Detecção de Retas e Círculos com OpenCV

Princípios de Visão Computacional - UnB - 2º/2013
Professor : Flávio Vidal
Aluno : Juarez Aires Sampaio Filho - 11/0032829

I. Objetivos

Desenvolver um código para detectar linhas e círculos em vídeos e em imagens.

II. Introdução

EM VISÃO COMPUTACIONAL a detecção de formas geométricas simples é utilizada no processamento de objetos mais complexos. A detecção de formas é útil quando conhecemos previamente a forma do objeto e em muitas aplicações isso é possível. A detecção de prédios e carros, por exemplo, pode ser feita facilmente se usarmos o fato de que suas arestas são retas e procurarmos por linhas na imagem.

O primeiro passa na detecção de linhas é a detecção de bordas. Isso pode ser feito usando o algoritmo de **Canny**. Esse algoritmo usa a variação da intensidade medida por meio de aproximações para o **gradiente** da função intensidade para determinar pontos de fronteiras entre duas superfícies. Após esse processo a **transformada Hough** é aplicada para detectar linhas. Uma versão melhorada dessa transformada pode também ser aplicada para detectar círculos.

Todas essas funções citadas já estão implementadas no OpenCV, de modo que aplicações que envolvem detecção de retas e círculos podem ser rapidamente desenvolvidas. Descrevemos brevemente as funcionalidade de tais funções;

- **Canny**(src, detected_edges, lowThreshold, lowThreshold*ratio, kernel_size) :
  salva em detected_edges as quinas detectadas em src. Nessa imagem pontos detectados como quinas estão marcados com branco e o resto é preto. Os outros argumentos são parâmetros que determinam a funcionalidade da função. De particular interesse lowThreshold determina a sensibilidade da rotina.Valores baixos fazem o método muito sensível à ruídos e valores elevados impossibilitam a detecção.
- **HoughLines**(dst, lines, rho, thetea, threshold, srn, stn) : lines é uma coleção de vetores que guarda os coeficientes das retas encontradas. Novamente threshold determina a sensibilidade do algoritmo.
- **HoughCircles**(src_gray, circles, CV_HOUGH_GRADIENT, dp, min_dist,param_1, threshold, min_r, max_r): semelhante ao anterior, mas para detecção de círculos.

III. Materiais

O código elaborado foi feito em C++ utilizando as bibliotecas:

- imgproc
- highgui

do **OpenCV** versão 2.4.6.

IV. Procedimentos

A. Detectando Formas

- linhas
  1) aplicamos o método Canny
  2) aplicamos o método HoughLines
- círculos
  1) convertemos para grayscale
  2) aplicamos um filtro gaussiano para diminuir ruídos
  3) aplicamos o método HoughCircles

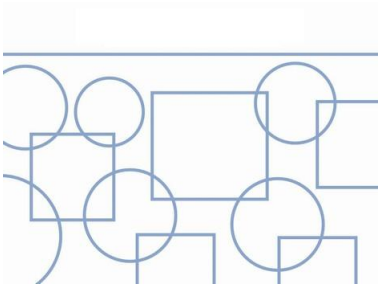
Agora é só desenhar as formas encontradas na figura com as funções **line** e **circle** do OpenCV. Para facilitar o processo de calibração dos parâmetros acrescentamos ainda trackbars para podermos variar os parâmetros em tempo de execução.

O código é feito de forma a aceitar entradas a partir de uma imagem ou vídeo em disco ou apartir de uma webcam instalada via usb.

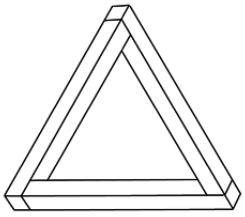
B. Testando o Código

As imagens 1a e 1b são entradas no programa e procuram-se os parâmetros que permitem uma boa detecção. O programa também é testado com um vídeo de um modelo de helicóptero voando sobre ambiente controlado e chão quadriculado. Um frame do vídeo é mostrado na figura 1c.

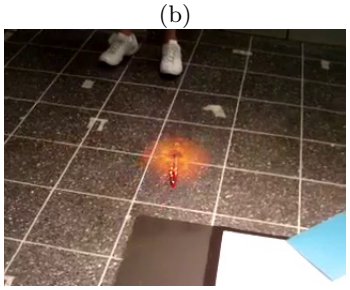
As imagens 1a e 1a são impressas e coladas em superfície rígida. Elas são então mostradas para a webcam e compara-se o resultado da detecção com o do processo inicial. A comparação é feita medindo-se o número de linhas detectadas nos dois casos. O processo de tomada de medidas no procedimento com a câmera é feito 10 vezes utiliza-se a média dos valores obtidos para efeito de comparação.



(a)

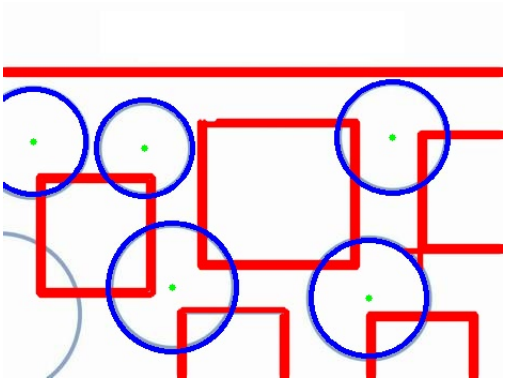


(b)

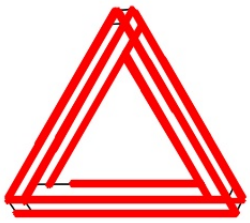


(c)

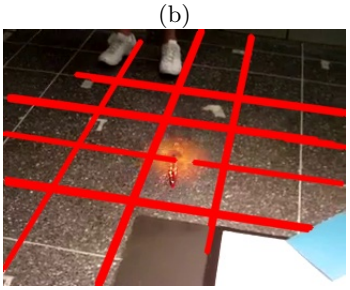
Figura 1: Imagens de teste do algoritmo



(a)



(b)



(c)

Figura 2: resultados obtidos

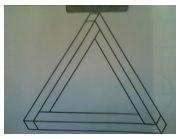
V. Resultados

Os resultados obtidos são mostrados a seguir nas figuras 1a, 1b e 1c. As duas primeiras figuras foram obtidas com parâmetro de thereshold mínimo para o algoritmo linear de 42, enquanto o parâmetro para a terceira foi de 145. A calibração foi feita manualmente por meio de trackbars até se atingir um resultado razoável.

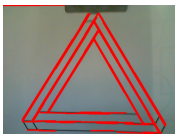
No teste de comparação obtivemos a seguinte tabela:

medida	baseado em imagem	em vídeo
média de linhas em 1b	24	62.9
média de círculos em 1b	0	0
média de linhas em 1a	35	24.4
média de círculos em 1a	5	3.9

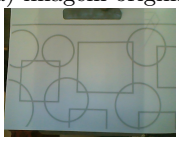
As figuras em 3 ilustram a detecção com câmera.



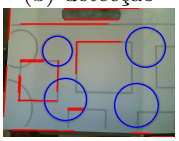
(a) imagem original



(b) detecção



(c) imagem original



(d) detecção

Figura 3: detectando linhas e círculos com câmera

VI. Discussão

Nos resultados da figura 2 vemos a eficiência e as limitações da detecção.

- Em 2a vemos que:
  1) o círculo no canto esquerdo superior que está cortado foi detectado.
  2) o círculo cortado na parte inferior não foi detectado
  3) todos os círculos inteiros foram detectados.

  Temos uma boa garantia portanto que círculos bem definidos serão detectados, já quanto a completação de círculos parcialmente ocultos não podemos ter certeza
- Em 2b vemos que:
  1) linhas compridas e bem definidas foram detectadas
  2) pequenas linhas não foram detectadas

  Vemos que o algoritmo trabalha bem com a detecção de linhas bem definidas mas que tem problemas com pequenas linhas. Provavelmente isso se deve ao fato de que o algoritmo Canny aplica um filtro para reduzir ruídos na imagem. Dessa forma pequenas linhas são identificadas como ruído.
- Em 2c vemos:
  1) novamente temos linhas que não foram detectadas, mas que as todas as linhas mais bem definidas foram

  Assim como nos anteriores, notamos que o **algoritmo não é ideal mas que lida bem com a maioria dos casos**. Em uma aplicação de localização de helicóptero por visão seria necessário reforçar os traço das linhas no chão e controlar a iluminação para melhores resultados. Vale ressaltar que poderíamos ter diminuído o threshold para o algoritmo. Nesse caso teríamos detectados mais linhas, mas teríamos também maior ruído. É necessário um processo de calibração desse parâmetro para cada aplicação.
- Notamos uma queda apreciável na qualidade da detecção quando as mesmas imagens são apresentadas a uma câmera e o processamento é feito em cima de um vídeo. Podemos citar algumas fontes de erro:
  1) **a iluminação ambiente** pode afetar as diferentes partes do vídeo diferentemente, de forma que as relações de gradiente se alteram. Se o gradiente diminui em um ponto pode-se ter que este, que era antes detectado como quina(e portanto tinha chance se ser uma reta), passa a ser detectado como ruído e não é processado.
  2) **o formato de compactação de vídeo** muda as propriedades da imagem de moda que podemos ter o mesmo efeito da iluminação.
  3) se apresentarmos o padrão de forma inclinada, teremos os **círculos sendo projetados como elipses** e não sendo mais detectados.
  4) imperfeições no padrão de cor obtido podem fazer com que **o que era interpretado como uma única reta seja quebrado em várias**. É isso o que acontece em 2b, onde temos um aumento no número de retas detectadas.
- é notável que os parâmetros de threshold para as imagens utilizadas e para o vídeo do modelo de helicóptero são muito distintos. Ressalta-se aqui a **importância do processo de calibração** nas aplicações com processamento de imagem. Um mesmo algoritmo pode precisar de parâmetros bem distintos dependendo da aplicação que se tem em mente.

Notamos nos resultados características comuns no trabalho com visão computacional: os algoritmos são sensíveis às condições de iluminação e os parâmetros são extremamente específicos para cada aplicação. Apesar dessas dificuldades presentes nota-se que resultados razoáveis podem ser obtidos rapidamente. Apesar de detecção não ser perfeita, o modelo de helicóptero do vídeo testado poderia utilizar a detecção de linhas no chão para se posicionar em um sistema de coordenadas apropriado.

VII. Conclusão

Foi possível utilizar as bibliotecas do OpenCV para desenvolver rapidamente um código capaz de detectar linhas e círculos em imagens e em vídeos. Os resultados obtidos mostram que os algoritmos utilizados não são perfeitos mas que funcionam adequadamente quando os objetos a serem detectados estão bem definidos. Notou-se ainda a importância da calibração dos parâmetros dos métodos para cada aplicação e a influência de condições internas e externas no processamento de vídeos obtidos por câmeras. O projeto reforça a ideia de que algoritmos de visão computacional são dificilmente robustos às condições do meio ambiente e que melhores resultados serão atingidos se for possível controlar parâmetros de cena como a iluminação.

Referências

[1] Forsyth, D.A. , *Computer Vision: a Modern Approach*, 1ªed.

[2] Documentação do OpenCV, *Tutoriais em Processamento de Imagem:Canny Edge Detector, Hough Line Transform e Hough Circle Transform* Disponível em: http://docs.opencv.org/doc/tutorials/imgproc/table_of_content_imgproc/table_of_content_imgproc.html#table-of-content-imgproc Acesso em 14 de Outubro de 2013.