Escola Politécnica da Universidade de São Paulo

PSI3471 – Fundamentos de Sistemas Eletrônicos Inteligentes

EP1 - Identificador de Placas do tipo Proibido Virar

Autor: Lucas Penna Saraiva | Prof. Dr. Hae Yong Kim

1. Introdução e Objetivos

Nesse exercício programa da disciplina PSI3471 tivemos a oportunidade de desenvolver um software capaz de identificar placas de trânsito do tipo “proibido virar”. Em termos práticos, queremos dar uma imagem de entrada para o software (figura 1) e queremos um obter uma imagem de saída com um círculo em volta da placa, se ela existir (figura 2).

Figura 1 Imagem de entrada

Figura 2 Imagem de saída desejada, com o círculo verde identificando a placa

O objetivo do exercício foi, a princípio, a introdução ao campo da visão computacional, bem como suas possíveis aplicações. Em uma época em que se discute muito questões como veículos autônomos, robótica, reconhecimento facial aplicado à segurança, a visão computacional e o processamento de imagens emergem como um campo de estudo muito relevante para a solução de problemas contemporâneos. Por exemplo, com este identificador desenvolvido, poderíamos integrá-lo a um carro autônomo que, a partir da da leitura da placa

de proibido virar, tomaria a decisão de não virar naquela rua. Outras possíveis leituras de placas de trânsito poderiam também ser desenvolvidas, de forma a criar um arsenal de reconhecimento de placas para o possível veículo autônomo em questão.

2. Ambiente de desenvolvimento e técnicas utilizadas

Para desenvolver o software reconhecedor de placas de trânsito, utilizou-se o sistema operacional Windows e a linguagem de programação ***Python 3.7.6*** além de, essencialmente, as bibliotecas ***NumPy*** (para performar algumas tarefas simples em Álgebra Linear e manipulação de *arrays)* e ***OpenCV*** (para executar processamento de imagens de forma otimizada).

A metodologia de projeto baseou-se na seguinte premissa: entregar uma função que, dada uma imagem, retorna outra imagem com o círculo verde desenhado na região onde a função identificou a placa de proibido virar.

No módulo *img_transf.py* desenvolvido, definiu-se três funções:

- transformarEmCinza(img**)**

- get_list_img_resizes(template_img, MAX_SIZE=100, q=0.9, save_temp=False, N=16**)**

- **get_placa_proibido(**img, template, color=GREEN, threshold_placa=0.8, MAX_SIZE=150, q=0.95, save_temp=False, N=60**)**

2.1 A função *transformarEmCinza* recebe uma imagem e utiliza o método do OpenCV inRange para criar máscaras indicando a região da imagem que satisfaz a determinada região de cor desejada. Utilizamos, nesse método, o código de cor no formato HSV.

O modelo HSV significa *Hue*, *Saturation* e *Value.*

➔ Nesse espaço de cor, **Hue** significa cor, variando de 0° a 360°, conforme a figura 3. Podemos perceber que as faixas de ângulo determinam as seguintes regiões de cor: vermelho (0°-60°), amarelo (60°-120°), verde (120-180), ciano (180-240), azul (240-300) e magenta (300-360).

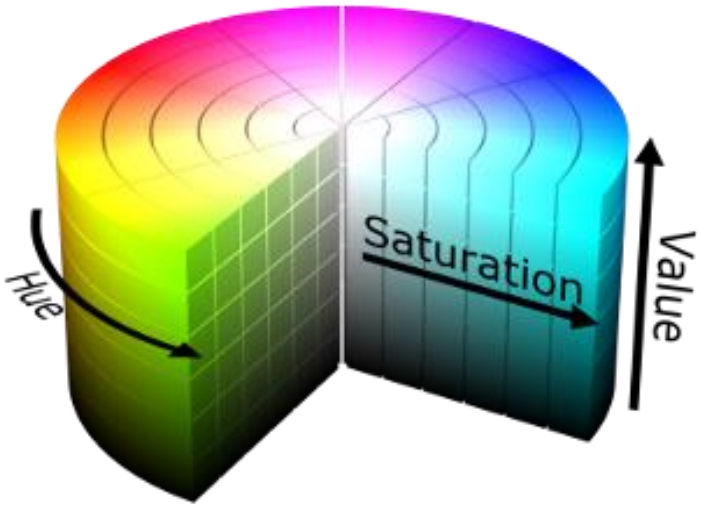

Figura 3: Espaço de cor HSV

➔ Já a **Saturação** indica o intervalo de cinza, conforme vê-se na figura 3. A Saturação 0 indica cinza e saturação 1 (máxima) indica a cor primária, pura, dentro daquele range de Hue especificado.

➔ Por fim, **Value** indica o brilho da cor. Ele varia de 0 a 100%. Quando 0%, a cor fica totalmente preta (pois não há brilho). Quando fica 100%, ela fica "pura", cor viva.

Para se ter uma cor vermelha típica e adequada nesse espaço, definiu-se o seguinte range de cores: o Hue nos intervalos de (0° a 20°) U (160° a 180°). Já a saturação adequada ficou entre (50 a 255) e o brilho (50 a 255).

Assim, criou-se máscaras que foram responsáveis por pintar de preto tudo o que estivesse fora desse range e pintar de branco tudo o que estivesse dentro desse range. Dessa forma, obtivemos a transformação da imagem em níveis de cinza.

2.2 A função *get_list_img_resizes* simplesmente pega a imagem template e retorna para o usuário uma lista contendo vários templates redimensionados. Essa função utiliza o método do OpenCV resize para redimensionar a imagem, fazendo isso dentro de um loop. As dimensões da imagem são diminuidas por um fator seguindo uma lei de progressão geométrica. Basta ao usuário definir o tamanho máximo de template que ele quer, bem como a razão dessa progressão geométrica. Para o devido funcionamento do programa, constatou-se que os parâmetros adequados podem ser: MAX_SIZE=300, q=0.98, N=130. Em que MAX_SIZE é o tamanho máximo, q é a razão da P.G. e N é o número de templates gerados.

2.3 A função get_placa_proibido simplesmente recebe uma imagem e retorna ao usuário a mesma imagem, só que com um círculo verde desenhado na região onde foi identificada a placa de proibido virar. Essa função implementa o seguinte pseudo-código:

```
define image as img
define template_image as temp
Img = transformarEmCinza(img)
list_temp = get_list_img_resize(temp)
for each temp in list_temp:
    img_correlation = matchTemplate(img)
    every_max_corr.append(img_corr.max())
    if max_correlation > threshold_placa:
            break loop;
n = argumento_maximo(every_max_corr)
img_correlation_definitiva = matchTemplate(img)
ponto_max_corr = getPontoMaxCorrelation( img_correlation )
img_output = desenharCirculoNoPonto( ponto_max_corr )
return img_output
```

Assim, a função recebe os seguintes inputs:

img: imagem que se quer identificar a placa

template: template da placa que se quer identificar

color: cor do círculo que se deseja desenhar ao redor da placa (default GREEN)

threshold_placa: limiar que indica a partir de quanto considerará a placa (default 0.8)

MAX_SIZE: tamanho máximo de templates a serem gerados. Deve ser alterado considerando o tamanho do diâmetro máximo das placas que se quer identificar (default 150)

q: fator geométrico de redução das imagens (default 150)

save_temp: se True, salva as imagens de template geradas (default False)

N: número de templates comparativos gerados (default 60)

No conjunto de imagens de treino, percebeu-se que os argumentos que melhor davam fit com o problema proposto foram:

```
get_placa_proibido(img, template, threshold_placa=0.4, MAX_SIZE=300, q=0.98, save_temp=False, N=130)
```

3. Como executar o programa?

Para executar o programa é muito simples, se estiver no Windows basta rodar:

```
python main.py CAMINHO_RELATIVO_DA_IMAGEM
```

Se estiver no Linux, será necessário especificar a versão do Python3 no terminal Bash:

```
$ python3 main.py CAMINHO_RELATIVO_DA_IMAGEM
```

Caso o usuário digite um caminho de imagem que não exista, o programa notificiará o usuário.

Exemplo de uso (Windows):

```
python main.py proibido_virar/09.jpg
```

4. Resultados obtidos

O programa conseguiu acertar praticamente todas as imagens fornecidas pelo professor na pasta proibido_virar, utilizando os parâmetros discutidos na seção 2.3. Já em relação ao tempo de processamento, ele varia bastante em relação a imagem. Para alguns casos, o programa levou 1.1s para identificar a placa. Já para outros casos, mais extremos, o programa elvou em torno de 25s. Na média, o programa demora em torno de 5s para retornar a imagem com a placa identificada. Nos testes, somente na imagem 25.jpg houve um resultado diferente do esperado: o programa foi capaz de detectar a plaquinha de proibido virar bem pequena lá do fundo ao invés de detectar a placa mais a frente.

Já a imagem 32.jpg, por ser bastante distorcida, exigiu uma “flexibilização” dos parâmetros de range de cor para que pudesse ser encontrada. Felizmente, o programa foi capaz de detectá-la conforme a figura abaixo:

5. Referências

- Notas de aula prof. Hae

- https://docs.opencv.org/2.4/

- https://docs.opencv.org/master/d4/dc6/tutorial_py_template_matching.html

- https://www.ginifab.com/feeds/pms/rgb_to_hsv_hsl.html