

Contagem de Vagas em Estacionamentos

Daniel Marcos Botelho Barbosa
17/0052427
DanielM.B.Barbosa@hotmail.com

Gabriel Filipe Botelho Barbosa
12/0050935
gabrielfbbarbosa@gmail.com

Abstract

Este projeto se caracteriza como uma pesquisa sobre um método de contagem de vagas na área da visão computacional sem a utilização de inteligência artificial.

Várias das atividades no campo de visão computacional se apoiam em atividades menores. Com isso, é bastante notório que o processo de contagem de vagas em um estacionamento tem uma grande abrangência no campo de estudos, não apenas da visão computacional. Técnicas como análise de texturas, detecção de gradiente, processamentos morfológicos, extração de características, clusterização e operações com histogramas estão bem presentes neste trabalho.

As tarefas realizadas envolvem desde abrir uma imagem simples do tipo BGR até as mais diversas técnicas para o desenvolvimento de uma solução da problemática. Para isso, a linguagem utilizada foi C++ com o padrão C++ 11 e a versão 3.2.0 do OpenCV.

1. Objetivos

O objetivo principal do desenvolvimento desse projeto é chegar à uma solução sobre um procedimento a ser seguido para se realizar um processo para a contagem das vagas sem utilizar técnicas de aprendizado de máquina. Além disso, a atividade como um todo objetiva a exploração dos conceitos aprendidos ao longo do curso de Princípios de Visão Computacional e ferramentas disponíveis na biblioteca em questão, OpenCV[5]. Com isso, tornar afim delas.

2. Introdução

Encontrar um espaço livre em estacionamentos de grandes áreas metropolitanas pode tornar-se cansativo. Além de estressante, essa tarefa desafiadora geralmente consome tempo e dinheiro consideráveis. Além disso, contribui para poluir o meio ambiente com emissões de CO₂. Tentando resolver esse problema, este projeto busca uma solução baseada em processamento de imagem.

Para se obter esse resultado da contagem, foi realizado um procedimento complexo desde detecção de gradiente até criação de algoritmos para determinar a similaridade de duas retas. Dos algoritmos pré-existentes que foram utilizados, o procedimento foi o seguinte:

Etapa 1 Abre-se uma imagem de estacionamento;

Etapa 2 Calcula-se diversas matrizes GLCM's;

Etapa 3 Aplica-se o algoritmo *Sobel Detector*[7];

Etapa 4 A partir de Sobel, aplica-se *Hough Line Transform*;

Etapa 5 A partir de Sobel, calcula-se a Transformada Probabilística de Hough;

Etapa 6 As retas obtidas na etapa 4 são clusterizadas e filtradas;

Etapa 7 Os segmentos de retas obtidos na etapa 5 são filtradas e clusterizadas de acordo com o obtido na etapa 6;

Etapa 8 Calcula a máscara da região a ser processada;

Etapa 9 Obtem o histograma da homogeneidade na região máscarada;

Etapa 10 Calcula a ocupação do estacionamento.

Após essas etapas, foram-se realizados procedimentos personalizados que serão descritos mais adiante.

2.1. Imagem

Os padrões de orientação e de pixelagem são tratados diferentemente pelo OpenCV.

A imagem utilizada para testes ao longo do projeto está representada em Figure 2.

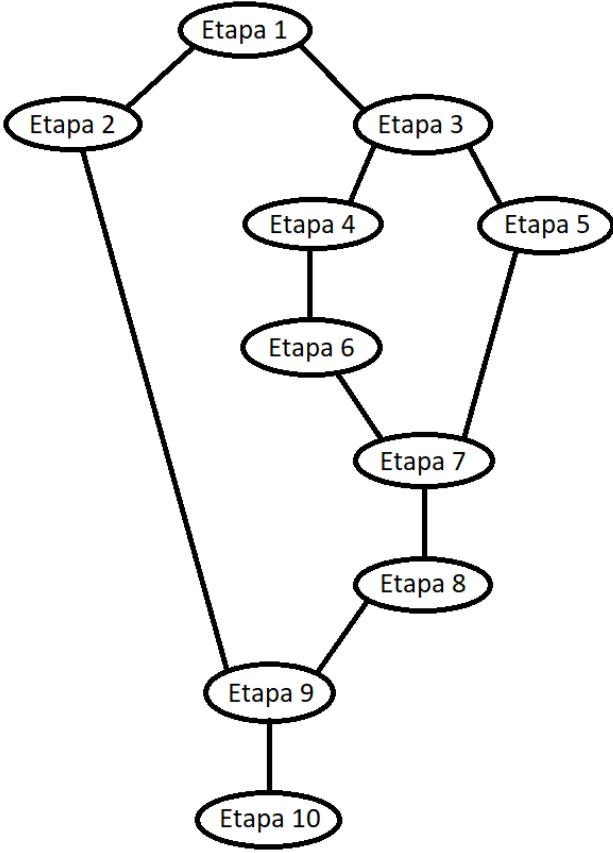


Figure 1. Diagrama em árvore da ordem e disposição das etapas realizadas



Figure 2. Imagem original de exemplo

2.1.1 Padrão RGB

Tanto imagens como vídeos são armazenados da mesma forma. A classe `cv::Mat` pode armazenar a matriz de pixels de uma imagem, bem como os frames de um vídeo.

Cada pixel tem seus canais armazenados, por padrão, na

ordem Vermelho (*Red*), Verde (*Green*) e Azul (*Blue*) (*RGB*). No entanto, a biblioteca em questão utiliza um padrão diferente, definido como *BGR*, invertendo a posição do valor do pixel azul com o vermelho.

É interessante converter a imagem do tipo *BGR* para *grayscale* porque diversos algoritmos usados não suportam imagens coloridas. Trabalhando com níveis de cinza só existe uma única informação a ser extraída: a intensidade do pixel, de 0 a 255. Todavia, ao trabalhar com imagem colorida, as informações triplicam. Três canais não relacionados aumentam o nível de complexidade por não terem uma ordenação linear que seja fácil de colocar em uma matriz, e que causam um *overhead* desnecessário sem nenhum ganho de precisão.

2.1.2 Coordenadas

O padrão de referências cartesianas computacionais é com origem no extremo noroeste da tela. Com eixo x crescendo para a direita e eixo y crescendo para baixo. No entanto, o padrão adotado pela biblioteca é situada, também, com origem no extremo norte-oeste da tela, porém com eixo x crescendo para baixo e eixo y crescendo para a direita.

2.1.3 Acesso dos dados

Para realizar o acesso dos dados matriciais da classe `cv::Mat`, utiliza-se, recomendada pela documentação[1], o método `cv::Mat::at<type T>(cv::Point(j, i))`.

Desse modo, como a imagem, neste ponto, está em níveis de cinza, o retorno do método `at<unsigned char>(j, i)`, em cada pixel (i, j), será um valor de 0 a 255 indicando a intensidade de pixel para cada pixel analisado.

3. Trabalhos relevantes

O artigo “*Car Parking Occupancy Detection Using Smart Camera Networks and Deep Learning*” é um trabalho de contagem de vagas em estacionamento utilizando, além do processamento de imagem, redes de aprendizagem profunda. Este artigo apresenta uma abordagem para a detecção de ocupação de estacionamento em tempo real que usa um classificador da Rede Neural Convolucional (CNN) que funciona a bordo de uma câmera inteligente com recursos limitados.

Com isso, decidiu-se explorar a viabilidade de realizar o processo de contagem de vagas de um estacionamento sem a utilização de redes neurais. Portanto, este projeto usa apenas processamento de imagens para a obtenção dos resultados almejados.

4. Metodologia proposta

Para realizar a contagem de vagas de estacionamento, utilizou-se de um procedimento um tanto empírico para atingir os resultados. Ao receber a imagem, utilizou-se de análise de texturas para extrair as matrizes de co-ocorrência, detecção de gradiente como um pré-processamento para reduzir informações inúteis, processamentos morfológicos, extrações de características, clusterizações e filtragens.

Nas duas últimas etapas da metodologia proposta, foram realizados alguns procedimentos já existentes, além de algoritmos próprios. Eles foram responsáveis por realizar a última etapa antes da segmentação para a contagem das vagas.

4.1. Análise de texturas

As medidas de textura de co-ocorrência de nível de cinza têm sido a força de trabalho da textura da imagem desde que foram propostas por Haralick[11] na década de 1970.

Uma matriz de co-ocorrência, ou distribuição de co-ocorrência,[10] é uma matriz que é definida sobre uma imagem para ser a distribuição de valores de pixel co-ocorrentes (valores de tons de cinza ou cores) a um dado deslocamento. Esta matriz aproxima a distribuição de probabilidade conjunta de um par de pixels.

Ela representa a relação entre distância e relação de angulação espacial sobre uma sub-relação de imagem de uma região específica e de tamanho específico. Ou seja, com essa matriz de coocorrência é possível detectar, de certa forma, a textura de objetos capturados em uma imagem. Neste projeto terão quatro matrizes finais que serão obtidas pelas direções 0° , 45° , 90° e 135° , em ambos sentidos, com offsets de 3 pixels.

As GLCM's não foram computadas na imagem toda, entretanto, em blocos de 32×32 pixels, deslocados de 8 pixels em ambos os eixos para cada iteração. O que significa que, por exemplo, cada uma das GLCM's seriam de 32×32 e começariam no pixel $(0, 0)$, depois $(0, 8)$, $(0, 16)$ e assim por diante.

A homogeneidade de cada uma dessas matrizes foi obtida e salva numa matriz que representa onde da imagem original aquele valor foi computado de tal forma que, para se obter a homogeneidade calculada para um bloco como $(8, 8, 40, 40)$, basta checar essa matriz final na posição $(1, 1)$.

4.2. Detecção de gradiente

O pré-processamento da imagem foi realizado com o algoritmo de Sobel.

Sobel O detector de bordas de Sobel[6] se baseia em algumas alterações e adaptações do operador matemático Laplaciano[3] para um espaço dimensional discreto e



Figure 3. GLCM $3\angle 0^\circ$



Figure 4. GLCM $3\angle 45^\circ$



Figure 5. GLCM $3\angle 90^\circ$



Figure 6. GLCM $3\angle 135^\circ$

bidimensional[4]. O efeito desse operador no espectro matemático é o mesmo quando aplicado numa imagem, ou seja, ele ameniza variações de baixa frequência e atenua variações de alta frequência.[6]

Ele realiza esse procedimento realizando uma convolução nos eixos da imagem usando um *kernel* direcional e realizando a média dos resultados. Ele é, geralmente, quadrado com tamanho ímpar, comumente 3, onde a soma de todos os elementos é 0.

-1	0	1
-2	0	2
-1	0	1

Figure 7. Operador Sobel horizontal para imagens bidimensionais

Além disso, se considerarmos o *kernel* utilizado para computador o gradiente horizontal (Figure 7), ou seja, na coordenada x , todos os elementos da coluna central será 0. Todos os *kernels* não são nada além de uma rotação ao redor do pixel central de algum outro.

Com Sobel, obtemos o gradiente em x , o gradiente em y , o ângulo da borda e a intensidade da transição mas, deste ponto em diante, somente a magnitude (Figure 8) será considerada no processamento.

4.3. Processamento morfológico

4.3.1 Limiarização e Binarização

A binarização realizada após as detecções serem finalizadas foi trivial, mas merece ser rapidamente explicada pois o valor final não foi 0 ou 1. Para que fosse possível se visualizar a binarização, ela foi escalada de forma que 1 seja 255 mas que não possua nenhum outro número entre 255 e 0. Ou seja, todo pixel maior que um *threshold* determinado seria setado como 255, e 0 caso contrário.

4.3.2 Dilatação e Erosão[8]

Filtros morfológicos exploram as propriedades geométricas dos sinais (níveis de cinza da imagem). Para filtros morfológicos, as máscaras são denominadas elementos estruturantes e apresentam valores 0 ou 1 na matriz que correspondem ao pixel considerado. Os filtros morfológicos básicos são o filtro da mediana, erosão e dilatação. No entanto, utilizou-se apenas a dilatação e a erosão.

Dilatação provoca efeitos de dilatação das partes escuras da imagem (baixos níveis de cinza), gerando imagens mais claras.

Erosão provoca efeitos de erosão das partes claras da imagem (altos níveis de cinza), gerando imagens mais escuras.

Foi-se realizado primeiro uma abertura mas com *kernels* diferentes para cada etapa, por isso preferiu-se descrevê-los como suas etapas separadas. Essas operações foram feitas pois o detector de borda computava duas linhas (uma borda para cada lado) nas laterais das faixas divisórias das vagas e era desejado uní-las em uma única linha.

Na etapa da dilatação, usou-se um *kernel* circular de diâmetro 7 pixels. Isso ajudou a conectar várias linhas



Figure 8. Matriz magnitude obtida ao realizar a detecção de Sobel

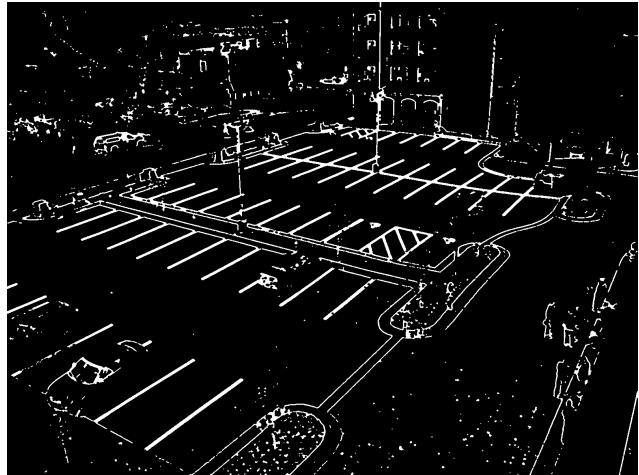


Figure 9. Imagem após realizada todos os processamentos morfológicos

que talvez tivessem sido desconectadas com o processo da limiarização.

A etapa de erosão utilizou um *kernel* também circular mas de diâmetro 5 pixels.

A decisão de se fazer assim foi de que, apesar vários pontos continuarem visíveis, as linhas permaneceriam conectadas e com um tamanho relativamente próximo ao da imagem original. Uma das vantagens advindas desse processo é que facilita e aumenta a quantidade de linhas geradas na próxima etapa e, com isso, aumentando a discrepância na quantidade de linhas geradas para cada direção e local da imagem.

4.4. Extração de características

4.4.1 Hough

A transformação de Hough[2] é uma técnica que pode ser usada para isolar características de uma determinada forma dentro de uma imagem. Pelo fato de necessitar que as características desejadas sejam especificadas em alguma forma paramétrica, a transformação Hough clássica é mais comumente usada para a detecção de curvas regulares, como linhas, círculos, elipses, etc. A transformada Hough generalizada pode ser empregada em aplicações onde uma descrição analítica simples de uma ou mais características não é possível. Apesar das suas restrições de domínio, a transformada clássica de Hough contém muitas aplicações.

A vantagem principal da transformada de Hough é que é uma técnica que não é afetada por espaços entre as características obtidas por detectores de borda e é relativamente invariante ao ruído nas imagens. A ideia de se utilizar esta transformada é para encontrar as linhas que dividem cada vaga.

Claro que, como as linhas são bem grossas, o algoritmo iria encontrar centenas de retas para cada linha divisória de

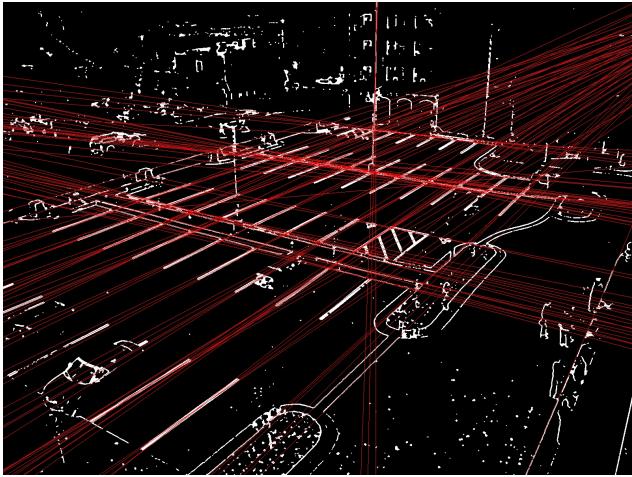


Figure 10. Retas encontradas pela Transformada de Hough

vagas, e esse era o objetivo. Usando um limiar de 300 para o acumulador de Hough, foi possível obter todas as retas mostrada na Figure 10. Uma rápida inspeção já nos permite perceber que as linhas detectadas, em sua grande maioria, apontam para os pontos de fuga da projeção da câmera. Isso significa que a distância entre as retas e a diferença entre seus ângulos não serão constantes por toda a imagem.

4.4.2 Probabilistic Hough Transform

Na transformada de Hough, percebe-se que mesmo para uma linha com dois argumentos é preciso muito processamento. A Transformada probabilística de Hough[12] é uma otimização da transformada clássica de Hough. Não leva todos os pontos em consideração, em vez disso, toma apenas um subconjunto aleatório de pontos que é suficiente para a detecção da linha. Apenas temos que definir o limiar.

Com esse processamento, podemos obter as linhas das vagas individualmente, ao invés de somente a sua orientação. Infelizmente isso vem com o custo de que outras linhas além das faixas divisórias das vagas também serão obtidas e usadas no cálculo.

4.5. Agrupamento

Clusterização é uma técnica que é usada para partitionar elementos em um conjunto de dados, de modo que elementos semelhantes sejam atribuídos ao mesmo cluster, enquanto elementos com propriedades diferentes são atribuídos a diferentes clusters. É utilizado para executar uma pesquisa eficiente de elementos em um conjunto de dados. O agrupamento é particularmente eficaz em dados multidimensionais que, de outra forma, podem ser difíceis de organizar de maneira efetiva.

Uma das primeiras técnicas de agrupamento na literatura é o método de *clustering* K-means. Nesta técnica, o agrupamento é baseado na identificação de elementos K

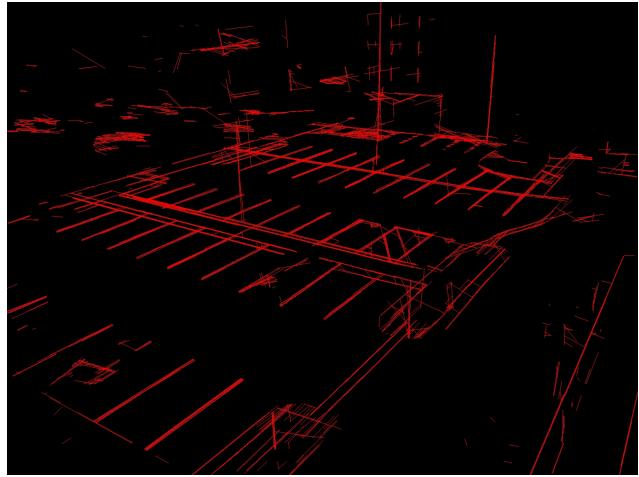


Figure 11. Segmento de Retas encontradas pela Transformada Probabilística de Hough

no conjunto de dados que podem ser usados para criar uma representação inicial de clusters. Esses elementos K formam as sementes do cluster. Os elementos restantes no conjunto de dados são então atribuídos a um desses clusters. Mesmo que o método pareça direto, sofre o fato de que pode não ser fácil identificar claramente os elementos K iniciais ou as sementes para os clusters.

Como não sabemos quantos núcleos K se precisaria, foi-se utilizado uma variação deste algoritmo. A clusterização K-means adaptativo por limiarização [9].

K-means foi utilizado para que as linhas retas ou segmentos de retas semelhantes geradas pelas transformadas Hough sejam agrupadas. Com isso, a contagem tornou-se algo já possível de se enxergar.

Um impedimento que foi encontrado ao se utilizar esse método é que fazia-se necessário existir uma forma de se classificar a similaridade entre duas retas, e de segmentos de retas. Esse não é um problema resolvido da matemática e, portanto, existem algumas possíveis formas de se ter esse valor computado. Usamos um método próprio e relativamente simples discutido a seguir.

4.5.1 Similaridade de retas

Para a aplicação da clusterização pelo método de K-means, criou-se um algoritmo para definir a similaridade entre as retas encontradas pela Transformada Hough. Esse algoritmo recebe como entrada duas retas, ou dois segmentos de retas, e retorna um número entre 0 a 1 que representa a similaridade entre elas de forma que, 1 significa que são identicas e 0 indica que não são correlacionadas.

Para que seja uma transformada linear, a função também recebe argumentos que representam qual o valor mínimo que alguma diferença pode ser para fazer com que a similaridade seja 0. Por exemplo, duas linhas paralelas cuja

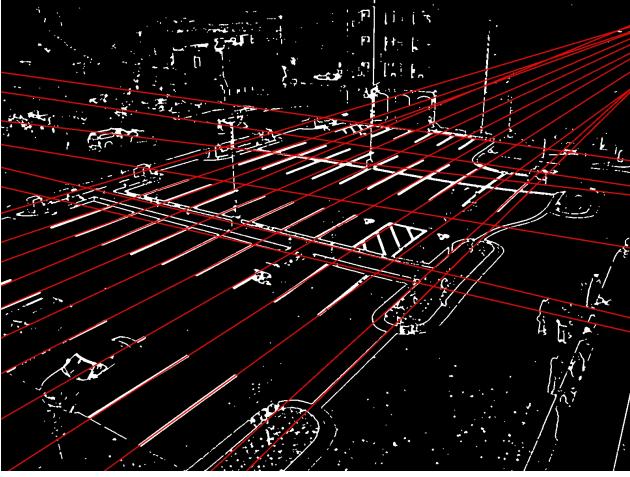


Figure 12. Retas geradas por Hough depois da clusterização e filtragem

distância máxima seja 10 pixels retornaria 0.2 de similaridade de a distância entre elas fossem de 8 pixels. A equação abaixo representa o cálculo realizado para se computar a similaridade entre retas.

$$\begin{aligned}
 angleSimilarity &= 1 - \frac{|\theta_1 - \theta_2|}{\pi/2}, \forall \theta \in [0, \pi/2] \\
 distanceSimilarity &= 1 - \frac{|\rho_1 - \rho_2|}{maxDistance}, \\
 \forall |\rho_1 - \rho_2| < maxDistance; \\
 &\quad = 0, otherwise \\
 similarity &= angleSimilarity * distanceSimilarity
 \end{aligned} \tag{1}$$

A diferença desse algoritmo para o de similaridades entre segmentos de retas é que o valor final é multiplicado por um outro *distanceSimilarity* adicional que computa a distância entre o centro de dois segmentos de retas dentro de um limite máximo, assim como o demonstrado acima.

4.6. Filtragem

Para se obter um melhor resultados, todas as linhas obtidas foram fitradas. Contudo, cada tipo de linha passou por uma filtragem diferente.

As retas obtidas pela transformada de Hough, após serem clusterizadas, foi-se realizado uma segunda clusterização sobre os centroides previamente encontrados. Todos esses centroides iniciais que ficaram ligados a um novo centroide médio que, ao final do processamento, possuísse menos de dois centroides, foram removidos da lista de centroides. O que resulta em somente linhas de média que possuem três ou mais similares.

Já os segmentos de retas obtidos pela transformada probabilística de Hoguh tiveram um processamento diferente.

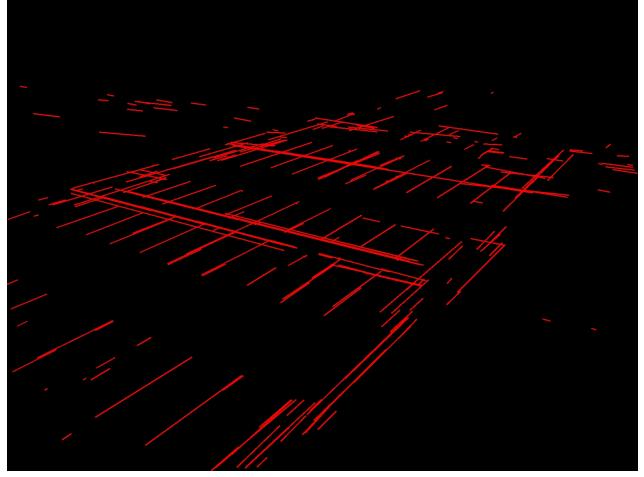


Figure 13. Segmentos de retas geradas por Hough probabilístico depois da clusterização e filtragem

Primeiro foi realizado uma filtragem para se remover todo e qualquer segmento de reta que não possuísse angulação similar e/ou estivesse acima de 50 pixels distante dos centroides encontrados na etapa anterior. Somente nesse ponto, os segmentos de retas restantes seriam clusterizados, resultando na Figure 13.

4.7. Mascaramento

Usando-se os segmentos de retas filtrados e clusterizados da etapa anterior como imagem, foi realizada mais três processamentos morfológicos para se obter a máscara. Duas dilatações com tamanhos diferentes que foram salvas em variáveis temporárias e uma subtração da maior pela menor. O resultado desta etapa foi uma imagem onde os únicos pixels em branco são os que estão relativamente próximo a algum segmento de reta obtido. A figura 14 monstra como é o resultado obtido nesta etapa.

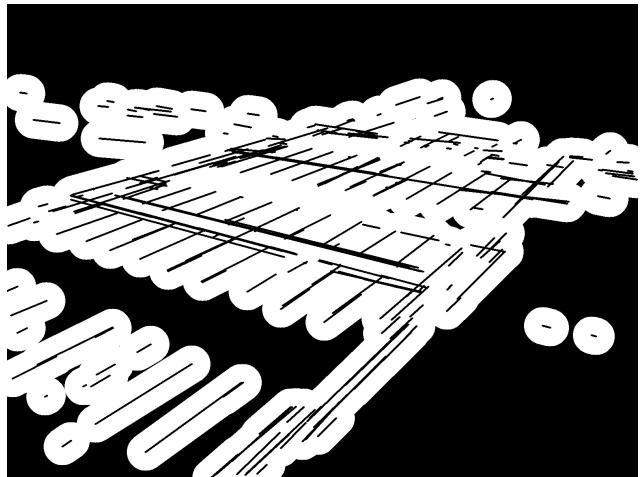


Figure 14. Mascaramento da imagem

4.8. Histogramização

Criou-se um vetor que armazenava todas as homogeneidades médias, ou seja, a média das homogeneidades obtidas pelas matrizes GLCM 0°, 45°, 90° e 135° para cada pixel, onde só eram adicionados nesse vetor as homogeneidades médias cujas janelas para sua computação estivesse completamente dentro da máscara obtida. Desta forma, este vetor possuía somente as homogeneidades computadas em posições relevantes.

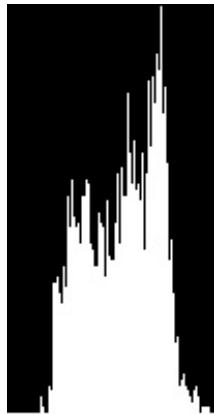


Figure 15. Histograma obtido para a imagem de exemplo

4.9. Computação da Ocupação

Nessa etapa, computa-se dois valores simultaneamente. Quantas vagas existem e o quanto do estacionamento está ocupado. Os valores não são exatos, são somente estimativas.

Para calcular quantas vagas foram encontradas, usamos as retas clusterizadas e filtradas encontradas. Aos reunimos através de uma clusterização de k-médias simples, ou seja, não adaptativo, e informando 2 como a quantidade de médias desejadas. Então, multiplicamos a quantidade de retas subtraído por um que uma média foi associada pela mesma medição da outra média.

Para se obter a estimativa de ocupação do estacionamento, procuramos pelo valor de homogeneidade de maior ocorrência com o auxílio do histograma. Partimos também do pressuposto que um estacionamento vazio não possui homogeneidade de 1 mas sim algum valor menor, e que um estacionamento lotado não possui homogeneidade de 0 mas sim de algum valor maior. Neste ponto, tivemos duas abordagens, uma de calcular a distância deste pico do histograma para os extremos de homogeneidade possíveis de classificar como uma porcentagem. A outra forma de se calcular a porcentagem de ocupação foi computar quantas ocorrências de homogeneidade foram encontradas depois do pico e dividir esse valor pela quantidade total de homogeneidade sendo computados.

5. Resultados

A figura 16 exibe os resultados que obtivemos para o teste de exemplo usado ao longo deste relatório. Ao se realizar uma contagem manual das vagas, encontramos 58 vagas (4 não visíveis, 2 parcialmente visíveis e 52 perfeitamente visíveis) sendo que dessas 58 vagas, 4 estão ocupadas. Isso resulta em uma ocupação de 6.89% se considerarmos todas as vagas, 7.4% se considerarmos somente as visíveis e 7.69% se considerarmos somente as vagas perfeitamente visíveis.

```
Computando GLCMs:  
0%... 50%... 100%  
GLCMs computadas. Calculando Transformada de Hough...  
Hough computado. Calculando Hough probabilístico...  
Hough probabilístico pronto. Computando ocupacao...  
Pronto!  
  
Encontramos 55 vagas.  
Estimamos que esteja entre 10.5% e 10.0% ocupado  
  
PS D:\Projetos\PVC\PVC-ContagemDeVagas> █
```

Figure 16. Print do console ao executar aplicação desenvolvida

Paralelização O código desse projeto foi implementado de forma completamente monolítica e sequencial, entretanto, ele é extremamente paralelizável. Isso se deve ao fato de que, de todos os processamentos realizados, somente um deles depende dos resultados obtidos por outros procedimentos pesados. Essa etapa é a da filtragem e clusterização que é relativamente rápida mas depende da transformada probabilística de Hough (que devido aos parâmetros e condições da imagem, podem levar até um minuto para ser computado).

Oclusões Uma desvantagem da técnica desenvolvida é sua falta de robustez com relação a oclusões. Todo e qualquer tipo de oclusão pode alterar o resultado obtido. Sejam árvores, pessoas ou até mesmo carros. Além do fato de que, vagas com marcações, como por exemplo vagas para deficientes, vagas reservadas, vagas para idosos, e afins, inflam nem que parcialmente o resultado da ocupação do estacionamento mesmo se estiverem desocupadas.

6. Conclusões

Considerando-se que houve a computação da ocupação de vagas em alguns poucos pontos fora do estacionamento e a baixa resistência a oclusões, podemos afirmar que a precisão do procedimento desenvolvido não é boa para se afirmar com segurança o valor encontrado, mas ainda assim o resultado foi de agrado, pois permitiu ter pelo menos uma estimativa de certa forma próxima aos valores reais.

Ao final desse projeto, percebeu-se que esta técnica é inviável para ser usada em casos que exigem precisão, a

não ser que seja utilizado algum processamento para dar resistência a oclusões. Além de que, é necessário se criar algum procedimento para se calcular e segmentar as vagas pois, da forma atual, está sendo computado a homogeneidade dos pixels próximos aos segmentos de retas detectadas e, através do uso de histograma, comparando-se os picos e se obtendo a relação ocupado/desocupado. E acredita-se que sejam esses os pontos que mais prejudicam a precisão dos cálculos.

References

- [1] Devdocs. <http://devdocs.io/>.
- [2] Hough transform. <http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm>.
- [3] Laplace operator. http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/laplace_operator/laplace_operator.html.
- [4] Laplacian edge detector. https://de.mathworks.com/matlabcentral/answers/uploaded_files/51086/Edge%20detection%20-%20Laplace%20edge%20detection%20algorithm.pdf. Accessed: 20/06/2017 - 21/06/2017.
- [5] OpenCV 3.2.0 documentation. <http://docs.opencv.org/3.2.0/>.
- [6] Sobel derivatives. http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/sobel_derivatives/sobel_derivatives.html.
- [7] Sobel edge detector. <http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm>. Accessed: 20/06/2017 - 21/06/2017.
- [8] Teoria: Processamento de imagens. <http://www.dpi.inpe.br/spring/teoria/filtrage/filtragem.htm>, Fevereiro, 2000.
- [9] S. K. Bhatia. *Adaptive K-Means Clustering*.
- [10] M. Hall-Beyer. The glcm tutorial. <http://www.fp.ucalgary.ca/mhallbey/tutorial.htm>, Fevereiro, 2007.
- [11] R. M. Haralick, K. Shanmugam, and I. Dinstein. Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3(6):610–621, Dezembro, 1973.
- [12] I. Macdonald. Probabilistic hough transform. http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV1011/macdonald.pdf, Abril, 2011.