OpenCV Rastreando Objetos

Paulo Renato Kinjo
Thiago da Silva Soares

Curso de Ciência da Computação - Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS) Av. Ipiranga, 6681 - Partenon - Porto Alegre/RS - CEP: 90619-900

(paulo.kinjo, thiago.soares.001)@acad.pucrs.br

Resumo. OpenCV (*Open Source Computer Vision Library*) é uma biblioteca open-source que inclui centenas de algoritmos de visão computacional[2]. O objetivo principal deste artigo é detectar objetos de uma cor especifica, podendo ser um vídeo pré-gravado ou mesmo o vídeo de uma câmera em tempo real. Após detectar o objeto, criaremos um algoritmo para rastrear este objeto na medida em que ele se move.

Palavras-chaves: OpenCV, algoritmos, rastreamento de objetos.

OpenCV Object Tracking

Abstract. OpenCV (*Open Source Computer Vision Library*) is an open source library that includes hundreds of computer vision algorithms. The main objective of the article is to detect objects of a specific color, which can be a pre-recorded video or video from a webcam in real time. After object detection, algorithm will tracking the object as it moves.

Keywords: OpenCV, algorithm, object tracking.

1 Introdução

OpenCV (*Open Source Computer Vision Library*) é uma biblioteca open-source que inclui centenas de algoritmos de visão computacional. Além disso, essa biblioteca tem uma estrutura modular, o que significa que ela inclui várias bibliotecas compartilhadas ou estáticas.

O que faremos aqui é utilizar os recurso da biblioteca OpenCV, para rastrear a localização de um objeto colorido em uma imagem. Em nosso caso, será uma bola amarela. Observe a figura 1.1 a seguir:

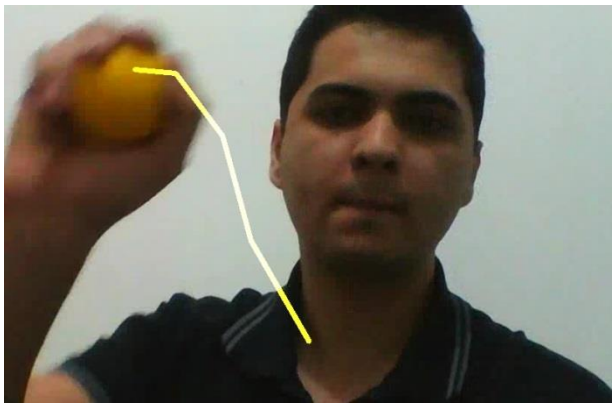

Figura 1.1: Exemplo de uma bola amarela sendo rastreada.

2 Capturando Imagens

O processo de captura de uma imagem usando o OpenCV é realizado utilizando a estrutura CvCapture que está definida dentro dos cabeçalhos da biblioteca. Os métodos desta estrutura podem ser utilizados, tanto para a captura de imagem de um vídeo pré-gravado, quanto para a captura de vídeo de uma câmera. Depois que a estrutura é criada podemos solicitar que o frame atual do vídeo seja capturado. Observe a linha de código a seguir:

```
CvCapture* captura = cvCaptureFromAVI("vídeos/video2.avi");
   IplImage* frame_atual = cvQueryFrame(captura);
```

A primeira linha de código carrega o vídeo armazenado no caminho de diretório passado como parâmetro, em seguida outra estrutura fornecida pela biblioteca é criada para capturar o frame corrente do vídeo em questão.

3 Cor

Para desenvolver este programa, optamos por usar o espaço de cor HSV, em vez de o espaço mais comum de cor RGB. Em HSV, para cada "tom" de cor é atribuído um número específico (Tonalidade). Para a "quantidade" de cor é atribuída outro número (saturação) e para o brilho da cor é atribuído outro número (intensidade). Isso nos dá a vantagem de ter um único número de tonalidade para a bola amarela, apesar de possuir vários tons de amarelo (todo o caminho de amarelo escuro a um amarelo brilhante)[3]. OpenCV também vem com uma função que converte imagens entre os espaços de cores. Observe o trecho de código a seguir:

```
cvCvtColor (origem, destino, codigo_conversor);
```

Aqui, o parâmetro origem é a imagem a ser transformada. O parâmetro destino é o local onde a imagem transformada será armazenada e o parâmetro código_conversor especifica o tipo de conversão que será realizado, no nosso caso, o código que será utilizado é o CV_BGR2HSV, por que é como o OpenCV carrega a maioria das imagens, ou seja, ele carrega em BGR e não em RGB[3].

4 Desenvolvimento do Projeto

Para desenvolver o projeto seguimos três passos: carregar a imagem da câmera ou vídeo, descobrir onde a bola amarela se encontra na imagem do frame corrente e adicionar a posição corrente da bola em variáveis de posições.

Para carregar a imagem da câmera ou vídeo, utilizaremos o procedimento apresentado no item 2 deste artigo.

Para descobrir a onde a bola está, primeiro vamos usar uma função de threshold[2] e então estimar a posição da bola no frame corrente.

Para rastrear a trajetória da bola, iremos usar outra imagem, que irá armazenar e rabiscar a imagem sempre que a bola se movimentar.

4.1 Função Threshold

Para começar vamos desenvolver a função de threshold. Observe o código a seguir:

```
IplImage* threshold(IplImage*)
```

O primeiro procedimento realizado dentro desta função é, converter a imagem recebida no formato BGR (lembrando que a biblioteca OpenCV lê imagens nesta ordem[3]) para HSV. Observe o seguinte código:

```
IplImage* imagemHSV;
imagemHSV = cvCreateImage(cvGetSize(imagem), 8, 3);
cvCvtColor(imagem, imagemHSV, CV_BGR2HSV);
IplImage* imagem_thresh = cvCreateImage(cvGetSize(imagem), 8, 1);
```

Primeiro definimos uma estrutura para armazenar a imagem convertida. Segundo criamos uma cópia da imagem passada como parâmetro, para preservarmos a imagem original para usos futuros. Terceiro efetuamos o processo de conversão propriamente dito e por ultimo definimos e criamos uma estrutura para armazenar a imagem que receberá o thresh[2]. Observe que a ultima estrutura definida utiliza a imagem original e não a imagem no formato HSV. Em seguida realizamos o threshold. Observe o código a seguir:

```
cvInRangeS(imagemHSV, cvScalar(10, 190, 80), cvScalar(90, 255, 255),
                              imagem_thresh);
```

Os parâmetros cvScalar[1], são funções de cor que representam a menor e a maior intensidade da cor. Com isso terminamos a função threshold, e podemos observar que para rastrearmos uma bola de outra cor, mudaríamos apenas os valores da função cvScalar.

4.2 Função main

O primeiro passo na função main é capturar o arquivo de vídeo ou acessar a câmera. Em seguida devemos testar se a estrutura conseguiu realizar a captura, por que se não nós finalizamos o programa.

Em seguida criamos uma janela que, irá exibir o vídeo na tela. Criamos a estrutura que ira armazenar a imagem rabiscada, um contador que tem a função de ajudar o programa a apagar o rastro depois de um determinado tempo para que a tela não fique toda rabiscada e uma variável que ira controlar a largura da linha.

Como estamos realizando a leitura de um vídeo, devemos utilizar um laço de repetição que ira capturar todos os frames que serão tradados separadamente. No nosso caso irá desenhar uma trajetória da bola caso ela se movimente, e então apresentará para o usuário a imagem tratada. Observe o código a seguir para um melhor entendimento:

```
while(true) {
	IplImage* frame = 0;
	frame = cvQueryFrame(captura);
		if(!frame) {break;}
	if(imagemRiscada == NULL){
		imagemRiscada = cvCreateImage(cvGetSize(frame), 8, 3);
	}
	IplImage* imagem_tresh = threshold(frame);
```

Primeiro definimos que o laço vai ser infinito, em seguida criamos uma estrutura para armazenar a captura que iremos fazer de cada frame do vídeo. Se não for mais possível capturar frames do vídeo que carregamos para a estrutura de captura, então terminamos o

laço de repetição e o programa segue o seu fluxo. Caso a captura do frame seja um sucesso, verificamos se a estrutura que irá armazenar a imagem riscada já foi inicializada, em caso negativo, pegamos a imagem capturada do frame corrente e atribuímos a esta estrutura para que o processo de desenho da trajetória da bola seja realizado. Observe que caso a imagem já esteja inicializada, o programa pula a estrutura condicional e inicia o processo de tratamento da imagem. Por fim criamos uma estrutura que armazenará a imagem do frame corrente, porém depois de aplicarmos a função de threshold (veja o item 4.1).

A biblioteca OpenCV possui uma estrutura de analise e descrição de formas que calcula todos os momentos até a terceira ordem de um polígono[1]. Vamos utilizar está estrutura para capturar o momento e estimar a posição da bola. Observe o código a seguir:

```
CvMoments *momentos = (CvMoments*)malloc(sizeof(CvMoments));
            cvMoments(imagem_tresh, momentos, 1);
```

Primeiro nós alocamos memória para a estrutura de momentos, calculamos os vários momentos. Em seguida, usando a estrutura de momentos, nós calculamos os momentos de primeira ordem e de segunda ordem. Observe o código a seguir:

```
double primeira_ordem_x   = cvGetSpatialMoment(momentos, 1, 0);
double primeira_ordem_y   = cvGetSpatialMoment(momentos, 0, 1);
double segunda_ordem      = cvGetCentralMoment(momentos, 0, 0);
```

As duas primeiras variáveis são de primeira ordem, e a ultima é de segunda ordem, que irá representar a área. A variável primeira_ordem_x dividida pela variável segunda_ordem resultará no valor da coordenada X da bola. E o resultado da divisão da variável primeira_ordem_y pela variável segunda_ordem, será a coordenada Y.

Agora precisamos capturar a posição das coordenadas x e y, e então armazenar em variáveis estáticas, por que os valores destas variáveis devem ser preservados para que possamos rastrear a posição da bola de forma continua sem perder as coordenadas. Observe o código a seguir:

```
static int pos_x = 0;
static int pos_y = 0;
int ultimo_x = pos_x;
int ultimo_y = pos_y;
pos_x = primeira_ordem_x  / segunda_ordem;
pos_y = primeira_ordem_y  / segunda_ordem;
```

Na primeira passagem do laço definimos as posições x e y como zero, e então criamos variáveis para armazenar temporariamente estes valores, para em seguida realizarmos o calculo da posição atual da bola. Observe que fazemos isto todas às vezes que percorremos o laço, ou seja, armazenamos temporariamente a ultima posição em que a bola se encontrava, com a leitura do frame atual calculamos as novas coordenadas da bola, e atribuímos às posições atuais de x e y que então, em uma próxima alteração serão atribuídas as variáveis temporárias novamente.

Finalmente nós desenhamos o rastro da bola utilizando a seguinte condição para realizar o desenho da linha.

```
if(ultimo_x > 0 && ultimo_y > 0 && pos_x > 0 && pos_y > 0) {
        cvLine(imagemRiscada,
        cvPoint(pos_x, pos_y),
        cvPoint(ultimo_x, ultimo_y),
```

```
                    cvScalar( 40, 255, 255), largura_linha);
            }
```

É fácil observar que o desenho da linha é realizado, pegando a posição anterior e fazendo o risco até a posição atual. Observe a função cvScalar que define a largura do rastro em pixel.

Agora temos que juntar a imagem a qual nós desenhamos a linha com a imagem do frame atual, para que a união das duas seja a imagem do frame atual com a trajetória da bola. Observe o código a seguir:

```
IplImage *nova_imagem = cvCreateImage(cvGetSize(frame_atual), 8, 3);
          cvAdd(frame_atual, imagem_riscada, nova_imagem);
```

Com isso, apenas vamos chamar a função cvShowImage e passar a imagem tratada para ser apresentada na tela que criamos anteriormente. A estrutura criada e definida como nova_imagem é utilizada com o propósito de preservar a imagem original do frame atual, observe que a função cvAdd junta as estruturas frame_atual e imagem_riscada, e adiciona esse resultado na estrutura nova_imagem. O código seguinte fundamente o motivo pelo qual o frame atual foi preservado:

```
if(contador % 14 == 0 ) {
     cvShowImage("video", frame_atual);
     imagem_riscada = NULL;
     largura_linha = 1;
     muda_cor_rastro(); // opcional
} else {
     cvShowImage("video", nova_imagem);
}
largura_linha += (largura_linha % 24) == 0 ? 0 : 1;
```

Lembra-se do contador que definimos antes? Então, ele é utilizado como controle, para que após 14 frames lidos, nós possamos utilizar o frame atual e eliminar o rastro da imagem, e também reinicializamos os valores da imagem_riscada para que o processo seja iniciado novamente, largura_linha para que a linha fique fina e volte a aumentar a largura novamente e por fim mudamos a cor. Durante os 14 frames que seguem, a bola é apresentada sendo rastreada.

Por fim liberamos a memória antes de prosseguir com o laço novamente e após o laço libera a estrutura de captura do vídeo. Observe o código a seguir:

```
cvReleaseImage(&imagem_tresh);
      delete momentos; }
  cvReleaseCapture(&captura);
            return 0; }
```

5 Conclusão

Na opinião do grupo foi interessante desenvolver um programa que através da movimentação de um objeto é capaz de simular um “mouse” 3D no espaço real. Com a elaboração deste trabalho, tivemos a oportunidade de aprender mais sobre Computação Gráfica e o Processamento de Imagens/Vídeo utilizando apenas alguns dos algoritmos de Visão Computacional que inclui a biblioteca OpenCV (*Open Source Computer Vision Library*). Conhecer esta biblioteca mostrou-se muito importante, pois diversas funções e estruturas estão disponíveis e prontas em OpenCV para serem utilizadas no tratamento e filtro de imagem, calibração de câmera, reconhecimento de objetos, análise estruturais entre outros, o

que facilita e possibilita a criação/desenvolvimento de diversos aplicativos na área de Visão Computacional.

6 Referências

[1] http://docs.opencv.org/index.html
[2] 4643B-04 – Computação Grafica I - Turma 128 - 2013/2 Profa. Dra. Soraia Raupp Musse. Notas de Aula e de Laboratório de Computação Grafica.
[3] http://www.aishack.in/