João Henrique de Oliveira Júnior

Reconhecimento de caracteres de placa veícular fazendo uso de visão computacional

Araquari – SC

Setembro de 2015

João Henrique de Oliveira Júnior

Reconhecimento de caracteres de placa veícular fazendo uso de visão computacional

Trabalho de conclusão de curso apresentado ao Instituto Federal Catarinense - Câmpus Araquari como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação sob orientação do professor Prof. Dr. Eduardo da Silva.

Instituto Federal Catarinense – IFC

Câmpus Araquari

Bacharelado em Sistemas de Informação

Orientador: Prof. Dr. Eduardo da Silva

Coorientador: Prof. Dr. Paulo César F. de Oliveira

Araquari – SC

Setembro de 2015

João Henrique de Oliveira Júnior
Reconhecimento de caracteres de placa veícular fazendo uso de visão computacional/ João Henrique de Oliveira Júnior. – Araquari – SC, Setembro de 2015-
Orientador: Prof. Dr. Eduardo da Silva

Monografia (Graduação) – Instituto Federal Catarinense – IFC
Câmpus Araquari
Bacharelado em Sistemas de Informação, Setembro de 2015.

1. Visão computacional. 2. OCR. 2. Inteligência artificial. I. Orientador. II. Instituto Federal Catarinense. III. Câmpus Araquari.

João Henrique de Oliveira Júnior

Reconhecimento de caracteres de placa veícular fazendo uso de visão computacional

Trabalho de conclusão de curso apresentado ao Instituto Federal Catarinense - Câmpus Araquari como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação sob orientação do professor Prof. Dr. Eduardo da Silva.

Trabalho aprovado. Araquari – SC, 24 de novembro de 2012:

Prof. Dr. Eduardo da Silva
Orientador

Prof. Dr. Paulo César F. de Oliveira
Coorientador

Professor
Convidado 1

Professor
Convidado 2

Araquari – SC
Setembro de 2015

Dedico esse trabalho de conclusão de curso a Deus
pois Ele é digno de mais essa vitória.

Agradecimentos

Agradeço a Jesus Cristo, a razão da minha existência, sem Ele nada eu teria ou seria. Agradeço também a minha família que sempre me apoiou, e a minha noiva Nicole, presente do Senhor para minha vida. Agradeço também ao orientador Eduardo da Silva por ter acreditado nesse trabalho.

"E tudo quanto fizerdes, fazei-o de todo o coração, como para o Senhor e não para homens"(Bíblia sagrada)

Resumo

Este trabalho trata-se da implementação de uma solução para reconhecimento de placa automotiva. Partindo do princípio que um computador não pode interpretar imagens como um ser humano, torna-se necessária a implementação de códigos que busquem emular a visão humana. Todo sistema que busca tal objetivo é chamado de sistema de visão computacional, que é uma área da inteligência artificial. Para a conclusão deste trabalho, foram utilizadas as bibliotecas OpenCV 3.0, GOCR e OCRAD, sendo que o OpenCV trata-se de uma biblioteca de visão computacional, e a GOCR e OCRAD são bibliotecas de OCR(Optical character recognition). O resultado obtido desse trabalho é de 50% de taxa de acerto devido a questões que são exploradas no capítulo 4 e na conclusão, podendo ser implementado por qualquer outro pesquisador. Todas as soluções são de licença gratuita, incluindo a plataforma onde a solução é executada.

Palavras-chave: OCR, Visão Computacional, Inteligência Artificial, OpenCV, GOCR, OCRAD, caracter.

Abstract

This work envisages the implementation of licence plate recognition solution. Assuming that a computer is not able to interpret images as a human, it is necessary an implementation of codes that try to emulate the human vision. All system that has this goal is called of computer vision system, that is an area of artificial intelligence. For the conclusion of this work, the libraries OpenCV 3.0, GOCR and OCRAD were used, in which the OpenCV is a library of computer vision, while the GOCR and OCRAD are OCR(Optical character recognition) libraries. The obtained result in this work, is of 50% of hit ratio due to some arguments that will be explored in the chapter 4 and in the conclusion, which may be implemented for any other research. All the solutions has free licence, including the platform where the solution is executing.

Keywords: OCR, Computer vision, Artificial intelligence, OpenCV, GOCR, OCRAD, character.

Lista de ilustrações

Lista de tabelas

Lista de Códigos

Lista de abreviaturas e siglas

HSV	Hue Saturation Value
OCR	Optical Character Recognition
RGB	Red Green Blue
OpenCV	Open Computer Vision
OffOCR	Offline Optical Character Recognition
OnOCR	Online Optical Character Recognition

Sumário

1 Introdução

Imagens e vídeos de forma geral podem conter uma quantidade significativa de informações valiosas, que com a ajuda da tecnologia, há possibilidade de serem extraídas em forma de texto de maneira automática. Essas informações registradas e processadas têm utilidade para os mais variados objetivos, como pesquisa, consultas ou até mesmo questões mais específicas. Entretanto, para que isso aconteça, é necessária uma série de técnicas como por exemplo detecção, localização, aprimoramento, extração e reconhecimento de texto (**??**).

Existe uma gama de aplicações que utilizam essas técnicas citadas, fazendo uso de bibliotecas de visão computacional para linguagens de programação. Algumas dessas aplicações são bem conhecidas, como a biometria por impressões digitais ou pela íris do olho humano, e o reconhecimento facial, encontrado em muitas redes sociais atuais (**??**). Existem também certas soluções para reconhecimento ótico de caracteres, geralmente utilizados em *scanners*. Essas soluções são conhecidas como OCRs (*Optical Character Recognition*) sendo também parte do conceito de visão computacional.

Diversas bibliotecas OCRs estão disponíveis para uso, como por exemplo a Tesseract-OCR (**??**), a GOCR (**??**), a OCRAD (**??**) e outras. Todas são bibliotecas de reconhecimento de texto com licença gratuita. A Tesseract, GOCR e OCRAD compõe grande parte deste trabalho.

O problema é que não existem meios de um sistema de computador interpretar uma imagem da mesma forma que um ser humano (**??**). Por esse motivo, tem-se a necessidade do uso de técnicas, pois a função básica realizada por um computador é a execução de programas, que consiste em um conjunto de instruções armazenadas na memória (**??**), e a visão computacional procura integrar as áreas de processamento digital de imagens e inteligência artificial, tendo como objetivo a obtenção de algoritmos capazes de interpretar o conteúdo visual das imagens (**??**). Um exemplo seria a leitura de placas de veículos, pois facilmente um ser humano consegue identificar as letras e números contidos na placa, entretanto um computador sem um algoritmo especializado não.

Existem alguns softwares e projetos que buscam fazer o reconhecimento de placas automotivas como (**??**), (**??**), (**??**), (**??**), (**??**), (**??**). Um dos que mais são conhecidos chama-se OpenALPR, que traz bons resultados de precisão, e também é capaz de encontrar uma placa em meio a um cenário, para então extrair as informações. Este software, que ainda está em desenvolvimento, possui uma comunidade ampla de desenvolvedores, não contendo ainda uma documentação formal disponível ao público. Há uma versão do projeto do software com licença AGPL e também versões comerciais, contudo ele reconhece apenas

placas norte americanas e europeias (**??**).

Este trabalho resume-se em reconhecer caracteres de imagens de placas de licenciamento de veículos brasileiros, transformando-os em texto simples. Assim podem ser aplicadas ações sobre este texto como consultas na base fiscal do Denatran. No âmbito de reconhecimento de placas brasileiras, **??**) apresenta uma solução que possui seis processos, observados na Figura 1, são esses:

a) Captura: Submete a imagem que será a entrada do sistema. A imagem pode ser obtida através de uma câmera *webcan* ou qualquer dispositivo semelhante.

b) Pré-processamento: Trata a imagem para que as condições da localização da placa aumentem.

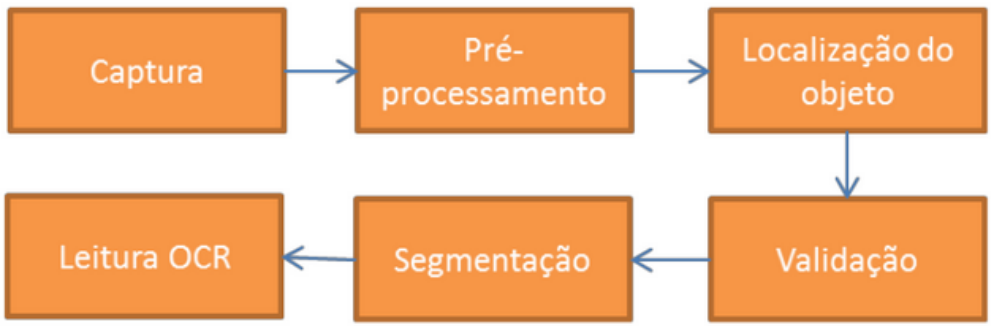
c) Localização: Identifica as bordas na imagem.

d) Validação: Encontra por definitivo a placa.

e) Segmentação: Corta a imagem da placa encontrada no processo anterior.

f) Leitura OCR: Trata a imagem recebida da etapa de segmentação. Passar por um leitor OCR identificando os caracteres contidos e retornando em forma de texto simples.

Figura 1 – Diagrama de sistema de detecção de placa automotiva



Fonte: **??**)

O autor fez uso da biblioteca Tesseract-OCR (**??**), mostrando um desempenho consideravelmente bom, dependendo das circunstâncias em que se encontram as etapas de validação e segmentação. Porém, foram necessários treinamentos do Tesseract-OCR, que foram considerados “trabalhosos” pelo autor. Certos testes resultaram em falhas, devido a alguns problemas que serão citados ao longo do trabalho.

Este trabalho então, tem como proposta principal implementar uma solução semelhante com o uso das bibliotecas de OCR "GOCR"e "OCRAD", testando dessa forma se é

possível manter o desempenho em relação a biblioteca Tessaract-OCR sem a utilização de treinamentos. Vale enfatizar que o fato de utilizar diferentes OCRs afeta diretamente no resultado, podendo ser melhor ou não.

O restante desse trabalho está dividido em cinco capítulos. O capítulo 2 trata a respeito do que é visão computacional e OCR, explicando conceitos de forma geral e apresentando certas aplicações que o utilizam. O capítulo 3 explora a ferramenta OpenCV e suas funcionalidades dando ênfase nas que serão utilizadas para a solução, também serão vistas algumas questões sobre o GOCR o OCRAD e o Tesseract, e para conclusão do capítulo serão abordados conceitos básicos de processamento de imagens. O Capítulo 4 aborda o desenvolvimento. O capítulo 5 trata-se da conclusão.

2 O que é visão computacional e OCR?

Este capítulo apresenta os conceitos básicos de visão computacional e OCR. Não serão abordados a fundo como ambos funcionam, mas sim para que e onde são utilizados.

2.1 Visão computacional

A Visão Computacional é um campo de estudo da computação que tem por objetivo tornar possível um computador ver e analisar imagens, extraindo informações úteis de componentes como uma câmera de vídeo, *scanners* e dispositivos semelhantes (**??**). Em outras palavras, visão computacional tem o objetivo final de usar computadores para emular a visão humana, incluindo a aprendizagem e a capacidade de fazer inferências e tomar decisões baseadas nas entradas fornecidas. Essa área é um ramo que pertence à inteligência artificial, que por sua vez tem o objetivo central de emular a inteligência humana (**??**). A análise de uma imagem está entre o processamento de imagens e a visão computacional (**??**).

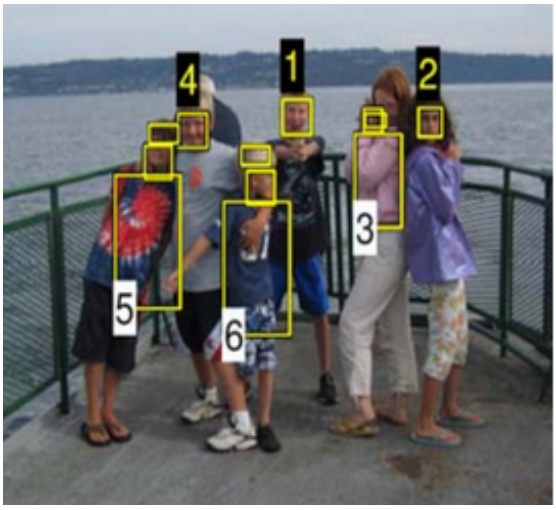
Entre processamento de imagens e visão computacional podem ser encontrados três níveis de processos, respectivamente, baixo-nível, nível-médio e alto-nível. Os processos de baixo-nível envolvem operações primárias, tais como a redução de ruído ou melhoria no contraste de uma imagem. Os processos de nível-médio são operações do tipo segmentação (particionamento da imagem em regiões) ou classificação (reconhecimento dos objetos na imagem). Os processos de alto-nível estão relacionados com as tarefas de cognição associadas com a visão humana (**??**).

Olhando para um grupo de pessoas em um porta-retratos, os seres humanos são capazes de identificar cada pessoa, reconhecer o nome e, através da expressão facial, até mesmo obter conhecimento do comportamento sentimental da pessoa naquele momento. Estudiosos passaram décadas tentando compreender como o sistema visual funciona, e chegaram a bons resultados em certas áreas como reconhecimento de texto. Mas ainda há muito para evoluir como na inteligência artificial de uma forma geral (**??**) (**??**).

Atualmente, com os avanços em visão computacional, pode-se identificar com alta probabilidade pessoas em meio a um cenário complexo, ou seja, que contenha diversos outros objetos incluídos, como árvores, carros, animais e outros, e dar-lhes os respectivos nomes. A Figura 2 ilustra a detecção de seis pessoas dentro de um barco, e a Figura 3 ilustra um sistema capaz de reconhecer uma pessoa em meio a um cenário que contém árvores, prédio, cadeiras e outros objetos, circulando a cabeça, os braços, mãos e pernas. Entretanto, não é possível dizer ainda que visão computacional consegue emular a visão

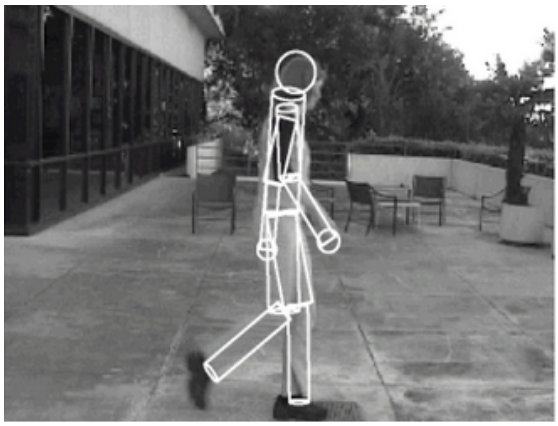
de uma criança de dois anos, devido principalmente, ao grande número de incógnitas para poucos dados (**??**).

Figura 2 – Reconhecimento de seis pessoas em um barco



Fonte:**??**)

Figura 3 – Reconhecendo e circulando pessoa em meio a cenário complexo



Fonte: **??**)

A visão computacional está sendo utilizada hoje em diversos meios, como (**??**):

a) no reconhecimento óptico de caracteres (OCR) aplicado a outros sistemas;

b) em sistemas de inspeção da máquina, em que são realizadas inspeções para garantia de qualidade, medindo as tolerâncias das asas de aviões, ou procurando defeitos em peças fundidas de aço, utilizando raios-X;

c) em construção de modelos 3D (fotogrametria) de forma totalmente automatizada a partir de fotografias aéreas utilizadas em sistemas como o Bing Maps (**??**);

d) na radiologia é utilizado para registrar imagens pré-operatórias e intra-operatórias, ou na realização de estudos de longo prazo da morfologia do cérebro das pessoas à medida que envelhecem;

e) na área de segurança automotiva, tem-se o sistema de detecção de obstáculos inesperados, como pedestres na rua;

f) em jogos de movimentos, fundindo imagens geradas por computador com cenas de ação ao vivo por rastreamento de pontos característicos no vídeo fonte. Tais técnicas são amplamente utilizadas em Hollywood (por exemplo, filme Jurassic Park) (**??**);

g) em autenticação visual com o uso da *webcan* para autenticação de usuário;

h) em câmeras digitais através da detecção facial, auxiliando a câmera digital a focar na região correta;

i) nos bancos, onde pode-se fazer autenticação por biometria, sendo as medidas biometricas a chave de segurança;

2.2 Reconhecimento ótico de caracteres

Reconhecimento ótico de caracteres é uma das áreas que envolve visão computacional como analisado anteriormente. **??**) faz a seguinte definição de OCR:

> O processo de reconhecimento óptico de caracteres (OCR) envolve o uso de sistemas de imagem e computadores para converter texto manuscrito ou impresso em texto que a máquina é capaz de codificar e reconhecer. OCR possui muitos usos, como a simplificação de entrada de dados, reconhecimento de placa de licença de veículos, e disponibilização de cópias editáveis de documentos impressos (**??**).

De maneira comercial, alguns sistemas OCRs começaram a surgir por volta de 1960, porém esses sistemas possuíam muitas limitações. Essa foi considerada a primeira geração de sistemas OCR. A segunda geração surgiu próxima do início de 1970, onde os sistemas OCRs já eram capazes de maneira singela, de reconhecer cartas escritas a mão. Pouco tempo depois, surge a terceira geração, dando resultados mais precisos sendo usados em *scanners* (**??**). Os estudos estão continuamente sendo feitos desde então buscando melhorias, mas não há registros muito relevantes, nem mesmo nomeações oficiais de novas gerações.

Há dois tipos de sistemas que podem ser encontrados de forma geral com o uso de OCRs: sistemas de reconhecimento de caracteres offline (OffOCR) e sistemas de reconhecimento de caracteres online (OnOCR). Ambos são muito utilizados e sua diferença principal é bem clara. Os sistemas OffOCRs fazem o reconhecimento do texto depois de ser impresso ou escrito a mão, necessitando apenas iniciar um processo de escaneamento para então o reconhecimento propriamente dito. Já os sistemas OnOCRs fazem o reconhecimento ainda enquanto o texto está sendo produzido. Em relação aos sistemas OffOCRs, todos os detalhes das condições do texto que será escaneado são importantes pois afetam diretamente na qualidade do resultado (**??**).

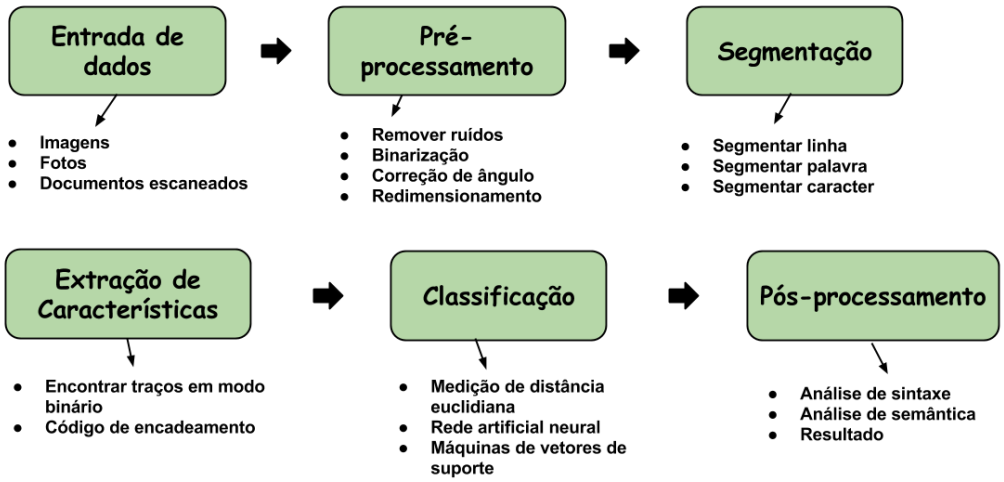
Todo o tipo de sistema OCR (OnOCR e OffOCR) passa pelos processos de entrada de dados, pré-processamento, segmentação, extração de traços, classificação e pós processamento, ilustrado na Figura 4.

As etapas ilustradas pela Figura 4 podem ser descritas da seguinte forma (**??**):

a) A etapa de entrada de dados é quando se adquire as imagens que serão analisadas, sejam fotografias, imagens coletadas de outras formas ou documentos escaneados.

b) O pré-processamento é onde a imagem de entrada deve ser trabalhada para que esteja na melhor condição possível antes de chegar no ponto onde será

Figura 4 – Etapas efetuadas por um sistema OCR



Fonte: o autor

segmentada. Isso envolve certos pontos importantes como a aplicação de *filtros* utilizados em processamento de imagens como:

- Remoção de ruídos: Erros de escaneamento, *pixels* indesejados e problemas semelhantes são removidos ou tratados;
- Binarização (Limiarização): Processo que converte em preto e branco as imagens coloridas e que também ajuda no destacamento do caracter;
- Correção de ângulo: Adequa o ângulo para que os caracteres possam ser reconhecidos corretamente;
- Redimensionamento: Passo que auxilia principalmente em relação ao tempo de processamento.

c) A segmentação é feita através de histogramas, dividida em duas partes, uma para identificar a linha onde estão as palavras e letras e outra para identificar colunas, separando caracter por caracter.

d) Extração de características é considerado o principal passo de extração de padrões em si. Dentro dessa fase encontram-se diversas técnicas, como a binarização, encadeamento dos resultados, histogramas dentre outros

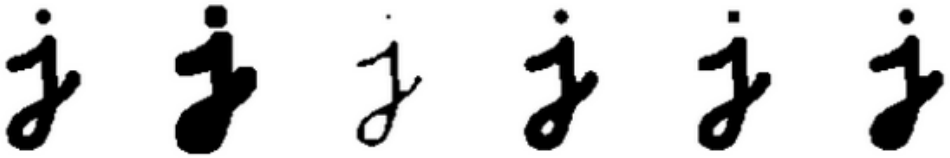
e) A classificação recebe as informações da extração de características e com isto, mede a distância entre os pontos semelhantes e faz uma comparação com um padrão pré-armazenado que a própria OCR contém, buscando em que classe a informação recebida como entrada melhor se encaixa. A fórmula da distância entre os pontos *euclidianos* será utilizada também em outras áreas no que diz respeito ao desenvolvimento deste projeto, e é dada por $d(u, v) = |u - v|$ (**??**).

f) O pós-processamento não é considerado como uma etapa essencial ou obrigatória. Porém pode ajudar num melhor resultado. Basicamente, trata de erros de caracteres muito parecidos, tais como a letra *O* e o número 0.

As bibliotecas OCRs que serão usadas no desenvolvimento, providenciam a leitura OCR. Isso significa que elas são capazes de fazer as etapas de segmentação, extração de características e de classificação descritas no item b. As etapas de entrada de dados, pré-processamento e pós-processamento são necessárias para aumento da taxa de acerto, e serão realizadas através da biblioteca OpenCV em conjunto com o Python.

A Figura 5 representa os principais filtros usados na etapa de pré-processamento. O item (a) indica uma imagem original, o item (b) a imagem passada pelo processo de dilatação, o item(c) pelo processo de erosão, o item (d) pelo processo de maioria, o item (e) pelo processo de abertura e por fim o item (f) passa pelo processo de fechamento. Alguns destes processos serão utilizados posteriormente na sessão de desenvolvimento, consequentemente receberão um melhor detalhamento de como funcionam na prática (**??**).

Figura 5 – Filtros de processamento de imagens aplicados a uma imagem



Fonte: **??**)

2.3 Conclusão

Nesse capítulo foram apresentados conceitos fundamentais sobre visão computacional dando exemplos de onde se é aplicado. Também foram analisados os conceitos básicos de um sistema OCR. O próximo capítulo tem o objetivo de demonstrar certas ferramentas que serão aplicadas no capítulo de desenvolvimento.

3 Ferramentas

Este capítulo introduz as ferramentas que serão utilizadas ao longo do desenvolvimento. Descreve-se aqui o OpenCV e suas funcionalidades principais, dando ênfase em funções que fazem parte da solução. Também são demonstradas as bibliotecas de OCR GOCR (**??**), OCRAD (**??**) e Tesseract (**??**).

3.1 O OpenCV

O OpenCV é uma biblioteca de visão computacional disponível de forma gratuita, escrita na linguagem de programação C e C++, sendo ela multiplataforma podendo ser usada em qualquer distribuição Linux, Windows e Mac OS X. O OpenCV pode ser integrado em linguagens como o Python, Ruby, Matlab, C e C++, dentre outras (**??**). Ele teve início como um projeto de pesquisa da Intel (**??**) no ano de 1998, ficando disponível com licença BSD em 2000. Foi registrado que o download dessa biblioteca foi efetuado mais de três milhões de vezes até 2012 (**??**), e até hoje vem crescendo e tem sido usado por profissionais da área, professores e estudantes.

Projetado para ter um bom desempenho diante de aplicações em tempo real, o OpenCV trabalha bem com processadores Intel. A própria empresa vende determinadas bibliotecas extras para melhorias em relação ao desempenho do OpenCV. Um dos principais objetivos do OpenCV é providenciar uma infraestrutura de visão computacional que auxilia profissionais específicos da área, professores e estudantes a desenvolverem sistemas sofisticados com simplicidade e eficiência, disponibilizando mais de quinhentas funções como por exemplo, inspeção de produtos de fábrica, imagem médica, segurança, interface para usuários, calibração de câmera, visão estéreo e robótica (**??**).

3.1.1 Funcionalidades do OpenCV

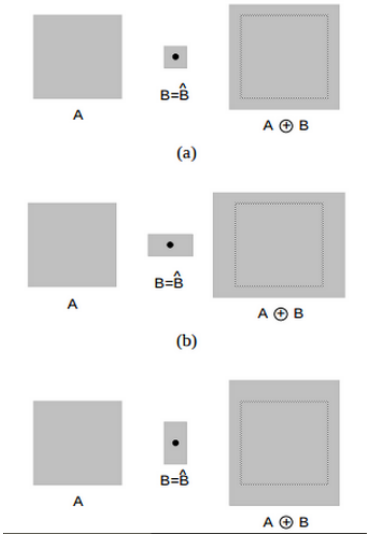
Algumas das funcionalidades do OpenCV serão exibidas nesta sub-sessão, todos os códigos deste e de outros capítulos estão na linguagem Python 2.7.2. Como analisado, existem mais de quinhentas funções no OpenCV, contudo, serão analisadas apenas algumas funções gerais de processamento de imagens e um exemplo de aplicação.

3.1.2 Dilatação e Erosão

O OpenCV disponibiliza funções de filtros que serão abordados na sessão de desenvolvimento, um destes é conhecido como dilatação (Figura 6). De acordo com os

autores (**??**) para compreender a fórmula matemática de dilatação faz-se necessário compreender também os seguintes conceitos:

Figura 6 – Demonstração do efeito de dilatação sobre um *pixel*



Fonte: **??**)

Sejam A e B conjuntos no espaço Z^2 onde respectivamente $A = (a1, a2)$ e $B = (b1, b2)$, e seja $\emptyset$ o conjunto vazio. A denotação $(A)_x$ é definida pela translação de A por um x qualquer onde $x = (x1, x2)$. Tem-se:

$$(A)_x = \{c|c = a + x\}, \text{para } a \in A$$

Isso quer dizer que, se x supostamente fosse igual ao ponto $(1, 2)$ e A fosse igual ao ponto $(3, 3)$ a translação de A por x seria igual ao ponto $(4, 5)$. Todos os *pixels* se deslocariam de A até x, resultando no ponto encontrado ao fim da equação.

A reflexão de B é dada por:

$$\hat{B} = \{x|x = -b \text{ para } b \in B\}$$

Logo, a reflexão de B trata-se da rotação de 180º em relação a origem.

O Complemento de A é:

$$A^c = \{x|x \notin A\}$$

Ou seja, o complemento de A são os pixels que não pertencem a A, em uma imagem binária (preto e branco) corresponderia aos *pixels* brancos da figura caso o objeto principal fosse em preto como na Figura 7(b).

A diferença entre A e B é definida por:

$$A - B = \{x|x \in A, x \notin B\} = A \cap B^c$$

Figura 7 – Filtros de dilatação e erosão sobre uma imagem

Fonte: **??**)

Portanto, sendo a diferença de A e B igual a intersecção de A por complemento de B, essa diferença será os pontos em comum de A e complemento de B. Com os conceitos apresentados compreendidos, segue a fórmula da dilatação:

$$A \oplus B = \{x|(\hat{B})_x \cap A \notin \emptyset\}$$

onde que A representa a imagem a ser operada, e B é um segundo conjunto de pixels, de uma forma que opera sobre os pixels de A para produzir o resultado. O conjunto B é chamado de elemento estruturante. A dilatação de A por B é o conjunto de todos os deslocamentos de x tais que A sobreponham-se em pelo menos um elemento não nulo.

No OpenCV, a dilatação pode ser aplicada de maneira simples, dando apenas duas entradas de dados, sendo elas a própria imagem e o kernel. O Kernel[1] de maneira bem resumida definirá a espessura do resultado final, basicamente é responsável por definir o quanto será dilatada a imagem (**??**). O Código 1 é um código de dilatação de uma imagem, onde inicia-se importando as bibliotecas OpenCV e Numpy, respectivamente cv2 e np. Após a importação das bibliotecas principais, uma variável chamada "img" utiliza a função *imread* do OpenCV, que apenas faz uma leitura da imagem "j.png" em escala de cinza, definida pelo parâmetro "0". Assim define-se o *kernel* e por fim a dilatação, feita através da função *dilate* também do OpenCV. Segue o código comentado:

Código 1 – Dilatação de uma imagem

```
1  import cv2     #Importando o OpenCV
2  import numpy as np   #Importando bibliotecas necessárias do Python

4  img = cv2.imread('j.png',0) #Leitura da imagem. 0 representa leitura em
       escala da cinza
5  kernel = np.ones((5,5),np.uint8) #definindo kernel
```

[1] Matriz de convolução utilizada para borrar, controlar nitidez dentre outras funções. No caso desse trabalho, define a expessura da dilatação e da erosão (**??**).

```
6 dilation = cv2.dilate(img,kernel,iterations = 1) #Dilatando
```

A erosão é o inverso da dilatação. Se por um lado o efeito da dilatação adiciona pixels a um objeto, a erosão retira (**??**). O efeito da erosão pode ser observado na Figura 7(c). A fórmula matemática do efeito de erosão é:

$$A \ominus B = \{c|(B)_c \subseteq A\}$$

O conjunto $A \ominus B$ é o conjunto de translações de B que se alinham sobre um conjunto de pixels pretos em A. Isto significa que nem todas as translações devem ser consideradas, mas apenas aquelas que, sobrepõe a origem de B em um dos membros de A. (**??**). Abaixo segue o Código 2, sendo ele muito semelhante ao Código 1, apenas diferenciando-se na aplicação final da função do OpenCV *erode* no lugar da função *dilate*.

Código 2 – Erosão de uma imagem com base na fonte

```
1 import cv2 #Importando o OpenCV
2 import numpy as np #Importando bibliotecas necessárias do Python

4 img = cv2.imread('j.png',0) #Leitura da imagem. 0 representa leitura em
      escala da cinza
5 kernel = np.ones((5,5),np.uint8) #definindo kernel
6 erosion = cv2.erosion(img,kernel,iterations = 1) #Erodindo
```

Observa-se que as funções de dilatação e erosão são aplicadas de forma praticamente idêntica. Não existe apenas uma forma de aplicar dilatação e erosão, pelo contrário, outras fórmulas dependente da situação podem ser encontradas (**??**), entretanto de forma geral estas apresentadas são base para compreensão do tema.

3.1.3 Limiarização - Thresholding

Limiarização[2] é um método de segmentação baseado em similaridade de níveis de cinza, buscando extrair objetos de interesse mediante a definição de um limiar T que separa os agrupamentos de níveis de cinza da imagem. A segmentação se dá varrendo a imagem, pixel a pixel, e rotulando cada pixel como sendo do objeto ou do fundo, em função da relação entre o valor do pixel e o valor do limiar (**??**). Limiarização é computacionalmente barato e rápido, é o método de segmentação mais antigo e ainda é amplamente utilizado em aplicações simples. Pode facilmente ser usado em tempo real. Limiarização é a transformação da imagem f para obter como saída a imagem g, como pode ser visto na fórmula abaixo (**??**):

$$g(i,j) = 1 \text{ para } f(i,j) > T$$

[2] Alguns autores traduzem o termo como *binarização*

$$g(i,j) = 0 \text{ para } f(i,j) \leq T$$

Em que T é o limiar que é definido conforme o histograma da imagem e também de acordo com quantos objetos se quer extrair da imagem, e os pontos g e f representam pixels da imagem g e da imagem f. Quando $g(i,j)$ é igual a 0, caracteriza o fundo da imagem, e quando $g(i,j)$ é igual a 1, caracteriza-se o objeto (**??**).

Este método de limiarização apresentado, é mais simples. Há outros métodos de limiarização, basicamente a diferença entre eles é a forma ou solução para encontrar o limiar T. O OpenCV disponibiliza cinco tipos diferentes de limiarização que são demonstrados no Código 3 e na Figura 8 (**??**). Não serão explicados aqui detalhes a respeito de todas as limiarizações, apenas uma pequena demonstração de cada uma.

Para que os códigos apresentados no Código 3 façam sentido, faz-se necessário o conhecimento básico do que é a biblioteca Matplotlib. Matplotlib é uma biblioteca de geração de gráficos bidimensional Python, que produz figuras de qualidade de publicação, em uma variedade de formatos e em várias plataformas. O Matplotlib permite gerar gráficos, histogramas, espectros de potência, gráficos de barras, gráficos de dispersão e outras funções (**??**).

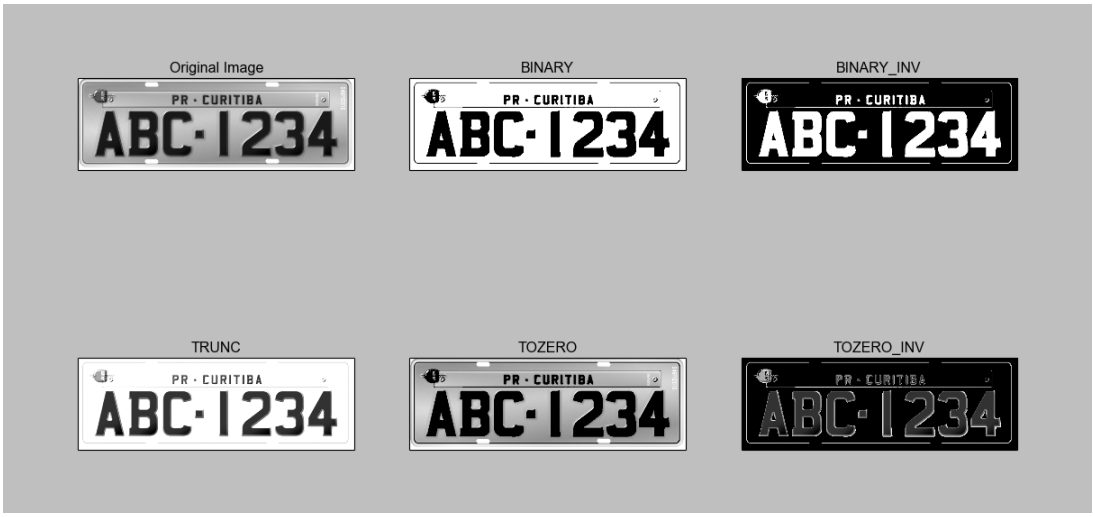
O Código 3 possui além das importações já comentadas anteriormente, a importação do matplotlib na linha 6. A leitura de uma imagem chamada "31.jpg" é feita, e então da linha 9 até a linha 13 são aplicadas as funções de limiarização disponíveis no OpenCV. Os valores 127 e 255 são *limiares* utilizados de forma geral por padrão para aplicar limiarização em imagens. O matplot é utilizado apenas para apresentar essas imagens uma ao lado da outra como pode ser verificado na Figura 8. Na linha 15 do Código 3, uma lista de nomes são agregados a uma lista de cada imagem *limiarizada*, e ambos são apresentados através do MatPlot nos comandos da linha 18 até a 22. Segue o Código 3:

Código 3 – Thresholding em uma placa

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-

4  import cv2
5  import numpy as np
6  from matplotlib import pyplot as plt

8  img = cv2.imread('31.jpg',0)
9  ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
10 ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
11 ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
12 ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
13 ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)
```

```
15  titles = ['Original Image','BINARY','BINARY_INV','TRUNC','TOZERO','
        TOZERO_INV']
16  images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]

18  for i in xrange(6):
19          plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
20          plt.title(titles[i])
21          plt.xticks([]),plt.yticks([])
22  plt.show()
```

Figura 8 – Thresholding(Limiarização) sobre uma imagem de uma placa



Fonte: o autor

3.1.4 Detecção de bordas, Histogramas e Detecção de cores

Detecção de bordas Canny é um dos algoritmos de detecção de borda que o OpenCV dispõe, e será usado aqui como exemplo embora existam outros algoritmos que possam ser tão úteis quanto ele. Ele foi desenvolvido por John F. Canny em 1986. É um algoritmo que passa por alguns estágios antes de sua solução final. Iniciando pela remoção de ruídos, encontrando a intensidade do gradiente da imagem, removendo pixels indesejados e finalmente encontrando as bordas (**??**).

Para utilizar o algoritmo de Canny, basta fazer uso da função "cv2.Canny" como feito na linha 6 do Código 4. Os valores 100 e 200 observados na linha 6, são também valores utilizados por padrão. A Figura 9 representa o resultado do Código 4 sobre uma imagem.

Código 4 – Detecção de bordas Canny, resultado representado pela Figura 9

```
1  import cv2
```

```
2  import numpy as np
3  from matplotlib import pyplot as plt

5  img = cv2.imread('placa3.jpg',0)
6  edges = cv2.Canny(img,100,200)

8  plt.subplot(121),plt.imshow(img,cmap = 'gray')
9  plt.title('Original Image'), plt.xticks([]), plt.yticks([])
10 plt.subplot(122),plt.imshow(edges,cmap = 'gray')
11 plt.title('Edge Image'), plt.xticks([]), plt.yticks([])

13 plt.show()
```

Figura 9 – Resultado do código de Detecção de bordas sobre uma imagem de uma placa

Fonte: o autor

Além dessa função de detecção de bordas o OpenCV disponibiliza visualização de histogramas, que são lidos através da biblioteca MatPlotLib. O histograma é uma das mais comuns formas de representação das distribuições de pixel em uma imagem. Quando observado é possível obter instantâneamente noções das características da imagem em questão, podendo inferir informações relevantes, como a intensidade das cores (**??**).

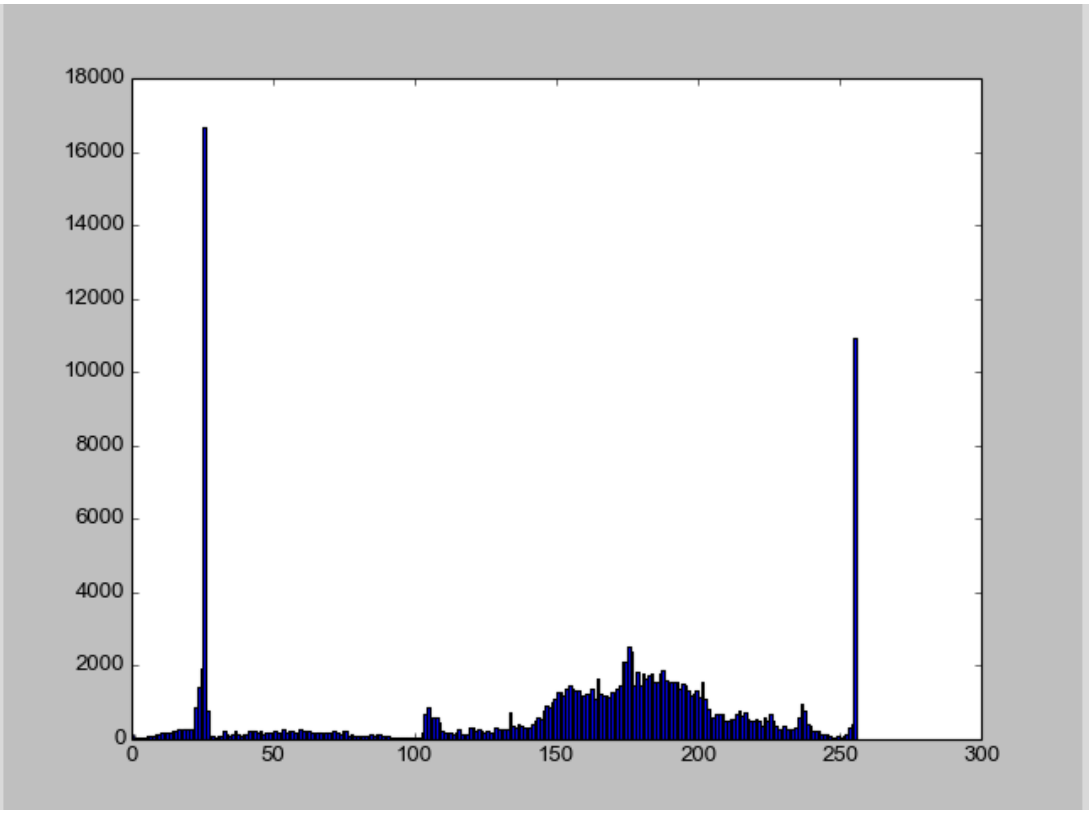
O Código 5 é um exemplo de código utilizando a mesma placa da Figura 9 e sequencialmente a Figura 10 apresenta o resultado do código. É possível verificar na linha 6 que para apresentar o histograma basta utilizar a função *hist* da MatPlotLib:

Código 5 – Histograma, resultado representado pela Figura 10

```
1  import cv2
2  import numpy as np
3  from matplotlib import pyplot as plt

5  img = cv2.imread('placa3.jpg',0)
6  plt.hist(img.ravel(),256,[0,256]); plt.show()
```

Para concluir a sessão do OpenCV, será mostrado uma pequena aplicação que destaca apenas a cor azul de objetos que possam estar em frente a câmera. Pode-se notar

Figura 10 – Resultado do código de histrograma



Fonte: o autor

que neste caso não é aberta uma imagem como entrada, mas sim captura direta da câmera, ou seja, tempo real. Este código não trabalha com o padrão RGB(*red green blue*) de cores, mas sim o padrão conhecido como HSV(*hue saturation value*).

Na linha 4 do Código 6 é iniciada a leitura da câmera, e então na linha 6 é criado um laço sem finalização. A linha 9 captura cada *frame* da câmera, e em na linha 12 é convertido para o padrão HSV. As linhas 15 e 16 definem os valores da cor azul, e na linha 19 é aplicada a função inRange para extrair apenas essa cor. A partir da linha 22 são apenas utilizados códigos para a apresentação na tela. O representação do resultado está na Figura 11.

Código 6 – Encontrando objetos da cor azul, resultado representado pela Figura 11

```
1  import cv2
2  import numpy as np

4  cap = cv2.VideoCapture(0)

6  while(1):

8      # Take each frame
```

```
9       _, frame = cap.read()

11      # Convert BGR to HSV
12      hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

14      # define range of blue color in HSV
15      lower_blue = np.array([110,50,50])
16      upper_blue = np.array([130,255,255])

18      # Threshold the HSV image to get only blue colors
19      mask = cv2.inRange(hsv, lower_blue, upper_blue)

21      # Bitwise-AND mask and original image
22      res = cv2.bitwise_and(frame,frame, mask= mask)

24      cv2.imshow('frame',frame)
25      cv2.imshow('mask',mask)
26      cv2.imshow('res',res)
27      k = cv2.waitKey(5) & 0xFF
28      if k == 27:
29          break

31  cv2.destroyAllWindows()
```

Figura 11 – Resultado do código de detecção de cor azul

Fonte: **??**)

3.2 O GOCR e o OCRAD

Nos dois capítulos anteriores, já foram identificadas as funções de uma OCR, e também foram citados os nomes de algumas bibliotecas OCR. O GOCR e o OCRAD sendo algumas dessas bibliotecas compõe parte importante neste trabalho, haja visto que trata-se da comparação de uma solução que não possui treinamento de reconheci-

mento de novos caracteres, utilizando ambas as bibliotecas, e uma solução utilizando o Tesseract(especificado na próxima sessão) com o uso de treinamento.

O GOCR é um motor OCR desenvolvido sob a Licença Pública GNU. Ele converte imagens digitalizadas de texto em arquivos de texto manipuláveis. Joerg Schulenburg foi quem criou a ferramenta, e passou a liderar uma equipe de desenvolvedores que procuraram melhorar a qualidade do GOCR. O GOCR pode ser usado em diferentes sistemas operacionais e arquiteturas. Ele pode abrir muitos formatos de imagem diferentes. Sua atualização mais recente é de 05/03/2013, sendo nomeada de "GOCR 0.50", estando disponível para download no site oficial (**??**).

O OCRAD também é um motor OCR de licença GNU, com base em um método de extração de características. A leitura de imagens são pelas extensões PBM(bitmap), pgm(escala de cinza) ou ppm(colorida) e extrai o texto nos formatos byte ou UTF-8. Também inclui um analisador de layout capaz de separar as colunas ou blocos de texto normalmente encontrados nas páginas impressas. Ocrad pode ser usado como um aplicativo de console autônomo, ou como um *backend* para outros programas. Assim como o GOCR, também pode-se contribuir para o melhoramento desse programa(**??**). O OCRAD também tem um manual online em sua página oficial.

3.3 Tesseract

Tesseract é o motor OCR de licença gratuita desenvolvido pela HP(Hewlett-Packard) (**??**) por volta de 1984 e 1994. Foi modificado e melhorado em 1995 tendo bons resultados. Tornou-se verdadeiramente licença gratuita em 2005. Hoje pertencente oficialmente a Google, o Tesseract possui suporte a diversas linguagens, tendo que ser treinado para o caso específico no qual esteja sendo submetido, para assim gerar os resultados desejados(**??**).

3.3.1 Conclusão

Nesse Capítulo foram apresentadas as ferramentas que serão utilizadas no Capítulo 4. Alguns exemplos práticos foram citados e demonstrados. Cada exemplo possui um código e uma imagem que demonstra o respectivo resultado.

4 Desenvolvimento

Neste capítulo serão abordadas as principais instalações e metodologias para a implementação dessa solução.

4.1 Principais instalações

Como descrito anteriormente, o sistema operacional utilizado para essa solução é o GNU/Linux Ubuntu 14.04. São necessárias as bibliotecas OpenCV 3.0 GOCR 0.50 e OCRAD 0.22.

4.1.1 OpenCV 3.0

Para instalar o OpenCV 3.0 são necessárias algumas dependências, para isso basta utilizar o Código 7 no terminal:

Código 7 – Comandos terminal - Dependências OpenCV

```
1 sudo apt-get -y install libopencv-dev build-essential cmake git libgtk2
    .0-dev pkg-config python-dev python-numpy libdc1394-22 libdc1394
    -22-dev libjpeg-dev libpng12-dev libtiff4-dev libjasper-dev
    libavcodec-dev libavformat-dev libswscale-dev libxine-dev
    libgstreamer0.10-dev libgstreamer-plugins-base0.10-dev libv4l-dev
    libtbb-dev libqt4-dev libfaac-dev libmp3lame-dev libopencore-amrnb-
    dev libopencore-amrwb-dev libtheora-dev libvorbis-dev libxvidcore-
    dev x264 v4l-utils unzip
```

Após instaladas todas as dependências, deve-se criar um diretório onde será instalado o OpenCV como observado na linha 1 do Código 8. Dentro desse diretório, via terminal, o próximo passo é fazer o *download* (**??**) do OpenCV visto na linha 3.

Código 8 – Comandos terminal - Download OpenCV

```
1 mkdir opencv
2 cd opencv
3 wget https://github.com/Itseez/opencv/archive/3.0.0-alpha.zip -O opencv
    -3.0.0-alpha.zip unzip opencv-3.0.0-alpha.zip
```

Quando o *download* for concluído, os comandos do Código 9 devem ser executados no terminal. Esse processo está apenas criando novo diretório onde definitivamente instala o OpenCV através do comando *Make*.

Código 9 – Comandos terminal - Processo de instalação I

```
1 cd opencv-3.0.0-alpha
2 mkdir build
3 cd build
4 cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D
      WITH_TBB=ON -D WITH_V4L=ON -D WITH_QT=ON -D WITH_OPENGL=ON ..
5 make -j $(nproc)
6 sudo make install
```

Para concluir a instalação, precisa-se ainda dos comandos do Código 10. Esses comandos têm a função de construir um *cache* das bibliotecas no sistema.

Código 10 – Comandos terminal - Processo de instalação II

```
1 sudo /bin/bash -c 'echo "/usr/local/lib" > /etc/ld.so.conf.d/opencv.
     conf'
2 sudo ldconfig
```

Dessa forma, o OpenCV 3.0 já estará pronto para uso, ao reiniciar o sistema operacional é possível testá-lo ao acessar a pasta *examples* dentro do diretório onde foi feita a instalação.

4.1.2 GOCR e OCRAD

As bibliotecas GOCR e OCRAD já estão nos repositórios do Ubuntu 14.04, e as versões que estão sendo utilizadas nessa solução são as mais recentes. Nesse caso, os únicos comandos necessários para instalação completa dessas ferramentas são apresentados no Código 11.

Código 11 – Comandos terminal - Instalando o GOCR e o OCRAD

```
1 sudo apt-get install gocr
2 sudo apt-get install ocrad
```

Considerando que o Python já vem instalado na distribuição e pronto para uso, não são mais necessárias outras instalações para a implementação da solução.

4.2 Metodologia

Essa sessão tratará da metodologia para a solução proposta, contendo os códigos e a forma como foram obtidas as imagens.

4.2.1 Solução

Para uma melhor compreensão, a solução pode ser dividida nas seguintes etapas:

a) obtenção da imagem;

b) tratamento da imagem;

c) leitura OCR;

d) tratamento da saída das OCRs.

A obtenção da imagem é a etapa mais simples de se compreender. Qualquer dispositivo com a função de fotografar pode ser usado como meio de fornecer as imagens, desde câmeras de dispositivos móveis, até *webcams* USB e câmeras digitais propriamente ditas, em qualquer nível de resolução. No caso dos testes realizados para esse trabalho, foram utilizadas câmeras de dispositivos móveis com a resolução de três megapixels. Essa imagem será a entrada da solução, e o próximo passo, que será o tratamento da imagem, necessariamente utilizará as funções do OpenCV. Dessa forma, a imagem de entrada deverá ser lida diretamente pelo OpenCV.

Para que o Python e o OpenCV estejam conectados é preciso que no código Python estejam importadas as bibliotecas que serão utilizadas. É importado então o OpenCV, chamado de *cv2*, o Numpy, chamado de *np*, e o Matplotlib como *plt*. Dessa forma poderá ser usado qualquer função do OpenCV do Numpy e do Matplotlib no restante do código. Após essas importações, uma imagem já pode ser lida através do OpenCV pela função *imread*. Ao verificar essas importações no Código 12, percebe-se que na função imread na linha 8 há dois parâmetros, um sendo o caminho relativo da imagem e o outro o valor 0, esse valor 0 representa que a imagem será lida em escala de cinza. O fato de estar em escala de cinza, facilita o processo de limiarização analisado no capítulo anterior, devido a imagem ter menos variações de cores.

Código 12 – Código da Solução I - Importações e leitura

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-

4  import cv2
5  import numpy as np
6  from matplotlib import pyplot as plt

8  img = cv2.imread('/home/user/example.jpg',0)
```

É válido citar que não serão descritas todas as linhas de código, mas sim as funções e alguns detalhes importantes, porém o código completo estará disponível em anexo. A solução inicia chamando a função *"tratamento"*(Código 13), que trata-se da etapa de tratamento da imagem. É importante dizer que nesse caso em específico, estão apenas sendo utilizadas imagens anteriormente fotografadas, mas pode-se fazer essa etapa em tempo real, pois o OpenCV disponibiliza a função *VideoCapture* que faz uso de qualquer dispositivo de câmera disponível, e também a função *imwrite* que é capaz de salvar o que está sendo capturado através da câmera.

Código 13 – Código da Solução II - Função principal

```
1  if __name__ == '__main__':
2          img = 'image.jpg'
3          tratamento_letras(img)
4          tratamento_números(img)
```

A etapa de tratamento da imagem efetua os filtros de remoção de ruídos, limiarização, dilatação, e erosão, exatamente nessa ordem, citados anteriormente nos capítulos 2 e 3. O filtro de remoção de ruídos pode ser encontrado em (**??**) na página 23. Cada filtro é salvo através da função *imread* do OpenCV. É importante destacar que a remoção de ruídos é aplicada sobre a imagem original e é salva como *remocao.png*, então *remocao.png* é lido e aplicado o filtro de limiarização, e assim sucessivamente. A Figura 12 ilustra os arquivos que são lidos, o filtro que é aplicado, e o nome salvo após a aplicação do novo filtro, deixando mais claro. O Código 14 é uma demonstração da função de tratamento da imagem.

Figura 12 – Sequência de aplicação de filtros

Ordem	Leitura	Filtro Aplicado	Escrita
1º	Arquivo original	Remoção de ruídos	remocao.png
2º	remocao.png	Limiarização	limiar.png
3º	limiar.png	Dilatação	dilatado.png
4º	dilatado.png	Erosão	erodido.png

Fonte: o autor

Código 14 – Código da Solução III - Função de Tratamento

```
1  def tratamento(img):
2          #removendo ruídos e salvando
3          remocao_ruidos.remover(img)
4          cv2.imread('remocao.png', remocao_ruidos)
5          #lendo remo ão
6          img = cv2.imread('remocao.png',0)
7          img = cv2.medianBlur(img,5)
8          #aplicando limiariza ão sobre a imagem sem ruídos
9          ret,thresh1 = cv2.threshold(img,0,255,cv2.THRESH_BINARY | cv2.
               THRESH_OTSU)
10         cv2.imwrite("limiar.png",thresh1)
11         #salvando a imagem limiarizada

13         #lendo a imagem limiarizada
14         img = cv2.imread("limiar.png",0)
15         #dilatando imagem limiarizada
16         dilate = cv2.dilate(img,np.ones((3,3),np.uint8),iterations = 1)
17         #salvando imagem dilatada
```

```
18          cv2.imwrite('dilato.png',dilate)
19          #abrindo imagem dilatada
20          img = cv2.imread("dilatado.png", 0)
21          #erodindo imagem dilatada
22          erosion = cv2.erode(img, np.ones((7,7),np.uint8), iterations =
                1)
23          #salvando imagem erodida
24          cv2.imwrite('erodido.png',erosion)
25          #conversões para formatos do OCRAD
26          os.system("convert erosion.png erosion.pgm")
27          os.system("convert dilate.png dilate.pgm")
28          os.system("convert preto_branco.png preto_branco.pgm")

30          return True
```

Depois de aplicados os filtros, as imagens salvas ficam como as que estão representadas na Figura 13. Essas imagens tratadas geram resultados muito melhores ao passar pelas OCRs, sendo que quando a imagem original não tratada é passada pelas OCRs, praticamente não são encontrados os caracteres.

Assim é possível iniciar a próxima etapa, que é a de leitura OCR. Cada imagem salva passará pela leitura OCR, pois existe a possibilidade de que as letras e os números de interesse sejam encontrados em um filtro, e não em outro.

Figura 13 – Imagens após o tratamento da placa via OpenCV e Python

Fonte: o autor

A execução da leitura OCR através do Python é simples de fazer; basta importar a biblioteca *commands* e executar o comando de leitura OCR que pode ser executado também no terminal do Linux. A biblioteca *commands* permite a execução de quaisquer comandos do sistema operacional. O Código 15 é uma função de busca das saídas da leitura OCR.

Código 15 – Código da Solução IV - Função de leitura OCR

```
1  #exemplo

3  from commands import *

5  def leitura_ocr(arquivo):
6          return (commands.getoutput("gocr -i "+arquivo))
```

O último passo a ser dado para a implementação, é o tratamento das saídas da leitura OCR. As vezes a leitura OCR encontra pixels indesejados e os traz como resultado, o que atrapalha muito para encontrar os valores que realmente interessam. A Figura 14 representa um exemplo de um dos piores casos testados, e é de grande importancia para compreensão. É possível notar que letras que representam o nome do estado e da cidade são reconhecidos e retornados, também é perceptível que letras podem ser confundidas com números.

Figura 14 – Formato de respostas da leitura OCR

Informação da placa	SC - SÃO FRANCISCO DO SUL - MJC - 1865
Dilatação OCRAD	r MJ_C. l?Qs
Erosão OCRAD	_____ ____J_C_ l ___6 5
Limiarização OCRAD	MJ_C.l?QS
Dilatação GOCR	___ _ _ _ _J_C_ l 8_6 5
Erosão GOCR	_____ ___J_C_ l ___0 5
Limiarização GOCR	____ ___ l _J_C l 8_6 5

Fonte: o autor

Na Figura 14, observa-se que na imagem que contém o filtro de dilatação, a leitura feita pelo OCRAD trouxe um pequeno “r” na frente do “MJ_C” que são os caracteres de interesse, isso é um fator que complica muito a decisão de quais são os caracteres que realmente correspondem as letras de registro da placa, e não as de informação de estado e cidade. É possível verificar que os únicos filtros que conseguiram as letras, foram a dilatação através do OCRAD e a limiarização também pelo OCRAD, em contrapartida, os únicos filtros que conseguiram os números completos, foram os de dilatação via GOCR e limiarização também através do GOCR. Um detalhe que será explorado na conclusão, é que as letras “MJC” e os números “1865” apareceram duas vezes, e nos testes realizados,

em todas as vezes que foi feita uma leitura OCR pelo menos uma vez as letras e os números apareceram em um dos filtros.

Para resolver essa situação encontrada, são necessárias duas funções, uma para encontrar as letras e outra para encontrar os números. Isso se deve a alguns erros comuns que ocorrem nas OCRs, como confundir a letra "O" com o número "0". A Figura 15 demonstra alguns dos erros mais comuns encontrados nos testes.

Figura 15 – Erros frequentes de leitura OCR

Letras	Números
O	0
l,I,i	1
B	8
b,	6
S,s	5
T	7

Fonte: o autor

Assumindo que esses erros apresentados são comuns, e que de maneira geral se conhece alguns erros como pontos, vírgulas, traços, interrogações e outros, a primeira ação no caso das letras é encontrar os caracteres incorretos e eliminá-los juntamente com os números, apenas deixando as letras maiúsculas. Essa limpeza resultaria nas saídas representadas na Figura 16.

Figura 16 – Primeira etapa de limpeza

Informação da placa	SC - SÃO FRANCISCO DO SUL - MJC - 1865
Dilatação OCRAD	MJCQ
Erosão OCRAD	JC
Limiarização OCRAD	MJCQS
Dilatação GOCR	JC
Erosão GOCR	JC
Limiarização GOCR	JC

Fonte: o autor

Nesse caso, onde houver no mínimo três caracteres, o resultado será os três primeiros, porém nem sempre os três primeiros serão as letras da placa, podendo ser uma letra da cidade ou estado como descrito anteriormente. Para os números o processo é semelhante, há uma limpeza de todas as letras e outros caracteres, deixando apenas os números e as letras que são geralmente confundidas(Figura 15) já substituídas pelo possível valor correto. No caso dos números, há uma taxa de acerto maior em relação as letras. O Código

14 demonstra um exemplo de função de limpeza para letras. Nesse código é utilizado somente a linguagem de programação, sem bibliotecas específicas para auxílio da tarefa.

Código 16 – Código da Solução V - Exemplo de limpeza de letras

```
1  #exemplo

3  def limpeza_letras(texto):

5          for i in texto:
6                  if i not in["A","B","8","C","D","E","F","G","H","I","J"
                       ,"K","L","M","N","O","P","Q","R","S","T","U","V","W
                       ","X","Y","Z"]:
7                          texto = texto.replace(i,"")
8          return texto
```

Dessa forma, o último passo é juntar os resultados e retornar a resposta, após isso, a solução está encerrada no estágio atual. A próxima sessão tem por objetivo apresentar os resultados obtidos.

4.3 Resultados

Essa sessão apresentará os resultados parciais, considerando que a solução pode ser continuada e melhorada.

A Tabela 1 mostra a taxa de acerto de três maneiras, sendo elas a placa completa, somente as letras e somente os números. Foram testadas trinta e cinco placas, em condições climáticas diferentes e em movimentação, fazendo uso de dispositivos móveis com no máximo oito *megapixels*. A taxa de acerto pode ser considerada baixa, devido ao problema citado na sessão anterior a respeito das letras que pertencem ao estado e a cidade contidos na placa.

Não existe um padrão de respostas, as letras podem estar em qualquer posição, misturadas entre outros conteúdos que não são de interesse. Os números também tem o mesmo problema.

Apesar das questões citadas foram encontradas as letras e os números correspondentes a placa de maneira visual, ou seja, ao verificar as saídas OCR uma a uma, é possível identificar de que os resultados estão envolvidos nessas saídas, e isso acontece em todas as placas.

Tendo em vista isso, uma solução seria encontrar um filtro ou efeito que de certa forma, causasse o efeito visual de embaçamento, fazendo assim com que as pequenas letras tornassem-se ilegíveis para o sistema.

Tabela 1 – Tabela de taxa de acerto

Descrição	Acertos	Erros	Taxa de acerto
Placas completas	17	17	50%
Letras	21	13	56.68%
Números	30	4	88.23%

Fonte: o autor

Para a questão dos números, a taxa de acerto pode ser considerada boa, obtendo quase 90%. Aprimorar o algoritmo de decisão de números seria um passo essencial, pois alguns erros provém de letras encontradas próximas aos números, que geralmente são confundidas, como por exemplo a letra “l” com o número “1”.

Caso haja por exemplo uma saída semelhante a “ABC l _ ?1234”, ao limpar para encontrar apenas os números e ao substituir a letra “l” por “1”, os quatro primeiros valores seriam “1123” e não “1234”, e também, se esse “l” estivesse por entre o “3” e o “4” devido a algum pixel indejesado(que pode ser causado por ausência ou excesso de luz, ou uma série de outros fatores), o resultado seria “1231”. Pode haver também o caso de um número não ser encontrado, obtendo apenas três números em vez de quatro.

Assim, o capítulo de desenvolvimento está finalizado, todas as conclusões e sugestões para continuação dessa solução, estarão disponíveis no próximo capítulo. O códigos que foram apresentados nesse capítulo não exatamente os que estão presentes na solução, porém como já informado no início do capítulo, nos anexos estará disponível o código completo.

5 Conclusão

Com esse trabalho, pode-se concluir de que essa solução não atingiu a eficiência desejada, porém há formas de conseguir melhorias significativas para os resultados. Alguns meios propostos para melhorar essa solução são o de aderir um filtro que cause o efeito visual de embaçamento, para que as letras da cidade e estado fiquem ilegíveis para o sistema, evitando letras que não correspondem a área de interesse serem encontradas. O outro meio de melhorar os resultados é aperfeiçoar a inteligência do código propriamente dito. Destaca-se o fato de que a solução não faz uso de nenhum tipo de treinamento para encontrar os caracteres.

Além das propostas de melhoramento, vale atentar para o fato de que essa solução foi feita em uma plataforma, sistema operacional, linguagem e bibliotecas de licença gratuíta, permitindo com que qualquer estudante ou pesquisador possa fazer uso e continuar os estudos. O OpenCV foi uma ferramenta que facilitou muito as aplicações de efeitos de processamento de imagens e ainda extende-se a capacidade de efetuar aplicações em tempo real, além de sua integração com o Python e outras linguagens ser simples. As OCRs utilizadas conseguiram resultados razoavelmente bons, considerando que visualmente as letras e números foram detectadas em quase 100% das placas, mas da forma em que se encontra a solução a respeito da decisão de quais são os caracteres de interesse, não foram suficientes, e dessa forma existe a possibilidade do Tesseract desempenhar melhor.

Anexos

ANEXO A – Código completo

Código 17 – Código final - Tratamento da imagem

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-

4  import cv2
5  import numpy as np
6  import remocao_ruidos
7  from PIL import Image
8  from pylab import *
9  import comparador



12 def conversoes_leitura():
13         #converte para extensão lida através do OCRAD
14         os.system("convert erosion.png erosion.pgm")
15         os.system("convert dilate.png dilate.pgm")
16         os.system("convert preto_branco.png preto_branco.pgm")

18 def erode_dil(img):
19         erosion = cv2.erode(img,np.ones((7,7),np.uint8),iterations = 1)
20         cv2.imwrite('erosion.png',erosion)
21         return cv2.imread("erosion.png", 0)

23 def dilata_limi(img):
24         dilate = cv2.dilate(img,np.ones((3,3),np.uint8),iterations = 1)
25         cv2.imwrite('dilate.png',dilate)
26         return cv2.imread("dilate.png", 0)

28 def limiariza_img(img):
29         img = cv2.medianBlur(img,5)
30         ret,thresh1 = cv2.threshold(img,0,255,cv2.THRESH_TRUNC | cv2.
               THRESH_OTSU)
31         ret,thresh1 = cv2.threshold(img,0,255,cv2.THRESH_BINARY | cv2.
               THRESH_OTSU)
32         cv2.imwrite("preto_branco.png",thresh1)
33         return cv2.imread("preto_branco.png")

35 def tratamento_letras(img):
36         #prepara para remo ão de ruídos
37         im = array(Image.open(img).convert('L'))
38         #aplica remo ão de ruídos
39         rm_ruidos, trash = remocao_ruidos.rr(im,im)
```

```
40          #deixa a imagem sem ruídos em escala de cinza
41          gray()
42          #retira pontos cartesianos da imagem em si
43          axis('off')

45          limiarizacao = limiariza_img(rm_ruidos)
46          dilatacao = dilata_limi(limiarizacao)
47          erosao = erode_dil(dilatacao)
48          return True

50  if __name__ == '__main__':
51          tratamento_letras('example_file.jpg')
52          conversoes_leitura()
53          print comparador.compara_saidas()
```

Código 18 – Código final - Remoção de ruídos

```
1   #!/usr/bin/env python
2   # -*- coding: utf-8 -*-

4   from numpy import *



7   def rr(im, U_init, tolerance=0.1, tau=0.125, tv_weight=100):
8           m,n = im.shape
9           U = U_init
10          Px = im
11          Py = im
12          error = 1

14          while(error > tolerance):
15                  Uold = U
16                  GradUx = roll(U, -1, axis=1)-U
17                  GradUy = roll(U, -1, axis=0)-U
18                  PxNew = Px + (tau/tv_weight)*GradUx
19                  PyNew = Py + (tau/tv_weight)*GradUy
20                  NormNew = maximum(1,sqrt(PxNew**2+PyNew**2))

22                  Px = PxNew/NormNew
23                  Py = PyNew/NormNew

25                  RxPx = roll(Px, 1, axis=1)
26                  RyPy = roll(Py, 1, axis=0)

28                  DivP = (Px-RxPx)+Py-RyPy

30                  U = im + tv_weight*DivP
31                  error = linalg.norm(U-Uold)/sqrt(n*m);
```

```
33          return U, im–U
```

Código 19 – Código final - Comparador de saídas

```
1   #!/usr/bin/env python
2   # -*- coding: utf-8 -*-

4   import os
5   import commands

7   def limpeza_letras(texto):
8           for i in texto:
9                   if i not in["A","B","8","C","D","E","F","G","H","I","1"
                        ,"J","K","L","M","N","O","0","P","Q","R","S","T","U
                        ","V","W","X","Y","Z"]:
10                          texto = texto.replace(i,"")
11                  if i == "8":
12                          texto = texto.replace(i,"B")
13                  if i == "0":
14                          texto = texto.replace(i,"O")
15                  if i == "1":
16                          texto = texto.replace(i,"I")
17          if len(texto) < 3:
18                  return False
19          return texto[:3]

21  def limpeza_numeros(img):

23          for i in texto:
24                  if i not in["0","1","2","3","4","5","6","b","7","8","B"
                        ,"9","l","L","i","I"]:
25                          texto = texto.replace(i,"")
26                  if i in ["l","L","i","I","t"]:
27                          texto = texto.replace(i,"1")
28                  if i in ["b"]:
29                          texto = texto.replace(i,"6")
30                  if i in ["B"]:
31                          texto = texto.replace(i,"8")

33          if len(texto) < 4:
34                          return False
35          return texto[(len(texto)-4):len(texto)]

37  def saidas():
38          texto1 = commands.getoutput("ocrad erosion.pgm")
39          texto2 = commands.getoutput("ocrad dilate.pgm")
40          texto3 = commands.getoutput("gocr erosion.png")
```

```
41          texto4 = commands.getoutput("gocr dilate.png")
42          texto5 = commands.getoutput("ocrad preto_branco.pgm")
43          texto6 = commands.getoutput("gocr preto_branco.png")
44          saidas = [texto1, texto2, texto3, texto4, texto5, texto6]
45          return saidas

47  def compara_saidas():
48          lista_saidas = saidas()
49          lista_comparacao = []
50          for texto in lista_saidas():
51                  lista_comparacao.append(limpeza_letras(texto))
52                  lista_comparacao.append(limpeza_numeros(texto))
53          return lista_comparacao
```