLang.

```
1  #:kivy 2.0.0
2
3  <CameraTela>:
4      name: 'CameraTela'
5      MDBoxLayout:
6          orientation: 'vertical'
7          MDToolbar:
8              md_bg_color: 100/255 , 100/255, 100/255, 1
9
10             title: "LockerSys"
11
12         Camera:
13             id: camera
14             #resolution: (640, 480)
15             play: False
16             size: root.width, root.height
17
18         Button:
19             text: "Camera"
20             pos_hint:{"center_x": .5, "center_y": .5}
21             size_hint: root.width,0.2
22             background_normal:''
23             background_color: 1, 100/255, 10/255, 1
24             on_touch_down:
25                 root.capture() if self.collide_point(*args[1].pos
                       ) else False
```

Fonte: Autor.

alcançado o requisito funcional RF03 e a regra de negócio RN03.

Visando atender o requisito funcional RF01 e a regra de negócio RN01 foi criado a segunda tela do sistema, essa tela foi desenvolvida seguindo a mesma maneira usada na tela CameraTela, na Figura 24 é apresentado a estrutura em KvLang, foi utilzado o widget ScrollView (linha 5) que permite um efeito de rolar os widgets filhos, que nesse caso são os botões das portas disponíveis. Na Figura 23 foi desenvolvido o comportamento da segunda tela, o principal comportamento é o do construtor, mais especificamente na linha 7 e 8, em que é instanciado os 28 botões na tela em forma de pilha, outra função importante está localizada nas linhas 10 à 19, aqui é configurado as cores de acordo com a disponibilidade da porta, sendo vermelho para em uso e verde para disponível, com essa indicação visual foi alcançado a regra de negócio RN04.

O botão para cada porta utilizado na tela PortasTela, é configurado seguindo a estrutura apresentada na Figura 26, no KvLang foi configurado o tamanho da fonte

Figura 23 – Implementação da tela chamada PortasTela, código em Python.

```python
class PortasTela(MDScreen):
    def __init__(self,sm, **kw):
        super().__init__(**kw)
        self.ids.stackportas.padding = 0.05*Window.width, "150dp"
        self.ids.stackportas.spacing = (Window.width/5 - 2*0.05*
           Window.width)/3

        for i in range(28):
            self.ids.stackportas.add_widget(Porta(i+1,sm))

    def callback(self):
        i = 27
        for child in self.ids.stackportas.children:
            print(i)
            print(armario_obj.get_portas()[i].get_disponivel())
            if(armario_obj.get_portas()[i].get_disponivel() ==
               True):
                child.background_color = 57/255,1,20/255,1
            else:
                child.background_color = 255,0,0,1
            i = i-1
```

Fonte: Autor.

Figura 24 – Implementação da tela chamada PortasTela, código em KvLang.

```
<PortasTela>:
    name: 'PortasTela'
    on_pre_enter: root.callback()

    ScrollView:
        bar_pos_y: 'right'
        bar_width: 10
        effect_cls: 'ScrollEffect'

        StackLayout:
            id: stackportas
            size_hint: (1, None)
```

Fonte: Autor.

em 22 (linha 2), a cor inicial que é verde (linha 4) e o evento on_press que dispara o método b_porta(). Na Figura 25 está a classe que descreve o botão para cada porta, pode-se ver que ela herda da classe Kivy Button, para indicar o comportamento de botão, a principal característica dessa classe é o método b_porta(), pois é a resposta ao evento de pressionar o botão da porta desejada.

Figura 25 – Configuração do Widget Porta, código em Python.

```
1  class Porta(Button):
2      def __init__(self,id,sm, **kwargs):
3          super().__init__(**kwargs)
4          self.screen_m = sm
5          self.size_hint=(None,None)
6          self.id = id
7          self.text = str(id)
8
9      def b_porta(self):
10         if(armario_obj.get_portas()[self.id-1].get_disponivel()
               == True):
11             print(f"Porta: {self.id}")
12             self.background_color = 255,0,0,1
13             armario_obj.get_ultimo_usuario().set_porta(
                   armario_obj.get_portas()[self.id-1])
14             armario_obj.set_porta(self.id-1,False)
15             self.screen_m.transition.direction = 'right'
16             self.screen_m.current = 'CameraTela'
17             try:
18                 con_serial.conexao.write((str(self.id) + '\n').
                       encode('utf-8'))
19             except:
20                 print("Falha na comunicação")
21         else:
22             snackbar = Snackbar(
23             text="Essa porta está em uso, favor escolha outra",
24             bg_color=(1,0,0,1),
25             snackbar_x="10dp",
26             snackbar_y="10dp"
27             )
28             snackbar.size_hint_x = (
29                 Window.width - (snackbar.snackbar_x * 2)
30             ) / Window.width
31             snackbar.open()
```

Fonte: Autor.

É importante notar que no código presente na Figura 25 é permitido utilizar uma porta apenas se ela estiver disponível (linha 10), fato que respeita o diagrama de sequência da Figura 16, caso ela esteja livre, o usuário e a porta escolhida (linha 13) são armazenados, e é enviado um sinal via serial para abertura dessa porta, caso a porta não esteja disponível é exibido uma mensagem de erro (linhas 21 - 31).

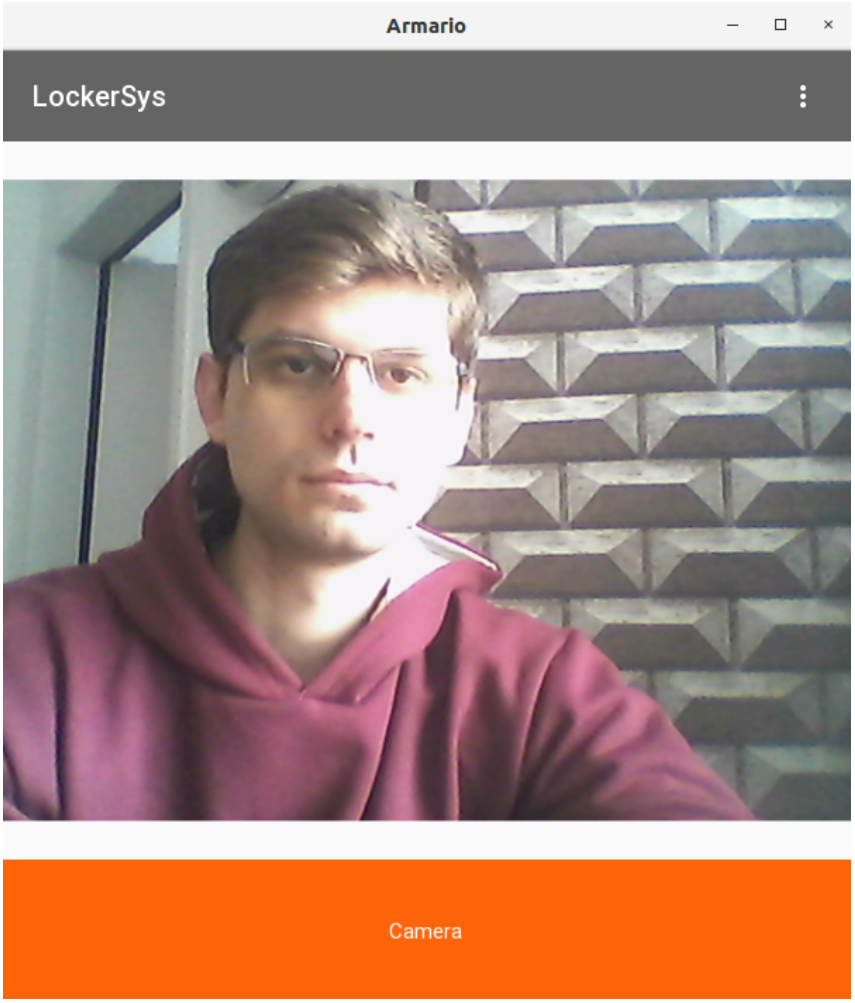
Figura 26 – Configuração do Widget Porta, código em KvLang.

```
<Porta>:
    font_size: 22
    background_normal: ''
    background_color: 57/255,1,20/255,1
    on_press:
        self.b_porta()
```

Fonte: Autor.

Na Figura 27 é apresentado o resultado da tela chamada CameraTela, nela pode-se ver o botão principal em laranja escrito Camera, no meio está a imagem da câmera embutida. A outra tela é mostrada na Figura 28 nela pode-se ver que existem 28 botões verdes, o que significa que todas as portas estão livres.

Figura 27 – Tela principal com a câmera e o botão em laranja para captar imagens.



Fonte: Autor.

Figura 28 – Segunda tela com os botões para escolha das portas.

Fonte: Autor.

3.2.3 Biblioteca Utilizada - pySerial

A biblioteca pySerial encapsula o acesso às portas seriais no dispositivo em que ela estiver operando (PYSERIAL, 2020), com a pySerial é possível comunicar dados via serial entre diferentes dispositivos. Para atender o requisito funcional RF04 e a regra de negócio RN10, foi utilizado a biblioteca pySerial para desenvolver um módulo de comunicação serial no sistema. O principal motivo para essa implementação, é a utilização de uma placa auxiliar (ver tópico 3.3.3) projetada para acionar e sensorear os estados das portas do armário.

Na Figura 29 é apresentado a classe mySerial desenvolvida, no construtor é realizado a identificação da porta onde está conectado a placa auxiliar (linha 8), após identificar a porta é feito uma tentativa de conexão utilizando o contrutor da biblioteca pySerial (linhas 9-10), nos parâmetros é inserido: porta, baudrate (9600), bytesize (8), paridade (’N’), stopbits (1), timeout=1 (tempo de expiração), caso não seja possível realizar a comunicação é informado que não foi possível conectar.

Figura 29 – Implementação da classe mySerial, utilizando a biblioteca pySerial.

```
1  from serial import Serial
2  from serial.tools import list_ports
3
4  class mySerial:
5      def __init__(self):
6          self.device_name = self.serial_init()
7          try:
8              self.conexao = Serial(self.device_name, 9600, 8, 'N',
                   1, timeout=1)
9          except:
10             print("Conexão mal sucedida")
11     def serial_init(self):
12         usb_device_list = list_ports.comports()
13         device_name_list = [ port.device for port in
              usb_device_list ]
14         if len(device_name_list) == 0:
15             return 'erro'
16         _device_name = device_name_list[0]
17         return _device_name
```

Fonte: Autor.

3.3 HARDWARE - ELETRÔNICA

Nessa seção é discutido os componentes de hardware utilizados para o desenvolvimento da parte eletrônica do trabalho, foi divido em tópicos, primeiramente é apresentado o micro-computador Raspberry Pi 3B, a próxima peça essencial para o sistema é a fechadura elétrica que realiza a abertura automática de uma porta, por último é relatado a respeito da placa auxiliar que permite o controle e sensoriamento das portas do armário.

3.3.1 Raspberry Pi 3B

Raspberry Pi 3B é um computador de tamanho e custo reduzidos, fabricado pela fundação de tecnologia Raspberry Pi. Os computadores da Raspberry Pi possuem uma porta para a inserção de cartão Micro SD, e nesse cartão é instalado um sistema operacional.

O sistema operacional oficial utilizado nos dispositivos da Raspberry Pi é o Raspberry Pi OS (FOUNDATION, 2022), no trabalho foi utilizado o Raspberry Pi OS (Legacy) com desktop, sistema baseado no Linux Debian (Buster). As especificações da Raspberry pi 3B utilizada são apresentadas no Quadro 1. Em conjunto com a Raspberry Pi 3B foi usado uma câmera de 5MP acoplada na porta CSI (ver Figura 10. Devido o emprego desses recursos, foi alcançado o requisito funcional RF08 e regras

Quadro 1 – Especificações da Raspberry Pi 3B.

Quantidade	Componente
1	CPU - Broadcom BCM2837 64bit, com quatro núcleos de 1.2GHz.
1	GB de memória RAM.
1	BCM43438 wireless LAN.
1	Bluetooth Low Energy (BLE).
1	100 Base Ethernet.
40	Pinos de GPIO.
4	Portas USB.
1	CSI (Camera Serial Interface) para conexão da Raspberry Pi camera.
1	Porta Micro SD para carregar o sistema operacional e armazenar os dados.
1	Porta HDMI.
1	Porta Micro USB, para fonte de alimentação, opera em tensão de 5V e correntes até 2.5A.

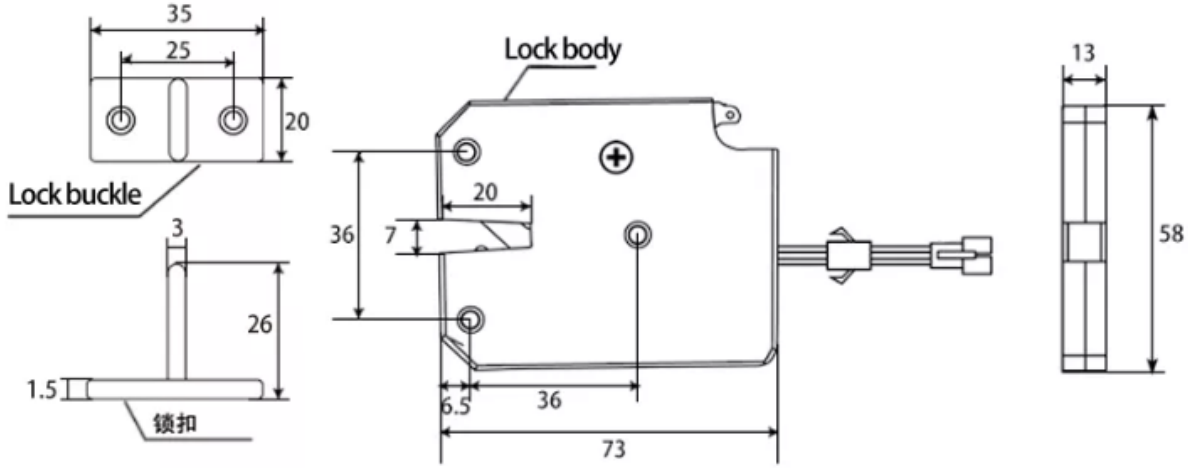
Fonte: (FOUNDATION, 2016).

de negócio RN07 e RN08.

3.3.2 Fechaduras elétricas

A fechadura utilizada no protótipo é fabricada pela SARY Eletronics (SARY, 2022), diferentemente das fechaduras de princípio eletromagnético (Solenoide) a fechadura utilizada nesse trabalho é ativada, por meio, do encolhimento de um fio, cujo o material é uma liga de titânio com níquel chamada Nitinol. Na Figura 30 é mostrado as dimensões da fechadura e a fivela de encaixe.

Figura 30 – Desenho técnico da fechadura, com cotas das dimensões em milímetros.

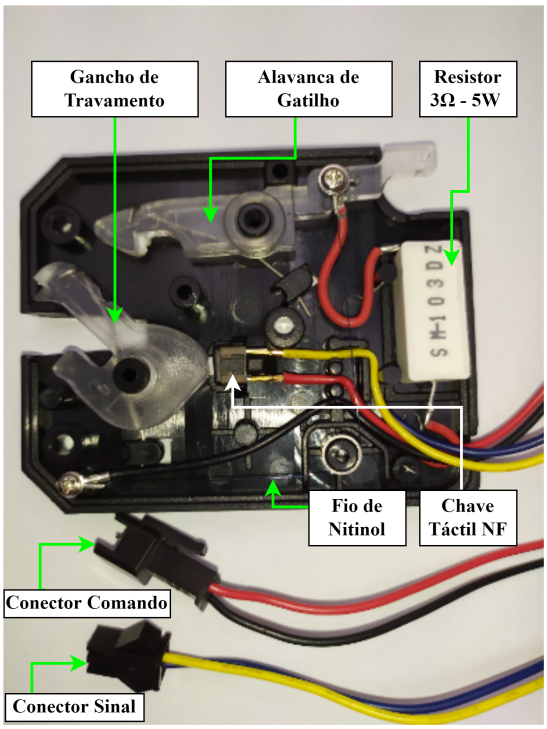


Fonte: (SARY, 2022)

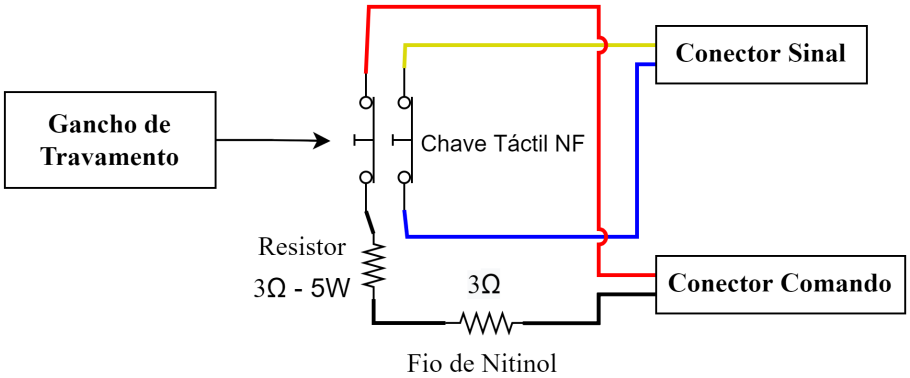
Características da fechadura utilizada:

1. Fabricada em plástico ABS (Acrylonitrile Butadiene Styrene).
2. Suporta trações de até 75kg, entre a fivela de encaixe em metal e o gancho de travamento da fechadura.
3. A prova da água.
4. Tensão de operação: 5V.
5. Quantidade de ciclos de abertura e fechamento até a falha: 500.000 vezes.
6. Algumas aplicações típicas: armários de entrega, cabines refrigeradas, caixas de correio, armários de auto-atendimento e vestiários.

Figura 31 – Fechadura utilizada no TCC, vista interna e diagrama das conexões.



(a) Vista interna dos componentes da fechadura.



(b) Diagrama com conexões internas dos componentes da fechadura.

Fonte: Autor.

Na Figura 31a pode-se ver a estrutura interna da fechadura, o sistema de trava é composto pela Alavanca de Gatilho, que retêm o Gancho de Travamento até que seja

fornecido uma diferença de potencial de 5V no conector de comando. A fechadura conta com dois sistemas de proteção contra a sobrecarga no Fio de Nitinol, o primeiro é composto pelo Resistor de 3Ω e 5W que limita a corrente, o segundo método de proteção é realizado pela Chave Táctil NF acionada pelo Gancho de Travamento, pois quando aberto empurra a Chave Táctil fazendo com que ela abra o circuito, na Figura 31b é mostrado os dois meios de proteção.

O sinal indicando se a porta está aberta ou fechada, é acionado em paralelo utilizando a mesma Chave Táctil, sendo atuado pelo Gancho de Travamento, quando ele está fechado a Chave Táctil é liberada fechando o circuito, como mostra o diagrama da Figura 31b. Ao empregar essa fechadura no sistema, foi alcançado o requisito não funcional RNF01 e a regra de negócio RN06.

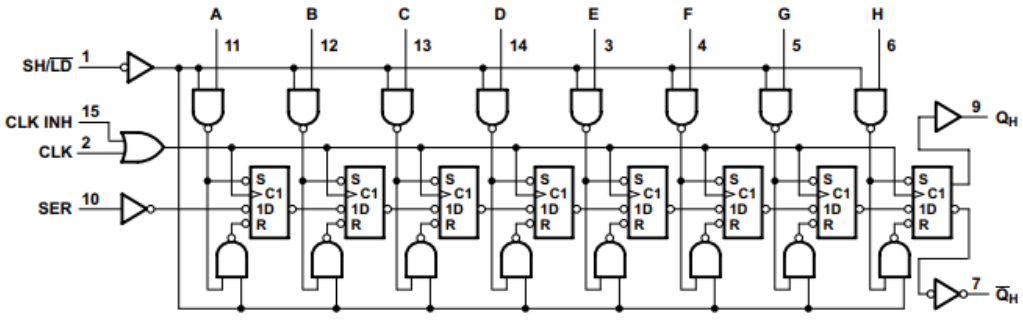
3.3.3 Projeto da placa auxiliar

Durante o desenvolvimento do projeto foi necessário projetar uma placa auxiliar (ver Figura 54 Apêndice B), devido à necessidade de controlar as 28 portas e receber os 28 sinais dos estados das mesmas. No trabalho essa placa foi utilizada para comunicar-se com a Raspberry Pi 3B via serial, as informações transmitidas referem-se aos estados das portas e ao acionamento das fechaduras. Essa placa tem como base os seguintes componentes: registradores de deslocamento 74HC165 e 74HC595, os mosfets AO3402 e o arduino nano.

3.3.3.1 74HC165

O registrador de deslocamento 74HC165 com oito bits é classificado como entrada paralela e saída serial (INSTRUMENTS, 2015), na Figura 32 pode-se ver o diagrama de funcionamento, que apresenta como as oito entradas de A-H são conectadas a portas do tipo nand, o pulso de carregamento dos flip-flops é realizado por SH, dessa maneira os bits são deslocados a cada borda de subida do CLK.

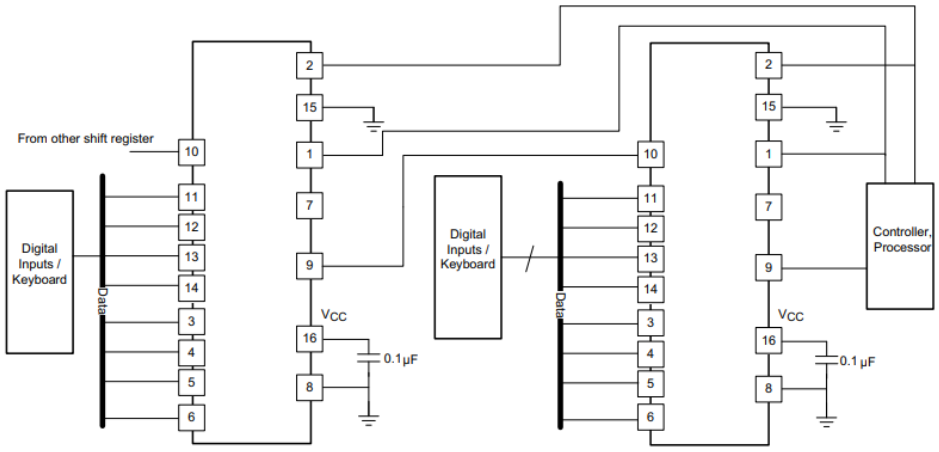
Figura 32 – Diagrama de funcionamento do registrador de deslocamento 74HC165.



Fonte: Adaptado de (INSTRUMENTS, 2015)

No circuito do trabalho foi utilizado quatro registradores 74HC165, com resistores de pull-down nos pinos de entrada dos oito bits. Foi realizado o cascateamento de quatro circuitos integrados para comportar trinta e dois sinais digitais, a Figura 33 e Figura 35 mostram como foi realizado essa topologia e as ligações elétricas.

Figura 33 – Aplicação típica do circuito integrado 74HC165.



Fonte: Adaptado de (INSTRUMENTS, 2015)

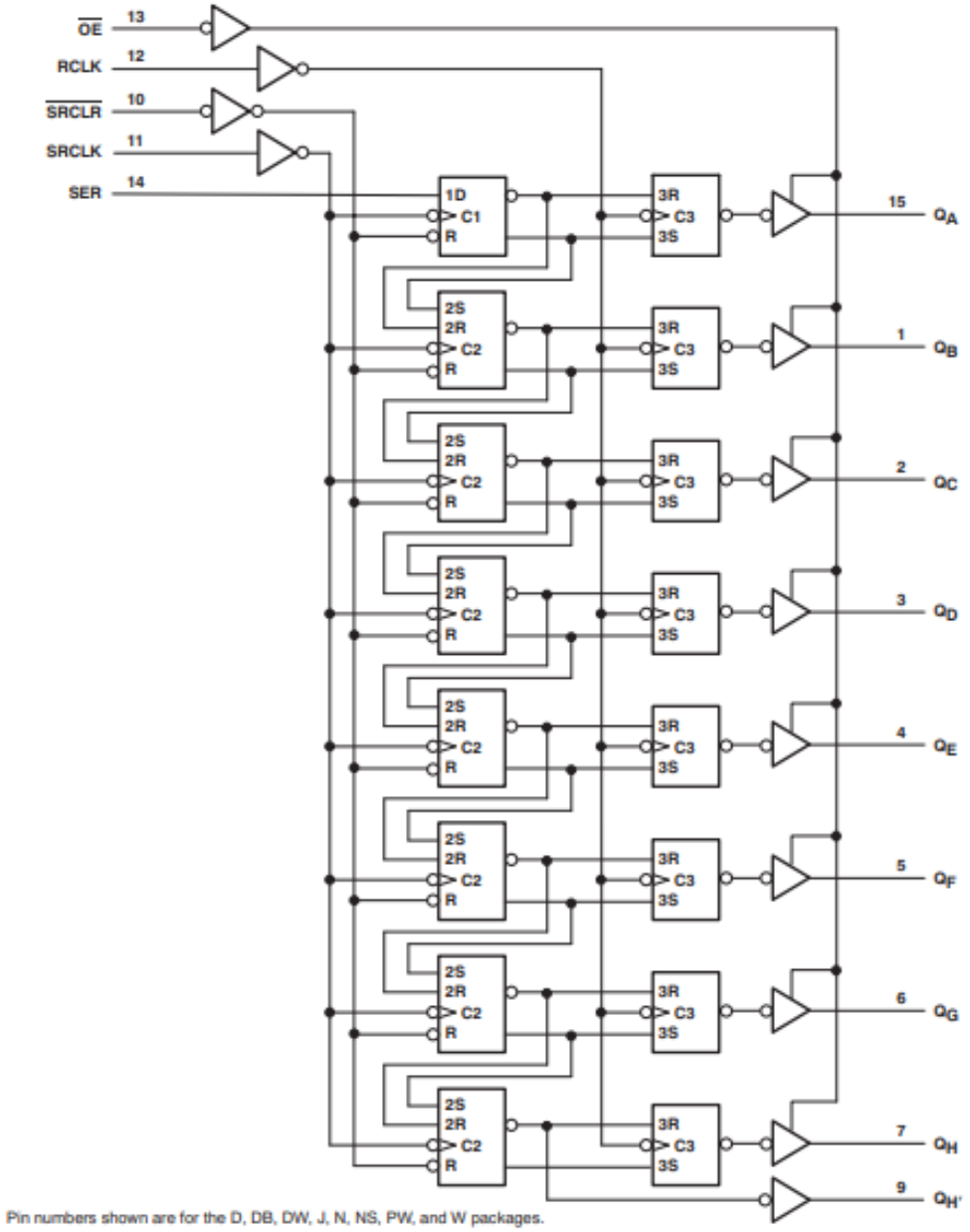
3.3.3.2 74HC595

O registrador de deslocamento e armazenamento 74HC595 é classificado como entrada serial e saída paralela com oito bits, tem oito estágios de deslocamento em conjunto com oito registradores de armazenamento, fato que permite clocks de entrada de dados e armazenamento separados (INSTRUMENTS, 2022). Na Figura 34 pode-se ver o diagrama de blocos funcional para o circuito integrado, existem oito registradores de deslocamento, ligados em série com oito registradores de armazenamento, que na borda de subida do RCLK é liberado os dados nas saídas de QA até QH. No circuito do trabalho foram utilizados quatro 74HC595 em cascata, como pode-se ver na Figura 35, cada saída QHť é ligada na entrada serial do próximo circuito integrado. As saídas de QA até QH de cada 74HC595 foram identificadas como Po's, indo de Po1 até Po32, a Figura 48 apresenta o destino desses sinais, com a função de polarizar o gate dos mosfets.

3.3.3.3 KiCad

Lançado inicialmente em 1992, o KiCad é um software de código aberto, desenvolvido para elaboração de esquemáticos eletrônicos e projetos de placas de circuito impresso (KICAD, 2021). No trabalho foi utilizado a versão 6.0 do KiCad, para modelar a placa auxiliar, na Figura 35 está representado o esquemático do circuito utilizado para comandar e sensoriar as portas do armário. A eletrônica é composta

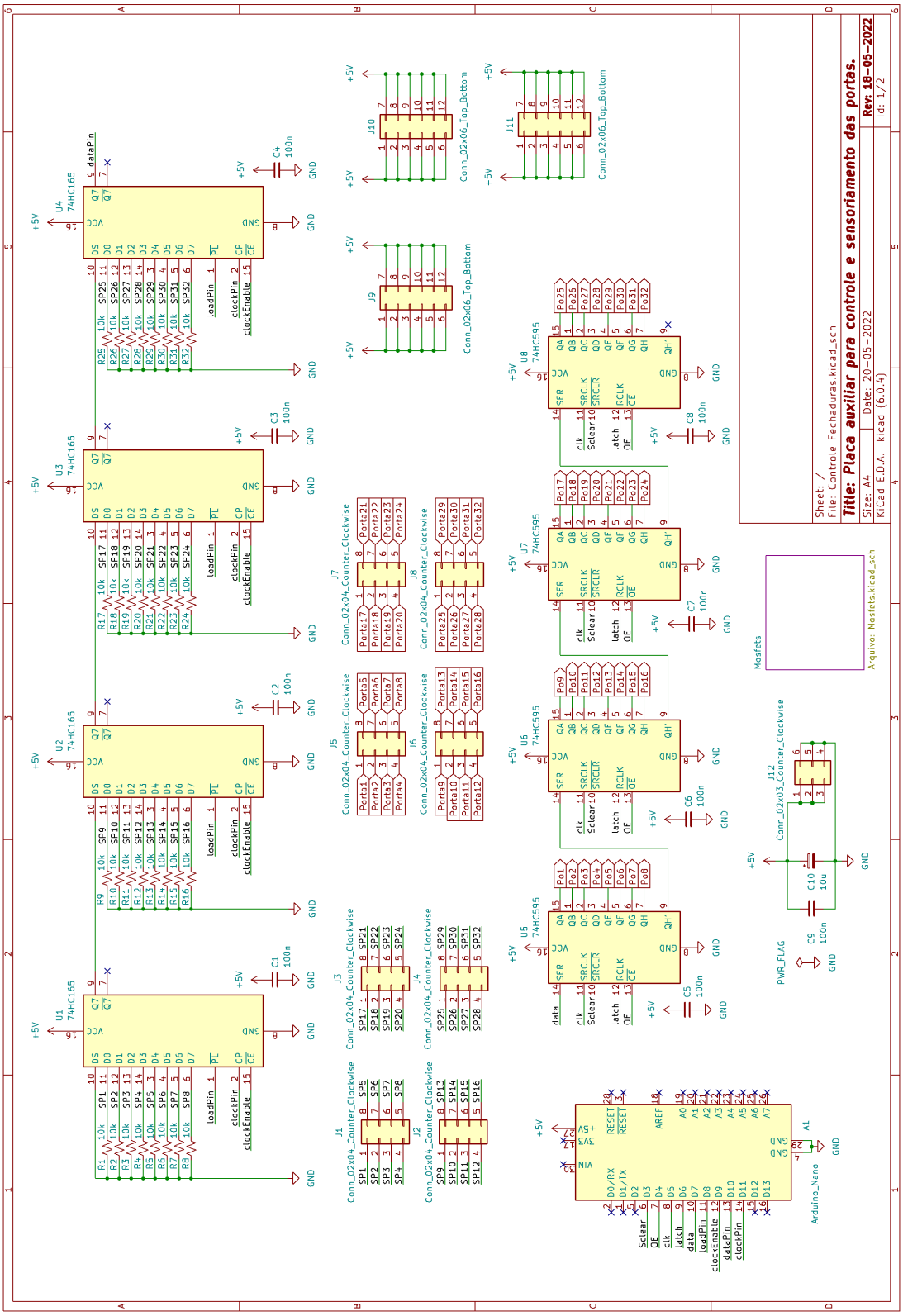
Figura 34 – Diagrama de funcionamento do registrador de deslocamento 74HC595.



Fonte: (INSTRUMENTS, 2022)

por quatro circuitos integrados 74HC165, utilizados para obter os estados das portas, outros quatro circuitos integrados 74HC595, usados para ativar o circuito de gatilho da porta. O circuito integrado 74HC595 permite uma corrente de saída de 35 mili amperes (INSTRUMENTS, 2022), devido a esse fato foi projetado o uso de trasistores mosfets como chaves para a comutação e acionamento das portas do armário. Na Figura 48 (Apêndice A) pode-se ver os trinta e dois mosfets, que a placa comporta, os cálculos utilizados no projeto da comutação dos mosfets estão no tópico 3.3.3.4. Na Figura 47 (Apêndice A) pode-se ver o projeto do roteamento das trilhas, foi utilizado duas camadas e ambas possuem um plano de terra.

Figura 35 – Esquemático da PCI criada no Kicad.



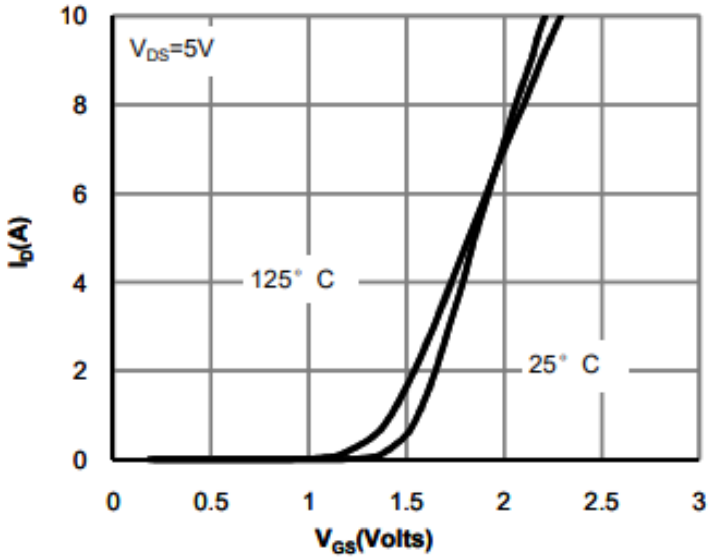
Fonte: Autor.

3.3.3.4 Projeto da comutação dos Mosfets

Nesse tópico é apresentado o projeto da comutação dos transistores mosfets AO3402, que são do tipo de intensificação de canal N. Foi utilizado uma polarização por divisão de tensão, que para o caso do mosfet é utilizado a Equação 4 (BOYLESTAD; NASHELSKY, 2013), considerando o circuito da Figura 37a é calculado o valor de $V_{gs} = 4.735V$, dessa maneira pode-se ver que de acordo com a tensão de threshold $V_{gs(th)} = 1.5V$ foi obtido a saturação do mosfet AO3402, pois $V_{gs} > V_{gs(th)}$, o resultado dos cálculos anteriores podem ser confirmados, por meio do gráfico da Figura 36, conforme a tensão V_{gs} ultrapassa a tensão de threshold $V_{gs(th)} = 1.5V$ no eixo das abcissas a I_D (Corrente de dreno) aumenta.

$$V_g = \frac{R_2 \cdot V_{DD}}{R_1 + R_2} \tag{4}$$

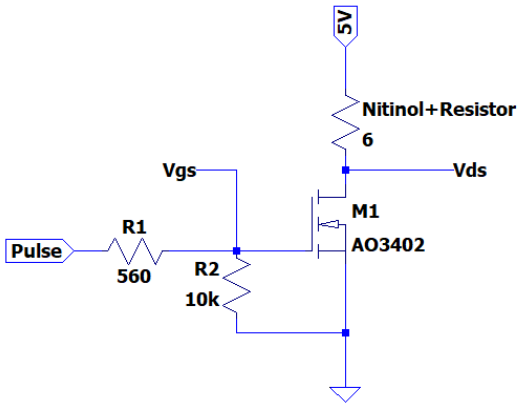
Figura 36 – Curvas relacionando a tensão V_{gs} com a corrente I_d (ALPHAEOMEGA, 2011).



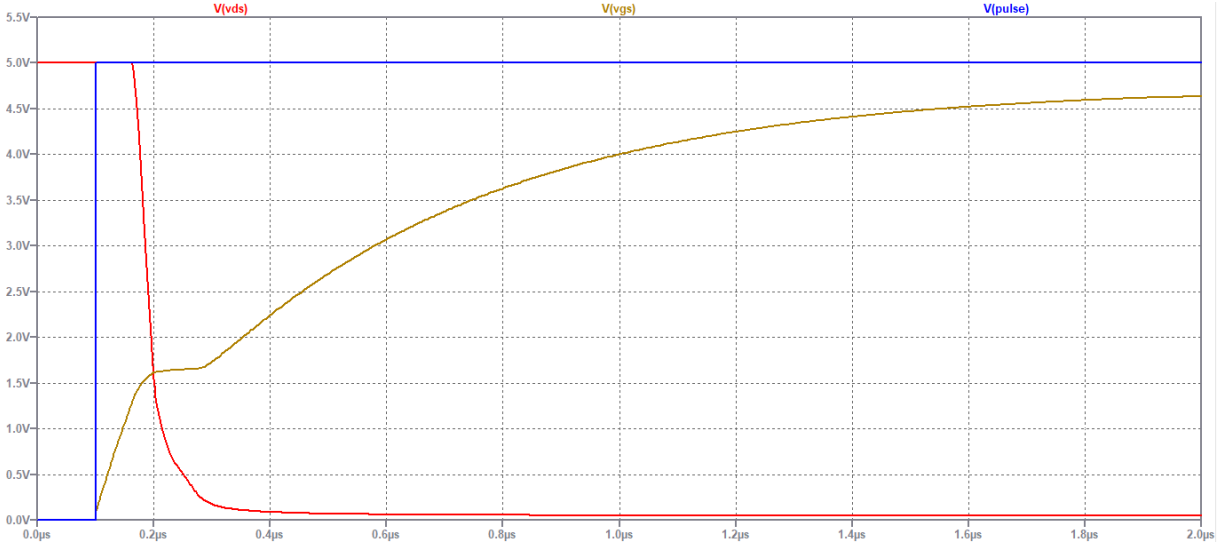
Fonte: (ALPHAEOMEGA, 2011).

No circuito foi considerado a proteção contra falha de curto-circuito no mosfet, dessa forma a corrente é limitada a $\frac{5V}{R_1} = \frac{5V}{560} = 8.93mA$ e o circuito integrado 74HC595 é preservado, outra proteção é em relação ao R_2 (ver Figura 37a), esse resistor evita que ruídos espúrios acionem o mosfet. Na Figura 37b pode-se ver a simulação realizada para analisar o tempo de chaveamento, considerando uma corrente mínima sobre o Nitinol+Resistor de $I_{min} = 0.8A$ o tempo é de $0.12us$, que está adequado para a aplicação, outra característica presente na figura é que a tensão V_{gs} tende a 4.735V que foi o valor projetado em regime permanente.

Figura 37 – Simulação realizada no LTspice do circuito de comutação.



(a) Esquemático do circuito de comutação.



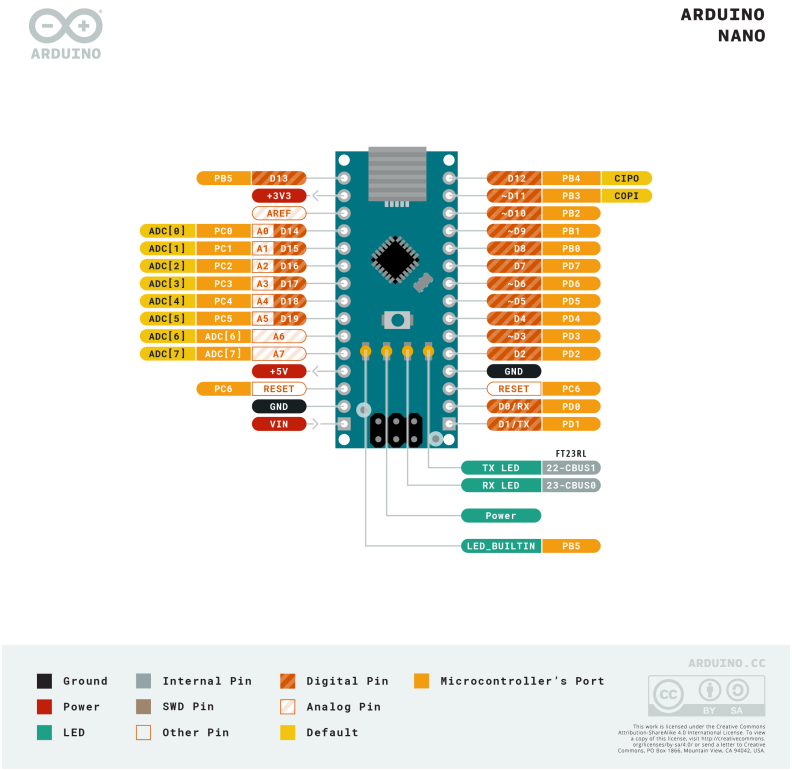
(b) Gráfico da simulação contendo a resposta transitória da comutação do mosfet AO3402.

Fonte: Autor.

3.3.4 Arduino Nano

Arduino é um projeto de código aberto, com o objetivo de facilitar o acesso a tecnologia de microcontroladores. O fato de ter uma grande comunidade, muitas bibliotecas que facilitam a interação com diversos componentes eletrônicos e pela acessibilidade no custo do hardware, faz do Arduino o microcontrolador de milhares de projetos ao redor do mundo (ARDUINO, 2018). Existem diversos modelos de placas, algumas são mais indicadas para aplicações em BLE (bluethooth low energy), WIFI (conexão wireless), IOT (internet of things), uma placa muito conhecida é a Arduino nano, que recebe um chip Atmel AVR de 8 bits, na Figura 38 pode-se ver um mapa de pinos dessa placa. A placa de desenvolvimento Arduino nano possui um microcontrolador AVR ATMega328P, esse microcontrolador é gravado normalmente através do USB de um computador que use o ambiente de desenvolvimento integrado da própria empresa Arduino, nesse software programa-se na Linguagem C ou C++

Figura 38 – Arduino Nano Diagrama dos pinos.



Fonte: (ARDUINO, 2022b)

(ARDUINO, 2022a).

3.3.4.1 Desenvolvimento do Firmware

O desenvolvimento do firmware foi direcionado a atender os requisitos funcionais RF05 e RF06, para isso foi implementado um firmware para o microcontrolador ATmega328P, utilizando a Linguagem de programação C++. O código foi escrito de maneira estruturada, sendo baseado nas seguintes funções: comunicação serial e leitura e escrita dos registradores. Na Figura 39 é apresentado a função ler_registradores, ela é responsável por ler os registradores 74HC165 e retornar os estados desses registradores, nesse intuito é realizado o carregamento de todos os 32 bits (linhas 7 - 11), após esse carregamento os bits são lidos de maneira serializada e armazenados na variável bytesVal, utilizando o operador de deslocamento binário a esquerda (linha 17), por último é retornado essa variável (linha 26), um exemplo de retorno dessa função é uma sequência de bits "01111110 11111111 11111111 01111111"esse resultado mostra que a porta 1, 8 e 25 estão abertas e as restantes estão fechadas.

Na função abrir ocorre o processo inverso do que foi realizado na função anterior ler registradores, o comportamento aqui é configurar um buffer chamado dataArray com tamanho de quatro bytes (linhas 3 - 6) e deslocar um bit até uma

Figura 39 – Implementação da função ler_registradores() desenvolvida em C++.

```
long ler_registradores()
{
  long bitVal;
  long bytesVal = 0;
  // carrega todos os bits de forma paralela
  digitalWrite(clockEnable, HIGH);
  digitalWrite(loadPin, LOW);
  delayMicroseconds(PULSE_WIDTH_USEC);
  digitalWrite(loadPin, HIGH);
  digitalWrite(clockEnable, LOW);
  // Loop para ler cada bit da saida serial dos quatro SN74HC165N
  for (int i = 0; i < DATA_WIDTH; i++)
  {
    bitVal = digitalRead(dataPin);
    bytesVal |= (bitVal << i);
    // Pulso no Clock desloca para o proximo bit.
    digitalWrite(clockPin, HIGH);
    delayMicroseconds(PULSE_WIDTH_USEC);
    digitalWrite(clockPin, LOW);
  }
  return (bytesVal);
}
```

Fonte: Autor.

Figura 40 – Implementação da função abrirPorta, desenvolvida em C++.

```
byte abrirPorta(int num) {
  digitalWrite(OE, LOW);
  dataArray[0] = 0x00;
  dataArray[1] = 0x00;
  dataArray[2] = 0x00;
  dataArray[3] = 0x00;
  if (num > 0 && num <= 8)
    dataArray[0] = B10000000 >> (num - 1);
  if (num > 8 && num <= 16)
    dataArray[1] = B10000000 >> (num - 1) - 8;
  if (num > 16 && num <= 24)
    dataArray[2] = B10000000 >> (num - 1) - 16;
  if (num > 24)
    dataArray[3] = B10000000 >> (num - 1) - 24;
  for (int i = 3; i >= 0; i--) {
    shiftOut(data, clk, LSBFIRST, dataArray[i]);
  }
  digitalWrite(latch, LOW);
  digitalWrite(latch, HIGH);
}
```

Fonte: Autor.

posição que corresponda a porta a ser aberta, para deslocar esse bit é empregado o operador de deslocamento binário a direita (linhas 8-19). Tendo configurado o buffer, é feito o envio dele por meio da função shiftOut que escreve nos registradores de armazenamento do 74HC595 (linha 21), por último realiza-se um pulso na latch para deslocar os bits e disponibilizá-los na saída, dessa maneira é polarizado um transistor e realizado o acionamento da porta selecionada.

Figura 41 – Implementação laço principal do firmware.

```
1  void loop() {
2      if (Serial.available() > 0){ //Se algo for recebido pela
           serial
3        Dado = Serial.read();
4        if(Dado == 'S'){
5          pinValues = ler_registradores();
6          Serial.println(pinValues, BIN);
7          oldPinValues = pinValues;
8          string = "";
9        }
10
11       if(Dado == '\n'){
12         Serial.println(string);
13         int n = string.toInt();
14         if(n<=32 and n>=0){
15           abrirPorta(string.toInt());
16           tempo = millis();
17
18         }
19         string = "";
20       }else{
21         string = string + (Dado-'0');
22       }
23
24     }
25     if(millis() - tempo > 600){
26       abrirPorta(0);
27       tempo = millis();
28     }
29 }
```

Fonte: Autor.

Na Figura 41 pode-se ver o laço principal do firmware, o comportamento é direcionado ao evento de recebimento de informação via serial (linha 2), se algo for recebido, é realizado a leitura desse byte (linha 3), caso o caractere recebido seja um 'S' (mnemônico para a palavra sinal) é feita a leitura dos registradores (linha 5) e enviado via serial os estados dos pinos, que nesse caso são exatamente os estados das portas (linha 6). Outro fluxo de execução do código é direcionado ao caso do Dado recebido for o caractere \n (linha 11), nessa parte é realizado o acionamento de

uma porta, por meio da função abrirPorta e do casting da string em um número inteiro (linhas 13-15).

3.3.5 Projeto Mecânico - SolidWorks

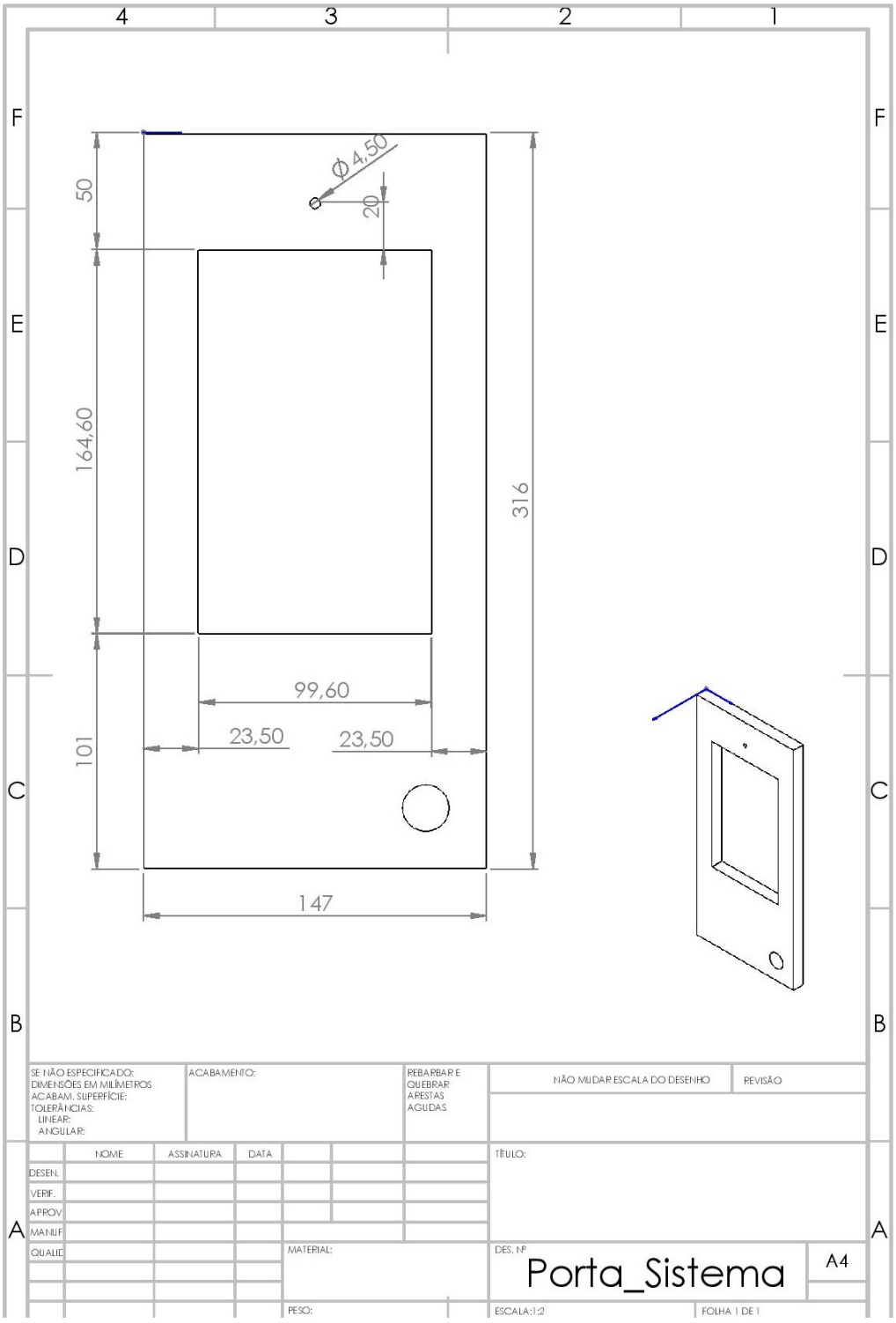
O software de CAD (Computer Aided Design) SolidWorks permite a criação de modelos tridimensionais de produtos, peças e mecanismos. Amplamente utilizado na engenharia pois permite projetar e analisar montagens de diferentes componentes mecânicos. No trabalho foi utilizado a versão 2017 do SolidWorks, para modelar todas as peças estruturais, na Figura 42 pode-se ver uma renderização da montagem do projeto mecânico do armário, pode-se ver que são 28 portas e há um compartimento central, que abriga a fonte de 5V, a Raspberry Pi, a tela touchscreen e a placa auxiliar.

Figura 42 – Renderização da montagem mecânica do armário.

Fonte: Autor.

As dimensões projetadas para o armário foram de 86.2 cm de largura 103.2 cm de altura e 25 cm de profundidade, cada porta tem um espaço útil de 12.6 cm de largura por 15.4 cm de altura e 17.6 cm de profundidade. Com o projeto mecânico em mãos e todos os desenhos técnicos das peças desenvolvidos, foi realizado a fabricação em uma marcenaria. Um exemplo de desenho de uma das peças é mostrado na Figura 43, que é em relação à porta central. Com o projeto do armário, foi alcançado o requisito funcional RF07 e a regra de negócio RN01.

Figura 43 – Desenho técnico da porta central.



Fonte: Autor.

3.4 CONSIDERAÇÕES PARCIAIS

Nesse capítulo foi apresentado as etapas realizadas para desenvolver o projeto do armário inteligente, tanto as etapas de software quanto as de hardware

foram direcionadas a atender os requisitos estabelecidos no início do projeto. Na Tabela 1 são apresentados os requisitos que foram e os que não foram alcançados durante o desenvolvimento do trabalho. A primeira coluna identifica o requisito, na segunda coluna é descrito como o requisito foi atendido, na terceira e quarta coluna é identificada a seção em que o requisito foi alcançado e informado se foi ou não atendido, entretanto todos os requisitos foram alcançados.

Tabela 1 – Análise do atendimento aos requisitos do projeto.

Requisito	**Descrição**	**Seção**	**Atendido**
RF01	Tela de escolha das portas	3.2.2	Sim
RF02	Aplicação de uma rede neural convolucional e algoritmo de distância euclidiana	3.2.1	Sim
RF03	Botão principal para captar fotos	3.2.2	Sim
RF04	Implementar interface serial com pySerial	3.2.3	Sim
RF05	Desenvolvimento de uma placa auxiliar, que apenas responde a mensagens seriais	3.3.4.1	Sim
RF06	Implementar uma condição de resposta para as respostas seriais	3.3.4.1	Sim
RF07	Projeto mecânico de um armário	3.3.5	Sim
RF08	Instalação de uma câmera na Raspberry Pi 3B	3.3.1	Sim
RN01	Tela para escolha da porta e projeto mecânico do armário	3.3.5 3.2.2	Sim
RN02	Controle por software para o uso do armário	3.2.1	Sim
RN03	Um botão principal apenas	3.2.2	Sim
RN04	Identificação gráfica dos estados das portas na tela	3.2.2	Sim
RN05	Controle via autenticação facial	3.2.1	Sim
RN06	Utilização de uma fechadura de 5V da empresa SARY	3.3.2	Sim
RN07	Utilizar uma Raspberry Pi 3B	3.3.1	Sim
RN08	Instalar o sistema operacional Raspbian Buster na Raspberry Pi 3B	3.3.1	Sim
RN09	Usar a biblioteca Kivy para a parte gráfica	3.2.2	Sim
RN10	Implementar a comunicação serial para o software LockerSys com pySerial	3.2.3	Sim
RNF01	Utilização de uma fechadura com recurso de sinal de estado	3.3.2	Sim
RNF02	Resposta é de <6s	4.2	Sim
RNF03	Utilização da Lingugem de programação Python	3.2.1	Sim
RNF04	Utilização de uma fonte de 5 Volts por 10 Amperes	3.1.1	Sim

Fonte: Autor.

4 AVALIAÇÃO E TESTES

Nesse capítulo é apresentado os dados e resultados obtidos durante o desenvolvimento do trabalho e dos testes. Para avaliar os acertos do modelo de rede neural, foi realizado uma análise com base em matrizes de confusão. Outro ponto importante é voltado a avaliação do tempo de processamento das principais partes do sistema, por último é avaliado os custos do sistema em relação a produtos comerciais. Nos testes foi utilizado o armário já montado como mostra a Figura 44.

Figura 44 – Armário completamente montado.

Fonte: Autor.

4.1 MATRIZ DE CONFUSÃO

As matrizes de confusão são utilizadas quando é desejado analisar métricas de desempenho de um modelo de classificação, nessa matriz é comparado um dado conhecido com um dado predito por um modelo. No Quadro 2 é apresentado um exemplo de matriz de confusão, pode-se ver que em $M_{22} = \text{TP}$ e $M_{33} = \text{TN}$ estão os resultados preditos corretos com relação às amostras, ou seja, verdadeiro positivo

(TP - True Positive) e verdadeiro negativo (TN - True Negative), o contrário ocorre em $M_{23} =$ FP e $M_{32} =$ FN estão os resultado incorretos com relação a predição, ou seja, falso positivo (FP - False Positive) e falso negativo (FN - False Negative).

Quadro 2 – Exemplo de uma matriz M de confusão.

Amostras n = 0	Predito: Sim	Predito: Não
Amostra: Sim	TP	FN
Amostra: Não	FP	TN

Fonte: Autor.

No intuito de avaliar as matrizes de confusão é utilizado a equação para acurácia (Equação 5) que trata de todos os erros, relacionando os erros com os acertos. Outra equação utilizada é a de precisão (Equação 6), aqui relaciona-se os dados de verdadeiro positivo com os falsos verdadeiros, dessa forma é obtida uma taxa para quantidade de acertos positivos.

· Acurácia:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

· Precisão:

$$P = \frac{TP}{TP + FP} \tag{6}$$

Foi realizado quatro classes de testes, para diferentes cenários de uso, nas três primeiras é verificado apenas se o sistema encontra um rosto, o primeiro teste é representado no Quadro 3, nesse teste foi considerado a entrada do rosto inclinado em 90 graus, ou seja, na horizontal, pode-se ver que para esse caso a acurácia é zero.

Quadro 3 – Matriz de confusão para detecção de rostos com ângulos de 90 ou -90 graus.

Amostras n = 10	Predito: Sim	Predito: Não
Amostra: Sim	0	10
Amostra: Não	0	0

Fonte: Autor.

O segundo teste (Quadro 4) foi realizado com entradas de rostos com inclinação de 45 graus, pode-se ver que a acurácia sobe para 0,5 e a precisão é igual a 0,5.

Quadro 4 – Matriz de confusão para detecção de rostos com ângulos de 45 ou -45 graus.

Amostras n = 10	Predito: Sim	Predito: Não
Amostra: Sim	5	5
Amostra: Não	0	0

Fonte: Autor.

No terceiro caso de teste foi utilizado entrada de rostos na vertical, pode-se ver que no Quadro 5 a predição do modelo é exata, ou seja, tanto a acurácia quanto

a precisão foram iguais a 1. Os três primeiros testes mostram que existe uma forte influência da inclinação do rosto em relação à identificação pelo sistema, sendo que os melhores resultados ocorrem com os rostos na vertical, fato que não é um problema, pois é o modo natural de utilização do sistema.

Quadro 5 – Matriz de confusão para detecção de rostos na vertical.

Amostras n = 30	Predito: Sim	Predito: Não
Amostra: Sim	15	0
Amostra: Não	0	15

Fonte: Autor.

Na última classe de teste foi considerada a tarefa de autenticação facial, ou seja, se o armário abre a porta apenas para o usuário correspondente, dessa forma verifica-se a segurança do sistema. No Quadro 6 é apresentado esse teste, pode-se ver que a acurácia e a precisão foram de 0,966 ou 96,6%, esse resultado vai ao encontro do valor de 99,38% para acurácia da rede neural *dlib_face_recognition_resnet_model_v1.dat.bz2* (KING, 2017).

Quadro 6 – Matriz de confusão, considerando a autenticação do rosto da mesma pessoa e fotos captadas na vertical.

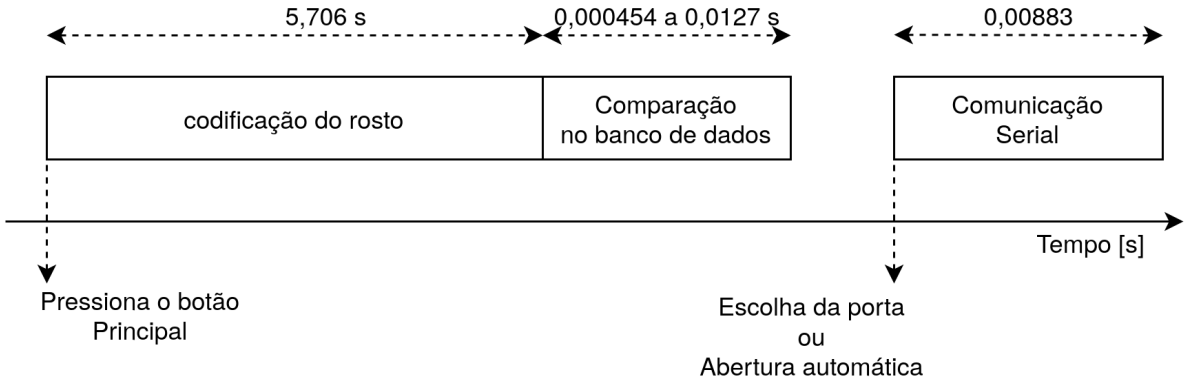
Amostras n = 60	Predito: Sim	Predito: Não
Amostra: Sim	29	1
Amostra: Não	1	29

Fonte: Autor.

4.2 ANÁLISE DO TEMPO DE PROCESSAMENTO

No intuito de entender quais componentes do sistema utilizam mais processamento, foi aplicada a biblioteca time de Python para extrair os tempos de execução de três principais fluxos do LockerSys, codificação do rosto, comparar no banco de dados e comunicação serial.

Figura 45 – Diagrama temporal do processamento no sistema.



Fonte: Autor.

Tabela 2 – Amostras do tempo de processamento em segundos [s] para as principais tarefas do sistema.

Amostra	**Codificação do Rosto [s]**	**Comparar no banco de dados [s]**	**Comunicação Serial [s]**
1	5,66	0,0005	0,012
2	5,72	0,0004	0,0085
3	5,73	0,00049	0,009
4	5,65	0,00046	0,01
5	5,72	0,0004	0,007
6	5,74	0,00045	0,0079
7	5,66	0,00047	0,01
8	5,74	0,0005	0,008
9	5,71	0,00044	0,0079
10	5,73	0,00043	0,008
média = $\overline{x}$	5,706	0,000454	0,00883
desvio padrão = σ	0,0335	0,000035	0,00139

Fonte: Autor.

Na Tabela 2 é apresentado dez amostras retiradas da execução de cinco utilizações do armário, ou seja, inserir um objeto e retirar. Utilizando a Tabela 2, mais especificamente o tempo médio $\overline{x}$, foi desenvolvido a Figura 45, que demonstra os tempos de maneira gráfica, pode-se ver que ao pressionar o botão principal é iniciado a codificação do rosto, processo que envolve a rede neural convolucional, por isso tem uma duração média de 5,7 segundos na Raspberry Pi 3B.

Seguindo na imagem é mostrado o tempo para realizar a comparação entre as codificações dos rostos armazenados, é importante notar que o menor tempo é de 0,45 milissegundos para um usuário e o maior tempo é de 12,7 milissegundos para vinte e oito usuários, por último está a comunicação serial que leva em torno de 8,83 milissegundos para executar.

4.3 CUSTOS DO SISTEMA

Com relação ao custo total do sistema foi construido a Tabela 3 que lista todos os materiais utilizados com relação ao preço unitário e subtotal, pode-se ver que os maiores valores foram da Raspberry Pi 3B, da estrutura mecânica do armário, das fechaduras e da tela touchscreen, que somados resultam no valor de R$ 2582,72, que é equivalente a 83,8% do valor total do armário.

Um ponto importante a ressaltar é sobre o valor da placa auxiliar, pois ela pode ser utilizada para outras funções que necessitem de até 32 sinais de feedback com estados binários, ou acionar até 32 dispositivos com até 4 Amperes por acionamento, o valor dessa placa resultou em R$ 98,9. Na Figura 54 e Figura 55 no Apêndice B é apresentado essa placa.

Tabela 3 – Relação de preço dos materiais utilizados no trabalho.

Componente	Quantidade	Preço Unitário (R$)	Sub Total (R$)
Fonte colmeia 5V/10A	1	70,29	70,29
Fechadura Sary	28	19,74	552,72
Tela Touchscreen	1	480	480
Conectores 12 vias MiniFit (Macho)	3	3,69	11,07
Conectores 12 vias MiniFit (Fêmea)	3	1,43	4,29
Cabo Vermelho 22 awg	35 m	1,12	39,2
Cabo Preto 22 awg	35 m	1,12	39,2
Cabo Amarelo	35 m	1,12	39,2
Resistor SMD 10k	56	0,06	3,36
Resistor SMD 560R	28	0,06	1,68
Conectores 6 vias MiniFit (Macho) 90 graus	1	1,97	1,97
Conectores 6 vias MiniFit (Fêmea) 90 graus	1	0,72	0,72
Capacitor smd	9	0,22	1,98
Espaçadores Para Cabos	30	0,96	29
Câmera Raspberry Pi	1	50	50
Conector engate Fechadura	28	3,05	85,4
Raspberry Pi 3B	1	900	900
Terminais para conector Mini Fit	100	0,22	22
Espaguete termo-retrátil 2mm2	4	1,14	4,56
Espaguete termo-retrátil 2mm2 Incolor	5	1,14	5,7
Conectores 8 vias MiniFit (Macho)	8	2,87	22,96
Conectores 8 vias MiniFit (Fêmea)	8	0,83	6,64
terminal olhal 1,5mm2 - M5	6	0,31	1,86
terminal olhal 1,5mm2 - M3	6	0,21	1,26
Estrutura em madeira	1	650	650
Placa auxiliar lisa	1	28,8	28,8
Arduio Nano	1	14	14
74HC595	4	0,47	1,88
74HC165	4	2	8
Mosfet AO3402	28	0,114	3,2
		Total	3081

Fonte: Autor.

4.3.1 Comparação de custos em relação a outros sistemas

Com relação à análise de custo em relação com outros sistemas semelhantes, foi considerado apenas a parte que envolve o software LockerSys, a Raspberry Pi 3B, a câmera e a tela touchscreen. Essa escolha de direcionar a comparação a essa parte do sistema, baseia-se no fato de que a maioria dos produtos relacionados a autenticação facial são orientados apenas ao controlador, e não a aplicação final, como exemplo, controle de armário, catraca ou portões.

Na Figura 46 é apresentado dois controladores de acesso comerciais, que trabalham com autenticação facial, o primeiro (Figura 46a) é fabricado pela empresa Intelbras, o segundo (Figura 46b) é fabricado pela empresa Hikvision. Os dois controladores apresentados foram utilizados para comparar atributos entre o sistema

desse trabalho.

Figura 46 – Controladores comerciais que trabalham com autenticação facial.

(a) Modelo: SS 3530 MF FACE W (Intelbras). (b) Modelo: DS-K1T343EWX (Hikvision).

Fonte: a) (INTELBRAS, 2021) b) (HIKVISION, 2022).

Com base nas folhas de dados dos dois produtos, foi criada a Tabela 4 que relaciona algumas das principais métricas para a análise desses sistemas, pode-se ver que o LockerSys (Software e Hardware) se diferencia dos outros equipamentos pelo fato de ter apenas uma câmera RGB, enquanto outros tem duas, uma RGB e outra infravermelha. Outro ponto importante é sobre o anti-fraude que o sistema LockerSys não possui em relação aos produtos comerciais, principalmente por conta da câmera infravermelho, uma característica é que todos os sistemas utilizam o Deep Learnig para captar características de rostos.

Tabela 4 – Comparação entre o equipamento do trabalho e produtos comerciais.

	LockerSys	**SS 3430 MF FACE W**	**DS-K1T343EWX**
Fabricante	Autor	Intelbras	Hikvision
Tempo de autenticação	<6s	<0,2s	<0,2s
Modelo para identificação de faces	Deep learnig	Deep learnig	Deep learnig
Precisão	>96,6%	>99,5%	>99%
Anti-fraude[1]	Não	Sim	Sim
Tamanho da tela	7"	4,3"	4,3"
Câmera	1x5MP	2x2MP(RGB + IR)[2]	2x2MP(RGB + IR)[2]
Compensação da iluminação[3]	Não	Sim	Sim
Compensação de luz infravermelha[4]	Não	Sim	Sim
Preço (R$)	1430	1700	1300

[1] Detecção facial utilizando a profundidade da face, dessa forma é negado o acesso, por meio de foto ou vídeo (necessita de uma câmera infravermelho).
[2] Duas câmeras de 2MP, uma para luz visível (RGB) e outra para luz infravermelha (IR).
[3] Leds brancos, que de maneira automática compensam a iluminação local.
[4] Leds infravermelhos, funcionam em conjunto com a câmera infravermelha.

Fonte: Autor.

5 CONCLUSÕES

O presente trabalho apresentou o desenvolvimento de um sistema de autenticação facial chamado LockerSys, e o utilizou como forma de facilitar o uso de um armário, para isso foram empregados ferramentas computacionais para o projeto mecânico e eletrônico, e o software foi implementado com funcionalidades disponíveis em bibliotecas para visão computacional e de desenvolvimento de interfaces gráficas, além disso, todo o processo de projeto e construção do sistema foi guiado por uma detalhada especificação de requisitos.

Os resultados de desempenho do trabalho vão ao encontro dos resultados do modelo de rede neural utilizado, apesar do LockerSys executando em uma Raspberry Pi 3B demorar mais no processo de codificação facial em relação aos produtos comerciais é destacado a precisão, que com algumas otimizações pode facilmente alcançar os níveis comerciais. As partes mecânicas, eletrônicas (Placa auxiliar) e fechaduras são facilmente desacopladas do controlador, fato que mostra a modularidade no sistema, e possibilidade e aprimoramentos.

Os objetivos foram alcançados, principalmente pois os requisitos funcionais e regras de negócio foram integralmente atendidas, sem dúvidas o objetivo geral de desenvolver um armário inteligente com autenticação facial foi alcançado, e os objetivos específicos foram discutidos ao longo do texto.

Durante o desenvolvimento do projeto, ficou claro a importância de estruturar uma especificação de requisitos do sistema, o seu uso deixa claro o que é preciso desenvolver para que o projeto alcance o objetivo desejado para os stakeholders (partes interessadas). A diagramação é outro ponto forte para a modelagem de um software, com ela é possível entender de forma clara as partes que devem integrar o sistema.

A implementação do LockerSys, foi possível por meio da utilização das bibliotecas Face Recognition, Dlib, OpenCV, Kivy e pySerial, cada uma desempenhou um papel importante, principalmente na parte da codificação das características dos rostos, no desenvolvimento da parte gráfica da tela touchscreen e comunicação serial com interfaces externas.

O projeto da placa auxiliar somou positivamente ao trabalho, foi possível trabalhar com prototipação rápida de PCI's utilizando o software Kicad e fabricantes internacionais, o projeto da eletrônica envolveu aplicações práticas de circuitos integrados, capacitores, resistores e microcontroladores. O projeto mecânico foi fundamental na estratégia para o chicote elétrico e disposição dos componentes, nesse sentido o Solidworks ofereceu um conjunto de ferramentas ideais para essa

finalidade.

A avaliação do projeto foi realizada principalmente, por meio da matriz de confusão que traz uma compreensão numérica dos resultados do modelo de rede neural utilizado, dentre os resultados destaca-se o fato da inclinação do rosto, pode-se afirmar que rostos com inclinações superiores a 45 graus não são captados pelo modelo. O LockerSys sendo executado em uma Raspberry Pi 3B mostrou características semelhantes em relação a produtos comerciais.

Como o trabalho atinge diferentes áreas do conhecimento, projetos futuros podem se aprofundar na implementação e treino da rede neural convolucional, outro ponto importante para se aprimorar está na câmera, em um próximo trabalho é indicado utilizar duas, uma com sensor para luz visível e outra para infravermelho, dessa forma é possível implementar identificação de profundidade e mecanismos anti-fraude. Em relação a eletrônica de potência pode-se pensar em um sistema de baterias que se comunique com o controlador do armário, por último, tendo em vista o tempo de computação, pode-se aplicar métodos para processamento em paralelo (threads) ou utilizar um hardware com mais recursos que a Raspberry Pi 3B.

REFERÊNCIAS

ALPHAEOMEGA. **AO340230V N-Channel MOSFET**. 2011. Disponível em: http://aosmd.com/pdfs/datasheet/AO3402.pdf. Acesso em: 21 jun. 2022.

ANUSHA, N.; SAI, A. D.; SRIKAR, B. Locker security system using facial recognition and One Time Password (OTP). In: **2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)**. Chennai: IEEE, 2017. p. 812–815.

ARDUINO. **What is arduino?** 2018. Disponível em: https://www.arduino.cc/en/Guide/Introduction. Acesso em: 16 jun. 2022.

ARDUINO. **Arduino IDE 1.8.19**. 2022. Disponível em: https://www.arduino.cc/en/software. Acesso em: 16 jun. 2022.

ARDUINO. **Nano**. 2022. Disponível em: https://docs.arduino.cc/hardware/nano. Acesso em: 16 jun. 2022.

BOYLESTAD, R. L.; NASHELSKY, L. **Dispositivos eletrônicos**: e teoria de circuitos. São Paulo: Pearson, 2013. ISBN 8564574217.

CORKE, P. **Robotics, vision and control**. Berlin: Springer, 2011. Disponível em: https://doi.org/10.1007/978-3-642-20144-8. Acesso em: 24 jan. 2022.

FLOYD, T. L. **Digital fundamentals**. England: Pearson Education, 2015.

FOUNDATION, R. P. **Raspberry Pi 3 Model B**. 2016. Disponível em: https://www.raspberrypi.com/products/raspberry-pi-3-model-b/. Acesso em: 16 jun. 2022.

FOUNDATION, R. P. **Raspberry Pi OS**. 2022. Disponível em: https://www.raspberrypi.com/software/. Acesso em: 16 jun. 2022.

GEITGEY, A. **Project description**. 2020. Disponível em: https://pypi.org/project/face-recognition/. Acesso em: 15 jul. 2022.

HE, K. et al. **Deep residual learning for image recognition**. arXiv, 2015. Disponível em: https://doi.org/10.48550/arXiv.1512.03385. Acesso em: 01 jun. 2022.

HIKVISION. **DS-K1T343EWX Face recognition terminal**. 2022. Disponível em: https://www.hikvision.com/content/dam/hikvision/products/S000000001/S000000080/S000000103/S000000107/OFR000141/M000048816/Data_Sheet/DS-K1T343EWX-Face-Recognition-Terminal_Datasheet_V1.0_20211101.pdf. Acesso em: 16 jul. 2022.

INSTRUMENTS, T. **SNx4HC165 8-bit parallel-load shift registers**. 2015. Disponível em: https://www.ti.com/lit/ds/symlink/sn74hc165.pdf?ts=1654442007110. Acesso em: 06 jun. 2022.

INSTRUMENTS, T. **CD74HC595 8-Bit Shift Registers With 3-State Output Registers**. 2022. Disponível em: https://www.ti.com/lit/ds/symlink/cd74hc595.pdf. Acesso em: 21 jun. 2022.

INTELBRAS. **SS 3530 MF FACE W**. 2021. Disponível em: https://backend.intelbras.com/sites/default/files/2021-09/DatasheetSS3530FACEW_V2.pdf. Acesso em: 16 jul. 2022.

JAIN, A. K.; ROSS, A. A.; NANDAKUMAR, K. **Introduction to biometrics**. Boston: Springer US, 2011. Disponível em: https://doi.org/10.1007/978-0-387-77326-1. Acesso em: 18 fev. 2022.

KAKANI, V. et al. A critical review on computer vision and artificial intelligence in food industry. **Journal of Agriculture and Food Research**, v. 2, p. 100033, dez. 2020.

KASSEM, A. et al. A smart lock system using Wi-Fi security. In: **2016 3rd International Conference on Advances in Computational Tools for Engineering Applications (ACTEA)**. Zouk Mosbeh: IEEE, 2016. p. 222–225.

KICAD. **About KiCad**. 2021. Disponível em: https://www.kicad.org/about/kicad/. Acesso em: 17 jun. 2022.

KING, D. **High Quality Face Recognition with Deep Metric Learning**. 2017. Disponível em: http://blog.dlib.net/2017/02/high-quality-face-recognition-with-deep.html. Acesso em: 21 jun. 2022.

KIVY. **Kivy - Open source Python library for rapid development of applications**. 2021. Disponível em: https://kivy.org/#home. Acesso em: 20 jun. 2022.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: PEREIRA, F. et al. (Ed.). **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2012. v. 25. Disponível em: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

LEAVERS, V. F. **Shape detection in computer vision using the hough transform**. London: Springer, 1992. Disponível em: https://doi.org/10.1007/978-1-4471-1940-1. Acesso em: 12 fev. 2022.

MLADENOVA, T.; VALOVA, I.; VALOV, N. Application of facial recognition with PCA and raspberry pi for access control to luggage lockers. In: **2021 International Conference Automatics and Informatics (ICAI)**. Varna: IEEE, 2021. p. 141–145.

OPENCV. **About**. 2022. Disponível em: https://opencv.org/about/. Acesso em: 20 jun. 2022.

PYSERIAL. **pySerial overview**. 2020. Disponível em: https://pyserial.readthedocs.io/en/latest/pyserial.html. Acesso em: 19 jun. 2022.

RUSSEL, S.; NORVIG, P. **Inteligência Artificial**. New Jersey: Pearson, 2013. ISBN 978-85-352-3701-6.

SARY. **Lockers**. 2022. Disponível em: https://sary.en.alibaba.com/. Acesso em: 17 jun. 2022.

SZELISKI, R. **Computer vision**: algorithms and applications. London: Springer, 2011. Disponível em: https://doi.org/10.1007/978-1-84882-935-0. Acesso em: 5 fev. 2022.

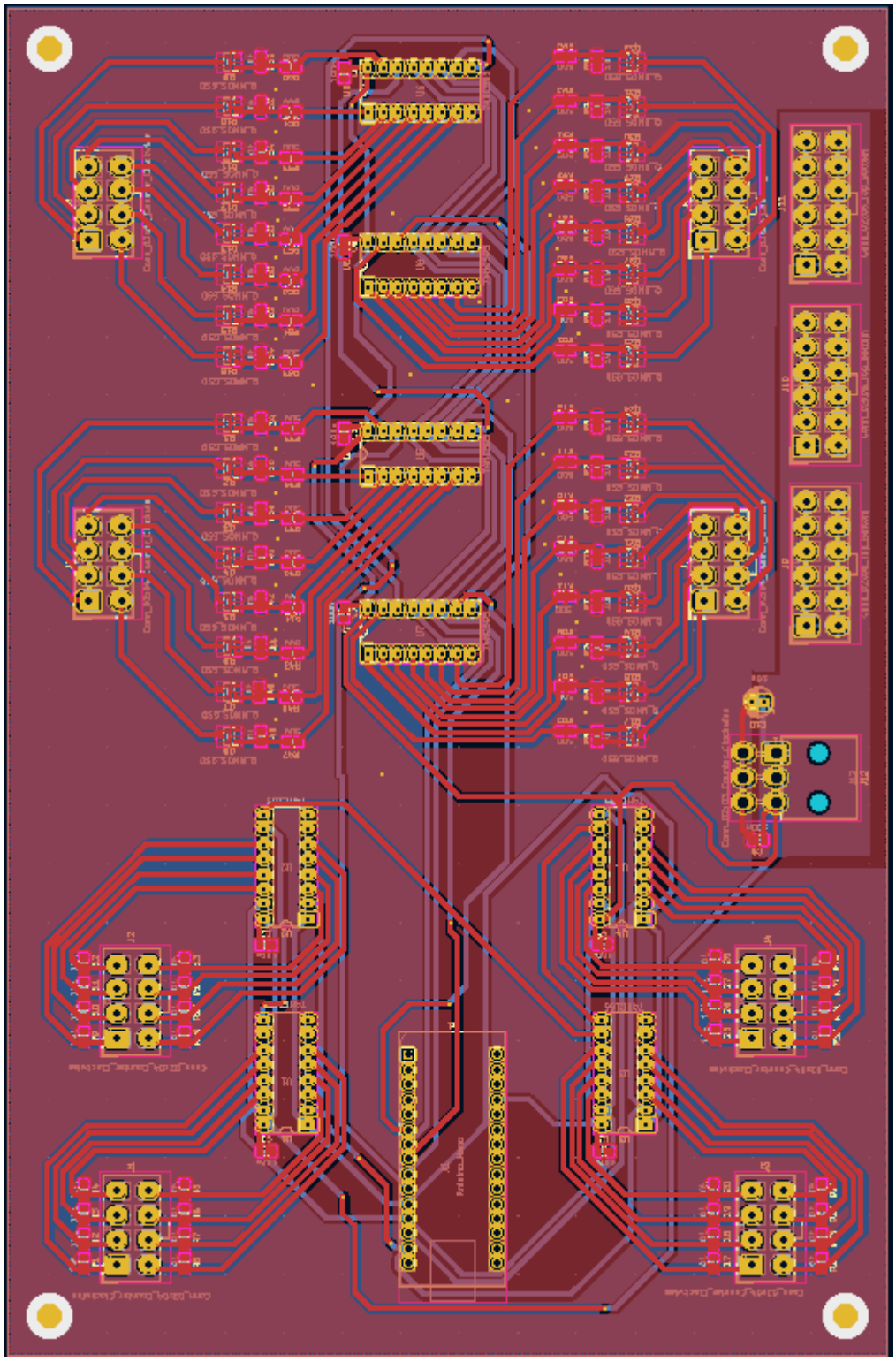
TAIGMAN, Y. et al. DeepFace: Closing the gap to human-level performance in face verification. In: **2014 IEEE Conference on Computer Vision and Pattern Recognition**. Columbus: IEEE, 2014. p. 1701–1708.

ZEILER, M. D.; FERGUS, R. **Visualizing and understanding convolutional networks**. arXiv, 2013. Disponível em: https://doi.org/10.48550/arXiv.1311.2901.

ZHANG, L.; LIN, W. **Selective visual attention**: computational models and applications. Singapore: IEEE COMPUTER SOC PR, 2013. Disponível em: https://ieeexplore.ieee.org/book/6497236. Acesso em: 2 fev. 2022.

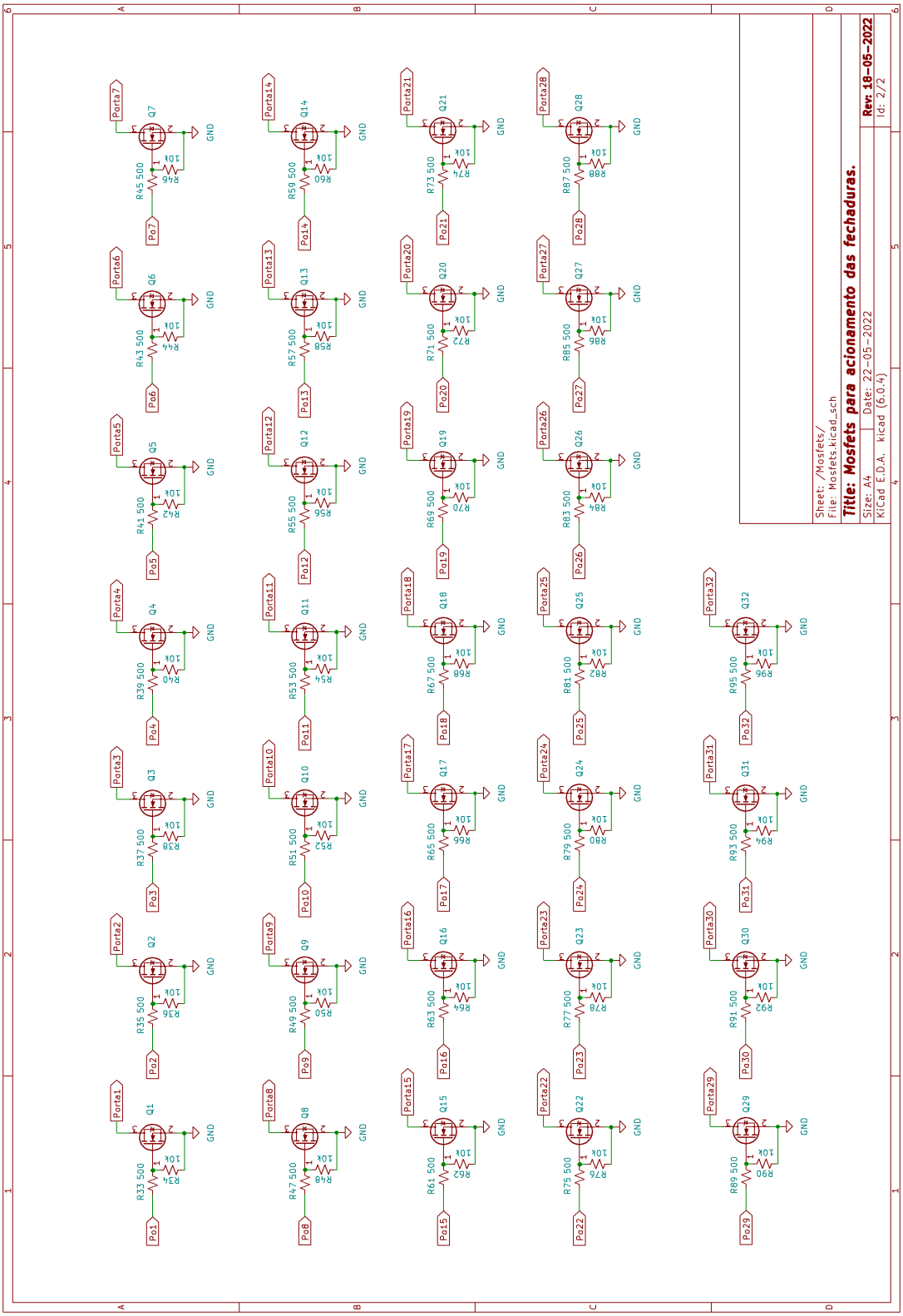
APÊNDICE A - PLACA AUXILIAR

Figura 47 – Trilhas da PCI.



Fonte: Autor.

Figura 48 – Esquemático mostrando o arranjo de mosfets, para acionamento das portas.

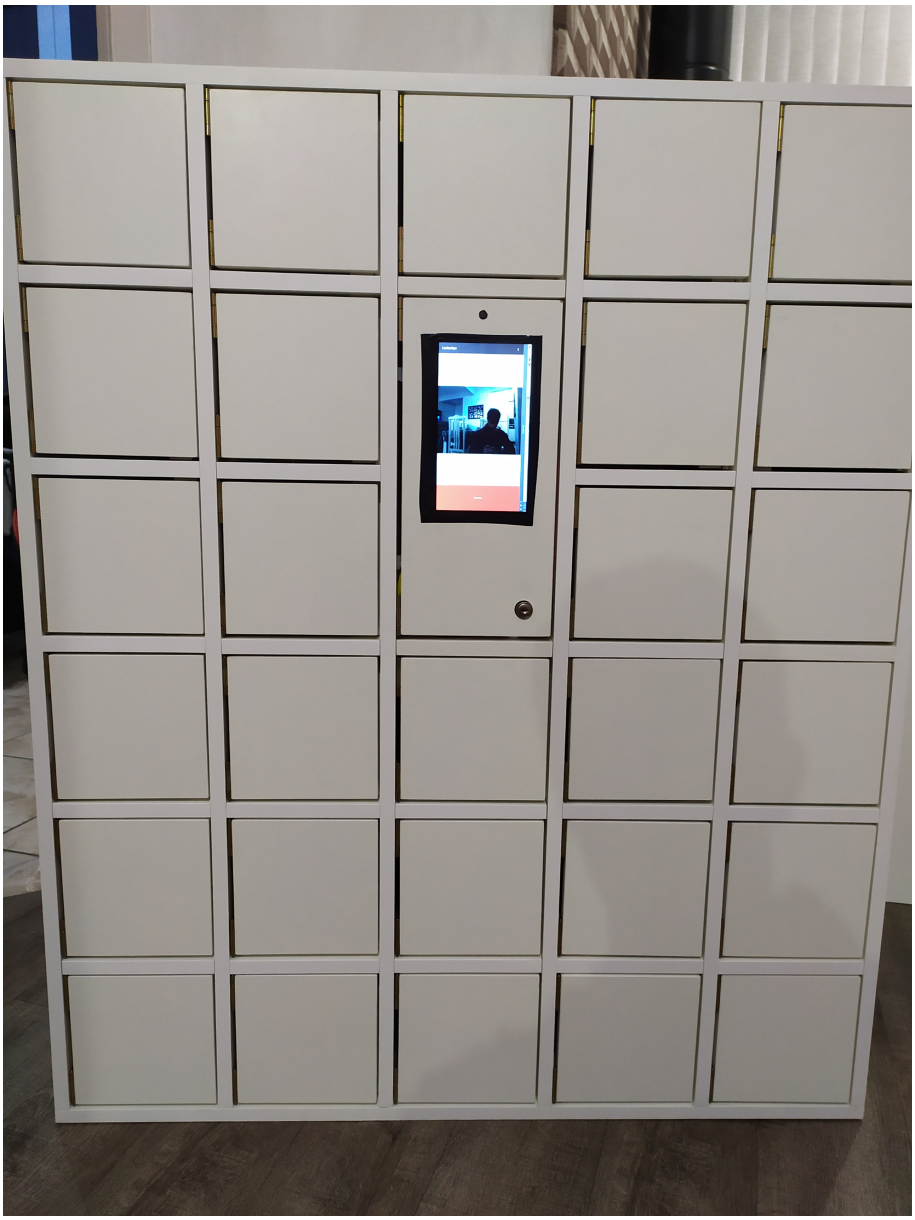


Fonte: Autor.

APÊNDICE B - FOTOS DO ARMÁRIO

Nesse apêndice é apresentado fotos do armário, para que o leitor tenha uma maior compreensão do sistema.

Figura 49 – Vista frontal do armário.

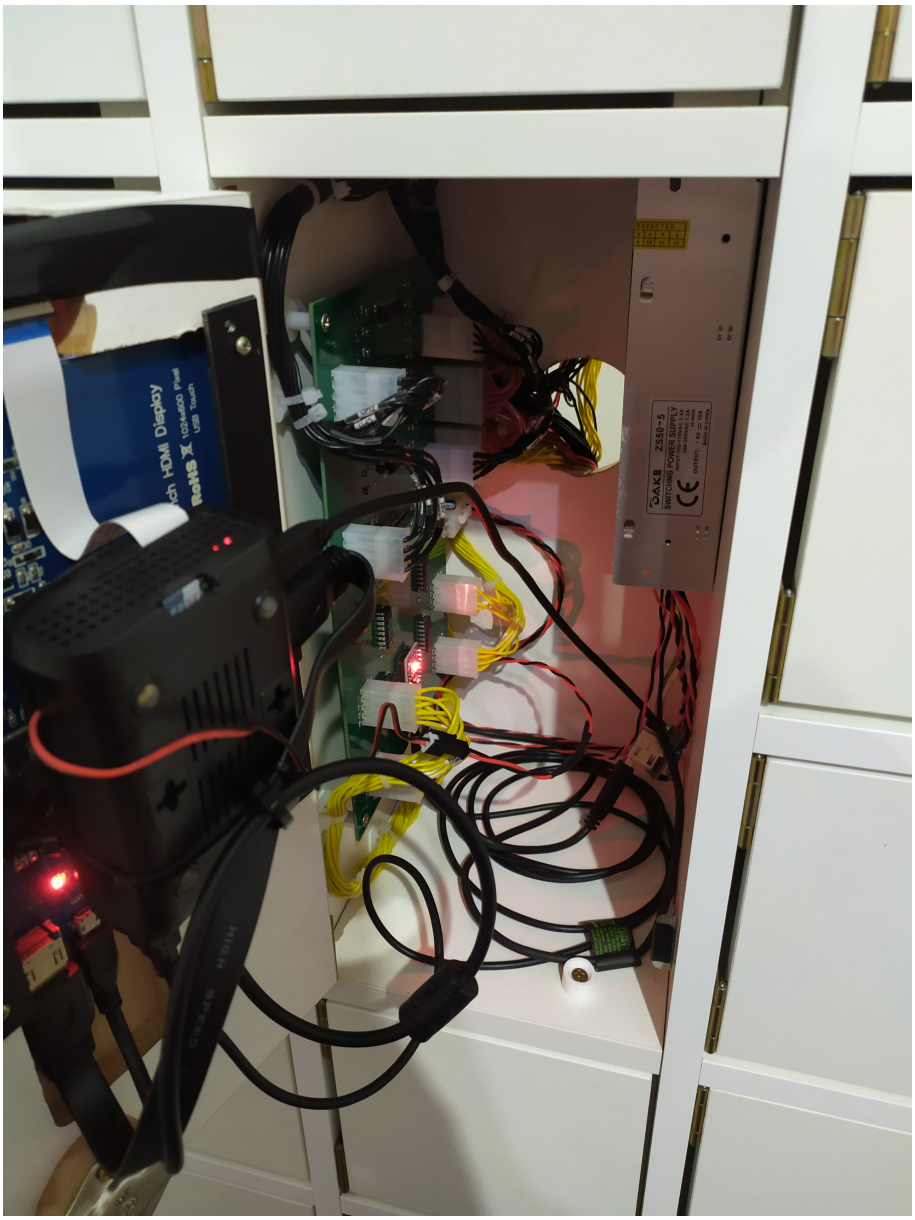


Fonte: Autor.

Figura 50 – Armário com a tampa de trás aberta, é visível o chicote que faz a ligação das fechaduras com a placa auxiliar.

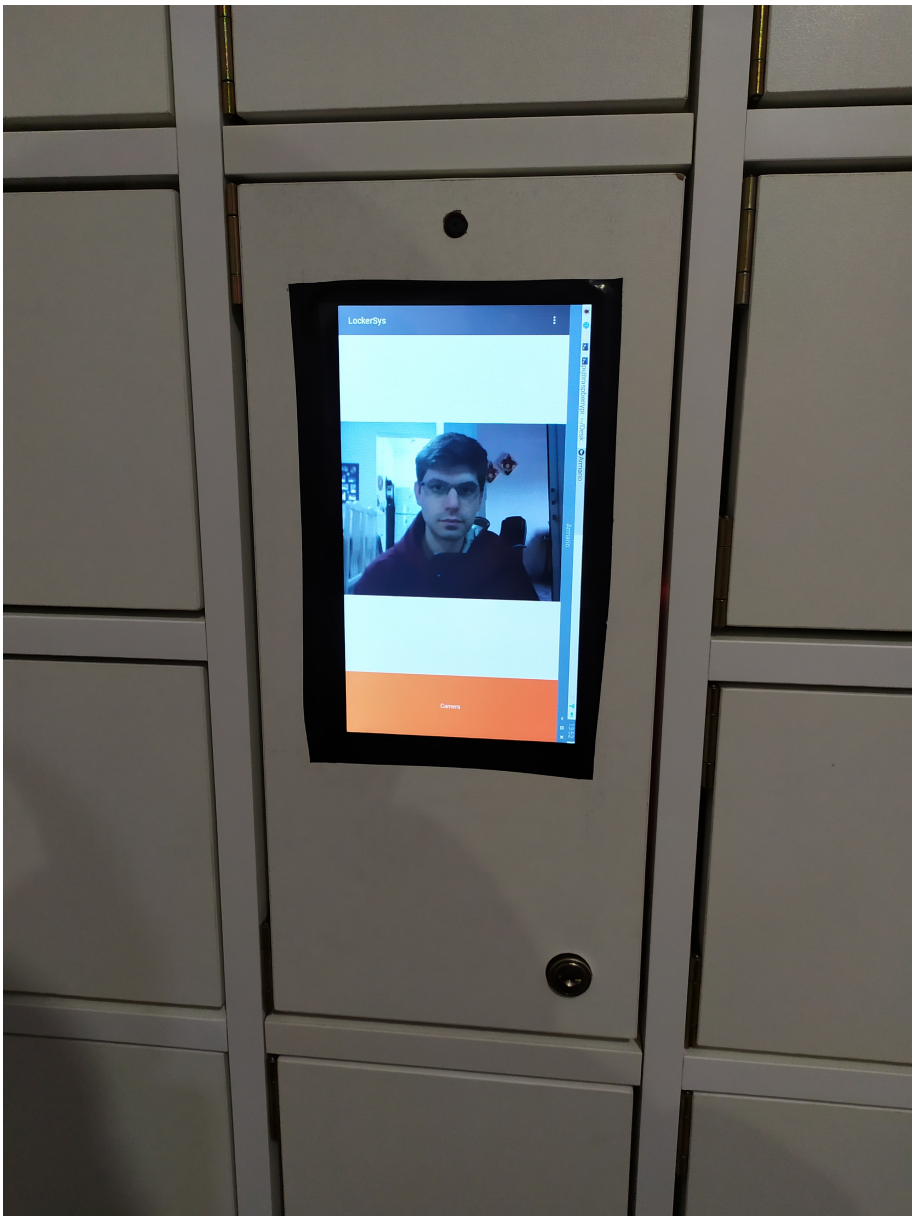
Fonte: Autor.

Figura 51 – Visão interna dos seguinte componentes da direita para esquerda: fonte 5V, placa auxiliar, Raspberry Pi 3B e tela touchscreen.



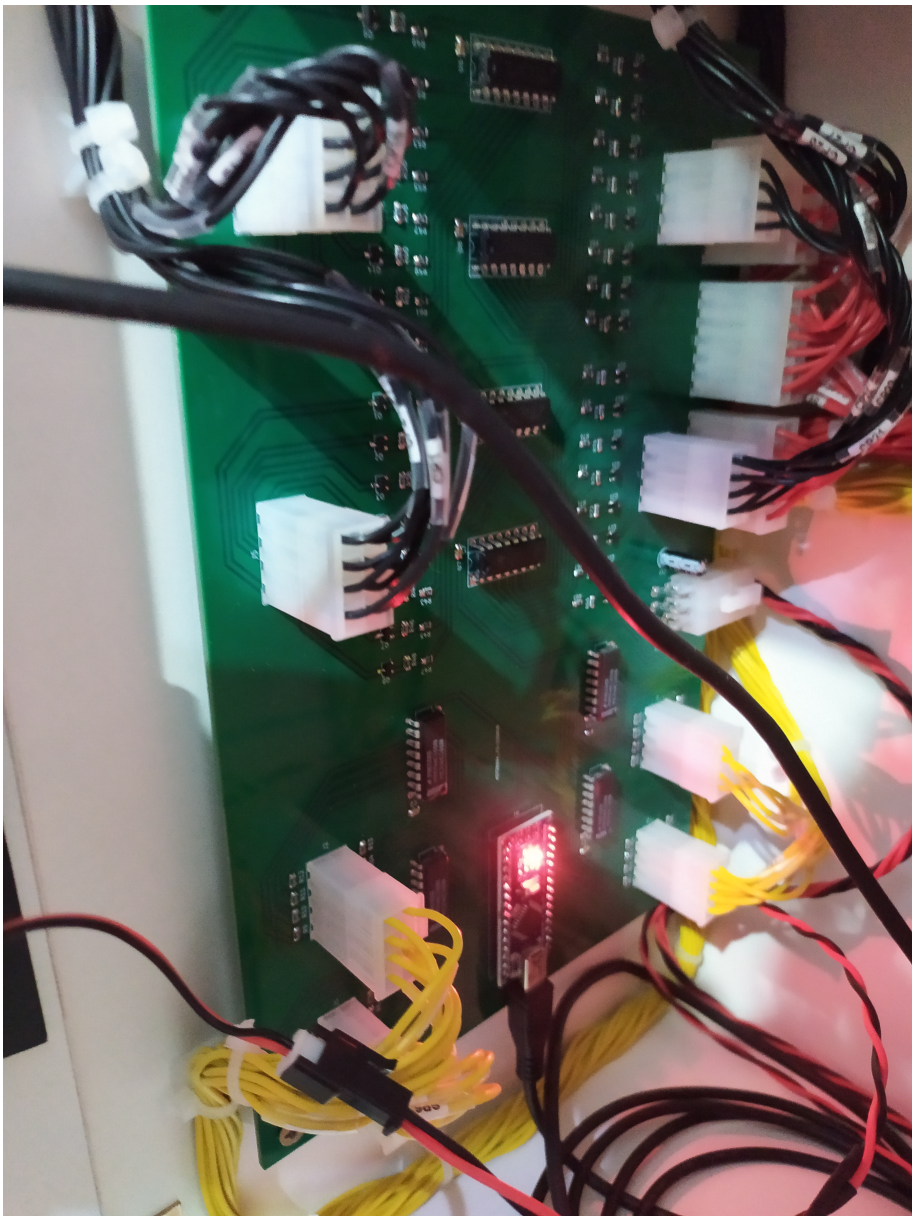
Fonte: Autor.

Figura 52 – Visão frontal do armário.



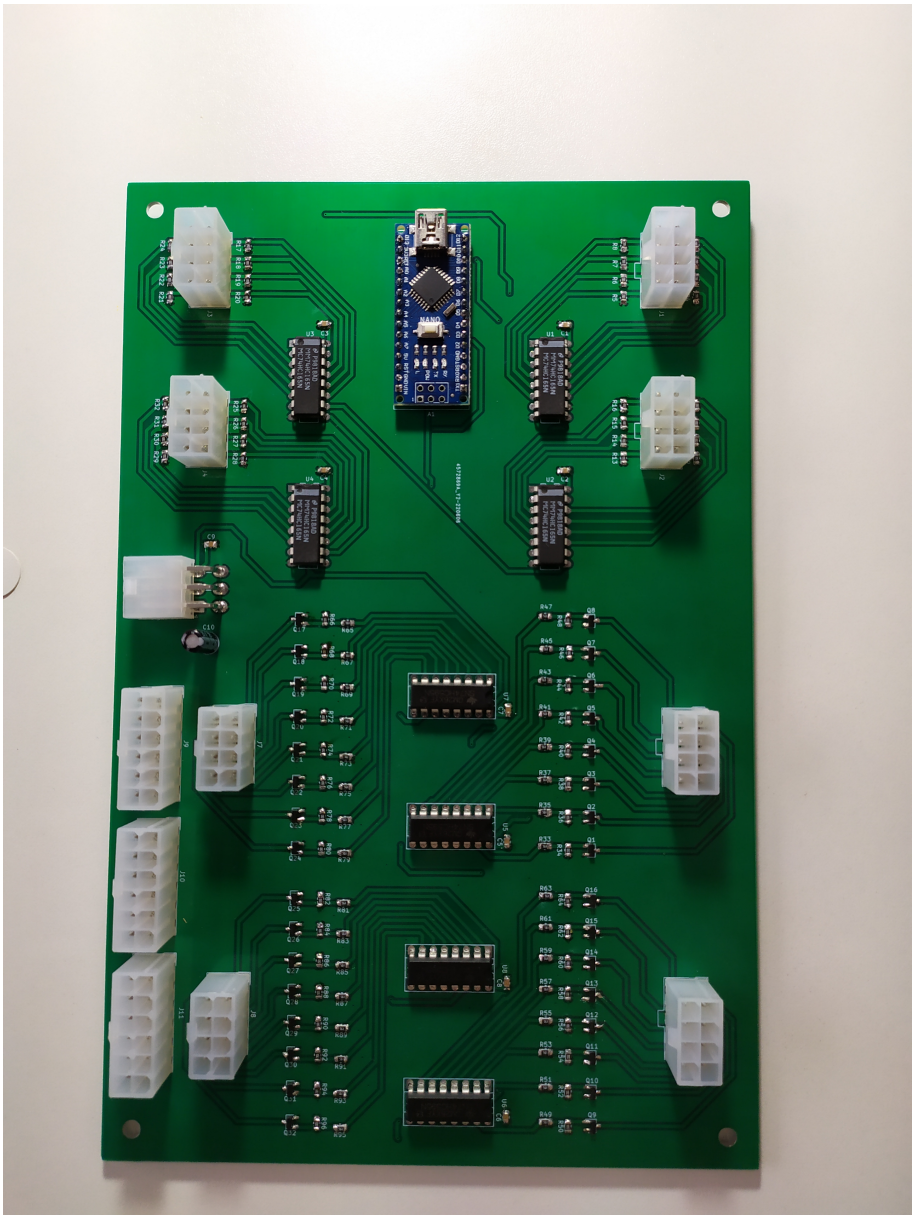
Fonte: Autor.

Figura 53 – Placa auxiliar instalada no armário.



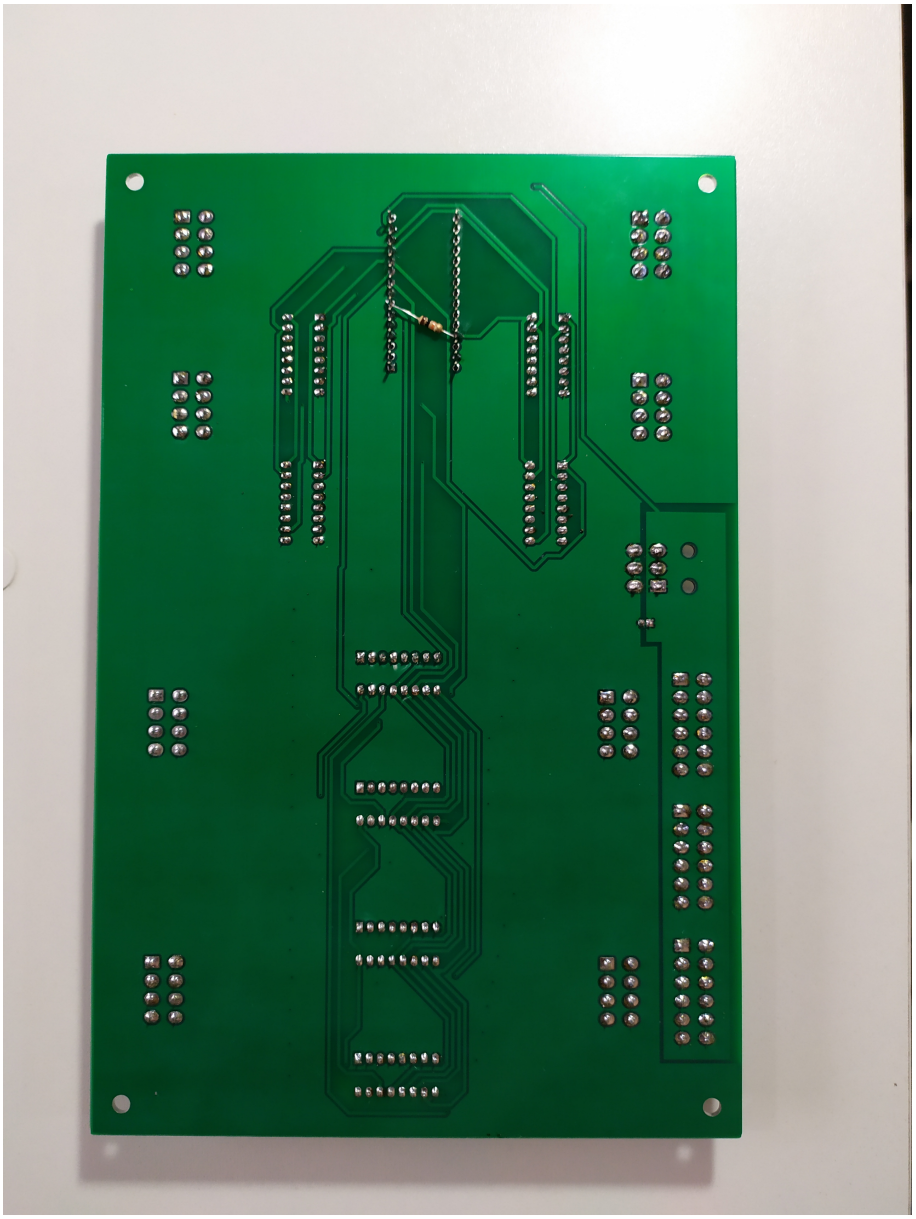
Fonte: Autor.

Figura 54 – Placa auxiliar lado de cima.



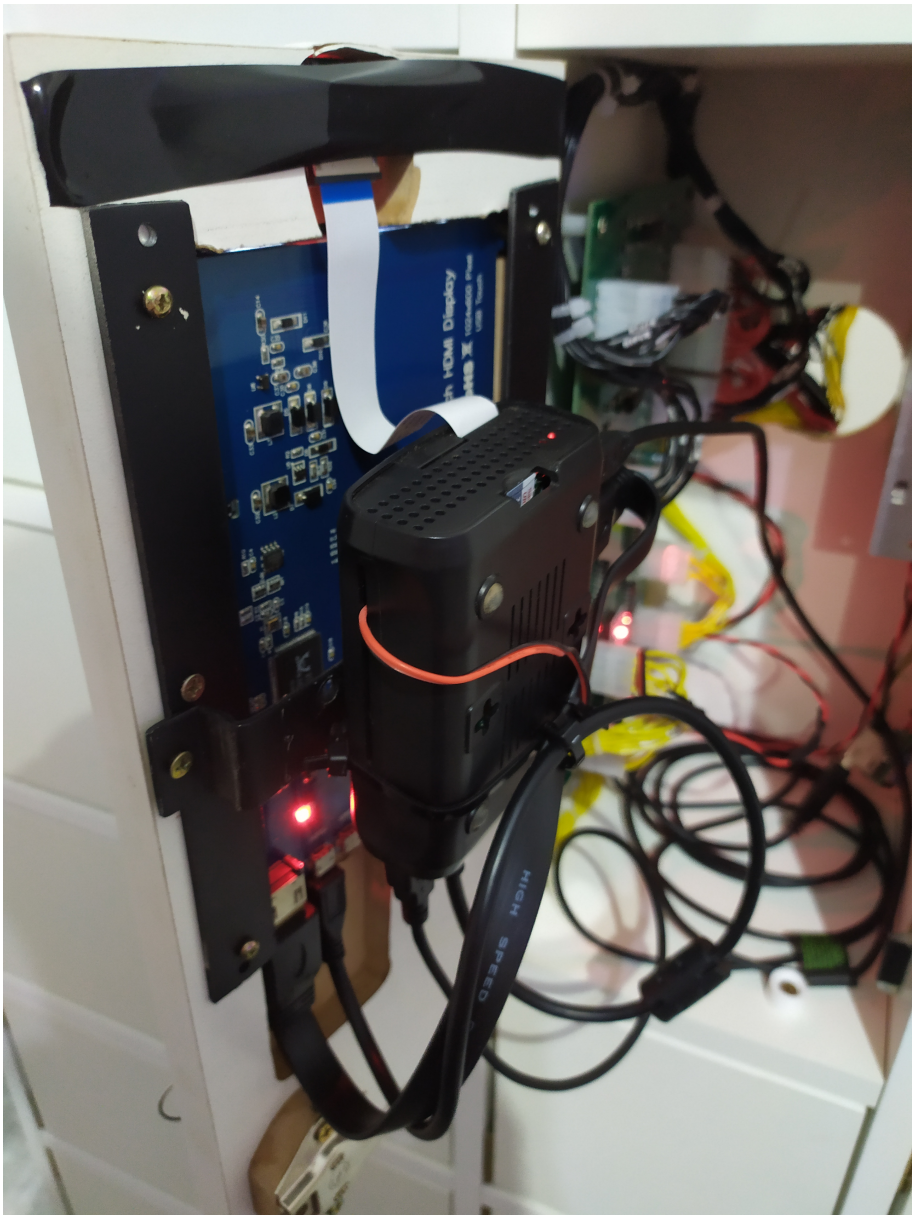
Fonte: Autor.

Figura 55 – Placa auxiliar lado de baixo.



Fonte: Autor.

Figura 56 – Parte interna, mostrando a tela touchscreen e a Raspberry Pi 3B.



Fonte: Autor.