Visão por Computador
On-road Vehicle Recognition and Tracking
Projeto BE - CEIIA

Diogo Corte, Diogo Silva
{diogo.corte, dbtds}@ua.pt

DETI, Universidade de Aveiro
Aveiro, Portugal

***Resumo* – Este artigo descreve um sistema desenvolvido para Visão por Computadores na Universidade de Aveiro. O objetivo principal é efetuar deteção de carros com recurso a técnicas de visão por computadores. Sendo que o artigo descreve detalhadamente a explicação de técnicas usadas e de decisões feitas para efetuar esta mesma deteção.**

I. DESCRIÇÃO DOS FICHEIROS

Durante este projeto foram desenvolvidos os seguintes ficheiros:

1. Pasta [haar_cascades] - Contém todo o treino que foi feito a volta das Haar Cascades;
2. Pasta [video_samples] - Contém exemplos de video usados, e uma pasta interior [camera] contém gravações feitas usando um telemóvel na parte interior de um carro;
3. DetectorHaarCascade.[h/cpp] - Contém um wrapper para o Haar Cascades (secção III-A);
4. DetectorMatchingFeatures.[h/cpp] - Contém um detector baseado em Matching de features (secção IV);
5. FilterFalsePositives.[h/cpp] - Contém um variedade de filtros falsos positivos usados;
6. Helper.[h/cpp] - Classe estática com funções auxiliares;
7. MultiTrackerOpenTLD.[h/cpp] - Multitracker para o Wrapper do OpenTLD;
8. MultiTrackerOpenCV.[h/cpp] - Wrapper para os trackers de OpenCV;
9. TrackerOpenTLD.[h/cpp] - Wrapper para o Tracker OpenTLD de uma biblioteca externa;
10. Ficheiros começados por letra minusculas contêm sempre uma função main associada sendo que são todos compilados pelo CMake.

II. PRÉ-PROCESSAMENTO

Antes de descrever qualquer algoritmo que tenha sido usado para a deteção, convém referir a existência de pré-processamento sobre a imagem de forma a facilitar a deteção.

A. *Equalização do histograma*

Por vezes a imagem captada por uma câmara tende a não estar bem dividida pelo na gama toda, sendo que existe muitas cores que ficam desvanecidas ou mesmo inexistentes. No caso de uma câmara exposta a uma luz muito forte. Por exemplo, um carro com uma luz forte em que a câmara capture essa luz vai causar com que a estrada pareça mais escura do que realmente é, tal como aparenta a imagem seguinte:

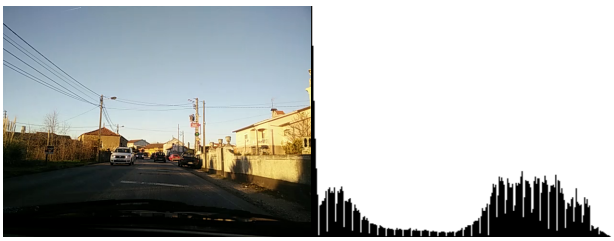

Figura 1: Imagem sem equalização e histograma

A ideia intensificar o contraste pelos pixeis das imagens, sendo que as cores da imagem vão ficar mais distintas, para isso aplicou-se equalização no histograma.

Comparando a imagem anterior (1) com a imagem seguinte (imagem 2) é possível verificar que a imagem ficou mais nítida, as cores estão mais distintas.

Figura 2: Imagem com equalização e histograma

É possível que determinados picos de intensidade no histograma foram espalhados por intensidades vizinhas ao longo da gama.

III. DETECTION

A. *Haar cascade*

A.1 Explicação teórica

Para o Haar Cascades validar se um dado objeto existe na imagem é feito através de várias fases, mais conhecidas por stages, estas stages são validadas de forma crescente, sendo que primeiro começam com pequenas validações mais genéricas e as ultimas já são mais especificas e detalhadas.

Sendo que se uma das stages falha, não é considerado como objeto detetado. Estas validações são pequenas features que são verificadas na área de interesse onde o Haar Cascades se encontra a avaliar [1].

Para melhorar o comportamento do mesmo ainda é feito variar o tamanho da área que é validada pelo HaarCascadas com detectMultiScale.

Também é possível visualizar no video seguinte o comportamento do algoritmo: `https://www.youtube.com/watch?v=nVbaNcRldmw`.

A.2 Abordagem prática

Existiam duas opções que se podiam ter seguido neste caso:

1. Treinar o próprio cascade e fazer classificação com este;
2. Usar um cascade já desenvolvido por fontes externas e fazer a classificação dos objetos com este cascade.

Pensou-se ser essencial fazer o treino de um cascade e usar este mesmo sendo que o objetivo seria treinar um cascade melhor do que existentes.

Para efectuar o treino seguiu-se o tutorial disponível pelo OpenCV [2].

A.3 Treino do Cascade - Obtenção de Dados

Tal como referido anteriormente, o Haar Cascades necessita de exemplos positivos do que se pretende detetar, em que esses exemplos devem conter pelo menos um objeto, neste caso, um carro ou mais. E também necessita de exemplos negativos.

Sendo que a estratégia inicial foi pegar num vídeo do YouTube que tivesse uma dashcam a gravar a estrada e fazer extração de partes de frames durante o vídeo todo. Para fazer um corte mais rápido do conteúdo do video usou-se **imageclipper** [5].

Mas devido a ser um processo demorado optou-se por usar a existência de datasets existentes. Neste caso usou-se um dataset [6] que contém partes de trás de carros e laterais, como também imagens negativas que é o pretendido.

A.4 Treino do Cascade - Geração do ficheiro de treino

As ferramentas de OpenCV para geração do ficheiro de treino exigem que exija um ficheiro que descreva as imagens positivas com a localização dos mesmos e outro ficheiro com uma lista das imagens negativas.

Sendo que se criou um pequeno script para fazer de forma automática (haar_cascades/trained/dataset_32/train.sh).

Após a criação destes ficheiros que descrevem o dataset, é só utilizar o opencv_traincascade fornecido pelo OpenCV. Sendo que exige uma configuração que se pretende efetuar para treinar o cascade (foram utilizadas sugestões para o treino de `https://abhishek4273.com/2014/03/16/traincascade-and-car-detection-using-opencv/`).

Foram treinados bastante cascades distintos com uma grande variedade de definições, sendo que se encontram todas em haar_cascades/trained_final. No entanto, o dataset nunca variou, foi sempre utilizado o mesmo dataset.

A.5 Treino do Cascade - Resultados Obtidos

A seguinte imagem apresenta um dos melhores cascades treinados:

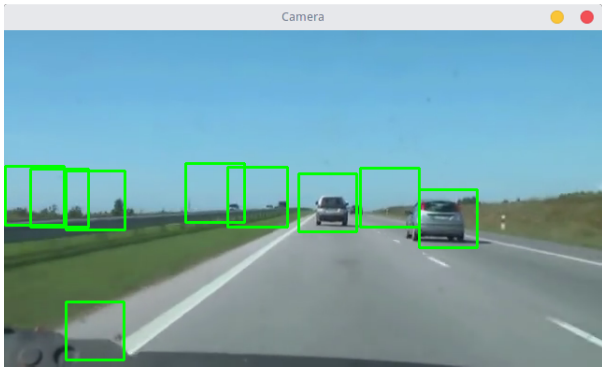

Figura 3: Haar cascades treinado

No entanto na maior parte das situações apresenta bastantes falsos positivos, sendo que determinadas situações não consegue fazer uma única deteção a um carro inteiro, mas sim a partes do carro.

O problema talvez se deva ao dataset conter imagens quase completas dos carros, mas não completas, sendo que a maior parte das imagens do dataset mostram cerca de 90% do conteúdo do carro. Talvez o corte das imagens tivesse de ser efetuado mais distante.

IV. DETEÇÃO COM LOCAL FEATURES ENTRE FRAMES (HISTÓRICO)

A ideia desta deteção consiste em encontrar um desfasamento entre frames, ou seja, o que obtiver movimento vai aparecer na imagem final, o que não obtiver, não aparece.

Esta ideia surgiu de tentar chegar a uma background subtraction para uma câmara em movimento, tendo em conta que não é possível aplicar background subtraction numa situação em que a câmara está em movimento, porque o background e o foreground estão ambos em movimento constante.

Sendo que o algoritmo consiste nos seguintes passos:

1. Extrair keypoints na frame atual e na frame antiga que está a ser comparada;
2. Calcular os descritores de cada keypoint;
3. Fazer a correspondência dos descritores relativos a cada keypoint de cada frame;
4. Usar findHomography para encontrar a matriz transformação que permite mapear os pontos antigos na frame atual;
5. Usar warpPerspective usando a matriz transformação e obtendo uma representação nova da frame antiga na perspetiva da frame recente.
6. Fazer a diferença entre a perspetiva e a frame actual para obter os desfasamento do que obteve movimento.

Na imagem seguinte é possível ver a correspondência, sendo que a da esquerda é a atual e a da direita é a atrasada.

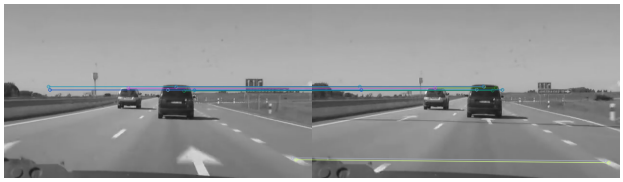

Figura 4: Correspondência de features entre a frame atrasada e a atual

Na imagem seguinte (imagem 5) pode-se ver um exemplo do resultado final.

Figura 5: Desfasamento das frames

O problema é que em cenários que tem casas ou se os keypoints forem mal mapeados, o imagem sai com uma distorção não suficientemente boa para aplicar um findContours sobre a imagem.

Um dos problemas que se deve ao desfasamento por vezes não ser o pretendido é por causa de o carro ser selecionado muitas vezes por keypoints, o que faz com que a perspetiva seja em relação ao carro em vez de ao cenário, sendo difícil de controlar desta forma.

Na figura 4 é possível ver um destes casos, existe 4 correspondências no carro e 5 no background, o que faz com que a perspetiva que aparece no desfasamento não seja a pretendida.

V. FILTRAGEM DOS FALSOS POSITIVOS - MEAN SQUARE ERROR FUNCTION

A. *Mean Square Error function*

Seja *F1* e *F2* duas frames com as mesma dimensão e *n* o total de pixeis para cada frame, o erro quadrático médio é dado por:

$$MSE = \frac{1}{n}\sum_{i=0}^{n}(F1[i] - F2[i])^2$$

B. *Abordagem prática*

Para o calculo do MSE, após dividida uma possível região de interesse nos respetivos recortes horizontais e verticais:

Figura 6: Recorte do lado esquerdo da região de interesse.

Figura 7: Recorte do lado direito da região de interesse com rotação sobre o eixo dos yy.

É possível verificar que após a simetria sobre o eixo dos yy na imagem, ao juntar os recortes verticais ou horizontais, uma metade do carro ficará bastante sobreposta com a outra metade.

Figura 8: Recorte do topo da região de interesse com rotação sobre o eixo dos xx.

Figura 9: Recorte do fundo da região de interesse.

Admitimos que todos os veículos em circulação encaixam no padrão de forte simetria horizontal e fraca simetria vertical como acima ilustrado, e comparamos os lados horizontais e verticais das regiões de interesse detetadas eliminando da lista de veículos instâncias com diferenças muito grandes horizontalmente e diferenças muito pequenas verticalmente.

VI. FILTRAGEM DE FALSOS POSITIVOS - ROAD COLOR

Esta filtragem consiste na expectativa de o carro que se está a tentar detetar, estar numa zona onde a estrada é igual (pelo menos parecida com a cor) onde o carro que contém a câmara está.

Por exemplo, parados em cima de relva têm uma forte probabilidade de ser eliminados. Porque a zona por baixo deles é verde enquanto que por baixo do carro da câmara o mais provável é ser cinzento.

Inicialmente tenta-se estimar a cor da estrada, para isso, extrai-se os pixeis mais próximos do carro e faz-se uma média das ultimas 50 frames desses pixeis, e será uma cor aproximada da estrada.

Para fazer a extração desta cor, converte-se a imagem em HSV porque é mais fácil de extrair a cor pelo canal H de Hue tal como apresenta a imagem 10.

Figura 10: Imagem convertida em HSV para distinção da estrada

Sendo que é possível verificar que a estrada tem uma cor mais roxa, casas uma cor amarelada, o céu uma cor vermelha. Aplicando um inRange tendo em conta a cor obtida pela extração dos pixeis próximos do carro obtém-se o seguinte resultado.

Figura 11: Segmentação da estrada relativamente ao resto

Após isto, é pegar nas imagens recebidas pelo detetor, que hão de conter os carros e o cenário a volta perto dos carros porque a deteção é uma zona quadrada e não um contorno perfeito a volta do carro. Retirando esses quadrados a uma imagem completamente preta obtêm-se a seguinte imagem 12.

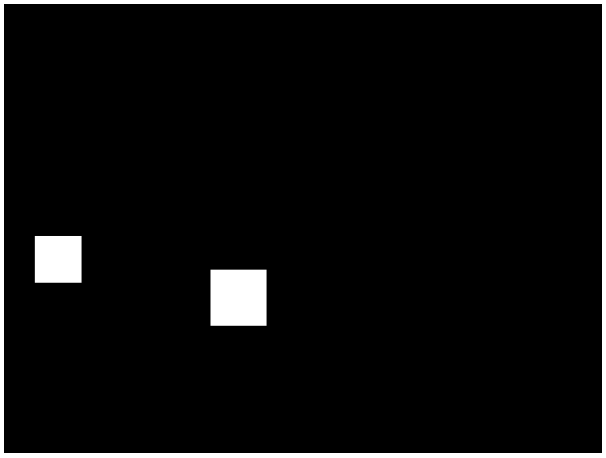

Figura 12: Imagem preta com os quadrados brancos das deteções

Se a imagem 12 for comparada com a imagem 10 é possível verificar que o quadrado branco mais a esquerda é um falso positivo. Para filtrar estes falsos positivos o que se faz é aplicar uma operação de AND sobre a imagem 12 com os quadrados brancos e a imagem 11 segmentada, obtendo assim uma imagem preta em que os quadrados brancos contêm porção da estrada, como mostra a figura 13.

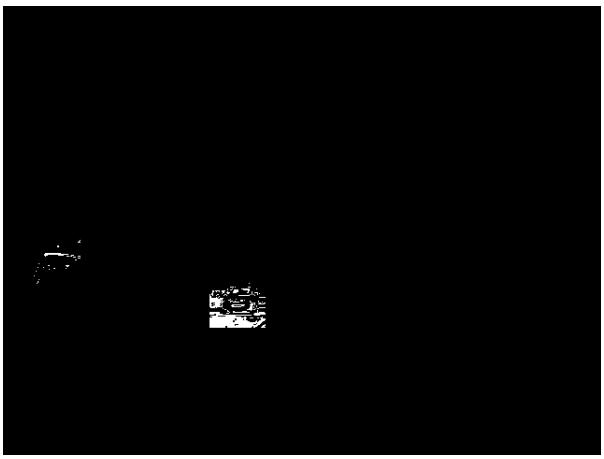

Figura 13: Operação bitwise AND sobre a imagem segmentada e as deteções

Agora basta voltar a verificar os quadrados e se contiveram abaixo de uma dada percentagem de branco, é porque são falsos positivos, significa que não se encontram na estrada. Se contiveram uma quantidade de brancos demasiado alta significa que é um falso positivo na estrada.

Esta solução funciona relativamente bem exceto em situações em que o sol está a incidir na estrada directamente. Sendo que altera complementa a cor que se encontra na estrada, por vezes também pode acontecer os murros terem uma cor bastante parecida com a estrada o que trás problemas.

VII. TRACKING

A. *OpenCV Tracker*

O OpenCV fornece uma ferramente que permite fazer o tracking de objectos, sendo que o objecto Tracker precisa de ser iniciado com uma box que se denomina de ROI (Region of Interest):

```
tracker->init(frame, roi);
```

Após o tracker ser inicializado a cada frame que se pretende localizar o objecto é necessário invocar o método seguinte:

```
tracker->update(frame, roi);
```

Sendo que o tracker recebe a frame e actualiza a posição da ROI, mantedo o sendo tamanho inalterado. O método update devolve true se o ROI não tiver sido modificado, o que permite evitar pós-processamento desnecessário.

B. *Diferentes tipos de Tracking usados*

Houve a necessidade de experimentar diferentes tipos de Tracking, sendo que os importantes para o trabalho são referidos abaixo. As caracteristicas importantes de um tracker são:

1. Não perder o tracking do objeto em questão;
2. Permitir redimensionar o objeto, sendo que o carro pode afastar-se ou aproximar-se;
3. Ser rápido o suficiente para funcionar em tempo real.

B.1 Tracker KFC

O tracker KFC é baseado em “color-names features” sendo que tem como foco ser rápido [7]. Este tracker permite fazer update a frame em questão de uma forma muito rápida sem nunca perder o carro, sendo que é um objeto minimamente visível após a sua deteção.
No entanto não contém um dos requisitos, que é os objetos poderem ser redimensionados. A medida que um carro se afasta ou se aproxima, o tracker mantém exactamente o tamanho da janela de interesse inicial.

B.2 Tracker MedianFlow

O tracker MedianFlow baseia-se em optical flow, no entanto isto pode levar a erros, desde erros devido a motion poder ser diferente do optical flow, tal como o exemplo habitual do barber pole.

Mas também pode dar outros erros devido a frame que foi detetada e que está a ser tracked poder conter mais do que objeto em questão, no caso deste trabalho, a frame detetada contém sempre um pouco de estrada e até pode vir a conter a faixa oposta que contém carros com o optical flow completamente oposto. Verifique-se a imagem 15.

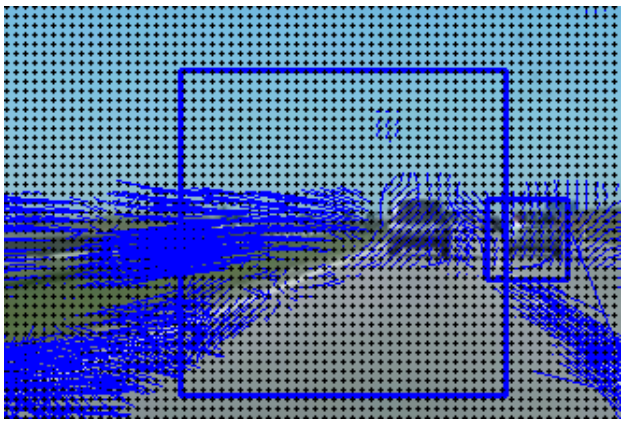

Figura 14: Optical flow durante o tracking

Nesta imagem o optical flow fez com que a janela de interesse fosse acumulando os erros relativos à mediana do optical flow da janela de interesse não ser igual ao motion do carro.

B.3 Tracker TLD

O tracker TLD, ’tracking, learning and detection’, tem em conta oclusões e também em redimensionar o objeto sempre que preciso. No entanto, é um algoritmo bastante lento, pelo menos o fornecido pelo OpenCV. O único tracker reduz as frames da deteção para 3-4 frames por segundo.

Para resolver este problema, usou-se `https://github.com/gnebehay/OpenTLD` que é um pouco mais rápido que o fornecido pelo OpenCV.

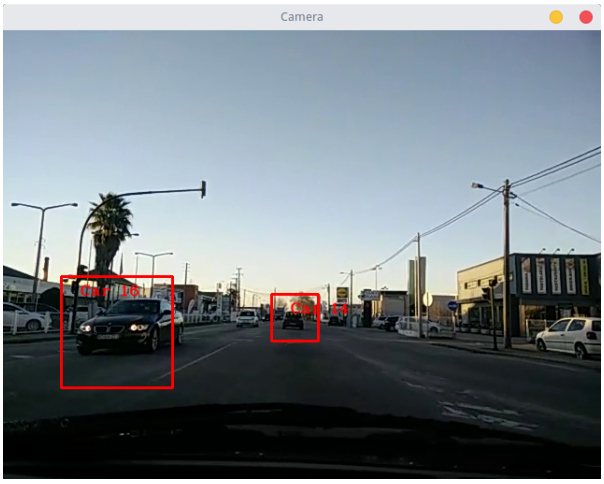

Figura 15: Tracking numerado usado o TrackerTLD

Sendo que se chega rapidamente a um multiplo tracker em que permite ter em conta vários carros, no entanto, começa a ficar igualmente lento como o desenvolvido pelo OpenCV.

O TLD também tem por base a mediana do Optical Flow, o que causa exactamente os mesmos problemas referidos acima (secção VII-B.2), ou seja, casos em que o optical flow é diferente do motion dos carros.

C. Abordagem prática com KCF

Dos três trackers referidos anteriormente optou-se pelo Tracker KCF, necessitando de fazer adaptações.

A deteção com filtro de falsos positivos apresenta resultados intermitentes, uma região de interesse pode ser detetada numa determinada frame e na seguinte deixar de o ser.

Para fazer face a este problema, e de forma a existir apenas Tracker por veículo, instânciamos um Tracker para cada nova região de interesse detetada, removendo as instâncias em que a região de interesse tivesse uma percentagem de área sobreposta de pelo menos 30% com a nova região de interesse detetada, sendo que nos casos em que excede o threshold de 30% consideramos que se trata do mesmo objecto.

Esta sobreposição ajuda o Tracker adaptar o tamanho, tendo em conta que o KCF mantém sempre o mesmo tamanho ao longo do tempo.

A cada nova frame removemos também os Trackers em que os limites da região de interesse excedem os limites da frame.

C.1 Resultados de Tracking

Na imagem seguinte (imagem 16) é possível verificar o tracking de 3 veículos:

Figura 16: Exemplo de Tracking das regiões de interesse detetadas

A imagem seguinte apresenta 3 veículos, sendo que um deles se encontra parcialmente fora de frame, pelo que apenas é feito o tracking dos 2 veículos restantes:

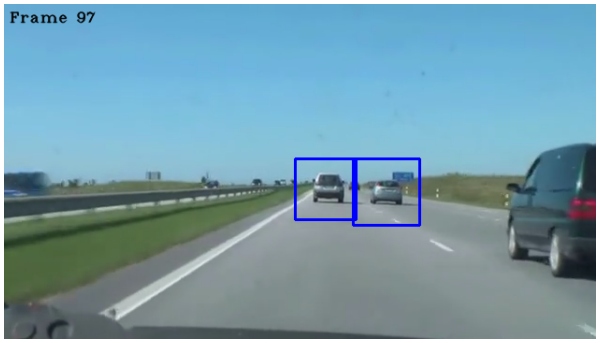

Figura 17: Exemplo de Tracking cancelado após exceder os limites da frame

Após fazer o tracking de cada carro facilmente se adiciona os respetivos labels a cada um, sendo possível fazer uma contagem, ou até mesmo estimar uma distância até ao carro.

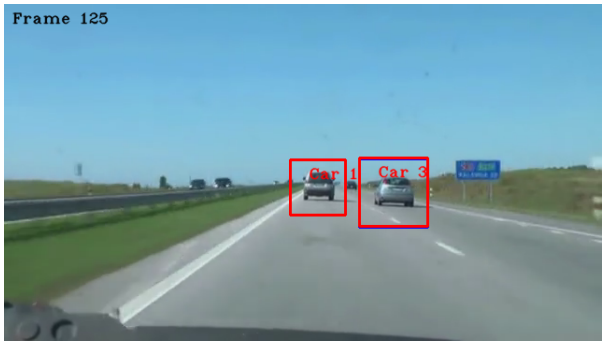

Figura 18: Labels em cada frame de tracking

VIII. CONCLUSÃO

Deteção e Tracking de carros usado uma câmara RGB são tarefas bastante desafiantes, sendo que foram efetuadas várias experiências de forma a obter os melhores resultados e mesmo assim continua-se a obter falsos positivos durante a deteção.

Variar as câmaras que são usadas para este tipo de tarefas implica mudanças a nível de parâmetros usados em cada função, tendo em conta que alguns parâmetros são otimizados para a câmara em questão.

O que talvez pudesse melhorar a deteção dos carros seria usar uma câmara com noção de profundidade, RGBD, ou até mesmo usar algoritmos relacionados com redes neuronais.

REFERÊNCIAS

[1] *Visual Recognition and Search Harshdeep Singh*.

[2] OpenCV, *Cascade Classifier Training*.

[3] OpenCV MSE Python Tutorial, *Car tracking with cascades*

[4] OpenCV, *Introduction to OpenCV Tracker*

[5] JoakimSoderberg, *Imageclipper fork*

[6] Grupo de Tratamiento de Imágenes (GTI), E.T.S.Ing. Telecomunicación, *Vehicle Image Database*

[7] OpenCV, *KFC Tracking*