

## PONTO DE CONTROLE 2

### CONTROLE DE ACESSO VIA RECONHECIMENTO DE FACE HUMANA

*Antônio Aldísio - 14/0130811 — Vitor Carvalho de Almeida - 14/0165380*

Programa de Graduação em Engenharia Eletrônica, Faculdade Gama  
Universidade de Brasília  
Gama, DF, Brasil

email: aldisiofilho@gmail.com — vitorcarvalhoamd@gmail.com

#### RESUMO

O projeto consiste em construir um sistema de controle de acesso ativado por reconhecimento facial. Será possível enviar os dados de acesso via rede para um banco de dados. Como validação será confeccionada uma porta em miniatura.

**Palavras-chave:** Controle de acesso, Raspberry Pi, OpenCV, reconhecimento facial, segurança.

Para este ponto de controle, é necessário comunicar a Raspberry Pi com os elementos que serão utilizados no projeto.

O projeto em questão utiliza uma trava solenoide, que trabalha com tensão e corrente maiores do que a placa consegue fornecer. Logo, é necessário usar um sistema de chaveamento.

Nas próximas seções são apresentadas as soluções para o problema.

#### 1. INTRODUÇÃO

O mundo encontra-se em uma grande evolução, nos dias atuais a automação utilizada para controle de acesso é a biometria por impressão digital. Porém o usuário tem quer ter uma interação direta e tátil com o sistema para a sua liberação. O controle de acesso via reconhecimento facial elimina a necessidade de interação direta do usuário e pode ser implementado juntamente ao de monitoramento por câmeras, utilizando o mesmo dispositivo para a aquisição das imagens.

Além da facilidade do uso e a eliminação da possibilidade de esquecer a chave de acesso, é possível armazenar as informações para utilizar como controle de ponto, ou adaptar para um sistema de controle/monitoramento de produtividade em uma empresa.

Com base nessa tendência e buscando uma facilidade para o usuário, esse artigo propõe a construção de um sistema de reconhecimento facial para abertura de portas.

O objetivo desse projeto é a construção de um sistema de abertura de porta através do reconhecimento do rosto de usuários cadastrados e enviar dados de acessos pela rede.

Um sistema de reconhecimento facial traz alguns benefícios como: praticidade, segurança. No caso desenvolvimento o enfoque é: a segurança, visto que a porta só se abrirá após o sistema reconhecer um usuário autorizado; e a possibilidade de utilizar essa validação de entrada como um ponto eletrônico para contagem de horas trabalhadas e geração de outros dados estatísticos

#### 2. DESENVOLVIMENTO

##### 2.1. Descrição do Hardware

Foi montado um sistema de ativação da trava eletrônica. Utilizando os seguintes materiais:

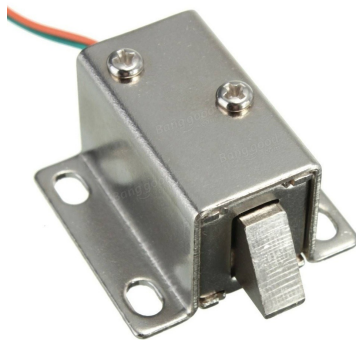
- Trava solenoide 12V (figura 1);
- Fonte DC 12V ;
- Resistor de 1 KOhm;
- Transistor NPN (TIP41);
- Jumpers
- Protoboard

Na protoboard foi montado o circuito da figura 2.

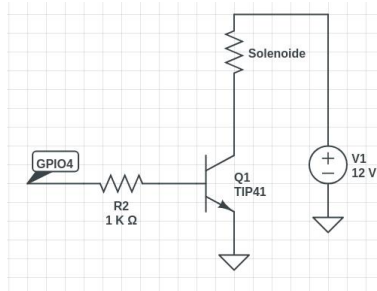
O pino de entrada foi conectado à GPIO4 da Raspberry Pi 3 para que fossem enviados os comandos para abrir a porta.

A trava solenoide mantém a porta fechada até que seja inserida uma tensão de 12V em seus terminais. Neste momento, o solenoide faz com que o "dente" da trava seja retraído, liberando a abertura da porta. Ao retirar a tensão dos terminais, uma mola retorna a trava para a posição original, travando a porta novamente. [1]

**Fig. 1.** Trava eletrônica solenoide 12V



**Fig. 2.** Ativação da trava eletrônica solenoide 12V



Foi utilizada uma fonte DC de 12V - 2A com conexão Jack P4, ligada na protoboard com um conector Jack P4 fêmea.

Foi conectada uma caixa de som à saída P2 da Raspberry Pi para reproduzir sons de confirmação ou negação de acesso.

Para receber a requisição de acesso, foi montado um circuito com botão em modo Pull-Up, como mostra o esquemático da figura 3

Foi utilizada uma câmera com conexão USB para testes (figura 4).

## 2.2. Descrição do Software

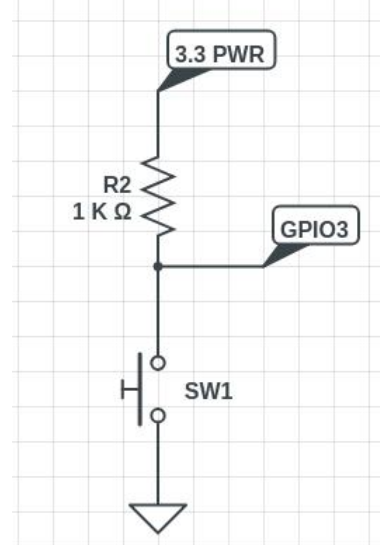
### 2.2.1. Cadastro

O projeto é composto de duas rotinas: *cadasturar usuario* e *abertura*. Para realização dessas rotinas será utilizada a raspberry como servidor e o cliente será um BOT no Telegram [2][3]. Pode-se observar as etapas do sistema de cadastro do ponto de vista do administrador na figura abaixo:

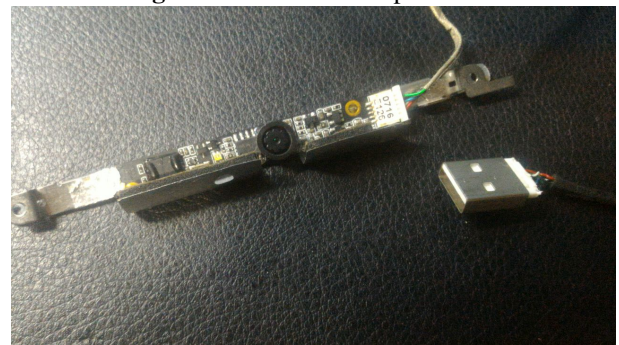
Na rotina *cadasturar* será enviada uma foto via BOT Telegram, que pode ser tirada do celular ou computador do usuário e essa foto servirá como base para o sistema. Este ficará aguardando o novo usuário pressionar o botão para

- 1 Modo de espera
- 2 Botão foi pressionado? N-Espera S-Segue

**Fig. 3.** Botão em modo Pull-Up



**Fig. 4.** Câmera utilizada para testes



enviada e validar o cadastro. Após isso, será enviada uma mensagem ao cliente (administrador) validando o cadastro do novo usuario.

Para tirar uma foto com a câmera instalada na Raspberry Pi é utilizado o seguinte comando no terminal [4]:

`fswebcam nome_imagem.jpg`

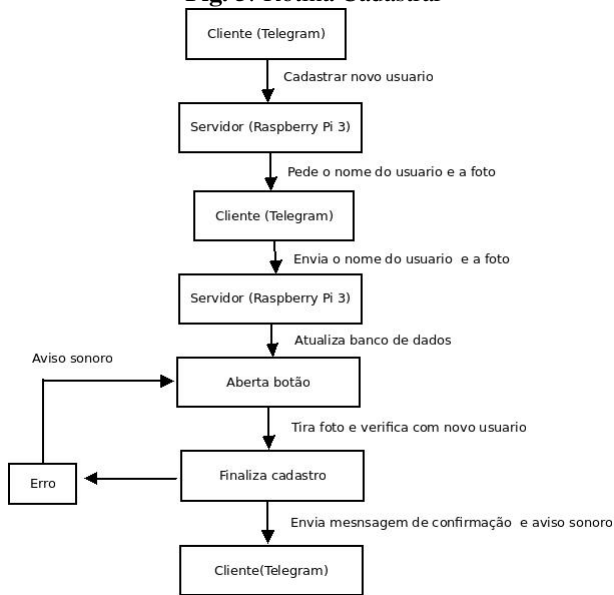
Onde pode ser escolhido qualquer nome para a imagem.

### 2.2.2. Acesso e resposta ao usuário

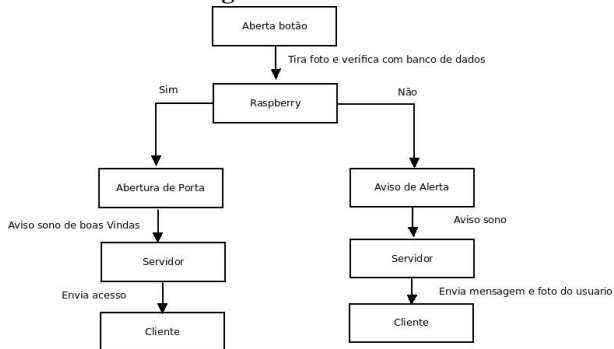
Para a abertura da porta, será executada a rotina descrita da figura 6:

A GPIO3 foi configurada de modo a ficar em modo de espera, utilizando a função *polling*. Abaixo é apresentado o pseudo-código da campainha. O código completo do teste utilizado pode ser encontrado no apêndice.

**Fig. 5. Rotina Cadastrar**



**Fig. 6. Rotina Acesso**



### 3 Envia requisição de acesso para o código principal.

A rotina para liberar a porta foi feita em um arquivo de instruções bash. O código *abre.sh* é mostrado no apêndice.

- 1 Primeiramente é reproduzido o som [5] de uma confirmação de acesso pela caixa de som, para que o usuário saiba que pode entrar
- 2 Depois a GPIO4 é definida como saída e colocada em nível lógico alto.
- 3 É definido um tempo de espera, no caso 3 segundos, para que a trava se mantenha retraída e o usuário possa empurrar a porta.
- 4 Ao final da contagem a GPIO4 volta para o nível lógico baixo. Neste momento, ao encostar a porta, esta será trancada.

5 Por fim, a GPIO4 é liberada para uso em outra rotina.

Caso o usuário não tenha acesso cadastrado, ao tocar a campainha deve ser executado o código *negado.sh*, mostrado abaixo.

```

1 #!/bin/bash
2
3 omxplayer -o local /home/pi/embarcados/projeto_final/sons/nao.mp3

```

Neste caso, a única função realizada é a reprodução de um som de negação na caixa de som, porém na prática, esse *script* será chamado pelo servidor presente na Raspberry Pi. Após a execução deste *script*, o servidor enviará para o cliente [6] a foto do sujeito que solicitou acesso.

Prevendo uma visita de alguém de confiança do administrador, porém não cadastrada no sistema, ou eventuais falhas no reconhecimento facial, o administrador terá a opção de abrir a porta remotamente, via um comando master de acesso sem a necessidade de reconhecimento facial.

#### 2.2.3. Servidor

O servidor é montado em cima da API disponibilizada pelo próprio Telegram, e foi escrito na linguagem Python [2][3]. Para esse ponto de controle é realizado apenas a identificação dos textos "oi" e "olá" com a resposta do bot "Olá Mestre, o que deseja?". O código pode ser visto no apêndice 3.

#### 2.2.4. Reconhecimento facial

Está sendo utilizado as seguintes bibliotecas:

- opencv 3.4.1 [7]
- face\_recognition [8]

A primeira responsável pela visão computacional, ou seja, fazer o sistema identificar onde está um rosto na imagem. Já o *face\_recognition* é responsável pela comparação do rosto com os cadastrados na base de dados.

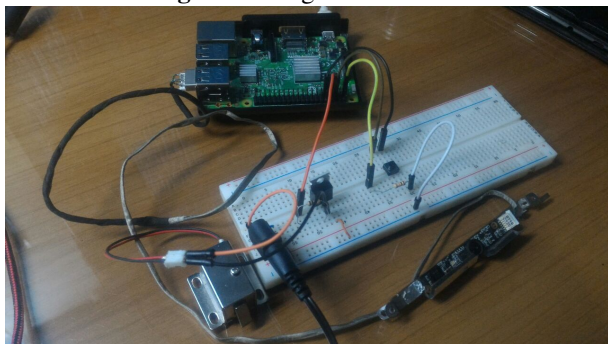
As duas bibliotecas estão na linguagem Python e para validação da factibilidade do projeto foi utilizado um exemplo da biblioteca *face\_recognition*, cuja, faz a verificação quase instantânea em um vídeo ao vivo. O código pode ser visto no apêndice.

## 3. RESULTADOS

O conjunto montado ficou como mostrado na figura 7:

A ativação da trava eletrônica foi realizada com sucesso, sem sobreaquecimento do transistor, nem falha na comunicação.

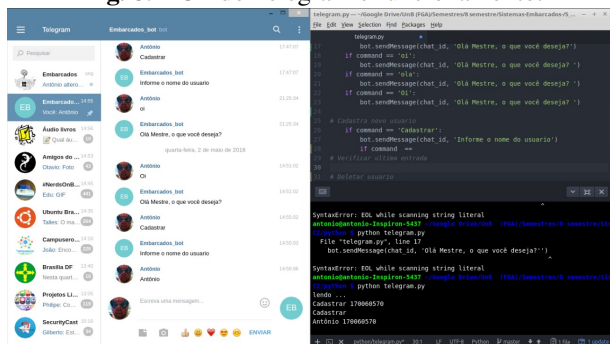
Fig. 7. Montagem do circuito



Na parte de software realizamos os teste apenas no computador, pois algumas bibliotecas não foram possíveis a usa instalação na raspberry até o momento.

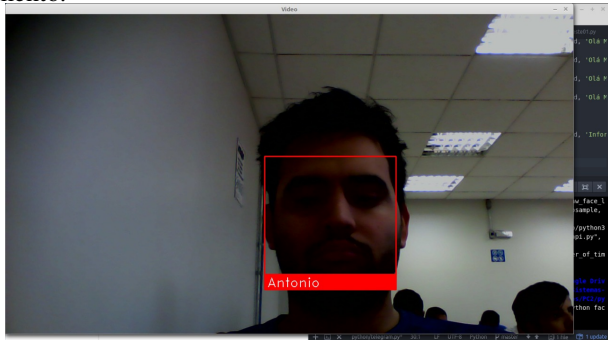
A figura 8 mostra o funcionamento da conversa do administrador do sistema com o Bot do Telegram.

Fig. 8. BOT do Telegram e funcionamento.



Na figura 9 pode ser visto o reconhecimento facial em funcionamento.

Fig. 9. Reconhecimento facial em funcionamento funcional.



## 4. DISCUSSÃO E CONCLUSÕES

Pelos experimentos realizados, concluiu-se que o projeto pode ser implementado da forma como foi pensado inicialmente, pois a comunicação da Raspberry Pi com os periféricos (trava, botão, câmera) funcionou corretamente.

Com o sistema rodando no computador so vimos um pequeno atraso da imagem com a indentificação, isso não será um problema para nosso projeto, pois iremos realizar uma foto. O que pode ser um problema é o delay do reconhecimento com a abertura da porta.

A bibliotecas de reconhecimento facial OpenCV e face\_recognition funcionaram corretamente no notebook, ainda falta implementá-las na Raspberry Pi.

Uma limitação das bibliotecas é que elas não diferenciam rostos reais de rostos em fotos mostradas para a câmera. Isso é um grande problema de segurança para o projeto, porém a dupla já está estudando técnicas de diferenciação destes casos.

Decidiu-se usar o bot do Telegram como interface do administrador com o sistema, porque assim o acesso é muito mais prático e podem ser recebidas notificações no celular sem necessidade de instalação de outros aplicativos.

Em relação ao servidor e cliente é uma boa escolha a utilização da API do telegram, pois assim tornamos o acesso ao sistema remotamente super fácil e acessível a qualquer usuário que tenha o mínimo de conhecimento em mensagens de comunicação. Por outro lado a sua programação será um pouco trabalhosa, pois será necessário considerar as variações de entrada do usuário para uma mesma ação.

Para este documento, foram escritos programas cuja rotina principal é específica para cada ação. Porém, para o próximo ponto de controle, os códigos serão unidos como subrotinas de um código principal de controle do sistema.

## 5. REFERENCIAS

- [1 ] <https://www.filipeflop.com/blog/acionando-trava-eletrica-com-rfid/>
- [2 ] <https://pypi.org/project/pyTelegramBotAPI/0.2.9/>
- [3 ] <https://core.telegram.org/bots/api>
- [4 ] <https://www.raspberrypi.org/documentation/usage/webcams/README.md>
- [5 ] <https://www.raspberrypi.org/documentation/usage/audio/README.md>
- [6 ] <https://medium.com/@rosbots/ready-to-use-image-raspbian-stretch-ros-opencv-324d6f8dcd96>
- [7 ] <https://github.com/opencv/opencv>
- [8 ] [https://github.com/ageitgey/face\\_recognition](https://github.com/ageitgey/face_recognition)

## 6. APENDICE

Códigos utilizados

Rotina do botão: *campainha.c*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <fcntl.h>
4 #include <sys/poll.h>
5 #include <unistd.h>
6
7 int main(void)
8 {
9     struct pollfd pfd;
10    char buffer;
11    system("echo 3 > /sys/class/gpio/
        export");
12    system("echo falling > /sys/class/
        gpio/gpio3/edge");
13    system("echo in > /sys/class/gpio/
        gpio3/direction");
14    pfd.fd = open("/sys/class/gpio/
        gpio3/value", ORDONLY);
15    if (pfd.fd < 0)
16    {
17        puts("Erro abrindo /sys/
            class/gpio/gpio3/value
            ");
18        puts("Execute este
            programa como root");
19        return 1;
20    }
21    read(pfd.fd, &buffer, 1);
22    pfd.events = POLLPRI | POLLERR;
23    pfd.revents = 0;
24    puts("Augardando campainha");
25    poll(&pfd, 1, -1);
26    if (pfd.revents) puts("Campainha
        pressionada: iniciar rotina de
        reconhecimento");
27    close(pfd.fd);
28    system("echo 3 > /sys/class/gpio/
        unexport");
29    return 0;
30 }

```

---

*abre.sh:*

```

1 #!/bin/bash
2
3 GPIO_PATH=/sys/class/gpio
4
5 omxplayer -o local /home/pi/embarcados/
    projeto_final/sons/sim.mp3
6 echo 4 >> $GPIO_PATH/export
7 sudo echo out > $GPIO_PATH/gpio4/direction
8 sudo echo 1 > $GPIO_PATH/gpio4/value
9 sleep 3
10 echo 0 > $GPIO_PATH/gpio4/value
11 echo 4 >> $GPIO_PATH/unexport

```

---

*telegram.py:* (Código do servidor para comunicação com o bot do telegram)

```

1 import sys
2 import time
3 import telepot
4 from telepot.loop import MessageLoop
5
6 def handle(msg):
7     chat_id = msg['chat']['id']
8     command = msg['text']
9     print(command, chat_id)
10
11 #Inicio de conversa
12 if command == 'Ola':
13     bot.sendMessage(chat_id, 'Olá
        Mestre, o que você deseja?')
14 if command == 'oi':
15     bot.sendMessage(chat_id, 'Olá
        Mestre, o que você deseja?')
16 if command == 'ola':
17     bot.sendMessage(chat_id, 'Olá
        Mestre, o que você deseja?')
18 if command == 'Oi':
19     bot.sendMessage(chat_id, 'Olá
        Mestre, o que você deseja?')
20 # Cadastra novo usuario
21 if command == 'Cadastrar':
22     bot.sendMessage(chat_id, 'Informe
        o nome do usuario')
23
24 # Entrar
25 if command == 'Abre':
26     os.system('abre.sh')
27
28 if command == 'Negado':
29     os.system('negado.sh')
30
31 # Deletar usuario
32
33
34 bot = telepot.Bot('556806366:
    AAH9OIYZwwKapLkZrs7fYQU-NdUF2H9MuDc')
35 MessageLoop(bot, handle).run_as_thread()
36 print('lendo...')
37
38 # Keep the program running.
39 while 1:
40     time.sleep(10)

```

---

*facerec\_from\_web.py:* (Código do reconhecimento facial)

```

1
2 import face_recognition
3 import cv2
4
5 # This is a demo of running face
    recognition on live video from your
    webcam. It's a little more complicated
    than the
6 # other example, but it includes some
    basic performance tweaks to make

```

```

    things run a lot faster:
7  # 1. Process each video frame at 1/4
    resolution (though still display it at
    full resolution)
8  # 2. Only detect faces in every other
    frame of video.
9
10 # PLEASE NOTE: This example requires
    OpenCV (the 'cv2' library) to be
    installed only to read from your
    webcam.
11 # OpenCV is *not* required to use the
    face_recognition library. It's only
    required if you want to run this
12 # specific demo. If you have trouble
    installing it, try any of the other
    demos that don't require it instead.
13
14 # Get a reference to webcam #0 (the
    default one)
15 video_capture = cv2.VideoCapture(0)
16
17 # Load a sample picture and learn how to
    recognize it.
18 obama_image = face_recognition.
    load_image_file("obama.jpg")
19 obama_face_encoding = face_recognition.
    face_encodings(obama_image)[0]
20
21 # Load a sample picture and learn how to
    recognize it.
22 luana_image = face_recognition.
    load_image_file("luana.jpg")
23 luana_face_encoding = face_recognition.
    face_encodings(luana_image)[0]
24
25 # Load a sample picture and learn how to
    recognize it.
26 vitinho_image = face_recognition.
    load_image_file("vitorio.jpg")
27 vitinho_face_encoding = face_recognition.
    face_encodings(vitorio_image)[0]
28
29 # Load a second sample picture and learn
    how to recognize it.
30
31 # Create arrays of known face encodings
    and their names
32 known_face_encodings = [
33     obama_face_encoding,
34     luana_face_encoding,
35     vitinho_face_encoding
36 ]
37 known_face_names = [
38     "Barack_Obama",
39     "Luana",
40     "vitorio"
41 ]
42
43 # Initialize some variables
44 face_locations = []
45 face_encodings = []
46 face_names = []
47 process_this_frame = True
48
49 while True:
50     # Grab a single frame of video
51     ret, frame = video_capture.read()
52
53     # Resize frame of video to 1/4 size
    for faster face recognition
    processing
54     small_frame = cv2.resize(frame, (0, 0)
    , fx=0.25, fy=0.25)
55
56     # Convert the image from BGR color (
    which OpenCV uses) to RGB color (
    which face_recognition uses)
57     rgb_small_frame = small_frame[:, :,
    ::-1]
58
59     # Only process every other frame of
    video to save time
60     if process_this_frame:
61         # Find all the faces and face
        encodings in the current frame
        of video
62         face_locations = face_recognition.
        face_locations(rgb_small_frame
        )
63         face_encodings = face_recognition.
        face_encodings(rgb_small_frame
        , face_locations)
64
65         face_names = []
66         for face_encoding in
        face_encodings:
67             # See if the face is a match
            for the known face(s)
68             matches = face_recognition.
            compare_faces(
            known_face_encodings,
            face_encoding)
69             name = "Unknown"
70
71             # If a match was found in
            known_face_encodings, just
            use the first one.
72             if True in matches:
73                 first_match_index =
                    matches.index(True)
74                 name = known_face_names[
                    first_match_index]
75
76                 face_names.append(name)
77

```

```

78     process_this_frame = not
        process_this_frame
79
80
81     # Display the results
82     for (top, right, bottom, left), name
        in zip(face_locations, face_names)
        :
83         # Scale back up face locations
            since the frame we detected in
            was scaled to 1/4 size
84         top *= 4
85         right *= 4
86         bottom *= 4
87         left *= 4
88
89         # Draw a box around the face
90         cv2.rectangle(frame, (left, top),
            (right, bottom), (0, 0, 255),
            2)
91
92         # Draw a label with a name below
            the face
93         cv2.rectangle(frame, (left, bottom
            - 35), (right, bottom), (0,
            0, 255), cv2.FILLED)
94         font = cv2.FONT_HERSHEY_DUPLEX
95         cv2.putText(frame, name, (left +
            6, bottom - 6), font, 1.0,
            (255, 255, 255), 1)
96
97         # Display the resulting image
98         cv2.imshow('Video', frame)
99
100        # Hit 'q' on the keyboard to quit!
101        if cv2.waitKey(1) & 0xFF == ord('q'):
102            break
103
104    # Release handle to the webcam
105    video_capture.release()
106    cv2.destroyAllWindows()

```