Universidade Federal de Pernambuco
Centro de Tecnologia e Geociências
Departamento de Engenharia Mecânica
Engenharia Mecânica

ÍTALO ALVES CARNEIRO

Aplicação do processamento de imagem digital em uma cancela no reconhecimento de placas de carro usando Python e Arduino

Recife
2018

ÍTALO ALVES CARNEIRO

Aplicação do processamento de imagem digital em uma cancela no reconhecimento de placas de carro usando Python e Arduino

Trabalho de Conclusão de Curso apresentado à banca examinadora do curso de Engenharia Mecânica da Universidade Federal de Pernambuco, como exigência Parcial para a obtenção do título de Bacharel em Engenharia Mecânica.

Área de concentração: Engenharia Mecânica

Orientador: Prof. DSc. João Paulo Cerquinho Cajueiro

Recife

2018

Catalogação na fonte
Bibliotecária Valdicéa Alves, CRB-4 / 1260

C289a Carneiro, Ítalo Alves.

Aplicação do processamento de imagem digital em uma cancela no reconhecimento de placas de carro usando Python e Arduino / Ítalo Alves Carneiro - 2018.

49folhas, Il.; e Tabs.

Orientador(a): Prof. DSc. João Paulo Cerquinho Cajueiro.

TCC (Graduação) – Universidade Federal de Pernambuco. CTG. Curso de Graduação de Engenharia Mecânica, 2018.

Inclui Referências e Anexos.

1. Engenharia Mecânica. 2. Python. 3. Arduino. 4. Processamento digital de imagem. 5. Placa de carro. 6. Reconhecimento de padrões.

I. Cajueiro. João Paulo Cerquinho. II. Título.

UFPE

621 CDD (22. ed.) BCTG/2019-010

ÍTALO ALVES CARNEIRO

Aplicação do processamento de imagem digital em uma cancela no reconhecimento de placas de carro usando Python e Arduino

Trabalho de Conclusão de Curso apresentado à banca examinadora do curso de Engenharia Mecânica da Universidade Federal de Pernambuco, como exigência Parcial para a obtenção do título de Bacharel em Engenharia Mecânica.

Aprovada em: 11 / dezembro / 2018.

BANCA EXAMINADORA

_____

Profº. Dr. João Paulo Cerquinho (Orientador)
Universidade Federal de Pernambuco

_____

Engo. Jeydson Lopes da Silva (Examinador Interno)
Universidade Federal de Pernambuco

_____

Profº. Dr. João Marcelo Teixeira (Examinadora Interna)
Universidade Federal de Pernambuco

AGRADECIMENTOS

Agradeço primeiramente a Deus, por ter me sustentado durante todos esses anos da graduação. Aos meus pais, Mary Janes e Almir Carneiro, por todo apoio e o estímulo a dar o meu melhor, fatos que me tornaram a pessoa que sou hoje, aos meus irmãos, Alexandra, André e Priscilla, pelo apoio constante nessa jornada, aos meus amigos da faculdade, por me darem conselhos e tornarem a caminhada mais fácil de suportar com grupos de estudos, dias de risadas e almoço no RU. Agradeço a Mario Jorge por sempre estar disponível em ajudar com o TCC. Agraço a Karynne Guimarães por todo apoio e ajuda nesse trabalho.

A todos os professores que contribuíram para a minha formação, em particular ao professor João Paulo, meu orientador, que me acompanhou durante o trabalho e o professor Jeydson Lopes por toda colaboração. Ao Centro de Ciências e Tecnologia – CTG, por ter me proporcionado estudar nesse ambiente para minha instrução.

RESUMO

O controle de acesso veicular pode ser automatizado com o uso de biometria, cartões eletrônicos, senhas, bem como explorando a identificação da placa do carro usando reconhecimento de padrões e processamento de imagem, melhorando a eficiência e trazendo agilidade ao processo. O processamento digital de imagens (PDI) objetiva a melhoria da imagem e a extração de informações através desta. Sob essa ótica, é possível compreender o uso do PDI para reconhecimento de padrões. Haja vista o alto número de veículos em circulação, surgiu a necessidade de identificá-los. Essa identificação é feita por placas de automóveis, que seguem padrões de acordo com sua região. No Brasil é usado o padrão alfanumérico. Dessa forma, é possível utilizar o processamento de imagem para seu reconhecimento. O presente trabalho realizou a identificação das placas de carro com a finalidade de obter um reconhecimento destas placas para a liberação de uma cancela. Para esse trabalho, foi utilizado um código escrito na linguagem *python* auxiliado de suas bibliotecas, principalmente *OpenCV*, *Pytesseract*, *Pandas* e *pyFirmata*. Foi utilizada também uma placa de Arduino, um notebook, uma *webcam* e um servo motor.

Palavras-chave: *Python*. *Arduino.* Processamento digital de imagem. Placa de carro. Reconhecimento de padrões.

ABSTRACT

Vehicle access control can be automated with the use of biometrics, electronic cards, passwords as well as exploring car plate identification using pattern recognition and image processing, improving efficiency and bringing agility to the process. Digital image processing (PDI) aims at improving the image and extracting information through it. From this perspective, it is possible to understand the use of PDI for pattern recognition. Given the high number of vehicles in circulation, the need to identify them arose. This identification is made by auto signs, which follow standards according to its region. In Brazil the alphanumeric pattern is used. In this way, you can use image processing for recognition. The present work carried out the identification of the car plates in order to obtain a recognition of these plates for the release of a gate. For this work, a code was written in the python language aided by its libraries, mainly OpenCV, Pytesseract, Pandas and pyFirmata. It also used an Arduino board, a laptop, a webcam and a servomotor.

Palavras-chave: *Python*. *Arduino.* Digital image processing. Car plate. Pattern recognition.

LISTA DE FIGURAS

LISTAS DAS TABELAS

SUMÁRIO

1 INTRODUÇÃO

Os avanços tecnológicos têm transformado muito a ciência, a educação e a saúde, assim, mudando a forma como vivemos. A troca de informação passou a ser mais rápida, quase instantânea (Gonçalves, 2006). Dessa forma, a adoção de sistemas automatizados proporciona maior segurança, conforto, agilidade, economia e possui a vantagem de que praticamente qualquer atividade pode ser parcial ou completamente automatizada (Garcia, 2013). Uma possível aplicação é a automação do controle de acesso veicular.

O controle de acesso veicular pode ser automatizado com o uso de biometria, cartões eletrônicos, senhas, bem como explorando a identificação da placa do carro, melhorando a eficiência e trazendo agilidade ao processo. O trabalho atual irá focar na identificação das placas dos carros para controle do acesso veicular por meio de cancela.

O sistema utilizado nas placas veiculares foi o alfanumérico, que entrou em vigor no ano de 1969 e sofreu alterações para o sistema de emplacamento atual. A estrutura atual, segundo a resolução 231 de 15/03/2007, consiste em 3 letras e 4 números pretos sobre um fundo cinza. A cor do fundo e letras pode ser diferente para alguns tipos de veículos, como exemplo, é possível citar os transportes públicos, que possuem fundo vermelho (Conheça o significado das placas de carro, acesso em outubro, 2017). Na **figura 1** a seguir, seguem exemplificadas algumas placas de veículos encontradas. Levando ao entendimento de que placas de carro seguem um padrão de caracteres.

Figura 1: Placas de veículos encontradas em diferentes estados.

Fonte:http://midiainteressante.com/2017/02/cor-das-placas-dos-carros-no-brasil.html. (2018)

Além disso, o processo de construção de imagens remonta a 40.000 a.C e permanece em constante evolução até os dias atuais (Lima, 2009). Uma dessas evoluções foi o processamento de imagens (PDI), que pode reconstruir uma cena tridimensional a partir de uma imagem real que, por sua vez, foi registrada por uma câmera e extrair alguma informação (De Queiroz & Gomes, 2006). Essa tecnologia está relacionada com a computação gráfica (CG), o processamento de dados (PD), a visão computacional (VC), a inteligência artificial (IA), o reconhecimento de padrões (RP), entre outros (De Queiroz & Gomes, 2006; Carvalho, 2004). Sua aplicação se dá em diversas áreas, como na classificação de malignidades em mamografias, projetos sobre cirurgia guiada por imagem, inúmeras aplicações industriais e, até mesmo no cotidiano em biometria, detecção de face ou objetos (Carvalho, 2004; Rocha, 2013). Sua aplicação no controle de acesso veicular é de fundamental importância para o conhecimento da placa do veículo.

O processamento de imagens tem como objetivo a melhoria da imagem e a extração de informações. Este processamento consiste em uma entrada de imagem ou uma fotografia ou quadro de vídeo, logo, essa imagem passa por um tratamento para melhorar a qualidade, para seus fins, então, é extraída alguma forma de informação. Sob essa ótica é possível compreender o uso do PDI para reconhecimento de padrões de placas automotivas e controlar acesso veicular (Lima, 2009).

Durante o processamento de imagem, são utilizadas as seguintes técnicas (Facon, 200-?):

- **Aquisição e digitalização**
  A imagem do sensor é transformada em uma imagem digital sob a forma de uma tabela de valores discretos inteiros chamados pixels.
- **Pré-processamento**
  Esta etapa permite corrigir alguns defeitos e imperfeições adquiridos no processo de aquisição e digitalização. Uma correção comum é a compensação de uma deficiência de iluminação.
- **Segmentação**
  Consiste em dividir uma imagem em partes constitutivas (segmentos ou primitivas) e armazená-las de forma adequada. A maioria dos processamentos posteriores é baseada na pesquisa dessas partes.

- **Interpretação**
  É a parte em que é interpretada a imagem. Ela representa o “alto nível” e permite obter a compreensão e a descrição final do fenômeno inicial.

1.1 APLICAÇÕES DO PDI

Os sistemas de visões de máquina, para que haja PDI, estão cada vez mais presentes na automação industrial. A cada dia surgem novas ideias e aplicações nas mais diversas áreas da ciência.

Uma dessas aplicações é a inspeção automática de peças, onde o sistema de visão é utilizado para inspeção em controle de qualidade. A grande vantagem do uso de sistemas de visão é a velocidade com que a inspeção pode ser realizada, pois as medições são transferidas eletronicamente para controle central da produção. Contudo, a precisão é, geralmente, menor que a inspeção convencional, feita por um operador usando um instrumento de medição.

A inspeção visual automática já é uma realidade em diversas indústrias. Como exemplos, temos inspeções de tubos metálicos, lâminas de madeira, qualidade de tecidos, entre várias outras (Zibetti, 2011).

Para a robótica, o processamento de imagem é muito usado em controle de direção. Uma dessas aplicações é a robótica de manufatura, com o objetivo de capturar e transportar peças para agrupamento, de forma inteiramente automática, por manipuladores industriais. Nesse exemplo, a visão robótica vai interpretar a localização do objeto e para onde ele deve ser transportado.

1.2 OBJETIVO

Esse projeto de pesquisa tem como objetivo o desenvolvimento de um protótipo de cancela automática.

Um programa, para a identificação por visão computacional das placas de veículos, checa o cadastro desta placa em um banco de dados e aciona um servo motor, que fará o papel de um motor de cancela para liberar a passagem do veículo.

A implementação fará uso de uma webcam, um notebook e uma placa Arduino, por facilidade de acesso. Serão explicadas as etapas de desenvolvimento deste projeto, bem como as possíveis falhas durante o processo.

2 METODOLOGIA

O presente projeto foi dividido em duas etapas: desenvolvimento da lógica computacional e montagem do protótipo, bem como, seu descritivo de desenvolvimento e resultado dos testes.

1. Desenvolvimento da lógica computacional: foi feito a conceituação e o desenvolvimento do projeto de programação de visão computacional. A implementação do código foi feita na linguagem *Python*, devido às facilidades de trabalho com uma linguagem *open source*, através da utilização das bibliotecas *OpenCV*, essencialmente para a implementação das técnicas de processamento de imagem, *PyTesseract*, entre outras que contribuem para o desenvolvimento. Serão explicadas as principais funções, seus argumentos e saídas usadas e algumas lógicas. Após a lógica concluída foi montado o protótipo usando uma Placa de Arduino e servo motor.

2. Análise e relatório dos testes: Os testes de identificação das placas e controle do acesso foram realizados assim que o protótipo foi concluído. Podem surgir modificações ao encontrar algumas falhas. Assim, será analisado e relatado.

Para isso, é preciso melhor entendimento de conceitos como imagem digital, processamento de imagem e placas veiculares.

3 Revisão

Na revisão de literatura, visando aprofundar o conhecimento sobre os assuntos que envolvem o presente trabalho, são apresentados conceitos e teorias sobre placas de carro, imagens e processamento de imagem.

3.1 Placas de Automóveis e Reconhecimento

Antes do sistema de placas atual, foram desenvolvidos outros que em sua época tinham apenas o objetivo de regulamentação dos veículos, devido ao desenvolvimento industrial e à alta produção. O sistema pioneiro ficou vigente até 1941. Basicamente, a placa era formada por uma letra e por números que podiam

variar de 1 a 5 algarismos (Conheça o significado das placas de carro, acesso em outubro, 2017).

Com o passar dos anos esse sistema já não satisfazia as necessidades e entrou em vigência, em 1969, o sistema alfanumérico. Ele foi dividido em cores, sendo cada cor responsável por representar um determinado tipo de uso do veículo. Para esse sistema as letras foram abolidas (Conheça o significado das placas de carro, acesso em outubro, 2017).

O sistema alfanumérico consistia em combinações de duas letras (nos dias atuais são três) e quatro números, com cores para identificar diferentes tipos de veículos

3.1.1 Características da placa

As características das placas brasileiras do ano 2000 em diante para carros são: Altura de 130mm, largura de 400mm, caracteres na altura de 53mm (tem outra resolução mais antiga que diz 63mm), fonte *Mandatory*, largura do traço de 10mm (LPR – License Plate Recognition Brazil – Parte 1, acesso em outubro, 2017). Como é possível visualizar a seguir na **figura 2**.

Figura 2: Placas de carro.

Fonte: http://coxipoplacas.com/legislacao/caracteristicas_das_placas.php. (2018).

As motos possuem um formato mais compacto, onde as letras ficam em cima e os números embaixo (LPR – License Plate Recognition Brazil – Parte 1, acesso em outubro, 2017). Suas características são: Altura de 170mm, largura de 200mm, caracteres na altura de 53mm, fonte *Mandatory*, largura do traço de 6mm.

3.2 MODELO DE IMAGEM DIGITAL

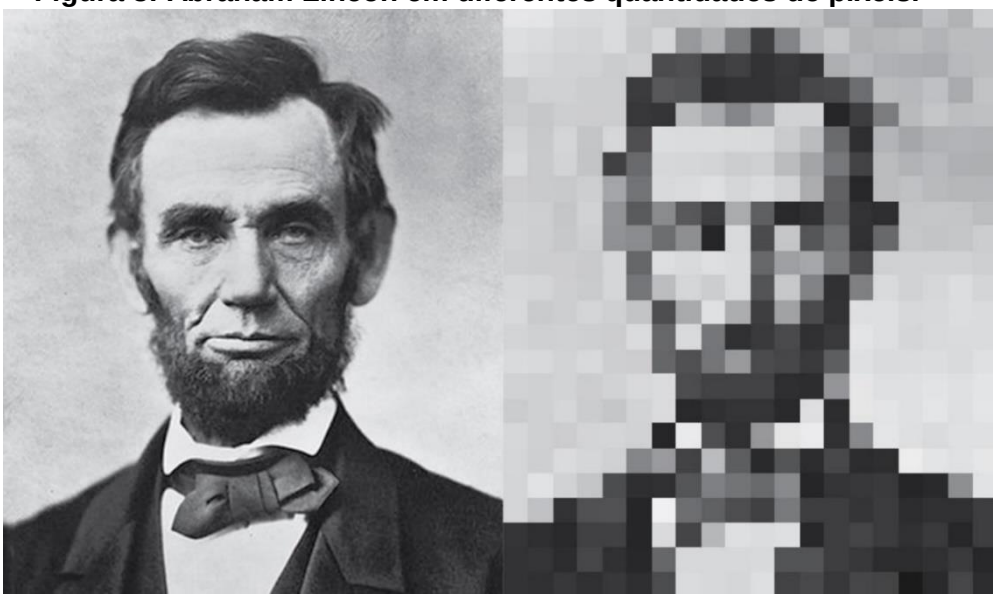
Os modelos de imagem em tons de cinza e colorida serão abordados nos próximos tópicos.

3.2.1 Imagem em tons de cinza

Uma imagem pode ser vista como uma função bidimensional contínua *f(x,y)*, na qual as variáveis x e y dessa função são coordenadas espaciais e o valor de f um ponto proporcional à intensidade luminosa (nível de cinza). Cada ponto dessa função é denominado pixel. O pixel pode ser entendido como o menor elemento em um dispositivo de exibição. Assim, o primeiro índice da função é a posição da linha na qual o pixel está, enquanto o segundo a posição da coluna (De Queiroz & Gomes, 2006; Acharya, 2005; Gonzalez, 2002; Lopes & Bonfim, 2013).

A imagem digital é composta por um número finito de elementos, chamados Pixels (Gonzalez, 2002). Na **figura 3** se faz possível visualizar a diferença em que a quantidade de Pixels em uma imagem impacta na sua qualidade de nitidez visual. A imagem de Abraham Lincon à esquerda possui 8.005.632 (3072 x 2606) pixels, já a imagem à direita é representada por apenas 480 (24 x 20) pixels (Gonçalves Filho, 2016).

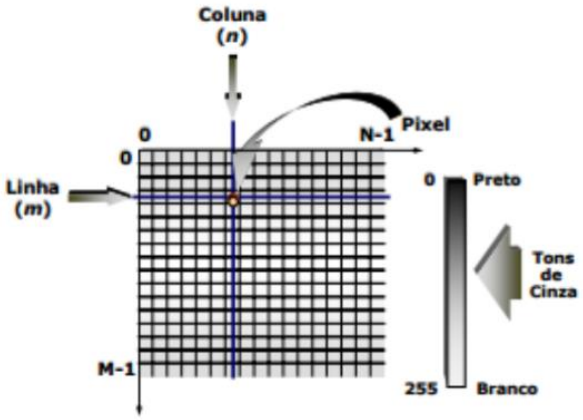
Figura 3: Abraham Lincon em diferentes quantidades de pixels.



Fonte: http://s-ben.github.io/Fractal-Mosaics/log_polar_registration_framework.html. (2018)

Se faz necessário representar imagem como arranjo bidimensional de pontos, devido à incapacidade de computadores processar imagens contínuas. Ao substituir-se o respectivo pixel pelo valor de tom de cinza, variando de 0 a 255 (1 byte), em uma matriz, onde 0 corresponde à cor preta e 255 à cor branca, há outras escalas como a 16 bits/pixel. Assim, o valor do tom de cinza do pixel da linha (m) e coluna (n) de uma imagem, será o valor do elemento (m, n) da matriz que representa esta imagem, como pode ser visualizada na **figura 4** (Queiroz, 2006).

Figura 4: Eixos (x,y) de uma imagem digital.

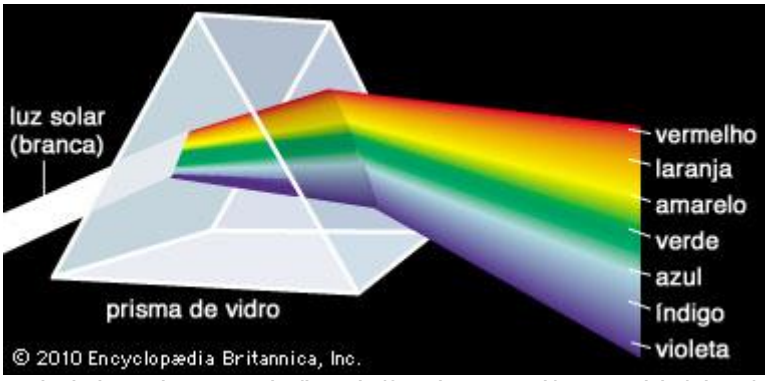


Fonte: De Queiroz & Gomes, 2006.

3.2.2 Imagem colorida

Em meados de 1660, Isaac Newton, após o experimento do prisma de vidro, que consiste em passar a luz do sol por esse prisma, descobriu que quando luz passa através dele, ela forma um desvio. Como resultado, as diferentes cores que formam a luz branca se separam. obtendo não mais a luz branca, do sol, e sim um espectro de cores no outro lado do prisma (Gonçalves Filho, 2016).

Figura 5: Prisma de vidro refratando a luz branca

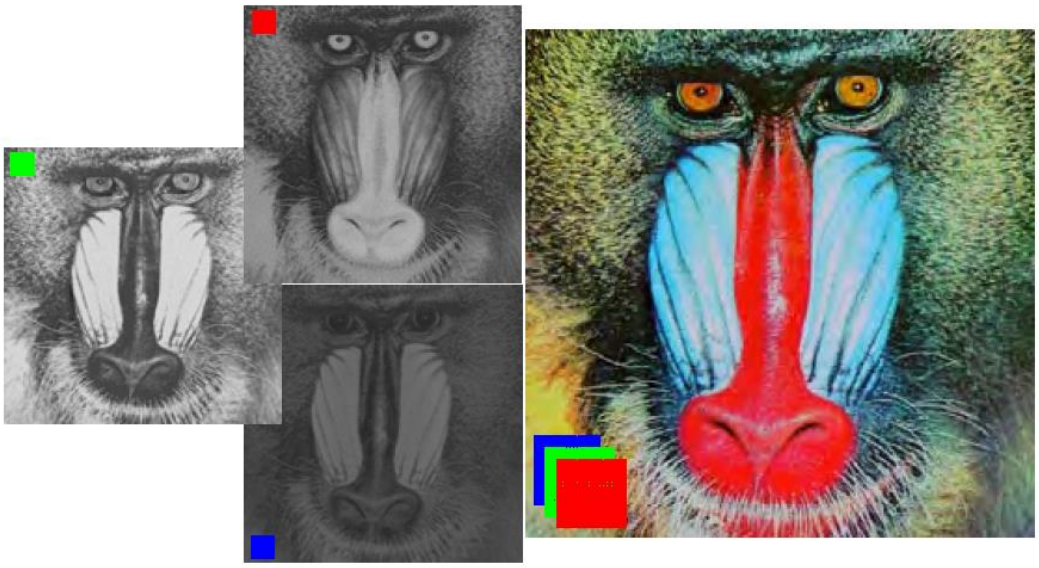


Fonte: https://escola.britannica.com.br/levels/fundamental/assembly/view/13534. (2018)

Além disso, a retina é o elemento do olho humano onde se encontram as células responsáveis por identificar as cores e luminosidade, permitindo, assim, visão a cores e com nitidez (Silva, 2013). Além disso, experimentalmente, se descobriu que os cones dos olhos humanos podem ser divididos em três categorias, correspondendo ao reconhecimento das luzes vermelha, verde e azul (Gonzalez, 2002). Dessa forma, a captação de cores pelos olhos humanos, é feita a partir da proporção destas três cores primárias: Vermelho (Red – R), verde (Green – G) e azul (Blue – B). (Gonçalves Filho, 2016). Assim, se faz possível o entendimento de como se consegue representar uma imagem digital colorida. Pode-se produzir todas as cores visíveis através das cores vermelho, verde e azul, além de sua intensidade (Gonçalves Filho, 2016). Dando origem ao sistema de cores RGB, que se trata de uma imagem composta pelas suas intensidades de vermelho, verde e azul, onde cada cor é representada por sua matriz, como mostra a **figura 6**.

O sistema RGB se trata de um sistema de síntese aditiva. Há também sistemas de síntese subtrativa, destaca-se o composto pelas cores secundárias deste: ciano (*cyan* – C), magenta (magenta – M) e amarelo (*yellow* – Y), o sistema CMY ou CMYK. Outro sistema de é o HSV (*hue* – H, *saturation* – S, *value* – V). Com estas variáveis, o modelo HSV aproxima-se muito do modelo intuitivo empregue em artes visuais que emprega os conceitos qualitativos de matriz, luz e tonalidade (Rocha, 2010) (Gonçalves Filho, 2016) (Mora, 2014).

Figura 6: Representação de uma imagem colorida digital bidimensional.

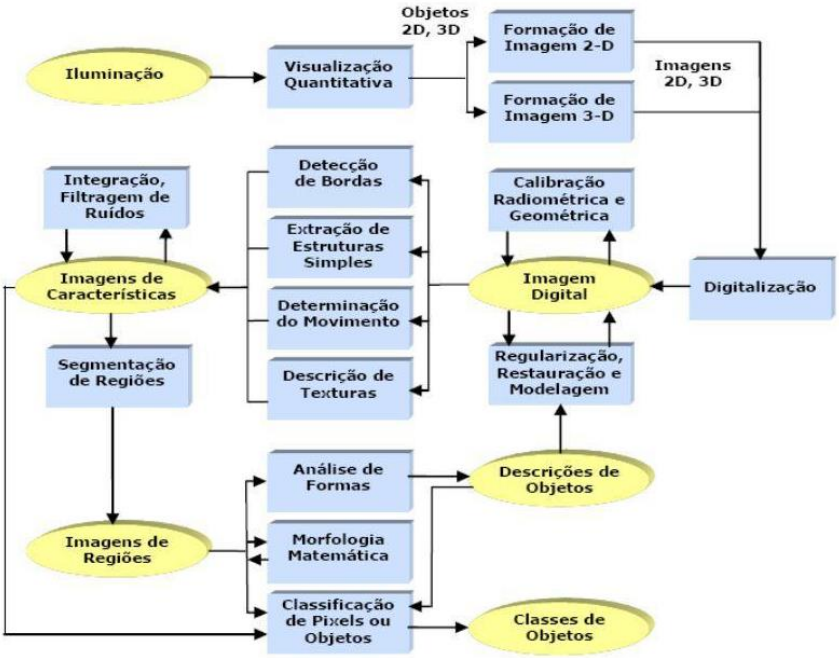


Fonte: De Queiroz & Gomes (2006)

3.3 Processamento de imagem

O processamento de imagem é uma ferramenta de crescente aplicação em diversas áreas, tendo como principais objetivos o aprimoramento de informações pictóricas para interpretações humanas e a análise automática por computador de informações extraídas de uma cena (Marques & Neto, 1999). Não é uma simples tarefa aplicar esta ferramenta, pois ela se inicia com a aquisição da imagem capturada, que corresponde à iluminação que é refletida dos objetos. Após isso, a imagem precisa ser representada de forma apropriada para tratamento computacional, envolvendo passos como a filtragem de ruídos e a correção de distorções geométricas. Dessa forma, é possível iniciar a análise e identificação de objetos. Primeiramente, características ou atributos, como bordas, texturas e vizinhanças, precisam ser extraídos da imagem. Em seguida, o objeto analisado precisa ser separado do plano de fundo, utilizando um procedimento de segmentação. A partir da forma geométrica dos objetos, resultantes da segmentação, pode-se utilizar operadores morfológicos com o objetivo de analisar, modificar e extrair informações adicionais do objeto e assim classifica-lo, tendo como objetivo o reconhecimento, a verificação ou inferência da identidade do objeto (De Queiroz & Gomes, 2006). Todas essas etapas serão aprofundadas posteriormente. Seus elementos principais estão ilustrados no fluxograma da **figura 7** a seguir.

Figura 7: Fluxograma de processamento de imagens.



Fonte: De Queiroz & Gomes (2006)

3.3.1 Sistema de visão artificial aplicado ao reconhecimento de padrões

A motivação do sistema de visão artificial é fazer com que máquinas possam enxergar, dando a elas uma capacidade semelhante a dos seres humanos. Dessa forma, são exigidas três caraterísticas principais, a saber: Uma base de dados, velocidade de processamento e capacidade de trabalhar sob condições variadas (Marques Filho & Neto, 1999).

Os avanços tecnológicos contribuíram com duas dessas características das máquinas, com os dispositivos de armazenamento cada vez maiores e o surgimento de unidades de processamento cada vez mais rápidas como CPUs, GPUs e TPUs, fornecendo condições cada vez melhores para modelar as características de base de dados e velocidade de processamento.

Fazer com que os sistemas de visão artificial trabalhem em diferentes condições de luminosidade, contraste e posicionamento relativo dos objetos em uma cena, sem perder a capacidade de interpretá-la, ainda é uma dificuldade atribuída à uma lógica computacional. Em outras palavras, os seres humanos podem identificar os amigos ou parentes em uma cena com maior facilidade e ainda conhecer se o cabelo cresceu ou se está mais magro, além das condições de luminosidade que conseguimos distinguir. Por exemplo, algumas horas do dia podem identificadas também pela luminosidade (Marques Filho & Neto, 1999).

Dessa forma, um sistema de visão artificial (SVA) pode ser utilizado como um sistema computadorizado capaz de adquirir, processar e interpretar imagens correspondentes a cenas reais. Sua interpretação pode ser o conhecimento de um padrão estabelecido por um banco de dados.

3.3.2 Problema do reconhecimento de placas

Segue adiante um exemplo do uso de SVA e de reconhecimento de padrões aplicado à problemática da identificação de placas de carro, seguindo a sequência dada por Marques Filho & Neto, 1999.

3.3.3 Domínio do problema

A problemática a ser trabalhada é fazer com que o SVA leia placas de carro e as reconheça, segundo um banco de dados com os caracteres destas placas. Dessa forma, é esperada uma sequência de três letras seguida de quatro números, separados por um hífen. Assim, ao identificar o padrão da placa, o sistema fica apto a determinar se a placa em questão está autorizada ou não. Em outras palavras, o sistema executará uma ação após a interpretação da imagem.

3.3.4 Aquisição da Imagem

A primeira etapa é a aquisição da placa e, para tal, é necessário um sensor capaz de captar a placa e digitalizá-la. Assim, pode ser utilizada uma câmera que capta a imagem real da placa por meio de frames, quadros de vídeos. Esses quadros de vídeos são imagens instantâneas que são usadas como um truque para dar movimento a um vídeo exibindo-se vários quadros consecutivamente (Machado, 2011). Ao fim dessa etapa, se tem na saída uma imagem digitalizada contendo a placa.

3.3.5 Pré-processamento

Ao adquirir a imagem, ela pode apresentar algumas imperfeições, a saber: presença de pixel ruidoso, contraste e/ou brilho inadequados, caracteres interrompidos ou indevidamente conectados, dentre outras. Dessa forma, a função do pré-processamento é ajustar a qualidade da imagem para as etapas posteriores. É um processo de baixo nível, pois trabalha diretamente com os valores de intensidade dos pixels. A imagem resultante desta etapa é uma imagem digitalizada, de melhor qualidade que a original.

3.3.6 Segmentação

A tarefa da segmentação é a divisão da imagem em áreas relativas aos objetos de interesse que a compõem. Essa tarefa é uma das mais difíceis de se implementar, apesar do simples entendimento da sua função.

É possível dividir o problema apresentado em duas etapas: no primeiro momento, os algoritmos de segmentação tentarão localizar a placa veicular do restante das informações para, no segundo momento, trabalhar sobre esta subimagem e segmentar cada dígito individualmente. Segundo essa linha de raciocínio, este bloco produzirá oito subimagens, cada qual correspondendo a um caractere da placa.

3.3.7 Extração de característica

Na etapa de extração de característica da imagem, o procedimento é extrair da imagem resultante da segmentação as devidas informações. No caso dos caracteres da placa, o processo deve extrair da imagem os caracteres com um bom poder de discriminação entre dígitos parecidos. Isso é possível devido a uma estrutura de dados adequada ao algoritmo de reconhecimento. Nessa etapa a entrada ainda é uma imagem, porém, na sua saída temos um conjunto de dados correspondente àquela imagem. Em linguagem computacional pode-se dizer que esses dados são *strings,* um conjunto de caracteres.

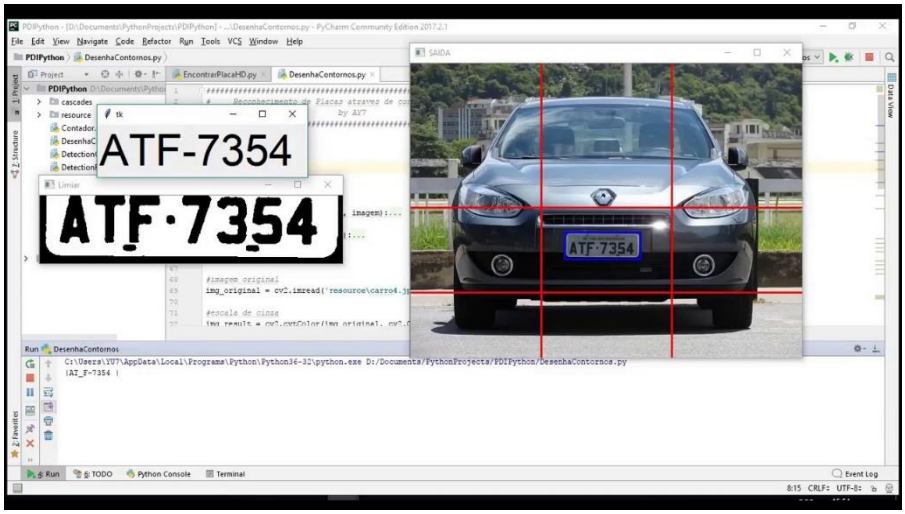
3.3.8 Reconhecimento e Interpretação

Nesta última etapa do sistema, se dá a interpretação dos dados finais recebidos da extração de características. Em outras palavras, a etapa consiste em atribuir significado a um conjunto de objetos reconhecidos. No problema proposto tem-se uma verificação dos caracteres da placa em um banco de dados de placas válidas, para descobrir se o conjunto dos oitos caracteres fazem sentido e assim tomar alguma decisão.

Para identificar placas de automóveis usando processamento de imagens, são utilizadas todas as etapas descritas anteriormente e, então, um reconhecimento de padrão da placa em questão.

Seguem exemplificadas nas **figuras 8A** e **8B** alguns reconhecimentos de placas automotivas.

Figura 8A: Imagem de reconhecimento de placa de automóvel.



Fonte: https://www.youtube.com/watch?v=oTlPAt6nupU. (2018).

Figura 8B: Imagem de reconhecimento de placa de automóvel.

Fonte: https://www.youtube.com/watch?v=cq7_L8v-55Y. (2018)

4 FERRAMENTAS

Nessa etapa será feito um levantamento das principais ferramentas usadas para o projeto.

4.1 PYTHON

Python é uma linguagem de programação orientada ao objeto que foi criada por Guido Van Rossum em 1991. Os objetivos do projeto da linguagem eram a produtividade e a legibilidade. Em outras palavras, *Python* é uma linguagem que suporta programação modular e funcional, além da orientação a objetos e foi criada para produzir código bom e fácil de manter, de maneira rápida. Entre as características da linguagem que ressaltam esses objetivos estão (PyScience-Brasil. Acesso em outubro 2018):

- Baixo uso de caracteres especiais, o que torna a linguagem muito parecida com *pseudo-código executável*;
- Uso de identação para marcar blocos;
- Quase nenhum uso de palavras-chave voltadas para a compilação;
- Coletor de lixo para gerenciar automaticamente o uso da memória, entre outros.

A escolha dessa linguagem foi devido ao grande número de *frameworks,* e bibliotecas desenvolvidos por terceiros que podem ser adicionados. Também inclui recursos encontrados em outras linguagens modernas, tais como geradores, introspecção, persistência, metaclasses e unidades de teste (Borges, 2014).

4.2 OPENCV

O OpenCV (Open Source Computer Vision Library) é uma biblioteca multiplataforma, totalmente livre ao uso acadêmico e comercial, para o desenvolvimento de aplicativos na área de Visão Computacional. Esta biblioteca foi desenvolvida nas linguagens de programação C/C++ (Sobral, 2013). Possui interfaces C ++, *Python* e *Java* e suporta *Windows, Linux, Mac OS, iOS e Android* (OpenCV, acesso em outubro, 2018).

Esta biblioteca possui mais de 2.500 algoritmos otimizados, tanto clássicos como no estado da arte de visão computacional e inteligência artificial. Esses algoritmos podem ser usados para detectar e reconhecer faces, identificar objetos, extrair modelos de objetos, converter em tons de cinza, modificar a resolução, entre outros (Cunha, 2013). Dessa forma, se faz útil em programas de processamento de imagens devido ao grande número de funções para auxiliar.

4.3 TESSERACT

O *Tesseract-OCR* (*Optical Character Recognition*) 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 à comunidade de programadores, essa biblioteca foi modificada, permitindo o funcionamento dela em programação *Python*. Por este motivo, o nome dessa biblioteca para a linguagem *Python* tornou-se "*PyTesseract"* (Antonello & Leite, 2017).

4.4 ARDUINO

O Arduino é uma plataforma de *hardware open source*, de fácil utilização para a criação de dispositivos que permitam interação com o ambiente (Souza et al., 2011). A placa de Arduino é baseada na linguagem C/C++. Entretanto pode-se utilizar a biblioteca PyFirmata para auxiliar na comunicação entre o microcontrolador e o computador usando a linguagem *python.* Dessa forma, teremos uma interação do software e o Arduino (Galiza, 2018).

Essa placa pode ser empregada na criação de protótipos. A figura X mostra um exemplo de um protótipo usando Arduino.

Figura 9: Protótipo de um carrinho com Arduino

Fonte:https://blog.usinainfo.com.br/carrinho-arduino-controlado-por-bluetooth-e-sistema-android/. (2018)

5 DESENVOLVIMENTO DO CÓDIGO

O código pode ser dividido em 5 etapas: Aquisição da imagem, pré-processamento, segmentação, extração de característica, reconhecimento, interpretação e execução da ação.

5.1 AQUISIÇÃO DA IMAGEM

Para a aquisição da imagem são utilizadas uma câmera (webcam) e algumas funções da biblioteca OpenCV. Ao iniciar a rotina, é preciso capturar a imagem por meio de frame. Esse papel é realizado com o auxílio da função ***cv2.VideoCapture***.

Há a necessidade de ler a imagem e salvá-la durante a execução do programa. Para isso temos ***cv2.imread*** e ***cv2.imwrite***. *A função* ***cv2.imwrite*** é utilizada para salvar a imagem. É utilizada quando se deseja salvar uma imagem importante na rotina como, por exemplo, quando na segmentação se quer apenas uma parte da imagem. A função ***cv2.imread*** realiza a posterior leitura da uma imagem previamente salva.

Um gatilho não automático é usado para garantir a aquisição do frame desejado. Dessa forma, as outras etapas do processo só ocorrem quando é disparado um gatilho (LPR – License Plate Recognition Brazil – Parte 1, acesso em outubro, 2017).

O gatilho manual desse sistema de visão computacional é implementado com na linha de código **cv2.waitKey(1) & 0xff == ord('q')**. ***cv2.waitKey*** retorna um valor inteiro de 32 bits. A entrada principal está em ASCII, que é um valor inteiro de 8 bits. Logo, só se deve preocupar com os 8 bits e deseja-se que todos os outros bits sejam equivalentes a 0. Para isso é usado **0xff** (constante hexadecimal). A função sai da rotina quando é pressionado o botão “q” do teclado (Paes, 2017; Antonello & Leite, 2017).

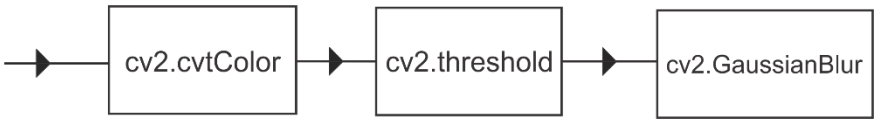
Quando temos um monitoramento, para ativação do gatilho, é interessante que se mostre algumas imagens geradas. A função que desempenha isso é a **cv2.imshow**, que cria uma janela com a imagem. Ao final é preciso fechar as janelas de uma forma automática. Assim, **cv2.destroyAllWindows**, fecha todas as janelas.

Uma vez que não é preciso usar mais a câmera, deve-se liberá-la através da função **video.release**.

5.2 PRÉ-PROCESSAMENTO

Para o pré-processamento, foram utilizadas funções da biblioteca *OpenCV2* para melhorar a imagem, de acordo com seus objetivos, além de converter em branco e preto para melhorar a extração da informação.

Figura 10: Fluxograma de pré-processamento



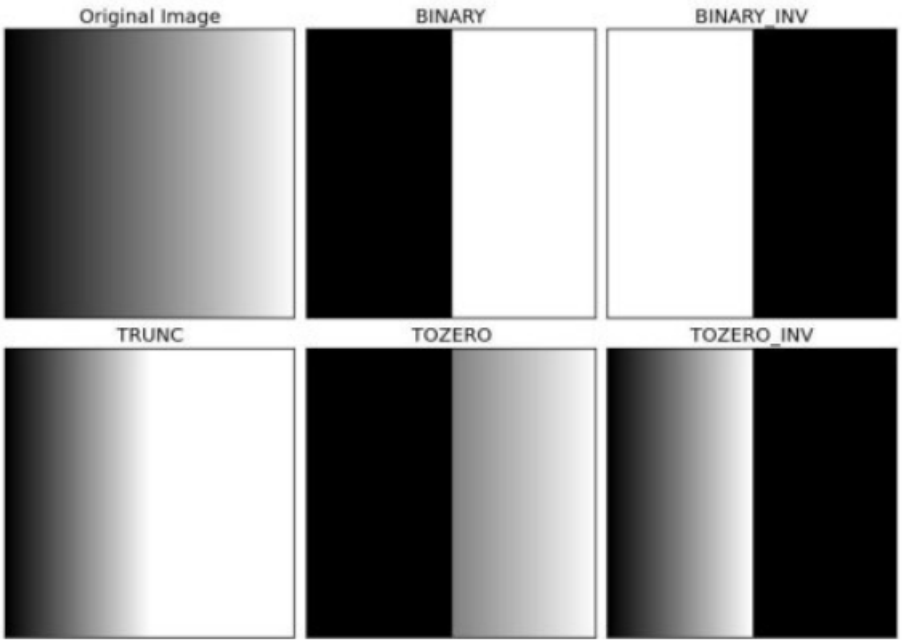
Fonte: O autor

A **figura 10** ilustra o fluxograma do pré-processamento inicial, onde a entrada é uma imagem que passa pelas três etapas em sequência.

Algumas funções da biblioteca *OpenCV* apenas aceitam imagens com espectro de cores em tons de cinza. Logo, para converter imagens de um espaço de cores em outro utiliza-se a função **cv2.cvtColor** que converte a imagem RGB capturada para tons de cinza.

Para uma melhor interpretação é preciso limiarizar a imagem. Se o valor do pixel for maior que um valor limite, é atribuído um valor a ele (pode ser branco), caso contrário, é atribuído outro valor (pode ser preto), obtendo uma imagem limiarizada em branco e preto. A função usada é ***cv2.threshold***. O primeiro argumento é a imagem de origem, que deve ser uma imagem em tons de cinza. O segundo argumento é o valor limite usado para classificar os valores de pixel. O terceiro argumento é o *maxVal*, que representa o valor a ser dado se o valor do pixel for maior ou menor que o valor do limite. O OpenCV fornece diferentes estilos de limiar e é decidido pelo quarto parâmetro da função. Ele possui diferentes tipos, a saber: ***cv2.THRESH_BINARY; cv2.THRESH_BINARY_INV; cv2.THRESH_TRUNC; cv2.THRESH_TOZERO; cv2.THRESH_TOZERO_INV.*** Exemplos do resultado desses tipos estão demonstrada na **figura 11**. O tipo utilizado foi o *binary*. Sua função é transformar a imagem em duas cores (OpenCV, Acesso em 2018).

Figura 11: Resultado dos diferente limiares.



Fonte: https://docs.opencv.org/3.4.3/d7/d4d/tutorial_py_thresholding.html (2018)

Duas saídas são obtidas. A primeira é um **retVal**. A segunda saída é a imagem limiarizada. Após se obter essa imagem é preciso desfocar a imagem e remover o ruído, para isso usa-se **cv2.GaussianBlur** obtendo uma imagem de melhor qualidade para a interpretação.

5.3 SEGMENTAÇÃO

Na etapa de segmentação é preciso separar o objeto de estudo da imagem como um todo. É uma das etapas mais importantes e difíceis do processamento de imagem. Para esse caso, precisa-se pegar apenas a placa de carro. Existem algumas formas de fazer esse tipo de segmentação que podem usar *machine learning* (aprendizado de máquina) com o uso de *Haar Cascades* (Antonello & Leite, 2017). Porém, foi utilizada uma ideia mais simples que foi aproveitar a geometria retangular das placas, limitando o espaço a procurar esse retângulo.

Os contornos são uma característica útil para análise de formas e detecção e reconhecimento de objetos. Para identificar contornos se tem a função **cv2.findContours**. Esta função produz os contornos e a hierarquia deles numa

lista. Cada contorno individual é um vetor de coordenadas (x, y) de pontos de limite do objeto.

Ao adquirir os contornos é possível atribuir suas dimensões a uma variável chamada perímetro usando a função **cv2.arcLength** e, assim, identificar se é pequeno ou grande demais, definido por parâmetro. Dessa forma, restringindo uma área aproximada de uma placa de carro. Em seguida, é aproximado o perímetro da forma correspondente, caso não se tenha obtido a forma desejada previamente. Essa aproximação é feita com o auxílio da função **cv2.approxPolyDP**. Neste, o segundo argumento é chamado *epsilon*, que é a distância máxima do contorno ao contorno aproximado. É um parâmetro de precisão. Uma seleção inteligente de *epsilon* é necessária para obter a saída correta. Abaixo, na **figura 11**, a linha verde mostra a curva aproximada para *epsilon* = 10% do comprimento do arco. A terceira imagem mostra o mesmo para *epsilon* = 1% do comprimento do arco (opencv.org, acesso em outubro de 2018).

Figura 12: Imagem de aproximação de contornos com a função cv2.approxPolyDP.

Fonte: https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html (2018)

No intuito de verificar se o contorno encontrado possui 4 lados, ou seja, é um quadrilátero, usa-se a lógica condicional "**if len(approx) == 4"**. Se há um quadrilátero, pega-se suas dimensões e atribui a variáveis que correspondem às coordenadas (x,y) da origem da imagem e largura (lar) e altura (alt) usando **cv2.boundingRect**. Essa função encontra um retângulo reto, não considera a rotação do objeto. Portanto, a área do retângulo delimitador não será mínima. Seu argumento é imagem (Paes, 2017).

Também há situações que se deseja desenhar na imagem, **cv2.line** e **cv2.rectangle**. O primeiro desenha uma linha na imagem. Seus parâmetros de entrada são coordenadas de dois pontos (x, y) e a cor da linha a ser desenhada. O segundo, desenha um retângulo direito, simples, grosso ou preenchido. Seus parâmetros são: Imagem, vértice do retângulo, vértice do retângulo oposto ao anterior,

especificação alternativa do retângulo desenhado, cor, espessura das linhas, tipo de linha e o número de bits fracionários (Antonello & Leite, 2017).

5.4 EXTRAÇÃO DE CARACTERÍSTICA

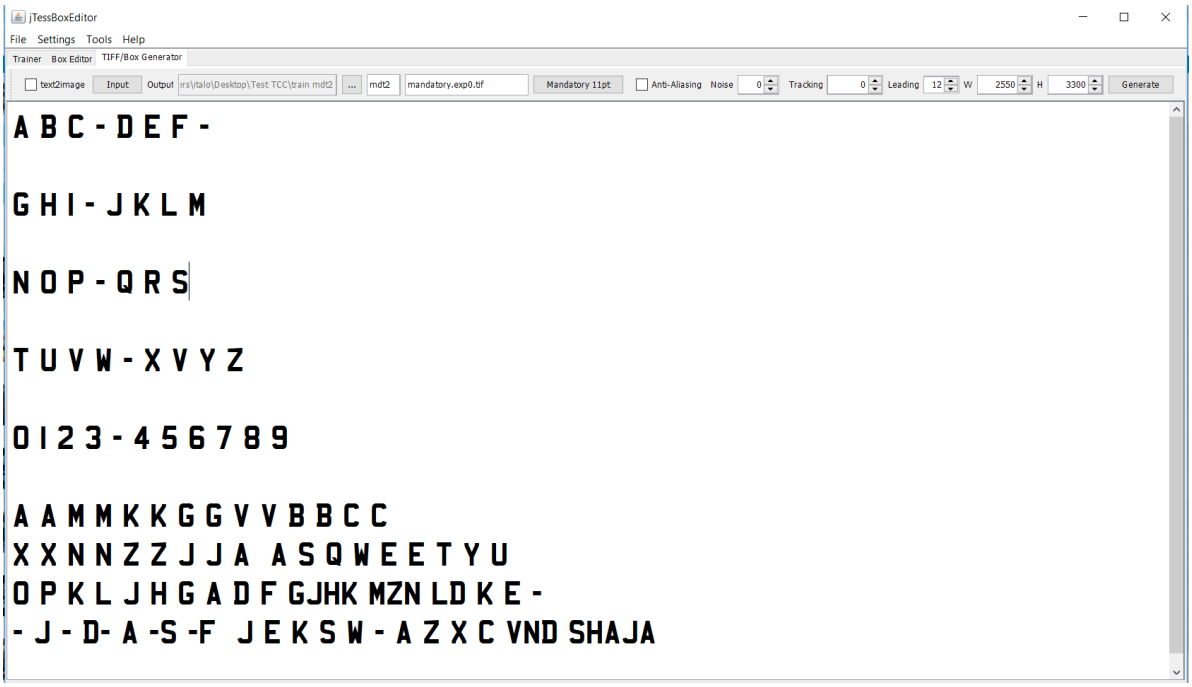
Uma das etapas mais importantes do programa é como será extraída a informação da placa. O método usado foi a utilização da biblioteca *tesseract-ocr.* Essa função é usada para scanear livros e converter em textos editáveis. A função que nos ajuda a fazer isso é a pytesseract.image_to_string. Ela tem a função de converter uma imagem em string, uma cadeia de caracteres. Seus argumentos são a imagem e a linguagem usada para essa transformação. Foi preciso criar uma “linguagem” de placa brasileira a qual foi dado o nome de ***mdt2***. O tesseract trabalha com reconhecimento de padrões, o qual precisa ser “treinada” através de exemplos desta linguagem.

5.5 TREINAMENTO DO TESSERACT PARA A LINGUAGEM “MDT2”

Como visto anteriormente, as placas brasileiras devem utilizar a fonte *Mandatory*. Como não há um dicionário no *PyTesseract* com essa fonte para ler caracteres, é preciso criá-lo. Para essa criação foram usados dois programas extras: *JTessBoxEditor* e o *Serak Tesseract Trainer* (Antonello & Leite, 2017).

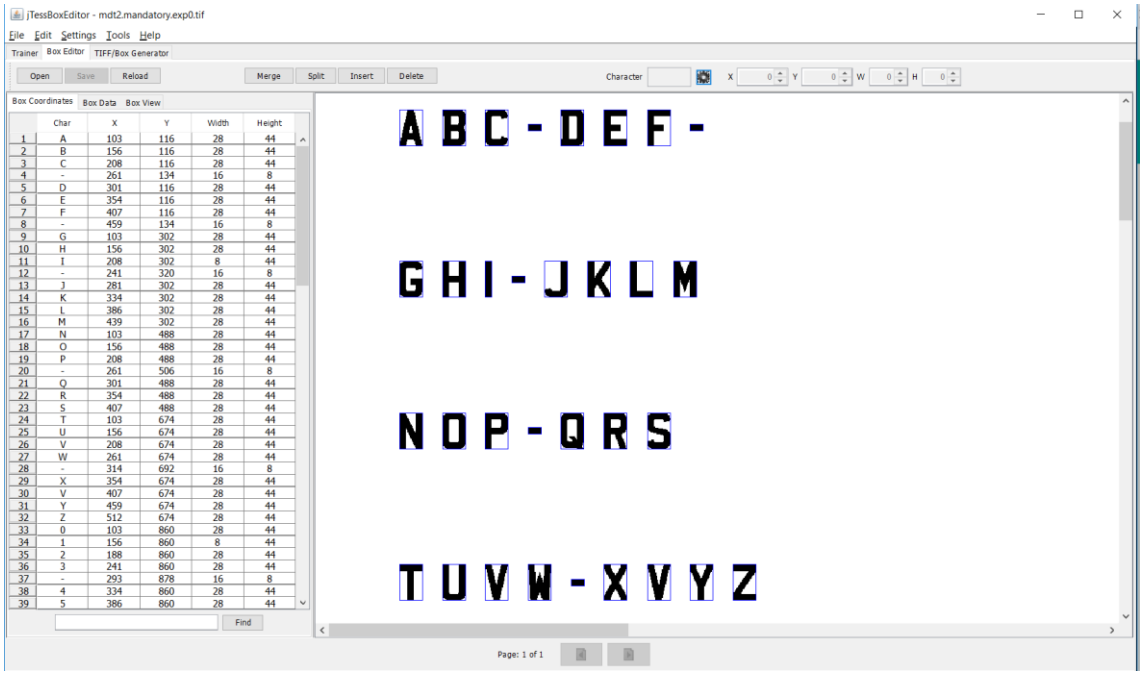
O programa *Serak Tesseract Trainer* precisa de um arquivo “.TIFF” para executar o treinamento do dicionário. O *JTessBoxEditor* é responsável por fazer essa transformação de um arquivo “.txt” para um arquivo “.TIFF”, no qual são salvos as coordenadas dos caracteres com a fonte *Mandatory*. Dessa forma, criou-se um arquivo *.txt* com todas as letras do alfabeto, números de 0 a 9 e hífen. Em seguida abrindo este arquivo no programa *JTessBoxEditor* e definida a fonte como *mandatory*, então, gerado o arquivo “.TIFF”. Na **figura 12** é possível ver o funcionamento do programa ao abrir o arquivo “.txt.”. Já na **figura 13** temos o resultado da transformação, o arquivo “.TIFF” propriamente dito.

Figura 12: *TIFF/Box Generator do JTessBoxEditor.*



Fonte: O autor

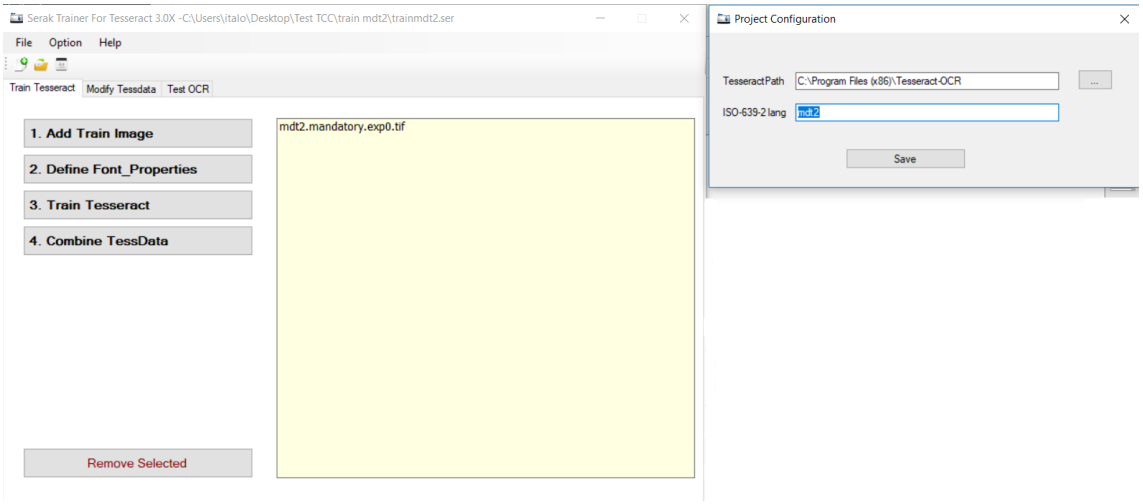
Figura 14: Coordenadas do arquivo TIFF no Box Editor do JTessBoxEditor.



Fonte: O autor

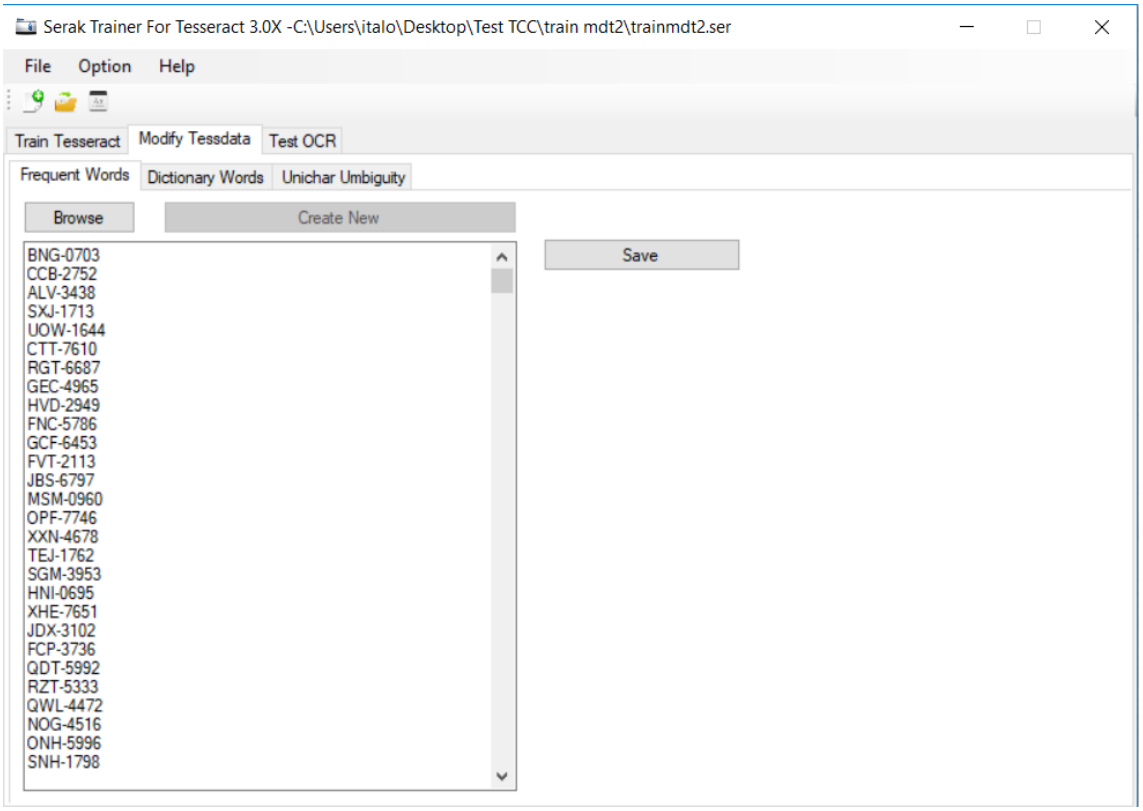
Para fazer o treinamento tendo já o arquivo da fonte, usa-se o programa *Serak Tesseract Trainer*, no qual carrega-se o arquivo ".TIFF" e introduz vários exemplos de placas para o treinamento.

Figura 15: Serak Tesseract Trainer em funcionamento e configuração.



Fonte: O autor

Figura 16: Palavras frequentes para o arquivo ".traineddata".



Fonte: O autor

Após abrir o programa, criamos um novo arquivo e abrimos o arquivo ".TIFF" criado anteriormente. Configuramos como será o nome do dicionário para "mdt2". Criamos várias palavras frequentes usando três letras, hífen e quatro números, em outras palavras, placas. Foram feitos os 4 passos vistos na **figura 14**, assim, gerando

o treinamento final que deve ser direcionado para a pasta "...*Tesseract-OCR**tessdata*".

Após a identificação da *string* da placa, foi preciso fazer uma separação entre a parte numérica e a alfabética. Desta forma, pode-se reparar possíveis erros, tal como, a troca da letra O pelo numeral 0 e vice-versa, sendo, para tal, elencado o código que se encontra descrito na **tabela 1** (Antonello & Leite, 2017).

TABELA 1: TRATAMENTO DA STRING

```
caracs = pytesseract.image_to_string(imagem, lang='mdt2')
caracs = caracs.replace(' ', '')
caracs = caracs.replace("-", "")
letras = caracs[:3]
num = caracs[3:]
num = num.replace("-", "")
letras = letras.replace("-", "")
num = num.replace('O', "0")
num = num.replace('Q', "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
```

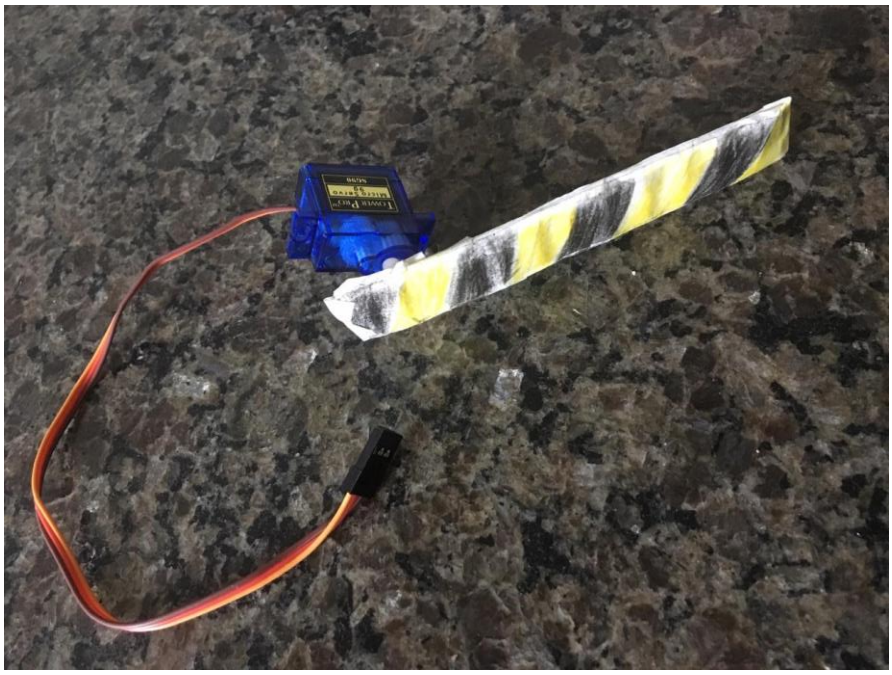
```
plate_= placa_escrita[:8]
print (placa_escrita[:8])
```

Fonte: O autor

5.6 RECONHECIMENTO E INTERPRETAÇÃO

O objetivo da extração da informação da placa é comparar com um banco de dados de um arquivo Excel e, assim, tomar uma decisão quanto à abertura ou não da cancela, implementada como mostra a **figura 16**. Sua movimentação é feita com um auxílio de uma placa *Arduino*.

Figura 17: Servo motor fazendo papel de uma cancela.



Fonte: O autor

O banco de dados das placas foi feito usando um arquivo Excel, vide exemplificado na **Tabela 2**. O Python conta com uma biblioteca chama **Pandas** que, entre outras coisas, auxilia a abertura e manipulação de arquivos ".xlsx". Neste banco de dados foram criados carros e placas fictícias, assim como seus respectivos donos, meramente ilustrativos.

Tabela 2: Exemplo do banco de dados.

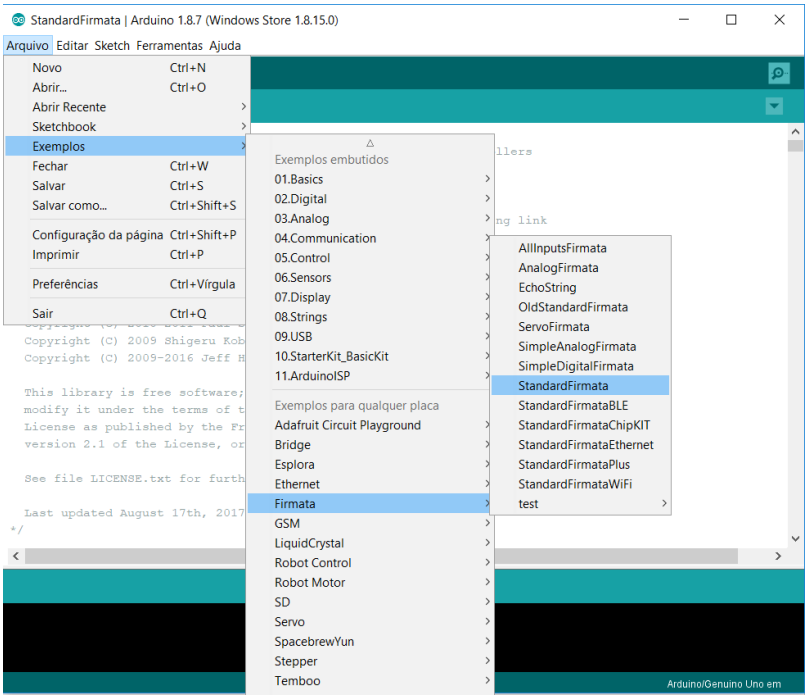
Nome	Modelo do Carro	Placa
José	Gol	PMN-2535
Italo	Camaro	ABC-1234
Maria	BH20	CSC-2013
Stan lee	BMW	XXT-2215
Marcos	Celta	PUV-5373

Fonte: O autor

A abertura do arquivo é feita usando a função **pd.read_excel**. Após a leitura, é possível fazer a comparação da placa extraída durante a execução do algoritmo com as placas presentes no banco de dados.

A comunicação com o *Arduino*, foi feita com a biblioteca pyFirmata, estando o Arduino executando o código StandardFirmata, presente como exemplo na própria IDE Arduino. Dessa forma, manda-se uma mensagem para o *Arduino* para, assim, executar uma rotação no servo motor simulando a abertura de uma cancela e, após 10 segundos, volta à posição inicial.

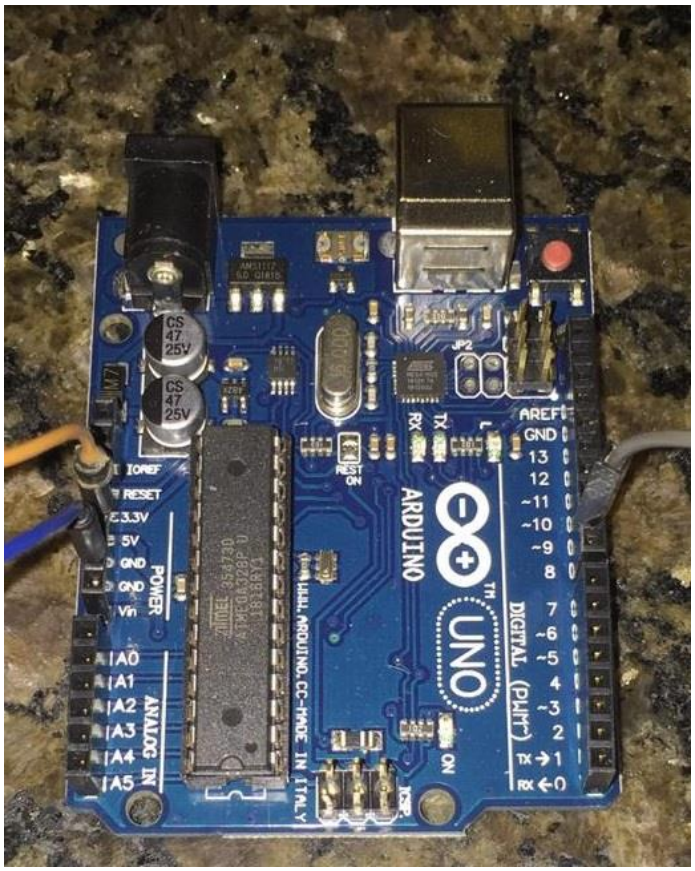
Figura 18: Procedimento para usar o exemplo StandardFirmata no Arduino IDE.



Fonte: O autor

A configuração eletrônica do Arduino pode ser visualizada na figura 15, na qual, é ligado um fio no 5V no GND (terra) e no pino 9. Por padrão do servo motor, o fio 5V é ligado ao fio vermelho, GND ao fio marrom e o pino ao fio laranja.

Figura 19: Imagem da configuração eletrônica do Arduino usado no projeto.



Fonte: O autor

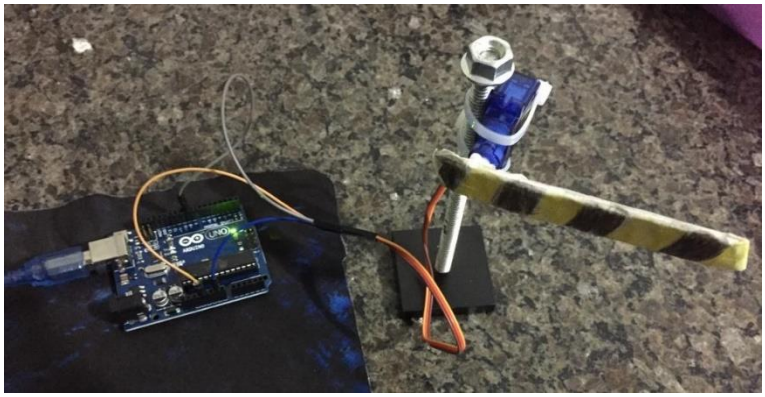
O código usado foi uma junção de dois códigos, modificando seus parâmetros e melhorando-os, adicionando bibliotecas e funções para implementação do *Arduino* e implementação de biblioteca, usando o banco de dados como um arquivo Excel (Antonello & Leite, 2017; Paes, 2017).

6 TESTES

O código usado foi uma junção de dois códigos, modificando seus parâmetros e melhorando-os, adicionando bibliotecas e funções para implementação do *Arduino* e implementação de biblioteca, usando o banco de dados como um arquivo Excel (Antonello & Leite, 2017; Paes, 2017). Esse código completo, usado nos testes, pode ser visto no anexo 1.

O sistema para os testes teve o *layout* do *Arduino* e servo motor ilustrado na **figura 19**. Os testes foram feitos com 10 placas encontradas na internet. A aquisição foi feita com o auxilio de um aparelho celular com a imagem da placa no *display,* ilustrado na **figura 20**.

Figura 20: *Layout* do sistema Arduino e Servo motor.



Fonte: O autor

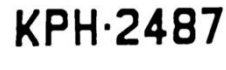
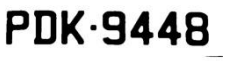
Figura 21: Demonstração do uso do aparelho celular como auxiliador na aquisição da imagem.

Fonte: O autor

Cada placa foi testada 5 vezes, modificando sua angulação e movimentação com relação ao ângulo de iluminação. Ao obter um resultado satisfatório em um teste, concluiu-se que aquela placa foi identificada no programa e, assim, que o programa foi funcional para aquela placa. A **tabela 3** demonstra os resultados, onde na primeira

coluna temos o número do teste, na segunda coluna temos a imagem da placa usada, na terceira coluna temos os caracteres da placa real, na quarta coluna estão as imagens após o processamento, a imagem onde, de fato, serão extraído os caracteres e na quinta e última coluna temos uma imagem do resultado impresso pelo *Python* da string identificada.

Tabela 3: Testes do código usando 10 placas

Número	Imagem	Placa	Imagem de identificação	Imagem do resultado
1		ABC-1234		ABC-1234 ABCI234I
2		BCD-3456		PIS-A0J0 PISAOJOSC005IIA5 BCD3456
3		KPH-2487		KPH-2487 KPH24B7
4		KYK-8608		KYK-8608 KYK8608
5		PDK-9448		PDK-9448 PDK9448
6		PFU-6000		PFU-6000 PFU6000
7		PLA-0000		PLA-0000 PLAOOOQI
8		CSC-2013		CSC-2013 CSC2OI3
9		KVA-1954		-

10		NQC-0876		NOC-0876 NOC0876

Fonte: O autor

Após os testes, identificamos que 70% das placas testadas foram identificadas corretamente. A placa de número 2 na tabela, obteve a *string* correta, devido aos caracteres situados na parte superior da placa referente a região. Já a placa de número 9 na tabela não obteve nenhum reconhecimento. O motivo identificado foi a iluminação da imagem e iluminação ambiente do teste. Dessa forma, ao executar a função **threshold**, resultou em uma imagem como vista na **tabela 3**. Entretanto, essa falha pode se entender como fora das limitações, citadas no próximo tópico.

Outro tipo de erro foi encontrado na placa de número 10. Essa placa possui a letra “Q” em sua parte alfabética e assim não se obteve um resultado satisfatório da função **pytesseract.image_to_string**, uma vez que ela troca a letra “Q” pela letra “O”.

O sistema de abertura da cancela e busca do banco de dados não apresentou falhas em nenhum dos testes. Assim, a cancela foi aberta quando identificada a placa no banco de dados e manteve-se fechada quando não.

6.1 LIMITAÇÕES

Para a execução dos testes, foi necessário executar uma edição em algumas placas para que restasse apelas o objeto desejado, três letras, quatro números e o hífen. Essa edição foi feita em placas em que ao usar a função **pytesseract.image_to_string** não era possível encontrar nada pois havia uma confusão de contornos. Nessas situações houve uma falha com relação ao pré-processamento.

Além disso, a iluminação ambiente e o ângulo de captura são variáveis bastante relevantes, podendo comprometer a identificação. Na situação hipotética do projeto se tem um ambiente estático, a entrada de um estacionamento por exemplo, além de sua iluminação manipulável, com luzes extras em posições desejadas, a fim de chegar próximo ao ideal para o funcionamento do equipamento.

7 APRENDIZADOS E SUGESTÕES

Houve a tentativa de se utilizar do *Haar-Cascad* como método de segmentação para se ter um equipamento mais automático, porém não se obteve sucesso. Esse método consiste em treinar o reconhecimento do padrão por um banco de dado. No experimento do trabalho foi utilizado 500 imagens com placas de carro no treinamento do *Haar-Cascad*. É um trabalho mecânico, fazendo de reconhecimento manual de cada foto, porem se feito corretamente é funcional.

Uma sugestão para não ser necessário o uso do notebook no equipamento é a utilizar de um *Raspbarry Pi*, em que é possível carregar essa placa com o programa em *Python*, carregar um banco de dados com placas registadas e fazer o controle do servo motor. Dessa forma, se tem um equipamento mais automático.

8 CONCLUSÃO

O trabalho desenvolvido identificou placas de carro com a finalidade de obter um reconhecimento da placa para a liberação de uma cancela. Para esse trabalho, foi utilizado um código escrito na linguagem *python* auxiliado de suas bibliotecas, principalmente *OpenCV*, *Pytesseract*, *Pandas* e *pyFirmata*. Foi utilizada também uma placa de Arduino, um notebook, uma *webcam* e um servo motor.

O presente projeto foi capaz de identificar, com algumas limitações, os caracteres de uma placa de carro, sendo estas, adquiridas de uma câmera de vídeo em tempo real. Além de poder tomar decisões com os caracteres identificados, também foi possível a comunicação com uma placa de *Arduino* e, assim, com um servo motor acoplado.

Espera-se que esse projeto possa auxiliar em trabalhos futuros, que visem o reconhecimento de padrões e visão artificial de máquina, sendo necessário, melhorar os parâmetros, independente da linguagem de programação utilizada.

Pode-se concluir que, com a visão artificial, a tecnologia e o reconhecimento de padrões é possível criar soluções para melhorar a vida social, seja no tráfego de veículos, na medicina, na indústria, entre outras áreas.

REFERÊNCIAS

ACHARYA, Tinku; RAY, Ajoy K. **Image processing**: principles and applications. John Wiley & Sons, 2005.

ANTONELLO, Ricardo; LEITE, Leonardo. **Identificação automática de placa de veículos através de processamento de imagem e visão computacional**. Artigo cientifico – Instituto Federal de Santa Catarina, campus Luzerna, 2017.

BORGES, Luiz Eduardo. Python para desenvolvedores: aborda python 3.3. Novatec Editora, 2014.

CARVALHO, Marco Antonio. Processamento digital de imagens aplicações. CESET/UNICAMP, 2004.

Conheça o significado das placas de carro. *Disponível em:* <http://blog.itaro.com.br/2016/02/conheca-o-significado-das-placas-de-carro/>*. Acesso em: 03 out, 2017.*

CUNHA, André Luiz Barbosa Nunes da. Sistema automático para obtenção de parâmetros do tráfego veicular a partir de imagens de vídeo usando OpenCV. 2013. Tese de Doutorado. Universidade de São Paulo.

DE QUEIROZ, José Eustáquio Rangel; GOMES, Herman Martins. Introdução ao processamento digital de imagens. **RITA**, v. 13, n. 2, p. 11-42, 2006.

DE SOUZA, Anderson R. et al. A placa Arduino: uma opção de baixo custo para experiências de física assistidas pelo PC. **Revista Brasileira de Ensino de Física**, 2011.

FACON, Jacques. Técnicas de processamento digital de imagens aplicadas à área da saúde. XIII Escola Regional de Informática da SBC-Paraná, 200-?.

GALIZA, Newton. **PyFirmata, programando Arduino com Python**. Disponível em: < https://medium.com/@newtongaliza/pyfirmata-programando-arduino-com-python-155ba5e7d965 >. Acesso em: 28, novembro. 2018.

GARCIA, Karla Maria. **Sistema de controle de acesso veicular utilizando tecnologia rfid**. Monografia do curso eletrônica – Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, 2013.

GONÇALEZ TENORIO, Aurio et al. **Implementação de um compressor de imagens na linguagem Python usando a transformada Wavelet**. Dissertação em engenharia elétrica e computação – universidade federal de Campinas 2003.

GONÇALVES FILHO, Paulo José. **Sensoriamento visual aplicado à competição "ieee humanoid robot racing" com o autônomo humanoide "nao".** TCC (Trabalho de conclusão de curso em engenharia mecânica) – Universidade Federal de Pernambuco, 2016.

GONÇALVES, Wesley Antonio. A grande revolução da tecnologia da informação na educação. Faculdade Atenas, 2006.

GONZALEZ, R., WOODS, P. Digital imagem Processing. Prentice Hall, 2002, 2. ed.

GOTARDO, Reginaldo Aparecido. Linguagem de Programação. Rio de Janeiro: Seses, 2015.

JARDIM, Laercio Arraes. **Sistema de visão robótica para reconhecimento e localização de objetos sob manipulação por robôs industriais em células de manufatura**. Dissertação (Mestrado em Sistemas Mecatrônicos) – Universidade de Brasília, Brasília, 2006.

LIMA, Antonio Vinicius G. Processamento de imagens para identificação de placas de automóveis, 2009.

LOPES, D.; SILVA, FH da; BONFIM, Matheus Ferreira. **Desenvolvimento do algoritmo para processamento de imagens digitais para diagnóstico de melanoma**. 2013. Tese de Doutorado. Tese (Doutorado), Centro Universitário Católico Lasensiano Auxilium-Ara ̧catuba SP.

LPR – License Plate Recognition Brazil – Parte 1. *Disponível em:* < https://www.dobitaobyte.com.br/lpr-license-plate-recognition-brazil-parte-1/ >*.* Acesso em: 02 out, 2017.

MACHADO, Jonathan. **O que são frames por segundo?** 2011. Disponível em: < https://www.tecmundo.com.br/video/10926-o-que-sao-frames-por-segundo-.htm>. Acesso em: 28 nov, 2018.

MARQUES FILHO, Ogê; NETO, Hugo Vieira. Processamento digital de imagens. Brasport, 1999.

MORA, André Damas; FONSECA, José Manuel. Metodologia para a detecção de artefatos luminosos em imagens de retinografia com aplicação em rastreio oftalmológico. **RISTI-Revista Ibérica de Sistemas e Tecnologias de Informação**, 2014.

MOREIRA, Fabiano Cordeiro et al. **Reconhecimento e classificação de padrões de imagens de núcleos de linfócitos do sangue periférico humano com a utilização de redes neurais artificiais**. Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Ciência da Computação, 2002.

NUNES, Marcelo Pereira. **Aspectos formais da linguagem python**. Relatório técnico, Fundaçao Universidade Federal do Rio Grande-FURG, 1998.

OpenCV. Disponível *em:* < https://opencv.org/>. Acesso em: 19, outubro. 2018.

PAES, Alexandre Santos Lima. Reconhecimento automático de placas automobilísticas, 2017.

PONCE, Jean et al. Computer vision: a modern approach. **Computer**, v. 16, n. 11, 2011.

PyScience-Brasil. Python: O que é? Por que usar? *Disponível em:* < *http://pyscience-brasil.wikidot.com/python:python-oq-e-pq >.* Acesso em: 21, outubro. 2018.

ROCHA, Ana Maria AC et al. Processamento de imagem digital com MatLab: uma aplicação em ambiente industrial. ENEGI 2013 – Atas do 2º Encontro Nacional de Engenharia e Gestão Industrial, 2013.

ROCHA, J. C. Cor Luz, Cor Pigmento e os Sistemas RGB e CMY, artigo, 2010.

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.

SILVA, J. V.; PINTO, H. S. R.; FERREIRA, B. F. A. Princípios Da Oftalmologia Anátomo-Histologia Funcional do Olho, artigo – UNIVERSIDADE DO CEARÁ, 2013.

SIQUEIRA, Mozart Lemos de. **Reconhecimento automático de padrões em imagens ecocardiográficas**. Dissertação (Doutorado) – Universidade Federal do Rio Grande do Sul, instituto de informática, 2010.

SOBRAL, Andrews. BGSLibrary: An opencv c++ background subtraction library. In: **IX** Workshop de Visao Computacional, 2013.

VACCARI, Maria Beatriz. **Placas de carro: conheça os seus significados**, 2016. Disponível em: < https://garagem360.com.br/placas-de-carro-conheca-os-seus-significados/ >. Acesso em: 28 nov, 2018.

ZIBETTI, Marcelo Victor Wust. Visão de máquina e suas aplicações na automação industrial. 2011.

ANEXO A

Código final usado para os testes

```
from PIL import Image
import numpy as np
import tkinter
import pytesseract
import cv2
import pandas as pd
from pyfirmata import Arduino,util
import time



def findPlace(contornos, imagem):

        for c in contornos:
        perimetro = cv2.arcLength(c, True)
        if perimetro > 200 and perimetro < 600:
        approx = cv2.approxPolyDP(c, 0.03 * perimetro, True)
         if len(approx) == 4:
         (x, y, lar, alt) = cv2.boundingRect(c)
         cv2.rectangle(imagem, (x, y), (x + lar, y + alt), (255, 255, 255), 12)
          roi = imagem[(y+10): y+alt, x: x+lar]
          cv2.line(img, (x, y+alt), (x+lar, y+al\t), (255, 255, 255), 200)
          #cv2.line(img, (1, 240), (700, 240), (255, 255, 255), 5)
           cv2.line(img, (x+lar, y), (x+lar, y+alt), (255, 255, 255), 200)
           cv2.line(img, (x, y), (x, y+alt), (255, 255, 255), 200)
           cv2.imwrite("imagem//roix.jpg", roi)

    return imagem

def reconhecimentoOCR(path_img):
    entrada = cv2.imread(path_img + ".jpg")
```

```
    #cv2.imshow("ENTRADA", img)

    img = cv2.resize(entrada, None, fx=4, fy=4,
interpolation=cv2.INTER_CUBIC)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # cv2.imshow("Escala Cinza", img)

    ret, img =
cv2.threshold(img,30,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    cv2.imshow("Limiar", img)

    img = cv2.GaussianBlur(img, (5, 5), 0)
    cv2.imshow("Desfoque", img)
    cv2.findContours(img, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
    cv2.imwrite(path_img + '-ocr.jpg', img)

    imagem = Image.open(path_img + '-ocr.jpg')
    caracs = pytesseract.image_to_string(imagem, lang='mdt2')
    caracs = caracs.replace(' ', '')
    caracs = caracs.replace("-", "")
    letras = caracs[:3]
    num = caracs[3:]
    num = num.replace("-", "")
    letras = letras.replace("-", "")
    num = num.replace('O', "0")
    num = num.replace('Q', "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
    plate_= placa_escrita[:8]
    print (placa_escrita[:8])

    if len(caracs) > 0:
        print(caracs)
    else:
        texto = "Reconhecimento Falho"

    return plate_

#####################################################################

video = cv2.VideoCapture(0)

while(video.isOpened()):

    ret, frame = video.read()

    if(ret == False):
        break

    area = frame[220:370 , 200:440]

    img_result = cv2.cvtColor(area, cv2.COLOR_BGR2GRAY)
```

```
    ret, img_result = cv2.threshold(img_result, 90, 255, cv2.THRESH_BINARY)

    img_result = cv2.GaussianBlur(img_result, (5, 5), 0)

    img, contornos, hier = cv2.findContours(img_result, cv2.RETR_TREE,
cv2.CHAIN_APPROX_NONE)

    cv2.imshow('FRAME', frame)

    img3 = findPlace(contornos, area)

    cv2.imshow('RES', area)
    if cv2.waitKey(1) & 0xff == ord('q'):
        break
video.release()

teste = reconhecimentoOCR('C://Users//italo//Desktop//Test
TCC//imagem//roix')
board = Arduino('COM3')
porteira = board.get_pin('d:9:s')
excelFile = 'placas.xlsx'
placasDB = pd.read_excel(excelFile)
print(placasDB)
placasDB['Placa']
value = teste

if value in placasDB['Placa'].values:
    print ('Veiculo identificado:\n')
    print (placasDB[placasDB['Placa']==value])
    porteira.write(90)
    time.sleep(10)
    porteira.write(0)
```

```
else:
    print('Veiculo não identificada!')


cv2.destroyAllWindows()
```