

Anderson Ferreira Rodriguez

MÓDULO DE AUTENTICAÇÃO POR RECONHECIMENTO FACIAL

Porto Alegre

2018

Anderson Ferreira Rodriguez

## MÓDULO DE AUTENTICAÇÃO POR RECONHECIMENTO FACIAL

Trabalho de Conclusão apresentado para obtenção do título de Tecnólogo em Sistemas Embarcados na Faculdade de Tecnologia SENAI Porto Alegre do Curso Superior de Tecnologia em Sistemas Embarcados.

Orientador: Professor Alexandre Gaspary Haupt

Porto Alegre

2018

## CIP – CATALOGAÇÃO NA PUBLICAÇÃO

RODRIGUEZ, Anderson Ferreira

Módulo de autenticação por reconhecimento facial / Anderson Ferreira Rodriguez; orientação [por] Alexandre Gaspary Haupt. – Porto Alegre: Faculdade de Tecnologia SENAI Porto Alegre, 2018.

66 f.: il.

1. Sistemas Embarcados. 2. Visão computacional. 3. Reconhecimento facial. 4. Processamento de imagem. I. Haupt, Alexandre Gaspary. II. Título.

FACULDADE DE TECNOLOGIA SENAI PORTO ALEGRE

Diretor: Márcio Rogério Basotti

Coordenador: Alexandre Gaspary Haupt

Bibliotecária: Gilmara Freitas Gomes

Anderson Ferreira Rodriguez

## MÓDULO DE AUTENTICAÇÃO POR RECONHECIMENTO FACIAL

Trabalho de Conclusão apresentado para obtenção do título de Tecnólogo em Sistemas Embarcados na Faculdade de Tecnologia SENAI Porto Alegre do Curso Superior de Tecnologia em Sistemas Embarcados.

Porto Alegre, 30/11/2018

---

Prof. Me. Alexandre Gaspary Haupt  
Orientador

---

Prof. Me. ....  
Avaliador

---

Prof. Me. Alexandre Gaspary Haupt  
Coordenador

## Agradecimentos

Agradeço aos colegas, por toda a assistência e troca de conhecimentos adquiridos. Agradeço aos familiares, por todos os momentos em que me apoiaram e me incentivaram a continuar buscando meus objetivos.

Agradeço à minha esposa, filha e filhos de quatro patas, pelo amor incondicional e por recarregarem minhas energias nos momentos mais difíceis desta trajetória. Esta conquista é tão minha quanto de vocês.

Ao Prof. Me. Alexandre Gaspary Haupt, um agradecimento especial pelo empenho dedicado à elaboração deste trabalho e por instigar a pesquisa na fascinante área do processamento de imagens.

Agradeço também a todos que me apoiaram e incentivaram das mais diversas maneiras durante toda a trajetória da minha graduação, em especial cito aqui àqueles que contribuíram ativamente nesta jornada: Cesar Antoniazzi, Cristiano Martins, Deivid Moraes, Filipe Estima, Jonatan Brochier, Leandro Borba, Marcelo Ávila, Laércio Muenchen, Michel Blanco e Me. Taciano Ares. A todos vocês e a quem eu possa ter injustamente esquecido, muito obrigado.

Agradeço finalmente a Deus. Sem o qual nada é possível.

*Jamais considere seus estudos como uma obrigação, mas como uma oportunidade invejável para aprender a conhecer a influência libertadora da beleza do reino do espírito, para seu próprio prazer pessoal e para proveito da comunidade à qual seu futuro trabalho pertencer.*

*Albert Einstein*

## RESUMO

Técnicas de processamento de imagem têm sido empregadas para reconhecimento facial a partir da extração de suas características. O monitoramento do movimento dos olhos é uma alternativa a sensores para testar a presença de vida. A autenticação a dispositivos e serviços é, em muitos casos, inserindo uma senha. O problema desta credencial é que dispositivos embarcados, como *smart-TV's* não possuem teclado físico, prejudicando a experiência do usuário. Considerando que câmeras com boa qualidade são facilmente encontradas, o objetivo geral desta proposta é desenvolver uma alternativa a autenticação por senha, implementando um método de autenticação por reconhecimento facial, utilizando ferramentas de código aberto. São objetivos específicos deste trabalho: Implementar um algoritmo para realizar detecção de faces utilizando a biblioteca OpenCV, aplicar o algoritmo criado para extrair informações biométricas da face, desenvolver uma aplicação que monitore os olhos para detectar a vitalidade. Este trabalho utiliza a metodologia experimental para desenvolver um módulo de autenticação por reconhecimento facial para plataformas embarcadas, baseando-se no kit de desenvolvimento Raspberry Pi 3 – model B. O módulo é testado em um grupo de 5 usuários. Os critérios avaliados são a precisão e desempenho do módulo, tendo sido realizados 10 experimentos para cada usuário simulando a autenticação no método proposto e no tradicional. A autenticação ocorre em duas etapas: o teste de detecção de vida monitorando o movimento dos olhos e o reconhecimento facial. Nos testes realizados com os 5 usuários obteve-se um tempo médio para acesso de 16,08 s, contrastando com os 23,99 s do método tradicional com uso de teclado virtual. Nos testes foi obtido um índice de reconhecimento facial de 89,6%. Recomenda-se para trabalhos futuros o aprimoramento do algoritmo de detecção de vida e melhorar a tolerância a variações de luminosidade.

Palavras-chave: Reconhecimento facial. Processamento de imagens. Sistemas Embarcados. Presença de vida. Visão Computacional.

## ABSTRACT

Image processing techniques has been employed for face recognition from the extraction of its characteristics. Eye tracking is an alternative to sensors in the liveness detection. Authentications to devices and services is, in many cases, entering a password. The problem with this credential is that embedded devices, like smart-TV's doesn't have a physical keyboard, reducing the user experience. Considering that good quality cameras are easily found, the main goal of this work is to develop an alternative to password authentication, using open-source tools. Specific goals of this work are: implement an algorithm that make face detection with the OpenCV library, apply the developed algorithm to extract biometric information from the face, develop an eye tracking application to make liveness detection. This work uses the experimental method to develop a face recognition authentication module suited to embedded systems, based on the Raspberry Pi 3 – model B development kit. The module is tested in a 5 users group. The criteria evaluated are the accuracy and performance of the module, 10 experiments were performed for each user simulating authentication in the proposed method and in the traditional one. Authentication occurs in two steps: the liveness detection test with the eye tracking and the face recognition. In the tests made up with 5 users, an average access time of 16.08 s was obtained, against the 23,99s of the traditional method using a virtual keyboard. In the tests, a facial recognition accuracy index of 89.6% was obtained. Future work is recommended to improve the liveness detection algorithm and improve tolerance to brightness variations.

Keywords: Facial recognition. Image processing. Embedded systems. Liveness detection. Computer vision.



## LISTA DE FIGURAS

<b>Figura 1 – Tipos de vizinhanças do pixel.....</b>	<b>15</b>
Figura 2 – Etapas da visão artificial.....	15
Figura 3 – Operações aritméticas em uma imagem.....	17
Figura 4 – Histograma RGB e grayscale.....	18
Figura 5 – Filtros aplicados em imagem com ruído Salt & Pepper.....	21
Figura 6 – Variação de intensidade luminosa e suas derivadas.....	22
Figura 7 – Máscaras para implementar o operador gradiente.....	23
Figura 8 – Operador Laplaciano.....	23
Figura 9 – Exemplos de dispositivos embarcados.....	24
Figura 10 – Diagrama de pinos da Raspberry Pi 3.....	25
Figura 11 – Remessas unitárias em todo o mundo de sistemas operacionais.....	26
Figura 12 – Etapas da extração de descritores SURF.....	28
Figura 13 – Exemplo de cálculo utilizando a imagem integral.....	29
Figura 14 – Pirâmides de escalas.....	30
Figura 15 – Representação das máscaras em relação às escalas.....	31
Figura 16 – Máscaras de convolução da matriz Hessiana.....	32
Figura 17 – Extração do conjunto de descritores SURF.....	33
Figura 18 – Representação dos atributos do descritor SURF.....	34
Figura 19 – Fluxo Óptico. a) quadro 1. b) quadro 2. c) Fluxo Óptico.....	35
Figura 20 – Trecho código Lucas e Kanade.....	36
Figura 21 – Código Horn-Shunck.....	37
Figura 22 – Gráfico comparativo dos métodos de fluxo óptico.....	39
Figura 23 – Montando instalação com Etcher.....	46
Figura 24 – Arquivo dphys-swapfile.....	47
Figura 25 – Desativando a interface gráfica.....	48
Figura 26 – Processos Internos do módulo.....	48
Figura 27 – Detecção e rastreamento da face.....	52
Figura 28 – Fluxo óptico utilizando Farneback.....	54
Figura 29 – Extração dos pontos de interesse.....	56
Figura 30 – Extração dos pontos de interesse.....	57
Figura 31 – Classe alvo gerando movimentos aleatórios.....	58
Figura 32 – Resultado do teste com algoritmo SURF.....	58
Figura 33 – Região de interesse do olho após filtro.....	59
Figura 34 – Gráfico resultado do teste.....	61



## LISTA DE TABELAS

<b>Tabela 1 – Comparativo de tempos de execução dos métodos.....</b>	<b>39</b>
Tabela 2 – Comparação entre características e requisitos.....	42
Tabela 3 – Lista de parâmetros acessíveis pela biblioteca OpenCV.....	50
Tabela 4 – Tempo de processamento.....	52
Tabela 5 – Resultado dos testes de reconhecimento facial.....	60
Tabela 6 – Resultado dos testes de autenticação com senha.....	62
Tabela 7 – Resultado dos testes de autenticação com reconhecimento facial.....	63

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO.....</b>	<b>12</b>
<b>2</b>	<b>REFERENCIAL TEÓRICO.....</b>	<b>14</b>
2.1	Processamento de imagens.....	14
2.1.1	Etapas do processamento de imagens.....	15
2.2	Técnicas e algoritmos de processamento de imagens.....	16
2.2.1	Operações por convolução de máscaras.....	19
2.2.2	Filtros para remoção de ruído.....	19
2.2.3	Detecção de bordas.....	21
2.3	Sistemas embarcados.....	23
2.3.1	Kit de desenvolvimento.....	24
2.3.2	Sistemas operacionais.....	25
2.3.3	Raspbian.....	26
2.4	Visão computacional.....	27
2.4.1	Detecção de faces e objetos.....	27
2.4.2	Reconhecimento facial a partir de descritores.....	28
2.4.3	Detecção dos pontos de interesse.....	28
2.4.4	Descrição dos pontos de interesse.....	32
2.5	Técnicas de fluxo óptico.....	34
2.5.1	Método de Lucas & Kanade.....	35
2.5.2	Método de Horn e Shunch.....	37
2.5.3	Comparação entre os métodos.....	38
2.6	Segurança e autenticação.....	40
2.6.1	Autenticação biométrica.....	40
<b>3</b>	<b>PROCEDIMENTO METODOLÓGICO.....</b>	<b>43</b>
3.1	Preparação do ambiente de desenvolvimento e de teste.....	43
3.1.1	Preparando o ambiente de desenvolvimento.....	43
3.1.2	Preparando o ambiente de testes.....	46
3.2	Estrutura do módulo de autenticação.....	48
3.3	Aquisição da imagem.....	50
3.4	Detecção da face e olhos.....	51
3.5	Tese de detecção de vida.....	53
3.6	Extração de características biométricas.....	55
3.7	Reconhecimento facial.....	56
3.8	Desenvolvimento do módulo.....	57

4	APLICAÇÃO E RESULTADOS.....	60
5	COMENTÁRIOS FINAIS.....	64
	REFERÊNCIAS.....	65

## 1 INTRODUÇÃO

Muitas pesquisas têm sido realizadas no mundo inteiro sobre o reconhecimento facial, seja para controle de acesso, seja para identificação de usuários. Técnicas de processamento de imagem são empregadas para detecção facial e extração de características da face, objetivando o reconhecimento dos padrões. Além disso, o monitoramento do movimento dos olhos pode ser útil para validar a presença de vida durante o reconhecimento. Digitar uma senha para obter acesso a dispositivos ou serviços na *web* ainda é o método mais utilizado de autenticação, o qual pode ser trabalhoso e prejudicar a experiência do usuário em dispositivos portáteis ou com restrições, como, por exemplo, uma *smart TV*.

Autenticação tradicional de sistemas tem sido predominantemente através de digitação de credenciais, no entanto, o problema é que esta forma não tem sido prática uma vez que alguns dispositivos não possuem teclado físico.

Tendo em vista que a grande maioria dos dispositivos móveis possuem câmeras com boa qualidade, melhorar a confiança no método de autenticação por reconhecimento facial pode ser a solução para muitos casos de uso. Uma forma de melhorar a segurança em uma autenticação com reconhecimento facial é realizando um teste de detecção de vida junto com o teste biométrico.

O objetivo geral desta proposta é desenvolver uma alternativa a autenticação por senha, implementando um método de autenticação por reconhecimento facial, utilizando ferramentas de código aberto.

São objetivos específicos deste trabalho:

- Implementar um algoritmo para realizar detecção de faces utilizando a biblioteca OpenCV;
- Aplicar o algoritmo criado para extrair informações biométricas da face;
- Desenvolver uma aplicação que monitore os olhos e faça a verificação da presença de vida;
- Testar a precisão e desempenho do algoritmo;
- Embarcar em uma plataforma de desenvolvimento;
- Portar o algoritmo em uma biblioteca no padrão dos serviços do Linux-PAM.

Este trabalho utiliza a metodologia experimental para desenvolver um módulo de autenticação por reconhecimento facial para plataformas embarcadas, baseando-se no kit de desenvolvimento *Raspberry PI 3 – model B*.

O reconhecimento facial será testado em um grupo de usuários com 5 indivíduos. Os usuários foram selecionados tendo como critério sua familiaridade com a utilização de tecnologias. Os critérios avaliados são a precisão do módulo, tendo sido realizadas 10 tentativas para cada usuário, e o tempo que o usuário demora a realizar a autenticação em comparação ao método tradicional de digitar senha.

Por fim, testar sua precisão e desempenho em sistemas embarcados com sistema operacional Linux. A autenticação ocorrerá em duas etapas: o reconhecimento facial e o monitoramento dos olhos em tempo real para realizar a detecção de vida (*liveness detection*).

## 2 REFERENCIAL TEÓRICO

Neste capítulo serão abordados os temas que foram pesquisados para a execução deste trabalho e, para cada tema, uma explicação introdutória com o objetivo de facilitar o entendimento do trabalho.

### 2.1 Processamento de imagens

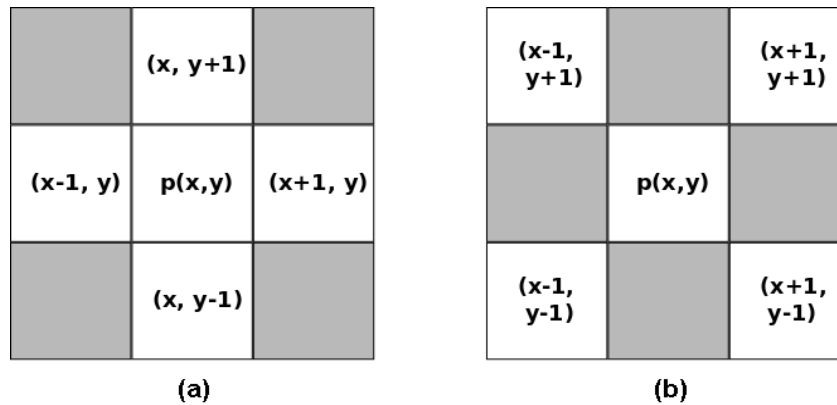
O processamento digital de imagens pode ser definido como o processo de extrair informações úteis a partir de uma imagem fornecida, interpretando-a e modificando-a se necessário. A imagem digital é uma coleção de pixels, organizados em uma matriz ou em algum formato que os algoritmos de processamento consigam interpretar para realizar posteriormente o processamento da informação. Podemos dizer que a matriz representa um plano em que a origem (0,0) da imagem digital é o pixel do canto superior esquerdo (SHAH, 2015).

O Pixel (Picture Element) é o elemento mais básico de uma imagem e guarda duas informações: uma coordenada espacial e um valor de intensidade luminosa para cada canal de cor. Imagens coloridas possuem geralmente três canais de cores e imagens em tons de cinza apenas um e, por esse motivo, o processamento digital de imagens é realizado predominantemente em imagens em tons de cinza, a fim de reduzir a carga computacional. A conversão de uma imagem colorida para uma em tons de cinza se dá através da média aritmética das intensidades luminosas para cada pixel.

Outra característica com grande importância no processamento de imagens é a vizinhança de um pixel. Cada pixel quando analisado individualmente possui quatro vizinhos horizontais e verticais e quatro vizinhos diagonais conforme ilustra a Figura 1.



Figura 1 – Tipos de vizinhanças do pixel



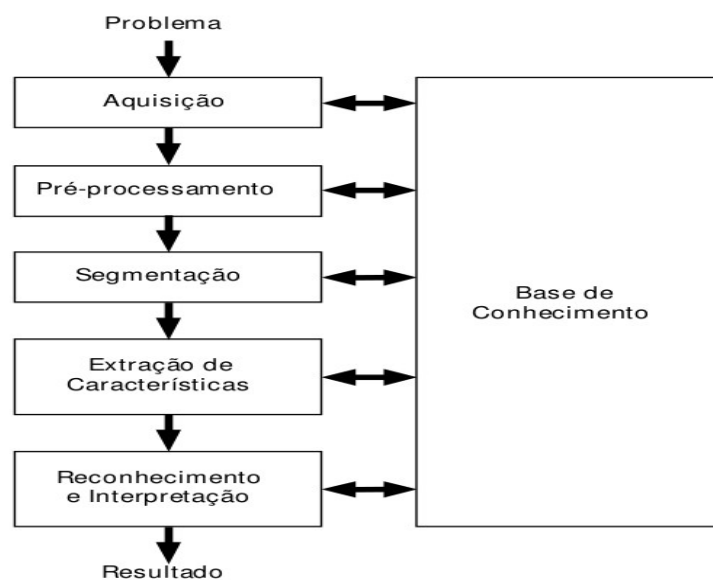
Fonte: Autor, 2018

O conjunto formado por vizinhos horizontais e verticais é denominado “4-vizinhança” de um pixel e a união deste conjunto com os vizinhos diagonais é denominado a “8-vizinhança” de um pixel. Esta característica é bastante utilizada em algoritmos de convolução.

### 2.1.1 Etapas do processamento de imagens

Um sistema de visão artificial completo para a solução de um problema pode ser dividido em 5 etapas principais, conforme ilustra a Figura 2.

Figura 2 – Etapas da visão artificial



Fonte: MARQUES FILHO, 1999

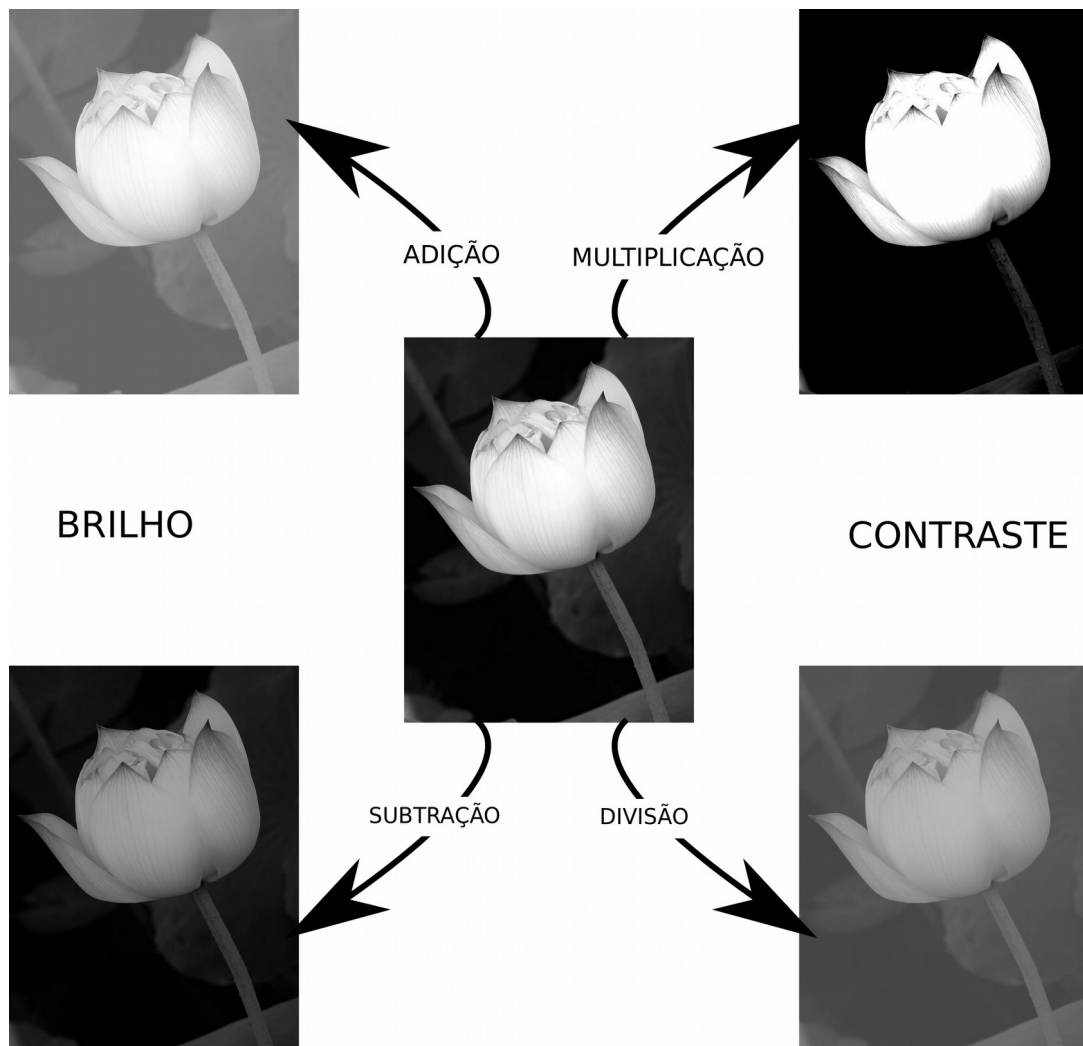
- Aquisição da imagem – Etapa inicial na qual é realizada a aquisição da imagem. Sinais ópticos são convertidos e digitalizados formando ao final desta etapa uma imagem digital. Nesta etapa deve-se selecionar adequadamente o tipo de sensor utilizado, ajustes de iluminação da cena e a resolução da imagem entre outros.
- Pré-processamento – Após a aquisição, pode ser necessário realizar ajustes na imagem para remover imperfeições causadas por diversos fatores. Entre estes fatores destacam-se ruídos resultantes da transmissão da imagem digital e ajustes de brilho e contraste que não puderam ser corrigidos na etapa anterior. A saída desta etapa é uma imagem com a qualidade aprimorada para as etapas subsequentes.
- Segmentação – Etapa onde a imagem é dividida em objetos ou regiões de interesse. Esta é uma das etapas com maior complexidade de implementação.
- Extração de características – Etapa onde são extraídos dados da imagem. A entrada desta etapa ainda é uma imagem digital, porém sua saída será um conjunto de dados correspondente à imagem.
- Reconhecimento e interpretação – Nesta etapa são atribuídos rótulos aos objetos identificados (reconhecimento). Após o reconhecimento, nesta etapa é atribuído um significado (interpretação), como uma tomada de decisão por exemplo (MARQUES FILHO, 1999).

## 2.2 Técnicas e algoritmos de processamento de imagens

Uma imagem após a etapa de aquisição pode ser interpretada como uma matriz de inteiros e por este motivo, a imagem pode ser manipulada numericamente por operações lógicas e aritméticas. Nesta sessão serão apresentados alguns algoritmos de processamento baseados nestas propriedades.

Conforme ilustrado na Figura 3, aplicando-se operações aritméticas em uma imagem digital é possível ajustar o brilho e o contraste. Ao somar ou subtrair um valor a toda a imagem, o brilho é ajustado e ao multiplicar ou dividir a imagem por um valor, o contraste é alterado (MARQUES FILHO, 1999).

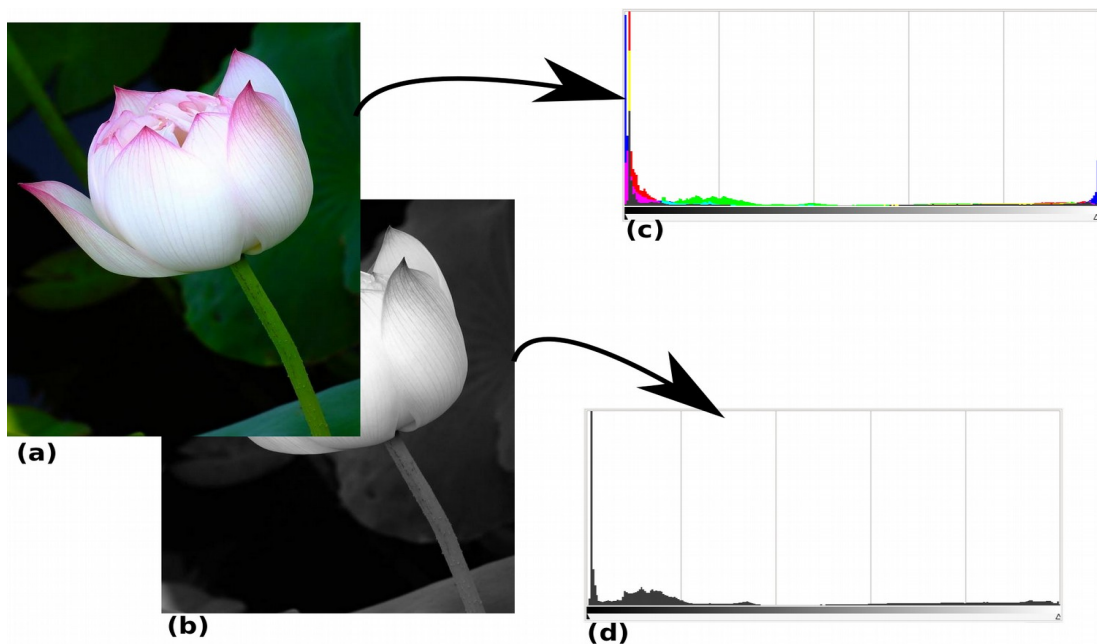
Figura 3 – Operações aritméticas em uma imagem



Fonte: Autor, 2018

O histograma é a representação da quantidade de pixels por nível de cinza. É geralmente representado como um gráfico de barras. A partir deste gráfico é possível obter informações como se a imagem apresenta um bom contraste, se a imagem é muito clara ou muito escura, entre outras observações possíveis. Também a partir do histograma é possível determinar limiares para aplicar outros processos como por exemplo a binarização da imagem. Para se obter o histograma de uma imagem colorida, a mesma deve ser decomposta seguindo algum padrão de cores, como por exemplo, o padrão RGB. A Figura 4 ilustra as imagens e histogramas de imagem RGB e *grayscale*: imagem colorida (a), imagem em tons de cinza (b), o histograma da imagem colorida (c) e o histograma da imagem em tons de cinza (d).

Figura 4 – Histograma RGB e grayscale



Fonte: AUTOR, 2018

A binarização (ou limiarização) é uma operação que tem como princípio segmentar regiões de uma imagem. A forma mais simples de binarização consiste em realizar uma bipartição do histograma, ou seja, determina-se um limiar de binarização e então todos os valores iguais ou acima do limiar são convertidos em pixels brancos e os demais em pixels pretos. A operação de binarização pode ser descrita matematicamente como uma técnica de processamento de imagens onde dado uma imagem  $f(x,y)$  de  $n$  níveis de cinza, é obtida como resultado uma imagem  $g(x,y)$ , conforme ilustra a equação 1.

$$g(x,y) = \begin{cases} 1 & \text{se } f(x,y) \geq T \\ 0 & \text{se } f(x,y) < T \end{cases} \quad (1)$$

Nela, pixels rotulados como 1 correspondem aos objetos e os pixels rotulados como 0 correspondem ao *background* (plano de fundo). A constante 'T' é o limiar predefinido empiricamente ou através de análises do histograma.

Outras operações muito frequentes em processamento de imagem são as técnicas de filtragem, realce e remoção de ruídos. O principal objetivo destas técnicas é preprocessar uma imagem de entrada para produzir na saída uma imagem que esteja mais adequada que a imagem original para um determinado fim. A afirmação de que a imagem de saída é mais adequada é subjetiva e depende do conhecimento prévio do observador. Uma técnica de filtragem idealizada para imagens obtidas através de um instrumento médico, como um tomógrafo, pode não

ter um resultado satisfatório para uma outra imagem obtida através de uma câmera convencional. Geralmente, são técnicas aplicadas na segunda etapa de um sistema de visão artificial.

### 2.2.1 Operações por convolução de máscaras

Operações de convolução com máscaras são utilizadas em diversas operações no processamento de imagens. O método consiste em passar sobre a imagem uma matriz menor (normalmente 3x3, 5x5 ou 7x7) composta por coeficientes. Inicia-se uma varredura pelo canto superior esquerdo e então movimenta-se a máscara da esquerda para a direita e de cima para baixo até a última posição possível no canto inferior direito. Para cada posição possível da máscara sobre a imagem, um novo valor é calculado e adicionado a uma nova matriz resultado. Este valor é definido pela seguinte equação:

$$Z = \sum_{i=1}^n W_i \cdot Z_i \quad (2)$$

Onde 'Z' é o valor do pixel que será armazenado na nova matriz resultado, 'n' é o total de coeficientes presentes na máscara, 'W<sub>i</sub>' é o coeficiente da máscara na posição 'i' e 'Z<sub>i</sub>' é o pixel correspondente na imagem original na posição 'i' (MARQUES FILHO, 1999).

### 2.2.2 Filtros para remoção de ruído

Um dos problemas mais recorrentes durante a aquisição da imagem, e que pode interferir no resultado do processamento, é o ruído. O ruído pode ser causado por perdas de dados durante a transmissão, conhecido como ruído *salt & pepper* (ou ruído sal e pimenta), ou até mesmo por características do sensor utilizado na aquisição, como por exemplo o ruído gerado ao aumentar a sensibilidade (ISO) de uma câmera. A remoção deste ruído pode ser feita através de filtros de média, gaussiana e mediana, entre outros.

O filtro de média utiliza a técnica de convolução com uma máscara na qual todos os coeficientes são 1, obtendo-se assim, a média dos pixels sobre a máscara. Segundo Gonzales (2010, pág. 211) “Um filtro de média atenua variações locais em uma imagem, e o ruído é reduzido em consequência do borramento”. Quanto maior

for a máscara, geralmente 3x3, 5x5 ou 7x7, mais borrada ficará a imagem. Outra forma de remover o ruído é utilizando o filtro gaussiano, consiste em realizar uma média ponderada dos pixels que estão sobre a máscara. Da mesma forma que o filtro de média, utiliza-se a convolução de máscara porém agora os coeficientes serão diferentes, dando maior importância ao valor do pixel no centro da máscara.

Os dois filtros descritos conseguem aprimorar a imagem porém possuem um efeito colateral, os filtros de média e gaussiano acabam danificando as bordas dos objetos na imagem. As bordas dos objetos são informações de grande interesse em vários casos e portanto os filtros citados podem não ser efetivos em certos níveis de ruído. Para a remoção do ruído sal e pimenta, a técnica mais indicada é o filtro por mediana. Esta técnica consiste em obter o valor mediano dos valores abaixo da máscara a cada passo da convolução. Como o valor obtido é um pixel presente na imagem e não um valor calculado que pode não pertencer a imagem original, as bordas sofrem menores danos. No código abaixo, segue um exemplo de como aplicar os filtros citados em uma imagem utilizando a biblioteca OpenCV em Python. Na Figura 5, é exibido o resultado do código. No canto superior direito a imagem original corrompida com ruído sal e pimenta; acima à esquerda a imagem original tratada com filtro de média; no canto inferior direito a imagem original tratada com filtro gaussiano; e, no canto inferior direito, a imagem original tratada com filtro mediana.

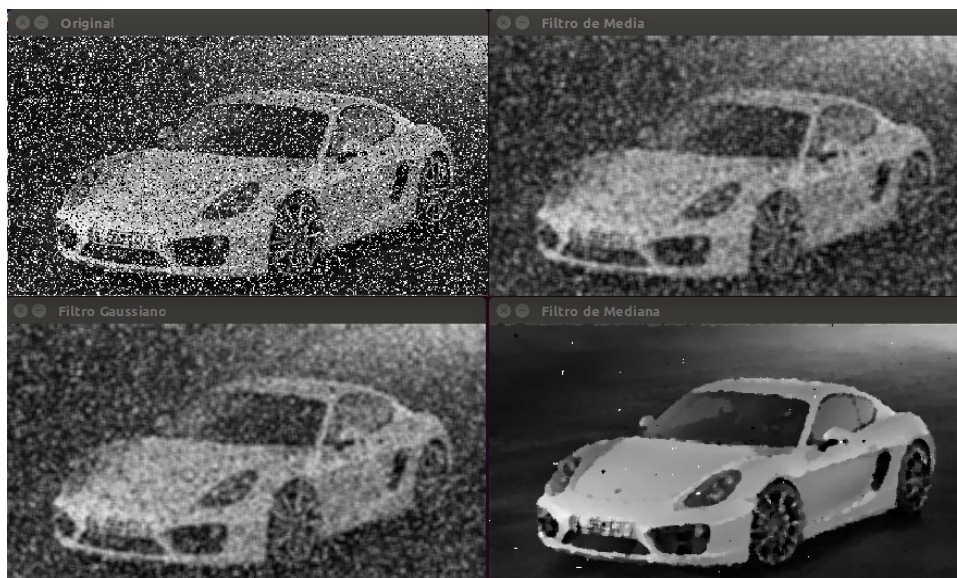
```
#Biblioteca OpenCV
import cv2
# Biblioteca para realizar cálculos com matrizes
import numpy as np
# Carrega a imagem em tons de cinza
img = cv2.imread('exemplo.bmp', 0)
# Exibe imagem original
cv2.imshow("Original", img)
# Aplica filtro de média com máscara 5x5
blur = cv2.blur(img, (5, 5))
# Exibe imagem com filtro de média
cv2.imshow("Filtro de Media", blur)
# Aplica filtro gaussiano
blur2 = cv2.GaussianBlur(img, (5,5),3)
# Exibe imagem com filtro gaussiano
cv2.imshow("Filtro Gaussiano", blur2)
# Aplica o filtro mediana
```

```

median = cv2.medianBlur(img,5)
# Exibe imagem com filtro mediana
cv2.imshow("Filtro Mediana",median)
# Aguarda usuário pressionar tecla
cv2.waitKey(0)
# Fecha todas as imagens exibidas
cv2.destroyAllWindows()

```

Figura 5 – Filtros aplicados em imagem com ruído Salt & Pepper



Fonte: AUTOR, 2018

### 2.2.3 Detecção de bordas

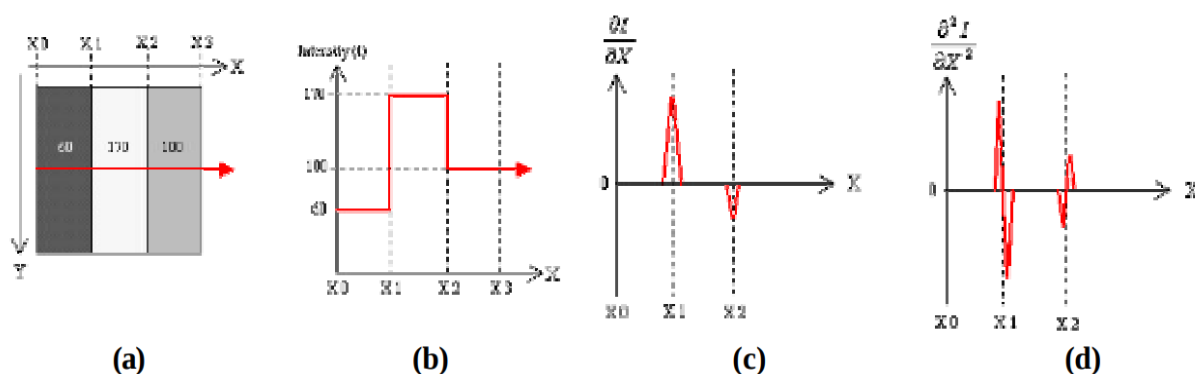
Bordas exercem um papel principal tanto na visão computacional quanto humana. Seres humanos conseguem facilmente reconhecer muitos objetos e suas posições apenas visualizando sua silhueta retroiluminada ou um esboço grosseiro (MINICHINO, 2015).

A detecção de bordas é uma técnica de processamento de imagens que visa realçar ou destacar regiões da imagem em que há uma descontinuidade nos tons de cinza. Estas regiões são, geralmente, os contornos de objetos ou defeitos na imagem como manchas ou ruídos. Por este motivo muitas vezes é necessário tratar a imagem antes de realizar a detecção de bordas, a fim de evitar que ruídos e imperfeições acabem sendo identificados como bordas.

Segundo HAUPT (2004), “a ideia por trás da maioria das técnicas para detecção de bordas é a computação do operador local derivativo em uma imagem que apresenta variações de intensidade luminosa”. Entre as técnicas mais utilizadas

podemos destacar os métodos de Roberts, Prewitt, Sobel e Laplace. Os métodos Roberts, Prewitt e Sobel utilizam a derivada de 1º ordem da imagem e o operador Laplaciano faz uso da derivada de 2ª ordem de uma imagem. Na Figura 6, a imagem (a) apresenta um exemplo de imagem com diferentes tons de cinza; o gráfico (b), a variação de intensidade; o gráfico (c), a derivada de 1º ordem da variação de intensidade luminosa; e o gráfico (d), a derivada de 2ª ordem da variação de intensidade luminosa.

Figura 6 – Variação de intensidade luminosa e suas derivadas



Fonte: HAUPT, 2004

Os operadores de 1º ordem, Roberts, Prewitt e Sobel, são sensíveis à direção e pelo fato de a imagem ser uma matriz de duas dimensões (x e y), a imagem é derivada em relação à x e à y. Para implementar a derivada da imagem através do método de convolução, é necessário o uso de duas máscaras: uma para a derivada em relação à x e outra para a derivada em relação à y, e então somam-se os resultados para compor a imagem derivada. Na Figura 7, estão listadas as máscaras utilizadas em cada método. A soma dos coeficientes em todos os operadores é zero para que, em regiões da imagem que não apresentam variação de cor, o valor da derivada seja zero.



Figura 7 – Máscaras para implementar o operador gradiente

	Gx		Gy		
Roberts	0	1	1	0	
	-1	0	0	-1	
Prewitt	-1	0	1	-1	-1
	-1	0	1	0	0
	-1	0	1	1	1
Sobel	-1	0	1	-1	-1
	-2	0	2	0	0
	-1	0	1	1	2

Fonte: GONZALES, 2010

O operador Laplaciano, Figura 8, utiliza a derivada de 2ª ordem da imagem e por este motivo tem alta sensibilidade ao ruído. As bordas são obtidas através da detecção de passagem por zero. “Os operadores Laplacianos geram um escalar, por isto utilizam apenas uma máscara” (HAUPT, 2004).

Figura 8 – Operador Laplaciano

$$\text{kernel} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Fonte: OPENCV, 2014.

### 2.3 Sistemas embarcados

Sistemas embarcados são computadores projetados para executar funções específicas. Alguns realizam apenas uma determinada função em repetição infinita (aplicações *one single loop*). Sistemas embarcados diferem de computadores convencionais em muitos pontos desde sua arquitetura até limitações de hardware. Na Figura 9 estão representados alguns exemplos de dispositivos embarcados.

Figura 9 – Exemplos de dispositivos embarcados



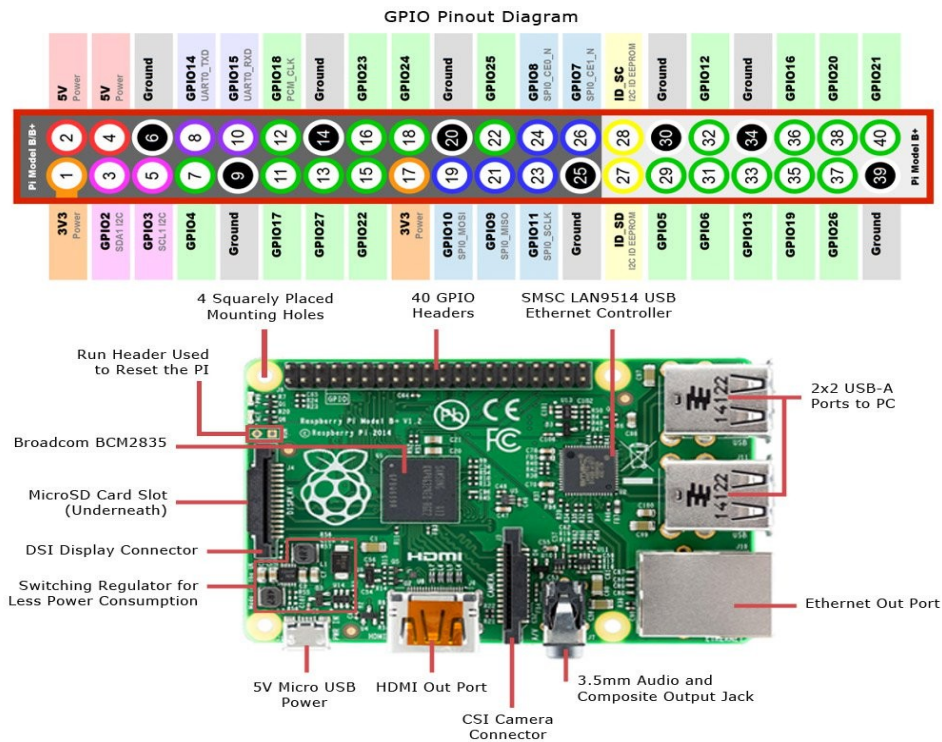
Fonte: Autor, 2018

### 2.3.1 Kit de desenvolvimento

O kit de desenvolvimento embarcado escolhido (*Raspberry pi 3*) possui basicamente todos os recursos que um computador convencional possui porém com sérias restrições de espaço de memória (RAM e ROM) e menor poder computacional. O kit de desenvolvimento foi escolhido devido sua versatilidade, suporte pela comunidade oficial e fácil aquisição.

A *Raspberry Pi* é um SBC (*Single-Board Computer*), ou seja, um computador onde todos os componentes estão em uma única placa. O modelo B (Figura 10) possui dimensões próximas a de um cartão de crédito, conta com 1GB de memória RAM, 40 GPIO e como processador, um SoC (System on a chip) da Broadcomm, o BCM2837. Ainda possui conectividade de rede Ethernet, Wi-Fi e Bluetooth e 4 portas USB. O SoC BCM2837 inclui um processador com 4 núcleos ARM com arquitetura Cortex-A53 de 64 bits que podem chegar a 1.2 GHz de *clock* e também uma GPU *Videocore IV* com *clock* de 400 MHz (SOUZA, 2016). Entre os 40 pinos GPIO encontram-se diversos padrões de comunicação como I2C, SPI e UART.

Figura 10 – Diagrama de pinos da Raspberry Pi 3

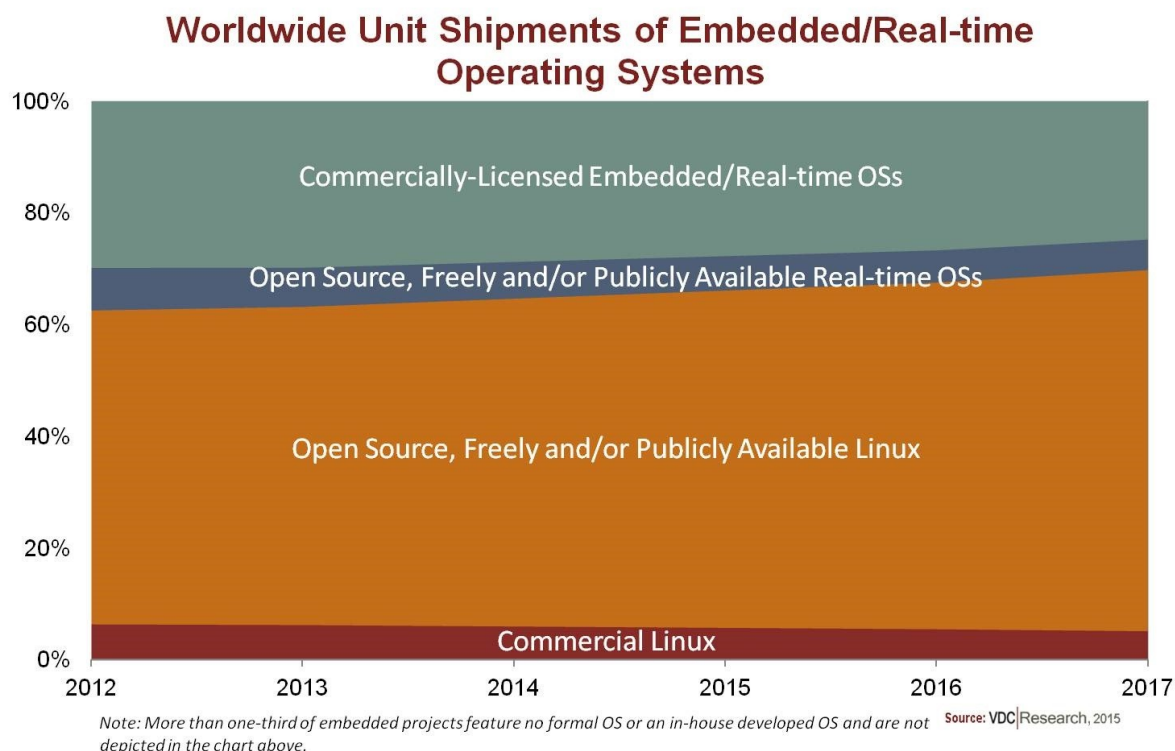


Fonte: JAMECO, 2016

### 2.3.2 Sistemas operacionais

O sistema operacional é um programa (ou conjunto de programas) responsável por realizar o gerenciamento dos recursos do sistema. Basicamente, o sistema operacional tem três atribuições: coordenar o uso do processador, gerenciar a memória e criar um sistema de arquivos. Em computadores convencionais existem predominantemente três sistemas operacionais (Windows, MAC e Linux), no entanto, para dispositivos embarcados a oferta de sistemas operacionais é muito maior, cada um visando arquiteturas e regras de projetos diferentes. Mesmo com a grande variedade de sistemas operacionais, estudos apontam (Figura 11) que, em 2012, 62,5% dos sistemas operacionais embarcados eram versões open-source (56,2%) e comerciais (6,3%) do Linux (BROWN, 2015). Uma importante tarefa que foi incorporada aos sistemas operacionais modernos é a autenticação. Uma grande parte dos sistemas operacionais derivados do Linux adota um padrão para gerenciar as tarefas de autenticação, o Linux-PAM (*Pluggable Authentication Modules for Linux*).

Figura 11 – Remessas unitárias em todo o mundo de sistemas operacionais



Fonte: BROWN, 2015

O Linux-PAM é um sistema de bibliotecas que tem a função de gerenciar as tarefas de autenticação de aplicações e serviços do sistema operacional. Estas bibliotecas proveem uma API (Application Programming Interface) para que programas que necessitam de autenticação possam fazer uso de tarefas de autenticação padrão do sistema. A principal característica abordada do PAM é que a natureza da autenticação pode ser configurada dinamicamente. Esta característica torna possível que o administrador do sistema configure quais métodos de autenticação serão utilizados para autenticar os usuários a um determinado serviço.

### 2.3.3 Raspbian

O *Raspbian* é o sistema operacional padrão para todos os modelos de *Raspberry Pi*. É uma distribuição baseada no sistema operacional Linux Debian e mantida pela *Raspberry Pi Foundation*. A versão utilizada neste trabalho é o *Raspbian Stretch*, lançado em Junho de 2018. Como trata-se de um sistema embarcado, não há um disco rígido para armazenamento de arquivos, o sistema operacional fica armazenado em um cartão de memória.

Como em todos os sistemas derivados do Unix, o Linux abstrai todos os componentes do sistema em arquivos e diretórios. Por exemplo, os dispositivos de hardware ficam expostos no diretório */dev* no sistema de arquivos. Ao utilizar uma câmera USB por exemplo, ela ficará disponível como um arquivo de texto em */dev/video0*.

## 2.4 Visão computacional

Dentre os cinco sentidos do ser humano, a visão é a maior porta de entrada de informações para o cérebro. A maior parte desta informação é redundante e é filtrada para que o cérebro processe apenas uma pequena fração desta quantidade de informação absorvida. O maior objetivo de áreas como a visão computacional é o de emular a visão humana, incluindo aprender e tomar decisões baseadas em entradas visuais (SHAH, 2015).

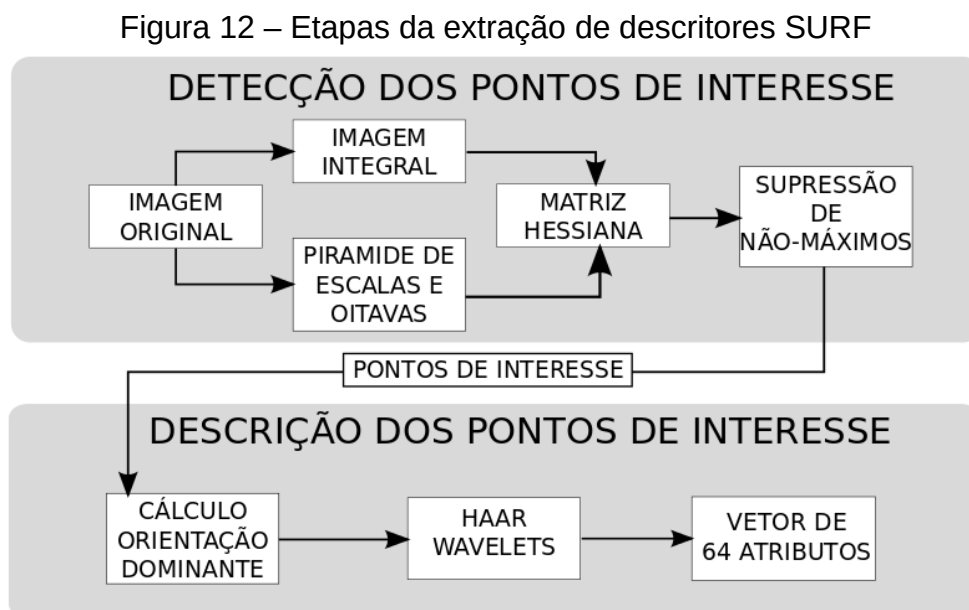
### 2.4.1 Detecção de faces e objetos

Em 2001, Viola e Jones propuseram um método efetivo de detecção de objetos baseado em técnicas de aprendizado de máquina. A detecção de objetos consiste em utilizar muitas imagens positivas (com o objeto) e imagens negativas (sem o objeto) pra treinar um classificador. Como são muitas imagens, o processo de extrair as características e selecionar as mais relevantes resulta em uma carga computacional muito grande. Como geralmente a maior parte da imagem não contem o objeto, uma solução melhor seria utilizar um método mais simples para descartar as janelas da imagem que não contem o objeto e focar o processamento somente nas janelas da imagem que contém o objeto. A partir deste pensamento Viola e Jones introduziram o conceito de cascata de classificadores (ou classificadores em cascata), onde a imagem é testada em um conjunto de vários estágios de classificadores. Nos primeiros estágios os classificadores testam poucas características e vão descartando as janelas da imagem sem o objeto, estágio por estágio, até que nas etapas finais, nas quais os classificadores testam mais características, somente a parte relevante é testada (SHAH, 2015).

### 2.4.2 Reconhecimento facial a partir de descritores

Um dos ramos de grande interesse da visão computacional é o reconhecimento facial, no qual o objetivo não é apenas identificar a presença da face mas também obter características únicas para garantir a identificação do indivíduo. As técnicas SIFT (*Scale Invariant Feature Transform*) e SURF (*Speeded-Up Robust Features*) varrem a imagem a procura por pontos de interesse (*keypoints*) e calculam descritores para cada um destes pontos de interesse. Os pontos de interesse são regiões com uma grande variação nos tons de cinza, pontos com maior gradiente (ou derivada). O objetivo destas técnicas é de ter uma aplicação genérica, ou seja, não foram elaborados com foco em reconhecimento facial, mas ambas apresentam boa precisão com um destaque para a técnica SURF que consegue atingir bons resultados com carga computacional menor em comparação com a técnica SIFT. A técnica SURF consegue esta melhora na carga computacional porque realiza aproximações em algumas etapas do processamento.

Conforme Figura 12, o algoritmo SURF (BAY et al. 2008) pode ser dividido em duas etapas principais, sendo cada uma dividida em subetapas.



Fonte: Autor, 2018



### 2.4.3 Detecção dos pontos de interesse

A primeira grande etapa é a detecção dos pontos de interesse na imagem de entrada. Os pontos de interesse são aqueles que caracterizam um objeto da forma mais eficiente, como os contornos por exemplo. Estes pontos geralmente são os que apresentam maior descontinuidade de tons de cinza em determinado eixo (x, y ou diagonal). Para detectar estes pontos de interesse então é preciso obter a derivada da imagem.

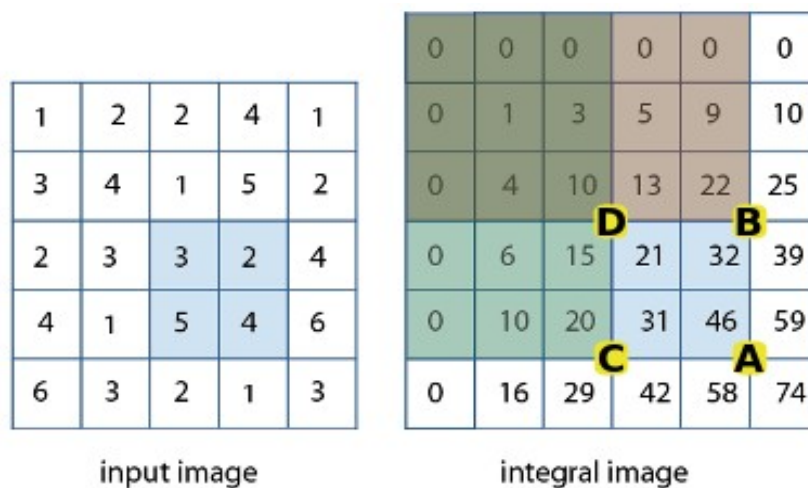
- Imagem integral – O conceito de Imagem integral, exemplificado na Figura 13, é uma representação da imagem original onde cada ponto é a somatória dos pixels dentro da área acima e a direita do ponto, até a origem (x=0, y=0). Desta forma, com apenas 4 operações de soma, como demonstrado na fórmula abaixo, é possível obter a soma de todos os pixels da área desejada.

$$\sum = A - B - C + D \quad (3)$$

Esta etapa foi adicionada no algoritmo SURF para deixá-lo mais eficiente, tornando mais ágil a soma dos pixels durante a convolução das máscaras nas etapas adjacentes. Na Figura 13 é possível verificar esta propriedade. Somando os pixels da área em azul da imagem de entrada (esquerda) obtemos o valor 14, e aplicando a fórmula acima obtemos:

$$\sum = 46 - 22 - 20 + 10 = 14 \quad (4)$$

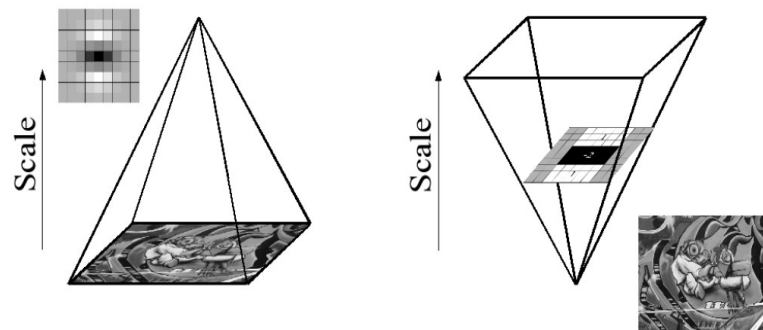
Figura 13 – Exemplo de cálculo utilizando a imagem integral



Fonte: Adaptado de MATHWORKS, 2018

- Pirâmide de escalas – Quando o algoritmo for utilizado para procurar um padrão na imagem de entrada, este padrão não será encontrado no mesmo tamanho que a amostra utilizada para montar o padrão. Por este motivo é necessário criar uma pirâmide de escalas. O algoritmo SIFT cria várias versões da imagem de entrada, cada uma com uma escala menor que a anterior como em uma pirâmide. O método SURF por outro lado, cria uma pirâmide a partir do mesmo conceito mas alterando a escala da máscara de convolução. Como no passo anterior foi gerado uma imagem integral, a área que a máscara ocupará terá sempre o mesmo custo computacional para ser processada independente da escala. Na Figura 14 é possível observar uma abstração do conceito no algoritmo SIFT (esquerda, diferentes escalas da imagem) e no algoritmo SURF (direita, diferentes escalas da máscara).

Figura 14 – Pirâmides de escalas

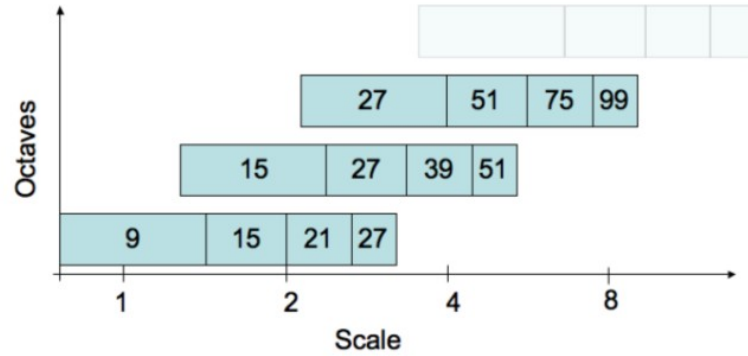


Fonte: BAY et al. 2008.

- Oitavas – outro conceito utilizado no algoritmo SURF é o de definir escalas intermediárias para as máscaras de convolução. Em vez de simplesmente dobrar o tamanho do filtro, o que ocasionaria em um intervalo de tamanho muito grande nas escalas finais, ele as separa em oitavas (conforme Figura 15), garantindo, assim, que o padrão será detectado também em escalas intermediárias.



Figura 15 – Representação das máscaras em relação às escalas



Fonte: BAY et al. 2008.

- Cálculo da matriz Hessiana – Esta etapa realiza a detecção dos pontos de interesse através de um método denominado como *Fast-Hessian Detector*. Este método consiste em uma aproximação de outros detectores com foco em reduzir o custo computacional admitindo uma queda não significativa na precisão do detector. Para isso, é necessário encontrar o determinante da matriz Hessiana. A matriz Hessiana é formada pelas derivadas parciais da imagem e representa a variação da intensidade luminosa nos sentidos x, y e na diagonal xy, segundo a fórmula abaixo:

$$H(x, \sigma) = \begin{bmatrix} L_{xx}(x, \sigma) & L_{xy}(x, \sigma) \\ L_{xy}(x, \sigma) & L_{yy}(x, \sigma) \end{bmatrix} \quad (4)$$

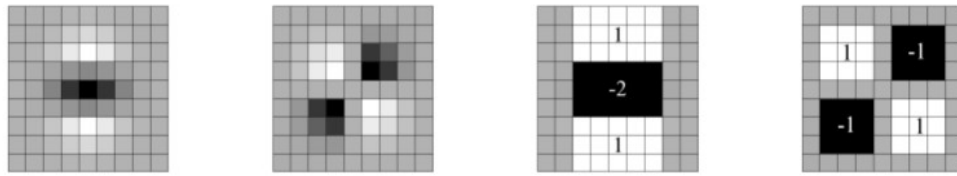
No algoritmo ela é discretizada em 3 máscaras de convolução, uma para cada sentido. No algoritmo SURF, é utilizado uma aproximação muito básica da função LoG (Laplaciana da Gaussiana), também chamada de filtro caixa. Na Figura 16 estão representadas as máscaras de convolução. As duas matrizes da esquerda são referentes à LoG e as duas matrizes da direita são referentes à aproximação simplificada (filtro caixa) utilizada no algoritmo SURF.

Após a composição da matriz hessiana é possível obter o determinante da matriz aplicando a fórmula abaixo:

$$\det(H_{\text{approx}}) = D_{xx} D_{yy} - (w D_{xy})^2 \quad (5)$$

Onde  $w$  representa um balanço do peso dos termos e varia de acordo com a escala da máscara utilizada.

Figura 16 – Máscaras de convolução da matriz Hessiana



Fonte: BAY et al. 2008.

- Supressão de não máximos – Nesta etapa, é feita uma comparação dos valores dos pixels com suas vizinhanças e também com a sua vizinhança correspondente nas matrizes em escalas diferentes da mesma oitava. Esta comparação serve para suprimir os pontos próximos aos pontos com gradientes mais elevados. Esta etapa garante a detecção de pontos de interesse de forma invariante a escala. Ao fim desta etapa obtêm-se duas informações: a coordenada do ponto de interesse e em qual escala ele foi detectado.

#### 2.4.4 Descrição dos pontos de interesse

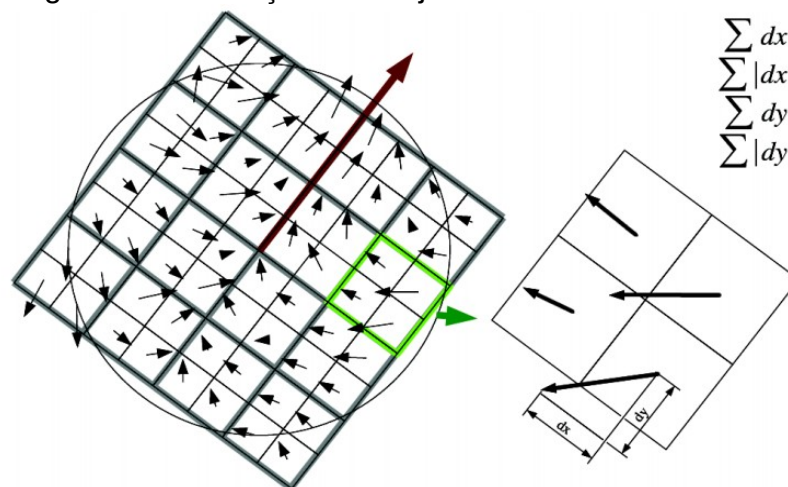
Ao fim da primeira etapa, obtêm-se uma relação de todos os pontos de interesse detectados. Na segunda grande etapa do algoritmo é calculado a orientação dominante e extraídos os descritores SURF.

- Cálculo da orientação dominante – Esta etapa tem a função de garantir que o descritor seja invariante a rotação. Primeiro é definido uma área circular de tamanho  $6s$  em torno do ponto de interesse, em que  $s$  é a escala em que o ponto foi detectado, e após é calculado a resposta do operador Haar-wavelet (GONZALEZ, 2010) nos eixos  $x$  e  $y$ . O tamanho das máscaras do operador também são definidas de acordo com a escala  $s$  para que o cálculo ocorra em apenas 6 operações, independente da escala. Existe uma variante do algoritmo, denominada U-SURF (Upright – Speeded Up Robust Feature), que não executa esta etapa tornando o algoritmo mais rápido porém, não detectando imagens rotacionadas e é indicado apenas para aplicações onde a câmera está sempre na posição horizontal em relação ao ambiente. Na

Figura 17 é possível observar a seta maior com origem no centro do ponto de interesse indicando a orientação dominante.

- Haar-wavelets e o vetor descritor – Nesta etapa são extraídos os descritores SURF. Uma região de tamanho  $20s$  (onde  $s$  é o valor da escala do ponto de interesse) é definida em torno do ponto de interesse e tendo como referência para o eixo  $x$ , a orientação dominante detectada na etapa anterior. Divide-se a nova área em regiões regulares de tamanho  $4 \times 4$ , conforme ilustra a Figura 17. Por último são calculadas as respostas do operador Haar-wavelet para todas as regiões do ponto de interesse em relação aos eixos  $x$  e  $y$  (tendo como referência a orientação dominante calculada).

Figura 17 – Extração do conjunto de descritores SURF



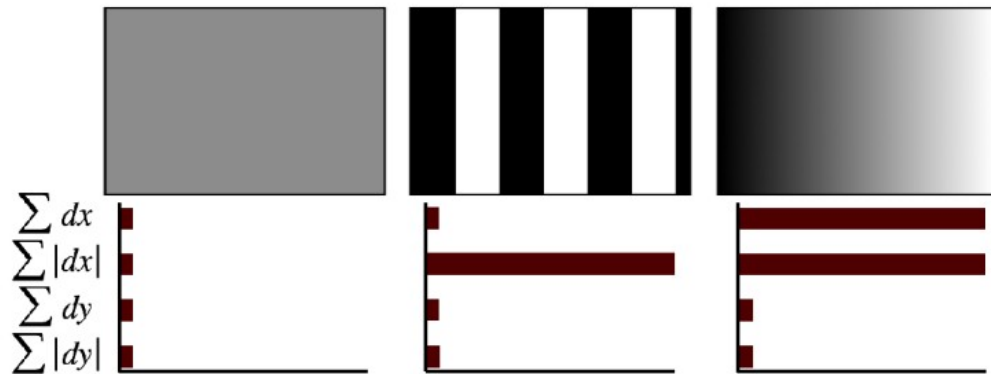
Fonte: BAY et al. 2008.

As respostas do operador (derivadas no eixo  $x$  e  $y$ ) são então somadas em quatro conjuntos para que seja possível descrever a polaridade característica das mudanças de intensidade de cada sub-região. No fim desta segunda etapa o resultado será, para cada ponto de interesse, um vetor contendo as quatro informações citadas para cada sub-região do ponto de interesse, totalizando 64 descritores.

A Figura 18 ilustra o significado de cada atributo que o descritor SURF extrai da imagem: Uma região sem variações na intensidade (esquerda) possui valores baixos nos quatro somatórios das derivadas. Uma região com variações altas no eixo  $x$  (centro) tem um valor alto para o somatório do módulo da derivada no eixo  $x$  e valores baixos para os demais somatórios e uma região com uma variação gradiente

no eixo x (direita) ocasiona em um valor alto tanto no somatório da derivada do eixo x quanto no somatório do módulo da derivada.

Figura 18 – Representação dos atributos do descritor SURF



Fonte: BAY et al. 2008.

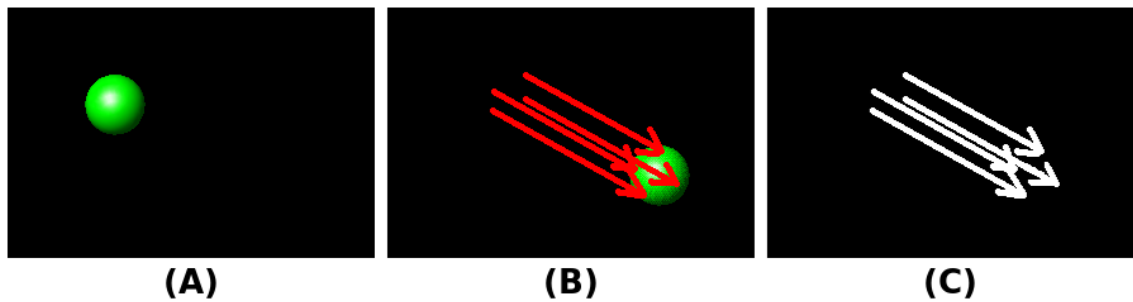
## 2.5 Técnicas de fluxo óptico

Diversas técnicas de fluxo óptico têm sido utilizadas para detectar um movimento em uma sequência de imagens de vídeo. Estas técnicas são empregadas quando se deseja, por exemplo, perseguir objetos em vídeos, ou mesmo estimar o tempo de colisão entre objetos na cena. Os métodos para detecção de fluxo óptico mais tradicionais são Lucas & Kanade e Horn & Schunck. As vantagens e desvantagens de cada um deles é aqui explorada, destacando características como: complexidade dos algoritmos e tempos de execução.

Fluxo óptico fornece um vetor contendo informações de velocidade aparente, direção e sentido do movimento. A base dos métodos é o operador derivativo, que realiza a análise de intensidade dos pixels em dois *quadros* adjacentes da imagem. O Fluxo Óptico pode, também, ser utilizado na segmentação de imagens, identificando regiões que correspondem a diferentes objetos.

A Figura 19 ilustra o fluxo óptico em dois *quadros* adjacentes em um vídeo. A Figura 19.a ilustra o *quadro* 1 com a bola em sua posição inicial. A Figura 19.b ilustra a bola em sua posição consecutiva, destacando as setas que indicam movimento entre os *quadros*. A Figura 19.c lustra os vetores de fluxo óptico dado pelos vetores de deslocamento.

Figura 19 – Fluxo Óptico. a) quadro 1. b) quadro 2. c) Fluxo Óptico



Fonte: WIKIPÉDIA, 2018

O fluxo óptico é utilizado com diferentes aplicações no processamento e análise de imagens, como por exemplo: interpretação de cena, navegação exploratória, acompanhamento de objetos, avaliação de tempo de um corpo contra o outro e segmentação de objetos.

Os métodos de Lucas & Kanade e Horn & Schunck são técnicas de fluxo óptico baseadas na derivada, considerando que a intensidade entre quadros adjacentes em uma sequência de imagens é praticamente constante. Desta forma, utilizando equações diferenciais é possível calcular o fluxo óptico.

Apenas as equações diferenciais não são suficientes para determinar o fluxo óptico, dessa forma são necessários os diferentes métodos. O Lucas & Kanade propõe um método onde, para encontrar o fluxo óptico, outro conjunto de equações é necessário. A solução dada por Lucas & Kanade é um método não iterativo que assume um fluxo óptico constante local. Já no método Horn & Schunk, a velocidade da imagem é computada a partir das derivadas espaço-temporal das intensidades na imagem (OLIVEIRA NETO; GOMES, 2018).

### 2.5.1 Método de Lucas & Kanade

A abordagem deste método consiste em dividir a imagem em janelas, e calcular se os pixels mudaram de lugar entre as janelas vizinhas.

O trecho de programa ilustrado na Figura 20 mostra uma implementação deste método para localização de fluxo óptico. O custo do método Lucas & Kanade será calculado a partir deste código.

Figura 20 – Trecho código Lucas e Kanade

```

1 function [u, v] = LucasKanade(im1, im2, ...
   windowSize);
2
3 [fx, fy, ft] = ComputeDerivatives(im1, im2);
4
5 u = zeros(size(im1));
6 v = zeros(size(im2));
7
8 halfWindow = floor(windowSize/2);
9 for i = halfWindow+1:size(fx,1)-halfWindow
10     for j = halfWindow+1:size(fx,2)-halfWindow
11         curFx = fx(i-halfWindow:i+halfWindow, ...
12                 j-halfWindow:j+halfWindow);
13         curFy = fy(i-halfWindow:i+halfWindow, ...
14                 j-halfWindow:j+halfWindow);
15         curFt = ft(i-halfWindow:i+halfWindow, ...
16                 j-halfWindow:j+halfWindow);
17
18         curFx = curFx';
19         curFy = curFy';
20         curFt = curFt';
21
22         curFx = curFx(:);
23         curFy = curFy(:);
24         curFt = -curFt(:);
25
26         A = [curFx curFy];
27
28         U = pinv(A'*A)*A'*curFt;
29
30         u(i,j)=U(1);
31         v(i,j)=U(2);
32     end;
33 end;
34
35 u(isnan(u))=0;
36 v(isnan(v))=0;

```

Fonte: OLIVEIRA NETO; GOMES, 2018.

O *loop* que inicia na linha 9, é onde tem-se o maior custo de processamento para o método. O custo deste *loop* depende do tamanho da janela dado por *windowSize*, que será chamado de  $n$ . Quanto maior o valor de  $n$ , menos vezes os laços executarão; por outro lado, as operações internas serão feitas sobre imagens de tamanho de  $w$ , fazendo que, com um valor do parâmetro menor as operações internas sejam menos custosas.

O método Lucas & Kanade é de complexidade quadrática, ou seja ( $n^2$ ). Entretanto existe o custo de multiplicar e calcular as inversas de  $w$  e embora  $w < n$ ,  $w$  possui graus altos e ainda é multiplicado por  $n^2$ , o que significa que, em prática, ele terá uma grande impacto no tempo de execução.

## 2.5.2 Método de Horn e Shunck

Inicialmente o algoritmo analisa se a imagem está ou não na escala de cinza e a conversão de tipos para permitir o cálculo. Depois é realizada a suavização da imagem. A Figura 21 ilustra o código do método Horn-Shunck para determinação do fluxo óptico.

Figura 21 – Código Horn-Shunck

```

1 function [u, v] = HS(im1, im2, alpha, ite, ...
   displayFlow, displayImg)
2
3 %% Converte as imagens para escala de cinza
4 if size(size(im1),2)==3
5     im1=rgb2gray(im1);
6 end
7 if size(size(im2),2)==3
8     im2=rgb2gray(im2);
9 end
10 im1=double(im1);
11 im2=double(im2);
12
13 im1=smoothImg(im1,1);
14
15 im2=smoothImg(im2,1);
16
17 % Definindo os valores iniciais para os ...
   vetores de velocidade
18 u = zeros(size(im1(:, :, 1)));
19 v = zeros(size(im2(:, :, 1)));
20
21 % Estimando as derivada espaço-temporais
22 [fx, fy, ft] = computeDerivatives(im1, im2);
23
24 % Criando Máscara
25 kernel_1=[1/12 1/6 1/12;1/6 0 1/6;1/12 1/6 ...
   1/12];
26
27 % Iterações
28 for i=1:ite
29     %Calculando as médias locais dos ...
       vetores de fluxo
30     uAvg=conv2(u,kernel_1,'same');
31     vAvg=conv2(v,kernel_1,'same');
32     % Computando vetores de fluxo limitado ...
       por sua média local e as restrições ...
       de fluxo óptico
33     u= uAvg - ( fx .* ( ( fx .* uAvg ) + ( ...
       fy .* vAvg ) + ft ) ) ./ ( alpha^2 ...
       + fx.^2 + fy.^2);
34     v= vAvg - ( fy .* ( ( fx .* uAvg ) + ( ...
       fy .* vAvg ) + ft ) ) ./ ( alpha^2 ...
       + fx.^2 + fy.^2);
35 end
36
37 u(isnan(u))=0;
38 v(isnan(v))=0;
39
40 % Plotando o fluxo
41 if displayFlow==1
42     plotFlow(u, v, displayImg, 5, 5);
43 end

```

O filtro gaussiano realiza a suavização da imagem, com uma máscara já definida, tem complexidade  $O(n^2)$  e nas linhas 13 e 14 são criadas os vetores de velocidades com valores 0, e na linha 22 são calculadas as derivadas entre as duas imagens, adjacentes.

Para os testes foram utilizadas 10 iterações. Dentro do laço, nas linhas 30 e 31 são feitas duas convoluções com uma máscara de tamanho fixo, que, como visto, possui complexidade quadrática. Assim como no método Lucas & Kanade, as últimas linhas apenas substituem os valores que não foram calculados pelo valor 0.

Com isso, pode-se dizer que a ordem de complexidade deste método é  $O(n^2)$ , ou seja o método Horn & Schunk tem complexidade quadrática. De fato, os termos de maior ordem são quadráticos e os demais são constantes.

### 2.5.3 Comparação entre os métodos

Enquanto o método Lucas & Kanade tem complexidade quadrática, existe o custo de multiplicar e calcular as inversas de  $w$  e embora  $w < n$ ,  $w$  possui graus altos e ainda é multiplicado por  $n^2$ , o que significa que, em prática, ele terá um grande impacto no tempo de execução.

Já o método de Horn & Schunk é  $O(n^2)$ , ou seja tem complexidade quadrática. Isto pode ser notado pelo fato de que os termos de maior ordem são quadráticos, e todos os outros são constantes, não variando de acordo com o tamanho da imagem.

Com base nos testes de desempenho realizados no MatLab, são apresentados alguns resultados. O fluxo óptico que é calculado para os dois métodos. Para os testes, foi usada uma janela de tamanho 8 para o Lucas & Kanade foram feitas 5 iterações com o método Horn & Schunck. Para medir o tempo de execução, foram feitos 10 testes com cada tamanho de imagem, e é apresentado a média do tempo de execução para cada um dos tamanhos de imagens testadas. A Tabela 1 ilustra os tempos de execução dos algoritmos para diferentes tamanhos de imagens quadradas. Observa-se que imagens de tamanho grande tem mais dificuldade em serem processadas pelo método de Lucas & Kanade.



Tabela 1 – Comparativo de tempos de execução dos métodos

Tamanho da imagem	Lucas&Kanade	Horn-Schunck
16	0,029407	0,00817
32	0,191916	0,024754
64	0,740171	0,060697
128	2,536055	0,177277
256	9,603240	1,066800
512	35,715323	2,996775
1024	133,563754	12,640983

Fonte: OLIVEIRA NETO; GOMES, 2018

A Figura 22 ilustra o comportamento assintótico do algoritmo Lucas & Kanade em função do tamanho da imagem. Assim, dependendo do tamanho da imagem, o algoritmo de Horn & Schunck é mais recomendado para aplicações que necessitam de resposta em pouco tempo.

Figura 22 – Gráfico comparativo dos métodos de fluxo óptico



Fonte: OLIVEIRA NETO; GOMES, 2018

## 2.6 Segurança e autenticação

Existem tradicionalmente três propriedades fundamentais que regem a segurança em sistemas computadorizados: confidencialidade, integridade e disponibilidade. Porém é comum atualmente adicionar outras duas propriedades: autenticação e não-repúdio. A autenticação garante que cada entidade (um usuário, por exemplo) seja quem alega ser; e o não-repúdio garante que uma terceira parte neutra possa ser convencida de que um evento tenha ou não ocorrido. Uma entidade solicita a autenticação ao sistema informando sua identidade em conjunto com uma credencial, que é a prova que evidencia sua identidade. A autenticação é concedida ou negada com base na identidade associada à entidade que solicita a autenticação, além de ser verificado se a credencial apresentada constitui prova suficiente (COSTA et. al, 2006).

As credenciais que uma entidade pode apresentar podem ser classificadas em três tipos:

- Posse – Qualquer um que possua o objeto pode obter acesso ao recurso. Como exemplo: a chave de uma casa. Todos que possuem a chave podem acessar a residência.
- Conhecimento – A identidade é comprovada através de algo que o usuário sabe. Uma autenticação baseada em senha, por exemplo.
- Biometria – A autenticação por biometria é baseada em traços da pessoa que possam ser medidos e computados. Como: a impressão digital ou o reconhecimento facial.

### 2.6.1 Autenticação biométrica

A autenticação por biometria se dá quando a credencial para autenticar o usuário é um conjunto de características fisiológicas ou comportamentais da pessoa. Porém para que uma característica seja utilizada como credencial ela precisa satisfazer alguns requisitos (PENTEADO, 2009):

- Universalidade – A característica biométrica tem que estar presente em todos os indivíduos, ou todos os que utilizarão o sistema. Por exemplo: existem pessoas que não possuem impressões digitais.

- Unicidade – A característica biométrica tem que ser única para cada indivíduo, ou seja, a probabilidade de dois usuários apresentarem a mesma característica tem que ser nula ou desprezível. Na prática, as características podem apresentar maior ou menor grau de unicidade mas devido às limitações tecnológicas (armazenamento, resolução do sensor entre outros), mesmo característica com elevado grau de unicidade, como a impressão digital, a unicidade absoluta não pode ser alcançada.
- Permanência – A característica deve ser imutável. A voz, por exemplo, muda em um indivíduo durante as etapas da vida e pode ser alterada devido a fatores como doenças.
- Coleta – A coleta dos dados da característica tem que ser de fácil aquisição por meio de um dispositivo.
- Desempenho – A acurácia e custo computacional exigidos devem respeitar as limitações do sistema.
- Aceitação – A coleta de dados deve ser tolerante pelo usuário. Ou seja, não pode gerar um transtorno ao indivíduo. Privacidade, higiene e questões culturais podem diminuir o grau de aceitação de determinadas características.
- Impostura – a característica deve ser de difícil imitação. Uma assinatura pode ser imitada por um especialista, por exemplo.

Nenhuma característica biométrica consegue atingir a todos os requisitos com perfeição. A Tabela 2 relaciona algumas características biométricas com os requisitos citados:

Tabela 2 – Comparação entre características e requisitos

Biometria	Univer- sidade	Unicida- de	Perma- nência	Coletabi- lidade	Desem- penho	Aceitabi- lidade	Impos- tura
Face	Alta	Baixa	Média	Alta	Baixa	Alta	Baixa
Impressã o digital	Média	Alta	Alta	Média	Alta	Média	Alta
Geome- tria das mãos	Média	Média	Média	Alta	Média	Média	Média
Íris	Alta	Alta	Alta	Média	Alta	Baixa	Alta
Veias das mãos	Média	Média	Média	Média	Média	Média	Alta
Orelha	Média	Média	Alta	Média	Média	Alta	Média
Digitação	Média	Média	Baixa	Média	Baixa	Média	Média
Odor	Alta	Alta	Alta	Baixa	Baixa	Média	Baixa
DNA	Alta	Alta	Alta	Baixa	Alta	Baixa	Baixa
Termo- grama	Alta	Alta	Baixa	Alta	Média	Alta	Alta
Retina	Alta	Alta	Média	Baixa	Alta	Baixa	Alta
Assina- tura	Baixa	Baixa	Baixa	Alta	Baixa	Alta	Baixa
Voz	Média	Baixa	Baixa	Média	Baixa	Alta	Baixa
Modo de andar	Média	Baixa	Baixa	Alta	Baixa	Alta	Média

Fonte: PENTEADO, 2018

### 3 PROCEDIMENTO METODOLÓGICO

Este trabalho utiliza a metodologia experimental para desenvolver um módulo de autenticação por reconhecimento facial para plataformas embarcadas, baseando-se no kit de desenvolvimento *Raspberry Pi 3 – model B* e tendo como sistema operacional uma distribuição Linux, a *Raspbian Stretch*.

#### 3.1 Preparação do ambiente de desenvolvimento e de teste

Para desenvolver o módulo, é necessário ter dois ambientes previamente configurados: um ambiente para o desenvolvimento do software e outro para o teste e validação. O ambiente de desenvolvimento utilizado é um notebook com as seguintes especificações:

- Processador: Intel® Core™ i3 2.40Ghz x 4 Núcleos
- Memória RAM: 6GB DDR3 1333 Mhz
- Gráfico: Intel® HD Graphics 3000/4000
- Sistema Operacional: Linux Ubuntu 14.04 LTS 64 bits
- Câmera USB: Microsoft LifeCam HD-5000

Para ambiente de testes foi utilizada uma Raspberry Pi 3 modelo B. Segue resumo das especificações:

- Processador: ARM Cortex-A53, 1.2GHz x 4 núcleos
- Memória RAM: 1GB LPDDR2 900 MHz
- Gráfico: Broadcom VideoCore IV
- Sistema Operacional: Raspbian Stretch Junho – 2018
- Câmera USB: Microsoft LifeCam HD-5000

##### 3.1.1 Preparando o ambiente de desenvolvimento

O sistema operacional Linux Ubuntu 14.04, assim como a maioria das distribuições Linux, já possui o interpretador do *Python* instalado por padrão nas versões 2.7 e 3.4.

A biblioteca OpenCV encontra-se nos repositórios de software oficiais do Ubuntu porém, a versão disponível para o Ubuntu 14.04 é o OpenCV 2.4 que está desatualizada e não possui todos os recursos que o projeto precisa. A opção

disponível é baixar o código fonte e compilar a biblioteca. A biblioteca *dlib*, que também será utilizada, está disponível para instalação utilizando o gerenciador de pacotes do *Python*.

Para poder compilar o código, é necessário satisfazer as dependências. No terminal do Ubuntu, entrar com os comandos abaixo:

```
# atualizar o sistema
sudo apt-get update && sudo apt-get upgrade
# instalar ferramentas de desenvolvedor
# que ajudam na configuração
sudo apt-get install build-essential cmake pkg-config
# instalar pacotes para lidar com
# diferentes formatos de imagem
sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev
libpng12-dev
# instalar pacotes para lidar com diferentes formatos de video
sudo apt-get install libavcodec-dev libavformat-dev libswscale-
dev libv4l-dev
sudo apt-get install libxvidcore-dev libx264-dev
# bibliotecas para lidar com interfaces graficas
sudo apt-get install libgtk2.0-dev libgtk-3-dev
# bibliotecas que otimizam operações com matrizes
sudo apt-get install libatlas-base-dev gfortran
# cabeçalhos do Python são necessários para compilar
# o OpenCV com suporte para python
sudo apt-get install python2.7-dev python3-dev
# pacote necessário para a biblioteca dlib
sudo apt-get install libboost-all-dev
```

Com as dependências instaladas, o próximo passo é baixar o código fonte. O código fonte está disponível no *Github* e pode ser baixado diretamente pelo navegador ou por linha de comando. Abaixo segue código para realizar a instalação por linha de comando:

```
# criar diretório temporário para a instalação
mkdir ~/opencv
cd ~/opencv
# baixar o código fonte do OpenCV e descompactar
wget -O opencv.zip \
https://github.com/Itseez/opencv/archive/3.3.0.zip
unzip opencv.zip
```

```
# baixar o código fonte dos pacotes
# opcionais mantidos por terceiros
wget -O opencv_contrib.zip \
https://github.com/Itseez/opencv_contrib/archive/3.3.0.zip
unzip opencv_contrib.zip
```

É recomendável que, para desenvolver *software* em *Python*, seja criado um ambiente virtual e, desta forma, as bibliotecas instaladas para um projeto ficam isoladas e não interferem no funcionamento de outros projetos. Mas com a finalidade de manter o texto o mais simples possível esta etapa será omitida e, as bibliotecas serão instaladas diretamente no ambiente global do sistema operacional.

Outra ferramenta necessária para facilitar a instalação é o gerenciador de pacotes do *Python*, o *pip*:

```
# baixar o código fonte e instalar o pip
wget https://bootstrap.pypa.io/get-pip.py
sudo python get-pip.py
```

A biblioteca *numpy* é uma dependência para que o OpenCV funcione corretamente com a linguagem *Python*, ela possui funções e classes de objetos para trabalhar com operações em matrizes e outros cálculos. As bibliotecas *scipy* e *scikit-image* são opcionais mas possuem recursos que podem ser utilizados em conjunto com a biblioteca *dlib*.

```
pip install numpy
pip install scipy
pip install scikit-image
# a biblioteca dlib pode demorar para completar a instalação
pip install dlib
```

Após instalar todas as dependências, é necessário configurar o *makefile* (arquivo de configuração para o compilador) para compilar a biblioteca. A configuração é feita utilizando a ferramenta *cmake* conforme código a seguir:

```
# criar diretório para compilar o código fonte
cd ~/opencv/opencv-3.3.0/
mkdir build
cd build
# comando para configurar e gerar o makefile
cmake -D CMAKE_BUILD_TYPE=RELEASE \
      -D CMAKE_INSTALL_PREFIX=/usr/local \
```

```

-D INSTALL_PYTHON_EXAMPLES=ON \
-D WITH_V4L=ON \
-D WITH_GTK=ON \
-D OPENCV_EXTRA_MODULES_PATH=~/.opencv/opencv_contrib-3.3.0/
modules \
-D BUILD_EXAMPLES=ON ..
# compilar utilizando 4 núcleos com o argumento "-j4"
make -j4
# instalar a biblioteca quando acabar de compilar
sudo make install
# testando a instalação
python
>>> import cv2
>>> cv2.__version__
'3.3.0'
>>> exit()

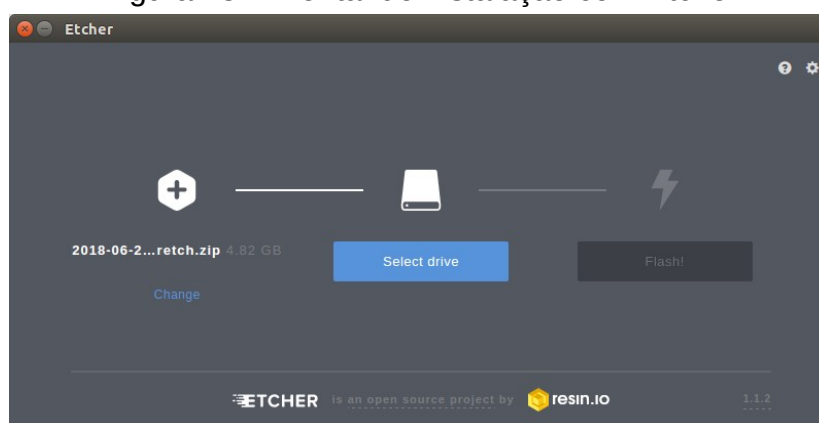
```

### 3.1.2 Preparando o ambiente de testes

O primeiro passo para preparar o ambiente de testes é montar uma instalação do sistema operacional para a plataforma Raspberry Pi em um cartão micro-SD. O sistema operacional *Raspbian* pode ser encontrado no site oficial da *Raspberry Pi Foundation*.

Para montar o sistema operacional no cartão micro-SD, foi utilizado a ferramenta *Etcher*, a qual é recomendada na documentação oficial para a instalação simplificada. A Figura 23 ilustra a instalação do sistema operacional no cartão micro-SD.

Figura 23 – Montando instalação com Etcher



Fonte: Autor, 2018



Quando o sistema operacional inicializa pela primeira vez na Raspberry Pi, um assistente realiza a configuração inicial do sistema. A partição contendo o sistema operacional é expandida para ocupar partes vazias do cartão micro-SD, o acesso à rede Wi-Fi é configurado, entre outras etapas que não impactam no objetivo deste trabalho.

Para instalar as bibliotecas OpenCV e dlib, são seguidos os mesmos procedimentos utilizados para configurar o ambiente de desenvolvimento porém, como a Raspberry Pi possui hardware com restrições, são necessárias algumas modificações para que as bibliotecas sejam compiladas sem erros.

O espaço disponível no cartão micro-SD utilizado é suficiente mas, caso seja utilizado um cartão com pouca capacidade, uma alternativa para liberar espaço é remover alguns softwares desnecessários. O código abaixo remove os softwares LibreOffice (aplicação de escritório) e Wolfram Engine (software matemático) liberando aproximadamente 1GB de armazenamento:

```
sudo apt-get purge wolfram-engine
sudo apt-get purge libreoffice*
sudo apt-get clean
sudo apt-get autoremove
```

A quantidade de memória RAM disponível não é suficiente para compilar a biblioteca *dlib*. Para aumentar a quantidade de memória RAM disponível, é necessário redimensionar a área de *swap*. A área de *swap* é um recurso dos sistemas Linux que permite utilizar uma partição do disco como uma extensão da memória RAM. Para redimensionar a área de *swap*, é necessário editar o campo *CONF\_SWAPSIZE* no arquivo */etc/dphys-swapfile* conforme ilustra a Figura 24.

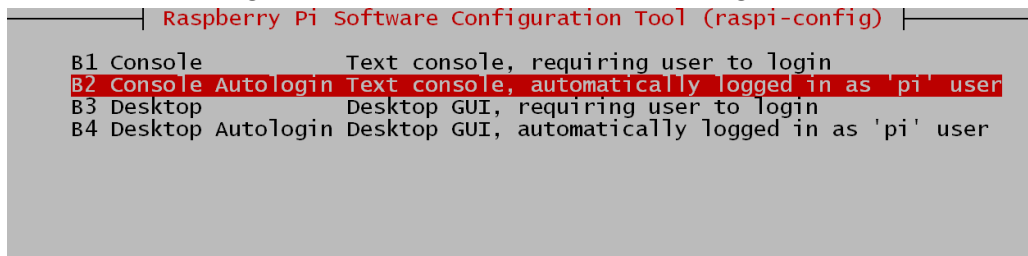
Figura 24 – Arquivo dphys-swapfile

```
8 # the default settings are added as commented out
9
10
11 # where we want the swapfile to be, this is the d
12 #CONF_SWAPFILE=/var/swap
13
14 # set size to absolute value, leaving empty (defa
15 #   you most likely don't want this, unless you h
16 #CONF_SWAPSIZE=100
17 CONF_SWAPSIZE=1024
18
19 # set size to computed value, this times RAM size
20 #   guarantees that there is enough swap without
21 #CONF_SWAPFACTOR=2
22
23 # restrict size (computed and absolute!) to maxim
24 #   can be set to empty for no limit, but beware
```

Fonte: Autor, 2018

Utilizando a ferramenta de configuração *raspi-config*, é possível desativar a interface gráfica e reduzir a quantidade de memória RAM compartilhada para gráficos. A interface gráfica é desativada seguindo as opções: *Boot Options > Desktop / CLI > Console Autologin* conforme destacado na Figura 25.

Figura 25 – Desativando a interface gráfica



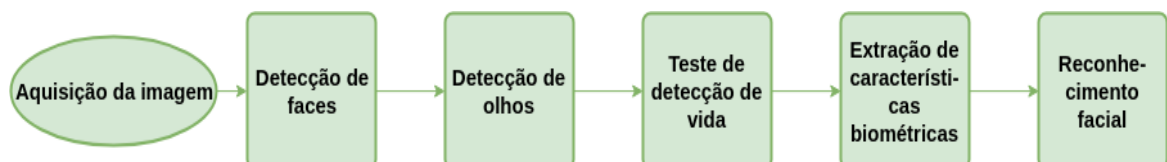
Fonte: Autor, 2018

O ajuste da quantidade de memória RAM compartilhada é encontrado na opção *Advanced Options => Memory Split*. Como a interface gráfica foi desabilitada, pode-se reduzir o valor de 64 para 16 MB. Todas as alterações terão efeito ao reiniciar o sistema operacional. Após realizar os procedimentos para instalar as dependências e compilar a biblioteca OpenCV, as alterações realizadas podem ser desfeitas.

### 3.2 Estrutura do módulo de autenticação

O módulo de autenticação é desenvolvido na linguagem de programação Python 2.7 e utiliza a biblioteca OpenCV 3.3, uma biblioteca de visão computacional de código aberto. O módulo é dividido em processos conforme Figura 26:

Figura 26 – Processos Internos do módulo



Fonte: Autor, 2018

- A aquisição da imagem é feita através de uma câmera USB Modelo Microsoft LifeCam HD-5000, posicionada acima do monitor (ou *display*) do dispositivo. A aquisição é feita de forma voluntária pelo usuário, por isso

não é necessário realizar etapas intermediárias para garantir o posicionamento ideal da face.

- A detecção de faces será feita utilizando o algoritmo proposto por Viola-Jones em conjunto com classificadores treinados e disponibilizados pela *Intel* sob licença livre para fins não comerciais e acadêmicos.
- A detecção de olhos também utilizará o algoritmo proposto por Viola-Jones e tem por objetivo delimitar a área dos olhos para realizar o teste de detecção de vida.
- O teste de detecção de vida consiste em monitorar o movimento dos olhos utilizando técnicas de fluxo óptico. Um objeto-alvo realizará movimentos aleatórios na tela do dispositivo e se os olhos acompanharem o movimento do objeto será possível afirmar que a face detectada pertence a um ser vivo.
- Na etapa de extração das características biométricas, as características serão extraídas utilizando o algoritmo SURF (*Speeded Up Robust Features*).
- Por último, na etapa de reconhecimento facial, as características extraídas irão compor a credencial do usuário. A credencial será comparada com as credenciais cadastradas e caso haja correlação, a autenticação é validada.

### 3.3 Aquisição da imagem

A etapa inicial do processamento é a aquisição da imagem. O objetivo desta etapa é capturar a imagem através da câmera USB e tratá-la de forma a melhorar os resultados nas etapas posteriores.

Para melhorar os resultados na etapa de aquisição da imagem, é necessário realizar ajustes nos parâmetros da câmera. A biblioteca OpenCV possui um método para realizar estes ajustes, o `cv2.VideoCapture.set(parâmetro, valor)` porém, é possível modificar apenas alguns parâmetros, conforme Tabela 3. Esta limitação dá-se por dois motivos: O primeiro é que estes parâmetros são modificados através da biblioteca v4l2 (*video-for-linux 2*) em plataformas Linux e as chamadas de sistema para esta biblioteca não tem suporte total em todas as versões. O segundo motivo é que as câmeras USB utilizam *drivers* de dispositivo diferentes de acordo com fabricante e modelo e nem sempre possuem todos os métodos de acesso aos parâmetros implementados no *driver*.

Tabela 3 – Lista de parâmetros acessíveis pela biblioteca OpenCV

Nome do parâmetro	Nº p/ acesso	Disponível	Val. Padrão	Leitura / Escrita
CV_CAP_PROP_POS_MSEC	0	Não	--	--
CV_CAP_PROP_POS_FRAMES	1	Não	--	--
CV_CAP_PROP_POS_AVI_RATIO	2	Não	--	--
CV_CAP_PROP_FRAME_WIDTH	3	Sim	640	Sim / Sim
CV_CAP_PROP_FRAME_HEIGHT	4	Sim	480	Sim / Sim
CV_CAP_PROP_FPS	5	Não	--	--
CV_CAP_PROP_FOURCC	6	Não	--	--
CV_CAP_PROP_FRAME_COUNT	7	Não	--	--
CV_CAP_PROP_FORMAT	8	Não	--	--
CV_CAP_PROP_MODE	9	Não	--	--
CV_CAP_PROP_BRIGHTNESS	10	Sim	0.45777	Sim / Sim
CV_CAP_PROP_CONTRAST	11	Sim	0.5	Sim / Sim
CV_CAP_PROP_SATURATION	12	Sim	0.41499	Sim / Sim
CV_CAP_PROP_HUE	13	Não	--	--

Fonte: Autor, 2018

Durante os testes com a aplicação, observou-se que alguns recursos da câmera USB interferem no funcionamento do software. O autofocus faz com que o algoritmo de detecção da face não consiga rastrear a face detectada e por este motivo deve estar desligado. Outro parâmetro com impacto é o controle automático da exposição que pode interferir na etapa de préprocessamento da imagem. Como solução para realizar os ajustes iniciais na câmera foi utilizado a ferramenta *uvcdynctrl* (UVCDYNCTRL, 2013), uma aplicação que permite ajustar e obter os valores dos parâmetros da câmera diretamente por linha de comando ou através de um arquivo de configuração. Esta ferramenta está disponível nos canais de software padrão do sistema.

O *quadro* após ser capturado ainda precisa passar por uma etapa de préprocessamento para diminuir o impacto dos fatores externos no processamento da imagem e para ressaltar as informações relevantes da imagem. Para esta aplicação foi identificado que converter o *quadro* para *grayscale* e equalizar o histograma foi suficiente para realizar a etapa de detecção da face.

### 3.4 Detecção da face e olhos

O algoritmo Viola & Jones foi utilizado para detectar faces na imagem. Este algoritmo precisa de um banco com descritores treinados para detectar os objetos. A *Intel* disponibiliza bancos previamente treinados distribuídos sob licença livre para uso não-comercial e acadêmico, o que permite pular a etapa de treinar um banco de descritores para detectar as faces e olhos.

Após realizar testes com este algoritmo na plataforma embarcada, verificou-se uma significativa redução do desempenho. O módulo desenvolvido estava ocupando totalmente apenas um dos núcleos do processador que possui 4 núcleos. Foi medido o tempo de processamento de cada etapa no ambiente de desenvolvimento, a fim de definir quais etapas precisariam ser divididas em subprocessos para utilizar melhor o processador. O resultado é apresentado na Tabela 4.

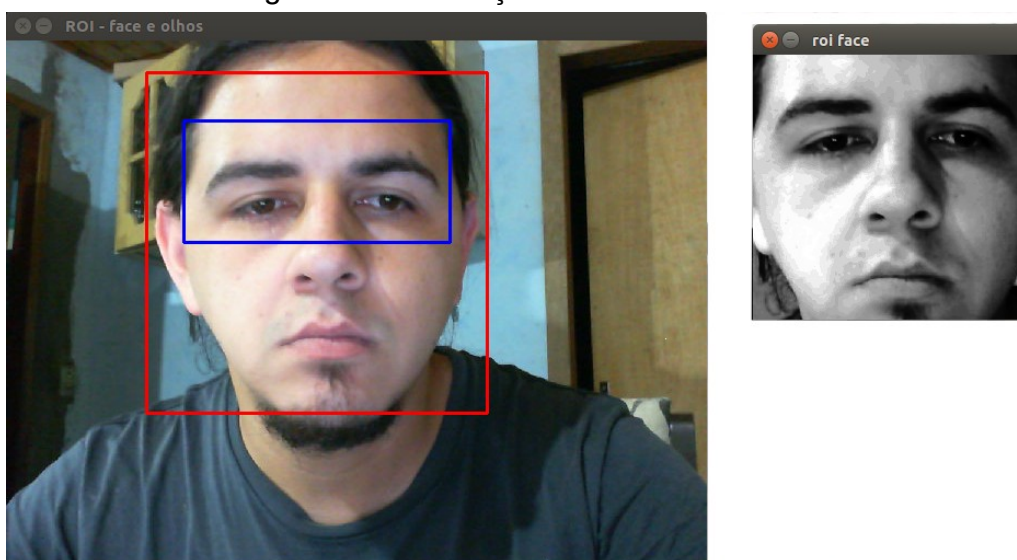
Tabela 4 – Tempo de processamento

Etapa	Tempo de processamento (ms)
Captura do <i>quadro</i>	56226,56
Converter p/ <i>grayscale</i>	1843,17
Detectar face	95593,69
Exibe face detectada	291,84
Detectar olhos	3171,16
Exibe olhos detectados	88,16
Mostra <i>quadro</i> completo	575,66

Fonte: Autor, 2018

O algoritmo de Viola & Jones tem um custo computacional elevado, principalmente na detecção da face, por ter uma área de imagem maior para processar. Por este motivo, optou-se por utilizar o algoritmo apenas para detectar a face e não para continuar rastreando-a após a detecção, a Figura 27 ilustra o algoritmo em funcionamento. Para rastrear a face após a detecção, foi utilizado o algoritmo *correlation tracker* presente na biblioteca *Dlib*.

Figura 27 – Detecção e rastreo da face



Fonte: Autor, 2018

A biblioteca *Dlib* é uma biblioteca de código aberto que implementa uma variedade de algoritmos de aprendizado de máquina e processamento de imagem. Entre estes algoritmos está o *correlation tracker* (que é baseado no artigo de Danelljan et al.'s 2014), o algoritmo realiza o rastreo do objeto na imagem e é tolerante a deslocamentos e alterações na escala do objeto. Utilizando este

algoritmo para rastrear a face, o percentual de utilização do processador da *Raspberry* caiu de 78% para 41%, apresentando picos de processamento somente quando a face precisava ser novamente detectada pelo algoritmo Viola & Jones.

Para a detecção dos olhos é realizado o mesmo procedimento descrito acima, porém tendo como entrada apenas a região dos olhos conforme destacado em azul na Figura 27. A região em azul é fixa e é calculada a partir do tamanho da região de interesse da face detectada com a equação a seguir:

$$\begin{aligned} X1 &= x + (w/9) \\ X2 &= x + w - (w/9) \\ Y1 &= y + (h/7) \\ Y2 &= y + (h/2) \end{aligned} \quad (7)$$

Onde: 'x' e 'y' são as coordenadas da região de interesse da face; 'h' e 'w' são as dimensões da região de interesse da face, altura e comprimento respectivamente.

### 3.5 Tese de detecção de vida

Para que a autenticação por características biométricas seja mais segura, é importante garantir que as informações extraídas venham de um ser vivo. Com a finalidade de dispensar o uso de sensores auxiliares para realizar a detecção de vida, este módulo de autenticação propõe utilizar o movimento dos olhos em um teste de detecção de vida (*liveness detection*).

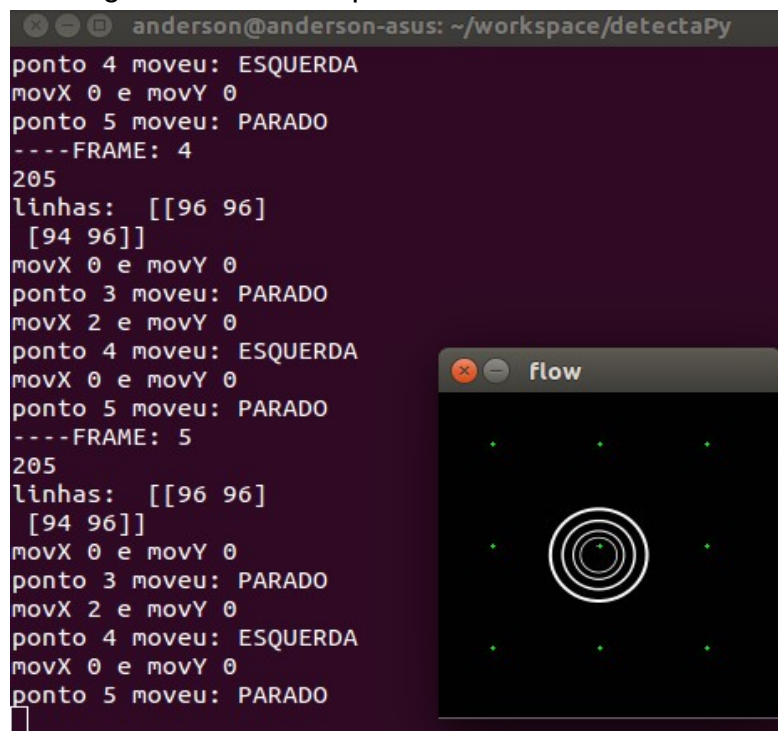
O teste consiste em movimentar um alvo na tela e solicitar que o usuário acompanhe o movimento com os olhos. Se a direção do movimento dos olhos coincidir com o movimento do alvo, é possível afirmar que a informação é proveniente de um ser vivo e não de um vídeo por exemplo.

O movimento do alvo é feito de forma aleatória respeitando apenas algumas regras para que não seja um movimento com pouca intensidade ou que aponte para uma coordenada fora da imagem original.

Para obter o sentido do movimento dos olhos, são utilizadas técnicas de fluxo óptico. O método utilizado para esta etapa é o *cv2.calcOpticalFlowFarneback()*, que é baseado no trabalho de Farneback (2003). Este método foi escolhido porque dispensa a necessidade de localizar um ponto de interesse (como calcular o centroide da pupila, por exemplo) para extrair o fluxo óptico. A Figura 28 ilustra o funcionamento do método utilizando um objeto circular para simular a movimentação

dos olhos, a saída no terminal é referente ao movimento identificado nos três pontos da linha central do *quadro*.

Figura 28 – Fluxo óptico utilizando Farneback



```

anderson@anderson-asus: ~/workspace/detectaPy
ponto 4 moveu: ESQUERDA
movX 0 e movY 0
ponto 5 moveu: PARADO
----FRAME: 4
205
linhas: [[96 96]
[94 96]]
movX 0 e movY 0
ponto 3 moveu: PARADO
movX 2 e movY 0
ponto 4 moveu: ESQUERDA
movX 0 e movY 0
ponto 5 moveu: PARADO
----FRAME: 5
205
linhas: [[96 96]
[94 96]]
movX 0 e movY 0
ponto 3 moveu: PARADO
movX 2 e movY 0
ponto 4 moveu: ESQUERDA
movX 0 e movY 0
ponto 5 moveu: PARADO

```

Fonte: Autor, 2018

O alvo é movimentado uma vez a cada 10 quadros do vídeo. Durante estes 10 quadros, os movimentos dos olhos são identificados e armazenados em uma lista. Quando o alvo for movimentado novamente, a lista é comparada, item a item, com o movimento gerado anteriormente e uma nota é atribuída ao usuário. A detecção de vida é confirmada após o usuário atingir pontuação maior ou igual a 20 pontos. Para mitigar a possibilidade de movimentos aleatórios serem validados, caso a pontuação em 10 quadros sequenciais seja zero, a pontuação do usuário é zerada. O trecho de código a seguir implementa o teste descrito:

```

# metodo que resolve a pontuacao do usuario
def placar(self, alvo):
    # zera o score parcial
    self.score = 0
    # registra a pontuacao parcial item a item
    for passo in range(0, len(self.movimentos)):
        if alvo == self.movimentos[passo]:
            self.score += 1
    # limpa a lista para os proximos 10 quadros
    self.movimentos = []

```



```

# armazena o score parcial
self.totalscore = self.totalscore + self.score
# se o score parcial for zero, reinicia o teste
if self.score == 0:
    self.totalscore = 0
# se o score for alcancado, encerra o teste
if self.totalscore >= 20:
    print 'vitalidade ok'
    self.totalscore = 0
    return 'vivo'
return self.score

```

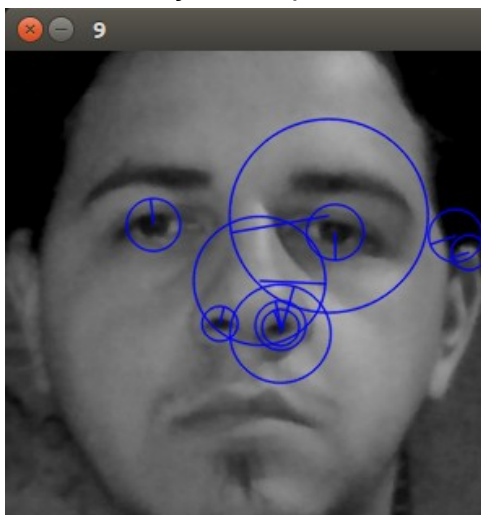
Por restrições no cronograma, este método de detecção de vida não foi exaustivamente testado e por este motivo não foi possível estimar o índice de falsos positivos com precisão. O critério validado durante os testes foi que, apresentando uma fotografia para a câmera, não foi possível atingir uma pontuação maior que 20 pontos.

### 3.6 Extração de características biométricas

Diferente de outros métodos de reconhecimento facial, onde as características biométricas são as medidas de tamanho e de distância entre os órgãos que compõem a face, utilizando o algoritmo SURF, são extraídos pontos de interesse (*keypoints*) na imagem. Com os ajustes realizados no algoritmo, são encontrados, em média, 100 pontos de interesse na face sem adornos. O conjunto de pontos de interesse encontrados na face irão compor a informação biométrica. A Figura 29 ilustra a extração dos pontos de interesse, considerando apenas 10 pontos para uma melhor visualização.

O algoritmo SURF deixou de ser mantido oficialmente na biblioteca OpenCV após a versão 2.4. Por este motivo, é necessário instalar junto com a biblioteca o pacote *opencv-contrib*, o qual contém algoritmos que são mantidos por terceiros e incorporados opcionalmente na biblioteca oficial durante a instalação.

Figura 29 – Extração dos pontos de interesse



Fonte: Autor, 2018

### 3.7 Reconhecimento facial

A etapa de reconhecimento facial é executada após o teste de detecção de vida. Nesta etapa, a região de interesse (a face) já está processada e então é submetida ao algoritmo SURF para extrair as características e compor a credencial do usuário. O algoritmo é tolerante a rotação e escala mas a variação da luminosidade do ambiente pode disfarçar pontos de interesse.

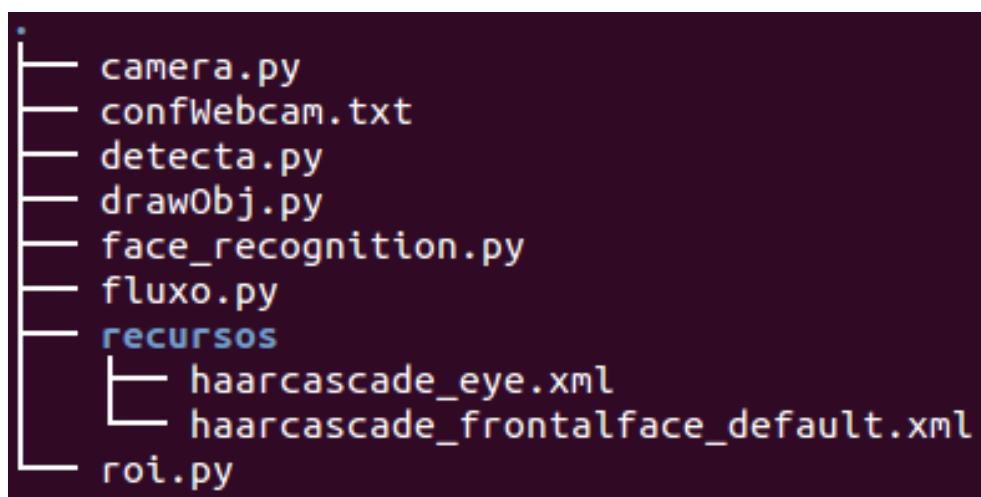
Para minimizar o efeito da luminosidade, uma etapa de pré-processamento foi incluída antes de executar o algoritmo SURF. A etapa adicional consiste em uma correção de gama seguida da equalização do histograma. Para este projeto, o valor ideal para a correção foi encontrado de maneira experimental por tentativa e erro. O valor ideal foi encontrado realizando testes com diferentes graus de iluminação e monitorando a quantidade de pontos de interesse registrados pelo algoritmo. Com o valor encontrado, é possível garantir uma média de 100 pontos de interesse em um ambiente com iluminação artificial. Não foram realizados testes com iluminação natural e em ambientes externos.

O usuário é autenticado realizando uma correlação dos pontos de interesse extraídos no momento da tentativa de autenticação com os pontos de interesse cadastrados anteriormente. O método utilizado para identificar esta correlação é conhecido como *Approximate Nearest Neighbor Search*. Se a taxa de correlação for maior que o limiar definido, a autenticação é validada.

### 3.8 Desenvolvimento do módulo

Nesta sessão serão apresentadas as partes que integram o software desenvolvido. O software foi implementado de forma modular com a finalidade de facilitar o reúso de código em projetos futuros. A Figura 30 apresenta em forma de árvore de diretórios os arquivos de código-fonte e recursos utilizados no projeto.

Figura 30 – Extração dos pontos de interesse



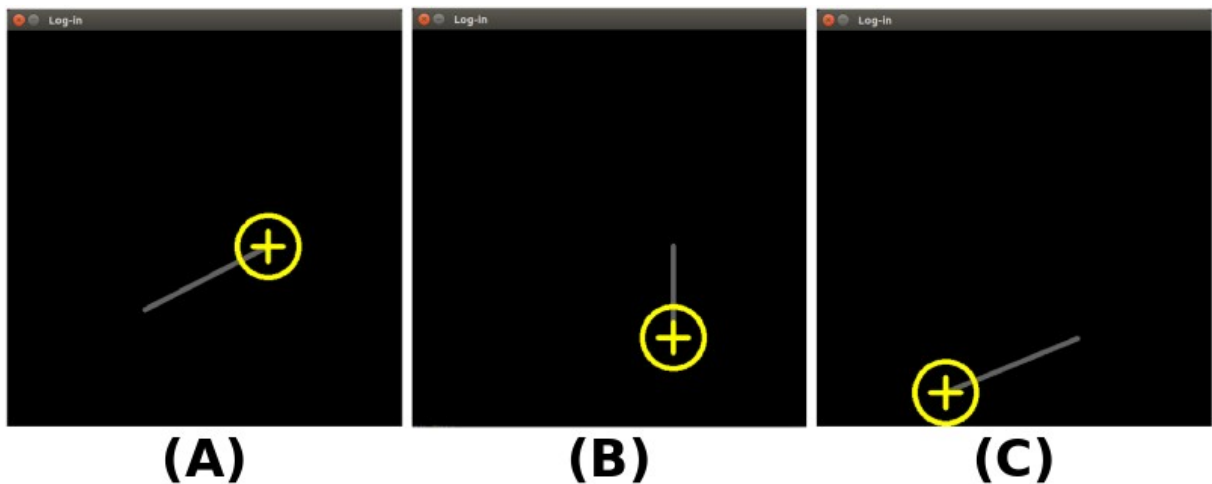
Fonte: Autor, 2018

O arquivo “*camera.py*” descreve a classe “*Camera*” que possui métodos para: inicializar o hardware da câmera, realizar a configuração inicial consultando o arquivo “*confWebcam.txt*”, capturar os quadros e preprocessar a imagem. Ainda implementa um método destrutor da classe que libera o hardware da câmera sempre que o software é encerrado.

Todo o funcionamento do módulo de autenticação está implementado no arquivo “*detecta.py*”, ele é o arquivo principal que realiza as chamadas aos componentes do módulo e realiza a interface com o usuário. É este arquivo que deve ser invocado pelo sistema para realizar a autenticação. Este arquivo não descreve nenhuma classe, apenas possui a lógica sequencial da aplicação.

O arquivo “*drawObj.py*” descreve a classe “*Alvo*” e os métodos necessários para desenhar o alvo na interface com o usuário e gerar os movimentos aleatórios. A Figura 31 ilustra seu funcionamento.

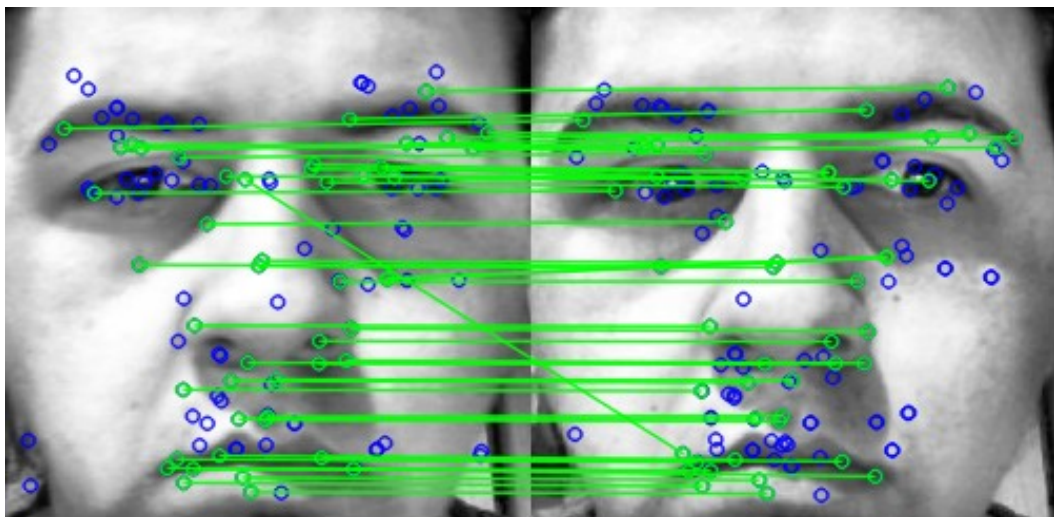
Figura 31 – Classe alvo gerando movimentos aleatórios



Fonte: Autor, 2018

O reconhecimento facial é realizado por meio da classe *FaceRecognition*, descrita no arquivo “face\_recognition.py”. A classe possui métodos para: inicializar o extrator de características com o algoritmo SURF, preprocessar o quadro corrigindo gama e equalizando o histograma, extrair as características da face e realizar o reconhecimento facial através da correlação com a face cadastrada no sistema. A Figura 32 apresenta o resultado do teste comparando a face cadastrada (esquerda) com a face em teste (direita), os pontos em azul são as características encontradas pelo algoritmo e os pontos verdes interligados são os quais apresentaram correlação.

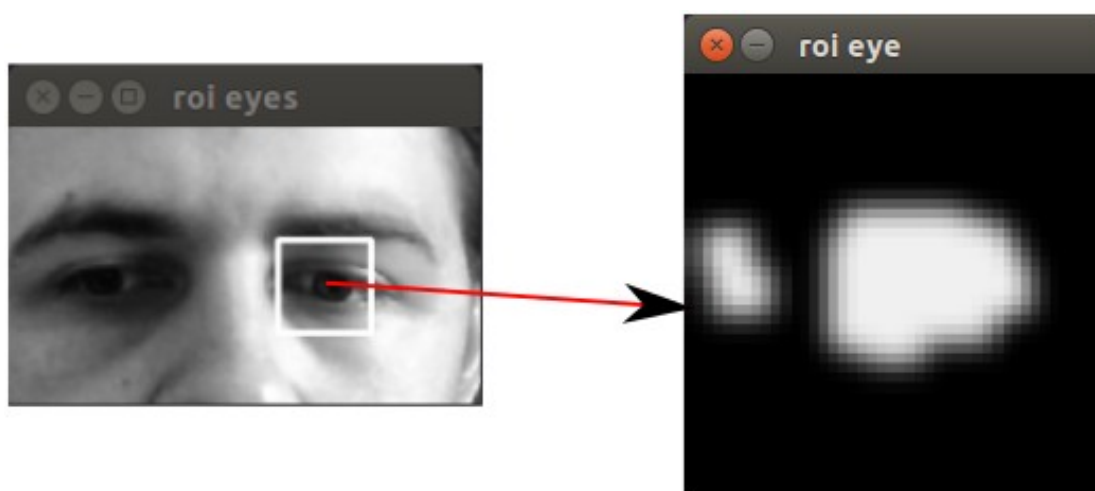
Figura 32 – Resultado do teste com algoritmo SURF



Fonte: Autor, 2018

O arquivo “fluxo.py” descreve a classe *Fluxo* que é responsável por realizar a detecção do movimento do olho através de técnicas de fluxo óptico. Para garantir que apenas o movimento dos olhos seja considerado, a classe possui um método para filtrar a imagem em uma imagem binária, aplica operações morfológicas a fim de expandir o tamanho do olho e por fim, para garantir que movimentos pequenos também sejam detectados, a imagem é borrada com um filtro gaussiano gerando um efeito de textura no objeto. O resultado do filtro é apresentado na Figura 33.

Figura 33 – Região de interesse do olho após filtro



Fonte: Autor, 2018

O arquivo “roi.py” descreve a classe *Detect* que implementa os métodos para detectar e rastrear as regiões de interesse (face e olhos). A detecção é feita através do algoritmo Viola & Jones, fazendo uso dos classificadores presentes no diretório “recursos” e o rastreamento utilizando o algoritmo *correlation tracker* presente na biblioteca *Dlib*.

#### 4 APLICAÇÃO E RESULTADOS

O módulo de autenticação descrito por este trabalho está disponível para download na plataforma *Github* e pode ser encontrado no seguinte endereço:

<https://github.com/andersonrs66/detectaPy/tree/master>.

A classe *FaceRecognition* foi testada para validar a precisão do reconhecimento facial. O teste foi realizado utilizando um banco de imagens formado por dois usuários com a face posicionada em diferentes ângulos. Para realizar o teste foi escolhido aleatoriamente uma amostra de cada um dos usuários e esta imagem foi comparada contra todas as outras presentes no banco de imagens. O resultado do teste está apresentado na Tabela 5. Como o algoritmo já teve sua eficácia comprovada em diversos artigos, a principal motivação para realizar este teste foi a de realizar ajustes no software e definir limiares para a verificação.

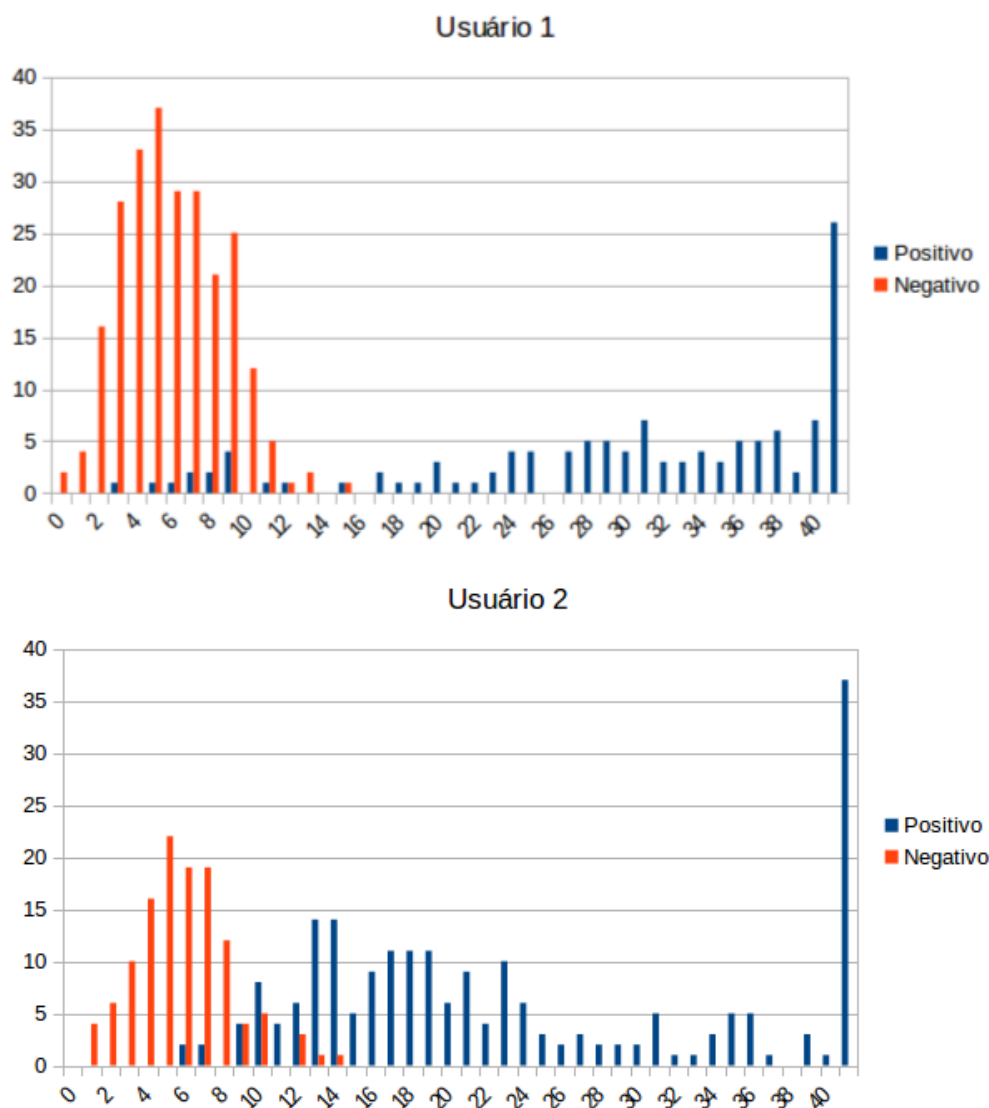
O limiar para considerar se o resultado do teste é positivo ou negativo foi obtido analisando os gráficos do resultado dos testes exibidos na Figura 34. Nos gráficos, as imagens que contém o usuário em teste estão representadas na cor azul e as imagens com outro usuário estão representados na cor laranja. É possível observar que as imagens negativas (que não contém o usuário em teste) não atingem correlações maiores que 15 pontos e, por este motivo, o valor adotado como limiar de decisão do software foi 16 correlações. Com este limiar definido, a precisão do reconhecimento facial ficou em 89,60%.

Tabela 5 – Resultado dos testes de reconhecimento facial

Usuário	Limiar	Total de amostras testadas	Amostras testadas contendo o usuário	Acertos	Erros		Precisão (%)
					Falsos negativos	Falsos positivos	
1	15	367	122	353	13	1	96,18
1	16	367	122	353	14	0	96,18
2	15	335	213	281	54	0	83,88
2	16	335	213	276	59	0	82,38

Fonte: Autor, 2018

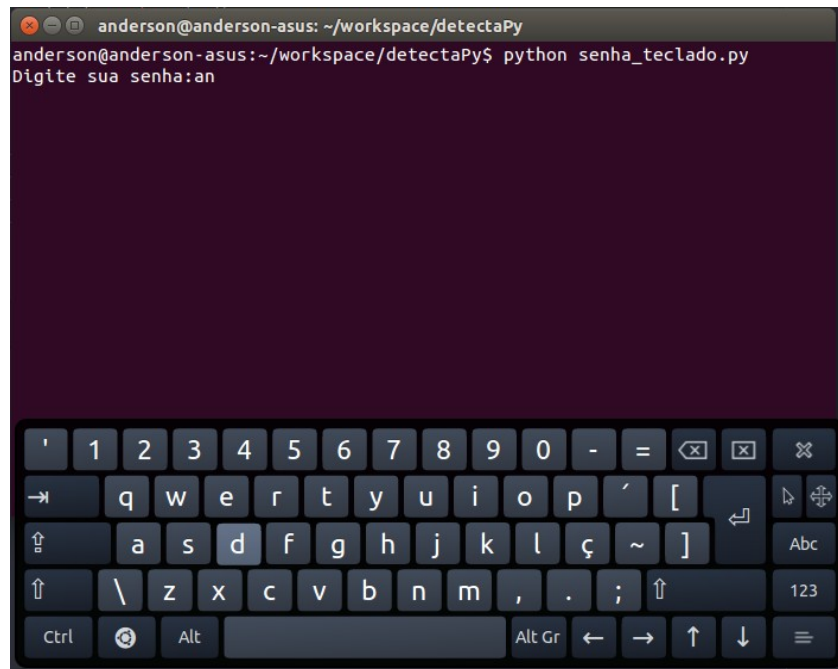
Figura 34 – Gráfico resultado do teste



Fonte: Autor, 2018

O módulo de autenticação foi testado em um grupo de 5 usuários. O objetivo do teste é avaliar a precisão e o desempenho do método de autenticação em comparação com o método tradicional de autenticação por senha. O teste consiste em realizar a autenticação 10 vezes com os dois métodos, cronometrando o tempo necessário para realizar o procedimento. O teste de autenticação por senha foi simulado utilizando um teclado virtual (conforme ilustra a Figura 35) e um *joystick* modelo “Xbox-360 controller” foi usado para controlar a seta do mouse.

Figura 35 – Simulação de autenticação com senha



Fonte: Autor, 2018

O software foi portado para a plataforma embarcada porém, não foi possível otimizar o código com o suporte a multiprocessing a tempo de encerrar este trabalho e, por este motivo, os testes foram executados no ambiente de desenvolvimento. A Tabela 6 e a Tabela 7 apresentam os resultados do teste.

Tabela 6 – Resultado dos testes de autenticação com senha

Usuário	Tempo de autenticação método com senha			
	Menor tempo (s)	Maior tempo (s)	Média dos tempos (s)	Média geral (s)
1	11,68	24,37	15,59	23,99
2	26,26	39,39	31,81	
3	15,45	37,59	27,56	
4	12,47	23,48	17,43	
5	26,60	39,39	32,38	

Fonte: Autor, 2018



Tabela 7 – Resultado dos testes de autenticação com reconhecimento facial

Usuário	Tempo de autenticação método reconhecimento facial			
	Menor tempo (s)	Maior tempo (s)	Média dos tempos (s)	Média geral (s)
1	5,34	30,95	18,78	16,08
2	7,62	31,22	19,08	
3	7,48	49,20	24,06	
4	2,96	24,21	8,04	
5	4,02	17,47	10,31	

Fonte: Autor, 2018

## 5 COMENTÁRIOS FINAIS

Foi objetivo geral deste trabalho desenvolver uma alternativa a autenticação por senha, implementando um método de autenticação por reconhecimento facial, associado à detecção de vida, utilizando ferramentas de código aberto.

Os objetivos específicos foram atingidos parcialmente tendo em vista que o código não foi otimizado para o ambiente de testes. A queda na taxa de quadros por segundo na plataforma de testes impossibilitou validar o software para uso em um sistema embarcado.

Nos testes foi obtido um índice geral de reconhecimento facial de 89,6% no entanto, por restrições no cronograma, não foi possível estimar a precisão do teste de detecção de vida.

O módulo desenvolvido obteve melhor desempenho, a autenticação por reconhecimento facial foi realizada, em média, com 16,8 segundos, em comparação com a média de 23,99 segundos da autenticação por senha.

O último objetivo específico não pôde ser alcançado. Portar o software para um módulo do *Linux-PAM*, ferramenta padrão de autenticação das distribuições Linux, só seria possível após finalizar todos os testes.

Recomenda-se para trabalhos futuros aprimorar o funcionamento da etapa de detecção de vida, que possui carga computacional elevada para um sistema embarcado. Um segundo problema que pode ser estudado em trabalhos futuros é tornar o software tolerante a variações de iluminação.

Deixo ainda como última recomendação, fazer o uso dos ambientes de desenvolvimento e teste descritos neste trabalho para a criação de outros projetos na área de processamento de imagens, tendo em vista que foi uma das etapas com maior dificuldade para encontrar documentação auxiliar.

## REFERÊNCIAS

BAY, Herbert et al. **Speeded-up robust features (SURF)**. Computer vision and image understanding, v. 110, n. 3, p. 346-359, 2008.

BROWN, Eric. **Embedded Linux keeps growing amid IoT disruption, says study**. Março 2015.

Disponível em: <<https://www.embarcados.com.br/hardware-da-raspberry-pi-3/>>

Acesso em: 10 Jul 2018.

COSTA, Luciano R.; OBELHEIRO, Rafael R.; FRAGA, Joni S. Introdução á biometria. **Livro texto dos minicursos do VI simpósio Brasileiro de segurança da informação e de sistemas computacionais (SBSeg2006)**. SBC: Porto Alegre, v. 1, p. 103-151, 2006.

DANELLIAN, Martin et al. Accurate scale estimation for robust visual tracking. In: **British Machine Vision Conference, Nottingham, September 1-5, 2014**. BMVA Press, 2014.

FARNEBÄCK, Gunnar. Two-frame motion estimation based on polynomial expansion. In: **Scandinavian conference on Image analysis**. Springer, Berlin, Heidelberg, 2003. p. 363-370.

GONZALEZ, Rafael C.; WOODS, Richard C. **Processamento digital de imagens**. 3ª Edição, São Paulo: Pearson Prentice Hall, 2010.

HAUPT, Alexandre Gaspar. **Detecção de movimento, acompanhamento e extração de informações de objetos móveis**. Porto Alegre: UFRGS, 2004.

JAMECO. **Raspberry Pi pinout diagram circuit notes**. 2018

Disponível em: <<https://www.jameco.com/Jameco/workshop/circuitnotes/raspberry-pi-circuit-note.html>>

Acesso em: 22 ago. 2018

MARQUES FILHO, Ogê; VIEIRA NETO, Hugo. **Processamento digital de imagens**. Rio de Janeiro: Brasport, 1999.

MATHWORKS Inc. **MathWorks Documentation – Integral image**. 2018.

Disponível em: <<https://www.mathworks.com/help/vision/ref/integralimage.html>>

Acesso em: 26 out. 2018.

MINICHINO, Joe; HOWSE, Joseph. **Learning OpenCV 3 computer vision with Python second edition**. Birmingham, Reino Unido: Packt Publishing Ltd, Setembro 2015.

OLIVEIRA NETO, Vantuil José de; GOMES, David Menotti. **Comparação de métodos para localização de fluxo óptico em sequências de imagens**. 2012. 8 p. Artigo (PPGCC - Programa de Pós-Graduação em Ciência da Computação)-PPGCC - Programa de Pós-Graduação em Ciência da Computação, UFOP - Universidade Federal de Ouro Preto, Ouro Preto, Minas Gerais, Brasil, 2012. Disponível em: <<http://www.decom.ufop.br/menotti/paa111/files/PCC104-111-ars-11.1-VantuilJoseDeOliveiraNeto.pdf>>. Acesso em: 26 out. 2018.

PENTEADO, Bruno E., **Autenticação biométrica de usuários em sistemas de E-learning baseada em reconhecimento de faces a partir de vídeo**. Baurú: UNESP, 2009.

SHAH, Samarth. **Learning Raspberry PI**. Birmingham, Reino Unido: Packt Publishing Ltd, Abril 2015.

SOUZA, Fábio. **O Hardware da Raspberry Pi 3**. Junho 2016.  
Disponível em: <<https://www.embarcados.com.br/hardware-da-raspberry-pi-3/>>  
Acesso em: 22 ago. 2018.

UVCDYNCTRL. **UVCDYNCTRL man page**. Agosto 2013.  
Disponível em: <<https://www.mankier.com/1/uvcdynctrl>>  
Acesso em: 11 nov. 2018.

WIKIPÉDIA. **Fluxo Óptico**.  
Disponível em <[https://pt.wikipedia.org/wiki/Fluxo\\_%C3%B3tico](https://pt.wikipedia.org/wiki/Fluxo_%C3%B3tico)>  
Acesso em: 26 out. 2018.