Relatório Final

Sistema de Segurança Aprimorado para Cofres Pessoais

Felipe Chermont Pereira
Faculdade do Gama (FGA)
Universidade de Brasília (UnB)
Brasília, Brasil
chernox0427@gmail.com

Guilherme Simões Dias
Faculdade do Gama (FGA)
Universidade de Brasília (UnB)
Brasília, Brasil
g.simoesdias@gmail.com

***Abstract*— This project is aimed at demonstrating an improved security system for personal safes, which in addition to protecting the property of the owner also allows remote monitoring of the safe. For this, a two-stage locking system was developed, with facial recognition and a numeric keypad with a password using a Raspberry Pi to perform the necessary processing. In this system, when an access occurs, a photo of the subject's face is captured and compared with those present in the database and, if necessary, a message will be sent to the main person responsible for the safe via telegram, informing that someone unknown is trying to access it. it.**

***Resumo*— Este projeto é voltado para a demonstração de um sistema aprimorado de segurança para cofres pessoais, que além de proteger os bens do proprietário também permite o monitoramento remoto do cofre. Para isso, foi desenvolvido um sistema de tranca de duas etapas, com reconhecimento facial e teclado numérico com senha, utilizando uma Raspberry Pi para realizar os processamentos necessários. Neste sistema, quando ocorre um acesso, uma foto do rosto do sujeito é capturada e comparada com as presentes no banco de dados e caso necessário, será enviado uma mensagem para o principal responsável pelo cofre via telegrama, informando que alguém desconhecido está tentando acessá-lo.**

Keywords—Segurança; Reconhecimento Facial; Raspberry PI;

I. INTRODUÇÃO

O projeto visa aprimorar a segurança dos cofres pessoais convencionais ao implementar um sistema de reconhecimento facial com uso de técnicas de "*Machine Learning*" para identificar o usuário antes de garantir o acesso. Ademais, outra medida de segurança adotada é o alerta emitido caso uma tentativa fracassada ocorra, em que um bot do Telegram que está integrado ao sistema realiza capturas de imagens da câmera e envia diretamente para o celular do proprietário do cofre, permitindo que este tome as ações necessárias. Em conjunto com a verificação facial, o sistema conta com uma segunda etapa de segurança realizada com uma senha de dígitos numéricos.

II. JUSTIFICATIVA

Dentro de residências ou estabelecimentos existem diversos objetos que precisam ser protegidos pelo seu valor ou que precisam ser mantidos fora do alcance de determinadas pessoas que frequentam o local por questões de segurança.
Em geral, os cofres pessoais convencionais cumprem com o seu papel de restringir o acesso ao seu conteúdo, no entanto estes não fazem distinção de quem está acessando, pois geralmente requerem apenas uma senha numérica. Fato que gera uma brecha no caso de algum indivíduo descobrir a senha.
Por este motivo, uma segunda etapa de verificação com um fator mais restrito e individual pode ser implementada para limitar o acesso.
Dentre os mecanismos de identificação que se adequam a esse requisito, temos biometria, reconhecimento de íris e fácil. No entanto, o reconhecimento de íris é um processo extremamente avançado e que requer sensores específicos que não podem ser obtidos para este projeto por limitações de verba.
Contudo, entre a autenticação por biometria e a identificação facial, optamos pela segunda opção pois ela nos dá a possibilidade de implementar um sistema de alerta mais completo e com informações mais relevantes para o proprietário, como uma foto do indivíduo que está tentando abrir o cofre.

III. OBJETIVOS

Desenvolver um protótipo de um cofre com um sistema de segurança aprimorado capaz de restringir o acesso ao seu conteúdo por meio de um sistema de reconhecimento facial auxiliado por aprendizado de máquina e proporcionar uma maior segurança e confiança ao proprietário.
Além de ser capaz de gerar alertas com informações relevantes sobre quem está acessando o cofre em tempo real via mensagem pelo aplicativo Telegram diretamente para o celular do proprietário.

IV. METODOLOGIA

O projeto foi dividido em pontos de controle, com o objetivo de analisar e marcar o desenvolvimento do mesmo. Cada ponto de controle possui objetivos e metas a serem alcançados para garantir o andamento do projeto.

- **PC1**: Proposta do projeto (justificativa, objetivos, requisitos, benefícios, revisão bibliográfica).

- **PC2**: Realização de testes nos componentes de forma individual, utilizando as ferramentas mais básicas da placa de desenvolvimento além de bibliotecas prontas para demonstrar a viabilidade do projeto e a capacidade do grupo de trabalhar com o hardware presente.

- **PC3**: Refinamento do protótipo, adicionando recursos básicos de sistema (múltiplos processos *ethreads*, pipes, sinais, semáforos, MUTEX etc.) e iniciando a integração entre os subsistemas.

- **PC4**: Refinamento do protótipo, montagem final do projeto físico e realização de adequações.

A cada ponto de controle foi realizado um relatório contendo o avanço do projeto conforme o cronograma para registrar a evolução do mesmo.
Além disso, na agenda dos membros do projeto foram dedicadas um mínimo de três horas semanais para o desenvolvimento. O projeto terá parte de desenvolvimento em *Software* e parte em *Hardware*.

1) Software: A funcionalidade deste depende da capacidade de executar as seguintes ações:
- Com um módulo de câmera capaz de identificar rostos de indivíduos e realizar o reconhecimento em tempo hábil, usando o banco de dados de referência, com uma acurácia de no mínimo 80%.
- Realizar comunicação com smartphones via internet para enviar alertas na forma de mensagens de texto e imagens registradas pelo módulo de câmera com o menor intervalo possível.
- Reconhecer a sequência numérica registrada como senha para desbloquear o cofre, além de realizar comandos básicos como limpeza de caracteres e realizar nova tentativa.

2) Hardware: Este deve atender a demanda do software, além de ser construído de forma a ter uma interface amigável e intuitiva. Também deve ser integrado a estrutura garantindo que a essência do produto não seja comprometida.

V. REQUISITOS

A. Hardware

- 1 processos ethreads pi 3B;
- 1 Módulo de câmera;
- 1 Teclado numérico matricial;
- 1 Protoboard;
- 1 Display LCD 16x2;
- 1 SD card 16GB;
- Diversos jumpers;
- 1 Cabo de alimentação tipo micro USB;
- 1 Transformador 12V 1.5ª
- Módulo relé 3.3v
- Tranca solenoide 12v

B. Software

- Integração raspberry com bots de Telegram;
- Sistema de reconhecimento facial;
- Identificação de entradas no *keypad*;
- Sistema de chave e cadeado;
- Comunicação com display.

VI. BENEFÍCIOS

O projeto demonstrará uma opção mais confiável e segura para guardar bens pessoais sem correr o risco de serem tomados por pessoas indesejadas ou indevidas. Garantindo que o proprietário do cofre tenha informações de qualidade sobre a situação dos seus bens a todo momento pelo seu smartphone.

VII. REVISÃO BIBLIOGRÁFICA

A ideia do projeto surgiu através de pesquisas acerca da utilização da raspberry pi para sistemas de segurança e de seu potencial envolvendo IoT (*internet of things*). Foi então que encontramos o projeto[1] "Sistema de monitoramento com Raspberry Pi" publicado pelo Paulo Lucas, onde foi utilizado uma Raspberry Pi, uma câmera e um sensor de movimento PIR para identificar movimentação anormal em determinado local e aciona a câmera para tirar uma foto do local.
Outra inspiração para o projeto foi o TCC[2] do Alef Kaian, onde ele descreveu como técnicas de Machine Learning e reconhecimento facial são e tendem a ser cada vez mais utilizadas como biometria para diversas aplicações.
Os projetos acima mostram como é possível utilizar técnicas de reconhecimento facial para biometria e como é possível utilizar das imagens fornecidas pela câmera associada a Raspberry Pi para desenvolver um sistema seguro para o usuário, demonstrando a viabilidade do projeto proposto.

VIII. DESENVOLVIMENTO

A. Componentes principais

1) Raspberry Pi 3B+: Modelo de 2018 da Raspberry Pi Foundantion com conectividade com Ethernet.

- Processador Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- 1GB LPDDR2 SDRAM
- 2.4GHz e 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
- Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)
- 40-pin GPIO header
- Conector de interface HDMI
- 4 portas USB 2.0
- Conector CSI para câmera Raspberry Pi
- Conector DSI para display capacitivo Raspberry Pi
- Saída de vídeo composto e áudio através do plug P4
- Slot cartão micro SD para carregamento do sistema operacional e armazenamento de arquivos
- Entrada de fonte DC micro USB 5V/2.5A
- Suporte a Power-over-Ethernet (PoE) (requer HAT PõE separadamente)

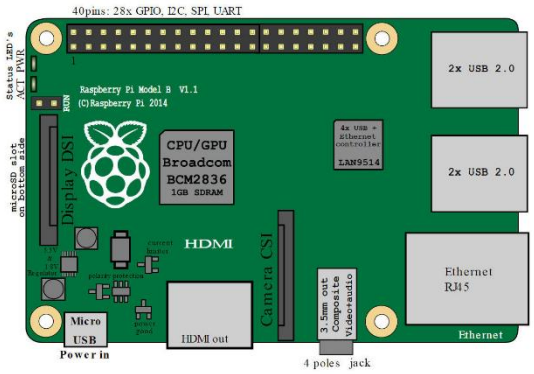


Figura 01 – Esquemático Raspberry Pi

2) Módulo de câmera: GOtech Webcam OFFICE HD 720p.

- Resolução 720P
- Ângulo da câmera 80°
- Tipo de interface USB 2.0
- Sensor 1/7" CMOS
- Vídeo chamada HD de 720p
- Correção automática de pouca luz
- Possui ajuste de foco automático
- Suporte aos sistemas operacionais Windows, Mac OS e Linux

Figura 02 – GOtech Webcam OFFICE

3) Teclado matricial

- Quantidade de Teclas: 16;
- Conector: 8 pinos (2,54mm);
- Limites de Operação: 35VDC, 100mA;
- Isolação: 100MΩ, 100V;
- Tempo de contato: <=5ms;
- Tamanho: 69,2 x 76,9 x 0,8mm;

Figura 03 – Teclado matricial 4x4.

4) Display Cristal Líquido (LCD 16X02 - AZ/BR) com Módulo I2C.

- Interface: Serial I2C;
- Tensão de trabalho: 2,5V a 6V;
- Pinos de conexão: SDA, SCL, GND e VCC;
- Displays suportados: LCD 16x02 e LCD 20x04;
- Corrente em standby: 10uA;
- Tipo de controlador: Expansor de portas I/O de 8 bits I2C;
- Endereços: 0x20 a 0x27 (Padrão 0x27), pode ser alterado.

Figura 04 – *Datasheet* Display LCD 16x2.

5) Mini Fechadura Solenoide Trava Elétrica 12v.

- Tensão de Operação: 12V
- Corrente de Operação: 0.42A
- Força: 0.07KG
- Lingueta: 7mm
- Tempo máximo de acionamento: 10s
- Tempo de abertura: 1s

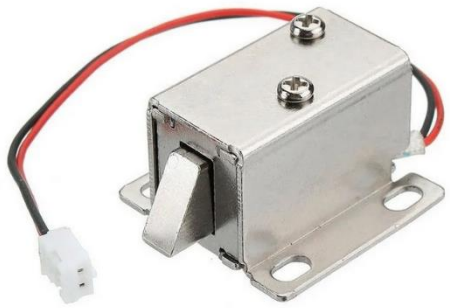


Figura 05 – Mini fechadura solenoide trava elétrica

B. Mapeamento de pinos e conexões

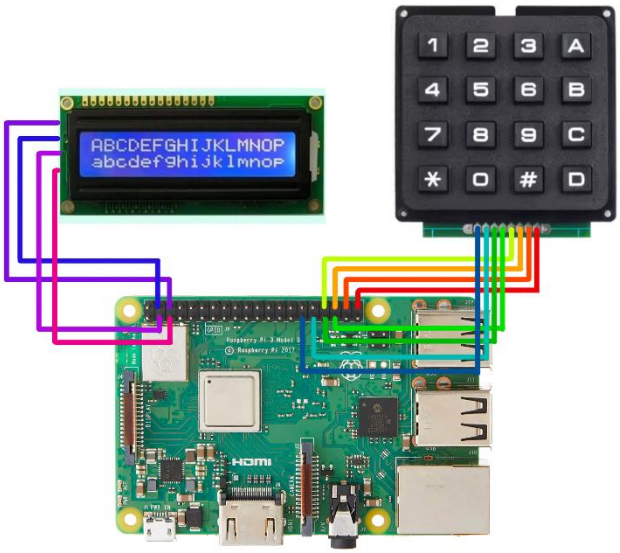


Figura 06: Conexão do Subsistema Senha numérica.

Teclado matricial: Contando os pinos da esquerda para a direita, foram realizadas a seguintes conexões.

PIN 1	GPIO 5
PIN 2	GPIO 6
PIN 3	GPIO 13
PIN 4	GPIO 19
PIN 5	GPIO 12
PIN 6	GPIO 16
PIN 7	GPIO 20
PIN 8	GPIO 21

Display LCD:

GND	5V Power
VCC	Ground
SDA	GPIO 2
SDL	GPIO 3

Módulo de câmera: Para este componente foi usada a entrada USB da *Raspberry*.

Módulo relé 3.3v: Devido à falta deste componente no mercado, foi realizada uma adaptação usando um relé de 5v.

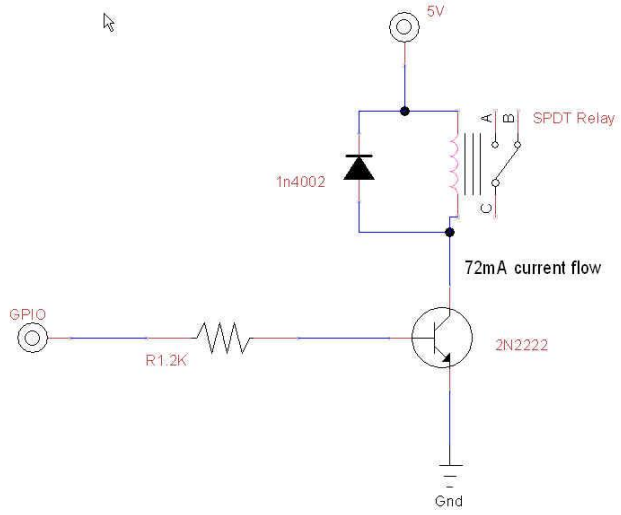


Figura 07: Diagrama de conexão módulo relé.

Como pode ser visto a partir do esquemático acima, foi utilizado um resistor entre a entrada GPIO e a base do transistor npn para limitar a corrente, um diodo conectado no coletor realizando uma ponte entre as bobinas do módulo relé para prevenir que a tensão reversa danifique o transistor quando o relé mudar o modo de operação e o terminal emissor foi conectado ao terra.

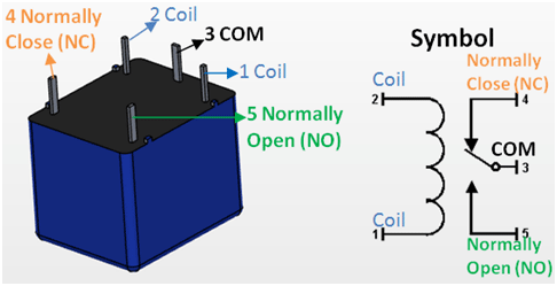


Figura 08: Diagrama de pinos do relé.

Tranca elétrica solenoide: Para realizar a alimentação deste componente, foi usada uma fonte de 12v. O terminal positivo da fonte foi conectado à entrada NO do relé, o neutro ao negativo da tranca e o COM foi conectado ao positivo.

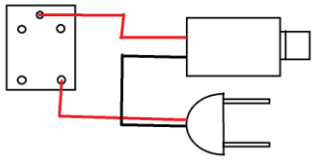
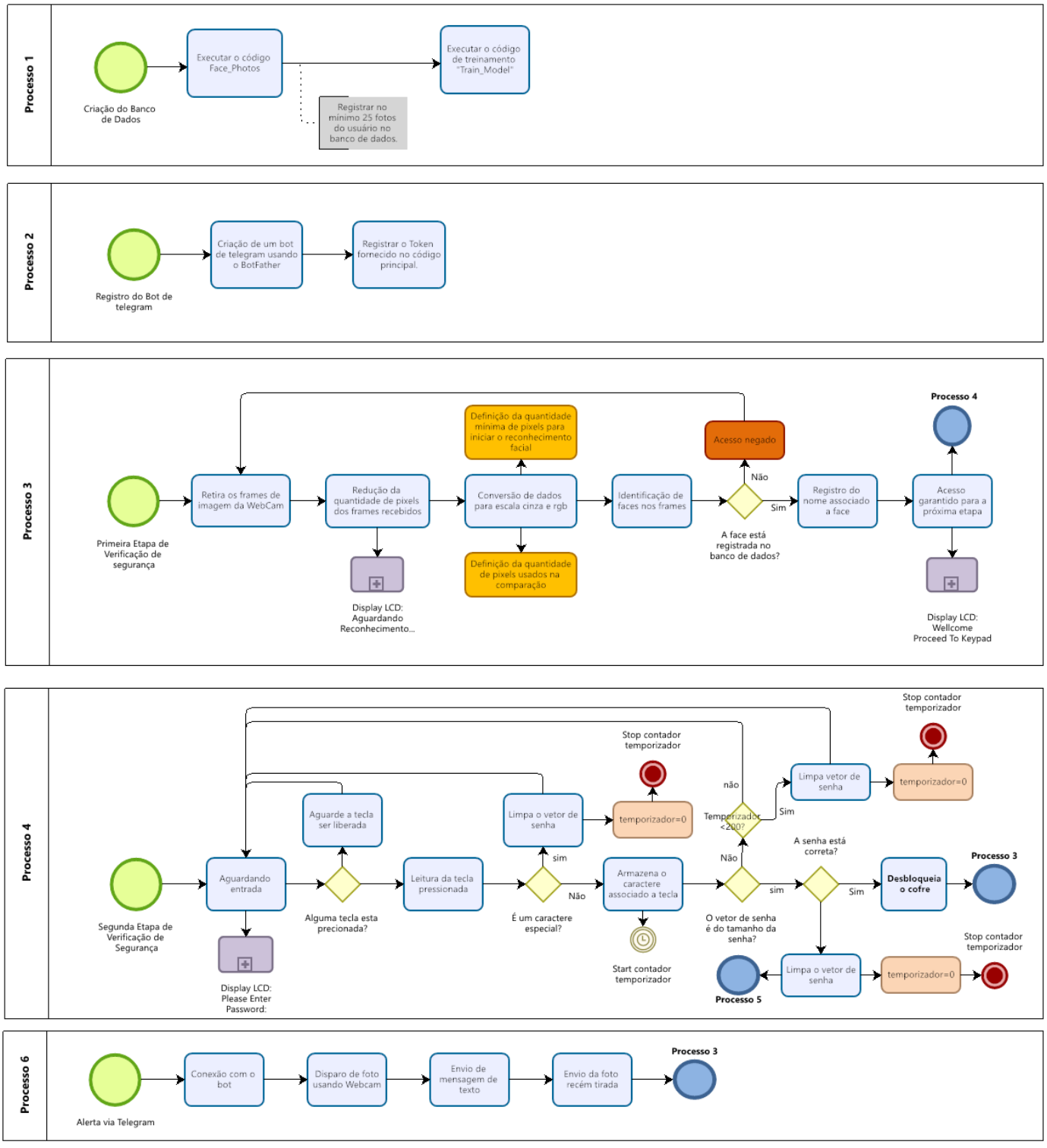


Figura 09: Conexão Relé-Fonte-Trava.

C) Fluxograma de funcionamento:



Powered by bizagi Modeler

D) Subsistemas:

1- Reconhecimento Facial

Para realizar o reconhecimento facial do usuário utilizando a raspberry pi 3B +, primeiramente criamos um algoritmo para extrair fotos contendo o rosto da pessoa que será utilizado posteriormente para treinamento. Para isso utilizamos a biblioteca Open CV, que realiza toda a operação de captura de vídeo e extração de uma imagem. O processo é feito abrindo a webcam, através da biblioteca citada, ao clicar na tecla selecionada para tirar a foto, é capturado o frame e salvo na base de dados definida pelo desenvolvedor "dataset" com o nome definido. Assim, o treinamento pode ser realizado para identificar qualquer pessoa, adicionando suas fotos na base de dados selecionada para o treinamento do algoritmo.

figura 10: armazenamento de fotos para banco de dados

A segunda etapa para o reconhecimento é o treinamento das imagens armazenadas no dataset da aplicação. Para esta etapa utilizamos as bibliotecas *imutils* e pickle para o algoritmo em python e a biblioteca <*vector*> para o algoritmo em C + +. O seu funcionamento começa com a abertura de cada imagem que será utilizada no treinamento, então é realizado o reconhecimento das faces utilizando face recognition em python e a opencv/face em C, em cada uma das imagens determinando as coordenadas (x e y) da face reconhecida na imagem. Esta face é transformada em um "embedding" que seria um vetor com a descrição da imagem. estes vetores são armazenados em um array de encodings, que são então transformados em um arquivo .pickle no caso do python e .xml no algoritmo em C.

Com o treinamento realizado, podemos realizar o reconhecimento facial. Inicializamos a pessoa como "unknown" que seria a desconhecida, determinamos as faces conhecidas como as do modelo .pickle ou .xml criado e utilizaremos o arquivo "*haarcascade_frontalface_default*.xml" como identificador das faces frontais. e então inicializamos o vídeo utilizando a biblioteca do OpenCV. Para cada Frame, realizamos um loop para cada frame, onde o sistema padroniza os frames e os converte para escala de cinza para detecção da face e para RGB para identificação do rosto. Ao ser reconhecido um rosto, o sistema cria uma "caixa" retangular em volta do rosto da pessoa reconhecida. Inicializando um segundo loop, onde é analisado, através dos encodings salvos no processo anterior se esta pessoa é a mesma salva em nosso dataset, ou não. E caso seja reconhecido a mesma pessoa salva em nosso banco de dados, é então passado para o subsistema de senha numérica.

figura 11: Identificação de rostos.

Primeiramente, o sistema pegava uma quantidade de 30 pixels, para formar o retângulo envolvendo o reconhecimento da face, porém, uma quantidade muito pequena apresentava alguns erros por conta de sua alta sensibilidade, reconhecendo alguns objetos com formas parecidas com faces, e tornando o sistema mais lento, por realizar o reconhecimento mais vezes. Resolvemos este problema aumentando a quantidade mínima de pixels para realizar o reconhecimento, forçando a pessoa a se aproximar mais da câmera para realizar o reconhecimento, o que elimina a possibilidade de reconhecer pequenos objetos ou outras pessoas passando ao fundo da câmera. Tornando o sistema mais preciso e rápido.

2- Senha numérica.

Este subsistema é composto pelo teclado matricial e tem como função ser a segunda etapa de verificação do sistema de segurança.

O teclado matricial é dividido em 4 linha e 4 colunas e para realizar a leitura das teclas, as linhas foram declaradas como GPIO.OUT e as colunas como GPIO.IN com resistor de *pull-up.* Ao se detectar uma borda de subida em uma das colunas, é enviado um pulso por todas as linhas para identificar qual botão foi pressionado e registrar o caractere associado a ele.

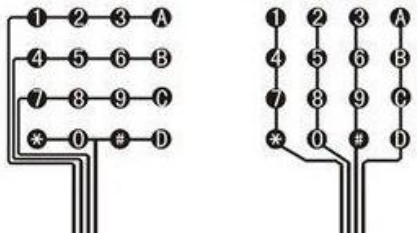


Figura 12: Diagrama do teclado matricial.

Como pode ser visto no fluxograma, o programa fica em espera de uma entrada e ao ser detectada ele dá início ao processo de identificação de comando para verificar qual ação deve ser tomada, no caso se é uma entrada especial para limpar o vetor de entrada de senha ou se é um caractere comum a ser armazenado. Com o início da inserção da senha, um contador é acionado que realiza a função de detecção de ociosidade, para caso o usuário começar a inserir a senha e desistir, fazendo com que o vetor sem entrada de senha seja limpo automaticamente após dez segundos se não houverem novas entradas.

Prosseguindo, quando o tamanho da senha inserida for o mesmo da senha registrada, o sistema realiza a comparação entre as senhas e caso a mesma seja correta o sistema ativa o

relé que por sua vez destranca o cofre, mas caso esteja incorreta o sistema segue para função de alarme.

3- Alarme com Bot Telegram.

Neste subsistema para realizar a comunicação com o smartphone do proprietário foi usada a biblioteca Telepot que funciona em python 2.7 e 3 e faz uso de urllib3 para criar solicitações HTTP.

Esta biblioteca está repleta de funções como: criar e remover eventos no calendário, enviar mensagens de texto, encaminhar mensagens, enviar fotos, áudio, documentos, localização dentre muitas outras funções.

Com a atualização do código para C, a biblioteca que substituiu a Telepot foi a tgbot-cpp que conta com menos recursos, mas que atende aos requisitos para o projeto.

Toda comunicação entre a Raspberry e o usuário é feita por um bot genérico do Telegram. Este bot foi criado usando o recurso disponível no próprio aplicativo chamado BotFather, que realiza todas as operações de criação e controle do bot. Após a criação, o token de acesso gerado foi adicionado ao código para que a biblioteca realizasse a comunicação.

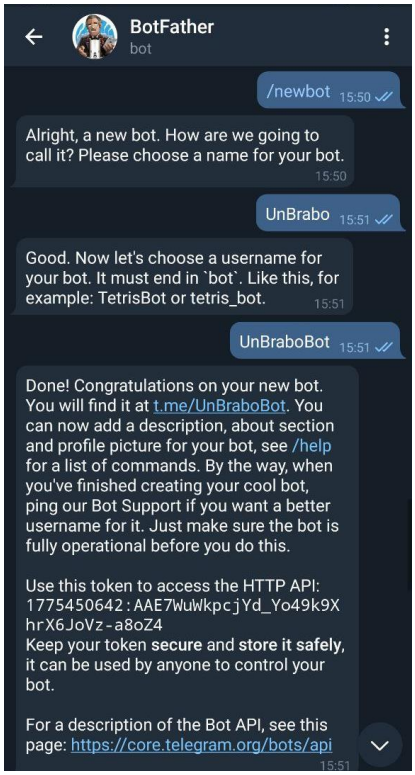


Figura 11: Conversa de criação de bot com o BotFather via Telegram.

De forma conjunta foi usada a biblioteca Open CV para realizar a captura da imagem que é enviada pelo aplicativo no momento em que o sistema de alarme for acionado.

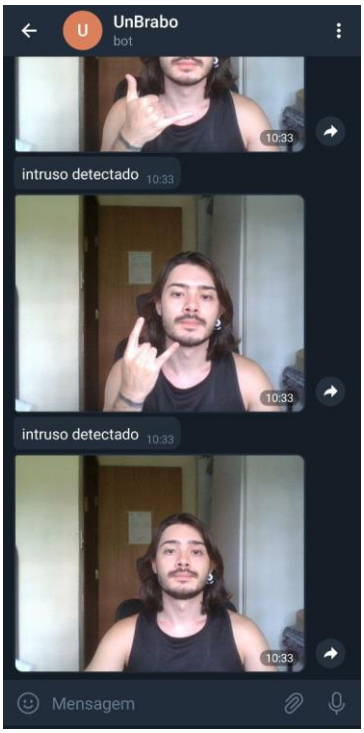


Figura 12: Alerta de intruso enviado pelo Bot do Telegram.

4. Display LCD

Para orientar o usuário durante a operação do cofre, é usado um display LCD 16x2 que informa quais ações devem ser tomadas para destrancar o cofre. Para sua aplicação foi desenvolvido o código que faz uso da biblioteca I2C_LCD_driver para realizar a comunicação usando as entradas DAS e SDL.

Foram definidas as seguintes instruções para serem mostradas:

- Waiting for face recognition
- Wellcome! Go to next step
- Please Enter Password:
- Password Correct, Unlocked!
- Wrong Password, Try Again
- Timeout! Try again

IX. RESULTADOS

Foi construído o cofre usando uma caixa de madeira ripada e nele foram feitos cortes para anexar o display LCD e o teclado matricial.

Figura 13: Cofre com sistema de segurança aprimorado.

Quando ligado, o cofre se encontra em loop filmando os arredores até o momento em que uma face é detectada dentro da distância mínima estipulada, durante este período, é mostrado uma mensagem no display LCD indicando que está aguardando o reconhecimento. Em seguida é criado um retângulo ao redor do rosto detectado em que todos pixel dentro da área serão comparados com os arquivos no Dataset. Se o rosto estiver cadastrado, o sistema segue para a segunda etapa de segurança.

Figura 14: Mensagens de instrução no display LCD.

Figura 15: Distância mínima para o reconhecimento.

Nesta etapa, é realizada a leitura das teclas do teclado matricial. Ao inserir a senha são mostrados caracteres "*" para que o usuário confirme que a leitura foi realizada com sucesso.

Ao fim da inserção da senha, a string é verificada e se tida como correta a tranca é liberada por 4 segundos, após esse período o sistema retorna ao loop inicial de reconhecimento facial. Já no caso da senha está incorreta, é acionada a câmera para tirar uma foto do indivíduo e esta é enviada para o celular do dono do cofre usando o bot do Telegram como mostrado na Figura 12.

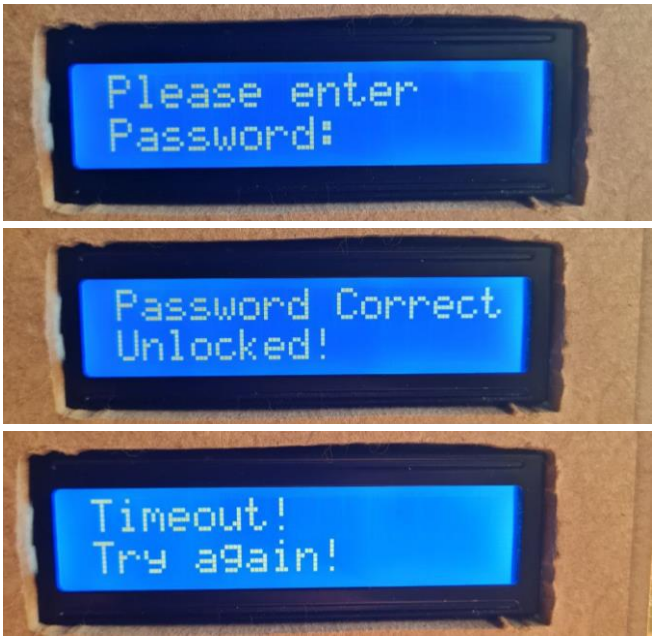


Figura 15: mensagens de instrução no display LCD.

Referências

[1] "Sistema de monitoramento com Raspberry PI", 11 fev. Disponível em: https://www.filipeflop.com/blog/sistema-de-monitoramento-com-raspberry-pi/ Acesso em 25 Fev. 2021

[2] "Sistema de autenticação por Biometria Utilizando Técnicas de Machine Learning e processamento Digital de Sinais / Alef Kaian Feitosa Barbos

[3] "Connect a Raspberry Pi Keypad – Code Lock" Disponível em: <https://tutorials-raspberrypi.com/connecz-raspberry-pi-kecpad-code-lock/> Acesso em 25 Fev. 2020.

[4] HERTZ, Daniel. "How to Use a Keypad With a Raspberry Pi 4", 2019. Disponível em: <https://maker.pro/raspberry-pi/tutorial/how-to-use-a-keypad-with-a-raspberry-pi-4>. Acesso em 25 Fev. 2020.

[5] IDRIS. "Interface 4×4 Keypad With Raspberry Pi", 2019. Disponível em: <https://tutorial.cytron.io/2019/09/18/interface-4x4-keypad-with-raspberry-pi/>. Acesso em 25 Fev. 202

[6] "Python framework for Telegram Bot API", Disponível em: <https://telepot.readthedocs.io/en/latest/> Acesso em 25 Mar 2021

ANEXOS

Anx01 – Código do Sistema de segurança em Python

```
1-  import RPi.GPIO as GPIO
2-  import time
3-  import I2C_LCD_driver
4-  import datetime
5-  import telepot
6-  import sys
7-  import os
8-  from imutils.video import VideoStream
9-  from imutils.video import FPS
10- import face_recognition
11- import imutils
12- import pickle
13- import cv2
14-
```

```
# These are the GPIO pin numbers where the
# lines of the keypad matrix are connected
C1 = 5
C2 = 6
C3 = 13
C4 = 19

# These are the four columns
L1 = 12
L2 = 16
L3 = 20
L4 = 21

# setting solenoid
RELAY = 17

# Initialize 'currentname' to trigger only when a new person is identified.
currentname = "unknown"
# Determine faces from encodings.pickle file model created from train_model.py
encodingsP = "encodings.pickle"
# use this xml file
cascade = "haarcascade_frontalface_default.xml"

# load the known faces and embeddings along with OpenCV's Haar
# cascade for face detection
print("[INFO] loading encodings + face detector...")
data = pickle.loads(open(encodingsP, "rb").read())
detector = cv2.CascadeClassifier(cascade)

# initialize the video stream and allow the camera sensor to warm up
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
# vs = VideoStream(usePiCamera=True).start()
time.sleep(2.0)

# start the FPS counter
fps = FPS().start()

doorUnlock = False

# Setting the LCD display
lcd = I2C_LCD_driver.lcd()

lcd.lcd_display_string("Esperando       ", 1, 0)
lcd.lcd_display_string("Reconhecimento..", 2, 0)

# The password must be 6 characters long
secretCode = "456789"
input = ""
hidden= ""

keypadPressed = -1
passCount = 0
passTimeout = 0

# Telepot bot key
bot = telepot.Bot('1775450642:AAE7WuWkpcjYd_Yo49k9XhrX6JoVz-a8oZ4')

# cam = cv2.VideoCapture(0) ----------------------------------------------------------------------------------------------------------

# The GPIO pin of the column of the key that is currently
# being held down or -1 if no key is pressed
# Setup GPIO
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

GPIO.setup(L1, GPIO.OUT)
GPIO.setup(L2, GPIO.OUT)
GPIO.setup(L3, GPIO.OUT)
GPIO.setup(L4, GPIO.OUT)

# Use the internal pull-down resistors
GPIO.setup(C1, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(C2, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(C3, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(C4, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

GPIO.setup(RELAY, GPIO.OUT)
GPIO.output(RELAY,GPIO.LOW)
# This callback registers the key that was pressed
# if no other key is currently pressed
def keypadCallback(channel):
    global keypadPressed
    if keypadPressed == -1:
        keypadPressed = channel

# Detect the rising edges on the column lines of the
# keypad. This way, we can detect if the user presses
# a button when we send a pulse.
GPIO.add_event_detect(C1, GPIO.RISING, callback=keypadCallback)
GPIO.add_event_detect(C2, GPIO.RISING, callback=keypadCallback)
GPIO.add_event_detect(C3, GPIO.RISING, callback=keypadCallback)
GPIO.add_event_detect(C4, GPIO.RISING, callback=keypadCallback)

# Sets all lines to a specific state. This is a helper
# for detecting when the user releases a button
def setAllLines(state):
    GPIO.output(L1, state)
    GPIO.output(L2, state)
    GPIO.output(L3, state)
    GPIO.output(L4, state)

def checkSpecialKeys():
    global input
    global hidden
    global passCount
    pressed = False

    GPIO.output(L4, GPIO.HIGH)

    if (GPIO.input(C1) == 1):
```

```
125-        lcd.lcd_display_string("Imput Reset!    ", 1, 0)
126-        lcd.lcd_display_string("Password:       ", 2, 0)
127-        time.sleep(0.5)
128-        passCount = 0
129-        pressed = True
130-
131-    GPIO.output(L4, GPIO.LOW)
132-    GPIO.output(L1, GPIO.HIGH)
133-    GPIO.output(L3, GPIO.LOW)
134-
135-    if pressed:
136-        input = ""
137-        hidden= ""
138-
139-    return pressed
140-
141-# reads the columns and appends the value, that corresponds
142-# to the button, to a variable
143-def readLine(line, characters):
144-    global input
145-    global hidden
146-    global passCount
147-    # We have to send a pulse on each line to
148-    # detect button presses
149-    GPIO.output(line, GPIO.HIGH)
150-    if(GPIO.input(C1) == 1):
151-        input = input + characters[0]
152-        hidden = hidden + "*"
153-        passCount = passCount+1
154-    if(GPIO.input(C2) == 1):
155-        input = input + characters[1]
156-        hidden = hidden + "*"
157-        passCount = passCount+1
158-    if(GPIO.input(C3) == 1):
159-        input = input + characters[2]
160-        hidden = hidden + "*"
161-        passCount = passCount+1
162-    if(GPIO.input(C4) == 1):
163-        input = input + characters[3]
164-        hidden = hidden + "*"
165-        passCount = passCount+1
166-    lcd.lcd_display_string("Password:" + hidden, 2, 0)
167-    GPIO.output(line, GPIO.LOW)
168-
169-try:
170-    while True:
171-        if doorUnlock == True:
172-            lcd.lcd_display_string("Please enter    ", 1, 0)
173-            name = "Unknown"
174-            # If a button was previously pressed,
175-            # check, whether the user has released it yet
176-            if keypadPressed != -1:
177-                setAllLines(GPIO.HIGH)
178-                if GPIO.input(keypadPressed) == 0:
179-                    keypadPressed = -1
180-                else:
181-                    time.sleep(0.1)
182-            # Otherwise, just read the input
183-            else:
184-                if not checkSpecialKeys():
185-                    if passCount < 6:
186-                        if passCount > 0:
187-                            readLine(L1, ["1","2","3","A"])
188-                            readLine(L2, ["4","5","6","B"])
189-                            readLine(L3, ["7","8","9","C"])
190-                            readLine(L4, ["*","0","#","D"])
191-                            passTimeout = passTimeout+1
192-                            time.sleep(0.1)
193-                        else:
194-                            readLine(L1, ["1","2","3","A"])
195-                            readLine(L2, ["4","5","6","B"])
196-                            readLine(L3, ["7","8","9","C"])
197-                            readLine(L4, ["*","0","#","D"])
198-                            time.sleep(0.1)
199-                    if passTimeout > 70:
200-                        lcd.lcd_display_string("TIMEOUT
    reseting", 1, 0)
201-                        lcd.lcd_display_string("Password:       ", 2,
    0)
202-                        # Reset vectors
203-                        passCount = 0
204-                        passTimeout = 0
205-                        input = ""
206-                        hidden = ""
207-                        doorUnlock = False
208-                        time.sleep(3)
209-                    else:
210-                        if passCount == 6:
211-                            if input == secretCode:
212-                                lcd.lcd_display_string("CORRECT!
    Opened!", 1, 0)
213-                                lcd.lcd_display_string("Password:
    ", 2, 0)
214-                                passCount = 0
215-                                passTimeout = 0
216-                                input = ""
217-                                hidden = ""
218-                                doorUnlock = False
219-                                GPIO.output(RELAY,GPIO.HIGH)
220-                                time.sleep(4)
221-                                GPIO.output(RELAY,GPIO.LOW)
222-                                # TODO: Unlock a door, turn a light
    on, etc.
223-                            else:
224-                                lcd.lcd_display_string("WRONG!
    Try again", 1, 0)
225-                                lcd.lcd_display_string("Password:
    ", 2, 0)
226-                                # Reset Vector
227-                                passCount = 0
228-                                passTimeout = 0
229-                                input = ""
230-                                hidden = ""
231-                                # Open the camera and take photo
232-                                frame = vs.read()
233-                                cv2.namedWindow("Taking
    picture")
234-                                cv2.imshow("Taking picture", frame)
235-                                cv2.destroyWindow("cam-test")
236-                                cv2.imwrite("Intruder.jpg",frame)
237-
    bot.sendMessage('1799177704','intruso detectado')
238-                                bot.sendPhoto('1799177704',
    photo=open('/home/pi/Documents/Intruder.jpg', 'rb'))
239-                                doorUnlock = False
```

```
240-                    time.sleep(2)
241-                    # TODO: Sound an alarm, send an email, etc.
242-                else:
243-                    time.sleep(0.1)
244-            else:
245-                # grab the frame from the threaded video stream and resize it
246-                # to 500px (to speedup processing)
247-                frame = vs.read()
248-                frame = imutils.resize(frame, width=500)
249-
250-                lcd.lcd_display_string("Esperando      ", 1, 0)
251-                lcd.lcd_display_string("Reconhecimento..", 2, 0)
252-
253-                # convert the input frame from (1) BGR to grayscale (for face
254-                # detection) and (2) from BGR to RGB (for face recognition)
255-                gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
256-                rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
257-
258-                # detect faces in the grayscale frame
259-                rects = detector.detectMultiScale(gray, scaleFactor=1.1,
260-                    minNeighbors=7, minSize=(220, 220),
261-                    flags=cv2.CASCADE_SCALE_IMAGE)
262-
263-                # OpenCV returns bounding box coordinates in (x, y, w, h) order
264-                # but we need them in (top, right, bottom, left) order, so we
265-                # need to do a bit of reordering
266-                boxes = [(y, x + w, y + h, x) for (x, y, w, h) in rects]
267-
268-                # compute the facial embeddings for each face bounding box
269-                encodings = face_recognition.face_encodings(rgb, boxes)
270-                names = []
271-
272-                # loop over the facial embeddings
273-                for encoding in encodings:
274-                    # attempt to match each face in the input image to our known
275-                    # encodings
276-                    matches = face_recognition.compare_faces(data["encodings"],
277-                        encoding)
278-                    name = "Unknown" #if face is not recognized, then print Unknown
279-
280-                    # check to see if we have found a match
281-                    if True in matches:
282-                        # find the indexes of all matched faces then initialize a
283-                        # dictionary to count the total number of times each face
284-                        # was matched
285-                        matchedIdxs = [i for (i, b) in enumerate(matches) if b]
286-                        counts = {}
287-                        # to unlock the door
288-                        lcd.lcd_display_string("Password:      ", 2, 0)
289-                        doorUnlock = True
290-
291-                        # loop over the matched indexes and maintain a count for
292-                        # each recognized face face
293-                        for i in matchedIdxs:
294-                            name = data["names"][i]
295-                            counts[name] = counts.get(name, 0) + 1
296-
297-                        # determine the recognized face with the largest number
298-                        # of votes (note: in the event of an unlikely tie Python
299-                        # will select first entry in the dictionary)
300-                        name = max(counts, key=counts.get)
301-
302-                        #If someone in your dataset is identified, print their name on the screen
303-                        if currentname != name:
304-                            currentname = name
305-                            print("Wellcome ", currentname)
306-                            print("Proceed to insert password")
307-
308-                    # update the list of names
309-                    names.append(name)
310-
311-                # loop over the recognized faces
312-                for ((top, right, bottom, left), name) in zip(boxes, names):
313-                    # draw the predicted face name on the image - color is in BGR
314-                    cv2.rectangle(frame, (left, top), (right, bottom),
315-                        (0, 255, 225), 2)
316-                    y = top - 15 if top - 15 > 15 else top + 15
317-                    cv2.putText(frame, name, (left, y), cv2.FONT_HERSHEY_SIMPLEX,
318-                        .8, (0, 255, 255), 2)
319-
320-                # display the image to our screen
321-                cv2.imshow("Facial Recognition is Running", frame)
322-                key = cv2.waitKey(1) & 0xFF
323-
324-                # quit when 'q' key is pressed
325-                if key == ord("q"):
326-                    break
327-
328-                # update the FPS counter
329-                fps.update()
330-
331- except KeyboardInterrupt:
332-     lcd.lcd_clear()
333-     cv2.destroyAllWindows()
334-     #vs.stop()
```

Anx 2 – Código: Armazenamento de fotos para o banco de dados em Python.

```
1-  import cv2
2-
3-  name = 'Joao' #replace with your name
```

```
4-
5-  cam = cv2.VideoCapture(0)
6-
7-  cv2.namedWindow("press space to take a photo", cv2.WINDOW_NORMAL)
8-  cv2.resizeWindow("press space to take a photo", 500, 300)
9-
10- img_counter = 0
11-
12- while True:
13-     ret, frame = cam.read()
14-     if not ret:
15-         print("failed to grab frame")
16-         break
17-     cv2.imshow("press space to take a photo", frame)
18-
19-     k = cv2.waitKey(1)
20-     if k%256 == 27:
21-         # ESC pressed
22-         print("Escape hit, closing…")
23-         break
24-     elif k%256 == 32:
25-         # SPACE pressed
26-         img_name = "dataset/"+ name +"/image_{}.jpg".format(img_counter)
27-         cv2.imwrite(img_name, frame)
28-         print("{} written!".format(img_name))
29-         img_counter += 1
30-
31- cam.release()
32-
33- cv2.destroyAllWindows()
```

Anx 3 – Código: Modelo de treino em Python.

```
1-  #! /usr/bin/python
2-
3-  # import the necessary packages
4-  from imutils import paths
5-  import face_recognition
6-  #import argparse
7-  import pickle
8-  import cv2
9-  import os
10-
11- # our images are located in the dataset folder
12- print("[INFO] start processing faces…")
13- imagePaths = list(paths.list_images("dataset"))
14-
15- # initialize the list of known encodings and known names
16- knownEncodings = []
17- knownNames = []
18-
19- # loop over the image paths
20- for (i, imagePath) in enumerate(imagePaths):
21-     # extract the person name from the image path
22-     print("[INFO] processing image {}/{}".format(i + 1, len(imagePaths)))
23-     name = imagePath.split(os.path.sep)[-2]
24-
25-     # load the input image and convert it from RGB (OpenCV ordering)
26-     # to dlib ordering (RGB)
27-     image = cv2.imread(imagePath)
28-     rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
29-
30-     # detect the (x, y)-coordinates of the bounding boxes
31-     # corresponding to each face in the input image
32-     boxes = face_recognition.face_locations(rgb, model="hog")
33-
34-     # compute the facial embedding for the face
35-     encodings = face_recognition.face_encodings(rgb, boxes)
36-
37-     # loop over the encodings
38-     for encoding in encodings:
39-         # add each encoding + name to our set of known names and
40-         # encodings
41-         knownEncodings.append(encoding)
42-         knownNames.append(name)
43-
44- # dump the facial encodings + names to disk
45- print("[INFO] serializing encodings…")
46- data = {"encodings": knownEncodings, "names": knownNames}
47- f = open("encodings.pickle", "wb")
48- f.write(pickle.dumps(data))
49- f.close()
```

Anx 4 – Código: Sistema de segurança em C

```
1-  #include <iostream>
2-  #include <vector>
3-  #include <string>
4-  #include <stdlib.h>
5-  #include <unistd.h>
6-
7-  #include <opencv2/core.hpp>
8-  #include <opencv2/videoio.hpp>
9-  #include <opencv2/imgproc.hpp>
10- #include <opencv2/face.hpp>
11- #include <opencv2/imgcodecs.hpp>
12-
13- #include "face_detect.hpp"
14- #include "config.hpp"
15-
16- #include <Keypad.h>
17-
18-
19- #define Password_Lenght 5
20- #define PIEZO   8
21- #define RELAY   A0
22-
23- #include <LiquidCrystal_I2C.h>
24-
25- #include <iostream>
26- #include <stdio.h>
27- #include <tgbot/tgbot.h>
28- #include<wiringPi.h> // alike GPIO
29-
30- #define TOKEN "1407581882:AAHl3q7nJ97I7eql6Gic0t76rde6gwlhlGs"
31- #define chat_ID 824356636
32-
33- LiquidCrystal_I2C lcd(0x27, 20, 2);
34-
35-
```

```
int pos = 0;    // variable to store the servo position

char Data[Password_Lenght];
char Master[Password_Lenght] = "5698";
byte data_count = 0, master_count = 0;
char customKey;

const byte ROWS = 4;
const byte COLS = 4;

char keys [ROWS] [COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
byte rowPins[ROWS] = {6, 5, 4, 3};
byte colPins[COLS] = {11, 10, 9, 7};

Keypad myKeypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS );

char inChar;
String inString;

std::string get_model_name(int argc, char *argv[]) {

    std::string model_name;
    if (argc != 2) {
        std::cout << "Error: Correct usage: ./train <model_name>" << std::endl;
        exit(1);
    } else {
        return argv[1];
    }

}

void setup() {
  pinMode(PIEZO, OUTPUT);
  pinMode(RELAY, OUTPUT);
  digitalWrite(RELAY, HIGH);
  lcd.init();
  lcd.backlight();
  lcd.clear();
  lcd.print(" ENTER PASSWORD");
}

void clearData()
{
  while (data_count != 0)
  { // This can be used for any array size,
    Data[data_count–] = 0; //clear array for new data
  }
  return;
}


int main(int argc, char *argv[]) {

    TgBot::Bot bot(TOKEN);
    bot.getEvents().onCommand("start", [&bot](TgBot::Message::Ptr message) {
        bot.getApi().sendMessage(message->chat->id, "Ola! Informaremos se algum desconhecido tentar abrir seu cofre!");
    });

    GPIO input_pin("17", "in");
    std::cout << input_pin.read_value() << std::endl;

    std::string model_name = get_model_name(argc, argv);

    std::cout << "Loading model..." << std::flush;
    cv::Ptr<cv::face::FaceRecognizer> model = cv::face::LBPHFaceRecognizer::create();
    model->read(std::string(MODEL_DIR) + model_name + ".xml");
    std::cout << "[DONE]" << std::endl;

    cv::VideoCapture camera = get_camera();

    bool loop = true;
    while (loop) {

            cv::Mat image;

            flush_capture_buffer(camera);
            camera >> image;
            cv::cvtColor(image, image, cv::COLOR_RGB2GRAY);

            std::vector<cv::Rect> face_regions = detect_faces(image);
            std::cout << "Deteced " << face_regions.size() << " faces." << std::endl;

  lcd.printf(" Waiting recognition...");

            if (face_regions.size() == 1) {

                image = image(face_regions[0]);
                cv::resize(image, image, cv::Size(FACE_WIDTH, FACE_HEIGHT), 0, 0, cv::INTER_LANCZOS4);

                int label;
                double confidence;
                model->predict(image, label, confidence);

                std::cout << "Results: " << label << ", " << confidence << std::endl;

                // If a face is recognized and authorized
                if (label != 0 && confidence <= POSITIVE_THRESHOLD) {
                    std::cout << "****** Detected allowed face with label: " << label << std::endl;
                     customKey = myKeypad.getKey();
            if (customKey)
            {
               Data[data_count] = customKey;
               data_count++;
```

```
144-            }
145-
146-            if (data_count == Password_Lenght – 1)
147-              if (!strcmp(Data, Master))
148-              {
149-              digitalWrite(RELAY, LOW);
150-              lcd.clear();
151-              lcd.print("    DOOR OPEN");
152-              sleep(5000);
153-              lcd.clear();
154-              lcd.print("    DOOR LOCK");
155-              digitalWrite(RELAY, HIGH);
156-              sleep(1000);
157-              lcd.clear();
158-              lcd.printf(" WAITING RECOGNITION...");
159-              {
160-            else {
161-              bot.getApi().sendMessage((message->chat->id,
    "INTRUDER ALERT");
162-              lcd.clear();
163-              lcd.print(" WRONG PASSWORD");
164-              }
165-              clearData();
166-              }
167-        }
168-          }
169-
170-                std::string input;
171-                std::cout << "Enter 'q' to quit, or any other key
    to take another picture: " << std::endl;
172-                std::cin >> input;
173-                if (input == "q" || input == "Q") {
174-                        loop = false;
175-                }
176-
177-      }
178-
179- }
```

Anx 05 – Código: Display LCD em C

```
1-  #include<wiringPi.h> // alike GPIO
2-  #include<string.h>
3-  #include<stdio.h>
4-
5-  #define m1 29
6-  #define m2 28
7-
8-  #define RS 11
9-  #define EN 10
10- #define D4 6
11- #define D5 5
12- #define D6 4
13- #define D7 1
14-
15- char pass[4];
16- char pass1[]={'1','2','3','4'};
17- int n=0;
18- char row[4]={21, 14, 13, 12};
19- char col[4]={22, 23, 24, 25};
20-
21- char num[4][4]={
22-              {'1','2','3','A'},
23-              {'4','5','6','B'},
24-              {'7','8','9','C'},
25-              {'*','0','#','D'}
26-             };
27-
28- void keypad();
29- void lcdcmd(unsigned int ch)
30-  {
31-    int temp=0x80;
32-    digitalWrite(D4, temp & ch<<3);
33-    digitalWrite(D5, temp & ch<<2);
34-    digitalWrite(D6, temp & ch<<1);
35-    digitalWrite(D7, temp & ch);
36-    digitalWrite(RS, LOW);
37-    digitalWrite(EN, HIGH);
38-    delay(10);
39-    digitalWrite(EN, LOW);
40-    digitalWrite(D4, temp & ch<<7);
41-    digitalWrite(D5, temp & ch<<6);
42-    digitalWrite(D6, temp & ch<<5);
43-    digitalWrite(D7, temp & ch<<4);
44-    digitalWrite(RS, LOW);
45-    digitalWrite(EN, HIGH);
46-    delay(10);
47-    digitalWrite(EN, LOW);
48-  }
49-
50- void write(unsigned int ch)
51-  {
52-    int temp=0x80;
53-    digitalWrite(D4, temp & ch<<3);
54-    digitalWrite(D5, temp & ch<<2);
55-    digitalWrite(D6, temp & ch<<1);
56-    digitalWrite(D7, temp & ch);
57-    digitalWrite(RS, HIGH);
58-    digitalWrite(EN, HIGH);
59-    delay(10);
60-    digitalWrite(EN, LOW);
61-    digitalWrite(D4, temp & ch<<7);
62-    digitalWrite(D5, temp & ch<<6);
63-    digitalWrite(D6, temp & ch<<5);
64-    digitalWrite(D7, temp & ch<<4);
65-    digitalWrite(RS, HIGH);
66-    digitalWrite(EN, HIGH);
67-    delay(10);
68-    digitalWrite(EN, LOW);
69-  }
70-
71- void clear()
72-  {
73-     lcdcmd(0x01);
74-  }
75-
76-  void setCursor(int x, int y)
77-  {
78-    int set=0;
79-    if(y==0)
80-    set=128+x;
81-    if(y==1)
82-    set=192+x;
83-    lcdcmd(set);
84-  }
85-
86-  void print(char *str)
```

```
87-   {
88-     while(*str)
89-     {
90-      write(*str);
91-      str++;
92-     }
93-   }
94-
95-   void begin(int x, int y)
96-   {
97-     lcdcmd(0x02);
98-     lcdcmd(0x28);
99-     lcdcmd(0x06);
100-    lcdcmd(0x0e);
101-    lcdcmd(0x01);
102-  }
103-
104-  void enter()
105-  {
106-    clear();
107-    print("Current Passkey:");
108-    setCursor(0,1);
109-    keypad();
110-    if(strncmp(pass,pass1,4)==0)
111-    {
112-      clear();
113-      print("Enter New Passkey");
114-      setCursor(0,1);
115-      keypad();
116-      for(n=0;n<4;n++)
117-      pass1[n]=pass[n];
118-      clear();
119-      print("Passkey Changed.");
120-      delay(3000);
121-      return;
122-    }
123-
124-    else
125-    {
126-      clear();
127-      print("Wrong password");
128-      setCursor(0,1);
129-      return;
130-    }
131-  }
132-
133-  void gate_open()
134-  {
135-     digitalWrite(m1, LOW);
136-     digitalWrite(m2, HIGH);
137-     delay(2000);
138-  }
139-
140-  void gate_stop()
141-  {
142-    digitalWrite(m1, LOW);
143-    digitalWrite(m2, LOW);
144-    delay(2000);
145-  }
146-
147-   void gate_close()
148-  {
149-    digitalWrite(m1, HIGH);
150-    digitalWrite(m2, LOW);
151-    delay(2000);
152-  }
153-
154-  void choice()
155-  {
156-    digitalWrite(col[0], LOW);
157-    digitalWrite(col[1],HIGH);
158-     if(digitalRead(row[3])==0)
159-     {
160-       clear();
161-       print("Enter Password:");
162-       setCursor(0,1);
163-       keypad();
164-       if(strncmp(pass,pass1,4)==0)
165-       {
166-        clear();
167-        setCursor(0,1);
168-        print("Door Unlock");
169-        gate_open();
170-        gate_stop();
171-        gate_close();
172-        gate_stop();
173-        return;
174-       }
175-
176-       else
177-       {
178-         clear();
179-         print("Access Denied");
180-         setCursor(0,1);
181-         return;
182-       }
183-     }
184-     digitalWrite(col[0], HIGH);
185-     digitalWrite(col[1], LOW);
186-     if(digitalRead(row[3])==0)
187-     {
188-      enter();
189-     }
190-  }
191-
192-  void setup()
193-  {
194-  if(wiringPiSetup()==-1)
195-  print("Error");
196-  pinMode(RS, OUTPUT);
197-  pinMode(EN, OUTPUT);
198-  pinMode(D4, OUTPUT);
199-  pinMode(D5, OUTPUT);
200-  pinMode(D6, OUTPUT);
201-  pinMode(D7, OUTPUT);
202-  pinMode(m1, OUTPUT);
203-  pinMode(m2, OUTPUT);
204-  for(n=0;n<4;n++)
205-  pinMode(row[n], INPUT);
206-  for(n=0;n<4;n++)
207-  pinMode(col[n], OUTPUT);
208-  for(n=0;n<4;n++)
209-  digitalWrite(col[n], HIGH);
210-  }
211-
```

Anx 06 – Código: Modelo de treino em C

```
1-  #include <iostream>
2-  #include <vector>
3-
4-  #include <opencv2/core.hpp>
5-  #include <opencv2/imgcodecs.hpp>
6-  #include <opencv2/imgproc.hpp>
7-  #include <opencv2/face.hpp>
8-
9-  #include "dataset.hpp"
10- #include "config.hpp"
11- #include "face_detect.hpp"
12-
13- std::string get_model_name(int argc, char *argv[]) {
14-
15-     std::string model_name;
16-     if (argc != 2) {
17-         std::cout << "Error: Correct usage: ./train <model_name>" << std::endl;
18-         exit(1);
19-     } else {
20-         return argv[1];
21-     }
22-
23- }
24-
25-
26- int main(int argc, char *argv[]) {
27-
28-     std::string model_name = get_model_name(argc, argv);
29-
30-     std::vector<int> labels;
31-
32-     std::vector<cv::Mat> training_images;
33-
34-     // Load positive training images
35-     std::vector<std::string> directories = get_directories(std::string(TRAINING_DIR) + "positive/");
36-     std::cout << "Loading positive training images for subjects:" << std::endl;
37-     for (int i = 0; i < directories.size(); i++) {
38-         int right_padding = 20 - directories[i].size();
39-         std::cout << "\t" << "(" << i + 1 << ") " << directories[i] << std::string(right_padding, ' ') << std::flush;
40-         std::vector<std::string> image_files = get_files(std::string(TRAINING_DIR) + "positive/" + directories[i]);
41-         for (int j = 0; j < image_files.size(); j++) {
42-
training_images.push_back(cv::imread(std::string(TRAINING_DIR) + "positive/" + directories[i] + "/" + image_files[j], cv::IMREAD_GRAYSCALE));
43-             labels.push_back(i + 1);
44-         }
45-         std::cout << "[DONE]" << std::endl;
46-     }
47-
48-     // Prepare images for training
49-     std::cout << "Preparing images for training ... ";
50-     for (int i = 0; i < training_images.size(); i++) {
51-         cv::resize(training_images[i], training_images[i], cv::Size(FACE_WIDTH, FACE_HEIGHT), 0, 0, cv::INTER_LANCZOS4);
52-     }
53-     std::cout << "[DONE]" << std::endl;
54-
55-     // Train model
56-     std::cout << "Training model with " << training_images.size() << " faces ... " << std::flush;
57-     cv::Ptr<cv::face::FaceRecognizer> model = cv::face::LBPHFaceRecognizer::create();
58-     model->train(training_images, labels);
59-     std::cout << "[DONE]" << std::endl;
60-
61-     // Test Prediction
62-     int label;
63-     double confidence;
64-     model->predict(training_images[0], label, confidence);
65-
66-     // Save model
67-     std::string filepath = std::string(MODEL_DIR) + model_name + ".xml";
68-     std::cout << "Saving model to " << filepath << " ... " << std::flush;
69-     model->write(filepath);
70-     std::cout << "[DONE]" << std::endl;
71-
72- }
```