Detecção de Movimentos em um Vídeo por meio de uma Câmera Estática

Davi Marques[1], Quelita A. D. S. Ribeiro[1], Íris A.e dos Santos[1], Chaina S. Olivera[1]

[1]Departamento de Computação – Universidade Federal de Sergipe (UFS)
Sergipe – SE – Brasil

{mrqsdavi,quelita.diniz,irisandradesantos,chaina.oliveira} @gmail.com

Abstract. *Motion detection is widely used in applications such as monitoring traffic of cars on the roads, movies, games, among others. Therefore, this article describes an implementation in C + + for motion detection in a video by a static camera. Some features of OpenCV library were used to assist in the implementation of the code. The tests results are visualized that are relevant and satisfying, because it was observed in practice and the performance of the motion detection algorithm.*

Resumo. *A detecção de movimentos é amplamente utilizada em aplicações como: monitoramento de trafego de carros nas vias públicas, filmes, jogos, entre outros. Por esse motivo, este artigo descreve uma implementação na linguagem C++ para detecção de movimentos em um vídeo por meio de uma câmera estática. Algumas funções da biblioteca OpenCV foram utilizadas para auxiliar na implementação do código. Através dos testes visualizou-se os resultados que foram relevantes e gratificantes, pois, observou-se na prática o comportamento do algoritmo e a detecção dos movimentos.*

Introdução

O intuito desse artigo é mostrar a implementação de um algoritmo de detecção de movimentos através de uma câmera estática. A captura de movimentos é realizada através de diferenças de cores numa seqüência de vídeo.

Existem diversos processos que utilizam a detecção de movimentos em vídeo, dentre as quais, pode-se destacar o cinema ilustrado pelo filme Senhor dos Anéis produzido através da captura de movimentos. Como esta captura no filme foi atrelado a movimentos de uma pessoa real, pode-se constatar que animações em filmes e jogos são cada vez mais realistas por esse motivo.

Um conjunto de ferramentas existentes, atualmente, podem auxiliar o desenvolvimento de detecção de movimentos em vídeo, pode-se citar, a biblioteca OpenCV. Esta é *open-source,* distribuída gratuitamente e aberta a colaborações de voluntários. Foi idealizada com o objetivo de tornar a visão computacional acessível a usuários e programadores em áreas tais como a interação humano-computador em tempo real e a robótica.

Vale ressaltar, que a implementação do algoritmo é feita na linguagem C++ e utiliza-se a biblioteca OpenCV para simplificar a produção do mesmo.

Os processos de detecção de movimentos em vídeo são descritos na seção 1. Na seção 2 são retratados os objetivos deste artigo e são apresentadas algumas aplicações que usam a captura de movimentos em vídeo. Na seção 3 é pincelado os recursos utilizados para a produção deste projeto, tanto em termos de *software* como de *hardware*. O código desenvolvido para a detecção de movimentos está descrito na seção 4, como também uma breve descrição da biblioteca OpenCV e funções desta. Os testes realizados e os resultados obtidos são demonstrados na seção 5. E por fim, as considerações finais está na seção 6.

1. Detecção de Movimentos

Captura de movimentos (*Motion Capture*) é o processo de converter eletronicamente os movimentos de uma pessoa ou de um objeto para uma base de dados digital. Esta base de dados depois pode ser usada para produzir gráficos de animações em filmes, jogos, segurança e vigilância, ou mesmo, outras aplicações.

O projeto desenvolvido teve por objetivo detectar movimentos através de uma câmera estática. Para isso, um *background* (imagem de referência) é *(i)* reconhecido; *(ii)* rastreado. Após esses processos de reconhecimento e rastreamento de *background*, a sequência de vídeo é *(iii)* estimada para que variações de movimento no vídeo fosse detectadas e mostradas. Os processos citados são vistos com detalhes, abaixo:

(i) No reconhecimento, um sistema de visão necessita uma base de conhecimento dos objetos a serem reconhecidos. Ou seja, "Reconhecer significa conhecer de novo, e isto implica num processo onde existe algum conhecimento prévio e algum tipo de armazenamento do conhecimento sobre o objeto a ser reconhecido" Marengoni e Stringhini (2009). O *background*, então, serve como referência a ser reconhecido através das variações de movimentos.

(ii) O processo de rastreamento é um processo de reconhecer um padrão em uma sequência de imagens. " Os processos de rastreamento atrelam um conhecimento prévio sobre o movimento do objeto que está sendo rastreado para minimizar a busca entre as imagens em uma sequência" Marengoni e Stringhini (2009).

Neste projeto usa-se o rastreamento nas diferenças de cores. Ou seja, trata a mudança dinâmica das distribuições de probabilidade das cores numa seqüência de vídeo. Para cada *frame*, a imagem de referência é convertida para outra de distribuição de probabilidade de cor através de uma subtração da cor entre o *background* e um *frame* de vídeo.

Ou seja, o *background* é criado para nortear a diferença entre ele e a variação de movimentos detectada na ocasião de uma mudança de cor entre os *pixels* da imagem.

(iii) O processo de estimar identifica a localização de uma pessoa ou um objeto em movimento numa sequência de vídeo. Primeiramente, faz-se uma predição, baseada em um conhecimento prévio de dados da imagem. E em seguida faz-se uma correção, que usa novas medidas para corrigir as medidas estimadas realizada na predição.

Segundo Ornai *et al.* (2013) "Existem várias abordagens sobre captura de movimentos numa *webcam* através de *stream* de vídeo contínuo. [...] São baseadas na comparação entre o *frame* atual do vídeo e o *frame* anterior ou entre uma imagem

previamente estabelecida no início do *streaming*, que é chamada de imagem de referência".

2. Objetivos

O objetivo principal aqui é apresentar um algoritmo que detecte movimentos em um vídeo através de uma câmera estática.

Para isso, faz se necessário salientar que existem diversas aplicações que utilizam o algoritmo de detecção de movimentos em vídeo, dentre as quais, pode-se destacar "operações militares, entretenimento, em vários desportos e até em aplicações médicas. É também usado em visão computacional e robótica. No entanto, a área que mais utiliza a captura de movimentos é sem dúvida o cinema" Ornai *et al.* (2013). A figura 1 mostra o filme Senhor dos Anéis sendo produzido através da captura de movimentos. Como a captura deriva de movimentos de uma pessoa real, as animações em filmes e jogos são cada vez mais realistas.

Figura 1: Exemplo de um "Motion Capture". Criação de uma personagem no filme Senhor dos anéis.
Fonte: Ornai *et al.* (2013).

Além disso, a captura de movimentos é aplicado em diversas áreas, como sistemas de segurança/vigilância, em sistemas de interface humano-computador. É utilizada, também, na leitura do tráfego numa estrada, onde o sistema captura rapidamente carros que se movimentam na mesma.

Figura 2: Captura de movimentos em tráfego de trânsito.
Fonte: Ornai *et al.* (2013).

Objetiva-se também demonstrar a importância da captura de movimentos através do vídeo, e ainda, afirmar que atualmente várias áreas utilizam aplicações com recurso ao uso da detecção de movimento em vídeo por *webcam* ou qualquer outro dispositivo de vídeo.

Dentre, a pesquisa realizada, foi visto que existe um conjunto de ferramentas que podem auxiliar o desenvolvimento deste tipo de aplicações, pode-se citar, a biblioteca OpenCV, utilizada para simplificar e auxiliar o desenvolvimento do algoritmo.

3. Recursos Utilizados para a produção do algoritmo

Os recursos utilizados, foram:

– Câmera fixa (*webcam*) em uma posição genérica;

– Eclipse Kepler (*Eclipse IDE for C/C++ Developers*) – plataforma de desenvolvimento de software livre extensível, baseada em Java;

– C++ Linguagem de programação usada para desenvolver o algoritmo;

– A Biblioteca OpenCV *(Open Source Computer Vision Library*) foi usada, pois, simplifica a implementação de operadores na área de processamento de imagens e visão computacional;

– Sistema operacional: MAC (OS X 10.8.5) e Windows 8.

Os recursos de hardwares são:

Notebook Macbook Pro 2010 mid, espaço em disco: SSD 60GB + HD250GB; Memória RAM: 4 GB 1067 MHz DDR3; Placa de vídeo: NVIDIA *Graphics GeForce* 320M; Processador: 2,4 GHz *Intel Core 2 Duo*;Webcam *iSight* Câmera.

E Notebook Positivo S5050, espaço em disco: 500 Gb; Memória RAM: 6 Gb DDR3; Placa de vídeo: Processamento de vídeo integrado *Intel HD Graphics* 3000; Processador: *Intel Core* i3 2330M (2.20 GHz, 3 MB Cache); *Webcam* de alta resolução embutido.

4. Descrição do Projeto

Abaixo, descreve-se os passos do algoritmo (seção 4.1), uma breve descrição da biblioteca OpenCV e funções desta que estão contidas no código (seção 4.2). Por fim, o código implementado é apresentado na seção 4.3.

4.1 Breve descrição dos passos do código

Os passos desenvolvidos no algoritmo são:

(1) Exibe vídeo através da *webcam;*

(2) Seleciona-se 100 frames em escala de cinza que não contenha objetos em movimento para representar o *background;*

(3) Faz-se o cálculo de média para cada *pixel* das 100 imagens;

(4) Faz-se o cálculo de desvio padrão para cada *pixel* das 100 imagens;

(5)Equação para obtenção de uma imagem auxiliar:

Imagem Auxiliar = frame - média *background*

(6) Subtrai o *frame* selecionado como referência de um *frame* contendo objetos em movimento. Os *pixels* resultantes cujos valores sejam maiores que uma margem de erro (calculada com o desvio-padrão) são considerados como movimentação de vídeo.

(7) Através dos valores dos *pixels* de movimento são encontrados os valores máximo e mínimo de x e y (coordenadas dos *pixels*) para calcular a área do retângulo visível na imagem em movimento.

4.2 OpenCV e funções utilizadas desta no algoritmo

Antes de expor as funções da biblioteca OpenCV é preciso compreendê-la. A biblioteca OpenCV foi desenvolvida pela Intel e traz consigo mais de 2500 algoritmos, tais como: segmentação, reconhecimento de faces, aprendizado de máquinas, filtragem de imagens etc. É utilizada no desenvolvimento de aplicativos para as áreas de processamento de imagens e visão computacional. Existem versões dessa biblioteca disponíveis para *Windows, Unix, Android* e até *GPUs Nvidia* (CUDA). É *open-source,* distribuída gratuitamente e aberta a colaborações de voluntários. Ela foi idealizada com o objetivo de tornar a visão computacional acessível a usuários e programadores em áreas tais como a interação humano-computador em tempo real e a robótica. A biblioteca está dividida em cinco grupos de funções: Processamento de imagens; Análise estrutural; Análise de movimento e rastreamento de objetos; Reconhecimento de padrões e Calibração de câmera e reconstrução 3D".

Neste projeto foram usadas, as seguintes, funções do OpenCV (tabela 1):

Quadro 1: Funções do OpenCV

Função	Descrição
CvCapture	Estrutura que serve para apontar a memória onde o vídeo se encontra, para isso, declara-se um ponteiro para a estrutura *CvCapture*.
cvQueryFrame()	Responsável por adquirir os *frames* em sequência. Está função deve estar dentro de um *loop* para a leitura do vídeo. Ela recebe como argumento um ponteiro da estrutura *CvCapture* depois pega o próximo *frame* do vídeo na memória, um ponteiro é retornado para o frame. Ele usa a memória já alocada na estrutura *CvCapture.*
cvShowImage()	Mostra o *frame*. Com ela, é possível exibir uma imagem que esteja sendo manipulada pelo OpenCV diretamente na tela. Mas para isso, é necessário invocar outra função antes (*cvNamedWindow*), que cria uma janela gráfica.
cvNamedWindow	Abre uma janela na tela que pode conter e exibir uma imagem.
cvWaitKey()	Depois de mostrado o *frame*, espera-se alguns milissegundos definidos na função *cvWaitKey*(). No nosso código, se a tecla Esc for pressionada (ESC = caractere 27) então o loop será interrompido, por causa, do comando *break* contido no programa, consequentemente, os *frames* não serão mais mostrados.

cvtColor()	Converte a imagem em escalas de cinza.
cvarrToMat()	Converte os objetos do tipo *IplImage* para *Mat*, ou seja, pode-se acessar a imagem como uma matriz.
cvSaveImage()	Salva uma imagem.
cvRectangle()	Cria um retângulo a partir das coordenadas dos pontos dispostas na função *cvPoint()*.
cvPoint()	Transforma as coordenadas dadas em uma estrutura que o OpenCV pode trabalhar.
cvScalar()	É uma função onde os valores são armazenados;
cvReleaseCapture()	Libera a memória alocada para estrutura *CvCapture*().
cvDestroyWindow()	Fecha a janela.

4.3. Código do Projeto

Abaixo, encontra-se o código em C++ da detecção de movimentos por uma *webcam*. Além, de alguns comentários referente a implementação do mesmo.

```
#include <opencv/cv.h>
#include <opencv/highgui.h>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>

usingnamespace cv;
usingnamespace std;

int main(int argc,char *argv[])
{

int c;
   IplImage* frame_video;
   Mat *framesDeBackground = new Mat[100];

   CvCapture* cv_cap = cvCaptureFromCAM(0);
   cvNamedWindow("Video",0); // Cria janela

//Loop utilizado para mostrar a webCAM
for(;;) {

          frame_video = cvQueryFrame(cv_cap); // Captura frame do vídeo
if(frame_video != 0){
          cvShowImage("Video", frame_video);
      }

//Ao pressionar a tecla ESC o processo de calibragem deve ser iniciado
      c = cvWaitKey(10); // wait 10 ms or for key stroke
if(c == 27){
break;
      }
   }

int numeroLinhas = 0;
int numeroColunas = 0;

//Captura 100 frames para calcular uma média e obter imagem de background
int temp = 0;
while(temp <100){
          frame_video = cvQueryFrame(cv_cap);
          if(frame_video != 0){
             cvShowImage("Video", frame_video);
             Mat capturaBackground = cvarrToMat(frame_video);
             if(capturaBackground.channels() == 3){
                          Mat convertida;
                          cvtColor(capturaBackground, convertida,  CV_RGB2GRAY);
                          framesDeBackground[temp] = convertida;
                          numeroLinhas = convertida.rows;
                          numeroColunas = convertida.cols;
```

```
                        temp++;
                    }
                }


    }

    cout << numeroColunas <<"\n";

    Mat calculoBackground = Mat(numeroLinhas, numeroColunas, CV_8U);
    Mat desvios_padrao = Mat(numeroLinhas, numeroColunas, CV_8U);


//Faz cálculo de média de cada pixel do backgorund
for(int i = 0; i < numeroLinhas; i++){
            for(int j = 0; j < numeroColunas; j++){
                int somaTotalPixelCanal1 = 0;

                for(int k = 0; k <100; k++){
                        Mat backgroundIteracao = framesDeBackground[k];
                        somaTotalPixelCanal1 += backgroundIteracao.at<uchar>(i, j, 0);
                }

                calculoBackground.at<uchar>(i, j, 0) = somaTotalPixelCanal1/100;
            }
    }

//Faz cálculo do desvio padrao de cada pixel do backgorund
for(int i = 0; i < numeroLinhas; i++){
            for(int j = 0; j < numeroColunas; j++){
                int somatorio = 0;

                for(int k = 0; k <100; k++){
                        Mat backgroundIteracao = framesDeBackground[k];
                        int subtracaoTemp = backgroundIteracao.at<uchar>(i, j, 0) - calculoBackground.at<uchar>(i, j, 0);
                        somatorio += (subtracaoTemp*subtracaoTemp);
                }

                desvios_padrao.at<uchar>(i, j, 0) = somatorio/99;
            }
       }

    IplImage backgroundIpl = calculoBackground;
    cvSaveImage("/Users/Davi/Desktop/Back.jpg", &backgroundIpl);

    cout << desvios_padrao <<"\n";
    Mat back_blur_mat = Mat(numeroLinhas, numeroColunas, CV_8UC1);
    blur(calculoBackground, back_blur_mat, Size(10,10), Point(-1,-1));

for(;;) {
            frame_video = cvQueryFrame(cv_cap);
          Mat frame_mat = cvarrToMat(frame_video);

if(frame_video != 0){
            Mat blur_mat = Mat(frame_mat.rows, frame_mat.cols, CV_8UC1);
            blur(frame_mat, blur_mat, Size(10,10), Point(-1,-1));

            int xMax = -1;
            int xMin = blur_mat.cols +1;
            int yMax = -1;
            int yMin = blur_mat.rows + 1;

            for(int i=0; i<blur_mat.rows; i++){
                for(int j=0; j<blur_mat.cols; j++){

                        int divisor = desvios_padrao.at<uchar>(i, j, 0);

                        if(divisor == 0){
                                divisor = 1;
                        }

                        int value = abs(blur_mat.at<uchar>(i, j, 0) - back_blur_mat.at<uchar>(i, j, 0));

                        if(value > divisor*30){
                                value = 255;
                        }else{
                            value = 0;
                        }

                        if(value == 255){
                                if(j < xMin){
                                xMin = j;
                            }
```

```
                        if(j > xMax){
                                xMax = j;
                           }

                                if(i < yMin){
                                yMin = i;
                                }

                                if(i > yMax){
                                yMax = i;
                                }
                        }

                    }
                }

                cvRectangle(frame_video,
                        cvPoint(xMin,yMin),
                        cvPoint(xMax, yMax),
                        cvScalar(0, 255, 0, 0),
                1, 8, 0);
                cvShowImage("Video", frame_video);
               }

                c = cvWaitKey(10);
                if(c == 27){
                    break;
                }

    }

    cvReleaseCapture( &cv_cap );
    cvDestroyWindow("Video");

return0;

}
```

5. Testes e Resultados

Para expor os resultados obtidos do algoritmo, faz-se necessário explicar como o algoritmo funciona, e em seguida os resultados são mostrados.

Ao rodar o algoritmo. Primeiramente, captura-se o *background* através da *webcam* do computador, ao criar essa imagem de referência é necessário que a tecla ESC seja pressionada. O processamento do *background* é realizado através de 100 (cem) imagens. Dessa forma, calcula-se uma média delas e desvio-padrão. Vale salientar, que essa média é feita *pixel* a *pixel* da imagem. O motivo disso, é obter um *background* com resultado mais confiável e menos propenso a ruídos. A maior parte das câmaras produzem ruído nas imagens, o que faz com que o programa assuma que existiu movimento numa região onde na realidade não existiu, além disso, variações de iluminação interferem na detecção de movimentos.

Após, isso aplica-se ao um *blur background* para reduzir, ainda mais, o ruído.

No modo vídeo, ou seja, quando a detecção de movimentos já está sendo processada, o algoritmo, no estado inicial possui o *background* já projetado e pronto para captar possíveis movimentos.

A partir, do momento que movimentos são detectados, um retângulo é delineado, ou seja, uma borda em retângulo é criada e posicionada ao redor do objeto ou da pessoa. A borda serve para indicar a localização do mesmo ao mesmo tempo que ocorre o movimento.

O usuário, então, percebe nitidamente onde existe movimento. A saída produzida por este algoritmo é a imagem atual, com a zona onde existiu movimento contornada com um retângulo.

Exemplificando, a execução do algoritmo em termos de coordenadas da imagem. Tem-se uma imagem com um eixo de coordenadas XY. Cada *pixel* vai ser comparado com o *pixel* da *frame* de referência (*background*) que esteja na mesma posição. Ou seja, o *pixel* da posição (3,10), vai ser comparado com o *pixel* da posição (3,10) da *frame* de referência. "Esta abordagem é muito útil em termos de compressão de vídeo pois apenas é necessário estimar os *pixels* diferentes e escrever apenas as diferenças, não sendo necessário escrever o *frame* totalmente de novo"Ornai *et al.* (2013).

Efetuou-se vários testes para verificar o desempenho do algoritmo. Tanto para redução de ruído na imagem, quanto para a detecção de movimentos entre o *background* (figura 3) e o *frame* atual. Dentre esses testes, obteve-se uma resposta aceitável no algoritmo. E os resultados foram, consideravelmente, favoráveis ao intuito do projeto. Nos testes, foi criado também a opção de salvar a imagem convertida, contendo a variação do movimento (figura 4). Por fim, quando a tecla ESC é pressionada a execução do algoritmo é encerrada.

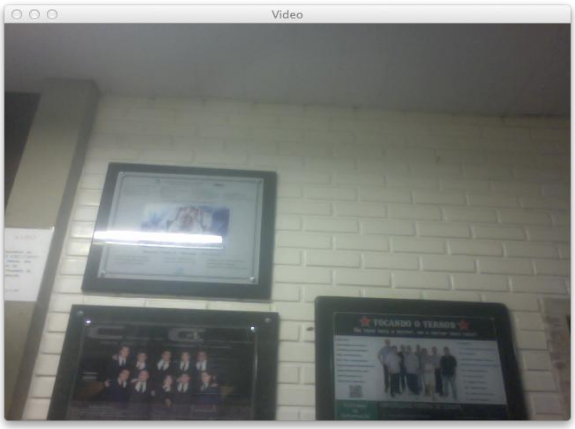

Figura 3:*Background.*

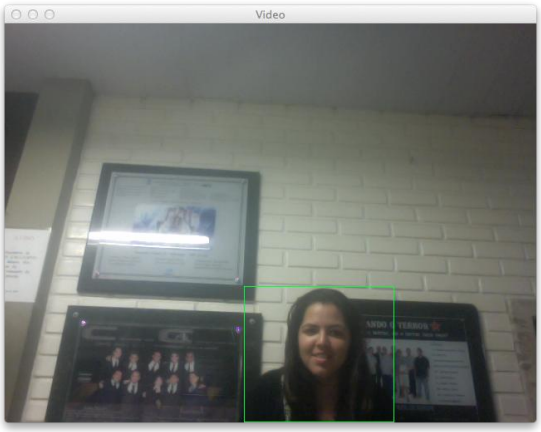

Figura 4: imagem convertida contendo a variação do movimento.

- Os problemas encontrados na realização do algoritmo foram:
  - – Sensibilidade a variação de iluminação da cena;
  - – Redução de ruído.

6. Considerações Finais

Demonstra-se que a captura de movimentos em vídeo tem grande relevância em aplicações usadas atualmente por diversas pessoas e que muitas destas aplicações fazem parte do dia-a-dia delas.

Consideramos que um dos aspectos mais complexos na detecção de movimentos é o tratamento do ruído e tratamento de sensibilidade de iluminação por parte da variação de cor refletida na câmera.

Enfim, nos sentimos felizes em poder terminar esse projeto com êxito. Principalmente, pelo fato de nos dedicarmos para que o objetivo deste fosse alcançado.

Agradecemos, aos professores Leonardo e Daniel pelas orientações e esforço em nosso auxílio.

7. Referências

MARENGONI, M. STRINGHINI, D.**Tutorial: Introdução à Visão Computacional usando OpenCV.**2009. Disponível em: <seer.ufrgs.br/rita/article/view/rita_v16_n1_p125>, acesso em 27 Set 2013.

ORNAI, A. SANTOS, E. PEREIRA, L. FERNANDES, P.**Detecção de Movimento - Criação de instrumento musical.**2013. Disponível em: <academia.edu/3024446/Deteccao_de_Movimento_-_Criacao_de_Instrumento_Musical>, acesso em 29 Set 2013.