

UNIVERSIDADE FEDERAL FLUMINENSE

Renato Moraes dos Santos

UM ESTUDO DE PROCESSAMENTO DE IMAGENS COM OPENCV

Niterói

2011

Renato Moraes dos Santos

UM ESTUDO DE PROCESSAMENTO DE IMAGENS COM OPENCV

Trabalho de Conclusão de Curso
submetido ao Curso de Tecnologia em
Sistemas de Computação da
Universidade Federal Fluminense como
requisito parcial para obtenção do Título
de Tecnólogo em Sistemas de
Computação.

Orientador(a):

Marcelo Panaro de Moraes Zamith

NITERÓI

2011

Renato Moraes dos Santos

UM ESTUDO DE PROCESSAMENTO DE IMAGENS COM OPENCV

Trabalho de Conclusão de Curso
submetido ao Curso de Tecnologia em
Sistemas de Computação da
Universidade Federal Fluminense como
requisito parcial para obtenção do Título
de Tecnólogo em Sistemas de
Computação.

Niterói, ____ de _____ de 2011.

Banca Examinadora:

Prof^a. Juliana Mendes Nascente Silva, Msc. – Avaliadora

UFF - Universidade Federal Fluminense

Prof. Marcelo Panaro de Moraes Zamith, Msc. – Orientador

UFF - Universidade Federal Fluminense

Dedico este trabalho a minha mãe que sempre me apoiou e ajudou nos estudos e em tudo na minha vida.

AGRADECIMENTOS

A minha família, que proporcionou a base,
que me fez chegar ao que sou hoje.

Aos meus amigos que me ajudaram, cada um
a sua maneira, e de inúmeras formas
diferentes.

Aos meus companheiros de curso que me
ajudaram sempre que precisei.

E aos meus professores que me ajudaram ao
longo da minha vida.

“O sucesso é um professor perverso. Ele seduz as pessoas inteligentes e as faz pensar que jamais vão cair”.

Bill Gates

RESUMO

A evolução da computação no mundo moderno é constante. Nesta evolução podemos citar o processamento de imagens digitais como um de seus principais ramos. O processamento de imagens é muito utilizado e relevante em diversas áreas, pois através deste processo, possibilitará a retirada das informações contidas em uma imagem. O processo de extração de informações de imagens, não é uma tarefa fácil e necessita do uso de técnicas muitas vezes complexas e de dados com boa qualidade, para se obter sucesso nos resultados.

Esse projeto tem como objetivo apresentar e implementar as operações básicas e fundamentais para o desenvolvimento de aplicações na área da visão computacional com o uso da biblioteca gráfica da Intel, OpenCV.

Palavras-chaves: Processamento de Imagens, Erosão, *Template Matching*, *Mean Shift* e Binarização.

ABSTRACT

The evolution of computing in the modern world is constant. In this evolution we can cite the digital image processing as one of its major branches. Image processing is widely used and important in many areas, because through this process, allow the withdrawal of the information contained in an image. The process of extracting information from images is not an easy task and requires the use of often complex and technical data with good quality, to achieve successful results. This project aims to present and implement the basic operations and fundamental to the development of applications in computer vision using the Intel graphics library, OpenCV.

Key words: **Image processing, Erosion, Template Matching, Mean Shift e Thresholding.**

GLOSSÁRIO

Pixel: É o menor elemento de uma imagem. A qualidade de uma imagem é definida pela quantidade presente de *pixels* em uma polegada linear.

Script: Arquivo de texto utilizado para designar uma sequência de comando e tarefas a serem executadas.

Template: Imagem Modelo.

Template matching: Correspondência por modelo, técnica de busca em imagem (Reconhecimento de padrões).

Python: Linguagem de programação interpretada ou compilada. Linguagem utilizada para desenvolvimento das aplicações do projeto.

Mean shift: Algoritmo de Deslocamento médio usado para rastreamento (Reconhecimento de padrões).

Hue: Matiz.

Tracker Polhemus: Rastreador da Famosa indústria de desenvolvimento de sistemas de rastreamento e identificação biométrica.

WebCam: Câmera de vídeo.

Web: A World Wide Web (WWW - Rede de alcance mundial) é um sistema de documentos em hipermídia que são interligados executados na internet.

Array: Matriz ou vetor.

Imagen digital: Imagem digital refere-se a uma imagem $f(x,y)$ que sofreu discretização tanto das suas coordenadas espaciais quantas do seu brilho. Uma imagem digital pode ser vista como uma matriz bidimensional cujas linhas e colunas identificam os pontos da imagem e cujos valores representam a intensidade dos níveis de cinza em cada ponto. Cada elemento desta matriz é denominado pixel.

Imagen: O termo imagem é definido como sendo uma função de intensidade luminosa bidimensional $f(x,y)$, onde a amplitude f , nas coordenadas (x,y) , indica o brilho da imagem neste ponto. As imagens percebidas pelo olho humano normalmente consistem de luz refletida pelos objetos. Pode-se considerar $f(x,y)$ como sendo composta de dois componentes: a iluminação $i(x,y)$ que é a quantidade de luz incidente sobre a cena e a refletância $r(x,y)$ que representa a quantidade de luz refletida pelos objetos na cena.

Assim, $f(x,y) = i(x,y) \cdot r(x,y)$.

Segmentação: Segmentação é o processo de subdivisão da imagem nas suas partes ou objetos constituintes. A segmentação é uma das principais operações na análise automática de imagem porque é nesta etapa que os objetos ou partes de interesse são extraídos para descrição ou reconhecimento subsequentes. Os algoritmos de segmentação baseiam-se em duas propriedades dos níveis de cinza: descontinuidade e similaridade. Na primeira categoria, a imagem é particionada com base em mudanças

bruscas nos níveis de cinza, permitindo a identificação dos pontos, linhas e bordas da imagem. Na segunda categoria, estão os algoritmos baseados em limiares, no crescimento de regiões ou na partição e aglomeração de regiões.

LISTA DE ILUSTRAÇÕES

1 Escala cinza.....	22
2 Imagem Colorida.....	22
3 Histograma.....	24
4 Vizinhança de pixels.....	28
5 Exemplo de estrutura para reconhecimento de faces.....	30
6 Imagem Origem.....	32
7 Template.....	32
8 Resultado.....	32
9 Trecho do script demhist.py.....	37
10 Histograma.....	38
11 Trecho do script edge.py.....	39
12 Binarização.....	39
13 Trecho do script morphology.py.....	41
14 Erosão.....	41
15 Dilatação.....	42
16 Trecho do script laplace.py.....	43
17 Laplace.....	43
18 Trecho do script squares.py.....	44
19 Quadrados.....	45
20 Trecho do script template.py.....	46
21 Equações dos métodos para cálculo do template matching e Opencv.....	47
22 Template matching - etapas.....	48
23 Template matching - resultado.....	49
24 Trecho do script match.py.....	50
25 Template matching - imagem resultante.....	51
26 Trecho do script camshift.py.....	53
27 Camshift: Tela Inicial.....	53
28 Camshift: Rastreamento.....	54
29 Camshift: Rastreamento.....	54

30 Camshift: Rastreamento.....	54
31 Camshift: Modo Retroprojeção.....	55
32 Trecho do script fback.py.....	56
33 Fluxo inicial.....	56
34 Fluxo pós movimentação.....	57
35 Trecho do script motempl.py.....	58
36 Fluxo - momento inicial.....	59
37 Fluxo - momento pós movimentação.....	60
38 Arquivo xml de mapeamento da face frontal.....	61
39 Trecho do script facedetect.py.....	62
40 Face detect.....	63
41 Face detect.....	63
42 Face detect.....	64
43 Face detect.....	64

LISTA DE ABREVIATURAS E SIGLAS

CAMSHIFT – *Continuously Adaptive Mean Shift (Transformação Contínua de Adaptação).*

OpenCV – *Open Source Computer Vision Library (Biblioteca de Computação Visual em Código Aberto).*

RGB – *Red, Green, Blue (Vermelho, Verde, Azul).*

YIQ – Y representa luminância, I significa em fase e Q significa quadratura.

HSI – *Hue, Saturation, Intensity (Matiz, Saturação, Intensidade).*

SAD – Soma das diferenças absolutas.

SSD – Soma dos desvios dos quadrados.

HSV – *Hue, Saturation, Value (Matiz, Saturação, Valor).*

GPU – *Graphic processor unit (Unidade de processamento gráfico).*

MHI – *Motion History Image (Histórico de movimentação na imagem).*

PIL – *Image Processing Library (Biblioteca de processamento de imagem)*

IPL – *Image Processing Library (Biblioteca de processamento de imagem)*

NTSC – National Television System Committee (*Comitê Nacional de Sistemas de Televisão*).

SUMÁRIO

<u>RESUMO</u>	7
<u>ABSTRACT</u>	8
<u>GLOSSÁRIO</u>	9
<u>LISTA DE ILUSTRAÇÕES</u>	12
<u>LISTA DE ABREVIATURAS E SIGLAS</u>	14
<u>1 INTRODUÇÃO</u>	18
<u>2 Fundamentação teórica</u>	20
<u>2.1 Imagem digital</u>	20
<u>2.2 Cores</u>	20
<u>2.3 Processamento de Imagens</u>	21
<u>2.3.1 ESCALA DE CINZA</u>	22
<u>2.3.2 HISTOGRAMA</u>	23
<u>2.3.3 BINARIZAÇÃO</u>	24
<u>2.3.4 FILTROS</u>	26
<u>2.3.4.1 SUAVIZAÇÃO</u>	27
<u>2.3.5 TRANSFORMAÇÕES MORFOLÓGICAS</u>	27
<u>2.3.5.1 EROSÃO</u>	27
<u>2.3.6 VISÃO COMPUTACIONAL</u>	28
<u>2.3.6.1 RECONHECIMENTO DE PADRÕES</u>	29
<u>2.3.6.1.1 TEMPLATE MATCHING</u>	30
<u>2.3.6.1.1.1 Distância Euclidiana</u>	31
<u>2.3.6.1.1.2 Correlação</u>	31
<u>2.3.6.1.2 RASTREAMENTO</u>	32
<u>2.3.6.1.2.1 Mean shift</u>	33
<u>2.3.6.1.2.2 Camshift</u>	33
<u>2.3.6.2 Fluxo óptico</u>	35
<u>3 Desenvolvimento</u>	36
<u>3.1 Implementando conceitos de PROCESSAMENTO DE IMAGENS</u>	36

3.1.1 Análise de Histograma	36
3.1.2 Análise de binarização	38
3.1.3 Erosão e dilatação	40
3.1.4 Filtro Laplace	42
3.2 Implementando conceitos de VISÃO COMPUTACIONAL	44
3.2.1 Busca de quadrados	44
3.2.2 Implementando Template matching	45
3.2.3 Implementando Camshift	52
3.2.4 Rastreamento de fluxo por pontos	55
3.2.5 Rastreamento de fluxo	57
3.2.6 Implementando um Detector de facial	60
4 CONCLUSÕES E TRABALHOS FUTUROS	65
5 Referências bibliográficas	66
ANEXOS	70

1 INTRODUÇÃO

A área de processamento de imagens vêm despertando maior interesse acadêmico e de mercado a cada dia, possibilitando o desenvolvimento de diversas aplicações. Podemos dividir estas aplicações em duas categorias bem distintas: (1) o aprimoramento de informações pictóricas para interpretação humana; e (2) a análise automática por computador de informações extraídas de uma cena. Reservaremos a expressão 'processamento de imagens' para designar a primeira categoria, adotando os termos 'análise de imagens', 'visão por computador' (ou 'visão computacional') e 'reconhecimento de padrões' para a segunda.

Esse projeto tem como objetivo apresentar e implementar as operações básicas e fundamentais para o desenvolvimento de aplicações na área da visão computacional. Primeiramente entenderemos a teoria processamento de imagem que são a base da maioria dos processos de visão computacional, apresentando ainda a evolução do processamento de imagens digitais em tempo real.

Através do uso da biblioteca OpenCV da Intel foi possível manipular as imagens oriundas de imagens estáticas (Fotografias) e imagens dinâmicas em tempo real (WebCam). OpenCV é uma biblioteca *Open Source* desenvolvida para C e C++, existe também uma versão para Python devido à facilidade de interação entre Python e C.

A biblioteca OpenCV foi desenvolvida pela Intel e possui mais de 500 funções [31] e 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 disponível com o código fonte e os executáveis (binários) otimizados para os processadores Intel. Um programa que utiliza a biblioteca OpenCV, ao ser executado, invoca automaticamente uma biblioteca que detecta o tipo de processador e carrega, por sua vez, a biblioteca otimizada para este. Juntamente com o pacote OpenCV é

oferecida a biblioteca IPL (*Image Processing Library*), da qual a OpenCV depende parcialmente.

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.

O pacote OpenCV está disponível na internet, assim como seu manual de referência [31].

A organização do trabalho é feita da seguinte forma, o primeiro capítulo refere-se às apresentações e objetivos referentes a este trabalho, trazendo uma introdução sobre a monografia em geral.

O segundo capítulo refere-se aos fundamentos teóricos do projeto em questão. A teoria por trás do software. Iremos falar sobre *Template Matching*, *CamSHIFT*, *Mean Shift*, Histogramas, Erosão, entre outros.

O terceiro capítulo refere-se à organização do projeto, um paralelo sobre como se desenvolveu o projeto e os detalhes da implementação da biblioteca OpenCV neste projeto, seu funcionamento através de Figuras e exemplos.

O quarto capítulo refere-se à conclusão do projeto, tecendo um paralelo entre os principais avanços conquistados e o que pode ser melhorado, para possíveis projetos futuros.

Localizadas posteriormente ao capítulo cinco, estão as referências bibliográficas e demais fontes de pesquisa indispensáveis para a elaboração deste trabalho.

Complementando este trabalho, está um apêndice, melhorando a compreensão dos capítulos propostos, conforme abaixo.

Nos apêndices temos os *scripts* desenvolvidos através das técnicas citadas.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentadas alguns fundamentos e técnicas fundamentais de processamento de imagens. Também serão abordados conceitos de visão computacional.

2.1 IMAGEM DIGITAL

A representação de uma imagem digital é feita por uma matriz que mapeia as cores da imagem do mundo real para o digital, transformando cada ponto em um pixel. Um pixel é a parte indivisível da imagem digital, em termos comparativos, poderíamos dizer que um pixel é como um átomo da imagem, sua área na imagem depende da resolução da mesma, ou seja, quanto maior a resolução menor será o tamanho do pixel. A medida dessa resolução é dada em *Dots Per Inch* ou pontos por polegada.

No padrão RGB (*RED, GREEN, BLUE*) existem três valores associados, um para o vermelho, outro para o verde, e outro para o azul. A mistura desses valores resulta numa gama imensa de cores, sendo assim, um processo igual ao usado por pintores na composição de suas pinturas.

Existem outros padrões, como o HSI que mapeia outras propriedade das cores – matriz, ou cor [1].

2.2 CORES

As cores são sensações que nós, seres humanos, temos em resposta à luz que incide nos nossos olhos e diversos tipos de materiais encontrados no mundo físico.

Os Tipos de processo de formação de cores são: Aditivo, Subtrativo e por Pigmentação. Discutiremos o processo Aditivo apenas, pois é o processo utilizado

na computacionalmente. O processo Aditivo, que é como obtemos o sistema padrão de três cores, como o RGB. Porém existem cores que não são igualadas pelas três cores padrão apenas. Nesse caso adiciona-se uma cor padrão ao lado esquerdo, ou seja, antes do valor de R (correspondendo matematicamente, a uma intensidade negativa).

Em editores gráficos é comum a utilização de sistemas de interface, pois oferecem uma interface adequada a especificação de cores por um usuário comum. Em geral, especificam cores através de três parâmetros: matiz, saturação, e luminância.

Tipos de sistemas de interface: baseados em coordenadas podemos citar o HSV e o HSL, baseados em amostras podemos citar Pantone e Munsell.

Entender os sistemas de cores é fundamental, para entendermos a análise de formação de imagens, e algumas técnicas de processamento de imagens.

2.3 PROCESSAMENTO DE IMAGENS

Os processos de visão computacional, que é o conjunto de métodos e técnicas através dos quais sistemas computacionais podem ser capazes de interpretar imagens, muitas vezes, necessitam de uma etapa de pré-processamento envolvendo o processamento de imagens. As imagens de onde queremos extrair alguma informação em alguns casos precisam ser convertidas para um determinado formato ou tamanho e precisam ainda ser filtradas para remover ruídos provenientes do processo de aquisição da imagem.

Os ruídos podem aparecer de diversas fontes, como por exemplo, o tipo de sensor utilizado, a iluminação do ambiente, as condições climáticas no momento da aquisição da imagem, a posição relativa entre o objeto de interesse e a câmera. Note que ruído não é apenas interferência no sinal de captura da imagem, mas também interferências que possam atrapalhar a interpretação ou o reconhecimento de objetos na imagem.

Os filtros são as ferramentas básicas para remover ruídos de imagens, neste caso, o ruído é aquele que aparece no processo de aquisição da imagem.

Os filtros podem ser espaciais, que são aqueles que atuam diretamente na imagem) ou de frequência, onde a imagem é inicialmente transformada para o domínio de frequência usando da transformada de Fourier (geralmente através da transformada de Fourier discreta) e então é filtrada neste domínio e em seguida a imagem filtrada é transformada de volta para o domínio de espaço.

2.3.1 ESCALA DE CINZA

Escala de cinza é uma escala em tons de cinza em que os valores *pixels* da imagem variam de acordo com sua intensidade (luminosidade), numa escala entre preto e branco. É calculada a partir da conversão RGB para YIQ que só normaliza os níveis de brilho da imagem [20]. YIQ é um modelo de cor usado para coloração de TV na América (NTSC = *National Television System Committee*). Y é a Luminância, I & Q são as cores (I=vermelho/verde, Q=azul/amarelo) [37].

Na escala de cinza os *pixels* possuem coloração de preto, variando de cinza até chegar a cor branca, não possuindo assim cores como azul, amarelo, vermelho ou verde. As Figuras 1 e 2 exemplificam a citação.



Figura 1: Escala de Cinza



Figura 2: Imagem Colorida

2.3.2 HISTOGRAMA

O histograma de uma imagem é um conjunto de números indicando o percentual de *pixels* naquela imagem que apresentam um determinado nível de cinza. Estes valores são normalmente representados por um gráfico de barras que fornece para cada nível de cinza o número (ou o percentual) de *pixels* correspondentes na imagem. Através da visualização do histograma de uma imagem obtemos uma indicação de sua qualidade quanto ao nível de contraste e quanto ao seu brilho médio (se a imagem é predominantemente clara ou escura).

Cada elemento deste conjunto é calculado conforme a Equação 2.1:

$$p_r(r_q) = \frac{n_q}{n}$$

Equação 2.1

onde:

$$0 \leq r_q \leq 1$$

$k = 0, 1, \dots, L-1$, onde L é o número de níveis de cinza da imagem digitalizada;

n = número total de *pixels* na imagem;

$p_r(r_q)$ = probabilidade do k -ésimo nível de cinza;

n_q = número de *pixels* cujo nível de cinza corresponde a k .

O conceito de histograma também é aplicável a imagens coloridas. Neste caso, a imagem é decomposta de alguma forma (por exemplo, em seus componentes R, G e B) e para cada componente é calculado o histograma correspondente [20].

2.3.3 BINARIZAÇÃO

O algoritmo de binarização é normalmente utilizado quando é preciso fazer a separação entre o fundo da imagem com os objetos que representam os caracteres. A binarização ou limiarização (do inglês, *thresholding*) é o método mais simples para segmentação de imagens que consiste em separar regiões de não interesse através da escolha de um ponto limiar. Em alguns casos não é possível dividir a imagem em apenas um limiar que resulte em resultados satisfatórios, nesses casos são definidos mais de um ponto de corte (*threshold*) da imagem [20].

Basicamente as escolhas dos limiares são feitas de acordo com o histograma dos *pixels* da imagem em escala de cinza, como exemplificado na Figura 3.

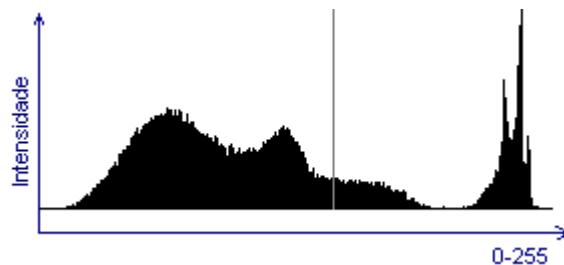


Figura 3: Histograma

Assim, a divisão da imagem em duas classes pode ser resolvida utilizando duas cores, por padrão, utilizamos as cores preto e branco. Portanto qualquer *pixel* com valor menor que o ponto de corte terá sua cor alterada para branco (valor 255) e qualquer valor acima terá sua cor alterada para preto (valor 0), função a função $p(x,y)$ definida na Equação 2.2 descreve essa valoração.

$$P(x,y) = \begin{cases} 255, & \text{se } f(x,y) < T \\ 0, & \text{se } f(x,y) > T \end{cases}$$

Equação 2.2 Equação que descreve a valoração das cores preto e branco.

A binarização é referida por alguns autores como um método que separa os objetos do fundo(*background*) com os objetos da imagem (*foreground*). Dessa forma, a escolha do ponto de corte é de extrema importância pois precisamos garantir uma melhor separação do conteúdo da imagem do seu fundo e/ou dos possíveis ruídos [20].

Através da observação do histograma se torna mais fácil a escolha do ponto de corte, e quanto mais bimodal for esse diagrama, mais fácil será a sua escolha. Esse trabalho utiliza um método automático para realizar a escolha dos pontos de corte, conforme descrito em [22].

Tal método separa a imagem em duas classes de maneira que o valor do *threshold* maximize a variância entre elas. Este método é conhecido como método de Otsu, o nome de seu criador [22].

O método de Otsu baseia-se no histograma normalizado (da imagem) como uma função de densidade de probabilidade discreta, conforme a Equação 2.1:

$$p_r(r_q) = \frac{n_q}{n} \quad q = 0, 1, 2, \dots, L - 1$$

Onde n é o número total de *pixels* na imagem, n_q é o número de *pixels* com intensidade r_q e L é o número total dos possíveis níveis de intensidade na imagem.

O método de Otsu escolhe o *threshod* de valor k (tal que k é um nível de intensidade onde $C_0 = [0, 1, \dots, k-1]$ e $C_1 = [k, k+1, \dots, L-1]$) que maximiza a variância entre classes, que é definido como:

$$\sigma_B^2 = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2$$

Equação 2.3

Onde: $\omega_0 = \sum_{q=0}^{k-1} p_q(r_q)$

$$\omega_1 = \sum_{q=k}^{L-1} p_q(r_q)$$

$$\mu_0 = \sum_{q=0}^{k-1} q p_q(r_q) / \omega_0$$

$$\mu_1 = \sum_{q=k}^{L-1} q p_q(r_q) / \omega_1$$

$$\mu_T = \sum_{q=0}^{L-1} q p_q(r_q)$$

Para seleção do ponto de corte de uma dada imagem, a classes de probabilidade ω_0 e ω_1 indicam porções das áreas ocupadas pelas classes C_0 e C_1 . As médias de classes μ_0 e μ_1 servem como estimativas dos níveis de cinza.

limite inferior da escala de cinza, quando a imagem tiver um único valor constante de cinza, e o limite superior quando a imagem apresentar apenas dois valores de cinza.

2.3.4 FILTROS

As técnicas de filtragem são transformações da imagem pixel a pixel, que não dependem apenas do nível de cinza de um determinado pixel, mas também do valor dos níveis de cinza dos pixels vizinhos. O processo de filtragem é feito utilizando matrizes denominadas máscaras, as quais são aplicadas sobre a imagem. Abordaremos a máscara de suavização [1].

2.3.4.1 SUAVIZAÇÃO

É uma operação simples de processamento de imagem e frequentemente utilizada. Há muitas razões para suavizar uma imagem, mas geralmente é feito para reduzir os ruídos de uma imagem. A suavização também é importante quando queremos reduzir a resolução de uma imagem de maneira fundamentada.

Em OpenCV através da função cvSmooth() é feita a suavização. Dentro desta função temos 5 tipos de suavização: CV_BLUR (Desfocar simples) , CV_BLUR_NO_SCALE (Desfocar sem escalar), CV_MEDIAN (Desfocar mediano), CV_GAUSSIAN (Desfocar Gaussiano), CV_BILATERAL (Desfocar bilateral).

2.3.5 TRANSFORMAÇÕES MORFOLÓGICAS

Nesta seção apresentaremos a erosão. Uma das muitas transformações morfológicas existentes, mas que em nosso caso será utilizada no desenvolvimento dos aplicativos em visão computacional.

2.3.5.1 EROSÃO

A erosão de uma imagem é um filtro simples e pode ter diversas configurações que serão alteradas de acordo com o necessário, como exemplo, excluir retas na vertical, retas horizontais, *pixels* isolados, entre outros [1].

O processo de erosão é normalmente utilizado para a limpeza de impurezas na imagem. Essas impurezas são provenientes de elementos irrelevantes que compõem a imagem ou por partes da imagem que precisamos extrair para um correto pós-processamento [22].

Para decidirmos se um pixel deve ou não ser excluído (erodido) precisamos saber a priori o valor dos seus *pixels* vizinhos, como mostra a Figura 4.

		$\text{pixel}(x,y-1)$
$\text{pixel}(x-1,y)$	$\text{pixel}(x,y)$	$\text{pixel}(x+1,y)$
		$\text{pixel}(x,y+1)$

Figura 4: Vizinhança de pixels

Se o *pixel* atual (x,y) for um *pixel* preto e os *pixels* vizinhos (acima, a frente, abaixo, atrás) também tiverem o valor preto, o *pixel* atual não é erodido.

Caso o *pixel* atual seja um *pixel* preto e algum de seus vizinhos não seja preto, isto é, possuí valor associado 0, então removemos o *pixel* atual da imagem. Dessa forma, as bordas dos elementos isolados são erodidas gradualmente a cada iteração, por isso precisamos aplicar o filtro na imagem várias vezes. Esse processo faz com que a imagem sofra diminuição do seu tamanho original sem que haja a perda das características geométricas da imagem.

2.3.6 VISÃO COMPUTACIONAL

Na parte relacionada ao processamento de imagens ficou caracterizado o caráter de processo de baixo nível, mais precisamente a eliminação de ruídos e melhoria no contraste das imagens. Neste item começamos a migrar para os processos mais relacionados com a visão computacional. Veremos inicialmente um processo de nível médio (reconhecimento de padrões) para em seguida analisarmos processos mais cognitivos, como o rastreamento de um objeto numa sequência de imagens.

2.3.6.1 RECONHECIMENTO DE PADRÕES

Para fazer o reconhecimento um sistema de visão é necessário uma base de conhecimento dos objetos a serem reconhecidos, esta base de conhecimento pode ser implementada diretamente no código, através, por exemplo, de um sistema baseado em regras, ou esta base de conhecimento pode ser aprendida a partir de um conjunto de amostras dos objetos a serem reconhecidos utilizando técnicas de aprendizado de máquina.

O reconhecimento de objetos é uma das principais funções da área de visão computacional e está relacionado diretamente com o reconhecimento de padrões. Um objeto pode ser definido por mais de um padrão (textura, forma, cor, dimensões, etc) e o reconhecimento individual de cada um destes padrões pode facilitar o reconhecimento do objeto como um todo. As técnicas de reconhecimento de padrões podem ser divididas em dois grandes grupos: estruturais, onde os padrões são descritos de forma simbólica e a estrutura é a forma como estes padrões se relacionam; o outro grupo é baseado em técnicas que utilizam teoria de decisão, neste grupo os padrões são descritos por propriedades quantitativas e deve-se decidir se o objeto possui ou não estas propriedades.

Os processos de reconhecimento de padrões podem ainda ser uma mistura das técnicas utilizadas nestes dois grupos, por exemplo, no processo de reconhecimento de faces apresentado em [24], é utilizado um modelo estrutural para determinar o local mais provável para se encontrar partes de uma face (boca, olhos e pele), conforme apresentado na Figura 5. Cada uma destas partes pode agora ser reconhecida utilizando outro tipo de técnica, por exemplo, os olhos podem ser reconhecidos utilizando uma rede neural, a pele pode ser reconhecida por uma análise estatística e a boca pode ser reconhecida por um critério de distância mínima, todas são técnicas de teoria de decisão.

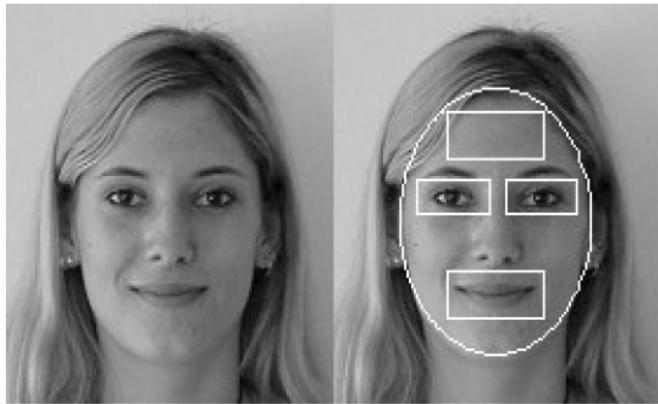


Figura 5: Exemplo de estrutura para o reconhecimento de faces. O modelo indica locais onde se esperam encontrar olhos, boca e pele. Reconhecer as partes pode levar ao reconhecimento da face.

É difícil encontrar técnicas estruturais prontas em bibliotecas, uma vez que estas técnicas dependem da estrutura de cada objeto. Para alguns objetos específicos, porém, é possível encontrar pacotes prontos. O OpenCV não possui uma ferramenta pronta que utilize este tipo de técnica [6]. As técnicas baseadas em teoria de decisão são mais gerais e podem ser adaptadas a diferentes tipos de objetos.

2.3.6.1.1 TEMPLATE MATCHING

Template matching é uma técnica em processamento de imagens digitais para encontrar pequenas partes de uma imagem que corresponda ou seja equivalente com uma imagem *template* (modelo). Esta técnica tem aplicações em processos industriais, como parte do controle de qualidade, em navegação de robôs móveis ou na detecção de bordas de imagens [25].

O processo de correspondência move a imagem *template* para todas as posições possíveis na imagem origem e calcula um índice numérico que indica quanto bem o modelo ajusta a imagem nessa posição. A correspondência é feita *pixel* por *pixel*.

Ao utilizarmos *template matching* em imagens em escala de cinza o nível de correspondência aumenta quase a perfeição.

2.3.6.1.1.1 Distância Euclidiana

Calculamos a distância Euclidiana para identificar diferenças entre o *template* e a imagem. Dada uma imagem em escala de cinza e g seja o valor de cinza do *template* de tamanho $n \times m$, a distância é dada pela Equação 2.5.

$$d(I, g, r, c) = \sqrt{\sum_{i=1}^n \sum_{j=1}^m (I(r+i, c+j) - g(i, j))^2}$$

Equação 2.5

Nesta fórmula r e c denotam o canto superior do *template* g .

2.3.6.1.1.2 Correlação

A correlação é uma medição do grau de igualdade de duas variáveis, não necessariamente no valor real, mas no comportamento em geral. As duas variáveis são os valores de *pixels* correspondentes em duas imagens, *template* e origem [30].

Fórmula da Correlação para Escala de cinza

$$cor = \frac{\sum_{i=0}^{N-1} (x_i - \bar{x}) \cdot (y_i - \bar{y})}{\sqrt{\sum_{i=0}^{N-1} (x_i - \bar{x})^2 \cdot \sum_{i=0}^{N-1} (y_i - \bar{y})^2}}$$

fórmulaEquação 2.6

x é o nível de cinza da imagem *template*

\bar{x} é média do nível de cinza da imagem *template*

y é a seção na imagem origem

\bar{y} é a média do nível de cinza da imagem origem

N é o número de *pixels* na seção da imagem

($N =$ tamanho da imagem *template* = colunas * linhas)

O valor de cor será um valor entre -1 e +1, quanto maior o valor maior a correspondência entre as duas imagens.

Exemplo:



Figura 6: Imagem origem



Figura 7:
Template

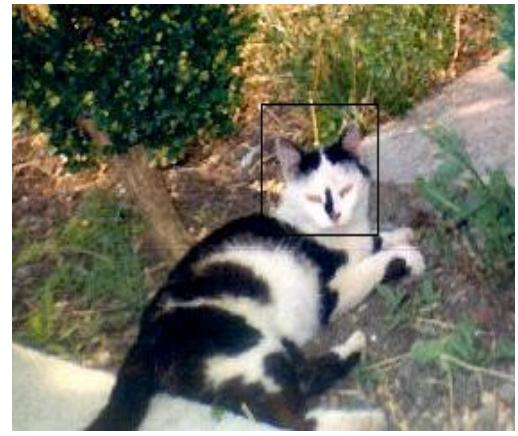


Figura 8: Resultado

2.3.6.1.2 RASTREAMENTO

O processo de rastreamento é um processo de reconhecer um padrão em uma sequência de imagens. O rastreamento poderia ser feito desta forma, porém, a busca em cada imagem de uma sequência sem o uso de qualquer conhecimento específico é relativamente lenta. Os processos de rastreamento atrelam um conhecimento sobre o movimento do objeto que está sendo rastreado para minimizar a busca entre as imagens em uma sequência. Os processos de rastreamento podem ser aplicados em diversas áreas, indo de sistemas de

segurança/vigilância até o uso em sistemas de interface humano – computador, por exemplo. Existem métodos para se prever a posição do objeto quadro a quadro, indo de filtros Kalman [26], até processos com filtros de partículas [27]. No OpenCV as técnicas de rastreamento incluem dois componentes principais: identificação de objetos e modelagem da trajetória. Existem algumas funções que são utilizadas para o rastreamento, baseadas nos algoritmos de *mean shift* e *camshift*.

2.3.6.1.2.1 Mean shift

É uma técnica de análise espacial não paramétrica. Cujos domínios de aplicação incluem *clusterização* em visão computacional e processamento de imagens.

O procedimento *Mean Shift* foi originalmente apresentado em 1975 por Fukunaga e Hostetler.

O algoritmo *Mean Shift* opera em distribuições de probabilidade. Para rastrear objetos coloridos em sequências de quadro de vídeo, os dados da cor da imagem tem de ser representados como uma distribuição de probabilidade. Nós usamos os histogramas de cores para fazer isso. Distribuições de cores derivadas de sequências de imagem de vídeo mudam ao longo do tempo, assim o algoritmo *Mean Shift* tem de ser modificado para adaptar-se dinamicamente à distribuição de probabilidade de rastreamento. O novo algoritmo que preenche todos esses requisitos é chamado *CAMSHIFT* [5].

2.3.6.1.2.2 Camshift

Utilizamos este algoritmo para encontrar um modo de distribuição de cores em uma cena de vídeo. Portanto, o algoritmo *Mean Shift* é modificado para lidar com a alteração dinâmica de distribuições de probabilidade de cor derivadas de sequências de quadro de vídeo. “O algoritmo modificado é chamado *Continuously Adaptive Mean Shift* (*CAMSHIFT*). A precisão do monitoramento *CAMSHIFT* é

comparado com um *tracker* Polhemus. Tolerância à ruído, distrações e desempenho são estudados. O CAMSHIFT é então utilizado como uma interface de computador para controlar jogos de computador comerciais e para explorar os imersivos mundos dos gráficos 3D” [5].

CAMSHIFT (*Continuously Adaptive Mean Shift*), em português, Transformação Contínua de Adaptação, é um algoritmo que para cada quadro do vídeo capturado, recebe a imagem bruta e a converte para uma distribuição probabilística de cores da imagem, usando um modelo de histograma da cor a ser rastreada [5]. Ele opera basicamente, se valendo de uma busca inicial, para ajustar uma janela para procura. Nesta janela, ele busca pela distribuição provável de cor, que mais se aproxima do padrão, previamente estabelecido. Após ter reconhecido um provável padrão, ele focaliza as buscas apenas no rastreio deste. Por isso, fazendo ajustes de tamanho máximo da janela de pesquisa, e do desvio máximo permitido, ele é capaz de rastrear com perfeição o objetivo. Além disso, internamente o CAMSHIFT faz a conversão do sistema de cores captado (RGB) para HSV, pois neste é separado a matiz da saturação, reduzindo problemas de intensidade luminosa.

Aumentando o contraste da imagem em HSV e mudando alguns parâmetros da janela de procura, obtém-se uma detecção probabilística mais pura, ou seja, apenas os *pixels* relevantes são destacados, possibilitando ajustar uma área de interesse com foco no alvo, que é a mão. Assim, fazendo os ajustes necessários ele pode definir esta área com muito mais precisão. Com esta área configurada, o reconhecimento de imagens é facilitado. Seu custo computacional é relativamente baixo e é bastante preciso, sendo por estas razões escolhido como método para o rastreio do padrão de cores. Após esta fase é necessário que se faça uma conversão da imagem para uma escala de cinza, facilitando os processamentos posteriores [8].

2.3.6.2 Fluxo óptico

Este tipo de técnica possibilita a identificação de movimento entre sequências de quadros sem que se conheça a *priori* o conteúdo destes. Tipicamente, o movimento em si indica que algo de interesse está acontecendo. O OpenCV possui funções que implementam técnicas de detecção de movimento esparsas e densas. Algoritmos de natureza esparsa consideram algum conhecimento prévio sobre os pontos que se deseja rastrear, como por exemplo as *bordas*. Os algoritmos densos, por sua vez, associam um vetor de velocidade ou de deslocamento a cada pixel na imagem, sendo, portanto desnecessário o conhecimento prévio de pontos específicos da imagem. Para a maioria das aplicações práticas, entretanto, as técnicas densas possuem um custo de processamento muito alto, sendo preferíveis, portanto, as técnicas esparsas.

Todas as técnicas apresentadas neste capítulo constituem os fundamentos e as operações base para o desenvolvimento dos mais diversos aplicativos na área de visão computacional.

3 DESENVOLVIMENTO

Foi escolhida a linguagem python para o desenvolvimento dos *scripts* devido à sua forte ligação com C. Foi utilizada com a biblioteca gráfica OpenCV da Intel que foi desenvolvida para C. Neste capítulo explicaremos os *scripts* desenvolvidos a partir dos exemplos disponíveis na biblioteca OpenCV, que implementam as técnicas de processamento de imagem apresentadas neste projeto.

Os *scripts* desenvolvidos que implementam os conceitos de visão computacional, foram baseados nos conceitos de reconhecimento de padrões.

O *hardware* utilizado no desenvolvimento e experimentos dos *scripts* foi um *notebook* com 1 processador Core 2 Duo, 4 GB de memória *ram* e placa de vídeo *on-board Mobile GM45 Integrated Graphics - Intel*.

3.1 IMPLEMENTANDO CONCEITOS DE PROCESSAMENTO DE IMAGENS

Nesta seção apresentaremos as implementações das técnicas de processamento de imagem.

3.1.1 Análise de Histograma

Os histogramas são normalmente representados por um gráfico de barras que fornece para cada nível de cinza o número (ou o percentual) de *pixels* correspondentes na imagem. Através da visualização do histograma de uma imagem obtemos uma indicação de sua qualidade quanto ao nível de contraste e quanto ao seu brilho médio (se a imagem é predominantemente clara ou escura) [25].

Ao executarmos o *script* “demhist.py”, desenvolvido a partir de exemplos do manual do OpenCV, podemos analisar o histograma da imagem quando a

imagem sofre alterações em contraste e em brilho. A funcionalidade deste *script* é reproduzir o gráfico de histograma para uma determinada imagem, assim que, a mesma sofra alterações nas suas propriedades de brilho e contraste.

Na Figura 9 temos a função responsável por modificar o histograma exibido de acordo com a variação de contraste e/ou brilho.

As alterações feitas ao aumentarmos ou diminuirmos tanto no contraste, quanto no brilho da imagem, são processadas e exibidas em tempo real.

Usamos a função `cv.CalcArrHist` que calcula o histograma da imagem e o transforma em valores de um *array*. E a função `cv.GetMinMaxHistValue` que nos retorna o máximo e o mínimo valores do histograma.

Arredondando e escalonando estes valores conseguimos obter a representação gráfica do histograma.

```
def update_brightcont(self):
    # The algorithm is by Werner D. Streidt
    # (http://visca.com/ffactory/archives/5-99/msg00021.html)
    if self.contrast > 0:
        delta = 127. * self.contrast / 100
        a = 255. / (255. - delta * 2)
        b = a * (self.brightness - delta)
    else:
        delta = -128. * self.contrast / 100
        a = (256. - delta * 2) / 255.
        b = a * self.brightness + delta

    cv.ConvertScale(self.src_image, self.dst_image, a, b)
    cv.ShowImage("image", self.dst_image)

    cv.CalcArrHist([self.dst_image], self.hist)
    (min_value, max_value, _, _) = cv.GetMinMaxHistValue(self.hist)
    cv.Scale(self.hist.bins, self.hist.bins, float(self.hist_image.height) / max_value, 0)

    cv.Set(self.hist_image, cv.ScalarAll(255))
    bin_w = round(float(self.hist_image.width) / hist_size)

    for i in range(hist_size):
        cv.Rectangle(self.hist_image, (int(i * bin_w), self.hist_image.height),
                    (int((i + 1) * bin_w), self.hist_image.height - cv.Round(self.hist.bins[i])),
                    cv.ScalarAll(0), -1, 8, 0)

    cv.ShowImage("histogram", self.hist_image)
```

Figura 9: Trecho do *script* demhist.py.

Na Figura 10 podemos observar o *script* em execução, e ver o histograma resultante das variações de brilho e contraste.

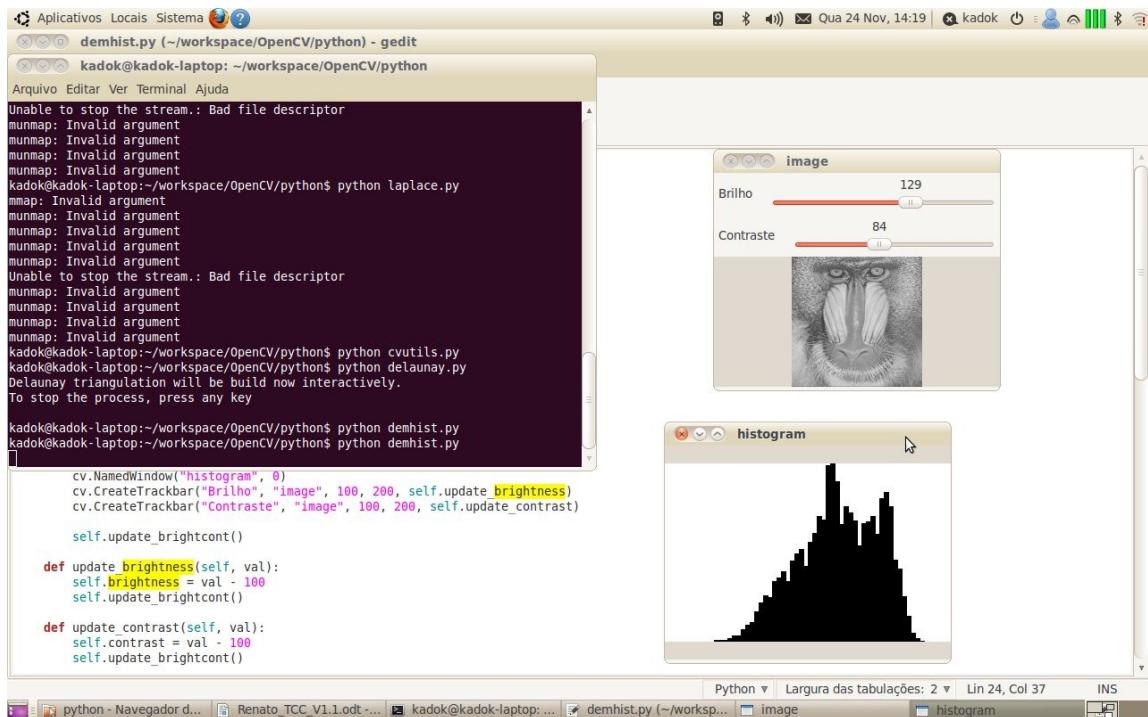


Figura 10: Histograma

3.1.2 Análise de binarização

A forma mais simples de limiarização consiste na bipartição do histograma, convertendo os *pixels* cujo tom de cinza é maior ou igual a um certo valor de limiar (T) em brancos e os demais em pretos. No caso de níveis de cinza divididos basicamente em duas classes, onde o histograma apresenta dois picos e um vale, a limiarização é trivial [25].

Ao executarmos o *script* “edge.py”, desenvolvido a partir de exemplos do manual do OpenCV, podemos analisar a aplicação da binarização sobre a imagem. A funcionalidade deste *script* é aplicar binarização sobre uma imagem, para funcionar como um detector de bordas.

Na Figura 11 temos a função de desenho que vai identificar as bordas e desenhá-las, o detector de bordas funciona em escala de cinza.

```

# the callback on the trackbar
def on_trackbar(position):

    cv.Smooth(gray, edge, cv.CV_BLUR, 3, 3, 0)
    cv.Not(gray, edge)

    # run the edge detector on gray scale
    cv.Canny(gray, edge, position, position * 3, 3)

    # reset
    cv.SetZero(col_edge)

    # copy edge points
    cv.Copy(im, col_edge, edge)

    # show the im
    cv.ShowImage(win_name, col_edge)

```

Figura 11: Trecho do script *edge.py*

Na Figura 12 podemos observar o resultado da execução do script de binarização, que efetua a detecção de bordas.

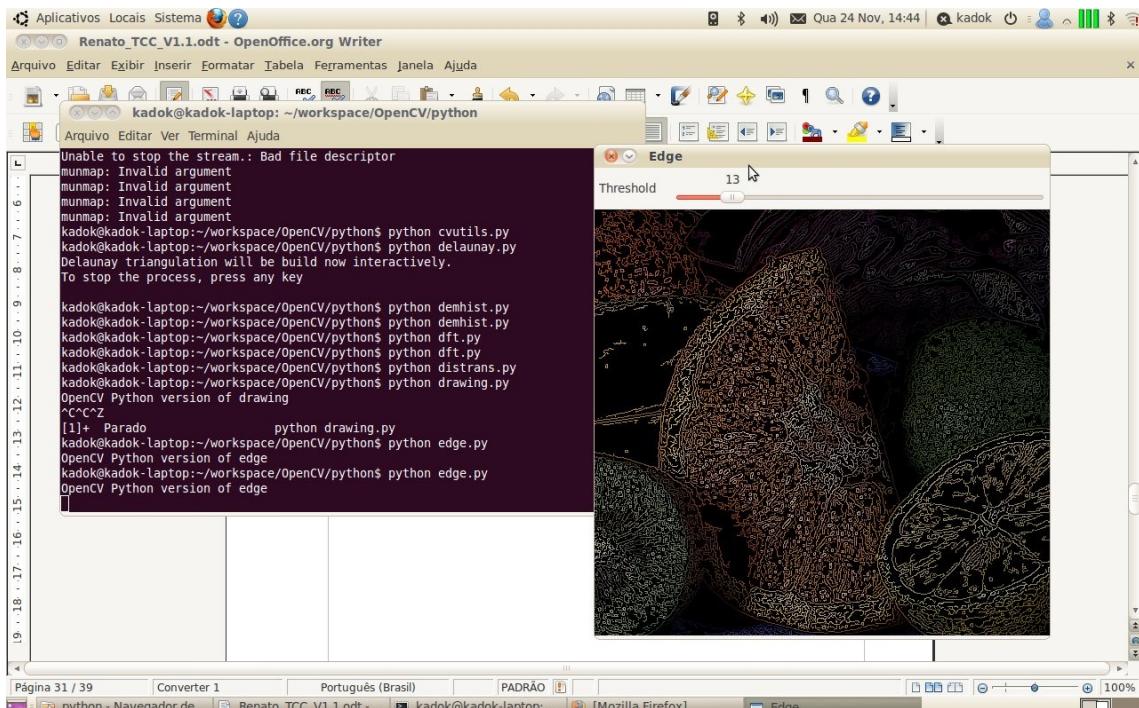


Figura 12: Binarização

3.1.3 Erosão e dilatação

As operações básicas morfológicas de erosão e dilatação, produzem resultados contrastantes quando aplicadas a qualquer escala de cinza ou imagens binárias. Erosão encolhe objetos de imagem, enquanto a dilatação expande-os.

A erosão geralmente diminui o tamanho dos objetos e remove pequenas anomalias subtraindo objetos com um raio menor que o elemento estruturante. Com tons de cinza, a erosão reduz o brilho (e consequentemente o tamanho) de objetos brilhantes sobre um fundo escuro. Com imagens binárias, a erosão remove completamente objetos menores do que o elemento estruturante e remove os *pixels* do perímetro de objetos da imagem maior.

A dilatação geralmente aumenta o tamanho dos objetos, o preenchimento de buracos e áreas quebradas e conecta áreas que estão separadas por espaços menores que o tamanho do elemento estruturante. Com tons de cinza, a dilatação aumenta o brilho dos objetos. Com imagens binárias, a dilatação conecta áreas que estão separadas por espaços menores que o elemento estruturante e adiciona *pixels* no perímetro de cada objeto da imagem.

Ao executarmos o *script* *morphology.py* podemos analisar a aplicação da Erosão e da Dilatação sobre a imagem. A funcionalidade deste *script* é aplicar a uma determinada imagem as operações morfológicas de erosão e dilatação combinadas.

Na Figura 21 podemos analisar um trecho do código do *script* *morphology.py*, neste trecho temos as funções que irão fazer a transformação da imagem através da aplicação de erosão pela função *Erosion* que aplica a função *cv.Erode* e a aplicação de dilatação pela função *Dilatation* que aplica a função *cv.Dilate* (Os nomes *Erosion* e *Dilatation* são apenas os nomes das funções criadas sem nenhum significado).

```

element_shape = cv.CV_SHAPE_RECT

def Opening(pos):
    element = cv.CreateStructuringElementEx(pos*2+1, pos*2+1, pos, pos, element_shape)
    cv.Erode(src, image, element, 1)
    cv.Dilate(image, dest, element, 1)
    cv.ShowImage("Opening & Closing", dest)

def Closing(pos):
    element = cv.CreateStructuringElementEx(pos*2+1, pos*2+1, pos, pos, element_shape)
    cv.Dilate(src, image, element, 1)
    cv.Erode(image, dest, element, 1)
    cv.ShowImage("Opening & Closing", dest)

def Erosion(pos):
    element = cv.CreateStructuringElementEx(pos*2+1, pos*2+1, pos, pos, element_shape)
    cv.Erode(src, dest, element, 1)
    cv.ShowImage("Erosao & Dilacao", dest)

def Dilation(pos):
    element = cv.CreateStructuringElementEx(pos*2+1, pos*2+1, pos, pos, element_shape)
    cv.Dilate(src, dest, element, 1)
    cv.ShowImage("Erosao & Dilacao", dest)

```

Figura 13: Trecho do script morphology.py

Nas Figuras 14 e 15 podemos ver o resultado destas transformações, tanto para Erosão quanto para Dilatação respectivamente.

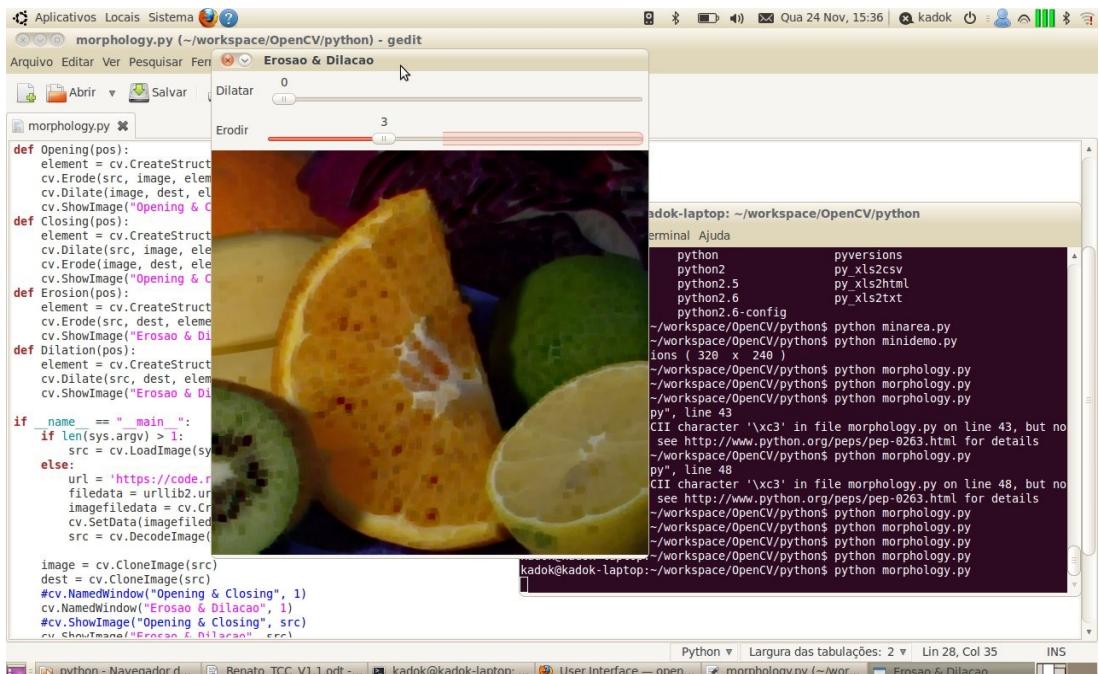


Figura 14: Erosão

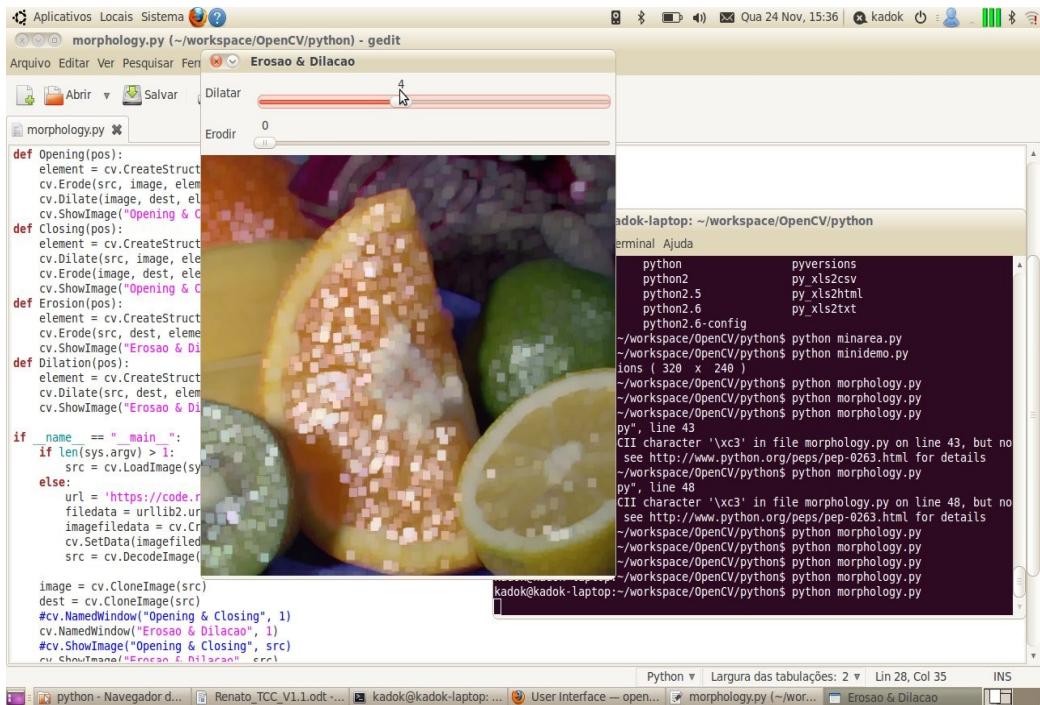


Figura 15: Dilatação

3.1.4 Filtro Laplace

Este filtro detecta bordas na imagem usando o método laplaciano, que produz bordas finas da largura de um pixel.

Ao executarmos o script “laplace.py” capturamos imagem da câmera e já a transformamos usando o efeito de bordas com filtro Laplace em tempo real.

Na Figura 16 temos um trecho do código do script “*laplace.py*”, analisando o fluxo de sucesso.

Inicialmente capturamos a imagem da câmera, criamos uma janela onde mostraremos a imagem capturada. Pegamos os quadros da captura e aplicamos a função `cv.Laplace` que é executada durante toda a captura. Assim, quadro a quadro temos em tempo real a aplicação do filtro Laplace na imagem capturada da câmera.

```

if len(sys.argv) == 1:
    capture = cv.CreateCameraCapture(0)
elif len(sys.argv) == 2 and sys.argv[1].isdigit():
    capture = cv.CreateCameraCapture(int(sys.argv[1]))
elif len(sys.argv) == 2:
    capture = cv.CreateFileCapture(sys.argv[1])

if not capture:
    print "Could not initialize capturing..."
    sys.exit(-1)

cv.NamedWindow("Laplacian", 1)

while True:
    frame = cv.QueryFrame(capture)
    if frame:
        if not laplace:
            planes = [cv.CreateImage((frame.width, frame.height), 8, 1) for i in range(3)]
            laplace = cv.CreateImage((frame.width, frame.height), cv.IPL_DEPTH_16S, 1)
            colorlaplace = cv.CreateImage((frame.width, frame.height), 8, 3)

            cv.Split(frame, planes[0], planes[1], planes[2], None)
            for plane in planes:
                cv.Laplace(plane, laplace, 3)
                cv.ConvertScaleAbs(laplace, plane, 1, 0)

            cv.Merge(planes[0], planes[1], planes[2], None, colorlaplace)

            cv.ShowImage("Laplacian", colorlaplace)

        if cv.WaitKey(10) != -1:
            break

cv.DestroyWindow("Laplacian")

```

Ilustração 16: Trecho do script laplace.py

Na Figura 17 podemos ver o resultado após a execução do script.

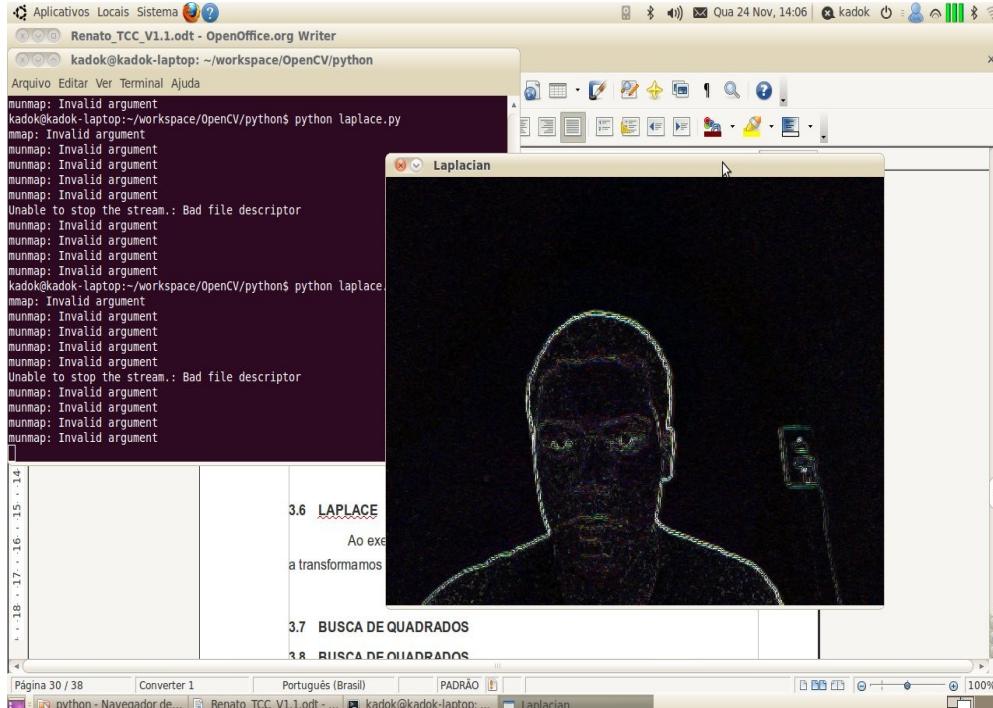


Figura 17: Laplace

3.2 IMPLEMENTANDO CONCEITOS DE VISÃO COMPUTACIONAL

Nesta seção apresentaremos as implementações de aplicações em visão computacional com ênfase em reconhecimento de padrões.

3.2.1 Busca de quadrados

Ao executarmos o *script* “squares.py” iniciamos a detecção de quadrados na imagem. Este *script* faz uma busca na imagem para identificar formas quadráticas e marcá-las na imagem.

A função na Figura 18 é a responsável pela identificação de quadrados na imagem. Ela se baseia na busca de contornos para encontrar quadrados em imagens binárias retornando quadriláteros de lados iguais.

```
def find_squares_from_binary( gray ):
    """
    use contour search to find squares in binary image
    returns list of numpy arrays containing 4 points
    """
    squares = []
    storage = cv.CreateMemStorage(0)
    contours = cv.FindContours(gray, storage, cv.CV_RETR_TREE, cv.CV_CHAIN_APPROX_SIMPLE, (0,0))
    storage = cv.CreateMemStorage(0)
    while contours:
        #approximate contour with accuracy proportional to the contour perimeter
        arclength = cv.ArcLength(contours)
        polygon = cv.ApproxPoly( contours, storage, cv.CV_POLY_APPROX_DP, arclength * 0.02, 0)
        if is_square(polygon):
            squares.append(polygon[0:4])
        contours = contours.h_next()

    return squares
```

Figura 18: Trecho do *script* squares.py.

Na Figura 19 podemos ver a imagem após o processo de detecção.

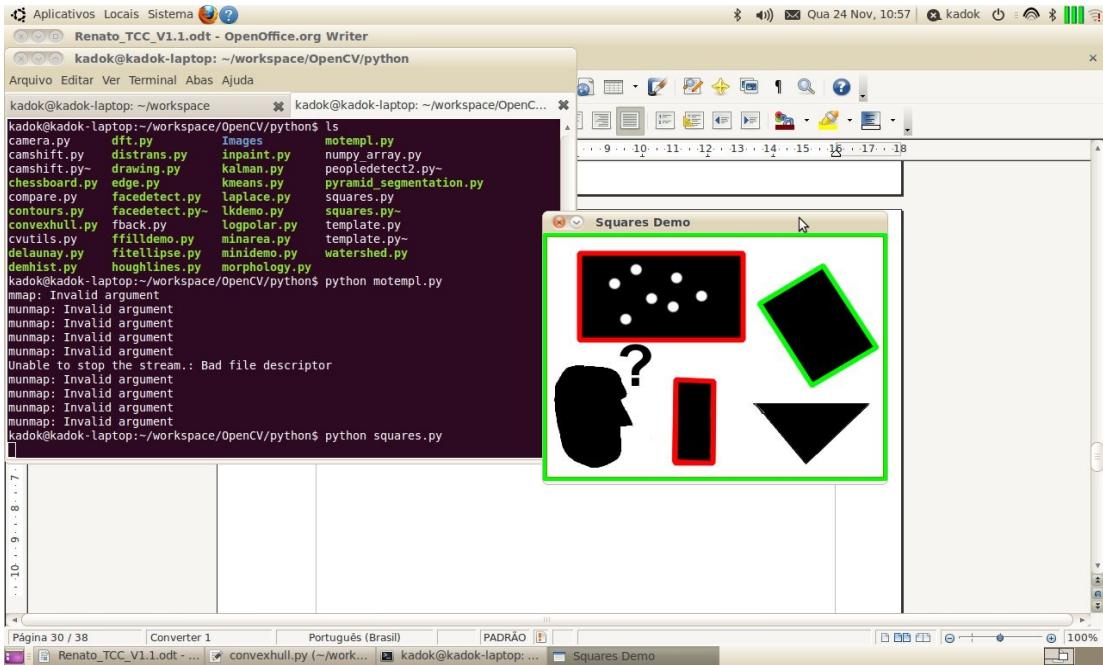


Figura 19: Quadrados

3.2.2 Implementando Template matching

Para a análise de *template matching* usaremos três *scripts* que seguirão diferentes formas de implementação e visualização da comparação, assim, poderemos fazer um comparativo sobre a comparação usando com OpenCV e sem OpenCV.

Ao executarmos o *script* `template.py` é feita a comparação e nos retorna as etapas do processo.

Na Figura 20 podemos analisar um trecho de código do *script* `template.py`, comparamos duas imagens, a imagem origem e a imagem *template*

com a função `cv.MatchTemplate` que irá percorrer a imagem origem procurando a imagem *template* pelos 6 métodos diferentes retornando o melhor resultado de comparação para cada método. As equações dos métodos estão na Figura 21. Após encontrarmos o melhor resultado para cada método, normalizamos estes resultados através da função `cv.Normalize` e exibimos as imagens resultantes.

```
# Allocate Output Images:  
wi = src.width;  
wii = templ.width;  
iwidth = wi - wii +1;  
he = src.height;  
hei = templ.height;  
iheight = he - hei +1;  
sz = (iwidth, iheight);  
ftmp = [1,2,3,4,5,6];  
  
i = 1;  
while i < 7:  
    ftmp[i-1]= cv.CreateImage( sz, 32, 1 );  
    i = i + 1;  
  
# Do the matching of the template with the image  
i = 1;  
while i < 7:  
    cv.MatchTemplate( src, templ, ftmp[i-1], i-1 );  
    cv.Normalize( ftmp[i-1], ftmp[i-1], 1, 0, cv.CV_MINMAX );  
    i = i + 1;
```

Figura 20: Trecho do script *template.py*

- method=CV_TM_SQDIFF

$$R(x, y) = \sum_{x', y'} (T(x', y') - I(x + x', y + y'))^2$$

- method=CV_TM_SQDIFF_NORMED

$$R(x, y) = \frac{\sum_{x', y'} (T(x', y') - I(x + x', y + y'))^2}{\sqrt{\sum_{x', y'} T(x', y')^2 \cdot \sum_{x', y'} I(x + x', y + y')^2}}$$

- method=CV_TM_CCORR

$$R(x, y) = \sum_{x', y'} (T(x', y') \cdot I(x + x', y + y'))$$

- method=CV_TM_CCORR_NORMED

$$R(x, y) = \frac{\sum_{x', y'} (T(x', y') \cdot I'(x + x', y + y'))}{\sqrt{\sum_{x', y'} T(x', y')^2 \cdot \sum_{x', y'} I'(x + x', y + y')^2}}$$

- method=CV_TM_CCOEFF

$$R(x, y) = \sum_{x', y'} (T'(x', y') \cdot I(x + x', y + y'))$$

Onde

$$\begin{aligned} T'(x', y') &= T(x', y') - 1/(w \cdot h) \cdot \sum_{x'', y''} T(x'', y'') \\ I'(x + x', y + y') &= I(x + x', y + y') - 1/(w \cdot h) \cdot \sum_{x'', y''} I(x + x'', y + y'') \end{aligned}$$

- method=CV_TM_CCOEFF_NORMED

$$R(x, y) = \frac{\sum_{x', y'} (T'(x', y') \cdot I'(x + x', y + y'))}{\sqrt{\sum_{x', y'} T'(x', y')^2 \cdot \sum_{x', y'} I'(x + x', y + y')^2}}$$

Figura 21: Equações dos métodos para cálculo de template matching em OpenCV.

Na Figura 22 podemos ver que todos os resultados para cada equação de comparação são exibidos como imagem. Este é um método bem eficiente, pois é ágil quanto ao tempo de comparação.

Podemos observar também que quando utilizamos imagens em escala de cinza o tempo de processamento de comparação torna-se menor e que a resolução da imagem também é um fator importante em relação ao tempo de processamento, pois quanto maior a resolução mais demorada será a comparação.

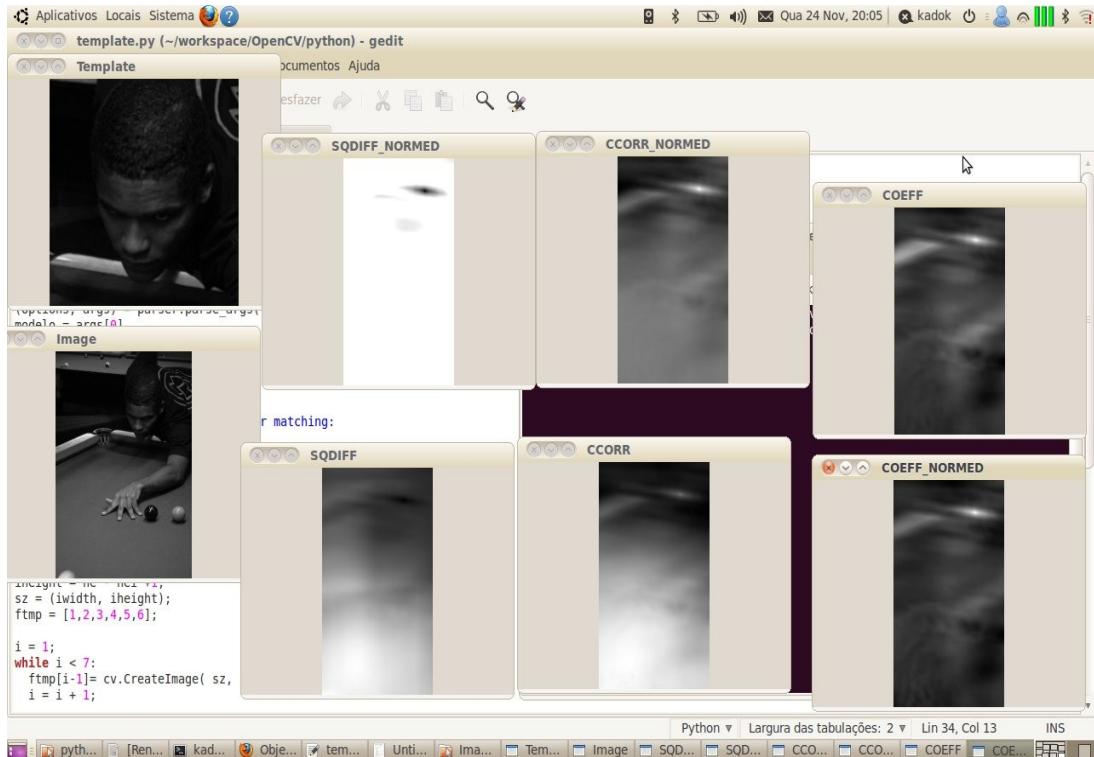


Figura 22: *Template matching* - etapas

Ao executarmos o script `compare.py` é feita a comparação e nos retorna o valor obtido de semelhança.

Este *script* é a base para o desenvolvimento do *script* anterior. A diferença entre os dois é que não analisamos todos os métodos, para este *script* foi escolhido o CV_TM_CCOEFF_NORMED e não retornamos a imagem como resultado e sim apenas o valor da comparação. O tempo de processamento deste *script* é menor que o anterior, por processar apenas um método.

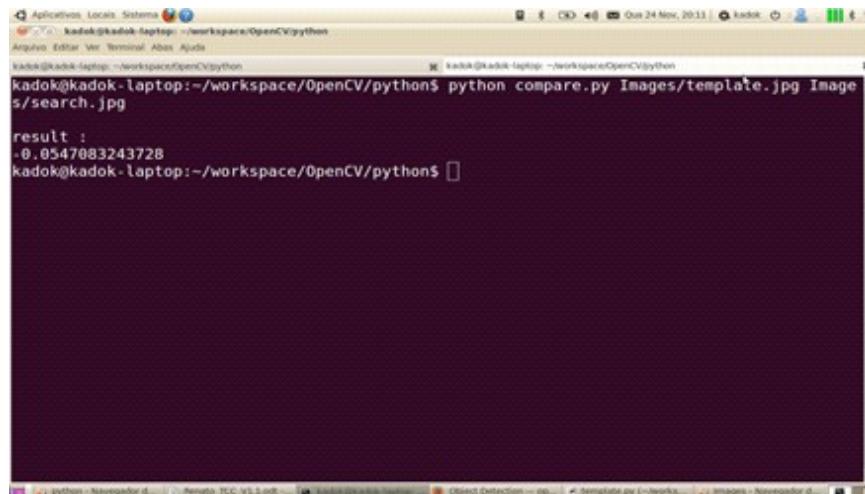


Figura 23: Template matching - resultado

Ao executarmos o *script* match.py não utilizaremos OpenCV, neste *script* utilizaremos a biblioteca PIL, para ver a diferença de técnicas de comparação. Neste *script* após o processamento da comparação é gerada uma imagem com o resultado da busca.

Na Figura 24 podemos analisar o trecho do código do *script* match.py, podemos analisar a função machTemplate, que recebe como parâmetro a imagem origem e a imagem *template*. Esta função executa o papel de um robô que percorre a imagem origem pixel a pixel comparando com a imagem *template*. Após encontrada na imagem origem. Cria-se uma imagem com fundo verde representando a imagem origem e colamos a imagem *template* na posição encontrada na imagem origem, para maior destaque.

```

def matchTemplate(searchImage, templateImage):
    minScore = -1000
    matching_xs = 0
    matching_ys = 0
    searchWidth = searchImage.size[0]
    searchHeight = searchImage.size[1]
    templateWidth = templateImage.size[0]
    templateHeight = templateImage.size[1]
    searchIm = searchImage.load()
    templateIm = templateImage.load()
    #loop over each pixel in the search image
    for xs in range(searchWidth-templateWidth+1):
        for ys in range(searchHeight-templateHeight+1):
            #for ys in range(10):
            #set some kind of score variable to 0
            score = 0
            #loop over each pixel in the template image
            for xt in range(templateWidth):
                for yt in range(templateHeight):
                    score += 1 if searchIm[xs+xt,ys+yt] == templateIm[xt, yt] else -1

            if minScore < score:
                minScore = score
                matching_xs = xs
                matching_ys = ys

    print "Location=", (matching_xs, matching_ys), "Score=", minScore
    im1 = Image.new('RGB', (searchWidth, searchHeight), (80, 147, 0))
    im1.paste(templateImage, ((matching_xs), (matching_ys)))
    #searchImage.show()
    #im1.show()
    im1.save('Images/template_matched_in_search.png')

```

Figura 24: Trecho do *script* match.py

Na Figura 25 podemos ver o resultado da execução do *script* match.py, a imagem *template* em destaque na imagem verde de fundo representando a imagem origem.

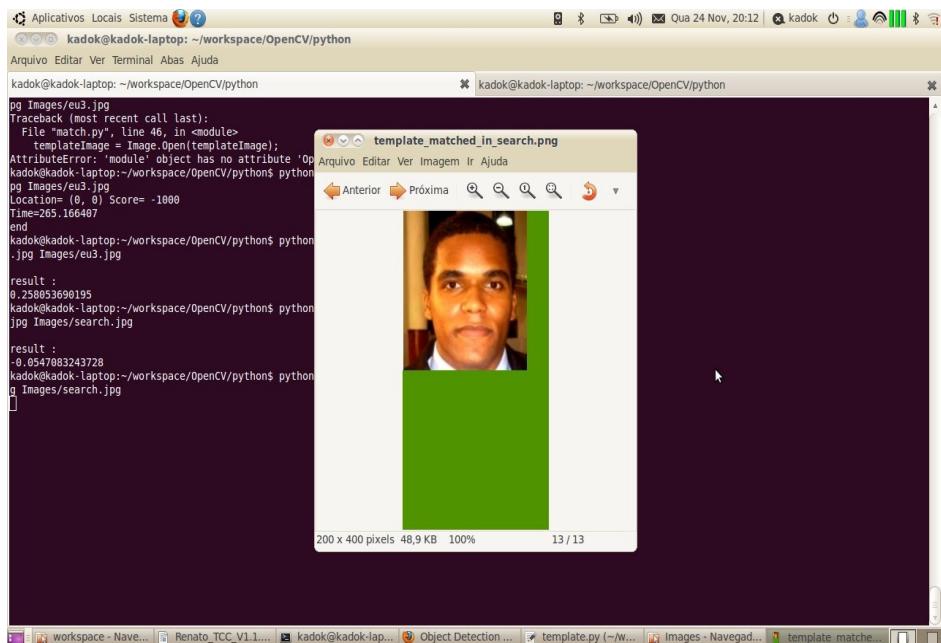


Figura 25: *Template matching* - imagem resultante

Podemos observar também que quando utilizamos imagens em escala de cinza o tempo de processamento de comparação torna-se menor, já que cada pixel terá uma combinação de duas cores possíveis (preto ou branco), assim sendo, será necessário comparar apenas duas cores, o que reduz o tempo de comparação. A resolução da imagem também é um fator importante em relação ao tempo de processamento, pois quanto maior a resolução mais demorada será a comparação, porque a resolução da imagem é definida pelo sua quantidade de *pixels*, ou seja, quanto maior a quantidade de *pixels* maior a resolução da imagem. Como analisamos pixel a pixel, quanto menor o número de *pixels* à comparar menor será o tempo de comparação.

Porém, ao compararmos este *script* aos que utilizam OpenCV, pode-se concluir que o tempo de comparação deste *script* é muito mais alto que os outros, sendo este um *script* que simplesmente compara pixel por pixel, como um simples robô, já os que utilizam a função da biblioteca tem meios mais eficientes de redução do tempo de busca.

3.2.3 Implementando Camshift

Camshift é um algoritmo desenvolvido para o rastreamento de cor, possibilitando também, o rastreamento de faces. É baseado numa técnica estatística onde se busca o pico entre distribuições de probabilidade em gradientes de densidade. Esta técnica é chamada de “média por deslocamento” (*mean shift*) e foi adaptada no Camshift para tratar a mudança dinâmica das distribuições de probabilidade das cores numa sequência de vídeo. Pode ser usada no rastreamento de objetos e no rastreamento de faces, como descrito a seguir.

Para cada quadro, a imagem (raw) é convertida para outra de distribuição de probabilidade de cor através de um modelo de histograma da cor da pele. O centro e o tamanho da face que se quer rastrear são encontrados através do CamShift operando na imagem de probabilidade de cores. O tamanho e a localização corrente da face são informados e usados para definir o tamanho e a localização da janela de busca da próxima imagem de vídeo.

A função cvCamShift chama o algoritmo CamShift para buscar o centro, o tamanho e a orientação do objeto sendo rastreado. Esta função tem como parâmetros de entrada a retroprojeção, a janela a ser monitorada (área a ser rastreada), e o critério de busca. Este trecho do código pode ser visto na Figura 26.

```

def run(self):
    hist = cv.CreateHist([180], cv.CV_HIST_ARRAY, [(0,180)], 1 )
    backproject_mode = False
    while True:
        frame = cv.QueryFrame( self.capture )

        # Convert to HSV and keep the hue
        hsv = cv.CreateImage(cv.GetSize(frame), 8, 3)
        cv.CvtColor(frame, hsv, cv.CV_BGR2HSV)
        self.hue = cv.CreateImage(cv.GetSize(frame), 8, 1)
        cv.Split(hsv, self.hue, None, None, None)

        # Compute back projection
        backproject = cv.CreateImage(cv.GetSize(frame), 8, 1)

        # Run the cam-shift
        cv.CalcArrBackProject( [self.hue], backproject, hist )
        if self.track_window and is_rect_nonzero(self.track_window):
            crit = ( cv.CV_TERMCRIT_EPS | cv.CV_TERMCRIT_ITER, 10, 1)
            (iters, (area, value, rect), track_box) = cv.CamShift(backproject, self.track_window, crit)
            self.track_window = rect

```

Figura 26: Trecho do script camshift.py

Ao executarmos o script “camshift.py” são abertas duas janelas, uma com a imagem da câmera e a outra com o histograma de cores da imagem da câmera. Como podemos observar na Figura 27.

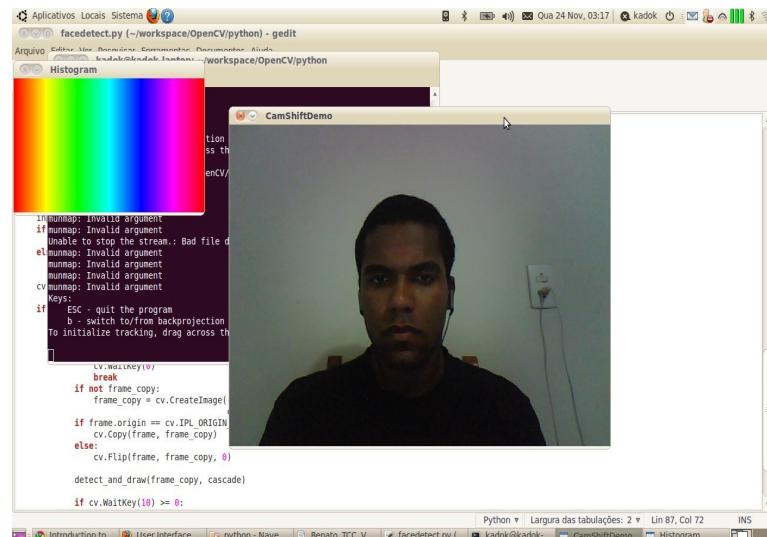


Figura 27: Camshift- Tela inicial

Selecionando uma área na imagem da câmera com o mouse, a área selecionada é rastreada em qualquer ponto da imagem da câmera. O rastreamento é feito em tempo real. Se pressionarmos a tecla “B” do teclado, poderemos ver a imagem no modo de retroprojeção, essa imagem é gerada pela razão do histograma da imagem de interesse e o histograma da cor da área a ser localizada, ambos na escala HSV. Nas Figuras 28, 29 e 30 podemos observar o rastreamento, e na Figura 31 a imagem no modo de retroprojeção.

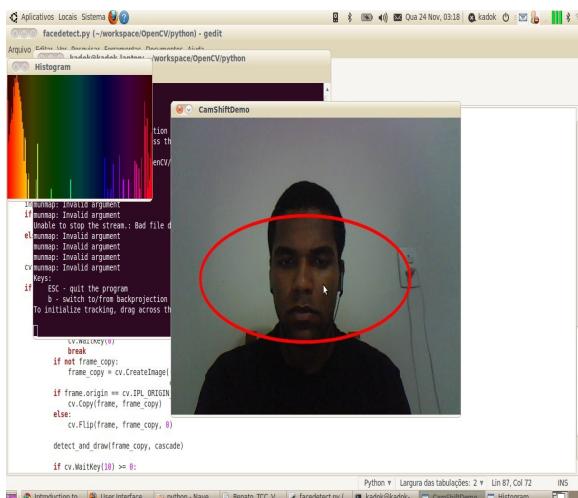


Figura 28: Camshift - Rastreamento

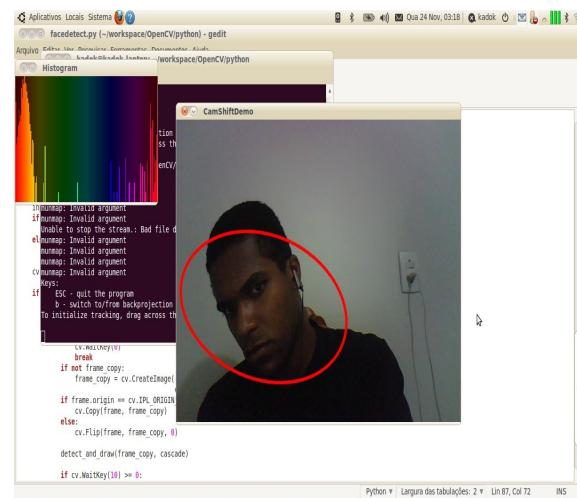


Figura 29: Camshift - Rastreamento

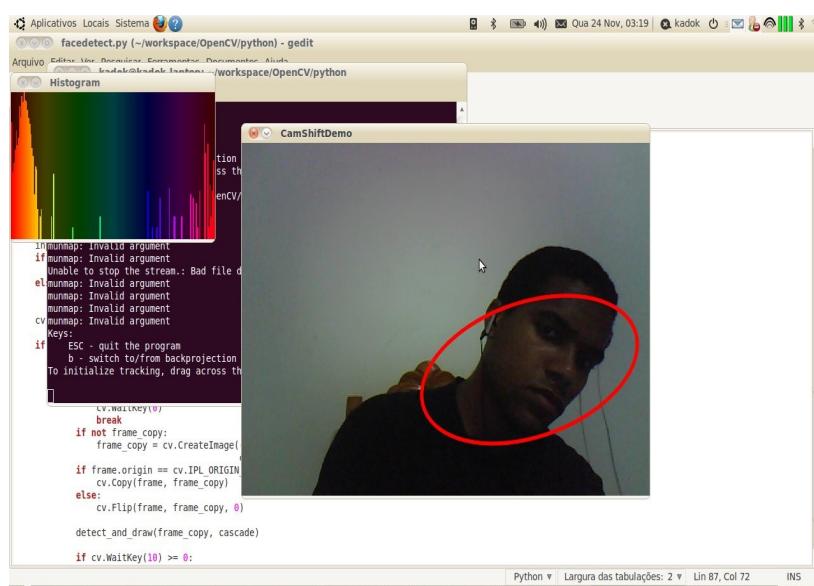


Figura 30: Camshift - Rastreamento

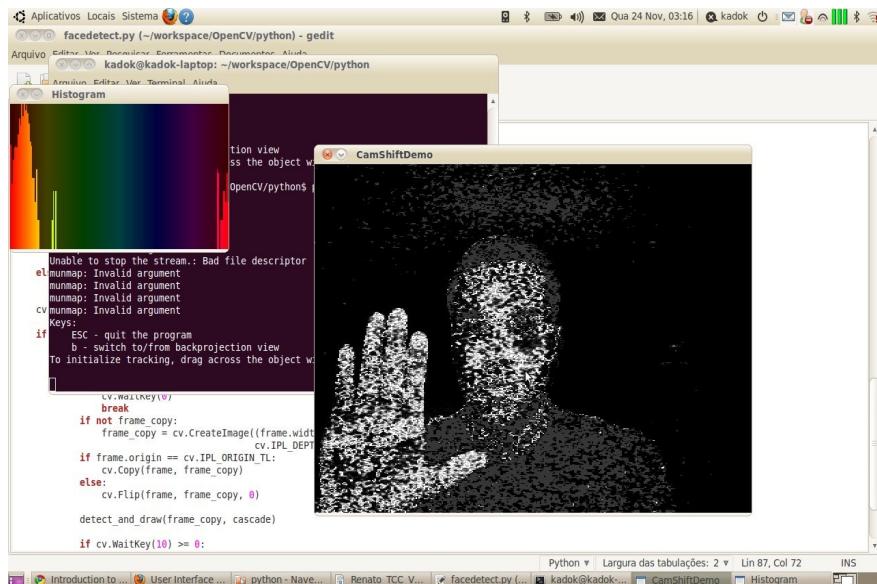


Figura 31: Camshift - Modo retroprojeção

3.2.4 Rastreamento de fluxo por pontos

Ao executarmos o script “fback.py”, capturamos a imagem da câmera quadro a quadro, analisando o momento em relação ao momento inicial. Usamos pontos fixos na imagem para indicar a posição zero para a imagem estática e quando ocorre alguma alteração (movimentação) na imagem, estes pontos geram retas para indicar a direção do movimento (fluxo), tudo em tempo real.

Na Figura 32 temos um treco do código do script fback.py. Neste trecho podemos observar que inicialmente capturamos a imagem da câmera, em seguida, vamos processando quadro a quadro. Se o quadro for o primeiro, definimos as posições iniciais das nossas imagens para comparação de momento. Se o quadro não for o primeiro fazemos a chamada da função CalcOpticalFlowFrameback, que irá calcular o quadro atual com o quadro inicial, caso haja diferença, esta função retornará as diferenças para que se possa desenhá-las através da função de desenho *draw_flow* que irá receber o sentido do fluxo e seu deslocamento em comparação ao quadro inicial. Após o desenhado o novo fluxo, tornamos a considerar o próximo quadro como primeiro quadro.

```

def run(self):
    first_frame = True

    while True:
        frame = QueryFrame( self.capture )

        if first_frame:
            gray = CreateImage(GetSize(frame), 8, 1)
            prev_gray = CreateImage(GetSize(frame), 8, 1)
            flow = CreateImage(GetSize(frame), 32, 2)
            self.cflow = CreateImage(GetSize(frame), 8, 3)

            CvtColor(frame, gray, CV_BGR2GRAY)
        if not first_frame:
            CalcOpticalFlowFarneback(prev_gray, gray, flow,
                                      pyr_scale=0.5, levels=3, winsize=15,
                                      iterations=3, poly_n=5, poly_sigma=1.2, flags=0)
            self.draw_flow(flow, prev_gray)
            c = WaitKey(7)
            if c in [27, ord('q'), ord('Q')]:
                break
            prev_gray, gray = gray, prev_gray
        first_frame = False

```

Figura 32: Trecho do script fback.py.

Na Figura 33 podemos observar o quadro inicial onde ainda não houve nenhuma movimentação.

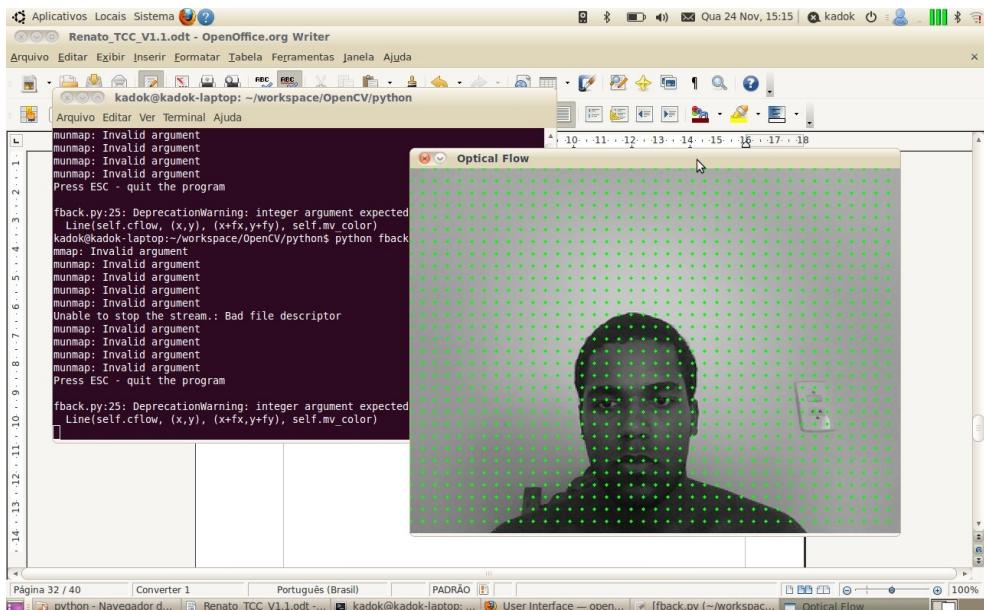


Figura 33: Fluxo Inicial

Na Figura 34 podemos observar o quadro seguinte, onde já foi realizado um movimento e o fluxo da direção e deslocamento deste movimento foram processados e desenhados em tempo real.

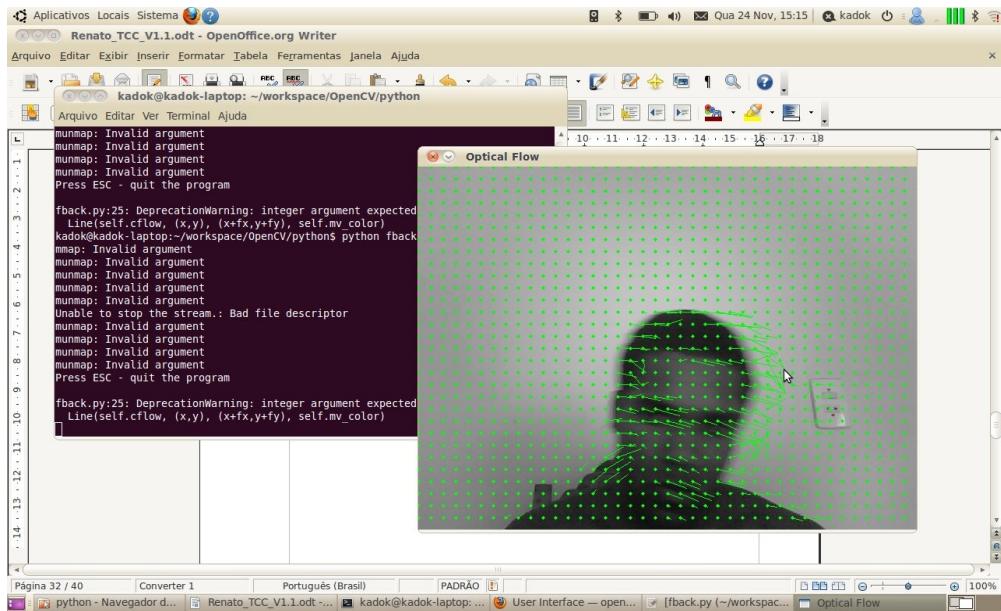


Figura 34: Fluxo pós movimentação

Durante a execução dos testes deste *script*, foi possível perceber que para que possamos executá-lo em tempo real de fato, é necessário um hardware muito poderoso pois o processamento deste *script* em tempo real torna-se inviável em hardwares obsoletos devido a sua complexidade.

3.2.5 Rastreamento de fluxo

Ao executarmos o *script* motempl.py, capturamos a imagem da câmera quadro a quadro, analisando o momento em relação ao momento inicial. Usando binarização sobre a imagem, quando ocorre alguma alteração (movimentação) na imagem, esta movimentação gera um rastro que são os momentos anteriores ao final do movimento.

Na Figura 35 podemos analisar um trecho do código do *script* motempl.py, este trecho é referente à função update_mhi que é a função responsável por desenhar o histórico e as marcações nas zonas de movimentação na imagem capturada. Recebemos um quadro ou imagem inicial, que é considerado o momento inicial, ou seja, iremos nos basear nesta imagem para compará-las com as próximas imagens. Ao recebermos a próxima imagem executamos a função cv.AbsDiff para verificar se houve alguma diferença (movimentação) em relação a imagem inicial. O conceito é o mesmo do rastreamento de fluxo por pontos, porém a implementação e o tipo de construção do fluxo são diferentes.

Após verificarmos as diferenças entre os quadros através da função cv.AbsDiff, aplicamos a limiarização (binarização) e fazemos a chamada da função cv.UpdateMotionHistory que irá criar o efeito na imagem capturada como um histórico do movimento realizado quadro a quadro. Ou seja, um fluxo do movimento realizado desde a imagem inicial até a última imagem capturada. Após este processo, aplicamos a função cv.CalcMotionGradient que irá calcular a derivação do MHI (*Motion History Image*) e a orientação do gradiente, assim atualizando o MHI.

```

if not mhi or cv.GetSize(mhi) != size:
    for i in range(N):
        buf[i] = cv.CreateImage(size, cv.IPL_DEPTH_8U, 1)
        cv.Zero(buf[i])
    mhi = cv.CreateImage(size, cv.IPL_DEPTH_32F, 1)
    cv.Zero(mhi) # clear MHI at the beginning
    orient = cv.CreateImage(size, cv.IPL_DEPTH_32F, 1)
    segmask = cv.CreateImage(size, cv.IPL_DEPTH_32F, 1)
    mask = cv.CreateImage(size, cv.IPL_DEPTH_8U, 1)

    cv.CvtColor(img, buf[last], cv.CV_BGR2GRAY) # convert frame to grayscale
    idx2 = (last + 1) % N # index of (last - (N-1))th frame
    last = idx2
    silh = buf[idx2]
    cv.AbsDiff(buf[idx1], buf[idx2], silh) # get difference between frames
    cv.Threshold(silh, silh, diff_threshold, 1, cv.CV_THRESH_BINARY) # and threshold it
    cv.UpdateMotionHistory(silh, mhi, timestamp, MHI_DURATION) # update MHI
    cv.CvtScale(mhi, mask, 255./MHI_DURATION,
                (MHI_DURATION - timestamp)*255./MHI_DURATION)
    cv.Zero(dst)
    cv.Merge(mask, None, None, None, dst)
    cv.CalcMotionGradient(mhi, mask, orient, MAX_TIME_DELTA, MIN_TIME_DELTA, 3)
    if not storage:
        storage = cv.CreateMemStorage(0)
        seq = cv.SegmentMotion(mhi, segmask, storage, timestamp, MAX_TIME_DELTA)

```

Figura 35: Trecho do *script* motempl.py

Após este processo aplicamos a função `cv.SegmentMotion` que irá encontrar todos os segmentos de movimentação e as marcações de zonas de movimentação e retorna uma estrutura do tipo `CVConnectedComp` que é representada por uma tupla para cada componente de movimentação, depois utilizamos a função `cv.CalcGlobalOrientation` para calcular a orientação global do movimento de cada componente.

Nas Figuras 36 e 37 podemos observar o resultado deste *script* para análise do fluxo gerado pela movimentação.

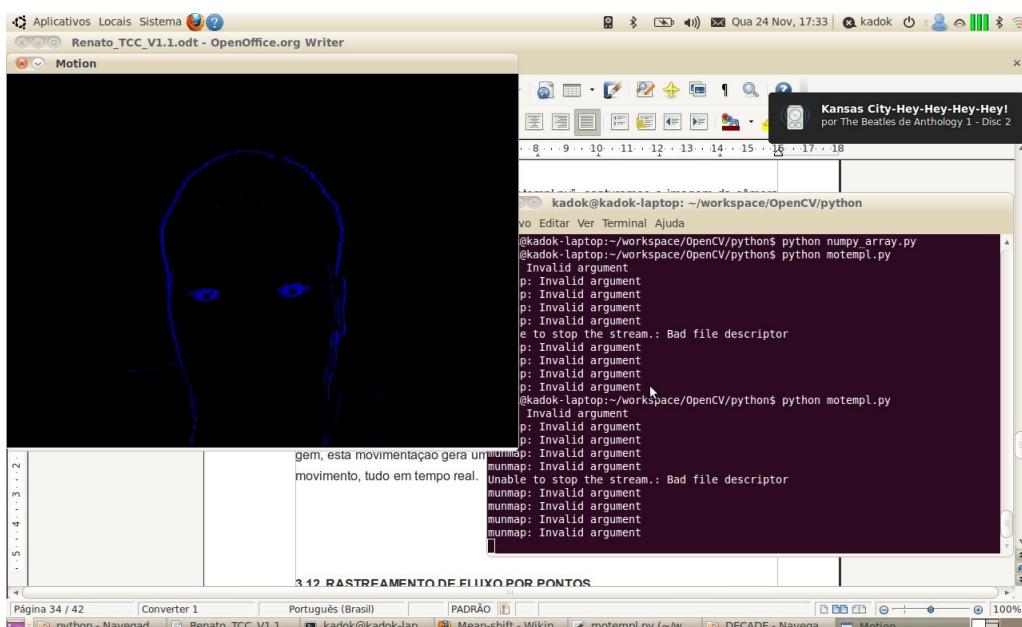


Figura 36: Fluxo - Momento inicial

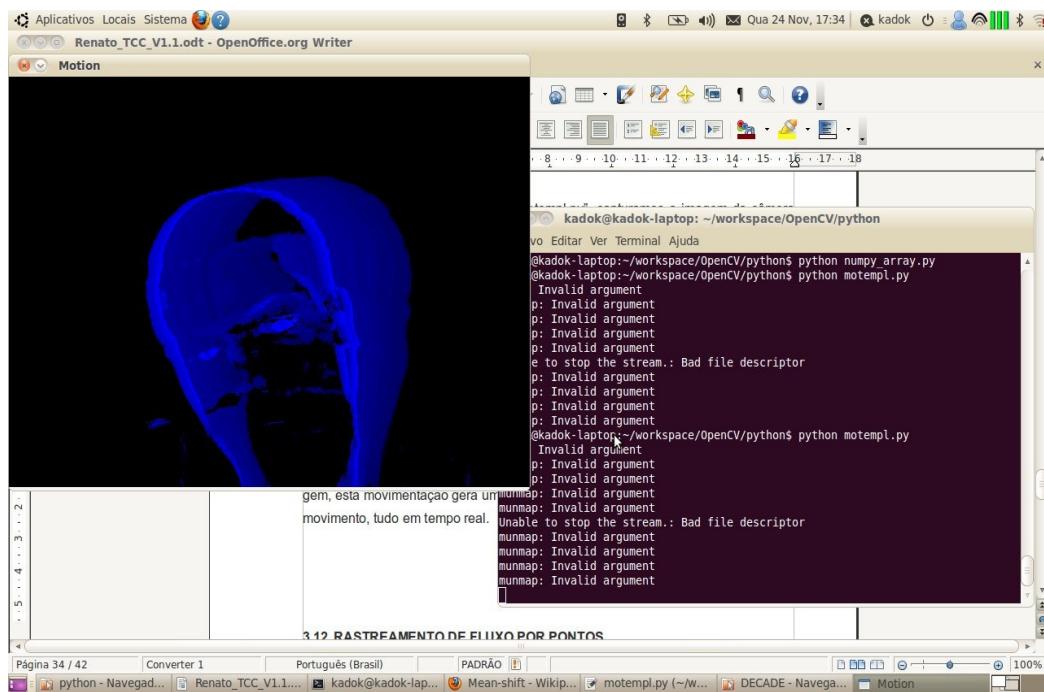


Figura 37 Fluxo - Momento pós movimentação

3.2.6 Implementando um Detector de facial

Ao executarmos o script “facedetect.py” é realizada a detecção de faces na imagem. Para o funcionamento da detecção de faces na imagem, é utilizado um mapeamento de valores que tem a grande probabilidade de resultar em faces.

Foi utilizado um arquivo XML disponibilizado pela própria biblioteca para apenas faces na posição frontal. Podemos ver um trecho deste arquivo na Figura 38.

```

<!-- tree 7 -->
<_>
<!-- root node -->
<feature>
<rects>
  <_>13 11 6 3 -1.</_>
  <_>13 12 6 1 3.</_></rects>
  <tilted>0</tilted></feature>
<threshold>-9.5083238556981087e-003</threshold>
<left_node>1</left_node>
<right_val>0.5318170189857483</right_val></_>
<_>
<!-- node 1 -->
<feature>
<rects>
  <_>12 12 6 4 -1.</_>
  <_>12 14 6 2 2.</_></rects>
  <tilted>0</tilted></feature>
<threshold>7.6601309701800346e-003</threshold>
<left_val>0.5411052107810974</left_val>
<right_val>0.2180687040090561</right_val></_></_>
<_>
<!-- tree 8 -->
<_>
<!-- root node -->
<feature>
<rects>
  <_>1 11 6 3 -1.</_>
  <_>1 12 6 1 3.</_></rects>
  <tilted>0</tilted></feature>
<threshold>7.6467678882181644e-003</threshold>
<left_node>1</left_node>
<right_val>0.1158960014581680</right_val></_>
<_>

```

Figura 38: Arquivo xml de mapeamento da face frontal

Este arquivo é composto com uma série de nós, que em XML são definidos por *tags*, na Figura 38 por exemplo, o início de um nó tem a *tag* `<feature>` e seu final é definido pela *tag* `</feature>`, entre as *tags* estão os valores, ou seja, para cada *tag*, podemos ter um valor, ou várias outras *tags* filhas com valores, a ideia é funcionar como uma lista de informações. Os nós (*nodes*) são responsáveis por guardar as informações para o mapeamento de um rosto, nestes nós temos o valor de binarização como peça chave para mapear formas.

Na Figura 39 temos o código da função responsável pela detecção facial. Analisando este código temos o seguinte fluxo:

Os parâmetros de entrada são a imagem e o arquivo xml de mapeamento. Alocamos a imagem temporariamente, convertemos a imagem para escala de cinza. Reduzimos o tamanho da imagem para agilizarmos o processamento de detecção. Se este arquivo existir então fazemos a chamada da função `cvHaarDetectObjects`, que interpreta o mapeamento do arquivo xml e faz a

detecção de objetos na imagem. No nosso caso utilizamos um arquivo para detecção de faces na posição frontal. Se alguma face for encontrada, então fazemos a chamada da função cv.Rectangle, para cada face encontrada.

```

min_size = (20, 20)
image_scale = 2
haar_scale = 1.2
min_neighbors = 2
haar_flags = 0

def detect_and_draw(img, cascade):
    # allocate temporary images
    gray = cv.CreateImage((img.width, img.height), 8, 1)
    small_img = cv.CreateImage((cv.Round(img.width / image_scale),
                               cv.Round (img.height / image_scale)), 8, 1)

    # convert color input image to grayscale
    cv.CvtColor(img, gray, cv.CV_BGR2GRAY)

    # scale input image for faster processing
    cv.Resize(gray, small_img, cv.CV_INTER_LINEAR)
    cv.EqualizeHist(small_img, small_img)

    if(cascade):
        t = cv.GetTickCount()
        faces = cv.HaarDetectObjects(small_img, cascade, cv.CreateMemStorage(0),
                                     haar_scale, min_neighbors, haar_flags, min_size)
        t = cv.GetTickCount() - t
        print "detection time = %gms" % (t/(cv.GetTickFrequency()*1000.))
        if faces:
            for ((x, y, w, h), n) in faces:
                # the input to cv.HaarDetectObjects was resized, so scale the
                # bounding box of each face and convert it to two CvPoints
                pt1 = (int(x * image_scale), int(y * image_scale))
                pt2 = (int((x + w) * image_scale), int((y + h) * image_scale))
                cv.Rectangle(img, pt1, pt2, cv.RGB(255, 0, 0), 3, 8, 0)

    cv.ShowImage("result", img)

```

Figura 39: Trecho do *script* facedetect.py.

Em imagens com pouca iluminação ou com excesso de iluminação, constatou-se que o *script* não consegue fazer a identificação, pois a identificação é obtida através do uso da binarização (limiarização), e o excesso ou falta de iluminação influencia os valores do histograma, assim deformando os valores padrões de um rosto, podemos observar este caso nas Figuras 40 e 41. Também podemos observar nestas Figuras que rostos que não aparecem por completo na foto, ou seja, só uma parte do rosto aparece na Figura, não são identificados.

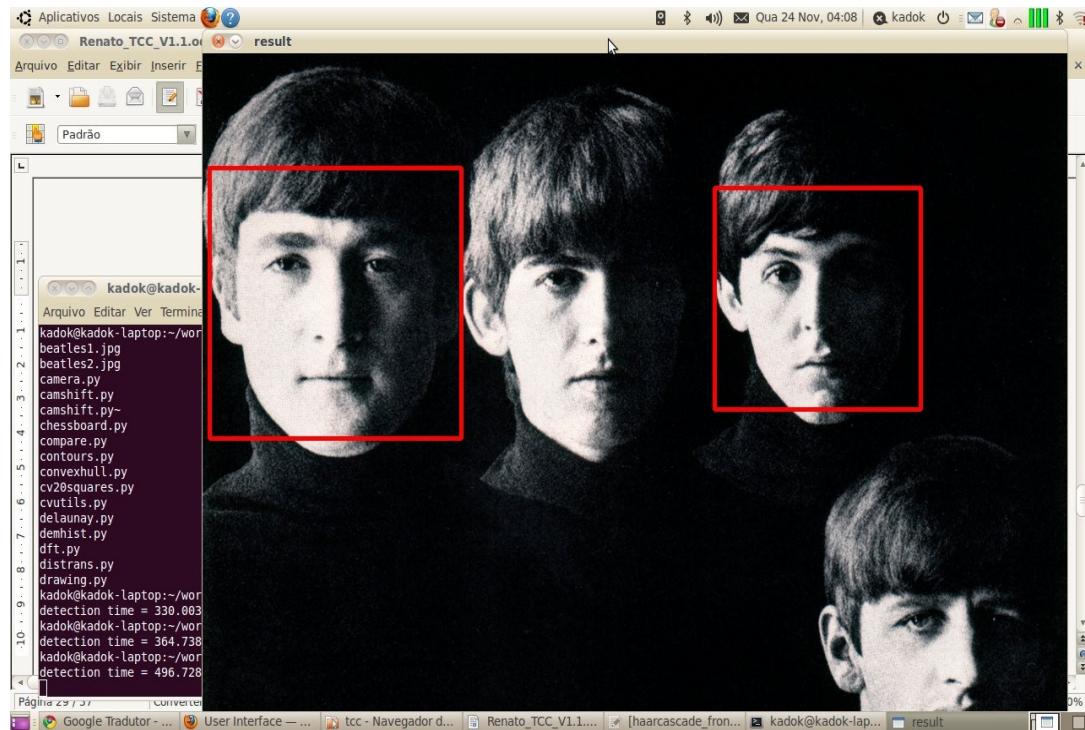


Figura 40: Face Detect

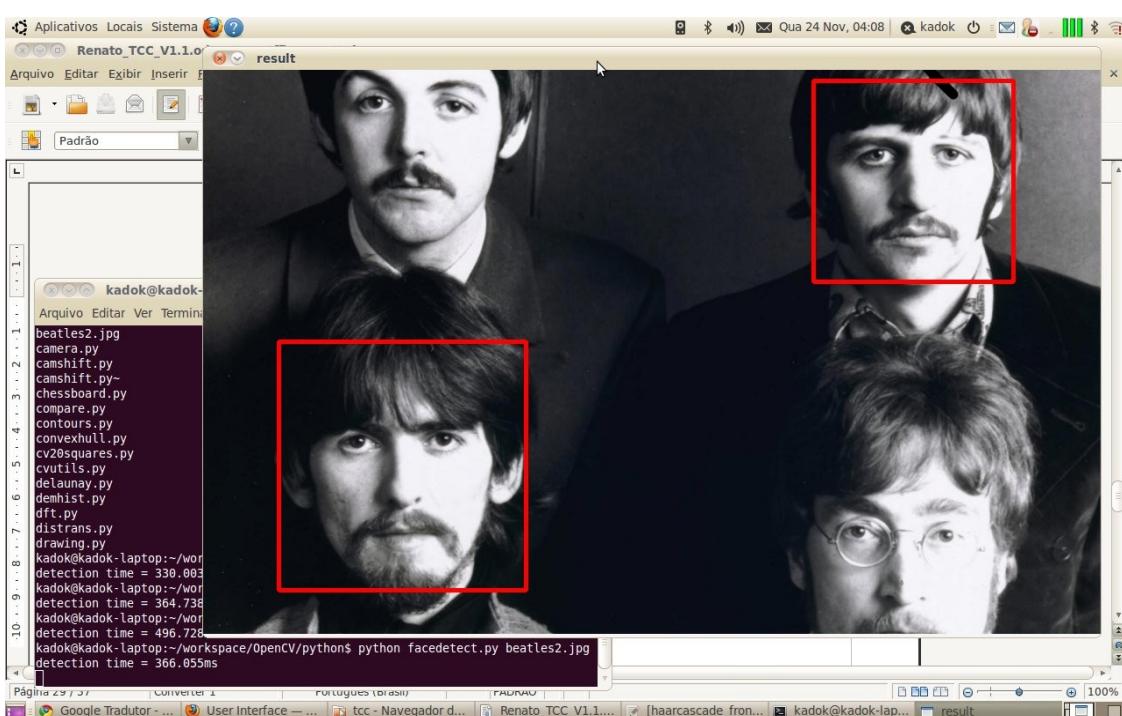


Figura 41: Face Detect

Imagens de baixa resolução também dificultam bastante o processo de detecção. Nas Figuras 42 e 43 podemos observar imagens de após a detecção.



Figura 42: Face Detect

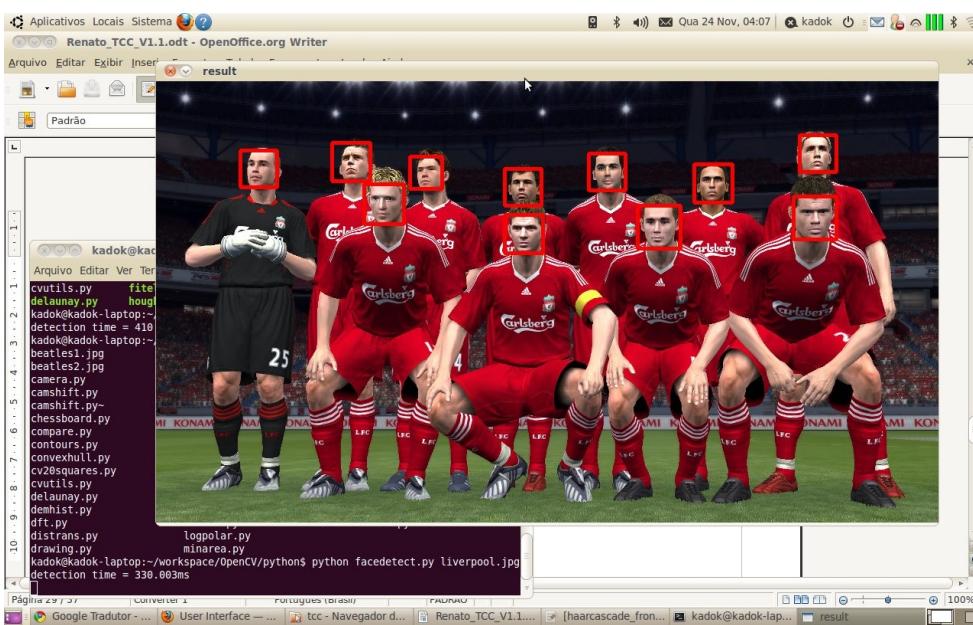


Figura 43: Face Detect

4 CONCLUSÕES E TRABALHOS FUTUROS

A área de processamento de imagens tem importantes aplicações e contribuições para as mais diversas áreas, por exemplo, medicina, industrial, e militar.

Este trabalho abordou os aspectos considerados fundamentais do processo de síntese de imagens digitais, que se fundamenta na utilização da biblioteca OpenCV com Python. Através destes aspectos construímos aplicações (*scripts*), na área de visão computacional, principalmente voltado ao reconhecimento de padrões. Estas aplicações foram implementadas de forma a apresentar o quanto fascinante é o ramo de processamento de imagens. Aplicações como o detector facial, ou o algoritmo CAMSHIFT, são aplicações que podem ser utilizadas em robôs ou máquinas em geral, para os mais diversos sistemas.

A biblioteca OpenCV possibilitou a criação de todas as aplicações, e o seu potencial foi confirmado, verificando os resultados obtidos e assim avaliando o seu desempenho, visto que, nosso ambiente para desenvolvimento e testes foi um computador pessoal.

Algumas linhas de atuação podem estender este trabalho, como por exemplo:

A implementação da biblioteca OpenCV para sistemas biométricos;

A implementação da biblioteca OpenCV para sistemas na internet;

Um estudo sobre a implementação da biblioteca OpenCV para controle de jogos através do uso do *script* CamSHIFT;

5 REFERÊNCIAS BIBLIOGRÁFICAS

1. ANDREAS, K. ; ABIDI, M. “**Digital Color Image Processing**”, Abril, 2008.
2. AKSOY, M. S., O. Torkul, e I. H. Cedimoglu, "An industrial visual inspection system that uses inductive learning." Journal of Intelligent Manufacturing, Agosto de 2004, Expanded Academic ASAP. Thomson Gale.
3. ATKIN, Denny. "Computer Shopper: The Right GPU for You". http://computershopper.com/feature/200704_the_right_gpu_for_you Acesso em 15 nov. 2010.
4. BERTOLI, I. M. “**Estudo dos Operadores Morfológicos Simulação de Erosão e Dilatação de imagens digitais usando MATLAB**”. Dissertação (Graduação em Engenharia Elétrica de Telecomunicações). Pontifícia Universidade Católica, Campinas, São Paulo, 2004.
5. BRADSKI, Gary. Computer Vision Face Tracking For Use in a Perceptual User Interface, Intel Technology Journal Q2. Microcomputer Research Lab, Santa Clara, CA, Intel Corporation, 1998.
6. BRADSKI, Gary, KAEHLER, Adrian. Learning OpenCV Computer Vision with the OpenCV Library. O'REILLY, ISBN 978-0-596-51613-0, 2008.
7. BRUNELLI, Roberto, *Template Matching Techniques in Computer Vision: Theory and Practice*, Wiley, ISBN 978-0-470-51706-2, 2009.
8. BURCHER B. BARBOSA, Bernardo, SILVA, Júlio César. Interação Computador – Humano Usando Visão Computacional, Revista TECEN –

- Edição Especial – volume 2 – número 1. Março de 2009. ISSN 1984-0993, p.11-12.
9. F. Jurie, M. Dhome. Real time robust template matching. In British Machine Vision Conference, p.123–131, 2002.
 10. GONZALEZ, R.C.; WOODS, R.E. “**Digital Image Processing**”. 3th ed. Person Prentice Hall. New Jersey, 2008.
 11. GONZALEZ, R. C.; WOODS, R. E.; EDDINS, S. “**Digital Image Processing Using Matlab**” – Julho, 2006.
 12. INTEL, Open Source Computer Vision Library, Reference Manual, 2000.
 13. KUNTZ, Noah. OpenCV Tutorials.
<http://www.pages.drexel.edu/~nk752/tutorials.html> Acesso em 10 out. 2010.
 14. KYRIACOU, Theocharis, GUIDO, Bugmann, e STANISLAO, Lauria. "Vision-based urban navigation procedures for verbally instructed robots." Robotics and Autonomous Systems 51.1, Abril de 2005, p.69-80, Expanded Academic ASAP. Thomson Gale.
 15. LI, Yuhai, L. Jian, T. Jinwen, X. Honbo. "A fast rotated template matching based on point feature." Proceedings of the SPIE 6043, 2005, p.453-459, MIPPR 2005, SAR and Multispectral Image Processing.
 16. OpenCV 2.1 Python Reference.
<http://opencv.willowgarage.com/documentation/python/index.html> Acesso em 18 ago. 2010.
 17. OpenCV Open Source Computer Vision Library Community.
<http://tech.groups.yahoo.com/group/OpenCV/> Acesso em 13 out. 2010.

18. Python / PIL template matching.
[<http://www.daniweb.com/forums/thread252384.html>](http://www.daniweb.com/forums/thread252384.html) Acesso em 13 out. 2010.
19. ROMERO, Mario. Fun with Pyhton, OpenCV and face detection.
 [<http://blog.jozilla.net/2008/06/27/fun-with-python-opencv-and-face-detection/>](http://blog.jozilla.net/2008/06/27/fun-with-python-opencv-and-face-detection/) Acesso em 13 out. 2010.
20. MARQUES FILHO, Ogê; VIEIRA NETO, Hugo. **Processamento Digital de Imagens**, Rio de Janeiro, RJ, Brasport, 1999. ISBN 8574520098, p.55-56.
21. MARTIN, A.; TOSUNOGLU, S. "Image Processing Techniques for machine.
22. SILVA GARCIA LEITE, Leonardo da; SANTOS MANGELLI, Tadeu; MOTTA WEYNE MARQUES, Pedro Leonardo. Reconhecimento de placas de veículos através de processamento de imagens. Barra Mansa, 2009. Monografia – Curso de Bacharelado em Ciência da Computação, UBM, Barra Mansa, RJ.
23. WANG, Ching Yang, Ph.D. "Edge detection using template matching". Duke University, 1985, AAT 8523046.
24. Cândido, J., e Marengoni, M, Combining Information in a Bayesian Network for Face Detection, Brazilian Journal of Probability and Statistics, 2009 (to appear).
25. Wikipédia, a enciclopédia livre, em português.
[<http://en.wikipedia.org/wiki/Template_matching>](http://en.wikipedia.org/wiki/Template_matching) Acesso em 1 out. 2010.
26. Kalman R. E., A new approach to linear filtering and prediction problems. Transactions of the ASME – Journal of Basic Engineering, 82:35-45, 1960.

27. Isard M. e Blake A., Condensation-conditional density propagation for visual tracking. International Journal in Computer Vision, IJCV 29(1):5-28, 1998.
28. Marr, D. E Hildreth, E. Theory of Edge Detection, Proc. Of The Royal Society of London, vol B207, pp. 187-217.
29. Canny, J., A Computational Approach for Edge Detection, IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 8, no 6, pp. 679-698, 1986.
30. Longin Jan Latecki, Template Matching based on a project by Roland Miezianko, Temple University, 2005.
31. Maurício Marengoni, Denise Stringhini, Introdução a Visão Computacional usando OpenCV, Universidade Presbiteriana Mackenzie, Dissertação, 2009
32. Claudio Esperança, Paulo Roma Cavalcanti. Introdução à Computação Gráfica, Coppe UFRJ, 2006.

ANEXOS

Nesta seção estão os scripts desenvolvidos para implementação das técnicas de processamento de imagens apresentadas neste projeto. A partir de exemplos disponibilizados pelo site da biblioteca OpenCV, foi possível fazer pequenas alterações nos scripts para a implementação. Os scripts foram desenvolvidos na linguagem python, portanto, para executá-los é necessário ter o interpretador python instalado em sua máquina, e também instalar a biblioteca OpenCV. Os scripts foram desenvolvidos na plataforma Linux, mas é possível executá-los na plataforma Windows, pois python e a biblioteca OpenCV tem versões para estas duas plataformas.

ANEXO A – camshift.py

```
#!/usr/bin/env python

import cv

def is_rect_nonzero(r):
    (_,_,w,h) = r
    return (w > 0) and (h > 0)

class CamShiftDemo:

    def __init__(self):
        self.capture = cv.CaptureFromCAM(0)
        cv.NamedWindow( "CamShiftDemo", 1 )
        cv.NamedWindow( "Histogram", 1 )
        cv.SetMouseCallback( "CamShiftDemo", self.on_mouse )

        self.drag_start = None      # Set to (x,y) when mouse starts drag
        self.track_window = None   # Set to rect when the mouse drag finishes

    print( "Keys:\n"
          "    ESC - quit the program\n"
          "    b - switch to/from backprojection view\n"
          "To initialize tracking, drag across the object with the mouse\n" )

    def hue_histogram_as_image(self, hist):
        """ Returns a nice representation of a hue histogram """

        histimg_hsv = cv.CreateImage( (320,200), 8, 3 )

        mybins = cv.CloneMatND(hist.bins)
        cv.Log(mybins, mybins)
        (_, hi, _, _) = cv.MinMaxLoc(mybins)
        cv.ConvertScale(mybins, mybins, 255. / hi)

        w,h = cv.GetSize(histimg_hsv)
        hdims = cv.GetDims(mybins)[0]
        for x in range(w):
            xh = (180 * x) / (w - 1) # hue sweeps from 0-180 across the image
```

```

val = int(mybins[int(hdims * x / w)] * h / 255)
cv.Rectangle( histimg_hsv, (x, 0), (x, h-val), (xh,255,64), -1)
cv.Rectangle( histimg_hsv, (x, h-val), (x, h), (xh,255,255), -1)

histimg = cv.CreateImage( (320,200), 8, 3)
cv.CvtColor(histimg_hsv, histimg, cv.CV_HSV2BGR)
return histimg

def on_mouse(self, event, x, y, flags, param):
    if event == cv.CV_EVENT_LBUTTONDOWN:
        self.drag_start = (x, y)
    if event == cv.CV_EVENT_LBUTTONUP:
        self.drag_start = None
        self.track_window = self.selection
    if self.drag_start:
        xmin = min(x, self.drag_start[0])
        ymin = min(y, self.drag_start[1])
        xmax = max(x, self.drag_start[0])
        ymax = max(y, self.drag_start[1])
        self.selection = (xmin, ymin, xmax - xmin, ymax - ymin)

def run(self):
    hist = cv.CreateHist([180], cv.CV_HIST_ARRAY, [(0,180)], 1 )
    backproject_mode = False
    while True:
        frame = cv.QueryFrame( self.capture )

        # Convert to HSV and keep the hue
        hsv = cv.CreateImage(cv.GetSize(frame), 8, 3)
        cv.CvtColor(frame, hsv, cv.CV_BGR2HSV)
        self.hue = cv.CreateImage(cv.GetSize(frame), 8, 1)
        cv.Split(hsv, self.hue, None, None, None)

        # Compute back projection
        backproject = cv.CreateImage(cv.GetSize(frame), 8, 1)

        # Run the cam-shift
        cv.CalcArrBackProject( [self.hue], backproject, hist )
        if self.track_window and is_rect_nonzero(self.track_window):
            crit = ( cv.CV_TERMCRIT_EPS | cv.CV_TERMCRIT_ITER, 10, 1)
            (iters, (area, value, rect), track_box) = cv.CamShift(backproject, self.track_window, crit)

```

```

    self.track_window = rect

    # If mouse is pressed, highlight the current selected rectangle
    # and recompute the histogram

    if self.drag_start and is_rect_nonzero(self.selection):
        sub = cv.GetSubRect(frame, self.selection)
        save = cv.CloneMat(sub)
        cv.ConvertScale(frame, frame, 0.5)
        cv.Copy(save, sub)
        x,y,w,h = self.selection
        cv.Rectangle(frame, (x,y), (x+w,y+h), (255,255,255))

        sel = cv.GetSubRect(self.hue, self.selection )
        cv.CalcArrHist( [sel], hist, 0)
        (_, max_val, _, _) = cv.GetMinMaxHistValue( hist)
        if max_val != 0:
            cv.ConvertScale(hist.bins, hist.bins, 255. / max_val)
    elif self.track_window and is_rect_nonzero(self.track_window):
        cv.EllipseBox( frame, track_box, cv.CV_RGB(255,0,0), 3, cv.CV_AA, 0 )

    if not backproject_mode:
        cv.ShowImage( "CamShiftDemo", frame )
    else:
        cv.ShowImage( "CamShiftDemo", backproject)
        cv.ShowImage( "Histogram", self.hue_histogram_as_image(hist))

    c = cv.WaitKey(7) % 0x100
    if c == 27:
        break
    elif c == ord("b"):
        backproject_mode = not backproject_mode

if __name__=="__main__":
    demo = CamShiftDemo()
    demo.run()

```

ANEXO B – compare.py

```
# -*- coding: utf-8 -*-
import sys
import cv
import Image
import math
import operator
from optparse import OptionParser

parser = OptionParser(usage = "usage: %prog [options] [filename|camera_index]");
(options, args) = parser.parse_args();
template = args[0];
img = args[1];

template = cv.LoadImage(template);
img = cv.LoadImage(img);

wi = img.width;
wii = template.width;
iwidth = wi - wii +1;
he = img.height;
hei = template.height;
iheight = he - hei +1;
sz = (iwidth, iheight);

result = cv.CreateImage(sz,cv.IPL_DEPTH_32F,1);

cv.MatchTemplate(img,template,result,cv.CV_TM_CCOEFF_NORMED);

print "\nresult :"
print cv.GetReal1D( result,0 );
```

ANEXO C – demhist.py

```
#!/usr/bin/python
import cv
import sys
import urllib2

hist_size = 64
range_0 = [0, 256]
ranges = [ range_0 ]

class DemHist:

    def __init__(self, src_image):
        self.src_image = src_image
        self.dst_image = cv.CloneMat(src_image)
        self.hist_image = cv.CreateImage((320, 200), 8, 1)
        self.hist = cv.CreateHist([hist_size], cv.CV_HIST_ARRAY, ranges, 1)

        self.brightness = 0
        self.contrast = 0

        cv.NamedWindow("image", 0)
        cv.NamedWindow("histogram", 0)
        cv.CreateTrackbar("Brilho", "image", 100, 200, self.update_brightness)
        cv.CreateTrackbar("Contraste", "image", 100, 200, self.update_contrast)

        self.update_brightcont()

    def update_brightness(self, val):
        self.brightness = val - 100
        self.update_brightcont()

    def update_contrast(self, val):
        self.contrast = val - 100
        self.update_brightcont()

    def update_brightcont(self):
        # The algorithm is by Werner D. Streidt
        # (http://visca.com/ffactory/archives/5-99/msg00021.html)
```

```

if self.contrast > 0:
    delta = 127. * self.contrast / 100
    a = 255. / (255. - delta * 2)
    b = a * (self.brightness - delta)
else:
    delta = -128. * self.contrast / 100
    a = (256. - delta * 2) / 255.
    b = a * self.brightness + delta

cv.ConvertScale(self.src_image, self.dst_image, a, b)
cv.ShowImage("image", self.dst_image)

cv.CalcArrHist([self.dst_image], self.hist)
(min_value, max_value, _, _) = cv.GetMinMaxHistValue(self.hist)
cv.Scale(self.hist.bins, self.hist.bins, float(self.hist_image.height) / max_value, 0)

cv.Set(self.hist_image, cv.ScalarAll(255))
bin_w = round(float(self.hist_image.width) / hist_size)

for i in range(hist_size):
    cv.Rectangle(self.hist_image, (int(i * bin_w), self.hist_image.height),
                (int((i + 1) * bin_w), self.hist_image.height - cv.Round(self.hist.bins[i])), 
                cv.ScalarAll(0), -1, 8, 0)

cv.ShowImage("histogram", self.hist_image)

if __name__ == "__main__":
    # Load the source image.
    if len(sys.argv) > 1:
        src_image = cv.GetMat(cv.LoadImage(sys.argv[1], 0))
    else:
        url = 'https://code.ros.org/svn/opencv/trunk/opencv/samples/c/baboon.jpg'
        filedata = urllib2.urlopen(url).read()
        imagefiledata = cv.CreateMatHeader(1, len(filedata), cv.CV_8UC1)
        cv.SetData(imagefiledata, filedata, len(filedata))
        src_image = cv.DecodeImageM(imagefiledata, 0)

    dh = DemHist(src_image)

    cv.WaitKey(0)

```

ANEXO D – edge.py

```
#! /usr/bin/env python

print "OpenCV Python version of edge"

import sys
import urllib2
import cv

# some definitions
win_name = "Edge"
trackbar_name = "Threshold"

# the callback on the trackbar
def on_trackbar(position):

    cv.Smooth(gray, edge, cv.CV_BLUR, 3, 3, 0)
    cv.Not(gray, edge)

    # run the edge dector on gray scale
    cv.Canny(gray, edge, position, position * 3, 3)

    # reset
    cv.SetZero(col_edge)

    # copy edge points
    cv.Copy(im, col_edge, edge)

    # show the im
    cv.ShowImage(win_name, col_edge)

if __name__ == '__main__':
    if len(sys.argv) > 1:
        im = cv.LoadImage( sys.argv[1], cv.CV_LOAD_IMAGE_COLOR)
    else:
        url = 'https://code.ros.org/svn/opencv/trunk/opencv/samples/c/fruits.jpg'
        filedata = urllib2.urlopen(url).read()
        imagefiledata = cv.CreateMatHeader(1, len(filedata), cv.CV_8UC1)
        cv.SetData(imagefiledata, filedata, len(filedata))
```

```
im = cv.DecodeImage(imagefiledata, cv.CV_LOAD_IMAGE_COLOR)

# create the output im
col_edge = cv.CreateImage((im.width, im.height), 8, 3)

# convert to grayscale
gray = cv.CreateImage((im.width, im.height), 8, 1)
edge = cv.CreateImage((im.width, im.height), 8, 1)
cv.CvtColor(im, gray, cv.CV_BGR2GRAY)

# create the window
cv.NamedWindow(win_name, cv.CV_WINDOW_AUTOSIZE)

# create the trackbar
cv.CreateTrackbar(trackbar_name, win_name, 1, 100, on_trackbar)

# show the im
on_trackbar(0)

# wait a key pressed to end
cv.WaitKey(0)
```

ANEXO E – facedetect.py

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
"""

This program is demonstration for face and object detection using haar-like features.
The program finds faces in a camera image or video stream and displays a red box around them.

Original C implementation by: ?
Python implementation by: Roman Stanchak, James Bowman
"""

import sys
import cv
from optparse import OptionParser

# Parameters for haar detection
# From the API:
# The default parameters (scale_factor=2, min_neighbors=3, flags=0) are tuned
# for accurate yet slow object detection. For a faster operation on real video
# images the settings are:
# scale_factor=1.2, min_neighbors=2, flags=CV_HAAR_DO_CANNY_PRUNING,
# min_size=<minimum possible face size

min_size = (20, 20)
image_scale = 2
haar_scale = 1.2
min_neighbors = 2
haar_flags = 0

def detect_and_draw(img, cascade):
    # allocate temporary images
    gray = cv.CreateImage((img.width,img.height), 8, 1)
    small_img = cv.CreateImage((cv.Round(img.width / image_scale),
                               cv.Round (img.height / image_scale)), 8, 1)

    # convert color input image to grayscale
    cv.CvtColor(img, gray, cv.CV_BGR2GRAY)

    # scale input image for faster processing
    cv.Resize(gray, small_img, cv.CV_INTER_LINEAR)

```

```

cv.EqualizeHist(small_img, small_img)

if(cascade):
    t = cv.GetTickCount()
    faces = cv.HaarDetectObjects(small_img, cascade, cv.CreateMemStorage(0),
                                 haar_scale, min_neighbors, haar_flags, min_size)
    t = cv.GetTickCount() - t
    print "detection time = %gms" % (t/(cv.GetTickFrequency()*1000.))
    if faces:
        for ((x, y, w, h), n) in faces:
            # the input to cv.HaarDetectObjects was resized, so scale the
            # bounding box of each face and convert it to two CvPoints
            pt1 = (int(x * image_scale), int(y * image_scale))
            pt2 = (int((x + w) * image_scale), int((y + h) * image_scale))
            cv.Rectangle(img, pt1, pt2, cv.RGB(255, 0, 0), 3, 8, 0)

cv.ShowImage("result", img)

if __name__ == '__main__':
    parser = OptionParser(usage = "usage: %prog [options] [filename|camera_index]")
    parser.add_option("-c", "--cascade", action="store", dest="cascade", type="str", help="Haar cascade
file,default%default",default=
"/home/kadok/OpenCV-2.1.0/data/haarcascades/haarcascade_frontalface_alt.xml")
    (options, args) = parser.parse_args()

    cascade = cv.Load(options.cascade)

    if len(args) != 1:
        parser.print_help()
        sys.exit(1)

    input_name = args[0]
    if input_name.isdigit():
        capture = cv.CreateCameraCapture(int(input_name))
    else:
        capture = None

    cv.NamedWindow("result", cv.CV_WINDOW_AUTOSIZE)

```

```
if capture:  
    frame_copy = None  
    while True:  
        frame = cv.QueryFrame(capture)  
        if not frame:  
            cv.WaitKey(0)  
            break  
        if not frame_copy:  
            frame_copy = cv.CreateImage((frame.width,frame.height),  
                                         cv.IPL_DEPTH_8U, frame.nChannels)  
        if frame.origin == cv.IPL_ORIGIN_TL:  
            cv.Copy(frame, frame_copy)  
        else:  
            cv.Flip(frame, frame_copy, 0)  
  
        detect_and_draw(frame_copy, cascade)  
  
        if cv.WaitKey(10) >= 0:  
            break  
    else:  
        image = cv.LoadImage(input_name, 1)  
        detect_and_draw(image, cascade)  
        cv.WaitKey(0)  
  
    cv.DestroyWindow("result")
```

ANEXO F – fback.py

```
#!/usr/bin/env python

from cv import *

class FBackDemo:
    def __init__(self):
        self.capture = CaptureFromCAM(0)
        self.mv_step = 16
        self.mv_scale = 1.5
        self.mv_color = (0, 255, 0)
        self.cflow = None
        self.flow = None

    NamedWindow( "Optical Flow", 1 )

    print( "Press ESC - quit the program\n" )

    def draw_flow(self, flow, prevgray):
        """ Returns a nice representation of a hue histogram """
        CvtColor(prevgray, self.cflow, CV_GRAY2BGR)
        for y in range(0, flow.height, self.mv_step):
            for x in range(0, flow.width, self.mv_step):
                fx, fy = flow[y, x]
                Line(self.cflow, (x,y), (x+fx,y+fy), self.mv_color)
                Circle(self.cflow, (x,y), 2, self.mv_color, -1)
        ShowImage("Optical Flow", self.cflow)

    def run(self):
        first_frame = True

        while True:
            frame = QueryFrame( self.capture )

            if first_frame:
                gray = CreateImage(GetSize(frame), 8, 1)
                prev_gray = CreateImage(GetSize(frame), 8, 1)
                flow = CreateImage(GetSize(frame), 32, 2)
```

```
self.cflow = CreateImage(GetSize(frame), 8, 3)

CvtColor(frame, gray, CV_BGR2GRAY)
if not first_frame:
    CalcOpticalFlowFarneback(prev_gray, gray, flow,
        pyr_scale=0.5, levels=3, winsize=15,
        iterations=3, poly_n=5, poly_sigma=1.2, flags=0)
    self.draw_flow(flow, prev_gray)
    c = WaitKey(7)
    if c in [27, ord('q'), ord('Q')]:
        break
    prev_gray, gray = gray, prev_gray
    first_frame = False

if __name__=="__main__":
    demo = FBackDemo()
    demo.run()
```

ANEXO G – laplace.py

```

#!/usr/bin/python
import urllib2
import cv
import sys

if __name__ == "__main__":
    laplace = None
    colorlaplace = None
    planes = [None, None, None]
    capture = None

    if len(sys.argv) == 1:
        capture = cv.CreateCameraCapture(0)
    elif len(sys.argv) == 2 and sys.argv[1].isdigit():
        capture = cv.CreateCameraCapture(int(sys.argv[1]))
    elif len(sys.argv) == 2:
        capture = cv.CreateFileCapture(sys.argv[1])

    if not capture:
        print "Could not initialize capturing..."
        sys.exit(-1)

    cv.NamedWindow("Laplacian", 1)

    while True:
        frame = cv.QueryFrame(capture)
        if frame:
            if not laplace:
                planes = [cv.CreateImage((frame.width, frame.height), 8, 1) for i in range(3)]
                laplace = cv.CreateImage((frame.width, frame.height), cv.IPL_DEPTH_16S, 1)
                colorlaplace = cv.CreateImage((frame.width, frame.height), 8, 3)

            cv.Split(frame, planes[0], planes[1], planes[2], None)
            for plane in planes:
                cv.Laplace(plane, laplace, 3)
                cv.ConvertScaleAbs(laplace, plane, 1, 0)

            cv.Merge(planes[0], planes[1], planes[2], None, colorlaplace)

```

```
cv.ShowImage("Laplacian", colorlaplace)

if cv.WaitKey(10) != -1:
    break

cv.DestroyWindow("Laplacian")
```

ANEXO H – match.py

```
# -*- coding: utf-8 -*-
from PIL import Image
import datetime
from optparse import OptionParser

def matchTemplate(searchImage, templateImage):
    minScore = -1000
    matching_xs = 0
    matching_ys = 0
    searchWidth = searchImage.size[0]
    searchHeight = searchImage.size[1]
    templateWidth = templateImage.size[0]
    templateHeight = templateImage.size[1]
    searchIm = searchImage.load()
    templateIm = templateImage.load()
    #loop over each pixel in the search image
    for xs in range(searchWidth-templateWidth+1):
        for ys in range(searchHeight-templateHeight+1):
            #for ys in range(10):
                #set some kind of score variable to 0
                score = 0
                #loop over each pixel in the template image
                for xt in range(templateWidth):
                    for yt in range(templateHeight):
                        score += 1 if searchIm[xs+xt,ys+yt] == templateIm[xt, yt] else -1

            if minScore < score:
                minScore = score
                matching_xs = xs
                matching_ys = ys

    print "Location=", (matching_xs, matching_ys), "Score=", minScore
    im1 = Image.new('RGB', (searchWidth, searchHeight), (80, 147, 0))
    im1.paste(templateImage, ((matching_xs), (matching_ys)))
    #searchImage.show()
    #im1.show()
    im1.save('Images/template_matched_in_search.png')
```

```
parser = OptionParser(usage = "usage: %prog [options] [filename|camera_index]")
(options, args) = parser.parse_args()
templateImage = args[0]
searchImage = args[1]

templateImage = Image.open(templateImage);
searchImage = Image.open(searchImage);

t1=datetime.datetime.now()
matchTemplate(searchImage, templateImage)
delta=datetime.datetime.now()-t1
print "Time=%d.%d"%(delta.seconds,delta.microseconds)
print "end";
```

ANEXO I – morphology.py

```

#!/usr/bin/python
import sys
import urllib2
import cv

src = 0
image = 0
dest = 0
element_shape = cv.CV_SHAPE_RECT

def Opening(pos):
    element = cv.CreateStructuringElementEx(pos*2+1, pos*2+1, pos, pos, element_shape)
    cv.Erode(src, image, element, 1)
    cv.Dilate(image, dest, element, 1)
    cv.ShowImage("Opening & Closing", dest)

def Closing(pos):
    element = cv.CreateStructuringElementEx(pos*2+1, pos*2+1, pos, pos, element_shape)
    cv.Dilate(src, image, element, 1)
    cv.Erode(image, dest, element, 1)
    cv.ShowImage("Opening & Closing", dest)

def Erosion(pos):
    element = cv.CreateStructuringElementEx(pos*2+1, pos*2+1, pos, pos, element_shape)
    cv.Erode(src, dest, element, 1)
    cv.ShowImage("Erosao & Dilacao", dest)

def Dilation(pos):
    element = cv.CreateStructuringElementEx(pos*2+1, pos*2+1, pos, pos, element_shape)
    cv.Dilate(src, dest, element, 1)
    cv.ShowImage("Erosao & Dilacao", dest)

if __name__ == "__main__":
    if len(sys.argv) > 1:
        src = cv.LoadImage(sys.argv[1], cv.CV_LOAD_IMAGE_COLOR)
    else:
        url = 'https://code.ros.org/svn/opencv/trunk/opencv/samples/c/fruits.jpg'
        filedata = urllib2.urlopen(url).read()
        imagefiledata = cv.CreateMatHeader(1, len(filedata), cv.CV_8UC1)
        cv.SetData(imagefiledata, filedata, len(filedata))
        src = cv.DecodeImage(imagefiledata, cv.CV_LOAD_IMAGE_COLOR)

```

```
image = cv.CloneImage(src)
dest = cv.CloneImage(src)
#cv.NamedWindow("Opening & Closing", 1)
cv.NamedWindow("Erosao & Dilacao", 1)
#cv.ShowImage("Opening & Closing", src)
cv.ShowImage("Erosao & Dilacao", src)
#cv.CreateTrackbar("Open", "Opening & Closing", 0, 10, Opening)
#cv.CreateTrackbar("Close", "Opening & Closing", 0, 10, Closing)
cv.CreateTrackbar("Dilatar", "Erosao & Dilacao", 0, 10, Dilation)
cv.CreateTrackbar("Erodir", "Erosao & Dilacao", 0, 10, Erosion)
cv.WaitKey(0)
#cv.DestroyWindow("Opening & Closing")
cv.DestroyWindow("Erosao & Dilacao")
```

ANEXO J – motempl.py

```
#!/usr/bin/python

import urllib2
import sys
import time
from math import cos, sin
import cv

CLOCKS_PER_SEC = 1.0
MHI_DURATION = 1
MAX_TIME_DELTA = 0.5
MIN_TIME_DELTA = 0.05
N = 4
buf = range(10)
last = 0
mhi = None # MHI
orient = None # orientation
mask = None # valid orientation mask
segmask = None # motion segmentation map
storage = None # temporary storage

def update_mhi(img, dst, diff_threshold):
    global last
    global mhi
    global storage
    global mask
    global orient
    global segmask
    timestamp = time.clock() / CLOCKS_PER_SEC # get current time in seconds
    size = cv.GetSize(img) # get current frame size
    idx1 = last
    if not mhi or cv.GetSize(mhi) != size:
        for i in range(N):
            buf[i] = cv.CreateImage(size, cv.IPL_DEPTH_8U, 1)
            cv.Zero(buf[i])
        mhi = cv.CreateImage(size, cv.IPL_DEPTH_32F, 1)
        cv.Zero(mhi) # clear MHI at the beginning
        orient = cv.CreateImage(size, cv.IPL_DEPTH_32F, 1)
        segmask = cv.CreateImage(size, cv.IPL_DEPTH_32F, 1)
```

```

mask = cv.CreateImage(size, cv.IPL_DEPTH_8U, 1)

cv.CvtColor(img, buf[last], cv.CV_BGR2GRAY) # convert frame to grayscale
idx2 = (last + 1) % N # index of (last - (N-1))th frame
last = idx2
silh = buf[idx2]
cv.AbsDiff(buf[idx1], buf[idx2], silh) # get difference between frames
cv.Threshold(silh, silh, diff_threshold, 1, cv.CV_THRESH_BINARY) # and threshold it
cv.UpdateMotionHistory(silh, mhi, timestamp, MHI_DURATION) # update MHI
cv.CvtScale(mhi, mask, 255./MHI_DURATION,
            (MHI_DURATION - timestamp)*255./MHI_DURATION)
cv.Zero(dst)
cv.Merge(mask, None, None, None, dst)
cv.CalcMotionGradient(mhi, mask, orient, MAX_TIME_DELTA, MIN_TIME_DELTA, 3)
if not storage:
    storage = cv.CreateMemStorage(0)
seq = cv.SegmentMotion(mhi, segmask, storage, timestamp, MAX_TIME_DELTA)
for (area, value, comp_rect) in seq:
    if comp_rect[2] + comp_rect[3] > 100: # reject very small components
        color = cv.CV_RGB(255, 0, 0)
        silh_roi = cv.GetSubRect(silh, comp_rect)
        mhi_roi = cv.GetSubRect(mhi, comp_rect)
        orient_roi = cv.GetSubRect(orient, comp_rect)
        mask_roi = cv.GetSubRect(mask, comp_rect)
        angle = 360 - cv.CalcGlobalOrientation(orient_roi, mask_roi, mhi_roi, timestamp,
                                              MHI_DURATION)

        count = cv.Norm(silh_roi, None, cv.CV_L1, None) # calculate number of points within
        silhouette ROI
        if count < (comp_rect[2] * comp_rect[3] * 0.05):
            continue

        magnitude = 30.
        center = ((comp_rect[0] + comp_rect[2] / 2), (comp_rect[1] + comp_rect[3] / 2))
        cv.Circle(dst, center, cv.Round(magnitude*1.2), color, 3, cv.CV_AA, 0)
        cv.Line(dst,
                center,
                (cv.Round(center[0] + magnitude * cos(angle * cv.CV_PI / 180)),
                 cv.Round(center[1] - magnitude * sin(angle * cv.CV_PI / 180))),
                color,
                3,

```

```
cv.CV_AA,  
0)  
  
if __name__ == "__main__":  
    motion = 0  
    capture = 0  
  
    if len(sys.argv)==1:  
        capture = cv.CreateCameraCapture(0)  
    elif len(sys.argv)==2 and sys.argv[1].isdigit():  
        capture = cv.CreateCameraCapture(int(sys.argv[1]))  
    elif len(sys.argv)==2:  
        capture = cv.CreateFileCapture(sys.argv[1])  
  
    if not capture:  
        print "Could not initialize capturing..."  
        sys.exit(-1)  
  
    cv.NamedWindow("Motion", 1)  
    while True:  
        image = cv.QueryFrame(capture)  
        if(image):  
            if(not motion):  
                motion = cv.CreateImage((image.width, image.height), 8, 3)  
                cv.Zero(motion)  
                #motion.origin = image.origin  
                update_mhi(image, motion, 30)  
                cv.ShowImage("Motion", motion)  
                if(cv.WaitKey(10) != -1):  
                    break  
            else:  
                break  
    cv.DestroyWindow("Motion")
```

ANEXO K – squares.py

```
"""
Find Squares in image by finding countours and filtering
"""

#Results slightly different from C version on same images, but is
#otherwise ok

import math
import cv

def angle(pt1, pt2, pt0):
    "calculate angle contained by 3 points(x, y)"
    dx1 = pt1[0] - pt0[0]
    dy1 = pt1[1] - pt0[1]
    dx2 = pt2[0] - pt0[0]
    dy2 = pt2[1] - pt0[1]

    nom = dx1*dx2 + dy1*dy2
    denom = math.sqrt( (dx1*dx1 + dy1*dy1) * (dx2*dx2 + dy2*dy2) + 1e-10 )
    ang = nom / denom
    return ang

def is_square(contour):
    """
    Squareness checker

    Square contours should:
    -have 4 vertices after approximation,
    -have relatively large area (to filter out noisy contours)
    -be convex.
    -have angles between sides close to 90deg ( $\cos(\text{ang}) \sim 0$ )

    Note: absolute value of an area is used because area may be
    positive or negative - in accordance with the contour orientation
    """

    area = math.fabs( cv.ContourArea(contour) )
    isconvex = cv.CheckContourConvexity(contour)
    s = 0

    if len(contour) == 4 and area > 1000 and isconvex:
```

```

for i in range(1, 4):
    # find minimum angle between joint edges (maximum of cosine)
    pt1 = contour[i]
    pt2 = contour[i-1]
    pt0 = contour[i-2]

    t = math.fabs(angle(pt0, pt1, pt2))
    if s <= t:s = t

    # if cosines of all angles are small (all angles are ~90 degree)
    # then its a square
    if s < 0.3:return True

return False

def find_squares_from_binary( gray ):
    """
    use contour search to find squares in binary image
    returns list of numpy arrays containing 4 points
    """
    squares = []
    storage = cv.CreateMemStorage(0)
        contours      =      cv.FindContours(gray,      storage,      cv.CV_RETR_TREE,
cv.CV_CHAIN_APPROX_SIMPLE, (0,0))
    storage = cv.CreateMemStorage(0)
    while contours:
        #approximate contour with accuracy proportional to the contour perimeter
        arclength = cv.ArcLength(contours)
        polygon = cv.ApproxPoly( contours, storage, cv.CV_POLY_APPROX_DP, arclength * 0.02, 0)
        if is_square(polygon):
            squares.append(polygon[0:4])
        contours = contours.h_next()

    return squares

def find_squares4(color_img):
    """
    Finds multiple squares in image
    """

Steps:
-Use Canny edge to highlight contours, and dilation to connect

```

the edge segments.

- Threshold the result to binary edge tokens
- Use cv.FindContours: returns a cv.CvSequence of cv.CvContours
- Filter each candidate: use Approx poly, keep only contours with 4 vertices, enough area, and ~90deg angles.

Return all squares contours in one flat list of arrays, 4 x,y points each.

=====

```
#select even sizes only
width, height = (color_img.width & -2, color_img.height & -2 )
timg = cv.CloneImage( color_img ) # make a copy of input image
gray = cv.CreateImage( (width,height), 8, 1 )

# select the maximum ROI in the image
cv.SetImageROI( timg, (0, 0, width, height) )

# down-scale and upscale the image to filter out the noise
pyr = cv.CreateImage( (width/2, height/2), 8, 3 )
cv.PyrDown( timg, pyr, 7 )
cv.PyrUp( pyr, timg, 7 )

tgray = cv.CreateImage( (width,height), 8, 1 )
squares = []

# Find squares in every color plane of the image
# Two methods, we use both:
# 1. Canny to catch squares with gradient shading. Use upper threshold
# from slider, set the lower to 0 (which forces edges merging). Then
# dilate canny output to remove potential holes between edge segments.
# 2. Binary thresholding at multiple levels
N = 11
for c in [0, 1, 2]:
    #extract the c-th color plane
    cv.SetImageCOI( timg, c+1 );
    cv.Copy( timg, tgray, None );
    cv.Canny( tgray, gray, 0, 50, 5 )
    cv.Dilate( gray, gray)
    squares = squares + find_squares_from_binary( gray )

# Look for more squares at several threshold levels
for l in range(1, N):
```

```

cv.Threshold( tgray, gray, (l+1)*255/N, 255, cv.CV_THRESH_BINARY )
squares = squares + find_squares_from_binary( gray )

return squares

RED = (0,0,255)
GREEN = (0,255,0)
def draw_squares( color_img, squares ):
    """
    Squares is py list containing 4-pt numpy arrays. Step through the list
    and draw a polygon for each 4-group
    """
    color, othercolor = RED, GREEN
    for square in squares:
        cv.PolyLine(color_img, [square], True, color, 3, cv.CV_AA, 0)
        color, othercolor = othercolor, color

    cv.ShowImage(WNDNAME, color_img)

WNDNAME = "Squares Demo"
def main():
    """
    Open test color images, create display window, start the search"""
    cv.NamedWindow(WNDNAME, 1)
    for name in [ "Images/pic%d.png" % i for i in [1, 2, 3, 4, 5, 6] ]:
        img0 = cv.LoadImage(name, 1)
        try:
            img0
        except ValueError:
            print "Couldn't load %s\n" % name
            continue

    # slider deleted from C version, same here and use fixed Canny param=50
    img = cv.CloneImage(img0)

    cv.ShowImage(WNDNAME, img)

    # force the image processing
    draw_squares( img, find_squares4( img ) )

```

```
# wait for key.  
if cv.WaitKey(-1) % 0x100 == 27:  
    break  
  
if __name__ == "__main__":  
    main()
```

ANEXO L – template.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys
from cv import *
import cv
from optparse import OptionParser

parser = OptionParser(usage = "usage: %prog [options] [filename|camera_index]")
(options, args) = parser.parse_args()
modelo = args[0]
pesquisa = args[1]

# Read in the source image to be searched
src = LoadImage(pesquisa);

# Read in the template to be used for matching:
templ = LoadImage(modelo);

# Allocate Output Images:
wi = src.width;
wii = templ.width;
iwidth = wi - wii +1;
he = src.height;
hei = templ.height;
iheight = he - hei +1;
sz = (iwidth, iheight);
ftmp = [1,2,3,4,5,6];

i = 1;
while i < 7:
    ftmp[i-1]= cv.CreateImage( sz, 32, 1 );
    i = i + 1;

# Do the matching of the template with the image
i = 1;
while i < 7:
```

```
cv.MatchTemplate( src, templ, ftmp[i-1], i-1 );
cv.Normalize( ftmp[i-1], ftmp[i-1], 1, 0, cv.CV_MINMAX );
i = i + 1;

# DISPLAY
cv.NamedWindow( "Template", 0 );
cv.ShowImage( "Template", templ );
cv.NamedWindow( "Image", 0 );
cv.ShowImage( "Image", src );
cv.NamedWindow( "SQDIFF", 0 );
cv.ShowImage( "SQDIFF", ftmp[0] );
cv.NamedWindow( "SQDIFF_NORMED", 0 );
cv.ShowImage( "SQDIFF_NORMED", ftmp[1] );
cv.NamedWindow( "CCORR", 0 );
cv.ShowImage( "CCORR", ftmp[2] );
cv.NamedWindow( "CCORR_NORMED", 0 );
cv.ShowImage( "CCORR_NORMED", ftmp[3] );
cv.NamedWindow( "COEFF", 0 );
cv.ShowImage( "COEFF", ftmp[4] );
cv.NamedWindow( "COEFF_NORMED", 0 );
cv.ShowImage( "COEFF_NORMED", ftmp[5] );

cv.WaitKey(0);
```