Relatório Final
Registrador de Presença por Detecção Facial

Leonardo Amorim de Araújo - 15/0039921 Josiane de Sousa Alves - 15/0038895
Email: leonardoaraujodf@gmail.com Email: josianealves.18@gmail.com
Universidade de Brasília
St. Leste Projeção A – Gama Leste, Brasília – DF, 72444 – 240

***Resumo*—Este documento apresenta um relatório final desenvolvido para o projeto final da disciplina Sistemas Embarcados, semestre 2017/2.**

***Keywords*—Biometria, Detecção Facial, Raspberry PI, OpenCV, Face Recognition**

I. Introdução

As tecnologias de reconhecimento facial não são novidade nos dias atuais. Diversos dispositivos conseguem realizar trabalhos com processamento de imagens, um exemplo são os filtros digitais para fotos, a detecção de faces em redes sociais, celulares que utilizam a face como senha pessoal e etc. Apesar de ser um trabalho no geral trivial para seres humanos, o reconhecimento facial é uma tarefa altamente desafiadora para computadores e mesmo assim, aparelhos que realizam esta tarefa estão bastante acessíveis nos dias atuais. Muitas são as aplicações possíveis para a detecção facial e aproveitando-se disso que foi construído um dispositivo para este projeto.

II. Objetivos

Construção um sistema de chamada eletrônica, com registro de presença via detecção facial utilizando a Raspberry Pi .

III. Descrição

É alta a demanda hoje por dispositivos que realizam detecção facial, um exemplo bem atual encontra-se na referência [1], onde a detecção facial já é de fato utilizada para determinar se existem potenciais fraldes nos cartões utilizados no sistemas de transporte público no Distrito Federal. Apesar de um problema antigo, uma das grandes preocupações ainda nos dias atuais é com a proteção de dados pessoais e com a eliminação de fraudes, que costumam acontecer ainda com o uso de senhas pessoais e cartões. Além disso, tarefas que exijam controle manual de pessoas que entram e saem em um estabelecimento, presença no trabalho ou em sala de aula são boas alternativas para a aplicação de projetos que envolvam biometria facial para uma possível evolução em projetos mais sofisticados, como o uso pela polícia para identificação de criminosos[3]. A biometria facial é, portanto, uma alternativa para estes e muitos outros problemas, e com base nisto que será proposto o projeto.

IV. Requisitos e Protótipo

Dentre os requisitos inseridos, visa-se:

- Obter o reconhecimento de todas as faces distintas cadastradas no banco de dados;
- Ter uma câmera com capacidade de detectar a face mesmo em ambientes com luz mais baixa;
- Conseguir um tempo máximo de 5 segundos para reconhecimento da imagem e registro da presença;
- Acrescentar a possibilidade do professor selecionar a turma em que a chamada será realizada para que não haja choque de dados;
- Implementar uma interface gráfica para que o aluno saiba que sua presença foi de fato registrada.

V. Descrição de Hardware

- Raspberry Pi 3 Modelo B: Sendo o componente principal, é o computador, ou sistema embarcado, que realiza o processamento de imagens e a tomada de decisão no projeto;
- Câmera: Componente por meio do qual a Rpi verifica a presença do aluno em sala de aula, fazendo análise dos frames para tomada de decisão. Testes de idenficação de face foram realizados com uma câmera de 1 MP relevando sucesso para esta escolha;
- Display: Componente que servirá de interface usuário-máquina, seu objetivo será possibilitar ao professor a escolha da turma em que a chamada será realizada (caso o professor tenha mais de uma turma), além de proporcionar ao aluno a confirmação da sua presença. Será utilizado um display LCD 16x2.
- Teclado: Um teclado inglês foi utilizado também. Este será necessário somente quando o professor decidir realizar alterar algum dado das turmas ou alunos. Para a presença, o teclado não é necessário, somente os botões.
- Push buttons: Componente de seleção. Será utilizado em conjunto com o display para que o professor escolha em qual turma a chamada será realizada.
- WebCam: Utilizada para tirar as fotos para cadastro e registro de presença dos usuários.

A tabela I mostra a lista de componentes em detalhe utilizados no projeto e a figura 1 mostra como foi realizada a ligação de cada componente.

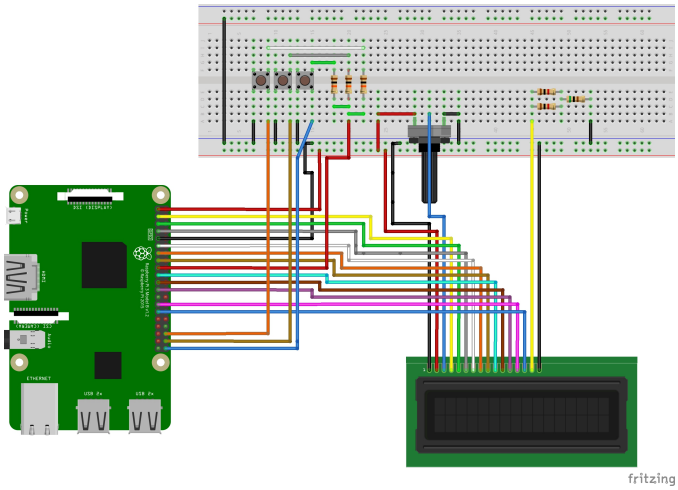

Figura 1. Montagem dos componentes do projeto

Tabela I
MATERIAIS UTILIZADOS NO PROJETO

Quantidade	Material
1	Raspberry Pi 3 Modelo B
1	Display LCD 16x2 JHD 162 A com luz de fundo verde
3	Push-Button sem trava
3	Resistor de $10,0 \pm 0,5\ k\Omega$ 1/4 W
2	Resistor de $1,00 \pm 0,05\ k\Omega$ 1/4 W
1	Resistor de $100 \pm 5\ \Omega$ 1/4 W
1	Mini Protoboard 170 Pontos
1	Webcam 1.3 Mp

VI. DESCRIÇÃO DE SOFTWARE

Para a confecção deste projeto, foi utilizada a biblioteca OpenCV [4] - *Open Source Computer Vision Library*. Esta biblioteca foi desenvolvida pela Intel no início dos anos 2000 e é de uso acadêmico e comercial, para desenvolvimento de projetos na área de Visão Computacional. Possui módulos de processamento de imagens e vídeo, estrutura de dados, Álgebra Linear, GUI e muitos outros algorítmos. Inicialmente, foram desenvolvidos/adaptados códigos na linguagem de programação Python para que se pudesse verificar se o projeto era factível. Posteriormente, os códigos criados em Python foram readaptados para a linguagem C++, linguagem esta utilizada em todo o projeto.

Diversas etapas do projeto foram sendo desenvolvidas em paralelo no decorrer da confecção deste, e desta forma, muitas funções e códigos tiveram que ser rearranjados e adaptados conforme a necessidade prática pedia.

Para que se possa explicar cada componente de código do projeto, irá-se aqui demonstrar a função de cada *header file* primeiramente, e assim poderá se mostrar as ideias principais de cada função nestes.

A. *Display.h*

Neste *header* está definido todas as funções, variáveis e classes utilizadas para a utilização do Display 16 x 2. No arquivo *Display.cpp* estão as implementações das funções declaradas no *header*. Uma classe chamada *Display* foi criada para auxiliar. Cada pino do display foi relacionado à uma GPIO da Raspberry Pi seguindo a tabela II definida pela dupla, de forma que o arranjo das GPIOs foram definidas nas variáveis privadas *GPIOs[11]*, que é um vetor de int que tem todos os valores das GPIOs cujos arquivos *direction* devem ser abertos no modo de saída e terem os arquivos *values* escritos, que serão inseridos no vetor *GPIOs_value[11]*.

O construtor *Display()* não tem inicialização alguma pois foi interessante para a dupla fazer com que a configuração inicial do display seja realizada no arquivo *main* do projeto.

O método *void Setup(void)* da classe Display realiza a configuração inicial. Ela está responsável por criar as GPIOs, definir as direções das GPIOs como saída e a abrir o arquivo *value* de cada GPIO, passando esses valores para a variável *GPIOs_value[11]*. Além disso, este inicializa o display, utilizando o método da classe *Display* chamado *Init()*.

O método *Init()* inicializa o display enviando três comandos 4 comandos: Inicialização da linha de dados, Ligar o LCD e ligar o Cursor, Incrementar o Cursor e Decrementar o Cursor. Este realiza esta configuração inicial utilizando três métodos: *Display::Send_Data()*, *Display::Selection* e *Display::Enable()*.

O método *Display::Selection* têm a função de selecionar entre os modos de comando ou dados, para que os dados recebidos nos pinos de DB0 à DB7 sejam armazenados ou no registrador de comando ou no registrador de dados. Este escreve no arquivo descrito em *GPIO_value[0]*.

O método *Display::Send_Data()* tem a função de receber dados de um caracter e enviar estes dados para os pinos de dados do display no formato hexadecimal. Esta método realiza *bit shifting* do dados em char recebidos e trata-os como valores em hexadecimal. A função somente envia os dados, para os pinos definidos em *GPIO_value[3]* à *GPIO_value[11]*, onde os pinos DB estão definidos.

O método *Display::Enable()* dá um pulso no pino de ENABLE do display para enviar os dados ou comandos. O tempo definido para o pulso foi escrito em *#define WAIT_TIME 1000*, ou seja, 1 ms.

O método *void Display::Command(string command_name)* recebe como parâmetro uma string que define o tipo de comando a ser aplicado no display. Os comandos disponíveis no *datasheet* são muitos, e dentre estes, os escolhidos foram:

- Increment cursor - Incrementa o cursor
- Clear screen - Limpa a tela
- Activate second line - Ativa a segunda linha
- Go to second line - Manda o cursor para o início da segunda linha
- LCD ON, Cursor ON, Cursor blinking ON - Liga o LCD,

liga o cursor, deixa o cursor piscando
- Go to first line - Ativa a primeira linha
- Display ON ,Cursor blinking OFF - Liga o LCD, mas o cursor permanece desligado

Tabela II
PINOS NO DISPLAY E RESPECTIVOS PINOS GPIO

Pino no Display	GPIO
RS	2
RW	3
E	4
DB0	17
DB1	27
DB2	22
DB3	10
DB4	9
DB5	11
DB6	5
DB7	6

O método *void Display::Print(char *word)* recebe uma string, ou char, e escreve na tela do display cada caracter incrementando sempre que um caracter é escrito. Neste método, ele escreve somente uma linha. Se for de desejo que se utilize outra linha, deve-se especificar utilizando o método de comandos para a segunda linha ser a ativa e que o cursor vá para o início da segunda linha, onde o método *Print* pode ser utilizando novamente para escrever os caracteres.

Por fim, o destrutor de Display() fecha todos os arquivos *value* das GPIOs e as desmonta usando o *unexport*.

B. Botao.h

O *header Botao.h* tem a função principal de tomar decisões conforme os botões vão sendo pressionados, atualizando uma struct do tipo *Decisao*. A struct *Decisao* tem os campos:

- navegar - variável que será atualizada com 0 - se o botão de navegar for pressionado pelo usuário e 1 - botão não pressionado.
- selecionar - variável que será atualizada com 0 - se o botão de selecionar for pressionado pelo usuário e 1 - botão não pressionado.
- voltar - varíavel que será atualizada com 0 - se o botão de voltar for pressionado pelo usuário e 1 - botão não pressionado.
- GPIO[3] - 3 GPIOs nos quais os botões serão ligados.
- opcao_atual - variável que define o estado atual da máquina de estados do menu.
- opcao_anterior - variável que define o estado anterior da máquina de estados do menu.
- thread_id[3] - id das threads que serão criadas para cuidar dos botões.

Todas as funções desse header foram definidas no arquivo *Botao.cpp*. A função *GPIO_setup(int num)* recebe o número de uma GPIO e cria esta GPIO definindo-a como entrada. Já a função *int GPIO_get(int GPIOnum)* recebe o número de uma GPIO já criada e lê no arquivo *value* desta GPIO, entregando o valor lido (1 - botão não pressionado ou 0 - botão pressionado, pois todos os botões estão com resistor de pull-up).

A função *void *Verifica_Navegar(void *d)* é uma função que será utilizada para inicializar a thread que cuida do botão navegar. Esta utiliza a função *GPIO_get* para receber o valor lido no botão e atualizar o ponteiro para a struct no campo de decisao→navegar com este valor lido, para que a função que cuida do menu possa saber que o botão foi pressionado ou não. A função *void *Verifica_Selecao(void *d)* realiza a mesma ação, mas para o botão seleção, atualizando o campo $decisao- > selecao$. Já a função *void *Verifica_Voltar(void *d)* também realiza a mesma ação, mas para o botão voltar, atualizando o campo $decisao- > voltar$.

A função *void Configurar_Botoes(Decisao *decisao)* utiliza a função *GPIO_setup* para criar as GPIOs, cria as threads passando o endereço das funções que as threads devem ser inicializadas, pega o endereço das threads e insere nos campos struct $decisao- > threadid$.

Por fim, a função *void GPIO_free* libera as GPIOs criadas para utilizar os botões, fechando os arquivos *value* e usando o arquivo *unexport*.

C. ProjetoFinal.h

Este é o terceiro e último *header file*. As funções foram implementadas no arquivo *ProjetoFinal.cpp*. Um menu principal com opções para aluno e professor foi implementado, de forma que o usuário possa ir navegando pela opção desejada, como em um dispositivo. O menu tem duas opções, 1. Opções de Professor, que são:

- Cadastrar Nova Turma;
- Ver Turmas Cadastradas;
- Excluir Turma;
- Cadastrar Aluno;
- Excluir Aluno;

E 2. Opções de aluno, que são:

- Registrar Presença
- Ver Presenças

Quatro funções principais realizam o papel essencial para funcionamento do display e seleção de modos pelo usuário:

- *void Tomar_Decisao(struct Decisao *decisao,Display *display)* - Esta função realiza a atualização da máquina de estados do menu conforme os botões forem sendo lidos, atualizando o campo do ponteiro para struct $decisao \rightarrow opcao\_atual$. Os estados representam todas as opções possíveis para serem acessadas no menu do registrador de presença, utilizando o botão navegar (para passar de uma opção à outra), o botão selecionar (para selecionar uma opção) e o botão voltar (para voltar de uma opção do menu principal).
- void *Realizar_Acao(struct Decisao *decisao, Display *display)* - Esta função irá escrever no display os dados da opção selecionada, ou seja, do estado atual. Depois disto, o campo do ponteiro $decisao \rightarrow opcao\_anterior$ é atualizado com o valor recebido do campo $decisao \rightarrow opcao\_atual$. Isto foi realizado porque não era necessário atualizar a todo momento o texto a ser exibido no

display, poupando processamento tanto do programa no terminal da Rpi, quanto do microcontrolador interno do display. Desta forma, os valores apresentados no menu do display foram atualizados somente quando uma ”interrupção”causada pelos botões no endereço da struct Decisao fosse detectada.

- void *Acao_Display(Display *display, char *str1, char *str2)* - Esta função serviu tão somente como um auxílio para a função *Realizar_Acao*, pois vários comandos são necessários para escrever nas duas linhas do display, e com esta função, a necessidade de repetir estes comandos foi retirada.
- *int Registrar_Presenca(Display *display)* - Esta função é basicamente o ”coração de todo o projeto”. O código da função utiliza o comando face_recognition[5], construído à partir da biblioteca dlib, e vários algorítmos prontos para uso de *Machine Learning*. Se o usuário selecionar através do menu as **Opções de Aluno**, conforme a figura 2, e selecionar **Registrar Presença**, esta função será chamada. O primeiro questionamento será para que o usuário selecione o número da turma a ser inserida. Após a inserção, aparecerá uma contagem no display, conforme a figura 4. Após a foto ser tirada, o sistema processará a imagem do usuário, ou seja, usará o programa pronto *face_recognition* e comparará com o banco de fotos de alunos cadastradas na turma especificada, usando o comando *face_recognition turma%d/fotos turma%d/unknown — cut -d ’,’ -f2 → resultado.txt*. Por exemplo, se a turma considerada for a turma de número 1, este pegará a foto unknown.jpg e comparará com as fotos dentro da pasta turma1/fotos, retornando nada para o arquivo resultado.txt, ou o nome da foto que foi reconhecida dentro da pasta turma1/fotos. O comando cut -d ’,’ -f2 pega o resultado do comando *face_recognition* e usa um pipe para cortar a parte .jpg da foto, caso encontrada. Como as fotos dos alunos cadastrados estarão no formato matricula.txt, onde matricula = número da matrícula do aluno, este escreverá o número da matrícula no arquivo resultado.txt. Após isto, a função *Registrar_Presenca()* lê o arquivo resultado.txt retornando no display ”Aluno Desconhecido”ou ”Presença Para: Matricula”. A figura 5 mostra o resultado usando uma das matrículas do autores do projeto. A função acima utiliza ainda o programa Computar_Presenca repassando o número da matrícula e a turma para o programa, e este atualiza a lista de presença no arquivo infoturmas%d.txt, onde %d recebe o número da turma especificado pelo usuário.

Os outros programas dentro do header file, que são acionados ao selecionar as opções, são:

1) Cadastro de Nova Turma: Uma nova turma pode ser cadastrada pelo professor. Neste código, o professor deve informar o nome da disciplina e a turma, e então uma pasta para esta nova turma será criada e dentro dela haverá o arquivo *infoturmas.txt*, que contém as informações de todas as turmas cadastradas. Dentro desta pasta, um arquivo chamado *listaturmax.txt*, com x = 1, 2, 3... será criado, este arquivo corresponde a lista de chamada da turma cadastrada.

Figura 2. Opções do Menu Principal para o Aluno e Professor

Figura 3. Opções do Menu do professor

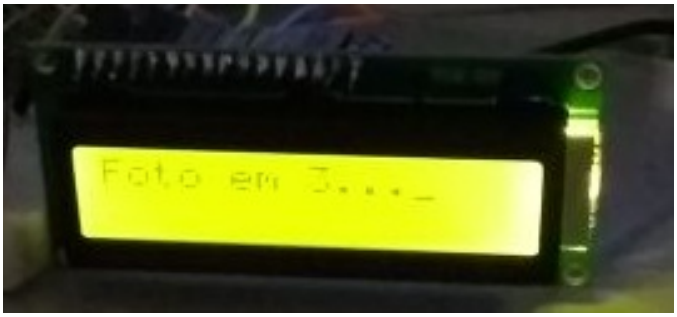

Figura 4. Contagem Regressiva para a foto a ser tirada do aluno

Figura 5. Resultado Final ao detectar uma face

2) Ver Turmas: Caso deseje, o professor pode ver o arquivo infoturmas.txt, onde é apresentado o número da turma, a disciplina e o nome real da turma cadastrada no display.

3) Excluir Turma: Se desejar, o professor pode excluir uma turma. Ao excluir, o programa verificará se a turma especificada existe (se não existir, o usuário é alertado da não existência). O programa exclui a pasta da turma e atualiza o arquivo infoturmas.txt.

4) Cadastro de Novo Aluno: Caso um aluno não esteja cadastrado no sistema, há a opção para o professor cadastrá-lo, sendo possível ser alertado do fato de o aluno já ter sido cadastrado. Na pasta da turma, a foto tirada do aluno é armazenada na pasta turmax/fotos, onde x é o número da turma.

5) Excluir Aluno: Ao excluir um aluno, o arquivo listaturmax.txt é atualizado com o novo número de alunos. O programa também alerta quando é passado como parâmetro a matrícula e turma de uma aluno não identificado.

A figuras 6 e 7 apresentam fluxogramas das opções **cadastro de uma nova turma** e **cadastro de uma novo aluno**.

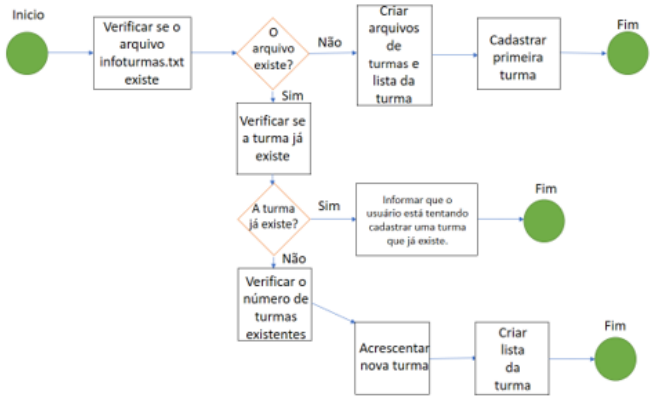

Figura 6. Fluxograma - Cadastro de nova turma

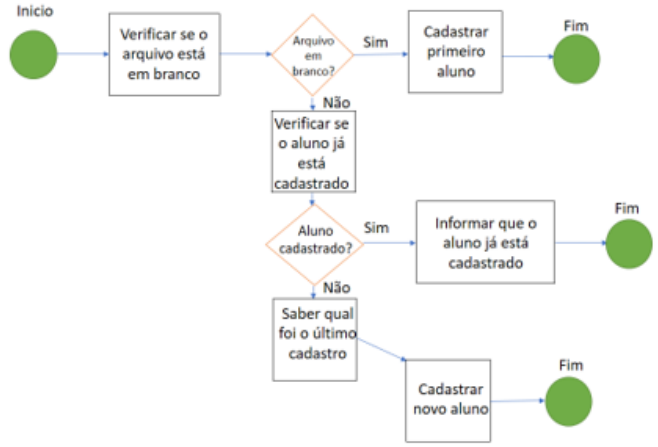

Figura 7. Fluxograma - Cadastro de novo aluno

VII. Resultados

A figura 8 mostra o projeto por completo, além de mostrar como ficou ligado os componentes. Conseguiu-se por partes mostrar que a proposta do registrador de chamadas era factível, pois foi possível verificar a presença de uma pessoa tirando foto da face desta e comparando com o banco de fotos de alunos cadastrados. Conseguiu-se também cadastrar, excluir e ver dados sobre uma turma, além de cadastrar, registrar presença e verificar dados sobre o aluno de uma turma.

Apesar de a dupla ter desenvolvido todo os códigos do menu, não foi possível em tempo juntar todos estes programas prontos e individuais com o programa principal, pois este projeto teve muitos contratempos relacionados principalmente com a parte principal, que era **reconhecimento facial**. A primeira parte dos contratempos esteve relacionada com o uso da biblioteca OpenCV, que requer muita afinidade com a linguagem de programação C++, além de um árduo processo de instalação, que além de demorado, gerava diversas falhas. A dupla teve que tentar instalá-la com diversos tutoriais na internet, que não funcionavam por completo.

Um outro grande problema encontrado foi o fato de conseguir, utilizando a biblioteca OpenCV, a confecção de um código para um programa conseguir realizar o reconhecimento facial de uma pessoa. Após o ponto de controle 3 toda a biblioteca OpenCV estava funcionando normalmente, mas como precisávamos do treinamento de faces utilizando os *H*aarCascades, a biblioteca não funcionava com os programas *c*reate_samples que cria um vetor de amostras positivas com as fotos a serem detectadas e O *t*raining_cascade. A dupla teve que mais uma vez passar árduos dias tentando configurar a biblioteca que por fim funcionou antes do ponto de controle 4.

Apesar disto tudo, outro contratempo foi enfrentado, pois as amostras positivas do rosto a ser detectado não geravam um *c*ascade que conseguia detectar uma face. A dupla tentou por mais vários dias, usando de 150 a 10000 amostras de uma face para treinar o classificador, mas não funcionou. Este fato foi apresentado no Ponto de Controle 4, e nos foi proposto utiliza um programa já pronto de detecção facial[5].

Este programa de detecção facial proposto também necessitava de instalação de uma biblioteca, e por mamis 6 dias a dupla ficou tentando instalar a biblioteca, mas sem êxito, até que conseguiu-se encontrar a solução em um dos fóruns aleatórios no site *S*tack Overflow.

Em paralelo, enquanto não funcionava a instalação do API [5], a dupla foi tentando ajustar os códigos para o display 16x2 e a interação máquina-usuário nesta última etapa do projeto, utilizando os push-buttons e integrando-os posteriormente ao display, criando um menu, que foi a interface gráfica, e possibilidade de selecionar opções e voltar.

Por fim, apresentou-se o projeto final, mas sem alguns dos componentes principais (programas), que apesar de terem

sido feitos, não conseguiu-se em tempo integrar estes a solução, devido aos problemas de instação das bibliotecas.

Um resultado importante também a ser comentado é o fato de que o programa *face_recognition* demorava em torno de 1,5 minutos para conseguir reconhecer a face de uma pessoa. Foi possível verificar, ao ver os comentários dos outros usuários do programa na internet, que computadores com processadores bem mais potentes e com muito mais capacidade de processamento que a Rpi, demoravam em torno de 20 segundos para reconhecer uma face utilizando o programa. Desta forma, batemos de frente com uma especificação do projeto que dizia que o registro de presença deveria ocorrer em menos de 5 segundos.

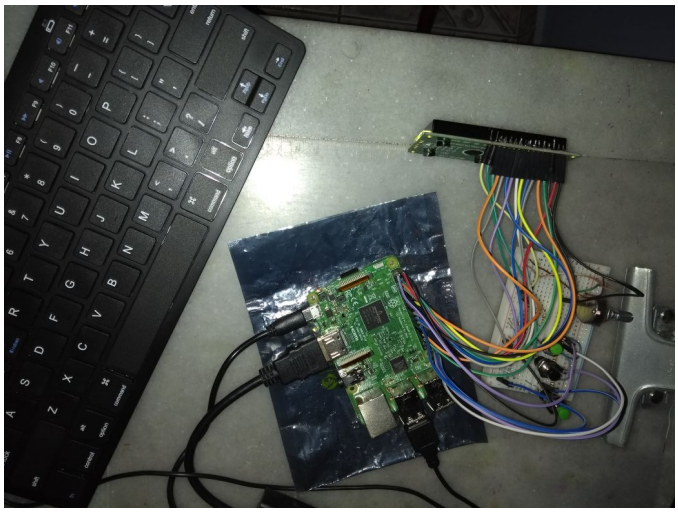

Figura 8. Montagem do projeto

VIII. Conclusão

Neste trabalho, apresentamos uma proposta para o problema de realizar chamadas dentro de sala de aula manualmente, utilizando a Rpi, um display 16x2 e botões, além de um teclado. Foi possível demonstrar que a ideia em si é factível, mas que depende de muitas outras variáveis que estão além do escopo da disciplina, que é, por exemplo, a detecção facial em um tempo adequado. A ideia de fato é realizável, mas precisa de diversor ajustes, principalmente no que tange a área de Processamento Digital de Imagens. Se um produto fosse de fato ser criado com esta ideia, outro ajustes deveriam ser necessários. O primeiro deles seria acoplar todo o projeto em um compartimento, o segundo realizar treinamento dos usuários e verificar se a interface usuário-máquina realmente responde as expectativas, ou se mudanças deveriam ser realizadas. Uma outra possível aplicação é usar a Rpi somente como coletora de imagens e de acesso a menus, e esta se comunicaria com um servidor via Wi-fi para determinar se uma face foi ou não reconhecida. Todo o trabalho de banco de dados seria, portanto, realizado pelo servidor, além do processamento das imagens.

Apesar de todos os problemas expostos, um outro aspecto positivo foi de que a dupla adquiriu muitos conhecimentos, principalmente no que tange o terminal do linux e na resolução de problemas comuns no terminal, que eram conhecimento esperados para a displina. Além disso, a dupla fez um curso da linguagem C++, de 16 horas no site Udemy[6], e um curso de OpenCV no mesmo site[7], de 6 horas.

IX. Anexos

A. Códigos

Todos os códigos podem ser visualizados no link para o github a seguir:
https://github.com/leonardoaraujodf/Sistemas_Embarcados/tree/master/2_PCs/codigos_pc4

Referências

[1] Biometria facial estará em toda a frota de ônibus até fevereiro, diz GDF. Acesso em 29/11/2017. http://www.correiobraziliense.com.br/app/noticia/cidades/2017/11/21/interna_cidadesdf,642367/biometria-facial-em-toda-a-frota-de-onibus-ate-fevereiro.shtml

[2] Aprenda a desbloquear seu notebook por reconhecimento facial, acesso em 04/10/2017. http://www.techtudo.com.br/dicas-e-tutoriais/noticia/2015/06/aprenda-desbloquear-seu-notebook-por-reconhecimento-facial.html

[3] 100+ Projects in Image Processing and Fingerprint Recognition, acesso em 04/10/2017. http://projectabstracts.com/list-of-projects-on-image-processing

[4] OpenCV library, acesso em 04/10/2017. http://projectabstracts.com/list-of-projects-on-image-processing

[5] The world's simplest facial recognition api for Python and the command line, acesso em 29/11/2017. https://github.com/ageitgey/face_recognition

[6] C++ for complete beginners, acesso em 29/11/2017. https://www.udemy.com/free-learn-c-tutorial-beginners/

[7] Learning Path: OpenCV: Master Image Processing with OpenCV 3, acesso em 29/11/2017. https://www.udemy.com/learning-path-opencv-master-image-processing-with-opencv-3/learn/v4/overview