# IDENTIFICAÇÃO AUTOMÁTICA DE PLACA DE VEÍCULOS ATRAVÉS DE PROCESSAMENTO DE IMAGEM E VISÃO COMPUTACIONAL

**Leonardo Leite[1] Ricardo Antonello[2]**
[1]Instituto Federal Catarinense, Campus Luzerna/leonardo.leitesc@gmail.com
[2]Instituto Federal Catarinense, Campus Luzerna/ricardo.antonello@ifc.edu.br

***Resumo:** O uso da visão computacional abrange diversos campos da tecnologia como identificação de faces, objetos animados e inanimados e a identificação de caracteres em uma imagem. Neste projeto, a utilização desta tecnologia visa a identificação de placas de veículos, que podem ser utilizados para autorizar a entrada de veículos em um ambiente e registrar o histórico de presenças daquele determinado veículo. O principal objetivo deste projeto é automatizar uma cancela para permitir a entrada e saída de veículos pré-cadastrados sem intervenção humana. O projeto foi aplicado na guarita principal do Instituto Federal Catarinense, campus Luzerna. O sistema utilizado foi composto por uma webcam com definição de 720p e um sistema computacional com processador Intel Core i5 com sistema operacional Windows. Os códigos foram desenvolvidos na linguagem Python com as bibliotecas OpenCV e Tesseract. A taxa de sucesso na identificação foi de 93,54% em um conjunto de 10 veículos cadastrados utilizando-se 50 tentativas de identificação. Como trabalhos futuros sugere-se a adaptação para utilização de um sistema compacto como o Raspberry Pi.***

***Palavras-Chave:** Visão Computacional, Identificação de veículos, Reconhecimento de caracteres.***

## 1. INTRODUÇÃO

Com o visível crescimento da quantidade carros nas áreas urbanas realizar o processo de leitura de placas e identificação de veículos de maneira manual se tornou um processo caro e sujeito a erros, além disso, é um trabalho insalubre um monitor de trânsito ficar posicionado em uma via anotando a placa dos veículos que circulam pelo local. Para tornar este processo mais rápido, barato e confiável este trabalho aplicou técnicas visão computacional em um sistema composto por um computador padrão PC e uma webcam convencional.

A visão computacional, em conjunto com a programação em linguagem Python (LABAKI, 2017), está sendo usada em diversos campos atualmente, como reconhecimento de pessoas, de padrões e também no reconhecimento de caracteres. Um programa que é capaz de ler placas de veículos pode ser utilizado para gerenciamento de estacionamentos, controlando a entrada e saída de veículos, bem como para uso do Estado, multando indivíduos que infringem a lei.

O objetivo principal deste trabalho foi realizar a identificação e a leitura dos caracteres de uma placa de veículo em uma imagem e em tempo real utilizando uma câmera com sensor RGB[1]. Salienta-se também que se utilizam componentes físicos de fácil aquisição no mercado conforme descritos e especificados no desenvolvimento deste trabalho, tornando este sistema barato e acessível.

[1] RGB: Acrônimo de Red/Green/Blue traduzido respectivamente para Vermelho, Verde e Azul. RGB é o padrão para sensores câmeras coloridas utilizadas em câmeras de smartphones, webcams e câmeras de vídeo digitais em geral.

## 2. FUNDAMENTAÇÃO TEÓRICA

### 2.1 Linguagem Python

*Python* é uma linguagem de código aberto e de alto nível, criada pelo holandês Guido Van Rossum (PYTHON, 2017). Grandes empresas, como a NASA, Nokia e Disney, utilizam a programação em *Python* em seus projetos, e em muitos casos, liberam tais códigos para a comunidade.

### 2.2 Biblioteca OpenCV

*OpenCV* é uma biblioteca de código aberto direcionada principalmente a visão computacional. Ela possui algoritmos e funções especializadas para edição de imagens, permitindo a leitura de caracteres, pessoas e objetos.

Ela foi desenvolvida inicialmente pela Intel Corporation, e tinha como objetivo tornar acessível o uso da visão computacional por diversos programadores atuando assim em diversas áreas, sendo estas educacionais, industriais ou até mesmo comerciais (OPENCV, 2017).

### 2.3 Haar-cascade

O princípio de funcionamento de um haar-cascade é baseado no uso de centenas de padrões de imagens listados em um arquivo *".xml"* que, quando utilizado mais tarde, em conjunto com a biblioteca *OpenCV*, é capaz de identificar padrões nas imagens e nos vídeos (REZAEI, 2017).

O processo de criação é relativamente simples se realizado com softwares criados especificamente para este fim. Um dos softwares que podem ser utilizados, destaca-se um que não um nome especifíco, que tem como criador Mahdi Rezaei (2017), que possui PHD em Ciências da Computação e atualmente é professor na Universidade de Auckland, onde desenvolveu um artigo onde contém todas as etapas necessárias no desenvolvimento do haar-cascade.

### 2.4 Raspberry Pi

O Raspberry Pi pode ser considerado por muitos profissionais como um pequeno sistema, que é capaz de exercer o mesmo papel que um computador normal. Além disso, possui entradas e saídas digitais e analógicas para diferentes necessidades (RASPBERRY PI, 2017).

Ele possui diferentes sistemas operacionais, sendo que qualquer pessoa tem acesso a tais arquivos pelo site oficial da empresa. O principal, e mais utilizado é o Debian, um sistema baseado no Linux que funciona no Raspberry Pi.

**Figura 01:** Raspberry Pi em perspectiva

**Fonte:** Raspberry Pi (2017)

### 2.5 Biblioteca PyTesseract

O Tesseract-OCR é uma biblioteca de código aberto desenvolvida pela Google, originalmente para a linguagem C++. O seu objetivo é a leitura de textos e caracteres de uma imagem. Ela é capaz de transformar a imagem de um texto em arquivos editáveis de texto (ROSEBROCK, 2017).

Graças a comunidade de programadores, essa biblioteca foi modificada, permitindo o funcionamento da mesma em programação *Python*. Por tal motivo, o nome desta biblioteca para a linguagem *Python* tornou-se "*PyTesseract"*.

## 3. METODOLOGIA

Visando uma boa organização das tarefas, o projeto foi separado em etapas, que podem ser observadas abaixo:

- *Estudo das placas:* esta foi uma das principais fases do projeto, onde centrou-se no estudo das placas. Notou-se que as placas possuem grande ruído, principalmente aquelas que possuíam fundo refletivo. Foi nesta etapa que foi constatado que a fonte utilizada era a chamada "*Mandatory"*.
- *Criação de um "Haar-Cascade":* como não há um haar-cascade para placas nacionais, deu-se como necessário a criação do primeiro *Haar-Cascade*, que tem como objetivo a identificação das placas.
- *Integração das placas com a visão computacional:* essa etapa teve como o objetivo tirar os ruídos das imagens provindas da câmera, e deixar apenas as letras e os números em evidência. Um exemplo desse processo pode ser visto nas imagens 4 e 5.
- *Estudo da biblioteca PyTesseract:* Criação de um dicionário especialmente direcionado a leitura das placas, já que não há nada na internet que atende as necessidades do projeto.

- *Leitura da placa:* a última etapa do projeto é a integração da biblioteca do OpenCV, com a biblioteca PyTesseract e seu dicionário destinado para a detecção de placas.
- *Integrá-lo a uma cancela:* o objetivo final do projeto é colocar o sistema de identificação de placas na cancela da entrada do Instituto Federal Catarinense. Junto a essa etapa está a integração de som, tornando o sistema "amigável" ao usuário. A ideia atual é o sistema emitir um som com a mensagem "Bem vindo(a)" e o nome do usuário após a identificação.


**Imagem 04**: Placa sem filtro
**Fonte**: Os autores


**Imagem 05:** Placa com filtro
**Fonte:** Os autores

## 3.1 FILTROS DO OPENCV

Como já comentado, as imagens de placas apresentam ruídos intensos, que para um programa de identificação são extremamente prejudiciais à leitura. Para uma correção utiliza-se a função *"Threshold"*, que serve nesse caso como uma espécie de filtro, intensificando o que é importante na imagem.

Nesta etapa do código também é realizado a parte do corte da placa, separando as duas partes escritas da placa: os caracteres da cidade e os da identificação. Assim, o código consegue ler as informações separadamente, diminuindo as chances de erro na leitura.

O código de filtragem, cortando o nome da cidade na placa, pode ser visto na tabela 01.

**Tabela 01**

```
def edicão_placa(img2):
    img = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
    imgcorte_alt_alt = int(img.shape[0] * 0.33)
    imgcorte_alt_baix = int(img.shape[0] * 0.87)
    imgcorte_lad_dir = int(img.shape[1] * 0.07)
    imgcorte_lad_esq = int(img.shape[1] * 0.97)
    recorte = img_figura[imgcorte_alt_alt:imgcorte_alt_baix,
imgcorte_lad_dir:imgcorte_lad_esq]
    suave = cv2.GaussianBlur(recorte, (3, 3), 10)
    T = mahotas.thresholding.otsu(suave)
    temp = recorte.copy()
    temp[temp > T] = 255
```

```
    temp[temp < 255] = 0
    temp = cv2.bitwise_not(temp)
    blur_mediana = cv2.medianBlur(temp, 13)
    blur = cv2.blur(blur_mediana, (9, 21))/
    cv2.imwrite("Placa filtrada.jpg", blur)
    return blur
```

## 3.3 CRIAÇÃO DO HAAR-CASCADE

Como já comentado, há a necessidade de criação de um *Haar-Cascade* destinado às placas nacionais de identificação de veículos. As etapas deste processo podem ser observadas abaixo:

- *Coleta de imagens positivas:* para a obtenção de placas positivas para a criação haar-cascade, foi necessário a construção de uma biblioteca com 200 imagens.
- *Imagens negativas:* as imagens negativas não podem conter uma placa.
- *Criação do "haar-cascade":* Necessário para o padrão de placas brasileiro.


**Figura 06:** Uma das etapas da criação do *"haar-cascade"* para as placas de veículos

**Fonte:** Os autores

## 3.4 CRIAÇÃO DO DICIONÁRIO

As placas nacionais são escritas pela fonte Mandatory, como já mencionado anteriormente. Infelizmente não há nenhum dicionário do *PyTesseract* destinado à leitura de tais caracteres. Para isso utilizou-se programas externos à biblioteca, criados pela comunidade. Os programas utilizados foram:

### 3.4.1 JTESSBOXEDITOR

Este é responsável por salvar os algoritmos de cada caractere em um arquivo ".TIFF". A primeira etapa deste processo é colocar os caracteres em um arquivo "*.txt"* para depois importá-lo para o programa. O programa abre esse arquivo, lê cada caractere e salva suas coordenadas para,

mais tarde, realizar a localização dos mesmos em uma imagem. Esse processo pode ser visto na imagem 06:

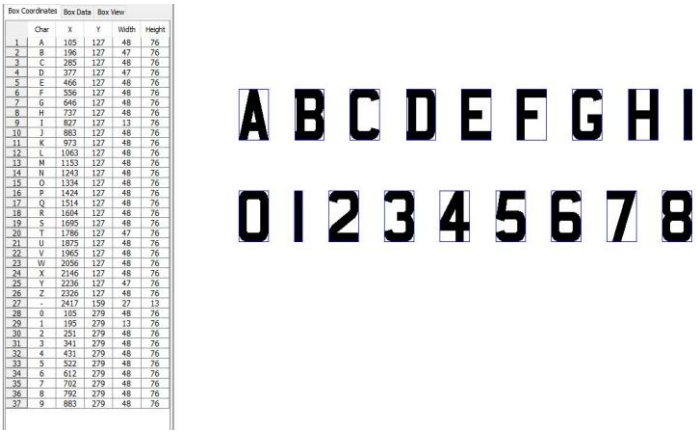


**Figura 06:** Programa *JTessBoxEditor* em funcionamento

**Fonte:** Os autores

Para a realização deste projeto utilizou-se apenas os caracteres do alfabeto, os números de 0 e 9 e o hífen, pois são apenas estes caracteres que estão presentes em uma placa de carro.

### 3.4.2 SERAK TESSERACT TRAINER

Este é responsável por unir todos os arquivos ".TIFF" necessários em um arquivo ".traineddata", formato padrão do *PyTesseract*. Este programa também foi criado pela comunidade, possuindo seu código aberto a todos. Seu nome deve-se ao seu criador Serak Shiferaw. Sua função é transformar o arquivo criado pelo *jTessBoxEditor* em um arquivo ".traineddata" para a biblioteca do *PyTesseract*. O funcionamento do programa pode ser visto na imagem 07.

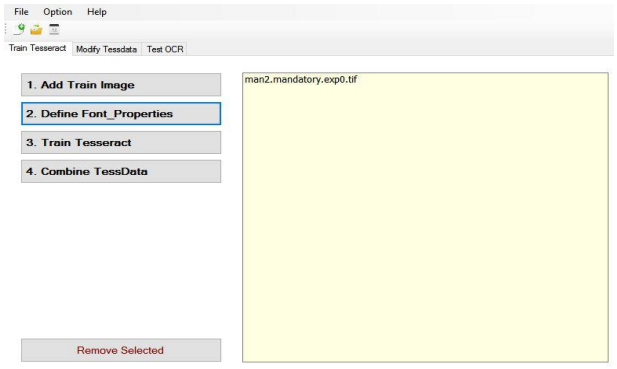


**Figura 07:** Programa *"Serak Tesseract Trainer"* em funcionamento

**Fonte:** Os autores

Este programa também permite colocar "padrões" de resultados, permitindo que o programa sempre tente esses padrões nas imagens. As placas brasileiras utilizam um padrão de 3 letras, um hífen e 4 números: ABC-XXXX.

Para isso criou-se um código básico em *Python* responsável pela criação randômica de milhares placas, para que o programa sempre tente procurar resultados parecidos com alguma dessas placas. Tal código pode ser visto na tabela 02 abaixo e o resultado de compilação na imagem 08.

**Tabela 02**

```
from random import *
letras = ["A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q"
,"R","S","T","U","V","W","X","Y","Z"]
num = ["0", "1", "2", "3", "4","5", "6", "7", "8", "9"]
a = 0
while(a>=0):
x = randint (0,25)
y = randint (0,25)
z = randint (0,25)
a = randint (0, 9)
b = randint (0, 9)
c = randint (0, 9)
d = randint (0, 9)
res = letras[x] + letras[y] + letras[z] + "-" + num[a] + num[b] + num[c] + num[d]
print(res)
a+=1
```

```
XWZ-1321
YZK-8580
KSO-3087
GAU-7012
IKW-2286
HUP-7904
```

**Figura 08:** Algumas placas randômicas criadas pelo programa

**Fonte:** Os autores

Por fim, o programa cria o arquivo ".traineddata" com todas essas informações já citadas, permitindo assim a leitura das placas.

## 4. RESULTADOS

Nos primeiros testes notou-se um problema de identificação, pois as letras "I" e "O" são iguais aos números "1" e "0", respectivamente. Por tal motivo o programa ao invés de identificar a placa como "PLA-0000", identificava como "PLA-OOOO". Para solucionar tal problema foi preciso dividir a *string* em duas partes: parte com letras e a parte com números. A partir é só colocar no código algumas linhas de substituição de caracteres. O resultado de tal compilação pode ser visto na imagem 10.


**Figura 09:** Placa de carro
**Fonte:** Ig (2017)

```
A placa é (sem modificações): PLA-0000
A placa é (com modifacações): PLA-0000

Process finished with exit code 0
```

**Figura 10:** Resultado da compilação
**Fonte:** Os autores

Com a leitura de caracteres funcionando corretamente, testou-se a identificação de placas em uma imagem. E o processo deu-se de maneira bem satisfatória, pois conseguiu identificar de maneira correta a placa na imagem em 93,54% dos casos para um conjunto de 10 placas cadastradas utilizando-se 50 tentativas de identificação.


**Figura 11:** Observa-se na esquerda, o software identificando a placa no vídeo, e à direita, o resultado da leitura dos caracteres.

**Fonte:** Os autores

O sistrema não possiu uma boa performance apenas quando a placa esta em movimento, pois no momento que ela se move os caracteres ficam borrados, tornando-os, naquele espaço pequeno de tempo, incompreensíveis ou semelhantes a outros. O código final encontra-se na tabela 03 deste artigo.

**Tabela 03**

```
import os, cv2, mahotas, pytesseract, time
from PIL import Image
################################################################################
def edição_placa(img2):
    img = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
    imgcorte_alt_alt = int(img.shape[0] * 0.33)
    imgcorte_alt_baix = int(img.shape[0] * 0.87)
    imgcorte_lad_dir = int(img.shape[1] * 0.07)
    imgcorte_lad_esq = int(img.shape[1] * 0.97)
    recorte = img[imgcorte_alt_alt:imgcorte_alt_baix, imgcorte_lad_dir:imgcorte_lad_esq]
    cv2.imshow("recorte", recorte)
    suave = cv2.GaussianBlur(recorte, (3, 3), 10)
    T = mahotas.thresholding.otsu(suave)
    temp = recorte.copy()
    temp[temp > T] = 255
    temp[temp < 255] = 0
    temp = cv2.bitwise_not(temp)

    blur_mediana = cv2.medianBlur(temp, 13)
    blur = cv2.blur(blur_mediana, (9, 21))
    cv2.imwrite("Placa filtrada.jpg", blur)
    return blur

################################################################################
blur = 0
camera = cv2.VideoCapture(0)
df = cv2.CascadeClassifier('placas_original.xml')
placa_escrita = 0
while True:
    (sucesso, frame) = camera.read()
    if not sucesso: #final do vídeo
       break
    imgPB = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

```
    placas = df.detectMultiScale(imgPB, scaleFactor=1.02, minNeighbors=10,
minSize=(60,60),flags=cv2.CASCADE_SCALE_IMAGE)
    imgPB_temp = imgPB.copy()
    imgC = frame
    cv2.imshow("Encontrando placas...", frame)
    if (b == 0):
        for (x, y, w, h) in placas:
            a = a +1
            placa = frame[y:y + h, x:x + w]
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 255), 2)
            cv2.imwrite("imagem_frame.jpg", frame)
            edicão_placa(placa)
            caracs = blur
            caracs = pytesseract.image_to_string(Image.open("Placa filtrada.jpg"),
lang="amh")
            caracs = caracs.replace(' ', '')
            caracs = caracs.replace("-", "")
            letras = caracs[:3]
            num = caracs[3:]
            num = num.replace("-", "")
            letras = letras.replace("-", "")
            num = num.replace('O', "0")
            letras = letras.replace('0', "O")
            num = num.replace('I', "1")
            letras = letras.replace('1', "I")
            num = num.replace('G', "6")
            letras = letras.replace('6', "G")
            num = num.replace('B', "8")
            letras = letras.replace('8', "B")
            num = num.replace('T', "1")
            letras = letras.replace('1', "T")
            num = num.replace('Z', "2")
            letras = letras.replace('2', "Z")
            num = num.replace('H', "11")
            letras = letras.replace('11', "H")
            num = num.replace('S', "5")
            letras = letras.replace('5', "S")
            placa_escrita = letras + '-' + num
            print (placa_escrita[:8])
    if cv2.waitKey(1) & 0xFF == ord("s"):
        break
camera.release()
cv2.destroyAllWindows()
```

## 5. CONSIDERAÇÕES FINAIS

O código atualmente é capaz de identificar os caracteres, estando estes presentes em um vídeo em tempo real. Vale ressaltar que o mesmo não identifica a cidade e estado, pois não foi desenvolvido ainda para tais fins. Além disso, pode ser utilizado como base para vários sistemas de detecção de padrões de placas, ou de leitura de caracteres.

## 6. TRABALHOS FUTUROS

Sugere-se para trabalhos futuros a otimização do código para utilização em Raspberry Pi.

## 7. AGRADECIMENTOS

Agradeçemos o Instituto Federal Catarinense, *campus* Luzerna, por permitir a realização deste trabalho através do fomento a pesquisa via edital 12/2016.

## 8. REFERÊNCIAS BIBLIOGRÁFICAS

CARRO ANTIGO (Comp.). **Década de 10.** Disponível em: <http://www.carroantigo.com/portugues/conteudo/fotos_10.htm>. Acesso em: 20 abr. 2017.

CRIE SEU CARRO (Org.). **Placas coloridas.** 2016. Disponível em: <http://www.crieseucarro.net/placas-coloridas/>. Acesso em: 01 maio 2017.

IG. **Carros terão de trocar placas no Estado de São Paulo.** Disponível em: <http://carros.ig.com.br/fotos/carros+terao+de+trocar+placas+no+estado+de+sao+paulo/3362.html >. Acesso em: 01 maio 2017.

ITARO (Comp.). **Conheça os significados das placas de carro.** 2016. Disponível em: <https://www.itaro.com.br/blog/2016/02/conheca-o-significado-das-placas-de-carro/>. Acesso em: 20 abr. 2017.

LABAKI, Josué. **Introdução a Python - Módulo A.** Disponível em: <http://www.dcc.ufrj.br/~fabiom/mab225/pythonbasico.pdf>. Acesso em: 20 abr. 2017.

MARENGONI, Maurício; STRINGHINI, Denise. **Tutorial: Introdução à Visão Computacional usando OpenCV.** 2009. Disponível em: <https://ai2-s2-pdfs.s3.amazonaws.com/a0a6/301a239519f117ca6c0398d5230ff15c67ff.pdf>. Acesso em: 20 abr. 2017.

NGUYEN, Quan (Comp.). **JTessBoxEditor.** Disponível em: <https://github.com/nguyenq/jTessBoxEditor>. Acesso em: 23 abr. 2017.

OFICIAL BLOG (Org.). **Consulte se a placa é de veículo roubado, furtado ou clonado.** 2014. Disponível em: <http://www.oficialblog.org/2014/04/consulta-situacao-de-veiculo-pela-placa.html>. Acesso em: 01 maio 2017.

OPENCV. About. Disponível em: <http://opencv.org/about.html >. Acesso em: 20 de jul. 2017.

PYTHON. Disponível em: <https://www.python.org/>. Acesso em: 01 de jul. 2017

RASPBERRY PI (Org.). **RASPBERRY PI 2 ON SALE NOW AT $35.** Disponível em: <https://www.raspberrypi.org/blog/raspberry-pi-2-on-sale/>. Acesso em: 20 abr. 2017.

REZAEI, Mahdi. **Creating a Cascade of Haar-Like Classifiers: Step by Step.** Disponível em: <https://www.cs.auckland.ac.nz/~m.rezaei/Tutorials/Creating_a_Cascade_of_Haar-Like_Classifiers_Step_by_Step.pdf>. Acesso em: 25 jun. 2017.

RICHARDSON, Matt; WALLACE, Shawn. **Primeiros Passos com o Raspberry Pi.** 2013. Disponível em: <http://www.martinsfontespaulista.com.br/anexos/produtos/capitulos/704370.pdf>. Acesso em: 20 abr. 2017.

ROSEBROCK, Adrian. **Using Tesseract OCR with Python.** 2017. Disponível em: <http://www.pyimagesearch.com/2017/07/10/using-tesseract-ocr-python/>. Acesso em: 01 jun. 2017.

SHIFERAW, Serak (Comp.). **Serak Tesseract Trainer for Tesseract 3.02**. Disponível em: <https://github.com/serak/serak-tesseract-trainer>. Acesso em: 23 abr. 2017.