

**Fundação Getulio Vargas  
Escola de Matemática Aplicada  
Curso de Graduação em Matemática  
Aplicada**

**Título da dissertação**

**Emanuel Bissiatti de Almeida**

Rio de Janeiro - Brasil  
2023

**Fundação Getulio Vargas  
Escola de Matemática Aplicada  
Curso de Graduação em Matemática  
Aplicada**

**Título da dissertação**

“Declaro ser o único autor do presente projeto de monografia que refere-se ao plano de trabalho a ser executado para continuidade da monografia e ressalto que não recorri a qualquer forma de colaboração ou auxílio de terceiros para realizá-lo a não ser nos casos e para os fins autorizados pelo professor orientador.”

---

**Nome**

Rio de Janeiro - Brasil  
2023

**Fundação Getulio Vargas  
Escola de Matemática Aplicada  
Curso de Graduação em Matemática  
Aplicada**

**Título da dissertação**

**“Projeto de Monografia apresentado à Escola de Matemática  
Aplicada como requisito parcial para continuidade ao trabalho de  
monografia.”**

Aprovado em \_\_\_\_ de \_\_\_\_\_ de \_\_\_\_\_  
Grau atribuído ao Projeto de Monografia: \_\_\_\_\_

---

**Professor Orientador:  
Escola de Matemática Aplicada  
Fundação Getúlio Vargas**

# 1 Introdução

O acervo de fotografias históricas ImagineRio (IMAGINERIO, 2023), ajuda a visualizar a evolução da cidade do Rio de Janeiro ao longo do tempo. Entretanto, esse retrato da cidade é limitado: as fotografias são escassas, um mesmo edifício possui poucas amostras de fotografias em um mesmo período de tempo e as fotografias não são apenas uma projeção bidimensional da realidade no sensor da câmera (HARTLEY; ZISSEMAN, 2004). O trabalho do Tomás Ferranti (FERRANTI, 2021) apresenta uma ferramenta para que dado uma fotografia de um edifício e a seleção manual das arestas que compõem esse edifício classificadas por direção, seja possível reconstruir tridimensionalmente a fachada desse edifício. Esse trabalho busca automatizar este processo, dado apenas uma imagem de um edifício, por meio de técnicas de visão computacional e aprendizado profundo de máquina seja possível selecionar automaticamente as arestas que compõem esse edifício e classificá-las por direção. Por fim, será proposto uma integrar essa ferramenta ao trabalho do Tomás Ferranti (FERRANTI, 2021) para que seja possível reconstruir as fachadas de um edifício a partir de uma única fotografia.

Para resolver esse problema, esse trabalho computacional foi subdividido em três problemas menores em que cada problema depende da solução do problema anterior. A **primeira etapa** corresponde a técnicas de detecção de arestas em imagens, na qual o objetivo é classificar cada pixel da imagem em aresta ou não aresta. A **segunda etapa** corresponde às técnicas de detecção de segmentos de retas, na qual o objetivo é identificar as equações dos segmentos de retas que compõem as arestas da imagem. Por fim, a **terceira etapa** corresponde às técnicas de classificação de segmentos de retas, na qual o objetivo é usar as propriedades matemáticas das equações das retas e classifica-las de acordo com o ponto de fuga correspondente.

Por hipótese, a solução proposta nesse trabalho considera que os ângulos entre as faces do edifícios são aproximadamente retos e que suas arestas pertencem a uma reta em 3 dimensões. Essa hipótese é razoável, pois, a maioria das construções possuem faces aproximadamente retas e suas arestas pertencem a uma reta em 3 dimensões. Entretanto, essa hipótese não é válida para todos os edifícios, pois, existem alguns edifícios como o Museu de Arte Contemporânea de Niterói, planejado pelo arquiteto Oscar Niemeyer, que possuem faces curvas. O artigo (FERRANTI, 2021) também carrega essas hipóteses, sendo assim, a solução do presente artigo, que é um complemento ao trabalho do Tomás, também leva essas hipóteses em consideração.

As três próximas seções desse trabalho discutirão as técnicas utilizadas para resolver cada um dos problemas propostos: a detecção de arestas, a detecção de segmentos de retas e a classificação de segmentos de retas. A seção INTEGRAÇÃO será dedicada a integrar a solução proposta com a ferramenta desenvolvida pelo Tomás Ferranti. Por fim, a seção CONCLUSÃO apresentará as conclusões desse trabalho e a avaliação dos resultados obtidos.

## 2 Detecção de arestas

A detecção de arestas é um problema clássico de processamento de imagens, onde o objetivo é identificar os limites de objetos contidos em uma imagem. Em especial, esse trabalho busca identificar as arestas que delimitam as faces de edifícios, que em geral são linhas retas.

### 2.1 Algoritmos clássicos

Os algoritmos clássicos de detecção de arestas são baseados na aplicação de filtros em uma imagem que identificam mudanças bruscas de intensidade. Observe que a imagem de uma aresta é uma região onde a intensidade da imagem muda bruscamente, sendo assim, os algoritmos de detecção de arestas são baseados em filtros que identificam essas mudanças de intensidade por meio de derivadas da intensidade da imagem ([SHRIVAKSHAN; CHANDRASEKAR, 2012](#)). A figura 1, mostra a fotografia do edifício da biblioteca nacional e o resultado da aplicação do algoritmo de Sobel e do algoritmo de Canny.

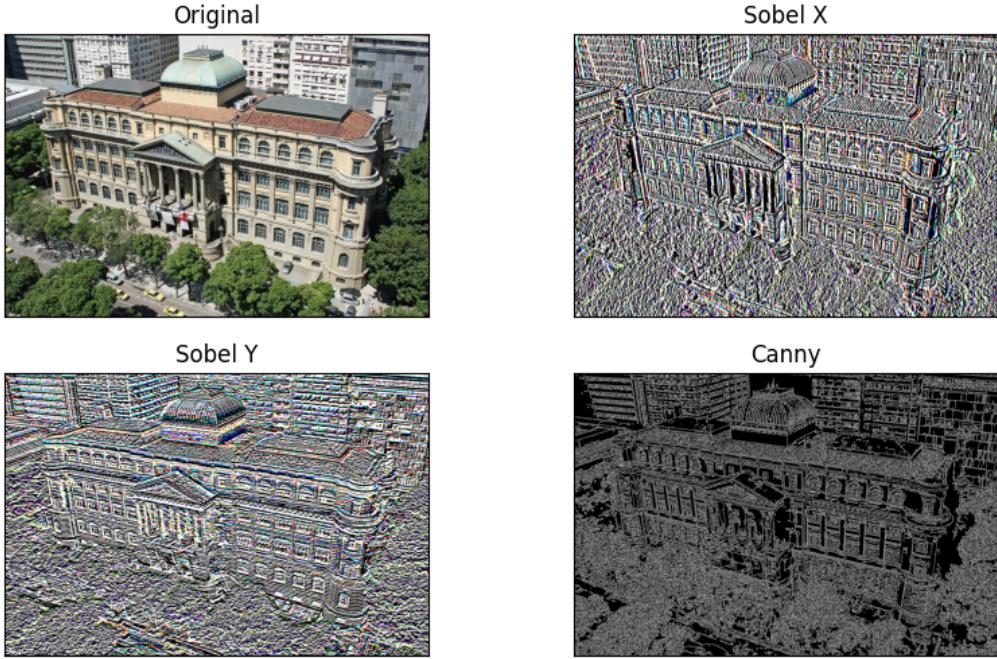


Figura 1: Imagem original e resultado do algoritmo de Sobel.

O algoritmo de Sobel é um filtro convolucional que calcula a magnitude do gradiente da intensidade da imagem. O algoritmo de Canny é um algoritmo mais sofisticado que utiliza o algoritmo de Sobel para calcular a magnitude do gradiente da intensidade da imagem e aplica uma série de filtros para remover ruídos e destacar as arestas da imagem. Em geral o algoritmo de Canny apresenta melhores resultados que o algoritmo de Sobel, porém, o algoritmo de Canny é mais custoso computacionalmente, apesar disso, o custo computacional não é está sendo considerado nesse trabalho.

Em análise, ambos os algoritmos não apresentam resultados satisfatórios para essa aplicação. O algoritmo de Sobel apresenta muitos ruídos e o algoritmo de Canny apresenta muitas arestas que não são relevantes para a aplicação. Essa aplicação busca encontrar as arestas que delimitam as faces de edifícios, entretanto, esses algoritmos detectam todas as arestas da imagem, incluindo as que compõem a textura da parede dos edifícios e as janelas.

Uma abordagem para resolver esse problema seria utilizar um desses algoritmos e depois aplicar um modelo que seleciona o subconjunto de arestas

ótimas para a aplicação, ou seja, do conjunto de arestas dado pelos algoritmos, selecionar apenas aquelas que delimitam os prédios por meio de aprendizado de máquina. Entretanto, essa abordagem não é eficiente, pois, as técnicas de aprendizado de máquina precisa de um conjunto de treinamento para aprender a selecionar as arestas ótimas, e para obter esse conjunto de treinamento é necessário selecionar manualmente as arestas ótimas. Existem alguns conjuntos de que já foram selecionados manualmente, como o banco de dados ([COUPRIE et al., 2013](#), Nyu Dataset) que realiza uma ótima segmentação de objetos em uma imagem. Porém, para trabalhar com modelos de aprendizado de máquina em imagens é necessário um grande conjunto de dados para treinamento e um grande poder computacional para processar esses dados a qual não está disponível para a realização desse trabalho. Sendo assim, foi decidido utilizar um modelo de aprendizado profundo pré-treinado para a seleção de arestas. Essa abordagem escolhida será melhor comentada na próxima seção.

## 2.2 Redes neurais convolucionais

As redes neurais ([BISHOP, 2016](#)) são uma combinação de funções matemáticas lineares e não lineares que, em uma sequência de camadas, transformam a informação produzindo uma saída única para cada entrada a partir dos pesos da rede. Em geral, esses pesos são iniciados aleatoriamente e são ajustados por meio de um algoritmo de otimização que minimiza uma função de perda. Entretanto, quanto maior a rede, maior é o número de parâmetros que precisam ser ajustados e maior é o custo computacional para realizar o ajuste desses parâmetros, e alguns dados precisam de um grande número de parâmetros para serem representados. Por exemplo, uma imagem de  $1000 \times 1000$  pixels precisa de  $1000 \times 1000 = 1000000$  parâmetros para ser representada, considerando apenas uma camada de entrada. Por conta disso, uma das técnicas utilizadas para trabalhar com imagens é por meio das redes convolucionais.

Uma rede neural convolucional ([GOODFELLOW; BENGIO; COURVILLE, 2016](#)) é um conjunto de técnicas de aprendizado profundo que foram desenvolvidas para trabalhar com dados tabulares. Essa técnica é uma extensão das redes neurais tradicionais, onde as camadas de neurônios são organizadas em camadas convolucional e camadas de pooling. As camadas convolucional são responsáveis por extrair as características da imagem e as camadas de pooling são responsáveis por reduzir a dimensionalidade da imagem. A figura 2 mostra um exemplo de uma rede neural convolucional, nesse

exemplo, após aplicar uma rede neural convolucional foi Aplicada uma rede neural tradicional para classificar a imagem.

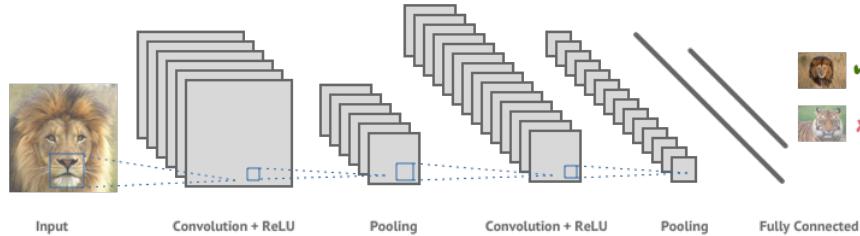


Figura 2: exemplo de rede neural convolucional

Durante o treinamento de uma rede neural convolucional, os pesos são ajustados por meio de um algoritmo de otimização que minimiza uma função de perda. Entretanto, o ajuste dos pesos de uma rede neural convolucional é mais eficiente que o ajuste dos pesos de uma rede neural tradicional, pois, os pesos são compartilhados entre as camadas convolucionais, ou seja, os pesos são ajustados para todas as regiões da imagem. Por exemplo, considere uma imagem de  $1000 \times 1000$  pixels e uma camada convolucional de  $3 \times 3$  pixels, nesse caso, os pesos são ajustados para todas as regiões de  $3 \times 3$  pixels da imagem. Sendo assim, o número de parâmetros que precisam ser ajustados é muito menor que o número de parâmetros de uma rede neural tradicional. Além disso, o cientista não precisa se preocupar em selecionar as características da imagem, pois, as camadas convolucionais são responsáveis por extrair as características da imagem automaticamente, os filtros possuem seus parâmetros ajustados durante o treinamento da rede neural, sendo que a única preocupação do cientista é definir a arquitetura da rede neural o que incluiu o número de camadas convolucionais e o número de filtros de cada camada convolucional.

Treinar uma rede neural convolucional do zero é um processo custoso, pois, é necessário um grande conjunto de dados para treinamento e um grande poder computacional para processar esses dados. Felizmente, após um modelo ser treinado, isto é, após encontrar os pesos ótimos de uma rede neural dado a função de erro e o conjunto de dados, é possível reutilizar esse modelo em outras aplicações, e, caso seja necessário, aplicar um pós-processamento para direcionar o modelo para uma aplicação específica. Sendo assim, foi encontrado na literatura um modelo de aprendizado profundo que já foi trei-

nando para a tarefa de detecção de arestas, um dos modelos já estabelecidos para essa tarefa é o Holistically-Nested Edge Detection (HED) (XIE; TU, 2015). Esse modelo foi um aprimoramento da rede VGG-16 (SIMONYAN; ZISSERMAN, 2014), uma rede convolucional genérica que possuiu seus pesos treinados com milhões de imagens para Classificação, especializando-o para a detecção de arestas a partir do banco de dados BSDS500 (ARBELAEZ et al., 2011) que contém 500 imagens de treinamento e 200 imagens de teste que tiveram suas arestas selecionadas manualmente. Após o treinamento, o HED foi capaz de selecionar as arestas das imagens do banco de dados BSDS500 com uma acurácia de 0.790.

A vantagem do modelo HED em relação os modelos clássicos de detecção de arestas é que, por meio do aprendizado de máquina, ele consegue delimitar as bordas que compõem os objetos da imagem. Isso ocorre porque esse modelo conseguiu aprender as representações hierárquicas das imagens, ou seja, ele resolve o problema de ambiguidade entre a detecção de bordas e a segmentação de objetos. A figura 3 mostra a fotografia do edifício da biblioteca nacional e o resultado da aplicação do modelo HED.



Figura 3: Imagem original e resultado do modelo HED.

### 2.3 Resultados preliminares

A seguir são apresentados os resultados comparativos da detecção de arestas utilizando o algoritmo de Canny e o modelo de aprendizado profundo pré-treinado HED. A figura 4 mostra a imagem original e o resultado da aplicação do algoritmo de Canny e do modelo HED. Observe que o modelo HED consegue selecionar as arestas que descreve melhor a forma dos edifícios, mas, distorce um pouco a forma das arestas. Por outro lado, o algoritmo de

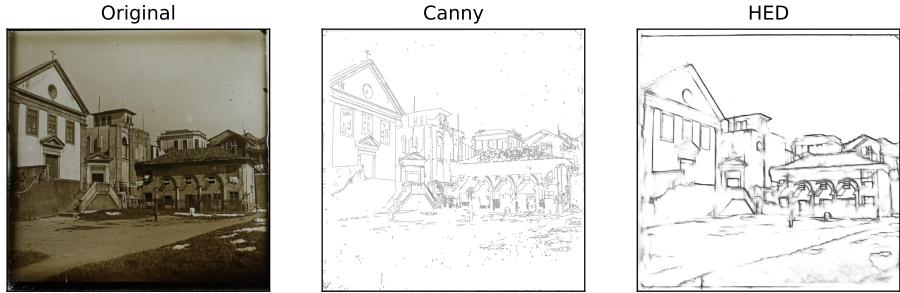


Figura 5: Imagem original e resultado da detecção de arestas na base de dados do ImageRio.

Canny não consegue selecionar as arestas que o descrevem melhor, mas, não distorce a forma das arestas. Para realizar o processamento foi utilizados a implementação dos modelos pela biblioteca ([ITSEEZ, 2015](#), OpenCV) na interface Python.

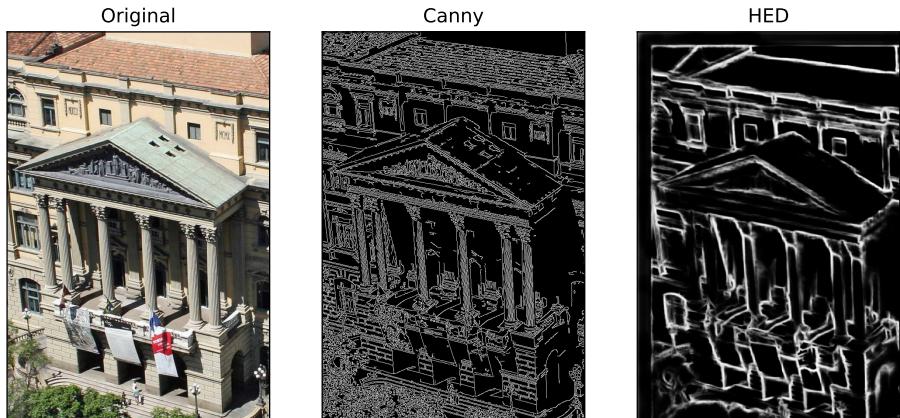


Figura 4: Imagem original e resultado da detecção de arestas.

O modelo HED será utilizado no pipeline desse trabalho, e, a saída dessa etapa é uma imagem binária na qual cada pixel é classificado como aresta, pontos brancos, ou não, pontos pretos. A figura 5 mostra a imagem original e o resultado da aplicação do modelo HED na base de dados do ImageRio.

## 3 Detecção de segmentos de retas

O retorno da etapa anterior é uma imagem binária na qual cada pixel é classificado como aresta, pontos brancos, ou não, pontos pretos. Tradicionalmente, uma imagem representada como um conjunto de pontos num sistema de coordenadas cartesianas, onde cada ponto é representado por um pixel, possuiu a origem  $(0, 0)$  no ponto superior esquerdo da imagem e o eixo  $x$  cresce para a direita e o eixo  $y$  cresce para baixo.

Sendo assim, o objetivo dessa etapa é identificar as equações dos segmentos de retas que compõem as arestas da imagem. Nesta etapa dois algoritmos foram testados: a transformada de Hough e a detecção de segmentos.

Antes de prosseguir pelo pipeline, foi observado que o modelo HED produz uma imagem com ruídos e que algumas arestas não são contínuas, ou seja, existem alguns pontos que não são conectados por uma aresta além de que existem arestas que são muito grossas o que pode gerar uma dupla detecção de arestas pelos algoritmos que se seguem. Para resolver esse problema, foi aplicado filtros morfológicos e filtros de suavização na imagem binária que serão discutidos na próxima subseção.

### 3.1 filtros morfológicos

Os filtros morfológicos ([FISHER et al., 2003](#)) são operadores que atuam em imagens binárias, ou seja, imagens que possuem apenas dois valores de intensidade, preto e branco. Esses filtros são baseados na teoria dos conjuntos e são utilizados para remover ruídos e para destacar as características de interesse da imagem. Os filtros morfológicos mais comuns são a erosão e a dilatação. A erosão remove os pixels brancos que estão próximos aos pixels pretos, ou seja, ela remove os pixels brancos que estão próximos as arestas. A dilatação adiciona pixels brancos próximos aos pixels brancos, ou seja, ela adiciona pixels brancos próximos as arestas. A abertura também é um filtro morfológico que é a composição da erosão com a dilatação, ou seja, a erosão é aplicada primeiro e depois a dilatação. A abertura remove os pixels brancos que estão próximos aos pixels pretos e adiciona pixels brancos próximos aos pixels brancos, ou seja, ela remove os pixels brancos que estão próximos as arestas e adiciona pixels brancos próximos as arestas. Outro filtro morfológico utilizado foi o fechamento, que é a composição da erosão com a dilatação, ou seja, a erosão é aplicada primeiro e depois a dilatação. O fechamento remove os buracos dentro das arestas e preenche as arestas

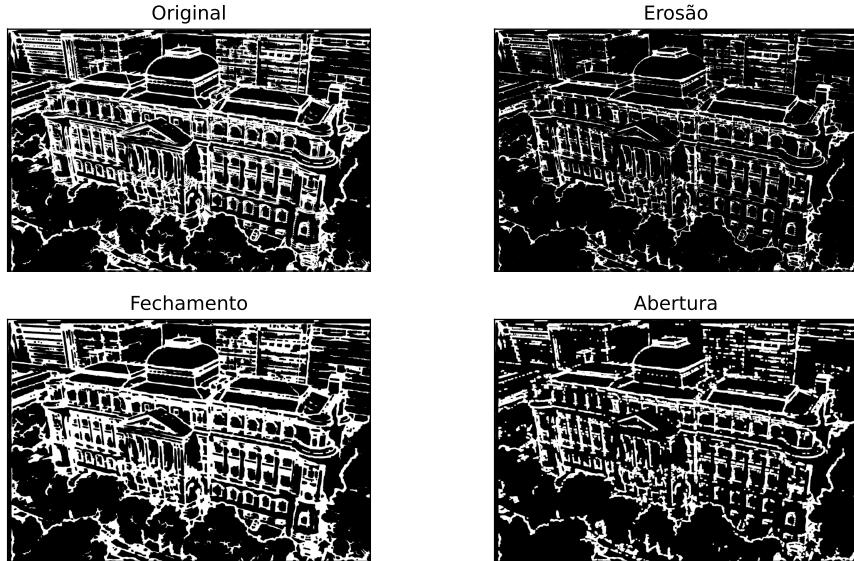


Figura 6: Imagem original e resultado da aplicação dos filtros morfológicos.

quebradas. A figura 6 mostra a imagem original e o resultado da aplicação do fechamento e da erosão utilizado o opencv ([OPENCV, 2023b](#)) com um elemento estruturante de tamanho  $(9 \times 9)$ .

É possível notar que o filtro de erosão consegue reduzir a espessura das arestas e o filtro de fechamento consegue remover os buracos dentro das arestas e preencher as arestas quebradas. Entretanto, o filtro de erosão ainda deixa arestas quebradiças e o filtro de fechamento ainda deixa arestas muito grossas. Para resolver o problema das arestas quebradiças, foi aplicado um filtro de suavização na imagem binária que será discutido na próxima subseção.

### 3.2 Filtros de suavização

Os filtros de suavização, também conhecidos como filtros de borramento, são operadores que amenizam as transições bruscas de intensidade da imagem. Neste caso, os filtros foram utilizados para suavizar as arestas quebradiças ao borrar a imagem. Os filtros de suavização que foram utilizados nesse trabalho foram o filtro de mediana e o filtro gaussiano. O filtro da mediana substitui o valor de um pixel pela mediana dos valores dos pixels vizinhos. O filtro gaussiano substitui o valor de um pixel pela média ponderada dos

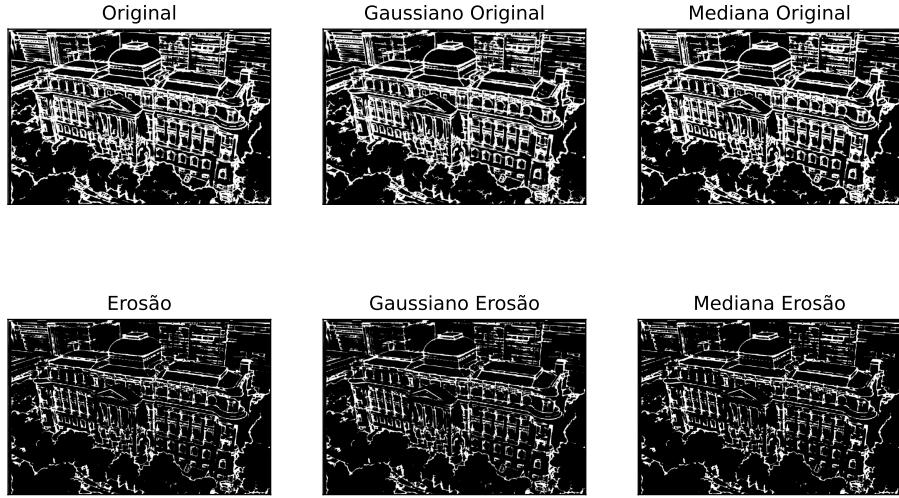


Figura 7: Imagem original e resultado da aplicação dos filtros de suavização.

valores dos pixels vizinhos. A figura 7 mostra a imagem original e o resultado da aplicação do filtro de mediana e do filtro gaussiano utilizado o opencv ([OPENCV, 2023c](#)) com um kernel de tamanho  $(5 \times 5)$ .

### 3.3 Transformada de Hough

A transformada de Hough ([ILLINGWORTH; KITTNER, 1988](#)) é um algoritmo de visão computacional que é capaz de detectar figuras geométricas em uma imagem, em especial retas. O algoritmo consiste em transformar a imagem de coordenadas cartesianas para coordenadas polares, onde cada ponto da imagem é representado por uma reta na imagem transformada. Nesse espaço, para encontrar uma reta, basta encontrar os pontos que se intersectam. Por isso, foi-se utilizado a transformada de Hough probabilística ([MATAS; GALAMBOS; KITTNER, 2000](#)) que é uma versão mais eficiente da transformada de Hough. A figura 8 mostra a imagem original e o resultado da aplicação da transformada de Hough probabilística utilizado o opencv ([OPENCV, 2023a](#)).

Nota-se que a transformada de Hough probabilística consegue detectar as arestas que compõem os edifícios, entretanto, ela não consegue detectar todas as arestas e algumas arestas são detectadas mais de uma vez. Por isso, foi testado um método mais robusto que será discutido na próxima subseção:

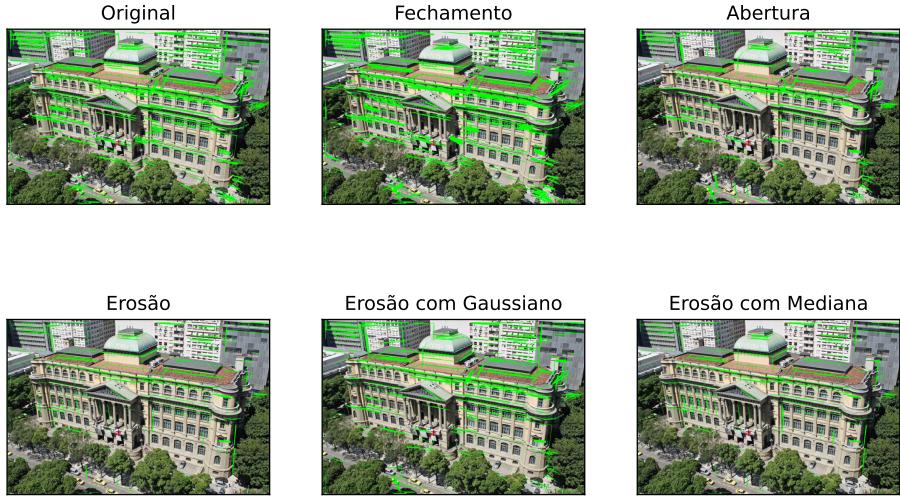


Figura 8: Imagem original e resultado da aplicação da transformada de Hough probabilística.

a detecção de segmentos de retas.

### 3.4 Detecção de segmentos a partir de pontos

Para a detecção de segmentos, foi utilizada a implementação do Line Segment Detector ([THE...](#), 2014) pela biblioteca opencv ([ITSEEZ](#), 2015) na interface Python. O LineSegmentDetector é um algoritmo de visão computacional que é capaz de detectar segmentos de retas em uma imagem, seu algoritmo é baseado em uma técnica de agrupamento de pontos chamada de **Progressive Probabilistic Hough Transform (PPHT)** ([MATAS; GALAMBOS; KITTNER](#), 2000). O algoritmo PPHT utilizado pelo Line Segment Detector opera de forma hierárquica, refinando gradualmente o processo de detecção de linha. Ele começa analisando a imagem em baixa resolução e aumenta progressivamente a resolução, permitindo a detecção de segmentos de linha grosseiros e finos. A desvantagem é que ele detecta todos os segmentos de linha selecionados pelo HED, incluindo os ruídos, e, por isso, é necessário desconsiderar os segmentos de linha que não são relevantes para a aplicação. A figura 9 mostra a imagem original e o resultado da aplicação do Line Segment Detector:

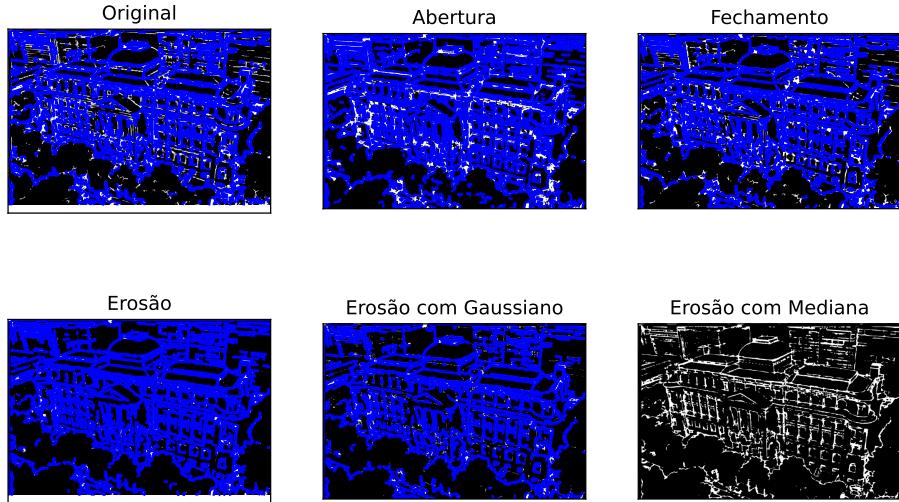


Figura 9: Imagem original e resultado da aplicação do Line Segment Detector.

## 4 Classificação de segmentos

Nota-se que o algoritmo de detecção de segmentos de retas do opencv obteve bom resultado em relação a transformada de Hough probabilística, sendo assim, o cálculo dessa seção será baseado nos segmentos de retas deste algoritmo. Nesta seção será discutido como classificar os segmentos de retas matematicamente a partir dos pontos extremos de cada segmento. A próxima subseção discutirá as propriedades matemáticas dos pontos de fuga e como utilizar essas propriedades para classificar os segmentos de retas de forma computacionalmente eficiente.

### 4.1 Modelo de câmera pinhole

O modelo matemático mais simples para representar a fotografia de uma cena é o modelo de câmera pinhole. Esse modelo é baseado na projeção de perspectiva, onde os raios de luz que saem de um ponto da cena e atingem o plano da imagem são representados por retas que passam pelo centro de projeção da câmera. Esse modelo despreza as distorções óticas da lente da câmera e considera que a câmera está alinhada com o plano da imagem, ou seja, a câmera está perpendicular ao plano da imagem, esse mesmo modelo

é utilizado pelo trabalho do Tomás Ferranti ([FERRANTI, 2021](#)).

Considerando esse modelo, tem-se cada ponto da cena é representado no plano projetivo por um raio de luz que passa pelo centro de projeção da câmera e atinge o plano da imagem. O plano projetivo é um plano bidimensional, um subespaço afim de dimensão 2, que contém a imagem projetada pela luz que passou pelas lentes da câmera. A principal propriedade do plano projetivo que é interessante para essa aplicação são os pontos de fuga. Os pontos de fuga são os pontos de intersecção de retas paralelas no  $R^3$  que são projetadas em retas concorrentes no plano projetivo. Como axioma, todas as retas paralelas no  $R^3$  se intersectam em um ponto no plano projetivo, nem que este seja no infinito. Um ponto de fuga é dito degenerado quando ele está no infinito.

## 4.2 Classificação de acordo com os pontos de fuga

Os dados dessa aplicação são fotografias de edifícios, e, esses dados possuem algumas hipóteses que serão utilizadas para classificar os segmentos de retas. A primeira hipótese é que os edifícios possuem faces aproximadamente retas, ou seja, os segmentos de retas que compõem as arestas dos edifícios são aproximadamente retos. A segunda hipótese é que os edifícios possuem faces que são perpendiculares entre si, ou seja, os segmentos de retas que compõem as arestas dos edifícios são perpendiculares entre si. A terceira hipótese é que a câmera está externa ao edifício. A quarta hipótese é que um dos pontos de fuga é aproximadamente degenerado verticalmente, ou seja, a câmera está aproximadamente alinhada com a base do edifício. Todas as hipóteses são razoáveis considerando os dados alvo: fotografias históricas de faxadas de edifícios.

Considerando a representação digitais binárias, como ocorre com a imagem de saída da etapa da etapa de detecção de arestas, a origem  $(0, 0)$  do plano cartesiano está no ponto superior esquerdo com o eixo  $x$  crescendo para a direita e o eixo  $y$  crescendo para baixo. Então, a equação da inclinação da reta que passa pelos pontos  $(x_1, y_1)$  e  $(x_2, y_2)$  é dada por:

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

Podemos obter o ângulo  $\theta$  que a reta faz com o eixo  $x$  por meio da função arco tangente:

$$\theta = \arctan(m)$$

A hipótese de que a câmera está alinhada com a base do edifício, ou seja, um dos pontos de fuga é aproximadamente degenerado verticalmente, implica que a reta que passa pelo ponto  $(x_1, y_1)$  e  $(x_2, y_2)$  é aproximadamente vertical, ou seja, o coeficiente angular  $m$ , em módulo, tende ao infinito. Sendo assim, foi definido um parâmetro  $\alpha$  tal que se  $m > \alpha$  então a reta é aproximadamente vertical e esta foi classificado com o ponto de fuga vertical de classe 0. Por padrão o parâmetro  $\alpha$  foi definido como  $\alpha = 2$ , arbitrariamente. Os demais seguimentos foram classificados em mais dois grupos de acordo com os pontos de fuga restantes, para essa divisão, houve a subdivisão do problema em 3 casos:

1. Existe um ponto de fuga horizontal degenerado, ou seja, a câmera está aproximadamente alinhada com a lateral do edifício.
2. Não existe um ponto de fuga horizontal degenerado, mas, os pontos de fugas estão distantes entre si e distantes do centro da imagem.
3. Não existe um ponto de fuga horizontal degenerado, mas, os pontos de fugas estão próximos entre si e próximos do centro da imagem.

#### 4.2.1 Ponto de fuga horizontal degenerado

A figura 10 de ([ADAMY GIANINNI POR PIXABAY, 2023](#)) é um exemplo de fotografia com o ponto de fuga horizontal degenerado. Nesse caso, a câmera está aproximadamente alinhada com a lateral do edifício, ou seja, a reta que passa pelo ponto  $(x_1, y_1)$  e  $(x_2, y_2)$  é aproximadamente horizontal, ou seja, o ângulo, em módulo, tende a zero. Sendo assim, foi definido um parâmetro  $\beta$  tal que se  $\theta < \beta$  então a reta é aproximadamente horizontal e esta foi classificado com o ponto de fuga horizontal degenerado de classe 1. Por padrão o parâmetro  $\beta$  foi definido como  $\beta = 0.1$ , arbitrariamente. Os demais seguimentos foram classificados na classe 2, que é a classe restante.



Figura 10: Exemplo de imagem com ponto de fuga horizontal degenerado.

Após aplicar o algoritmo nesse caso foi obtido o seguinte resultado na figura 11:

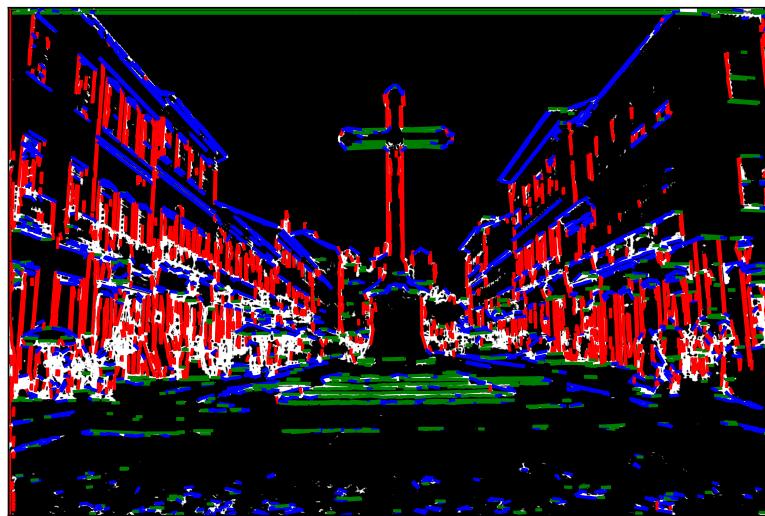


Figura 11: Resultado final da classificação de segmentos de retas no caso de ponto de fuga horizontal degenerado.

endfigure

#### 4.2.2 Pontos de fuga distantes

Caso os pontos de fuga estejam distantes da imagem, então, as retas originalmente paralelas, possuem uma inclinação ligeiramente diferente, pois elas irão se encontrar em um dos pontos de fuga distantes. Neste caso, como a câmera está externa ao edifício e os ângulos entre as faces do edifício são aproximadamente retos então, os ângulos dessas retas no plano projetivo são ângulos agudos por conta dos pontos de fugas estarem distantes. Sendo assim, o ângulo em relação ao eixo  $x$  da reta que passa pelos pontos  $(x_1, y_1)$  e  $(x_2, y_2)$  foi classificado para a classe 1 se  $\theta < 0$  e para a classe 2 se  $\theta > 0$ , pois, necessariamente, um dos pontos de fuga é ascendente e o outro é descendente dado que os ângulos são agudos. A figura 12 mostra um exemplo de pontos de fuga distantes classificados de acordo com essa lógica.

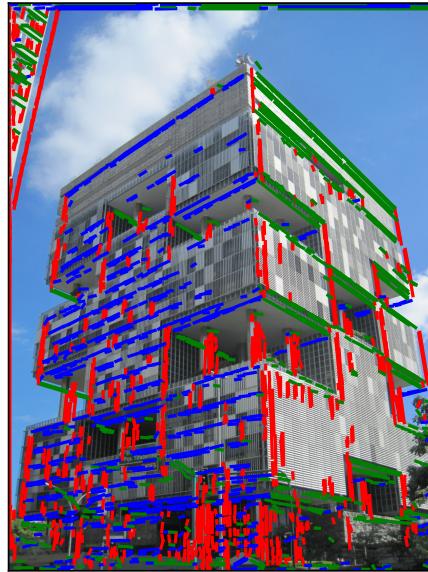


Figura 12: Exemplo de imagem com ponto de fuga distantes

#### 4.2.3 Pontos de fuga próximos

Por outro lado, caso os pontos de fuga estejam próximos da imagem, então, as retas originalmente paralelas, possuem uma inclinação muito diferente, pois elas irão se encontrar em um dos pontos de fuga próximos. Neste caso, como a câmera está externa ao edifício e os ângulos entre as faces do edifício

são aproximadamente retos então, os ângulos dessas retas no plano projetivo são ângulos obtusos já que os pontos de fuga estão próximos. Sendo assim, os pontos de fuga se encontram com seu  $y$  entre essas retas, portanto, a partir da altura  $h$  do ponto de fuga, foi-se necessário inverter a classificação das retas, ou seja, o ângulo em relação ao eixo  $x$  é classificado para a classe 1 se  $\theta < 0$  e se  $y_i < h^*$  ou se  $\theta > 0$  e se  $y_i > h^*$  e para a classe 2 caso contrário. A figura 13 mostra um exemplo de pontos de fuga próximos classificados de acordo com essa lógica.

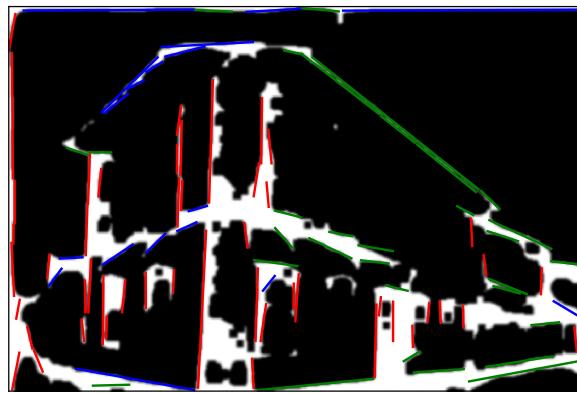


Figura 13: Exemplo de imagem com ponto de fuga próximos

### 4.3 Algoritmo de classificação

Após os comentários em relação a lógica da classificação, o algoritmo de classificação é apresentado a seguir:

---

**Algorithm 1** Funções auxiliares

---

```
1: function CALCULA_INCLINACAO(pontos)
2:    $x_1, y_1, x_2, y_2 \leftarrow$  pontos
3:    $m \leftarrow \frac{y_2 - y_1}{x_2 - x_1}$ 
4:   return m
5: end function
6: function CALCULA_ANGULO(pontos=None, m=None)
7:   if type(m) == None then
8:     m  $\leftarrow$  calcula_inclinacao(pontos)
9:   end if
10:  angulo  $\leftarrow$  np.arctan(m)
11:  return angulo
12: end function
13: function VETOR_INCLINACAO(pontos)
14:   func  $\leftarrow \lambda x : \text{calcula\_inclinacao}(x)$ 
15:   return np.apply_along_axis(func, 1, pontos)
16: end function
17: function VETOR_ANGULO(pontos)
18:   func  $\leftarrow$  np.vectorize(calcula_angulo)
19:   m  $\leftarrow$  func(pontos)
20:   return m
21: end function
22: function VETOR_ANGULO_DADO_M(m)
23:   angulo  $\leftarrow$  calcula_angulo(m = m)
24:   return angulo
25: end function
```

---

---

**Algorithm 2** Função principal de classificação de segmentos de retas

---

```
function CLUSTER_FINAL(n_cluster=3, alpha=2, image_path='t',
image_opencv=None, swap=False, swapPercent=0.75)
2:   if image_opencv ≠ None then
        lines ← getLines(image_opencv=image_opencv)
4:   else
        lines ← getLines(image_path)
6:   end if
        lines ← lines[:, 0 : 4]
8:   m ← vetor_inclinacao(lines)
        angulo ← vetor_angulo_dado_m(m)
10:  class0, class1 ← filtro_inclinacao(m, α)
        class0 ← np.array(class0)
12:  class1 ← np.array(class1)
        labels ← cluster_lines_angulo(angulo[class1], lines[class1], n_cluster, swap=swap, swapPercent=swap)
14:  allClass ← np.zeros([lines.shape[0]])
        allClass[class0] ← 0
16:  allClass[class1] ← labels + 1
        return allClass, lines
18: end function
```

---

---

**Algorithm 3** Função de clusterização de segmentos de retas de acordo com o ângulo para as retas não verticais

```
function CLUSTER_LINES_ANGULO(angulo, lines, n_clusters=2, beta=0.1,
                                swap=False, swapPercent=0.75)
    count ← 0
    3:   for i ← 0 to angulo.shape[0] do
        if |angulo[i]| ≤ beta then
            count ← count + 1
        6:   end if
        end for
        PRINT( $\frac{\text{count}}{\text{angulo.shape[0]}}$ )
        count
    9:   if  $\frac{\text{count}}{\text{angulo.shape[0]}} > 0.25$  then
        values ← []
        for i ← 0 to angulo.shape[0] do
    12:       if |angulo[i]| ≤ beta then
            values.append(0)
        else
    15:           values.append(1)
        end if
        end for
    18:   return np.array(values)
    end if
    h ← max(max(lines[:, 1]), max(lines[:, 3]))
    21:   values ← []
    for i ← 0 to angulo.shape[0] do
        if angulo[i] > 0 then
    24:           if not swap then
                values.append(0)
            else if lines[i, 1] > h × swapPercent or lines[i, 3] > h × swapPercent
            then
                values.append(1)
            else
                values.append(0)
    30:           end if
            else
                if not swap then
    33:                   values.append(1)
            else if lines[i, 1] > h × swapPercent or lines[i, 3] > h × swapPercent
            then
                values.append(0)
    36:           else
                values.append(1)
            end if
    39:       end if
        end for
        return np.array(values)
    42: end function
```

---

## 5 Referências

### Referências

- ADAMY GIANINNI POR PIXABAY, Imagem de. **Centro histórico de Salvador, Bahia, Brasil.** [S.l.: s.n.], 2023. Disponível em: [https://pixabay.com/pt/users/adamygianinni-8676966/?utm\\_source=link-attribution&utm\\_medium=referral&utm\\_campaign=image&utm\\_content=3323275](https://pixabay.com/pt/users/adamygianinni-8676966/?utm_source=link-attribution&utm_medium=referral&utm_campaign=image&utm_content=3323275).
- ARBELAEZ, Pablo et al. Contour Detection and Hierarchical Image Segmentation. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 33, n. 5, p. 898–916, 2011. DOI: [10.1109/TPAMI.2010.161](https://doi.org/10.1109/TPAMI.2010.161).
- BISHOP, C.M. **Pattern Recognition and Machine Learning.** [S.l.]: Springer New York, 2016. (Information Science and Statistics). ISBN 9781493938438. Disponível em: <https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf>.
- COUPRIE, Camille et al. Indoor semantic segmentation using depth information. **arXiv preprint arXiv:1301.3572**, 2013.
- FERRANTI, Tomás. **Single image 3D building reconstruction using adjacent rectangles parallel to an axis.** [S.l.: s.n.], 2021. Trabalho de conclusão de curso. Acessado em 21 de novembro de 2023. Disponível em: <https://hdl.handle.net/10438/33214>.
- FISHER, R. et al. **Morphology.** [S.l.: s.n.], 2003. Disponível em: <https://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm>.
- GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. **Deep Learning.** [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.
- HARTLEY, Richard; ZISSELMAN, Andrew. **Multiple View Geometry in Computer Vision.** 2nd. New York, NY, USA: Cambridge University Press, 2004. ISBN 0521540518. Disponível em: <https://www.cambridge.org/core/books/multiple-view-geometry-in-computer-vision/0B6F289C78B2B23F596CAA76D3D43F7A>.

- ILLINGWORTH, John; KITTLER, Josef. A survey of the Hough transform. **Computer vision, graphics, and image processing**, Elsevier, v. 44, n. 1, p. 87–116, 1988.
- IMAGINERIO. **Situated Views of Rio de Janeiro**. [S.l.: s.n.], 2023. Instituto Moreira Salles. Acessado em 21 de novembro de 2023. Disponível em: <https://www.imaginario.org/pt>.
- ITSEEZ. **The OpenCV Reference Manual**. [S.l.], abr. 2014.
- ITSEEZ. **Open Source Computer Vision Library**. [S.l.: s.n.], 2015. <https://github.com/itseez/opencv>.
- MATAS, Jiri; GALAMBOS, Charles; KITTLER, Josef. Robust Detection of Lines Using the Progressive Probabilistic Hough Transform. **Comput. Vis. Image Underst.**, v. 78, p. 119–137, 2000. Disponível em: <https://api.semanticscholar.org/CorpusID:35118732>.
- OPENCV. **Hough Line Transform**. [S.l.: s.n.], 2023. [https://docs.opencv.org/3.4/d9/db0/tutorial\\_hough\\_lines.html](https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html).
- \_\_\_\_\_. **Morphological Transformations**. [S.l.: s.n.], 2023. [https://docs.opencv.org/3.4/d9/d61/tutorial\\_py\\_morphological\\_ops.html](https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html).
- \_\_\_\_\_. **Smoothing Images**. [S.l.: s.n.], 2023. [https://docs.opencv.org/3.4/dc/dd3/tutorial\\_gaussian\\_median\\_blur\\_bilateral\\_filter.html](https://docs.opencv.org/3.4/dc/dd3/tutorial_gaussian_median_blur_bilateral_filter.html).
- SHRIVAKSHAN, GT; CHANDRASEKAR, Chandramouli. A comparison of various edge detection techniques used in image processing. **International Journal of Computer Science Issues (IJCSI)**, Citeseer, v. 9, n. 5, p. 269, 2012.
- SIMONYAN, Karen; ZISSERMAN, Andrew. Very Deep Convolutional Networks for Large-Scale Image Recognition. **arXiv preprint arXiv:1409.1556**, 2014.
- XIE, Saining; TU, Zhuowen. **Holistically-Nested Edge Detection**. [S.l.: s.n.], 2015. arXiv: [1504.06375 \[cs.CV\]](https://arxiv.org/abs/1504.06375).