Visão Artificial Para a Indústria

Manual de Desenvolvimento de Operações

Luis Fonseca Carvalho de Matos

(luis.matos@ua.pt)

Julho de 2007

Índice de conteúdos

1. Advertência

Este manual é um guia para expansão de funcionalidades da aplicação VAPI – Visão Artificial Para a Indústria.

Por adição de funcionalidades subentende-se o desenvolvimento de novas operações e a sua inclusão na aplicação.

O leitor deste manual deve estar familiarizado com a linguagem C, estruturas de dados e funções.

2. Introdução

Uma operação é um procedimento elementar de qualquer tarefa no panorama industrial e, sobretudo, no campo da Visão Artificial. Poderá ser uma função para processamento de imagem, um acto de comunicação, ou simplesmente uma operação matemática.

O procedimento de Desenvolvimento de Operação implica a criação de 2 elementos:

- A operação.
- O registo na Aplicação.

Após o registo na aplicação, a operação é adicionada à Lista de Operações, onde se encontram todas as operações disponíveis na aplicação. É desta lista que o utilizador selecciona as operações que utiliza.

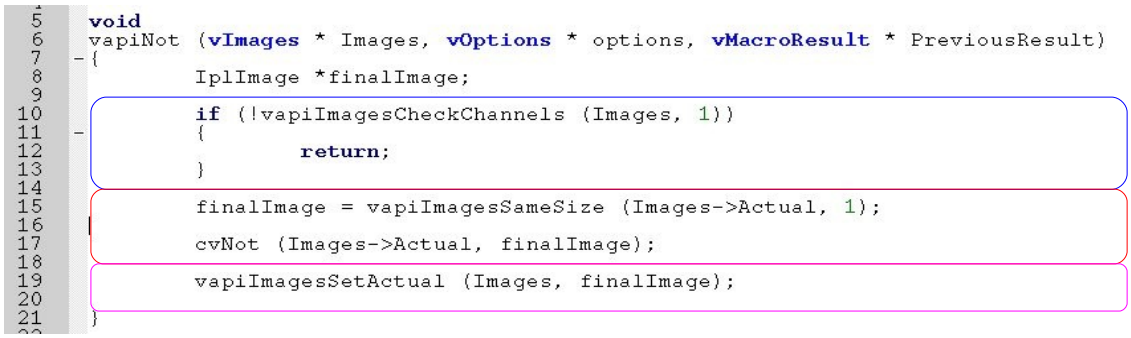
A nível de programação, uma operação é uma função em *C*. A sua simplicidade permite a mais variada aplicação.

```
 5  void
 6  vapiNot (vImages * Images, vOptions * options, vMacroResult * PreviousResult)
 7  {
 8          IplImage *finalImage;
 9
10          if (!vapiImagesCheckChannels (Images, 1))
11          {
12                  return;
13          }
14
15          finalImage = vapiImagesSameSize (Images->Actual, 1);
16
17          cvNot (Images->Actual, finalImage);
18
19          vapiImagesSetActual (Images, finalImage);
20
21  }
```

Figura 2.1: Exemplo de uma operação

3. Estrutura

N figura 3.1 encontra-se a operação de “Negação”, subdividida:



- Verificações
- Algoritmo
- Definição de Outputs e Resultados

Figura 3.1: Exemplo de uma operação comentada

Uma operação divide-se em 3 grupos:

- Verificações: Utilização de funções integradas para verificações e acesso a dados de aplicação.
- Algoritmo: desenvolvido pelo programador.
- Definição de output e resultados.

Esta estrutura possibilita ao programador focar-se no desenvolvimento do algoritmo, devido à simplificação da integração com a restante aplicação.

4. Inputs/Outputs

Tomando ainda como base a operação de Negação, podemos verificar o seu protótipo:

vapi*Not (vImages * Images, vOptions * Options, vResult * PreviousResult)*

Figura 4.1: Prototipo de Uma operação

Existem 3 estruturas de Dados que são requeridas e que servem de entrada e saída de Dados:

- Imagens – Imagens que estão a ser utilizadas;
- Opções – As opções requeridas para a operação;
- Resultados – Estrutura de dados para armazenar resultados.

O esquema seguinte pode ajudar a compreender o fluxo de dados:

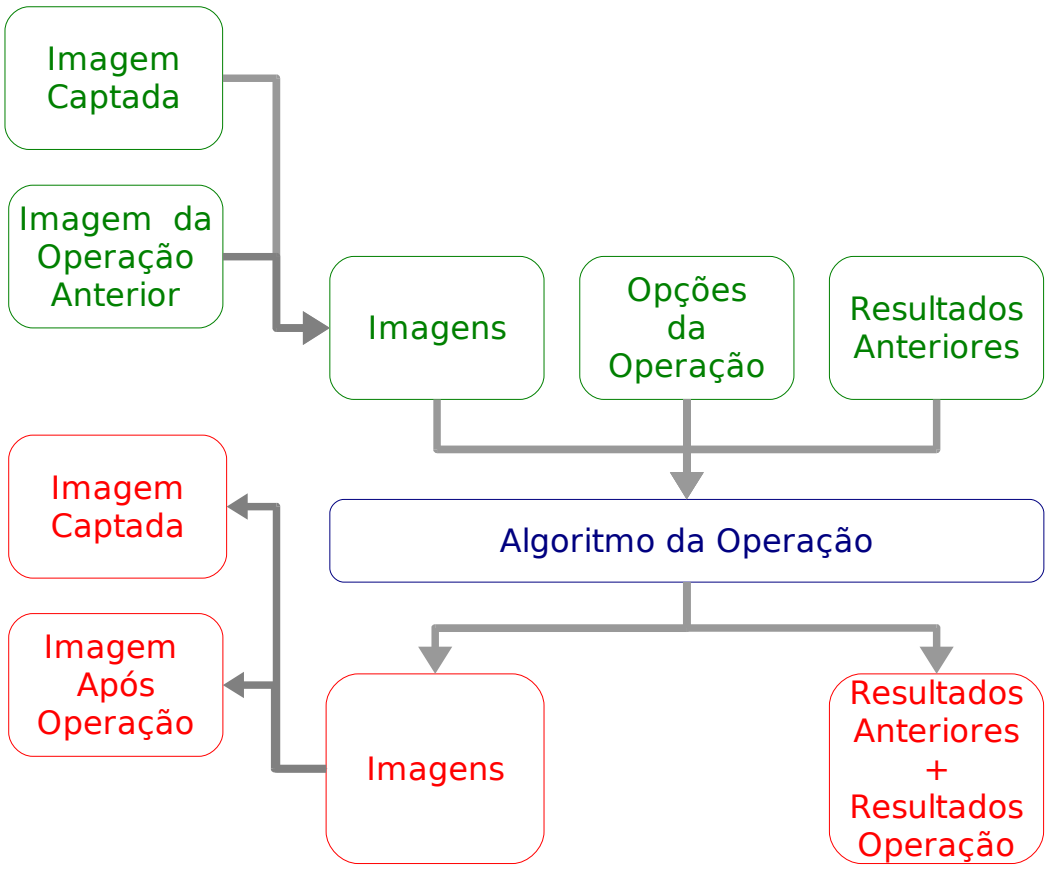


Figura 4.1: Esquema de inputs/outputs de uma operação

4.1. Imagens

A estrutura de imagens subdivide-se em 2 estruturas do tipo IPLimage. IPLimage é o tipo de dados em que o OpenCV guarda as imagens (ver Anexo I). Essas sub-estruturas são:

- I*mages->Original:* estrutura IPLimage que contém a imagem captada. Apenas deve ser alterada pela função *vapiGetSnapshot()* ;
- *Images->Actual*: estrutura IPLimage que contém a imagem Actual, resultado da última operação.

Existem algumas funções disponíveis para verificação e manipulação de imagem, bem como aquisição de nova imagem da origem. Estas funções são disponibilizadas com o objectivo de aumentar a fiabilidade e a liberdade para criar algoritmos.

Em caso algum a estrutura de imagens deverá ser alterada directamente. Deve-o ser sempre recorrendo às funções auxiliares.

Duas das funções mais importantes são:

- *vapiImageCheckChannel* – que verifica o número de canais da imagem Actual;
- *vapiImageCopy* – cria uma copia directa da imagem de input;
- *vapiImageSetActual* – que define uma nova imagem resultado.

Para detalhes destas e de outras funções de imagens, consultar Anexo II.

4.2. Opções

Esta estrutura de dados (vOptions) contém as opções preenchidas pelo utilizadores, requeridas pela operação. As opções contêm vários tipos de dados, nomeadamente:

- Vector Inteiros;
- Vector Decimais;
- Vector Frases;
- Vector Imagens.

O programador pode aceder a esta estrutura de dados utilizando as funções existentes para o efeito, por forma a tornar o acesso aos dados mais fiável.

Os dados são disponibilizados sob a forma de vectores. A ordenação dos vectores segue a ordem das questões. Se o programador definir 2 questões em que o utilizador deve colocar um número inteiro, a primeira terá a posição 0 e a segunda a posição 1, sendo acedidas desta forma:

vapiOptionsGetInts (Options, 0)

Figura 4.2: Exemplo para a obtenção do inteiro na posição 0

Esta estrutura não deve ser alterada. Apenas o deve ser pelo utilizador.

No Anexo III estão listadas as funções para acesso à estrutura de Opções.

4.3. Resultados

Resultados (vResult) é a estrutura de dados que faz a transferência da dados entre operações de uma tarefa. Se se pretende passar um ponto que se pretende assinalar, esse ponto é adicionado à estrutura de resultados.

Desta forma a operação seguinte terá acesso ao ponto a assinalar.

A estrutura de dados de resultados só deve ser acedida pelo grupo de funções que a ela está relacionada e nunca directamente, pois essas funções efectuam múltiplas verificações e expansões. A estrutura de resultados cria vectores para cada tipo de dados, tal como as opções.

Se for adicionado um ponto e não exista nenhum na estrutura de resultados, esse ponto terá o índice 0.

Para consultar as funções para os resultados, consulta o Anexo IV.

5. Definição de Operações

O desenvolvimento da operação em si é um passo. A sua integração no VAPI é outro. Para integrar a operação no Vapi é necessário adiciona-la ao programa, em runtime. Isso é efectuado utilizando a função *VapiLoadAllOptions()* no ficheiro VapiOptionsData.h.

Esta função chama outras funções criadas por programadores que registam as suas operações no Vapi (*registerOperations()*). Por sua vez, estas sub-funções são desenvolvidas para unicamente registarem as operações.

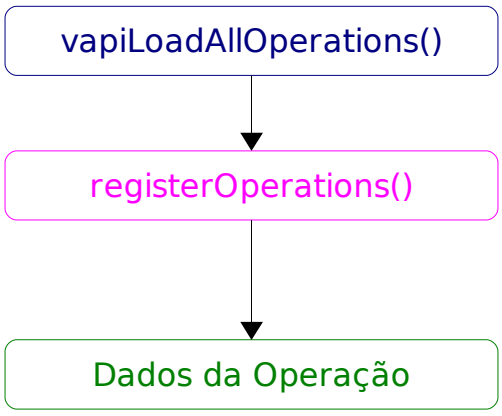


Figura 5.1
Esquema da cadeia de funções de registo de operações. Função de Registo global e função de registo específico (registerOperations), que contém os dados da operação. Esta última é desenvolvida em conjunto com a operação.

A função *registerOperations()* - nome de referência - divide-se em 3 áreas:

- Declaração – declaração do número de operações a adicionar.
- Definição – definir as novas operações: nome, descrições e opções.
- Adição – adições das operações à Lista de Operações disponíveis.

No início do ficheiro onde será criada a função de registo da operação, deverá existir uma declaração de Pré-Processamento com o número de operações.

```
#Define Operations 1
```

Figura 5.2: Definição do número de operações no início do ficheiro que contém a registerOperations()

Depois, a função deverá criar um vector de estruturas de operações:

```
vapiRegisterOperations()
{
        vOperation *operation_array[Operations];

        (...)
```

Figura 5.3: Exemplo do início da função registerOperations().

Para iniciar cada estrutura de operações (vOperation) é utilizada a função *vapiOperationInit()* que requer o nome da operação, o grupo e o ponteiro da função (ver Anexo V para funções para declarar operações).

Esta função devolve uma estrutura de operações preenchida com os valores por defeito. Podemos então definir questões ao utilizador, descrição breve e pormenorizada da operação. As funções que se encarregam desses pormenores estão definidas no *vapiOperations.h* (Anexo 5).

5.1. Registo de Operações

De modo a que as Operações sejam reconhecidas pelo VAPI, é necessário o seu registo na aplicação. Tal é efectuado em 2 passos:

- Declarar a função *registerOperations()* no ficheiro *ExtraOperations.h*, que se encontra dentro do directório ExtraOperations.
- Incluir a função de definição na função *vapiLoadAllOperations()*, ao ficheiro *vapiOperationsData.c*.

De salientar que a operação e a função de definição têm que estar num qualquer ficheiro .c (preferencialmente no directório ExtraOperations) e incluídos na compilação.

6. Execução

As operações são executadas sequencialmente quando dentro de uma tarefa (modo mais comum).

Podemos imaginar um ciclo contínuo de execução:

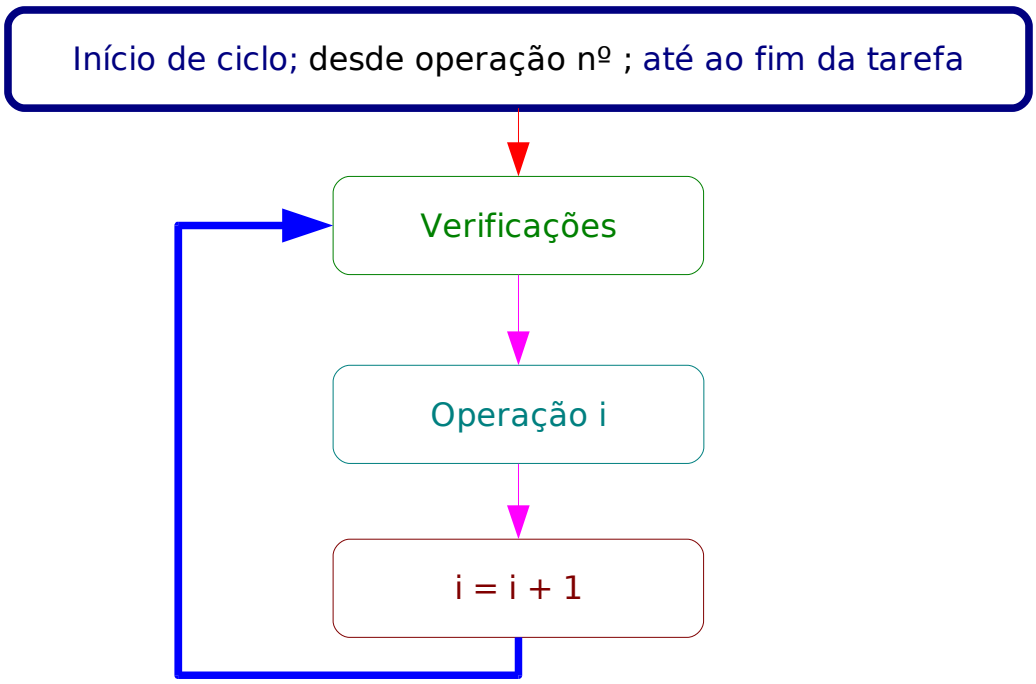


Figura 6.1: ciclo de execução de uma operação numa tarefa

Existem, no entanto, formas de manipular este ciclo utilizando funções de resultado (ver Anexo IV) tais como o *vapiResultSetEnd(),* que termina a execução da tarefa.

7. Exemplo

A operação a desenvolver terá como objectivo desenhar uma cruz num ponto pré-definido pelo utilizador. Chamemos à função da operação *vapiMarkPoint,* com o protótipo:

vapiMarkPoint *(vImages * Images, vOptions * Options, vResult * PreviousResult)*

Figura 7.1: Prototipo da função da operação Assinalar Ponto

Requer-se então que o utilizador forneça a coordenada ***x*** e ***y*** do ponto. Serão 2 inteiros e por essa ordem – 1º o ***x*** e depois o ***y***. Na estrutura de Opções, O ***x*** é o inteiro da posição 0 do vector de inteiros das opções e o ***y*** é a posição 1.

O objectivo é marcar o ponto na imagem original, pelo que temos então que ter a "Imagem Original + cruz" como saída da operação.

Recorrendo à função auxiliar *vapiDrawCross* (ver Anexo II):

*void vapiImagesDrawCross (IplImage * src, int line, int column, int lenght, int thick, int *color)*

Figura 7.2: Prototipo da função vapiDrawCross

Além do ponto, esta função necessita que sejam definidas as 3 componentes de cor RGB, o tamanho da cruz e a sua espessura. Deve-se requerer tudo isto ao utilizador.

Em resumo, o vector de inteiros da estrutura de opções fica definido na tabela 7.1:

Tabela 7.1 – planificação do Vector de Inteiros da estrutura de opções

Índice Vector	Descrição da Opção
0	coordenada x do ponto (0 … tamanho da imagem)
1	coordenada y do ponto (0 … tamanho da imagem)
2	componente de vermelho da cruz (0.255)
3	componente de verde da cruz (0.255)
4	componente de azul da cruz
5	tamanho da cruz (em pixeis)
6	espessura da linha da cruz (em pixeis)

Pode-se então esquematizar a operação:

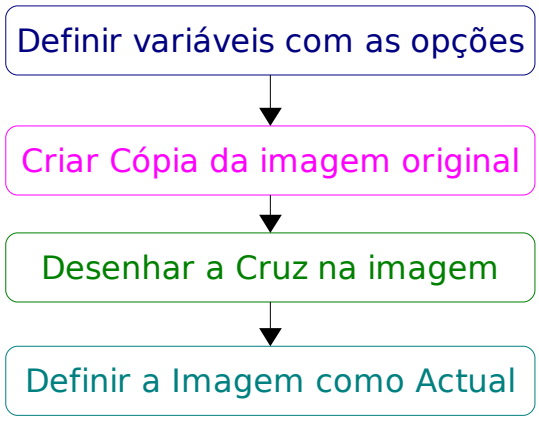


Figura 7.3: Planeamento da Operação

Em termos de código:

```
void
vapiMarkPoint(vImages * Images, vOptions * Options, vResult * PreviousResult)
{
	vPoint ponto;
	int RGB[3], crossSize, thickness;
	IPLImage * newImage;
	// Definir as opções em modos mais perceptíveis.
	// Ponto a Marcar
	ponto.x = vapiOptionsGetInt(Options, 0);
	ponto.y = vapiOptionsGetInt(Options, 1);

	//Definição das Cores da Cruz
	RGB[0] = vapiOptionsGetInt(Options, 2);
	RGB[1] = vapiOptionsGetInt(Options, 3);
	RGB[2] = vapiOptionsGetInt(Options, 4);

	//Definição do tamanho da Cruz
	crossSize = vapiOptionsGetInt(Options, 5);

	//Definição da espessura da Cruz
	thickness = vapiOptionsGetInt(Options, 6);

	//Copiar a Imagem Original para uma nova imagem
	newImage = vapiImagesCopy(Images->Original);

	//Desenhar a cruz
	vapiImagesDrawCross (newImage, ponto.y, ponto.x, crossSize,thickness,
&RGB);

	//Definir a nova imagem como actual
	vapiImagesSetActual(Images, newImage);
}
```

Figura 7.4: Função da Operação

De forma a definir a operação na aplicação, criar-se-á a função RegisterMark().

Existe a necessidade de definir

- nome da Operação : Marcar Ponto.
- nome do Grupo: Operações Auxiliares.
- descrição breve: Assinala com uma cruz o ponto definido pelo utilizador.
- descrição longa: Assinala o ponto definido pelo utilizador com uma cruz. É também possível definir a cor, tamanho e espessura da cruz.
- opção Inteira 0: Coordenada x do ponto.
- opção Inteira 1: Coordenada y do ponto.
- opção Inteira 2: Quantidade de vermelho.
- opção Inteira 3: Quantidade de verde.
- opção Inteira 4: Quantidade de azul.
- opção Inteira 5: Tamanho da cruz.
- opção Inteira 6: Espessura da cruz.

O código da função *RegisterMark()* será:

```
#define Operations 1

void
RegisterMark()
{
        // Definir variáveis
        vOperation * operation_array[Operations];

        //Criar a Operação
        operation_array[0] = vapiOperationInit ("Marcar Ponto", "Operações Auxiliares",
                                        &vapiMarkPoint);
        /***************************
         *          Definir Opções       *
         ***************************/
        // As opções são adicionadas segundo a ordem estabelecida:
        //Coordenada X
        vapiOperationSettingsSetInt (operation_array[0], "Coordenada X",
                        "coordenada X do Ponto a Marcar", 0, 2000, 0, 1, vSpin);
        //Coordenada Y
        vapiOperationSettingsSetInt (operation_array[0], "Coordenada Y",
                        "coordenada Y do Ponto a Marcar", 0, 2000, 0, 1, vSpin);
        /***************
         * Definir RGB *
         ***************/
        // Por defeito, R=255, G e B = 0, o que faz a cruz ser, por defeito, vermelha.

        //Quantidade de Vermelho
        vapiOperationSettingsSetInt (operation_array[0], "Quantidade de vermelho",
                        "Definição da Cor da Cruz, em RGB. Defina a Quantidade de Vermelho",
                                0, 255, 255, 1, vSpin);
```

```
        //Quantidade de Verde
        vapiOperationSettingsSetInt (operation_array[0], "Quantidade de verde",
                        "Definição da Cor da Cruz, em RGB. Defina a Quantidade de Verde",
                                0, 255, 0, 1, vSpin);

        //Quantidade de Azul
        vapiOperationSettingsSetInt (operation_array[0], "Quantidade de Azul",
                        "Definição da Cor da Cruz, em RGB. Defina a Quantidade de Azul",
                                0, 255, 0, 1, vSpin);

        /****************************************
         * Definir o tamanho e espessura da Cruz *
         ****************************************/
        //Tamanho da Cruz
        vapiOperationSettingsSetInt (operation_array[0], "Tamanho da Cruz (em pixeis)",
                        "Definir o tamanho da cruz, em pixeis, de um extremo ao outro.",
                                0, 2000, 0, 1, vSpin);
        //Espessura da linha
        vapiOperationSettingsSetInt (operation_array[0], "Espessura da Cruz (em pixeis)",
                        "Definir a espessura da linha da cruz, em pixeis.",
                                0, 2000, 0, 1, vSpin);
        /*********************************************************************
         * Adicionar o array de operações à lista de Operações Disponíveis *
         *********************************************************************/

        vapiOperationListAddOperation (operation_array, Operations);
        /* FIM */
}
```

Figura 6.5: Função de Registo da Operação

Por último, adicionar o protótipo da função no ficheiro ExtraOperations / ExtraOperations.h e adicionar ficheiro.c a este directório. Dentro desse ficheiro deve estar a função RegisterMark e a operação.

Adicionar então a função *RegisterMark()* à função *vapiLoadAllOperations()* no ficheiro *vapiOperationsData.c*.

A N E X O S

Anexo I – Estrutura IPLImage

```
typedef struct _IplImage
    {
        int  nSize;         /* sizeof(IplImage) */
        int  ID;            /* version (=0)*/
        int  nChannels;     /* Most of OpenCV functions support 1,2,3 or 4 channels */
        int  alphaChannel;  /* ignored by OpenCV */
        int  depth;         /* pixel depth in bits: IPL_DEPTH_8U, IPL_DEPTH_8S, IPL_DEPTH_16U,
                            IPL_DEPTH_16S, IPL_DEPTH_32S, IPL_DEPTH_32F and IPL_DEPTH_64F are
supported */
        char colorModel[4]; /* ignored by OpenCV */
        char channelSeq[4]; /* ditto */
        int  dataOrder;     /* 0 - interleaved color channels, 1 - separate color channels.
                            cvCreateImage can only create interleaved images */
        int  origin;        /* 0 - top-left origin,
                            1 - bottom-left origin (Windows bitmaps style) */
        int  align;         /* Alignment of image rows (4 or 8).
                            OpenCV ignores it and uses widthStep instead */
        int  width;         /* image width in pixels */
        int  height;        /* image height in pixels */
        struct _IplROI *roi;/* image ROI. when it is not NULL, this specifies image region to process */
        struct _IplImage *maskROI; /* must be NULL in OpenCV */
        void  *imageId;     /* ditto */
        struct _IplTileInfo *tileInfo; /* ditto */
        int  imageSize;     /* image data size in bytes
                            (=image->height*image->widthStep
                            in case of interleaved data)*/
        char *imageData;  /* pointer to aligned image data */
        int  widthStep;   /* size of aligned image row in bytes */
        int  BorderMode[4]; /* border completion mode, ignored by OpenCV */
        int  BorderConst[4]; /* ditto */
        char *imageDataOrigin; /* pointer to a very origin of image data
                               (not necessarily aligned) -
                               it is needed for correct image deallocation */
    }
    IplImage;
```

Anexo II - Funções Auxiliares para manipulação de Imagens

- vImages *
  vapiImagesAuxCloneImages (vImages *src)
- int
  vapiImagesCheckChannels (vImages *Images, int nChannels)
- IplImage *
  vapiImagesCopy (IplImage *src)
- void
  vapiImagesDrawCross (IplImage *src, int line, int column, int lenght, int thick, int *color)
  Função Para desenhar cruzes, dado o centro e o tamanho.
- IplImage *
  vapiImagesEnlarge (IplImage *src, int enlarge)
  Cria uma imagem vazia com um tamanho maior ou menor que original.
- void
  vapiImagesPrint (IplImage *src, double level)
  Esta função mostra na linha de comandos os valores existentes na Matriz src que forem maiores que level.
- IplImage *
  vapiImagesSameSize (IplImage *src, int n_channels)
  Cria uma cópia da imagem com a mesma dimensão (nÃºmero de pixels) mas com número diferente de canais.
- IplImage *
  vapiImagesSameSize16 (IplImage *src, int n_channels)
  Cria uma cópia da imagem com a mesma dimensão (número de pixeis) mas com número diferente de canais e 16 bits.
- void
  vapiImagesShow (const char *title, IplImage *image)
- void
  vapiOriginal2Actual (vImages *Images)
  Copia a imagem Original para a Actual.

Anexo III - Funções Para aceder às opções

Enumerações

- enum

vapiInterfaceType { vBar = 0, vSpin }

Types of interfaces to Options.

Funções

- vBoolean

vapiOptionsGetBooleans (vOptions *Options, int position)
Retrieves the boolean option from the position in the Array of Booleans.

- char *

vapiOptionsGetChars (vOptions *Options, int position)
Retrieves the string from the position in the Array of Strings.

- int

vapiOptionsGetChoices (vOptions *Options, int position)
Retrieves the number of the choice of the list of choices in the position required in the choices' list array.

- double

vapiOptionsGetFloats (vOptions *Options, int position)
Retrieves the float from the position in the Array of floats.

- int

vapiOptionsGetInts (vOptions *Options, int position)
Retrieves the integer from the position in the Array of integers.

- int

vapiOptionsGetTimes (vOptions *Options)
Retrieves the number of repetitions.

Anexo IV - Funções para aceder aos Resultados

- void
vapiResultAddChar (vMacroResult *PreviousResult, const char *string)
- void
vapiResultAddFloat (vMacroResult *PreviousResult, double value)
- void
apiResultAddInt (vMacroResult *PreviousResult, int value)
- void
apiResultAddPoint (vMacroResult *PreviousResult, int x, int y)
- char *
vapiResultGetChar (vMacroResult *PreviousResult, int position)
- int
vapiResultGetCharsNumber (vMacroResult *PreviousResult)
- vBoolean
vapiResultGetDecision (vMacroResult *PreviousResult)
- double
vapiResultGetFloat (vMacroResult *PreviousResult, int position)
- int
vapiResultGetFloatsNumber (vMacroResult *PreviousResult)
- int
vapiResultGetInt (vMacroResult *PreviousResult, int position)
- int
vapiResultGetIntsNumber (vMacroResult *PreviousResult)
- int
vapiResultGetJumpTo (vMacroResult *PreviousResult)
- vPoint
vapiResultGetPoint (vMacroResult *PreviousResult, int position)
- int
vapiResultGetPointsNumber (vMacroResult *PreviousResult)
- char *
vapiResultPrintf (const char *inString, vMacroResult *PreviousResult)
Detecta variaveis nas strings e substitui-as por variaveis internas do vapi. Utiliza a mesma sintaxe do printf, mas procura variÃ¡veis em memÃ³ria. Por exemplo %rd0 indica o valor de Resultado (d) inteiro de indice 0.
- void
vapiResultSetDecision (vMacroResult *PreviousResult, vBoolean Decision)

- void
vapiResultSetEnd (vMacroResult *PreviousResult)
- void
vapiResultSetJumpTo (vMacroResult *PreviousResult, int OperationOrder)
- void
vResultReset (vMacroResult *PreviousResult)
- void
vResultResetChars (vMacroResult *PreviousResult)
- void
vResultResetInts (vMacroResult *PreviousResult)
- void
vResultResetPoints (vMacroResult *PreviousResult)

Anexo V - Funções de Operações

vEffect * vapiOperationInit (const char *name, const char *Group, vOperationFunction func)

Initiates an Operation.

void vapiOperationSettingsSetBoolean (vEffect *Operation, const char *name, const char *description, vBoolean defaultValue)

Adds a boolean option.

void vapiOperationSettingsSetChar (vEffect *Operation, const char *name, const char *description)

Adds a character or string option.

void vapiOperationSettingsSetChoose (vEffect *Operation, const char *name, const char *description, int numberOfChoices, int defaultOption,...)

Adds a "option list" option.

void vapiOperationSettingsSetFloat (vEffect *Operation, const char *name, const char *description, double min, double max, double defaultValue, double scale, vInterfaceType interfaceType)

Adds a float option.

void vapiOperationSettingsSetHasOriginalOption (vEffect *Operation)

Set if the operation can ask the user for the use of the original image, or not.

void vapiOperationSettingsSetInt (vEffect *Operation, const char *name, const char *description, int min, int max, int defaultValue, int scale, vInterfaceType interfaceType)

Adds an integer option.

void vapiOperationSettingsSetMacros (vEffect *Operation, int MacrosNumber)

Sets the use of Macros, so that space can be spared for them.

void vapiOperationSettingsSetNeedOfAuxImage (vEffect *Operation, int NeedOfAuxImage)

Sets the need of a Template space.

void vapiOperationSettingsSetOperationBriefDescription (vEffect *Operation, const char *briefDescription)

Define the operation's brief Description.

void vapiOperationSettingsSetOperationLongDescription (vEffect *Operation, const char *longDescription)

Define the operation's long Description.

void vapiOperationSettingsSetRepeatInside (vEffect *Operation)

Sets if the operation handles the repetitions.

void vapiOperationSettingsSetRunSeveralTimes (vEffect *Operation)

Sets if the Operation Can run several times (repetitions).
