# PONTO DE CONTROLE 5
# 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. Foi possivel controlar a porta remotamente, atravez do bot do telegram, e salvar o histórico de acesso dos usuários. O tempo de processamento foi elevado devido à capacidade limitada da Raspberry Pi 3.

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

## 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 abrir após o sistema reconhecer um cadastrado 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.

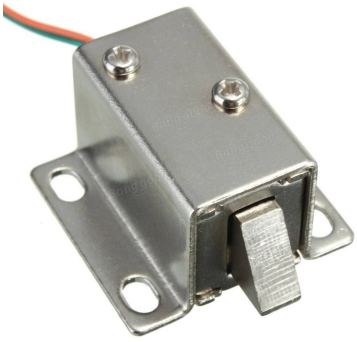
## 2. DESENVOLVIMENTO

### 2.1. Descrição do Hardware

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

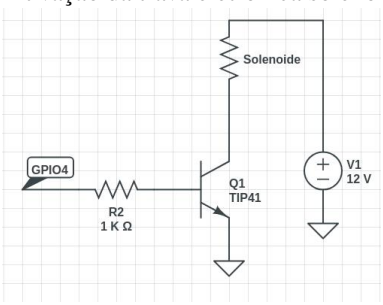
- Trava solenoide 12V (figura 1);
- Fonte DC 12V;
- Resistor de 1 KOhm;
- Transistor NPN (TIP41);
- Jumpers
- Protoboard
- Push-buttom
- Chave 3 pinos
- LED

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



Em uma placa universal perfurada foi montado o circuito da figura 2.

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




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

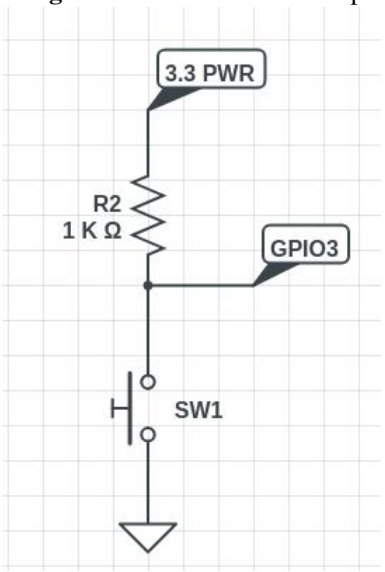
A trava solenoide mantém a porta fechada até que seja inserida uma tensão de 12V em seus terminais. Neste momento, o solenoide faz com que o ”dente” da trava seja retarda, liberando a abertura da porta. Ao retirar a tensão dos terminais, uma mola retorna a trava para a posição original, travando a porta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

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 é saída P2 da Raspberry Pi para reproduzir sons de confirmação ou negado de acesso.

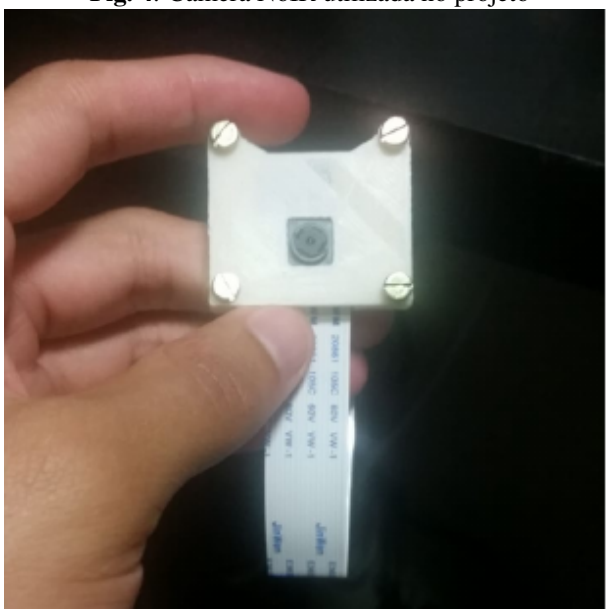
Para receber a requisição de acesso, foi montado um circuito com botão em modo Pull-Up, como mostra o esquemático da figura 3

**Fig. 3**. Botão em modo Pull-Up




Foi utilizada a câmera NoIR da Raspberry Pi, conectada por meio do cabo flat (figura 4).

**Fig. 4**. Câmera NoIR utilizada no projeto



Prevendo que um malfeitor poderia arrombar a porta, notou-se a necessidade de instalar uma chave de fim de curso nesta, para identificar se ela encontra-se aberta ou fechada.

Utilizando MDF, foi construído o protótipo da porta, e instalados todos os periféricos.

Com isso, a protoboard foi substituída por uma placa perfurada universal, com os componentes soldados.

## 2.2. Descrição do Software

### *2.2.1. Cliente - Servidor*

Foi criado um sistema cliente-servidor utilizando o protocolo TCP para efetuar a comunicação com o administrador de forma remota. O servidor foi instalado na Raspberry Pi presente na central de comando da porta.

O cliente envia os comandos pela rede, e o servidor os escreve no arquivo *msgs_admin.txt*, assim, o programa principal pode ler os comandos.

O servidor foi configurado para escutar pela porta 8080.

Quando o bot do telegram executa o cliente, este está na própria Raspberry Pi, logo, o IP de destino é definido como 127.0.0.1 (localhost). Ao executar o cliente pelo terminal de uma outra máquina, é necessário indicar o IP da Raspberry Pi.

Protótipo para execução do cliente: cliente [IP Raspberry Pi] 8080 [comando]

O sistema possui suporte a três comandos, todos ativados por apenas um caractere:

Liberar (l): Seta a flag *reconhecido*, para não haver alarme falso, e abre a porta, sem passar pela função de reconhecimento.

Desativar alarme (d): Zera a flag *alerta*, desativando o alarme.

Bloquear (b): Zera a flag *reconhecido*, bloqueando a entrada, independentemente do reconhecimento facial, até que seja enviado outro caractere qualquer.

Requisitar histórico (h): (Este comando é tratado diretamente no servidor, e não no programa principal) É retornado ao cliente a lista com as identificações dos usuários que entraram e os horários dos respectivos acessos.

### *2.2.2. Programa principal*

Foi criada uma função principal contendo todas as chamadas necessárias para a execução do sistema. No programa, são criados três processos filhos, mostrados na figura 5.

Filho 1: Executa as rotinas de verificação e controle da porta, tais como: verificação da campainha, verificação do estado da porta, ativação do alarme (caso a porta seja aberta sem permissão).

Filho 2: Executa o servidor.

Filho 3: Executa o bot.

No filho 1, são criadas threads para cada elemento, pois todos precisam ser verificados simultaneamente.

A thread da campainha, cujo funcionamento é mostrado na figura 6, é responsável por verificar mudanças no estado do botão (através da função poll), iniciar a rotina de verificação, e decidir se a porta pode ser aberta ou não. Caso o acesso seja negado, é dada a opção de cadastro. A rotina de reconhecimento sai é acionada com a porta fechada e quando o alerta de invasão está desativado.

**Fig. 5**. Visão geral

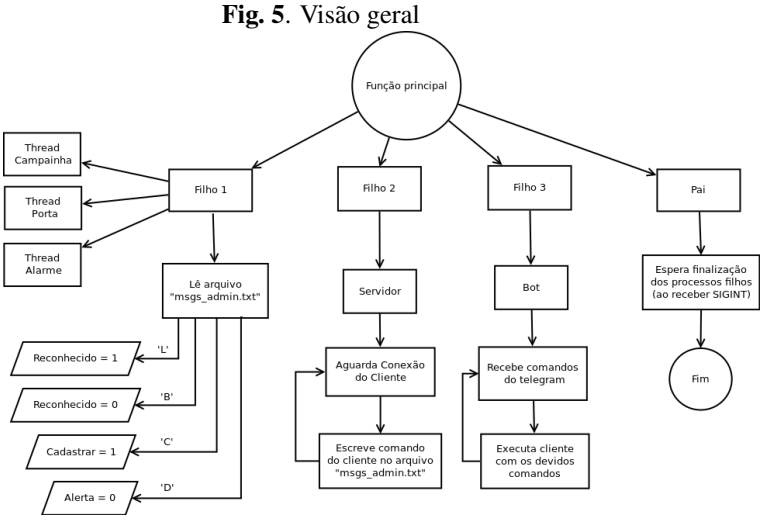
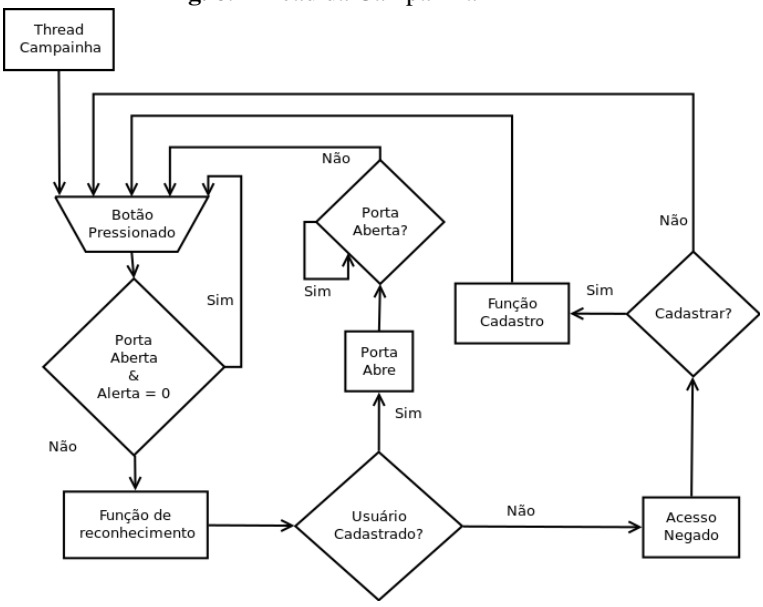



**Fig. 6**. Thread da Campainha

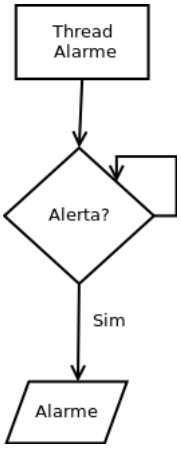



A thread da porta, explicada pelo diagrama da figura 7 é responsável por verificar se a porta encontra-se aberta ou fechada. Como sistema de segurança, se a porta estiver aberta com a flag *reconhecido* = 0, é emitido um alerta, indicando uma invasão.

**Fig. 7**. Thread da Porta


A thread do alarme é responsavel apenas por manter o alarme sonoro ligado, caso a flag *alerta* esteja setada, como mostra a figura 8.

**Fig. 8**. Thread do Alarme




O cadastro, mostrado na figura 9 é responsável pela inclusão do arquivo da foto e a inserção do nome do usuário no arquivo de cadastro, mas isso só pode acontecer se o programa reconhecer que tem um rosto na foto tirada pela sistema, caso isso não aconteça o sistema tira outra foto até ele encontrar um rosto.

**Fig. 9**. Cadastro

*2.2.3. Reconhecimento*

No código do reconhecimento,exemplificado na figura 10é necessário fazer a verificação das imagens no banco de dados e estamos utilizando a semelhança de 80 por centos para ele identificar que a pessoa existe no cadastro.O cadastro é realizado via bot e são permitidos apenas 3 rosto cadastrados.

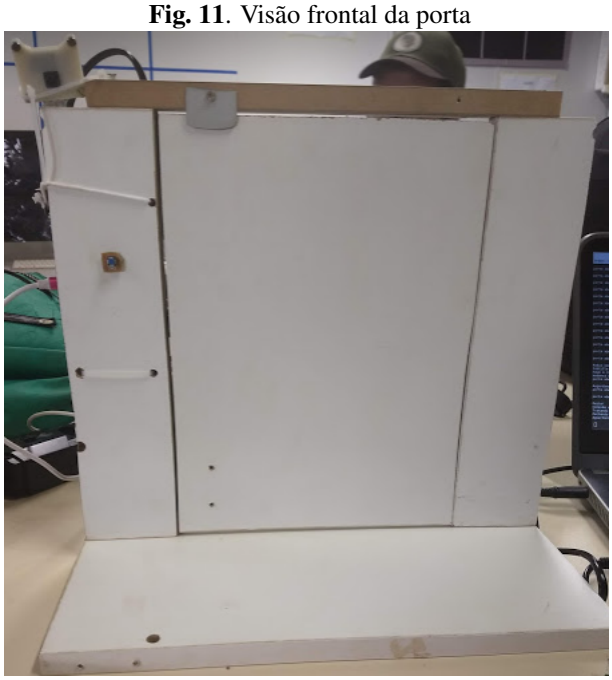
**Fig. 10**. Reconhecimento


Houve o desenvolvimento de um bot no telegram que irar servir de interface gráfica para o usuário, este será responsável pelo cadastramento de pessoas, abertura da porta sem verificação e consulta de histórico de acesso.

## 3. RESULTADOS

O conjunto montado ficou como mostrado nas figuras 11 e 12.

**Fig. 11**. Visão frontal da porta



Foi possível comunicar o sistema com o cliente com sucesso, assim como executar os comandos recebidos, como mostram as figuras 13 e 14.

**Fig. 12**. Visão traseira da porta

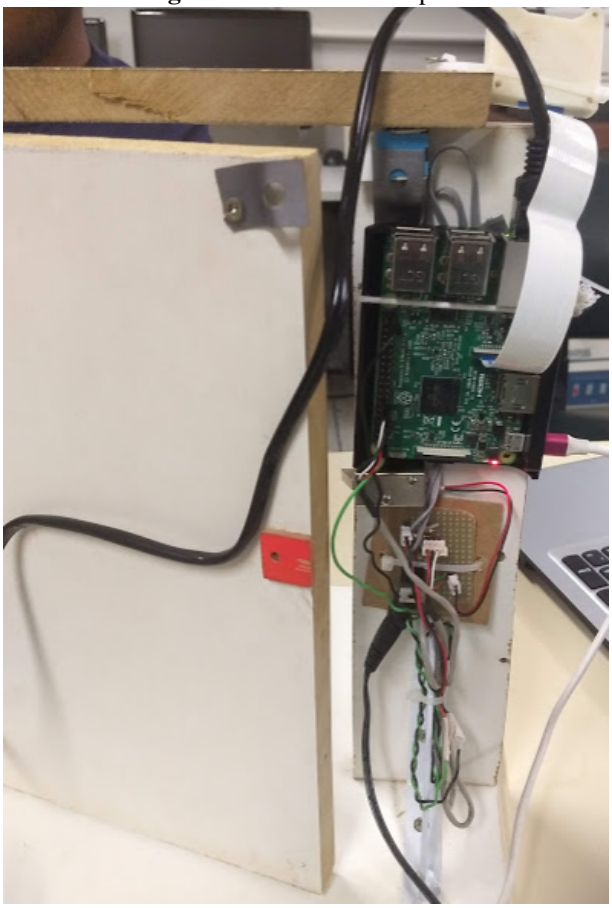
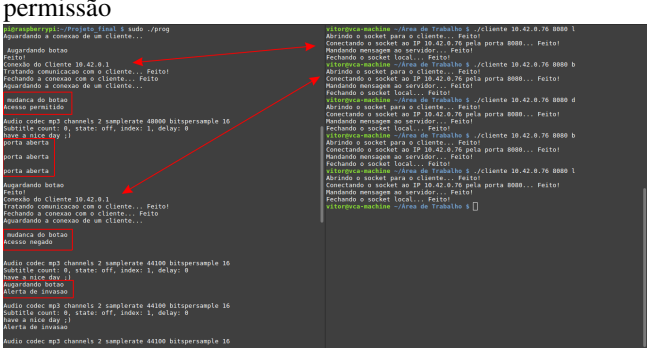
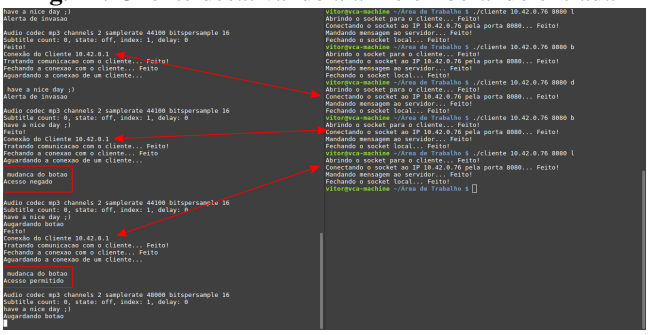


**Fig. 13**. Cliente liberando porta, porta abrindo com e sem permissão



Aqui, o primeiro comando do cliente é para liberar a entrada. Pode-se observar que o programa só passa a obedecer novamente quando a porta é fechada. Na segunda instrução, o cliente pede que o acesso seja bloqueado. E então ao pressionar o botão, o sistema informa "Acesso negado". Após isso foi simulada a abertura da porta por arrombamento, sem que o usuário fosse reconhecido. Imediatamente o alerta de invasão foi ativado.

**Fig. 14**. Cliente desativando alarme e liberando entrada



Aqui, a primeira instrução do cliente é para desativar o alarme. Ocorrem atrasos até que o alarme seja completamente desligado. Na segunda instrução, o cliente bloqueia o acesso e com isso também ativa o alarme. Ao pressionar o botão, o acesso é negado. Então o cliente envia o comando para liberar a porta, e assim ao pressionar o botão o acesso é liberado.

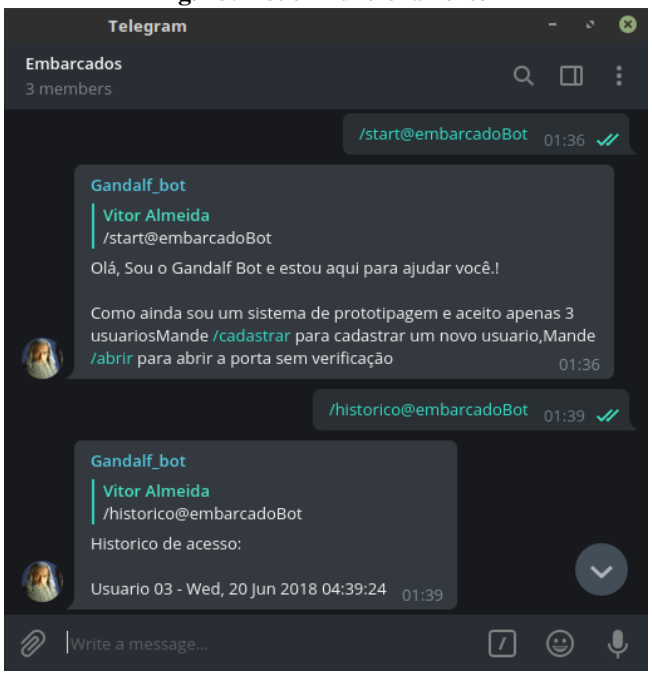
Como pode ser visto no apêndice temos o código do reconhecimento ,face reco.py, esse código é desenvolvido em python 2 e nele podemos ver que toda vez que ele roda temos a verificação de 3 imagens, que é os usuários cadastrados, quando ele realiza a comparação entre o banco de dados com a foto tirada ele retorna true e false. Assim, escrevemos em um arquivo o resultado dessa comparação, porém precisamos saber qual usuário entrou para conseguir montar o banco de dados, assim escrevemos em outro arquivo essa informação. O reconhecimento é baseado na biblioteca fornecida pela comunidade do python[8], ela é desenvolvida em cima do método eigenface e utilizando uma rede neural dlib[9], como é uma biblioteca aberta podemos ver a modelagem e conseguir diminuir alguns recursos para deixar import mais rápido, entretrando para importar está é tem um tempo considerável, sendo o principal fator da demora do reconhecimento, pois em média o tempo de importação na Raspberry pi 3 é 8 segundos.

Analisando o bot do telegram ele foi desenvolvido para servir de cliente, e como não queríamos desenvolver todo o cliente em python dividimos o cliente em dois, o primeiro é o bot do telegram e o segundo é propriamente o cliente. O bot ele apenas executa o programa cliente com os parâmetros enviados no bot. Em suma, o bot pode ser considerado apenas com um controlado do cliente verdadeiro. Isso fica mais claro na hora que vemos os comandos que são executados quando pedimos para abrir a porta, no bot teremos a seguinte instrução: os.system('sudo ./cliente ip 8080 l'), como pode ser visto estamos executando o cliente - desenvolvido em c- dentro da programação do bot. Para cada operação temos flags que são enviadas ao cliente, por exemplo: a flag

d é para desativar o alarme e a flag h para requisitar o histórico.

A figura 15 mostra um exemplo do Bot em funcionamento quando é requisitado o histórico de acesso.

**Fig. 15**. Bot em funcionamento




## 4. DISCUSSÃO E CONCLUSÕES

A arquitetura multi-thread permitiu que os periféricos fossem controlados simultaneamente. Isso é fundamental para o projeto, tanto do ponto de vista de experiência do usuário, que não precisa esperar o término de alguma rotina para que a função de interesse seja executada, quanto para a segurança do sistema, visto que há uma vigilância permanente do estado da porta.

Um dos problemas que a dupla teve durante o desenvolvimento do software, foi o compartilhamento de variáveis entre as threads e processos pai e filho. Inicialmente a alteração das variáveis utilizadas como flags foi feita no processo pai, e as threads criadas no processo filho realizava as leituras. Porém isso não funcionou, porque os processos não compartilham valores de variáveis, apenas suas declarações. Esse problema foi resolvido realizando todas as operações com variáveis flags dentro do mesmo processo.

Para o encerramento do programa via comando CTRL+C, foi necessário utilizar a captura do sinal SIGINT e encaminhar para uma função de encerramento. Esta realiza o cancelamento das threads e o encerramento do programa.

O servidor é um programa separado, e não uma função e nem um processo filho. Logo, para o código principal receber comandos através do servidor, foi necessário utilizar métodos de escrita em arquivo para comunicar os dois processos.

O processo de reconhecimento foi relativamente reduzido diminuindo o tamanho das imagens, tanto as do banco de dados, quando a de requisição de acesso. Isso aconteceu porque o algoritmo de comparação precisa passar por menos pixels, reduzindo o tempo. Além disso a importação da biblioteca demora um certo tempo, pois ela é bem grande para realizar todos os processos necessários para conhecimento.

O algoritmo precisa realizar a comparação da foto de requisição com todas as fotos cadastradas. Sendo assim, o tempo de processamento também foi um fator limitante no número de usuários cadastrados, então foram cadastrados apenas 3. Isso levando em conta do poder computacional na Raspiberry Pi 3 , se esse programa for rodado em um computador ou um servidor mais poderoso computacionalmente pode-se adicionar mais usuários que o programa irá responder.

Ao utilizar o telgram ganhamos em relação a UX - experiencia do usuário-, pois por ser um aplicativo altamente distribuído e sendo necessário apenas ser adicionado ao um grupo para controla a porta traz uma respectividade muito alta. Analisando o comportamento do bot do telegram temos ele operando como o esperado realizando os comandos do cliente em C a partir dos comandos enviados.

O projeto funciona até um certo momento que ele passa a ter um algum erro , que não foi encontrado , que libera acesso de qualquer pessoa e ate mesmo se não for idetificado uma rosto. A solução disso é simplesmente dar reboot na raspberry e assim o sistema volta a funcionar sem erros.

Uma limitação do nosso projeto 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. É também o bot do telegram não receber aviso se aconteceu uma invasão no 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.mc

**[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

**[9** ] http://dlib.net/

**[10** ] http://eyalarubas.com/face-detection-and-recognition.html

## 6. APENDICE

Códigos utilizados

Função principal: *main.c*

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <fcntl.h>
4  #include <sys/poll.h>
5  #include <unistd.h>
6  #include <pthread.h>
7  #include <signal.h>
8  #include <wiringPi.h>
9  #include <sys/wait.h>
10 #include <string.h>
11
12 #include "bib_arqs.h"
13 #include"funcoes.h"
14
15 int fim_curso = 2;
16 //int campainha = 0;
17 int alarme = 3;
18 int cadastrar = 0;
19
20 int porta_aberta = 0; //sensor fim de curso
       na porta
21 int reconhecido = 0; //sinal
22 int alerta = 0; //alarme de invasao
23 int child_pid;
24
25 int encerrar=0;
26
27
28 pthread_t id_campainha;
29 pthread_t id_alarme;
30 pthread_t id_porta;
31
32 void encerra_prog(int sig);
33 void encerra_threads(int sig);
34 void* thread_campainha(void*arg);
35 void* thread_alarme(void*arg);
36 void* thread_porta(void*arg);
37
38
39 int main(int argc, char const *argv[]) {
40
41 //  reconhecido = (char) *argv[1];
42 //  printf("%c\n",reconhecido );
43   wiringPiSetup();
44   pinMode(fim_curso, INPUT);
45   pinMode(alarme, OUTPUT);
46 //  pinMode(campainha, IN);
47
48
49
50   if (fork() == 0){
51       child_pid = getpid();
52       signal(SIGINT,encerra_threads); //
         direcionando sinal de interrupcao
         (CTRL+C)
53
54       pthread_create(&id_campainha,NULL,&
             thread_campainha,NULL); //criando
             thread para Campainha
55       pthread_create(&id_alarme,NULL,&
             thread_alarme,NULL); //criando
             thread para Campainha
56       pthread_create(&id_porta,NULL,&
             thread_porta,NULL); //criando
             thread para Campainha
57         int a;
58
59       char comando;
60       while (!encerrar) {
61         a = open("msgs_admin.txt", O_RDONLY)
               ;
62         read(a,&comando,1);
63         close(a);
64 //        le_arq_texto("msgs_admin.txt",
       comando);
65 //          printf("%c\n", comando);
66         sleep(1);
67         if (comando=='d'){
68       //     printf("admin quer desativar\n")
               ;
69           alerta = 0;
70           char aux = 'a';
71           a = open("msgs_admin.txt", O_RDWR)
                 ;
72                 write(a,&aux,1);
73                 close(a);
74         }
75
76         if (comando=='1'){
77           reconhecido = 1;
78           system("sudo ./abre.sh");
79
80           while(porta_aberta==1 && !encerrar
                 ){
81               printf("porta aberta\n\n");
82               sleep(1);
83           } //espera porta fechar
84           reconhecido = 0;
85
86 //          printf("Acesso liberado,
       aguardando campainha. status reconhecido
        = %c\n",reconhecido);
87   //        sleep(1);
88           char aux = 'a';
89           a = open("msgs_admin.txt", O_RDWR)
                 ;
90                 write(a,&aux,1);
91                 close(a);
92         }
93         if (comando=='b'){
94           reconhecido = 0;
```

```
95            }
96
97
98
99          }
100
101     }
102
103     if (fork()==0){//filho 2
104         system("./servidor 8080"); // Executa
                o servidor
105     }
106    if (fork()==0){//filho 3
107         system("sudo python bot.py"); //
                Executa o bot
108     }
109
110     signal(SIGINT,encerra_prog); //
            direcionando sinal de interrupcao (
            CTRL+C)
111
112
113
114     while (!encerrar) {
115
116
117
118     }
119
120     wait(NULL);
121     wait(NULL);
122
123     return 0;
124 }
125 void encerra_prog(int sig){
126     encerrar = 1;
127
128 }
129 void encerra_threads(int sig){
130     alerta = 0;
131
132     puts("Encerrando...");
133     encerrar = 1;
134     if (pthread_cancel(id_campainha) ==-1){
135       puts("tread da campainha nao foi
              cancelada");
136     }
137     if (pthread_cancel(id_alarme) ==-1){
138       puts("tread do alarme nao foi cancelada"
              );
139     }
140
141     if (pthread_cancel(id_porta) ==-1){
142       puts("tread da port nao foi cancelada");
143     }
144 printf("threads canceladas\n" );
145     pthread_join(id_campainha,NULL);
146 //    pthread_join(id_alarme,NULL);
147     pthread_join(id_porta,NULL);
148
149     system("gpio unexportall");
150    puts("Programa encerrado pelo administrador
          !");
151
152     exit(1);
153
154 }
155
156 void* thread_campainha(void*arg){
157 pthread_setcancelstate(PTHREAD_CANCEL_ENABLE
      , NULL);
158
159     while (!encerrar) {
160
161       poll_bot();
162       if(porta_aberta == 0 && alerta== 0) { //
              so inicia reconhecimento se a porta
              estiver fechada
163         //reconhecido = funcao de
                reconhecimento;
164          // tirando foto para reconhecimento
165         char aux [100] = "raspistill -w 640 -h
              480 -q 75 -o ./imagens/unknown.
              jpg";
166         system(aux); //tira a foto
167
168         printf("Foto foi tirada");
169
170         system("sudo ./face.sh");
171         FILE *verifica;
172         verifica = fopen("lista_verificacao.
              txt","r");
173         char ch [20];
174         char verdade [20] = "[True]";
175
176          //testando se o arquivo foi realmente
                aberto
177         if(verifica == NULL)
178         {
179         printf("Erro na abertura do arquivo!")
              ;
180         }
181         else
182         {
183                 while( (fgets(ch,20,verifica))
                      != NULL )
184             {
185               //printf("%s",ch);
186                       }
187               fclose(verifica);
188
189         }
190
191         // Comparando string  com arquivo
192         if (strcmp (verdade,ch) == 0 )
193         {
```

```
194           reconhecido  = 1;
195           printf("Usuario reconhecido! \n\n");
196         }
197         else
198         {
199           reconhecido = 0;
200           printf("Usuario nao reconhecido\n\n"
                );
201         }
202
203         if (reconhecido == 1){ //se o usuario
               for cadastrado
204           printf("Acesso permitido\n\n");
205           system("sudo ./abre.sh");
206
207
208           while(porta_aberta==1 && !encerrar){
209               printf("porta aberta\n\n");
210               sleep(1);
211           } //espera porta fechar
212           reconhecido = 0;
213         }
214         else{ //usuario nao cadastrado
215           puts("Acesso negado\n\n");
216           system("sudo ./negado.sh");
217            if (cadastrar){
218             cadastro("nome");
219           }
220
221         }
222
223       }
224     }
225     pthread_exit(0);
226
227   }
228
229   void* thread_alarme(void*arg){
230     while(!encerrar){
231         if(alerta == 1 && !encerrar){
232           /*  digitalWrite(alarme,HIGH);
233             sleep(1);
234             digitalWrite(alarme,LOW);
235             sleep(1); */
236             printf("Alerta de invasao\n\n");
237             system("sudo ./alarme.sh");
238   /*          puts("--------------------");
239             puts("CTRL+C para sair");
240
241             sleep(3);
242             puts(" ");
243             puts("ESPERE");
244             puts("--------------------"); */
245             if (encerrar ==1){
246               pthread_exit(0);
247             }
248
249         } //espera administrador desativar
250           alarme;
251       }
252     pthread_exit(0);
253   }
254   void* thread_porta(void*arg){
255     pthread_setcancelstate(
          PTHREAD_CANCEL_ENABLE, NULL);
256     while(!encerrar){
257   //printf("alerta = %d",alerta);
258       porta_aberta = digitalRead(fim_curso);
259    //    printf("porta aberta = %d",
           porta_aberta);
260       sleep(1);
261       if(porta_aberta == 1 && reconhecido ==
            0){
262         alerta = 1;
263         while (alerta == 1 && !encerrar);
264       }
265     }
266     pthread_exit(0);
267
268   }
```

---

*servidor.c*

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <unistd.h>
4   #include <arpa/inet.h>
5   #include <string.h>
6   #include <signal.h>
7   #include <sys/socket.h>
8   #include <sys/un.h>
9   #include "bib_arqs.h"
10  #include "rdwr_arq.h"
11
12  int socket_id;
13  void sigint_handler(int signum);
14  void print_client_message(int client_socket)
        ;
15  void end_server(void);
16
17  int main(int argc, char* const argv[]){
18
19          unsigned short servidorPorta;
20          struct sockaddr_in servidorAddr;
21
22          if (argc < 2)
23          {
24                  puts("   Este programa cria
                        um servidor TCP/IP ");
25                  puts("   conectado a porta
                        especificada pelo
                        usuario.");
26                  puts("   Para permitir que o
                        cliente comunique-se");
27                  puts("   com este servidor, 
                        o servidor deve ser");
```

```
28                  puts("␣␣␣executado␣
                        inicialmente␣com␣uma␣
                        porta␣definida,");
29                  puts("␣␣␣e␣o␣cliente␣devera␣
                        ser␣executado␣em␣outra")
                        ;
30                  puts("␣␣␣janela␣ou␣em␣outra␣
                        aba␣do␣terminal,␣
                        utilizando");
31                  puts("␣␣␣a␣mesma␣porta.␣O␣
                        servidor␣escreve␣na␣tela
                        ");
32                  puts("␣␣␣todo␣texto␣enviado␣
                        pelo␣cliente.␣Se␣o␣
                        cliente");
33                  puts("␣␣␣transmitir␣o␣texto␣
                        \"sair\",␣o␣servidor␣se"
                        );
34                  puts("␣␣␣encerra.␣Se␣o␣
                        usuario␣pressionar␣CTRL-
                        C,");
35                  puts("␣␣␣o␣servidor␣tambem␣
                        se␣encerra.");
36                  puts("␣␣␣Modo␣de␣Uso:");
37                  printf("␣␣␣␣␣␣%s␣<Numero␣da␣
                        porta>\n", argv[0]);
38                  printf("␣␣␣Exemplo:␣%s␣8080\
                        n", argv[0]);
39                  exit(1);
40          }
41
42          servidorPorta = atoi(argv[1]);
43
44  //      fprintf(stderr, "Definindo o
        tratamento de SIGINT... ");
45          signal(SIGINT, sigint_handler);
46  //      fprintf(stderr, "Feito!\n");
47
48  //      fprintf(stderr, "Abrindo o socket
        local... ");
49          socket_id = socket(PF_INET,
                SOCK_STREAM, IPPROTO_TCP);
50          if(socket_id < 0)
51          {
52                  fprintf(stderr, "Erro␣na␣
                        criacao␣do␣socket!\n");
53                  exit(0);
54          }
55  //      fprintf(stderr, "Feito!\n");
56
57  //      fprintf(stderr, "Ligando o socket a
        porta %d... ", servidorPorta);
58          memset(&servidorAddr, 0, sizeof(
                servidorAddr)); // Zerando a
                estrutura de dados
59          servidorAddr.sin_family = AF_INET;
60          servidorAddr.sin_addr.s_addr = htonl
                (INADDR_ANY);
61          servidorAddr.sin_port = htons(
                servidorPorta);
62          if(bind(socket_id, (struct sockaddr
                *) &servidorAddr, sizeof(
                servidorAddr)) < 0)
63          {
64                  fprintf(stderr, "Erro␣na␣
                        ligacao!\n");
65                  exit(0);
66          }
67  //      fprintf(stderr, "Feito!\n");
68
69  //      fprintf(stderr, "Tornando o socket
        passivo (para virar um servidor)... ");
70          if(listen(socket_id, 10) < 0)
71          {
72                  fprintf(stderr, "Erro!\n");
73                  exit(0);
74          }
75  //      fprintf(stderr, "Feito!\n");
76
77          while(1)
78          {
79                  int socketCliente;
80                  struct sockaddr_in
                        clienteAddr;
81                  unsigned int clienteLength;
82
83                  fprintf(stderr, "Aguardando␣
                        a␣conexao␣de␣um␣cliente
                        ...\n\n␣");
84                  clienteLength = sizeof(
                        clienteAddr);
85                  if((socketCliente = accept(
                        socket_id, (struct
                        sockaddr *) &clienteAddr
                        , &clienteLength)) < 0)
86                          fprintf(stderr, "
                                Falha␣no␣accept
                                ().\n");
87                  fprintf(stderr, "Feito!\n");
88
89                  fprintf(stderr, "Conexao␣do␣
                        Cliente␣%s\n", inet_ntoa
                        (clienteAddr.sin_addr));
90
91                  fprintf(stderr, "Tratando␣
                        comunicacao␣com␣o␣
                        cliente...␣");
92                  print_client_message(
                        socketCliente);
93                  fprintf(stderr, "Feito!\n");
94
95                  fprintf(stderr, "Fechando␣a␣
                        conexao␣com␣o␣cliente...
                        ␣");
96                  close(socketCliente);
97                  fprintf(stderr, "Feito\n");
```

```
        }
        return 0;
}

void sigint_handler(int signum)
{
        fprintf(stderr, "\nRecebido o sinal CTRL+C... vamos desligar o servidor!\n");
        end_server();
}

void print_client_message(int client_socket)
{
  FILE *arq;
  arq = fopen("msgs_admin.txt", "wb");

        int length;
//      char* text;
        char text;

//      fprintf(stderr, "\nMensagem enviada pelo cliente tem ");
        read(client_socket, &length, sizeof(length));
//      fprintf(stderr, "%d bytes.", length);
//      text = (char*) malloc (length);
        read(client_socket, &text, 1);
//      fprintf(stderr,"\n\n   Mensagem = %s \n\n", text);
  putc(text,arq); //Escreve no arquivo de transicao;
  fclose(arq);
        if (text=='s')
        {
//              free (text);
                fprintf(stderr, "Cliente pediu para o servidor fechar.\n");
                end_server();
        }
        if (text == 'h'){ // requisitando historico
                char historico[1000];
//              char buf[100];
//              le_arq_texto("historico.txt",buf);
//              system("sudo rm historico.txt");
//              system("touch historico.txt");
//              rdwr_arq(buf);
                le_arq_texto("historico.txt",historico);
                send(client_socket, historico, 1000, 0);

        }
//      free (text);
}

void end_server(void)
{
        fprintf(stderr, "Fechando o socket local... ");
        close(socket_id);
        fprintf(stderr, "Feito!\n");
        exit(0);
}
```

*abre.sh*:

```
#!/bin/bash

GPIO_PATH=/sys/class/gpio

omxplayer -o local /home/pi/embarcados/projeto_final/sons/sim.mp3
echo 4 >> $GPIO_PATH/export
sudo echo out > $GPIO_PATH/gpio4/direction
sudo echo 1 > $GPIO_PATH/gpio4/value
sleep 3
echo 0 > $GPIO_PATH/gpio4/value
echo 4 >> $GPIO_PATH/unexport
```

*negado.sh*:

```
#!/bin/bash
omxplayer -o local ./sons/nao.mp3
```

*alarme.sh*:

```
#!/bin/bash
omxplayer -o local ./sons/alarme.mp3
```

*poll_bot.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/poll.h>
#include <unistd.h>

#include "funcoes.h"


int poll_bot()
{
        struct pollfd pfd;
        char buffer;
        system("echo 17 > /sys/class/gpio/export");
        system("echo falling > /sys/class/gpio/gpio17/edge");
```

```
16          system("echo in > /sys/class/gpio/
                gpio17/direction");
17          pfd.fd = open("/sys/class/gpio/
                gpio17/value", O_RDONLY);
18          if(pfd.fd < 0)
19          {
20                  puts("Erro abrindo /sys/
                        class/gpio/gpio17/
                        value");
21                  puts("Execute este
                        programa como root");
22                  return -1;
23          }
24          read(pfd.fd, &buffer, 1);
25          pfd.events = POLLPRI | POLLERR;
26          pfd.revents = 0;
27          puts("Augardando botao");
28          poll(&pfd, 1, -1);
29          if(pfd.revents) puts("mudanca do
                botao");
30          usleep(500000);
31          close(pfd.fd);
32          system("echo 17 > /sys/class/gpio/
                unexport");
33          return 0;
34  }
```

Obs: *poll_fim_curso.c* é identico ao *poll_bot.c*, apenas trocando a GPIO 17 para a 27

---

*negado.sh*:

```
1  #!/bin/bash
2  omxplayer -o local ./sons/nao.mp3
```

---

*alarme.sh*:

```
1  #!/bin/bash
2  omxplayer -o local ./sons/alarme.mp3
```

---

*face_reco.py*

```
1  # -*- coding: utf-8 -*-
2  import face_recognition
3  import time
4  #from datetime import datetime
5  #from time import ctime
6
7  arq = open('lista_verificacao.txt','w')
8  his = open('historico.txt','a')
9
10 compare = '[True]'
11 f = '[False]'
12
13 # Banco de dados
14 known_image_01 = face_recognition.
       load_image_file("imagens/01.jpg")
15 known_image_02 = face_recognition.
       load_image_file("imagens/02.jpg")
16 known_image_03 = face_recognition.
       load_image_file("imagens/03.jpg")
17
18 # Foto tirada
19 unknown_image = face_recognition.
       load_image_file("imagens/unknown.jpg")
20
21 # Codificacao do Banco de Dados
22 biden_encoding_01 = face_recognition.
       face_encodings(known_image_01)[0]
23 biden_encoding_02 = face_recognition.
       face_encodings(known_image_02)[0]
24 biden_encoding_03 = face_recognition.
       face_encodings(known_image_03)[0]
25
26 # Codificacao foto tirada
27 try:
28      unknown_encoding = face_recognition.
            face_encodings(unknown_image)[0]
29 except Exception:
30      arq.write(f)
31
32
33 results_01 = face_recognition.
       compare_faces([biden_encoding_01],
       unknown_encoding)
34 results_02 = face_recognition.
       compare_faces([biden_encoding_02],
       unknown_encoding)
35 results_03 = face_recognition.
       compare_faces([biden_encoding_03],
       unknown_encoding)
36
37
38
39 if compare == str((results_01)):
40      arq.write(compare)
41      now = time.strftime("%c")
42      his.write('\nUsuario 01 - ' + now)
43 elif compare == str((results_02)):
44      arq.write(compare)
45      now = time.strftime("%c")
46 #      aux = datetime.now().strftime('%Y-%m
       -%d %H:%M:%S\n')
47      his.write('\nUsuario 02 - ' + now)
48 elif compare == str((results_03)):
49      arq.write(compare)
50      now = time.strftime("%c")
51      his.write('\nUsuario 03 - ' + now)
52 else:
53      arq.write(f)
54
55 his.close()
56 arq.close()
```

---

*bot.py*:

```
1  # -*- coding: utf-8 -*-
2  import telegram
```

```
3  from telegram.ext import Updater,
       CommandHandler
4  import sys
5  import os
6
7  # Define a few command handlers. These
       usually take the two arguments bot and
8  # update. Error handlers also receive the
       raised TelegramError object in error.
9  def start(bot, update):
10     """Manda␣a␣mensagem␣quando␣o␣comando␣/
           start␣e␣enviado."""
11     update.message.reply_text(
12         'Ola, Sou o Gandalf Bot e estou
               aqui para ajudar voce.!\n\n'
13         'Como ainda sou um sistema de
               prototipagem e aceito apenas 3
                usuarios'
14         'Mande /cadastrar para cadastrar
               um novo usuario,'
15         'Mande /abrir para abrir a porta
               sem verificacao')
16
17
18 def cadastrar(bot, update):
19     """Manda␣a␣mensagem␣quando␣o␣comando␣/
           cadastrar␣e␣enviado."""
20
21     update.message.reply_text(
22         'Ok, Como ja falei eu tenho apenas
                3 espacos para cadastro.
               Entao informe em qual espaco
               deseja:\n'
23         'Mande para mim o nome do usuario
                da seguinte forma:\n/primeiro
                Fulano \n'
24         'Agora agora irei tirar a foto,
               por favor sem oculos ou bone
               ')
25
26 def primeiro(bot,update):
27     """␣Cadastrando␣o␣primeiro␣usuario␣"""
28
29     usuario = update.message.text
30     FILE = 'lista_cadastrados.txt'
31     resposta_file = open(FILE,'a')
32     resposta_file.write(usuario + "\n")
33
34     aux = ('raspistill -w 640 -h 480 -q 75
           -o ./imagens/01.jpg')
35     os.system (aux)
36
37     update.message.reply_text(
38         'Nome registado\n\n'
39         'Foto Tirada, se desejar ver a
               foto manda /verificarPrimeiro
               ')
40
41 def segundo(bot,update):
42     """␣Cadastrando␣o␣primeiro␣usuario␣"""
43
44     usuario = update.message.text
45     FILE = 'lista_cadastrados.txt'
46     resposta_file = open(FILE,'a')
47     resposta_file.write(usuario + "\n")
48
49     aux = ('raspistill -w 640 -h 480 -q 75
           -o ./imagens/02.jpg')
50     os.system (aux)
51
52     update.message.reply_text(
53         'Nome registado\n\n'
54         'Foto Tirada, se desejar ver a
               foto manda /verificarSegundo')
55
56 def terceiro(bot,update):
57     """␣Cadastrando␣o␣primeiro␣usuario␣"""
58
59     usuario = update.message.text
60     FILE = 'lista_cadastrados.txt'
61     resposta_file = open(FILE,'a')
62     resposta_file.write(usuario + "\n")
63
64     aux = ('raspistill -w 640 -h 480 -q 75
           -o ./imagens/03.jpg')
65     os.system (aux)
66
67     update.message.reply_text(
68         'Nome registado\n\n'
69         'Foto Tirada, se desejar ver a
               foto manda /verificarTerceiro
               ')
70
71 def verificaPrimeiro(bot,update):
72     """␣Manda␣a␣foto␣que␣foi␣cadastrada"""
73     photo = open('/imagens/01.jpg','rb')
74     update.sendPhoto(chat_id,photo)
75
76     update.message.reply_text(
77         'Foto do ultimo cadastro 01 foi
               enviada\n'
78     )
79
80 def verificaSegundo(bot,update):
81     """␣Manda␣a␣foto␣que␣foi␣cadastrada"""
82     photo = open('/imagens/02.jpg','rb')
83     update.sendPhoto(chat_id,photo)
84
85     update.message.reply_text(
86         'Foto do ultimo cadastro 02 foi
               enviada\n'
87     )
88 def verificaTerceiro(bot,update):
89     """␣Manda␣a␣foto␣que␣foi␣cadastrada"""
90     update.sendPhoto(chat_id ,photo =open
           ('imagens/03.jpg','rb'))
```

```
91
92      update.message.reply_text(
93          'Foto do ultimo cadastro 03 foi
                enviada\n'
94      )
95
96  def ultimo(bot,update):
97      """ Manda a foto que foi cadastrada"""
98      update.send_photo(chat_id =chat_id,
            photo =open('imagens/unknown.jpg
            ','rb'))
99
100     update.message.reply_text(
101         'Foto do ultima tentativa de log
                foi enviada\n'
102     )
103
104
105 def abrir(bot,update):
106     aux = ('sudo ./Cliente/cliente
            127.0.0.1 8080 1')
107     os.system(aux)
108     update.message.reply_text(
109         'Porta Aberta\n'
110     )
111
112 def historico(bot,update):
113     aux = ('sudo ./Cliente/cliente
            127.0.0.1 8080 h')
114     os.system(aux)
115     hist = open('./Cliente/historico.txt
            ','rb')
116     aux = hist.read()
117     hist.close()
118     update.message.reply_text(
119         'Historico de acesso:\n\n' + aux
120     )
121
122 def main():
123     """Start the bot."""
124     # Create the EventHandler and pass it
            your bot's token.
125     updater = Updater("Token")
126
127     print('Lendo...')
128     # Get the dispatcher to register
            handlers
129     dp = updater.dispatcher
130
131     # on different commands - answer in
            Telegram
132     dp.add_handler(CommandHandler("start",
             start))
133     dp.add_handler(CommandHandler("
            cadastrar", cadastrar))
134     dp.add_handler(CommandHandler("
            primeiro", primeiro))
135     dp.add_handler(CommandHandler("ultimo"
            , ultimo))
136     dp.add_handler(CommandHandler("
            verificaTerceiro",verificaTerceiro
            ))
137     dp.add_handler(CommandHandler("
            verificaSegundo",verificaSegundo))
138     dp.add_handler(CommandHandler("
            verificaPrimeiro",verificaPrimeiro
            ))
139     dp.add_handler(CommandHandler("segundo
            ",segundo))
140     dp.add_handler(CommandHandler("
            terceiro",terceiro))
141     dp.add_handler(CommandHandler("abrir",
             abrir))
142     dp.add_handler(CommandHandler("
            historico", historico))
143
144     # Start the Bot
145     updater.start_polling()
146
147     # Run the bot until you press Ctrl-C
            or the process receives SIGINT,
148     # SIGTERM or SIGABRT. This should be
            used most of the time, since
149     # start_polling() is non-blocking and
            will stop the bot gracefully.
150     updater.idle()
151
152
153 if __name__ == '__main__':
154     main()
```

*cadastro.c*:

```
1  #include<stdio.h>
2  #include <stdlib.h>
3  #include <fcntl.h>
4  #include <sys/poll.h>
5  #include <unistd.h>
6  #include "funcoes.h"
7
8  void cadastro(char nome[100]){
9
10         FILE *escrita;
11   // Escrevendo no arquivo
12         escrita = fopen("lista.txt","a");
13     char  aux[1000];
14
15   if (escrita == NULL)
16         {
17                 printf("Erro na abertura
                       do arquivo");
18                 //return 1;
19         }
20
21   sprintf(aux,"raspistill -w 640 -h 480 -q
         75 -o ./imagens%s.jpg",nome);
22   fprintf(escrita ," %s",nome);
```

```
23      system(aux); //tira a foto
24
25      fclose(escrita);
26      printf("Os dados foram gravados com
            sucesso! \n");
27
28
29  }
```