CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Monitoramento veicular utilizando câmeras de segurança convencionais e visão computacional

Jorge Takano Junior

Marília, 2016

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Monitoramento veicular utilizando câmeras de segurança convencionais e visão computacional

Monografia apresentada ao Centro Universitário Eurípides de Marília como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador:
Prof. Me. Rodolfo Barros Chiaramonte

Marília, 2016

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA - UNIVEM
MANTIDO PELA FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Jorge Takano Júnior

Monitoramento Veicular Utilizando Câmera de Segurança Convencional e Visão Computacional.

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Nota: 9,0 (nove)

Orientador:Rodolfo Barros Chiaramonte _____

1º.Examinador:Fabio Piola Navarro _____

2º.Examinador:Allan Mariano de Souza _____

Marília, 06 de dezembro de 2016.

AGRADECIMENTOS

Ao meu Senhor e Salvador Jesus Cristo, pois sem Ele nada seria possível. A minha esposa e companheira que me apoiou desde o princípio da graduação, a minha filha que me deu forças para me superar a cada dia, aos familiares e amigos pelo apoio.

Ao meu orientador Rodolfo, pela ajuda, pela preocupação e pela paciência que teve em todos esses anos, a todos os professores e colaboradores do Centro Universitário Eurípides de Marília.

Sumário

Lista de Figuras

Lista de Abreviações

CTB	Código de Trânsito Brasileiro.
DETRAN	Departamento Estadual de Trânsito.
CCD	Charge Coupled Device.
CMOS	Complementary Metal Oxide Semicondutor
CRT	Cathode Ray Tube.
LCD	Liquid Crystal Display.
RoI	Region of Interest

TAKANO, Jorge Junior. **Monitoramento veicular utilizando câmeras de segurança convencionais e visão computacional**. 2016. Monografia. (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha". Marília, 2016.

Resumo

O advento de aplicações móveis colaborativas voltadas ao trânsito têm ajudado condutores de veículos de diversas formas, como por exemplo, encontrar uma rota livre de congestionamento. No entanto, essas mesmas tecnologias permitem também a identificação de localização de radares, possibilitando que os condutores infratores não sejam penalizados. Por esse motivo o monitoramento de trânsito utilizando apenas radares tem se tornado ineficaz. Com isso propõe-se uma aplicação para monitoramento da velocidade de veículos em tempo real a partir de imagens obtidas de câmeras de segurança convencional. O uso deste sistema pode permitir à um operador facilmente identificar veículos em alta velocidade para fornecer informações aos órgãos de transito responsáveis.

Palavras-chave: Visão Computacional, Rastreamento de Veículos, Cálculo de Velocidade, Subtração de Fundo.

TAKANO, Jorge Junior. **Monitoramento veicular utilizando câmeras de segurança convencionais e visão computacional**. 2016. Monografia. (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha". Marília, 2016.

Abstract

The advent of collaborative mobile applications focused on traffic have been helping vehicle drivers in various ways e.g., finding a congestion-free route. However, these same technologies also allow the identification of radars' location, allowing that offenders drivers to not be penalized. For this reason traffic, monitoring using only radars has become ineffective. Thus, this work proposes an application to monitor the vehicles' speed in real time from images obtained from conventional security cameras. The use of this system may allow an operator easily identify vehicles at high speed to provide information to responsible traffic organs.

Keywords: Computer Vision, Vehicle Tracking, Speed Calculation, Background Subtraction.

1. INTRODUÇÃO

Em 2016, de janeiro até outubro, a polícia de Marília registrou 1.627 acidentes de trânsito com vítimas, sendo 75 fatais. Na comparação proporcional (incidência a cada grupo de mil habitantes), a cidade tem o pior indicador que São Paulo, Bauru, Prudente, Rio Claro e São Carlos.

No mesmo período, a região da cidade com trânsito mais perigoso foi a zona oeste, onde está localizado o campus universitário e parte da rodovia do Contorno. Foram 587 registros de acidentes. A zona norte é a segunda com maior número de ocorrências, com 435 registros. Em seguida aparece a região sul, com 324 registros. No centro, foram contabilizados 281 acidentes com vítima.

A assessoria de imprensa da SSP esclarece que os números não são referentes a quantidade de pessoas lesionadas, mas sim de boletins de ocorrência registrados pela polícia. Em parte dos casos, há mais de uma vítima.

Um balanço divulgado pelas Polícias Militar, Rodoviária federal e Rodoviária Estadual contabiliza 32.459 multas aplicadas durante as fiscalizações e acidentes na cidade e região.

Em vista desse problema, os gestores dos municípios espalham por suas cidades, milhares de equipamentos buscando controlar o tráfego de veículos, a fim de garantir a integridade e o bem estar de sua população. Existem vários tipos de equipamentos para auxiliar na busca pelos infratores, dentre eles os radares.

Eles podem ser classificados em dois tipos: metrológicos e não-metrológicos. Radares metrológicos são equipamentos que medem a velocidade dos veículos, tais como: fixos, portáteis e de lombadas. Radares não-metrológicos são equipamentos que não medem a velocidade, geralmente são específicos a uma determinada finalidade, por exemplo, os radares de semáforo que identificam veículos que passam pelo sinal vermelho ou radares de faixa de pedestre, que identificam os veículos que param em cima da faixa.

A tecnologia tem auxiliado nessa procura de infratores, porém ao longo do tempo este quadro tem mudado, novas tecnologias vem surgindo para auxiliar os condutores a se

beneficiarem de várias situações. Utilizando aplicativos de smartphones motoristas tem trocado informações, como por exemplo, a localização dos radares para que neste determinado local os condutores reduzam a velocidade não sendo autuados, outro exemplo é grupos formados em redes sociais, como por exemplo o WhatsApp, que alerta todos do grupo sobre a localização de radares móveis ou mesmo blitz, prejudicando a operação policial.

Por esse motivo os radares têm se tornado cada vez menos eficazes na busca de infratores, este projeto propõe disponibilizar outro método, o monitoramento através de câmera de segurança convencional e visão computacional como ferramenta para o reconhecimento de veículos trafegando acima da velocidade permitida, alertando o usuário da aplicação quando ocorrer as possíveis infrações. Desta forma os órgãos competentes podem agir com mais eficiência na busca pelos infratores. Tendo em vista que os radares convencionais não conseguem alertar as autoridades responsáveis no ato da infração, este projeto propõe auxilia-los, identificando os veículos que se movimentam com a velocidade acima da permitida, possibilitando rastrear criminosos em perseguições, identificando o sentido da via pela qual os veículos estão se deslocando.

1.1. Objetivos

Este trabalho tem como objetivo criar uma aplicação que monitore o tráfego veicular em vias urbanas, através de câmeras de segurança convencionais, utilizando técnicas como, tratamento das imagens tirando ruídos, extraindo pontos de interesse e segmentação de imagens baseada em subtração de fundo, utilizando o framework OpenCV para o processamento das imagens e codificado na linguagem Python. A aplicação será capaz de identificar os veículos e se algum deles estiver trafegando acima da velocidade permitida, alertar o usuário da aplicação, a fim de auxiliar as autoridades policiais no monitoramento do tráfego veicular.

Objetivos Específicos:

- Fazer revisão bibliográfica e estudo das técnicas necessárias para o desenvolvimento da aplicação;
- Obter conhecimento necessário para desenvolvimento em python (linguagem definida para desenvolver a aplicação);

- Utilizar técnicas de processamento de imagens tais como: Subtração de Fundo para facilitar o reconhecimento dos veículos e Gaussian Blur para filtrar os ruídos da imagem;
- Calcular a velocidade do tráfego do veículo;

1.2. Metodologia

O projeto foi dividido em duas partes principais:

- Levantamento bibliográfico e de trabalhos correlatos. Nesta etapa será realizada uma pesquisa buscando referências para as principais técnicas empregadas no projeto, comparações de algoritmos para utilizar os mais apropriados, definição das tecnologias e linguagem que seria codificada.
- Desenvolvimento da aplicação: foi adotado um vídeo "teste" de uma câmera de segurança convencional de aproximadamente cinco minutos do MIT (Massachusetts Institute of Technology), um *dataset* de trânsito para pesquisa sobre movimentações em cenas Wang, Ma e Grimson (2009) para utilizar no desenvolvimento do projeto.

1.3. Trabalhos Correlatos

O monitoramento de tráfego de veículos através de visão computacional, já foi utilizado em outros projetos. Cunha (2013) propôs um sistema automático para extrair dados do tráfego de veículos a partir do pós-processamento de vídeos, utilizando conceitos de visão computacional, desenvolvido em C++ com o framework OpenCV.

Para detectar os veículos, foram adotadas duas etapas: a modelagem do background onde foram investigados seis modelos estatísticos de métodos de geração de background, constatando que o método *Scoreboard*, que combina média e mediana, como melhor método considerando processamento e exatidão. A segunda etapa investigou seis métodos de segmentação desde a subtração de fundo até a segmentação por texturas, destacado o LFP dentre os descritores de textura.

Cunha (2013) conclui que analisando o desempenho desses métodos, o *Background Subtraction* foi mais adequado, apresentando mais performance e taxa de acerto totalizando 95,1% de média. Comparando os resultados obtidos pelo sistema proposto, com os resultados dos radares, foi obtido uma taxa de acerto de 92,7%, apresentando em alguns momentos valores acima, em outros abaixo dos valores obtidos nas coletas manuais.

Neste projeto Cunha retira a perspectiva da via para o processamento, diferente da aplicação proposta neste projeto. Cunha visa o pós-processamento dos vídeos, a aplicação proposta neste projeto visa em trabalhos futuros, utilizar em tempo real, para o auxilio imediato a autoridades policiais e órgãos competentes, porém nesta proposta de projeto o processamento será realizado com o pós-processamento de vídeos.

2. VISÃO COMPUTACIONAL

Para entender melhor o conceito de visão computacional empregado neste projeto, é necessário compreender sua definição. Segundo Marr (1982) psicólogo e pesquisador inglês que desenvolveu modelos sobre o sistema visual, utilizando informática no Instituto de Tecnologia de Massachusetts citado por Schneider (2007) afirma que "visão é o processo que produz, a partir de imagens do mundo externo, uma descrição que é útil ao usuário e que não é repleta de informações irrelevantes".

Outra definição muito conhecida é a de Trivedi e Rosenfeld (1989), afirmando que "visão computacional é a disciplina que investiga as questões computacionais e algorítmicas associadas à aquisição, ao processamento e à compreensão de imagens".

2.1. Elementos de Sistema de Visão Computacional

A seguir serão mostrados os elementos físicos básicos de um sistema de visão computacional. Segundo Gallon (2013) um sistema de visão computacional necessita de vários componentes interligados e dependentes uns dos outros. Abaixo são representados os componentes com suas respectivas descrições:

- **Aquisição da Imagem:** Segundo Schneider (2007), a aquisição da imagem é feita por um sensor sensível a banda do espectro eletromagnético, o qual converte a energia percebida em um sinal elétrico proporcional, sua saída pode ser em sinal elétrico na forma analógica, como também pode ser utilizado um conversor analógico-digital, para converter este sinal no formato digital.

  Ainda segundo Schneider (2007), existem vários tipos de sensores, específicos para cada banda do espectro. Sensores infravermelhos são capazes de detectar a temperatura dos corpos e transforma-la em imagem, por isso também são chamados de sensores térmicos. "As tecnologias de sensores de imagens mais comuns no mercado são o CCD e o CMOS", Schneider (2007).
- **Armazenamento:** Como as imagens já foram convertidas no formato digital na etapa anterior pelo dispositivo de aquisição, elas podem ser armazenadas como qualquer dado digital para que possam ser processadas posteriormente.
- **Processamento:** "A etapa de processamento de imagens envolve operações expressas em forma algorítmica", Schneider (2007). Segundo Cunha (2013) citando

Badenas (2001) a execução em tempo real com baixo custo em equipamento é um dos maiores desafios em visão computacional.

- **Interfaces de Comunicação:** São responsáveis pela troca de informação entre os dispositivos, podendo transmitir as imagens do dispositivo de aquisição para o de processamento, do processamento ao de armazenamento ou de exibição da imagem, ou até mesmo para um dispositivo externo do sistema.

  Segundo Schneider (2007), Os sensores tendem a possuir cada vez mais resolução, deixando as imagens com uma grande quantidade de informações para serem transmitidas, mesmo existindo varias técnicas de compactação, a quantidade de dados tende a ser cada vez maior.

  As imagens são exibidas por dispositivos que convertem os sinais digitais elétricos em imagens, tais como televisão, monitores, displays monocromáticos ou coloridos, do tipo CRT, LCD ou plasma, ou até mesmo impressoras.

2.2. Operações Em Imagens

Para o bom reconhecimento dos veículos no vídeo, foram utilizadas algumas operações nas imagens. Nesta seção serão abordadas as técnicas utilizadas no projeto, assim como seu funcionamento.

2.2.1. Gaussian Blur

A função Gausian Blur é utilizada para tirar os ruídos da imagem, realizando um processo de borrar a imagem através de uma função de Gauss, onde é calculada a média ponderada do pixel com relação aos seus pixels vizinhos.

Segundo Ferhat(2015), os pixels com desfocagem têm o valor de pico quando comparado com pixels vizinhos com a média ponderada e a distância máxima é de cinco pixels. Na Figura 1 é possível visualizar a imagem antes da utilização da função e depois de aplicado a máscara do filtro.

Figura 1 - Ilustração da função GaussianBlur.

Fonte: Ilustração adaptada da Documentação do OpenCV 3.0.0-dev.

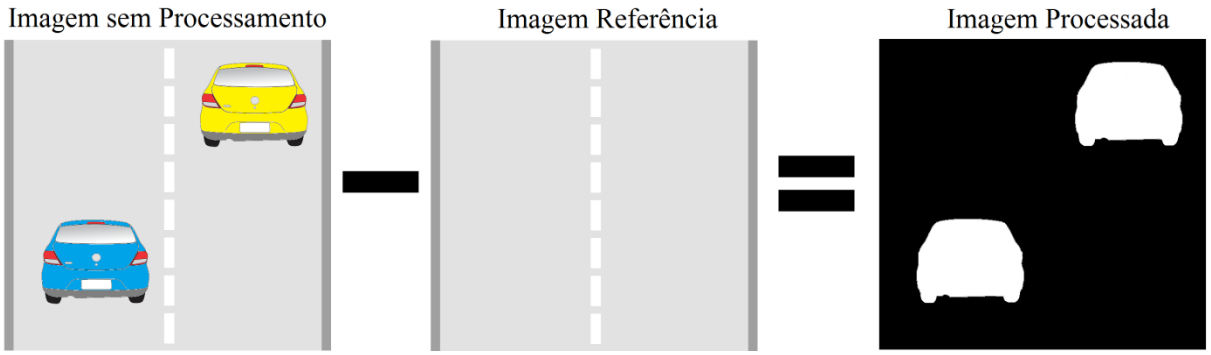
2.2.2. Subtração de Fundo

Para auxiliar na detecção dos veículos na via, foi utilizada a técnica de segmentação baseada em subtração de fundo. Antes de falar sobre subtração de fundo é necessário entender melhor o que é segmentação de imagens. Segundo Ballard e Brown (1982, apud FONSECA, 2007, p. 33), em visão computacional, segmentação da imagem é o resultado do agrupamento de partes de uma imagem generalizada, em unidades homogêneas considerando um ou mais aspectos.

Segundo Gonzalez e Woods (2007), no processamento de imagens a segmentação automática é uma das tarefas mais difíceis e isso que determina o sucesso ou fracasso da aplicação. Existem vários métodos de segmentação, que vai do mais simples que opera pixel-a-pixel até os mais complexos que interpretam as imagens por modelos matemáticos.

Existem várias técnicas de segmentação, porém as mais comuns fazem parte do método de subtração de fundo. A subtração do fundo consiste em retirar da imagem atual (sem processamento) todo o fundo a partir de uma imagem referência, como ilustrado na Figura 2, ou seja, é o resultado da comparação da imagem com os veículos e outra imagem somente da via sem os veículos.

Figura 2 - Ilustração do processo de Subtração de Fundo.



Fonte: Ilustração adaptada de Britto (2011).

Cheung e Kamath (2004) classificaram as técnicas de modelagem de fundo, em recursiva e não-recursiva. Modelagem de fundo é a base da subtração de fundo, é nesta etapa que a imagem de referência é construída. Na seção seguinte serão abordadas as técnicas de modelagem recursiva e não-recursiva.

2.2.2.1. Modelagem não-recursiva

As técnicas não-recursivas armazenam uma determinada quantidade de imagens no buffer e estima o fundo baseado na variação temporal de cada pixel dentro do buffer. Abaixo é apresentado um pequeno resumo de algumas técnicas citadas por Schneider (2007) tais como:

- *Diferenciação de frame:* utiliza o frame do instante t-1 como modelo de fundo no frame do instante t, é a técnica mais simples.
- *Filtro da Média:* é calculada a média de cada pixel dentro do buffer de cada imagem amostrada, estimando o fundo.
- *Filtro linear preditivo:* o coeficiente do filtro é estimado para cada frame baseado na covariância de cada amostra. O fundo é estimado aplicando a técnica para cada pixel dentro do buffer, deixando inviável a aplicação em tempo real, SCHNEIDER (2007).

2.2.2.2. Modelagem recursiva

As técnicas recursivas diferente das técnicas não-recursivas não armazenam um buffer para estimar o fundo, o modelo de fundo é atualizado recursivamente através de cada frame

de entrada. Por não armazenar buffer não é necessário ter grande capacidade de memoria, porém se houver um erro de modelagem, este pode alterar os resultados por um longo período de tempo. Schneider (2007), resume algumas técnicas de modelagem recursiva:

- *Aproximação do filtro da média:* Técnica apresentada por McFarlane e Schopfeld (1995) que consiste em um filtro recursivo simples para estimar a média. Aplicando a técnica, a estimação da média é incrementada em um se o pixel de entrada for maior que o estimado, ou decrementada em um se for menor que o estimado. Com isso é armazenado somente uma imagem de referência na memoria.

2.2.3. Threshold

A função Threshold ou limiarização, tem como objetivo separar os pixels em duas regiões através de um limiar, a região de interesse, ou seja, a região onde os objetos de interesse se destacam e a região de contraste com a região anterior.

A imagem é representada por uma matriz *I(i,j)* onde *i* é o número de linhas e *j* é o número de colunas, cada elemento da matriz representa a cor de um pixel, que pode assumir o valor 0 ou 1 para imagens binárias, de 0 a 255 para imagens em tons de cinza e três valores de 0 a 255 para imagens coloridas. Esse processo também é conhecido como binarização, segundo Cunha(2013) comparando os pixels da imagem com o valor *threshold* podemos classificar os pixels com a Equação 1 abaixo:

$$T(x,y) = \begin{cases} 0, se\ f(x,y) < \tau \\ 1, se\ f(x,y) \geq \tau \end{cases} \quad \text{(Equação 1)}$$

Ainda segundo Cunha(2013), os pixels da imagem *f(x,y)*, onde *x* é a linha e *y* a coluna, são comparados com os valores do limiar, se os valores dos pixels forem menores que τ os valores nas imagens resultantes *T(x,y)* recebem 0 (preto) e caso os valores forem maiores ou iguais a τ, *T(x,y)* recebem 1 (branco). Na Figura 3 é possível visualizar o resultado da binarização da imagem, após ser transformada em tons de cinza.

Figura 3 - Ilustração do processo de Binarização.

Fonte: Ilustração adaptada da Documentação do OpenCV 3.0.0-dev.

2.2.4. Morfologia

Morfologia é uma biologia que estuda as formas e estruturas dos animais e plantas. A Morfologia Matemática foi apresentada por pesquisas conjuntas de Georges Mathelon e Jean Serra em 1964, inicialmente com imagens binárias. A morfologia matemática é baseada na teoria dos conjuntos, por isso suas operações utilizam operações entre conjuntos. O objetivo é extrair informações de um conjunto desconhecido, através de um conjunto definido denominado "*elemento estruturante*".

Nas próximas seções serão apresentadas algumas operações fundamentais, assim como suas expressões algébricas e exemplos visuais. Para os exemplos foi utilizado um elemento estruturante retangular MORPH_RECT.

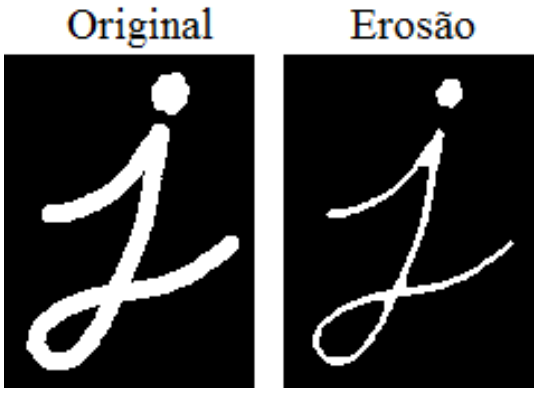
2.2.4.1. Erosão

A operação é realizada fazendo a translação de um conjunto A por x, definida pela Equação 2 abaixo:

$$A \ominus B = \{ x \mid (B)_x \subseteq A \} \qquad \text{(Equação 2)}$$

A erosão de um conjunto A pelo elemento estruturante B é o conjunto de todos os pontos x, tais que, B transladado por x, esteja contido em A. A Figura 4, demonstra a operação de erosão.

Figura 4 - Ilustração da Operação de Erosão.



Fonte: Ilustração adaptada da Documentação do OpenCV 3.0.0-dev.

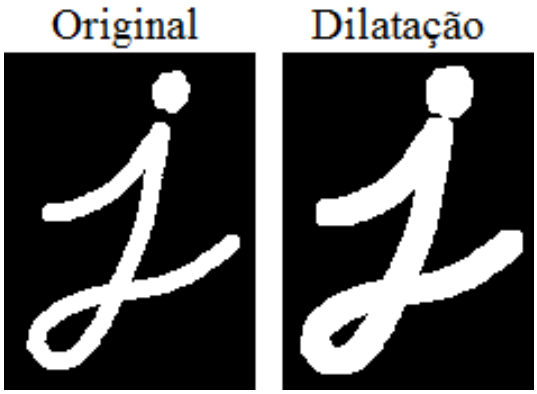
2.2.4.2. Dilatação

A operação de Dilatação de um conjunto *A* por *B* (*A*⊕*B*), pode ser definida pela Equação 3 abaixo:

$$A \oplus B = \{ x \mid \left(\hat{B}\right)_x \cap A \neq \emptyset \} \qquad \text{(Equação 3)}$$

A dilatação é um conjunto de todos os elementos *x*, de forma que as translações da reflexão de B por *x*, sobreponham-se ao conjunto *A* por pelo menos um elemento não-nulo. A Figura 5, demonstra a operação da dilatação.

Figura 5 - Ilustração da Operação de Dilatação.



Fonte: Ilustração adaptada da Documentação do OpenCV 3.0.0-dev.

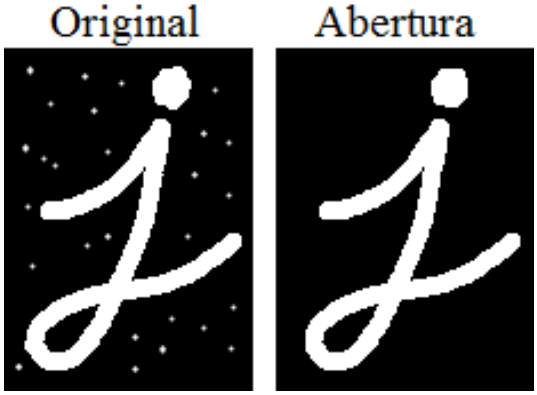
2.2.4.3. Abertura

A operação de Abertura de um conjunto *A* por um elemento estruturante *B* (*A*∘*B*), pode ser definida pela Equação 4 abaixo:

$$A \circ B = (A \ominus B) \oplus B \qquad \text{(Equação 4)}$$

A Abertura é o resultado de uma erosão de *A* por *B* seguida de uma dilatação por *B*. A Figura 6, demonstra a operação de abertura.

Figura 6 - Ilustração da Operação de Abertura.



Fonte: Ilustração adaptada da Documentação do OpenCV 3.0.0-dev.

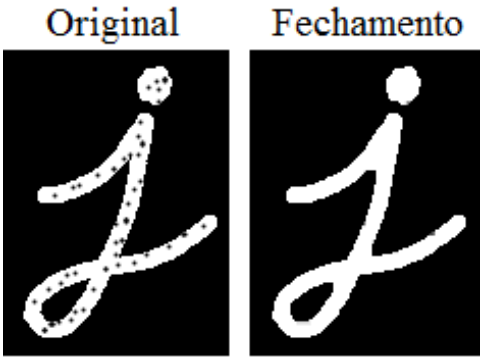
2.2.4.4. Fechamento

A operação de Fechamento de um conjunto *A* por um elemento estruturante *B* (*A*•*B*), pode ser definida pela Equação 5 abaixo:

$$A \bullet B = (A \oplus B) \ominus B \qquad \text{(Equação 5)}$$

O Fechamento é o resultado de uma dilatação de *A* por *B* seguida de uma erosão por *B*. Oposto da abertura que suavisa os bicos da imagem, o fechamento suavisa o bico dentro do plano de fundo da imagem. A Figura 7, demonstra a operação de fechamento.

Figura 7 - Ilustração da Operação de Fechamento.



Fonte: Ilustração adaptada da Documentação do OpenCV 3.0.0-dev.

2.2.5. Pontos de Interesse

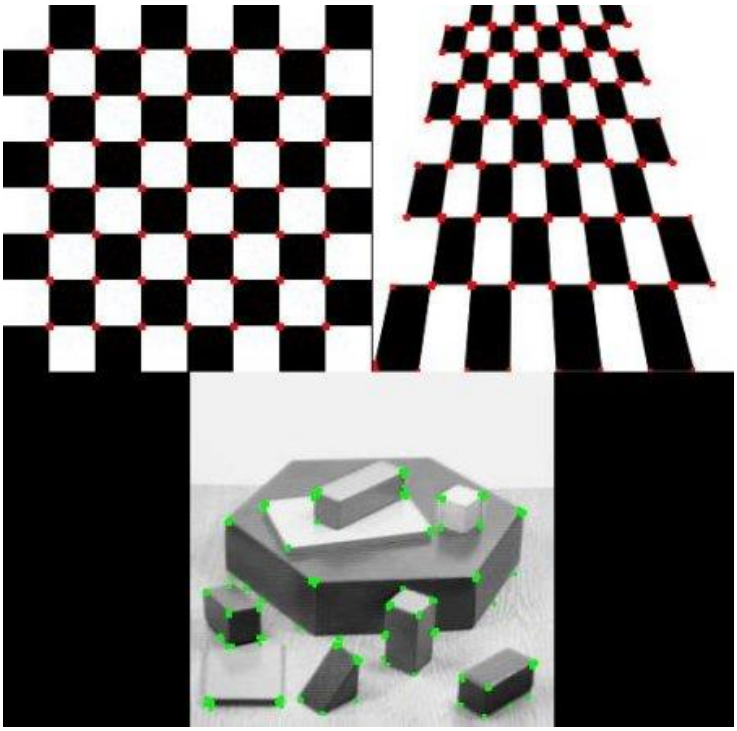
Segundo Codevilla(2015), pontos de interesse são definidos por padrões de uma imagem que diferem de seus vizinhos. São pontos que tem algumas características em comum, que em uma imagem tem alguma relevância. Um exemplo de ponto de interesse é uma quina de um monitor, assim como seu botão de ligar e desligar.

A utilização de pontos de interesse reduz significativamente o custo computacional, descarta ruídos na imagem por utilizar os pontos relevantes, possibilita sem a necessidade de segmentar o reconhecimento de cenas, dentre outras vantagens, porém para que um ponto de interesse seja eficaz, algumas propriedades são imprescindíveis (TUYTELAARS; MIKOLAJCZYK, 2008), dentre as mais importantes tem-se: Repetibilidade, onde um ponto de interesse representam características que podem ser encontradas em determinados objetos e quando comparados em imagens diferentes, os pontos de interesse são detectados em ambas as imagens.

Distintividade, onde um ponto de interesse deve representar características distintas, destacando-os sobre as demais características e que sejam especificas de um determinado objeto, dessa forma o objeto é detectado dentre outros objetos, Codevilla(2015).

Para o projeto foi utilizado a função goodFeaturesToTrack da biblioteca OpenCV. A Figura 8, demonstra os pontos de interesse detectados na imagem.

Figura 8 – Detecção de pontos de interesse.



Fonte: Ilustração adaptada da Documentação do OpenCV 3.1.0

2.2.6. OpenCV

Em seu próprio site OpenCV (*Open Source Computer Vision Library*) é definido como uma biblioteca de visão computacional liberada sob licença BSD(licença de código aberto), portanto, pode ser utilizada tanto no meio acadêmico quanto no meio comercial gratuitamente.

Com mais de 2500 algoritmos otimizados de visão computacional e aprendizado de máquina, foi projetada para ter eficiência computacional com grande foco em aplicações de tempo real, podendo ser aproveitado com processamento multi-core. Possui interface de C, C++, Python e Java com suporte para Windows, Linux, Mac OS, iOS e Android.

OpenCV é utilizado no mundo inteiro, sua comunidade possui mais de 47 mil usuários e estima-se que foram realizados mais de 9 milhões de downlods, empresas bem estabelecida como Google, Intel, IBM, Microsoft, Sony, Honda, Toyota, também utilizam a biblioteca. OpenCV é uma biblioteca bastante utilizada em vários projetos para o processamento de imagens, por ter uma documentação simples e fácil de entender, por encontrar muitos exemplos de utilização das funções, por esses motivos, foi determinado que seria utilizada neste projeto.

3. MONVIC – MONITORAMENTO VEICULAR COM VISÃO COMPUTACIONAL

Para permitir o monitoramento do trânsito em tempo real propõe-se o Monvic. O Monvic é uma aplicação de monitoramento veicular utilizando câmeras de segurança convencional e visão computacional. O objetivo do projeto é identificar veículos trafegando em vias urbanas, medir a velocidade com que os veículos estão se movimentando e emitir um alerta ao usuário do sistema identificando-os.

A próxima seção irá abordar como foi desenvolvido o projeto, que foi dividido em três etapas:

1. As técnicas de processamento de imagem utilizando a biblioteca OpenCV;
2. O algoritmo que identifica um conjunto de pontos de interesse em um veículo;
3. O cálculo da velocidade, também será apresentado alguns resultados obtidos.

3.1. Técnicas de processamento.

Para auxiliar no reconhecimento dos veículos, foram aplicadas algumas técnicas de processamento de imagens utilizando a biblioteca OpenCV. Serão abordadas as funções de tratamento das imagens para facilitar a identificação de cada veículo e consequentemente medir sua velocidade.

3.1.1. Subtração de Fundo

A função cv2.absdiff calcula a diferença entre duas imagens ou matrizes, ou seja, esta função faz a subtração do fundo. Para cada frame do vídeo é retirada a imagem da via sem nenhum objeto, facilitando o reconhecimento dos veículos e pedestres que estão trafegando na via, como foi demonstrado na Figura 2 da seção 2.2.2. Na Figura 9 é demonstrada a imagem de referência para a subtração de fundo e na Figura 10 a imagem que esta sendo processada.

Figura 9 – Imagem utilizada como fundo para a subtração.

Fonte: Autoria Própria

Figura 10 – Imagem que esta sendo processada.

Fonte: Autoria Própria

Nota-se que na Figura 9 não tem nada que se desloque na imagem, para que ao subtrair essa imagem de fundo do frame processado (Figura 10), seja possível identificar todos os objetos adicionais, ou seja, os possíveis veículos. Na Figura 11, é demonstrado o resultado da subtração, é através dessa diferença que o processamento continuará a ser realizado.

Figura 11 – Resultado da Subtração de Fundo.

Fonte: Autoria Própria

Nota-se que na Figura 11, foram reconhecidos dois veículos e se prestar atenção onde está marcado com um circulo na figura, tem alguns pontos reconhecidos, se trata do reconhecimento do movimento das folhas das árvores, porém não serão considerados, nas próximas seções será abordado o tratamento para desconsiderá-los.

3.1.2. Redução de Ruídos

Feita a subtração das imagens, os objetos desnecessários foram retirados da imagem, ou seja, tudo que não é relevante foi removido, porém para facilitar ainda mais no reconhecimento dos veículos é necessário remover os ruídos das imagens, utilizando a função cv2.GaussianBlur ou a "técnica de borrar".

Com a função cv2.GaussianBlur foram filtrados os ruídos das imagens como descrito na seção 2.2.1, porém antes de utiliza-la é necessário converter a imagem em tons de cinza com a função cv2.cvtColor. A Figura 12 demonstra a Figura 10 em tons de cinza utilizando a propriedade cv2.COLOR_BGR2GRAY.

Figura 12 – Imagem convertida em tons de cinza.

Fonte: Autoria Própria

A Figura 13 demonstra o resultado da função cv2.GaussianBlur, utilizando a imagem convertida em tons de cinza, apenas para visualização da técnica, pois para o projeto foi utilizada a imagem resultante da subtração de fundo.

Figura 13 – Imagem convertida em tons de cinza utilizando a função cv2.GaussianBlur.

Fonte: Autoria Própria

3.1.3. Binarização

Como descrito anteriormente, foi necessário utilizar no projeto as imagens binarizadas. Com o resultado do filtro de ruídos, as imagens foram binarizadas, ou seja, ao invés de tons de cinza a imagem passa a ter apenas duas cores preto e branco, para isso foi utilizada a propriedade cv2.THRESH_BINARY. A função cv2.threshold realiza a binarização da imagem, mas para que isso seja possível é necessário que a imagem esteja em tons de cinza. A Figura 14 demonstra a imagem binarizada.

Figura 14 – Imagem resultante da função cv2.GaussianBlur binarizada.

Fonte: Autoria Própria

3.1.4. Processamento Morfológico

A dilatação é o ultimo passo no tratamento das imagens. Após binarizar a imagem é utilizada a operação morfológica de dilatação a fim de juntar os “pedaços” do objeto encontrado, para que se torne um objeto único e consequentemente poder classificá-lo como um veículo e posteriormente medir sua velocidade.

Para isso a função cv2.morphologyEx foi utilizada com a propriedade cv2.MORPH_DILATE, para aplicar a dilatação, o elemento estruturante utilizado foi

np.ones((10,10),np.uint8), que representa uma matriz com o valor “1” de 10 linhas e 10 colunas, a Figura 15 demonstra a dilatação da imagem binarizada.

Figura 15 – Imagem binarizada dilatada.

Fonte: Autoria Própria

É possível notar na Figura 14, que existem partes dos veículos separadas umas das outras, na Figura 15 com a aplicação da dilatação as partes se juntaram e se tornaram um único formato, para que este objeto esteja inteiro e não em “partes”.

3.2. Cálculo da Velocidade

Finalizado o processo de tratamento da imagem, inicia-se o processo de calcular a velocidade com que o veículo está se movimentando, porém para que isso seja possível é necessário passar por alguns passos: Detecção dos pontos de interesse da imagem; Classificação dos objetos, para identificar se o objeto é um veículo ou um pedestre; Encontrar os contornos dos objetos para agruparem em conjuntos de pontos de interesse; Cálculo da Velocidade e reconhecimento da infração.

3.2.1.1. Detecção dos Pontos de Interesse.

Para detectar os pontos de interesse da imagem foi utilizada a função cv2.goodFeaturesToTrack. Esta função encontra os pontos de interesse na imagem como descrito no item 2.2.5. Com a imagem tratada é possível encontrar os seguintes pontos demonstrados na Figura 16.

Figura 16 – Detecção dos pontos de interesse na imagem tratada.

Fonte: Autoria Própria

Esses pontos são necessários para desenhar os contornos dos objetos, esse assunto será abordado na seção seguinte.

3.2.1.2. Classificação dos Objetos.

Neste ponto do processamento os objetos serão classificados em *Veículos* e *Pedestres*. Para realizar a classificação é necessário obter o contorno dos objetos, para isso foi utilizada a função cv2.findContours.

Com o resultado da função é possível calcular a área do contorno através da função cv2.contourArea, por fim através da área dos objetos, foi possível classifica-los como veículos e pedestres. A Figura 17 demonstra a imagem vista pelo usuário e com o contorno dos veículos, o contorno está apenas para ilustração, pois o usuário não visualizará o contorno e a Figura 18 demonstra o contorno na imagem binarizada.

Figura 17 – Demonstração do contorno dos veículos na visão do usuário.

Fonte: Autoria Própria

Figura 18 – Demonstração do contorno dos veículos na imagem binarizada.

Fonte: Autoria Própria

Com os pedestres descartados, ou seja, descartando os objetos que tem uma área pequena, restaram apenas os veículos na imagem, porém o problema foi agrupar um conjunto de pontos e classifica-los como um único veículo. Para resolver esse problema, foi considerado apenas um único ponto colocado no centro do contorno como ilustra a Figura 19.

Figura 19 – Demonstração do contorno com o ponto central.

Os veículos são representados por seus respectivos pontos centrais. Para calcular a velocidade, será considerada a distância do ponto do frame anterior com o frame atual, mas outro problema encontrado foi como identificar o mesmo ponto em frames diferentes, ou seja, como identificar o mesmo ponto nos frames anterior e atual.

Considerando que entre o frame anterior e o frame atual o ponto vai percorrer uma pequena distância, bastaria percorrer ponto do frame anterior e percorrer ponto a ponto do frame atual para encontrar seu respectivo ponto, porém por ter que percorrer ponto a ponto nos dois frames o algoritmo ficaria custoso.

Para resolver esse problema bastaria apenas ordenar as listas de pontos dos frames anteriores e atuais e posteriormente calcular suas distâncias. Um problema encontrado por esta abordagem é que quando um veículo entra na área de monitoramento, a lista de pontos anteriores terá menos pontos que a lista de pontos atuais, resultando em erro na aplicação.

Esta questão foi resolvida verificando se as duas listas tem a mesma quantidade de elementos, antes de calcular a distância, se não houver não é calculada a distância neste instante, porém para o próximo frame do vídeo, as listas terão a mesma quantidade de pontos.

Portanto, pode-se concluir que para identificar seu respectivo ponto entre os frames, basta ordenar as listas e antes de calcular a distância verificar se as listas tem a mesma quantidade de elementos. Para calcular a distância entre os pontos, foi utilizada a fórmula euclidiana que segue abaixo na Equação 6, onde p representa a quantidade de pontos, x representa os pontos anteriores, y os pontos atuais

$$DE(x,y) = \sqrt{\sum_i^p (x_i - y_i)^2} \qquad \text{(Equação 6)}$$

Visualizando a formula, pode-se concluir que distância euclidiana é o resultado da raiz quadrada do somatório do ponto anterior menos o atual elevado ao quadrado. Para exemplificar segue abaixo a Equação 7, com o calculo da distância entre os pontos P_1(1,1) e P_2(3,3).

$$DE(P_1,P_2) = \sqrt{(3-1)^2 + (3-1)^2} = \sqrt{8} \approx 2.8284 \qquad \text{(Equação 7)}$$

O cálculo da velocidade é muito simples, basta dividir a distância percorrida, pelo tempo percorrido, como este cálculo é feito frame a frame, temos que o tempo percorrido é igual a 0,033, pois o vídeo foi gravado a 30 quadros por segundo.

A distância percorrida foi calculada com a fórmula descrita acima, porém foi necessário converter a distância que estava em pixels para metros, para fazê-lo é necessário

ter a distância da câmera (já calibrada) com os objetos da imagem (carros e pedestres), a partir destas informações foi possível encontrar o valor de conversão de pixels em metros.

Com todos os dados necessários, a velocidade foi calculada, mas ainda não é a velocidade final, pois a velocidade considerada nas vias são medidas em quilômetros por hora e a resposta estava em metros por segundo, por fim para obter a velocidade basta converter de metros por segundo (m/s) para quilômetros por hora(km/h).

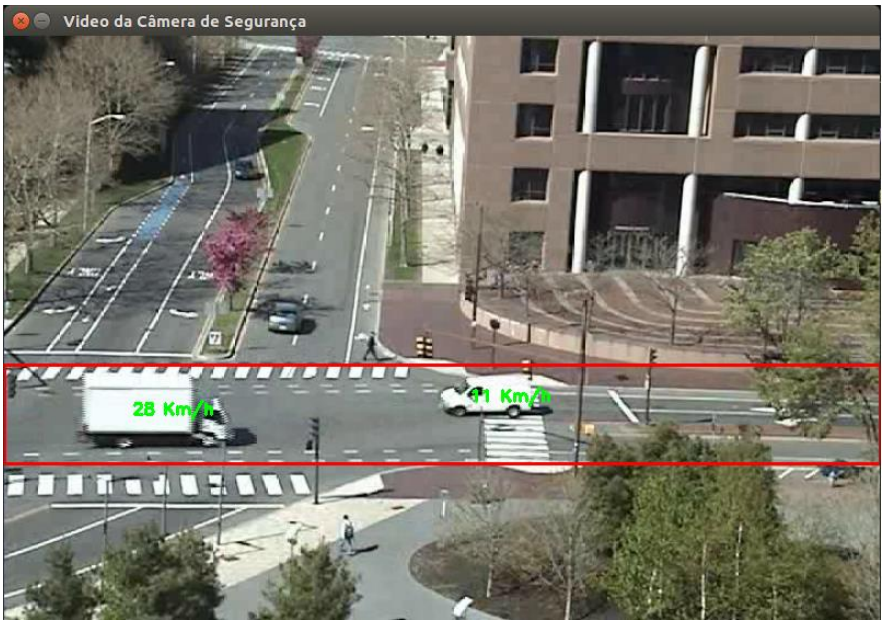
Para identificar os veículos trafegando acima da velocidade permitida, foi verificado se a velocidade calculada está acima da permitida, se estiver, a cor da velocidade mostrada na tela passará a ser vermelha, como mostra a Figura 20.

Figura 20 – Demonstração da velocidade excedida.

Fonte: Autoria Própria

O resultado final da aplicação é demonstrado pela Figura 21, onde a área que está sendo processada, ou área de interesse (RoI), é apresentada pelas linhas em vermelho, e a velocidade calculada é identificada com a cor verde e se o veículo estiver acima da velocidade permitida a velocidade passa a ter a cor vermelha como descrito na seção anterior.

Figura 21 – Demonstração da aplicação concluída.



Fonte: Autoria Própria

4. CONCLUSÃO

O objetivo deste trabalho foi o desenvolvimento de uma aplicação capaz de medir a velocidade com que os veículos se movimentam através do uso de câmeras de segurança convencionais e técnicas de visão computacional.

O desenvolvimento foi realizado utilizando a linguagem python que facilitou muito por ser uma linguagem de alto nível, de fácil entendimento e com muitos exemplos para consulta, principalmente utilizando a biblioteca OpenCV que é muito completa e também bastante utilizada, por esse motivo foi possível encontrar vários exemplos e explicações com relação as suas funções.

As dificuldades encontradas no trabalho foi conseguir identificar como um objeto único, um conjunto de pontos de interesse encontrados na imagem e obter vídeos de câmeras de segurança para o desenvolvimento da aplicação.

Portanto, conclui-se o trabalho com sucesso, pois os objetivos propostos foram alcançados e apresentados como base para trabalhos relacionados ou para o aperfeiçoamento deste trabalho.

Um diferencial do trabalho, é que a aplicação utiliza câmeras de segurança convencionais, por ter um valor baixo de aquisição, o projeto se torna viável de aplicação. Por realizar o monitoramento com imagens de câmeras de segurança convencionais também será possível realizar com câmeras mais sofisticadas.

5. TRABALHOS FUTUROS

Com o desenvolvimento desta aplicação e os estudos realizados, é possível que em trabalhos futuros possa ser possível realizar o monitoramento de veículos em perspectiva, ou seja, calcular a velocidade de veículos retirando a perspectiva para que se torne completo.

Para trabalhos futuros também poderá se pensar em monitorar o tráfego em tempo real e desenvolver uma interface para o usuário do sistema.

6. BIBLIOGRAFIA

BORBA, Victor Ubiracy. **Estudo Comparativo de Algoritmos de Extração de Fundo**. Monografia. Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2014.

BRITTO, Gabriel Ramires. **Desenvolvimento de algoritmo para "Traking" de veículos**. Monografia. Escola Politécnica da Universidade de São Paulo, Departamento de Engenharia de Sistemas Eletrônicos, São Paulo, 2011.

CHEUNG, S. C.; Kamath, C. **Robust Techniques for Background Subtraction in Urban Traffic Video**. Video Communications and Image Processing, SPIE Electronic Imaging, 2004.

CODEVILLA, Felipe Moraes. **Um estudo sobre detecção de pontos de interesse e classificação utilizando contexto**. Dissertação de Mestrado – Universidade Federal de Rio Grande, 2015.

CUNHA, A. L. B. N. **Sistema automático para obtenção de parâmetros do tráfego veicular a partir de imagens de vídeo usando OpenCV**. Tese de Doutorado. Escola de Engenharia de São Carlos, Universidade de São Paulo. São Carlos, 2013.

FERHAT, Bozkurt; METE, Yağanoğlu; FARUK, Baturalp Günay. **Effective Gaussian Blurring Process on Graphics Processing Unit with CUDA**. International Journal of Machine Learning and Computing, Vol. 5, No. 1, 2015.

FONSECA, Thiago Corrêa da Fonseca. **Processamento de Imagens para Detecção de Nódulos em Mamogramas Digitalizados**. Monografia. Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2007.

GALLON, Lucas. **Sistema de visão computacional para classificação de pedras naturais através de vídeos em tempo real**. Monografia. Centro de Tecnologia da Informação do Centro Universitário UNIVATES. Lajeado, 2013. Universidade de São Paulo. São Carlos, 2013.

GONZALEZ, R. C., WOODS, R. E. **Digital Image Processing**. New York: Addison-Wesley Publishing Company, 1992.

KALMAN, R. E. **On the general theory of control systems**. in Proc. First Internat. Congress Automat. Contr., Moscow, 1960.

MARR, D. (1982). Vision: **A Computational Investigation into the Human Representation and Processing of Visual Information**. New York: Freeman, 1982.

TRIVEDI, M.M., ROSENFELD, A., On making computers "See". **IEEE transactions on systems, man and cybernetics**, v.19, n.6, p.1333-6. 1989.

TUYTELAARS, Tinne; MIKOLAJCZYK, Krystian. **Local invariant feature detectors: a survey**. Foundations and Trends in Computer Graphics and Vision, Now Publishers Inc., v. 3, n. 3, p. 177–280, 2008.

SCHNEIDER, Marcos Roberto. **Sistema de segurança e proteção baseado em visão computacional**. Tese de Mestrado. Universidade Tecnológica Federal do Paraná. Curitiba, 2007.

WANG, X.; MA, X.; GRIMSON, E.**, Unsupervised Activity Perception in Crowded and Complicated scenes Using Hierarchical Bayesian Models**, *IEEE Transactions on Pattern Analysis and Machine Intelligence* (PAMI), Vol. 31, pp. 539-555, 2009.

7. APÊNDICE

```
#!/bin/python
#coding: utf-8

import numpy as np
import os
import cv2
from math import sqrt
from datetime import datetime

##### CONSTANTES ######
#Constante de fotos por segundo
cns_Fps = 0.033
#Constante da Velocidade Permitida
cns_VelPermitida = 30
#Constante para transformar pixels em metros
cns_PxToMts = 0.051
#Constante para transformar mts/s em km/h
cns_KmH = 3.6
#Constante que controla o delay entre uma foto e outra
cns_Delay = 120

#Atribui a imagem de fundo da subtração de fundo
cns_BaseSubFundo = 'documentos/video.jpeg'

#Atribui o video que vai ser processado
cns_Video = 'documentos/video.avi'

##### VAIRIÁVEIS ######
#Inicializa as variáveis que dimencionam a imagem dos veículos infratores
x1 = -10
x2 = -10
y1 = -10
y2 = -10

#Variável que controla o tempo restante para tirar outra foto
delay = 0
#Variável que controla á quantidade de fotos na tela
cont = 0
#Variável que controla a quantidade
veiculos = 0
#Variáveis que controlam a posição das imagens
top  = 590
left = 65

#Configura o caminho onde serão armazenadas as imagens
caminho = '../Veículos Infratores'

if (__name__ == '__main__'):

        #Recupera a data atual
        data = datetime.today().strftime("%Y-%m-%d")

        #Verifica se existe a pasta raiz
        if(not os.path.isdir(caminho)):
                #Cria a pasta raiz e a pasta com a data atual
                os.mkdir (caminho)
                os.mkdir (caminho+'/'+data)

        else:
                #Verifica se existe a pasta do dia atual
                if(not os.path.isdir(caminho+'/'+datetime.today().strftime("%Y-%m-%d"))):
                        #Cria a pasta do dia atual
                        os.mkdir (caminho+'/'+str(datetime.today().strftime("%Y-%m-%d")))

        #Atribui o caminho da pasta do dia atual para armazenar as imagens
        caminho = caminho+'/'+data

        #Inicializando as listas
        pontosAnteriores = []
        pontosAtuais     = []
```

```
	#Inicializa a velocidade em 0
	velKmh = 0

	#Informa o usuário que para sair da aplicação tem q digitar "ESC"
	print 'Digite "ESC" para fechar a aplicação'

	#Carrega a imagem base para a Subtração de Fundo
	fundo = cv2.imread(cns_BaseSubFundo);

	#Intância o vídeo que vai ser processado
	video = cv2.VideoCapture(cns_Video);

	#Verifica se ocorreu algum erro de leitura do vídeo
	if(not video.isOpened()):

		#Mostra a mensagem de erro de leitura do vídeo
		print "Atenção ocorreu um erro ao abrir o video!"

	#Percorre o laço infinitamente
	while(1):

		#Captura a imagem frame a frame
		ret,principal = video.read()

		#Verifica se o video chegou ao fim
		if ret == True:

			#Redimenciona a imagem
			principal = cv2.resize(principal, (800, int(principal.shape[0] * 800 /
principal.shape[1])), interpolation = cv2.INTER_AREA)

			#Define a área de interesse para o processamento
			frame = principal[300:390, 0:800]

			#Calcula a diferenca entre a imagem atual e a imagem de fundo
			diferenca = cv2.absdiff(frame, fundo)

			#Converte a imagem para escala de cinza
			imagemTomCinza = cv2.cvtColor(diferenca, cv2.COLOR_BGR2GRAY)

			#Filtra os ruídos da imagem
			imagemFiltrada = cv2.GaussianBlur(imagemTomCinza,(5,5),2)

			#Faz a binarização da imagem
			ret,imagemBinarizada = cv2.threshold(imagemFiltrada, 30, 255,
cv2.THRESH_BINARY)

			#Faz uma erosão apos uma dilatação para filtrar mais a imagem
			imagemDilatada = cv2.morphologyEx(imagemBinarizada, cv2.MORPH_DILATE,
(np.ones((10,10),np.uint8)))

			#Extrai os pontos de interesse
			pontoAtual = cv2.goodFeaturesToTrack(imagemDilatada, 200, 0.01, 30)

			#Busca o contorno da imagem binarizada
			contours, hierarchy = cv2.findContours(imagemDilatada,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

			#Limpa a lista de pontos atuais
			pontosAtuais = []

			#Percorre por todos os contornos
			for contorno in contours:

				#Calcula a area do contorno
				areaContorno = cv2.contourArea(contorno)

				#Verifica se a area do contorno é maior que 1000 para não pegar
os pedestres
				if areaContorno > 2000:

					#Extraindo as caracteristicas de cada contorno
					m = cv2.moments(contorno)

					#Extraindo o x e y da imagem2
					x = int (m['m10'] / m['m00'])
```

```
                                        y = int (m['m01'] / m['m00'])

                                        #Atribui "True" a variável que controla se vai incluir o
ponto na lista
                                        incluiLista = True

                                        #Verifica se tem valor em "x"
                                        if x:
                                                #Percorre a lista de pontos atuais
                                                for ptsAtuais in pontosAtuais:
                                                        #Verifica se o ponto já esta na lista
                                                        if ptsAtuais == [x,y]:
                                                                #Atribui "False" para não incluir o
ponto, pois já existe na lista
                                                                incluiLista = False

                                                #Verifica se vai incluir o ponto na lista
                                                if incluiLista:
                                                        #Inclui o ponto na lista
                                                        pontosAtuais.append([x,y])


                        #Verifica se tem a mesma quantidade de pontos nas duas listas para
calcular a velocidade
                        if (len(pontosAnteriores) == len(pontosAtuais)):

                                #Verifica se o delay ainda não chegou a 0
                                if(delay > 0):
                                        #Decrementa o valor do delay
                                        delay -= 1

                                #Ordena as duas listas para comparar os pontos proximos
                                pontosAtuais.sort()
                                pontosAnteriores.sort()

                                #Percorre os pontos anteriores para medir a velocidade
                                for i in range(len(pontosAnteriores)):

                                        #Calcula a distância euclidiana
                                        distPxs = sqrt((pontosAnteriores[i][0]-
pontosAtuais[i][0])**2) + ((pontosAnteriores[i][1]-pontosAtuais[i][1])**2)
                                        #Converte a distância em pixels em distância em metros
                                        distMts = distPxs * cns_PxToMts
                                        #Calcula a velocidade em metros por segundo
                                        velMts  = distMts / cns_Fps
                                        #Converte a velocidade de metros por segundo (mt/s) em
Kilometros por hora (km/h)
                                        velKmh  = velMts * cns_KmH

                                        #Verifica se a velocidade do veículo é maior que a
velocidade permitida
                                        if velKmh < cns_VelPermitida:
                                                #Printa a velocidade calculada de verde indicando
que a velocidade é permitida
                                                cv2.putText(frame, "%.0f Km/h" % velKmh,
(pontosAtuais[i][0] - 20, pontosAtuais[i][1]), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

                                                #Limpa as variáveis que controlam a posição onde
os veículos infratores estão
                                                if(delay == 0):
                                                        x1 = -10
                                                        x2 = -10
                                                        y1 = -10
                                                        y2 = -10

                                        else:
                                                #Printa a velocidade calculada de vermelho
indicando que está acima da velocidade permitida
                                                cv2.putText(frame, "%.0f Km/h" % velKmh,
(pontosAtuais[i][0] - 20, pontosAtuais[i][1]), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

                                                #Verifica se o delay chegou a 0 para recomeçar
tudo de novo
                                                if(delay == 0):

                                                        #Reinicia o valor do delay
```

```
                                                delay = cns_Delay

                                                #Verifica se não está fora da imagem
                                                if(((pontosAtuais[i][1]+200) < 390) and
((pontosAtuais[i][0]-80) > 0) and ((pontosAtuais[i][0]+80) < 800)):

                                                        #Verifica se existe 5 janelas de
imagens tiradas
                                                        if(cont == 5):
                                                               #Percorre o laço 5 vezes
para destruir as janelas
                                                               for j in range(5):

                                                                       #Destroi a janela

       cv2.destroyWindow('Veículo ' + str(veiculos-j))
                                                               #Reinicia as variáveis que
indica a posição e quantidade de janelas
                                                               left = 65
                                                               cont = 0

                                                        #Atribui as variáveis que controlam
a posição do veículo infrator
                                                        x1 = pontosAtuais[i][0]-80
                                                        y1 = pontosAtuais[i][0]+80
                                                        x2 = pontosAtuais[i][1]-200
                                                        y2 = pontosAtuais[i][1]+200

                                                        #Atribui na variável a imagem do
veículo infração
                                                        veiculoInfrator = frame[x2:y2,
x1:y1]
                                                        #Atribui na variável o nome da
janela e consequentemente da imagem também
                                                        nomeJanela      = 'Veículo ' +
str(veiculos+1)

                                                        #Mostra o veículo que está acima da
velocidade permitida
                                                        cv2.imshow(nomeJanela,
veiculoInfrator)
                                                        cv2.moveWindow(nomeJanela, left,
top)

                                                        #Armazena a imagem na pasta da data
atual
                                                        cv2.imwrite(caminho + '/' +
nomeJanela + ' - ' + datetime.now().strftime("%H:%M:%S") + '.jpeg', frame)

                                                        #Atualiza as variáveis de controle
das janelas
                                                        left     += 160
                                                        veiculos += 1
                                                        cont     += 1

                        #Atualiza o ponto anterior com o ponto atual para os calculos do próximo
frame
                        pontosAnteriores = pontosAtuais

                        #Identifica com um retangulo qual veículo foi tirado a foto
                        cv2.rectangle(principal, (x1,y2+40),(y1,y2+150), (0, 0, 255), 2)

                        #Desenha o retangulo vermelho na tela principal para identificar a área
de interesse
                        cv2.rectangle(principal, (0,300),(800,390), (0, 255, 0), 2)

                        #Mostra e posiciona a imagem Principal da aplicação
                        cv2.imshow('MONVIC - Video da Câmera de Segurança',principal)
                        cv2.moveWindow('MONVIC - Video da Câmera de Segurança', 0, 0)

                        #Recupera a tecla pressionada
                        tecla = cv2.waitKey(20) & 0xFF

                        #Verifica se a tecla ESC foi pressionada
                        if(tecla == 27):
                               #Sai do laco encerrando o processamento
```

```
                        break
        else:
                #Sai do laco encerrando o processamento
                break

#Destroi todas as janelas encerrando a aplicacao
cv2.destroyAllWindows()
#Informa que acabou o processamento
print "Processamento realizado com sucesso"
```