UNIVERSIDADE DE SÃO PAULO ESCOLA POLITÉCNICA
DEPARTAMENTO DE ENGENHARIA MECATRÔNICA

Nota final
9,1 (nove e um)

ESTUDO VERIFICAÇÃO DE DESGASTE DE FERRAMENTAS DE USINAGEM COM AUXÍLIO DE TÉCNICAS DE PROCESSAMENTO DE IMAGENS

Eduardo Peterson Finamore

São Paulo
2005

UNIVERSIDADE DE SÃO PAULO ESCOLA POLITÉCNICA
DEPARTAMENTO DE ENGENHARIA MECATRÔNICA

ESTUDO VERIFICAÇÃO DE DESGASTE DE FERRAMENTAS DE USINAGEM COM AUXILIO DE TÉCNICAS DE PROCESSAMENTO DE IMAGENS

Trabalho de formatura apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Graduação em Engenharia.

Eduardo Peterson Finamore

Orientador: Prof. Dr. Amauri Hassui.

Área de Concentração:
Engenharia Mecatrônica

São Paulo
2005

AGRADECIMENTOS

Aos meus pais, pelo apoio incondicional a todos os meus sonhos.

Ao meus irmão, por tudo que vivemos juntos.

Ao Prof Amauri Hassui, pela orientação e confiança depositada.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE FIGURAS

Palavras chaves: Desgaste de ferramentas, Processamento de imagens, Edge detection.

RESUMO

Este trabalho esta vinculado à necessidade do estudo do desgaste da ponta de ferramentas de usinagem para detecção de possíveis irregularidades. Para este estudo pretende-se utilizar técnicas de processamento de imagens adquiridas em microscópio, sendo que além do processamento através do desenvolvimento de algoritmos específicos e utilização de métodos já conhecidos, será necessário ainda um aprofundamento sobre as melhores condições para aquisição da imagem, envolvendo iluminação adequada ao problema e ampliação que atenda a analise proposta. Ainda na primeira etapa após definidas as características ideais de aquisição será realizado um estudo sobre as características básicas de imagens, composição, cores, etc, afim de conseguir ferramentas básicas que permita o desenvolvimento de algoritmos específicos para o problema da analise do desgaste e que torne possível quantificar o desvio geométrico da ferramenta em relação a uma nova.

Pretende-se neste projeto implementar um software que a partir de um conjunto de imagens consiga automaticamente identificar as coordenadas das curvas que representam o desvio geométrico da ferramenta ocorrido durante o processo de usinagem. O software fornecerá ainda uma interface que permitirá visualizar o processamento em suas etapas. Para o desenvolvimento do software serão utilizados mathlab,C++ e Delphi.

ABSTRACT

A method to provide measurement of cutting tools is described. The tools are illuminated to make possible find out irregularities that cold cause surface finishing problems, overheat and others undesired behaviors. We expect to use machine vision techniques developing customized algorithms and study the influence of illumination in the problem solution. To accomplish these tasks we intend to use tools developed to assist image processing operations, such as openCv an open source library and mathlab image processing tool box, allowing the use of algorithms of edge detection and color segmentation.

This study is complemented with a software prototype projected to automate the process described above.

1 INTRODUÇÃO

O desgaste de ferramentas pode diminuir consideravelmente a produtividade. Só na indústria automobilística apenas o desgaste de componentes e ferramentas é responsável pela perda de bilhões de dólares. Percebe-se então a necessidade de estudos sobre como este desgaste ocorre, uma vez que o mesmo impacta na diminuição da vida útil da ferramenta e eventuais danos à peça que esta sendo usinada. Uma forma de contornar este problema, já que não é possível eliminar o desgaste, seria associarmos o desgaste da ferramenta ao longo de sua vida útil com a rugosidade da peça. Para isto, uma boa alternativa seria obter parâmetros que permitissem **quantificar o desvio geométrico da ferramenta em relação a uma nova** a partir de algum método de medição direta. Contudo, existem algumas dificuldades inerentes a analise deste processo de desgaste, uma vez que não temos apenas uma medição pontual e sim toda uma região de desgaste, tornando simplória a avaliação quando não inviável para a maioria dos casos. Uma forma automatizada e explorando o maior numero de pontos possíveis para analise possivelmente traria avanços no desenvolvimento de técnicas de controle do processo de usinagem e seria de grande utilidade numa indústria onde se perde tanto dinheiro com ferramentas utilizadas de forma errônea ou subutilizadas, assim como estabeleceria um método útil no desenvolvimento de novas ferramentas.

1.1 OBJETIVO

Observando as dificuldades anteriormente relacionadas e o desejo crescente das indústrias em minimizar perdas no processo de usinagem, percebe-se a necessidade do desenvolvimento de um sistema capaz de automatizar o processo de analise do desgaste de ferramentas.

Uma forma de obtermos alta precisão e ao mesmo tempo facilidade de processamento dos dados é através da análise de imagens coletadas de uma mesma ferramenta em diferentes níveis de desgaste. Isto agiliza o processo uma vez que não é necessário que alguém faça medidas manualmente e aumenta a precisão do resultado, eliminando as dificuldades de medição sobre uma superfície sem apoios, como é o caso da área que será analisada. Com isso pretende-se eliminar as dificuldades relacionadas a analise do desgaste de ferramentas e criar ferramentas de apoio que de forma automatizada consigam relacionar quantitativamente o desgaste ao longo do tempo, assim como poderá se tornar em trabalho futuro a viabilização da verificação em tempo real da ferramenta sugerindo alteracões aos parametros - usinagem.

Sendo assim pretende-se desenvolver um software que auxilie no estudo da verificação de desgaste de ferramentas de usinagem com auxilio de técnicas de processamento de imagens, comumente chamadas de Visão computacional.

2 CONCEITOS FUNDAMENTAIS

Atualmente, existe uma grande confusão acerca dos termos Processamento de Imagens, Visão Computacional, Visão Artificial e Reconhecimento de Padrões quando se discutem sistemas computacionais onde os dados de entrada são imagens. O primeiro passo para o correto entendimento das técnicas a serem descritas ao longo deste capítulo é definir qual o significado exato e o domínio de aplicação de cada um dos termos citados.

Define-se Reconhecimento de Padrões como o conjunto de métodos e técnicas que se ocupam da extração e identificação de informações em grandes quantidades de dados, que podem ser numéricos, como imagens ou sons, ou simbólicos, como bases de dados, informações estatísticas, etc [Duda, Hart & Stork, 2000].

Comumente chama-se Processamento de Imagens o aprimoramento de informações pictóricas para a interpretação por humanos ou por sistemas computacionais. Já **Visão Computacional** pode ser definida como o conjunto de métodos e técnicas através dos quais sistemas computacionais podem ser capazes de interpretar imagens. A interpretação de uma imagem pode ser definida em termos computacionais como a transformação de um conjunto de dados digitais representando uma imagem (um sinal mono-, bi-, tri- ou tetradimensional) em uma estrutura de dados descrevendo a semântica deste conjunto de dados em um contexto qualquer. Resume-se, portanto, Visão Computacional como o agrupamento das técnicas de Processamento de Imagens e Reconhecimento de Padrões com o objetivo de realizar a análise automática por computador de informações extraídas de cenas.

Visão Artificial, por sua vez, é junção das referidas técnicas de Visão Computacional com as de Aquisição de Imagens (ótica, iluminação, eletrônica, sensoriamento etc.), onde a análise, por sistemas computacionais, de atributos visuais de objetos é tratada num contexto mais amplo, envolvendo a aquisição, o processamento e a interpretação das suas imagens [Burke, 1996].

Este capítulo descreve, nas seções seguintes, os conceitos fundamentais aplicados no projeto algumas técnicas de **Visão Computacional**, bem como a estrutura de um **Sistema de Visão Artificial**.

2.1 VISÃO COMPUTACIONAL

Visão Computacional é a área da ciência que se dedica a desenvolver teorias e métodos voltados à extração automática de informações úteis contidas em imagens. Entretanto, a extração automática de informações úteis em imagens começa pela separação ou a segmentação dos objetos dentro destas imagens. O objetivo final da segmentação de imagens consiste em particionar a imagem em unidades que correspondam a objetos de interesse na cena. Segmentação de imagens é uma tarefa bastante complexa, pois envolve diversos conceitos e processos: a radiometria da formação das imagens, a geometria envolvida no processo de imageamento, as propriedades e características da cena e seus objetos constituintes [Costa, 2003].

Os algoritmos para segmentação são baseados em duas propriedades: descontinuidade e similaridade.

A descontinuidade numa imagem pode ser: um ponto, uma linha ou uma borda de um objeto. Neste contexto, serão apresentados alguns algoritmos de segmentação por detecção de bordas.

Já similaridade numa imagem está relacionada a características em comum num grupo de pixels. A segmentação por cores, que poderá ser utilizada, explora a cor dos pixels e a sua conectividade como estas características comuns.

2.1.1 SEGMENTAÇÃO POR DETECÇÃO DE BORDAS

Bordas são definidas como picos da magnitude do gradiente, ou seja, são variações abruptas que ocorrem ao longo de curvas baseadas nos valores do gradiente da imagem. As bordas são regiões da imagem onde ocorre uma mudança de intensidade em um certo intervalo do espaço, em uma certa direção. Isto corresponde a regiões de alta derivada espacial, que contém alta freqüência espacial.

Desde que uma borda é definida por uma mudança no nível de cinza, quando ocorre uma descontinuidade na intensidade, ou quando o gradiente da imagem tem uma variação abrupta, um operador que é sensível a estas mudanças operará como um detector de bordas.

Um operador de derivada faz exatamente esta função. Uma interpretação de uma derivada seria a taxa de mudança de uma função, e a taxa de mudança dos níveis de cinza em uma imagem é maior perto das bordas e menor em áreas constantes. Extraindo-se os valores da intensidade da imagem e determinando-se os pontos onde a derivada é um ponto de máximo, tem-se marcadas as suas bordas. Dado que as imagens são bidimensionais, é importante considerar mudanças nos níveis de cinza em muitas direções. Por esta razão, derivadas parciais das imagens são usadas, com as respectivas direções X e Y. Uma estimativa da direção atual da borda pode ser obtida usando as derivadas x e y como os componentes da direção ao longo dos eixos, e computar o vetor soma. O operador envolvido é o gradiente, e se a imagem é vista como uma função de duas variáveis $A(x,y)$ então o gradiente é definido como:

$$\nabla A(x,y) = \frac{\partial A}{\partial x}, \frac{\partial A}{\partial y} \tag{2.22}$$

Nas seções seguintes são apresentados os algoritmos mais tradicionais de detecção de bordas.

2.1.1.1 OPERADOR DE ROBERTS

É o mais antigo e simples algoritmo de detecção de bordas. Utiliza uma matriz 2x2 para encontrar as mudanças nas direções x e y [Lim, 1990]. A expressões 2.23 e 2.24 apresentam a máscara de Roberts:

$$G_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{2.23}$$

$$G_y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.24}$$

Para determinar onde o pixel avaliado é ou não um pixel de borda, o gradiente é calculado da seguinte forma:

$$|G| = \sqrt{G_x^2 + G_y^2} \tag{2.25}$$

Se a magnitude calculada é maior do que o menor valor de entrada (definido de acordo com a natureza e qualidade da imagem que esta sendo processada), o pixel é considerado ser parte de um borda. A direção do gradiente da borda, perpendicular a direção da borda, é encontrada com a seguinte fórmula:

$$\alpha = \arctan\frac{G_y}{G_x} \tag{2.26}$$

O pequeno tamanho da máscara para o operador de Roberts o torna fácil de se implementar e rápido para calcular a resposta. Entretanto, as respostas são muito sensíveis ao ruído na imagem.

2.1.1.2 OPERADOR DE SOBEL

Utiliza duas máscaras para encontrar os gradientes vertical e horizontal das bordas [Lim, 1990].

$$G_x = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.27}$$

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{2.28}$$

As fórmulas para encontrar o gradiente e o ângulo são as mesmas do operador de Roberts. O ângulo do gradiente corresponde à direção de máxima variação da intensidade. Devido às máscaras serem 3X3 ao invés de 2X2, Sobel é muito menos sensível ao ruído do que Roberts e os resultados são mais precisos. A computação de $|G|$, contudo, torna-se mais complexa. Na prática, $|G|$ é aproximada da seguinte

forma: $|G| = |G_x| + |G_y|$. O módulo do gradiente é proporcional a derivada local da intensidade.

2.1.1.3 OPERADOR DE ROBINSON

É similar em operação ao de Sobel, porém usa um conjunto de oito máscaras, onde quatro delas são as seguintes:

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.29}$$

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{2.30}$$

$$\begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix} \tag{2.31}$$

$$\begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix} \tag{2.32}$$

As outras quatro máscaras são simplesmente negações destas quatro. A magnitude do gradiente é o valor máximo obtido ao aplicar todas as oito máscaras ao pixel vizinho, e o ângulo do gradiente pode ser aproximado como o ângulo na linha de zeros na máscara dando a resposta máxima. Este algoritmo aumenta a precisão de $|G|$, mas requer mais computação do que Roberts e Sobel, devido à quantidade das máscaras.

2.1.1.4 DETECTOR DE BORDAS DE CANNY

Canny (1986) definiu um conjunto de metas que um detector de bordas deveria ter, são elas:

- **Taxa de erro**: o detector de bordas deveria detectar e achar somente bordas, nenhuma borda deveria faltar;
- **Localização**: a distância entre os pixels de borda encontradas pelo detector de bordas e a borda atual deveriam ser o menor possível;
- **Resposta**: o detector de bordas não deveria identificar múltiplos pixels de borda onde somente exista um único pixel.

O detector de bordas de Canny é um filtro de convolução *f* que uniformiza o ruído e localiza as bordas [Canny, 1986]. Contudo, o problema é identificar um filtro que otimize os três critérios do detector de bordas. Se for considerada uma borda de uma dimensão variando no contraste e então convolucionando a borda com a função de uniformização de Gauss, o resultado será uma variação contínua do valor inicial ao final, com uma inclinação máxima no ponto onde existe um "degrau". Se esta continuidade é diferenciada em relação à x, esta inclinação máxima será o máximo da nova função em relação a original (Figura 2.1).

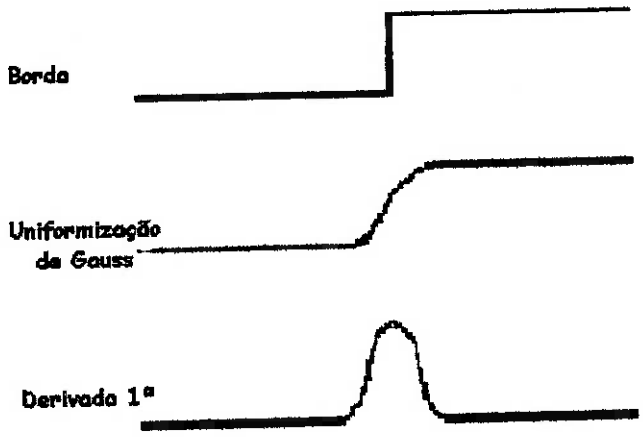

Figura 2.1. Detector de Bordas de Canny.

Os máximos da convolução da máscara e da imagem indicarão bordas na imagem. Este processo pode ser realizado através do uso de uma função de Gauss de 2-dimensões na direção de x e y. Os valores das máscaras de Gauss dependem da escolha do sigma na equação:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-x}{2\sigma}} \tag{2.33}$$

$$G'(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-x}{2\sigma}} \tag{2.34}$$

A aproximação do filtro de Canny para detecção de bordas é *G'*. Convolucionando a imagem com *G'* obtemos uma imagem *I* que mostrará as bordas, mesmo na presença de ruído. A convolução é relativamente simples de ser implementada, mas é cara computacionalmente, especialmente se for em 2-dimensões. Entretanto, uma convolução de Gauss de 2-dimensões pode ser separada em duas convoluções de Gauss de 1-dimensão.

A intensidade computacional do detector de bordas de Canny é relativamente alta, e os resultados são geralmente pós-processados para maior clareza. Entretanto, o algoritmo é mais eficiente no processamento de imagens com ruídos ou com bordas difusas.

O algoritmo de Canny [Canny, 1986] está descrito abaixo:

- Ler a imagem *I* a ser processada;
- Criar uma máscara de Gauss de 1-D *G* para convolucionar *I*. O desvio *S* de Gauss é um parâmetro para o detector de bordas;
- Criar uma máscara de 1-D para a primeira derivada de Gauss nas direções x e y; nomear como G_x e G_y. O mesmo valor *S* é usado;
- Convolucionar a imagem *I* com *G* percorrendo as linhas na direção *x*(*Ix*) e percorrer as colunas na direção *y*(*Iy*);
- Convolucionar *Ix*, com G_x, para dar *I'x* (o componente x de I convolucionado com a derivada de Gauss), e convolucionar *Iy*, com G_y para dar *I'y*;
- Para ver o resultado neste ponto os componentes x e y devem ser "combinados". A magnitude do resultado é computada para cada pixel (x,y).

2.1.1.5 OPERADOR DE MARR-HILDRETH

Um máximo da derivada primeira ocorrerá a cada *zero-crossing* (mudançado positivo para o negativo) da derivada segunda [Parker, 1997]. Para localizar bordas horizontais e verticais procura-se a derivada segunda nas direções de x e y. Isto é chamado laplaciano de *I*:

$$\nabla^2 I = \frac{\partial^2 I}{\partial^2 x}, \frac{\partial^2 I}{\partial^2 y} \tag{2.35}$$

O laplaciano é linear e simétrico rotacionalmente [Parker, 1997]. Pode-se procurar pelos *zeros-crossings* da imagem primeiro uniformizando com uma máscara de Gauss e então calculando a derivada segunda (Figura 2.2). Isto define o operador de Marr-Hildreth.

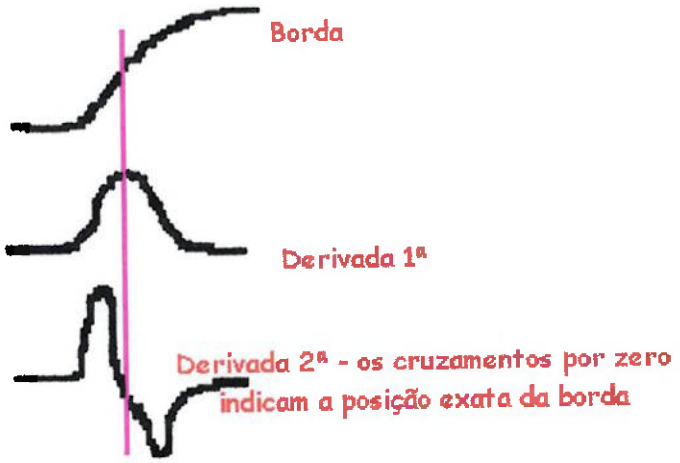

Figura 2.2. Operador de Marr-Hildreth (Laplaciano da Gaussiana).

O operador de Marr-Hildreth tornou-se largamente utilizado devido:

- a reputação de Marr;
- pesquisadores encontraram campos nos olhos de animais que se comportavam de forma semelhante com este operador;

- o operador é simétrico. Bordas são encontradas em todas as direções, diferente dos operadores que usam a primeira derivada (são unidirecionais).

Considerações sobre os operadores de derivadas de segunda ordem:

- *Zero-crossing* da derivada segunda são mais simples de determinar do que o ponto de máximo em uma derivada de primeira ordem;
- *Zero-crossing* sempre formam contornos fechados. Isto é bom quando se está tentando separar objetos em uma imagem;
- No operador da segunda derivada, o ruído é considerado. Este método é muitas vezes usado em conjunção com limiarização normal para reduzir sua sensibilidade a ruído;
- Marca bordas em algumas localizações que não são bordas.

Método de Marr-Hildreth:

- Aplica-se um filtro gaussiano à imagem;
- Obtém-se o Laplaciano; Localiza-se os cruzamentos por zero;
- Os pontos localizados formam contornos fechados.

2.1.2 SEGMENTAÇÃO POR COR

Segmentar uma imagem consiste em agrupar partes (pixels) desta imagem em unidades que sejam homogêneas em relação a uma ou mais características (atributos ou propriedades). A segmentação por cores define que a homogeneidade de tais unidades deve ser em relação às suas cores: pixels de mesma cor devem ser agrupados nas mesmas unidades. Desta forma, o processo de **classificação de cada pixel** na imagem como pertencente ou não a um certo grupo (ou classe) que possui determinada cor é tarefa de extrema importância na segmentação.

Embora aparentemente imediato, o conceito que envolve a palavra cor esconde muitas noções de ordem física e psicológica, o que torna a cor um conceito difícil de modelar. A classificação de cores consiste em particionar um espaço n-dimensional,

dado pela transformação da imagem de interesse num espaço apropriado de representação de cores [Costa, 2003].

Normalmente, uma restrição extra, **conectividade**, pode ser estipulada: essas partes com mesmas características, para serem agrupadas numa mesma unidade, devem ainda ser vizinhas entre si na imagem. Esta restrição vem do fato de que muitas vezes se deseja, no processo de segmentação de uma imagem, dividi-la em partes constituintes que representem objetos ou propriedades da cena.

O nível no qual esta segmentação é conduzida depende da aplicação. Assim, o processo de segmentação deve parar logo que os objetos de interesse em uma aplicação tiverem sido separados. Normalmente, o processo autônomo de segmentação de imagens é uma das tarefas mais difíceis do processamento de imagens.

2.1.2.1 Espaços de Representação de Cores

Os dispositivos digitais para representação de imagens coloridas utilizam o princípio da tricromaticidade, onde quase todas as cores podem ser reproduzidas como combinação linear de três cores básicas, chamadas cores primárias.

A generalização tricromática estabelece que, sobre uma ampla variedade de observações, muitos estímulos de cor podem ser obtidos através da mistura aditiva dos três estímulos de cor primária [Foley et al., 1990]. A proporção de cada estímulo, medida ao longo de uma escala arbitrária, pode variar. Uma determinada cor é, portanto, descrita por seus valores de triestímulos.

Observadores distintos têm a impressão de que uma mesma cor é formada por diferentes proporções dos triestímulos; assim, seus valores médios são usados para descrever uma cor. Em 1931, através de várias observações, definiram-se os valores médios adotados pela "Comission Internacionale de L'Éclairage - C.I.E." surgindo então o *observador padrão*; posteriormente, trabalhos complementares fizeram aparecer o *observador padrão C.I.E. de 1964* [Foley et al., 1990]

Os espaços tricromáticos de representação de cores mais usados são o *Red Green Blue* (RGB), o *Hue Saturation Value* (HSV) e o YUV.

RGB

Os terminais RGB possuem como primárias as cores aditivas vermelho (R), verde (G) e azul (B); baseiam-se na propriedade que estabelece ser a retina humana constituída por três tipos de fotopigmentos diferidos entre si pela sensibilidade relativa aos comprimentos de onda [Foley et al., 1990]

O modelo mais simples para tais dispositivos é o modelo RGB (*Red-Green-Blue*); esse modelo baseia-se na sensibilidade do olho, e usa um sistema de coordenadas cartesianas R, G, B, cujo subespaço de interesse é o cubo unitário mostrado na Figura 2.3.

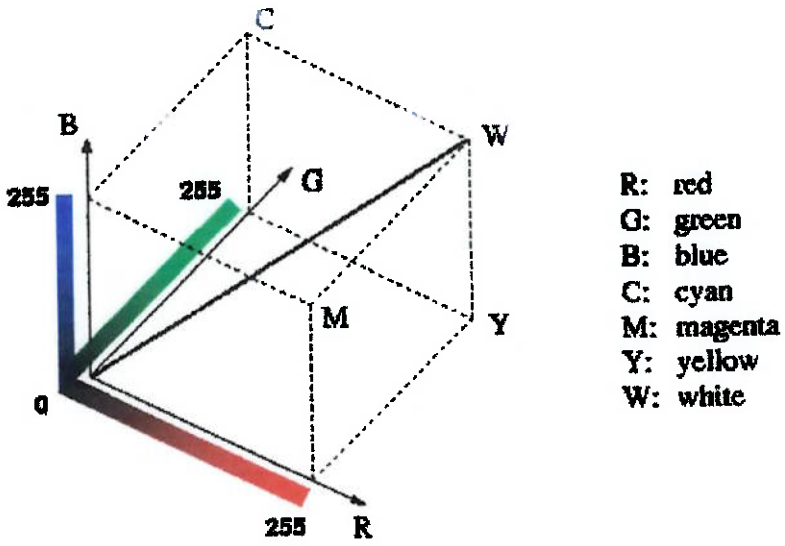

Figura 2.3. Espaço de representação de cores RGB.

As cores primárias RGB são aditivas. A diagonal principal do cubo possui quantidades iguais de cores primárias e representa a escala de cinza. Cada ponto colorido, dentro dos limites do cubo, pode ser representado por (R, G, B), onde os valores de R, G e B variam entre zero e um valor máximo.

Segundo Jackson; MacDonald; Freeman (1994), são vantagens do Cubo RGB:

- Grande simplicidade geométrica;
- Suporta um controle direto sobre o dispositivo, o que requer uma computação mínima;

- Facilidade de implementação.

Entretanto, apresenta as seguintes desvantagens:

- As coordenadas RGB em geral não são transferíveis, isto é, os mesmos valores de coordenadas não reproduzem exatamente a mesma cor em diferentes dispositivos;
- Não suporta dispositivos não aditivos (dispositivos não lineares como uma impressora que usa o modelo subtrativo CMYK); simplesmente assume que as primárias subtrativas são complementos das primárias aditivas (cyan do vermelho, etc.), o que leva à uma suposição de que as tintas também são aditivas, o que não é verdade;
- Não é perceptualmente uniforme, o que significa que uma variação de unidade da coordenada corresponde à diferentes variações perceptuais da cor dependendo da região do cubo;
- Como não se baseia em estímulos visuais e sim em sinais dos dispositivos (voltagens) o que faz com que não seja facilmente relacionado com a aparência da cor;
- Como não é um modelo intuitivo, não é de uso fácil para tarefas perceptuais.

HSV

O modelo HSV (*Hue, Saturation, Value*) orientado ao usuário, foi desenvolvido em 1978 por *Alvey Ray Smith* [Foley et al., 1990] que se baseou na maneira com que o artista descreve e mistura as cores. O sistema de coordenadas é um cilindro, e o sub-conjunto dentro do qual o modelo é definido, é um hexágono, ou uma pirâmide de seis lados, como mostrado na Figura 2.4.

A altura do hexágono é de uma unidade ao longo do eixo V. No eixo V, o valor de S é zero; trata-se da escala de "cinzas", onde o ponto V = 0 e S = 0 corresponde ao preto e o ponto V = 1 e S= 0 ao branco. No topo do hexágono encontram-se as cores "brilhosas" (mas elas não necessariamente possuem o mesmo brilho) e corresponde a V = 1.

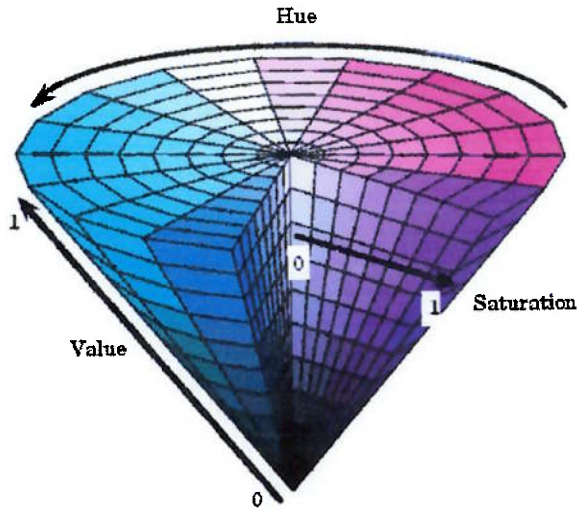

Figura 2.4. Espaço de representação de cores HSV.

A matiz ou H, é dada pelo ângulo ao redor do eixo vertical, sendo o vermelho igual a 0°, o verde 120° e assim por diante. Nesse modelo, as cores complementares encontram-se separadas uma da outra por um ângulo de 180°. O valor da saturação varia de 0, na linha central (eixo V), até 1, nos lados triangulares do hexágono.

As cores cujos valores de V e S são respectivamente 1, assemelham-se aos pigmentos primitivos usados por artistas; o acréscimo de pigmento branco corresponde à diminuição de S (sem alteração de V). As sombras são criadas pela diminuição de V (sem alteração de S). Os tons são gerados através do decréscimo de ambos, S e V. Alterações na componente H correspondem a uma escolha de um novo pigmento puro.

Segundo Jackson; MacDonald; Freeman (1994), são vantagens do modelo HSV:

- Simplicidade e facilidade de para implementação;
- Muito popular entre os programadores de Computação Gráfica.

Entretanto, apresenta as seguintes desvantagens:

- As coordenadas são definidas diretamente em termos dos sinais do monitor RGB, o que faz com que a cor produzida dependa das características do fósforo do monitor. Ao se trocar de equipamento, pode-se obter resultados

indesejados, uma vez que uma cor não é exatamente a mesma em diferentes dispositivos.

- Nenhum dos eixos é percetivamente uniforme; uma alteração em qualquer coordenada resulta em uma mudança aparentemente variável, de acordo com a localização no espaço de cor. O ângulo de matiz é notavelmente o menos uniforme, com mudanças lentas próximo aos ângulos referentes às primárias RGB e mudanças rápidas entre eles. Isso dificulta a tarefa do operador em precisar o efeito na cor devido a um ajuste dos controles HSV.
- Os três eixos não são de fato ortogonais (independentes um do outro). Por exemplo, a medida que se locomove ao redor do plano de cor H-S, a claridade (lightness) aparente no monitor sofrerá mudanças, mesmo que os valores da componente V permaneça constante. A maior discrepância ocorre entre o verde e o azul, que tipicamente difere na luminância por um fator próximo a cinco. Já uma mudança em V produz normalmente uma alteração na saturação aparente, mesmo que o componente S permaneça numericamente constante.

YUV

O YUV (ou YCbCr, que na verdade é o YUV normalizado) é um modelo derivado do RGB, onde busca-se separar as componentes cromáticas da luminância. Um modo simples de obter a decomposição crominância-luminância é proceder a subtração da componente Y (luminância) das outras componentes do sistema, obtendo (R-Y), (G-Y) e (B-Y). Uma vez que a quantidade de informação da componente verde é grande no eixo da luminância (expressão 2.35), adotaram-se os eixos (R-Y) e (B-Y), dando origem aos eixos U e V, a menos de alguma transformação linear que visa uma melhor distribuição dessas coordenadas pelo espaço [Simões, 2000]. A transformação do modelo RGB para o YUV pode ser vista nas equações 2.36, 2.37 e 2.38.

$$Y = 0{,}299R + 0{,}587G + 0{,}114B \tag{2.36}$$

$$U = 0{,}494(B\text{-}Y) \tag{2.37}$$

$$V = 0{,}877(R\text{-}Y) \tag{2.38}$$

O YUV apresenta as seguintes vantagens com relação aos outros modelos:

- O custo computacional do seu cálculo é baixo quando comparado ao HSV (coordenadas cilíndricas), já que realiza apenas operações de soma e subtração sobre as coordenadas dos RGB;
- É o modelo utilizado para a representação de cor no formato JPEG;
- Muitos dispositivos de captura de imagens comerciais fornecem saídas diretamente neste formato.

Entretanto apresenta as seguintes desvantagens:

- Apresenta alguns dos mesmos defeitos do RGB, já que é apenas uma simples conversão deste;
- A separação entre a luminância e a crominância não é tão nítida como no HSV;
- Em altos valores de luminância, as componentes cromáticas tornam-se instáveis.

A escolha do espaço de cores adequado para classificação depende de diversos fatores, incluindo disponibilidade do hardware de digitalização empregado e da utilidade para uma aplicação particular. A Figura 2.6 ilustra a distribuição das cores nestes três espaços.

2.2 SISTEMAS DE VISÃO ARTIFICIAL

Como já citado no começo deste capítulo, **Sistema de Visão Artificial** (SVA) é um sistema computadorizado capaz de adquirir, processar e interpretar imagens correspondentes a cenas reais [Marques Filho & Vieira Neto, 1999]. Muitos autores costumam descrever SVAs conforme o diagrama de blocos apresentado na Figura 2.5. Nesta figura, também está destacado o domínio de aplicação de cada uma das técnicas apresentadas.

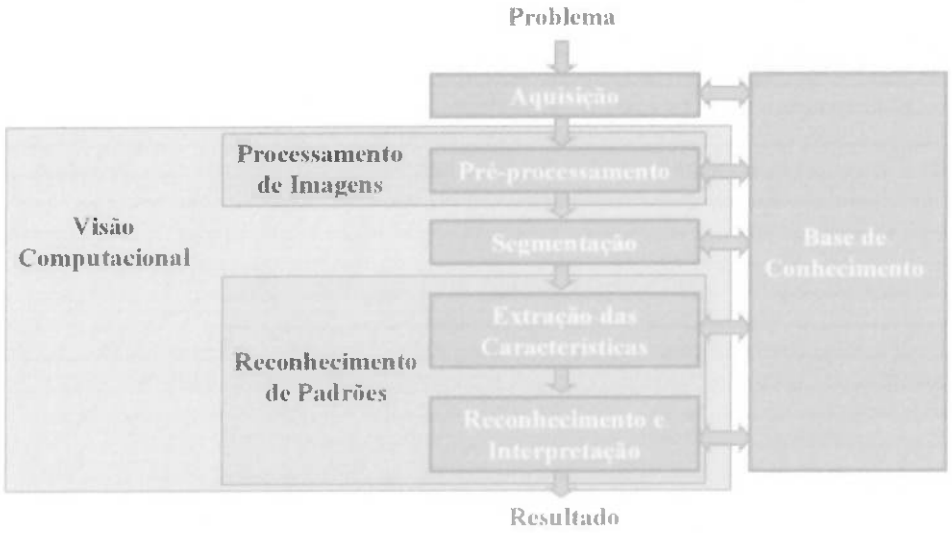

Figura 2.5. Um Sistema de Visão Artificial (SVA) e suas principais etapas.

2.2.1 AQUISIÇÃO

O primeiro passo de processamento de um SVA é a aquisição das imagens do objeto de interesse da análise. Para tanto, são necessários um sensor e um digitalizador. O sensor converterá a informação óptica em sinal elétrico e o digitalizador transformará a imagem analógica em imagem digital [Marques Filho & Vieira Neto, 1999].

2.2.2 PRÉ-PROCESSAMENTO

A imagem resultante do passo anterior pode apresentar diversas imperfeições tais como: presença de pixels ruidosos, contraste e/ou brilho inadequados, etc. A função da etapa de pré-processamento é aprimorar a qualidade da imagem para as etapas subseqüentes. As operações efetuadas nesta etapa são ditas de baixo nível porque trabalham diretamente com a intensidade e a conectividade dos pixels sem nenhum conhecimento sobre quais deles pertencem ao objeto de interesse ou ao fundo [Marques Filho & Vieira Neto, 1999]. Basicamente é nesta etapa onde as técnicas de processamento de imagens são aplicadas.

O resultado deste pré-processamento é uma imagem digitalizada de melhor qualidade que a original.

2.2.3 SEGMENTAÇÃO

A tarefa básica da etapa de segmentação é a de destacar os objetos de interesse em uma imagem. Apesar de intuitiva para os seres humanos, esta tarefa, dependendo do que se deseja analisar, é um das mais complexas de um SVA. Esta é uma fase típica de aplicação de algumas técnicas de visão computacional, dentre elas as segmentações por cor e por bordas, apresentadas na Seção 2.2.

O resultado desta etapa são as "sub-imagens" da imagem original contendo os objetos de interesse da análise.

2.2.4 EXTRAÇÃO DAS CARACTERÍSTICAS

Nesta etapa procura-se extrair características das "sub-imagens" resultantes da segmentação através de descritores que permitam definir com precisão o objeto de interesse e que apresentem bom poder de discriminação entre diferentes objetos. Estes descritores devem ser representados por uma estrutura de dados adequada ao algoritmo de reconhecimento [Marques Filho & Vieira Neto, 1999].

O resultado desta etapa são conjuntos de dados estruturados correspondente às "sub-imagens" de entrada.

2.2.5 RECONHECIMENTO E INTERPRETAÇÃO

Nesta última etapa do sistema, denomina-se reconhecimento o processo de atribuição de um rótulo a um objeto baseado em suas características, traduzidas pelos seus descritores. A tarefa de interpretação, por outro lado, consiste em atribuir significado a um conjunto de objetos reconhecidos [Marques Filho & Vieira Neto, 1999].

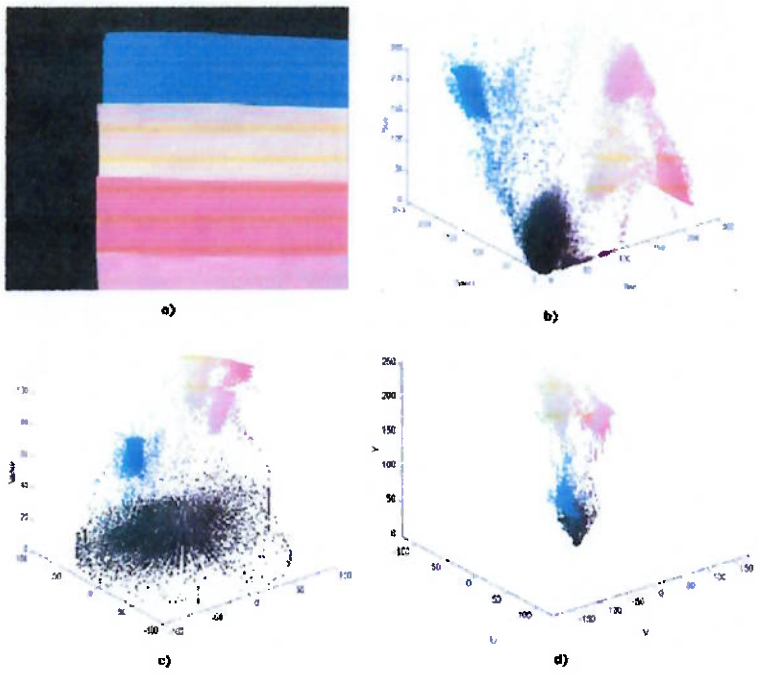

Figura 2.6. Exemplo de a) uma imagem representada no b) espaço RGB de cores, c) espaço HSV e d) espaço YUV [Costa, 2003].

2.2.6 BASE DE CONHECIMENTO

Todas as tarefas descritas anteriormente pressupõem a existência de um conhecimento sobre o problema a ser resolvido, armazenado em uma base de conhecimento, cujo tamanho e complexidade podem variar enormemente. Idealmente, esta base de conhecimento deveria não somente guiar o funcionamento de cada etapa, mas também permitir a realimentação entre elas [Marques Filho & Vieira Neto, 1999]. Em muitos casos, esta base de conhecimento pode não estar explícita, mas dividida e incorporada nas diversas etapas citadas.

3 MATERIAIS E MÉTODOS

3.1 MÓDULO DE ENTRADA DE DADOS

O módulo de entrada é responsável pela comunicação entre o usuário e o computador. Através deste módulo o usuário será solicitado a fornecer informações do mundo externo de modo a criar condições ao sistema para extrair os parâmetros necessários. Será possível ainda entrar com informações assim como escolher o lote de imagens a serem processadas.

3.2 MÓDULO DE AQUISIÇÃO DE IMAGENS

Para a aquisição serão reaproveitadas as imagens colidas em experimentos prévios de desgaste de ferramenta, assim como, novas imagens obtidas em ambiente com iluminação controlada. Tarefa que tem demonstrado ser um dos maiores desafios do projeto, uma vez que não é possível definir uma situação ideal de iluminação para todas as geometrias das ferramentas a serem testadas devido à vasta gama de modelos que encontramos. Algumas ferramentas apresentam ainda particularidades que dificultam a iluminação, tal como cavidades profundas para saída de cavaco que ocasionalmente geram sombra nas imagens diminuindo a precisão de medida do sistema.

O módulo de aquisição de imagens, embora não faça parte do escopo do projeto desenvolver um sistema para este fim, tem essencial importância nos resultados como um todo. Sistemas de visão artificial não possuem informações externas sobre o sistema que esta sendo analisado, dificultando bastante qualquer tipo de correção em imagens que não apresentem qualidade suficiente para processamento e extração de atributos.

3.3 MÓDULO DE DETECÇÃO DA FERRAMENTA

O Módulo de detecção da ferramenta visa isolar a ferramenta criando uma imagem com fundo neutro e com intensidade máxima. Desta forma poderemos processar apenas a região de interesse.

Ver figura:

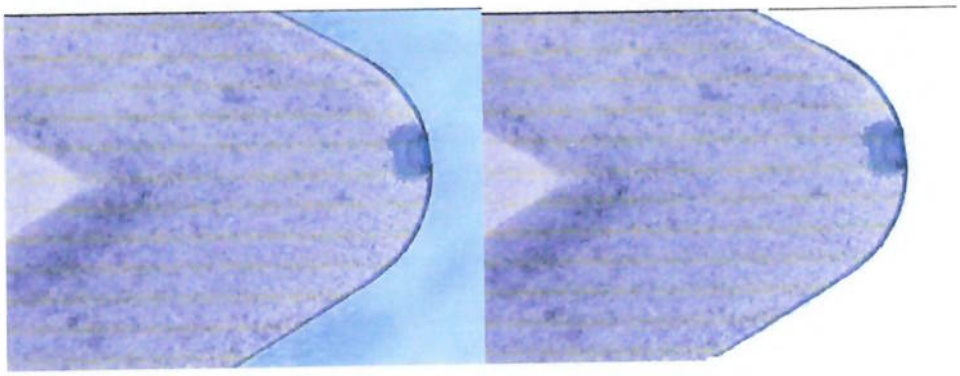

Figura 3.1 Exemplo de subtração do fundo de imagem

Através deste método, poderemos também isolar a região onde a ferramenta se encontra de modo a diminuir o tempo de processamento, pois não mais precisaremos analisar a imagem inteira e sim apenas a região conforme figura.

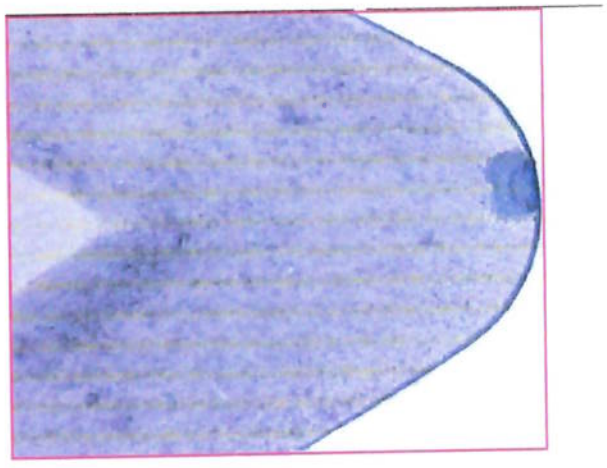

Figura 3.2 Limitando a região de processamento

Espera-se ainda que este método diminua o tempo de procura pela melhor posição de encaixe entre duas imagens que possam apresentar variação no seu posicionamento frente à câmera. Este método poderá ser implementado futuramente.

3.4 Módulo de Detecção de Bordas

O módulo de detecção de borda conforme discutido anteriormente, identificará o contorno da ferramenta possibilitando a criação de uma imagem máscara para o processamento e extração dos parâmetros necessários. Este módulo tem como saída um vetor com os pontos do contorno identificado, este contorno será limitado pelo seu tamanho máximo e mínimo a fim de evitar problemas com ruídos (identificação de contornos falsos).

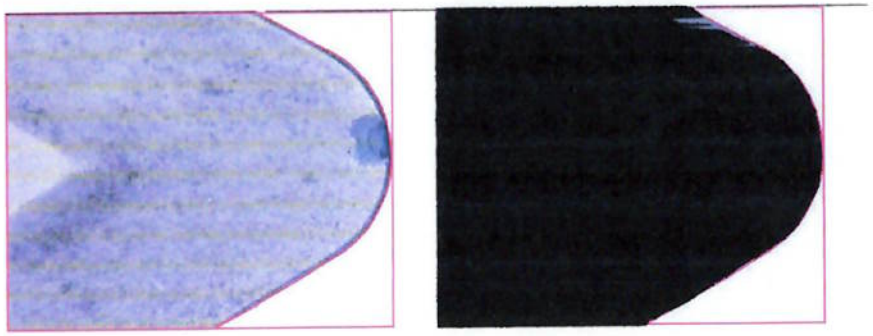

Figura 3.1 Exemplo de mascara para processamento

3.5 Módulo de Extração de Parâmetros

O programa deverá identificar, plotar e gerar um arquivo com as coordenadas de cada ponto da imagem que está sendo analisada.

3.6 MÓDULO DE EXIBIÇÃO

O módulo de exibição apresentará ferramentas para que seja possível selecionar a área da imagem que será processada. Além disso, será possível visualizar a imagem processada ou não e obter ajustes em tempo real dos parâmetros de processamento.

4 PROTÓTIPO

4.1 VISÃO GERAL

O protótipo foi desenvolvido utilizando ferramentas de visão computacional, apoiados por ferramentas de programação para Windows.

4.2 LINGUAGEM DE PROGRAMAÇÃO

4.2.1 C++

Os algoritmos de processamento de imagens foram desenvolvidos utilizando C++ como linguagem de programação, pois alem de ser esta uma ferramenta de desenvolvimento de baixo nível, aumentando assim o processo de otimização da velocidade de processamento do código, também é a linguagem de programação utilizada em alguns SDKs (software Development Kit) para Windows.

Algumas opções foram analisadas para a escolha do SDK a ser utilizado, dentre elas GDI, GDI+, OpenCV highgui, DirectXSDK e MFC. Apesar da maior complexidade da estrutura envolvida na utilização do SDK DirectX, optou-se pelo mesmo por apresentar ampla integração da estrutura de processamento de imagens com os diversos sistemas de cor e de exibição operantes atualmente, não sendo necessário a conversão entre os principais métodos de compressão de imagens, tais como JPG, GIF para BMP (imagem sem compressão). Esta plataforma de desenvolvimento possibilita ainda tanto o processamento de vídeos quanto imagens utilizando a mesma estrutura.

4.2.2 DELPHI

A interface com o usuário foi desenvolvida utilizando Delphi como linguagem de programação, sendo esta uma das mais poderosas ferramentas disponível no mercado para esta finalidade. Fornecendo rápida integração com operações rotineiras do sistema operacional, como diálogos de abertura e salvamento de arquivos, e inúmeras ...gens operacionais que agilizam o processo de criação de interfaces com o usuário.

Para a integração entre os dois módulos, processamento e interface com usuário, foi utilizado uma estrutura de encapsulamento e integração de sistemas chamada ActiveX, permitindo mobilidade entre dados dos dois módulos e tornando possível a integração do sistema.

4.3 ENSAIO PRELIMINARES

Como ferramenta de apoio, utilizou-se o MathLab para ensaiar o melhor módulo de detecção de borda para o problemas proposto. Conforme citado na seção 2.1, são muitos os algoritmos de detecção de borda propostos atualmente, sendo então necessário a analise daquele que melhor se encaixe na analise da detecção do desgaste da ponta da ferramenta. Abaixo seguem algumas imagens processadas utilizando esses algoritmos.

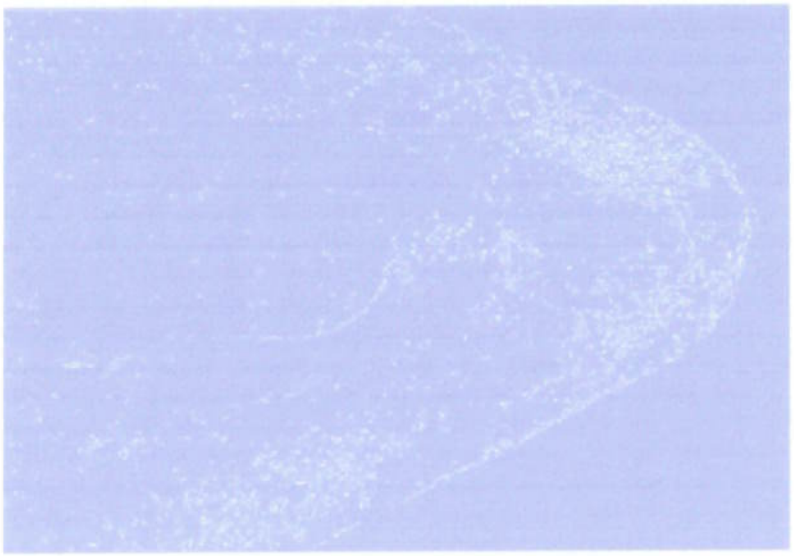

Figura 4.1. Exemplo de detecção de borda utilizando o algoritmo de Sobel

Figura 4.2. Exemplo de detecção de borda utilizando o algoritmo de Prewitt

Figura 4.3. Exemplo de detecção de borda utilizando o algoritmo de Roberts

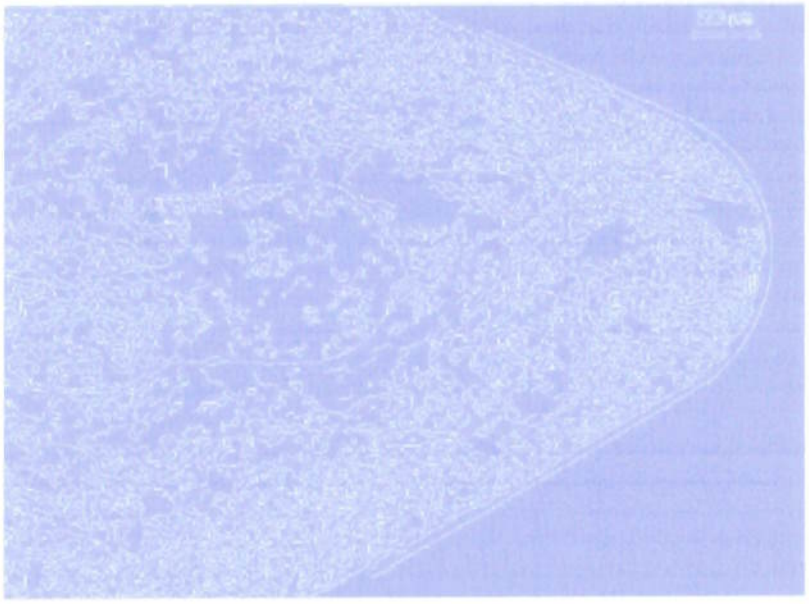

Figura 4.3. Exemplo de detecção de borda utilizando o algoritmo de Roberts

Figura 4.3. Exemplo de threshold

4.3.1 ANALISE DOS ALGORITIMOS

Acima encontramos algumas das imagens processadas com os algoritmos estudados na seção 2.1.1 através de testes no MathLab. A analise dos resultados permitiu observar que os algoritmos de Sobel, Prewitt, Roberts e Canny por utilizarem informações dos pixels vizinhos apresentam um auto grau de ruído nos contornos obtidos. Já no threshold simples foi possível obter uma maior uniformidade na analise do contorno, além disso, as condições de iluminação a cada ensaio poderão ser reavaliadas para facilitar a identificação da borda por este processo, aumentando o contraste entre o objeto de interesse e o background.

Sendo assim, o software foi projetado para trabalhar com a identificação de borda a partir de uma imagem binária.

4.4 ENTRADA DO SISTEMA

O programa aceitará imagens no formato bmp ou jpg. Para facilitar o processamento de uma seqüência de imagens, foi projeto um modulo que permite a navegação através de um grupo de imagens a serem processados, não sendo necessário a abertura individual de cada arquivo.

4.5 INTERFACE COM O USUÁRIO

A interface com o usuário é composta pelos componentes demonstrados abaixo:

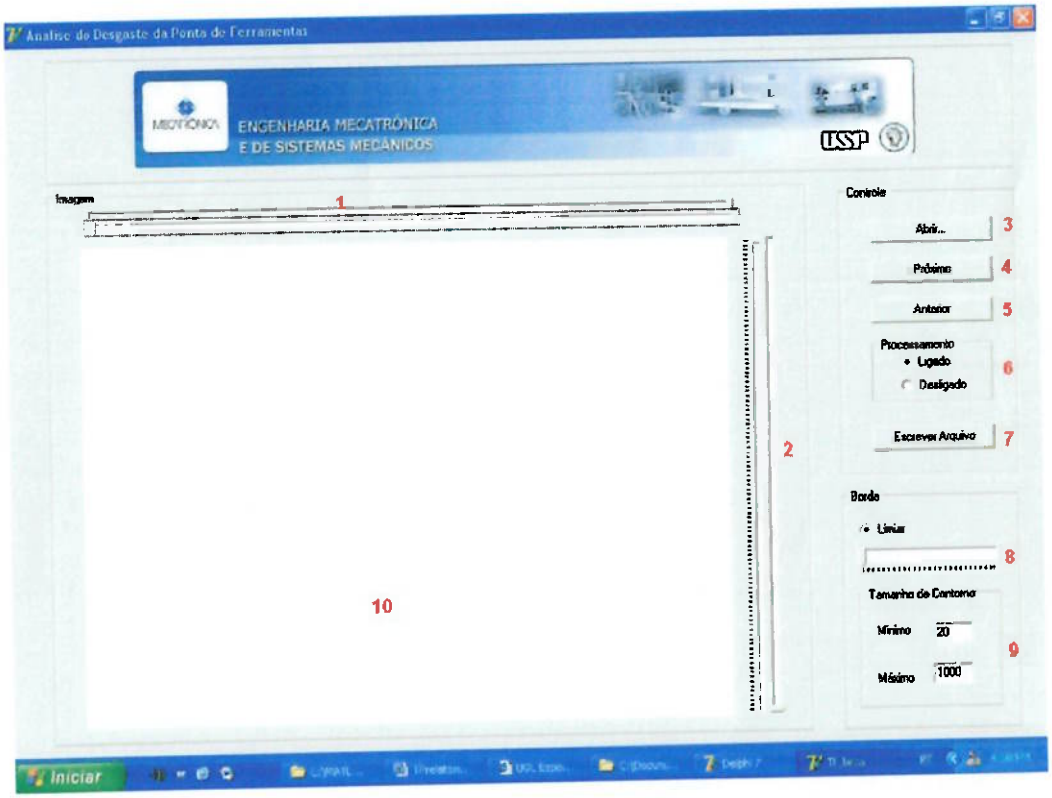

Figura 4.5.1 Interface do Programa

1. **Barra de rolagem horizontal** - permite selecionar a região da imagem que será processada.
2. **Barra de rolagem vertical** - permite selecionar a região da imagem que será processada.
3. **Botão abrir** – abre as imagens a serem processadas
4. **Botão próximo** – avança para a próxima imagem
5. **Botão anterior** – retorna para a imagem anterior
6. **Processamento (ligado-desligado)** – alterna entre processamento ligado e desligado (imagem original)
7. **Escrever Arquivo** – escreve um arquivo texto com todas as coordenadas de todos os pontos que compõem o contorno encontrado.
8. **Limiar** – abaixo desse valor todo pixel será considerado negro, acima todo pixel será considerado branco
9. **Tamanho do Contorno (Maximo-Minimo)** – Mínimo e maximo valores para que um contorno encontrado não seja desprezado pelo algoritimo.

Pode-se assim eliminar ruídos da imagem eliminando os contornos isolados menores que um determinado valor.

4.6 REGIÃO DE PROCESSAMENTO

Devido às irregularidades dos objetos de analise e como nem sempre desejamos que toda imagem seja processada, foi desenvolvida uma ferramenta de isolamento da área de analise. Para facilitar a visualização do processo, foi ainda mantida a imagem original, sendo que apenas a região processada apresenta alguma alteração.

Ainda com o intuito de facilitar o ajuste durante o processo de obtenção da borda, optou-se por utilizar um threshold não binário, criando uma região unicolor e mantendo as características em toda a região que estiver abaixo do limiar estipulado.

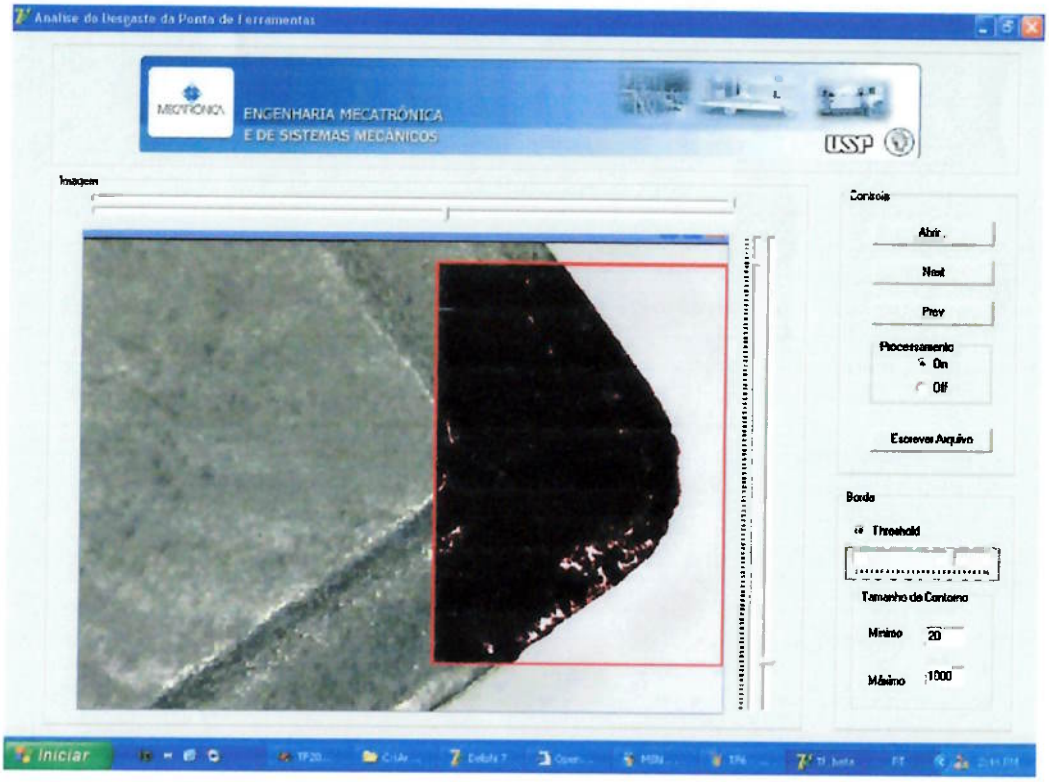

Figura 4.6.1 Regiao de Processamento

Na figura acima, podemos observar que o fundo da imagem manteve suas características originais, facilitando o ajuste visual da borda. A região de processamento pode ser alterada a qualquer instante sendo que o processamento da imagem será automaticamente refeito.

4.7 AJUSTE DO LIMIAR E TAMANHO DO CONTORNO

Uma vez que as condições de iluminação e de fundo variam muito a cada ensaio, foi necessária a criação de uma ferramenta de ajuste do limiar de corte para identificação da borda, dando assim liberdade de mudanças das condições de aquisição de novas imagens e tornando possível o ajuste para aquelas já adquiridas. Os limites de tamanho de contorno são medidos em pixels e foram necessários para a eliminação de ruídos, manchas e outras impurezas do processo que não estão no escopo de analise.

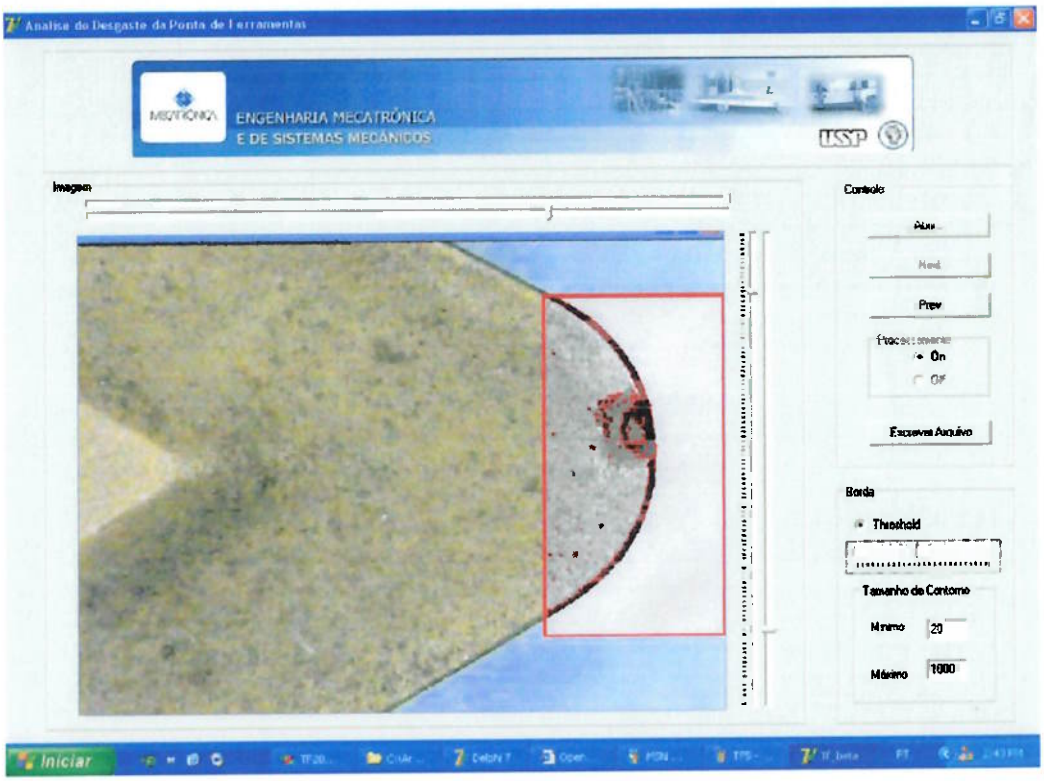

Figura 4.7.1 Limite do tamanho do contorno

Exemplo de imagem de ferramenta com manchas secundaria que podem ser eliminadas através da correta seleção da região de analise, ajuste do limiar e tamanhos máximos e mínimos dos contornos.

5 RESULTADOS E DISCUSSÃO

O sistema apresenta grande versatilidade quanto ao ajuste de parâmetros para obtenção da identificação de borda. Uma vez que todo o processo esta amarrado as condições de aquisição de imagem, os resultados foram os esperados permitindo que as coordenadas do contorno da ferramenta fossem adquiridas de modo automático e com a possibilidade de ajustes para atender a variação de cor das peças e eventuais alterações nas condições de iluminação.

Sendo o protótipo uma ferramenta que além de fornecer dados quantitativos permite que haja um retorno visual para o processo, torna possível o ajuste das condições de iluminação de maneira integrada, bastando para isso processar uma imagem e visualmente verificar se as características da imagem obtida atenderão ao propósito de avaliação do desgaste da ponta da ferramenta.

Os resultados quantitativos da analise do desgaste da ponta de ferramentas poderá ser executado em trabalho futuro tendo como entrada do sistema o arquivo com as coordenadas do contorno da ferramenta geradas a partir do protótipo desenvolvido neste trabalho.

6 CONCLUSÕES

A vasta variedade de ferramentas existentes no mercado reduziu a abordagem do sistema como um todo, sendo que alguns dos algoritmos propostos inicialmente não puderam ser aproveitados. Como exemplo pode citar a segmentação por cor que, apesar de ser uma poderosa ferramenta de analise, quando submetida a sistemas com grande variedade de cores e intensidades acaba tornando impraticável a obtenção de padrões que pudessem gerar uma correta modelagem do sistema.

Para estudos futuros, é possível a integração do sistema com equipamentos de aquisição em tempo real, toda a estrutura foi montada levando em consideração essas ampliações do projeto. Além disso, a estrutura atual permite que vídeos sejam analisados bastando para isso pequenas alterações no programa atual. Tornando possível a avaliação de um grande numero de imagens e acompanhamento da evolução do desgaste em um ensaio controlado.

Fica ainda como desafio a obtenção das melhores condições de iluminação para a detecção do desgaste da ponta da ferramenta, pois esta depende não só do arranjo a ser formado, mas do formato, cor e tipo de desgaste que a ferramenta será submetida.

LISTA DE REFERÊNCIAS

BALLARD, D.; BROWN, C. **Computer Vision**. Englewood Cliffs, NJ: Prentice Hall, 1982.

BURKE, M.W. **Image Aquisition**. 1.ed. Oxford, UK: Chapman & Hall, 1996. (Handbook of Machine Vision Engineering).

CANNY, J.A. Computational Approach to Edge Detection. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. PAMI-8, n. 6, pp.679-698, 1986.

GONZALES, R. C.; WOODS, R. E. **Digital Image Processing**. 2.ed. Boston, MA: Addison-Wesley publisher Company, 1990.

COSTA, A.H.R. **Robótica Móvel Inteligente: progressos e desafios**. 2003. Tese (Livre-Docência) – Escola Politécnica, Universidade de São Paulo. São Paulo.

www.microsoft.com.br

```
// TF2005_AnaliseDesgProcessor.h
//
// Description:
//          Interface for the CTF2005_AnaliseDesgProcessor class.
//
// Author:  // TODO: Eduardo Finamore.
//
// Version: // TODO: Version number.
// Date:    // TODO: Version release date.
// Version Note:
//          // TODO: Notes about the version.
//
// Note:    // TODO: Write general notes here
//////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////
// Compiler macros

#if !defined(AFX_TF2005_ANALISEDESGPROCESSOR_H__6903746A_20B9_46C0_BC20_915AE4A585EE__INCL
UDED_)
#define AFX_TF2005_ANALISEDESGPROCESSOR_H__6903746A_20B9_46C0_BC20_915AE4A585EE__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "cv.h"
#include "Control.h"

//////////////////////////////////////////////////////////////////////
// Preview declaration.

struct ImageData;

//////////////////////////////////////////////////////////////////////
// CTF2005_AnaliseDesgProcessor class interface.

class CTF2005_AnaliseDesgProcessor
{
public:
    //////////////////////////////////////////////////////////////////////
    // Constructor and destructor.
    //////////////////////////////////////////////////////////////////////
    CTF2005_AnaliseDesgProcessor();
    virtual ~CTF2005_AnaliseDesgProcessor();

    //////////////////////////////////////////////////////////////////////
    // Buffer processing function.
    //////////////////////////////////////////////////////////////////////
    void Process(ImageData* );
    CControl* m_control;



};

#endif // !defined(AFX_TF2005_ANALISEDESGPROCESSOR_H__6903746A_20B9_46C0_BC20_915AE4A585EE
__INCLUDED_)
```

```
// Control.h: interface for the CControl class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_CONTROL_H__B2B73935_59C1_40D0_881E_1B0AC766E26A__INCLUDED_)
#define AFX_CONTROL_H__B2B73935_59C1_40D0_881E_1B0AC766E26A__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CControl
{
protected:
    CControl();
public:
    void setROI(int xLeft,int xRight,int yBack,int yTop);
    virtual ~CControl();
    static CControl* instance();
    static CControl* object;

    int m_xLeft;
    int m_xRight;
    int m_yBack;
    int m_yTop;


    int m_thresh1;
    int m_thresh2;
    int m_aperture;

    int m_thresh1_canny;
    int m_thresh2_canny;
    int m_aperture_canny;

    bool m_isThreshold;

    int minimal_contour_size;
    int maximal_contour_size;

    double m_comprimentoMedidoCalibracao;

    CString m_strFileName2Save;
    bool m_bWriteFile;


    int m_atualImageWidth;
    int m_atualImageHeight;

    CString m_fileName;

};

#endif // !defined(AFX_CONTROL_H__B2B73935_59C1_40D0_881E_1B0AC766E26A__INCLUDED_)
```

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by TF2005_AnaliseDesg.rc
//
#define IDS_VIDEOCONTROL                1
#define IDD_ABOUTBOX_VIDEOCONTROL       1
#define IDB_VIDEOCONTROL                1
#define IDI_ABOUTDLL                    1
#define IDS_VIDEOCONTROL_PPG            2
#define IDS_VIDEOCONTROL_PPG_CAPTION    200
#define IDD_PROPPAGE_VIDEOCONTROL       200
#define IDD_DIALOG_SNAPSHOT             201
#define IDC_BUTTON_CAPTURE              201
#define IDC_BUTTON_ERASE                202
#define IDC_STATIC_COUNT                203

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        202
#define _APS_NEXT_COMMAND_VALUE         32768
#define _APS_NEXT_CONTROL_VALUE         204
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

```
   SnapShot.h
//
// Description:
//          Interface of CSnapShot class.
//
// Author:  // TODO: Author name.
//
// Version: // TODO: Version number.
// Date:    // TODO: Version release date.
// Version Note:
//          // TODO: Notes about the version.
//
// Note:    // TODO: Write general notes here
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// Compiler macros

#if !defined(AFX_SNAPSHOT_H__B48F0F4C_8418_48AE_95F1_F6BCF98741CB__INCLUDED_)
#define AFX_SNAPSHOT_H__B48F0F4C_8418_48AE_95F1_F6BCF98741CB__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

////////////////////////////////////////////////////////////////////////////
// Headers

#include "resource.h"
#include "TF2005_AnaliseDesgProcessor.h"
#include <list>

////////////////////////////////////////////////////////////////////////////
// ImageData structure
// Store image data with extra information
////////////////////////////////////////////////////////////////////////////
struct ImageData{
    ImageData() : pData(NULL),iPixelSize(0),iSizeX(0),iSizeY(0),lDataLen(0) {}
    BYTE*   pData;
    int     iPixelSize;
    int     iSizeX;
    int     iSizeY;
    int     iSizeBytes;
    int     iNumPixels;
    long    lDataLen;
};

////////////////////////////////////////////////////////////////////////////
// CSnapShot dialog

class CSnapShot : public CDialog
{

public:
    ////////////////////////////////////////////////////////////////////////
    // Constructor and destructor
    ////////////////////////////////////////////////////////////////////////
    CSnapShot(CWnd* pParent = NULL);
    ~CSnapShot();

    ////////////////////////////////////////////////////////////////////////
    // Methods used for the video acquisition class.
    ////////////////////////////////////////////////////////////////////////
    /// Add a copy of the image to buffer.
    void addImage(ImageData* frame);
    /// Tests if a snapshot were requested.
    BOOL isCapturing();
    /// Fills current buffer with the last captured image,
    void fillBuffer(BYTE*);
    /// Set Processor
    void setProcessor(CTF2005_AnaliseDesgProcessor*);

    ////////////////////////////////////////////////////////////////////////
    // Methods related to the use of SnapShot class.
```

```
        ////////////////////////////////////////////////////////////////////////
        // If the last snapshot is to be displayed.
        void ShowSnapShot(BOOL b);
        // Capture a single frame.
        void CaptureFrame();
        // Capture multiple frames.
        void CaptureSequence(long num_frames);
        // If a capture dialog box is to be shown.
        void ShowDialog(BOOL);
        // Saves the last snapshot image to a bitmap file.
        BOOL SaveToBitmap(CString);

        ////////////////////////////////////////////////////////////////////////
        // Automation methods.
        ////////////////////////////////////////////////////////////////////////
    // Dialog Data
        //{{AFX_DATA(CSnapShot)
        enum { IDD = IDD_DIALOG_SNAPSHOT };
        //}}AFX_DATA


    // Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CSnapShot)
        public:
        virtual void OnFinalRelease();
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

    // Implementation
    protected:

        // Generated message map functions
        //{{AFX_MSG(CSnapShot)
        afx_msg void OnButtonCapture();
        afx_msg void OnButtonErase();
        virtual BOOL OnInitDialog();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
        // Generated OLE dispatch map functions
        //{{AFX_DISPATCH(CSnapShot)
        //}}AFX_DISPATCH
        DECLARE_DISPATCH_MAP()
        DECLARE_INTERFACE_MAP()

    // Attributes
    private:
        int                         m_iDlgID;           //   Dialog box id.
        BOOL                        m_bShowSnapShot;    ///< If the snap shot is to be shown.
        long                        m_lCapturing;       ///< Number of frames to be captured.
        std::list<ImageData>        m_imageBuffer;      ///< Image buffer.
        CTF2005_AnaliseDesgProcessor*   m_pProcessor;   ///< Image processor.
    };

    //{{AFX_INSERT_LOCATION}}
    // Microsoft Visual C++ will insert additional declarations immediately before the previou
s line.

    #endif // !defined(AFX_SNAPSHOT_H__B48F0F4C_8418_48AE_95F1_F6BCF98741CB__INCLUDED_)
```

```
#if !defined(AFX_STDAFX_H__AA297431_6427_4570_9452_6A6BBF13AA51__INCLUDED_)
#define AFX_STDAFX_H__AA297431_6427_4570_9452_6A6BBF13AA51__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

// stdafx.h : include file for standard system include files,
//      or project specific include files that are used frequently,
//      but are changed infrequently

#define VC_EXTRALEAN        // Exclude rarely-used stuff from Windows headers

#include <afxctl.h>         // MFC support for ActiveX Controls
#include <afxext.h>         // MFC extensions
#include <afxdtctl.h>       // MFC support for Internet Explorer 4 Comon Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>         // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT

// Delete the two includes below if you do not wish to use the MFC
//  database classes
#include <afxdb.h>          // MFC database classes
#include <afxdao.h>         // MFC DAO database classes

//////////////////////////////////////////////////////////////////////
// Debug code

#ifdef _DEBUG
    #define     DEBUG_MESSAGE(x,y)  MessageBox(NULL,_T(x),_T(y),0)
#else
    #define     DEBUG_MESSAGE(x,y)
#endif

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previou
s line.

#endif // !defined(AFX_STDAFX_H__AA297431_6427_4570_9452_6A6BBF13AA51__INCLUDED_)
```

```
////////////////////////////////////////////////////////////////////////////
// TF2005_AnaliseDesg.h
//
// Description:
//          Main header file for TF2005_ANALISEDESG.DLL
//
// Author:  // TODO: Author name.
//
// Version: // TODO: Version number.
// Date:    // TODO: Version release date.
// Version Note:
//          // TODO: Notes about the version.
//
// Note:    // TODO: Write general notes here
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// Compiler macros

#if !defined(AFX_TF2005_ANALISEDESG_H__D74D1699_FDF9_47FE_95CB_2D63CA22CB8E__INCLUDED_)
#define AFX_TF2005_ANALISEDESG_H__D74D1699_FDF9_47FE_95CB_2D63CA22CB8E__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#if !defined( __AFXCTL_H__ )
    #error include 'afxctl.h' before including this file
#endif

////////////////////////////////////////////////////////////////////////////
// Headers
#include "resource.h"       // main symbols

////////////////////////////////////////////////////////////////////////////
// CTF2005_AnaliseDesgApp : See TF2005_AnaliseDesg.cpp for implementation.

class CTF2005_AnaliseDesgApp : public COleControlModule
{
public:
    BOOL InitInstance();
    int ExitInstance();
};

extern const GUID CDECL _tlid;
extern const WORD _wVerMajor;
extern const WORD _wVerMinor;

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previou
s line.

#endif // !defined(AFX_TF2005_ANALISEDESG_H__D74D1699_FDF9_47FE_95CB_2D63CA22CB8E__INCLUDE
D)
```

```
TF2005_AnaliseDesgAcquisition.h

// Description:
//          Interface of the CTF2005_AnaliseDesgAcquisition class
//
// Author:  // TODO: Eduardo Finamore.
//
// Version: // TODO: Version number.
// Date:    // TODO: Version release date.
// Version Note:
//          // TODO: Notes about the version.
//
// Note:    // TODO: Write general notes here
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
#pragma once

////////////////////////////////////////////////////////////////////////////
// Headers
#include "VideoGraphBase.h"
#include "SnapShot.h"

class CTF2005_AnaliseDesgProcessor;

////////////////////////////////////////////////////////////////////////////
// Class that allows you to process the buffer.
//
// CTF2005_AnaliseDesgAcquisition : Video playback, capturing, seeking and grabbing
//
//  1. Functions:
//      StartPreview();     // Begin capture using specified device
//      void CaptureToFile();       // setup live capture
//  2. Message handler:
////////////////////////////////////////////////////////////////////////////
class CTF2005_AnaliseDesgAcquisition : public CVideoGraphBase
                                , public ISampleGrabberCB
{
public:
    ////////////////////////////////////////////////////////////////////////
    // Class constructor and destructor.
    ////////////////////////////////////////////////////////////////////////
    CTF2005_AnaliseDesgAcquisition();
    virtual ~CTF2005_AnaliseDesgAcquisition();

protected:
    ////////////////////////////////////////////////////////////////////////
    // The interface needed by ISampleGrabber Filter
    ////////////////////////////////////////////////////////////////////////
    // fake out any COM ref counting
    STDMETHODIMP_(ULONG) AddRef() { return 2; }
    STDMETHODIMP_(ULONG) Release() { return 1; }
    // fake out any COM QI'ing
    STDMETHODIMP QueryInterface(REFIID riid, void ** ppv) {
        if( riid == IID_ISampleGrabberCB || riid == IID_IUnknown ) {
            *ppv = (void *) static_cast<ISampleGrabberCB*> ( this );
            return NOERROR;
        }
        return E_NOINTERFACE;
    }

    // The frame callback function called on each frame
    // We can add the processing code in this function
    STDMETHODIMP SampleCB(double SampleTime, IMediaSample *pSample);
    // Another callback function. I am not using it now.
    STDMETHODIMP BufferCB(double SampleTime, BYTE *pBuffer, long BufferLen) {
        return E_NOTIMPL;
    }

    // We have to set the mediaType as initialization, we can not get it
    // through the pSample->GetMediaType(), it will be NULL if the media
    // type did not change.
    void  SetGrabMediaType( AM_MEDIA_TYPE mt ) { m_mediaType = mt; }
```

```
protected:
    ////////////////////////////////////////////////////////////////////////
    // Members for adding the ISampleGrabber to the filter graph
    ////////////////////////////////////////////////////////////////////////

    /// Overide this function to add extra filters to process media data
    virtual HRESULT addExtraFilters(IPin * pIn, IPin ** ppOut);

    // The media type we are processing
    AM_MEDIA_TYPE       m_mediaType;
    TCHAR **            m_strDevStrNames;   // Cap device friendly name
    TCHAR **            m_strDevDispNames;  // Cap devive name
    CSnapShot           m_snapShot;         // SnapShot class.
    BOOL                m_bIsProcessing;    // If it is processing video.
    CTF2005_AnaliseDesgProcessor*   m_pProcessor;   // Video processing member.

// Operations
public:
    /// Init capturing video.
    void StartPreview(int i, bool fSetup = false);
    /// Set if video processing is active or not.
    void IsProcessing(BOOL b);
    /// Set if snapshot processing is active or if it is not.
    void ActivateSnapshotProcessing(BOOL b);
    /// Captures to a file.
    void CaptureToFileDialog();
    /// Gets the interface for snapshot
    CSnapShot* getSnapShotDlg();
};
```

```
// TF2005_AnaliseDesgCtl.h
//
// Description:
//          Declaration of the CTF2005_AnaliseDesgCtrl ActiveX Control class.
//
// Author:  // TODO: Eduardo Finamore.
//
// Version: // TODO: Version number.
// Date:    // TODO: Version release date.
// Version Note:
//          // TODO: Notes about the version.
//
// Note:    // TODO: Write general notes here
/////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////
// Compiler macros

#if !defined(AFX_TF2005_ANALISEDESGCTL_H__9A299EF0_4B4D_4A59_8FC4_011D2AE13A9A__INCLUDED_)
#define AFX_TF2005_ANALISEDESGCTL_H__9A299EF0_4B4D_4A59_8FC4_011D2AE13A9A__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <list>
#include "Control.h"


/////////////////////////////////////////////////////////////////////////////
// Debug code

#ifdef _DEBUG
    #define     DEBUG_CONTROL_MESSAGE(x,y)  MessageBox(_T(x),_T(y),0)
#else
    #define     DEBUG_CONTROL_MESSAGE(x,y)
#endif

/////////////////////////////////////////////////////////////////////////////
// Preview declaration.
class CTF2005_AnaliseDesgAcquisition;

/////////////////////////////////////////////////////////////////////////////
// CTF2005_AnaliseDesgCtrl : See TF2005_AnaliseDesgCtl.cpp for implementation.
//
//      Implements the following operations:
//      - Open          : Video (InitVideoFile())
//      - Start         : Capture (InitCapture())
//      - Play()        : Start video or capturing.
//      - Pause()       : Pause video or capturing.
//      - Stop()        : Stop and restart video.
//      - SnapShot()    : Takes current frame snapshot.
//      Has the following attributes:
//      - Capture device number ( CaptureDevice )
//      - Path of the file to be oppened ( FileName )
//      - If its to show capture device setup dialogs ( ShowCaptureDialog )
/////////////////////////////////////////////////////////////////////////////

class CTF2005_AnaliseDesgCtrl : public COleControl
{
    DECLARE_DYNCREATE(CTF2005_AnaliseDesgCtrl)

// Constructor
public:
    CTF2005_AnaliseDesgCtrl();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CTF2005_AnaliseDesgCtrl)
    public:
    virtual void OnDraw(CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid);
    virtual void DoPropExchange(CPropExchange* pPX);
    virtual void OnResetState();
    //}}AFX_VIRTUAL
```

```
    // Implementation
    protected:
        ~CTF2005_AnaliseDesgCtrl();

        DECLARE_OLECREATE_EX(CTF2005_AnaliseDesgCtrl)    // Class factory and guid
        DECLARE_OLETYPELIB(CTF2005_AnaliseDesgCtrl)      // GetTypeInfo
        DECLARE_PROPPAGEIDS(CTF2005_AnaliseDesgCtrl)     // Property page IDs
        DECLARE_OLECTLTYPE(CTF2005_AnaliseDesgCtrl)     // Type name and misc status

    // Message maps
        //{{AFX_MSG(CTF2005_AnaliseDesgCtrl)
        afx_msg void OnSize(UINT nType, int cx, int cy);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()

    // Dispatch maps
        //{{AFX_DISPATCH(CTF2005_AnaliseDesgCtrl)
        CString m_fileName;
        afx_msg void OnFileNameChanged();
        short m_captureDevice;
        afx_msg void OnCaptureDeviceChanged();
        BOOL m_showCaptureDialog;
        afx_msg void OnShowCaptureDialogChanged();
        afx_msg float GetFrameRate();
        afx_msg void SetFrameRate(float newValue);
        afx_msg long GetFrameWidth();
        afx_msg void SetFrameWidth(long nNewValue);
        afx_msg long GetThreshold1();
        afx_msg void SetThreshold1(long nNewValue);
        afx_msg short GetAperture();
        afx_msg void SetAperture(short nNewValue);
        afx_msg long GetThreshold2();
        afx_msg void SetThreshold2(long nNewValue);
        afx_msg BOOL GetIsThreshold();
        afx_msg void SetIsThreshold(BOOL bNewValue);
        afx_msg long GetMinContour();
        afx_msg void SetMinContour(long nNewValue);
        afx_msg double GetPixelPerMilimeter();
        afx_msg void SetPixelPerMilimeter(double newValue);
        afx_msg BOOL InitVideoFile();
        afx_msg BOOL InitCapture();
        afx_msg void Play();
        afx_msg void Pause();
        afx_msg void Stop();
        afx_msg void SnapShot(BOOL show);
        afx_msg void ShowSnapShotDlg(BOOL show);
        afx_msg void Save();
        afx_msg void SaveSnapshot();
        afx_msg void StartProcessing();
        afx_msg void StopProcessing();
        afx_msg void StartProcessingSnapshot();
        afx_msg void StopProcessingSnapshot();
        afx_msg void openImage();
        afx_msg void next();
        afx_msg void previous();
        afx_msg void setROI(long xLeft, long xRight, long yBack, long yTop);
        afx_msg void setThresholdParameters(long thresh1, long thresh2, short Aperture);
        afx_msg void setCannyParameters(long thresh1_canny, long thresh2_canny, short aperture
_canny);
        afx_msg void saveContourFile(LPCTSTR fileName);
        afx_msg void setMinMaxContourSize(double minContourSize, double maxContourSize);
        afx_msg double getMMPerPixel();
        afx_msg void writeFile(LPCTSTR name);
        afx_msg void setMMperPixel(long p1, long p2, double realWorld);
        afx_msg void calibrate(long p1, long p2, long realWorld, short x1_y2);
        //}}AFX_DISPATCH
        DECLARE_DISPATCH_MAP()

        afx_msg void AboutBox();

    // Event maps
        //{{AFX_EVENT(CTF2005_AnaliseDesgCtrl)
        //}}AFX_EVENT                          2
```

```
        DECLARE_EVENT_MAP()

    // Dispatch and event IDs
    public:
        enum {
        //{{AFX_DISP_ID(CTF2005_AnaliseDesgCtrl)
        dispidFileName = 1L,
        dispidCaptureDevice = 2L,
        dispidShowCaptureDialog = 3L,
        dispidFrameRate = 4L,
        dispidFrameWidth = 5L,
        dispidThreshold1 = 6L,
        dispidAperture = 7L,
        dispidThreshold2 = 8L,
        dispidIsThreshold = 9L,
        dispidMinContour = 10L,
        dispidPixelPerMilimeter = 11L,
        dispidInitVideoFile = 12L,
        dispidInitCapture = 13L,
        dispidPlay = 14L,
        dispidPause = 15L,
        dispidStop = 16L,
        dispidSnapShot = 17L,
        dispidShowSnapShotDlg = 18L,
        dispidSave = 19L,
        dispidSaveSnapshot = 20L,
        dispidStartProcessing = 21L,
        dispidStopProcessing = 22L,
        dispidStartProcessingSnapshot = 23L,
        dispidStopProcessingSnapshot = 24L,
        dispidOpenImage = 25L,
        dispidNext = 26L,
        dispidPrevious = 27L,
        dispidSetROI = 28L,
        dispidSetThresholdParameters = 29L,
        dispidSetCannyParameters = 30L,
        dispidSaveContourFile = 31L,
        dispidSetMinMaxContourSize = 32L,
        dispidGetMMPerPixel = 33L,
        dispidWriteFile = 34L,
        dispidSetMMperPixel = 35L,
        dispidCalibrate = 36L,
        //}}AFX_DISP_ID
        };

    private:
        // Control states
        enum PlayerState {      // Possible states of the player.
            INITIAL_STATE,      // Video stoped.
            PLAYING_VIDEO_FILE, // Player is playing a video.
            CAPTURING           // Player is capturing from device.
        };

        std::list<CString>      m_sFileNames;
        std::list<CString>::iterator m_it;
        // Sets the render window.
        void SetRenderWindow();

        //Video manipulation class.
        CTF2005_AnaliseDesgAcquisition* m_pVideoPlayer;
        PlayerState                 m_eCurrentState;

        CControl* m_control;
    };

    //{{AFX_INSERT_LOCATION}}
    // Microsoft Visual C++ will insert additional declarations immediately before the previou
s line.

    #endif // !defined(AFX_TF2005_ANALISEDESGCTL_H__9A299EF0_4B4D_4A59_8FC4_011D2AE13A9A__INCL
UDED)
```

```
// TF2005_AnaliseDesgPpg.h
//
// Description:
//          Declaration of the CTF2005_AnaliseDesgPropPage property page class.
//
// Author:  // TODO: Eduardo Finamore.
//
// Version: // TODO: Version number.
// Date:    // TODO: Version release date.
// Version Note:
//          // TODO: Notes about the version.
//
// Note:    // TODO: Write general notes here
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// Compiler macros

#if !defined(AFX_TF2005_ANALISEDESGPPG_H__81E16C5B_736A_4691_9BC8_4A34FF5D18EC__INCLUDED_)
#define AFX_TF2005_ANALISEDESGPPG_H__81E16C5B_736A_4691_9BC8_4A34FF5D18EC__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

////////////////////////////////////////////////////////////////////////////
// CTF2005_AnaliseDesgPropPage : See TF2005_AnaliseDesgPpg.cpp.cpp for implementation.

class CTF2005_AnaliseDesgPropPage : public COlePropertyPage
{
    DECLARE_DYNCREATE(CTF2005_AnaliseDesgPropPage)
    DECLARE_OLECREATE_EX(CTF2005_AnaliseDesgPropPage)

public:
    ////////////////////////////////////////////////////////////////////////
    // Constructor
    ////////////////////////////////////////////////////////////////////////
    CTF2005_AnaliseDesgPropPage();

    ////////////////////////////////////////////////////////////////////////
    // Dialog Data
    ////////////////////////////////////////////////////////////////////////
    //{{AFX_DATA(CTF2005_AnaliseDesgPropPage)
    enum { IDD = IDD_PROPPAGE_VIDEOCONTROL };
        // NOTE - ClassWizard will add data members here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_DATA

// Implementation
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

// Message maps
protected:
    //{{AFX_MSG(CTF2005_AnaliseDesgPropPage)
        // NOTE - ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previou
s line.

#endif // !defined(AFX_TF2005_ANALISEDESGPPG_H__81E16C5B_736A_4691_9BC8_4A34FF5D18EC__INCL
UDED)
```

```
// VideoGraphBase.h
//
// Description:
//          CVideoGraphBase class interface declaration.
//
// Author:  // TODO: Author name.
//
// Version: // TODO: Version number.
// Date:    // TODO: Version release date.
// Version Note:
//          // TODO: Notes about the version.
//
// Note:    This program was based on DirectX Doc/View template.
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// Compiler tags

#pragma once

////////////////////////////////////////////////////////////////////////////
// Headers

#include <dshow.h>   // DirectShow
#include <atlbase.h> // ATL classes
#include <qedit.h>

////////////////////////////////////////////////////////////////////////////
// Compiler macros

// File filter for OpenFile dialog. It includes all the media file
// extension DirectX can process.
#define MEDIA_FILE_FILTER_TEXT \
    TEXT("Images (*.jpg;*.gif;*.bmp)|*.jpg;*.gif;*.bmp|")\
    TEXT("Video Files (*.avi;*.asf;*.mpg;*.mpeg)|*.avi;*.asf;*.mpg;*.mpeg|")\
    TEXT("All Files (*.*)|*.*||")

// Safe release the resouces.
#define SafeRelease(x) { if(x) { (x)->Release(); (x)= NULL; } }

#ifdef _DEBUG
    // Check for error, return false if failed. (Jump-If-Failed)
    #define JIF(x) if (FAILED(hr=(x))) { \
        errorHandler(TEXT("FAILED(hr=0x%x) in ") TEXT(#x) TEXT("\n"), hr);\
        return hr; \
    }
#else
    // Check for error, return false if failed. (Jump-If-Failed)
    #define JIF(x) if (FAILED(hr=(x))) { \
        return hr; \
    }
#endif

////////////////////////////////////////////////////////////////////////////
//
//  CVideoGraphBase encloses the methods to build a simple DirectX
//  Graph to redener and seek in a video file or preview live video.
//
//  To use it: ( You need to install DirectX 8.0 or upper SDK )
//  1. Denfine an object:  CVideoGraphBase videoGraphObj
//  2. Set the window (have been created) to display the video
//        videoGraphObj.SetRenderWindow(hwnd,pRC);
//  3. Playback a video file:
//        videoGraphObj.RenderFile(...);
//  4. Capture from cameras
//     a. videoGraphObj.GetNumCapDev() to know # cap devices
//     b. videoGraphObj.PreviewLive() to preview live video from camera
//        -- specified by an index or the device name.
//     c. videoGraphObj.CaptureToFile() to capture video to file.
//  5. Stop(), Pause(), Play() the video file.
//  6. Seek: Search through the video file (for media file only).
//
//  Note: the Stop() or ReleaseGraph() should be called before
//  the video window is destroyed.
```

```
//  To derive your own graph, re-implement the following functions:
//      createFilterGraph()  to build a special graph;
//      ReleaseFilterGraph() to release the graph after usage;
//////////////////////////////////////////////////////////////////////
class CVideoGraphBase
{
public:
    // Video Source ( Video file, capture device 0, 1, ... )
    #define MAX_CAP_DEVICES 10  // maximum number of capture devices
    enum VideoSource {
        VideoFilePlayback = -1,  // video file
        VideoCapDevice0   = 0,   // cap device 0
        VideoCapDevice1, // 1    // cap device 1
        VideoCapDevice2, // 2    // ...
        VideoCapDevice3, // 3
        VideoCapDevice4, // 4
        VideoCapDevice5, // 5
        VideoCapDevice6, // 6
        VideoCapDevice7, // 7
        VideoCapDevice8, // 8
        VideoCapDevice9, // 9
        VideoCapDevice10 // 10
    };

public: // Operations

    //////////////////////////////////////////////////////////////////
    // Constructor and Deconstructor
    //////////////////////////////////////////////////////////////////
    CVideoGraphBase();
    virtual ~CVideoGraphBase();
    // Set the window to render the video
    virtual HRESULT SetRenderWindow(HWND hWnd, RECT * pImgRect = NULL);
    virtual void    ReleaseFilterGraph();    // Release the whole Graph

    //////////////////////////////////////////////////////////////////
    // Functions to Capture, Playback, Setup Cameras ...
    //////////////////////////////////////////////////////////////////
    virtual HRESULT RenderFile(const TCHAR * vname);// Playback a file
    virtual int     GetNumCapDev();                 // # capture devices
    // Get specified device name
    virtual bool    GetCapDevName(int i, TCHAR *pDevStrName=NULL,
                                         TCHAR *pDevDispName=NULL);
    // Specify capture device by index or device name
    virtual HRESULT PreviewLive(int capInd = 0, bool fSetup = false);
    virtual HRESULT PreviewLive(TCHAR * capName, bool fSetup = false);
    // Capture video to file.
    virtual HRESULT CaptureToFile(const TCHAR *vname);
    virtual HRESULT StartCapture();           // begin capture to file.
    virtual HRESULT StopCapture();            // stop  capture to file.
    virtual HRESULT AdjustCamera();           // Setup current camera.
    // Set and get capture properties
    void SetIsScaling(BOOL scale);
    float GetFrameRate();
    int GetFrameWidth();
    void SetFrameRate(float rate);
    void SetFrameWidth(int w);

    //////////////////////////////////////////////////////////////////
    // Media Constrol Functions: Play, Stop, Seek
    //////////////////////////////////////////////////////////////////
    void        Play();             // Play the video
    void        Pause();            // Pause the video
    void        Stop();             // Stop the video
    LONGLONG    Seek(int offset);   // seek w.r.t current position
    LONGLONG    SeekToStart();      // Goto the begin of the video
    LONGLONG    SeekToEnd();        // Goto the end of the video

    //////////////////////////////////////////////////////////////////
    // Get the attribute of the filter graph
    //////////////////////////////////////////////////////////////////
    VideoSource GetVideoSource(TCHAR * vn = NULL); // Get source name
    SIZE        GetVideoSize();         // Get the vidoe size
```

```
    double      GetAvgFrameRate();      // Get the rendering frame rate
    LONGLONG    GetFrameNumber();       // Get current frame number
    bool        IsGraphRunning();       // Get current graph status

protected: // Operations
    ////////////////////////////////////////////////////////////////
    // Build the graph needed to render the video and control it
    ////////////////////////////////////////////////////////////////
    virtual HRESULT createFilterGraph(); // Create the filter graph
    virtual HRESULT startGraph();        // Start the playing
    virtual void    stopGraph();         // Stop playing
    virtual HRESULT setVideoWindowSize();// Set render window

    ////////////////////////////////////////////////////////////////
    // Build the essensial filters (called in createFilterGraph())
    ////////////////////////////////////////////////////////////////
    virtual HRESULT addSrcFilter(IBaseFilter **ppSrc);
    virtual HRESULT addExtraFilters(IPin * pIn, IPin ** ppOut);
    virtual HRESULT addRenderFilters(IPin *pIn);

    ////////////////////////////////////////////////////////////////
    // Helper functions:
    ////////////////////////////////////////////////////////////////
    HRESULT setupCapture(IBaseFilter * pSrcFilter); // resolution, rate
    HRESULT showPropertyPage(IUnknown* pIU, const WCHAR* name);  // COM
    HRESULT get_pin(IBaseFilter * pFilter, PIN_DIRECTION dir, int n, IPin **ppPin);
    void    errorHandler(TCHAR * szFormat, ...);   // Show Error message

    ////////////////////////////////////////////////////////////////
    // Helper functions for debug filter graph
    // register the filter graph built so that GraphEdt can view it
    ////////////////////////////////////////////////////////////////
    HRESULT addGraphToRot(IUnknown *pUnkGraph, DWORD *pdwRegister);
    void    removeGraphFromRot(DWORD dwRegister);

protected: // Attributes
    ////////////////////////////////////////////////////////////////
    // DirectShow interface pointers
    ////////////////////////////////////////////////////////////////
    IGraphBuilder *         m_pGraphBuilder; // build render graph
    IFilterGraph*           m_pFilterGraph;  // Filter Graph
    ICaptureGraphBuilder2 * m_pCaptureGraphBuilder2; // capture graph
    IMediaControl*          m_pMediaControl; // MediaControl
    IVideoWindow*           m_pVideoWindow;  // window to play video
    IMediaSeeking*          m_pMediaSeeking; // Seeking interface
    IBaseFilter *           m_pSrcFilter;    // Source filter
    IBaseFilter *           m_pRenderFilter; // Render filter

    ////////////////////////////////////////////////////////////////
    // Source filter: (render file or capture live)
    ////////////////////////////////////////////////////////////////
    VideoSource      m_nVideoSource;      // Source (file/cap devices)
    WCHAR            m_wcMediaName[1024]; // Video source name (string)
    IMoniker *       m_pCapMonikers[MAX_CAP_DEVICES];  // cap devices
    int              m_nNumOfCapDev;           // # cap devices
    bool             m_fShowCaptureProperties; // custom cap setting
    AM_MEDIA_TYPE *  m_pCaptureSetting;        // rate, resolution...

    ////////////////////////////////////////////////////////////////
    // Output: display video in m_oVideoRect in m_hRenderWnd or output
    // video to a file
    ////////////////////////////////////////////////////////////////
    HWND    m_hRenderWnd;          // The Window to display the video
    RECT    m_oVideoRect;          // The viewport to display the video
    BOOL    m_fIsScaling;          // scale video to the whole window
    WCHAR   m_wcOutputFile[1024]; // captured to this file

    ////////////////////////////////////////////////////////////////
    // Video properties
    ////////////////////////////////////////////////////////////////
    float   m_fAvgFrmRate;      // Desired average frame rate
    int     m_iSizeX;           // Frame width

    ////////////////////////////////////////////////////////////////
```

```
    // VMR device Control interface
    //////////////////////////////////////////////////////////////////////
#ifdef _VMR
    IVMRWindowlessControl *m_pWc;    //VMR9-test
    bool m_VMRSucceeded;             //VMR9-test
#endif

    //////////////////////////////////////////////////////////////////////
    // Do we want the filter graph to be viewable by GraphEdt
    //////////////////////////////////////////////////////////////////////
    bool    m_fRegisterFilterGraph;
    DWORD   m_dwGraphRegister;
};
```

```
// TF2005_AnaliseDesgCtl.cpp
//
// Description:
//          Implementation of the CTF2005_AnaliseDesgCtrl ActiveX Control class.
//
// Author:  // TODO: Eduardo Finamore
//
// Version: // TODO: Version number.
// Date:    // TODO: Version release date.
// Version Note:
//          // TODO: Notes about the version.
//
// Note:    // TODO: Write general notes here
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// Headers

#include "stdafx.h"
#include "TF2005_AnaliseDesg.h"
#include "TF2005_AnaliseDesgCtl.h"
#include "TF2005_AnaliseDesgPpg.h"
#include "TF2005_AnaliseDesgAcquisition.h"


////////////////////////////////////////////////////////////////////////////
// Compiler macros

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


IMPLEMENT_DYNCREATE(CTF2005_AnaliseDesgCtrl, COleControl)


////////////////////////////////////////////////////////////////////////////
// Message map

BEGIN_MESSAGE_MAP(CTF2005_AnaliseDesgCtrl, COleControl)
    //{{AFX_MSG_MAP(CTF2005_AnaliseDesgCtrl)
    ON_WM_SIZE()
    //}}AFX_MSG_MAP
    ON_OLEVERB(AFX_IDS_VERB_PROPERTIES, OnProperties)
END_MESSAGE_MAP()


////////////////////////////////////////////////////////////////////////////
// Dispatch map

BEGIN_DISPATCH_MAP(CTF2005_AnaliseDesgCtrl, COleControl)
    //{{AFX_DISPATCH_MAP(CTF2005_AnaliseDesgCtrl)
    DISP_PROPERTY_NOTIFY(CTF2005_AnaliseDesgCtrl, "FileName", m_fileName, OnFileNameChange
d, VT_BSTR)
    DISP_PROPERTY_NOTIFY(CTF2005_AnaliseDesgCtrl, "CaptureDevice", m_captureDevice, OnCapt
ureDeviceChanged, VT_I2)
    DISP_PROPERTY_NOTIFY(CTF2005_AnaliseDesgCtrl, "ShowCaptureDialog", m_showCaptureDialog
, OnShowCaptureDialogChanged, VT_BOOL)
    DISP_PROPERTY_EX(CTF2005_AnaliseDesgCtrl, "FrameRate", GetFrameRate, SetFrameRate, VT_
R4)
    DISP_PROPERTY_EX(CTF2005_AnaliseDesgCtrl, "FrameWidth", GetFrameWidth, SetFrameWidth,
VT_I4)
    DISP_PROPERTY_EX(CTF2005_AnaliseDesgCtrl, "threshold1", GetThreshold1, SetThreshold1,
VT_I4)
    DISP_PROPERTY_EX(CTF2005_AnaliseDesgCtrl, "aperture", GetAperture, SetAperture, VT_I2)
    DISP_PROPERTY_EX(CTF2005_AnaliseDesgCtrl, "threshold2", GetThreshold2, SetThreshold2,
VT_I4)
    DISP_PROPERTY_EX(CTF2005_AnaliseDesgCtrl, "isThreshold", GetIsThreshold, SetIsThreshol
d, VT_BOOL)
    DISP_PROPERTY_EX(CTF2005_AnaliseDesgCtrl, "minContour", GetMinContour, SetMinContour,
VT_I4)
    DISP_PROPERTY_EX(CTF2005_AnaliseDesgCtrl, "pixelPerMilimeter", GetPixelPerMilimeter, S
```

```
etPixelPerMilimeter, VT_R8)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "InitVideoFile", InitVideoFile, VT_BOOL, VTS_NO
NE)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "InitCapture", InitCapture, VT_BOOL, VTS_NONE)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "Play", Play, VT_EMPTY, VTS_NONE)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "Pause", Pause, VT_EMPTY, VTS_NONE)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "Stop", Stop, VT_EMPTY, VTS_NONE)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "SnapShot", SnapShot, VT_EMPTY, VTS_BOOL)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "ShowSnapShotDlg", ShowSnapShotDlg, VT_EMPTY, V
TS_BOOL)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "Save", Save, VT_EMPTY, VTS_NONE)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "SaveSnapshot", SaveSnapshot, VT_EMPTY, VTS_NON
E)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "StartProcessing", StartProcessing, VT_EMPTY, V
TS_NONE)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "StopProcessing", StopProcessing, VT_EMPTY, VTS
_NONE)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "StartProcessingSnapshot", StartProcessingSnaps
hot, VT_EMPTY, VTS_NONE)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "StopProcessingSnapshot", StopProcessingSnapsho
t, VT_EMPTY, VTS_NONE)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "openImage", openImage, VT_EMPTY, VTS_NONE)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "next", next, VT_EMPTY, VTS_NONE)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "previous", previous, VT_EMPTY, VTS_NONE)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "setROI", setROI, VT_EMPTY, VTS_I4 VTS_I4 VTS_I
4 VTS_I4)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "setThresholdParameters", setThresholdParameter
s, VT_EMPTY, VTS_I4 VTS_I4 VTS_I2)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "setCannyParameters", setCannyParameters, VT_EM
PTY, VTS_I4 VTS_I4 VTS_I2)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "saveContourFile", saveContourFile, VT_EMPTY, V
TS_BSTR)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "setMinMaxContourSize", setMinMaxContourSize, V
T_EMPTY, VTS_R8 VTS_R8)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "getMMPerPixel", getMMPerPixel, VT_R8, VTS_NONE
)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "writeFile", writeFile, VT_EMPTY, VTS_BSTR)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "setMMperPixel", setMMperPixel, VT_EMPTY, VTS_I
4 VTS_I4 VTS_R8)
            DISP_FUNCTION(CTF2005_AnaliseDesgCtrl, "calibrate", calibrate, VT_EMPTY, VTS_I4 VTS_I4
 VTS_I4 VTS_I2)
            //}}AFX_DISPATCH_MAP
            DISP_FUNCTION_ID(CTF2005_AnaliseDesgCtrl, "AboutBox", DISPID_ABOUTBOX, AboutBox, VT_EM
PTY, VTS_NONE)
        END_DISPATCH_MAP()


        ////////////////////////////////////////////////////////////////////////
        // Event map

        BEGIN_EVENT_MAP(CTF2005_AnaliseDesgCtrl, COleControl)
            //{{AFX_EVENT_MAP(CTF2005_AnaliseDesgCtrl)
            // NOTE - ClassWizard will add and remove event map entries
            //    DO NOT EDIT what you see in these blocks of generated code !
            //}}AFX_EVENT_MAP
        END_EVENT_MAP()


        ////////////////////////////////////////////////////////////////////////
        // Property pages

        // TODO: Add more property pages as needed.  Remember to increase the count!
        BEGIN_PROPPAGEIDS(CTF2005_AnaliseDesgCtrl, 1)
            PROPPAGEID(CTF2005_AnaliseDesgPropPage::guid)
        END_PROPPAGEIDS(CTF2005_AnaliseDesgCtrl)


        ////////////////////////////////////////////////////////////////////////
        // Initialize class factory and guid

        IMPLEMENT_OLECREATE_EX(CTF2005_AnaliseDesgCtrl, "TF2005_AnaliseDesgCtrl.1",
            0x61484206, 0x756e, 0x419f, 0xb5, 0xba, 0xfb, 0x5a, 0xe9, 0xc8, 0x6b, 0x55)
```

```
////////////////////////////////////////////////////////////////////////////
// Type library ID and version

IMPLEMENT_OLETYPELIB(CTF2005_AnaliseDesgCtrl, _tlid, _wVerMajor, _wVerMinor)


////////////////////////////////////////////////////////////////////////////
// Interface IDs

const IID BASED_CODE IID_DVideoControl =
        { 0x8ddca8b0, 0x6122, 0x42cb, { 0xac, 0x8, 0xc9, 0x86, 0x8d, 0x86, 0xe9, 0x1f } };
const IID BASED_CODE IID_DVideoControlEvents =
        { 0x941cf825, 0xee89, 0x4b3f, { 0xba, 0x4d, 0x79, 0x6b, 0x74, 0x48, 0x7c, 0x2a } }
;


////////////////////////////////////////////////////////////////////////////
// Control type information

static const DWORD BASED_CODE _dwVideoControlOleMisc =
    OLEMISC_ACTIVATEWHENVISIBLE |
    OLEMISC_SETCLIENTSITEFIRST |
    OLEMISC_INSIDEOUT |
    OLEMISC_CANTLINKINSIDE |
    OLEMISC_RECOMPOSEONRESIZE;

IMPLEMENT_OLECTLTYPE(CTF2005_AnaliseDesgCtrl, IDS_VIDEOCONTROL, _dwVideoControlOleMisc)

////////////////////////////////////////////////////////////////////////////
// CTF2005_AnaliseDesgCtrl::CTF2005_AnaliseDesgCtrlFactory::UpdateRegistry -
// Adds or removes system registry entries for CTF2005_AnaliseDesgCtrl

BOOL CTF2005_AnaliseDesgCtrl::CTF2005_AnaliseDesgCtrlFactory::UpdateRegistry(BOOL bRegiste
r)
{
    // TODO: Verify that your control follows apartment-model threading rules.
    // Refer to MFC TechNote 64 for more information.
    // If your control does not conform to the apartment-model rules, then
    // you must modify the code below, changing the 6th parameter from
    // afxRegApartmentThreading to 0.

    if (bRegister)
        return AfxOleRegisterControlClass(
            AfxGetInstanceHandle(),
            m_clsid,
            m_lpszProgID,
            IDS_VIDEOCONTROL,
            IDB_VIDEOCONTROL,
            afxRegApartmentThreading,
            _dwVideoControlOleMisc,
            _tlid,
            _wVerMajor,
            _wVerMinor);
    else
        return AfxOleUnregisterClass(m_clsid, m_lpszProgID);
}

////////////////////////////////////////////////////////////////////////////
// CTF2005_AnaliseDesgCtrl: Constructor
////////////////////////////////////////////////////////////////////////////
CTF2005_AnaliseDesgCtrl::CTF2005_AnaliseDesgCtrl()
:m_pVideoPlayer(new CTF2005_AnaliseDesgAcquisition()), m_eCurrentState(INITIAL_STATE)
{
    InitializeIIDs(&IID_DVideoControl, &IID_DVideoControlEvents);

    m_fileName.Empty();
    m_control=CControl::instance();
}

////////////////////////////////////////////////////////////////////////////
// ~CTF2005_AnaliseDesgCtrl: Destructor
////////////////////////////////////////////////////////////////////////////
CTF2005_AnaliseDesgCtrl::~CTF2005_AnaliseDesgCtrl()
{
```

```
    delete m_pVideoPlayer;
}

////////////////////////////////////////////////////////////////////////////
// InitVideoFile: Open a video file. If there is a file name defined in
// FileName attribute, it opens automatically, else it will open a dialog box.
////////////////////////////////////////////////////////////////////////////
BOOL CTF2005_AnaliseDesgCtrl::InitVideoFile()
{

    m_pVideoPlayer->SetIsScaling(true); // Scale the video.
    this->SetRenderWindow();            // Set this control as render window.

    if(m_fileName.IsEmpty()){
        // Choose the video file by fileopen dialog (MEDIA_FILE_FILTER_TEXT)
        // specifies the file types we support
        CFileDialog open_dlg( true, 0, 0, OFN_FILEMUSTEXIST,
            MEDIA_FILE_FILTER_TEXT, NULL );
        if( open_dlg.DoModal() == IDOK ){
            // Get the video file name and open it.
            m_fileName = open_dlg.GetPathName();
            // Open the video file
            if(!SUCCEEDED(m_pVideoPlayer->RenderFile(m_fileName.GetBuffer(0)))){
                DEBUG_CONTROL_MESSAGE(_T("Failed to open video"),_T("InitVideo()"));
                return FALSE;
            }
            DEBUG_CONTROL_MESSAGE(_T("Video opened successfully"),_T("InitVideo()"));
        }
    } else {
        if(!SUCCEEDED(m_pVideoPlayer->RenderFile(m_fileName.GetBuffer(0)))){
            DEBUG_CONTROL_MESSAGE(_T("Failed to open video"),_T("InitVideo()"));
            m_fileName.Empty();
            return FALSE;
        }
    }

    // Ajustar o estado atual do componente.
    m_eCurrentState = PLAYING_VIDEO_FILE;

    return TRUE;
}

////////////////////////////////////////////////////////////////////////////
// InitCapture: Starts capturing from capture devices.
////////////////////////////////////////////////////////////////////////////
BOOL CTF2005_AnaliseDesgCtrl::InitCapture()
{
    m_pVideoPlayer->SetIsScaling(true);     // Scale the video.
    this->SetRenderWindow();                // Set this control as render window.
    // If camera wasn't set up yet...
    if(m_eCurrentState != CAPTURING && m_showCaptureDialog){
        if(m_captureDevice < 0){
            // Gets the default camera device registered in WIN.INI file.
            // If no value is found, returns 0.
            m_captureDevice = AfxGetApp()->GetProfileInt(_T("VideoPlayer"),
                _T("Default Capture Device"), 0);
        }
        // Shows the camera dialog.
        m_pVideoPlayer->StartPreview(m_captureDevice, true);
    } else {
        // If we have not adjusted it yet...
        if(m_captureDevice < 0){
            // Gets the default camera device registered in WIN.INI file.
            // If no value is found, returns 0.
            m_captureDevice = AfxGetApp()->GetProfileInt(_T("VideoPlayer"),
                _T("Default Capture Device"), 0);
        }
        m_pVideoPlayer->StartPreview(m_captureDevice, false);
    }

    // If it was successful
    if(m_pVideoPlayer->IsGraphRunning()){
        // Now, camera was successfully set up.
        m_eCurrentState = CAPTURING;
```

```
    } else {
        DEBUG_CONTROL_MESSAGE("Failed to open the device","Warning!");
        m_captureDevice = AfxGetApp()->GetProfileInt(_T("VideoPlayer"),
                _T("Default Capture Device"), 0);
        return FALSE;
    }

    return TRUE;
}

////////////////////////////////////////////////////////////////////////////
// Play: Play the video or starts capturing, depending on the mode initiated.
////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgCtrl::Play()
{
    // Starts the video only if there is a source.
    if(m_eCurrentState != INITIAL_STATE){
        m_pVideoPlayer->Play();
    }
}

////////////////////////////////////////////////////////////////////////////
// Pause: Pause the video or capturing
////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgCtrl::Pause()
{
    m_pVideoPlayer->Pause();
}

////////////////////////////////////////////////////////////////////////////
// Stop: Stops and restarts video.
////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgCtrl::Stop()
{
    // If it's playing a video, restart the video first.
    if(m_eCurrentState == PLAYING_VIDEO_FILE){
        m_pVideoPlayer->SeekToStart();
    }
    m_pVideoPlayer->Stop();
}

////////////////////////////////////////////////////////////////////////////
// GetFrameRate: Gets the desired frame rate.
// Return value: Desired frame rate.
////////////////////////////////////////////////////////////////////////////
float CTF2005_AnaliseDesgCtrl::GetFrameRate()
{
//  return m_pVideoPlayer->GetFrameRate();
    return (float) m_pVideoPlayer->GetAvgFrameRate();
}

////////////////////////////////////////////////////////////////////////////
// SetFrameRate: Sets desired frame rate.
// Arguments:
//      newValue    : desired frame rate.
////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgCtrl::SetFrameRate(float newValue)
{
    m_pVideoPlayer->SetFrameRate(newValue);

    SetModifiedFlag();
}

////////////////////////////////////////////////////////////////////////////
// GetFrameWidth: Gets the processing frame width
// Return value: Frame width
////////////////////////////////////////////////////////////////////////////
long CTF2005_AnaliseDesgCtrl::GetFrameWidth()
{
    return m_pVideoPlayer->GetFrameWidth();
}

////////////////////////////////////////////////////////////////////////////
// SetFrameWidth: Sets the processing frame width
```

```
// Arguments:
//      nNewValue   : Width
////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgCtrl::SetFrameWidth(long nNewValue)
{
    m_pVideoPlayer->SetFrameWidth(nNewValue);

    SetModifiedFlag();
}

////////////////////////////////////////////////////////////////////////////
// Save: Capture video to a file. If FileName isn't set open a dialog box.
// Opens automatically a dialog asking to start and stop capture.
////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgCtrl::Save()
{
    if(m_fileName.IsEmpty()){
        m_pVideoPlayer->CaptureToFileDialog();
    } else {
        if(SUCCEEDED(m_pVideoPlayer->CaptureToFile(m_fileName))){
            m_pVideoPlayer->StopCapture();
            MessageBox(_T("Start Capturing?"),_T(""),0);
            m_pVideoPlayer->StartCapture();
            MessageBox(_T("Stop Capturing?"),_T(""),0);
            m_pVideoPlayer->StopCapture();
        }
        m_fileName.Empty();
    }
}

////////////////////////////////////////////////////////////////////////////
// SnapShot: Gets a snap shot.
// Arguments:
//      show    : If false, snap shot immediately
//                Else, opens a snapshot dialog
////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgCtrl::SnapShot(BOOL show)
{
    m_pVideoPlayer->getSnapShotDlg()->ShowSnapShot(show);
    m_pVideoPlayer->getSnapShotDlg()->CaptureFrame();
}

////////////////////////////////////////////////////////////////////////////
// ShowSnapShot: Gets a snap shot.
// Arguments:
//      show    : If true, snap shot immediately
////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgCtrl::ShowSnapShotDlg(BOOL show)
{
    m_pVideoPlayer->getSnapShotDlg()->ShowDialog(show);
}

////////////////////////////////////////////////////////////////////////////
// SaveSnapshot: Saves snapshot image.
////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgCtrl::SaveSnapshot()
{
    if(m_fileName.IsEmpty()){
        // Choose the video file by fileopen dialog (MEDIA_FILE_FILTER_TEXT)
        // specifies the file types we support
        CFileDialog open_dlg( true, 0, 0, OFN_CREATEPROMPT,
            MEDIA_FILE_FILTER_TEXT, NULL );
        if( open_dlg.DoModal() == IDOK ){
            // Get the video file name and open it.
            m_fileName = open_dlg.GetPathName();
            // Open the video file
            if(!m_pVideoPlayer->getSnapShotDlg()->SaveToBitmap(m_fileName)){
                DEBUG_CONTROL_MESSAGE(_T("Failed to open image"),_T("SaveSnapshot()"));
                return;
            }
            DEBUG_CONTROL_MESSAGE(_T("Image opened successfully"),_T("SaveSnapshot()"));
        }
    } else {
        if(!m_pVideoPlayer->getSnapShotDlg()->SaveToBitmap(m_fileName)){
```

```
            DEBUG_CONTROL_MESSAGE(_T("Failed to open image"),_T("SaveSnapshot()"));
            m_fileName.Empty();
            return;
        }
    }
}

////////////////////////////////////////////////////////////////////////////
// StartProcessing: Starts to process the video.
////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgCtrl::StartProcessing()
{
    m_pVideoPlayer->IsProcessing(TRUE);
}

////////////////////////////////////////////////////////////////////////////
// StopProcessing: Stops processing the video.
////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgCtrl::StopProcessing()
{
    m_pVideoPlayer->IsProcessing(FALSE);
}

////////////////////////////////////////////////////////////////////////////
// StartProcessingSnapshot: Starts processing snapshots.
////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgCtrl::StartProcessingSnapshot()
{
    m_pVideoPlayer->ActivateSnapshotProcessing(TRUE);
}

////////////////////////////////////////////////////////////////////////////
// StopProcessing: Stops processing snapshots.
////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgCtrl::StopProcessingSnapshot()
{
    m_pVideoPlayer->ActivateSnapshotProcessing(FALSE);
}

////////////////////////////////////////////////////////////////////////////
// DoPropExchange: Persistence support
////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgCtrl::DoPropExchange(CPropExchange* pPX)
{
    ExchangeVersion(pPX, MAKELONG(_wVerMinor, _wVerMajor));
    COleControl::DoPropExchange(pPX);

    PX_String(pPX, _T("FileName"), m_fileName, _T(""));
    PX_Short(pPX, _T("CaptureDevice"), m_captureDevice, -1);
    PX_Bool(pPX, _T("ShowCaptureDialog"), m_showCaptureDialog, TRUE);
}

////////////////////////////////////////////////////////////////////////////
// AboutBox: Display an "About" box to the user
////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgCtrl::AboutBox()
{
    CDialog dlgAbout(IDD_ABOUTBOX_VIDEOCONTROL);
    dlgAbout.DoModal();
}

////////////////////////////////////////////////////////////////////////////
//
////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgCtrl::SetRenderWindow()
{
    // Sets this control to be the render window
    RECT rec;
    ::GetClientRect(this->GetSafeHwnd(), &rec); // Get the client area

    // Sets the render window owner and RECT.
    m_pVideoPlayer->SetRenderWindow(this->GetSafeHwnd(), &rec);
}
```

```
/////////////////////////////////////////////////////////////////////////////
// CTF2005_AnaliseDesgCtrl message handlers

/////////////////////////////////////////////////////////////////////////////
// OnDraw: Drawing function
/////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgCtrl::OnDraw(
            CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)
{
    // TODO: Replace the following code with your own drawing code.
    pdc->FillRect(rcBounds, CBrush::FromHandle((HBRUSH)GetStockObject(WHITE_BRUSH)));
}

/////////////////////////////////////////////////////////////////////////////
// OnResetState: Reset control to default state
/////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgCtrl::OnResetState()
{
    COleControl::OnResetState();  // Resets defaults found in DoPropExchange

    // TODO: Reset any other control state here.
}

/////////////////////////////////////////////////////////////////////////////
// OnFileNameChanged: Action to be taken when the file name changes.
/////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgCtrl::OnFileNameChanged()
{
    SetModifiedFlag();

    // TODO: Put event handling code here.
}

/////////////////////////////////////////////////////////////////////////////
// OnCaptureDeviceChanged: Action to be taken when the choosen device changes.
/////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgCtrl::OnCaptureDeviceChanged()
{
    SetModifiedFlag();

    // TODO: Put event handling code here.
}

/////////////////////////////////////////////////////////////////////////////
// OnShowCaptureDialogChanged:
/////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgCtrl::OnShowCaptureDialogChanged()
{
    SetModifiedFlag();

    // TODO: Put event handling code here.
}

/////////////////////////////////////////////////////////////////////////////
// OnSize: Action to be taken when the control size is changed.
/////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgCtrl::OnSize(UINT nType, int cx, int cy)
{
    COleControl::OnSize(nType, cx, cy);
    // Reset render window.
    this->SetRenderWindow();

    // TODO: Put event handling code here.
}

void CTF2005_AnaliseDesgCtrl::openImage()
{
    m_sFileNames.clear();

    OPENFILENAME ofn;
    char buf[100000]; buf[0] = NULL;
    char buf2[100000];buf[0]=NULL;
```

```
            memset(&ofn,0,sizeof(ofn));
            ofn.lpstrTitle="Abrir Arquivo(s)";
            ofn.lStructSize = sizeof(ofn);//Specifies the length, in bytes, of the structure.
            ofn.hwndOwner = NULL;          //Handle to the window that owns the dialog box
            ofn.lpstrFile = buf;           //Pointer to a buffer that contains a file name used to
initialize the File Name edit control
            ofn.lpstrFileTitle=buf2;
            ofn.nMaxFileTitle=sizeof(buf2);
        //  ofn.lpstrInitialDir=dlg->m_CSTRING_FolderPath;
            ofn.nMaxFile = sizeof(buf);    //Specifies the size, in TCHARs, of the buffer pointed t
o by lpstrFile
            ofn.lpstrFilter = "Arquivos de Imagem(*.bmp;*.jpg;*.avi)\0*.bmp;*.jpg;*.avi\0\0";
  //Pointer to a buffer that receives the file name and extension (without path information) of th
e selected file.
            ofn.Flags = OFN_HIDEREADONLY|OFN_ALLOWMULTISELECT|OFN_NOCHANGEDIR|OFN_EXPLORER;
            ofn.lpstrDefExt="tst";
            ofn.lpstrTitle="Abrir";

            bool apenasUmArquivoSelecionado=false;

            ////////////////////////////////////
            //Pegando cada arquivo
            ////////////////////////////////////


            if(GetOpenFileName(&ofn))
            {

                char* path =ofn.lpstrFile ;
                int size = strlen(ofn.lpstrFile);
                CString dir(path);
                dir += "\\";
                char* lastfile = ofn.lpstrFile + size + 1;
                char* firstfile;
                int lastFileSize = strlen(lastfile);


                if(lastFileSize)
                    apenasUmArquivoSelecionado=false;
                else
                    apenasUmArquivoSelecionado=true;

                bool exit=false;

                if(apenasUmArquivoSelecionado==false)
                while(!exit)
                {
                    char* file = ofn.lpstrFile + size + 1; // skip the null terminator
                    CString fileName(file);
                    int fileSize = strlen(file);

                    if(!fileSize)
                    {
                        exit = true;

                        m_sFileNames.push_back(dir+lastfile);
                    }
                    else
                        {
                            size += (fileSize + 1);
                            fileName=dir+fileName;
                            m_sFileNames.push_back(fileName);

                        }

                }
                else m_sFileNames.push_back(path);

                m_sFileNames.sort();

            }

            m_it=m_sFileNames.begin();
```

```
        m_pVideoPlayer->SetIsScaling(true); // Scale the video.
        this->SetRenderWindow();            // Set this control as render window.

        if(*m_it){

            if(!SUCCEEDED(m_pVideoPlayer->RenderFile(LPCTSTR(*m_it)))){
                DEBUG_CONTROL_MESSAGE(_T("Failed to open video"),_T("InitVideo()"));
            }
        }

        // Ajustar o estado atual do componente.
        m_eCurrentState = PLAYING_VIDEO_FILE;
        return ;
    }

    void CTF2005_AnaliseDesgCtrl::next()
    {

            this->m_it++;
            this->m_it++;
            if(m_it!=m_sFileNames.end())
            {

                if(*m_it){

                    if(!SUCCEEDED(m_pVideoPlayer->RenderFile(LPCTSTR(*m_it)))){
                    DEBUG_CONTROL_MESSAGE(_T("Failed to open video"),_T("InitVideo()"));

                }

                }
            }

        return;

    }

    void CTF2005_AnaliseDesgCtrl::previous()
    {
        this->m_it--;
            if(m_it!=m_sFileNames.end())
            {
                    if(*m_it){

                    if(!SUCCEEDED(m_pVideoPlayer->RenderFile(LPCTSTR(*m_it)))){
                        DEBUG_CONTROL_MESSAGE(_T("Failed to open video"),_T("InitVideo()"));

                    }
                }
            }

    }

    void CTF2005_AnaliseDesgCtrl::setROI(long xLeft, long xRight, long yBack, long yTop)
    {
        // TODO: Add your dispatch handler code here
        m_control->setROI(xLeft,xRight,yBack,yTop);

    }

    void CTF2005_AnaliseDesgCtrl::setThresholdParameters(long thresh1, long thresh2, short Ape
rture)
    {
        m_control->m_thresh1=thresh1;
        m_control->m_thresh2=thresh2;
        m_control->m_aperture=Aperture;

    }

    long CTF2005_AnaliseDesgCtrl::GetThreshold1()
    {
        // TODO: Add your property handler here

        return  m_control->m_thresh1_canny;
```

```
}

void CTF2005_AnaliseDesgCtrl::SetThreshold1(long nNewValue)
{
    // TODO: Add your property handler here

    SetModifiedFlag();
}

short CTF2005_AnaliseDesgCtrl::GetAperture()
{
    // TODO: Add your property handler here

    return  m_control->m_aperture_canny;
}

void CTF2005_AnaliseDesgCtrl::SetAperture(short nNewValue)
{
    // TODO: Add your property handler here

    SetModifiedFlag();
}

long CTF2005_AnaliseDesgCtrl::GetThreshold2()
{
    // TODO: Add your property handler here

    return m_control->m_thresh2_canny;
}

void CTF2005_AnaliseDesgCtrl::SetThreshold2(long nNewValue)
{
    // TODO: Add your property handler here

    SetModifiedFlag();
}

void CTF2005_AnaliseDesgCtrl::setCannyParameters(long thresh1_canny, long thresh2_canny, s
hort aperture_canny)
{
    m_control->m_thresh1_canny=thresh1_canny;
    m_control->m_thresh2_canny=thresh2_canny;
    m_control->m_aperture_canny=aperture_canny;

}

BOOL CTF2005_AnaliseDesgCtrl::GetIsThreshold()
{
    // TODO: Add your property handler here

    return  m_control->m_isThreshold;
}

void CTF2005_AnaliseDesgCtrl::SetIsThreshold(BOOL bNewValue)
{
    // TODO: Add your property handler here
    m_control->m_isThreshold=bNewValue;
    SetModifiedFlag();
}

void CTF2005_AnaliseDesgCtrl::saveContourFile(LPCTSTR fileName)
{
    // TODO: Add your dispatch handler code here
        m_control->m_fileName=fileName;

}

void CTF2005_AnaliseDesgCtrl::setMinMaxContourSize(double minContourSize, double maxContou
rSize)
{
    if(minContourSize&&maxContourSize)
    {
    m_control->minimal_contour_size=minContourSize;
    m_control->maximal_contour_size=int(11xContourSize);
```

```
    }

}

double CTF2005_AnaliseDesgCtrl::getMMPerPixel()
{
    // TODO: Add your dispatch handler code here

    return 0.0;
}

long CTF2005_AnaliseDesgCtrl::GetMinContour()
{
    // TODO: Add your property handler here

    return 0;
}

void CTF2005_AnaliseDesgCtrl::SetMinContour(long nNewValue)
{
    m_control->minimal_contour_size=nNewValue;

    SetModifiedFlag();
}

void CTF2005_AnaliseDesgCtrl::writeFile(LPCTSTR name)
{
    m_control->m_strFileName2Save=name;
    m_control->m_bWriteFile=true;

}

void CTF2005_AnaliseDesgCtrl::setMMperPixel(long p1, long p2, double realWorld)
{
//  fabs(p1-p2)

}

void CTF2005_AnaliseDesgCtrl::calibrate(long p1, long p2, long realWorld, short x1_y2)
{
    if(x1_y2==1)
    m_control->m_comprimentoMedidoCalibracao=double(realWorld)/double(double(double(abs(p1
-p2))/double(1000))*double(m_control->m_atualImageWidth));
    else
    m_control->m_comprimentoMedidoCalibracao=double(realWorld)/double(double(double(abs(p1
-p2))/double(1000))*double(m_control->m_atualImageHeight));
}

double CTF2005_AnaliseDesgCtrl::GetPixelPerMilimeter()
{
    // TODO: Add your property handler here

    return m_control->m_comprimentoMedidoCalibracao;
}

void CTF2005_AnaliseDesgCtrl::SetPixelPerMilimeter(double newValue)
{
    // TODO: Add your property handler here

    SetModifiedFlag();
}
```

```
// TF2005_AnaliseDesgPpg.cpp
//
// Description:
//          Implementation of the CTF2005_AnaliseDesgPropPage property page class.
//
// Author:  // TODO: Eduardo Finamore
//
// Version: // TODO: Version number.
// Date:    // TODO: Version release date.
// Version Note:
//          // TODO: Notes about the version.
//
// Note:    // TODO: Write general notes here
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// Headers

#include "stdafx.h"
#include "TF2005_AnaliseDesg.h"
#include "TF2005_AnaliseDesgPpg.h"

////////////////////////////////////////////////////////////////////////////
// Compiler macros

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


IMPLEMENT_DYNCREATE(CTF2005_AnaliseDesgPropPage, COlePropertyPage)


////////////////////////////////////////////////////////////////////////////
// Message map

BEGIN_MESSAGE_MAP(CTF2005_AnaliseDesgPropPage, COlePropertyPage)
    //{{AFX_MSG_MAP(CTF2005_AnaliseDesgPropPage)
    // NOTE - ClassWizard will add and remove message map entries
    //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()


////////////////////////////////////////////////////////////////////////////
// Initialize class factory and guid

IMPLEMENT_OLECREATE_EX(CTF2005_AnaliseDesgPropPage, "TF2005_AnaliseDesgPropPage.1",
    0xb517a9df, 0x842a, 0x4007, 0x91, 0x58, 0x86, 0xa0, 0x2c, 0x40, 0x7f, 0x40)


////////////////////////////////////////////////////////////////////////////
// CTF2005_AnaliseDesgPropPage::CTF2005_AnaliseDesgPropPageFactory::UpdateRegistry -
// Adds or removes system registry entries for CTF2005_AnaliseDesgPropPage

BOOL CTF2005_AnaliseDesgPropPage::CTF2005_AnaliseDesgPropPageFactory::UpdateRegistry(BOOL
bRegister)
{
    if (bRegister)
        return AfxOleRegisterPropertyPageClass(AfxGetInstanceHandle(),
            m_clsid, IDS_VIDEOCONTROL_PPG);
    else
        return AfxOleUnregisterClass(m_clsid, NULL);
}

////////////////////////////////////////////////////////////////////////////
// CTF2005_AnaliseDesgPropPage: Constructor
////////////////////////////////////////////////////////////////////////////
CTF2005_AnaliseDesgPropPage::CTF2005_AnaliseDesgPropPage() :
    COlePropertyPage(IDD, IDS_VIDEOCONTROL_PPG_CAPTION)
{
    //{{AFX_DATA_INIT(CTF2005_AnaliseDesgPropPage)
```

```
    // NOTE: ClassWizard will add member initialization here
    //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_DATA_INIT
}

/////////////////////////////////////////////////////////////////////////////
// DoDataExchange: Moves data between page and properties
/////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgPropPage::DoDataExchange(CDataExchange* pDX)
{
    //{{AFX_DATA_MAP(CTF2005_AnaliseDesgPropPage)
    // NOTE: ClassWizard will add DDP, DDX, and DDV calls here
    //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_DATA_MAP
    DDP_PostProcessing(pDX);
}

/////////////////////////////////////////////////////////////////////////////
// CTF2005_AnaliseDesgPropPage message handlers
```

```
// TF2005_AnaliseDesgProcessor.cpp
//
// Description:
//          Implementation of the CTF2005_AnaliseDesgProcessor class.
//
// Author:  // TODO: Eduardo Finamore
//
// Version: // TODO: Version number.
// Date:    // TODO: Version release date.
// Version Note:
//          // TODO: Notes about the version.
//
// Note:    // TODO: Write general notes here
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
// Headers

#include "stdafx.h"
#include "TF2005_AnaliseDesg.h"
#include "TF2005_AnaliseDesgProcessor.h"
#include "SnapShot.h"


////////////////////////////////////////////////////////////////////////////////
// Compiler macros

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

////////////////////////////////////////////////////////////////////////////////
// Construction/Destruction
////////////////////////////////////////////////////////////////////////////////

CTF2005_AnaliseDesgProcessor::CTF2005_AnaliseDesgProcessor()
{
    // TODO: Initalize helper variables for video processing.
    m_control=CControl::instance();
}

CTF2005_AnaliseDesgProcessor::~CTF2005_AnaliseDesgProcessor()
{
    // TODO: Release taken resources.
}

////////////////////////////////////////////////////////////////////////////////
// Process: Process image buffer.
// Arguments:
//      frame   : Pointer to a data structure encapsulating the
//                frame buffer and some extra information.
////////////////////////////////////////////////////////////////////////////////
void CTF2005_AnaliseDesgProcessor::Process(ImageData *frame)
{
#ifdef _DEBUG
    // Sample processing.
    for(int i = 0; i < frame->lDataLen; i++){
        frame->pData[i] = 255;
    }
#else


    IplImage*       pCvData;
    IplImage*       pCvDataGray;
    IplImage*       pCvCopy;


    CvSize size=cvSize(frame->iSizeX,frame->iSizeY);
    pCvData = cvCreateImageHeader(size, IPL_DEPTH_8U, 3);
    int stride = (size.width * 3); // byte offset between row
    cvSetImageData(pCvData,frame->pData,stride);
```

```
                CvSize size2=cvSize(pCvData->width,pCvData->height);
                pCvDataGray=cvCreateImage(size2, IPL_DEPTH_8U, 1);
                pCvCopy=cvCreateImage(size2, IPL_DEPTH_8U, 1);

                cvCvtColor(pCvData,pCvDataGray, CV_BGR2GRAY);

                 CvPoint pt1;
                 CvPoint pt2;

                 pt1.x=int(m_control->m_xLeft*pCvData->width/1000);
                 pt1.y=int(pCvData->height-m_control->m_yBack*pCvData->height/1000);
                 pt2.x=int(m_control->m_xRight*pCvData->width/1000);
                 pt2.y=int(pCvData->height-m_control->m_yTop*pCvData->height/1000);

                 m_control->m_atualImageWidth=pCvData->width;
                 m_control->m_atualImageHeight=pCvData->height;


                 CvRect Roi;
                 Roi.x=pt1.x;
                 Roi.y=pt2.y;
                 Roi.width=pt2.x-pt1.x;
                 Roi.height=pt1.y-pt2.y;


                 //cvRectangle(pCvDataGray,pt1,pt2,255,-1);

                 cvSetImageROI(pCvDataGray, Roi );
                 cvSetImageROI(pCvCopy, Roi );
                 cvSetImageROI(pCvData, Roi );

           //  if(m_control->m_isThreshold==true)//m_control->m_thresh1
                cvThreshold(pCvDataGray,pCvDataGray,m_control->m_thresh1,255, CV_THRESH_TOZERO );
           //  else
           //  cvCanny(pCvDataGray,pCvDataGray,m_control->m_thresh1_canny,150);

                CvMemStorage* storage = cvCreateMemStorage(0);
                CvSeq* pSeq = 0;



                cvCopy(pCvDataGray, pCvCopy);
                //Find image contours points

                 CvPoint pt1b;
                 CvPoint pt2b;

                 pt1b.x=0;
                 pt1b.y=0;
                 pt2b.x=100;
                 pt2b.y=100;


                //cvRectangle(pCvCopy,pt1b,pt2b,255,-1);
                cvFindContours( pCvCopy, storage, &pSeq, sizeof(CvContour), CV_RETR_LIST, CV_CHAIN
_APPROX_NONE );


                CvContour* pContour = NULL;

                //Find saturated and isolator regions, the other regions will be find using geomet
ric relations



                 cvCvtColor(pCvDataGray,pCvData, CV_GRAY2BGR);



           //    m_control->maximal_contour_size=100;
```

```
            //fprintf(temp,"\n\n%d ",pSeq->total);
            //          fprintf(temp,"%d \n",m_control->minimal_contour_size);

                    while(pSeq != 0){


                    if(pSeq->total > m_control->minimal_contour_size && pSeq->total < m_control->m
aximal_contour_size&& pSeq->total !=2*Roi.width+2*Roi.height-100)
                    {

                        CvContour* contour = (CvContour*)pSeq;

                        //if(pSeq->total <500)
                        cvDrawContours(pCvData, pSeq, RGB(0,0,255), RGB(0,0,255), 0, 1, 8);

                        //if(temp)
                        //  fprintf(temp,"pSeq->total=%d maximacontour=%d\n",pSeq->total,m_control
->maximal_contour_size);

                        CvPoint pontoAtual;
                        CvSeqReader reader;
                        cvStartReadSeq(pSeq, &reader, 0 );

                        //fprintf(temp,"\n\n%d ",pSeq->total);
                        //fprintf(temp,"%d \n",m_control->minimal_contour_size);
                        //  fprintf(temp,"%d \n",Roi.height);
                        //      fprintf(temp,"%d \n",Roi.width);
                        FILE* temp;

                        if(m_control->m_bWriteFile==true)
                        {
                            temp=fopen(m_control->m_strFileName2Save,"a+");
                            //  fprintf(temp,"\n\npSeq->total=%d ",pSeq->total);
                            //  fprintf(temp,"minimal_contour_size=%d \n",m_control->minimal_conto
ur_size);

                            //  fprintf(temp,"Roi.height=%d \n",Roi.height);
                            //  fprintf(temp,"Roi.width=%d \n",Roi.width);
                            for(int i = 0; i < pSeq->total; i++ )
                            {


                                CV_READ_SEQ_ELEM(pontoAtual,reader );

                                if(pontoAtual.x>4&&pontoAtual.x<abs(pt2.x-pt1.x)-5&&pontoAtual.y>4
&&pontoAtual.y<abs(pt2.y-pt1.y)-5)
                                fprintf(temp,"%d %d\n",pontoAtual.x,pontoAtual.y);
                            }

                            fclose(temp);
                        }
                    }


                    pSeq= pSeq->h_next;
                }

                if(m_control->m_bWriteFile==true)
                m_control->m_bWriteFile=false;


                cvResetImageROI(pCvDataGray);
                cvResetImageROI(pCvCopy);
                cvResetImageROI(pCvData);
            /*
                 FILE* teste;
                 teste=fopen("c:\\teste.log","a+");
                 if(teste)
                 {
                      fprintf(teste,"x1=%d,y1=%d,x2=%d,y2=%d\n",m_control->m_xLeft,m_control->m_yB
ack,m_control->m_xRight,m_control->m_yTop);
                     fprintf(teste,"x1=%d,y1=%d,x2=%d,y2=%d\n",pt1.x,pt1.y,pt2.x,pt2.y);
                     fclose(teste);
                 }
```

```
     */

    cvRectangle(pCvData,pt1,pt2,RGB(0,0,255),1);

    cvReleaseImageHeader(&pCvData);
    cvReleaseImage(&pCvDataGray);
    cvReleaseImage(&pCvCopy);


    // TODO: Put your video processing algorithm here.

#endif
}
```

```
// VideoGraphBase.cpp
//
// Description:
//          Implementation of CVideoGraphBase class.
//
// Author:  // TODO: Eduardo Finamore
//
// Version: // TODO: Version number.
// Date:    // TODO: Version release date.
// Version Note:
//          // TODO: Notes about the version.
//
// Note:    This program was based on DirectX Doc/View template.
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// Headers

// Standard precompiled headers.
#include "stdafx.h"
// Class interface declaration.
#include "VideoGraphBase.h"


////////////////////////////////////////////////////////////////////////////
// Compiler macros

// Debug mode definitions.
#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif


////////////////////////////////////////////////////////////////////////////
// Helper function that uses a windowless render.

////////////////////////////////////////////////////////////////////////////
// Either VMR7 or VMR9 must be installed.
////////////////////////////////////////////////////////////////////////////
#ifdef _VMR
HRESULT InitWindowlessVMR(
    HWND hwndApp,                   // Window to hold the video.
    IGraphBuilder* pGraph,          // Pointer to the Filter Graph Manager.
    IVMRWindowlessControl** ppWc    // Receives a pointer to the VMR.
    )
{

////////////////////////////////////////////////////////////////////////////
// If VMR7 is installed
////////////////////////////////////////////////////////////////////////////
#ifdef _VMR7
    if (!pGraph || !ppWc) return E_POINTER;
    IBaseFilter* pVmr = NULL;
    IVMRWindowlessControl* pWc = NULL;
    // Creates the VMR.
    HRESULT hr = CoCreateInstance(CLSID_VideoMixingRenderer, NULL,
        CLSCTX_INPROC, IID_IBaseFilter, (void**)&pVmr);
    if (FAILED(hr))
    {
    #ifdef _DEBUG
        MessageBox(hwndApp,_T("CoCreateInstance VideoMixing FAILED!!"),_T("InitWindowlessV
MR"),0);
    #endif
        return hr;
    }

    // Add the VMR to the filter graph.
    hr = pGraph->AddFilter(pVmr, L"Video Mixing Renderer");
    if (FAILED(hr))
    {
        pVmr->Release();
        return hr;
```

```
        }

        // Set the rendering mode.
        IVMRFilterConfig* pConfig;
        hr = pVmr->QueryInterface(IID_IVMRFilterConfig, (void**)&pConfig);
        if (SUCCEEDED(hr))
        {
            hr = pConfig->SetRenderingMode(VMRMode_Windowless);
            pConfig->Release();
        } else {
        #ifdef _DEBUG
            MessageBox(hwndApp,_T("Quering FilterConfig FAILED!!"),_T("InitWindowlessVMR"),0);
        #endif
        }
        if (SUCCEEDED(hr))
        {
            // Set the window.
            hr = pVmr->QueryInterface(IID_IVMRWindowlessControl, (void**)&pWc); //VMR7
            if( SUCCEEDED(hr))
            {
                hr = pWc->SetVideoClippingWindow(hwndApp);
                if (SUCCEEDED(hr))
                {
                    *ppWc = pWc; // Return this as an AddRef'd pointer.
                }
                else
                {
        #ifdef _DEBUG
                    MessageBox(hwndApp,_T("Setting Clipping Window FAILED!!"),_T("InitWindowle
ssVMR"),0);
        #endif
                    // An error occurred, so release the interface.
                    pWc->Release();
                }
            }
        } else {
        #ifdef _DEBUG
            MessageBox(hwndApp,_T("Rendering Mode NOT SET"),_T("InitWindowlessVMR"),0);
        #endif
    #endif

    //////////////////////////////////////////////////////////////////////
    // If VMR9 is installed
    //////////////////////////////////////////////////////////////////////
    #ifdef _VMR9
        if (!pGraph || !ppWc) return E_POINTER;
        IBaseFilter* pVmr = NULL;
        IVMRWindowlessControl* pWc = NULL;
        // Creates the VMR.
        HRESULT hr = CoCreateInstance(CLSID_VideoMixingRenderer9, NULL,
            CLSCTX_INPROC, IID_IBaseFilter, (void**)&pVmr);
        if (FAILED(hr))
        {
        #ifdef _DEBUG
            MessageBox(hwndApp,_T("CoCreateInstance VideoMixing FAILED!!"),_T("InitWindowlessV
MR"),0);
        #endif
            return hr;
        }

        // Add the VMR to the filter graph.
        hr = pGraph->AddFilter(pVmr, L"Video Mixing Renderer 9");
        if (FAILED(hr))
        {
            pVmr->Release();
            return hr;
        }

        // Set the rendering mode.
        IVMRFilterConfig* pConfig;
        hr = pVmr->QueryInterface(IID_IVMR9FilterConfig, (void**)&pConfig);
        if (SUCCEEDED(hr))
        {
            hr = pConfig->SetRenderingMode(VMR9Mode_Windowless);
```

```
            pConfig->Release();
        } else {
        #ifdef _DEBUG
            MessageBox(hwndApp,_T("Quering FilterConfig FAILED!!"),_T("InitWindowlessVMR"),0);
        #endif
        }
        if (SUCCEEDED(hr))
        {
            // Set the window.
            hr = pVmr->QueryInterface(IID_IVMR9WindowlessControl, (void**)&pWc);
            if( SUCCEEDED(hr))
            {
                hr = pWc->SetVideoClippingWindow(hwndApp);
                if (SUCCEEDED(hr))
                {
                    *ppWc = pWc; // Return this as an AddRef'd pointer.
                }
                else
                {
        #ifdef _DEBUG
                    MessageBox(hwndApp,_T("Setting Clipping Window FAILED!!"),_T("InitWindowle
ssVMR"),0);
        #endif
                    // An error occurred, so release the interface.
                    pWc->Release();
                }
            }
        } else {
        #ifdef _DEBUG
            MessageBox(hwndApp,_T("Rendering Mode NOT SET"),_T("InitWindowlessVMR"),0);
        #endif
    #endif

        pVmr->Release();
        return hr;
    }
    #endif


    //////////////////////////////////////////////////////////////////////
    // CVideoGraphBase: The class handle the DirectShow graph building


    //////////////////////////////////////////////////////////////////////
    // Global helper functions
    //////////////////////////////////////////////////////////////////////
    //  Free an existing media type (ie free resources it holds)

    void WINAPI FreeMediaType(AM_MEDIA_TYPE& mt)
    {
        if (mt.cbFormat != 0) {
            CoTaskMemFree((PVOID)mt.pbFormat);

            // Strictly unnecessary but tidier
            mt.cbFormat = 0;
            mt.pbFormat = NULL;
        }
        if (mt.pUnk != NULL) {
            mt.pUnk->Release();
            mt.pUnk = NULL;
        }
    }

    // general purpose function to delete a heap allocated AM_MEDIA_TYPE structure
    // which is useful when calling IEnumMediaTypes::Next as the interface
    // implementation allocates the structures which you must later delete
    // the format block may also be a pointer to an interface to release

    void WINAPI DeleteMediaType(AM_MEDIA_TYPE *pmt)
    {
        // allow NULL pointers for coding simplicity

        if (pmt == NULL) {
            return;
```

```
    }

    FreeMediaType(*pmt);
    CoTaskMemFree((PVOID)pmt);
}


//////////////////////////////////////////////////////////////////////
// Constructor : Init member variable and search for available devices
//////////////////////////////////////////////////////////////////////
CVideoGraphBase::CVideoGraphBase()
{
    CoInitialize(NULL);             // Init COM

    m_pCaptureGraphBuilder2 = NULL;
    m_pGraphBuilder         = NULL;
    m_pFilterGraph          = NULL;
    m_pMediaControl         = NULL;
    m_pVideoWindow          = NULL;
    m_pMediaSeeking         = NULL;
    m_pSrcFilter            = NULL;
    m_pRenderFilter         = NULL;

    m_hRenderWnd            = NULL;
    m_pCaptureSetting       = NULL;
    m_fIsScaling            = false; // no scaling

    m_fRegisterFilterGraph  = true;  // register for GraphEdt to view
    m_dwGraphRegister       = 0;     // the register no. of the graph

    m_wcMediaName[0]        = '\0';
    m_wcOutputFile[0]       = '\0';

    // enumerate all available video capture devices
    m_nNumOfCapDev = 0;
    m_iSizeX = 320;
    m_fAvgFrmRate = 30;

    HRESULT hr;
    CComPtr<ICreateDevEnum> pCreateDevEnum;
    CComPtr<IEnumMoniker> pEnumMoniker = NULL;

    hr = CoCreateInstance(CLSID_SystemDeviceEnum, NULL,
                          CLSCTX_INPROC_SERVER,
                          IID_ICreateDevEnum,
                          (void**)&pCreateDevEnum);
    if (FAILED(hr)) return; // no capture device enumerator

    hr = pCreateDevEnum->
            CreateClassEnumerator(CLSID_VideoInputDeviceCategory,
                                  &pEnumMoniker, 0);
    if (FAILED(hr) || !pEnumMoniker) return; // no capture device

    // Search all available devices and save in m_pCapMonikers[]
    ULONG cFetched;
    // Note that if the Next() call succeeds but there are no monikers,
    // it will return S_FALSE (which is not a failure).  Therefore, we
    // check that the return code is S_OK instead of using SUCCEEDED()
    while (S_OK == pEnumMoniker->Next(1, m_pCapMonikers+m_nNumOfCapDev,
                                      &cFetched))
        m_nNumOfCapDev ++;
}

//////////////////////////////////////////////////////////////////////
// Deconstructor : Release all resources
//////////////////////////////////////////////////////////////////////
CVideoGraphBase::~CVideoGraphBase()
{
    // Release the stored available capture device
    for (int i = 0; i < m_nNumOfCapDev; i++)
        SafeRelease(m_pCapMonikers[i]);

    ReleaseFilterGraph();       // destroy the filter graph
```

```
    CoUninitialize();           // release COM
}

//////////////////////////////////////////////////////////////////////
// SetRenderWindow: Set the window to display the video
// Auguments:
//      hWnd:       the display window handler
//      pImgRect:   display the video in the rectangle in window
//                  if NULL, display video in up-left conner (0, 0) at
//                  original video size
// return value:    error code of set video window
//////////////////////////////////////////////////////////////////////
HRESULT CVideoGraphBase::SetRenderWindow(HWND hWnd, RECT * pImgRect)
{
    m_hRenderWnd = hWnd;  // set the render window

    if (pImgRect == NULL) {
        // display video in original size at up-left corner.
//      m_fIsScaling = false;
        m_oVideoRect.left = m_oVideoRect.top = 0;
    }
    else {
        // Set the rect to display the video and mark the flag
        m_oVideoRect = *pImgRect;
//      m_fIsScaling = true;
    }
    // update the video window size
    return setVideoWindowSize();
}

//////////////////////////////////////////////////////////////////////
// ReleaseFilterGraph : Release the whole graph
//////////////////////////////////////////////////////////////////////
void CVideoGraphBase::ReleaseFilterGraph()
{
    // Destroy the whole filter graph
    stopGraph();
    SafeRelease( m_pMediaControl );
    SafeRelease( m_pMediaSeeking );
    SafeRelease( m_pSrcFilter );
    SafeRelease( m_pRenderFilter );
    SafeRelease( m_pVideoWindow  );
    SafeRelease( m_pFilterGraph  );
    SafeRelease( m_pGraphBuilder );
    SafeRelease( m_pCaptureGraphBuilder2 );
}

//////////////////////////////////////////////////////////////////////
// RenderFile: Playback the given media file
// Auguments:
//   vname: the name of the media file
// Return value: Error code of building playback graph
//////////////////////////////////////////////////////////////////////
HRESULT CVideoGraphBase::RenderFile(const TCHAR * vname)
{
    // set the VideoSource to indicate we are play video file
    m_nVideoSource = VideoFilePlayback;
    // change the given file name to wide character in m_wcMediaName
    USES_CONVERSION;
    TCHAR tempVName[MAX_PATH];
    _tcscpy(tempVName, vname);
    wcscpy(m_wcMediaName, T2W(tempVName));

    HRESULT hr;
    // Create the essensial element of a video graph
    JIF(createFilterGraph());

    // Create seeking control to navigate through the video file
    // May fail for live camera. (but no need to return even we fail)
    hr = m_pGraphBuilder->QueryInterface(IID_IMediaSeeking,
                                (void**)&m_pMediaSeeking);
    if (SUCCEEDED(hr))
        m_pMediaSeeking->SetTimeFormat(&TIME_FORMAT_FRAME);
    else
```

```
            m_pMediaSeeking = NULL;

        // Start the graph (start playing)
        JIF(startGraph());
        return hr;
    }

    ////////////////////////////////////////////////////////////////////////
    // GetNumCapDev : Return the total number of available capture devices
    ////////////////////////////////////////////////////////////////////////
    int  CVideoGraphBase::GetNumCapDev()
    {
        return m_nNumOfCapDev;
    }

    ////////////////////////////////////////////////////////////////////////
    // GetCapDevName: Get the name of given capture device.
    // Auguments:
    //   i:           the index of the capture device we want
    //   StrName:     the string to store the friendly name of given device
    //                if NULL, the name is not retrieve
    //   DispName:    the string to store the device name (for machine)
    //                if NULL, the name is not retrieve
    // return value: false if there is no ith cpature device. else true.
    ////////////////////////////////////////////////////////////////////////
    bool CVideoGraphBase::GetCapDevName(int i, TCHAR *pDevStrName, TCHAR *pDevDispName)
    {
        if (pDevStrName)  pDevStrName[0] = '\0';
        if (pDevDispName) pDevDispName[0] = '\0';

        HRESULT   hr;
        if ( i < m_nNumOfCapDev ) {
            IMoniker* pM = m_pCapMonikers[i];
            // search through the capture devices
            // Get friendly name (for user to see it)
            if (pDevStrName) {
                IPropertyBag *pBag;
                hr = pM->BindToStorage(0, 0, IID_IPropertyBag, (void **)&pBag);
                if(SUCCEEDED(hr)) {
                    VARIANT var;
                    var.vt = VT_BSTR;
                    hr = pBag->Read(L"FriendlyName", &var, NULL);
                    if (hr == NOERROR) {
                        USES_CONVERSION;
                        _tcscpy(pDevStrName, OLE2T(var.bstrVal));
                        SysFreeString(var.bstrVal);
                    }
                    pBag->Release();
                }
            }
            // Get Device Name (machine will understand this string)
            if (pDevDispName) {
                WCHAR *szDispName;
                if (SUCCEEDED(pM->GetDisplayName(0, 0, &szDispName)))
                {
                    wsprintf(pDevDispName, TEXT("%S"), szDispName?szDispName:L"");
                    CoTaskMemFree(szDispName);
                }
            }
            return true;
        }
        else return false;
    }

    ////////////////////////////////////////////////////////////////////////
    // PreviewLive: Find specified capturing device to preview live video
    // Auguments:
    //   capInd: the index of the capture device to use
    //   fSetup: true, a dialog is popped up first to set capture format
    //           (color space, video size and framerate)
    //           false, default capturing (320 x 240 at 30fps)
    // Return value: Error code of building preview graph
    ////////////////////////////////////////////////////////////////////////
    HRESULT CVideoGraphBase::PreviewLive(int capInd, bool fSetup)
```

```
{
    // check if the specified capture device exists
    if ( capInd >= m_nNumOfCapDev  || capInd < 0 ) return E_FAIL;
    // set VideoSource so that createFilterGraph() knows what to build
    m_nVideoSource = (enum VideoSource)capInd;
    m_fShowCaptureProperties = fSetup;
    HRESULT hr;
    // Build graph according to the setting: e.g VideoSource ...
    JIF(createFilterGraph());
    // Start the graph
    JIF(startGraph());
    return hr;
}

////////////////////////////////////////////////////////////////////////
// PreviewLive: Find specified capturing device to preview live video
// Auguments:
//   capName: the name of capture device to use
//   fSetup : true, a dialog is popped up first to set capture format
//            (color space, video size and framerate)
//            false, default capturing (320 x 240 at 30fps)
// Return value: Error code of building preview graph
////////////////////////////////////////////////////////////////////////
HRESULT CVideoGraphBase::PreviewLive(TCHAR* capName, bool fSetup)
{
    int ind = 0;
    // if capName is not empty, find corresponding capture device
    // Otherwise use the first available capture device (ind == 0).
    if (_tcsclen(capName) > 0) {
        // change to WCHAR
        WCHAR wcCapName[1024];
        USES_CONVERSION;
        TCHAR tempVName[MAX_PATH];
        _tcscpy(tempVName, capName);
        wcscpy(wcCapName, T2W(tempVName));
        // Create IMoniker from DisplayName
        IBindCtx *lpBC;
        HRESULT hr = CreateBindCtx(0, &lpBC);
        IMoniker *pmVideo = 0;
        if (SUCCEEDED(hr))
        {
            DWORD dwEaten;
            hr = MkParseDisplayName(lpBC, wcCapName, &dwEaten,
                                    &pmVideo);
            lpBC->Release();
        }
        else return E_FAIL;  // not a valid capture device name

        // Find the IMoniker from the available capture devices
        for (ind = 0; ind < m_nNumOfCapDev; ind++)
            if (S_OK == m_pCapMonikers[ind]->IsEqual(pmVideo)) break;
        SafeRelease(pmVideo);
    }

    // Build capture graph using this capture device
    return PreviewLive(ind, fSetup);
}

////////////////////////////////////////////////////////////////////////
// CaptureToFile: set the output file to save the captured video
// Auguments:
//   vname : the name of the file
////////////////////////////////////////////////////////////////////////
HRESULT CVideoGraphBase::CaptureToFile(const TCHAR *vname)
{
    // change the given file name to wide character in m_wcOutputFile
    if( vname && _tcsclen(vname) > 0 ) {
        // change the given file name to wide character in m_wcMediaName
        USES_CONVERSION;
        TCHAR tempVName[MAX_PATH];
        _tcscpy(tempVName, vname);
        wcscpy(m_wcOutputFile, T2W(tempVName));
    }
```

```
    HRESULT hr;
    // Build graph according to the setting: e.g VideoSource ...
    JIF(createFilterGraph());

    // Start the graph
    JIF(startGraph());

    m_wcOutputFile[0] = '\0';
    return hr;
}

////////////////////////////////////////////////////////////////////////
// StartCapture: Begin capture to file
////////////////////////////////////////////////////////////////////////
HRESULT CVideoGraphBase::StartCapture()
{
    const LONGLONG MAX_TIME = 0x7FFFFFFFFFFFFFFF;    /* Maximum LONGLONG value */
    // Set the start time to 0 and stop time to MAX_TIME
    REFERENCE_TIME stop = MAX_TIME;
    HRESULT hr = m_pCaptureGraphBuilder2->ControlStream(&PIN_CATEGORY_CAPTURE, NULL,
                NULL, NULL, &stop, 0, 0);
    return hr;
}

////////////////////////////////////////////////////////////////////////
// StopCapture: Stop capturing to file
////////////////////////////////////////////////////////////////////////
HRESULT CVideoGraphBase::StopCapture()
{
    const LONGLONG MAX_TIME = 0x7FFFFFFFFFFFFFFF;    /* Maximum LONGLONG value */
    // Set the start time to MAX_TIME to prevent capture
    REFERENCE_TIME start = MAX_TIME;
    HRESULT hr = m_pCaptureGraphBuilder2->ControlStream(&PIN_CATEGORY_CAPTURE, NULL,
                NULL, &start, NULL, 0, 0);
    return hr;
}

////////////////////////////////////////////////////////////////////////
// AdjustCamera: Setup the video camera currently being used
// ( e.g exposure, white balance, contrast ... )
// Return value: Error code
////////////////////////////////////////////////////////////////////////
HRESULT CVideoGraphBase::AdjustCamera()
{
    // Setup the camera by property page
    HRESULT hr = S_FALSE;  // Treated as success.
    // If the current graph is for capturing.
    if (m_nVideoSource >= 0 && m_pSrcFilter) {
        // display the property page of source filter
        JIF(showPropertyPage(m_pSrcFilter, L"Adjust Camera"));
    }
    return hr;
}

////////////////////////////////////////////////////////////////////////
// Play : Play the video (media file or video capture device)
////////////////////////////////////////////////////////////////////////
void CVideoGraphBase::Play()
{
    // Begin the palying
    if (m_pMediaControl) m_pMediaControl->Run();
}

////////////////////////////////////////////////////////////////////////
// Pause : Pause the video (media file or video capture device)
// (Different from Stop(). It does not stop the capture source.)
////////////////////////////////////////////////////////////////////////
void CVideoGraphBase::Pause()
{
    // Pause the video
    if (m_pMediaControl) {
        m_pMediaControl->Pause();
        m_pMediaControl->StopWhenReady();
    }
```

```
}

//////////////////////////////////////////////////////////////////////
// Stop : Stop the video (media file or video capture device)
// (Different from Pause(). It stops all the filters in the graph.)
//////////////////////////////////////////////////////////////////////
void CVideoGraphBase::Stop()
{
    // Stop the video
    if (m_pMediaControl) {
        m_pMediaControl->Stop();
        m_pMediaControl->StopWhenReady();
    }
}

//////////////////////////////////////////////////////////////////////
// Seek: Seek in the video file by relative position
// Auguments:
//   offset : offset to current position
// Return value: Current position after seeking
//////////////////////////////////////////////////////////////////////
LONGLONG CVideoGraphBase::Seek(int offset)
{
    LONGLONG cur_pos = -1;
    OAFilterState  fs;
    if (m_pMediaSeeking) {
        // Wait for the other seeking operation to finish.
        m_pMediaControl->GetState(INFINITE, &fs);
        // Seek with respect to current position.
        m_pMediaSeeking->GetCurrentPosition(&cur_pos);
        cur_pos += offset;
        m_pMediaSeeking->SetPositions(&cur_pos,
                                      AM_SEEKING_AbsolutePositioning,
                                      NULL, AM_SEEKING_NoPositioning);
        // wait until the seeking is finished
        m_pMediaControl->GetState(INFINITE, &fs);
        // display the new frame
        m_pMediaControl->StopWhenReady();
    }
    return cur_pos;
};

//////////////////////////////////////////////////////////////////////
// SeekToStart: Seek to the begin of the video file
// Return value: Current position after seeking
//////////////////////////////////////////////////////////////////////
LONGLONG CVideoGraphBase::SeekToStart()
{
    // Goto the start of the video file
    LONGLONG cur_pos = 0;
    OAFilterState  fs;
    if (m_pMediaSeeking) {
        // Wait for the other seeking operation finish.
        m_pMediaControl->GetState(INFINITE, &fs);
        // Seek to start
        m_pMediaSeeking->SetPositions(&cur_pos,
                                      AM_SEEKING_AbsolutePositioning,
                                      NULL, AM_SEEKING_NoPositioning);
        // this function will wait until the seeking is finished
        m_pMediaControl->GetState(INFINITE, &fs);
        // display the new frame
        m_pMediaControl->StopWhenReady();
    }
    return cur_pos;
}

//////////////////////////////////////////////////////////////////////
// SeekToEnd: Seek to the end of the video file
// Return value: Current position after seeking
//////////////////////////////////////////////////////////////////////
LONGLONG CVideoGraphBase::SeekToEnd()
{
    // Goto the end of the video file
    LONGLONG cur_pos = 0;
```

```
    OAFilterState  fs;
    if (m_pMediaSeeking) {
        // Wait for the other seeking operation finish.
        m_pMediaControl->GetState(INFINITE, &fs);
        // Seek to the last frame
        m_pMediaSeeking->GetDuration(&cur_pos);
        m_pMediaSeeking->SetPositions(&cur_pos,
                                      AM_SEEKING_AbsolutePositioning,
                                      NULL, AM_SEEKING_NoPositioning);
        // this function will wait until the seeking is finished
        m_pMediaControl->GetState(INFINITE, &fs);
        // display the new frame
        m_pMediaControl->StopWhenReady();
    }
    return cur_pos;
}

//////////////////////////////////////////////////////////////////////
// GetVideoSourceName: Get the video source name
// Auguments:
//   vname: where to store the name. need allocate outside of this func
//     If playing file, vname = media file name
//     If capturing,    vname = capture device name
// return value: index of the video source (See enum VideoSource)
//////////////////////////////////////////////////////////////////////
CVideoGraphBase::VideoSource CVideoGraphBase::GetVideoSource(TCHAR *vn)
{
    if (vn) {
        if (m_nVideoSource >= 0 )
            GetCapDevName(m_nVideoSource, vn);
        else if( m_nVideoSource < 0 ) {
            USES_CONVERSION;
            _tcscpy(vn, W2T(m_wcMediaName));
        }
    }
    return m_nVideoSource;
}

//////////////////////////////////////////////////////////////////////
// GetVideoSize: Get the size of the video
//////////////////////////////////////////////////////////////////////
SIZE CVideoGraphBase::GetVideoSize()
{
    // Get the size of the video through the IBasicVideo interface.
    SIZE    s = {0, 0};
    HRESULT hr = E_FAIL;
    IBasicVideo *pBasicVideo = NULL;
    if (m_pGraphBuilder)
        hr = m_pGraphBuilder->QueryInterface(IID_IBasicVideo, (void **)&pBasicVideo);
    if (SUCCEEDED(hr)) {
        pBasicVideo->get_SourceWidth(&(s.cx));
        pBasicVideo->get_SourceHeight(&(s.cy));
        SafeRelease(pBasicVideo);
    }
    return s;
}

//////////////////////////////////////////////////////////////////////
// GetAvgFrameRate: Get the frame rate achieved by the Rendering filter
//////////////////////////////////////////////////////////////////////
double CVideoGraphBase::GetAvgFrameRate()
{
    int frameRate = 0;
    HRESULT hr = E_FAIL;
    CComPtr<IQualProp> pQualProp = NULL;
    if (m_pRenderFilter)
        hr = m_pRenderFilter->QueryInterface(IID_IQualProp, (void **)&pQualProp);
    if (SUCCEEDED(hr))
        hr = pQualProp->get_AvgFrameRate(&frameRate);

    return double(frameRate/100.);
}

//////////////////////////////////////////////////////////////////////
```

```
// GetFrameNumber: Get the current frame position
//////////////////////////////////////////////////////////////////////
LONGLONG CVideoGraphBase::GetFrameNumber()
{
    // Get current frame number
    LONGLONG cur_pos = -1;
    // return the current position in video file in frames number
    // (we set the unit to frame number format)
    if (m_pMediaSeeking)
        m_pMediaSeeking->GetCurrentPosition(&cur_pos);
    return cur_pos;
}

//////////////////////////////////////////////////////////////////////
// IsGraphRunning: Check if the graph is running or not
//////////////////////////////////////////////////////////////////////
bool CVideoGraphBase::IsGraphRunning()
{
    // Is the graph in running mode
    OAFilterState  fs;
    if (m_pMediaControl) {
        m_pMediaControl->GetState(INFINITE, &fs);
        if (fs == State_Running) return true;
    }
    return false;
}

//////////////////////////////////////////////////////////////////////
// createFilterGraph: Create a graph to capture or playback the video
// return value: error code of creating graph.
//////////////////////////////////////////////////////////////////////
HRESULT CVideoGraphBase::createFilterGraph()
{
    HRESULT hr;
    // First destroy and release all the old graph
    ReleaseFilterGraph();
    // Then, create the essential interface for building filter graph
    // IGraphBuilder, ICaptureGraphBuilder2, IFilterGraph,
    // IMediaControl and IVideoWindow
    JIF(CoCreateInstance( CLSID_FilterGraph, NULL, CLSCTX_INPROC,
                        IID_IGraphBuilder, (void **)&m_pGraphBuilder ));
    JIF(CoCreateInstance (CLSID_CaptureGraphBuilder2 , NULL, CLSCTX_INPROC,
        IID_ICaptureGraphBuilder2, (void **) &m_pCaptureGraphBuilder2));
    JIF(m_pCaptureGraphBuilder2->SetFiltergraph(m_pGraphBuilder));

    JIF(m_pGraphBuilder->QueryInterface(IID_IFilterGraph,(void**)&m_pFilterGraph));
    JIF(m_pGraphBuilder->QueryInterface(IID_IMediaControl,(void**)&m_pMediaControl));
    JIF(m_pGraphBuilder->QueryInterface(IID_IVideoWindow, (void**)&m_pVideoWindow));

    // Create essential component and connect the graph
    // 1. Source Filter and get its output pin for further processing
    JIF(addSrcFilter(&m_pSrcFilter));
    CComPtr< IPin > pSrcOut = NULL, pProcessedOut = NULL;
    JIF(get_pin(m_pSrcFilter, PINDIR_OUTPUT, 0, &pSrcOut));
    // 2. Extra processing filters for the output pin
    JIF(addExtraFilters(pSrcOut, &pProcessedOut));
    // 3. Render the output pin of output of Extra Processing
    JIF(addRenderFilters(pProcessedOut));

    return hr; // success build the graph
}

//////////////////////////////////////////////////////////////////////
// startGraph: start to play the graph
// return value: error code
//////////////////////////////////////////////////////////////////////
HRESULT CVideoGraphBase::startGraph()
{
    // Start the graph to render the video
    HRESULT hr = E_FAIL;

    if( m_pMediaControl && m_pVideoWindow ){

#ifdef _VMR
```

```
        // Try to use a VMR render.
        m_pWc = NULL;
        m_VMRSucceeded = false;

        hr = InitWindowlessVMR(m_hRenderWnd, m_pGraphBuilder, &m_pWc);
        errorHandler(TEXT("InitWindowlessVMR called!"), hr);
        if (SUCCEEDED(hr)) {
            m_VMRSucceeded = true;
            errorHandler(TEXT("O dispositivo Render VMR foi localizado"), hr);
        } else {    // If there's no VMR render.
            errorHandler(TEXT("Nao existe VMR!! Usando Video Render"), hr);
#endif

            // set render window to IVideoWindow.
            // If m_hRenderWnd == NULL, video will be displayed in new win
            if (m_hRenderWnd) {
                JIF(m_pVideoWindow->put_Owner((OAHWND)m_hRenderWnd));
#ifdef _VMR
                JIF(m_pVideoWindow->put_WindowStyle(WS_CHILD | WS_CLIPSIBLINGS));
                JIF(m_pVideoWindow->put_MessageDrain((OAHWND)m_hRenderWnd));
#endif
                // Set the display position
            }
#ifdef _VMR
        }
#endif
        // If we want the filter graph accessible by GraphEdt.exe
        if (m_fRegisterFilterGraph) {
            // Add the graph to Rot so that the graphedit can view it
            hr = addGraphToRot(m_pGraphBuilder, &m_dwGraphRegister);
            if (FAILED(hr)) m_dwGraphRegister = 0;
        }
        // Start the graph
        JIF(m_pMediaControl->Run());
        JIF(setVideoWindowSize());
    }
    return hr;
}

////////////////////////////////////////////////////////////////////////
// stopGraph: Stop playing the graph
// return value: error code
////////////////////////////////////////////////////////////////////////
void CVideoGraphBase::stopGraph()
{
    if( m_pMediaControl && m_pVideoWindow ){
        // Stop the graph
        m_pMediaControl->Stop();
#ifdef _VMR
        if(this->m_VMRSucceeded){   // If VMR render was used.
            // Release the VMR interface when you are done.
            m_pWc->Release();
        }
        else {      // If a simple video render was used.
#endif
            m_pVideoWindow->put_Visible(OAFALSE);
            m_pVideoWindow->put_Owner(NULL);
            m_pVideoWindow->put_MessageDrain(0);
#ifdef _VMR
        }
#endif

        if (m_fRegisterFilterGraph) {
            if (m_dwGraphRegister)
                removeGraphFromRot(m_dwGraphRegister);
        }
    }
}

////////////////////////////////////////////////////////////////////////
// setVideoWindowSize: set the window to render video
//   (size is decided by m_oVideoRect and m_fIsScaling)
// return value: error code
////////////////////////////////////////////////////////////////////////
```

```
HRESULT CVideoGraphBase::setVideoWindowSize()
{
    // if no graph exist, return S_FALSE which is treated as success
    HRESULT hr = S_FALSE;

    // If no scaling is set, set display rectangle to video size
    if (m_pGraphBuilder && !m_fIsScaling) {     // display at original size
        SIZE s = GetVideoSize();
        // Store the dimension of the image into the m_oVideoRect
        m_oVideoRect.right = m_oVideoRect.left + s.cx;
        m_oVideoRect.bottom = m_oVideoRect.top + s.cy;
    }

#ifdef _VMR
    // Set the video window position showing contours
    if (m_pVideoWindow) {
        JIF(m_pVideoWindow->SetWindowPosition( m_oVideoRect.left,
                                m_oVideoRect.top,
                                m_oVideoRect.right - m_oVideoRect.left,
                                m_oVideoRect.bottom - m_oVideoRect.top ));
    }
#else
    // Set the video window position hidding frame contours
    if (m_pVideoWindow) {
        JIF(m_pVideoWindow->SetWindowPosition( m_oVideoRect.left - 3,
                                m_oVideoRect.top - 22,
                                m_oVideoRect.right - m_oVideoRect.left + 6,
                                m_oVideoRect.bottom - m_oVideoRect.top + 25));
    }
#endif
    return hr;
}

//////////////////////////////////////////////////////////////////////////
// addSrcFilter: Add a source filter into graph according the setting
// of m_nVideoSource ( video file, capture devices ... )
// Auguments:
//   ppSrc   : to receive the created source filter
// return value: error code
//////////////////////////////////////////////////////////////////////////
HRESULT CVideoGraphBase::addSrcFilter(IBaseFilter **ppSrc)
{
    // create the source filter
    HRESULT hr;
    if (m_nVideoSource >= 0) { // create the capture filter
        // Bind the specified capture Moniker to a filter object
        JIF(m_pCapMonikers[m_nVideoSource]->BindToObject(0,0,IID_IBaseFilter,
                                                    (void**)ppSrc));
        JIF(setupCapture(*ppSrc)); // setup the capture
    }
    else if (m_nVideoSource == -1) { // video file source filter
        JIF(m_pGraphBuilder->AddSourceFilter(m_wcMediaName,
                                        L"Video File", ppSrc));
    }

    return hr;
}

//////////////////////////////////////////////////////////////////////////
// addExtraFilter: Add extra filter to process output of source filter
// Auguments:
//   pIn  : the input pin
//   ppOut: To receive the output (processed by Extra filters) pin
// return value: error code
//////////////////////////////////////////////////////////////////////////
HRESULT CVideoGraphBase::addExtraFilters(IPin * pIn, IPin ** ppOut)
{
    // no extra filters is needed in this class
    // if you need extra filters, override this function
    *ppOut = pIn;
    // remember to increase the ref count after assign pointer
    // Otherwise, it will release pIn twice and cause memory problem.
    pIn->AddRef();
    return S_OK;
```

```
}

//////////////////////////////////////////////////////////////////////
// addRenderFilter: Render the given *ppPinIn with a render filter. Now
// we have two kinds of rendering:
// 1. Output file is not set, we use video renderer to render it.
// 2. Output to a file, both a preview (render in a window) and a
// capture (save to a avi file) is setup.
// Auguments:
//   ppPinIn: the pin to render.
// return value: error code
//////////////////////////////////////////////////////////////////////
HRESULT CVideoGraphBase::addRenderFilters(IPin *pIn)
{
    HRESULT hr;
    // render the pPinIn (Preview Pin) with video renderer
    // we need m_pRenderFilter to get achieved frame rate. Otherwise,
    // just call m_pGraphBuilder->Render(*ppPinIn) is ok.
    JIF(CoCreateInstance(CLSID_VideoRenderer , NULL, CLSCTX_INPROC_SERVER,
                IID_IBaseFilter, (void **)&m_pRenderFilter));
    JIF(m_pFilterGraph->AddFilter(m_pRenderFilter, L"Video Renderer"));

    // If output file is not set, we just use video renderer.
    if (m_wcOutputFile[0] == '\0') {
        return (m_pGraphBuilder->Render(pIn));
    }

    // Otherwise, we need to split input to 2 (preview & capture)
    // Use Smart Tee to split the input pin to Preview & Capture Pins
    // Different from Inf Pin Tee, Smart Tee support independent
    // control of capture pin and preview pin so that we can start
    // preivew and capture independently. Inf Pin Tee does not.
    CComPtr <IPin>            pTmpPin   = NULL;
    CComPtr <IBaseFilter>     pf        = NULL;
    CComPtr <IFileSinkFilter> pSink     = NULL;
    CComPtr <IBaseFilter>     pSmartTee = NULL;
    JIF(CoCreateInstance(CLSID_SmartTee , NULL, CLSCTX_INPROC_SERVER,
                    IID_IBaseFilter, (void **)&pSmartTee));
    JIF(m_pFilterGraph->AddFilter(pSmartTee, L"Smart Tee"));
    JIF(get_pin(pSmartTee, PINDIR_INPUT, 0, &pTmpPin));
    JIF(m_pGraphBuilder->Connect(pIn, pTmpPin));

    // Preview branch
    pTmpPin = NULL;
    JIF(get_pin(pSmartTee, PINDIR_OUTPUT, 1, &pTmpPin));// Preview Pin
    JIF(m_pGraphBuilder->Render(pTmpPin)); //use added m_pRenderFilter

    // Capture branch
    pTmpPin = NULL;
    JIF(get_pin(pSmartTee, PINDIR_OUTPUT, 0, &pTmpPin));// Capture pin
    // Render the capture pin with a file writer to save video
    JIF(m_pCaptureGraphBuilder2->SetOutputFileName(&MEDIASUBTYPE_Avi,
                                m_wcOutputFile, &pf, &pSink));
    JIF(m_pCaptureGraphBuilder2->RenderStream(NULL, NULL, pTmpPin, NULL, pf));

    return hr;
}

//////////////////////////////////////////////////////////////////////
// setupCapture: Add capture source filter and set the capture format
// (frame rate, resolution...)
// Auguments:
//   pSrcFilter: the capture filter to set
// return value: error code
//////////////////////////////////////////////////////////////////////
HRESULT CVideoGraphBase::setupCapture(IBaseFilter * pSrcFilter)
{
    HRESULT hr;
    // Add Capture filter and init ICaptureGraphBuilder2 interface
    // we may need it if want to capture video to files
    JIF(m_pGraphBuilder->AddFilter(pSrcFilter, L"Video Capture"));


    // Creates the crossbar option menu.
```

```
    CComPtr<IAMCrossbar> pX;  // Media Stream config interface

    // Query for the MEDIATYPE_Interleaved crossbar.
    hr = m_pCaptureGraphBuilder2->FindInterface(&PIN_CATEGORY_CAPTURE,
        &MEDIATYPE_Interleaved, pSrcFilter, IID_IAMCrossbar, (void **)&pX);
    if(hr != S_OK){
        // Query for the MEDIATYPE_Video crossbar.
        DEBUG_MESSAGE(_T("MEDIATYPE_Interleaved"),_T("ERROR !!"));
        hr = m_pCaptureGraphBuilder2->FindInterface(&PIN_CATEGORY_CAPTURE,
            &MEDIATYPE_Video, pSrcFilter, IID_IAMCrossbar, (void **)&pX);
        if(hr == S_OK){
            DEBUG_MESSAGE(_T("MEDIATYPE_Video"),_T("OK !"));
        } else {
            DEBUG_MESSAGE(_T("MEDIATYPE_Video"),_T("ERROR !!"));
        }
    } else {
        DEBUG_MESSAGE(_T("MEDIATYPE_Interleaved"),_T("OK !!"));
    }
    if (!m_fShowCaptureProperties) { // default capture setup
    }
    // set capture channel by property page
    else {
        showPropertyPage(pX, L"Setup the channel");
    }

    // Now decide the capturing config by the property page or set a
    // default one. (i.e. Frame rate, resolution.....)
    CComPtr<IAMStreamConfig> pSC;  // Media Stream config interface
    // check capture device capabilities
    JIF(m_pCaptureGraphBuilder2->FindInterface(&PIN_CATEGORY_CAPTURE,
        &MEDIATYPE_Video, pSrcFilter, IID_IAMStreamConfig, (void **)&pSC));

    AM_MEDIA_TYPE *pmt;
    int iCount, iSize, ind;
    if (!m_fShowCaptureProperties) { // default capture setup
        // default capture: 320 * 240 resolution, 15f/s
        VIDEO_STREAM_CONFIG_CAPS caps;
        pSC->GetNumberOfCapabilities(&iCount, &iSize);
        for (ind = 0; ind < iCount; ind++) {
            pSC->GetStreamCaps(ind, &pmt, (BYTE *)&caps);
            if (caps.InputSize.cx == m_iSizeX) {
                // Set the capturing frame rate
                VIDEOINFOHEADER *pvi = (VIDEOINFOHEADER *)pmt->pbFormat;
                pvi->AvgTimePerFrame = (LONGLONG)(10000000 / m_fAvgFrmRate);
                JIF(pSC->SetFormat(pmt));
                DeleteMediaType(pmt);
                break; // found default setup, quit the loop
            }
            DeleteMediaType(pmt);
        }
    }
    // if m_fShowCaptureProperties == 1
    // set capture format by property page
    else {
        showPropertyPage(pSC, L"Setup the capture");
    }

    return hr;
}

//////////////////////////////////////////////////////////////////////////
// showPropertyPage: Display the property page of a COM object
// Auguments:
//   pIU:    A interface of the COM object
//   name:   the name of the dialog box of the property page
// return value: error code
//////////////////////////////////////////////////////////////////////////
HRESULT CVideoGraphBase::showPropertyPage(IUnknown* pIU, const WCHAR* name)
{
    HRESULT hr;
    if (pIU) {
        ISpecifyPropertyPages* pispp = 0;
        JIF(pIU->QueryInterface(IID_ISpecifyPropertyPages, (void **)&pispp));
```

```
            CAUUID caGUID;
            if (SUCCEEDED(pispp->GetPages(&caGUID))) {
                OleCreatePropertyFrame(m_hRenderWnd, 0, 0,
                    name,      // Caption for the dialog box
                    1,         // Number of filters
                    (IUnknown**)&pIU,
                    caGUID.cElems,
                    caGUID.pElems,
                    0, 0, 0);
                // Release the memory
                CoTaskMemFree(caGUID.pElems);
            }
            SafeRelease(pispp);
        }
        return hr;
    }

    ////////////////////////////////////////////////////////////////////////
    // get_pin: Get the specified pin of a given filter
    // Auguments:
    //   pFilter: Get the pin of the pFilter
    //   dir    : Which pin (input or output pin)
    //   n      : the nth pin of given direction
    //   ppPin  : To receive the pin we found
    // return value: error code
    ////////////////////////////////////////////////////////////////////////
    HRESULT CVideoGraphBase::get_pin(IBaseFilter * pFilter, PIN_DIRECTION dir, int n, IPin **p
pPin)
    {
        CComPtr< IEnumPins > pEnum;
        *ppPin = NULL;
        HRESULT hr;
        JIF(pFilter->EnumPins(&pEnum));

        ULONG ulFound;
        IPin *pPin;
        hr = E_FAIL;
        while(S_OK == pEnum->Next(1, &pPin, &ulFound))
        {
            PIN_DIRECTION pindir = (PIN_DIRECTION)3;
            pPin->QueryDirection(&pindir);
            if(pindir == dir) {
                if(n == 0) {
                    hr = S_OK, *ppPin = pPin;
                    break;
                }
                 --
            } // if
            pPin->Release();
        } // while

        return hr;
    }

    ////////////////////////////////////////////////////////////////////////
    // addGraphToRot: Register the graph for the GraphEdt.exe to view it
    // Auguments:
    //   pUnkGraph:   the graph to register
    //   pdwRegister: to retrieve the register number (to unregister later)
    // return value: error code
    ////////////////////////////////////////////////////////////////////////
    HRESULT CVideoGraphBase::addGraphToRot(IUnknown *pUnkGraph, DWORD *pdwRegister)
    {
        CComPtr<IMoniker> pMoniker;
        CComPtr<IRunningObjectTable> pROT;
        WCHAR wsz[128];
        HRESULT hr;

        JIF(GetRunningObjectTable(0, &pROT));

        wsprintfW(wsz, L"FilterGraph %08x pid %08x", (DWORD_PTR)pUnkGraph,
                  GetCurrentProcessId());

        hr = CreateItemMoniker(L"!", wsz, &pMoniker);
```

```
    if (SUCCEEDED(hr)) {
        hr = pROT->Register(0, pUnkGraph, pMoniker, pdwRegister);
    }
    return hr;
}

//////////////////////////////////////////////////////////////////////
// removeGraphFromRot: unregister the previous registered graph
// Auguments:
//   pdwRegister: specify which graph to unregister
//////////////////////////////////////////////////////////////////////
void CVideoGraphBase::removeGraphFromRot(DWORD dwRegister)
{
    CComPtr<IRunningObjectTable> pROT;
    if (SUCCEEDED(GetRunningObjectTable(0, &pROT))) {
        pROT->Revoke(dwRegister);
    }
}

//////////////////////////////////////////////////////////////////////
// errorHandler: Display a error message in Dialogbox according the
// given format in szFormat
//////////////////////////////////////////////////////////////////////
void CVideoGraphBase::errorHandler(TCHAR *szFormat, ...)
{
    // Get the aregument of the function
    TCHAR szBuffer[512];
    va_list pArgs;
    va_start(pArgs, szFormat);
    _stprintf(szBuffer, szFormat, pArgs);
    va_end(pArgs);
    // Display the error message
    MessageBox(NULL, szBuffer, _T("DirectShow Error Message"), MB_OK | MB_ICONERROR);
}

//////////////////////////////////////////////////////////////////////
// SetIsScaling: Perform scaling depending on the argument passed.
// Arguments:
//      scale:  true if it is to scale the frame.
//              false else.
//////////////////////////////////////////////////////////////////////
void CVideoGraphBase::SetIsScaling(BOOL scale)
{
    m_fIsScaling = scale;
}

//////////////////////////////////////////////////////////////////////
// SetFrameWidth:   Sets the frame width (capturing frame, not
// the window displayed.
// Arguments:
//      w:      New value of width.
//////////////////////////////////////////////////////////////////////
void CVideoGraphBase::SetFrameWidth(int w)
{
    this->m_iSizeX = w;
}

//////////////////////////////////////////////////////////////////////
// GetFrameWidth:   Gets frame width
// Return value:    Frames width.
//////////////////////////////////////////////////////////////////////
int CVideoGraphBase::GetFrameWidth()
{
    return this->m_iSizeX;
}

//////////////////////////////////////////////////////////////////////
// SetFrameRate:    Sets desired frame rate.
//////////////////////////////////////////////////////////////////////
void CVideoGraphBase::SetFrameRate(float rate)
{
    this->m_fAvgFrmRate = rate;
}
```

```
//////////////////////////////////////////////////////////////////////////
// GetFrameRate:    Gets desired frame rate. Not the actual frame rate.
// To get actual frame rate, use GetAvgFrameRate()
//////////////////////////////////////////////////////////////////////////
float CVideoGraphBase::GetFrameRate()
{
    return this->m_fAvgFrmRate;
}
```