Felipe Freitas de Carvalho

Reconhecimento facial utilizando o raspberry pi e visão computacional

Belo Horizonte

Junho, 2017

Felipe Freitas de Carvalho

Reconhecimento facial utilizando o raspberry pi e visão computacional

Monografia apresentada durante o Seminário dos Trabalhos de Conclusão do Curso de Graduação em Engenharia Elétrica da UFMG, como parte dos requisitos necessários à obtenção do título de Engenheiro Eletricista.

Universidade Federal de Minas Gerais – UFMG

Escola de Engenharia

Curso de Graduação em Engenharia Elétrica

Orientador: Prof. Cristiano Leite de Castro

Belo Horizonte

Junho, 2017

Este trabalho é dedicado a todas as pessoas que estiveram comigo ao longo destes anos, que acreditaram em mim e me motivam a ser sempre melhor.

Agradecimentos

Gostaria de agradecer a todos que estiveram juntos comigo nesta caminhada e que me ajudaram a realizar o sonho de me formar em um excelente curso.

Gostaria de agradecer a todos meus colegas, meus pais e minha namorada por estarem sempre presentes e pelo suporte que me deram durante todos estes anos.

Agradeço à Universidade Federal de Minas Gerais, todo seu corpo docente e agradeço a meu orientador, Prof. Cristiano Leite de Castro, por acreditar em meu potencial e me incentivar a seguir meu sonho de trabalhar com aquilo que mais gosto e acredito.

E por fim, gostaria de agradecer a Deus por ter vivido anos maravilhosos de muita alegria e por me dar saúde e forças permitindo que tudo isso fosse possível.

“ Let me tell you something you already know. The world ain’t all sunshine and rainbows. It’s a very mean and nasty place, and I don’t care how tough you are, it will beat you to your knees and keep you there permanently if you let it. You, me, or nobody is gonna hit as hard as life. But it ain’t about how hard you hit. It’s about how hard you can get hit and keep moving forward; how much you can take and keep moving forward.

(Rocky Balboa)

Resumo

Neste trabalho será utilizado um microcomputador, em conjunção com um sensor de presença ultrassónico e uma câmera para realizar a tarefa de reconhecimento facial de um usuário conhecido em tempo real. Para isto serão utilizados algoritmos de visão computacional clássicos, que utilizam internamente técnicas de aprendizado de máquina, para que se extraia de uma imagem capturada a face do usuário para que essa possa ser reconhecida.

A motivação do trabalho é criar um dispositivo, altamente portátil, que tenha a capacidade identificar um usuário cadastrado mediante a aproximação deste, podendo ser usado para diversos fins em momentos futuros. Dentre as possibilidades de uso, destacam-se aplicações para segurança residencial ou empresarial, ou como sensor altamente inteligente que poderia ser usado para tarefas de IOT.

Palavras-chave: Visão computacional, raspberry pi, reconhecimento facial.

Abstract

In this project a microcomputer will be used together with a motion sensor and a camera to accomplish the task of facial recognition of a known user in real time. To achieve that, classic computer vision algorithms, that make use of machine learning techniques, will be used to extract the face of an image and recognize it.

The motivation behind this work is to create a highly portable device that has the ability of recognizing a known user when it approaches, that can be used for several other tasks in a future moment. Among the possibilities of use there are security applications for houses or enterprises, or even a very intelligent sensor to be used as an IOT device.

Key-words: Computer vision, raspberry pi, facial recognition.

Lista de ilustrações

Lista de abreviaturas e siglas

CNN	*Convolutional neural networks*
GPU	*Processing Unit*
AI	*Artificial Intelligence*
IOT	*Internet of Things*
CSI	*Camera Serial Interface*
NOOBS	*New out of the box software*
APT	*Advanced Package Tool*
LAN	*Local Area Network*
SSH	*Secure Shell*
VNC	*Virtual Network Connection*
TTL	*Transistor–transistor logic*
GND	*Ground*
LBPH	*Local Binary Patterns Histograms*
k-NN	*K-nearest neighbors*
XML	*eXtensible Markup Language*
ROC curve	*Receiver Operating Characteristic curve*

Sumário

1 Introdução

1.1 Contextualização

O avanço da inteligência artificial é algo que poderá ser muito benéfico para o futuro da humanidade. A inteligência artificial será capaz de nos auxiliar em nossas tarefas, desde as mais simples até as mais complexas. Estima-se que em cerca de duas décadas grande parte das profissões façam uso de algum tipo de inteligência artificial e que outras desapareçam completamente por causa dela. Esse processo de mudança será semelhante ao que ocorreu com a revolução industrial, só que ao em vez de utilizarmos máquinas para realizar esforços mecânicos intensos que não seriamos capazes nós mesmo, usaremos maquinas inteligentes, que são capazes de realizar tarefas específicas que demandam raciocínio, mais eficientemente que nós mesmos. Um exemplo de uma situação em que utilizamos uma certa forma de inteligência provinda de maquinas está no uso dos diversos serviços de geo-localização disponíveis; estes têm uma inteligência espacial muito maior que a nossa. E a tendência para o futuro é que surjam outras maquinas que sejam mais inteligentes que nós em outros tipos de tarefas específicas, como dirigir, fornecer diagnósticos médicos baseados em sintomas, fazer a contabilidade para uma empresa, dentre muitas outras. Isso não significa que profissões tradicionais desaparecerão, mas que elas receberam auxilio de novas ferramentas, que facilitarão muito o trabalho.

Um dos ramos da inteligência artificial que vem evoluindo bastante nos últimos tempos é o da visão computacional. Para nós, o que é uma tarefa simples como distinguir rostos ou objetos, pode ser complicado para um computador, até mesmo pelo fato de que nós mesmo não conseguimos definir como conseguimos diferenciar os objetos que vemos, é algo natural para nós, evolutivo. Diversos algoritmos foram introduzidos com o intuito de melhor decifrar a tarefa de visão computacional. Os algoritmos estado da arte hoje em dia utilizam CNNs(CS231, 2017), as redes neurais convolucionais. Estas utilizam várias camadas que fazem diversas abstrações, filtram e por fim classificam a imagem como determinada classe. Estes algoritmos podem diferenciar diferentes objetos, reconhecer diferentes rostos e realizar as mais diversas tarefas de visão computacional, desde que sejam bem utilizados. Esses métodos são computacionalmente custosos, necessitam de hardware de última geração e normalmente utilizam *clusters* de GPUs para o processamento da rede criada.

Existem, no entanto, algoritmos clássicos que se mostraram excelentes para tarefas de reconhecimento mais específicas de visão computacional, como por exemplo, a tarefa exclusiva de reconhecimento facial. Estes são mais bem mais leves e utilizam técnicas clássicas de aprendizado de máquina. Essa classe de algoritmos pode ser utilizada em

hardware mais acessível, que não precisa ser robusto ou de última geração.

A viabilidade destes algoritmos perante hardware mais simples, permitem que eles sejam embutidos em microcomputadores portáteis, e que realizem a tarefa de reconhecimento facial localmente e com precisão. Aproveitando deste fato, neste trabalho será apresentado um projeto que envolve o uso da biblioteca OpenCV (OPENCV, 2017), de visão computacional, em conjunção com o microcomputador Raspberry Pi (RASPBERRYPI, 2017), e um conjunto de periféricos, incluindo câmera e sensor de presença, para realizar a tarefa de reconhecimento facial em tempo real de um usuário conhecido.

1.2 Objetivos

- **Objetivo principal:** Criar um dispositivo capaz de realizar a tarefa de reconhecimento facial de um usuário cadastrado em tempo real de maneira automática.
  - O dispositivo deve sinalizar de forma clara que o usuário correto foi reconhecido, e caso contrário sinalizar que o usuário reconhecido não é o correto.
  - O dispositivo deve ter um funcionamento robusto, rápido e que seja pouco susceptível a falhas, principalmente na parte do reconhecimento, onde deve ser evitado o máximo possível um resultado positivo para uma pessoa não desejada.
  - O dispositivo deve ser portátil e ser passível de ser utilizado posteriormente para outras tarefas, destacando-se a probabilidade deste ser utilizado para segurança residencial ou empresarial ou como um sensor inteligente.
  - O dispositivo deve realizar todos os cálculos necessários e rodar todo e qualquer tipo de software localmente, para que esse seja autossuficiente e não dependa de rede externa. Isso também garante que o dispositivo seja personalizado para a aplicação específica.

1.3 Metodologia

1.3.1 Materiais

Serão listados os materiais utilizados e será feita uma análise sobre cada um, incluindo os motivos pelos quais eles foram selecionados.

- Raspberry Pi 3 model B

O Raspberry Pi é um microcomputador de tamanho reduzido, desenvolvido pela Raspberry Pi Foundation, com o intuito de ser utilizado em diversos tipos de projetos,

e que foi amplamente popularizado e hoje em dia é utilizado para tarefas que saem até mesmo de seu escopo original, como tarefas de robótica, por exemplo.

O Raspberry Pi possui um processador potente baseado em arquitetura ARM (ARM, 2017), com boa quantidade de memória RAM, 1Gb em seu último modelo, sendo seu armazenamento através de cartão SD. O Raspberry permite acesso a diferentes periféricos, contendo entradas USB, HDMI, e de áudio. Ele também possui placa de Wi-Fi *on-board* e entradas analógicas de pinos, conhecida como GPIO, *General Purpose Input Output*, perfeito para inclusão de componentes eletrônicos periféricos adicionais. Além disso, o raspberry tem entrada para câmera via conexão própria e módulo de câmera próprio para ele, a PiCamera.

Altamente portátil, e com capacidade de processamento razoavelmente elevada, dado o seu tamanho, o raspberry é ideal para projetos onde se requer portabilidade, mas ao mesmo tempo não se deve abrir mão da capacidade de processamento.

O Raspberry roda diversos tipos de sistemas operacionais, sendo o mais comumente utilizado o Raspbian, que é um sistema operacional baseado em Debian (DEBIAN, 2017), gratuito e fornecido pela própria Raspberry Pi Foundation. Outros sistemas operacionais podem ser escolhidos, como o Windows IOT (MICROSOFT, 2017), porém o mais comumente usado é o Raspbian. Este tem nativo ambientes de programação que utilizam a linguagem Python (PYTHON, 2017), que é uma linguagem de alto nível, muito popular. Além disso, a instalação de bibliotecas e pacotes é fácil neste tipo de arquitetura, sendo que com simples comandos pode-se instalar bibliotecas como o OpenCV (OPENCV, 2017).

O Raspberry Pi será utilizado neste trabalho como o microcomputador escolhido, e será responsável por todo o processamento de imagens e algoritmos. Sua escolha é baseada no fato deste ser potente o suficiente para a tarefa de reconhecimento facial, além de contar com entradas que possibilitam um fácil acoplamento de periféricos. Somado a isso, o Raspberry é extremamente portátil, leve e acessível.

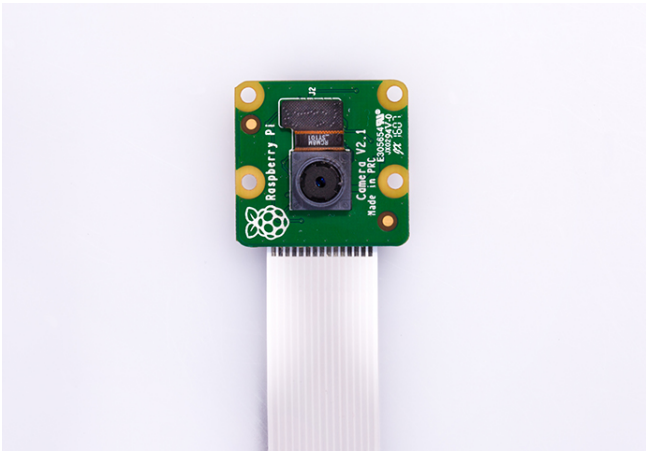
- Câmera e sensor

Uma câmera de baixo custo, boa resolução e qualidade de imagem, que é ideal para ser usada em conjunto com o raspberry pi para realizar a tarefa de reconhecimento facial em tempo real proposta nesse trabalho é o modulo Raspberry Pi Camera v2. Este é capaz de gravar imagens e vídeos de alta resolução, graças a seu sensor de 8 megapixels. Sua conexão com o Raspberry é feita através da entrada CSI -*Camera Serial Interface*- presente no Raspberry. Por ser feita especificamente para o Raspberry, conta com módulos, interfaces e bibliotecas próprias, que facilitam sua utilização em conjunto.

Figura 1 – Raspberri Pi 3 model B

Fonte: Evan-Amos (2017)

Figura 2 – The Raspberry Pi Camera Module v2



Fonte: PiCamera (2017)

Será utilizado em conjunção com a câmera, o sensor de presença SR04. Este é um sensor ultrassônico, de baixo custo, que mede a distância de um objeto ao mesmo, sendo capaz de medir distâncias de 2cm a 2m com precisão. O motivo da escolha deste sensor e o papel que este desempenhará será detalhado na próxima seção.

- Acessórios adicionais
  - Cartão SD para ser usado como HD para o raspberry
  - Cabos, resistores e protoboard
  - LEDs, vermelho e verde para sinalizar o resultado do reconhecimento facial
  - Madeira para produzir uma caixa para encapsular todo o projeto, caixa de acrílico para proteção do raspberry

Figura 3 – Sensor Ultrassonico HC-SR04

Fonte: HCSR04 (2017)

- OpenCV

O OpenCV (OPENCV, 2017) é uma biblioteca de visão computacional *open-source*, ou seja, de uso livre. Possui módulos de processamento de imagens e vídeo e vários algoritmos interessantes que podem ser utilizados para tarefa de reconhecimento e detecção facial e que fazem uso de técnicas de aprendizado de máquina. É escrito em C/C++ e tem interfaces para C++, C, Python e Java. O fato de ser escrito em C/C++ o torna mais otimizado, e ideal para processamento de imagens em tempo real. Além disso, a maioria dos algoritmos presentes para detecção e reconhecimento facial no OpenCV são relativamente leves. O OpenCV é amplamente utilizado atualmente, sendo que alguns outros pacotes de visão computacional também fazem seu uso de forma indireta.

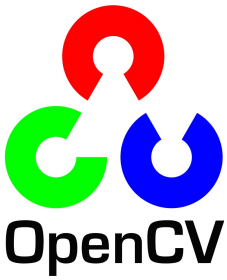
O OpenCV será utilizado como a biblioteca principal para a tarefa de reconhecimento e detecção facial. O OpenCV é leve o suficiente de forma que seu funcionamento não é restrito pela capacidade de processamento limitada do Raspberry Pi. Além disso, o OpenCV conta com algoritmos já implementados e de eficiência comprovada que podem ser utilizados para reconhecimento e detecção facial e também conta com ferramentas de processamento de imagens que facilitarão o trabalho de maneira significativa.

A interface para Python, permite que sejam desenvolvidos scripts em linguagem de alto nível, mas que ao mesmo tempo tenham robustez e rapidez pelo fato de suas funções serem internamente compiladas em liguagem de baixo nível.

- Python

A linguagem de programação escolhida para realizar a parte de desenvolvimento deste trabalho é Python (PYTHON, 2017). Python é uma linguagem de alto nível, de fácil uso e atualmente, junto com a linguagem R (R, 2017), a linguagem escolhida pela maior parte daqueles que trabalham em áreas que fazem uso de ferramentas de inteligência arti-

Figura 4 – OpenCV



Fonte: Shavit (2017)

ficial/computacional. Python tem diversas bibliotecas e pacotes que facilitam o trabalho com análise de dados, algoritmos de aprendizado de máquina e tarefas relacionadas.

Figura 5 – Python

Fonte: Wikipedia (2017)

1.3.2 Métodos

Como passo inicial para a realização dos objetivos deste trabalho, serão embutidas no Raspberry Pi as bibliotecas, suas respectivas dependências e todos os pacotes necessários para garantir que seja possível desenvolver scripts que controlarão o funcionamento do dispositivo e permitirão a realização das tarefas propostas. Dentre as bibliotecas a serem instaladas destaca-se o OpenCV (OPENCV, 2017), que conforme mencionado na seção anterior, contém os algoritmos que serão utilizados para as tarefas de detecção e reconhecimento facial e que também contém ferramentas excelentes para processamento de imagens.

O sistema operacional que será utilizado no Raspberry é o Raspbian, o que faz com que este tenha nativamente interfaces de programação em Python (PYTHON, 2017), além de diversos pacotes em Python que serão úteis. Isso é importante já que todo código, na forma de scripts, utilizados no projeto serão criados utilizando Python como linguagem de programação e eles serão responsáveis pela lógica e execução do projeto.

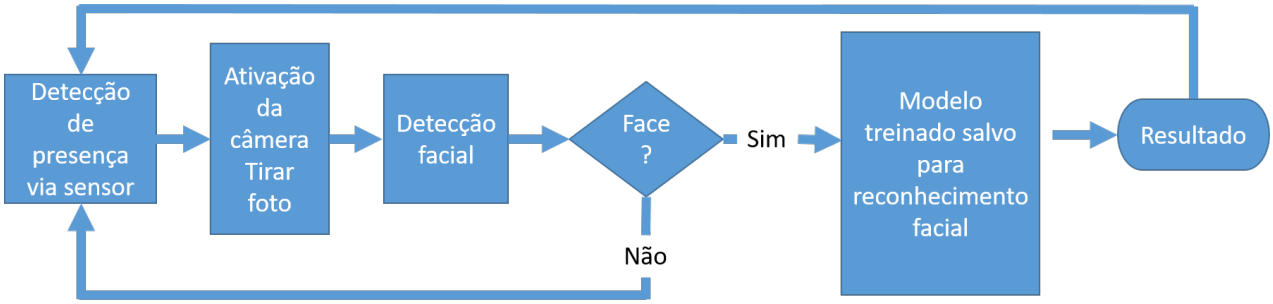
Serão conectados ao Raspberry o sensor e a câmera escolhidos, através das interfaces disponíveis. No caso da câmera, será utilizada a entrada CSI - entrada serial da câmera. O sensor será conectado via GPIO - entrada de pinos analógicos - e com auxílio

do protoboard. Seu propósito será dar início ao processo de reconhecimento facial. Além disso serão conectados os LEDs de maneira similar, que serão responsáveis por sinalizar se o reconhecimento foi da pessoa desejada ou não, para isso acendendo o LED verde ou o vermelho.

Os acessórios adicionais, como cabos e resistores, servirão para possibilitar a conexão dos diferentes periféricos e os demais, como a caixa de acrílico, para permitir que o conjunto seja encapsulado em um pacote portátil.

O funcionamento do dispositivo pode ser analisado através do fluxograma abaixo:

Figura 6 – Fluxograma: Funcionamento do dispositivo



Fonte: Produzido pelo autor

Em um primeiro momento o sensor de presença detectará se algum objeto se aproxima à frente do dispositivo, caso isso ocorra a câmera será ativada e tirará uma foto. Essa foto passará por um processo de detecção facial. Caso seja detectada uma face, esta será testada pelo modelo treinado para reconhecimento facial, onde será indicado caso a pessoa pertence à classe de usuários cadastrados ou é uma pessoa desconhecida. Essa indicação será sinalizada através dos LEDs verde e vermelho. Após isso volta-se para a etapa de detecção de presença via sensor onde o loop é continuado. Caso a foto tirada, após passar pelo processo de detecção facial, não tenha uma face identificada, retorna-se para a etapa de detecção de presença via sensor.

Todo esse processo será programado via scripts escritos em Python. Estes serão responsáveis pelos algoritmos e pela realização da interface entre os diversos dispositivos.

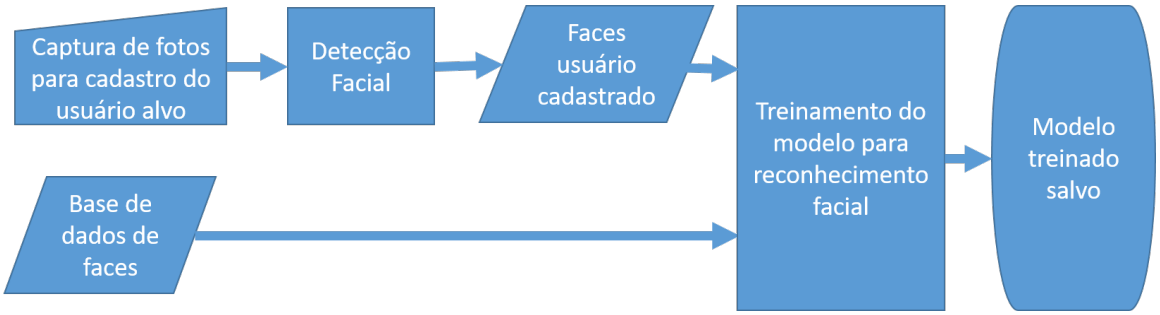
A escolha do sensor como ativador do processo é estratégica. Ela visa reduzir a quantidade de processamento feito pelo dispositivo, fazendo com que a etapa de processamento só seja necessária caso haja algum objeto presente perto do dispositivo. Além disso, como o sensor fornece a distância do objeto, esta será usada para detectar objetos apenas a distancias desejadas. Essas escolhas agilizam o processo como um todo pois garantem que o usuário não tenha que esperar um ciclo de processamento terminar, ao se aproximar dispositivo, para ter sua face reconhecida, e, portanto, garante que sua foto seja

tirada de maneira imediata e que logo após isso seja processada. Além disso, reduzindo a quantidade de processamento realizado, impedindo que ele seja continuo, evita o desgaste precoce do dispositivo e também é reduzido seu consumo de energia.

Os processos de detecção facial e reconhecimento facial utilizarão algoritmos que utilizam técnicas de aprendizado de máquina. Esses algoritmos aprendem a realizar tarefas sem serem explicitamente programados para elas, como definido por Arthur Samuel em 1959. Estes usam dados a partir dos quais o algoritmo acha padrões e mapeia entradas a saídas e é treinado para realizar isso com o menor erro possível. O modelo treinado por este processo pode ser usado para prever dados nunca antes vistos (AMAZON, 2017).

A parte de detecção facial será por parte de algoritmos disponíveis no OpenCV (OPENCV, 2017), pré-treinados e feitos para esta tarefa específica. Já a parte de reconhecimento facial será realizada por um algoritmo também contido na biblioteca do OpenCV, porém treinado com dados contidos em uma base de dados com fotos do usuário cadastrado e fotos de pessoas aleatórias, quer servem como exemplos de pessoas que não são as desejadas. Este processo pode ser exemplificado no fluxograma abaixo:

Figura 7 – Fluxograma: Treinamento do modelo de reconhecimento facial



Fonte: Produzido pelo autor

Pode ser notado que as fotos capturadas do usuário alvo devem ser passadas pela detecção facial, para que então as faces deste usuário sejam utilizadas para o treinamento do modelo para reconhecimento facial.

2 Desenvolvimento

Os procedimentos e os aspectos técnicos do trabalho serão detalhados nesta seção. Estes seguem a metodologia apresentada anteriormente.

2.1 Setup do Raspberry

O primeiro passo do desenvolvimento deste trabalho é a configuração do ambiente de programação, e por consequência, do Raspberry Pi, para que este fique propicio para realizar a tarefa proposta.

2.1.1 Sistema Operacional

O sistema operacional escolhido foi o Raspbian, que conforme discutido anteriormente é um sistema baseado em Debian, que por sua vez é uma distribuição GNU/linux. O Raspbian foi adquirido através do site da Raspberry Pi Foundation (RASPBERRYPI, 2017) e é um software livre, sua última versão, que é a utilizada neste trabalho é o Raspbian Jessie. Esse sistema tem como nativo ambientes de programação na linguagem Python, o que é ideal para o desenvolvimento dos scripts necessários para a execução do projeto.

Para instalação do sistema operacional o método mais simples é baixando o software NOOBS - acrônimo de *New out of the box software*. Uma vez baixado basta extrair os arquivos em um cartão SD, que servirá como HD para o raspberry. Ao ligar o raspberry pela primeira vez, será mostrado uma interface gráfica onde deverá ser selecionado o Raspbian como sistema operacional e iniciar o processo de instalação. Este dura cerca de 15 minutos.

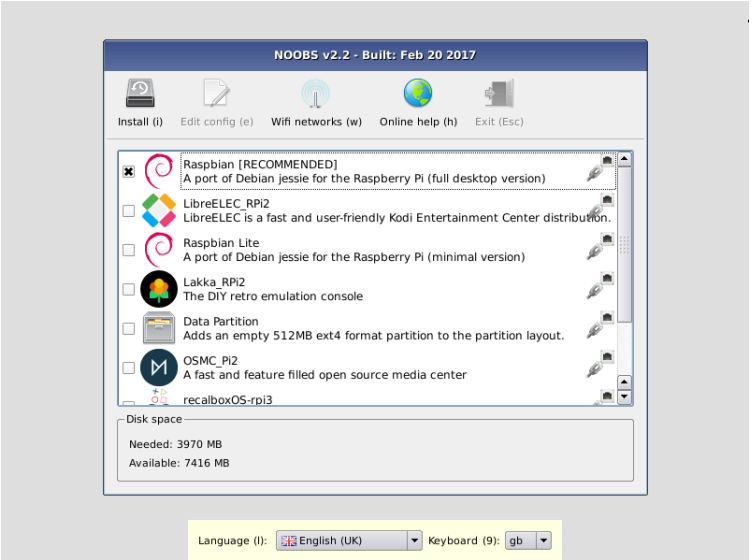
2.1.2 Pacotes importantes

A instalação de pacotes importantes será feita através do sistema de distribuição de pacotes do Debian, o APT. Os pacotes mais relevantes para o desenvolvimento do trabalho são o OpenCV (OPENCV, 2017) e o Scikit-Learn (SCIKIT-LEARN, 2017). Os demais já vêm pré-instalados com a instalação do sistema operacional.

Para instalar o OpenCV e o Scikit-learn deve-se usar os seguintes comandos no terminal de comandos:

```
$ sudo apt-get install libopencv-dev python-opencv
$ sudo apt-get install python-sklearn
```

Figura 8 – Instalando o Raspbian Jessie através do NOOBS



Fonte: RaspberryPi (2017)

Pode ser necessário instalar algumas dependências adicionais para o funcionamento correto destes pacotes, porém isto é normalmente feito de maneira automática no momento da instalação, ou no caso específico deste sistema operacional, grande parte das dependências já vieram instaladas. Estas dependências incluem pacotes como o NumPy (NUMPY, 2017), usado para cálculo cientifico em Python e o Matplotlib (MATPLOTLIB, 2017), utilizado principalmente para criação de gráficos 2d.

2.1.3 Acesso ao raspberry

Como qualquer computador, o raspberry pode ser utilizado com teclado, mouse e monitor, porém, existem maneiras de acessa-lo remotamente que podem facilitar o seu uso, principalmente quando o objetivo é utiliza-lo como um dispositivo para realização de tarefa específica em vez de como um computador normal, que é o caso deste trabalho.

A possibilidade de utilizar o raspberry com ferramentas de acesso remoto é importante, e esta é facilitada pelo sensor Wi-Fi presente no raspberry. O raspberry pode ser acessado via SSH ou VNC, tanto via LAN quanto via internet. O acesso via SSH garante controle do dispositivo por seu terminal, enquanto o acesso via VNC garante acesso ao Desktop, como se este estivesse sendo utilizado via monitor. O programa utilizado para ter acesso via SSH chama-se PuTTY (PUTTY, 2017) e via VNC chama-se VNC (VNC, 2017). Para garantir acesso a esses serviços os mesmos devem ser habilitados no raspberry. Isso pode ser feito via o seguinte comando no terminal:

```
$ sudo raspi-config
```

Este comando abrirá uma interface onde é possível escolher opções para habilitar esse tipo de conexão, assim como habilitar entradas via GPIO, a entrada CSI para câmera, e várias outras opções e configurações do dispositivo.

Ainda existe a possibilidade de usar os programas através de um cabo Ethernet, que pode ser bastante útil quando não há rede disponível para se conectar remotamente ao raspberry.

2.2 Setup da câmera

A câmera Raspberry pi camera v2, por ser projetada especificamente para o raspberry é simples de ser instalada, configurada e operada. O raspberry, com o sistema operacional Raspbian Jessie instalado, já vem com os pacotes necessários para a câmera. Para conecta-la ao raspberry basta inserir seu conector na entrada CSI.

Figura 9 – Camera conectada ao Raspberry pela entrada CSI

Fonte: CameraModule (2017)

A câmera deve ser habilitada via raspi-config, seguindo o mesmo procedimento anteriormente descrito de acessar o terminal e digitar o comando $sudo raspi-config, e selecionar a opção de permitir a câmera.

Para operar a câmera via software, pode-se utilizar a biblioteca chamada *picamera* em python, ou através de comandos via terminal. No caso deste trabalho, a câmera foi utilizada em scripts em python, por isso a primeira opção foi a utilizada. Para importar a biblioteca em python deve-se usar o comando:

```
import picamera
```

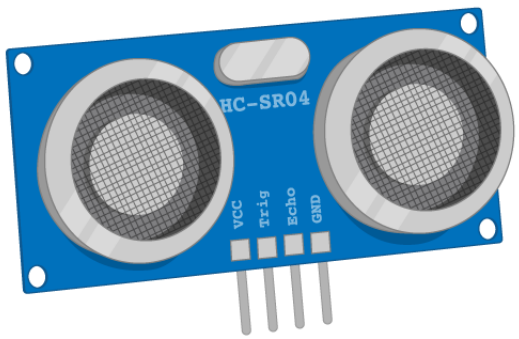
Detalhes nas funções e classes disponíveis nesta biblioteca podem ser encontrados no site que contém a documentação (PICAMDOCUMENTATION, 2017). Neste trabalho, no entanto, apenas funções básicas serão utilizadas para a operação da câmera, como a função de tirar uma foto por exemplo.

2.3 Setup do sensor

O sensor HC-SR04 funciona através do envio de um pulso ultrassónico, de frequência igual a 40kHz, que atinge os objetos a sua volta e é refletido nestes. Quando o pulso retorna ao sensor, através de um receptor ele o capta e, utilizando um circuito de controle, calcula a diferença entre o tempo em que o pulso foi emitido e recebido de volta e gera um sinal para representar este tempo. Usando uma lógica simples e leis da física, é possível calcular a distância do objeto ao sensor.

O sensor HC-SR04 contém 4 pinos, Vcc, trig, Echo e GND. Os pinos VCC e GND são o pino de alimentação 5V e o terra, respectivamente. O trigger é um pino que recebe um pulso de 5V e 10 µs TTL, e é responsável pelo disparo do pulso ultrassônico. O Echo é um pino que fica em nível logico alto, de 5V, após o retorno do pulso ultrassônico, por um tempo igual ao gasto do envio ao retorno do pulso.

Figura 10 – Pinos do sensor HC-SR04



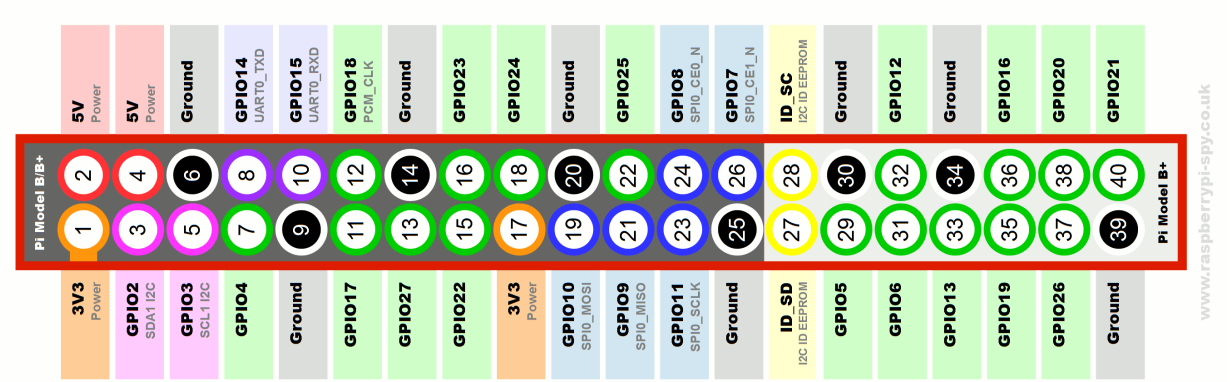
Fonte: RaspberryPiSensor (2017a)

A saída echo pode ser utilizada para calcular a distância do objeto mais próximo ao sensor. Sabendo a velocidade do som, de 343m/s aproximadamente, e sendo o tempo de

duração do pulso echo o tempo gasto do pulso para ir e voltar, pode-se calcular a distância ao objeto de forma simples pela equação $Distancia = (343m/s * t_{pulso})/2$. O divido por dois na formula anterior se baseia no fato de que o tempo corresponde ao percurso de ida e volta do pulso, logo deve ser divido por dois para se calcular a distância.

A conexão do sensor ao Raspberry é feita através dos pinos analógicos presentes na entrada GPIO.

Figura 11 – Layout das entradas GPIO



Fonte: raspberrypispy (2017)

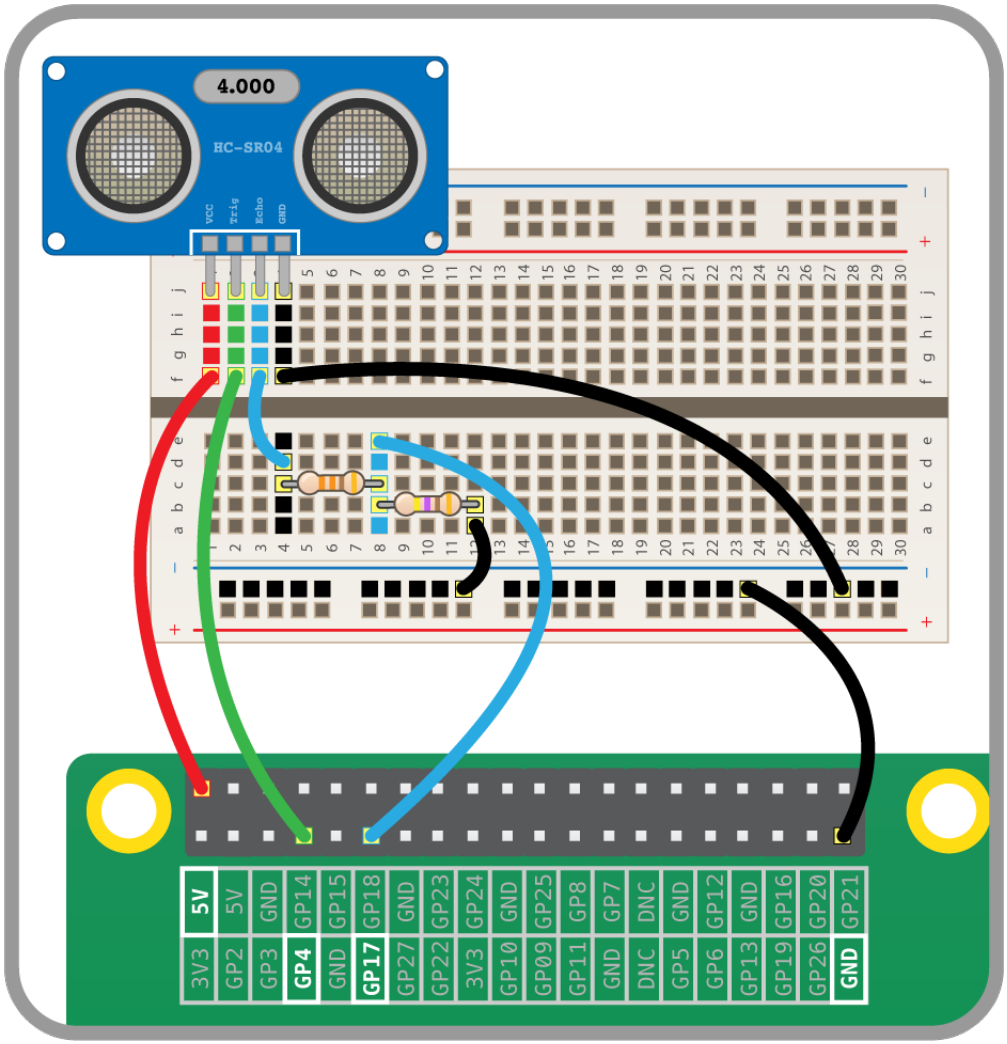
Como pode ser visto, a entrada GPIO tem vários pinos, dos quais os de alimentação 5V poderão ser conectados ao pino VCC do sensor, e analogamente os terras do raspberry e do sensor podem ser conectados. Qualquer um dos pinos GPIO gerais podem ser usados para conectar o raspberry ao trigger do sensor, sendo que os pinos gerais são os que não tem nenhum subscrito como pode ser visto na figura acima. A saída Echo do sensor, que deve entrar em algum dos pinos do raspberry para ser interpretada e com isso calcular a distância do objeto ao sensor, não pode ser conectada diretamente pois o raspberry só suporta tensões de até 3.3V como entrada. Logo, com um protoboard e resistores, pode ser feito um simples divisor de tensão para conexão do pino Echo ao raspberry.

Para o divisor de tensão as resistências escolhidas foram de 1.2kΩ e 2.4kΩ. O divisor faz com que os 5V de saída do pino Echo entrem no raspberry como $5V * 2.4k/(1.2k + 2.4k) = 3,33V$, ou seja, a tensão ideal de entrada. Os pinos trigger e echo foram conectados aos pinos 4 e 17, como mostrado na figura 12.

Para controle dos pinos do GPIO através de código em Python deve-se utilizar a biblioteca *RPi.GPIO*.

```
import RPi.GPIO as GPIO
```

Figura 12 – Diagrama de ligação do sensor



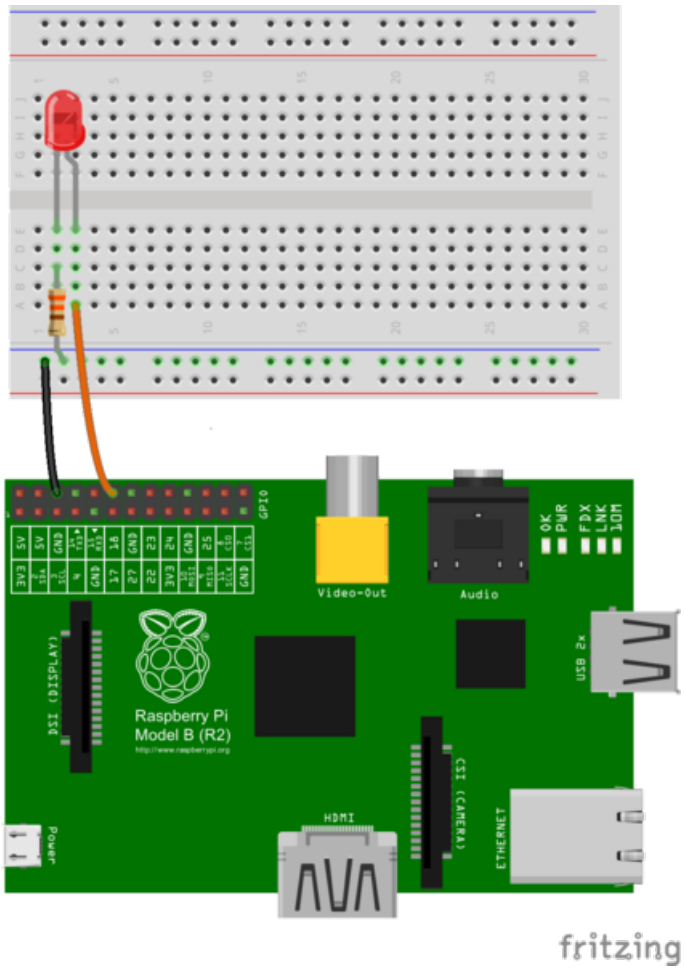
Fonte: RaspberryPiSensor (2017b)

Através de comandos utilizando esta biblioteca pode-se utilizar o pino desejado como input ou output e ler ou enviar sinais ao sensor da forma desejada.

2.4 Setup dos LEDs

O setup dos LEDs no raspberry é muito simples, basta conecta-los com seu terminal positivo a algum pino de propósito geral da GPIO, seguidos de um resistor de 330Ω ou 470Ω, ou algum valor razoavelmente próximo, e este resistor conectado ao terra da GPIO. Neste trabalho foram usados resistores de 470Ωs. Através da atribuição do pino conectado do GPIO como um output, é possível ligar ou desligar o LED. Os LEDs verde e vermelho foram conectados nos pinos 24 e 18 respectivamente.

Figura 13 – Diagrama de ligação dos LEDs



Fonte: thepihut (2017)

2.5 Algoritmo de detecção facial

O algoritmo de detecção facial utilizado neste trabalho é base para que possa ser utilizado o algoritmo de reconhecimento facial. Este algoritmo é capaz de extrair de uma imagem todas as faces presentes e retorna a posição do vértice superior esquerdo junto com a altura e largura das faces encontradas. Advindo da biblioteca do OpenCV (OPENCV, 2017), é utilizado com quatro versões de algoritmos pré-treinados para reconhecimento facial: *haarcascade_frontalface_alt.xml, haarcascade_frontalface_alt_tree.xml, haarcascade_frontalface_alt2.xml* e *haarcascade_frontalface_default.xml*. Cada versão tem sua particularidade, no caso desse projeto foi utilizado a versão que na pratica apresentou os melhores resultados para este projeto especifico, o *haarcascade_frontalface_alt_tree.xml*.

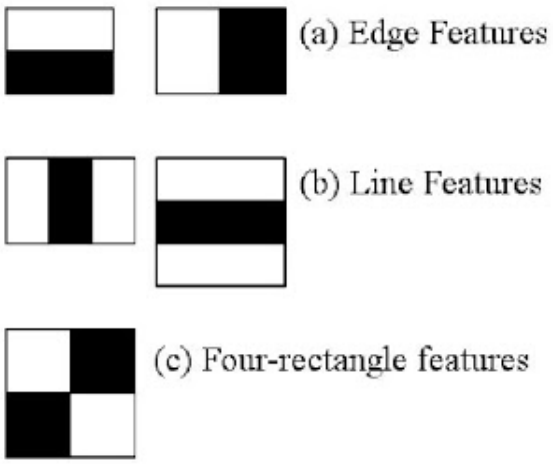
O algoritmo de detecção facial provem de um algoritmo para detecção de objetos no geral, porém foi treinado para a tarefa de reconhecimento facial. Existem versões deste algoritmo para reconhecimento de olhos, placas de carro e vários outros tipos de objeto, e

ainda é possível treinar uma versão para reconhecer o objeto desejado, qualquer este seja, mas isso foge ao escopo do trabalho.

Consultando a documentação do OpenCV (HAARCASCADES, 2017) é possível notar que o algoritmo é uma implementação do método de detecção baseado em uma cascata de classificadores, introduzido originalmente em (VIOLA; JONES, 2001). Os classificadores são responsáveis por dizer em qual classe a saída pertence em um algoritmo de aprendizado de máquina. O algoritmo faz uso de técnicas de aprendizado de máquina para treinar uma função com imagens positivas e negativas, ou seja, imagens que contém o objeto desejado e imagens que não o contém, e a partir disso, faz um modelo que permite reconhecer o objeto treinado.

O método de (VIOLA; JONES, 2001) utiliza características na forma de retângulos pretos e brancos advindos de operações de soma e subtração de pixels em regiões da imagem, em vez dos próprios pixels da imagem, como as características para o classificador realizar sua tarefa. Esses representam bordas, linhas e diagonais em uma região da imagem.

Figura 14 – Tipos de características geradas e usadas pelo classificador



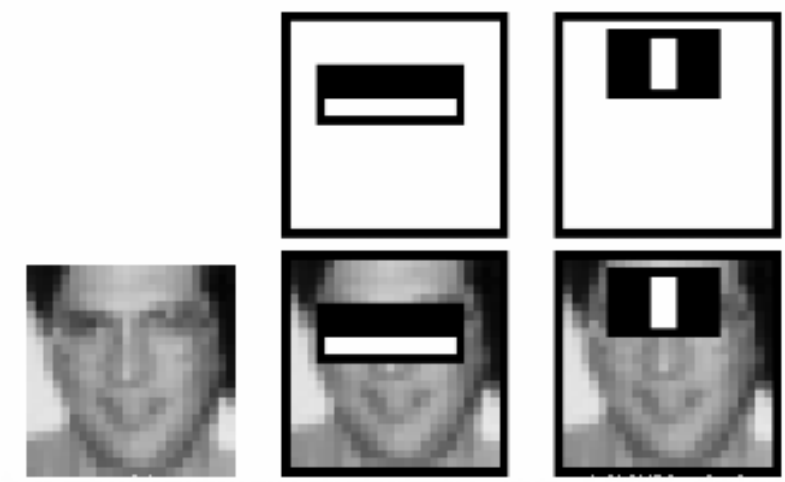
Fonte: FeatureTypes (2017)

Mesmo uma imagem pequena gera um número de características enorme e computacionalmente proibitivo, pois no processo de geração se explora todos os tamanhos e regiões de pixels para produção destas, usando janelamento da imagem com janelas de tamanho variável. Portanto, o número de características geradas é muito superior ao próprio número de pixels presentes na imagem, dado a forma como elas são extraídas. Para solucionar este problema os autores utilizaram a técnica chamada *integral images* (VIOLA; JONES, 2001), que simplifica e agiliza o processo de cálculo dessas características, fazendo com que menos computações sejam necessárias para gera-las.

Uma vez geradas, as características devem ser selecionadas, pois várias são irrelevantes dado seu enorme número. Isso é feito usando o método AdaBoost (FREUND; SCHAPIRE, 1997). Este é um método complexo, mas amplamente utilizado que utiliza vários classificadores fracos, chamados assim pois não conseguem classificar a imagem sozinhos, mas que juntos formam um classificador forte. Segundo (VIOLA; JONES, 2001), as 200 características mais relevantes selecionadas por este método, conseguem atingir 95% de acurácia na detecção do objeto desejado, sendo que os testes foram realizados utilizando faces. O autor, no entanto, sugere o uso de 6000 características para cada janela de 24x24 pixels.

As características que o método seleciona são as mais relevantes, um exemplo pode ser visto abaixo, onde é visto que a região dos olhos de uma pessoa é considerada importante para detecção facial.

Figura 15 – Características importantes selecionadas pelo AdaBoost



Fonte: Viola e Jones (2001), figura 3

O método AdaBoost é utilizado por este algoritmo tanto para selecionar as características mais relevantes quanto como classificador, uma vez as características selecionadas. E, por fim, o algoritmo introduz o conceito de *Cascade of Classifiers* - em português cascata de classificadores. Para cada região percorrida na tentativa de detecção de face, ou seja, cada janela criada da imagem, uma quantidade de características seria avaliada pelo classificador, após serem selecionadas, que embora reduzidas em quantidade ainda seriam muitas considerando percorrer toda a figura. Através da cascata de classificadores cada janela é testada em etapas, usando cada vez mais características. Isso permite que, se no estágio inicial, não se tenha características que mostrem que aquela região contém o objeto desejado, já se descarta a região, reduzindo a quantidade de processamento necessário, por não ter que testar todas as características no classificador. Isso faz com que o algoritmo foque nas regiões onde há mais chance de haverem os objetos desejados, no

caso deste trabalho as faces, e perca menos tempo com regiões onde não há faces. Isso melhora drasticamente a performance do algoritmo.

Aplicado neste trabalho, para a tarefa de detecção facial, o algoritmo de (VIOLA; JONES, 2001), apresenta ótimos resultados e é rápido o suficiente para ser processado no raspberry.

2.6 Algoritmo de reconhecimento facial

O algoritmo de reconhecimento facial que será utilizado neste trabalho também é proveniente da biblioteca do OpenCV (OPENCV, 2017). Mais especificamente ele pertence à classe de algoritmos para reconhecimento facial da biblioteca do OpenCV. A documentação do algoritmo pode ser encontrada em (FACEOPENCV, 2017) e seu nome é *LBPHFaceRecognizer*. Este está implementado como uma classe e é baseado também em técnicas de aprendizado de máquina. Uma vez criado o modelo base, ou seja, um objeto da classe, é possível utilizar funções para treinar o algoritmo com uma base de dados, salvar o algoritmo treinado, importa-lo novamente e utiliza-lo para realizar previsões.

Diferentemente do algoritmo de detecção facial, o algoritmo de reconhecimento facial não vem pré-treinado, até mesmo porque deve-se escolher o usuário desejado para ser reconhecido, que no caso deste trabalho, será o próprio autor. Por isso, é importante usar uma base de dados com faces de pessoas aleatórias e a pessoa desejada para que através destes dados o algoritmo ache padrões e aprenda a classificar corretamente a pessoa desejada como sendo da classe positiva, e estranhos como sendo da classe negativa.

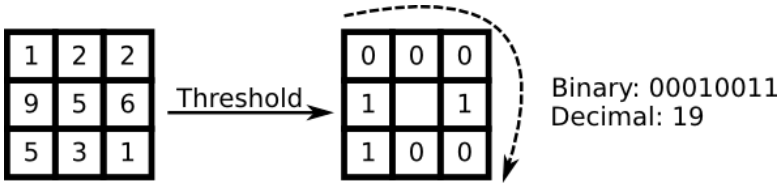
Como fica claro, a entrada para o algoritmo de reconhecimento facial são um conjunto de faces, de pessoas aleatórias e da pessoa conhecida, que possibilitam que o modelo seja treinado. Com este modelo treinado, o resultado da análise de uma nova face, desconhecida para o algoritmo, pode ser previsto e classificado. Essa classificação é feita utilizando a função *predict*, que fornece, dado a imagem testada, a classe a qual esta pertence e uma medida de confiança baseada na distância entre essa imagem e a imagem que mais se assemelha dela dentro da classe positiva ou negativa, dependendo da classificação dada.

Assim como para o algoritmo de detecção facial, o algoritmo de reconhecimento facial não utiliza os próprios pixels da imagem como características a serem treinadas no modelo. Em vez disso a imagem é descrita como um histograma baseado em texturas da imagem, utilizando a técnica de LBPH - *Local Binary Patterns Histograms* (AHONEN T.; PIETIKAINEN, 2004).

O descritor utilizado pelo algoritmo, que extrai as texturas da imagem para que posteriormente possam ser utilizadas como um vetor de características do classificador, é

uma versão modificada do descritor LBP original de (OJALA; HARWOOD, 1994), e tem o nome de *Circular LBP* ou de *Extended LBP*. Originalmente, no LBP clássico, utiliza-se janelamento, onde cada janela da imagem é dividida em células e cada pixel dessas células é comparado com os 8 vizinhos adjacentes, em formato de um círculo. A partir disso, constrói-se um código em que quando o pixel adjacente é menor que o central, atribui um 0 e quando é maior atribui um 1, girando em sentido horário ou anti-horário. Isso gera um número binário de 8 dígitos, que representa o pixel central.

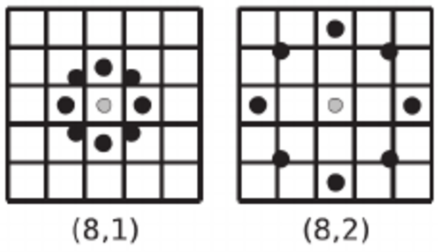
Figura 16 – Representação binária para o pixel central usando LBP



Fonte: lbptobinary (2017)

A diferença do *Extended LBP* está no fato de que o tamanho da janela, e por consequência, o tamanho da vizinhança de pixels adjacente é variável. Ou seja, o raio do círculo a partir do pixel central é variável.

Figura 17 – Raio variável do *extended LBP*



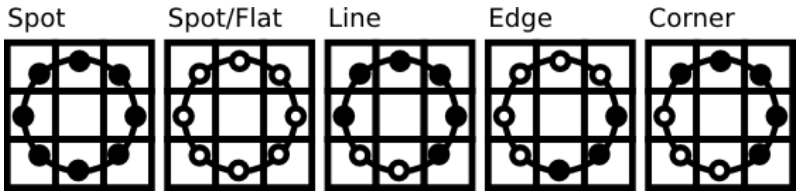
Fonte: lbptobinary (2017)

A vantagem do raio ser variável é que capta padrões que estão em escalas diferentes, como pontos, vazios, linhas, curvas e bordas.

Com os binários que representam os pixels de uma imagem é construído um histograma que serve como o vetor de características gerado, com 256 dimensões, representando a frequência onde cada pixel é maior que o adjacente. Os histogramas de todas as janelas de tamanho variável são utilizados como as observações do vetor de características 256-dimensional.

A vantagem do LBP, é sua baixa sensibilidade a diferenças de iluminação nas imagens capturadas. Geralmente, as imagens são lidas nos algoritmos de visão compu-

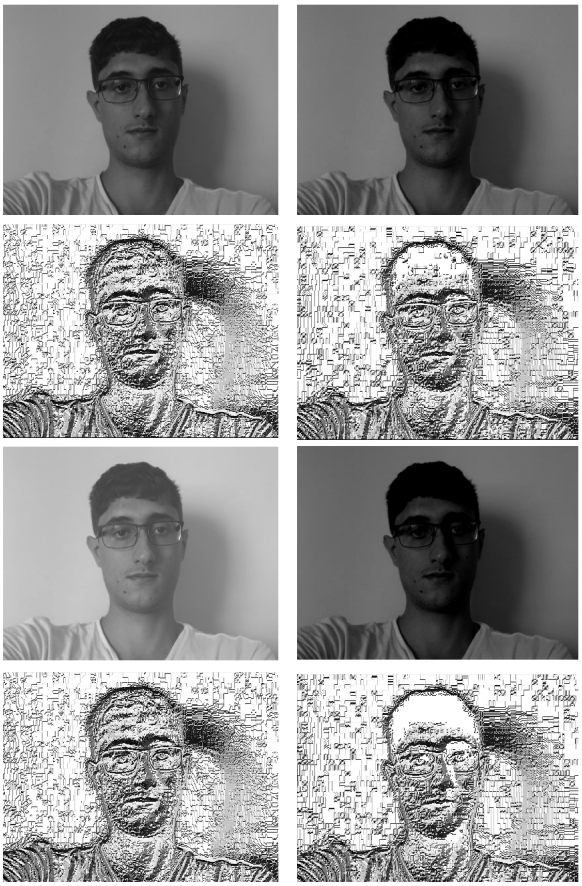
Figura 18 – Padrões captados pelo *extended LBP*



Fonte: ExtendedLBPpatterns (2017)

tacional como imagens cinza monotonicas, isso se deve ao fato de que ajuda a reduzir a dimensionalidade e complexidade. Os algoritmos de detecção e reconhecimento facial também utilizam imagens cinzas e monotonicas. Pela figura 19 pode ser visto como o LBPH é pouco sensível a diferenças de iluminação, uma vez as que as imagens passam pela transformação para tons de cinza.

Figura 19 – Baixa sensibilidade do LBP a diferenças de iluminação



Fonte: Produzido pelo autor

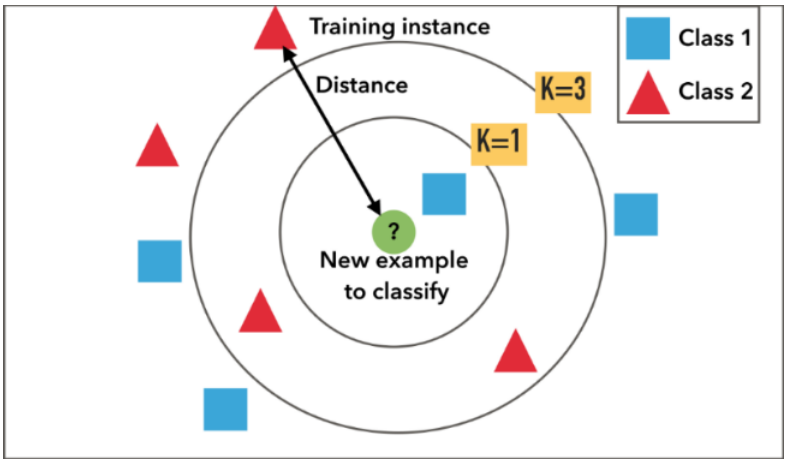
O método do LBPH no algoritmo como um todo é o que se pode chamar de

descritor, que é justamente a parte responsável por extrair as características a serem usadas. Uma vez extraídas, estas devem passar por um classificador que selecionará a classe da imagem baseada nas características dessa extraídas.

O classificador utilizado no algoritmo de reconhecimento facial *LBPHFaceRecognizer* deste trabalho é um k-NN (COVER T.M., 1967)- *k-nearest neighbors algorithm*. O k-NN é um algoritmo clássico, muito utilizado para a tarefa de classificação, porém um dos mais simples algoritmos de aprendizado de máquina.

O funcionamento do k-NN é basicamente classificar o objeto em questão como pertencente à classe de objetos mais comuns perto dele no espaço, ou seja, o objeto pertence à classe onde ele tem o maior número de vizinhos próximos. Para o k-NN utilizado no *LBPHFaceRecognizer*, k = 1, o que significa que a face a ser avaliada é classificada como pertencente à classe que tem o vizinho mais próximo.

Figura 20 – Exemplo do classíficador k-NN para duas dimensões



Fonte: SFLScientific (2017)

Embora o classificador pareça simples, quando usado em conjunção com o LBP, consegue excelente resultados, e faz isso em pouco tempo, sendo de rápida execução. Isso faz com que o algoritmo como um todo seja muito eficiente.

2.7 Implementação em código

Uma vez feito o setup do raspberry e especificados e estudados os algoritmos principais a serem utilizados, pode ser desenvolvido o código que junta as peças e torna possível a tarefa proposta de reconhecimento facial de um usuário em tempo real.

O código como um todo está dividido em vários scripts, cada um responsável por uma tarefa específica. Um script principal faz uso de todos os scripts desenvolvidos, e

é o responsável por realizar a tarefa de reconhecimento facial em tempo real. Primeiramente será detalhado o funcionamento de cada script individualmente, depois explicado o funcionamento deles como um todo, ao ser detalhado o script principal.

Todos os scripts desenvolvidos encontram-se disponíveis no Apêndice. Eles também estão em um repositório virtual no GitHub em <https://github.com/felipefreitas93/RaspiPiFaceReco>

Os scripts utilizados no desenvolvimento do trabalho, que serão analisados nas subseções subsequentes são:

- sensor.py
- ledblink.py
- variables.py
- make_captures.py
- edit_imgs.py
- train_model.py
- main.py

2.7.1 sensor.py

Esse script é responsável pelo controle do sensor de presença HC-SR04. No script é implementada uma função que recebe os pinos do sensor e uma lista em python para a distância máxima e mínima que uma pessoa deve se posicionar do sensor para que seja retirada uma foto com a câmera. Além disso recebe um parâmetro adicional opcional que é a frequência em que se checa a distância de objetos ao sensor, tendo como padrão 2 segundos.

O script implementa, através de comandos para os pinos GPIO, a lógica de ativar o Trigger do sensor com uma onda TTL de 5V, 10 µs e depois mede o tempo em nível logico alto da saída Echo. Em seguida, parte para etapa de realização das contas para cálculo da distância do objeto ao sensor, utilizando a informação anterior. Após realizar estes procedimentos o script imprime na tela a distância do objeto, e caso este esteja na distância ideal, passada na entrada da função, a função retorna verdadeiro e termina sua execução. O objetivo disso é que, ao retornar verdadeiro este valor pode ser usado para dar início as outras etapas do processo, como a retirada de fotos seguida de seu processamento até o reconhecimento facial.

2.7.2 ledblink.py

Esse script é responsável por controlar o LED, mais especificamente os momentos em que este acende. É um script simples, que recebe o pino GPIO onde o LED está conectado, o número de vezes que o LED piscará e o tempo entre cada piscar. Da mesma forma que o script para o controle do sensor, o script ledblink.py faz uso de funções da GPIO para controlar o LED. Ao ativar e desativar a entrada GPIO passada como entrada da função, ele consegue ligar e desligar o LED, e faze-lo piscar. Esse script, por receber como entrada o pino onde o LED está colocado, é capaz de ascender tanto o LED verde, para simbolizar um reconhecimento positivo, quanto o vermelho, que simboliza o oposto, simplesmente fornecendo o pino apropriado. Este script, implementado como função, não retorna nada. O simples piscar do LED já é a saída necessária.

Vale ressaltar que, foram feitos scripts para o controle do LED e do sensor, porém não foi feito o mesmo para a câmera, pois desta será usada somente uma função para tirar foto, não necessitando de uma lógica específica.

2.7.3 variables.py

Este script não tem nenhuma função além de servir como um arquivo que contém as definições e variáveis utilizadas no projeto. Seu objetivo é facilitar a mudança de parâmetros para realização de testes ou tarefas relacionadas. Ele contém informações sobre a pinagem dos LEDs, sensor, a distância de ativação do sensor, definições de nomes de pastas e arquivos gerados, por exemplo.

2.7.4 make_captures.py

Este script captura imagens do usuário que se deseja reconhecer. Estas são colocadas em um diretório, para serem utilizadas posteriormente como dados para treinar o algoritmo de reconhecimento facial.

Este script, em teoria seria dispensável se fossem retiradas e colocadas as imagens do usuário de forma manual no diretório apropriado, no entanto ele se mostra útil pois automatiza este processo. Para retirar as fotos ele faz uso da Raspberry Pi camera v2, através das funções contidas na bilbioteca *picamera*. Especificamente ele faz o uso da função *capture*, que é responsável por tirar as fotos.

2.7.5 edit_imgs.py

Uma vez que existam imagens positivas, ou seja, do usuário desejado para serem utilizadas para treinar o modelo, estas antes devem ser editadas para estarem em con-

formidade com o tipo de imagem ideal para o algoritmo de reconhecimento facial. Este script tem este objetivo, o de editar as imagens capturadas com a câmera.

O script conta com um conjunto de funções que operam em conjunto para transformar as imagens capturadas e armazenadas do usuário em faces e coloca-las em um novo diretório, para serem utilizadas pelo algoritmo de reconhecimento facial. Como entrada a função principal deste script recebe o nome do diretório onde estão as imagens a serem editadas e o nome para onde irão após serem tratadas.

Cada imagem para por uma série de procedimentos: primeiramente são transformadas para imagens em escalas de cinza, como requisito para serem avaliadas pelo algoritmo de detecção facial, e em sequência, passam pelo algoritmo de detecção facial onde são extraídas as coordenadas da face presente na imagem. Neste processo é assegurado que seja reconhecida apenas uma face, logo, para que a imagem analisada tenha sua face extraída deve-se ter apenas uma face presente na imagem. Após o processo de extração das coordenadas da face, estas são utilizadas para recortar e redimensionar a face para um tamanho padrão e por fim as faces são salvas no diretório de saída que contem imagens positivas. Neste processo o algoritmo de detecção facial foi utilizado com sua versão pré-treinada *haarcascade_frontalface_alt.xml*.

A razão pelo qual as faces são redimensionadas, é devido ao fato de que para ser treinado o algoritmo de reconhecimento facial com as faces de saída, todas devem ser do mesmo tamanho.

2.7.6 train_model.py

Uma vez presente o diretório contendo imagens positivas, que no caso deste trabalho contém 10 imagens, o script train_model é responsável por treinar um modelo para previsão de novas faces, usando as faces positivas, um conjunto de faces negativas e o algoritmo de reconhecimento facial.

A database de faces negativas utilizada foi retirada de (CAMBRIDGE, 2017), que contém 400 imagens de pessoas 40 pessoas distintas sob diferente iluminação e fazendo diferentes expressões faciais. Estas têm tamanho fixo de 92x112 pixels e são em tons de cinza. Seu tamanho propositalmente coincide com o tamanho das faces positivas, pois juntando estes conjuntos, que devem possuir o mesmo tamanho de imagem, é treinado o modelo de reconhecimento facial.

Nota-se que os dados para este script estão desbalanceados pois existem 400 observações da classe negativa e apenas 10 da classe positiva, porém, visto que o algoritmo de reconhecimento facial utiliza como classificador o 1-NN, ou seja, o k-NN com k = 1, este efeito do desbalanceamento é atenuado, devido ao funcionamento do próprio 1-NN. Em outras, palavras, o 1-NN é por natureza pouco sensível ao desbalanceamento de classes,

por isso as classes desbalanceadas como estão utilizadas não tem grande influência no resultado do reconhecimento facial.

É fácil entender porque o 1-NN é pouco sensível ao desbalanceamento de classes; como ele atribui ao novo objeto a classe equivalente ao vizinho mais próximo, não importa muito a proporção de vizinhos nos dados, mas sim sua proximidade.

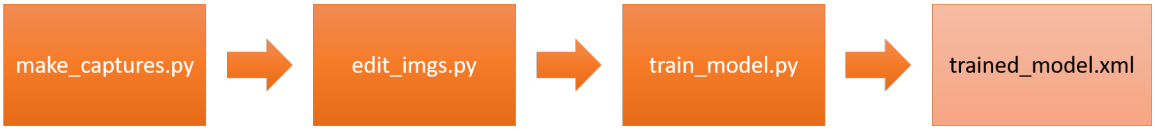
O script, utilizando os dados acima mencionados, cria um modelo e o salva como um arquivo XML, utilizando o algoritmo de reconhecimento facial descrito neste trabalho, o *LBPHFaceRecognizer*. Este modelo é o modelo de aprendizado de máquina que pode ser usado para realizar previsões para novas imagens.

2.7.7 main.py

O script main.py é responsável pelo reconhecimento facial em tempo real e usa de forma direta ou indireta todos os scripts discutidos anteriormente, é o script principal. No entanto, antes de entrar em detalhes sobre o script em si, será explicado como os scripts anteriores trabalham em conjunto para possibilitar a tarefa de reconhecimento facial em tempo real.

Os scripts *make_captures.py*, *edit_imgs.py* e *train_model.py* trabalham em conjunto para construir o modelo de reconhecimento facial, que será utilizado no main.py. O script *make_captures.py* captura as imagens do usuário a ser cadastrado e as salva em um diretório. Em seguida, o script *edit_imgs.py* transforma estas imagens em faces, através do uso do algoritmo de detecção facial, e recorta e redimensiona as faces extraídas e as coloca em um novo diretório. Por fim o script *train_model.py* extrai deste último diretório, e de um banco de fotos negativas os dados necessários para construção do modelo, e através do algoritmo de reconhecimento facial constrói o modelo para ser usado para previsão e o salva como um arquivo *.xml*. Todos estes scripts consultam o script *variables.py* para ter informações relativas ao nome dos diretórios, tamanho das imagens após serem editadas e todas outras definições relevantes.

Figura 21 – Fluxograma: scripts utilizados para construção do modelo de reconhecimento facial



Fonte: Produzido pelo autor

Este processo descrito pelo fluxograma acima é feito de maneira manual, visto que deverá ser executado somente uma vez. Além disso, o fato de ser feito manualmente

auxilia na supervisão das etapas.

Agora, explicado o processo de construção do modelo de reconhecimento facial e como os scripts realizam esta tarefa, será explicada o script principal, *main.py*.

O script *main.py* carrega o *.xml* salvo com o modelo de reconhecimento facial, *trained_model.xml*, como seu primeiro passo. Após isso executa um loop infinito, que é responsável pelo funcionamento ininterrupto do dispositivo para a tarefa de reconhecimento facial.

Dentro do loop o primeiro passo é utilizar a função *sensor.py* como desencadeador do resto do processo, isso é feito com um condicional *if*. Uma vez em que um objeto se encontrar na distância apropriada, a função retorna True e este resultado é avaliado posteriormente, permitindo a entrada para a segunda etapa, que é tirar uma foto com a câmera.

Diferentemente de como era feito nos scripts anteriores, onde as fotos tiradas iam para um diretório e depois eram lidas e processadas, no script *main.py* a foto é salva em memória virtual diretamente, para garantir um processamento mais rápido. Isso é feito com a função *cameracap*

Com a foto tirada, esta é transformada em uma imagem em tons de cinza e passa pelo mesmo procedimento utilizado no script *edit_images.py*, porém deste script agora são utilizados somente suas funções para detecção facial e redimensionamento e recorte da face, e não o scripts em si, visto que não são feitas operações em cima de um diretório inteiro. Da tarefa de detecção facial, são obtidas as coordenadas da face na imagem, caso não exista uma face ou exista mais de uma face, a função *recon_face*, responsável pela obtenção das coordenadas, e proveniente do script *edit_images.py*, retorna um valor *None*. Caso isso aconteça, o loop volta para a parte de detecção com o sensor e, caso contrário a face única encontrada é redimensionada e recortada com a função *cut_resize*.

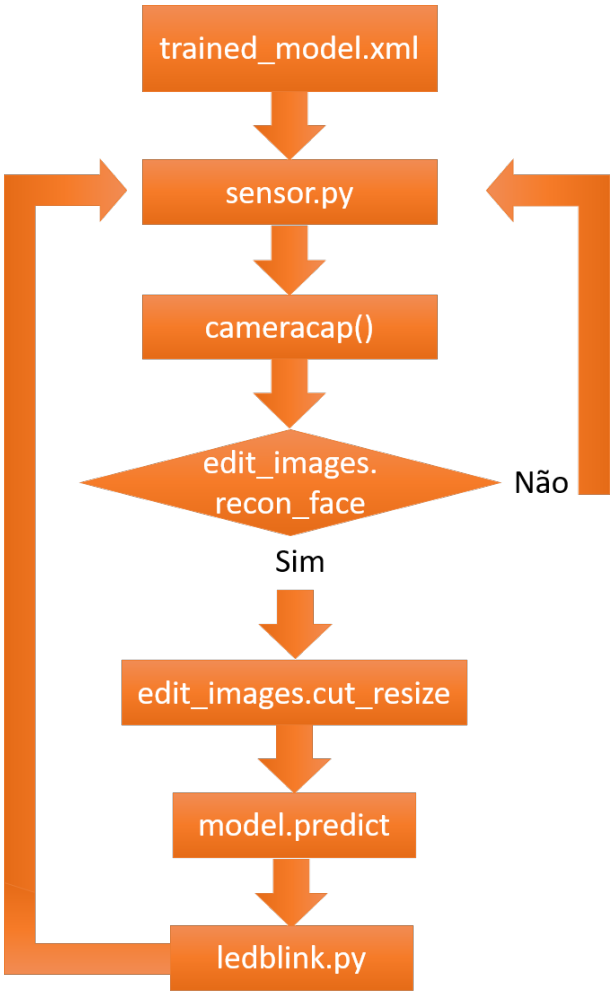
Com a face redimensionada e recortada, esta está pronta para ser testada pelo modelo treinado para reconhecimento facial, carregado no início do script. Este procedimento é feito utilizando a função *predict* do modelo salvo.

O resultado é obtido e caso seja positivo, o script *ledblink.py* é chamado com o LED verde e, caso seja negativo, é chamado com o LED vermelho. Esse procedimento simboliza se o reconhecimento foi de uma pessoa conhecida ou não.

Vale ressaltar, que o fato de durante o reconhecimento facial da foto tirada só ser aceito uma cara na imagem é planejado. Isso serve para evitar que uma pessoa ao fundo atrapalha o funcionamento do reconhecedor. E em relação ao detalhe do script *sensor.py* retornar True apenas em uma distância apropriada também é uma escolha de projeto, como descrito na seção 1.3.2.

O funcionamento do dispositivo, portanto é garantido pelo script *main.py*. Na imagem a seguir, será mostrado um fluxograma do funcionamento do script *main.py*.

Figura 22 – Fluxograma: script *main.py*, que é responsável por realizar toda a tarefa de reconhecimento facial em tempo real de um usuário cadastrado



Fonte: Produzido pelo autor

2.8 Automatização do código

Como visto na seção anterior, o script principal *main.py* é responsável e capaz de realizar a tarefa de reconhecimento facial em tempo real de um usuário cadastrado. Este faz uso dos outros scripts disponíveis para esta tarefa.

A maneira de começar o processo de reconhecimento no raspberry é rodar o script principal. Porém, para isto ser feito, é necessário o acesso ao raspberry e que manualmente se clique e rode o script.

A fim de evitar isto, de forma a garantir maior autonomia e, por consequência, portabilidade do dispositivo, foi implementado um código que atua na inicialização do sistema operacional que faz com que o script *main.py* seja executado automaticamente ao ligar o dispositivo.

Esse código é um arquivo *.service*, gerido pelo *systemclt*, que é um gerenciador do *systemd* do Linux, que por sua vez é responsável pela inicialização dos programas e do sistema operacional. Logo, criando esse arquivo, o script *main.py* é adicionado à lista de programas a serem inicializados quando o dispositivo é inicializado.

Para o funcionamento deste procedimento, o script *main.py* deve ser transformado em um executável, e deve ser colocado nele uma linha de comando acima do código, conhecida como *shebang*, que aponta o interpretador de Python para ser usado para a execução do mesmo.

A principal vantagem de realizar este procedimento, como já discutido, é que o raspberry consegue automaticamente executar o script principal, que permite o funcionamento dele como um reconhecedor de faces e para isso só é necessário conecta-lo na tomada.

Os códigos utilizados para realização deste procedimento estarão no Anexo.

2.9 Montagem e encapsulamento

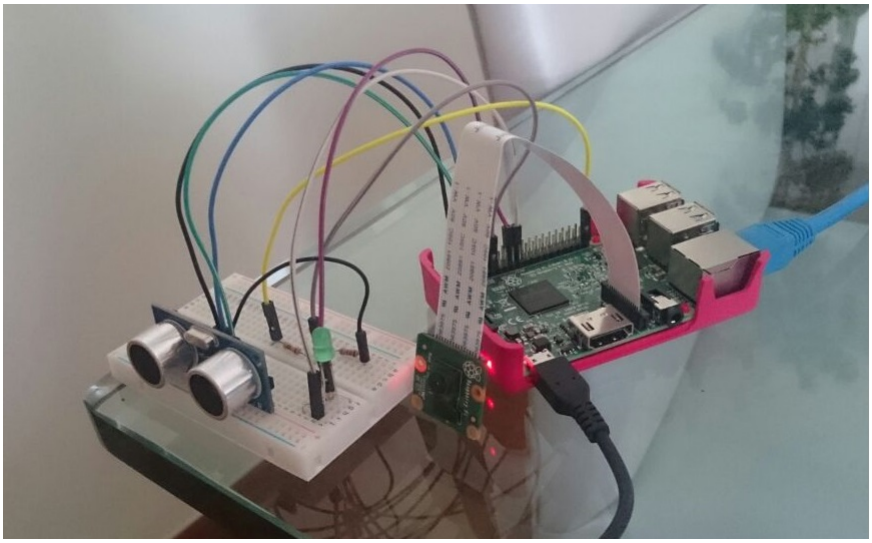
Uma vez implementado no dispositivo os códigos necessários para seu funcionamento, e tendo feito o setup dos periféricos a serem utilizados no mesmo, foi feito o encapsulamento deste em uma caixa compacta.

Foi primeiro assegurado que os componentes estavam todos encaixados perfeitamente, com os cabos ligados no GPIO e os resistores, LEDs e sensor no conectados protoboard. A figura 23 mostra o dispositivo antes de ser encapsulado, com as conexões necessárias realizadas, apenas faltando acrescentar o LED vermelho.

Após esta fase, o raspberry foi colocado dentro de uma caixa de acrílico, para sua proteção. O dispositivo foi então encapsulado utilizando madeira como material, onde foi construído à mão uma caixa de madeira pequena, de 9.7cm x 7.3cm de base e 7.4cm de altura, que depois de pronta foi pintada de preto. A figura 24 mostra o dispositivo sendo encapsulado na caixa, ainda em fase de construção.

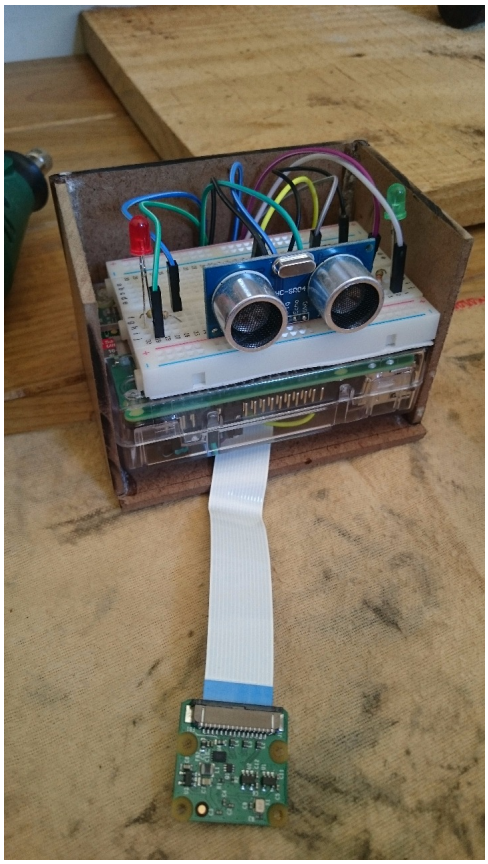
Ao final, o dispositivo foi completamente encapsulado, e na caixa foram recortados buracos para o sensor, câmera, LEDs, conexão de energia e entrada ethernet, esta última para acesso ao dispositivo com facilidade via VNC, caso necessário. O resultado pode ser visto na figura 25. É também possível ter uma noção do tamanho do dispositivo, que por ser muito portátil cabe na palma da mão.

Figura 23 – Dispositivo antes de ser encapsulado



Fonte: Produzido pelo autor

Figura 24 – Dispositivo sendo encapsulado, com a caixa em construção



Fonte: Produzido pelo autor

Figura 25 – Dispositivo encapsulado e finalizado

Fonte: Produzido pelo autor

3 Resultados

3.1 Produto Final

O produto final gerado neste trabalho é um dispositivo compacto, portátil, que realiza a tarefa de reconhecimento facial de um usuário cadastrado de forma independente e rápida. O dispositivo necessita somente de energia para funcionar, ou seja, ser conectado na tomada, o resto é feito automaticamente.

Figura 26 – Produto final

Fonte: Produzido pelo autor

De forma sucinta, o dispositivo reconhece um usuário cadastrado caso este esteja próximo, e sinaliza isto com os LEDs.

A metodologia proposta e os algoritmos utilizados geraram um reconhecedor facial robusto, que raramente classifica alguém erroneamente. O dispositivo é pouco sensível a mudanças de iluminação ambiente e os sensores e LEDs raramente falham.

Algumas ocasiões onde o dispositivo tem limitações é quando se encontra contra a luz, ou seja, com uma luz forte atrás da pessoa a ser reconhecida ou em condições de

muito pouca luz. Além disso, se a pessoa estiver em movimento rápido a imagem retirada pode ficar desfocada, embaçada e atrapalhar o processo de reconhecimento.

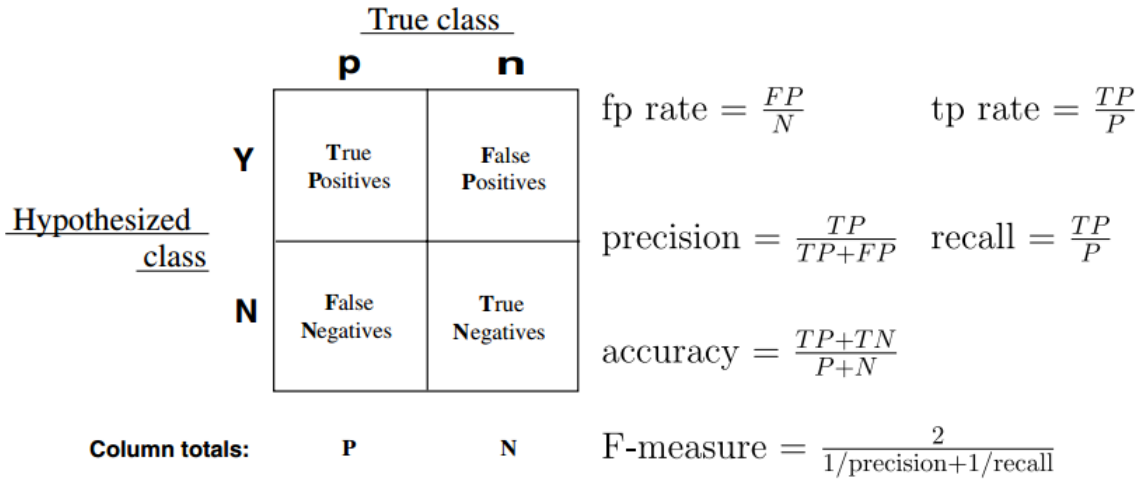
3.2 Análise curva ROC

A fim de validar a eficiência do dispositivo criado na tarefa de reconhecimento facial, verificando sua confiabilidade e a taxa de erro, será feita uma análise utilizando uma curva ROC (FAWCETT, 2006).

A curva ROC é um plot 2D que ilustra a performance de um sistema de classificação binário conforme seu limiar de classificação é variado. É uma tecnica gráfica eficaz para visualizar, organizar e selecionar e ajustar classificadores baseado na sua performance. A curva ROC testa novas observações no classificador, para à partir dessas produzir a curva que permite análisar a performance deste.

O primeiro passo para entender como analisar uma curva ROC é entender uma matriz de confusão.

Figura 27 – Matriz de confusão



Fonte: Fawcett (2006), Figura 1

Essa matriz resume todas as possibilidades a qual está sujeito uma observação classificada por qualquer tipo de classificador. A classe hipotetizada, ou seja, que o objeto foi classificado pelo classificador, quando comparada à classe real da qual aquele objeto pertence gera quatro saídas na matriz de confusão: *True Positives*, ou verdadeiros positivos, são aquelas observações previstas pelo classificador como da classe positiva e que realmente pertencem a esta, *False Positives*, ou falsos positivos são as observações previstas como positivas mas que na verdade são negativas, e ainda *False Negatives*, ou falso nega-

tivos que são previstos como negativos mas na realidade são positivos e *True Negatives*, ou negativos verdadeiros, que são previstos como negativos e de fato são negativos.

O número de verdadeiros positivos, *True Positives*, sobre o número de verdadeiros totais representa a taxa de verdadeiros positivos, *TP rate*. O número de falsos positivos, *False Positives*, sobre o número de negativos totais representa a taxa de falsos positivos, *FP rate*. Essas duas métricas são as mais comumente utilizadas para a construção da curva ROC. O outro passo necessário para construção da curva ROC é a variação do limiar utilizado para simbolizar a classe de saída do classificador.

Os classificadores, quando binários, possuem dois tipos possíveis de saída, ou fornecem uma probabilidade do objeto testado pertencer à classe positiva, ou fornecem diretamente a classe do objeto. Os primeiros, para decidir se o objeto pertence à classe positiva precisam que a probabilidade fornecida esteja acima de um certo valor, ou limiar, normalmente usado 0.5, ou seja, caso a probabilidade de pertencer à classe positiva fornecida pelo classificador seja acima do limiar, o objeto é dito pertencer à classe positiva. Os segundos utilizam métodos que diretamente associam o objeto a alguma classe, como ocorre com o 1-NN usado neste trabalho, que usa métricas de distância, por exemplo. Este tipo de classificador pode ser chamado de classificador discreto.

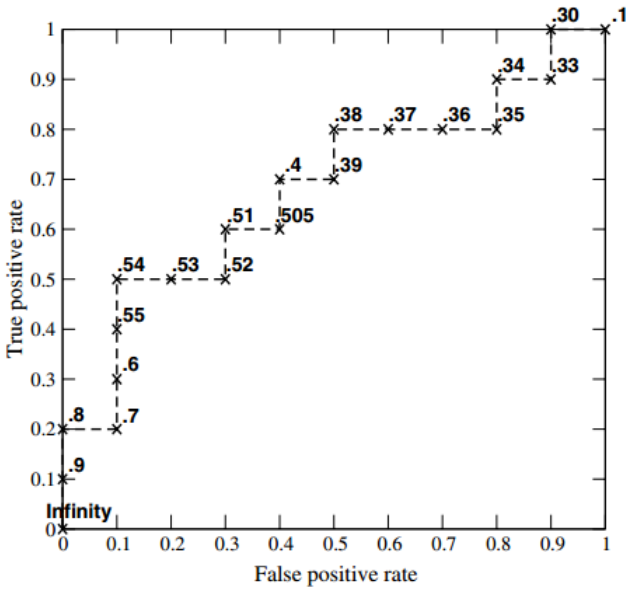
Variando o limiar do classificador, ou seja, alterando com qual probabilidade classificar as classes como positivas, muda-se a matriz de confusão para novas observações testadas no classificador, e consequentemente a taxa de verdadeiros positivos, *True Positive rate*, e falsos positivos, *False Positive Rate* para estas novas observações. Com vários valores de limiar é possível gerar uma curva no espaço, usando como ordenada a *True Positive rate*, e como abscissa a *False Positive Rate*, obtidas das diferentes matrizes de confusão, por sua vez obtidas com os novos limiares sobre o conjunto de observações inéditas. Esta é a curva ROC. Equivalentemente, para classificadores discretos é possível utilizar a *True Positive rate* e *False Positive Rate* para produzir uma saída no espaço ROC. No entanto, por não ter um limiar que permite variar a maneira como é feita a classificação, em vez de curva estes geram somente um ponto no espaço ROC, pois só geram uma matriz de confusão.

Na figura 28, é possível ver a saída do classificador, representado como o *score*, para as observações, numeradas de 1 a 20 e também a classe cuja qual elas realmente pertencem, através da coluna *Class*. A curva ROC é construída variando o limiar, que tem seu valor escolhido mostrado sobre a própria curva, sendo cada valor escolhido responsável por gerar um ponto com uma taxa de verdadeiros positivos e falsos positivos, devido à alteração da matriz de confusão causada pela mudança do limiar.

Na figura 29, vemos a saída de classificadores discretos no espaço ROC, cada classificador é representado por uma letra e fornece apenas um ponto no espaço.

Figura 28 – Curva ROC, com seus pontos gerados para diferentes valores de limiar escolhidos, plotados sobre a própria curva

Inst#	Class	Score	Inst#	Class	Score
1	p	.9	11	p	.4
2	p	.8	12	n	.39
3	n	.7	13	p	.38
4	p	.6	14	n	.37
5	p	.55	15	n	.36
6	p	.54	16	n	.35
7	n	.53	17	p	.34
8	n	.52	18	n	.33
9	p	.51	19	p	.30
10	n	.505	20	n	.1

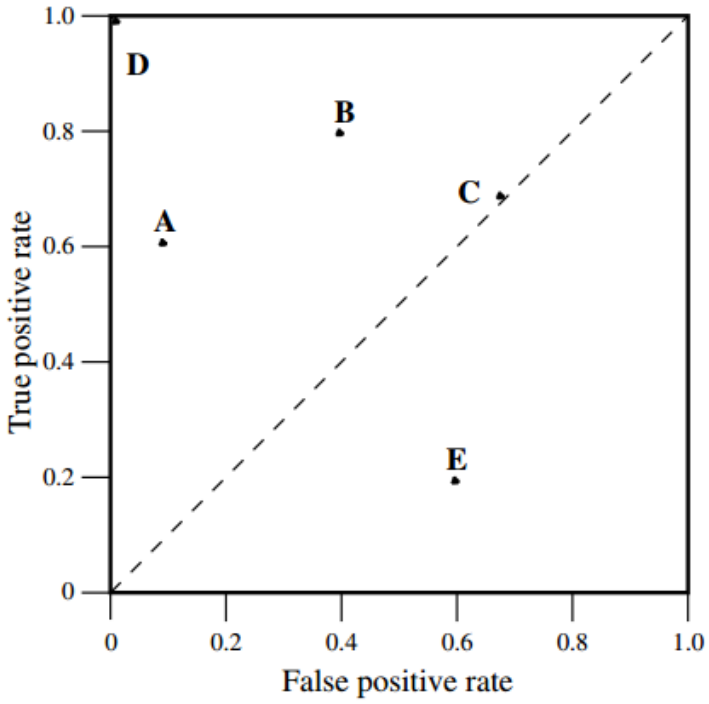


Fonte: Fawcett (2006), Figura 3

Na curva ROC, o quanto mais acima e a esquerda o ponto estiver, melhor sua performance. Pois significa que tem mais positivos verdadeiros e menos falsos positivos. A linha diagonal na curva ROC representa o classificador aleatório, se um classificador atribui aleatoriamente a classe positiva para metade das observações, ele corresponde ao ponto (0.5,0.5) no espaço ROC. Se ele classifica a classe positiva para 90% das observações, ele provavelmente acertará 90% das observações que serão realmente positivas, mas ao mesmo tempo sua taxa de falsos positivos também deve ser em torno de 90%, o que o coloca no ponto (0.9,0.9) no espaço ROC (FAWCETT, 2006).

Neste trabalho, a saída final do processo de detecção facial é por parte do algo-

Figura 29 – Curva ROC para classificadores discretos



Fonte: Fawcett (2006), Figura 2

ritmo de reconhecimento facial - *LBPHFaceRecognizer* e antes disso a imagem retirada é classificada pelo detector facial. Após ser reconhecida a face, na saída final, o *LBPHFaceRecognizer* usando como classificador o 1-NN, fornece a classe da imagem analisada diretamente, de forma discreta, logo a representação do sistema no espaço ROC seria um único ponto. Entretanto, junto com a classe de saída, o 1-NN fornece uma confiança, que representa a distância desta imagem à imagem mais próxima utilizada durante o treinamento, que foi responsável por classificar a imagem avaliada como pertencente à classe desta. Esta informação pode ser transformada em uma probabilidade, de forma a possibilitar a produção de uma curva no espaço ROC.

A métrica de distância fornecida ao final do *LBPHFaceRecognizer*, junto com a classe, representa o quão próximo da sua classe, positiva ou negativa, a imagem analisada está. Portanto, uma imagem com um valor de distância alto na classe negativa representa uma imagem que estaria em termos de probabilidade de pertencer à classe positiva, próxima à 0.5 pela esquerda. Equivalentemente uma imagem com distancia elevada, porém pertencente a classe positiva também estaria próxima de 0.5 de probabilidade de pertencer à classe positiva, porém pela direita. Uma imagem com distancia que tende a zero, da classe negativa, corresponde a uma imagem com probabilidade próximo de 0 de pertencer a classe positiva e uma imagem com distancia que tende a zero da classe positiva,

corresponde a uma imagem com probabilidade próximo de 1 de pertencer a classe positiva.

A partir disso, foram criadas duas equações, uma para cada classe, que mapeiam as distancias em termos de probabilidade, a fim de produzir uma curva ROC que poderá avaliar o classificador e descobrir um bom limiar para a operação do dispositivo.

Uma vez descoberto este limiar, a porcentagem encontrada pode ser convertida de volta em distância, e então o classificador pode ser utilizado com esta como limiar para o reconhecimento, de forma a potencializar sua taxa de verdadeiros positivos e reduzir a taxa de falsos positivos em sua operação.

Equação para a classe positiva:

$$Probabilidade = (2 * distancia_{max} - distancia)/(2.0 * distancia_{max})$$

Equação para a classe negativa:

$$Probabilidade = distancia/(2.0 * distancia_{max})$$

Essas equações funcionam transformando a distância fornecida na classe positiva como uma probabilidade de 0.5 a 1, e para a classe negativa uma probabilidade de 0 a 0.5, seguindo a lógica descrita acima e usando o máximo valor de distância encontrado com as observações testadas no classificador.

As observações testadas são provenientes de uma base com 50 imagens, 25 da classe positiva, explorando diferentes expressões faciais e iluminação e 25 da classe negativa, contendo 25 imagens de pessoas diferentes com face e iluminação padrão, ou seja, expressão facial neutra e luz indireta retiradas de (UTRECHT, 2017).

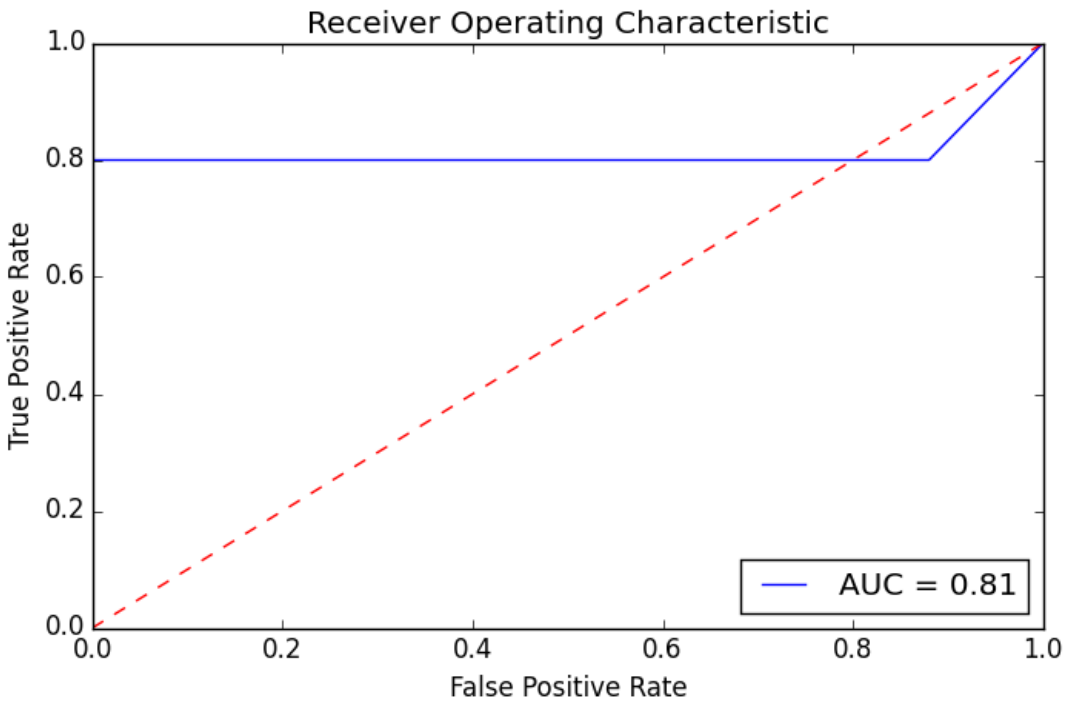
Adicionalmente, caso uma imagem não passasse pela primeira etapa no processo de reconhecimento facial, ou seja, se não passasse pela detecção facial, a essa era atribuído classe negativa com distancia 0, ou seja, em porcentagem isso significaria 0 de chance de pertencer à classe positiva.

Então, usando estas imagens como observações, as equações anteriormente descritas para transformar a distância em porcentagem e o pacote *sklearn.metrics* em Python, foi produzida a curva ROC, mostrada na figura 30, que representa o característica de funcionamento do dispositivo.

A partir da curva ROC, podemos ver que o dispositivo é bem rosbusto. Existe uma região no canto esquerdo superior onde o dispositivo apresenta uma taxa de verdadeiros positivos alta, de 0.8 e 0 falsos postivos.

Esta região, coincidentemente, corresponde ao limiar de 0.5 e, ao fazer a conversão desta porcentagem novamente para distância, percebe-se que este limiar representa a distância máxima de saída. Isto por sua vez significa que o dispositivo já está no ponto

Figura 30 – Curva ROC do dispositivo criado



Fonte: Produzido pelo autor

ótimo da curva ROC, pois não existe uma distância que limita um valor máximo de distância aceitável para que a imagem seja prevista como a classe positiva, portanto qualquer distância, desde que pertencente à classe desejada, representa bem a classe desejada e gera um número baixo de falsos positivos. Isso significa que não é necessário regular o dispositivo com um novo limiar para a distância pois ele já está em sua faixa de melhor operação.

Estes resultados atestam o bom funcionamento do dispositivo, principalmente no quesito de evitar falsos positivos, que para o caso deste trabalho seriam uma falha grave.

Dado ao fato do número de observações utilizadas na análise ter sido relativamente baixo, possuindo apenas 50 imagens, os resultados mostraram a ausência de falsos positivos para o melhor limiar. Entretanto, em situações reais, verifica-se que é possível que ocorram falsos positivos, no entanto estes são muito poucos e tendem a 0, como pode ser atestado pela curva ROC.

4 Conclusões

Neste trabalho foi desenvolvido um dispositivo capaz de realizar a tarefa de reconhecimento facial de um usuário cadastrado em tempo real utilizando algoritmos de aprendizado de máquina e visão computacional. Estes foram implementados em um raspberry pi, que tinha como periféricos uma camera, um sensor de presença e LEDs.

O dispositivo atendeu todos os objetivos a este propostos, sendo portátil, eficiente e confiável. Além disso o dispositivo é faz sua tarefa de forma automática, necessitando apenas de energia para sua operação, não necessita de ser conectado à rede e faz todo seu processamento localmente.

Este trabalho mostra que é possível embutir em um hardware muito compacto algoritmos extremamente eficientes que realizam tarefas inteligentes. Além disso, este trabalho mostra o enorme potencial presente hoje em dia no estudo e desenvolvimento de algoritmos de aprendizado de máquina, visão computacional e inteligência artificial.

4.1 Trabalhos futuros

Como trabalhos futuros, o dispositivo poderia ser adaptado para ser utilizado como um dispositivo IOT, de forma que o resultado do reconhecimento fosse transmitido via rede ou via internet. Além disso, mais de um usuário poderia ser cadastrado e fazer com que as presenças de determinados usuários, em determinados momentos, pudessem ser usadas como dados para algum outro propósito. Como uma outra ideia, a saída do dispositivo poderia ser utilizada para permitir acesso a determinada porta, ou armário. O dispositivo ainda poderia ser adaptado para ser utilizado como dispositivo de controle de ponto em um escritório.

Em relação a análise do dispositivo como este está implementado, mais testes poderiam ser feitos, com imagens sob condições mais adversas para testar a robustez do dispositivo em situações pouco comuns. Além disso, baseado no resultado dos testes, novos algoritmos, mais complexos, poderiam ser implementados para substituir os existentes na busca de mais rapidez ou performance.

Referências

AHONEN T., H. A.; PIETIKAINEN, M. Face recognition with local binary patterns. *Computer Vision - ECCV*, p. 469–481, 2004. Citado na página 36.

AMAZON, M. *Training ML Models*. 2017. Disponível em: <http://docs.aws.amazon.com/machine-learning/latest/dg/training-ml-models.html>. Citado na página 26.

ARM. *ARM Processor Architecture*. 2017. Disponível em: <https://www.arm.com/products/processors/instruction-set-architectures/index.php>. Citado na página 21.

CAMBRIDGE, A. L. *AT&T Database of Faces*. 2017. [Online;Acessado em 03 de junho de 2017]. Disponível em: <http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html>. Citado na página 42.

CAMERAMODULE. *PiCamera connected to Raspberry*. 2017. [Online;Acessado em 02 de junho de 2017]. Disponível em: <https://cdn-shop.adafruit.com/1200x900/3099-02.jpg>. Citado na página 29.

COVER T.M., H. P. Nearest neighbor pattern classification. *IEEE Trans. Inform. Theory, IT*, p. 21–27, 1967. Citado na página 39.

CS231, S. *CS231n: Convolutional Neural Networks for Visual Recognition*. 2017. Disponível em: <http://cs231n.github.io/convolutional-networks/>. Citado na página 19.

DEBIAN. *Debian*. 2017. Disponível em: <https://www.debian.org/index.pt.html>. Citado na página 21.

EVAN-AMOS. *Raspberry Pi 3 model B*. 2017. [Online;Acessado em 30 de maio de 2017]. Disponível em: <https://commons.wikimedia.org/w/index.php?curid=49898536>. Citado na página 22.

EXTENDEDLBPPATTERNS. *Extended LBP neighborhood patterns*. 2017. [Online;Acessado em 03 de junho de 2017]. Disponível em: <http://docs.opencv.org/2.4/_images/patterns.png>. Citado na página 38.

FACEOPENCV. *LBPH face recognizer OpenCV*. 2017. Disponível em: <http://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html#ahp04>. Citado na página 36.

FAWCETT, T. An introduction to roc analysis. *Pattern Recognition*, v. 27, p. 861–874, 2006. Citado 3 vezes nas páginas 50, 52 e 53.

FEATURETYPES. *Features for the cascade*. 2017. [Online;Acessado em 03 de junho de 2017]. Disponível em: <http://docs.opencv.org/trunk/haar_features.jpg>. Citado na página 34.

FREUND, Y.; SCHAPIRE, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, Elsevier, v. 55, n. 1, p. 119–139, 1997. Citado na página 35.

HAARCASCADES. *Face Detection using Haar Cascades*. 2017. Disponível em: <http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html>. Citado na página 34.

HCSR04. *Sensor Ultrassonico HC-SR04*. 2017. [Online;Acessado em 30 de maio de 2017]. Disponível em: <http://s3.amazonaws.com/img.iluria.com/product/6B8A2/2AE50F/450xN.jpg>. Citado na página 23.

LBPTOBINARY. *Binary representation LBP*. 2017. [Online;Acessado em 03 de junho de 2017]. Disponível em: <http://docs.opencv.org/2.4/_images/lbp.png>. Citado na página 37.

MATPLOTLIB. *Matplotlib*. 2017. Disponível em: <https://matplotlib.org/>. Citado na página 28.

MICROSOFT. *WindowsIOT*. 2017. Disponível em: <https://developer.microsoft.com/pt-br/windows/iot>. Citado na página 21.

NUMPY. *NumPy*. 2017. Disponível em: <http://www.numpy.org/>. Citado na página 28.

OJALA, M. P. T.; HARWOOD, D. Performance evaluation of texture measures with classification based on kullback discrimination of distributions. *Proceedings of the 12th IAPR International Conference on Pattern Recognition*, v. 1, p. 582 – 585, 1994. Citado na página 37.

OPENCV. *OpenCV*. 2017. Disponível em: <http://opencv.org/>. Citado 8 vezes nas páginas 20, 21, 23, 24, 26, 27, 33 e 36.

PICAMDOCUMENTATION. *picamera documentation*. 2017. Disponível em: <https://picamera.readthedocs.io/en/release-1.13/>. Citado na página 30.

PICAMERA. *The Raspberry Pi Camera Module v2*. 2017. [Online;Acessado em 30 de maio de 2017]. Disponível em: <https://www.raspberrypi.org/app/uploads/2016/02/Pi-Camera-web.jpg>. Citado na página 22.

PUTTY. *PuTTY*. 2017. Disponível em: <http://www.putty.org/>. Citado na página 28.

PYTHON. *Python*. 2017. Disponível em: <https://www.python.org/>. Citado 3 vezes nas páginas 21, 23 e 24.

R. *R language*. 2017. Disponível em: <https://www.r-project.org/>. Citado na página 23.

RASPBERRYPI, F. *Raspberry Pi*. 2017. Disponível em: <https://raspberrypi.org/>. Citado 3 vezes nas páginas 20, 27 e 28.

RASPBERRYPISENSOR. *Physical Computing with the Raspberry Pi*. 2017. [Online;Acessado em 03 de junho de 2017]. Disponível em: <https://www.raspberrypi.org/learning/physical-computing-with-python/distance/>. Citado na página 30.

RASPBERRYPISENSOR. *Wiring of the HC-SR04 and Raspberry Pi*. 2017. [Online;Acessado em 03 de junho de 2017]. Disponível em: <https://www.raspberrypi.org/learning/physical-computing-with-python/distance/>. Citado na página 32.

RASPBERRYPISPY. *GPIO layout*. 2017. [Online;Acessado em 03 de junho de 2017]. Disponível em: <http://www.raspberrypi-spy.co.uk/wp-content/uploads/2012/06/Raspberry-Pi-GPIO-Layout-Model-B-Plus-rotated-2700x900.png>. Citado na página 31.

SCIKIT-LEARN. *Scikit-Learn*. 2017. Disponível em: <http://scikit-learn.org/stable/>. Citado na página 27.

SFLSCIENTIFIC. *k-NN example in 2D*. 2017. [Online;Acessado em 03 de junho de 2017]. Disponível em: <https://static1.squarespace.com/static/55ff6aece4b0ad2d251b3fee/t/5752540b8a65e246000a2cf9/1465017829684/>. Citado na página 39.

SHAVIT, A. *OpenCV logo*. 2017. [Online;Acessado em 30 de maio de 2017]. Disponível em: <https://commons.wikimedia.org/w/index.php?curid=26547749>. Citado na página 24.

THEPIHUT. *LED setup*. 2017. [Online;Acessado em 03 de junho de 2017]. Disponível em: <https://thepihut.com/blogs/raspberry-pi-tutorials/27968772-turning-on-an-led-with-your-raspberry-pis-gpio-pins>. Citado na página 33.

UTRECHT, E. C. on V. P. *Utrecht ECVP database of faces*. 2017. [Online;Acessado em 03 de junho de 2017]. Disponível em: <http://pics.psych.stir.ac.uk/2D_face_sets.htm>. Citado na página 54.

VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. In: . [S.l.: s.n.], 2001. p. 511–518. Citado 3 vezes nas páginas 34, 35 e 36.

VNC. *VNC*. 2017. Disponível em: <https://www.realvnc.com/>. Citado na página 28.

WIKIPEDIA. *Python Logo*. 2017. [Online;Acessado em 31 de maio de 2017]. Disponível em: <https://upload.wikimedia.org/wikipedia/commons/thumb/f/f8/Python_logo_and_wordmark.svg/2000px-Python_logo_and_wordmark.svg.png>. Citado na página 24.

Apêndices

APÊNDICE A – Código utilizado no desenvolvimento

A.1 sensor.py

```
"""
Felipe Freitas de Carvalho
TCC - Reconhecimento facial utilizando o raspberry pi
Scrip que contem a funcao que ativa o sensor ultrassonico e calcula
a distancia de um objeto deste. Caso este objeto esteja entre valores
desejados a funcao retorna True
"""

import variables
import RPi.GPIO as GPIO
import time


def sensor(trig,echo,distance,sleep=2):

  #modo de chamar os pinos
  GPIO.setmode(GPIO.BCM)
  #setar os pinos como entrada ou saida
  GPIO.setwarnings(False)
  GPIO.setup(trig,GPIO.OUT)
  GPIO.setup(echo,GPIO.IN)

  while True:
    #desliga o trigger
    GPIO.output(trig, False)
    time.sleep(sleep)
    #pulso de 10us no trigger para medir a distancia
    GPIO.output(trig, True)
    time.sleep(0.00001)
    GPIO.output(trig, False)
```

```
    #duracao do sinal ir e voltar
    while GPIO.input(echo)==0:
      inicio = time.time()
    while GPIO.input(echo)==1:
      final = time.time()
    duracao = final - inicio

    distancia = duracao * 17150
    print(distancia)

    if(distancia>distance[0] and distancia<distance[1]):
      print("Distancia de {} cm".format(distancia))
      return True



if __name__=="__main__":
  a = sensor(variables.trig,variables.echo,variables.distance)
```

A.2 ledblink.py

```
import variables

def ledblink(pin,n,t):
    '''Funcao que faz um led piscar, pin e o pino GPIO onde o led esta
    conectado, n e o numero de vezes que o led pisca e t e o tempo entre
    cada piscada do led
    '''
    import RPi.GPIO as GPIO
    import time
    GPIO.setmode(GPIO.BCM)

    GPIO.setwarnings(False)

    GPIO.setup(pin,GPIO.OUT)
    GPIO.output(pin,False)
    for i in range(1,n+1):
        GPIO.output(pin,True)
        time.sleep(t/2)
```

```
        GPIO.output(pin,False)
        time.sleep(t/2)

if __name__ == '__main__':
    ledblink(variables.ledpingreen,variables.ntimes,variables.tled)
```

A.3 variables.py

```
"""
Felipe Freitas de Carvalho
TCC - Reconhecimento facial utilizando o raspberry pi
Arquivo que contem variaveis pertinentes ao projeto para serem usadas em outros
scrips e funcoes
"""

#caminho para as capturas com a camera (manualmente ou pelo script)
path_capture = './training/capture'
#exemplos positivos e negativos
path_positive = './training/positive'
path_negative = './training/negative'
#labels para exemplos positivos e negativos
positive_label = 1
negative_label = 0
#extrator de face pre-treinado
cascade = 'haarcascade_frontalface_alt.xml'
#parametros padrao para o cascade
haar_scale_factor  = 1.1
haar_min_neighbors = 5
haar_min_size      = (30, 30)
#tamanho padrao das imagens das faces (que deve ser consistente com a database
#de faces que servirao de exemplos negativos) - AT&T database
#http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html
face_width  = 92
face_height = 112
#modelo trainado salvo
trained_model = 'trained_model.xml'
#GPIO pinos usados para o sensor
trig = 4
echo = 17
```

```
#distancia limite inferior e superior
distance = [20,45]
#pino do led verde
ledpingreen = 24
ledpinred = 18
#n de vezes que o led pisca
ntimes = 3
#tempo entre cada piscada
tled = 0.3
```

A.4 make_captures.py

```
"""
Felipe Freitas de Carvalho
TCC - Reconhecimento facial utilizando o raspberry pi
Scrip para capturar imagens para a pasta capture que serao utilizadas
posteriormente como o grupo de imagens positivas
"""

if __name__ == "__main__":
    from picamera import PiCamera
    camera = PiCamera()
    from time import sleep
    import variables
    import os

    if not os.path.exists(variables.path_capture):
        os.makedirs(variables.path_capture)


    #dependendo da orientacao da camera e necessario vira-la verticalmente
    camera.vflip = True

    count_cap = 0
    while True:
        decisao = input("Digite 1 para capturar uma imagem ou 2 para sair: ")
        if decisao == 1:
            sleep(1)
            camera.capture(variables.path_capture + '/capture' + str(count_cap) + '.jpg'
```

```
            count_cap += 1
            print("{} imagens ja capturadas".format(count_cap))
        elif decisao == 2:
            print("Verifique as imagens capturadas em {}".format(variables.path_cap
            break
        else:
            print("Opcao invalida, digite novamente")
            continue
```

A.5 edit_imgs.py

```
"""
Felipe Freitas de Carvalho
TCC - Reconhecimento facial utilizando o raspberry pi
Scrip para editar as imagens contidas na pasta .training/capture para que estas
estejam adequadas para serem treinadas
"""
import variables
import cv2
import os

def recon_face(image):
    '''Retorna as coordenadas, largura e altura de uma cara
detectada, caso mais de uma cara seja detectada retorna None
'''
    haar = cv2.CascadeClassifier(variables.cascade)

    coords = haar.detectMultiScale(image,
                                   scaleFactor=variables.haar_scale_factor,
                                   minNeighbors=variables.haar_min_neighbors,
                                   minSize=variables.haar_min_size,
                                   flags=cv2.CASCADE_SCALE_IMAGE)
    if len(coords) != 1:
        return None
    return coords[0]

def cut_resize(image, x, y, w, h):
    '''Recorta a cara da imagem, definida pelas coords (x,y,w,h)
sendo (x,y) o canto esquero superior. Alem disso, mantem o mesmo
```

```
aspect ratio das imagens negativas usadas para treinamento. Depois,
altera o tamanho da imagem para o mesmo das imagens negativas
'''
    height = int((variables.face_height / float(variables.face_width)) * w)
    meioy = int(y + h/2)
    y1 = int(max(0, meioy - height/2))
    y2 = int(min(image.shape[0]-1, meioy + height/2))
    image = image[y1:y2, x:x+w]
    image = cv2.resize(image,(variables.face_width,variables.face_height),
                       interpolation=cv2.INTER_LANCZOS4)
    return image

def editimgs(pathcap,pathpos):
    count = 1
    for root, dirs, files in os.walk(pathcap,pathpos):
        for filename in files:
            fullpath = root + '/' + filename
            image = cv2.imread(fullpath)
            image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
            coords = recon_face(image)
            if coords !=None:
                x,y,w,h = coords
                image = cut_resize(image,x,y,w,h)
                writepath = pathpos + '/' + str(count) + '.pgm'
                print(writepath)
                count += 1
                cv2.imwrite(writepath,image)
            else:
                print('imagem em {} nao teve a face reconhecida'.format(fullpath))



if __name__ == '__main__':
    #Cria o diretorio para as imagens positivas caso este
    #nao exista ainda
    if not os.path.exists(variables.path_positive):
        os.makedirs(variables.path_positive)

    editimgs(variables.path_capture, variables.path_positive)
```

A.6 train_model.py

```
"""
Felipe Freitas de Carvalho
TCC - Reconhecimento facial utilizando o raspberry pi
Scrip cujo o objetivo e treinar um modelo e salva-lo como um arquivo
xml utilizando as imagens positivas e negativas contidas na pasta training
"""

import cv2
import os
import variables
import numpy



def train_model():

    examples = []
    labels = []


    count_pos = 0
    for root, dirs, files in os.walk(variables.path_positive):
        for filename in files:
            fullpath = root + '/' + filename
            image = cv2.imread(fullpath, cv2.IMREAD_GRAYSCALE)
            examples.append(numpy.asarray(image))
            labels.append(1)
            count_pos += 1
    print("Foram obtidas {} faces positivas".format(count_pos))

    count_neg = 0
    for root, dirs, files in os.walk(variables.path_negative):
        for subdirs in dirs:
            allpaths = root + '/' + subdirs
            for filename in os.listdir(allpaths):
            #each image in path (eiip)
                eiip = allpaths + '/' + filename
                image = cv2.imread(eiip,cv2.IMREAD_GRAYSCALE)
```

```
                examples.append(numpy.asarray(image))
                labels.append(0)
                count_neg += 1
    print("Foram obtidas {} faces negativas".format(count_neg))

    print("Treinando modelo...")
    #model = cv2.createEigenFaceRecognizer()
    model = cv2.createLBPHFaceRecognizer()
    model.train(examples,numpy.asarray(labels))
    model.save(variables.trained_model)

    print("Modelo salvo como {}".format(variables.trained_model))

if __name__ == '__main__':
    train_model()
```

A.7 main.py

```
#!/usr/bin/python2

import picamera
import cv2
import variables
import numpy as np
import io
import time
import edit_imgs
import sensor
import ledblink


def cameracap():
    stream = io.BytesIO()
    with picamera.PiCamera() as camera:
        #dependendo da orientacao da camera e necessario vira-la verticalmente
        #camera.vflip = True
        camera.capture(stream, format='jpeg')
    data = np.fromstring(stream.getvalue(), dtype=np.uint8)
    image = cv2.imdecode(data, 1)
```

```
    return image


if __name__ == "__main__":
    print("Scrip para reconhecimento facial de um usuario cadastrado")
    print("Para sair do scrip digite ctrl-c")
    #model = cv2.createEigenFaceRecognizer()
    model = cv2.createLBPHFaceRecognizer()
    model.load(variables.trained_model)
    print("Modelo carregado")
    while True:
        a = sensor.sensor(variables.trig,variables.echo,variables.distance)
        if a == True:
            image = cameracap()
            cv2.imwrite("last_capture.jpg",image)
            print("Imagem capturada")
        #cv2.imshow("image",image)
        #cv2.waitKey(0) & 0xFF
        #cv2.destroyAllWindows()
            image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
            face = edit_imgs.recon_face(image)
            if face is None:
                print("Nao foi detectada nenhuma face ou foi detectado mais de um u
                continue
            x, y, w, h = face
            image = edit_imgs.cut_resize(image, x, y, w, h)
            print("Face detectada, reconhecendo...")
            label, confidence = model.predict(image)
            print("Imagem reconhecida como pertencente a class {} com confianca {}"
            if label == 1:
                ledblink.ledblink(variables.ledpingreen,variables.ntimes,variables.
            elif label == 0:
                ledblink.ledblink(variables.ledpinred,variables.ntimes,variables.tl
```

APÊNDICE B – Código utilizado para a automatização do dispositivo

B.1 script.service

```
[Unit]
Description=AutoStartScript
After=multi-user.target

[Service]
Type=simple
ExecStart=/usr/bin/python /home/pi/Desktop/TCC/main.py
WorkingDirectory=/home/pi/Desktop/TCC

[Install]
WantedBy=multi-user.target
```

B.2 Comandos

```
$ chmod +x /home/pi/Desktop/TCC/main.py

$ sudo systemctl enable script.service
$ sudo systemctl daemon-reload
$ sudo systemctl start script.service
$ sudo systemctl status script.service
```

APÊNDICE C – Código utilizado para a criação da curva ROC

C.1 make_captures_roc.py

```
"""
Felipe Freitas de Carvalho
TCC - Reconhecimento facial utilizando o raspberry pi
Scrip para capturar imagens para a pasta roc que serao utilizadas
para construir a curva ROC para analise do algoritmo
"""

from picamera import PiCamera
import cv2
import os
import io
from time import sleep
import numpy as np

path = "/home/pi/Desktop/TCC/roc/roccapture"


if __name__ == "__main__":
    if not os.path.exists(path):
        os.makedirs(path)

    camera = PiCamera()

    count_cap = 0
    while True:
        decisao = input("Digite 1 para capturar uma imagem ou 2 para sair: ")
        if decisao == 1:
            sleep(1)
            camera.capture(path + '/capture' + str(count_cap) + '.jpg')
            count_cap += 1
            print("{} imagens ja capturadas".format(count_cap))
```

```
        elif decisao == 2:
            print("Verifique as imagens capturadas em {}".format(path))
            break
        else:
            print("Opcao invalida, digite novamente")
            continue
```

C.2 buildroc.py

```
"""
Felipe Freitas de Carvalho
TCC - Reconhecimento facial utilizando o raspberry pi
Scrip para realizar todos os passos necessarios para construcao da curva ROC,
que envolvem o tratamento das imagens a serem utilizados e a passagem das
mesmas pelo detector.
"""

from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
import variables
import cv2
import os
import numpy as np


def recon_face(image):
    '''Retorna as coordenadas, largura e altura de uma cara
detectada, caso mais de uma cara seja detectada retorna None
'''
    haar = cv2.CascadeClassifier(variables.cascade)

    coords = haar.detectMultiScale(image,
                                   scaleFactor=variables.haar_scale_factor,
                                   minNeighbors=variables.haar_min_neighbors,
                                   minSize=variables.haar_min_size,
                                   flags=cv2.CASCADE_SCALE_IMAGE)
    if len(coords) != 1:
        return None
    return coords[0]
```

```
def cut_resize(image, x, y, w, h):
    '''Recorta a cara da imagem, definida pelas coords (x,y,w,h)
sendo (x,y) o canto esquero superior. Alem disso, mantem o mesmo
aspect ratio das imagens negativas usadas para treinamento. Depois,
altera o tamanho da imagem para o mesmo das imagens negativas
'''
    height = int((variables.face_height / float(variables.face_width)) * w)
    meioy = int(y + h/2)
    y1 = int(max(0, meioy - height/2))
    y2 = int(min(image.shape[0]-1, meioy + height/2))
    image = image[y1:y2, x:x+w]
    image = cv2.resize(image,(variables.face_width,variables.face_height),
                       interpolation=cv2.INTER_LANCZOS4)
    return image


if __name__ == '__main__':

    model = cv2.createLBPHFaceRecognizer()
    model.load(variables.trained_model)

    print("Modelo carregado")
    print("Construindo a curva ROC")

    #labels de cada imagem
    labels = []
    #classe de saida
    output = []
    #saida do lpb (baseado em distancia)
    confidence = []


    nofacecounter = 0
    for root, dirs, files in os.walk("/home/pi/Desktop/TCC/roc/roctrain"):
        for subdirs in dirs:
            allpaths = root + '/' + subdirs
            for filename in os.listdir(allpaths):
                #each image in path (eiip)
```

```
            eiip = allpaths + '/' + filename
            if "myface" in eiip:
                labels.append(1)
            else:
                labels.append(0)
            image = cv2.imread(eiip)
            image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
            coords = recon_face(image)
            if coords !=None:
                #reconheceu a face
                x,y,w,h = coords
                image = cut_resize(image,x,y,w,h)
                classe, threshold = model.predict(image)
                output.append(classe)
                confidence.append(threshold)

            else:
                #nao reconheceu a face
                output.append(0)
                confidence.append(0)
                print(eiip + " nao reconhecida")
                nofacecounter += 1

print("{} faces nao foram detectadas".format(nofacecounter))

#valores de 0 a 1 para saida do algoritmo, normalizados pela distancia
predictions = []
#valor maximo de distancia para normalizar
maxvalue = max(confidence)


for classeout, distancia in zip(output, confidence):
    if classeout == 1:
        result = (2*maxvalue - distancia)/(2.0*maxvalue)
        predictions.append(result)
    elif classeout == 0:
        result = distancia/(2.0*maxvalue)
        predictions.append(result)
```

```
false_positive_rate, true_positive_rate, thresholds = roc_curve(labels, predict
roc_auc = auc(false_positive_rate, true_positive_rate)

plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate, true_positive_rate, 'b',
label='AUC = %0.2f'% roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
plt.savefig("ROC.jpg")
```