Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Aplicativo de escaneamento de quadro branco para smartphones da plataforma Android

Edgar Fabiano de Souza Filho

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof. Dr. Bruno Luiggi Macchiavello Espinoza

Brasília
2019

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Aplicativo de escaneamento de quadro branco para smartphones da plataforma Android

Edgar Fabiano de Souza Filho

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof. Dr. Bruno Luiggi Macchiavello Espinoza (Orientador)
CIC/UnB

Prof. Dr. Camilo Chang Dorea
CIC/UnB

Prof. Dr. Wilson Henrique Veneziano
CIC/UnB

Prof. Dr. José Edil Guimarães de Medeiros
Coordenador do Curso de Engenharia da Computação

Brasília, 1 de maio de 2019

Dedicatória

Dedico este trabalho

Aos docentes e discentes do ensino básico, médio e superior do nosso país. Que este trabalho contribua de alguma forma para a melhoria do modelo de ensino que nos acostumamos até o presente momento.

Aos pesquisadores e entusiastas, apaixonados por tecnologia, computação móvel, e processamento de imagens digitais.

Agradecimentos

Agradeço

Pela paciência e colaboração de todas as pessoas que envolveram-se direta ou indiretamente de alguma forma no desenvolvimento desse projeto e tiveram que gastar uma energia extra para compreensão e análise dos resultados do mesmo.

Agradeço carinhosamente aos meus amigos, à minha noiva e aos familiares que fazem valer a pena isso que chamo de vida.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

Neste trabalho serão apresentadas metodologias de processamento de imagens aplicadas para o desenvolvimento de um aplicativo para celulares *Android* capaz de processar uma foto tirada de um quadro branco em um contexto escolar, com objetivo de que imagem final tenha somente o conteúdo escrito no quadro branco, removendo pessoas e objetos que aparecerem na frente, e por fim melhorando o aspecto visual. Em um contexto escolar é muito importante tomar anotações do que é escrito no quadro pelo professor, isso permite que os alunos se lembrem do que foi ensinado em aula. Desde sempre estamos acostumados a tomar anotações em cadernos, uma atividade que pode consumir certo tempo de aula. A cada dia os *smartphones* estão sempre superando novas barreiras de processamento a cada geração, então possuir no bolso uma solução como a que este trabalho propõe é bastante conveniente. Para validar a proposta, um banco de mais de 100 imagens foi preparado, contemplando imagens de quadros brancos com e sem anotações, e com e sem pessoas ou objetos na frente. Foi desenvolvido um aplicativo de celular que implementava a solução proposta neste trabalho, permitindo testar as imagens diretamente no celular. Os resultados das imagens finais foram analisados subjetivamente. A partir dos resultados obtidos, obteve-se uma taxa de sucesso média de 88% para os módulos. 83% para o módulo de detecção do quadro branco, 89% para o de *inpainting*, e 92% para o módulo de melhoria visual. O sistema cumpre seus requisitos na maior parte dos casos, dadas algumas circunstâncias que dependem diretamente da imagem de entrada. Também observou-se que algumas etapas podem ser melhoradas futuramente, como a detecção do quadro na imagem original em alguns casos que a borda do quadro não está tão evidente, ou que algum reflexo interfira em sua continuidade. A otimização velocidade do processamento para remover pessoas ou objetos em primeiro plano da imagem final também pode ser objeto de trabalhos futuros.

Palavras-chave: aplicativo *Android*, processamento de imagens, scanner de quadro branco

Abstract

This work will present image processing techniques applied to the development of a mobile *Android* app that processes a photo taken from a whiteboard in a classroom, until the final image has only the board's written content, removing appearing people and objects in front of it, and enhancing it's visual aspect in the end. On a school context, note taking is a very important activity, allowing students to remember what was written by the teacher in class, we are all used to take notes on notebooks, a very time consuming task. Our everyday smartphones are always getting better and better in computer processing power for each generation, so having a solution like this is very convenient for students. To validate the proposal, a bank containing more than 100 images was prepared, contemplating whiteboard images with and without annotations, with and without people or objects in front of it. An Android app implementing the proposed solution was developed, allowing tests directly on the phone. The final image results were evaluated subjectively. Within the results, the average success rate is 88% for the modules. 83% for the whiteboard detection module, 89% for the *inpainting* module, and 92% for the visual enhancing module. The system meets it's requirements in most of the cases, given the circumstances that directly depend on the input image. It was concluded that some steps can be further improved, such as the frame detection in the original whiteboard image. In some cases that the edge of the frame is not so clear, or some interfering reflex takes place in front of it affecting it's continuity. The optimization of the processing time to execute *inpainting* for hiding people or objects in front of the final image can also be subject from future works.

Keywords: *Android* app, image processing, whiteboard scanning

Sumário

Lista de Figuras

Lista de Algoritmos

Glossário

inter-frame É uma técnica utilizada em compressão de vídeos que tira vantagem da redundância temporal dos quadros, e utiliza relações entre os frames para prever isso.

opensource Ou *software* de código aberto é um modelo de desenvolvimento criado em 1998, que promove o licenciamento livre para o design ou esquematização de um produto, e a redistribuição universal desses, com a possibilidade de livre consulta, examinação ou modificação do produto sem a necessidade de pagar uma licença comercial, promovendo um modelo colaborativo de produção intelectual..

aliasing O aliasing é um efeito que faz com que diferentes sinais se tornem indistinguíveis quando amostrados. Também se refere à distorção resultante quando o sinal reconstruído das amostras é diferente do sinal contínuo original.

anelamento Em processamento de imagens, anelamento é uma distorção no sinal que provoca um efeito de anéis ao redor de uma transição aguda.

cluster É um termo vindo do inglês que em português significa aglomerado, podendo ser aplicado em diversos contextos.

histograma É uma representação gráfica de um conjunto de dados divididos em classes.

interpolação É método utilizado em processamento de sinais que permite construir um novo conjunto de dados a partir de um conjunto discreto de dados pontuais previamente conhecidos.

Lista de Abreviaturas e Siglas

API Application Programming Interface.

AS Android Studio.

FMM Fast marching method (método de marcha rápida).

IDE Integrated Development Environment.

OpenCV Open Source Computer Vision Library.

RGB Red Green Blue.

Capítulo 1

Introdução

O desenvolvimento da tecnologia trouxe conforto e qualidade de vida para a humanidade em situações nunca antes imaginados por gerações passadas. Atualmente já é possível realizar diversas tarefas do dia-a-dia, desde contas simples até prevenção de doenças cardiovasculares [1] utilizando um aplicativo em um dispositivo móvel que carregamos no bolso.

Em um ambiente escolar, tomar anotações do que é escrito no quadro branco pelo professor é muito importante. Isso garante que os alunos se lembrem da aula no momento dos estudos em casa. Assim que o professor termina de escrever no quadro branco, os alunos precisam terminar de copiar logo, caso contrário não mais terão tempo de copiar antes que o professor comece a apagar algumas partes das anotações para escrever novamente.

Partindo do princípio de que a tecnologia serve para facilitar a vida das pessoas, neste trabalho será desenvolvido um sistema implementado em um aplicativo de celular que utiliza algoritmos de processamento de imagens para melhorar o aspecto visual de uma foto tirada de um quadro branco em um ambiente escolar. Uma solução muito conveniente para o problema elencado acima, pois a partir de uma foto tirada do quadro branco diretamente com o celular, é possível processá-la para extrair somente o conteúdo do quadro branco [2], e remover pessoas ou objetos da frente [3], garantindo que independente do ângulo, perspectiva, ou objetos na frente, é possível ter uma imagem final contendo somente a parte do quadro branco que interessa para os alunos.

Somente tirar a foto do quadro branco já poderia ser suficiente para muitos casos, porém espera-se que com este trabalho que alunos possam executar um procedimento equivalente a escanear o quadro branco durante uma aula. A imagem final que será produzida pelo sistema deve ter características visuais semelhantes a uma imagem de um papel escaneado, guardando na imagem somente o que seja de interessante para os alunos: o conteúdo que foi escrito pelo professor.

O sistema a ser desenvolvido deve cumprir os requisitos para que de fato a imagem

final do quadro branco tenha somente o conteúdo do mesmo, com um aspecto visual melhorado, otimizado para uma boa leitura posterior. Para isso, as melhores técnicas de processamento de imagens desenvolvidas até o momento em trabalhos anteriores serão utilizadas em conjunto com outros conhecimentos para o desenvolvimento deste sistema aqui proposto.

Para testar e validar a proposta, será utilizado um banco de mais de 100 imagens contendo quadros brancos e diversas situações em um contexto escolar ou organizacional. Essas fotos contemplam as várias circunstâncias necessárias para validar essa proposta, fotos em ambientes claros, escuros, com e sem pessoas na frente do quadro, com e sem distorção de perspectiva.

1.1 Objetivo geral

O objetivo principal deste trabalho é relatar o desenvolvimento de um sistema implementado em um aplicativo *Android* capaz de escanear o conteúdo do quadro branco a partir de uma imagem digital tirada de um quadro branco. O sistema deve ser capaz de identificar os limites do quadro branco e o recortá-lo da imagem original, retificando-a posteriormente para remover distorções de perspectiva. O sistema deve ser capaz de remover objetos em primeiro plano, como partes do corpo de alguém ou objetos na frente do quadro branco, executando o procedimento de *inpainting* (mais detalhes podem ser vistos na Seção 2.9). Ao final, o sistema também deve ser capaz de melhorar a qualidade visual da imagem final, tornando todos os pixeis do quadro branco homogêneos, destacando os traços nele escritos, deixando tudo com um aspecto visual de uma imagem gerada por um *scanner* de escritório. Tudo isso independente da conectividade do celular à rede internet, ou seja, podendo ser executado de forma *offline*.

1.2 Objetivos específicos

Implementar em um aplicativo, os algoritmos de:

1. Detecção de bordas do quadro branco;
2. Recorte e retificação da imagem do conteúdo do quadro branco;
3. Segmentação por cor para distinguir objetos em 1º plano do quadro branco;
4. Inpainting de objetos em 1º plano;
5. Melhoria visudo aspecto visual da imagem do quadro branco.

1.3 Organização

No Capítulo 2 veremos a fundamentação teórica deste trabalho, abordando todos os conceitos de processamento de imagens necessários para seu entendimento.

No Capítulo 3 faremos uma revisão dos trabalhos relacionados ao que é proposto neste trabalho, e que também serviram como inspiração deste.

No Capítulo 4 abordaremos a metodologia utilizada para o desenvolvimento deste trabalho, explicitando todas as técnicas utilizadas para implementar cada módulo do sistema.

No Capítulo 5 mostraremos detalhes sobre a implementação do aplicativo em si, bem como as tecnologias e ferramentas escolhidas para o seu desenvolvimento.

No Capítulo 6 explicitaremos como os experimentos para validar a proposta deste trabalho foram conduzidos.

No Capítulo 7 retomaremos tudo que foi desenvolvido no trabalho, concluiremos as considerações finais e proporemos o que poderia ser pauta de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Neste capítulo serão descritos os conceitos e conhecimentos de processamento de imagens e algoritmos de programação necessários para o entendimento comum deste trabalho.

2.1 Imagem digital

Uma imagem digital é uma representação bidimensional de uma imagem utilizando números binários codificados para serem armazenados ou transmitidos. Ela pode ser definida como uma função bidimensional *f(x, y)* em que x e y são coordenadas espaciais em um plano, e a amplitude de f em qualquer par de coordenada *(x, y)* é chamado intensidade ou nível de cinza da imagem naquele ponto [4]. Toda imagem digital passa pelo processo de amostragem, e quantização, o resultado é uma matriz de números reais. O termo *pixel* é utilizado para representar cada um desses elementos da imagem, cada pixel guarda a informação de posicionamento *(x, y)* e também de intensidade *f(x, y)*. Uma imagem digital monocromática, também chamada de níveis de cinza é uma imagem em que cada pixel armazena um valor que representa uma única amostra do espaço de cores, geralmente algum tom de cinza. Apenas uma matriz (ou canal) é suficiente para representá-la. Os valores que podem ser utilizados para representar um tom de cinza dependem de como a imagem foi quantizada, em muitos casos 8 bits são utilizados para representar 1 cor em uma imagem, isso faz com que existam 2^8, ou seja, 256 níveis possíveis, onde 0 é totalmente preto e 255 totalmente branco. O mesmo vale para uma imagem colorida, porém canais adicionais são necessários para armazenar os valores referentes às cores. No sistema Red Green Blue (RGB), um canal é dedicado ao vermelho, outro ao verde e outro ao azul, e quando se superpõe-se os pixeis de mesma posição de cada um dos 3 canais, forma-se a imagem colorida.

(a) Imagem original

(b) Imagem em escala de cinza

Figura 2.1: Exemplo de conversão para escala de cinza

2.2 Conversão para escala de cinza

Converter imagem colorida para tons de cinza é requisito para muitas aplicações em processamento de imagem. O objetivo geral é usar o número limitado de tons de cinza para preservar o máximo possível do contraste da imagem original. Imagens em escala de cinza, ou monocromáticas, podem ser resultado de se medir a intensidade de luz em cada pixel de acordo com uma combinação de frequências de luz com pesos. Por exemplo, para se converter uma imagem RGB para escala de cinza, pode-se atribuir a soma com pesos dos valores de cada canal RGB para compor o único canal da imagem em escala de cinza.

$$f(x, y) = \alpha * \mathbf{R} + \beta * \mathbf{G} + \gamma * \mathbf{B} \tag{2.1}$$

Como visto na Equação 2.1 a imagem final Y em escala de cinza é formada pela composição dos valores de RGB com pesos α, β e γ representando porcentagens dos canais da imagem colorida original. A Figura 2.1 mostra um exemplo de uma imagem colorida (com os 3 canais RGB) e a mesma imagem após sofrer o processo de conversão para escala de cinza, contendo somente informação sobre intensidade luminosa em cada pixel.

2.3 Convolução

A convolução é um procedimento matemático linear na área de análise funcional e processamento de sinais, em que dadas duas funções de entrada, obtém-se uma terceira função que resulta da soma do produto dessas funções ao longo da região compreendida pela superposição delas em função do deslocamento entre elas. A convolução pode ser aplicada tanto em funções contínuas quanto discretas. Para funções discretas, a Equação 2.2

$$f(x,y) * h(x,y) = \sum_{i=0;j=0}^{i=M;j=N} f(i,j) * h(x-i, y-j) \tag{2.2}$$

descreve o seu funcionamento, em que *f(x, y)* é uma função discreta bidimensional, como uma imagem, e *h(x, y)* outra função discreta bidimensional, M e N são as dimensões de *f*.

Em imagens digitais o processo de convolução se dá por calcular a intensidade de um determinado pixel em função da intensidade de seus vizinhos. Esse cálculo geralmente é baseado em uma ponderação, isto é, utilizam-se pesos diferentes para pixeis vizinhos diferentes. A matriz de pesos é chamada de *kernel* da convolução. Para obter o novo valor do pixel, multiplica-se o *kernel* pelo valor de cada pixel da imagem original em torno do pixel, elemento a elemento, e soma-se o produto, obtendo-se o valor do pixel na nova imagem.

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.3}$$

Kernel de tamanho 3x3

2.4 Filtros espaciais

Os métodos de filtragem que trabalham no domínio espacial operam diretamente sobre os pixeis, normalmente utilizando operações lineares de convolução com máscaras. O uso de máscaras nas imagens no domínio espacial é usualmente chamado de filtragem espacial e as máscaras são chamadas de filtros espaciais.

2.4.1 Filtro Gaussiano

O filtro Gaussiano é muito utilizado para suavização e remoção de ruídos de imagens [5], uma de suas principais características é a de não conservação de arestas, uma vez que não considera a diferença das intensidades. Possui um comportamento semelhante a de um filtro passa-baixas, porém sua aplicação não gera o efeito de anelamento, em que o nível de desfocagem está relacionado ao desvio padrão σ segundo a Equação 2.4 [6]

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \tag{2.4}$$

em que *G(x, y)* é a função bidimensional que representa a imagem com o filtro Gaussiano aplicado. A Figura 2.2a mostra uma imagem antes de se aplicar o filtro Gaussiano,

(a) Imagem original

(b) Filtro Gaussiano aplicado

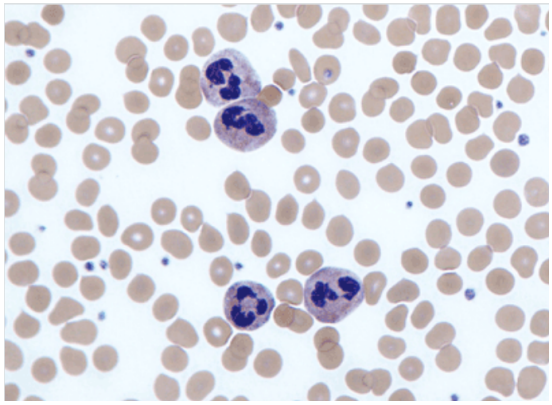
Figura 2.2: Exemplo do filtro Gaussiano

e a Figura 2.2b mostra o resultado final após aplicar o filtro com $\sigma = 4.0$, no caso, um *kernel* de tamanho 9x9. Percebe-se que qualquer ruído que existia na Figura 2.2a foi eliminado, e suas bordas e arestas foram suavizadas.

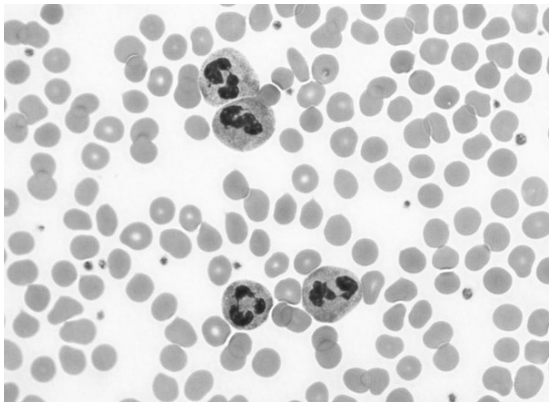
2.5 Binarização

É o processo de converter uma imagem composta por pixeis em uma imagem binária, ou seja, composta por pixeis com valores de 0 ou 1. Esse processo de conversão é precedido por uma operação de *threshold* (limiar) [7]. Uma estratégia de implementação consiste na análise do histograma em escala de cinza de uma imagem, os valores abaixo do limiar são considerados como 0, e os valores acima são considerados o valor máximo na imagem final [8], ou o contrário no caso da operação de binarização inversa. Após a binarização normalmente chama-se de objetos as regiões de primeiro plano, ou seja, que possuem o valor máximo (ou 1), e chama-se de buracos, os objetos de segundo plano (que possuem o valor 0).

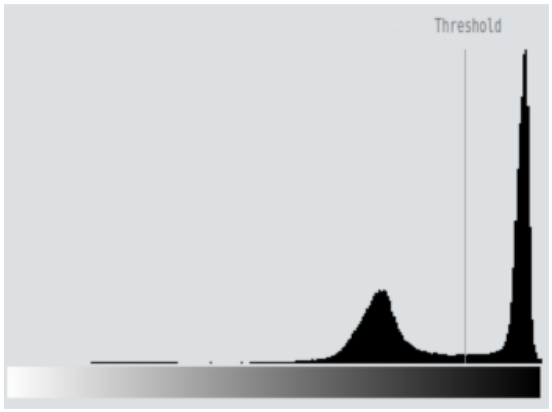
Na Figura 2.3a podemos ver a imagem original em cores, e na Figura 2.3b vemos a imagem original convertida para escala de cinza, na Figura 2.3c vemos o histograma da versão em escala de cinza da imagem original. Está marcado nele o limiar para binarizar essa imagem. Na Figura 2.3d vemos o resultado final da binarização da imagem original. A Equação 2.5 descreve como é feita a binarização em uma imagem,



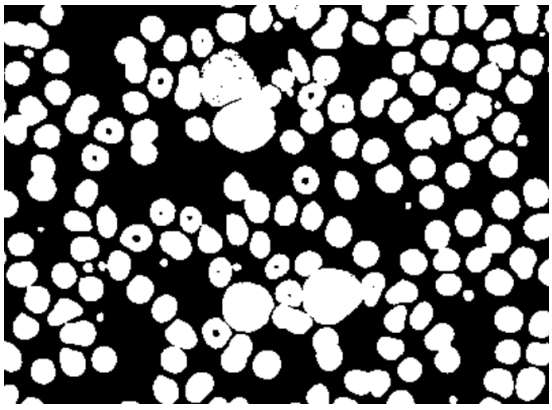
(a) Imagem original



(b) Imagem original em escala de cinza

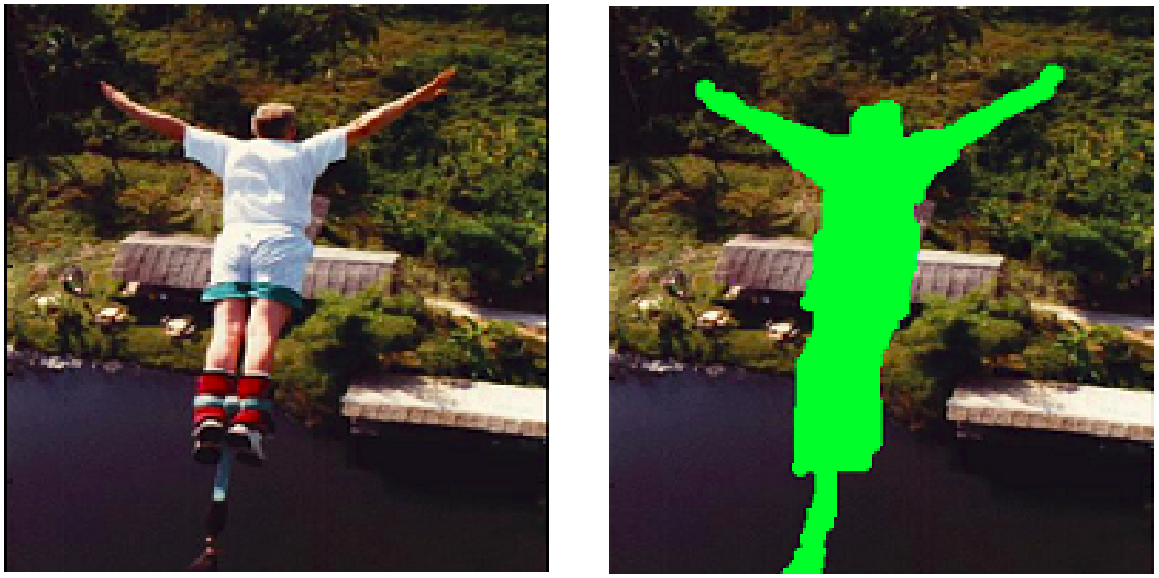


(c) Histograma da imagem escala de cinza



(d) Imagem binarizada com limiar

Figura 2.3: Exemplo de binarização



(a) Imagem original (b) Segmentando a pessoa na imagem

Figura 2.4: Exemplo de segmentação

$$f(x,y) = \begin{cases} 1 \; se \; g(x,y) >= \alpha \\ 0 \; se \; g(x,y) < \alpha, \end{cases} \tag{2.5}$$

em que *f(x, y)* é a imagem binária resultante, *g(x, y)* é a imagem de entrada, e α é o limiar de binarização.

2.6 Segmentação

Em muitos casos, quando estamos no contexto de processamento de imagens digitais, é comum querer se obter dados relacionados com os objetos presentes na imagem [9]. A segmentação é um procedimento que procura dividir e isolar uma imagem em regiões de pixeis que possuem certos parâmetros em comum, como a cor, luminância, formas, objetos, etc, em que as regiões adjacentes devem possuir uma diferença significativa com respeito a essas características.

2.6.1 Detector de bordas Canny

O algoritmo de detecção de bordas Canny é uma técnica de extração de informações estruturais a partir de objetos na imagem. Canny estabeleceu requisitos para os algoritmos de detecção de bordas, com critérios em comum que deveriam atender:

1. Boa Detecção - O algoritmo deve ser capaz de identificar todas as bordas possíveis na imagem.

(a) Imagem original (b) Bordas detectadas na imagem

Figura 2.5: Exemplo de detector de bordas Canny

2. Boa Localização - As bordas encontradas devem estar o mais próximo possível das bordas da imagem original.

3. Resposta Mínima - Cada borda da imagem deve ser marcada apenas uma vez. O ruído da imagem não deve criar falsas bordas.

Para satisfazer esses requisitos, Canny utilizou o cálculo das variações [10]. A função ótima no detector de Canny é descrita pela soma de quatro termos exponenciais, mas pode ser aproximada pela primeira derivada de um filtro Gaussiano (Equação 2.4).

$$G(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{\frac{-x^2}{2\sigma_x^2}+\frac{-y^2}{2\sigma_y^2}} \tag{2.6}$$

Na Figura 2.5a podemos ver a imagem original, e na Figura 2.5b somente as bordas detectadas na imagem original. perceba que a imagem gerada pela operação de detecção de bordas Canny é binária, em que os objetos brancos representam as bordas da imagem original.

2.6.2 Transformada Hough

A transformada Hough é um tipo de transformada de domínio que gera uma operação de extração de características da imagem, identificando mudanças locais de intensidade significativas na imagem, ocorrendo tipicamente na separação de duas regiões diferentes. Originalmente a transformada Hough era focada em identificar pontos pertencentes à

mesma reta em uma imagem, mas evoluiu para também identificar posições de formas arbitrárias, em geral circunferências e elipses [11].

O propósito da transformada de Hough é abordar esse problema de detecção de retas possibilitando agrupamentos de pontos de borda em candidatos a pontos colineares executando um procedimento de eleição explícito sobre um conjunto de objetos de imagem. O caso mais simples da transformada de Hough é a detecção de linhas retas, em geral, a linha reta $y = mx + b$ que pode ser representada como um ponto (b, m) no espaço. Porém, essa representação introduz um problema computacional, as linhas verticais produzem um valor infinito para o parâmetro m, portanto representaremos a reta com a *Hesse Normal form* [12], em que a reta é representada pela Equação 2.7:

$$\rho = x * cos(\theta) + y * sen(\theta), \tag{2.7}$$

onde ρ é a distância da origem para o ponto mais próximo da linha reta, e θ é o ângulo entre o eixo x e a linha conectando a origem com esse ponto mais próximo. Dessa forma é possível computacionalmente associar cada reta da imagem com o par de coordenadas (ρ, θ). O plano formado por essas coordenadas é chamado de espaço Hough, dado um único ponto nesse plano, então o conjunto de todas as retas que atravessam esse ponto corresponde a uma curva sinusoidal no plano (ρ, θ) exclusiva para esse ponto. O conjunto de pontos que formam uma linha reta no plano cartesiano produzirá sinusóides que se cruzam no espaço Hough, dessa forma o problema de encontrar pontos colineares no domínio espacial se torna o problema de encontrar curvas coincidentes no domínio de Hough, vulgo solucionar equações de primeiro grau.

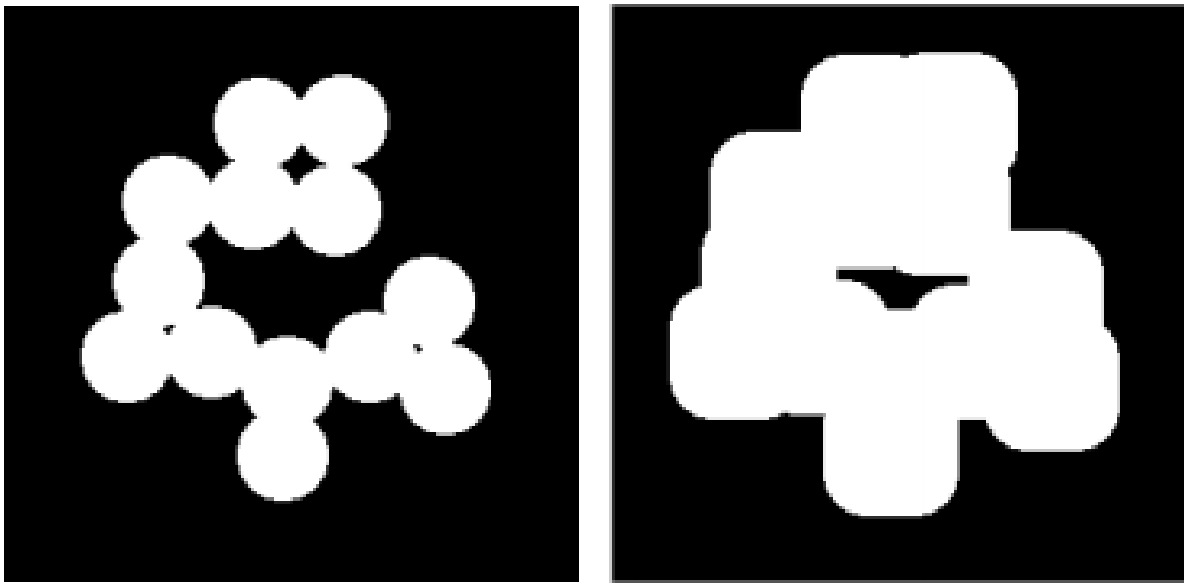
2.7 Operações morfológicas

São operações matemáticas não lineares aplicadas a imagens binárias, ou escala de cinza. Essas operações visam modificar a forma da imagem ou remover imperfeições, em sua maioria causados pelo processo de binarização.

2.7.1 Elemento estruturante

O elemento estruturante é uma pequena imagem binária, ou seja, uma pequena matriz quadrada de pixeis cada um com valor de 0 ou 1, que possui dimensão ímpar, e origem definida como centro da matriz.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & \mathbf{1} & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{2.8}$$



(a) Imagem original (b) Após a dilatação

Figura 2.6: Exemplo de dilatação

Elemento estruturante em cruz de dimensão 3x3

Elementos estruturantes desempenham em imagens binárias um papel análogo ao do *kernel* nos processos de filtragem linear, ou seja, é através do processo de convolução da imagem com o elemento estruturante que se obtém o resultado morfológico esperado.

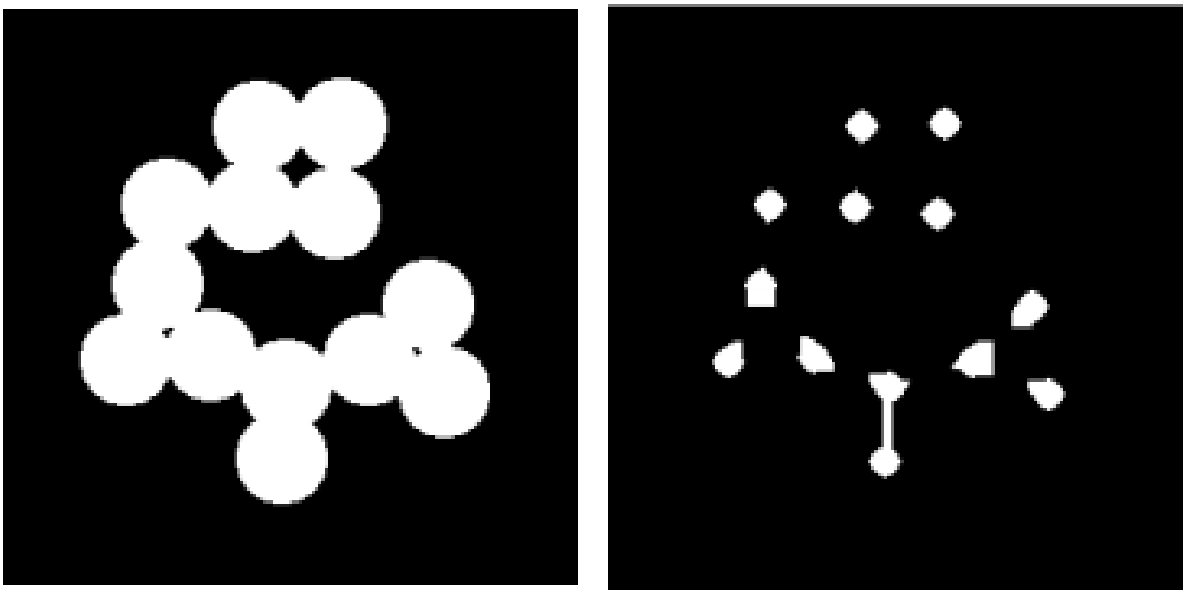
2.7.2 Dilatação

É uma das operações morfológicas primitivas em processamento de imagens, em que são aplicados elementos estruturantes em uma imagem resultando em outra imagem que é a união de diversos deslocamentos na original definidos pelo elemento estruturante. Buracos menores que o elemento estruturante serão preenchidos, unindo os objetos, e os objetos serão expandidos, fazendo com que a imagem final tenha uma área maior que a imagem original. A dilatação é definida pela a Equação 2.9, onde "A"é a imagem original e "O"o elemento estruturante.

$$A \bigoplus O = \left\{ x \| (\hat{O})x \cap A \neq \oslash \right\} \tag{2.9}$$

2.7.3 Erosão

A erosão é outra operação morfológica primitiva, porém produz o efeito inverso da dilatação. Uma imagem binária erodida por um elemento estruturante terá uma área menor do que a imagem original, pois o processo combina os vetores de subtração resultado da



(a) Imagem original

(b) Após a erosão

Figura 2.7: Exemplo de erosão

convolução entre ambas. Objetos maiores que o elemento estruturante terão suas áreas reduzidas, os menores serão removidos e buracos serão expandidos. A erosão é definida pela a Equação 2.10, onde "A"é a imagem original e "O"o elemento estruturante.

$$A©O = \{x|(B)x \subseteq A\} \tag{2.10}$$

2.7.4 Fechamento

A operação de fechamento, assim como a sua operação simétrica a abertura, são muito importantes no campo da morfologia matemática, e podem ser derivadas das operações primitivas. O fechamento é similar em alguns aspectos à dilatação, uma vez que tende a ampliar os limites dos objetos e reduzir o tamanho dos buracos, porém é menos destrutiva quanto aos limites da forma original. Ela é definida simplesmente por uma dilatação seguida por uma erosão utilizando o mesmo elemento estruturante.

2.7.5 Flood fill

Flood fill é um algoritmo de preenchimento, em que dado um ponto em uma matriz, ele busca encontrar a área conectada relativa a esse ponto. Muito utilizado em programas de pintura na ferramenta "Preenchimento com cor"para preencher áreas conectadas com uma cor diferente. Ele pode ser implementado para imagens binárias dessa forma:

O Algoritmo 1 mostra a implementação do *flood-fill* a partir de um pixel.

Algoritmo 1 Flood-fill(pixel)

1: **if** pixel.cor = 1 **then return**
2: Define a cor como 1
3: Executar Flood-fill(pixel acima)
4: Executar Flood-fill(pixel abaixo)
5: Executar Flood-fill(pixel à direita)
6: Executar Flood-fill(pixel à esquerda) **return**

2.8 Retificação planar

A retificação de imagens é um processo de transformação utilizado para projetar imagens em um plano comum de imagem [13]. Isso está relacionado a processes de visão estéreo, como o nosso sistema de visão, ou também duas câmeras enxergando uma cena 3D a partir de posições diferentes, há um certo número de relações geométricas entre os pontos 3D e as suas projeções para as imagens 2D que levam a restrições entre os pontos de imagem, geometria epipolar estuda esse fenômeno, em que os epipolos são as imagens projetadas pelos centros ópticos de cada uma das câmeras envolvidas. O procedimento de retificação pode ser implementado matematicamente como combinações de transformações lineares entre matrizes. Operações de rotação nos eixos X e Y colocam as imagens no mesmo plano, o escalonamento ajusta o tamanho das imagens, e uma rotação no eixo Z fazem com que as linhas de pixeis da imagem se alinhem diretamente. Todas as imagens retificadas devem satisfazer as duas propriedades a seguir [14]:

- Todas as linhas epipolares são paralelas ao eixo horizontal.
- Pontos correspondentes têm coordenadas verticais iguais.

Então para se transformar o par original de imagens em um par retificado de imagens, é necessário encontrar uma transformação linear de projeção H que satisfaça as duas restrições acima.

$$r(x, y) = f\left(\frac{H_{11}x+H_{12}y+H_{13}}{H_{31}x+H_{32}y+H_{33}}, \frac{H_{21}x+H_{22}y+H_{23}}{H_{31}x+H_{32}y+H_{33}}\right) \tag{2.11}$$

A Equação 2.11 mostra a função da retificação planar. Em que *r(x, y)* é a imagem retificada, f é a imagem original e H é uma matriz de transformação de tamanho 3x3.

A Figura 2.8 mostra um exemplo de retificação. O retângulo vermelho seria uma imagem com distorção de perspectiva, e o retângulo azul é a projeção do retângulo circunscrito nela, que deve ser o resultado da retificação.

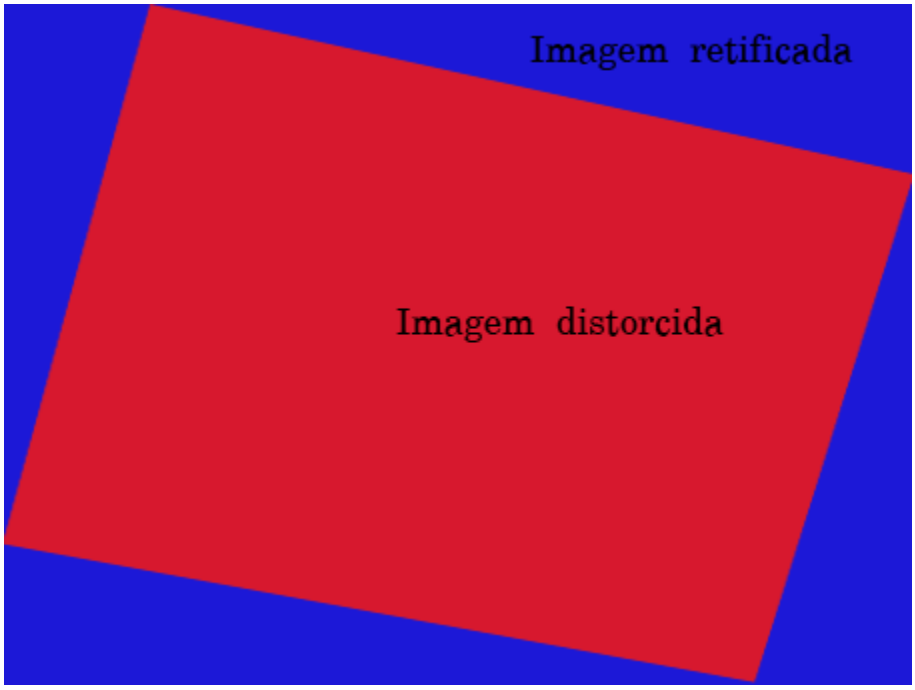


Figura 2.8: Exemplo da retificação.

2.9 *Inpainting*

Inpainting é o processo de reconstrução de partes perdidas ou deterioradas de imagens e vídeos, também é conhecido como interpolação. Em imagens digitais, o *inpainting* é definido como a aplicação de algoritmos sofisticados para substituir regiões perdidas ou corrompidas, em casos particulares pode ser utilizado quando se deseja remover ou mascarar objetos indesejáveis na cena, preenchendo com partes da imagem ao redor [3].

As Figuras 2.9 a 2.11 mostram o exemplo de como o *inpainting* funciona. A Figura 2.9 apresenta uma imagem com uma falha, um risco, ou corte. A Figura 2.10 mostra a máscara que foi utilizada para ocultar a parte indesejada na imagem original. A Figura 2.11 mostra o resultado final após aplicar o *inpainting* com a máscara na imagem original.

Uma implementação do *inpainting* utiliza o Fast marching method (método de marcha rápida) (FMM) [15] como metodologia de propagação de informação para dentro da região a ser pintada.

O Algoritmo 2 mostra a implementação do *inpainting* por Alexandru Telea [3]. Ele percorre a máscara do *inpainting* enquanto ela não estiver vazia, preenchendo o interior da imagem original pelas bordas com pixeis próximos.

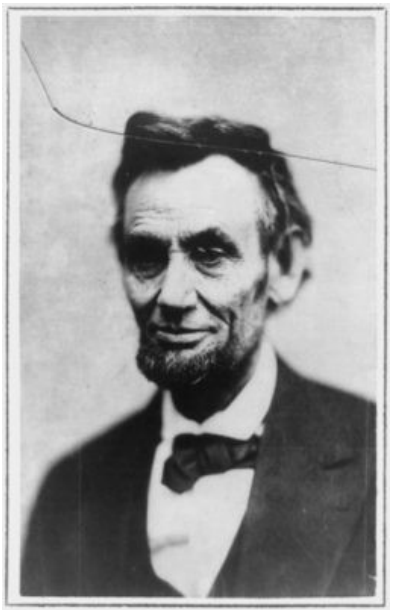


Figura 2.9: Imagem original

Figura 2.10: Máscara

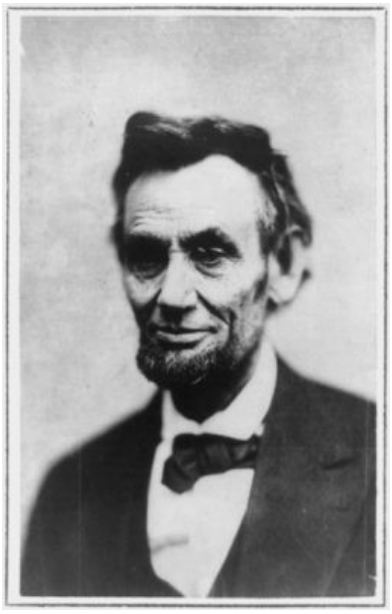


Figura 2.11: *Inpainting*

Algoritmo 2 Inpaint(img, mascara)

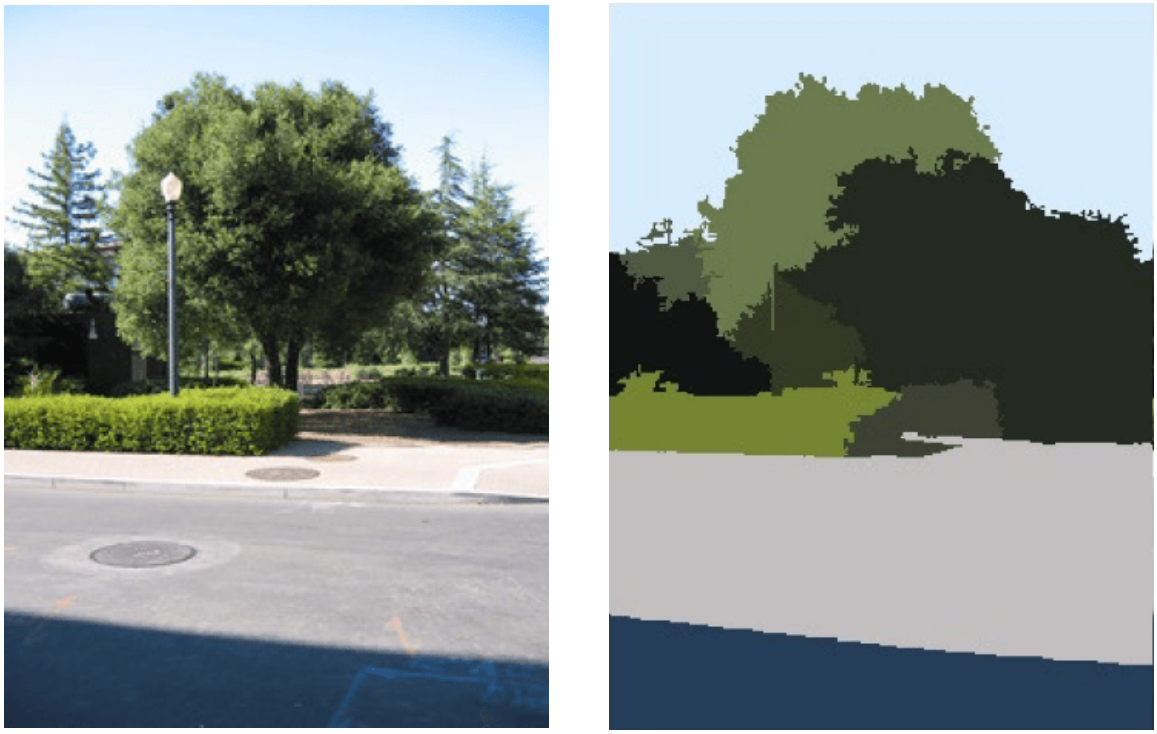
```
b = borda da região da mascara
while mascara not empty do
    p = pixel de img mais proximo do pixel da mascara
    pintar(p)                          ▷ com o pixel externo à b mais próximo
    avançar b 1 pixel para dentro da mascara
return img
```

2.10 Clusterização (*K-means*)

O método de clusterização *K-means* utilizado em processamento de sinais é uma forma de executar a quantização de um sinal. O objetivo dessa técnica é agrupar n valores de um sinal em k clusters, em que valores com características suficientemente em comum pertence a um cluster [16]. Em processamento de imagens *K-means* pode ser utilizado para classificar os pixeis de uma imagem com relação à sua cor.

Na Figura 2.12 pode-se ver o efeito da clusterização *K-means*. Dado o parâmetro K=10, o algoritmo busca classificar todos os elementos da imagem com características em comum, de forma que se tenha 10 clusters, ou agrupamentos, no final. Na Figura 2.12b para se facilitar a visualização, a cor média de todos os elementos de cada um dos clusters foi utilizada para representá-los.

No capítulo seguinte serão abordados os trabalhos relacionado que somados aos conhecimentos abordados neste capítulo, serviram como fonte de inspiração para este trabalho.



(a) Imagem original

(b) Clusterização com k=10

Figura 2.12: Exemplo de clusterização *K-means*

Capítulo 3

Revisão de Literatura

Neste capítulo revisaremos os trabalhos anteriores relacionados ao tema, que inspiraram e contribuíram para o desenvolvimento desse projeto.

3.1 Note-taking with a camera: whiteboard scanning and image enhancement (2004)

Esse artigo relacionado foi o maior motivador para este trabalho, nele é descrito um sistema capaz de digitalizar do conteúdo de um quadro branco por meio de uma câmera digital, e também capaz de melhorar a qualidade visual da imagem do quadro branco [2]. Sua maior motivação é a de que como as câmeras digitais estavam se tornando acessíveis e populares (em 2004) para as pessoas, é proposto então que elas poderiam utilizar essas câmeras digitais para tirar fotos dos quadros ao invés de copiá-los manualmente, aumentando significativamente sua produtividade. O principal problema de utilizar somente a foto crua tirada da câmera é que algumas vezes a imagem poderia ter sido tirada em um ângulo, resultando em uma imagem com distorção na perspectiva, e também poderia conter outras distrações como paredes e outros objetos que não faziam parte do conteúdo principal do quadro branco. O sistema desenvolvido no artigo é capaz de localizar automaticamente os limites do quadro branco, recortar essa região e corrigir as distorções de perspectiva, e tornar o branco do quadro completamente branco, fazendo com que a imagem se pareça com uma folha escaneada. Quando várias imagens eram tiradas do mesmo quadro branco, o sistema também era capas de costurar essas imagens sobrepostas em uma só, realizando o procedimento de *stitching* da fotos. Portanto produziam o conteúdo do quadro branco em um único documento eletrônico fiel, que podia ser arquivado ou compartilhado com todos. Esse sistema foi testado extensivamente, e produziu resultados muito bons.

3.2 Automated Detection of Handwritten Whiteboard Content in Lecture Videos for Summarization (2018)

Neste trabalho relacionado, os autores desenvolveram um método baseado em aprendizagem de máquina para detecção de texto de cena de vídeos de palestras online, com a finalidade de detecção de texto manuscrito, expressões matemáticas e esboços em vídeos de aula. Detectam elementos manuscritos no quadro branco para gerar um resumo de todo o conteúdo ao longo do tempo na palestra, ao mesmo tempo em que lidam com conteúdo ocluso devido ao movimento do palestrante. A principal motivação deste trabalho relacionado [17], se dá ao fato de que vídeos de palestras *on-line* são um recurso valioso para estudantes em todo o mundo, então processar e extrair conteúdos destes vídeos poderia ser mais importante ainda. Os métodos para extração automática do conteúdo do quadro branco reduzem a quantidade de esforço manual necessário para possibilitar a indexação e a recuperação de tais vídeos.

3.3 Whiteboard Documentation through Foreground Object Detection and Stroke Classification(2008)

Esse trabalho relacionado trás por meio dos seus autores uma abordagem muito semelhante ao anterior, nele é proposto um modelo para documentação de conteúdo de quadro branco através da detecção de objetos em primeiro plano [18], em que regiões de objetos detectadas são impressas com o conteúdo recente do quadro branco, isso virtualmente proporciona uma experiência de visualização do quadro branco sem qualquer oclusão, ao final do processamento, as imagens do quadro branco eram exportadas como vídeo, podendo ser armazenadas ou compartilhadas. Sua motivação se baseia praticamente na necessidade de preservar o conteúdo do quadro branco (e também quadro negro segundo o escopo do artigo), pois é um componente valioso do que é apresentado em sala de aula. O grande desafio relacionado a esse contexto é capturar os conteúdos do quadro branco na presença de objetos em primeiro plano, como o professor passando na frente.

3.4 Real-Time Whiteboard Capture and Processing Using a Video Camera for Teleconferencing (2005)

Esse trabalho, ao autores desenvolveram um sistema que captura os traços de um pincel em um quadro branco em tempo real a partir de uma câmera de vídeo [19]. Sua técnica se baseia em analisar a sequência de *frames* do vídeos capturado, e em tempo real e classificar os pixeis que pertencem ao fundo do quadro branco, os que pertencem aos traços do pincel e os que pertencem a objetos em primeiro plano, como pessoas que passam na frente do quadro branco, assim é possível extrair os traços recém escritos. A motivação principal do artigo se baseia em resolver um problema de compartilhamento de anotações durante uma reunião em vídeo conferência, com esse sistema é possível transmitir somente o conteúdo do quadro branco para os demais participantes remotos da reunião, sem interferências de pessoas ou objetos na frente inclusive quando novos traços estivessem sendo escritos no quadro branco.

3.5 Whiteboard Content Extraction and Analysis for the Classroom Environment (2008)

O principal foco desse trabalho é voltado para o desenvolvimento de um sistema de captura do conteúdo de um quadro branco de apresentações organizadas a partir de palestras dentro de um ambiente de sala de aula, ou seja, aulas ministradas por professores [20]. O sistema desenvolvido no artigo é responsável por adquirir uma sequência de imagens a partir de câmeras fixas de alta resolução, e a partir dessas imagens, ser capaz de extrair *frames* que tenham o conteúdo do quadro branco. Os *frames* são derivados ao analisar as imagens da câmera (que deve ser posicionada no meio da sala), selecionando imagens que mostram a progressão do material apresentado, no caso, os traços sendo escritos progressivamente nas imagens projetadas, removendo então digitalmente a imagem do professor que porventura pode ter passado na frente e posteriormente aprimorando as imagens para maior legibilidade. Essas imagens então poderiam então ser distribuídas em tempo real, ou armazenadas para posteriormente serem sincronizadas com outros conteúdos de vídeo ou capturas de computador. A grande vantagem desse sistema em relação aos trabalhos nele relacionados, é que não há a necessidade de nenhum hardware especializado que não seja câmeras de campo de visão fixos, não atrapalha o palestrante, e é capaz de lidar tanto com quadros brancos e apresentações projetadas em telas. Em média, o sistema extrai aproximadamente 70 *frames* de aproximadamente 70000 imagens capturadas durante uma

apresentação de 75 minutos, sendo executado quase em tempo real.

No capítulo seguinte abordaremos a metodologia utilizada para o desenvolvimento deste trabalho, todas as técnicas, a arquitetura principal, o fluxo do aplicativo, etc.

Capítulo 4

Metodologia

Este capítulo mostra como a arquitetura dos módulos do sistema foi desenvolvida, descrevendo todas as etapas em sequência desde a detecção do quadro branco na imagem com retificação, e também o *inpainting* e melhoria visual. Todos os conceitos abordados no Capítulo 2 serão apresentados na prática como foram aplicados ao desenvolvimento deste trabalho. Muitos parâmetros dos procedimentos desse sistema foram obtidos de forma empírica verificando subjetivamente qual produzia o melhor resultado esperado na imagem final, embasado nos trabalhos relacionados já se sabia o resultado esperado, então bastava um ajuste com tentativa e erro para produzir a melhor imagem final. Para efeito de comparação, a Figura 4.1 será utilizada para mostrar como uma imagem evolui ao passar por cada etapa no sistema.

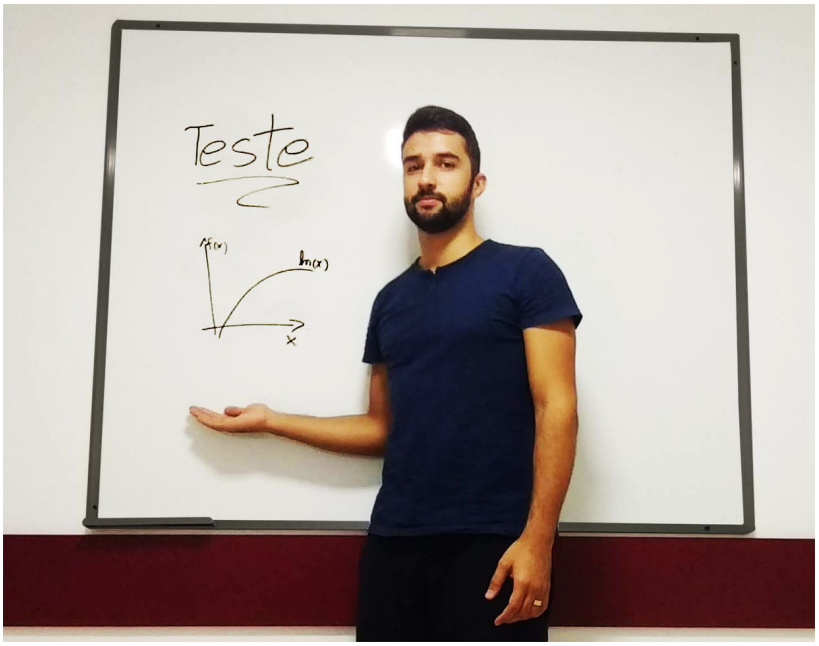


Figura 4.1: Imagem para ser utilizada no fluxo do sistema.

4.1 Arquitetura principal

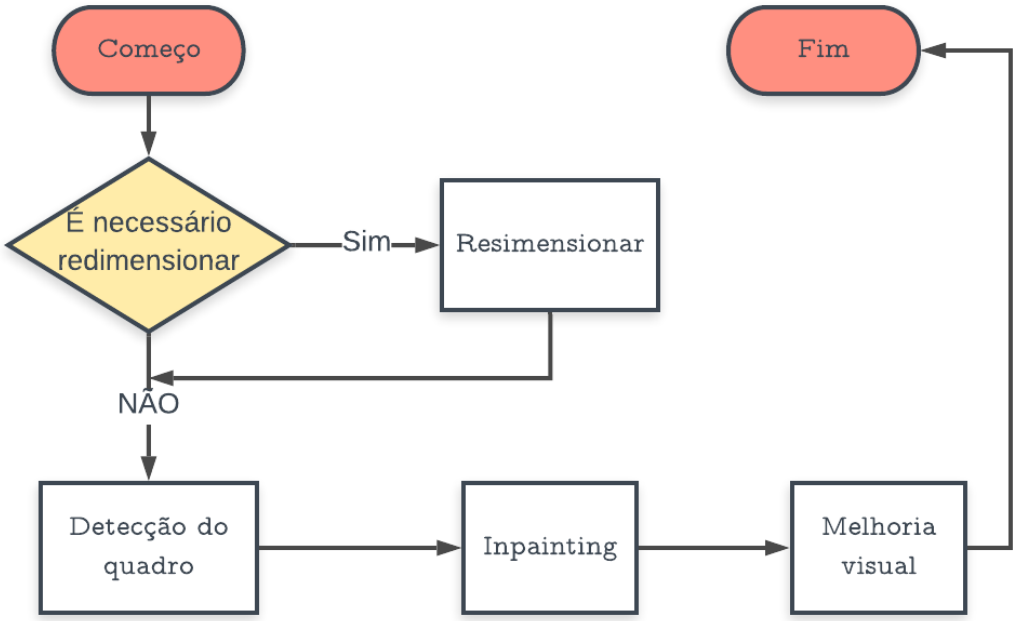


Figura 4.2: Arquitetura principal do sistema.

A Figura 4.2 mostra como é o fluxo principal de execução do sistema. Quatro módulos são responsáveis por produzir o resultado final (imagem do quadro branco processada), eles não são exclusivos nem possuem interdependência a partir do momento que avança do primeiro passo de redimensionameno, podendo ser invocados individualmente pela interface do aplicativo, deixando o usuário livre para escolher o que deseja fazer com a sua imagem original conforme as suas necessidades.

4.1.1 Redimensionamento

Este módulo possui uma grande importância para o funcionamento como um todo do sistema, partindo do princípio que ele será implementado em um *smartphone*, existem algumas limitações relacionadas ao *hardware* que devem ser consideradas. O poder de processamento de um celular tende a ser relativamente inferior ao de um computador de mesma geração, tendo isso em mente, embora não seja o foco nem foram feitos testes de performance, é desejável que este sistema não demande muito tempo de processamento entre as operações fornecidas no aplicativo.

Muitos algoritmos de compressão de imagens com perdas [21] tiram vantagem da redundância existente entre os pixeis para produzir imagens com menor tamanho, ne-

cessitando de menos espaço na memória para armazená-las. Com base nesse conceito o sistema executa o procedimento de escalonamento (Seção **??**) da imagem para proporcionar maior velocidade de processamento entre as etapas sem prejudicar muito a qualidade da imagem final produzida.

Grande parte dos algoritmos envolvidos nos processos do sistema envolvem o passo de percorrer todos os pixeis da imagem um por um, para realizar operações ou inferir conclusões necessárias às outras etapas. A partir do momento que se tem uma imagem com uma dimensão menor, menos pixeis são necessários visitar a cada iteração, reduzindo proporcionalmente o tempo de execução dos processos. A desvantagem introduzida por essa técnica é que a imagem agora possui menos informação nela contida, dependendo das proporções que o escalonamento é aplicado pode acabar deixando o conteúdo semântico contido na imagem ilegível, ainda mais que o foco desse sistema é processar imagens de quadros brancos, a legibilidade da imagem final é de extrema importância.

Para solucionar esse critério, foi estabelecido um parâmetro de redimensionamento que tanto produzia imagens finais suficientemente legíveis e não demandava muito tempo de execução dos algoritmos que iteravam sobre cada pixel da imagem (no máximo 2 minutos). Foi estabelecido então que o tamanho máximo da maior dimensão (altura ou largura) da imagem deveria ser de 1280 pixeis. Se a sua maior dimensão fosse maior que 1280 pixeis, o sistema deveria escalonar a imagem para que a maior dimensão tivesse 1280 pixeis, e a outra dimensão, o tamanho proporcional ao novo tamanho.

Algoritmo 3 Redimensionar se necessário (imagem)

1: $maxDimensao \leftarrow max(imagem.width, imagem.height)$
2: **if** $maxDimensao > 1280$ **then**
3: reduzirTamanho(imagem)
return *imagem*

4.2 Arquitetura da detecção do quadro branco

A Figura 4.3 mostra o fluxo executado pelo sistema para detectar o quadro branco dentro da imagem e recortá-lo.

4.2.1 Aplicar filtro Gaussiano

Como visto na Seção 2.4.1, o processo de aplicação do filtro Gaussiano tem o objetivo de suavização e remoção de ruídos de imagens. Para esse sistema, esse passo é necessário justamente para remover possíveis ruídos da imagem que possam interferir nos passos posteriores, visto que algumas imagens do banco de imagens utilizado para teste possuiam

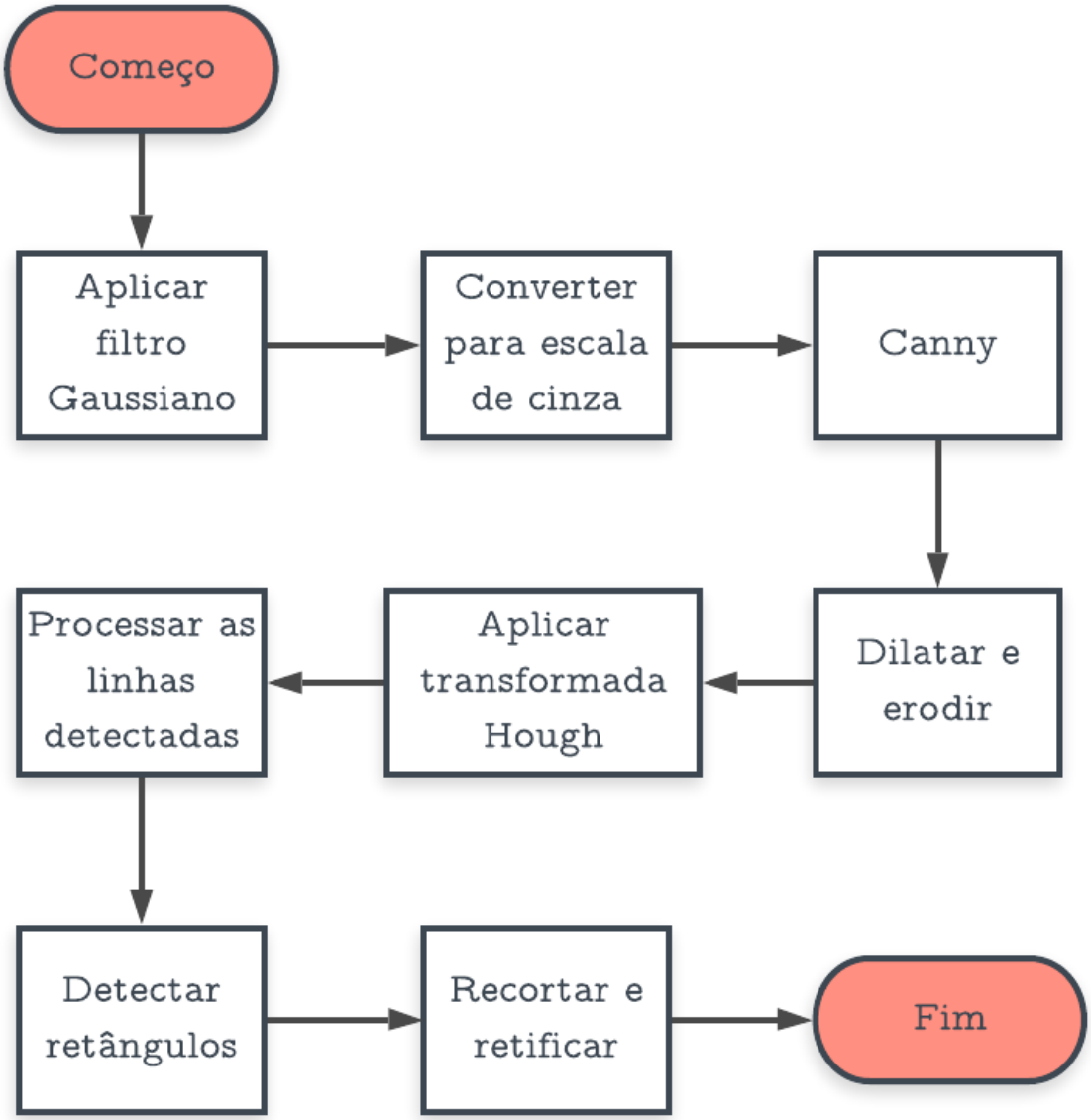


Figura 4.3: Arquitetura da detecção do quadro.

algum tipo de granulação decorrente da forma com que foram capturadas ou comprimidas. Esses ruídos acabavam por prejudicar um processo posterior de detecção de bordas, pois em alguns casos o sistema acabava detectando bordas falso-positivas.

O filtro Gaussiano tem a característica de não conservação de arestas, o que pode ser um problema dependendo da sua intensidade aplicado na imagem, uma vez que um dos objetivos posteriores nesse sistema é de detectar as bordas do quadro, ou seja, as arestas que delimitam os objetos que pertencem ao quadro e os que não. Para isso, um *kernel* de tamanho 3x3 pixel foi utilizado para filtrar todas as imagems que passam pelo sistema,

um *kernel* desse tamanho não afeta a definição das bordas da imagem, nem deixa de remover as granulações das imagens do banco de dados de exemplo.

Uma cuidado a mais necessário a ser tomado, é que quando se realiza a operação de redimensionamento de uma imagem para reduzir seu tamanho, o que acontece na prática é o descarte de amostras na imagem, o que pode provocar o efeito de *aliasing*. Uma vez que o sistema somente exeuta operações de redimensionamento para diminuir o tamanho da imagem, o filtro Gaussiano serve como *anti-alliasing* aplicado imediatemente após a redução.

4.2.2 Converter para escala de cinza

A conversão para escala de cinza é um processo fundamental na etapa de detecção do quadro branco na imagem, uma vez que as cores não são um elemento fundamental neste momento, apenas os contornos da imagem, e por consequência, do quadro branco são os principais objetos a serem detectados. Como visto na Seção 2.2 esse processo faz com que a imagem tenha apenas um canal armazenando a intensidade luminosa em cada pixel da imagem.

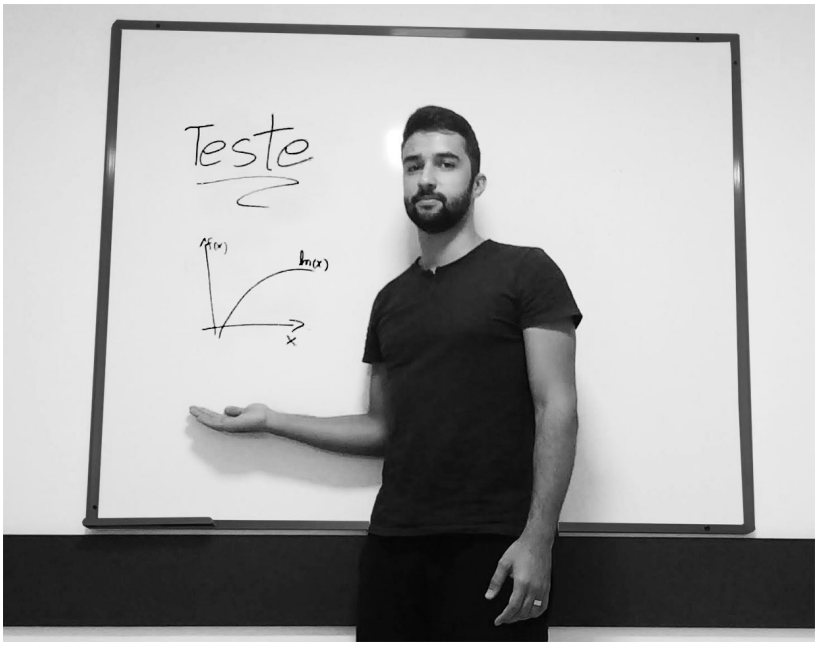


Figura 4.4: Imagem em escala de cinza.

A Figura 4.4 mostra a imagem original em escala de cinza.

4.2.3 Aplicar o detector de bordas Canny

O algoritmo de detecção de bordas Canny é uma das partes mais importantes desse módulo de detecção do quadro branco. Ele é o responsável por detectar e evidenciar todas as bordas da imagem original. A operação de Canny já transforma a imagem original em binária, o que irá facilitar as etapas seguintes do fluxo. As bordas detectadas por Canny são representadas por objetos na imagem binária (possuem valor 1) de espessura de 1 pixel, e o que não é considerado borda é representado pelo valor 0.

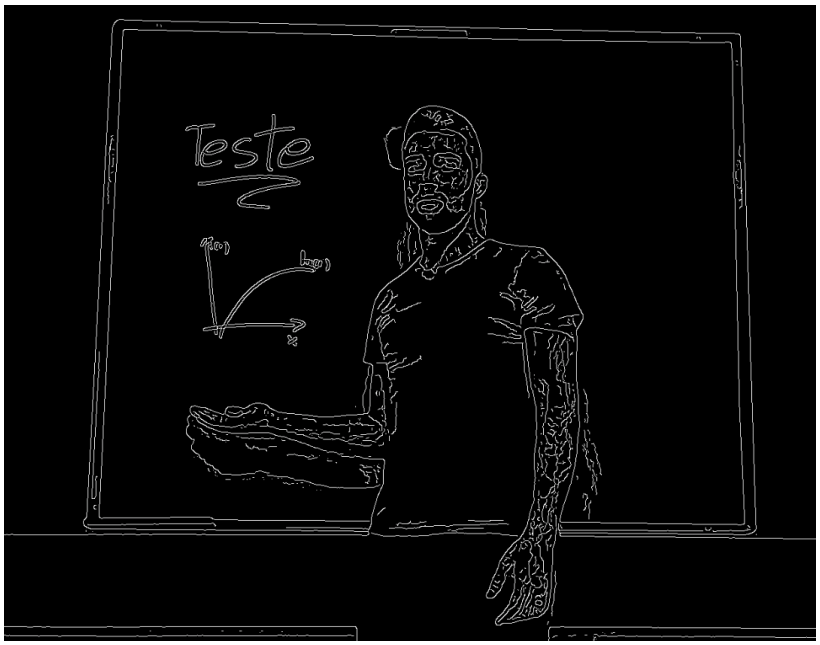


Figura 4.5: Imagem após aplicar o detector de bordas Canny.

A Figura 4.5 mostra como ficam as bordas da imagem ao utilizar o detector de bordas canny, evidenciando em objetos de uma imagem binária o que seriam todas as bordas da imagem original.

4.2.4 Dilatar

Esse passo é responsável por evidenciar o que seriam as possíveis bordas do quadro branco. A operação morfológica de dilatação é necessária para evidenciar os objetos de borda detectados na imagem gerada pelo algoritmo de Canny, uma vez que possuem uma espessura de 1 pixel. Um elemento estruturante de tamanho 10x10 é utilizado na dilatação, dessa forma a espessura das bordas detectadas pelo algoritmo de Canny aumentam em 10x de tamanho. Um efeito colateral benéfico que também é produzido pela dilatação é fechar as pequenas interrupções que podem existir entre as bordas detectadas, deixando-as perfeitamente contínuas. Isso garante um melhor funcionamento da próxima etapa.

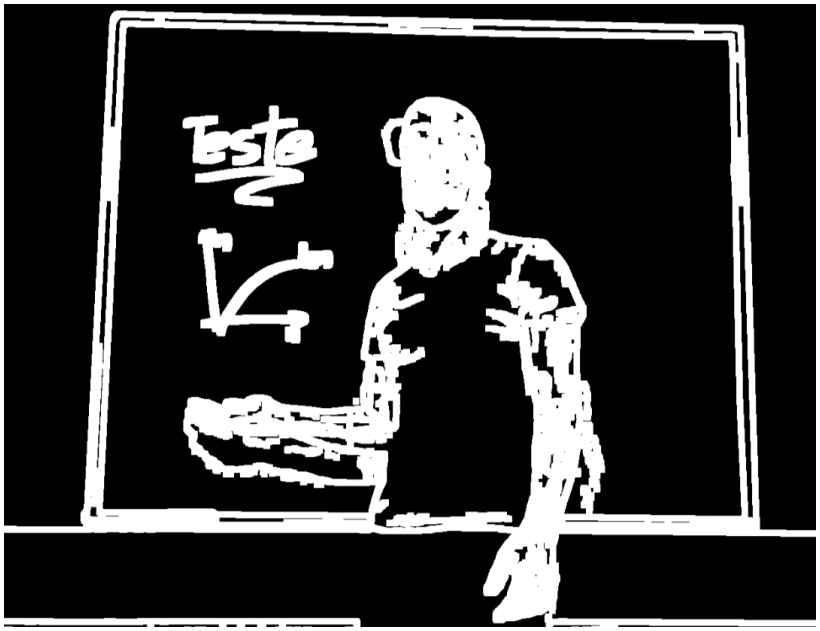


Figura 4.6: Imagem após dilatar as bordas.

A Figura 4.6 mostra a imagem após dilatar as bordas com o elemento estruturante de dimensões 10x10.

4.2.5 Aplicar transformada Hough

Como visto no Capítulo 2.6.2, a transformada Hough muda a imagem original do seu domínio para o domínio Hough com o principal objetivo de encontrar os pontos na imagem original que pertencem à mesma reta. No domínio Hough, os pontos que pertencem à mesma reta na imagem original produzem sinusóides que se cruzam em um mesmo ponto. A partir dessas sinusóides coincidentes são extraídas as coordenadas de cada ponto que pertencem a uma mesma reta na imagem original. Esse passo permite identificar todos os segmentos de linhas retas da imagem original. Dois pontos são utilizados para representar um segmento de reta, um ponto de origem (x1, y1) e um ponto de fim (x2, y2). Ao final desse passo teremos uma lista de pares de pontos que representam todas os segmentos de reta detectados na imagem original.

A Figura 4.7 mostra todos os segmentos de reta (em azul) detectados na imagem original ao aplicar a transformada Hough.

4.2.6 Processar linhas detectadas e detectar retângulos

Como no passo anterior teremos todos os segmentos de reta detectados na imagem original, certamente segmentos não farão parte dos quatro que pertencem às bordas do quadro

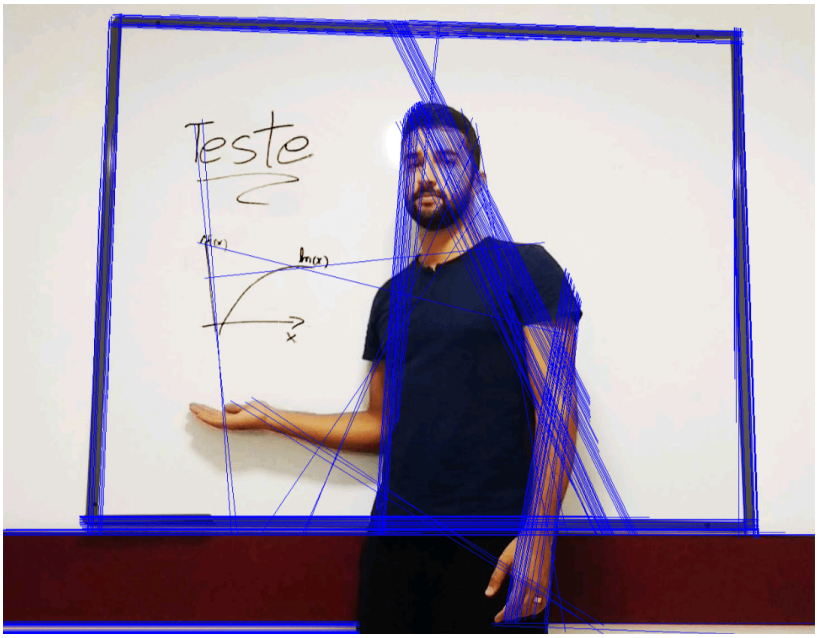


Figura 4.7: Segmentos de reta detectados na imagem.

branco. Esse passo processa a lista de segmentos de reta detectados para agrupá-los em clusters com algumas características em comum, todas respeitando as seguintes regras:

- Linhas opostas devem ter a mesma orientação (com até 30º de tolerância);
- Linhas opostas devem estar a uma distância mínima umas das outras (1/5 da maior dimensão da imagem original no nosso caso);
- O ângulo entre 2 linhas vizinhas ou concorrentes deveria ser de 90º (com até 30º de tolerância);
- A orientação das linhas deveria ser consistente (só formar grupos com pares de linhas opostas de mesma orientação).

Ao final desse passo teremos uma lista de grupos de segmentos de retas, em que todos os grupos respeitam às regras acima.

Em alguns casos os grupos formados podem conter segmentos de reta que não necessariamente compreendem os contornos do quadro branco. O objetivo neste momento é verificar se os grupos de segmentos de reta realmente formam retângulos. A forma mais simples de verificar isso é pela quantidade de segmentos de retas contidos nos grupos. Todos os grupos que possuem um número diferente do que 4 segmentos são descartados. Então só restam os retângulos candidatos à borda do quadro branco. Então o que possui a maior área é escolhido como representante das bordas do quadro branco na imagem.

Essa metodologia assume que a foto foi tirada com o quadro branco o cupando a maior área possível.

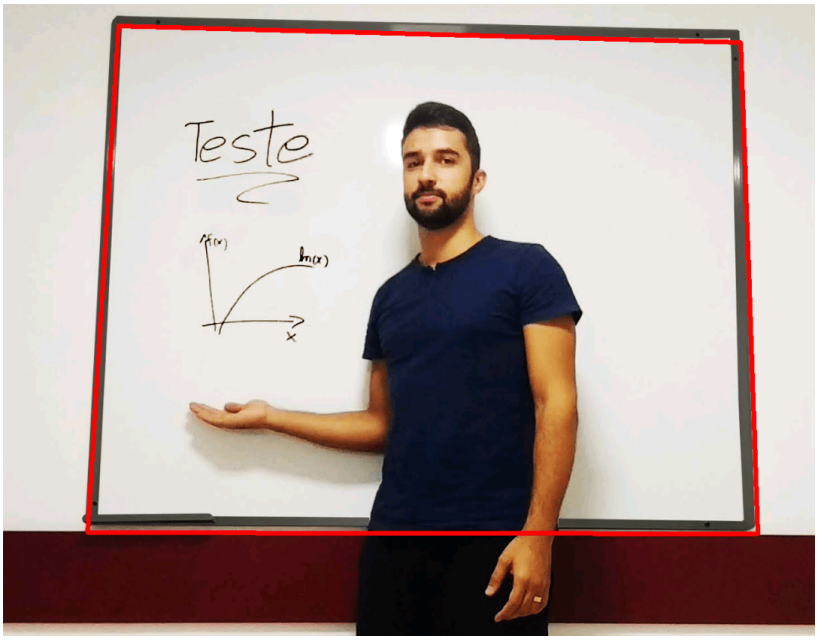


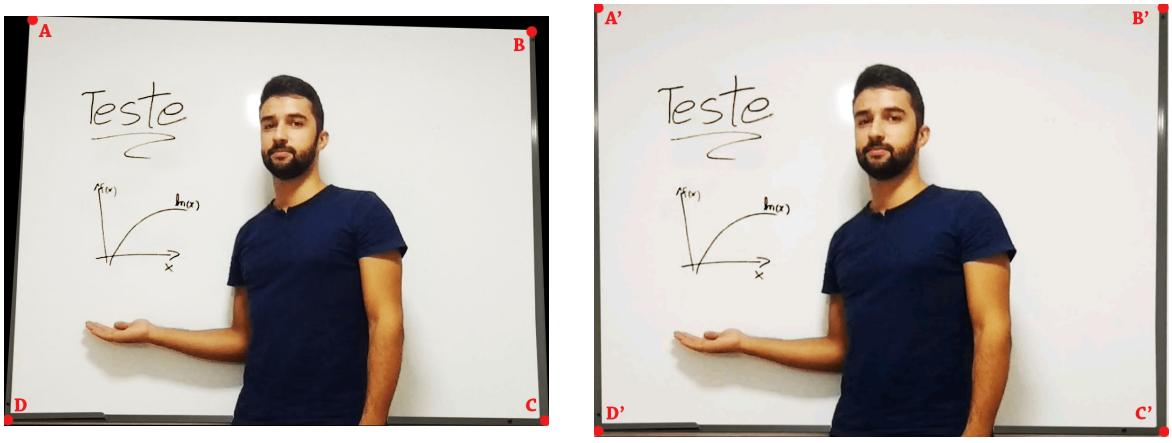
Figura 4.8: Segmentos de reta eleitos como contornos do quadro branco.

A Figura 4.8 mostra os 4 segmentos de reta (em vermelho) escolhidos dentre os detectados na Figura 4.7 para representar os contornos do quadro branco de acordo com os critérios estabelecidos conforme as regras.

4.2.7 Recortar e retificar

Uma vez detectada a região que foque compreende as bordas do quadro, tudo que está dentro dela então faz parte do conteúdo do quadro. O principal objetivo desse módulo é detectar o quadro branco na imagem original. Nesse ponto já temos a posição relativa à imagem original ocupada pelo conteúdo do quadro. Podemos com essas posições então recortar a imagem original, extraindo somente o conteúdo do quadro branco. Mas como sabemos, a imagem pode ter sido tirada em ângulo em relação ao quadro, causando uma distorção de perspectiva. Para solucionar esse problema, neste momento é executada a retificação planar da imagem do conteúdo do quadro, em que a imagem é projetada para que ocupe a região do retângulo circunscrito nela.

A Figura 4.9 mostra como a retificação acontece no sistema. A Figura 4.9a mostra a imagem delimitada pelos segmentos de reta demarcando os limites do quadro branco. Para efeito visual, o retângulo nela circunscrito foi preenchido com a cor preta. O objetido da retificação é projetar a imagem de forma que ela preencha esse retângulo, removendo o



(a) Imagem com distorção de perspectiva

(b) Imagem após a retificação e recorte

Figura 4.9: Retificação

efeito da distorção de perspectiva, como pode ser visto na Figura 4.9b retificada, em que todos os ângulos dos cantos da imagem são de 90º. Os pontos A', B', C' e D' mostram na Figura 4.9b retificada o local para onde os pontos A, B, C e D foram respectivamente da Figura 4.9a original.

4.3 Arquitetura do *inpainting*

A Figura 4.10 mostra o fluxo de execução no sistema para fazer a mascaragem na imagem de objetos em primeiro plano em relação ao quadro branco.

4.3.1 Segmentação (K-means)

Como nesse passo deseja-se mascarar somente alguns objetos na imagem, é necessário determinar nela quais regiões deverão ser mascaradas e quais não. Primeiramente devemos encontrar a máscara que compreenda os objetos indesejáveis na imagem. A máscara é uma imagem binária em que os objetos representam na imagem original o que se deseja mascarar. Como dispomos somente de uma imagem do quadro, não é possível utilizar técnicas *inter-frame* [18] para fazer a segmetação. Uma solução encontrada para esse problema, foi então utilizar o método de clusterização K-means.

Observando o banco de dados de imagens utilizadas como exemplo para este trabalho, percebemos que a maioria das imagens que tinham objetos em primeiro plano em geral eram partes do corpo de pessoas como braço, tronco, etc. E na maioria dos casos, o quadro branco ao fundo apresentava uma cor homogênea e uniforme em relação ao resto da imagem, incluindo os traços nele escritos. A partir dessa informação, o algoritmo K-means serviu para segmentar a imagem utilizando o parâmetro K=2 com o objetivo de classifi-

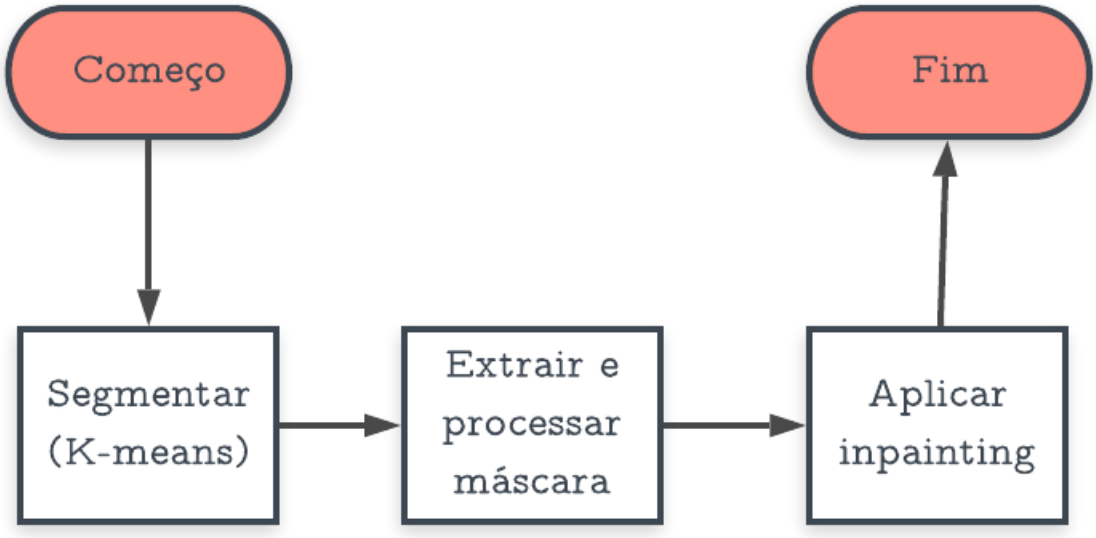


Figura 4.10: Arquitetura do *inpainting*.

car as cores da imagem em 2 grupos: `objetos que fazem parte do quadro branco` e `objetos que não fazem`. O algoritmo implementado do K-means utiliza sementes iniciais randômicas, pois as imagens de entrada do sistema não possuem um padrão de cor definido ou previsível.

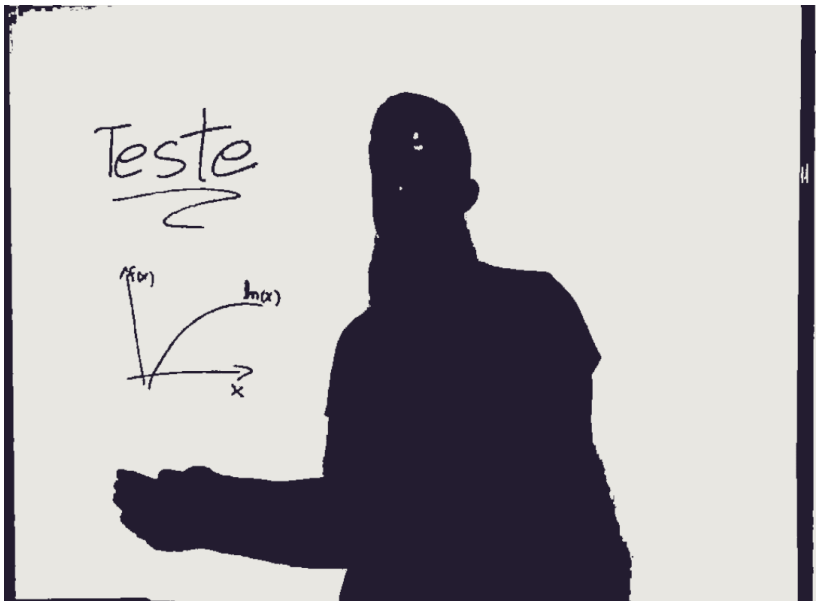


Figura 4.11: Imagem clusterizada em 2 grupos.

A Figura 4.11 mostra a imagem final dessa etapa em 2 cores. Os objetos de uma cor

são os pixeis na imagem original que têm uma cor suficientemente parecida com o quadro branco em si, e a outra agrupa os demais pixeis da imagem.

4.3.2 Extrair e processar a máscara

Como a segmentação é feita por meio da cor do elementos da image, espera-se que neste momento teremos todos os objetos que teoricamente não fazem parte do quadro branco segmentados, isso inclui traços escritos no quadro branco e partes de pessoas e objetos. Como o único objetivo desse módulo é mascarar somente objetos em primeiro plano, é necessário remover da máscara os objetos que fazem parte dos traços escritos no quadro. Normalmente os traços escritos no quadro branco estão em uma cor diferente do restante do quadro, e possuem traços mais finos em relação aos demais objetos segmentados. Para resolver esse problema, algumas operações morfológicas são aplicadas sobre a máscara.

Algoritmo 4 Processar máscara (mascara)

1: $maiorDim \leftarrow max(mascara.width, mascara.height)$
2: $kernel \leftarrow ElementoEstruturante(maiorDim/50)$
3: $erodir(mascara, kernel)$
4: $dilatar(mascara, kernel)$
5: $floodFill(mascara)$
6: $dilatar(mascara, ElementoEstruturante(5))$ **return** $mascara$

O algoritmo 4 mostra o processo utilizado para processar a máscara após a segmentação. A máscara é erodida com um elemento estruturante quadrado de dimensão 1/50 da maior dimensão da máscara para remover dela os traços escritos no quadro. Logo em seguida a máscara é dilatada com o mesmo elemento estruturante para restaurar os demais objetos ao seu tamanho original. O algoritmo de *flood fill* é aplicado para preencher os possíveis buracos existentes dentro da máscara, uma vez que as operações morfológicas anteriormente aplicadas não fazem isso. Por fim um elemento estruturante circular de dimensão 5 é utilizado para dilatar a máscara, para servir como borda de segurança e cobrir uma pequena região ao redor da máscara para envolver completamente os objetos no processo de *inpainting*.

A Figura 4.12 mostra como a máscara fica ao final do processamento. Os traços escritos no quadro branco e os buracos nos objetos como apareciam na Figura 4.11 já não estão mais na máscara após o processamento.

4.3.3 *Inpainting*

Esse processo é o *inpainting* em si. Nesse ponto já teremos a máscara que compreende toda a região que não deve aparecer na imagem final. O processo do *inpainting* consiste

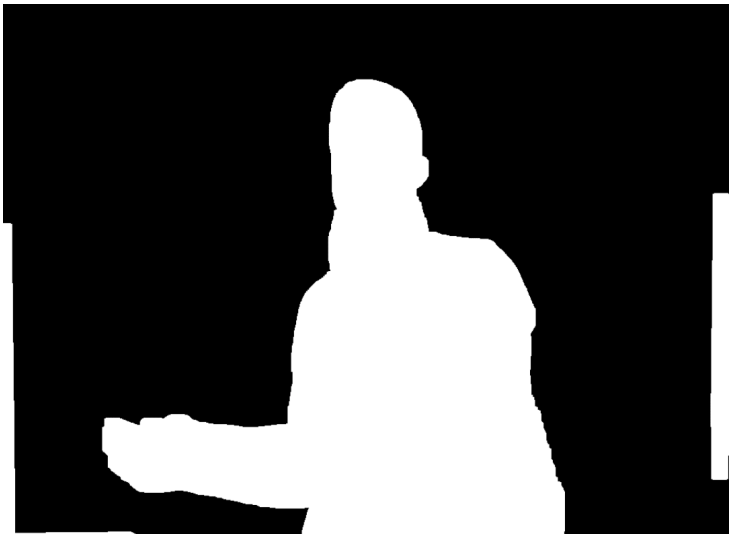


Figura 4.12: Máscara pós processada.

em pegar pixeis de regiões externas e próximas à máscara na imagem original, e copiá-los para a região dentro da máscara. Como o objetivo é mascarar partes do corpo de pessoas na frente do quadro, na prática o que acontece com a imagem após o inpainting é o preenchimento da máscara com partes do quadro branco, ocultando os objetos indesejados, deixando a imagem somente com os objetos de interesse do conteúdo do quadro..

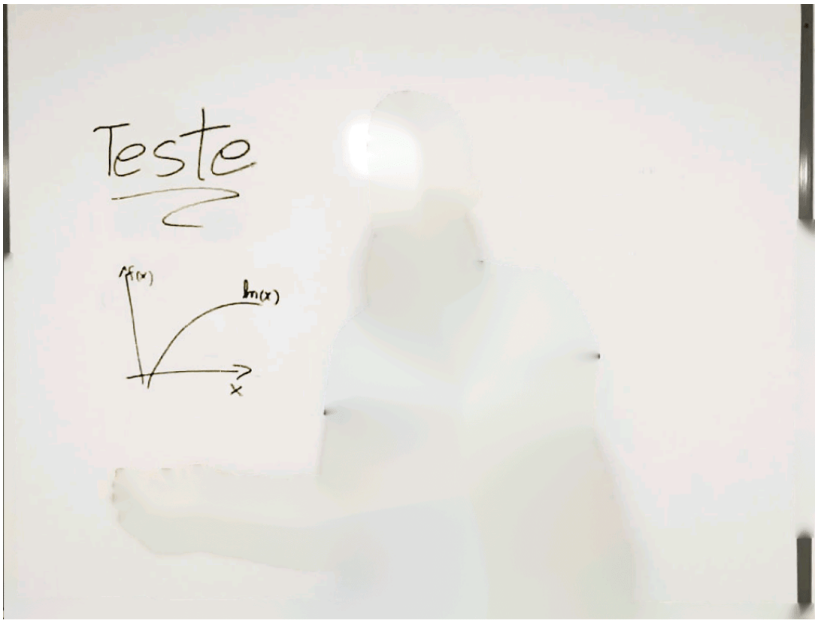


Figura 4.13: Imagem após a aplicação do *inpainting*.

(a) Imagem antes da melhoria visual

(b) Imagem após a melhoria visual

Figura 4.14: Antes e depois de aplicar a melhoria visual na imagem

4.4 Melhoria visual

A melhoria visual é um processamento que pode ser aplicado em qualquer momento a partir de quando a imagem é carregada no aplicativo. Ele consiste em melhorar visualmente a qualidade da imagem final do quadro. Apesar de tratarmos de quadros brancos neste trabalho, uma foto tirada de um deles não apresenta um padrão 100% homogêneo de cores. Decorrente da forma com que a imagem foi capturada, efeitos de luz e sombra podem introduzir pequenos gradientes de tonalidade de cinza ao longo da superfície do quadro branco na imagem.

Algoritmo 5 Melhoria visual (pixel)

```
1: lim ← 165
2: max ← 255
3: if pixel.r >= lim and pixel.g >= lim and pixel.b >= lim then
4:     pixel.r = max
5:     pixel.g = max
6:     pixel.b = max
```

O algoritmo 5 descreve como esse passo foi implementado no sistema. Ao percorrer toda a imagem cada pixel é analisado, se todos os valores dos seus canais RGB forem maior do que o limiar (definido empiricamente como 165), então o valor de todos os canais desse pixel é estourado para branco, ou seja, definido 255. Isso produz na imagem final um efeito homogêneo, pois somente os pixeis que compõem o quadro branco ficam branco, enquanto os demais, que compõem os traços escritos permanecem na cor que estão. Essa parte do sistema limita as imagens para quadros brancos somente. A foto de um quadro negro iria falhar nesse módulo de melhoria visual.

A Figura 4.14 mostra o que seria o resultado final bem sucedido de uma imagem no sistema. Percebe-se também que alguns resíduos da borda do quadro branco também permanecem na imagem final. Isso ocorre pois durante a detecção essas bordas foram incluídas dentro do que foi considerado conteúdo do quadro branco e durante as etapas de processamento do *inpainting*, elas foram removidas da máscara.

Neste capítulo veremos detalhes sobre o desenvolvimento do aplicativo.

Capítulo 5

Aplicativo

Neste capítulo veremos detalhes sobre a implementação do aplicativo em si. Quais tecnologias e ferramentas foram escolhidas para o desenvolvimento e os motivos para serem selecionadas neste trabalho.

5.1 Plataforma (sistema operacional)

A ideia central desse sistema é que pudesse ser implementado diretamente em um dispositivo móvel, agilizando e melhorando a forma com que os estudantes tomam notas em um ambiente de sala de aula. Para isso, era necessário então desenvolver um aplicativo móvel que atendesse os requisitos. Os dois maiores sistemas operacionais da atualidade foram elencados como opção: Android e IOS. Os demais sistemas operacionais móveis disponíveis foram descartados devido à sua baixa penetrabilidade no mercado, representando menos de 2% do total de *smartphones* [22]. O sistema operacional Android foi escolhido pelas suas diversas vantagens em relação ao sistema competidor IOS da Apple, dentre elas:

- Custo zero para desenvolvimento nativo
  - Como a plataforma Android é *opensource*, desenvolver uma aplicação para ela não há nenhum custo envolvido.
  - A plataforma concorrente exige um software de desenvolvimento que custa US$4,99 sem agregar um valor significativo ao desenvolvimento.
  - Para desenvolver um aplicativo Android é necessário apenas um computador com qualquer sistema operacional.
  - Desenvolver e testar um aplicativo IOS exige pelo menos um computador com o sistema operacional Mac.

- Maturidade da linguagem de programação utilizada na plataforma
  - Um aplicativo móvel para Android pode ser desenvolvido em Java, uma linguagem de programação estável, com uma comunidade bem consolidada para suporte, e muitas bibliotecas auxiliares já implementadas na linguagem.
  - Aplicativos IOS são desenvolvidos em *swift*, uma linguagem relativamente recente (5 anos), que não tem as vantagens de uma linguagem madura como java.

Outro fator importante na tomada de decisão para escolher o sistema operacional foi a qualidade da documentação. A documentação do Android [23] fornece uma variedade de referências à sua Application Programming Interface (API) de recursos, dicas de design e boas práticas, bem como e utilitários do Android compatíveis com uma ampla variedade de versões de plataforma cobrindo até mesmo os dispositivos vestíveis e televisores.

5.2 Ambiente de desenvolvimento

A Integrated Development Environment (IDE) para o Android utilizado nesse trabalho Android Studio (AS) foi escolhido pois apresenta uma série de funcionalidades que facilitam o desenvolvimento. O AS possui um editor de código que aumenta bastante a produtividade no desenvolvimento de uma aplicação, ele faz o preenchimento e geração de trechos do código automaticamente para as linguagens Kotlin, Java e C / C ++, faz sugestões de melhorias, organiza estruturalmente, dentre outros recursos. Ele também vem equipado com um editor visual de layout que permite criar telas e interfaces gráficas com o usuário de uma forma tão simples quanto arrastar componentes na posição desejada na tela e atribuir funções que serão executadas ao detectar interações com o usuário. O AS também possui um emulador de Android que permite instalar e executar aplicativos com mais rapidez do que um dispositivo físico e também simular diferentes configurações para testes.

5.3 OpenCV

O Open Source Computer Vision Library (OpenCV) é uma biblioteca multiplataforma, totalmente livre ao uso acadêmico e comercial, para o desenvolvimento de aplicações na área de Visão computacional. O OpenCV possui módulos de processamento de imagens e vídeo, estrutura de dados, álgebra Linear, interface gráfica do usuário básica com sistema de janelas independentes, controle de mouse e teclado, além de mais de 350 algoritmos de visão computacional como: filtros de imagem, calibração de câmera, reconhecimento de

objetos, análise estrutural e outros. O seu processamento é em tempo real de imagens. Esta biblioteca foi desenvolvida nas linguagens de programação C/C++. Também, dá suporte a programadores que utilizem Java, Python e Visual Basic e desejam incorporar a biblioteca a seus aplicativos. A versão 1.0 foi lançada no final de 2006.

Por esses motivos, o OpenCV pareceu ser adequado para atender às necessidades ferramentais do projeto, e portanto, foi escolhido como a biblioteca central de processamento de visão computacional neste trabalho.

5.4 Porte do OpenCV para Android

Uma abordagem para solucionar o problema proposto neste trabalho seria utilizar o celular apenas como método de entrada para as imagens, explorando apenas seus recursos de câmera. A partir dos dados obtidos, enviá-los a um servidor, e então executar o processamento na imagem lá, para retornar somente a imagem final para o celular. Porém, um dos objetivos desse trabalho é que ele seja executado em um dispositivo móvel independente da sua conectividade com a internet, ou seja, de forma *offline*, todo o processamento deveria ser executado diretamente no *smartphone*.

Para resolver esse problema, foi utilizado o porte da biblioteca OpenCV para Android. Esse porte encapsula toda a implementação em C++ do OpenCV em classes Java com métodos nativos, ou seja, as classes Java servem como uma interface de comunicação para fazer as chamadas aos métodos implementados em C++ [1] , preservando a implementação original testada e validada do OpenCV, sem introduzir falhas devido à mudança de linguagem.

5.5 Aparelho utilizado para realizar os testes do sistema

Para testar o aplicativo, foi utilizado o *smartphone* Moto Z2 Play da Motorola, mostrado na figura Figura 5.1, utilizando a versão do Android 7.1.1 Nougat. Ele possui uma tela de 5.5 polegadas com uma de 1920x1080 pixels, seu sensor principal de câmera possui 12 megapixels, com tamanho de 1/2.55 ; e abertura de F 1.7 que permite tirar fotos com uma resolução de até 4032x3024 pixels. Seu processador é o Cortex-A53, com o chipset Snapdragon 626 Qualcomm MSM8953 Pro, GPU Adreno 506, e 4 GB de memória RAM.

[1] Veja mais detalhes em https://opencv.org/android/

Figura 5.1: *Smartphone* utilizado para testes do sistema.

5.6 Interface gráfica do aplicativo

A Figura 5.2 mostra um *printscreen* da tela principal do aplicativo. Nela é possível interagir com todas as funcionalidades dele. Ela não segue nenhum padrão de projeto ou design específico, apenas utiliza os componentes de botão e visualização de imagem com zoom do próprio Android. O botão `GALLERY` é responsável por invocar a galeria do sistema operacional para que usuário selecione a foto que deseja processar no sistema. O botão `CONTOUR` aplica o módulo de **detecção do quadro branco** na imagem carregada, e exibe a imagem atualizada. O botão `INPAINT` aplica o módulo de ***inpainting*** na imagem, e exibe o resultado. O botão `ENHANCE` quando acionado aplica o módulo de **melhoria visual** na imagem exibida, e mostra o resultado. Antes de aplicar qualquer módulo sobre a imagem, é executado em segundo plano o módulo de redimensionamento, para reduzir o tamanho da imagem se for necessário. Só é possível aplicar algum módulo se houver uma imagem carregada. Os módulos podem ser aplicados fora de ordem, de acordo com o que o usuário seleciona.

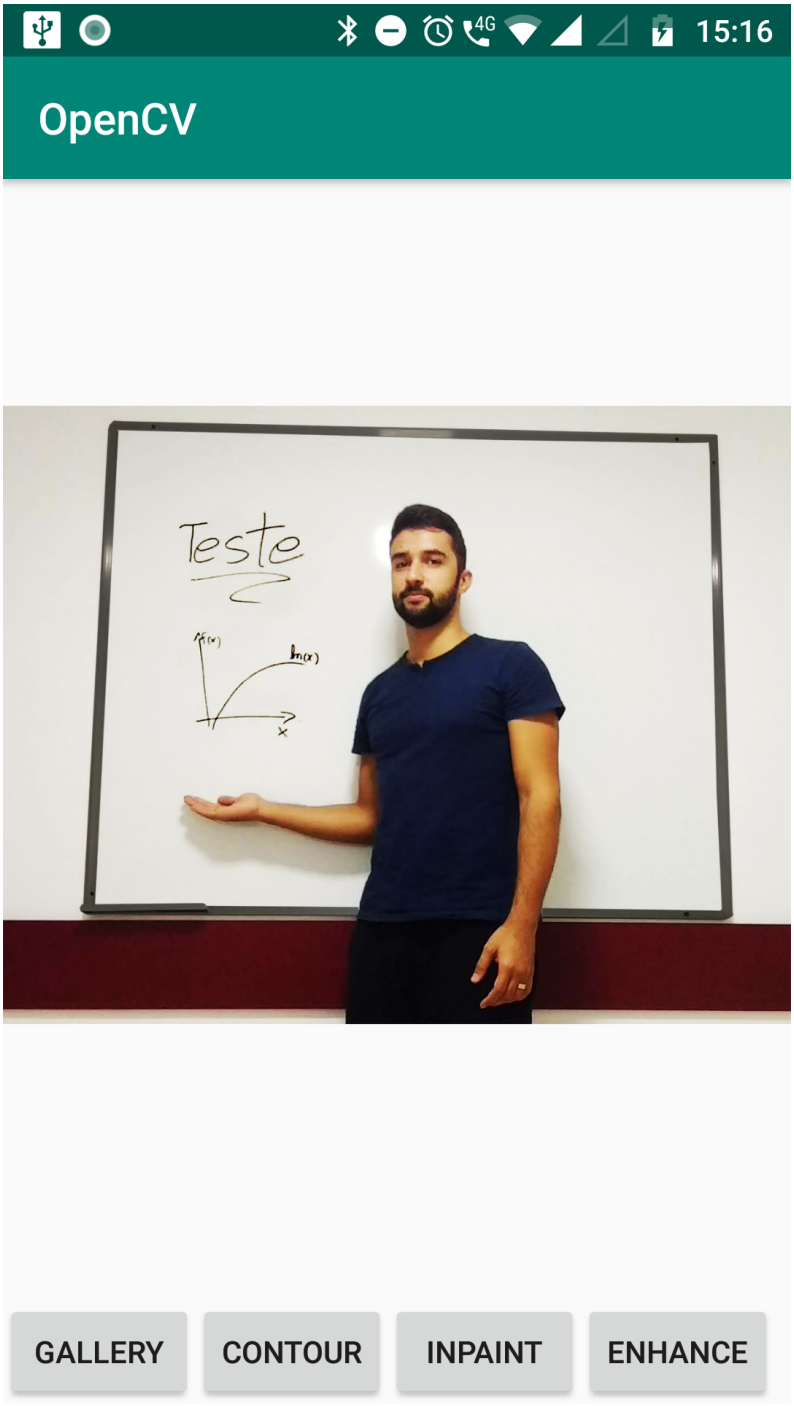


Figura 5.2: *Printscreen* da tela principal do aplicativo.

No capítulo seguinte será abordada a metodologia de como os experimentos foram conduzidos para validar a proposta deste trabalho.

Capítulo 6

Experimentos

Neste capítulo serão apresentadas as opções e explicados os motivos de escolha das ferramentas e tecnologias utilizados para o desenvolvimento do sistema proposto neste trabalho, bem como os experimentos utilizados para validar a sua proposta.

6.1 Detalhes do banco de dados

O banco de dados utilizado para desenvolver e testar os módulos deste sistema consiste em uma sequência de mais de 100 imagens de quadros brancos capturadas em ambientes escolares reais, simulados e também de imagens da internet sem direitos autorais. O banco de imagens contempla os mais diversos cenários em que o sistema poderia ser utilizado na prática para melhor validar sua proposta. Podendo validar individualmente cada módulo, e o sistema como um todo, aplicando sequencialmente cada módulo na mesma imagem.

A Figura 6.1 mostra algumas amostras de imagem do banco de dados real utilizado para validar a proposta deste trabalho.

6.2 Condução dos experimentos

Os experimentos foram conduzidos com as imagens do banco. Cada uma delas foi carregada no aplicativo instalado no *smartphone* (visto na Seção 5.5). Então os módulos foram testados individualmente e sequencialmente. Os resultados foram avaliados subjetivamente pela tela do *smartphone*. Uma planilha de controle foi utilizada para registrar pelo nome da imagem o seu resultado obtido em cada módulo e no sistema como um todo.

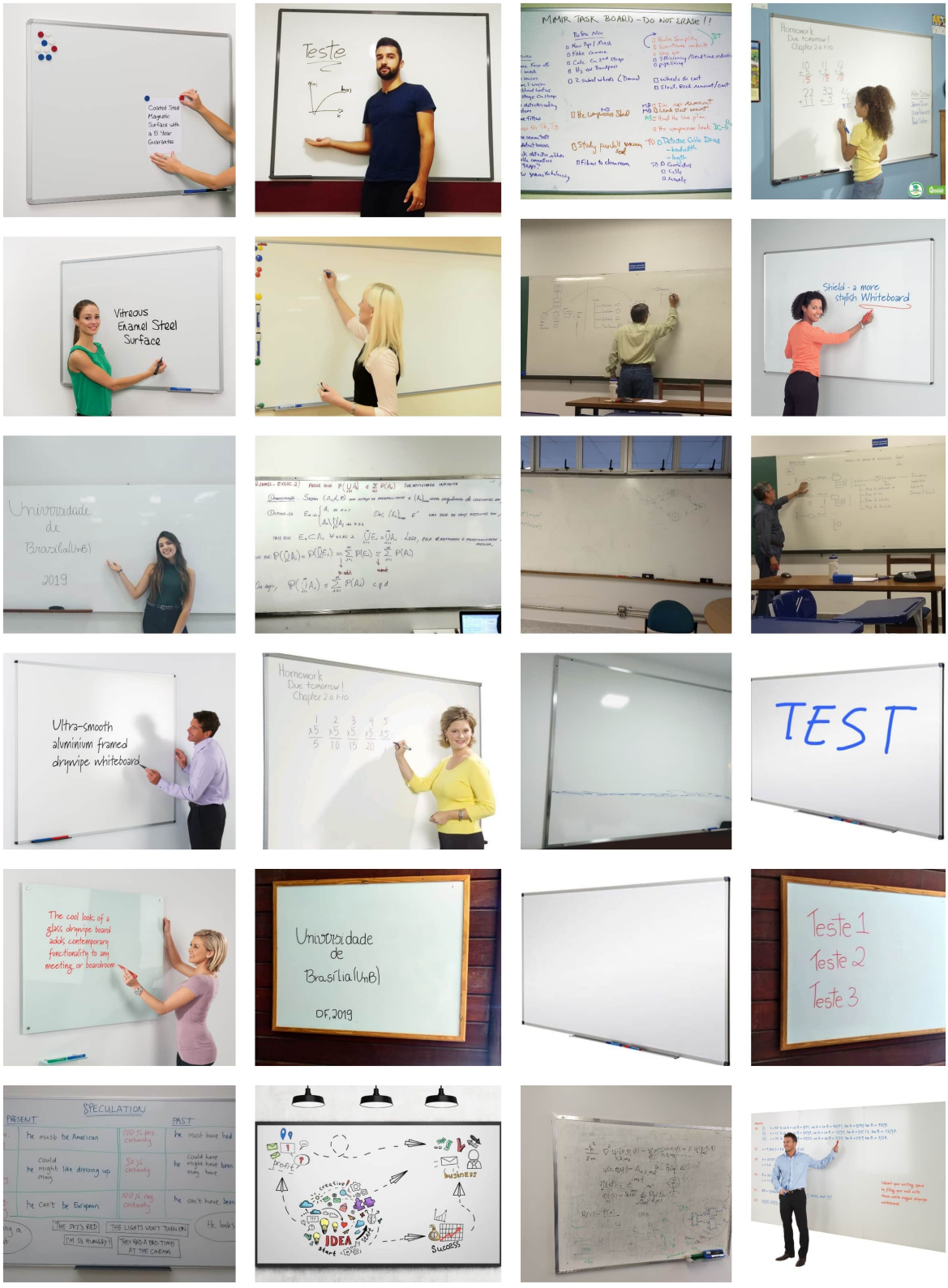


Figura 6.1: Exemplos de imagens do banco.

6.3 Resultados

Nesta seção veremos os resultados do sistema aplicados diretamente sobre imagens do banco de dados. O fator de sucesso é plenamente subjetivo, e o resultado de cada módulo deve ser analisado em cada imagem caso a caso. As causas de sucesso e falha dependem diretamente da imagem e dos fatores externos no momento da sua captura, com o iluminação e cor. Também serão discutidas as razões pelas quais as imagens na maioria tiveram resultado positivo e negativo em cada módulo.

6.3.1 Detecção do quadro branco

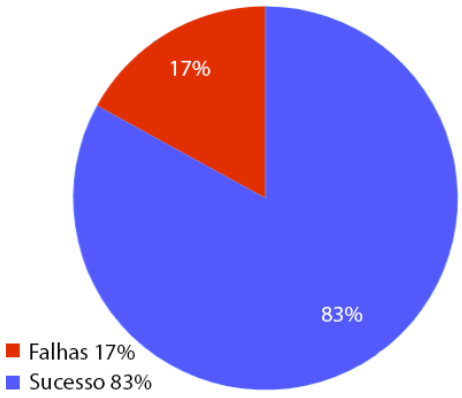


Figura 6.2: Gráfico com a taxa de sucesso do módulo de detecção do quadro branco no sistema.

A Figura 6.2 mostra a taxa de sucesso do módulo de detecção de bordas do quadro branco e retificação. Podemos ver que a maior parte das imagens testadas obteve sucesso ao aplicar o procedimento do módulo do sistema.

A Figura 6.3 mostra exemplos de imagens com resultado bem sucedido no módulo de detecção. As linhas em vermelho nas imagens do meio mostram as retas que foram consideradas bordas do quadro branco nas imagens originais (à esquerda). Nessas imagens há pessoas na frente do quadro, mas ainda assim a detecção é bem sucedida. Isso se deve ao fato de que a transformada Hough vista na Seção 2.6.2 leva em consideração a reta que pertence ao contorno do quadro branco como um todo, portanto o algoritmo consegue inferir que os pontos pertencentes à borda do quadro são colineares, ainda que oclusos por um objeto. À direta vemos as imagens recortadas e retificadas.

O maior responsável pelas falhas deste módulo é a presença de elementos físicos retos ao redor do quadro branco. Eles interferem no processo de detecção pois também produzem segmentos de reta durante a transformada Hough que se encaixam nos critérios de

Figura 6.3: Exemplos de sucesso no módulo de detecção do quadro branco.

definição de retângulo. Em alguns exemplos, objetos físicos próximos ao quadro branco eram considerados como suas bordas pelo algoritmo durante o processamento das linhas, então na imagem final, o conteúdo do quadro branco não era corretamente detectado tampouco recortado.

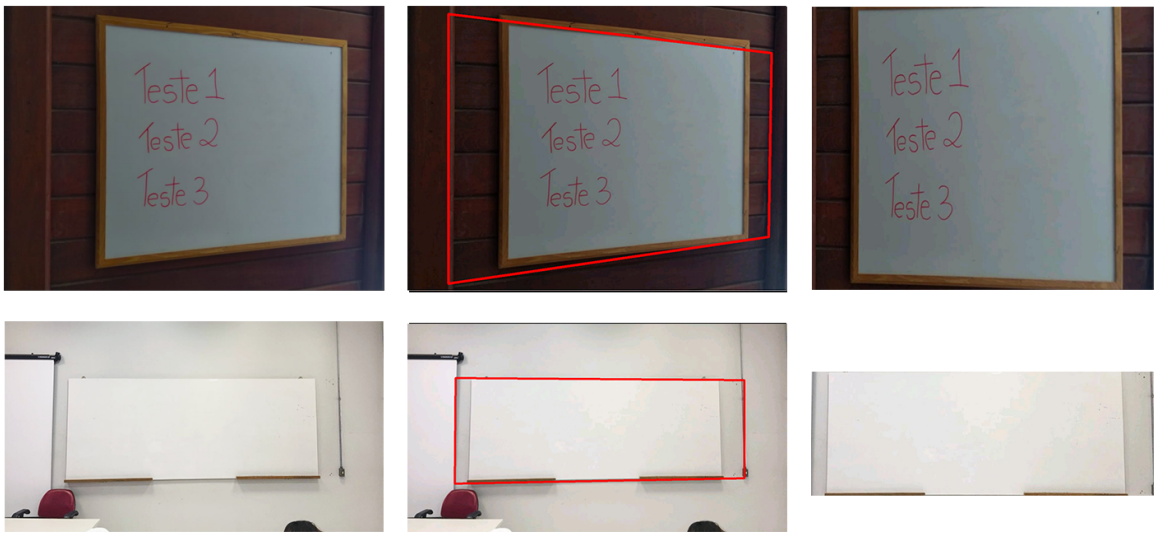


Figura 6.4: Exemplos de falha no módulo de detecção do quadro branco.

A Figura 6.4 mostra alguns exemplos que falharam no módulo de detecção das bordas do quadro branco na imagem. Podemos observar nas imagens originais (à esquerda)

haviam objetos físicos que interferiram na detecção das bordas do quadro branco. A pilastra na parede de madeira foi considerada como borda do quadro branco na primeira imagem, e na segunda foi a combinação da tela do projetor com a extensão de tomada na parede.

6.3.2 *Inpainting*

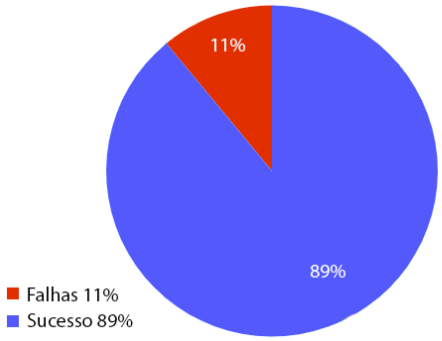


Figura 6.5: Gráfico com a taxa de sucesso do módulo de *inpainting* do quadro branco no sistema.

A Figura 6.5 mostra a taxa de sucesso do módulo de *inpainting* implementado no sistema. Para ser considerado sucesso, a máscara do *inpainting* deve cobrir a maior área possível do objeto em primeiro plano (90%), deixando o mínimo de resíduos ou nenhum. Os casos em que uma parte maior do que 10% do objeto em primeiro não é contemplada é considerado falha. Para imagens em que o professor se encontra na frente dos traços escritos no quadro branco, o critério que foi levado em conta é o mesmo dos 90% de cobertura da máscara, ainda que a imagem seja preenchida posteriormente de forma incorreta, pois com uma imagem somente não há como inferir o que está por de trás do objeto em primeiro plano.

A Figura 6.6 mostra algumas imagens de exemplo do banco que o *inpainting* foi bem sucedido. Da esquerda pra direita temos as imagens originais, a máscara resultado da segmentação e processamento, e a imagem final após o *inpainting*. Podemos ver que e a máscara foi capaz de contemplar totalmente os objetos em primeiro plano, fazendo por consequência que o *inpainting* tivesse um bom resultado. A máscara envolveu uma região maior do que a área dos objetos em primeiro plano na imagem por causa da sombra projetada abaixo dela, mesmo que imperceptível, ela altera a cor dos pixeis em volta do objeto. Isso não é um problema para este módulo pois tudo será preenchido com os pixeis do fundo do quadro branco.

Figura 6.6: Exemplos de sucesso no módulo de inpainting do quadro branco.

Figura 6.7: Exemplo de falha no módulo de inpainting do quadro branco.

A Figura 6.7 mostra um exemplo que o módulo de *inpainting* falhou. Percebemos que a máscara ficou com alguns buracos grandes dentro da área da pessoa em primeiro plano, isso fez com que o sobrasse boa parte da blusa dela na imagem final. Podemos ver também que a máscara contemplou uma grande área à direita que não faz parte da pessoa. Ainda que o critério de qualidade para o *inpainting* não seja esse, isso mostra que neste caso a clusterização não foi capaz de agrupar corretamente os pixeis da imagem pela cor.

Na Figura 6.8 podemos ver um exemplo em que o professor encontra-se em frente às anotações. Esse caso ainda é considerado sucesso para o *inpainting*, pois a máscara

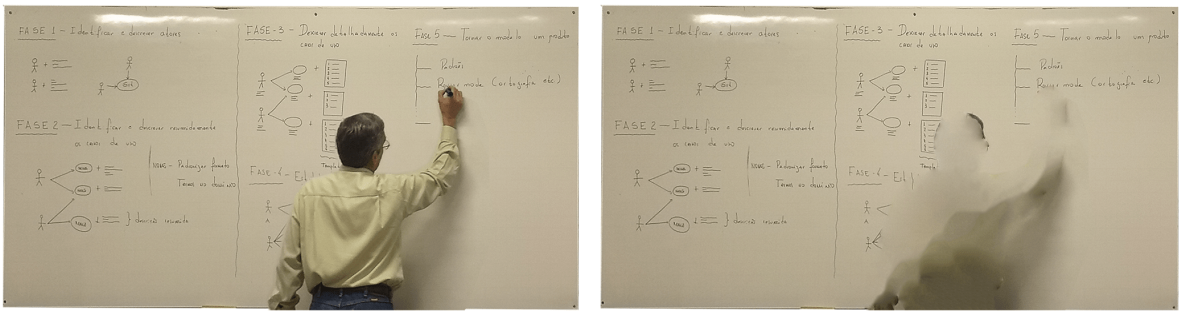


Figura 6.8: Exemplo de inpainting com o professor na frente do quadro branco.

contemplou corretamente a maior parte da área do objeto em primeiro plano. Apesar do professor estar na frente dos traços escritos no quadro, o sistema não consegue inferir a partir de uma única imagem o que está ocluso por ele.

6.3.3 Melhoria visual

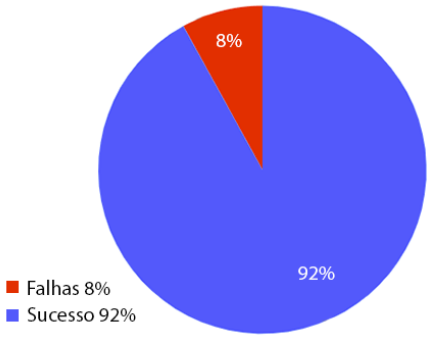


Figura 6.9: Gráfico com a taxa de sucesso do módulo de melhoria visual do quadro branco no sistema.

A Figura 6.9 mostra a taxa de sucesso do módulo de melhoria visual do quadro branco no sistema. O critério utilizado para verificar se a imagem era um caso de sucesso ou falha era se o módulo de melhoria era capaz de preencher grande parte da área na imagem (90%) que faz parte do quadro branco com a cor branca, sem prejudicar a leitura dos traços nele escritos. Isso é necessário pois esse módulo depende exclusivamente da iluminação do ambiente e da cor do quadro para funcionar. Como ele faz apenas uma operação de *threshold* com o limiar de 165 para cada canal, é necessário que as imagens tenham valores de pixel do quadro branco maiores ou iguais a esse limiar.

Na Figura 6.10 podemos ver exemplos de sucesso do módulo de melhoria visual. O objetivo é que a imagem final tenha um aspecto visual uniforme, ressaltando os traços

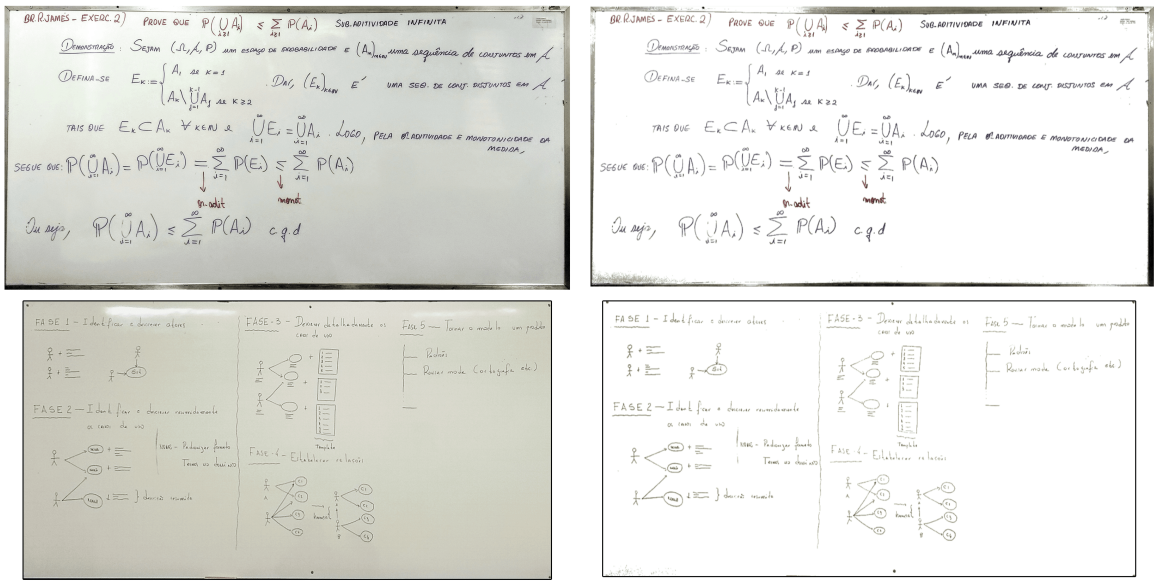


Figura 6.10: Exemplos de sucesso do módulo de melhoria visual.

escritos no quadro branco. Então foi feito com que todo o resto do quadro tivesse a mesma cor branca.

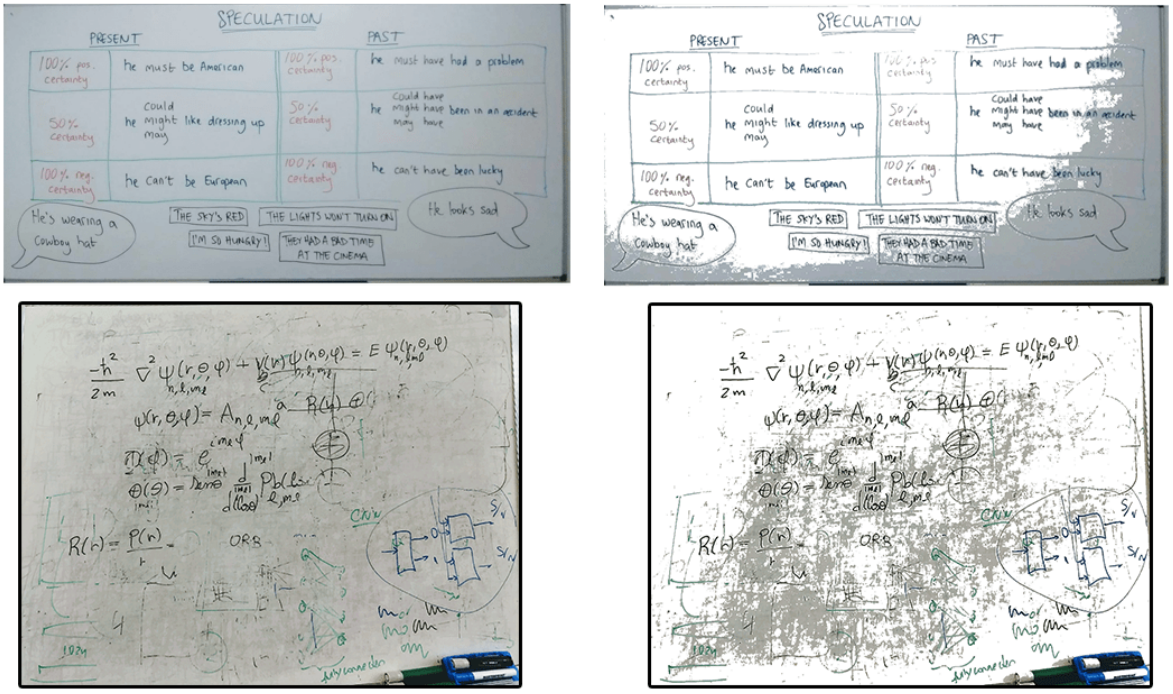


Figura 6.11: Exemplos de falha do módulo de melhoria visual.

Na Figura 6.11 podemos ver quando esse módulo não funciona corretamente. O principal fator para que ele não funcione é a imprevisibilidade de algumas imagens de entrada. Se a imagem do quadro não tiver pixeis que atendam aos requisitos vistos no Algoritmo 5 da Seção 4.4, a melhoria não será executada de forma correta. Isso acontece com imagens escuras, ou que alguma sombra ficou na frente do quadro. Também pode acontecer uma falha quando o algoritmo deixa ilegível os traços escritos no quadro branco. Isso acontece geralmente quando o traço é muito fino, ou a foto foi tirada a uma distância muito grande do quadro branco.

6.4 Análises finais

A Figura 6.12 mostra exemplos de imagens que passaram por todo o fluxo de execução do sistema, e foram subjetivamente classificados como bem sucedidos. As imagens à esquerda são as fotos originais, as da direita são os resultados após utilizar todos os módulos do sistema diretamente nelas pelo aplicativo. Uma borda preta foi manualmente aplicada nelas para melhorar a visualização.

Essas imagens mostram o que seria um resultado esperado do sistema. Devem possuir um aspecto visual de uma imagem escaneada, contendo somente o conteúdo do quadro branco, ou seja, os traços nele escritos. Analisando esses resultados subjetivamente, eles cumprem com os requisitos.

Figura 6.12: Resultados de todo o fluxo.

Capítulo 7

Conclusão

O principal foco deste trabalho era o de produzir um sistema capaz de escanear o conteúdo do quadro branco a partir de uma imagem digital tirada de um quadro branco, implementado em um aplicativo Android. O objetivo de ser implementado em um aplicativo é que atualmente muitas pessoas já possuem acesso a um *smartphone*, no Brasil já há mais celulares do que pessoas [24]. Os requisitos mínimos para utilizar o sistema, é ter um celular com Android 7.1.1 ou superior instalado. Não é preciso ter câmera necessariamente, pois o sistema busca as fotos da galeria do aparelho.

Este trabalho foi desenvolvido utilizando as técnicas de processamento de imagens introduzidas em trabalhos de pesquisa anteriores. O fluxo de execução do sistema foi criado a partir dos conhecimentos de obtidos a partir de estudos sobre essas técnicas mais adequadas até o presente momento para resolver o problema proposto. Também foram utilizados conhecimentos empíricos inferidos ao longo do seu desenvolvimento para produzir os melhores resultados.

O sistema utiliza 3 módulos para produzir o resultado final: `detecção do quadro branco`, `inpainting`, e `melhoria visual`. Também utiliza um módulo embutido de `redimensionamento` quando é necessário reduzir o tamanho da imagem final. Para fins de usabilidade do aplicativo final pelos usuários, a solução foi implementada de uma forma que provesse a maior liberdade. Portanto independente do estado com que a foto do quadro branco foi tirada, o usuário pode decidir qual módulo do sistema deseja aplicar sobre ela, não importando se aplicou os outros módulos anteriormente.

Os módulos foram testados independentemente em um banco com mais de 100 imagens de quadros brancos em diversas situações. Os resultados se mostraram subjetivamente satisfatórios para a maioria das imagens, dadas as circunstâncias de iluminação, cor e estrutura do quadro branco que produzem os melhores resultados. Obteve-se uma taxa de sucesso média de 88% para os módulos. 83% para o módulo de detecção do quadro branco, 89% para o de *inpainting*, e 92% para o módulo de melhoria visual

Posteriormente em trabalhos futuros, algumas melhorias não implementadas neste trabalho poderiam ser desenvolvidas para sanar suas impropriedades, como:

- Detecção de contornos de quadros brancos sem bordas evidentes, ou com objetos retos próximos que interfiram na detecção;
- Velocidade do processamento do *inpainting*, chega a demorar mais de 2 minutos dependendo da imagem.
- O processo de melhoria visual apresenta a maior parte das falhas concentradas em imagens muito escuras.

Particularmente, o produto aqui desenvolvido serve como uma prova de conceito para validar a solução proposta. Ele ainda não está maduro o suficiente para ser implantado como uma solução comercial viável para o público amplo, mas abre as portas para que possa ser evoluído para esse fim.

Referências

[1] Neubeck, Lis, Nicole Lowres, Emelia J Benjamin, S Ben Freedman, Genevieve Coorey e Julie Redfern: *The mobile revolution—using smartphone apps to prevent cardiovascular disease*. Nature Reviews Cardiology, 12(6):350, 2015. 1

[2] Zhang, Zhengyou e Li wei He: *Note-taking with a camera: whiteboard scanning and image enhancement*. Em *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 3, páginas iii–533. IEEE, 2004. 1, 18

[3] Telea, Alexandru: *An image inpainting technique based on the fast marching method*. Journal of graphics tools, 9(1):23–34, 2004. 1, 15

[4] Gonzalez, Rafael C, Richard E Woods *et al.*: *Digital image processing [m]*. Publishing house of electronics industry, 141(7), 2002. 4

[5] Deng, Guang e LW Cahill: *An adaptive gaussian filter for noise reduction and edge detection*. Em *1993 IEEE Conference Record Nuclear Science Symposium and Medical Imaging Conference*, páginas 1615–1619. IEEE, 1993. 6

[6] Haddad, Richard A e Ali N Akansu: *A class of fast gaussian binomial filters for speech and image processing*. IEEE Transactions on Signal Processing, 39(3):723–727, 1991. 6

[7] Sauvola, Jaakko e Matti Pietikäinen: *Adaptive document image binarization*. Pattern recognition, 33(2):225–236, 2000. 7

[8] Otsu, Nobuyuki: *A threshold selection method from gray-level histograms*. IEEE transactions on systems, man, and cybernetics, 9(1):62–66, 1979. 7

[9] Scuri, Antonio Escaño: *Fundamentos da imagem digital*. Pontifícia Universidade Católica do Rio de Janeiro, 1999. 9

[10] Canny, John: *A computational approach to edge detection*. Em *Readings in computer vision*, páginas 184–203. Elsevier, 1987. 10

[11] Duda, Richard O e Peter E Hart: *Use of the hough transformation to detect lines and curves in pictures*. Relatório Técnico, SRI INTERNATIONAL MENLO PARK CA ARTIFICIAL INTELLIGENCE CENTER, 1971. 11

[12] Bôcher, Maxime: *Plane analytic geometry, with introductory chapters on the differential calculus*. Bull. Amer. Math. Soc, 23:102, 1916. 11

[13] Fusiello, Andrea, Emanuele Trucco e Alessandro Verri: *A compact algorithm for rectification of stereo pairs*. Machine Vision and Applications, 12(1):16–22, 2000. 14

[14] Loop, Charles e Zhengyou Zhang: *Computing rectifying homographies for stereo vision*. Em *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, volume 1, páginas 125–131. IEEE, 1999. 14

[15] Sethian, James A: *A fast marching level set method for monotonically advancing fronts*. Proceedings of the National Academy of Sciences, 93(4):1591–1595, 1996. 15

[16] MacQueen, James *et al.*: *Some methods for classification and analysis of multivariate observations*. Em *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, páginas 281–297. Oakland, CA, USA, 1967. 16

[17] Kota, Bhargava Urala, Kenny Davila, Alexander Stone, Srirangaraj Setlur e Venu Govindaraju: *Automated detection of handwritten whiteboard content in lecture videos for summarization*. Em *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, páginas 19–24. IEEE, 2018. 19

[18] Prabhu, Nagesh, R Pradeep Kumar, T Punitha e Raman Srinivasan: *Whiteboard documentation through foreground object detection and stroke classification*. Em *2008 IEEE International Conference on Systems, Man and Cybernetics*, páginas 336–340. IEEE, 2008. 19, 31

[19] He, Li wei e Zhengyou Zhang: *Real-time whiteboard capture and processing using a video camera for teleconferencing*. Em *Proceedings.(ICASSP'05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.*, volume 2, páginas ii–1113. IEEE, 2005. 20

[20] Dickson, Paul E, W Richards Adrion e Allen R Hanson: *Whiteboard content extraction and analysis for the classroom environment*. Em *2008 Tenth IEEE International Symposium on Multimedia*, páginas 702–707. IEEE, 2008. 20

[21] Zhang, Xinpeng: *Lossy compression and iterative reconstruction for encrypted image*. IEEE transactions on information forensics and security, 6(1):53–58, 2010. 23

[22] StatCounter: *Mobile operating system market share worldwide*, 2019. `http://gs.statcounter.com/os-market-share/mobile/worldwide`, acesso em 2019-06-03. 37

[23] Google®: *Documentation for app developers*, 2019. `https://developer.android.com/docs`, acesso em 2019-07-06. 38

[24] StatCounter: *Brasil tem 230 milhões de smartphones em uso*, 2019. `https://epocanegocios.globo.com/Tecnologia/noticia/2019/04/brasil-tem-230-milhoes-de-smartphones-em-uso.html`, acesso em 2019-06-21. 52