Deteção de linhas para condução autónoma com OpenCV

André Ribeiro Almeida 88960

Introdução

Condução autónoma é uma meta que quem aprecia tecnologia está bastante ansioso para ver. É um processo demorado e difícil, principalmente devido a razões de segurança dos "condutores". No entanto, neste trabalho é dada uma primeira visão de como a etapa de processamento de imagem é realizada. Este projeto é feito no âmbito da unidade curricular de Computação Visual do 4º ano do curso Mestrado Integrado em Engenharia de Computadores e Telemática.

Organização do trabalho

O processamento de imagem está dividido em múltiplas etapas de execução.
Em primeiro lugar, a imagem é transposta para escala de cinzentos visto que a cor não é particularmente importante para o objetivo em questão.
Seguidamente, é realizado uma operação de desfoque de modo a reduzir algum eventual ruído na imagem ou outra disrupção do tipo (como veremos em seguida, por exemplo o asfalto pode ser disruptivo na deteção das linhas).
Em terceiro lugar, é realizada uma operação de *Canny Edge Detection* para detetar as fronteiras das linhas da estrada com o pavimento. Isto facilita a realização da última etapa de processamento.
Para reduzir a imagem ao seu essencial, isto é, à própria estrada, é feito uma "máscara", cortando da imagem o resto que não é necessário.
Por último lugar, é realizado a operação disponível nas bibliotecas de opencv chamado transformada de Hough. Este passo deteta segmentos de reta na imagem, e é útil para a obtenção do ângulo para o carro seguir na direção certa.
Em cada passo, irá ser mostrada a imagem resultante. No processo da transformada de Hough, será desenhada, em cima da imagem origina, as linhas que este detetou, no local onde foram detetadas.
A imagem base onde estas operações serão realizadas é a seguinte:

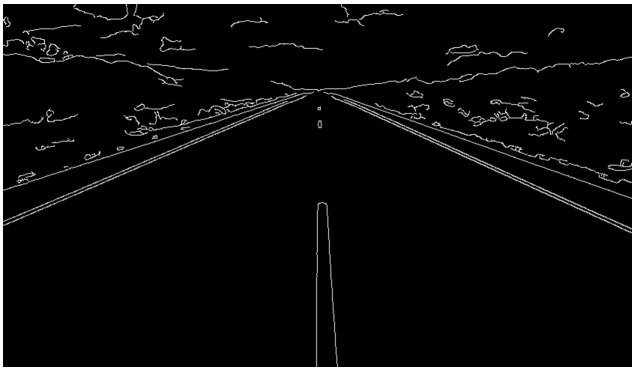
Pré-processamento

Grayscale

A imagem é primeiro convertida de rgb para escala de cinzentos, com o seguinte comando:

```
lane_image = cv2.imread(utils.get_abs_path(img_path))
gray = cv2.cvtColor(lane_image, cv2.COLOR_RGB2GRAY)
```

Resultando, então, na seguinte imagem:



Blurring

Para realizar a etapa *blurring*, temos várias opções:

- Average
- Gaussian
- Median

Para qualquer destes métodos, é fornecida a opção de ajustar a intensidade:

```
blur = (blur_level, blur_level)
blur = cv2.GaussianBlur(
    src=gray, ksize=blur, sigmaX=0
)
```

Em que *blur_level* é o tamanho do kernel de processamento que produz o desfoque: quanto maior, entre os possíveis valores, maior o desfoque.
O valor default foi definido a 5 porque, com verificações empíricas, mostrou-se eficaz para a maioria dos casos.

Canny Edge Detection

É realizado o procedimento *canny edge detection* para detetar as linhas na estrada, visto que estas têm grande contraste, são facilmente detetadas:

```
canny_image = cv2.Canny(
    image=blur, threshold1=thresholds[0],
threshold2=thresholds[1]
)
```

Os *threshold* default são 50 e 150 para o *threshold1* e *threshold2*, respetivamente no procedimento por histerese, embora outros valores próximos na ordem das dezenas são aceitáveis. É possível também ajustar estes valores se necessário.

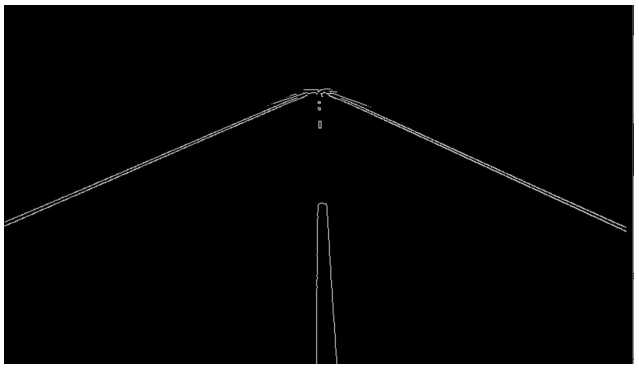
Masking

De forma a reduzir o espaço de análise da última etapa (Transformada de Hough) definiremos uma região de interesse da imagem para ser processada. Para tal, é definido um conjunto de pontos que formarão um polígono (neste caso, para simplificar, usar-se-á um triângulo) no entanto, outros polígonos poderão ser usados. Seguidamente, será realizado um masking da seguinte forma:

```
polygon = [(0, 0), (0, .4), (.5, .78), (.99, .4), (.99, 0)]
…
polygon = np.array(polygon, np.int32)
mask = np.zeros_like(image)
cv2.fillPoly(mask, [polygon], 255)
cropped_image = cv2.bitwise_and(image, mask)
```

Nota: os valores dos pontos do triângulo são definidos em fração dos limites da imagem ((0.5, 0.5) seria um ponto no centro
A "máscara" default é definida manualmente. É necessário testar a câmara do carro e ver a melhor configuração possível dependendo do ambiente.
Para o caso mais genérico, isto é, para definir uma máscara dinâmica conforme o ambiente é necessário um estudo mais aprofundado.
Foi gerada então, neste trabalho, esta máscara:

E com essa foi gerada esta imagem reduzida apenas aos seus pontos de interesse:



Mapping

Hough Transform

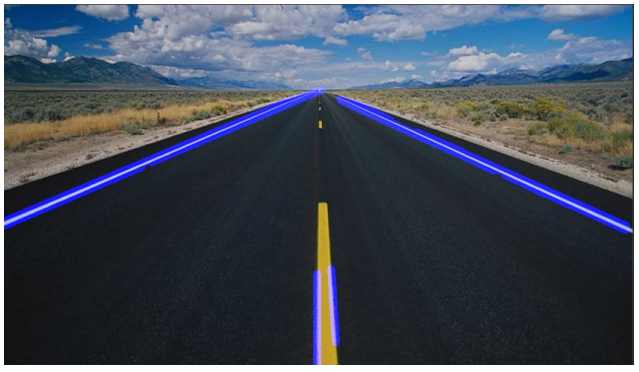
Para detetar as linhas é usado o método das transformadas de hough. Na condução autónoma, idealmente, serve para:

- Fazer o “desenho” das retas por estimativa das linhas da estrada;
- Realizar o cálculo do declive dessas retas;
- A partir do declive calcular o ângulo de viragem.

A partir das funções fornecidas do opencv, temos 2 opções:

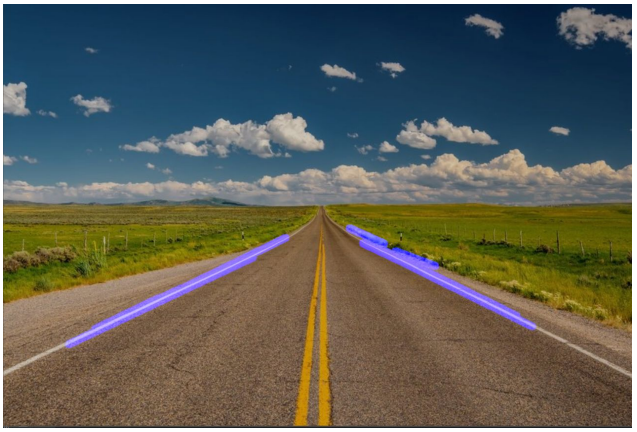
- usar a função simples de *hough*
- usar a função probabilística de hough

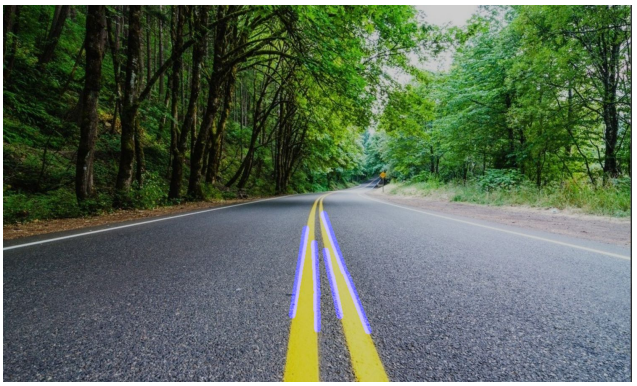
Apesar da primeira ser também útil em questão da condução autónoma, em que o que é necessário é apenas o declive das mesmas para obter o ângulo de viragem, foi escolhida a função probabilística visto que é mais eficiente em questões de esforço computacional.
Com esse método foi possível chegar ao seguinte resultado:



Em que, se for considerado necessário, é possível ajustar a máscara para detetar mais ou menos do que o que já está detetado da estrada.

Resultados



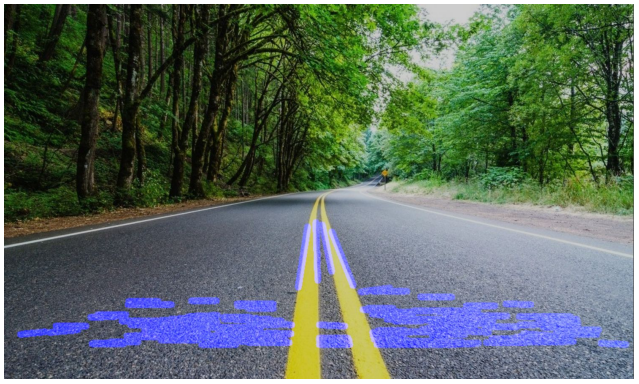


Notas finais

Face ao largo espetro de estradas possíveis, é difícil encontrar os parâmetros certos de processamento de imagem que sirvam para tudo. Face a isto, é fornecida uma configuração mais rápida e simples dos mesmos ao nível do código, de forma a ser mais rápido o estudo das combinações mais eficazes.

```
im_path = "road_photos/road7.jpg"
lane_detector(im_path,
              blur_level=5,
              blur_funct=cv2.GaussianBlur,
              thresholds=(85, 150),
              polygon=[(0, 0), (0, .4), (.5, .78), (.99, .4), (.99,
0)]
              )
```

Exemplificando, a imagem 3 dos resultados foi processada com blur_level de 7, visto que inicialmente a gravilha do pavimento da estrada levou a serem desenhadas linhas onde não deviam existir:



Um pequeno extra foi adicionado, que imprime na consola o valor do ângulo final que o carro deve virar, baseado nos declives das linhas.
Exemplificando, 0 graus significa seguir em frente, +90 significa virar à esquerda e -90 significa virar à direita.

Conclusão

Usando ferramentas de opencv para processamento de imagem, nomeadamente:

- *Grayscale*;
- Desfocamento (*blur*);
- *Canny edge detection*;
- Masking;
- Hough Transform,

é possível fazer a deteção de linhas de estrada para condução autónoma.
É necessária, no entando, alguma configuração manual dependendo do ambiente onde o veículo se encontra.
Ajustar os parâmetros do processamento de imagem pode ser realizado em trabalhos futuros.

Referências

1. https://www.learnpython.org/en/Multiple_Function_Arguments
2. https://medium.com/@yogeshojha/self-driving-cars-beginners-guide-to-computer-vision-finding-simple-lane-lines-using-python-a4977015e232
3. https://docs.opencv.org/master/d6/d10/tutorial_py_houghlines.html