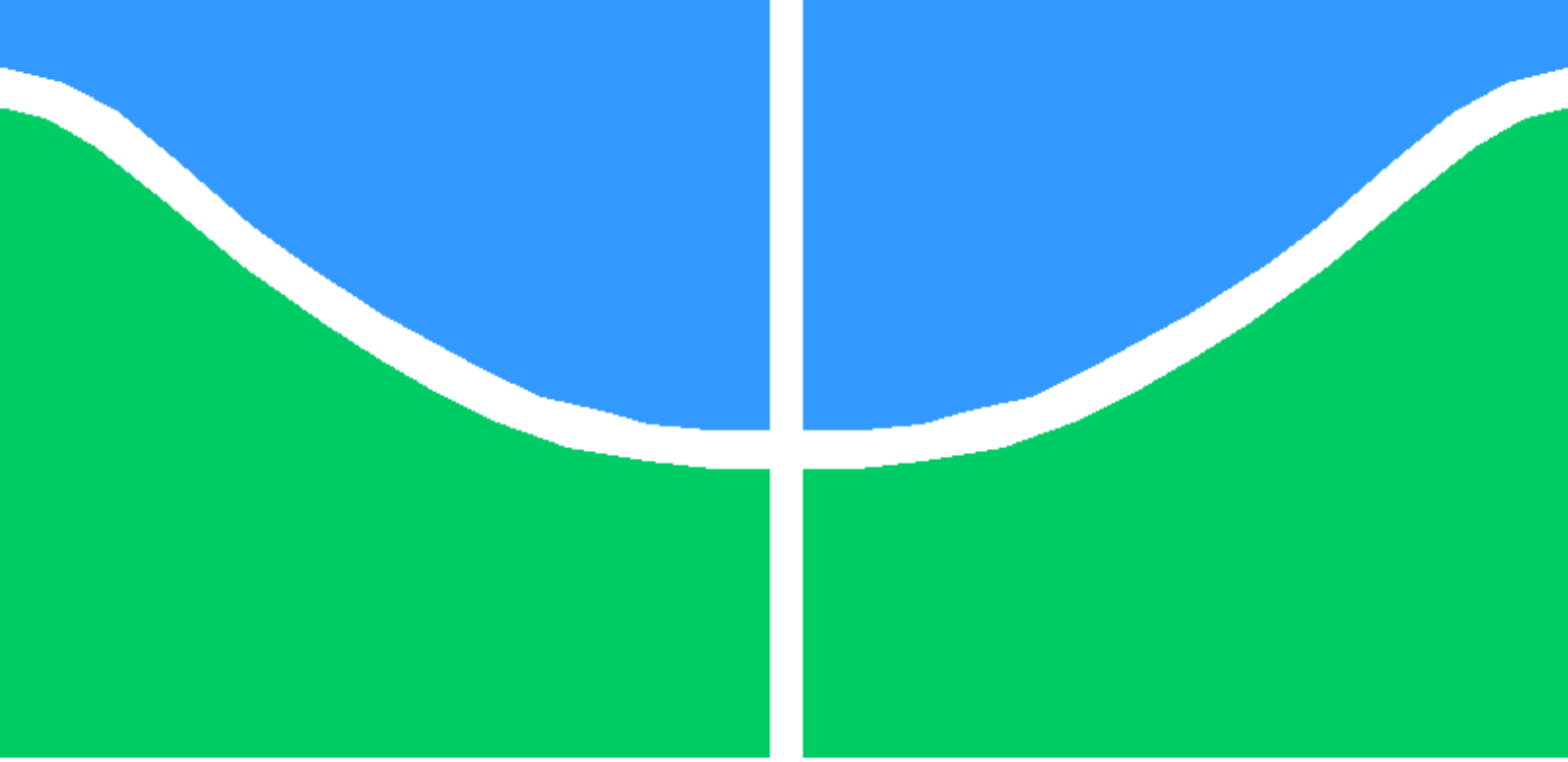


TRABALHO DE GRADUAÇÃO

USO DE VISÃO COMPUTACIONAL E RECONHECIMENTO ÓPTICO DE CARACTERES PARA RASTREAMENTO E CONTROLE DE VEÍCULOS

Raphael Pires de Mello Alvares dos Prazeres

Brasília, Julho de 2018

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

USO DE VISÃO COMPUTACIONAL E RECONHECIMENTO ÓPTICO DE CARACTERES PARA RASTREAMENTO E CONTROLE DE VEÍCULOS

Raphael Pires de Mello Alvares dos Prazeres

Relatório submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiro de Redes de Comunicação

Banca Examinadora

Prof. Georges Daniel Amvame Nze, Dr., ENE/UnB _____
Orientador

Prof. Ugo Silva Dias, Dr., EnE/UnB _____
Examinador interno

Fábio Lúcio Lopes de Mendonça, Msc., Latitude/UnB _____
Examinador interno

Dedicatórias

À minha família e aos meus amigos, que acompanharam toda minha caminhada, dedico este trabalho. Essa conquista é de todos.

.

Raphael Pires de Mello Alvares dos Prazeres

Agradecimentos

Agradeço o suporte que recebi de todos que colaboraram com a conclusão dessa etapa tão importante da minha vida.

Raphael Pires de Mello Alvares dos Prazeres

RESUMO

O campo multidisciplinar da visão computacional permite que máquinas "enxerguem" como humanos ou tão bem quanto eles. A partir de métodos que incluem a captura, processamento e análise de imagens ou vídeos digitais, dados são extraídos do mundo real (através de uma câmera, por exemplo) e através de algoritmos apropriados, essas máquinas podem realizar tarefas específicas com base nas informações que estão recebendo. A engenharia busca, através desses métodos, automatizar, monitorar e tornar mais seguros processos que antes eram impraticáveis ou que demandavam constante intervenção humana. As áreas de atuação cobrem diversos setores como uso industrial, na inspeção e controle de qualidade de produtos, no auxílio para desenvolvimento de inteligência artificial em robôs de última geração e monitoramento e controle de objetos.

Este projeto foi construído com foco no último tópico supracitado. Especificamente, trata do monitoramento e controle de veículos baseado no reconhecimento de suas respectivas placas. O principal objetivo é apresentar uma solução integradora de *software* e *hardware* que seja capaz de funcionar de forma autônoma, adaptável ao ambiente monitorado, realizando reconhecimento de placas dos veículos e executando tarefas pré-determinadas de acordo com regras vigentes e adequadas ao cenário.

ABSTRACT

The multidisciplinary field of computer vision allows machines to "see"as humans or as well as they do. From methods that include the capture, processing and analysis of digital images or videos, data are extracted from the real world (through a camera, for example) and using appropriate algorithms, these machines can perform specific tasks based on the information they are receiving. Engineering seeks by these methods to automate, monitor and render processes that were previously impractical or that require constant human intervention. Areas of practice cover various sectors such as industrial use, inspection and quality control of products, assistance in the development of artificial intelligence in state-of-the-art robots, and object monitoring and control.

This work was built with focus on the last topic mentioned above. Specifically, it deals with the monitoring and control of vehicles based on the reading of their respective plates. The main objective is to present a software and hardware integrator solution that is able to operate autonomously, adaptable to the monitored environment, reading vehicle license plates and performing tasks predetermined according to current and scenario-appropriate rules.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

Acrônimos

BSIA	*British Security Industry Association*
ABESE	Associação Brasileira de Segurança Eletrônica
DENATRAN	Departamento Nacional de Trânsito
LPR	*License Plate Recognition*
DFT	*Discrete Fourier Transform*
OpenCV	*Open Source Computer Vision*
IEEE	*Instituto de Engenheiros Eletricistas e Eletrônicos*
OCR	*Optical Character Recognition*
JPEG	*Joint Photographic Experts Group*
PNG	*Portable Network Graphics*
HP	*Hewlett-Packard*
CMOS	*Complementary metal-oxide semiconductor*
CCD	*Charge-coupled Device*
RGB	*Red, Green, Blue*
CMY	*Cyan, Magenta, Yellow*
CMYK	*Cyan, Magenta, Yello, Black*
HSI	*Hue, Saturation, Intensity*
BD	Banco de Dados
SO	Sistema Operacional
GUI	*Graphical User Interface*
API	*Application Programming Interface*
SQL	*Structured Query Language*
CPU	*Central Process Unit*
RAM	*Random Access Memory*
LAN	*Local Area Network*
GPIO	*General Purpose Input-Output*
FTP	*File Transfer Protocol*
HDTV	*High Definition Television*

Capítulo 1

Introdução

1.1 Motivação

O uso de imagens e vídeos para monitoramento de ambientes é comum nos diversos âmbitos da sociedade. Seja em pequenos ou grandes empreendimentos, residências e condomínios, áreas públicas ou privadas, esses sistemas, que em grande parte são compostos por câmeras e *softwares* de gerenciamento de vídeo, servem principalmente para fiscalizar ou proteger os locais onde estão instalados. A tendência natural é que o número de câmeras aumente à medida que o acesso a essa tecnologia é facilitado e o preço diminui. O relatório emitido pela *British Security Industry Association* (BSIA):*The Picture is not clear*, estima que existem entre 4 milhões e 5,9 milhões de câmeras de vigilância no Reino Unido entre ambientes públicos e privados. No cenário brasileiro, uma pesquisa conduzida pela Associação Brasileira de Segurança Eletrônica (ABESE) aponta que, somente em São Paulo, há mais de um milhão de câmeras de vigilância. Esses números confirmam o potencial de mercado e de exploração para esse nicho de tecnologia. Além disso, reflete a crescente busca por maior segurança e controle do que ocorre dentro dos limites de poder ou responsabilidade de algum agente, seja ele dono de algum empreendimento ou o próprio Estado, buscando a manutenção da segurança e controle do seu território e dos seus cidadãos.

As aplicações de vídeo abrangem diversos campos que vão desde segurança residencial até uso industrial. Dependendo do ambiente é necessário que aja algum tipo de filtro ou mecanismo que possa extrair informações específicas dentre a enorme quantidade de ocorrências ou eventos que são registrados através das lentes das câmeras de monitoramento. Detecção de movimento, reconhecimento facial, visão noturna e reconhecimento de texto são exemplos desses mecanismos.

A demanda por *softwares* que forneçam a identificação de veículos através da leitura de placas se dá pelo crescente aumento nacional da frota veicular e pelo devido controle sobre essa frota que deve ser promovido por autoridades competentes (públicas ou privadas). Um documento disponibilizado publicamente pelo Departamento Nacional de Trânsito (DENATRAN) aponta que no Brasil existem atualmente cerca de 53 milhões de automóveis registrados, sendo aproximadamente 1,2 milhões somente no Distrito Federal. A figura 1.1 apresenta o crescimento dessa frota nacional nos últimos 8 anos. Há, de fato, uma média de crescimento de praticamente 2,3 milhões de auto-

móveis por ano, mesmo excluídas outras categorias como caminhões, motocicletas e reboques, por exemplo. Utilizando a tecnologia de reconhecimento de placas comumente conhecida como *License Plate Recognition* ou LPR, como ferramenta governamental, seria possível acompanhar a situação do veículo do ponto de vista de documentação e legalidade.

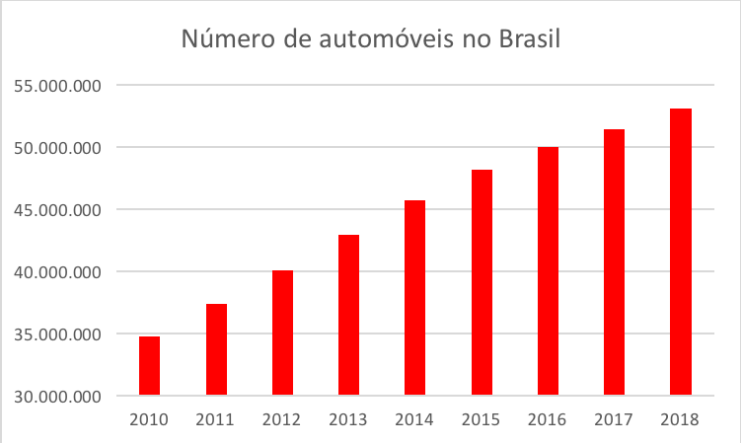


Figura 1.1: Aumento da frota veicular no Brasil. (Fonte: DENATRAN)

Além disso é possível utilizar essa mesma tecnologia para promover o controle de acesso de veículos em áreas onde se queira ter tal controle, classificando as placas entre autorizadas ou não autorizadas. A ideia é que esses *softwares* de LPR promovam artifícios que otimizem processos e facilitem operações do ponto de vista do operador.

1.2 Objetivos gerais

Criar um sistema que integre *hardware* e *software* capaz de extrair e tratar informações necessárias de uma imagem fornecida por uma câmera. Essa imagem, adequada para seguir em frente, servirá de entrada para o mecanismo de *Optical Character Recognition* (OCR) para realização do reconhecimento de placa de veículos. Os registros de reconhecimentos gerados devem ser armazenados em um banco de dados para análise, consulta e formação de histórico. O conjunto físico deve ser composto principalmente pelo par câmera/computador e um eventual terceiro componente que por enquanto será trata como "controlador". A solução deve funcionar de forma autônoma, ou seja, sem necessidade de intervenção humana que não seja para fins de configuração ou definição de regras. Além disso ela deve se propor a realizar a identificação de veículos em dois cenários específicos: rodovias e áreas de acesso restrito.

1.2.1 Objetivos Específicos

- A primeira proposta do sistema deve permitir ao operador cadastrar determinada coleção de placas que são do seu interesse e que devem gerar uma notificação visual ao serem detectadas pelo sistema. Consiste na implementação de ferramentas de visão computacional, com foco em OCR, para rastreamento de veículos em vias públicas através do reconhecimento de

placas. Aqui basta o conjunto físico câmera/computador para o funcionamento. A figura 1.2 ilustra a proposta em questão. A câmera deve estar posicionada adequadamente para capturar a faixa da rodovia que deseja-se monitorar e se encontrar na mesma rede local do computador que possui o sistema de reconhecimento das placas.

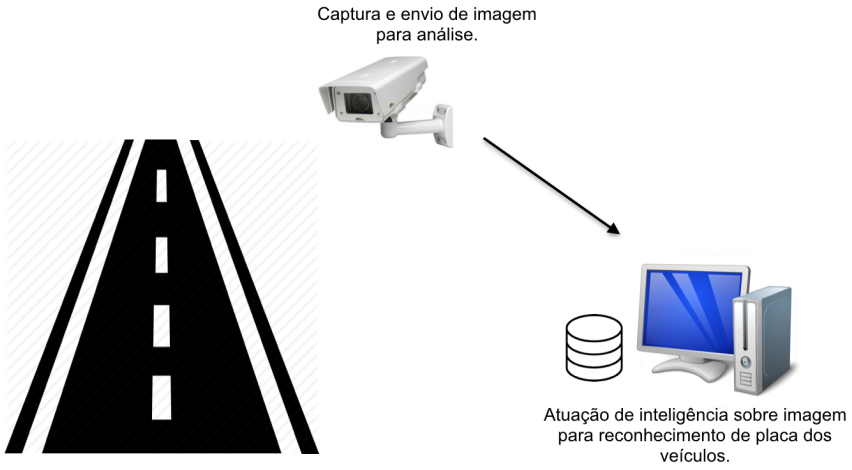


Figura 1.2: Primeira proposta. (Fonte: elaborada pelo autor)

- Na segunda proposta espera-se que o sistema permita ao operador cadastrar as placas que permitirão que os veículos que as possuem tenham o acesso garantido. Ele deve ainda atuar no controle de barreiras como cancelas ou portões. Com relação à diferença entre a proposta anterior, destaca-se a necessidade do acréscimo de *hardware* adequado para implementar a automação do controle de acesso dos veículos às áreas controladas. A figura 1.3 acrescenta o *hardware* adicional, bem como a barreira a ser controlada.

Figura 1.3: Segunda proposta. (Fonte: elaborada pelo autor)

- Desenvolver uma interface gráfica amigável que seja capaz de interagir com o banco de dados e que permita que o operador não faça modificações direta e exclusivamente através do arquivo que define as tabelas e registros do banco de dados;

1.3 Estrutura do Trabalho

No Capítulo 2, Fundamentação Teórica, serão abordados os principais fundamentos e conceitos que compõem o projeto. Busca-se dessa forma criar o conhecimento necessário para melhor entendimento do que ocorrerá nos capítulos seguintes.

No Capítulo 3, serão descritas as ferramentas computacionais utilizadas, bem como os equipamentos de *hardware* que permitem o funcionamento e a operação do sistema como um todo.

No Capítulo 4, o foco está no estudo de caso. São apresentados os tópicos de implementação prática do projeto e como operá-lo. Na área de implementação serão abordados os principais elementos técnicos e *scripts* de programação que permitem alcançar o proposto pelos objetivos citados acima. Já na parte de operação será demonstrado o funcionamento do sistema do ponto de vista do operador. Serão apresentadas as interfaces gráficas que permitem a interação de forma mais intuitiva com o banco de dados e que possibilitam, através de campos apropriados, criar as regras sobre placas que devem gerar notificações ou atuações sobre barreiras físicas.

No Capítulo 5, Análise, busca-se o entendimento e a representação do desempenho e de fatores decisivos inerentes ao processo de reconhecimento das placas.

Por fim, o Capítulo 6 apresentará a conclusão do autor no que tange ao desenvolvimento deste projeto. Serão tratados ainda pontos que podem ser otimizados ou substituídos em trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Buscando um melhor entendimento dos componentes e ferramentas que tornam possível o funcionamento do sistema, é importante esclarecer tópicos e conceitos que são inerentes ao escopo do projeto. Eles estão apresentados em ordem que possibilita a aquisição sequencial do conhecimento necessário.

2.1 Processamento de Imagens Digitais

Uma imagem pode ser definida com uma função bidimensional, $f(x, y)$, em que x e y são coordenadas espaciais, e a amplitude de f é chamada de intensidade ou nível de cinza da imagem nesse ponto (GONZALEZ; WOODS, 2009). Quando esses valores são quantidades finitas e discretas, temos a chamada imagem digital. Os elementos que compõe a imagem digital, cada um com localização e valores específicos, são chamados de pixels.

Não existe um acordo geral entre autores em relação ao ponto em que o processamento de imagens termina e outras áreas relacionadas, como a análise de imagens e a visão computacional, começam. Uma distinção pode ser traçada definindo o processamento de imagens com uma disciplina na qual tanto a entrada quanto a saída de um processo são imagens. Essa fronteira pode ser considerada um tanto quanto superficial mas ficará mais clara quando compararmos com o campo da visão computacional, cuja meta é utilizar computadores para emular a visão humana, incluindo o aprendizado e a capacidade de fazer inferências e agir com base em informações visuais. Aplicando essa ideia ao contexto do projeto, trata da diferença entre capturar uma imagem contendo um veículo e tratá-la através de transformações morfológicas (processamento de imagem) para em seguida extrair dessa imagem os caracteres alfanuméricos que compõe a placa do veículo, salvando-os em um arquivo editável de texto (visão computacional). O lado esquerdo da figura 2.1 representa uma imagem sem processamento aplicado. Percebe-se que que há um excesso de preto e que não é possível identificar os caracteres da placa do veículo. Já do lado direito da figura, após uma etapa de processamento conhecida como equalização de histograma, que será abordada posteriormente, a imagem possui uma maior distribuição dos tons de cinza na imagem. Dessa forma, a aplicação de um algoritmo de reconhecimento de caracteres se torna possível.

Figura 2.1: Processamento e Visão Computacional (MARENGONI; STRINGHINI, 2009)

2.1.1 Componentes de um sistema de processamento de imagens

Foi-se a época em que sistemas de processamento de imagens consistiam em dispositivos periféricos bastantes substanciais conectados a computadores igualmente substanciais. A evolução da tecnologia e consequentemente miniaturização de componentes de *hardware* possibilitou a migração dos padrões de barramento da indústria para gabinetes e estações de trabalho de computadores pessoais. A figura 2.2 identifica os componentes básicos que constituem um sistema de uso geral típico para o processamento de imagens digitais. Na sequência há explicação daqueles que são pertinentes ao projeto.

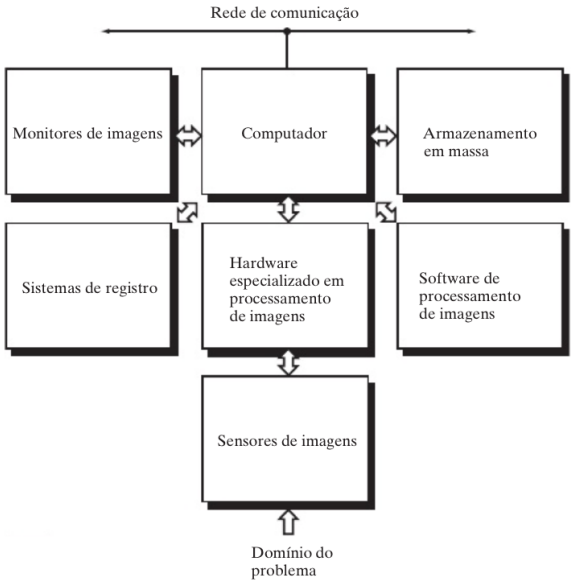


Figura 2.2: Componentes de um sistema de processamento de imagens (GONZALEZ; WOODS, 2009).

2.1.1.1 Sensores de imagem

São os elementos necessários para aquisição de imagens digitais. Para atender o escopo desse projeto podemos resumi-los aos dispositivos físicos sensíveis à energia irradiada pelo objeto cuja imagem deve ser capturada e aos conversores para formato digital. De forma mais direta, estão sendo especificados aqui os sensores CMOS (*Complementary Metal Oxide Semiconducto*) e CCD (Charge-Coupled Device) presentes nas câmeras digitais.

2.1.1.2 *Hardware* especializado

Geralmente está relacionado e presente no mesmo dispositivo que contém os sensores de imagem. Ele é responsável por operações primárias, aritméticas e lógicas em paralelo em toda a imagem. São operações como a redução de ruído, por exemplo, que não são suficientes para atender às necessidades do sistema que aqui será construído. Além das operações realizadas pelo *hardware*, é necessário amplo trabalho sobre a imagem através do processamento de imagem via *software*.

2.1.1.3 Computador

Pode variar entre um supercomputador e um computador pessoal. Para as aplicações que serão aqui tratadas, basta um computador de uso pessoal para realizar as tarefas de processamento e de visão computacional.

2.1.1.4 *Software* de processamento de imagens

De grande importância, esse tópico será intensamente explorado e é de vital importância para alcançar os objetivos propostos anteriormente. Trata da utilização de ferramentas que possibilitam ajustar a imagem para o uso em atividade específica, como é o caso do reconhecimento de placas.

2.1.1.5 Rede de comunicação

Componente praticamente indispensável em qualquer cenário onde há presença de máquinas e dispositivos que precisam se comunicar, também está presente no ambiente de operação da solução proposta pelo projeto. Através da rede será possível a comunicação entre os componentes câmera, computador e controlador.

2.2 Processamento de Imagens Coloridas

2.2.1 Modelo de cores

O objetivo de um modelo de cores é facilitar a especificação das cores em alguma forma padronizada amplamente aceita. A maioria dos modelos de cores utilizados atualmente é orientada ou em direção ao *hardware* (como no caso de monitores e impressoras coloridas) ou em direção a aplicações envolvendo a manipulação de cores (como a criação de imagens coloridas para uma animação) (GONZALEZ; WOODS, 2009). Em termos de processamento de imagens, aqueles modelos que mais se destacam são o modelo RGB (*red*, *green*, *blue* - vermelho, verde, azul) para monitores coloridos e uma ampla classe de câmeras de vídeo em cores; o modelo CMY (*cyan*, *magenta*, *yellow* - ciano, magenta, amarelo) e o modelo CMYK (*cyan*, *magenta*, *yellow*, *black* - ciano, magenta, amarelo, preto) para impressão colorida; e o modelo HSI (*hue*, *saturation*, *intensity* - matiz, saturação, intensidade), que corresponde estreitamente à forma como os seres humanos descrevem e interpretam as cores.

O principal modelo para o processamento de imagens, sobre o qual foram realizadas as operações sobre as imagens obtidas, é o modelo RGB. Nele, cada cor aparece em seus componentes espectrais primários de vermelho, verde e azul, e se baseia em um sistema de coordenadas cartesianas. A figura 2.3 apresenta o esquema de cores RGB. Os pontos ao longo da diagonal principal representam os valores de cinza, do preto na origem ao branco no ponto (1,1,1).

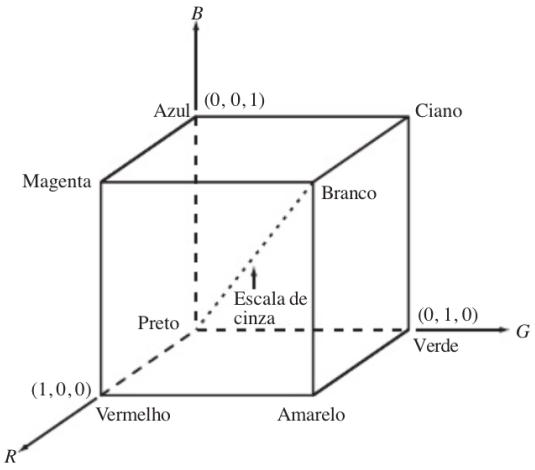


Figura 2.3: Esquema do cubo de cores RGB. (GONZALEZ; WOODS, 2009)

O número de bits utilizados para representar cada pixel em espaço RGB é chamado de profundidade de pixel. Considerando uma imagem RGB na qual cada uma das imagens: vermelha, verde e azul, seja uma imagem de 8 bits, é possível dizer que cada pixel de cores RGB tem uma profundidade de 24 bits (3 planos de imagem multiplicado pelo número de bits de cada plano). O termo imagem colorida costuma ser utilizado para expressar uma imagem de cores RGB de 24 bits. O cubo mostrado na figura 2.4 é um sólido, composto das $(2^8)^3 = 16.777.216$ cores possíveis nesse cenário.

Figura 2.4: Cubo de cores RGB de 24 bits. (GONZALEZ; WOODS, 2009)

2.3 Filtros

A aplicação de filtros em uma imagem digital significa a execução de algum tipo de processamento à mesma com o objetivo de se obter uma imagem resultante mais adequada a um determinado propósito do que a imagem original. É necessário enfatizar que o resultado estar mais apropriado ou não depende da interpretação do observador, já que as técnicas de aplicação de filtros em imagem são, de uma forma geral, direcionadas a um determinado problema a ser resolvido.

Os filtros podem ser espaciais (filtros que atuam diretamente na imagem) ou de frequência, onde a imagem é inicialmente transformada para o domínio da frequência usando a transformada de Fourier (geralmente através da transformada de Fourier discreta) e então é filtrada neste domínio. Em seguida, a imagem filtrada é transformada de volta para o domínio do espaço.

2.3.1 Domínio do Espaço

No domínio espacial os processos atuam na manipulação direta dos pixels da imagem. São caracterizados pela equação 2.1.

$$g(x, y) = T(f(x, y)) \tag{2.1}$$

onde $f(x, y)$ é a imagem original, $T(.)$ é uma transformação na imagem e $g(x, y)$ é a imagem transformada. T é uma operação definida sobre uma vizinhança de influência do pixel que está localizado na posição (x, y). Essa ideia de vizinhança de influência está associada aos pixels que estão ao redor da posição (x, y). Ela é definida por uma região retangular e de lado ímpar. A figura 2.5 mostra alguns exemplos de vizinhança com tamanhos variados. Essas regiões que definem matrizes nas operações de transformação também são chamadas de máscaras (MARENGONI; STRINGHINI, 2009).

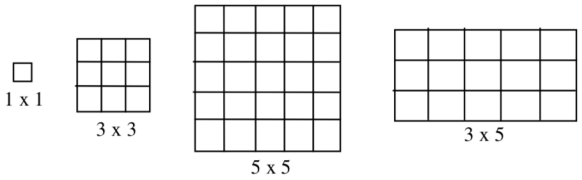


Figura 2.5: Máscaras de tamanhos diferentes. (GONZALEZ; WOODS, 2009)

A. Transformação de intensidade

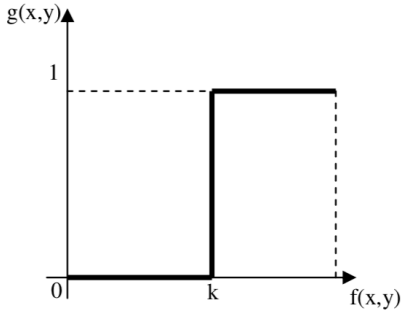
No caso mais simples o operador T é computado em uma vizinhança de tamanho 1x1, ou seja, apenas o valor do pixel no ponto é suficiente para determinar o valor na imagem processada. Este tipo de operação, chamada de função de transformação de intensidade, é utilizada para alterar a intensidade da imagem e pode ser aplicada a toda imagem ou a uma parte dela. Uma operação bastante útil é a binarização da imagem original em escala de cinza, que utiliza um certo valor de corte k. Esse tipo de transformação é definido pela equação 2.2

$$g(x,y) = \begin{cases} 1, & \text{se } f(x,y) \geq k. \\ 0, & \text{caso contrário.} \end{cases} \tag{2.2}$$

Para uma imagem em escala de cinza, os valores de intensidade de pixel variam entre 0 e 255. A figura 2.6 mostra a aplicação dessa função para um valor de $k = 127$. Essa técnica tem grande valia quando se quer isolar objetos de interesse.

É importante observar que na binarização o corte é feito de forma abrupta, ou seja, todos os valores acima de k são mapeados para o valor 1 (branco) e os valores abaixo de k são mapeados para 0 (preto).

(a) Imagem original. (OPENCV,2018)



(b) Função de binarização para um valor k. (OPENCV, 2018)

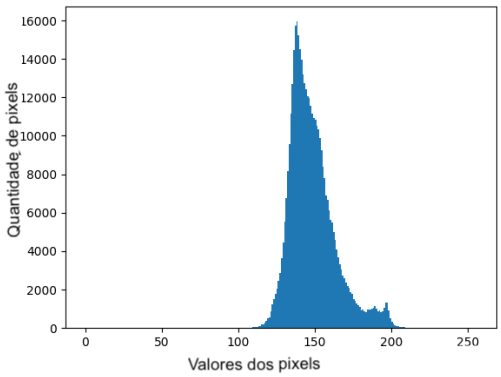
(c) Imagem binarizada. (Fonte: elaborada pelo autor)

Figura 2.6: Processo de binarização de uma imagem

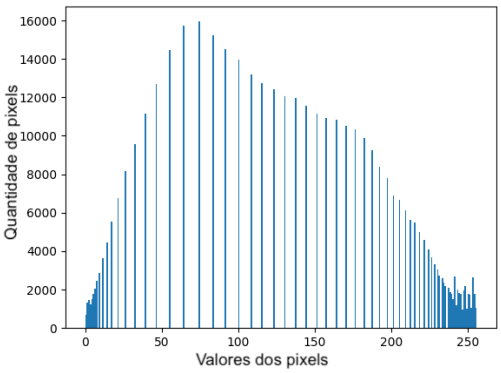
B. Histogramas

Os histogramas também são determinados a partir de valores de intensidade dos pixels. Uma das suas principais aplicações é o ajuste dos valores de intensidade de forma a melhorar o contraste em uma imagem. Esta operação é conhecida como equalização de histogramas. A ideia é mapear os valores de intensidade de uma imagem de um intervalo pequeno (pouco contraste) para um intervalo maior (muito contraste) e ainda distribuir os pixels ao longo da imagem de forma a obter uma distribuição uniforme de intensidades. A figura 2.7 mostra um exemplo de uma imagem que foi ajustada utilizando equalização de histogramas.

Figura 2.7: Pré-equalização (esquerda) e pós-equalização (direita). (OPENCV,2018)



(a) Histograma pré-equalização. (Fonte: elaborada pelo autor)



(b) Histograma pós-equalização. (Fonte: elaborada pelo autor)

Figura 2.8: Histogramas

A figura 2.8 mostra a diferença de distribuição dos valores dos pixels antes e depois da equalização de histograma. Percebe-se que ela gera uma distribuição de valores de intensidades muito mais equilibrada, retirando a concentração de uma pequena faixa para uma maior. Como consequência a imagem possui contornos e definição de objetos da imagem melhores.

C. Filtros Gaussianos

O filtro gaussiano tem os valores da máscara determinados a partir de uma função bi-dimensional Gaussiana discreta, com média igual a zero e desvio padrão θ, como mostrado abaixo pela equação 2.3

$$Gauss(x, y) = \frac{1}{2\pi\sigma^2} exp \frac{-(x^2 + y^2)}{2\sigma^2} \tag{2.3}$$

onde x e y são as posições na máscara e $Gauss(x, y)$ dá o valor a ser colocado na posição (x, y) da máscara. Os filtros Gaussianos são filtros de média e são utilizados para suavizar a imagem de forma ponderada e simétrica.

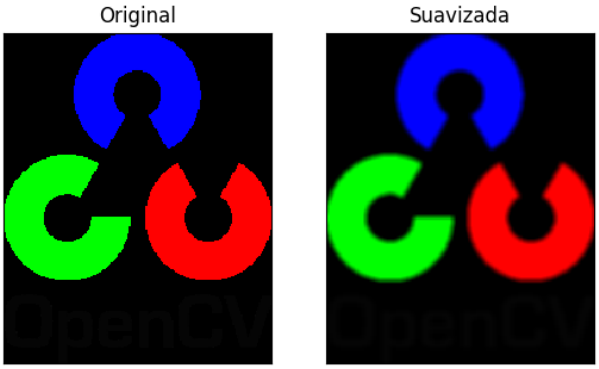


Figura 2.9: Imagem suavizada com filtro gaussiano. (Fonte: elaborada pelo autor)

2.3.2 Domínio da frequência

É possível fazer uma troca de base em uma imagem e representá-la em termos de uma soma ponderada infinita de um conjunto de senoides (MARENGONI; STRINGHINI, 2009). Mudanças rápidas na imagem são caracterizadas por frequências altas e mudanças suaves são caracterizadas por frequências baixas. Essa mudança de base pode ser realizada através da transformada de Fourier. Para implementação em computador geralmente é utilizada a transformação discreta de Fourier (DFT).

Definir um filtro no domínio da frequência corresponde a encontrar uma máscara para ser utilizada em conjunto com a imagem transformada e assim obter a imagem filtrada desejada (MARENGONI; STRINGHINI, 2009). Apesar da sua importância no processamento de imagens, permitindo o uso de filtros passa-baixa e passa-alta, por exemplo, os filtros sobre o domínio espacial que foram amplamente utilizados no projeto.

2.4 Processamento Morfológico de Imagens

A morfologia aqui é tratada no contexto da matemática, no campo da teoria dos conjuntos. Ela é usada como uma abordagem unificada e poderosa para vários problemas de processamento de imagens. Pode ser utilizada para extrair componentes e informações das imagens que são úteis na representação e descrição da forma de uma região, como fronteiras, por exemplo.

Em imagens binárias, os conjuntos em questão são membros do espaço 2-D de números inteiros Z^2, em que cada elemento de um conjunto é um vetor bidimensional, cujas coordenadas são (x, y) de um pixel branco (ou preto, dependendo da convenção) de uma imagem (GONZALEZ; WOODS, 2009). Já as imagens digitais em escalas de cinza podem ser representadas como conjuntos cujos componentes estão em Z^3. Neste caso, dois componentes de cada elemento do conjunto referem-se às coordenadas de um pixel e o terceiro corresponde ao seu valor discreto de intensidade. Os conjuntos em espaços dimensionais maiores podem conter outros atributos de imagem, como cor e componentes que variam no tempo.

Além das operações básicas envolvendo conjuntos, isto é, união, intersecção, diferença e complementares, os conceitos de reflexão e de translação são amplamente utilizados em morfologia. A reflexão de um conjunto B, indicada por $\hat{B}$, é definida pela equação 2.4 abaixo

$$\hat{B} = \{w | w = -b, \ para \ b \ \epsilon \ B\} \tag{2.4}$$

Se B é o conjunto de pixels que representa um objeto em uma imagem, então $\hat{B}$ é simplesmente o conjunto dos pontos em B cujas coordenadas (x, y) foram substituídas por $(-x, -y)$. As figuras 2.4 (a) e (b) mostram um conjunto simples e sua reflexão.

A translação de um conjunto B no ponto $z = (z_1, z_2)$, é indicada por $(B)_z$, é definida como

$$(B)_z = \{c | c = b + z, \ para \ b \ \epsilon \ B\} \tag{2.5}$$

Se B é o conjunto de pixels que representa um objeto em uma imagem, então $(B)_z$ é o conjunto de pontos em B, cujas coordenadas (x, y) foram substituídas por $(x + z_1, y + z_2)$. A figura 2.10 (c) ilustra esse conceito usando o conjunto B da figura 2.10 (a).

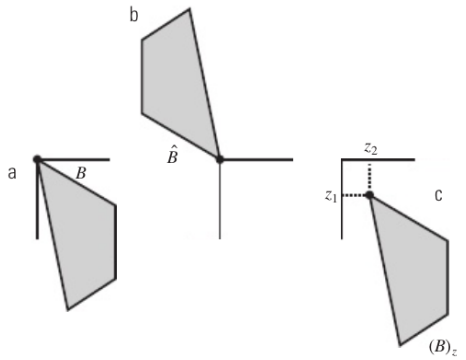


Figura 2.10: (a)Um conjunto, (b)sua reflexão e (c)sua translação. (GONZALEZ; WOODS, 2009)

A. Dilatação

De forma sucinta, a dilatação é uma transformação morfológica que combina dois conjuntos usando adição vetorial. Como seu nome sugere, o resultado será uma imagem com bordas maiores ou "engordada".

A dilatação de A por B é definida pela equação 2.6

$$A \oplus B = \{c \, \epsilon \, Z^2 | c = a + b, a \, \epsilon \, A \; e \; b \, \epsilon \, B\} \tag{2.6}$$

Onde

- A e B são conjuntos de Z^2 (imagens binárias)
- A é a imagem sendo operada
- B é chamado de elemento estruturante e sua natureza define como a dilatação irá ocorrer.

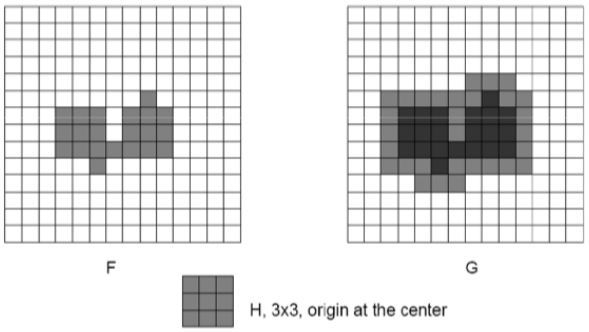


Figura 2.11: Processo de Dilatação (UFU, 2014).

Na figura 2.11 o elemento estruturante é definido pelo quadrado H de tamanho 3x3 com origem no centro. Já a imagem que será dilatada está definida no quadriculado F. O elemento estruturante "navegará" pela imagem que deve ser dilatada. Quando o pixel central do elemento estruturante coincidir com algum pixel da outra imagem, os pixels ao redor do elemento estruturante vão se "aderir" à imagem, tornando-a dilatada. A imagem contida no quadriculado G da figura 2.11 exemplifica o procedimento citado.

B. Erosão

Ao contrário da dilatação, a erosão é uma transformação morfológica que combina dois conjuntos usando vetores de subtração. O resultado, dessa vez, será uma imagem "encolhida".

A erosão de A por B é definida pela equação 2.7

$$A \ominus B = \{c \, \epsilon \, Z^2 | c + b, \; \epsilon \, A, \; paratodo \; b \, \epsilon \, B\} \tag{2.7}$$

Onde

- A e B são conjuntos de Z^2 (imagens binárias)
- A é a imagem sendo operada
- B é chamado de elemento estruturante e sua natureza define como a erosão irá ocorrer.

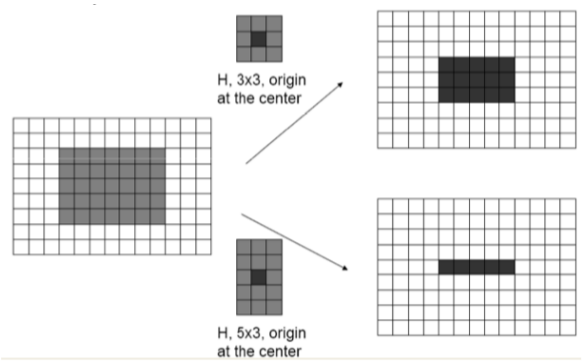


Figura 2.12: Processo de Erosão (UFU, 2014).

A figura 2.12 mostra o processo de erosão. Ele é inverso ao de dilatação, onde dessa vez, há o encolhimento da imagem operada.

2.4.1 Visão Computacional

A visão computacional é o processo de modelagem e replicação da visão humana usando *software* e *hardware* (ACADEMY, 2017). Basicamente, sistemas de visão computacional capturam imagens através de dispositivos óptico-eletrônicos (como câmeras digitais) e buscam produzir descrições úteis dos dados contidos nas imagens, sendo que as informações produzidas podem ser utilizadas, por exemplo, na classificação de objetos ou no controle automático de algum dispositivo atuador, como uma barreira física que controla entrada e saída de automóveis. As primeiras pesquisas sobre a visão computacional tomaram uma abordagem *Top-down* (de cima para baixo). Em situações controladas funcionou bem, mas em situações mais diversas ficou claro que seriam necessários conjuntos de dados impraticáveis, sabendo que nessa abordagem buscava-se descrever todo o cenário antes de aplicar a visão computacional.

Uma abordagem *Bottom-up* (de baixo para cima) imitando o que é encontrado no cérebro se mostrou muito mais efetiva e promissora. Dessa forma, um computador pode aplicar uma série de transformações a uma imagem e descobrir aquilo que é relevante.

As aplicações mais difundidas encontram-se na área médica, com o processamento médico de imagens. Fala-se aqui da extração de informação de imagens para realizar diagnósticos sobre os pacientes. As fontes de imagem podem ser advindas, por exemplo, de microscopia, radiografia e ressonância magnética. É possível citar também seu uso na indústria, onde a informação processada pode auxiliar no controle de qualidade de produtos, por exemplo. Outros grandes campos de atuação são o militar, veículos autônomos e exploração espacial.

Dentre as etapas comumente encontradas em sistemas de visão computacional envolvendo reconhecimento de padrões encontram-se a segmentação - processo utilizado para suprimir da imagem o que não tem importância, mantendo-se apenas os objetos de interesse (GONZALEZ; WOODS, 2009) -, a extração de características (DUIN; MAO, 2000), que retira desses objetos características que sirvam para identificá-los (como tamanho, curvatura, comprimento, ocorrência de determinadas formas geométricas, etc.) e a classificação (GONZALEZ; WOODS, 2009), que recebe os valores do módulo extrator de características e busca enquadrar os objetos de interesse em uma das classes pré-definidas.

Um sistema de visão computacional típico possui 6 etapas: Aquisição de imagens, pré-processamento, segmentação, extração dos atributos, identificação dos padrões previamente estabelecidos e, por fim, é gerado um relatório com as análises e informações obtidas (GONZALEZ; WOODS, 2009), como mostra a figura 2.13

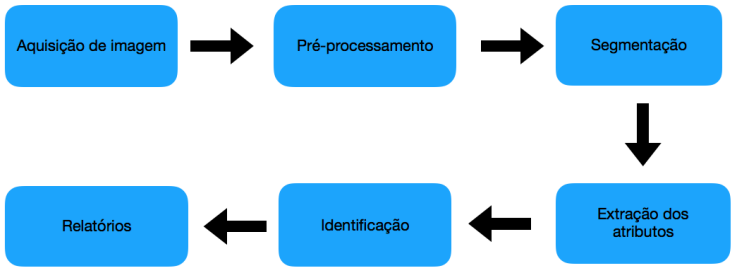


Figura 2.13: 6 etapas da visão computacional (Fonte: elaborada pelo autor)

As etapas de aquisição da imagem e do pré-processamento foram tratadas anteriormente. Quanto a etapa de segmentação, é o processo de separar as várias regiões que compõem uma imagem, de acordo com suas propriedades. A técnica de segmentação a ser utilizada varia com a aplicação do sistema. Esse processo é de fundamental importância, de tal forma que o desempenho do sistema é essencialmente dependente do desempenho do processo de segmentação. Uma segmentação realizada de forma adequada em um sistema de visão computacional é um passo substancialmente importante para seu correto funcionamento (A.R, 2005). Vale ressaltar a utilização da técnica *Canny Edge Detection*, de vital importância para o projeto, que cobre as etapas de segmentação e de extração de atributos.

2.4.1.1 *Canny Edge Detection*

As bordas constituem informação de alta frequência e encerram propriedades significativas de uma imagem (MAIA DO VALE; PORFÍRIO DAL POZ, 2002). Estas propriedades incluem descontinuidades geométricas e as características físicas dos objetos.

Para que se obtenha resultados consistentes nos passos subsequentes à detecção de bordas, é necessário que esta detecção seja eficiente e confiável. É difícil formular um algoritmo de detecção de bordas que possua um bom desempenho em diferenciados contextos e capture os requisitos necessários aos estágios subsequentes de processamento. Nesse contexto, Canny (1986), desenvolveu

um processo de detecção de bordas a partir de critérios de quantificação de desempenho de operadores de bordas conhecidos como: critério de detecção e critério de localização. Essa abordagem baseia-se em três objetivos básicos:

- Baixa taxa de erro: todas as bordas devem ser encontradas e serem as mais próximas possível das bordas verdadeiras.
- Os pontos de bordas devem estar bem localizados: as bordas detectadas devem estar mais próximas possíveis das bordas verdadeiras.
- Resposta de um único ponto de borda: o detector não deve detectar múltiplos *pixels* de borda onde apenas um único ponto de borda existe

Para satisfazer esses aspectos, o detector de bordas Canny utiliza uma complexa sequência de operações matemáticas, que fogem ao escopo do projeto. Aqui, basta compreender a importância e a capacidade de isolar elementos específicos da imagem através dessa técnica. O resultado de sua aplicação pode ser observado na figura 2.14

Figura 2.14: Detecção de bordas Canny.

2.4.2 Reconhecimento Óptico de Caracteres (OCR)

Reconhecimento Óptico de caracteres, ou OCR, é a tecnologia que permite converter diferentes tipos de documentos, como páginas escaneadas, arquivos PDF ou imagens capturadas por uma câmera digital, transformando o conteúdo do documento em um arquivo de texto editável. Essa tecnologia é amplamente utilizada em *softwares* que requisitam uma imagem contendo um texto, como a foto de um documento ou uma página manuscrita, por exemplo, e a transforma em arquivo de texto editável pelo usuário. Este é o processo chamado de "ocerização". A maioria dos *softwares* gratuitos disponíveis na internet impõe ao usuário uma quantidade mínima de iluminação, posicionamento do texto a ser "ocerizado", resolução e outros aspectos. Trazendo para o escopo do projeto, seria inviável fornecer ao *software* uma imagem capturada através do processo

de aquisição sem nenhum tratamento posterior. Esta imagem contém milhares de aspectos que não estão em formato de texto, o que impossibilitaria a extração exclusiva do conteúdo contido na placa do veículo. Esse fator justifica todas as etapas de pré-processamento e de segmentação.

2.4.3 Banco de Dados

(i) "Um banco de dados é um conjunto de arquivos relacionados entre si" (Chu, 1983)

(ii) "Um banco de dados é uma coleção de dados operacionais armazenados, sendo usados pelos sistemas de aplicação de uma determinada organização" (C. J. Date, 1985)

(iii) "Um banco de dados é uma coleção de dados relacionais" (Elmasri & Navathe, 1989)

(iv) "Um banco de dados é um conjunto de dados armazenados, cujo conteúdo informativo representa, a cada instante, o estado atual de uma determinada aplicação" (Laender, 1990)

Depreende-se das definições acima que o Banco de Dados (BD) é:

- Coleção de dados relacionados;
- Coleção logicamente coerente de dados com algum significado;
- Um BD está sempre associado a aplicações e a usuários que têm interesse nele.

Dentre outros aspectos almeja-se com o uso de banco de dados: redundância controlada de dados, compartilhamento de dados por aplicações diversas, controle de autorização de acesso a dados e redução do tempo de desenvolvimento de aplicações. O escopo do projeto visa principalmente manter um registro dos reconhecimentos de placas realizadas pelo sistema e criar regras baseadas em tabelas presentes nesse mesmo banco de dados. Será utilizado o modelo relacional, onde representa-se dados e relacionamentos entre dados por um conjunto de tabelas, cada uma contendo um número de colunas com nomes únicos. O uso do banco de dados é essencial para o efetivo funcionamento do sistema. A geração de notificações ou o controle sobre barreiras físicas são realizados com base em verificações realizadas em tabelas presentes nesse banco de dados.

Capítulo 3

Ferramentas e equipamentos

Diversas ferramentas são utilizadas para a execução deste projeto. As duas principais são: *OpenCV*, usada para fazer toda a etapa de processamento sobre imagem e *Tesseract*, responsável por realizar o reconhecimento de caracteres. Existem ainda ferramentas auxiliares que servem de suporte, como o *Python*, *Virtualenv* e *VirtualBox*, por exemplo. Esse capítulo visa à apresentação das ferramentas, bem como suas respectivas funções no projeto.

3.1 *Oracle VM VirtualBox*

Virtualbox é uma aplicação de virtualização *cross-plataform*. Pode ser instalado em dispositivos dos mais diversos sistemas operacionais, por exemplo, Windows, Mac e Linux. O *Virtualbox* estende as capacidades de um computador existente para que este possa funcionar com múltiplos sistemas operacionais (dentro de múltiplas máquinas virtuais) ao mesmo tempo. Também é a única solução profissional que está disponível gratuitamente (ORACLE, 2017). Para realização do projeto, o sistema operacional escolhido para ser virtualizado foi o Ubuntu 16.04 LTS.

3.2 *Python*

Python é uma linguagem de programação poderosa e fácil de aprender. Tem estruturas de dados de alto nível eficientes e uma simples, mas eficiente abordagem à programação orientada a objeto. Esta linguagem permite a divisão de programas em módulos que podem ser reusados em outros programas em *Python* (PYTHON-SOFTWARE-FOUNDATION, 2017).

Python é uma das linguagens de programação mais populares da atualidade, sendo ranqueada em 2017 pelo Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) como a mais popular entre as linguagens utilizadas em projetos *open source* (DIAKOPOULOS; CASS, 2016). A possibilidade de integração com outras ferramentas (principalmente com o *OpenCV*) e vasta documentação foram os dois principais fatores que justificaram a escolha de tal linguagem de programação. A versão 2.7 foi escolhida para compor as soluções do projeto.

3.2.1 *Virtualenv*

Virtualenv é uma ferramenta para criar ambientes *Python* isolados. O problema básico a ser solucionado é o de dependências, versões e indiretamente permissões. Estes problemas ocorrem quando aplicações diferentes precisam de, por exemplo, versões diferentes de uma mesma biblioteca *Python*, ou então alguma aplicação que ainda não foi testada com a nova versão de tal biblioteca (BICKING, 2017). O último problema, de permissões, envolve o fato de que um certo usuário não pode instalar pacotes no diretório global do *Python*. Para todos estes casos, o *Virtualenv* tem os seus próprios diretórios de instalação que não compartilham bibliotecas com outros ambientes *Virtualenv* e, opcionalmente, com o ambiente global (BICKING, 2017).

Uma vez instalado, criou-se o ambiente "cv", abreviação para *computer vision*. A instalação das ferramentas subsequentes foram feitas dentro desse ambiente virtual isolado. O acesso pode ser feito pelo terminal do Ubuntu através do comando "*workon cv*" e comprova-se que estamos nele pela identificação "cv" anterior ao *prompt*, como mostra a figura 3.1

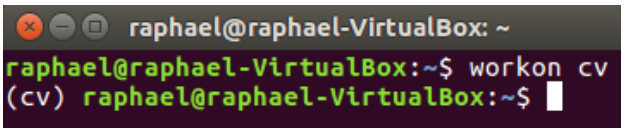


Figura 3.1: Ambiente virtual isolado.

3.2.2 *Numpy*

O *NumPy* é um pacote fundamental para computação científica em *Python*. É uma biblioteca que fornece um objeto de matriz multidimensional, vários objetos derivados e uma variedade de rotinas para operações rápidas em matrizes, incluindo manipulação matemática, lógica, de formas, classificação, seleção, I / O , transformada discreta de Fourier, álgebra linear básica, operações estatísticas básicas, simulação aleatória e muito mais (SCIPY, 2017).

3.2.3 *Watchdog*

O *Watchdog* é uma API e um utilitário do *Shell* para monitorar eventos no sistema de arquivos. Em outras palavras, um diretório é passado como argumento para essa API que passa a ser observado pelo programa, gerando alertas sobre eventos de criação, deleção, modificação e movimentação de arquivos e diretórios. Esta é uma ferramenta muito útil que serve de gatilho para algumas tarefas que devem ser executadas pelo sistema desenvolvido.

3.2.4 Tkinter

Tkinter é uma biblioteca que acompanha a instalação padrão do *Python* e permite desenvolver interfaces gráficas. Isso significa que qualquer computador que tenha o interpretador *Python* instalado é capaz de criar interfaces gráficas usando o Tkinter, com exceção de algumas distribuições

Linux, exigindo que seja feita o download do módulo separadamente.

O principal motivo para utilização do Tkinter é a sua facilidade de uso e recursos disponíveis. Outro motivo significativo é que é nativo da linguagem *Python*, o que significa que basta importá-lo no momento que se quiser utilizá-lo. A criação de interfaces gráficas vem como um recurso adicional ao projeto, visto que não tem papel primário, e sim, serve como um intermediário entre o usuário do sistema e o banco de dados. Evita-se dessa forma a operação direta sobre o banco de dados e o ofereça uma interface mais simples e amigável. A figura 3.2 ilustra parte de como é feita a estrutura em *python* para que se crie a janela e funções dentro da janela.

```
11  class Main:
12      def __init__(self,master):...
86
87
88      def add(self):...
103
104     def apagar(self):...
111
112     def atualizar(self):
113         self.conectar.commit()
114         self.listbox.delete(0,END)
115         self.listbox2.delete(0,END)
116         lista1 = self.cursor.execute("SELECT placa FROM placas")
117         for i in lista1:
118             self.listbox.insert(END,i)
119
120         lista2 = self.cursor.execute("SELECT proprietário FROM placas")
121         for i in lista2:
122             self.listbox2.insert(END,i)
123
124     def blacklist(self):...
127
128     def realTime(self):...
131
132     def streamVideo(self):...
```

Figura 3.2: Exemplo de estrutura *python* para construção da janela e funções.

É possível identificar entre as linhas 88 e 132 da imagem a definição dos botões, com atenção especial à construção da conexão com o banco de dados para realização da atualização dos dados que são exibidos na janela, entre as linhas 112 e 122.

A figura 3.3 apresenta o resultado final. Neste caso trata-se da interface gráfica principal que permite acesso a outras janelas e promove interação com o banco a partir de dados adicionados nos campos disponíveis. Essas interações serão abordadas posteriormente.

Figura 3.3: Interface gráfica interativa.

3.3 OpenCV

OpenCV (Open Source Computer Vision Library) é uma biblioteca de software de visão computacional e aprendizado de máquina em código aberto. A *OpenCV* foi construída para fornecer uma infra-estrutura comum para aplicações de visão computacional e para acelerar o uso da percepção da máquina nos produtos comerciais (OPENCV, 2018).

A biblioteca tem mais de 2500 algoritmos otimizados (OPENCV, 2018), o que inclui um conjunto abrangente de algoritmos de visão computacional e de aprendizado de máquina clássicos e de última geração. Esses algoritmos podem ser usados para detectar e reconhecer rostos, identificar objetos, classificar ações humanas em vídeos, rastrear movimentos de câmera, rastrear objetos em movimento, extrair modelos 3D de objetos, unir imagens para produzir uma alta resolução de uma cena inteira, encontrar imagens semelhantes de um banco de dados de imagens, etc. *OpenCV* tem mais de 47 mil pessoas de usuário comunidade e número estimado de downloads superiores a 14 milhões (OPENCV, 2018). A biblioteca é amplamente utilizada em empresas, grupos de pesquisa e órgãos governamentais.

Os usos implantados da *OpenCV* abrangem desde unir imagens de ruas, detectar intrusões em vídeo de vigilância, monitorar equipamentos de minas, ajudar robôs a navegar e pegar objetos, detectar acidentes de afogamento, entre outros. Este projeto visa entrar no rol de implantações inteligentes que buscam otimizar e facilitar tarefas através das possibilidades disponibilizadas pela ferramenta em questão.

3.4 *Tesseract*

Tesseract é um software de reconhecimento óptico de caracteres que foi originalmente desenvolvida pela HP entre 1985 e 1994. Em 2005 o *Tesseract* se tornou *open source* (GITHUB, 2018). Durante um período foi mantido pela Google e atualmente o projeto está hospedado no *GitHub*.

O Tesseract tem suporte a unicode (UTF-8) e pode reconhecer mais de 100 idiomas "*out of the box*". Suporta vários formatos de saída: texto simples, hocr (html), pdf, tsv, pdf em texto invisível. Em muitos casos, para obter melhores resultados de OCR, será necessário melhorar a qualidade da imagem que está sendo fornecendo ao Tesseract, o que justifica todo o processamento realizado pela ferramenta OpenCV. A ferramenta não inclui um aplicativo GUI.

A versão estável mais recente é 3.05.01, lançada em 1 de junho de 2017. O código-fonte mais recente 3.05 está disponível no *GitHub*.

3.5 *SQLite*

O *SQLite* é uma biblioteca que implementa um mecanismo de banco de dados SQL transacional autônomo, sem servidor e com configuração zero. O código para o *SQLite* é de domínio público e, portanto, livre para uso para qualquer finalidade, comercial ou privada (SQLITE.ORG, 2018).

O *SQLite* é um mecanismo de banco de dados SQL incorporado. Ao contrário da maioria dos outros bancos de dados SQL, o *SQLite* não possui um processo de servidor separado. Ele lê e grava diretamente em arquivos de disco comuns. Um banco de dados SQL completo com várias tabelas, índices, acionadores e visualizações está contido em um único arquivo de disco. O formato de arquivo de banco de dados é multi-plataforma.

3.6 *DB Browser for SQLite*

O *DB Browser for SQLite* é uma ferramenta visual de código aberto de alta qualidade para criar, projetar e editar arquivos de banco de dados compatíveis com o SQLite (SQLITE, 2018)

Seu público alvo são usuários e desenvolvedores que desejam criar bancos de dados, pesquisar e editar dados. Ele usa uma interface semelhante a uma planilha eletrônica e não é necessário utilizar os comandos SQL.

Dentre as tarefas, destacam-se as possibilidades para:

- Criar e compactar arquivos de banco de dados;
- Criar, definir, modificar e excluir tabelas;
- Navegar, editar, adicionar e excluir registros;
- Importar e exportar registros como texto.

Essa ferramenta foi utilizada inicialmente para visualização dos registros criados através do processo de reconhecimento dos caracteres das placas. Posteriormente, por meio da ferramenta Tkinter e das janelas criadas a partir dela, a visualização desses registros passou a ser, também, possível por meio dessas janelas.

3.7 Raspberry Pi

O Raspberry Pi é um computador do tamanho de um cartão de crédito de baixo custo que se conecta a um monitor de computador ou TV, usa um teclado e mouse padrão e foi desenvolvido no Reino Unido pela Fundação Raspberry Pi. (RASPBERRYPI.ORG, 2018). O Raspberry Pi tem a capacidade de interagir com o mundo exterior e tem sido usado em uma ampla gama de projetos que vão desde robótica e automação residencial a interfaces de aprendizado de linguagem de programação voltadas ao público infantil.

Atualmente estão disponíveis 6 modelos diferentes (RASPBERRYPI.ORG, 2018), cada um dotado de *hardware* apropriado para aplicações diferentes. O modelo escolhido para atender as necessidades desse projeto foi o Raspberry Pi Model 3 B. Ele foi lançado em fevereiro de 2016, usa uma CPU ARM Cortex-A53 quad-core de 64 bits de 1.2GHz, 1GB de RAM, LAN sem fio 802.11n integrada e Bluetooth 4.1. Em termos de funcionalidade, o fator importante para o projeto são as chamadas entradas e saídas de uso geral, ou, em inglês, *General purpose input-output* (GPIO). Através dessas saídas é possível controlar um módulo relé, que por sua vez controla dispositivos de cargas maiores. Dessa forma, o Raspberry Pi atua como controlador de barreiras físicas, isto é, portões ou cancelas, por exemplo. A figura 3.4 apresenta o Raspberry Pi Model 3, bem como as GPIO.

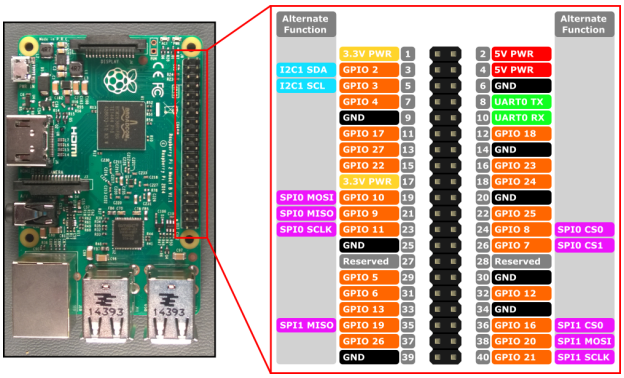


Figura 3.4: Raspberry Pi Model 3 e GPIO.

3.7.1 Raspbian

Raspbian é um sistema operacional livre baseado no Debian otimizado para o *hardware* Raspberry Pi (RASPBIAN, 2018). O sistema operacional é o conjunto de programas básicos e utilitários que fazem o Raspberry Pi rodar. No entanto, o Raspbian oferece mais do que um sistema operacional puro: ele vem com mais de 35.000 pacotes e *software* pré-compilado em um formato agradável

para fácil instalação (RASPBIAN, 2018). A compilação inicial de mais de 35.000 pacotes, otimizada para melhor desempenho, foi concluída em junho de 2012. No entanto, Raspbian ainda está em desenvolvimento ativo com ênfase na melhoria da estabilidade e desempenho de tantos pacotes.

3.7.2 Módulo Relé

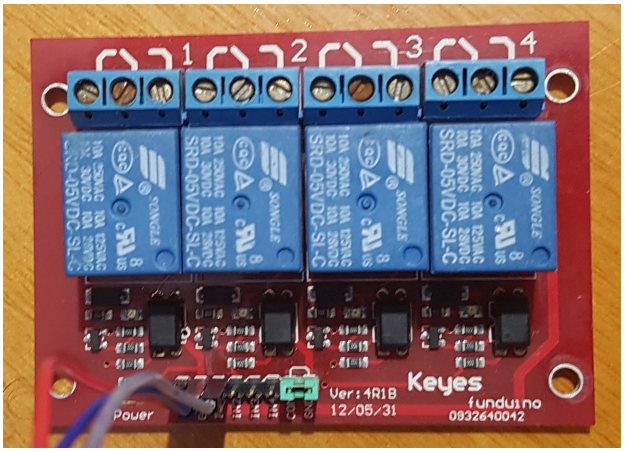


Figura 3.5: Módulo Relé 4 canais.

Este Módulo Relé 5V com 4 canais permite integração com uma ampla gama de microcontroladores como Arduino, AVR, PIC, ARM. A partir das saídas digitais pode-se, através do relé, controlar cargas maiores e dispositivos como motores AC/DC, eletroímãs, solenóides, lâmpadas incandescentes e eletrodomésticos por exemplo.

Especificações:

- Tensão de operação: 5VDC;
- Permite controlar cargas de 220V AC;
- Corrente típica de operação: 15 20mA;
- LED indicador de status;
- Pinagem: Normal Aberto, Normal Fechado e Comum;
- Tensão de saída: (30 VDC a 10A) ou (250VAC a 10A);
- Tempo de resposta: 5 10ms;
- Dimensões: 8 x 6 x 2cm.

3.8 Câmera Axis P1355

Figura 3.6: Axis P1355. (AXIS, 2018)

Com relação a câmera, dentre suas diversas especificações técnicas e capacidades, são aspectos que merecem atenção dentro do escopo do projeto: resolução de até 1080p HDTV, detecção de movimento e suporte ao protocolo FTP para envio de imagens. A seção 4.1 do próximo capítulo apresenta claramente o propósito de uso dessa câmera.

Capítulo 4

Estudo de Caso

4.1 Implementação

O projeto consiste em reunir todas as ferramentas apresentadas no capítulo anterior e criar uma solução unificadora, como proposta nos objetivos da seção 1.2. O projeto tem um apelo maior para o segundo cenário apresentado, ou seja, realizar o reconhecimento de placas e, com base em verificações nas tabelas criadas no banco de dados, acionar o módulo relé para liberação ou não da barreira física, criando assim, um controle de acesso de veículos.

Neste capítulo serão apresentadas as etapas que compõe o funcionamento do sistema, destacando os principais *scripts* que permitem a evolução dessas etapas.

Como forma de prezar pela segurança e pela privacidade, as imagens que contêm placas de veículos reais que foram utilizados como exemplo para especificar as etapas foram propositalmente embaçadas. Independentemente, é possível compreender a evolução das etapas por meio do restante visível da placa.

4.1.1 Aquisição de Imagem

O processo tem início a partir da aquisição de imagem. A aquisição, ou o que inicialmente foi uma inserção, pode ser realizada manualmente pelo operador. Nesse modo, a imagem deve ser adicionada em diretório específico para que que o *watchdog* responsável pelo "*start*" do processo possa agir.

Durante grande parte do desenvolvimento deste projeto, essa era a única forma de interagir com o programa. No entanto, percebeu-se que existem no mercado câmeras capazes de realizar a captura e envio de imagem baseados em detecção de movimento. Nesse período anterior ao uso da câmera, tentou-se criar essa funcionalidade de detecção de movimento e envio da imagem para o diretório apropriado no intuito de não restringir o funcionamento pleno do sistema a câmeras específicas. Entretanto, o desenvolvimento das funcionalidade não evoluiu como esperado e não atendeu às expectativas do projeto. Por fim, aderiu-se ao uso da câmera que é capaz de realizar o

processo de aquisição e envio da imagem. É importante frisar que o uso da câmera justifica-se por tornar o processo mais fluido, sem necessidade de inserção manual de imagens no diretório. De qualquer forma, a retirada desse equipamento não afeta as outras funcionalidades do projeto como reconhecimento das placas, interação com banco de dados e acionamento de barreiras físicas.

A figura 4.1 apresenta a tela de configuração de detecção de movimento da câmera utilizada.

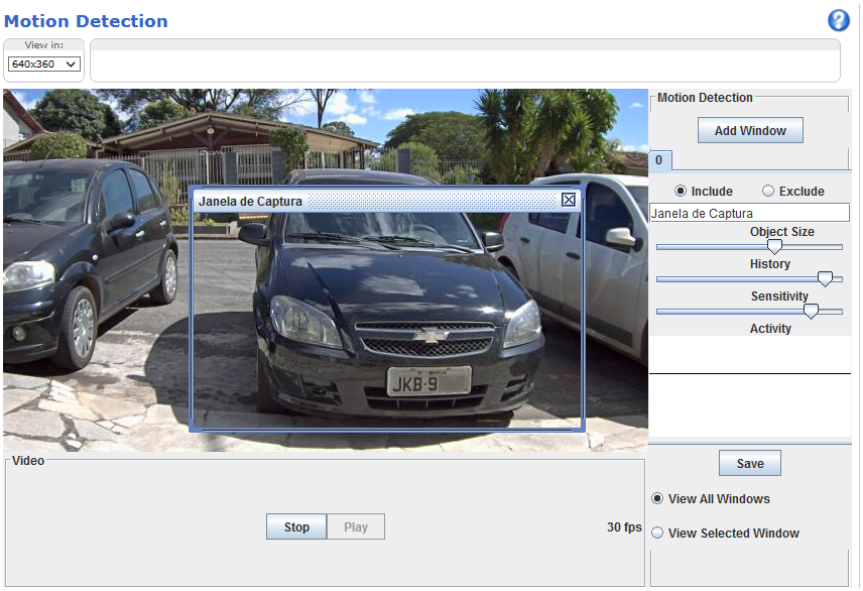


Figura 4.1: Detecção de movimento.

A figura 4.2 apresenta a tela de configuração de envio da imagem baseada em detecção por movimento e via FTP.

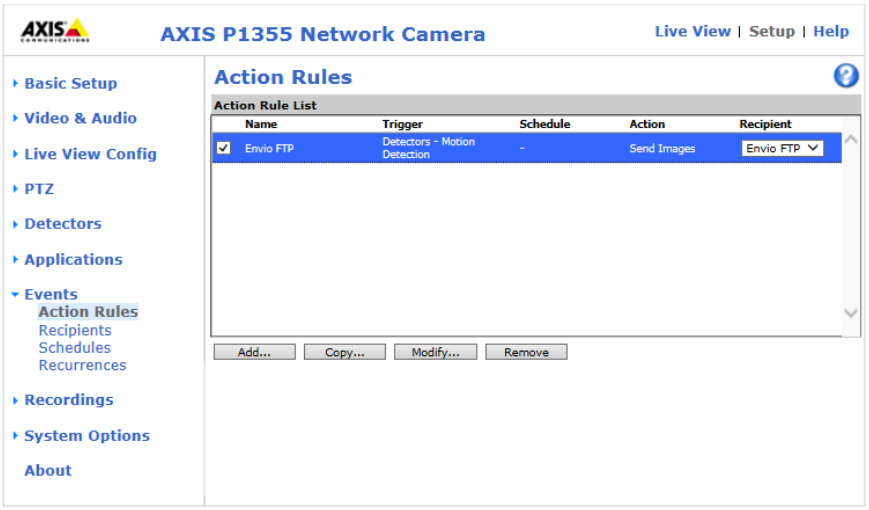


Figura 4.2: Envio via FTP.

Desde que obedecidos os parâmetros de tamanho do objeto e sensibilidade de movimento definidos nas configurações da câmera, haverá o envio para o diretório especificado.

4.1.1.1 Requisitos de Imagem

Diversos testes foram realizados buscando uma configuração ótima de imagem. Isso envolveu alteração na resolução da imagem, variação de iluminação, posicionamento da câmera e o relativo tamanho do automóvel no *frame* da câmera.

Os melhores resultados foram obtidos de tal forma que os requisitos padrões se tornaram:

- Resolução da imagem: 1280x800. A resolução da imagem está intrinsecamente relacionada com a capacidade da ferramenta *Tesseract* realizar o processo de OCR. Inicialmente acreditou-se que quanto maior a resolução, melhor seriam os resultados do reconhecimento de caracteres. Tal expectativa se mostrou frustrada visto que a ferramenta busca por caracteres com características específicas como altura e largura em um *range* específico de pixels. Esse comportamento será melhor abordado na seção relativa ao reconhecimento de caracteres. É importante destacar que as imagens devem ser capturadas nativamente na resolução proposta. Testes foram realizados capturando-se imagens em resoluções superiores e em seguida reduzidas por meio de *software* para a resolução ideal e o comportamento do sistema não foi o mesmo, apresentando uma queda considerável no desempenho do reconhecimento dos caracteres;

- Formato da imagem: JPEG ou PNG. Ambos os formatos produzem resultados semelhantes mas aconselha-se o uso do PNG, já que este retém melhor a qualidade da imagem, enquanto o JPEG compacta as imagens, podendo reduzir a qualidade.

- Iluminação: os testes sempre foram realizados em períodos com boa iluminação e sem incidência de sombra sobre o veículo. Entenda-se: dias de sol com imagens adquiridas na parte da manhã ou tarde. Nesses períodos a atuação do programa foi mais satisfatória. Não foram realizados teste noturnos, porém, acredita-se que com o uso de outros equipamentos específicos como iluminador infravermelho, o programa possa atuar de forma semelhante;

- Posicionamento da câmera: as capturas devem ser realizadas de tal forma que a câmera sempre esteja paralela ao nível do solo sobre qual o veículo se encontra e em altura semelhante ao nível do teto ou retrovisor do veículo, buscando uma visada frontal da placa;

- Tamanho do veículo em relação ao *frame*: a definição dessa etapa também foi bastante experimental. Imagens adquiridas em diversas distâncias foram postas à atuação do sistema a fim de se observar o comportamento do mesmo. Chegou-se à conclusão de que o cenário ideal é aquele em que o veículo preencha a maior parte do *frame* de vídeo da câmera, visto de forma frontal.

A imagem 4.5 da seção posterior é um exemplo que atende a todos os requisitos especificados acima.

Apesar dos requisitos estarem separados em categorias e estarem sendo apresentados na primeira seção do capítulo de Implementação, é importante frisar que os testes de requisitos de

imagem foram realizados simultaneamente com os testes de processamento da imagem, já que é praticamente impossível separar essas duas etapas.

As etapas de definição de requisitos da imagem, em conjunto com a de processamento, foram responsáveis por grande parte do tempo e do esforço necessários para o desenvolvimento deste projeto.

4.1.2 Reconhecimento das Placas

4.1.2.1 Etapa 1: início

O processamento de imagem inicia toda vez que uma nova imagem é adicionada em diretório determinado previamente. Pode ser qualquer diretório do sistema de arquivos, desde que acessível pelo administrador. Isso é possível através do *watchdog* especificado no capítulo anterior; que deve estar no mesmo diretório do *sript python* que deverá ser executado quando a inserção da nova imagem for capturada pelo *watchdog*. Essa espera pela imagem a ser processada pode ser representada pelo diagrama da figura 4.3.

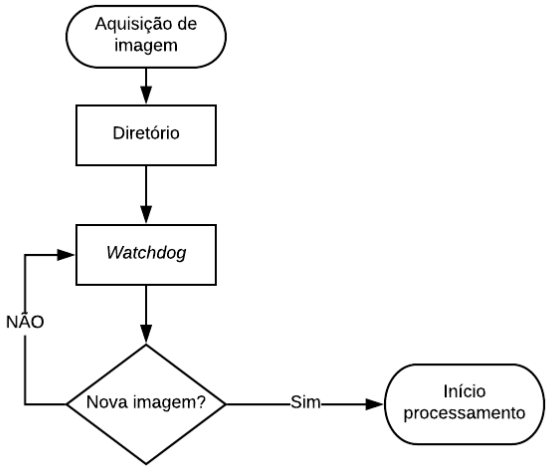


Figura 4.3: Atuação *watchdog*.

Ainda tratando do *watchdog*, é possível observar os dois fatores essenciais de seu funcionamento através da figura 4.4. A linha 16 define o diretório que deve ser monitorado e a linha 10 define o *script python* que deverá ser executado caso uma nova imagem seja adicionada nesse diretório.

```python
import time
import subprocess
import os
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler

class ExampleHandler(FileSystemEventHandler):
    def on_created(self, event):
        print "Evento capturado na pasta %s" % event.src_path
        subprocess.call(["python2.7", "/home/raphael/Desktop/Main1/Main.py"])
        #os.system("python /home/raphael/Desktop/parte2/check.py")

observer = Observer()
event_handler = ExampleHandler()

observer.schedule(event_handler, path='/home/raphael/Desktop/Main1')
observer.start()

try:
    while True:
        time.sleep(3)
except KeyboardInterrupt:
    observer.stop()

observer.join()
```

Figura 4.4: Código *watchdog*.

O processamento de imagem não se resume a uma etapa única e direta (uma entrada e uma saída); na verdade, durante o processo alguns arquivos são gerados antes de se obter o resultado final. Estes arquivos (que podem ser chamados de temporários) servem para marcar etapas que são determinantes ao longo do processo, sendo utilizados para verificar o andamento e localizar possíveis erros ao longo do processo.

Dadas as devidas adaptações de diretórios a serem observados e ações a serem tomadas, o uso dos *watchdogs* é recorrente em outras etapas do programa.

4.1.2.2 Etapa 2: Processamento (*Main.py*)

O *script python* (*Main.py*) invocado pelo primeiro *watchdog* realiza a maior parte do processamento da imagem antes da atuação da ferramenta de reconhecimento de caracteres. Aqui cabe informar os processos pelos quais a imagem passa.

1° Toma-se como base a imagem da figura 4.5. A primeira operação é transformá-la para escala de cinza pois torna as etapas seguintes mais eficientes.

Figura 4.5: Imagem em escala de cinza.

2° Em seguida é aplicado detecçaão de bordas pelo método *Canny Edge Detection* como abaixo na figura 4.6.

Figura 4.6: Método *Canny Edge Detection.*

A detecção de bordas facilita a etapa seguinte que é a de encontrar formas retangulares que obedeçam a critérios de altura e comprimento em unidades de pixels. Por formas retangulares deve-se entender que está sendo procurada a placa do veículo. Justifica-se aqui o motivo de fazer com que o veículo preencha a maior parte da imagem: dessa forma evita-se que formas indesejáveis sejam consideradas como placas. Como elas têm tamanhos muito semelhantes e foi definido anteriormente quais são os parâmetros de tamanho que as compõem, o programa dificilmente as confunde com algum outro elemento na imagem. Uma vez detectada a placa, ela é recortada da imagem e uma cópia sua é salva em outro diretório. Dessa forma, espera-se que o *script Main.py* forneça uma imagem como a da figura 4.7

Figura 4.7: Placa recortada da imagem original.

4.1.2.3 Etapa 3: Processamento (*thresh_ binary.py*)

Supondo que a etapa anterior tenha ocorrido com sucesso, tem-se que o *script thresh_ binary.py* será responsável por começar a ajustar a placa recortada da imagem original para ser reconhecida pela ferramenta de reconhecimento de caracteres. O acionamento dele também é feito por outro *watchdog* responsável por monitorar o diretório onde a imagem da placa recortada foi salva. Esse *script* realiza o processo conhecido como *binary threshold*.

O processo é bastante simples. Se o valor do pixel for maior que um valor limite, ele é atribuído a um valor (pode ser preto), caso contrário, é atribuído outro valor (pode ser branco). A função usada é *cv2.threshold*. O primeiro argumento é a imagem de origem, que deve ser uma imagem em escala 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 que (às vezes menor que) o valor do limite. A *OpenCV* fornece diferentes estilos de limiar e a decisão é feita pelo quarto parâmetro da função. A figura 4.8 ilustra essa configuração de forma mais clara.

```
7   img = cv2.imread('imgPlate.png')
8   gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
9
10  retval, threshold = cv2.threshold(gray, 50, 255, cv2.THRESH_BINARY)
```

Figura 4.8: Parâmetros da função.

Na linha 7 está sendo realizada a leitura da imagem base (neste caso é a placa recortada na etapa anterior) e em seguida, na linha 8, é feita a conversão da imagem para escala de cinza. Na linha 10 a função de *threshold* é de fato executada e é possível observar os parâmetros citados anteriormente. A nova imagem fica armazenada na variável "threshold" e é salva em outro diretório.

O *script* deve fornecer uma imagem como a da figura 4.9

Figura 4.9: Resultado da operação.

4.1.2.4 Etapa 4: Processamento e Reconhecimento (*cut_ read.py*)

O *script cut_ read.py* é responsável por duas tarefas importantes. A primeira é limitar a imagem gerada anteriormente, retirando as áreas em preto nas duas laterais. A solução foi recortar pois em testes anteriores à sua implementação, a ferramenta de reconhecimento comumente entendia como algum caractere, geralmente algum parêntese ou alguma chave, a junção entre a área lateral preta com a área branca. É possível fazer com que alguns símbolos ou caracteres sejam ignorados, mas essa seria uma solução pós reconhecimento. O resultado aguardado é como o da figura 4.10

Figura 4.10: Imagem recortada.

A segunda tarefa inclui o uso do chamado *Python-tesseract* ou simplesmente *pytesseract*. O *Python-tesseract* nada mais é do que um *wrapper* para a ferramenta *Tesseract* (FOUNDATION, 2018) especificada anteriormente. Ele é útil pois pode ser utilizado dentro de outro *script* como forma de invocação independente para o *Tesseract*. Vale destacar que a ferramenta foi projetada para aplicar reconhecimento de caracteres sobre textos em formatos de página nas línguas que possuem suporte. Para atender às especificações desse projeto, os arquivos de configuração tiveram que sofrer modificações que fizessem com que a ferramenta não procurasse por palavras ou textos em língua específica. Antes dessas modificações a ferramenta forçava a aparição de palavras da língua inglesa e errava completamente o reconhecimento dos caracteres das placas dos veículos. Afinal, diferente de palavras, os caracteres que compõem as placas não seguem uma sequência lógica, com a exceção da definição de 3 letras seguidas de 4 números. A figura 4.11 ilustra como é feita a invocação da ferramenta. Percebe-se que o reconhecimento de caracteres será será realizado sobre a imagem cortada anteriormente, identificada pelo nome "cortada.png" e o resultado armazenado na variável "text".

```
17   text = pytesseract.image_to_string(Image.open("cortada.png"))
```

Figura 4.11: Uso do *pytesseract*.

Em seguida, no mesmo *script*, como mostra a figura 4.12, é criado um arquivo que irá conter o texto com o reconhecimento realizado pelo *pytesseract*. O arquivo tem o nome "remover.txt" pois este ainda irá passar por outra etapa (que será chamada pela função da linha 27) antes de adicioná-lo aos registros do banco de dados.

```
20  file = open("remover.txt","w")
21  file.write(text.encode("utf-8"))
22  print "A placa gravada no arquivo remover.txt foi: "+(text)
23  file.seek(0)
24
25  time.sleep(2)
26
27  subprocess.call(["python2.7", "/home/raphael/Desktop/parte3/remover.py"])
28
29  cv2.waitKey(0)
```

Figura 4.12: Armazenamento do texto em arquivo.

4.1.2.5 Etapa 5: Tratamento (*remover.py*)

Essa etapa é considerada como um tratamento do texto reconhecido pela ferramenta de reconhecimento óptico de caracteres. As etapas de processamento de imagem chegaram a tal ponto onde o nível de acerto fosse aceitável. Entretanto, a chamada leitura das placas não é perfeita; existem caracteres que eventualmente aparecem no reconhecimento, geralmente no lugar do hífen separador das letras e números da placa, quando na verdade não existem. São eles: aspas simples, traço inferior, parêntesis ou chaves. Sendo assim, essa etapa busca eliminar esses caracteres, inclusive o hífen separador das letras e números. Alcança-se assim um padrão de como serão armazenadas as placas: somente letras e números. A figura 4.13 mostra o *script* responsável por tal tratamento, com atenção especial ao intervalo de linhas 9-15 que especificam os caracteres que devem ser eliminados.

```
1   #coding=utf-8
2   import string
3   import subprocess
4   import time
5
6   s = open("remover.txt").read()
7   print "A placa lida do arquivo remover.txt foi: "+(s)
8
9   s = s.replace("'","")
10  s = s.replace("-","")
11  s = s.replace("_","")
12  s = s.replace("(","")
13  s = s.replace(")","")
14  s = s.replace("[","")
15  s = s.replace("]","")
16
17  print "A nova placa do arquivo remover.txt: "+(s)
18  f = open("remover.txt", 'w')
19  f.write(s)
20  f.close()
21
22  time.sleep(3)
23
24  subprocess.call(["python2.7", "/home/raphael/Desktop/parte3/leitura.py"])
```

Figura 4.13: Remoção de caracteres.

Realizadas as modificações, o arquivo é atualizado e está pronto para seguir para a última etapa do processo como um todo. Nesse momento o reconhecimento da placa já foi finalizado, a etapa que segue é relativa à interação com o banco de dados.

4.1.2.6 Etapa 6: Banco de Dados (*addToDB.py*)

A partir do arquivo de texto gerado, o *sript addToDB.py* será responsável por ler o conteúdo do arquivo e adicioná-lo ao banco de dados, na respectiva tabela de leituras realizadas. Essa tabela conta com uma coluna com registros relativos às placas reconhecidas (leitura) e outra (data) com registro dos dias e horários em que o reconhecimento foi adicionado ao banco de dados. A figura 4.14 (retirada do *DB Browser for SQLite*) mostra um exemplo de testes de reconhecimento que foram simulados ao programa.

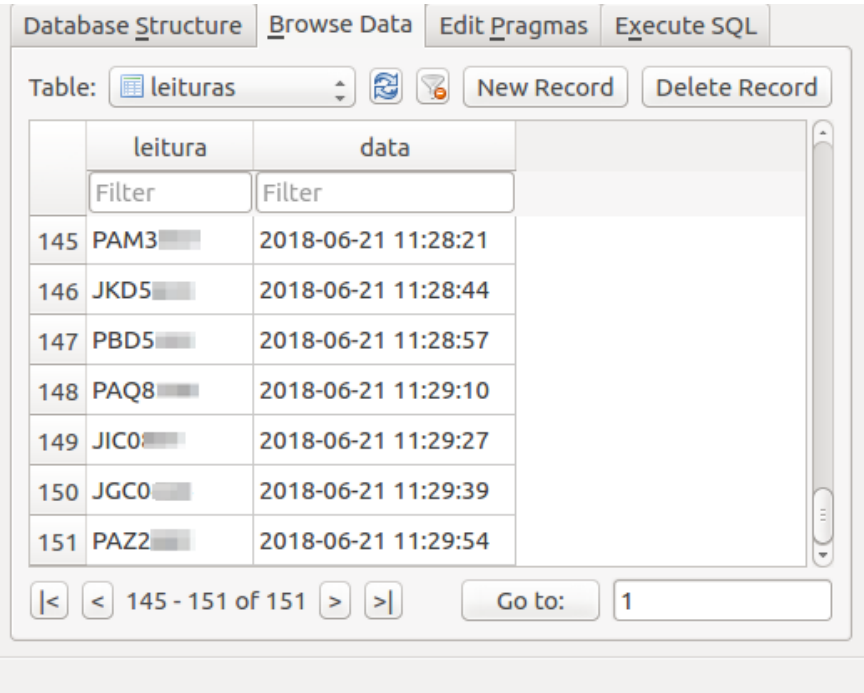


Figura 4.14: Tabela de reconhecimento (vista pelo DB Browser for SQLite).

A finalização da etapa 6, e consequentemente das anteriores com sucesso, se traduz em como o programa deve se comportar sempre que houver a inserção de uma imagem de um veículo visto de frente, seja de forma manual ou pela captura automática feita pela câmera.

4.1.2.7 Etapa paralela: *find_ delete.py*

Há uma etapa que ocorre paralelamente a todas as outras. Definida pelo *script find_ delete.py*, sua função é verificar a cada vinte segundos (esse período é ajustável) pelos arquivos de imagem que são criados entre as etapas 2 e 4 definidas acima. O tempo de vinte segundos foi definido como uma margem extremamente segura que possibilite a operação sobre todos os arquivos envolvidos nos processos antes que eles sejam deletados. Isso é necessário para evitar a acumulação de arquivos que servem temporariamente para realização da operação de reconhecimento. Posteriormente eles não são mais usados. Sendo assim, faz-se com que eles sejam excluídos dos diretórios e abram espaço para criação de novos arquivos. A figura 4.15 apresenta parte do *script* responsável pela tarefa.

```
1  import threading
2  import os
3
4
5  def printit():
6    threading.Timer(20.0, printit).start()
7    filename="/home/raphael/Desktop/Main1/image.jpg"
8    filename2="/home/raphael/Desktop/parte2/imgPlate.png"
9    filename3="/home/raphael/Desktop/PlacasLidas/letrapreta.png"
10   filename4="/home/raphael/Desktop/PlacasLidas/cortada.png"
11
12   ## deletar arquivo somente se ele existir no diretório ##
13   if os.path.exists(filename):
14     os.remove(filename)
15   else:
16     print("Desculpe, impossível remover o arquivo: %s" % filename)
```

Figura 4.15: Busca e exclusão dos arquivos.

Entre as linhas 7 e 10 são definidos os diretórios, e consequentemente os arquivos que servirão de alvo de busca e exclusão. Caso alguns, ou todos, não sejam encontrados, uma mensagem é exibida no terminal e o processo continua sequencialmente a cada vinte segundos.

4.2 Operação

Esta seção consiste na exposição dos mecanismos de operação do programa; não do ponto de vista do programador, e sim do ponto de vista de um possível usuário final. A abordagem aqui não é de cunho técnico como quando explicados os mecanismos nos capítulos anteriores. Esses mecanismos são transparentes ao usuário final e a ele basta operar o programa adicionando ou removendo placas que são de seu interesse e de acordo com uma das propostas explicitadas na seção 1.2.1: Objetivos Específicos. No entanto, somente quando necessário, será feita referência a algum elemento técnico para melhor desenvolvimento das ideias propostas.

4.2.1 Tela Principal

Abaixo segue imagem da tela inicial do programa, a partir da qual serão feitas todas as interações entre usuário e sistema.

Figura 4.16: Tela principal.

As interações possíveis estão identificadas pelos números que executam tarefas distintas:

- 1 (Digite a placa): campo onde deve ser introduzida a placa de interesse do usuário, somente letras e números. A imagem trata da tabela "*Whitelist*". Seu uso será abordado em seção seguinte;
- 2 (Digite o proprietário): identifica o proprietário da placa digitada no campo superior;
- 3 (Adicionar registro): adiciona a dupla "placa"e "proprietário"ao banco de dados;
- 4 (Apagar registro): realiza o processo para deletar a dupla de registros selecionada pelo nome do proprietário da placa;
- 5 (Atualizar): realiza um *refresh* da tabela de registros;
- 6 (Tempo Real): abre uma nova janela que contém os registros que estão sendo realizados através do processo de reconhecimento das placas;

- 7 (*Stream* de Video): abre uma nova janela que mostra a imagem vista pelas lentes da câmera;
- 8 (*Blacklist*): abre uma janela semelhante à principal. Seu uso será abordado na seção seguinte.

4.2.2 Proposta 1: *Blacklist*

Essa proposta está associada à possibilidade de emitir alertas gerados a partir de placas consideradas importantes ou que possuam algum *status* que necessite atenção. Para atender essa demanda foi criada a tabela *blacklist* no banco de dados. O botão "*blacklist*" identificado pelo número 8 da figura 4.16 leva o usuário a uma janela que possibilitará a inserção de placas na *blacklist*. Vide figura 4.17.

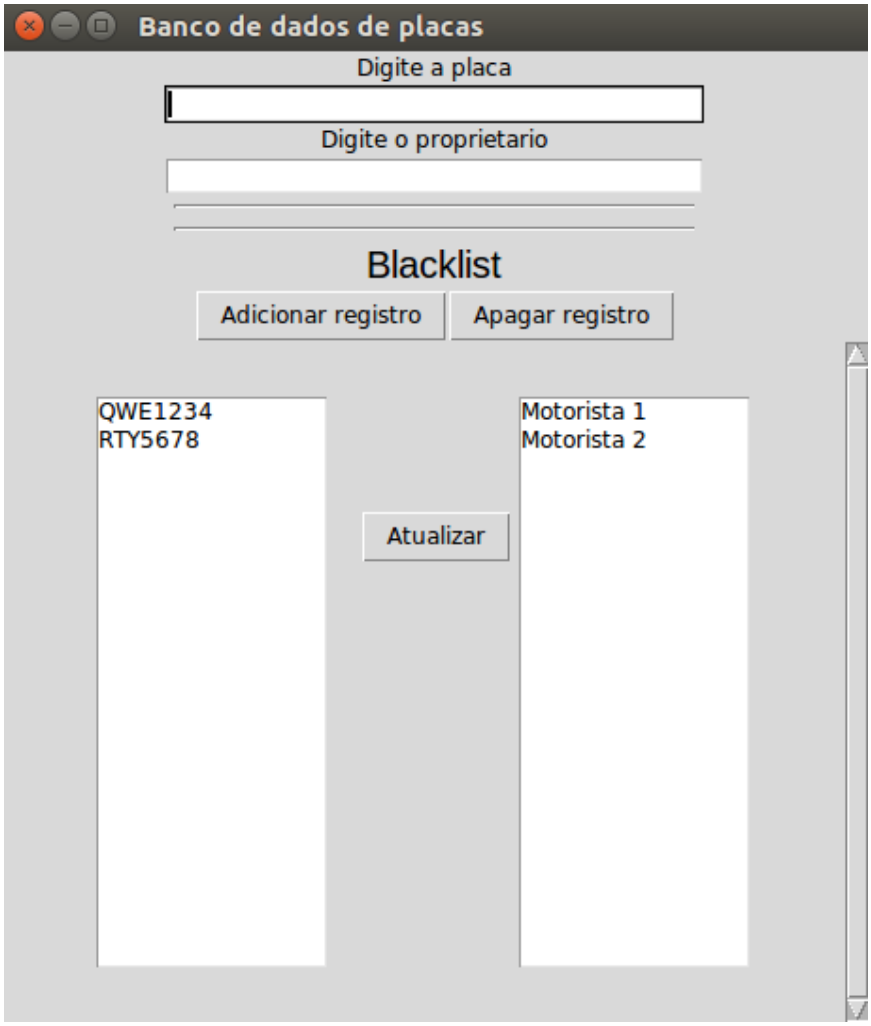


Figura 4.17: Janela *Blacklist*.

Para toda e qualquer placa reconhecida pelo programa haverá um processo de varredura na tabela *blacklist* em busca pela placa em questão. Caso ela esteja lá, como é o caso das placas

fictícias pertencentes ao motoristas 1 e 2 da imagem acima, automaticamente será gerado um alerta visual ao usuário contendo a placa reconhecida e o horário do registro realizado (figura 4.18). Em seguida é adicionada à tabela de reconhecimentos realizados (figura 4.19). Em caso negativo, o registro é adicionado à tabela de reconhecimentos realizados e nenhuma ação extra é realizada.

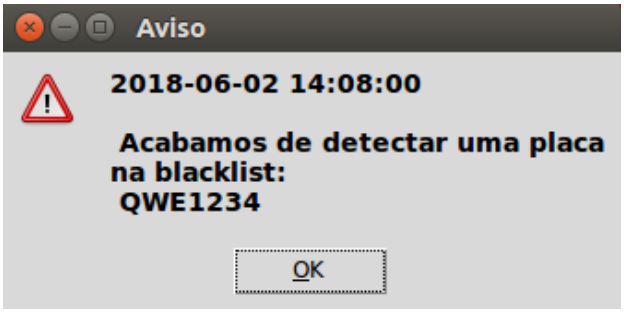


Figura 4.18: Alerta *Blacklist*.

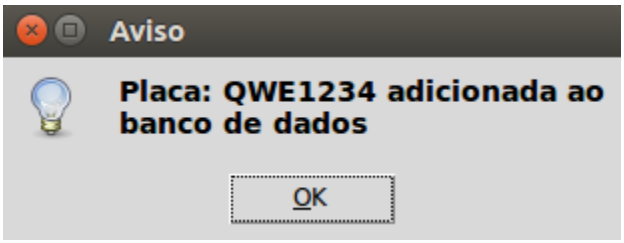


Figura 4.19: Alerta de placa adicionada ao banco de dados.

4.2.3 Proposta 2: *Whitelist*

A segunda proposta está associada à função de controle de acesso. Aqui faz-se necessário o uso do Raspberry Pi como controlador e atuador, através de comandos enviados ao módulo relé, das barreiras físicas que controlam o acesso ao ambiente.

Neste caso trabalha-se com a tabela *Whitelist* da figura 4.16. Para que determinado veículo tenha acesso liberado ao ambiente controlado, basta cadastrar a placa e o nome do proprietário nos campos apropriados. Caso algum veículo se aproxime e não tenha a placa cadastrada, o sistema vai reagir apenas reconhecendo a placa e adicionando o registro no banco dados, na tabela de reconhecimentos realizados. No entanto, a barreira física não será acionada (o acesso permanece bloqueado). Caso o usuário queira, além de bloquear o acesso do veículo, receber um alerta de que determinado veículo se aproximou, basta fazer o cadastro na *Blacklist* conforme instruções da seção 4.2.2.

O processo segue três etapas principais:

1° É realizado o reconhecimento da placa e em seguida verificado se ela consta na tabela *Blacklist*. Em caso positivo, o alerta é emitido e nenhuma outra ação é tomada. Caso negativo, o processo segue para a segunda etapa;

2° É feita a verificação na tabela *Whitelist*. Caso a placa conste nessa tabela, um *script* é chamado para realizar a comunicação com o Raspberry Pi e assume o que transcreve a etapa 3. Em caso negativo, o registro é apenas adicionado à tabela de reconhecimentos realizados;

3° Feita a comunicação com o Raspberry Pi, o comando enviado pelo sistema também invoca um *script python* que está armazenado no próprio Raspberry Pi e será responsável por acionar o módulo relé, atuando assim, na barreira física que controla o ambiente.

A figura 4.20 ilustra o *script*, simples e objetivo, responsável por realizar a comunicação e invocar o comando de atuação da barreira física.

```
1  import subprocess
2
3  subprocess.call("ssh pi@192.168.0.54 'cd Desktop && python teste.py'", shell = True)
```

Figura 4.20: *Script* responsável pela atuação do Raspberry Pi.

Tem-se dessa forma o controle de acesso baseado no reconhecimento de placa dos veículos. Para aqueles que possuem placas cadastradas na tabela "*whitelist*" o acesso será liberado. Em contra partida, os que não tiverem, não terão acesso ao ambiente controlado.

Capítulo 5

Análise

Este capítulo versa sobre a análise de desempenho dos principais componentes executados. São eles: *watchdogs*, processamento da imagem, reconhecimento das placas, operações no banco de dados e atuação do Raspberry Pi.

5.1 Obtenção dos dados

As análises foram realizadas após a definição de conclusão do sistema, ou seja, são dados que o representam em seu melhor estado de operação. Os resultados foram obtidos a partir da execução do sistema tendo como base uma amostra de 120 imagens de veículos diferentes. Visando uma maior agilidade para formação dessa amostra, as imagens foram obtidas manualmente e atendendo aos critérios da seção 4.1.1.1. A captura foi manual pois não exige a devida estruturação e posicionamento de câmera como no modo de captura automático.

5.2 *Watchdogs*

Os *watchdogs* são parte integrante essencial do processo geral. São eles que permitem a chamada de programas apropriados para operar sobre determinada etapa do processamento da imagem. Ao total são três *watchdogs* que devem estar sendo executados no *background*. Ao final do projeto foi avaliada a possibilidade de diminuir a quantidade dos *watchdogs* com o intuito de "enxugar" o sistema. Na verdade, foi verificado que é possível eliminar dois dos três *watchdogs*; só não seria possível eliminar aquele citado na seção 4.1.2.1, responsável por iniciar o processo através da chegada de uma nova imagem no diretório apropriado. Para eliminar os outros dois faz-se necessário incluir uma chamada interna à próxima etapa dentro dos *scripts* de processamento das imagens. No entanto, o preço a se pagar observado foi a diminuição na velocidade de execução do sistema. Em outras palavras, o processo ficou lento. Sendo assim, apesar da necessidade da inicialização de três *watchdogs* diferentes, justifica-se tal fato pelo ganho em desempenho de velocidade de execução.

Os tempos entre captura de evento e chamada para execução de *script* através do uso dos *watchdogs* refletem algo entre 1 e 1,5 segundos. A imagem 5.1 mostra a medição do tempo em entre a inserção de nova imagem no diretório inicial e a chamada do *script* que inicia o processo.

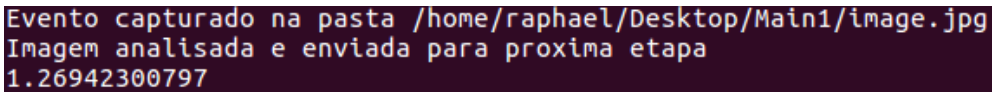


Figura 5.1: Tempo de execução *watchdog*.

Além disso, no que diz respeito à confiabilidade dos *watchdogs*, é possível afirmar que eles se comportam como se espera: sem falhas e sem queda de desempenho com maior carga de atividade. Ao longo do desenvolvimento do projeto os *watchdogs* foram postos em prática todas as vezes e não foi observado nenhum tipo de erro causado pela não execução ou por execução indesejada dos *watchdogs*.

5.3 Processamento da imagem

O objetivo final do processamento é entregar uma imagem que contenha apenas os caracteres da placa do veículo de forma que a ferramenta de reconhecimento óptico possa atuar da melhor forma. Sendo assim, o processamento da imagem está intrinsecamente ligado ao nível de assertividade do reconhecimento das placas. Para fins de análise o processamento pode ser dividido em dois aspectos principais: capacidade de identificar a placa na imagem fornecida e realização de operações para tornar os caracteres mais definidos para a etapa de reconhecimento. Na seção 4.1.2.4 ficou claro que basta limitar a imagem através do corte de áreas indesejadas e em seguida já pode ser realizado o reconhecimento dos caracteres. Desde que se obedeçam os requisitos impostos sobre a imagem, esse é o cenário que apresenta melhor resultado.

Ao longo do desenvolvimento do projeto a etapa de definição do *script* responsável por "achar" a placa do veículo na imagem e salvá-la em outro arquivo passou por diversas tentativas com "pontos de ataques"diferentes: inicialmente a mesma região da imagem era recortada e definia-se que ali era a placa do veículo. Tal abordagem se mostrou falha imediatamente, visto que é praticamente impossível que em capturas de imagens diferentes e de veículos diferentes o resultado seja sempre o recorte ideal da placa do veículo.

Em seguida, partindo para um cenário mais próximo do atual, definiu-se que era necessário achar na imagem um retângulo e que aquele seria a placa do veículo. Novamente essa tentativa de solução não foi efetiva pois diversas vezes outros elementos (que não a placa do veículo) eram identificados como retângulos e acarretavam no evidente erro de reconhecimento de caracteres.

Por fim foi elaborada uma solução capaz de identificar os possíveis retângulos da imagem e somente identificar como uma placa aqueles que obedecessem a critérios de largura e comprimento em pixels. Tal abordagem se mostrou extremamente eficaz, e, novamente, desde que sejam obedecidos os requisitos para captura de imagem, acredita-se que não haverá erros de detecção da placa na imagem. No universo das imagens utilizadas para teste, houve detecção correta em todos os

casos.

5.4 Operações no banco de dados

As operações no banco de dados se mostraram eficazes desde que foram implementadas. Processos de adição e exclusão de registros nas tabelas apropriadas do banco de dados: *whitelist*, *blacklist* e reconhecimentos realizados não apresentaram falhas em nenhum dos testes realizados. Associadas ao banco de dados existem as atividades que devem gerar notificações baseadas em leituras realizadas na tabelas. Tais notificações também nunca deixaram de ser exibidas ou foram exibidas quando não deveriam. Ou seja, dentro da amostra utilizada, todas foram armazenadas corretamente nas tabelas do banco de dados.

5.5 Atuação do Raspberry Pi

Quanto ao Raspberry Pi não houve momento em que a solicitação de sua atuação não foi correspondida. Por realizar tarefa de considerável simplicidade, resumida a executar o programa responsável pela atuação de um dos relés do módulo, não há grande margem para execução errônea ou para falhas. Durante os testes realizados com a amostra de imagens adquiridas, algumas placas foram cadastradas na tabela *whitelist* e quando apresentadas ao sistema, ocasionaram na atuação correta do Raspberry Pi em todas as vezes.

5.6 Reconhecimento das placas/ *Tesseract*

A operação da ferramenta *Tesseract* resulta no reconhecimento da placa. Não há dúvida de que representa a parte mais sensível do projeto no que se refere ao nível de assertividade e consequentemente confiabilidade do resultado apresentado. Além da amostra utilizada para produzir os resultados finais, vários testes foram realizados buscando melhor adaptação à ferramenta ao longo do desenvolvimento do projeto. A análise quanto ao desempenho do reconhecimento das placas pode ser feita de forma bastante objetiva. A melhor forma de apresentar os dados é separando o reconhecimento entre correto (quando o texto reconhecido representa, de fato, a placa do veículo da imagem), parcialmente correto (quando há acerto parcial dos caracteres contidos na placa do veículo da imagem) e incorretos (quando não há relação alguma entre o texto reconhecido e a placa do veículo em questão).

O *status* "parcialmente correto" merece melhor elaboração. Essa categoria foi criada pois há um aumento perceptível de reconhecimentos incorretos quando constata-se a presença de determinados caracteres nas placas do veículos. A ferramenta *Tesseract* eventualmente faz a troca de caracteres específicos por outros que são, em suas formas, semelhantes. As trocas geralmente ocorrem entre: B e 8; 5 e S; W e M; O e D. Portanto, os reconhecimentos que acertam todos os caracteres da placa, com exceção de algum desses citados acima, entram na categoria de reconhecimento "parcialmente correto".

Das 120 placas usadas como base de análise, 87 foram reconhecidas corretamente, 25 foram reconhecidas de forma parcialmente correta e 8 foram reconhecidas incorretamente. A figura 5.2 representa por meio de gráfico essas informações.

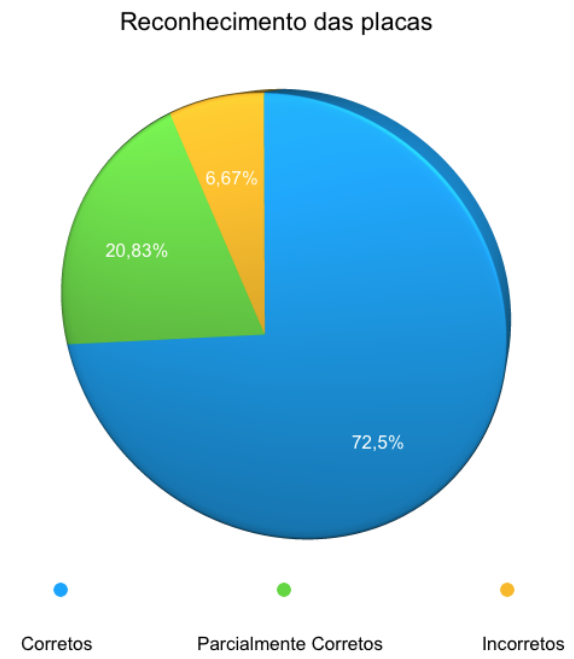


Figura 5.2: Desempenho do reconhecimento.

Outros projetos como os desenvolvidos por (OLIVEIRA, 2010) e (SILVA; ALVES, 2015) afirmam atingir taxas de acerto maiores, com valores de 90% e 92% respectivamente. Apesar da taxa de acerto do projeto aqui desenvolvido estar menor, percebe-se que com melhores ajustes para evitar a troca dos caracteres que decorrem em acertos parcialmente corretos, é possível atingir o nível dos projetos supracitados. Já empresas do mercado que atuam nesse ramo de reconhecimento de placas afirmam atingir taxas de acerto superiores a 98%, como fazem as empresas Genetec e *Intelligent Security Systems*.

Com relação à porcentagem de reconhecimentos incorretos não houve nenhuma constatação óbvia entre as placas ou entre as imagens capturadas. São pontos fora da curva que não possuem uma base consiste de motivos para tais ocorrências incorretas.

Capítulo 6

Conclusão

Neste projeto foi desenvolvido um sistema de reconhecimento de placas de veículos que pode operar em dois cenários diferentes. O primeiro sendo utilizado como uma espécie de sentinela, que emite alertas de acordo com os parâmetros definidos pelo usuário e o outro como controle de acesso, atuando na liberação de barreiras físicas. As principais ferramentas que possibilitaram tal projeto foram: *OpenCV* e *Tesseract*. Neste capítulo são tratadas as conclusões obtidas da análise e também as propostas de trabalhos futuros.

6.1 Considerações

O crescimento da frota veicular demanda um maior controle por parte das entidades competentes. O uso de *softwares* que auxiliem nesse controle e possibilitem o tratamento de grande quantidade de informações, filtrando e armazenando aquelas mais importantes, é de grande valia para a comunidade privada e pública; cada uma com seu foco de atuação.

A junção das ferramentas corretas permitiu criar um sistema capaz de atender essa demanda. Constatou-se que além das capacidades aqui exploradas, a biblioteca *OpenCV* possui funcionalidades e aplicações impressionantes, que vão além do necessitado para este projeto. Escolhida como meio para tratar as imagens, a biblioteca apresentou várias possibilidades de combinações de ferramentas de processamento de imagem para alcançar o objetivo a ela imposto.

Além disso, o projeto proporcionou uma maior imersão na linguagem *Python* e, principalmente, na sua interação com o Sistema Operacional Ubuntu. O Raspberry Pi, mesmo com participação discreta, mostrou como pode extrapolar o alcance do sistema permitindo a operação de outros equipamentos por meio do módulo relé.

De vital importância, o *Tesseract* cumpriu sua tarefa bem. É necessário fazer uma abstração e destacar, como feito na seção 4.1.2.4, que a ferramenta foi configurada para sair da sua "zona de conforto", visto que inicialmente foi projetada para reconhecimento de textos escritos em línguas pré-definidas em imagens com nada além do próprio texto; e não uma pequena sequência de letras e números com aparentemente nenhuma relação lógica com alguma língua.

Entra nesse momento a métrica mais importante para definir o desempenho do sistema: a porcentagem de acertos de reconhecimento das placas dos veículos. Como constatado no capítulo anterior, o valor de aproximadamente 72% não alcança os mesmos números prometidos e/ou obtidos por outros projetos e *softwares* de mercado que se propõem a realizar a mesma tarefa. Entretanto, comprova a viabilidade, através de mais ajustes, do sistema desenvolvido através deste projeto nos cenários propostos.

6.2 Trabalhos Futuros

Em trabalhos futuros será possível considerar a implementação do reconhecimento de caracteres por meio de técnicas como *machine learning*, que podem apresentar resultados mais consistentes baseados nos mecanismos do chamado aprendizado de máquina.

Pode ser considerado também um avanço técnico no que se refere aos requisitos de imagem, que são de certa forma limitados aos aspectos de resolução de imagem, iluminação e posicionamento da câmera expostos na seção 4.1.1.1.

Em comparação com *softwares* que realizam tarefas semelhantes, percebe-se que o tempo entre a captura da imagem e o reconhecimento da placa do veículo, algo em torno de 5 segundos, pode ser melhorado por meio de codificação mais enxuta e mais objetiva. As várias etapas pela qual o arquivo de imagem passa é fator que pode ser otimizado em busca de um tempo de reconhecimento menor.

Além disso, o programa possui capacidade para expansão e personalização de funções que podem ser extremamente úteis, por exemplo: emissão de relatórios diários, campo de busca de placas ou proprietários e adição de campos que vão além dos dois supracitados. Quanto à atuação das barreiras físicas é possível definir regras baseadas em horários e ainda realizar inserção manual de placas para verificação de autorização de entrada ou saída.

Finalizando, o projeto deixou claro que a participação do Raspberry Pi ficou limitada a simplesmente gerar o comando que atua no módulo relé. Quanto a isso, outra possibilidade que foi observada para implementações futuras é a de reunir todas as ferramentas e implementá-las dentro do raspberry pi, eliminando assim, a necessidade de um computador separado e dedicado. No entanto, é necessário fazer um estudo a cerca das limitações de *hardware* que ele possui e verificar se tal possibilidade é viável ou não.

Bibliografia

GONZALEZ, Rafael; WOODS, Richard. Processamento Digital de Imagens. In: [s.l.]: Pearson, 2009. cap.6, p. 259–301. ISBN 978-85-8143-586-2.

MARENGONI, Maurício; STRINGHINI, Denise. Tutorial: Introdução à Visão Computacional usando OpenCV. In:

OPENCV. **Histogram Equalization**. 2018. Disponível em: <http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_histograms/py_histogram_equalization/py_histogram_equalization.html#histogram-equalization>. Acesso em: 20 jun. 2018.

UFU. **Morfologia Matemática**. 2014. Disponível em: <http://www.facom.ufu.br/~backes/gsi058/Aula08-Morfologia.pdf>. Acesso em: 26 abr. 2018.

ACADEMY, Data Science. **O que é visão computacional?** 2017. Disponível em: <http://datascienceacademy.com.br/blog/o-que-e-visao-computacional/>. Acesso em: 30 abr. 2018.

DUIN, A. JAIN; R.; MAO, J. Statistical pattern recognition: A review. In: IEEE Transactionson Pattern Analysis and Machine Intelligence. [S.l.: s.n.], 2000.

A.R, Alexandria. Sistema de reconhecimento óptico de algarismos para medidores convencionais de energia. In:

MAIA DO VALE, Giovane; PORFÍRIO DAL POZ, Aluir. O processo de detecção de bordas de Canny: Fundamentos, algoritmos e avaliação experimental. In:

ORACLE. **Virtualbox Manual**. 2017. Disponível em: <https://www.virtualbox.org/manual/>. Acesso em: 1 nov. 2017.

PYTHON-SOFTWARE-FOUNDATION. **Python 2.7.13 Documentation**. 2017. Disponível em: <https://docs.python.org/2/tutorial/index.html>. Acesso em: 28 jun. 2017.

DIAKOPOULOS, Nick; CASS, Stephen. **The 2017 Top Programming Languages**. 2016. Disponível em: <https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages>. Acesso em: 17 nov. 2017.

BICKING, Ian. **Virtualenv**. 2017. Disponível em: <https://virtualenv.pypa.io/en/stable/>. Acesso em: 1 nov. 2017.

SCIPY. **What is NumPy?** 2017. Disponível em: <https://docs.scipy.org/doc/numpy-1.13.0/user/whatisnumpy.html>. Acesso em: 10 maio 2018.

OPENCV. **About**. 2018. Disponível em: <https://opencv.org/about.html>. Acesso em: 10 maio 2018.

GITHUB. **Tesseract OCR**. 2018. Disponível em: <https://github.com/tesseract-ocr/tesseract/blob/master/README.md>. Acesso em: 10 maio 2018.

SQLITE.ORG. **About SQLite**. 2018. Disponível em: <https://www.sqlite.org/about.html>. Acesso em: 10 maio 2018.

SQLITE, DB Browser for. **What it is**. 2018. Disponível em: <https://sqlitebrowser.org>. Acesso em: 2 jun. 2018.

RASPBERRYPI.ORG. **WHAT IS A RASPBERRY PI?** 2018. Disponível em: <https://www.raspberrypi.org/help/what-%5C%20is-a-raspberry-pi/>. Acesso em: 16 maio 2018.

RASPBIAN. **Welcome to Raspbian**. 2018. Disponível em: <http://www.raspbian.com>. Acesso em: 16 maio 2018.

AXIS. **AXIS P1355 Network Camera**. 2018. Disponível em: <https://www.axis.com/products/axis-p1355>. Acesso em: 21 jun. 2018.

FOUNDATION, Python Software. **Project Description**. 2018. Disponível em: <https://pypi.org/project/pytesseract/>. Acesso em: 28 maio 2018.

OLIVEIRA, Leonardo Augusto de. **Localização e reconhecimento de caracteres em placas de automóveis**. 2010. Diss. (Mestrado) – Escola de Engenharia de São Carlos, São Paulo.

SILVA, Marcelo da; ALVES, Robinson. Reconhecimento automático de veículos utilizando processamento digital. In: CONGRESSO de Métodos Numéricos em Engenharia. [S.l.: s.n.], 2015.