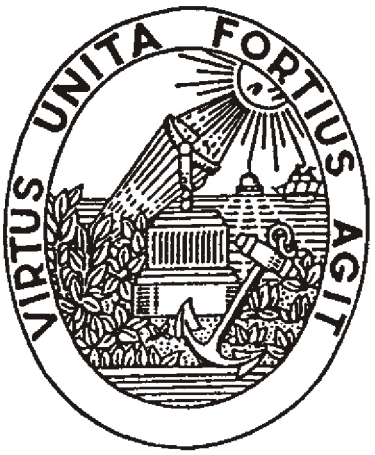
João Manuel Ribeiro da Silva Tavares

Algumas Ferramentas para Visão Tridimensional por Computador



Faculdade de Engenharia da Universidade do Porto

Junho de 1995

Algumas Ferramentas para Visão Tridimensional por Computador

Publicação inserida na dissertação:

Obtenção de Estrutura Tridimensional a Partir de Movimento de Câmara

Dissertação submetida ao Departamento de Engenharia Electrotécnica e de Computadores para satisfação parcial dos requisitos do:

Mestrado em Engenharia Electrotécnica e de Computadores

Perfil de Informática Industrial

Por:

João Manuel Ribeiro da Silva Tavares

Licenciado em Engenharia Mecânica pela Faculdade de Engenharia da Universidade do Porto (1992)

Orientador:

A. Jorge Padilha

Prof. Auxiliar do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

J. Tavares *Algumas Ferramentas para Visão Tridimensional por Computador* **1995**

Agradecimentos

Ao Prof. A. Jorge Padilha por todo o apoio prestado ao longo desta dissertação; nomeadamente, pela sua orientação e total disponibilidade sempre demonstrada.

Ao Eng. Jorge Alves por todo o apoio prestado; em particular, nos domínios de Calibração de Câmaras e Extracção de Coordenadas 3D.

Ao Eng. Miguel Velhote por todo o apoio prestado; em particular, pela disponibilidade de utilização de funções em linguagem C por si desenvolvidas.

À Junta Nacional de Investigação Científica e Tecnológica pela Bolsa de Mestrado atribuída ao abrigo do Programa Ciência com a ref. BM / 3258 / 92 - RM com a qual foi possível frequentar este Curso de Mestrado a tempo inteiro.

Ao Instituto de Engenharia Biomédica pelos recursos utilizados ao longo desta dissertação.

A todos aqueles que tornaram esta publicação possível...

Sumário

Nesta publicação é apresentado um conjunto de ferramentas para utilização em visão tridimensional por computador. Assim, são apresentadas as várias implementações desenvolvidas ao longo desta dissertação para os domínios: transformações geométricas; calibração de câmaras; seguimento de linhas e consequente aproximação poligonal utilizando faixas dinâmicas; fusão de segmentos de recta próximos e de direcções similares; seguimento de segmentos de recta ao longo de uma dada sequência de imagens; obtenção de informação tridimensional.

As várias apresentações estão divididas em três fases; na primeira, são apresentados os vários módulos que constituem a implementação; na segunda, os respectivos fluxogramas dos mesmos; e, na terceira e última fase, é apresentada a listagem da implementação em linguagem C.

Índice

Introdução

Nesta publicação são apresentadas, na óptica do utilizador, algumas ferramentas para visão tridimensional por computador. Designadamente, as várias implementações desenvolvidas no âmbito da dissertação "Obtenção de estrutura tridimensional a partir de movimento de câmara", para satisfação parcial dos requisitos do Mestrado em Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto. Trata-se, portanto, de um manual para utilização das implementações desenvolvidas no domínio da visão tridimensional por computador, não contendo assim uma apresentação detalhada de todos os fundamentos teóricos nos quais se baseiam.

Optou-se por dividir esta publicação em cinco capítulos, consoante o domínio mais específico das respectivas implementações. Assim esta publicação é constituída pelos capítulos:

- ***Capítulo I*** ***Transformações Geométricas em 2D e em 3D***

Neste capítulo é apresentada uma implementação capaz de aplicar as diferentes transformações geométricas, utilizadas quer em visão por computador quer em computação gráfica, a pontos no espaço bidimensional e no espaço tridimensional; nomeadamente, as transformações geométricas:

- ✓ Rotação em torno do eixo *x*;
- ✓ Rotação em torno do eixo *y*;
- ✓ Rotação em torno do eixo *z*;
- ✓ Translação ao longo dos três eixos principais;
- ✓ Escalonamento segundo os três eixos principais;
- ✓ Distorção segundo os três eixos principais;
- ✓ Projecção perspectiva, quando o plano de projecção é anterior ao centro de projecção e quando lhe é posterior.

- ***Capítulo II*** ***Calibração de Câmaras para Utilização em Visão Tridimensional***

Neste capítulo são apresentadas cinco implementações com interesse no domínio de calibração de câmaras para utilização em visão tridimensional. Assim, são apresentadas as seguintes implementações:

- ✓ Calibração do factor de incerteza horizontal de uma câmara: ***findsx***.
- ✓ Calibração de todos os parâmetros intrínsecos e extrínsecos de uma câmara, exceptuando o factor de incerteza horizontal: ***calcamera***.
- ✓ Determinação das coordenadas imagem na memória *frame* dos pontos utilizados na calibração: ***calpoint***.
- ✓ Simulação de uma dada câmara: ***simcamera***.
- ✓ Formatação conveniente dos dados de entrada para a implementação ***calcamera***: ***input***.

- ***Capítulo III*** ***Seguimento de Linhas e Consequente Aproximação Poligonal Utilizando Faixas Dinâmicas***

Neste capítulo é apresentada uma implementação para seguimento de linhas, baseado na direcção e em dois níveis de amplitude do gradiente, e para a consequente aproximação poligonal utilizando faixas dinâmicas.

- ***Capítulo IV*** ***Fusão de Segmentos de Recta Próximos e de Direcções Similares***

Neste capítulo são apresentadas duas implementações com interesse no domínio da fusão de segmentos de recta que pertencem a uma mesma entidade. Assim, são apresentadas as seguintes implementações:

- ✓ Fusão de segmentos próximos e de direcções similares: ***linelink***.
- ✓ Desenho numa imagem de saída dos segmentos de recta definidos num ficheiro de entrada: ***showlines***.

- ***Capítulo V*** | ***Seguimento de Segmentos de Recta ao Longo de Sequências de Imagens e Obtenção de Coordenadas 3D***

Neste capítulo são apresentadas quatro implementações com interesse nos domínios do seguimento de segmentos de recta ao longo de sequências de imagens e da obtenção de coordenadas 3D. As quatro implementações apresentadas são:

- ✓ Determinação da área visível por uma dada câmara sobre um plano de coordenadas especificadas pelo utilizador: ***simspace***.
- ✓ Simulador de uma dada câmara móvel para segmentos de recta no espaço tridimensional cujas coordenadas 3D dos pontos iniciais e finais são especificadas pelo utilizador: ***simoca***.
- ✓ Seguimento de segmentos de recta e posterior obtenção de coordenadas 3D dos pontos inicial e final de cada entidade emparelhada em duas imagens da sequência respectiva, quando o movimento da câmara é conhecido: ***deep***.
- ✓ Obtenção de coordenadas 3D de pontos cujas coordenadas 2D em duas imagens, obtidas por uma mesma câmara para duas posições e orientações distintas, são especificadas pelo utilizador: ***compdeep***.

Todas as implementações apresentadas neste capítulo foram experimentadas quer ao nível de simulação quer ao nível real, exceptuando a implementação ***findsx***, cujos requisitos de ordem prática não puderam ser realizados com as condições existentes. Em todas as implementações experimentadas, os valores dos respectivos argumentos opcionais foram escolhidos de forma a se obterem, em termos médios, resultados de boa qualidade.

As diferentes apresentações estão divididas em três fases; na primeira, são descritos os diferentes módulos que constituem a implementação; na segunda, são apresentados os respectivos fluxogramas dos mesmos; e, na terceira e última fase, é apresentada a listagem da implementação em linguagem C.

Na *Fig. 1*, é apresentada a simbologia que é utilizada nos diversos fluxogramas.

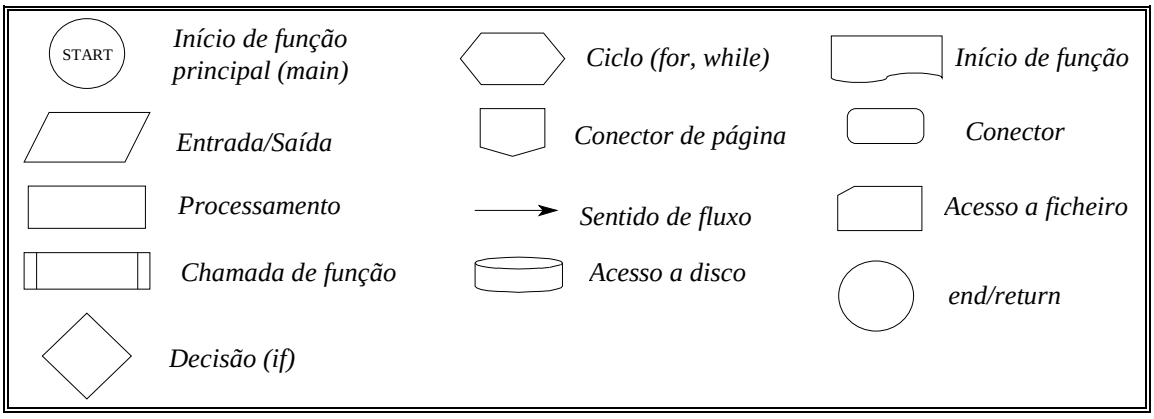


Fig. 1 - Simbologia utilizada nos fluxogramas apresentados nesta publicação.

Capítulo I

Transformações Geométricas em 2D e em 3D

Introdução

Neste capítulo é apresentada uma implementação capaz de aplicar as diferentes transformações geométricas, utilizadas quer em visão por computador quer em computação gráfica, a pontos no espaço bidimensional e no espaço tridimensional; nomeadamente, as transformações geométricas:

- ✓ Rotação em torno do eixo *x*;
- ✓ Rotação em torno do eixo *y*;
- ✓ Rotação em torno do eixo *z*;
- ✓ Translação ao longo dos três eixos principais;
- ✓ Escalonamento segundo os três eixos principais;
- ✓ Distorção segundo os três eixos principais;
- ✓ Projecção perspectiva, quando o plano de projecção é anterior ao centro de projecção e quando lhe é posterior.

Para o estudo das transformações geométricas recomenda-se [Foley 1991], [Schalkoff, 1989], [Hall , 1993] e [Tavares, 1995].

1 - Descrição dos diferentes módulos

A implementação é constituída pelos módulos: ***principal***, ***rotationx***, ***rotationy***, ***rotationz***, ***translation***, ***scale***, ***distortion***, ***perspective1***, ***perspective2*** e ***multmatvec***, *Fig. 1*. Apresenta-se de seguida a descrição de cada um destes módulos.

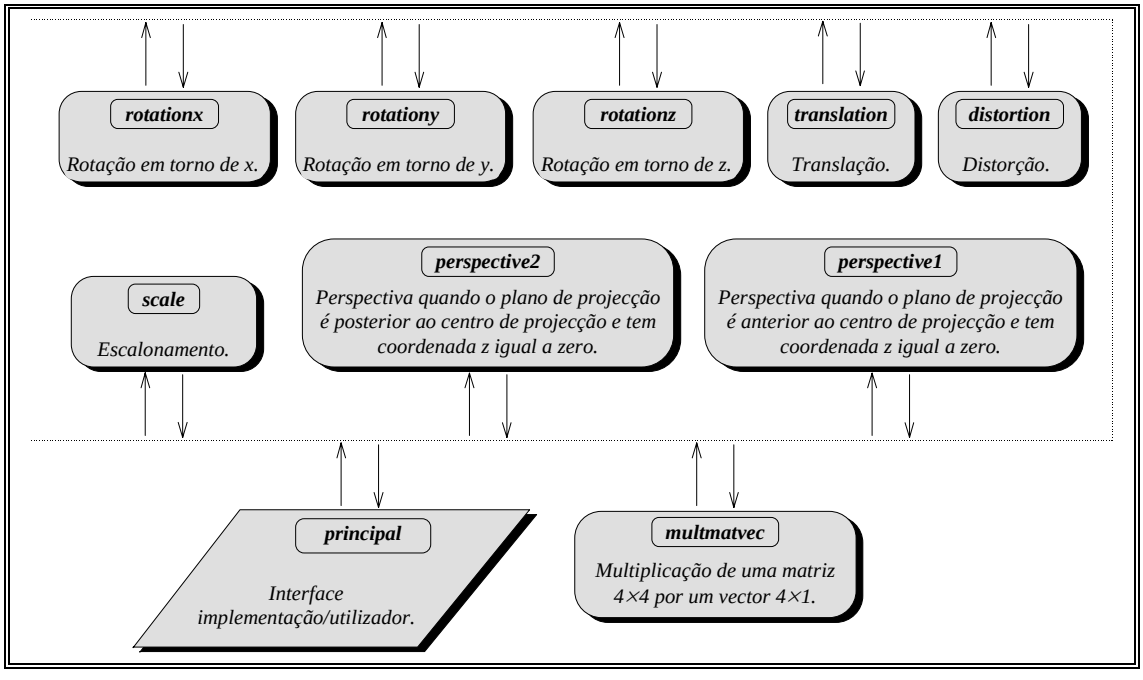


Fig. 1 - Módulos integrantes desta implementação e suas relações.

1.1 - Módulo *principal*

Este módulo é responsável pela interface implementação/utilizador. A seu cargo está:

- ✓ Leitura dos argumentos opcionais ("*switches*"):

- *xpoint*, ("[-px <xpoint>]"), coordenada *x* do ponto de entrada. Por defeito, o seu valor é *0.0*.
- *ypoint*, ("[-py <ypoint>]"), coordenada *y* do ponto de entrada. Por defeito, o seu valor é *0.0*.
- *zpoint*, ("[-pz <zpoint>]"), coordenada *z* do ponto de entrada. Por defeito, o seu valor é *0.0*.
- *parameter1*, ("[-p1 <parameter1>]"), parâmetro *1* para a transformação geométrica, ver *Tab. I*. Por defeito, o seu valor é *0.0*.
- *parameter2*, ("[-p2 <parameter2>]"), parâmetro *2* para a transformação geométrica, ver *Tab. I*. Por defeito, o seu valor é *0.0*,
- *parameter3* ("[-p3 <parameter3>]"), parâmetro *3* para a transformação geométrica, ver *Tab. I*. Por defeito, o seu valor é *0.0*.
- *parameter4*, ("[-p4 <parameter4>]"), parâmetro *4* para a transformação geométrica, ver *Tab. I*. Por defeito, o seu valor é *0.0*.
- *parameter5*, ("[-p5 <parameter5>]"), parâmetro *5* para a transformação geométrica, ver *Tab. I*. Por defeito, o seu valor é *0.0*.
- *parameter6*, ("[-p6 <parameter6>]"), parâmetro *6* para a transformação geométrica, ver *Tab. I*. Por defeito, o seu valor é *0.0*.
- *parameter7*, ("[-p7 <parameter7>]"), parâmetro *7* para a transformação geométrica, ver *Tab. I*. Por defeito, o seu valor é *0.0*.
- *parameter8*, ("[-p8 <parameter8>]"), parâmetro *8* para a transformação geométrica, ver *Tab. I*. Por defeito, o seu valor é *0.0*.
- *parameter9*, ("[-p9 <parameter9>]"), parâmetro *9* para a transformação geométrica, ver *Tab. I*. Por defeito, o seu valor é *0.0*.
- *filename*, ("[-f <filename>]"), nome do ficheiro para escrita dos resultados obtidos pela implementação. Por defeito é "*NULL*", o que implica a não abertura do ficheiro. Quando especificado, o ficheiro, se ainda não existir, é criado; caso contrário, os resultados são acrescentados ao ficheiro já existente.

✓ Leitura do argumento passado à implementação pela linha de comando: *transformation*, ("<transformation>"), nome da transformação geométrica desejada, deverá ser igual a:

- *rotationx*, para rotação em torno do eixo *x*;
- *rotationy*, para rotação em torno do eixo *y*;
- *rotationz*, para rotação em torno do eixo *z*;
- *translation*, para translação;
- *scale*, para escalonamento;
- *distortion*, para distorção;
- *perspective*, para projecção perspectiva.

✓ Comunicação da existência de erros em argumentos de entrada ou ao longo do decurso da execução:

	rotationx	**rotationy**	**rotationz**	**translation**	**scale**	**distortion**	**perspective**
<parameter1>	θ	θ	θ	d_x	s_x	d_1	d
<parameter2>				d_y	s_y	d_2	
<parameter3>				d_z	s_z	d_3	
<parameter4>						d_4	
<parameter5>						d_5	
<parameter6>						d_6	
<parameter7>						d_7	
<parameter8>						d_8	
<parameter9>						d_9	

Onde:

θ é o ângulo de rotação em radiano, d_x é a translação ao longo do eixo *x*, d_y é a translação ao longo do eixo *y*, d_z é a translação ao longo do eixo *z*, s_x é o escalonamento segundo o eixo *x*, s_y é o escalonamento segundo o eixo *y*, s_z é o escalonamento segundo o eixo *z*, d_1, ..., d_9 são os elementos da matriz de distorção e *d* é a distância do plano de projecção perspectiva ao centro de projecção.

Tab. I - Correspondência entre <transformation> e <parameter(i)> com $i = 1, \ldots, 9$.

- Através de valores de retorno:
  - 1, ("Value of d (<parameter1>) must not be zero."), o valor especificado para *parameter1* é igual a zero, o que não é possível para a transformação projecção perspectiva.
  - 2, ("Value of z/d (<zpoint>/<parameter1>) must be different from -1."), os valores especificados para *zpoint* e para *parameter1* não são correctos, porque para a transformação projecção perspectiva *zpoint/parameter1* deverá ser diferente de *-1*.
  - 3, ("Wrong name for <transformation>."), o nome especificado para a transformação geométrica não é correcto.
  - 4, ("Can´t open file <filename>."), o módulo não consegue abrir o ficheiro para escrita de resultados obtidos pela implementação.
- Através de mensagens:
  - 1, (*InitMessage*), quando o número de argumentos é igual a um, isto é, apenas o nome da implementação.
  - 2, ("Bad number of arguments."), quando o número de argumentos de especificação necessária é diferente de dois, isto é, o nome da implementação e a transformação geométrica desejada através do argumento <*transformation*>.

✓ Chamada do módulo para execução da transformação geométrica desejada, com a passagem dos parâmetros devidos.

✓ Abertura, se pretendido pelo utilizador, do ficheiro para escrita dos resultados obtidos pela implementação, com a descrição de como estes foram obtidos.

✓ Apresentação ao utilizador dos resultados obtidos pela implementação, com a descrição de como estes foram obtidos.

Como restrições na utilização deste módulo têm-se: quando a transformação geométrica desejada é projecção perspectiva, o valor de *d* (<*parameter1*>) deverá ser diferente de zero; assim como, *z/d* (<*zpoint*>/<*parameter1*>) deverá ser diferente de *-1*.

1.2 - Módulo *rotationx*

Este módulo é responsável pela transformação geométrica rotação em torno do eixo *x*.

Como parâmetros de entrada para este módulo têm-se:

- *teta*, do tipo real, ângulo de rotação em radiano.
- *b[ ]*, do tipo real, coordenadas do ponto homogéneo original.
- *c[ ]*, do tipo real, coordenadas do ponto homogéneo transformado.

No início da sua tarefa, este módulo inicializa a matriz de transformação geométrica respectiva; de seguida, utiliza o módulo ***multmatvec*** para a execução do produto da matriz transformação pelo ponto homogéneo original.

Este módulo retorna *0* como indicação que a sua execução decorreu com normalidade, e não apresenta nenhum tipo de restrições quanto à sua utilização.

1.3 - Módulo *rotationy*

Este módulo é responsável pela transformação geométrica rotação em torno do eixo *y*.

Como parâmetros de entrada para este módulo têm-se:

- *teta*, do tipo real, ângulo de rotação em radiano.
- *b[ ]*, do tipo real, coordenadas do ponto homogéneo original.
- *c[ ]*, do tipo real, coordenadas do ponto homogéneo transformado.

No início da sua tarefa, este módulo inicializa a matriz de transformação geométrica respectiva; de seguida, utiliza o módulo ***multmatvec*** para a execução do produto da matriz transformação pelo ponto homogéneo original.

Este módulo retorna *0* como indicação que a sua execução decorreu com normalidade, e não apresenta nenhum tipo de restrições quanto à sua utilização.

1.4 - Módulo *rotationz*

Este módulo é responsável pela transformação geométrica rotação em torno do eixo *z*.

Como parâmetros de entrada para este módulo têm-se:

- *teta*, do tipo real, ângulo de rotação em radiano.
- *b[ ]*, do tipo real, coordenadas do ponto homogéneo original.
- *c[ ]*, do tipo real, coordenadas do ponto homogéneo transformado.

No início da sua tarefa, este módulo inicializa a matriz de transformação geométrica respectiva; de seguida, utiliza o módulo ***multmatvec*** para a execução do produto da matriz transformação pelo ponto homogéneo original.

Este módulo retorna *0* como indicação que a sua execução decorreu com normalidade, e não apresenta nenhum tipo de restrições quanto à sua utilização.

1.5 - Módulo *translation*

Este módulo é responsável pela transformação geométrica translação ao longo dos três eixos principais.

Como parâmetros de entrada para este módulo têm-se:

- *dx*, do tipo real, translação segundo o eixo *x*.
- *dy*, do tipo real, translação segundo o eixo *y*.

- *dz*, do tipo real, translação segundo o eixo *z*.
- *b[ ]*, do tipo real, coordenadas do ponto homogéneo original.
- *c[ ]*, do tipo real, coordenadas do ponto homogéneo transformado.

No início da sua tarefa, este módulo inicializa a matriz de transformação geométrica respectiva; de seguida, utiliza o módulo ***multmatvec*** para a execução do produto da matriz transformação pelo ponto homogéneo original.

Este módulo retorna *0* como indicação que a sua execução decorreu com normalidade, e não apresenta nenhum tipo de restrições quanto à sua utilização.

1.6 - Módulo *scale*

Este módulo é responsável pela transformação geométrica escalonamento segundo os três eixos principais.

Como parâmetros de entrada para este módulo têm-se:

- *sx*, do tipo real, escalonamento segundo o eixo *x*.
- *sy*, do tipo real, escalonamento segundo o eixo *y*.
- *sz*, do tipo real, escalonamento segundo o eixo *z*.
- *b[ ]*, do tipo real, coordenadas do ponto homogéneo original.
- *c[ ]*, do tipo real, coordenadas do ponto homogéneo transformado.

No início da sua tarefa, este módulo inicializa a matriz de transformação geométrica respectiva; de seguida, utiliza o módulo ***multmatvec*** para a execução do produto da matriz transformação pelo ponto homogéneo original.

Este módulo retorna *0* como indicação que a sua execução decorreu com normalidade, e não apresenta nenhum tipo de restrições quanto à sua utilização.

1.7 - Módulo *distortion*

Este módulo é responsável pela transformação geométrica distorção segundo os três eixos principais.

Como parâmetros de entrada para este módulo têm-se:

- *d[ ]*, do tipo real, parâmetros de distorção: *d[0]* parâmetro de distorção d_{xx}, *d[1]* parâmetro de distorção d_{xy}, *d[2]* parâmetro de distorção d_{xz}, *d[3]* parâmetro de distorção d_{yx}, ..., *d[8]* parâmetro de distorção d_{zz}.
- *b[ ]*, do tipo real, coordenadas do ponto homogéneo original.
- *c[ ]*, do tipo real, coordenadas do ponto homogéneo transformado.

No início da sua tarefa, este módulo inicializa a matriz de transformação geométrica respectiva; de seguida, utiliza o módulo ***multmatvec*** para a execução do produto da matriz transformação pelo ponto homogéneo original.

Este módulo retorna *0* como indicação que a sua execução decorreu com normalidade, e não apresenta nenhum tipo de restrições quanto à sua utilização.

1.8 - Módulo *perspective1*

Este módulo é responsável pela transformação geométrica perspectiva quando o plano de projecção é anterior ao centro de projecção e tem coordenada *z* igual a zero.

Como parâmetros de entrada para este módulo têm-se:

- *d*, do tipo real, distância do centro de projecção ao plano de projecção.

- *b[ ]*, do tipo real, coordenadas do ponto homogéneo original.
- *c[ ]*, do tipo real, coordenadas do ponto homogéneo transformado.

No início da sua tarefa, este módulo inicializa a matriz de transformação geométrica respectiva; de seguida, utiliza o módulo ***multmatvec*** para a execução do produto da matriz transformação pelo ponto homogéneo original.

Este módulo retorna *0* como indicação que a sua execução decorreu com normalidade, e tem como restrição na sua utilização que o valor de *d* deve ser maior do que zero.

1.9 - Módulo *perspective2*

Este módulo é responsável pela transformação geométrica perspectiva quando o plano de projecção é posterior ao centro de projecção e tem coordenada *z* igual a zero.

Como parâmetros de entrada para este módulo têm-se:

- *d*, do tipo real, distância do centro de projecção ao plano de projecção.
- *b[ ]*, do tipo real, coordenadas do ponto homogéneo original.
- *c[ ]*, do tipo real, coordenadas do ponto homogéneo transformado.

No início da sua tarefa, este módulo inicializa a matriz de transformação geométrica respectiva; de seguida, utiliza o módulo ***multmatvec*** para a execução do produto da matriz transformação pelo ponto homogéneo original.

Este módulo retorna *0* como indicação que a sua execução decorreu com normalidade, e tem como restrição na sua utilização que valor de *d* deve ser maior do que zero.

1.10 - Módulo *multmatvec*

Este módulo é responsável pela multiplicação de uma matriz de dimensões 4×4 por um vector coluna de dimensão 4×1.

Como parâmetros de entrada para este módulo têm-se:

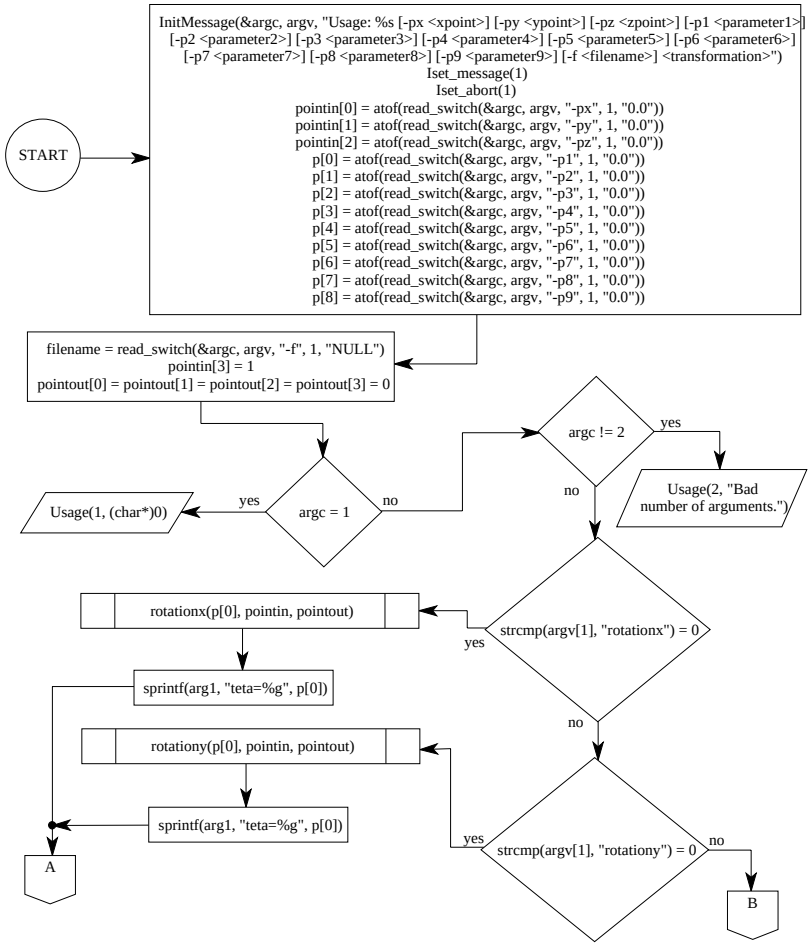
- *a[ ][4]*, do tipo real, matriz de dimensões 4×4.
- *b[ ]*, do tipo real, vector coluna de dimensão 4×1 original.
- *c[ ]*, do tipo real, vector coluna de dimensão 4×1 resultante do produto.

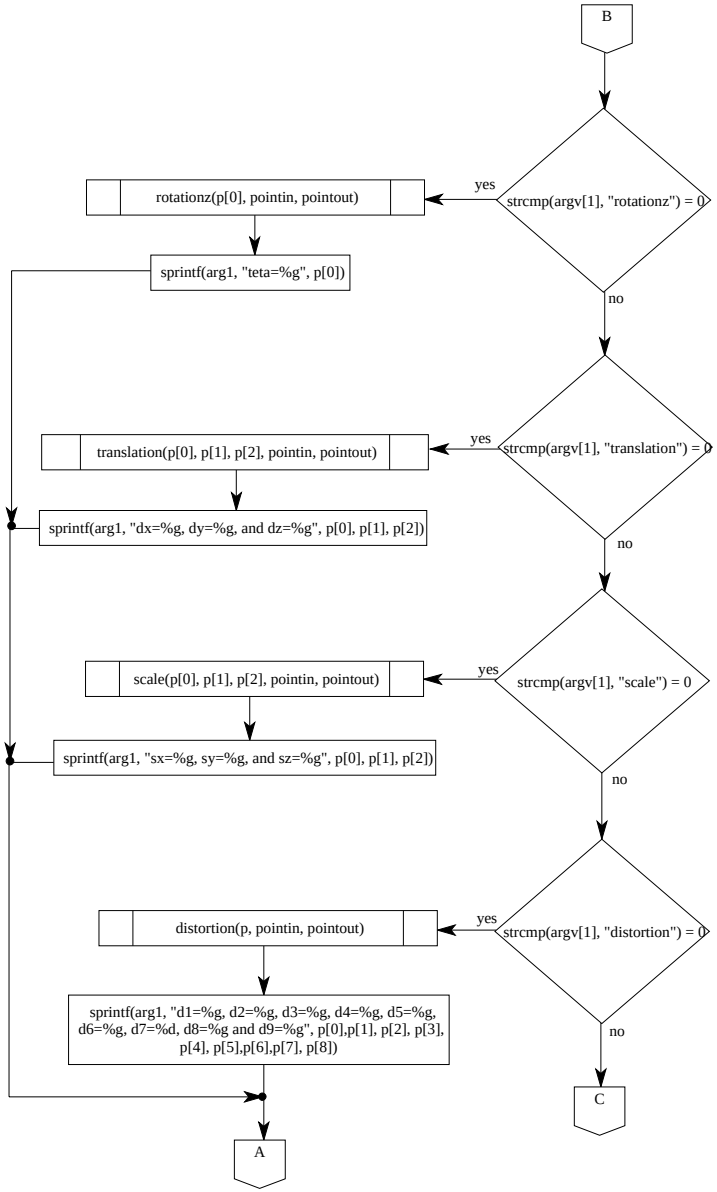
Este módulo retorna *0* como indicação que a sua execução decorreu com normalidade, e não apresenta nenhum tipo de restrições quanto à sua utilização.

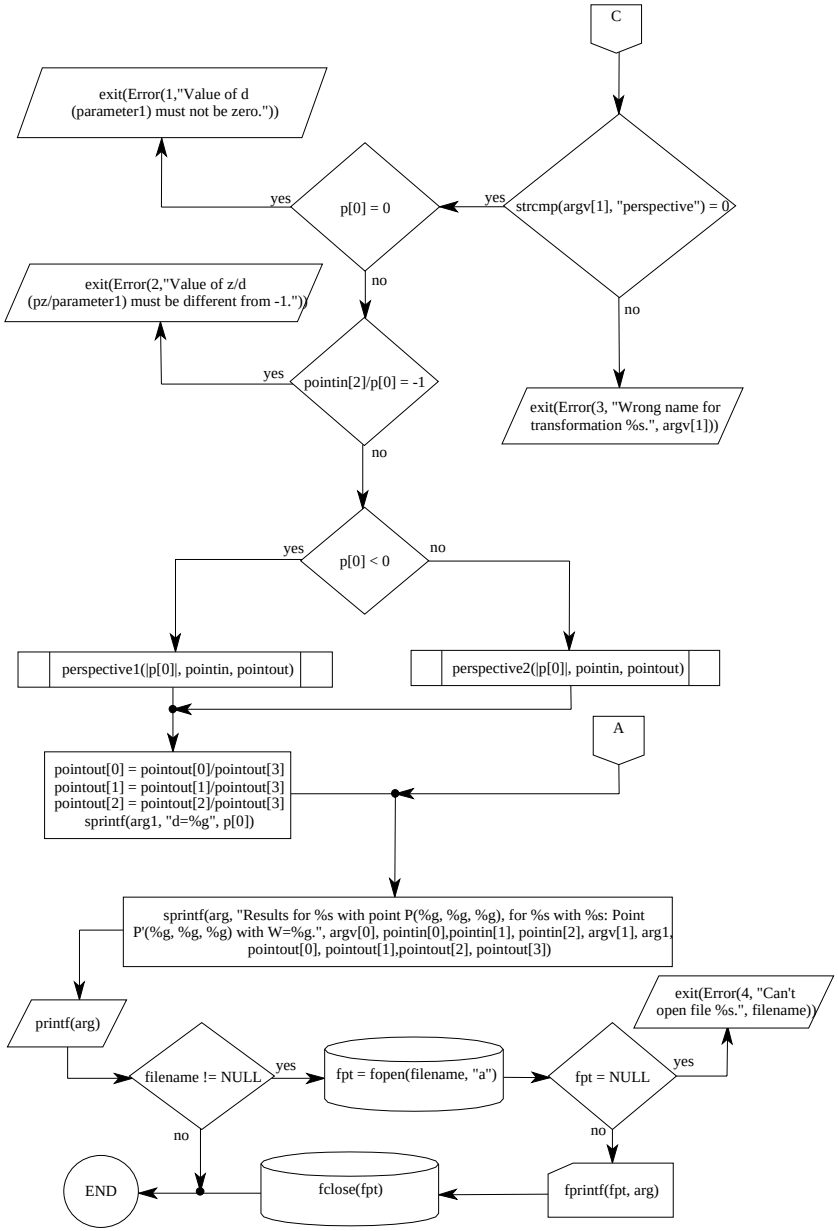
2 - Fluxogramas dos diferentes módulos

Apresentam-se, de seguida, os fluxogramas dos diferentes módulos que constituem a implementação ***geotransf***.

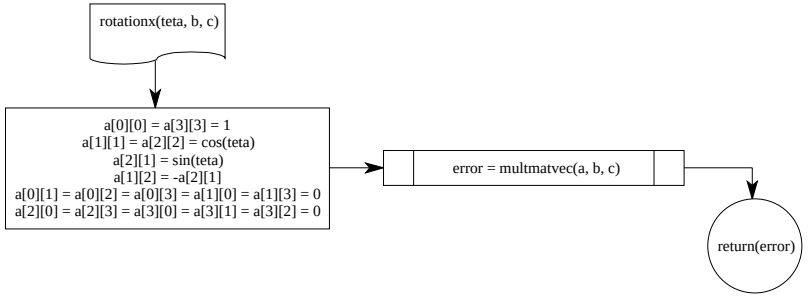
2.1 - Módulo *principal*



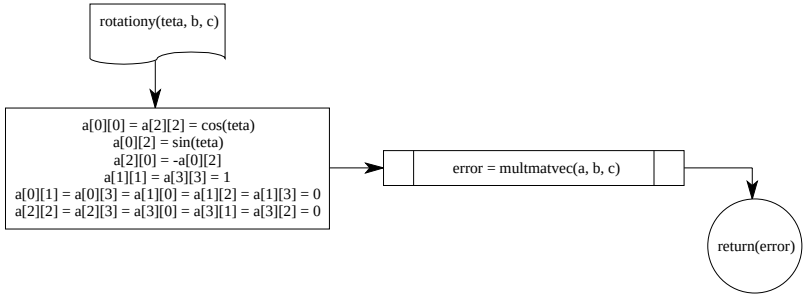




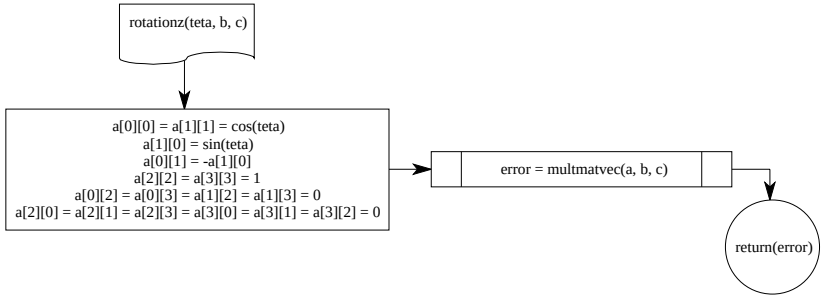
2.2 - Módulo *rotationx*



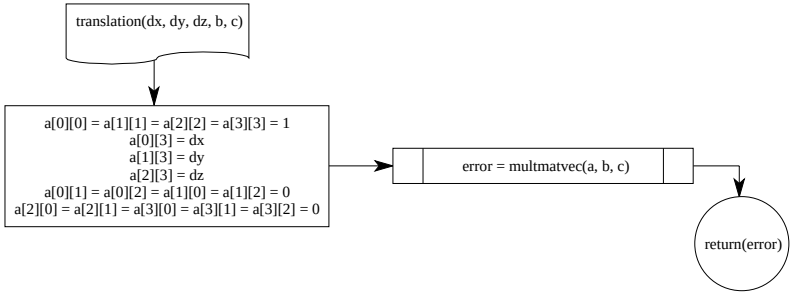
2.3 - Módulo *rotationy*



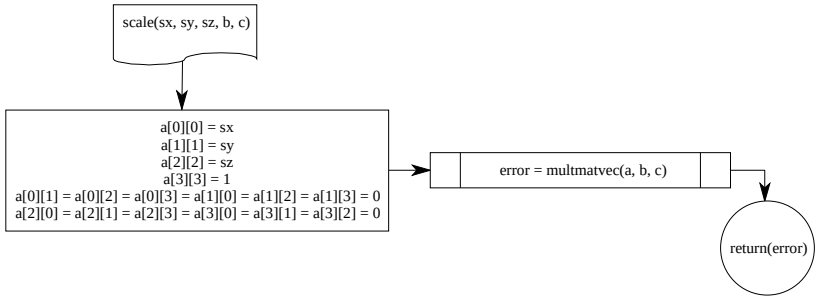
2.4 - Módulo *rotationz*



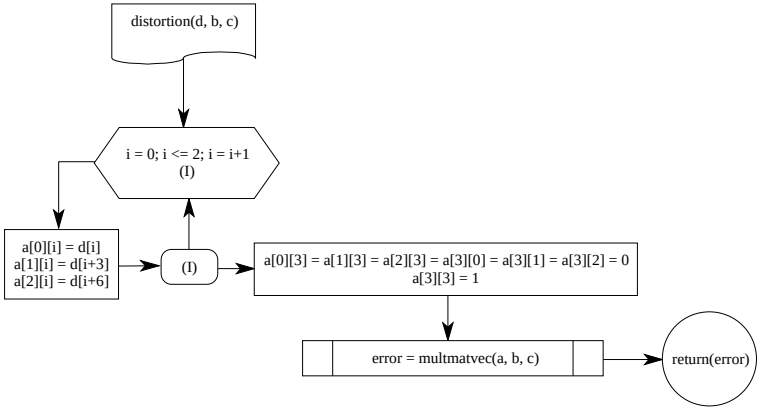
2.5 - Módulo *translation*



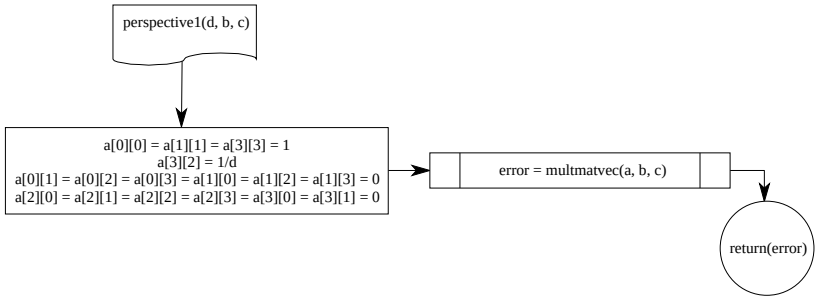
2.6 - Módulo *scale*



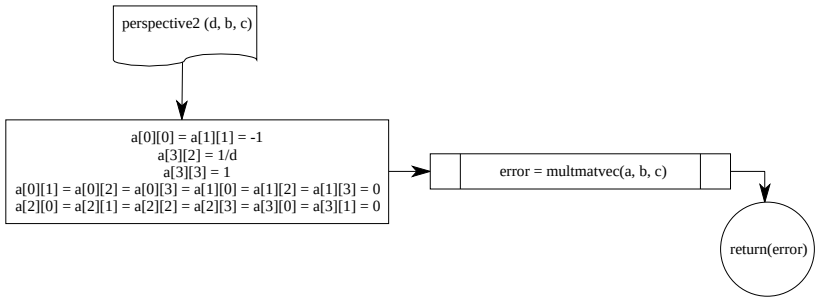
2.7 - Módulo *distortion*



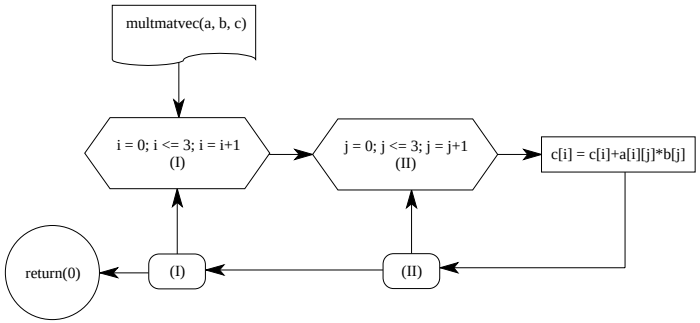
2.8 - Módulo *perspective1*



2.9 - Módulo *perspective2*



2.10 - Módulo *multmatvec*



3 - Listagem da implementação

Apresenta-se, de seguida, a listagem da implementação ***geotransf*** desenvolvida em linguagem *C* em ambiente *UNIX* numa estação de trabalho *SunSparc* e que se destina ao ambiente de processamento de imagem *XITE*[1], do qual utiliza algumas funções e tipos de variáveis já previamente definidas. Esta implementação contém comentários para auxílio à sua compreensão, assim como informação sobre os diferentes módulos, para sua posterior extracção através da ferramenta *CDOC(1)* existente no *XITE*.

[1] *X-based Image processing Tools an Environmemt*, [Lønnestad, 1992].

```
/*
_____________________________________________________________________

   geotransf.c
   @(#)geotransf.c   1.00 93/12/03, Copyright 1993, DEEC, FEUP
   Departamento de Engenharia Electrotecnica e de Computadores
   Faculdade de Engenharia
   Rua dos Bragas
   4099 PORTO CODEX
   PORTUGAL
   E-mail: gpai@obelix.fe.up.pt
_____________________________________________________________________

   readarg.h
   @(#)readarg.h   1.3 92/01/15, Copyright 1990, Blab, UiO
   Image processing lab, Department of Informatics
   University of Oslo
   E-mail: blab@ifi.uio.no
_____________________________________________________________________

   message.h
   @(#)message.h   1.3 92/01/15, Copyright 1990, Blab, UiO
   Image processing lab, Department of Informatics
   University of Oslo
   E-mail: blab@ifi.uio.no
_____________________________________________________________________

  Permission to use, copy, modify and distribute this software and its documentation for any purpose
and without fee is hereby granted, provided that this copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in supporting documentation and that the
name of B-lab, Department of Informatics or University of Oslo not be used in advertising or publicity
pertaining to distribution of the software without specific, written prior  permission.

  B-LAB DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT
SHALL B-LAB BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL
DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA
OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER
TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
*/

/*****************************************************************/

static char *SccsId = "@(#)geotransf.c 1.00 93/12/03, DEEC, FEUP";
```

```
/*********************************************************************/

/* INCLUDES */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <blab/readarg.h>
#include <blab/message.h>

/*********************************************************************/

/* GLOBAL VARIABLE */

FILE *fpt;    /* file's pointer */

/*********************************************************************/

/*F:rotationx*
_____________________________________________________________________

  rotationx
_____________________________________________________________________

Name:          rotationx - rotation of an homogeneous point about the x axis

Syntax:        | rotationx(teta, b, c)
               | double teta, b[], c[];

Description:    'rotationx' performs a rotation about the x axis of the
                homogeneous point 'b' by an angle 'teta'.
                'b[0]' is the x coordinate of the homogeneous input point.
                'b[1]' is the y coordinate of the homogeneous input point.
                'b[2]' is the z coordinate of the homogeneous input point.
                'b[3]' is the fourth coordinate of the homogeneous input point,
                most of time b[3]=1.0.
                'c[0]' is the x coordinate of the homogeneous output point.
                'c[1]' is the y coordinate of the homogeneous output point.
                'c[2]' is the z coordinate of the homogeneous output point.
                'c[3]' is the fourth coordinate of the homogeneous
                output point, if b[3]=1.0 then c[3]=1.0.

Return value:  | 0 => Ok.

Example:       | Try yourself.

Restrictions:   None.

Author:         Joao Tavares

Id:             @(#)rotationx.c   1.00   93/12/03
_____________________________________________________________________
*/

int rotationx(teta, b, c)
double teta, b[], c[];

{

 int error;
 double a[4][4];

 /* initialisation of the transformation matrix a */

 a[0][0] = a[3][3] = 1.0;
 a[1][1] = a[2][2] = cos(teta);
 a[2][1] = sin(teta);
 a[1][2] = -a[2][1];
 a[0][1] = a[0][2] = a[0][3] = a[1][0] = a[1][3] = 0.0;
 a[2][0] = a[2][3] = a[3][0] = a[3][1] = a[3][2] = 0.0;

 /* call multmatvec for the multiplication of matrix a by vector b, vector c */

 error = multmatvec(a, b, c);

 return(error);

}

/*********************************************************************/

/*F:rotationy*
_____________________________________________________________________

  rotationy
_____________________________________________________________________

Name:          rotationy - rotation of an homogeneous point about the y axis
```

Syntax: | rotationy(teta, b, c)
| double teta, b[], c[];

Description: 'rotationy' performs a rotation about the y axis of the homogeneous point 'b' by an angle 'teta'.
'b[0]' is the x coordinate of the homogeneous input point.
'b[1]' is the y coordinate of the homogeneous input point.
'b[2]' is the z coordinate of the homogeneous input point.
'b[3]' is the fourth coordinate of the homogeneous input point, most of time b[3]=1.0.
'c[0]' is the x coordinate of the homogeneous output point.
'c[1]' is the y coordinate of the homogeneous output point.
'c[2]' is the z coordinate of the homogeneous output point.
'c[3]' is the fourth coordinate of the homogeneous output point, if b[3]=1.0 then c[3]=1.0.

Return value: | 0 => Ok.

Example: | Try yourself.

Restrictions: None.

Author: Joao Tavares

Id: @(#)rotationy.c 1.00 93/12/03

---

*/

```
int rotationy(teta, b, c)
double teta, b[], c[];

{

  int error;
  double a[4][4];

  /* initialisation of the transformation matrix a */

  a[0][0] = a[2][2] = cos(teta);
  a[0][2] = sin(teta);
  a[2][0] = -a[0][2];
  a[1][1] = a[3][3] = 1.0;
  a[0][1] = a[0][3] = a[1][0] = a[1][2] = a[1][3] = 0.0;
  a[2][2] = a[2][3] = a[3][0] = a[3][1] = a[3][2] = 0.0;

  /* call multmatvec for the multiplication of matrix a by vector b, vector c */

  error = multmatvec(a, b, c);

  return(error);

}
```

/**/

/*F:rotationz*

---

rotationz

Name: rotationz - rotation of an homogeneous point about the z axis

Syntax: | rotationz(teta, b, c)
| double teta, b[], c[];

Description: 'rotationz' performs a rotation about the z axis of the homogeneous point 'b' by an angle 'teta'.
'b[0]' is the x coordinate of the homogeneous input point.
'b[1]' is the y coordinate of the homogeneous input point.
'b[2]' is the z coordinate of the homogeneous input point.
'b[3]' is the fourth coordinate of the homogeneous input point, most of time b[3]=1.0.
'c[0]' is the x coordinate of the homogeneous output point.
'c[1]' is the y coordinate of the homogeneous output point.
'c[2]' is the z coordinate of the homogeneous output point.
'c[3]' is the fourth coordinate of the homogeneous output point, if b[3]=1.0 then c[3]=1.0.

Return value: | 0 => Ok.

Example: | Try yourself.

Restrictions: None.

Author: Joao Tavares

Id: @(#)rotationz.c 1.00 93/12/03

```
*/

int rotationz(teta, b, c)
double teta, b[], c[];

{

  int error;
  double a[4][4];

  /* initialisation of the transformation matrix a */

  a[0][0] = a[1][1] = cos(teta);
  a[1][0] = sin(teta);
  a[0][1] = -a[1][0];
  a[2][2] = a[3][3] = 1.0;
  a[0][2] = a[0][3] = a[1][2] = a[1][3] = 0.0;
  a[2][0] = a[2][1] = a[2][3] = a[3][0] = a[3][1] = a[3][2] = 0.0;

  /* call multmatvec for the multiplication of matrix a by vector b, vector c */

  error = multmatvec(a, b, c);

  return(error);

}

/*****************************************************************************/

/*F:translation*
_____________________________________________________________________________

  translation
_____________________________________________________________________________

Name:         translation - translation of an homogeneous point

Syntax:       | translation(dx, dy, dz, b, c)
              | double dx, dy, dz, b[], c[];

Description:   'translation' performs a translation of the homogeneous point
               'b' by 'dx' units parallel to the x axis, 'dy' units parallel to the y
               axis and 'dz' units parallel to the z axis.
               'b[0]' is the x coordinate of the homogeneous input point.
               'b[1]' is the y coordinate of the homogeneous input point.
               'b[2]' is the z coordinate of the homogeneous input point.
               'b[3]' is the fourth coordinate of the homogeneous input
               point, most of time b[3]=1.0.
               'c[0]' is the x coordinate of the homogeneous output point.
               'c[1]' is the y coordinate of the homogeneous output point.
               'c[2]' is the z coordinate of the homogeneous output point.
               'c[3]' is the fourth coordinate of the homogeneous output
               point, if b[3]=1.0 then c[3]=1.0.

Return value: | 0 => Ok.

Example:      | Try yourself.

Restrictions:  None.

Author:        Joao Tavares

Id:            @(#)translation.c   1.00   93/12/03
_____________________________________________________________________________
*/

int translation(dx, dy, dz, b, c)
double dx, dy, dz, b[], c[];

{

  int error;
  double a[4][4];

  /* initialisation of the transformation matrix a */

  a[0][0] = a[1][1] = a[2][2] = a[3][3] = 1.0;
  a[0][3] = dx;
  a[1][3] = dy;
  a[2][3] = dz;
  a[0][1] = a[0][2] = a[1][0] = a[1][2] = 0.0;
  a[2][0] = a[2][1] = a[3][0] = a[3][1] = a[3][2] = 0.0;

  /* call multmatvec for the multiplication of matrix a by vector b, vector c */

  error = multmatvec(a, b, c);

  return(error);

}
```

```
/*************************************************************************/

/*F:scale*
_________________________________________________________________________

  scale
_________________________________________________________________________

Name:         scale - scale of an homogeneous point about the origin

Syntax:       | scale(sx, sy, sz, b, c)
              | double sx, sy, sz, b[], c[];

Description:   'scale' performs a scale of the homogeneous point 'b' by 'sx'
               along the x axis, 'sy' along the y axis and 'sz' along the z axis.
               'b[0]' is the x coordinate of the homogeneous input point.
               'b[1]' is the y coordinate of the homogeneous input point.
               'b[2]' is the z coordinate of the homogeneous input point.
               'b[3]' is the fourth coordinate of the homogeneous input point,
               most of time b[3]=1.0.
               'c[0]' is the x coordinate of the homogeneous output point.
               'c[1]' is the y coordinate of the homogeneous output point.
               'c[2]' is the z coordinate of the homogeneous output point.
               'c[3]' is the fourth coordinate of the homogeneous output
               point, if b[3]=1.0 then c[3]=1.0.

Return value: | 0 => Ok.

Example:      | Try yourself.

Restrictions:  None.

Author:       Joao Tavares

Id:           @(#)scale.c   1.00   93/12/03
_________________________________________________________________________
*/

int scale(sx, sy, sz, b, c)
double sx, sy, sz, b[], c[];

{

  int error;

  double a[4][4];

  /* initialisation of the transformation matrix a */

  a[0][0] = sx;
  a[1][1] = sy;
  a[2][2] = sz;
  a[3][3] = 1.0;
  a[0][1] = a[0][2] = a[0][3] = a[1][0] = a[1][2] = a[1][3] = 0.0;
  a[2][0] = a[2][1] = a[2][3] = a[3][0] = a[3][1] = a[3][2] = 0.0;

  /* call multmatvec for the multiplication of matrix a by vector b, vector c */

  error = multmatvec(a, b, c);

  return(error);

}

/*************************************************************************/

/*F:distortion*
_________________________________________________________________________

  distortion
_________________________________________________________________________

Name:         distortion - distortion of an homogeneous point

Syntax:       | distortion(d, b, c)
              | double d[], b[], c[];

Description:   'distortion' performs a distortion of the homogeneous point 'b'.
               'b[0]' is the x coordinate of the homogeneous input point.
               'b[1]' is the y coordinate of the homogeneous input point.
               'b[2]' is the z coordinate of the homogeneous input point.
               'b[3]' is the fourth coordinate of the homogeneous input
               point, most of time b[3]=1.0.
               'c[0]' is the x coordinate of the homogeneous output point.
               'c[1]' is the y coordinate of the homogeneous output point.
               'c[2]' is the z coordinate of the homogeneous output point.
               'c[3]' is the fourth coordinate of the homogeneous output
               point, if b[3]=1.0 then c[3]=1.0.
               'd[0]' is the dxx parameter of distortion.
               'd[1]' is the dxy parameter of distortion.
```

```
                'd[2]' is the dxz parameter of distortion.
                'd[3]' is the dyx parameter of distortion.
                ...
                'd[8]' is the dzz parameter of distortion.

Return value:   | 0 => Ok.

Example:        | Try yourself.

Restrictions:   None.

Author:         Joao Tavares

Id:             @(#)distortion.c   1.00   93/12/03
_____________________________________________________________________
*/

int distortion(d, b, c)
double d[], b[], c[];

{

  register int i;
  int error;
  double a[4][4];

  /* initialisation of the transformation matrix a */

  for (i = 0; i <= 2; ++i) {

    a[0][i] = d[i];
    a[1][i] = d[i+3];
    a[2][i] = d[i+6];

  }
  a[0][3] = a[1][3] = a[2][3] = a[3][0] = a[3][1] = a[3][2] = 0.0;
  a[3][3] = 1.0;

  /* call multmatvec for the multiplication of matrix a by vector b, vector c */

  error = multmatvec(a, b, c);

  return(error);

}
```

```
/*********************************************************************/

/*F:perspective1*
_____________________________________________________________________

  perspective1
_____________________________________________________________________

Name:         perspective1 - perspective of an homogeneous point when the
                             projection plane is before the projection center

Syntax:       | perspective1(d, b, c)
              | double d, b[], c[];

Description:   'perspective1' performs a perspective of the homogeneous point
               'b' when the projection plane is before the center of projection
               and have the z coordinate zero.
               'b[0]' is the x coordinate of the homogeneous input point.
               'b[1]' is the y coordinate of the homogeneous input point.
               'b[2]' is the z coordinate of the homogeneous input point.
               'b[3]' is the fourth coordinate of the homogeneous input point,
               most of time b[3]=1.0.
               'c[0]' is the x coordinate of the homogeneous output point.
               'c[1]' is the y coordinate of the homogeneous output point.
               'c[2]' is the z coordinate of the homogeneous output point.
               'c[3]' is the fourth coordinate of the homogeneous output
               point, if b[3]=1.0 then c[3]=1.0.
               'd' is the distance from projection plane and the center of
               projection.

Return value: | 0 => Ok.

Example:      | Try yourself.

Restrictions: 'd' must be greater than zero.

Author:       Joao Tavares

Id:           @(#)perspective1.c   1.00   93/12/03
_____________________________________________________________________
*/

int perspective1(d, b, c)
double d, b[], c[];
```

```
{

  int error;
  double a[4][4];

  /* initialisation of the transformation matrix a */

  a[0][0] = a[1][1] = a[3][3] = 1.0;
  a[3][2] = 1.0/d;
  a[0][1] = a[0][2] = a[0][3] = a[1][0] = a[1][2] = a[1][3] = 0.0;
  a[2][0] = a[2][1] = a[2][2] = a[2][3] = a[3][0] = a[3][1] = 0.0;

  /* call multmatvec for the multiplication of matrix a by vector b, vector c */

  error = multmatvec(a, b, c);

  return(error);

}

/*****************************************************************************/

/*F:perspective2*
_____________________________________________________________________

  perspective2
_____________________________________________________________________

Name:         perspective2 - perspective of an homogeneous point when the
                             projection plane is after the projection center

Syntax:       | perspective2(d, b, c)
              | double d, b[], c[];

Description:   'perspective2' performs a perspective of the homogeneous point 'b'
               when the projection plane is after the center of projection and have
               the z coordinate zero.
               'b[0]' is the x coordinate of the homogeneous input point.
               'b[1]' is the y coordinate of the homogeneous input point.
               'b[2]' is the z coordinate of the homogeneous input point.
               'b[3]' is the fourth coordinate of the homogeneous input point,
               most of time b[3]=1.0.
               'c[0]' is the x coordinate of the homogeneous output point.
               'c[1]' is the y coordinate of the homogeneous output point.
               'c[2]' is the z coordinate of the homogeneous output point.
               'c[3]' is the fourth coordinate of the homogeneous output point,
               if b[3]=1.0 then c[3]=1.0.
               'd' is the distance from projection plane and the center of
               projection.

Return value:  | 0 => Ok.

Example:       | Try yourself.

Restrictions:  'd' must be greater than zero.

Author:        Joao Tavares

Id:            @(#)perspective2.c   1.00   93/12/03
_____________________________________________________________________
*/

int perspective2 (d, b, c)
double d, b[], c[];

{

  int error;
  double a[4][4];

  /* initialisation of the transformation matrix a */

  a[0][0] = a[1][1] = -1.0;
  a[3][2] = 1.0/d;
  a[3][3] = 1.0;
  a[0][1] = a[0][2] = a[0][3] = a[1][0] = a[1][2] = a[1][3] = 0.0;
  a[2][0] = a[2][1] = a[2][2] = a[2][3] = a[3][0] = a[3][1] = 0.0;

  /* call multmatvec for the multiplication of matrix a by vector b, vector c */

  error = multmatvec(a, b, c);

  return(error);

}

/*****************************************************************************/

/*F:multmatvec*
```

---

multmatvec

Name: multmatvec - multiplication of a 4x4 matrix by a 4x1 vector

Syntax: | multmatvec(a, b, c)
| double a[][], b[], c[];

Description: 'multmatvec' performs a multiplication of the 4x4 matrix 'a' by the 4x1 vector 'b', 'c' is the 4x1 output vector.

Return value: | 0 => Ok.

Example: | Try yourself.

Restrictions: None.

Author: Joao Tavares

Id: @(#)multmatvec.c 1.00 93/12/03

```
*/

int multmatvec(a, b, c)
double a[][4], b[], c[];

{

  register int i, j;

  /* multiplication of matrix a by vector b, vector c */

  for (i = 0; i <= 3; ++i) {

    for (j = 0; j <= 3; ++j) c[i] += a[i][j]*b[j];

  }

  return(0);

}

/*************************************************************************/
```

/*P:geotransf*

---

geotransf

Name: geotransf - geometrical transformations of a point

Syntax: | geotransf [-px <xpoint>] [-py <ypoint>] [-pz <zpoint>]
| [-p1 <parameter1>] [-p2 <parameter2>] [-p3 <parameter3>]
| [-p4 <parameter4>] [-p5 <parameter5>] [-p6 <parameter6>]
| [-p7 <parameter7>] [-p8 <parameter2>] [-p9 <parameter9>]
| [-f <filename>] <transformation>

Description: 'geotransf' perform the geometrical 'transformation' of a point.
'xpoint' is the x coordinate of the input point, by default is zero.
'ypoint' is the y coordinate of the input point, by default is zero.
'zpoint' is the z coordinate of the input point, by default is zero.
The output results can be writing in the file 'filename' if the flag '-f' is on, by default this flag is off.

'transformation' can be:

rotationx - rotation about x axis, 'parameter1' is the rotation angle.
rotationy - rotation about y axis, 'parameter1' is the rotation angle.
rotationz - rotation about z axis, 'parameter1' is the rotation angle.
translation - translation, 'parameter1' is dx, 'parameter2' is dy and 'parameter3' is dz,
scale - scale about the origin, 'parameter1' is sx, 'parameter2' is sy and 'parameter3' is sz.
distortion - distortion, with the 'parameteri' 1<=i<=9 the parameter's of distortion ('parameter1' is dxx, ...,'parameter3' is dxz, ..., 'parameter9' is dzz).
perspective - perspective with a distance d from projection plane and the center of projection, 'parameter1' is d.

'parameteri' with 1<=i<=9 is by default zero.

Return value: | 1 => Value of d (<parameter1>) must not be zero.
| 2 => Value of z/d (<zpoint>/<parameter1>) must be different
| from -1.
| 3 => Wrong name for <transformation>.
| 4 => Can't open file <filename>.

```
Examples:       | Try yourself.

Restrictions:   If the transformation is perspective then <parameter1> must not be
                zero and <zpoint>/<parameter1> must be different from -1.

Author:         Joao Tavares

Id:             @(#)geotransf.c   1.0   93/12/03
_____________________________________________________________________
*/

#ifdef MAIN

main(argc, argv)
int argc;
char **argv;

{

  double p[9], pointin[4], pointout[4];
  char *filename, arg[200], arg1[80];

  InitMessage(&argc, argv, "Usage: %s\n [-px <xpoint>] [-py <ypoint>] [-pz <zpoint>] [-p1
<parameter1>] [-p2 <parameter2>] [-p3 <parameter3>] [-p4 <parameter4>] [-p5 <parameter5>] [-p6
<parameter6>] [-p7 <parameter7>] [-p8 <parameter8>] [-p9 <parameter9>] [-f <filename>]
<transformation>\n");
  Iset_message(1);
  Iset_abort(1);

  /* initialisation of elements of p, pointin, pointout */

  /* read switches */

  pointin[0] = atof(read_switch(&argc, argv, "-px", 1, "0.0"));
  pointin[1] = atof(read_switch(&argc, argv, "-py", 1, "0.0"));
  pointin[2] = atof(read_switch(&argc, argv, "-pz", 1, "0.0"));
  p[0] = atof(read_switch(&argc, argv, "-p1", 1, "0.0"));
  p[1] = atof(read_switch(&argc, argv, "-p2", 1, "0.0"));
  p[2] = atof(read_switch(&argc, argv, "-p3", 1, "0.0"));
  p[3] = atof(read_switch(&argc, argv, "-p4", 1, "0.0"));
  p[4] = atof(read_switch(&argc, argv, "-p5", 1, "0.0"));
  p[5] = atof(read_switch(&argc, argv, "-p6", 1, "0.0"));
  p[6] = atof(read_switch(&argc, argv, "-p7", 1, "0.0"));
  p[7] = atof(read_switch(&argc, argv, "-p8", 1, "0.0"));
  p[8] = atof(read_switch(&argc, argv, "-p9", 1, "0.0"));
  filename = read_switch(&argc, argv, "-f", 1, NULL);

  pointin[3] = 1.0;
  pointout[0] = pointout[1] = pointout[2] = pointout[3] = 0.0;

  if (argc == 1) Usage(1, (char*)0);
  if (argc != 2) Usage(2, "\nBad number of arguments\n");

  if (strcmp(argv[1], "rotationx") == 0) {

   /* call rotationx */

   rotationx(p[0], pointin, pointout);
   sprintf(arg1, "teta=%g", p[0]);

  }

  else {

   if (strcmp(argv[1], "rotationy") == 0) {

    /* call rotationy */

    rotationy(p[0], pointin, pointout);
    sprintf(arg1, "teta=%g", p[0]);

   }

   else {

    if (strcmp(argv[1], "rotationz") == 0) {

     /* call rotationz */

     rotationz(p[0], pointin, pointout);
     sprintf(arg1, "teta=%g", p[0]);

    }

    else {

     if (strcmp(argv[1], "translation") == 0) {

      /* call translation */
```

```
        translation(p[0], p[1], p[2], pointin, pointout);
        sprintf(arg1, "dx=%g, dy=%g, and dz=%g", p[0], p[1], p[2]);

      }

     else {

       if (strcmp(argv[1], "scale") == 0) {

         /* call scale */

         scale(p[0], p[1], p[2], pointin, pointout);
         sprintf(arg1, "sx=%g, sy=%g, and sz=%g", p[0], p[1], p[2]);

       }

       else {

         if (strcmp(argv[1], "distortion") == 0) {

           /* call distortion */

           distortion(p, pointin, pointout);
           sprintf(arg1, "d1=%g, d2=%g, d3=%g, d4=%g, d5=%g, d6=%g, d7=%d, d8=%g and
d9=%g", p[0],p[1], p[2], p[3], p[4], p[5],p[6],p[7], p[8]);

         }

         else {

           if (strcmp(argv[1], "perspective") == 0) {

             if ( p[0] == 0.0) exit(Error(1,"\nValue of d (parameter1) must not be zero.\n"));

             if ( pointin[2]/p[0] == -1) exit(Error(2,"\nValue of z/d (pz/parameter1) must be different
from -1.\n"));

             if (p[0] < 0.0) {

               /* call perspective1 */

               perspective1(fabs(p[0]), pointin, pointout);

             }

             else {

               /* call perspective2 */

               perspective2(p[0], pointin, pointout);

             }

             /* determination of the cartesian point */

             pointout[0] /= pointout[3];
             pointout[1] /= pointout[3];
             pointout[2] /= pointout[3];

             sprintf(arg1, "d=%g", p[0]);


           }

           else exit(Error(3, "\nWrong name for transformation %s.\n", argv[1]));

         }

       }

     }

   }

  }

  sprintf(arg, "\nResults for %s with point P(%g, %g, %g), for %s with %s:\nPoint P'(%g, %g, %g)
with W=%g.\n", argv[0], pointin[0],pointin[1], pointin[2], argv[1], arg1, pointout[0],
pointout[1],pointout[2], pointout[3]);

  /* output results */

  printf(arg);

  /* open file, write arg, and close, if it was desire */

  if (filename != NULL) {
```

```
    fpt = fopen(filename, "a");
    if (fpt == NULL) exit(Error(4, "\nCan't open file %s.\n", filename));
    fprintf(fpt, arg);
    fclose(fpt);

  }

}

#endif

/*****************************************************************************/
```

Capítulo II

Calibração de Câmaras para Utilização em Visão Tridimensional

Introdução

Neste capítulo, são apresentadas cinco implementações com interesse no domínio da calibração de câmaras para utilização em visão tridimensional; assim, são apresentadas as seguintes implementações:

- ✓ Calibração do factor de incerteza horizontal de uma câmara: ***findsx***.
- ✓ Calibração de todos os parâmetros intrínsecos e extrínsecos de uma câmara, exceptuando o factor de incerteza horizontal: ***calcamera***.
- ✓ Determinação das coordenadas memória *frame* dos pontos utilizados na calibração: ***calpoint***.
- ✓ Simulação de uma dada câmara: ***simcamera***.
- ✓ Formatação conveniente dos dados de entrada para a implementação ***calcamera***: ***input***.

O modelo utilizado para a câmara nas implementações ***findsx***, ***calcamera*** e ***simcamera***, foi inicialmente proposto por Roger Tsai em [Tsai, 1987] e está representado na *Fig. 1*.

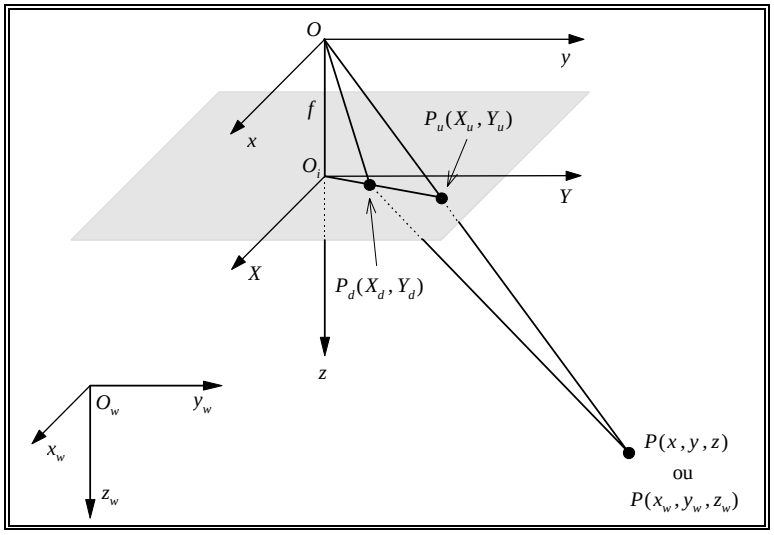


Fig. 1 - Modelo utilizado para a câmara.

Na *Fig. 1*:

- (x_w, y_w, z_w) representa as coordenadas 3D do ponto *P* no sistema mundo;
- (x, y, z) representa as coordenadas 3D do ponto *P* no sistema câmara, o qual é centrado no ponto *O* (o centro óptico) e tem o eixo *z* coincidente com o eixo óptico;
- (X, Y) representa o sistema de coordenadas imagem centrado em O_i (intersecção do eixo óptico *z* com o plano imagem) e com os eixos *X* e *Y* paralelos, respectivamente, aos eixos *x* e *y* do sistema câmara;
- *f* é a distância entre o plano imagem e o centro óptico (isto é, a distância focal efectiva);
- (X_u, Y_u) representa as coordenadas imagem de *P*, se for utilizado um modelo *pinhole* perfeito para a câmara (isto é, as coordenadas imagem sem distorção);
- (X_d, Y_d) representa as coordenadas imagem real, que diferem de (X_u, Y_u) devido à distorção da lente.

Como é possível depreender pelo modelo utilizado para a câmara, os parâmetros a calibrar são: os três componentes do vector rotação *R* e os três componentes do vector translação *T*, que definem a

transformação geométrica global do sistema de coordenadas 3D mundo para o sistema 3D câmara, a distância focal efectiva *f*, e a distorção radial imposta pela lente k_1. Contudo, como a unidade para as coordenadas utilizadas no computador, (X_f, Y_f), é o número de *pixels* para a imagem discreta na memória *frame*, parâmetros adicionais necessitam de ser especificados e calibrados; nomeadamente, as coordenadas discretas do centro óptico da imagem na memória *frame* do computador (C_x, C_y), o factor de incerteza horizontal s_x, a distância entre centros dos elementos sensores vizinhos na direcção *x* e a distância entre centros dos sensores CCD vizinhos na direcção *y*.

1 - Calibração do factor de incerteza horizontal de uma câmara: findsx

Nesta secção, é apresentada uma implementação para a calibração do factor de incerteza horizontal de uma câmara s_x, segundo o método descrito em [Tavares, 1995] e inicialmente apresentado em [Lenz, 1988].

1.1 - Descrição dos diferentes módulos

A implementação é constituída pelos módulos: ***principal***, ***dft*** e ***find***, *Fig. 1*. Apresenta-se de seguida a descrição de cada um destes módulos.

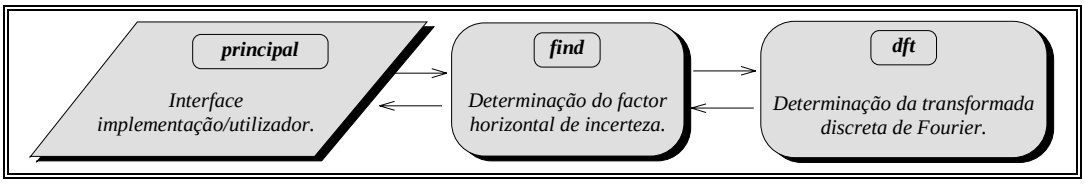


*Fig. 1 - Módulos integrantes da implementação **findsx** e suas relações.*

1.1.1 - Módulo principal

Este módulo é responsável pela interface implementação/utilizador. A seu cargo está:

- ✓ Leitura dos argumentos opcionais ("*switches*"):
  - *filename*, ("[-f <filename>]"), nome do ficheiro para escrita dos resultados obtidos pela implementação, nomeadamente as condições da realização da calibração e o factor de incerteza horizontal. Por defeito é "*NULL*", o que implica a não abertura do ficheiro. Quando especificado, o ficheiro, se ainda não existir, é criado; caso contrário, os resultados são acrescentados ao ficheiro já existente.
  - *nlines*, ("[-l <nlines>]"), número de linhas centradas no centro da imagem a considerar para obtenção de uma linha de valores médios. Por defeito, é igual a *19*.
  - *ncolumns*, ("[-c <ncolumns>]"), número de colunas centradas no centro da imagem a considerar. Por defeito, é igual a *19*.
  - *nzeros*, ("[-z <nzeros>]"), número de zeros a acrescentar às amostras antes de realizar a transformada discreta de Fourier para interpolar a representação espectral. Por defeito, é igual a *60*.
- ✓ Leitura dos argumentos passados à implementação pela linha de comando:
  - *Ncx/fc*, ("<Ncx/fc>"), número de elementos sensores na direcção *x* (N_{cx}), ou frequência de relógio dos elementos sensores da câmara f_c.
  - *Nfx/ff*, ("<Nfx/ff>"), número de elementos imagem numa linha amostrados pelo computador N_{fx}, ou frequência de amostragem do conversor analógico/digital f_f.

- *input*, (“<input>”), nome da imagem obtida sem o filtro passa-baixo entre a câmara e o conversor analógico/digital e com um objecto difuso colocado sobre a lente.

✓ Chamada do módulo ***find***.

✓ Apresentação dos resultados obtidos.

✓ Abertura do ficheiro de saída de resultados, escrita dos resultados obtidos e respectivo fecho, se desejado pelo utilizador.

✓ Comunicação da existência de erros em argumentos de entrada ou ao longo do decurso da execução:

- Através de valores de retorno:
  - 1, (“Wrong number of lines; it must be odd.”), o valor especificado para *nlines* não é ímpar.
  - 2, (“Wrong number of columns; it must be odd.”), o valor especificado para *ncolumns* não é ímpar.
  - 3, (“Nfx or ff must be different from zero.”), o valor especificado para o número de elementos imagem numa linha amostrados pelo computador, ou para a frequência de amostragem do conversor analógico/digital, não é diferente de zero.
  - 4, (“Can’t read image <input>.”), o módulo não consegue realizar com êxito a leitura da imagem especificada.
  - 5, (“The <input> has more than one band.”), a imagem de entrada é constituída por mais de uma banda.
  - 6, (“Bad input pixel type.”), os *pixels* da imagem de calibração não são do tipo *“UNS_BYTE”*.
  - 7, (“Wrong value for the number of lines; it must be smaller than ysize of input image.”), o valor especificado para *nlines* não é menor que o tamanho da imagem de entrada segundo a direcção *y*.
  - 8, (“Wrong value for the number of columns; it must be smaller than xsize of input image.”), o valor especificado para *ncolumns* não é menor que o tamanho da imagem de entrada segundo a direcção *x*.
  - 9, (“Can’t open file <filename>.”), o módulo não consegue abrir o ficheiro de saída especificado.
- Através de mensagens:
  - 1, (*InitMessage*), quando o número de argumentos é igual a um, isto é, apenas o nome da implementação.
  - 2, (“Bad number of arguments.”), quando o número de argumentos é diferente de oito.

Como restrições na utilização deste módulo têm-se:

⌦ *nlines* deve ser ímpar e menor que o tamanho da imagem segundo a direcção *y*.

⌦ *ncolumns* deve ser ímpar e menor que o tamanho da imagem segundo a direcção *x*.

⌦ *Nfx/ff* deve ser diferente de zero.

⌦ *input* deve ser do tipo *“UNS_BYTE”* com apenas uma banda.

1.1.2 - Módulo dft

Este módulo é responsável pelo cálculo da transformada discreta de Fourier[1], DFT, de uma sequência de *n* amostras.

Como parâmetros de entrada para este módulo têm-se:

- *n*, do tipo inteiro, é o número de amostras da sequência a transformar.
- *x[ ]*, do tipo real, é o vector que contém as *n* amostras reais.
- *xreal[ ]*, do tipo real, é o vector que contém, para as várias amostras, a parte real dos resultados.
- *ximag[ ]*, do tipo real, é o vector que contém, para as várias amostras, a parte imaginária dos resultados.

Como desvantagem na utilização deste módulo, salienta-se o tempo consumido para um número elevado de amostras, o que não é o caso da utilização prevista. Para utilizações em que este módulo seja demasiado lento, seria mais adequado e recomendável uma implementação da transformada discreta de Fourier[1] rápida, FFT.

Este módulo retorna *0* como indicação que a execução decorreu com normalidade, e não apresenta qualquer tipo de restrição na sua utilização.

1.1.3 - Módulo find

Este módulo é responsável pela calibração do factor de incerteza horizontal de uma câmara, segundo o método descrito em [Tavares, 1995].

Como parâmetros de entrada para este módulo têm-se:

- *band*, do tipo "*IBAND*", banda da imagem obtida sem o filtro passa-baixo entre a câmara e o conversor analógico/digital e com um objecto difuso colocado sobre a lente.
- *nl*, do tipo inteiro, número de linhas centradas no centro da imagem, para obtenção de uma linha de valores médios.
- *nc*, do tipo inteiro, número de colunas, centradas no centro da imagem, a considerar.
- *nz*, do tipo inteiro, número de zeros a acrescentar às amostras, para interpolar a representação espectral.
- *ncx*, do tipo real, número de elementos sensores na direcção *x*, ou frequência de relógio dos elementos sensores da câmara.
- *nfx*, do tipo real, número de elementos imagem numa linha amostrados pelo computador, ou frequência de amostragem do conversor analógico/digital.

Este módulo retorna o factor de incerteza horizontal, e como restrições à sua utilização têm-se:

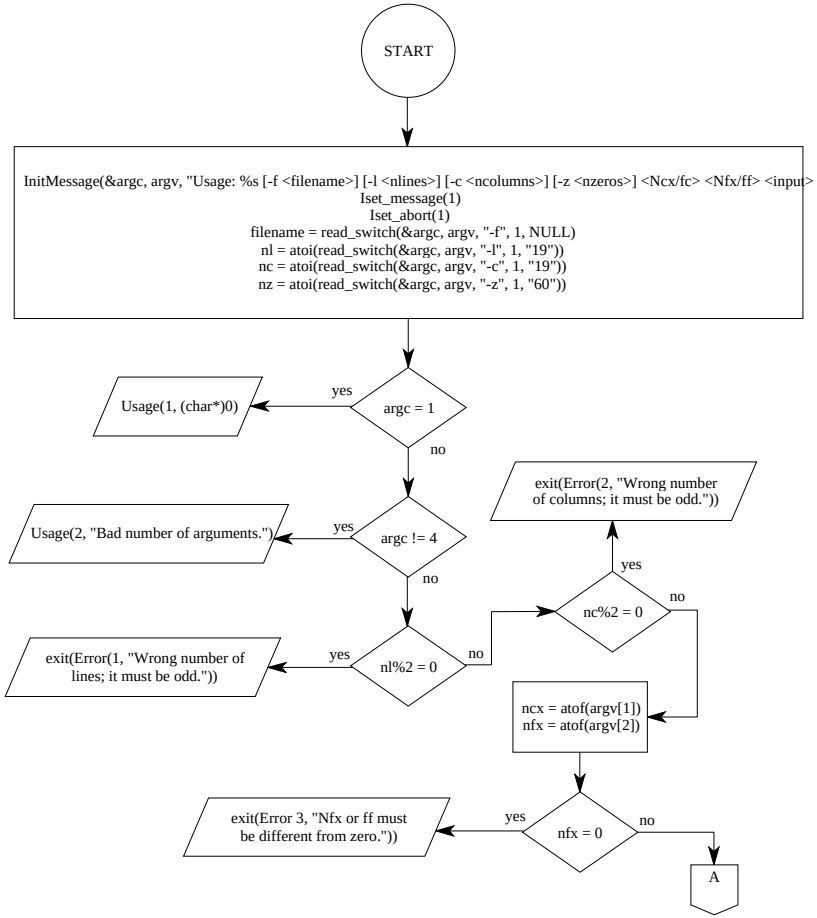
- *nl* deve ser ímpar e menor que o tamanho da imagem segundo a direcção *y*;
- *nc* deve ser ímpar e menor que o tamanho da imagem segundo a direcção *x*;
- *nfx* deve ser diferente de zero;
- *band* deve ser do tipo "*UNS_BYTE*".

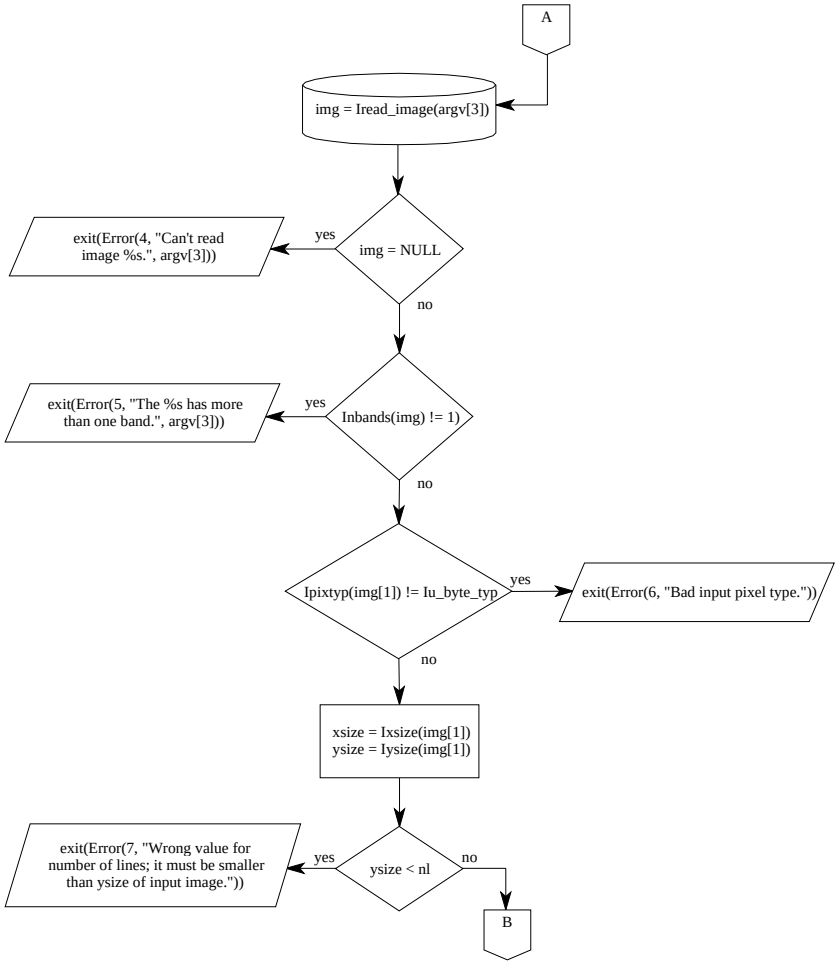
[1] Ver, por exemplo [Chapra, 1988].

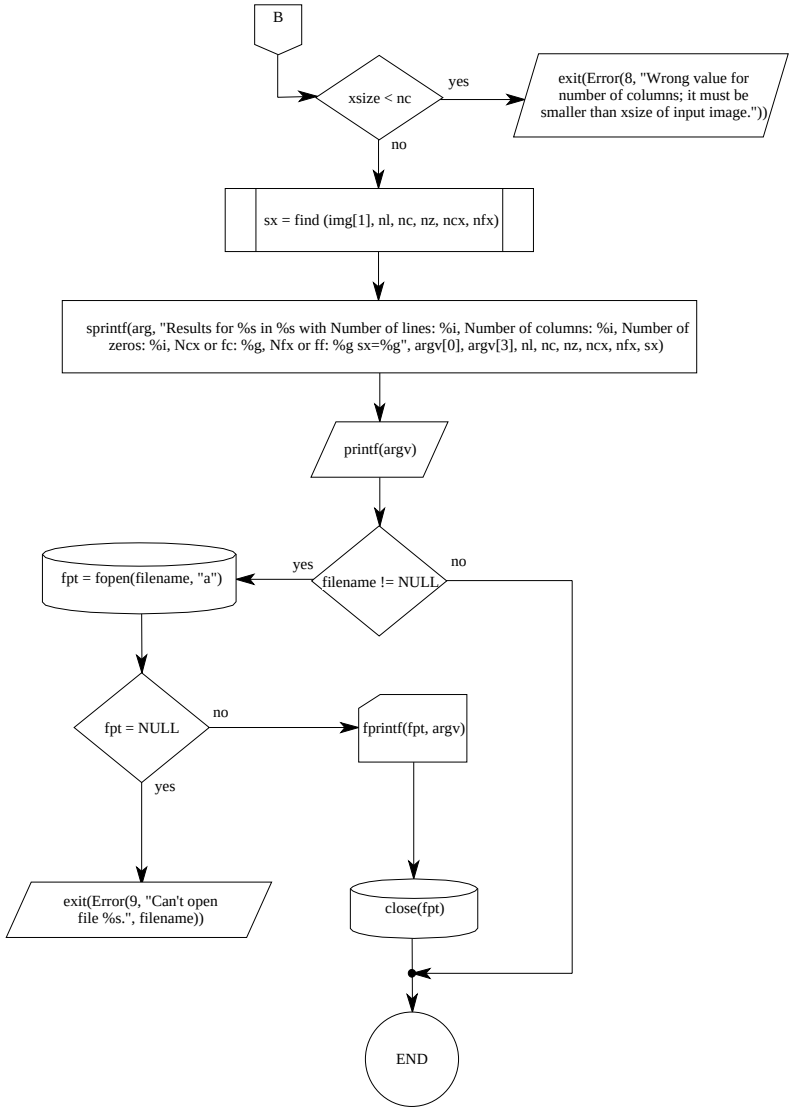
1.2 - Fluxogramas dos diferentes módulos

Apresentam-se, de seguida, os fluxogramas dos módulos que constituem a implementação ***findsx***.

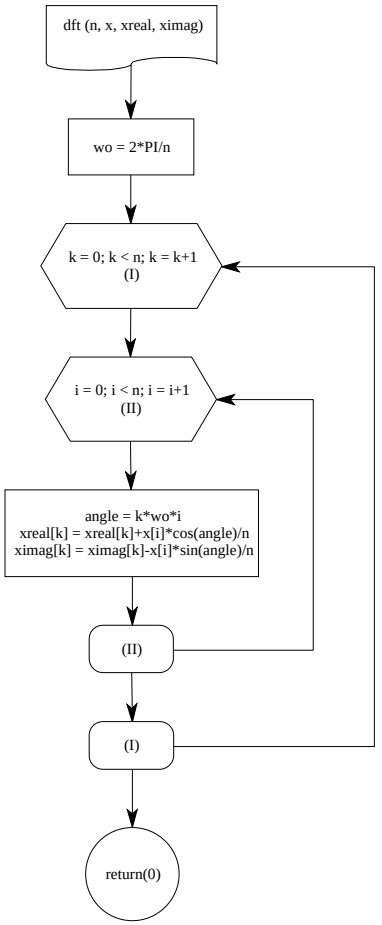
1.2.1 - Módulo principal



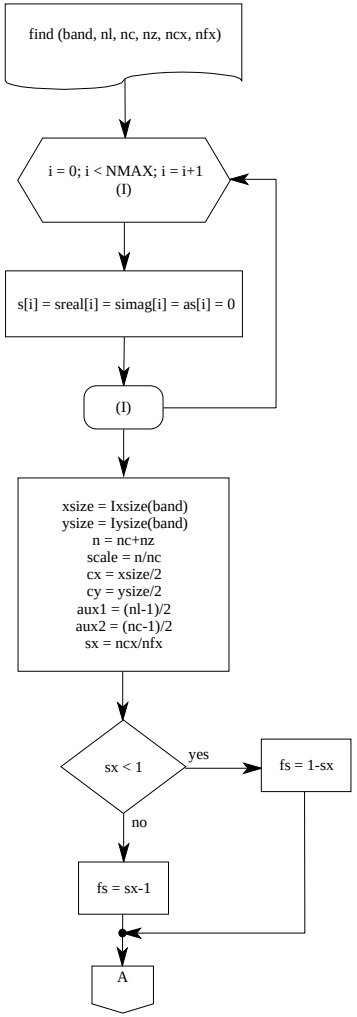


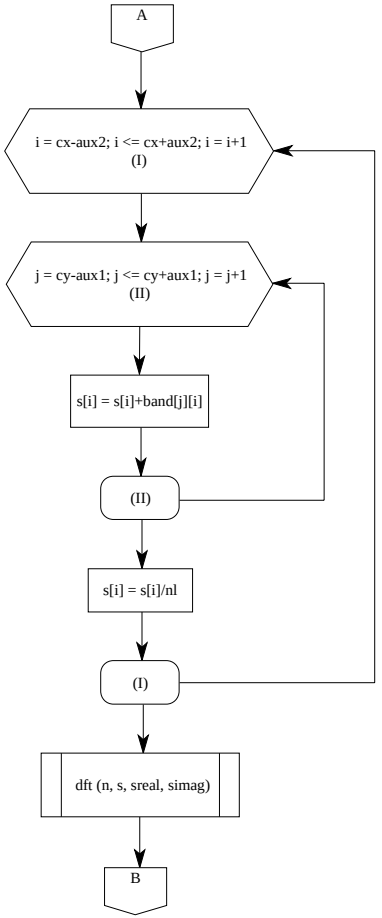


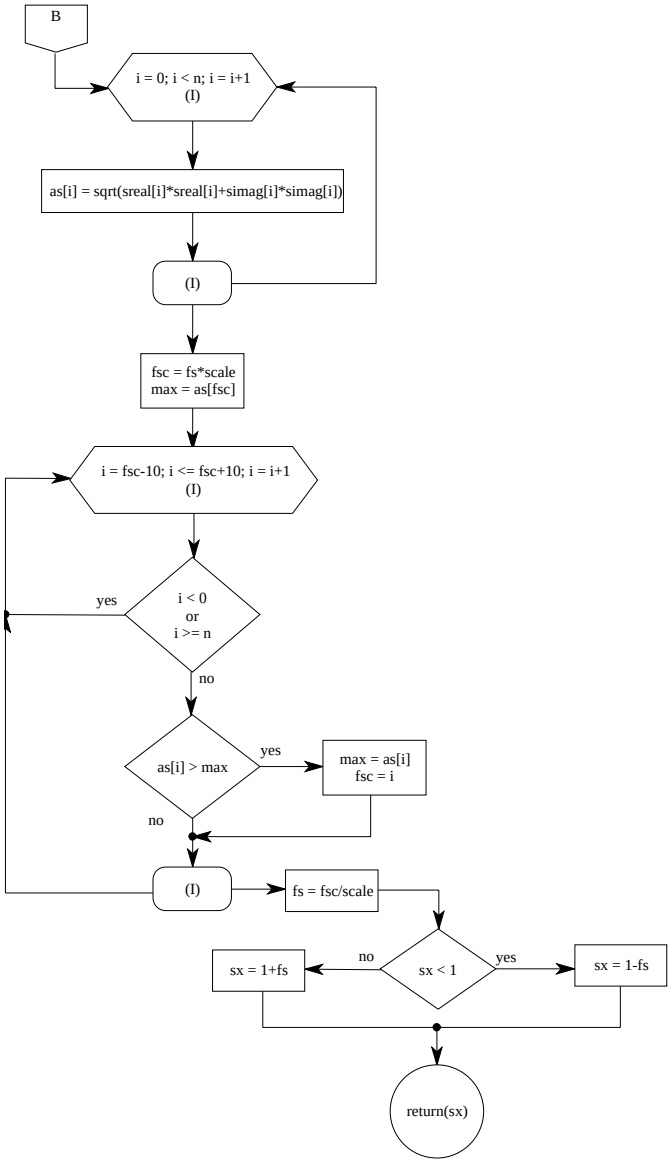
1.2.2 - Módulo dft



1.2.3 - Módulo find







1.3 - Listagem da implementação

Apresenta-se, de seguida, a listagem da implementação ***findsx*** desenvolvida em linguagem *C* em ambiente *UNIX* numa estação de trabalho *SunSparc* e que se destina ao ambiente de processamento de imagem *XITE*[2], do qual utiliza algumas funções e tipos de variáveis já previamente definidas. Esta implementação contém comentários para auxílio à sua compreensão, assim como informação sobre os diferentes módulos, para sua posterior extracção através da ferramenta *CDOC(1)* existente no *XITE*.

```
/*
_____________________________________________________________________

findsx.c
@(#)findsx.c   1.00 94/01/14, Copyright 1994, DEEC, FEUP
Departamento de Engenharia Electrotecnica e de Computadores
Faculdade de Engenharia
Rua dos Bragas
4099 PORTO CODEX
PORTUGAL
E-mail: gpai@obelix.fe.up.pt
_____________________________________________________________________

biff.h
@(#)biff.h   1.23 92/04/10, Copyright 1990, Blab, UiO
Image processing lab, Department of Informatics
University of Oslo
E-mail: blab@ifi.uio.no
_____________________________________________________________________

readarg.h
@(#)readarg.h   1.3 92/01/15, Copyright 1990, Blab, UiO
Image processing lab, Department of Informatics
University of Oslo
E-mail: blab@ifi.uio.no
_____________________________________________________________________

message.h
@(#)message.h   1.3 92/01/15, Copyright 1990, Blab, UiO
Image processing lab, Department of Informatics
```

[2] *X-based Image processing Tools an Environmemt*, [Lønnestad, 1992].

University of Oslo
E-mail: blab@ifi.uio.no

---

*/

```
/*****************************************************************************/

static char *SccsId = "@(#)findsx.c 1.00 94/01/14, DEEC, FEUP";

/*****************************************************************************/

/* INCLUDES */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <blab/biff.h>
#include <blab/readarg.h>
#include <blab/message.h>

/*****************************************************************************/

/* DEFINES */

#define NMAX 800            /* maximum number of line's points to be considered */
#define MPI 3.141592654     /* pi's value */

/*****************************************************************************/

/*F:dft*
```

---

dft

Name: dft - performs the discrete FOURIER transformation

Syntax:
```
| int dft(n, x, xreal, ximag)
| int n;
| double x[], xreal[], ximag[];
```

Description: 'dft' performs the discrete fourier transformation of 'n' samples.
The input array 'x[]' contain the values of samples.
The array 'xreal[]' and the array 'yreal[]' contain the real and the imaginary part of the output results.

Return value: | 0 => Ok.

Example: | Try yourself.

Restrictions: None.

Author: Joao Tavares

Id: @(#)dft.c 1.00 94/01/10

```
*/

int dft(n, x, xreal, ximag)
int n;
double x[], xreal[], ximag[];

{

  register int i, k;
  double w0, angle;

  w0 = 2.0*MPI/n;

  for (k = 0; k < n; ++k) {

    for (i = 0; i < n; ++i) {

      angle = k*w0*i;
      xreal[k] += x[i]*cos(angle)/n;
      ximag[k] -= x[i]*sin(angle)/n;
```

```
   }

  }

  return(0);

}

/*************************************************************************/

/*F:find*
_______________________________________________________________________

  find

_______________________________________________________________________

Name:  find - compute the horizontal uncertainty factor of a discrete array camera

Syntax:          | double find(band, nl, nc, nz, ncx, nfx)
                 | IBAND band;
                 | int nl, nc, nz;
                 | double ncx, nfx;

Description:  'find' compute the horizontal uncertainty factor by the fourier analysis
              over the central 'nc' columns of the 'nl' lines in the input 'band'.
              The 'band' must be obtain without lowpass filter and a diffusing
              lens cap on the camera.
              'nz' is the number of zeros to padding before taking the DFT for the
              interpolation.
              'ncx' is the number of sensor elements in x (scan line) direction or
              pel clock frequency of the camera.
              'nfx' is the number of picture elements in a line as sampled by the
              computer or the frequency of the A/D-converter.

Return value:  | The horizontal uncertainty factor 'sx'.

Example:       | Try yourself.

Restrictions:  The value of 'nl' must be odd and smaller than ysize of input 'band'.
               The value of 'nc' must be odd and smaller than xsize of input 'band'.
               The value of 'nfx' must be different from zero.
               The input band must have UNS_BYTE pixels.

Author:        Joao Tavares

Id:            @(#)find.c   1.00   94/01/14
_______________________________________________________________________
*/

double find(band, nl, nc, nz, ncx, nfx)
IBAND band;
int nl, nc, nz;
double ncx, nfx;

{

 register int i, j;
 int  n, xsize, ysize, cx, cy, aux1, aux2, fsc;
 double sx, fs, s[NMAX], sreal[NMAX], simag[NMAX], as[NMAX], scale, max;

 /* initialisation of some variables */

 for (i = 0; i < NMAX; ++i) s[i] = sreal[i] = simag[i] = as[i] = 0.0;
 xsize = Ixsize(band);
 ysize = Iysize(band);
 n = nc+nz;
 scale = n/nc;
 cx = xsize/2;
 cy = ysize/2;
 aux1 = (nl-1)/2;
 aux2 = (nc-1)/2;

 /* compute an approximation of sx and fs */

 sx = ncx/nfx;

 if (sx < 1.0) fs = 1.0-sx;
 else fs = sx-1.0;

 /* compute the average of lines */

 for (i = cx-aux2; i <= cx+aux2; ++i) {

  for (j = cy-aux1; j <= cy+aux1; ++j) s[i] += band[j][i];
  s[i] /= nl;

 }

 /* call dft function */
```

```
  dft(n, s, sreal, simag);

  /* compute the amplitude spectrum */

  for (i = 0; i < n ; ++i) as[i] = sqrt(sreal[i]*sreal[i]+simag[i]*simag[i]);

  /* compute the real sx and fs */

  fsc = fs*scale;
  max = as[fsc];

  for (i = fsc-10; i <= fsc+10; ++i) {

    if (i < 0 || i >= n) continue;
    if (as[i] > max) {

      max = as[i];
      fsc = i;

    }

  }

  fs = fsc/scale;

  if (sx < 1.0) sx = 1.0-fs;
  else sx = 1.0+fs;

  return(sx);

}

/*****************************************************************************/

/*P:findsx*
_____________________________________________________________________________

  findsx
_____________________________________________________________________________

Name:  findsx - compute the horizontal uncertainty factor of a discrete array camera

Syntax:      | findsx [-f <filename>] [-l <nlines>] [-c <ncolumns>] [-z <nzeros>]
             | <Ncx/fc> <Nfx/ff> <input>

Description:  'findsx' compute the horizontal uncertainty factor by the fourier analysis
              over the central 'ncolumns' columns of the 'nlines' lines in the
              'input' image.
              The 'input' image must be obtain without lowpass filter and a
              diffusing lens cap on the camera.
              The output results can be writing in the file 'filename' if the flag
              '-f' is on, by default this flag is off.
              The number of lines to be considered is indicate through flag '-l',
              by default 'nlines' is 19.
              The number of columns to be considered is indicate through flag
              '-c', by default 'ncolumns' is 19.
              'nzeros' is the number of zeros to padding before taking the DFT for
              the interpolation, and is indicate through flag '-z'. By default
              'nzeros' is 60.
              'Ncx/ff' is the number of sensor elements in x (scan line) direction
              or pel clock frequency of the camera.
              'Nfx/ff' is the number of picture elements in a line as sampled by
              the computer or the frequency of the A/D-converter.

Return value:  | 1 => Wrong number of lines; it must be odd.
               | 2 => Wrong number of columns; it must be odd.
               | 3 => Nfx or ff must be different from zero.
               | 4 => Can't read image <input>.
               | 5 => The <input> has more than one band.
               | 6 => Bad input pixel type.
               | 7 => Wrong value for the number of lines; it must be smaller than
               |       ysize of input image.
               | 8 => Wrong value for the number of columns; it must be smaller
               |       than xsize of input image.
               | 9 => Can't open file <filename>.

Examples:     | Try yourself.

Restrictions:  Accepts only UNS_BYTE pixels.
               Images with only one band.
               The number of lines 'nl' must be odd and smaller ysize of 'input'
               image.
               The number of columns 'nc' must be odd and smaller xsize of
               'input' image.
               The value for 'Nfx/ff' must be different from zero.

Author:       Joao Tavares

Id:           @(#)findsx.c   1.0   94/01/14
```

```
_____________________________________________________________
*/

#ifdef MAIN

main(argc, argv)
int argc;
char **argv;

{

  IMAGE img;
  int nl, nc, nz, xsize, ysize;
  double ncx, nfx, sx;
  char *filename, arg[200];
  FILE *fpt;

  InitMessage(&argc, argv, "Usage: %s\n [-f <filename>] [-l <nlines>] [-c <ncolumns>] [-z
<nzeros>] <Ncx/fc> <Nfx/ff> <input>\n");
  Iset_message(1);
  Iset_abort(1);

  /* read switches */

  filename = read_switch(&argc, argv, "-f", 1, NULL);
  nl = atoi(read_switch(&argc, argv, "-l", 1, "19"));
  nc = atoi(read_switch(&argc, argv, "-c", 1, "19"));
  nz = atoi(read_switch(&argc, argv, "-z", 1, "60"));

  if (argc == 1) Usage(1, (char*)0);
  if (argc != 4) Usage(2, "\nBad number of arguments.\n");
  if (nl%2 == 0) exit(Error(1, "\nWrong number of lines; it must be odd.\n"));
  if (nc%2 == 0) exit(Error(2, "\nWrong number of columns; it must be odd.\n"));

  /* read argv[] */

  ncx = atof(argv[1]);
  nfx = atof(argv[2]);

  if (nfx == 0.0) exit(Error(3, "\nNfx or ff must be different from zero.\n"));

  /* read input image */

  img = Iread_image(argv[3]);
  if (img == NULL) exit(Error(4, "\nCan't read image %s.\n", argv[3]));
  if (Inbands(img) != 1) exit(Error(5, "\nThe %s has more than one band.\n", argv[3]));
  if (Ipixtyp(img[1]) != Iu_byte_typ) exit(Error(6, "\nBad input pixel type.\n"));
  xsize = Ixsize(img[1]);
  ysize = Iysize(img[1]);

  if (ysize < nl) exit(Error(7, "\nWrong value for the number of lines; it must be smaller than ysize
of input image.\n"));
  if (xsize < nc) exit(Error(8, "\nWrong value for the number of columns; it must be smaller than
xsize of input image.\n"));

  /* call find function */

  sx = find(img[1], nl, nc, nz, ncx, nfx);

  /* output results */

  sprintf(arg, "\nResults for %s in %s with Number of lines: %i, Number of columns: %i, Number
of zeros: %i, Ncx or fc: %g, Nfx or ff: %g\n\n\tsx=%g", argv[0], argv[3], nl, nc, nz, ncx, nfx, sx);
  printf(arg);

  /* open file if it was desire, write arg and close it */

  if (filename != NULL) {

    fpt = fopen(filename, "a");
    if (fpt == NULL) exit(Error(9, "\nCan't open file %s.\n", filename));
    fprintf(fpt, arg);
    close(fpt);

  }

}

#endif

/*****************************************************************************/
```

2 - Calibração de uma dada câmara: calcamera

Nesta secção é apresentada uma implementação para calibração de uma dada câmara, segundo o método descrito em [Tavares, 1995]. Com esta implementação, é possível a calibração de todos os parâmetros de uma câmara, excepto o factor de incerteza horizontal.

2.1 - Descrição dos diferentes módulos

A implementação é constituída pelos módulos: ***principal***, ***linleastsquar***, ***solve*** e ***findcenter***, *Fig. 2*. Apresenta-se de seguida a descrição de cada um destes módulos.

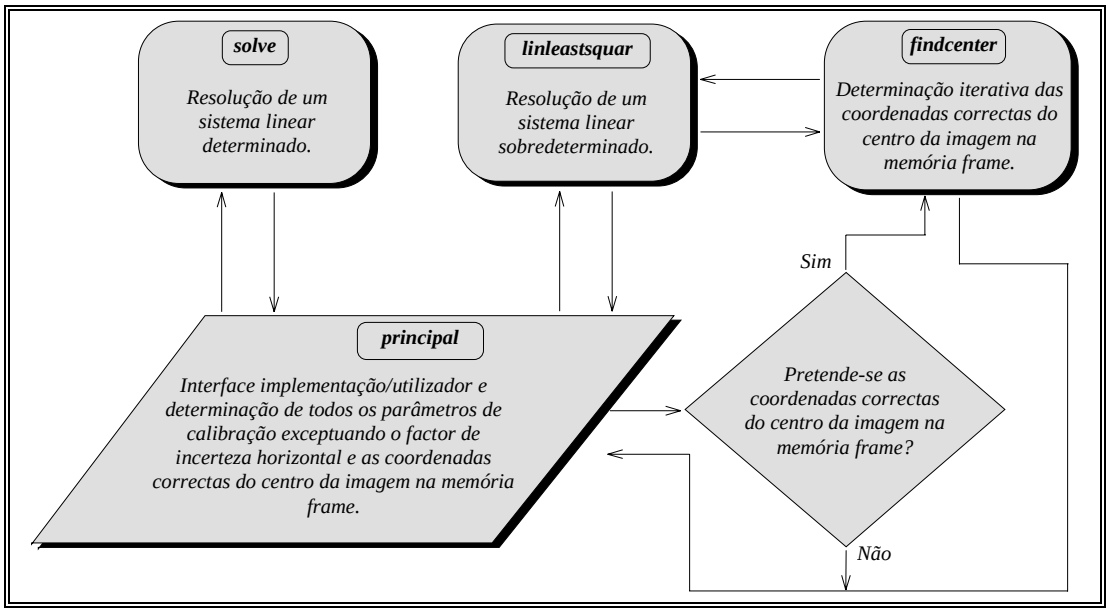


*Fig. 2 - Módulos integrantes da implementação **calcamera** e suas relações.*

2.1.1 - Módulo principal

Este módulo é responsável pela interface implementação/utilizador e pela calibração de todos os parâmetros de uma câmara, excepto o factor de incerteza horizontal e as coordenadas do centro da imagem na memória *frame*. A seu cargo está:

- ✓ Leitura dos argumentos opcionais ("*switches*"):
  - *outfilename*, ("[-f <outfilename>]"), nome do ficheiro para escrita dos resultados obtidos, nomeadamente as condições da realização da calibração e os parâmetros obtidos. Por defeito é "*NULL*", o que implica a não abertura do ficheiro. Quando especificado, o ficheiro, se ainda não existir, é criado; caso contrário, os resultados são acrescentados ao ficheiro já existente.
  - *nlstop*, ("[-n <nlstop>]"), tolerância de erro para as soluções exactas: da distância focal efectiva *f*, da componente t_z do vector de translação, e do factor de distorção radial da lente k_1, determinadas pela resolução do sistema de equações não linear respectivo, pelo método Levenberg-Marquardt[3]. Por defeito, é igual a *0.0000001*.
  - *sx*, ("[-s <sx>]"), factor de incerteza horizontal s_x. Por defeito, é igual a *0.710935*.

[3] Ver, por exemplo [Press, 1992].

- *findcenter*, ("[-c <findcenter>]"); quando *findcenter* é especificado pelo utilizador como igual a *1*, a implementação determina as coordenadas exactas do centro da imagem na memória *frame*, (C_x, C_y). Por defeito, esta determinação não é realizada.

✓ Leitura dos argumentos passados à implementação pela linha de comando:

- *dx*, ("<dx>"), distância entre centros dos elementos sensores vizinhos na direcção *x*, d_x.
- *dy*, ("<dy>"), distância entre centros dos sensores CCD vizinhos na direcção *y*, d_y.
- *Cx*, ("<Cx>"), coordenada do centro da imagem na memória *frame* segundo a direcção *x*, C_x. Quando C_x não é conhecido exactamente, o utilizador deve especificar C_x igual a metade da dimensão da memória *frame* segundo a direcção *x* e *findcenter* igual a *1*. Deste modo, o valor especificado para C_x é apenas a solução inicial, sendo a implementação responsável pela determinação da solução exacta para C_x.
- *Cy*, ("<Cy>"), coordenada do centro da imagem na memória *frame* segundo a direcção *y*, C_y. Quando C_y não é conhecido exactamente, o utilizador deve especificar C_y igual a metade da dimensão da memória *frame* segundo a direcção *y* e *findcenter* igual a *1*. Deste modo, o valor especificado para C_y é apenas a solução inicial, sendo a implementação responsável pela determinação da solução exacta para C_y.
- *infilename*, ("<infilename>"), nome do ficheiro de entrada que contém as coordenadas 3D mundo $(x_{w_i}, y_{w_i}, z_{w_i})$ e as coordenadas na memória *frame* (X_{f_i}, Y_{f_i}), para todos pontos de calibração *i*.

✓ Abertura do ficheiro de entrada.

✓ Leitura do ficheiro de entrada das coordenadas 3D mundo $(x_{w_i}, y_{w_i}, z_{w_i})$ e das coordenadas na memória *frame* (X_{f_i}, Y_{f_i}), para todos pontos de calibração *i*.

✓ Fecho do ficheiro de entrada.

✓ Determinação das coordenadas do centro da imagem na memória *frame* (X_{f_i}, Y_{f_i}) quando pretendido pelo utilizador; para tal, este módulo utiliza o módulo ***findcenter***.

✓ Calibração de todos os restantes parâmetros de uma câmara, excepto o factor de incerteza horizontal; para tal, este módulo utiliza os módulos ***linleastsquar*** e ***solve***.

✓ Apresentação dos resultados obtidos.

✓ Abertura do ficheiro de saída de resultados, escrita dos resultados obtidos e respectivo fecho, se desejado pelo utilizador.

✓ Comunicação da existência de erros em argumentos de entrada ou ao longo do decurso da execução:

- Através de valores de retorno:
  - 1, ("Can't open file <infilename>."), o módulo não consegue abrir o ficheiro de entrada especificado.
  - 2, ("The point(n) doesn't have the world coordinate z equal to zero."), o ponto de calibração especificado não tem a coordenada 3D mundo *z* igual a zero.
  - 3, ("The number of calibration points must be larger than 5."), o número de pontos de calibração tem de ser maior do que cinco.
  - 4, ("The horizontal uncertainty factor (sx) must be different from zero."), o factor de incerteza horizontal tem de ser diferente de zero.

- 5, ("Can't open file <outfilename>."), o módulo não consegue abrir o ficheiro de saída especificado.

- Através de mensagens:
  - 1, (*InitMessage*), quando o número de argumentos é igual a um, isto é, apenas o nome da implementação.
  - 2, ("Bad number of arguments."), quando o número de argumentos é diferente de dez.

Como restrições na utilização deste módulo têm-se:

- O número de pontos de calibração deve ser maior do que cinco.
- O sistema de coordenadas 3D mundo deve ser escolhido de forma a que todos os pontos de calibração tenham a coordenada 3D mundo *z* igual a zero, e a que a componente t_y do vector de translação não seja exactamente igual a zero.
- O factor de incerteza horizontal deve ser diferente de zero.

2.1.2 - Módulo linleastsquar

Este módulo é responsável pela resolução de um sistema sobredeterminado de equações lineares, $z[m][n] \cdot a[n] = y[m]$. O método de resolução utilizado é o dos mínimos quadrados com resolução do respectivo sistema de equações normais. Para tal, utiliza a decomposição de Cholesky[4] da matriz $[z[m][n]^T z[m][n]]$, com posterior substituição para a frente e para trás. Esta decomposição é das mais eficientes em termos de memória consumida; no entanto, a sua utilização só é possível em matrizes definidas positivamente.

Como parâmetros de entrada para este módulo têm-se:

- *m*, do tipo inteiro, número de amostras disponíveis do modelo.
- *n*, do tipo inteiro, número de incógnitas a determinar.
- *z[ ][ ]*, do tipo real, matriz com os valores observados para as variáveis independentes.
- *a[ ]*, do tipo real, vector das incógnitas a determinar.
- *y[ ]*, do tipo real, vector dos valores observados para as variáveis dependentes.

Este módulo tem como única restrição à sua utilização a necessidade de a matriz $[z[m][n]^T z[m][n]]$ ser definida positivamente, e apresenta como valores de retorno:

- 0, a execução do módulo decorreu com normalidade.
- 1, 2, ("Can't make CHOLESKY decomposition."), a matriz $[z[m][n]^T z[m][n]]$ não é definida positivamente.

Neste módulo, é utilizada a variável definida globalmente *NMAX* para o dimensionamento de matrizes auxiliares, que deverá ser igual ao número de amostras disponíveis do modelo.

2.1.3 - Módulo solve

Este módulo é responsável pela resolução de um sistema de equações lineares $alfa[NMAX1][NMAX1] \times delta[NMAX1] = beta[NMAX1]$, onde *NMAX1* é uma variável definida globalmente. O método de resolução utilizado é a decomposição de Cholesky[4] da matriz *alfa*, com posterior substituição para a frente e para trás. Esta decomposição é das mais eficientes em termos de memória consumida; no entanto, a sua utilização só é possível em matrizes definidas positivamente.

Como parâmetros de entrada para este módulo têm-se:

[4] Ver, por exemplo [Chapra, 1988].

- *alfa[ ][ ],* do tipo real, matriz dos coeficientes.
- *delta[ ],* do tipo real, vector das incógnitas a determinar.
- *beta[ ],* do tipo real, vector das constantes.

Este módulo tem como única restrição à sua utilização a necessidade da matriz *alfa* ser definida positivamente, e apresenta como valores de retorno:

- 0, a execução do módulo decorreu com normalidade.
- 1, 2, ("Can't make CHOLESKY decomposition."), a matriz *alfa* não é definida positivamente.

Neste módulo, é utilizada a variável definida globalmente *NMAX* para o dimensionamento de matrizes auxiliares, que deverá ser igual ao número de incógnitas do sistema.

2.1.4 - Módulo findcenter

Este módulo é responsável pela determinação das coordenadas do centro da imagem na memória *frame*, segundo o método do alinhamento radial apresentado em [Tavares, 1995] e inicialmente proposto em [Lenz, 1988]. Durante a determinação, todos os restantes parâmetros de calibração são também determinados novamente, até o módulo atingir uma solução global óptima. Para tal, existem duas variáveis definidas globalmente: *STOP1* e *STOP2*, que servem de controlo à tolerância de erro da solução global encontrada. No fim da execução, as coordenadas determinadas como centro da imagem são arredondadas para o inteiro mais próximo, pois são coordenadas discretas da memória *frame*. Este módulo, durante a sua execução, utiliza o módulo ***linleastsquar*** descrito anteriormente.

Como parâmetros de entrada deste módulo têm-se:

- *n*, do tipo inteiro, número de pontos de calibração a considerar.
- *point[ ]*, do tipo estrutura *{float x_w; float y_w; float x_f; float y_f; double x_d; double y_d}*, sendo: $(x_w, y_w, 0)$ as coordenadas 3D mundo, (x_f, y_f) as coordenadas na memória *frame*, e (x_d, y_d) as coordenadas imagem distorcida, para todos os pontos de calibração.
- *dxl*, do tipo real, igual a $s_x d_x$, onde s_x é o factor de incerteza horizontal e d_x é a distância entre centros dos elementos sensores vizinhos na direcção *x*.
- *dy*, do tipo real, distância entre centros dos sensores CCD vizinhos na direcção *y*.
- *cx*, do tipo inteiro, coordenada do centro da imagem na memória *frame* segundo a direcção *x*, passado ao módulo por endereço.
- *cy*, do tipo inteiro, coordenada do centro da imagem na memória *frame* segundo a direcção *y*, passado ao módulo por endereço.

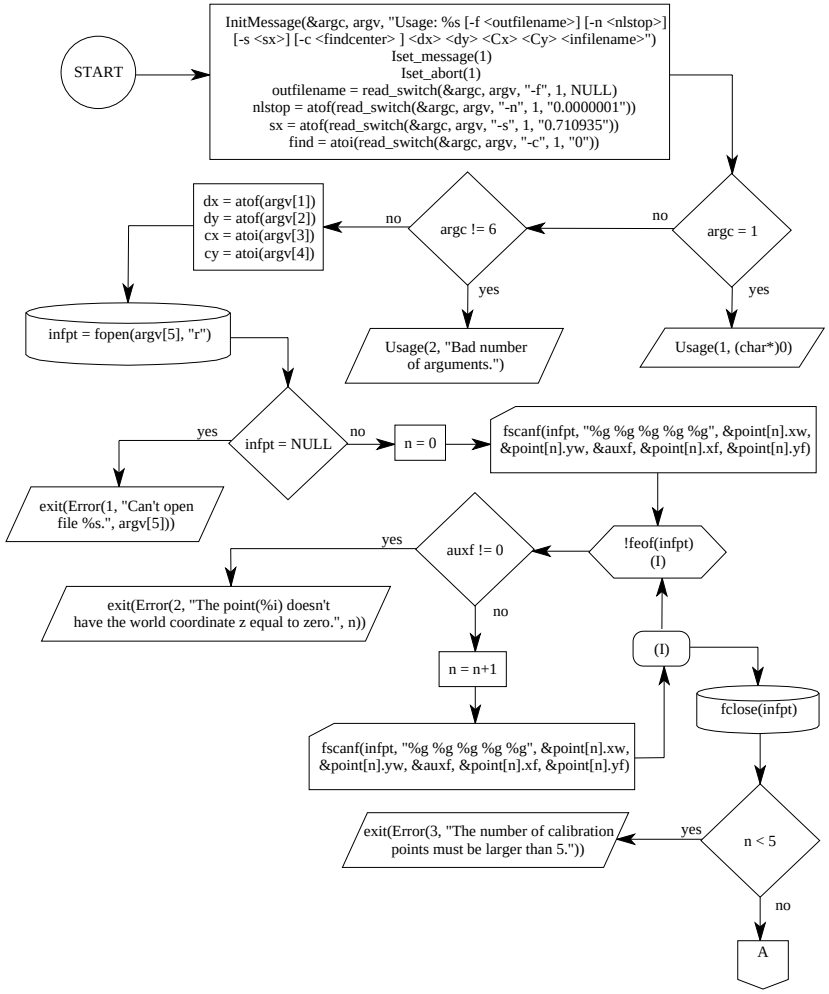
Este módulo retorna *0* como indicação de que a execução do módulo decorreu com normalidade, e apresenta como restrições quanto à sua utilização:

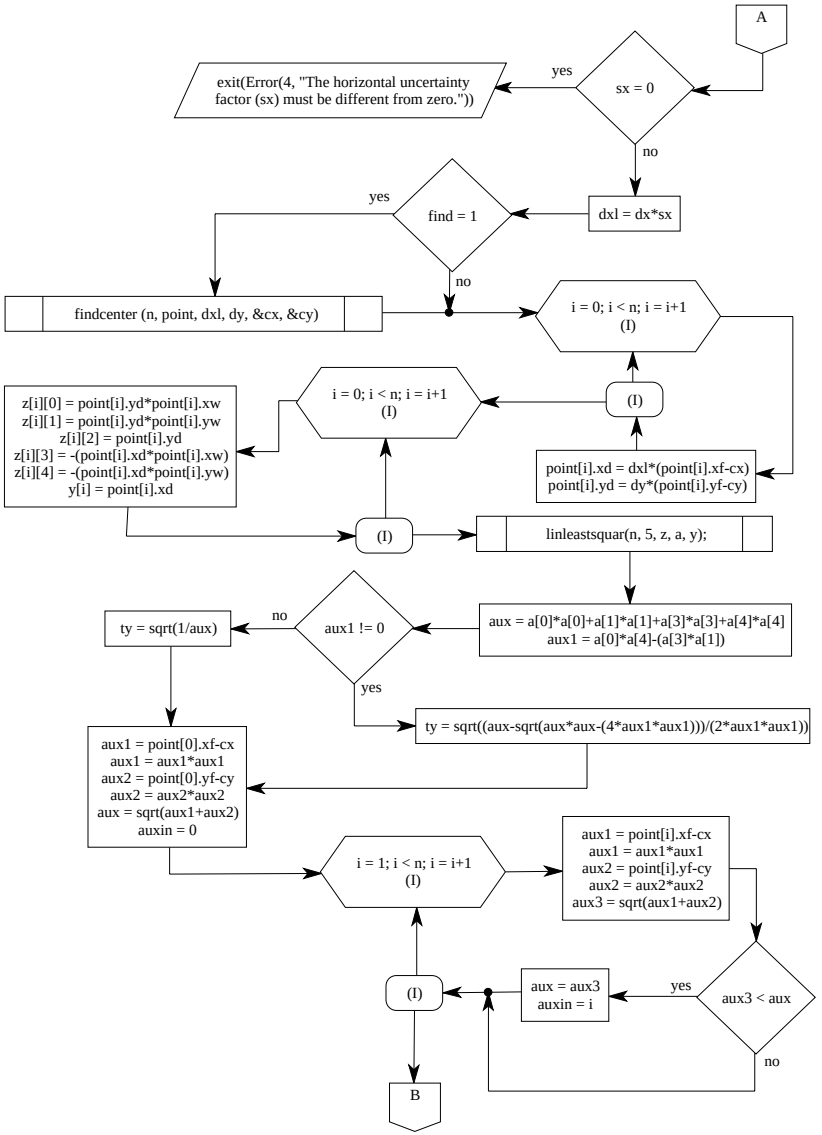
- O número de pontos de calibração deve ser maior do que cinco.
- O sistema de coordenadas 3D mundo deve ser escolhido de forma a que todos os pontos de calibração tenham a coordenada 3D mundo *z* igual a zero, e a que a componente t_y do vector de translação não seja exactamente igual a zero.
- O valor da distorção radial não pode ser demasiado pequeno, pois este método baseia-se na existência de distorção radial da lente.

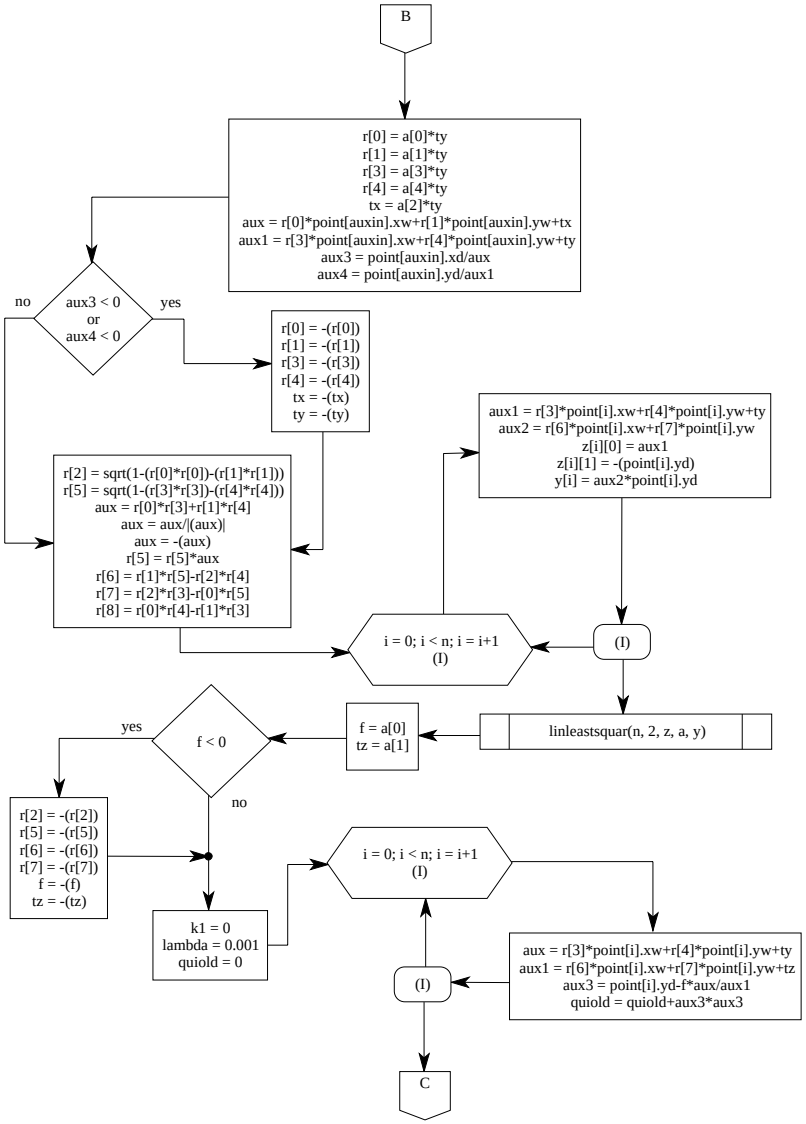
2.2 - Fluxogramas dos diferentes módulos

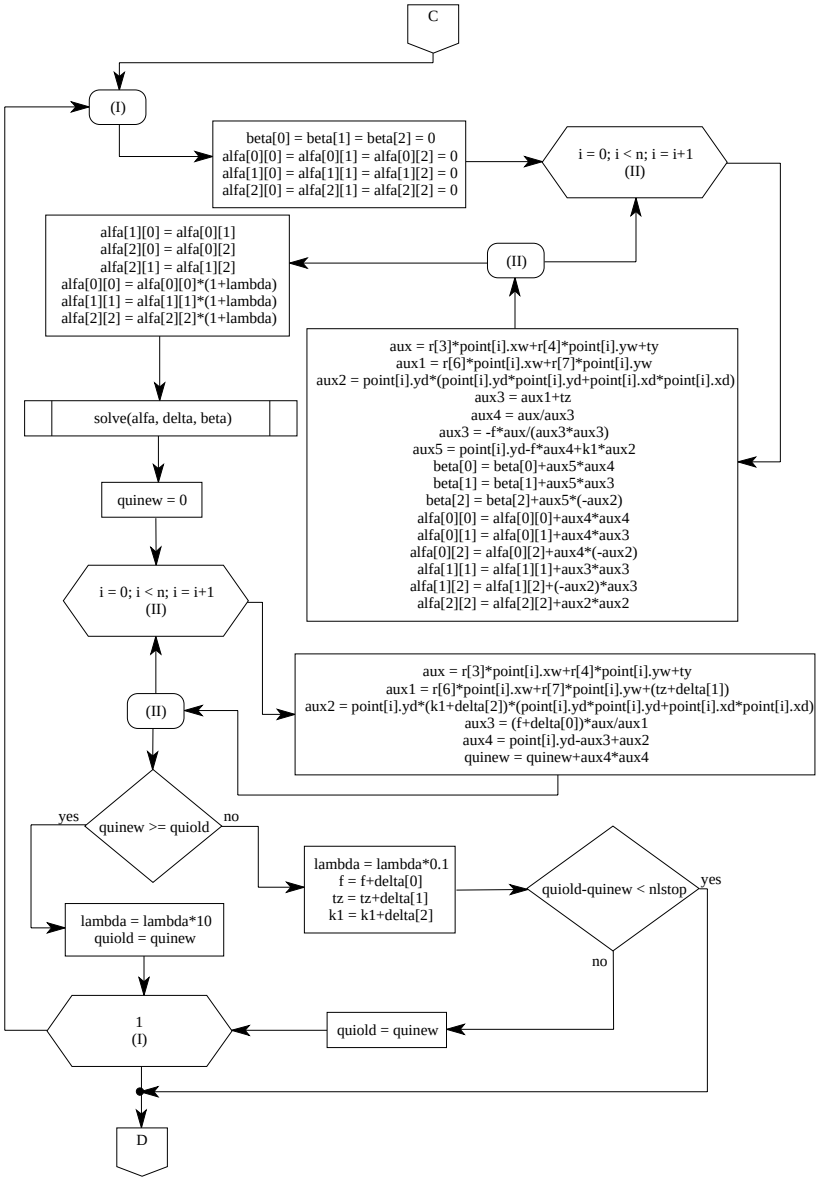
Apresentam-se, de seguida, os fluxogramas dos módulos que constituem a implementação ***calcamera***.

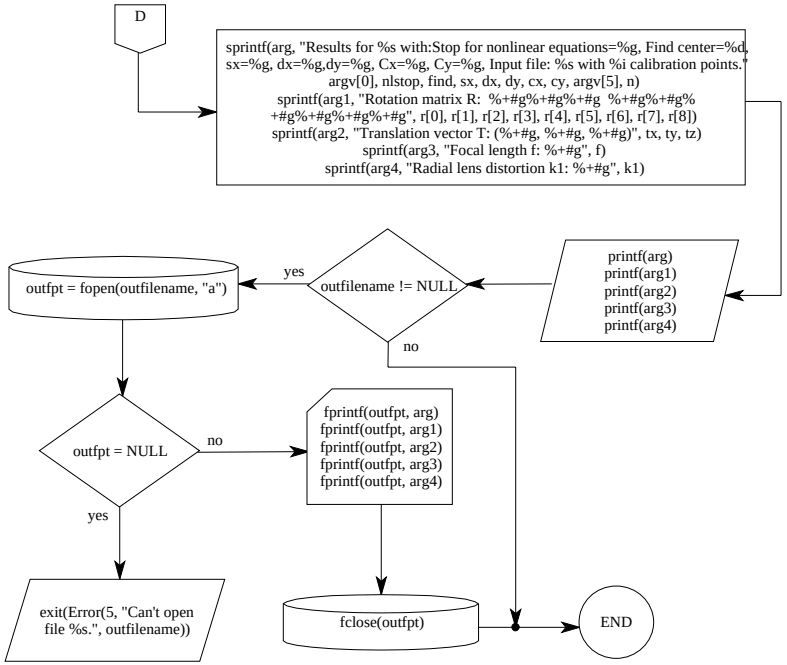
2.2.1 - Módulo principal



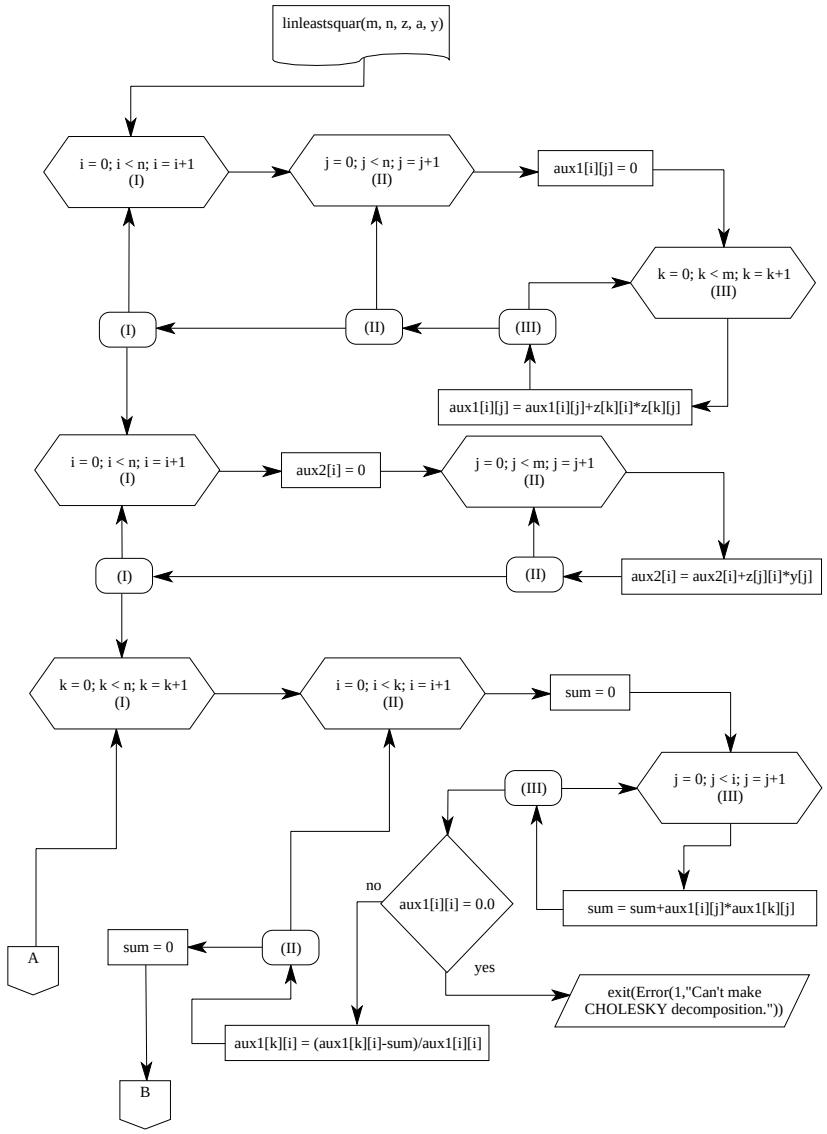


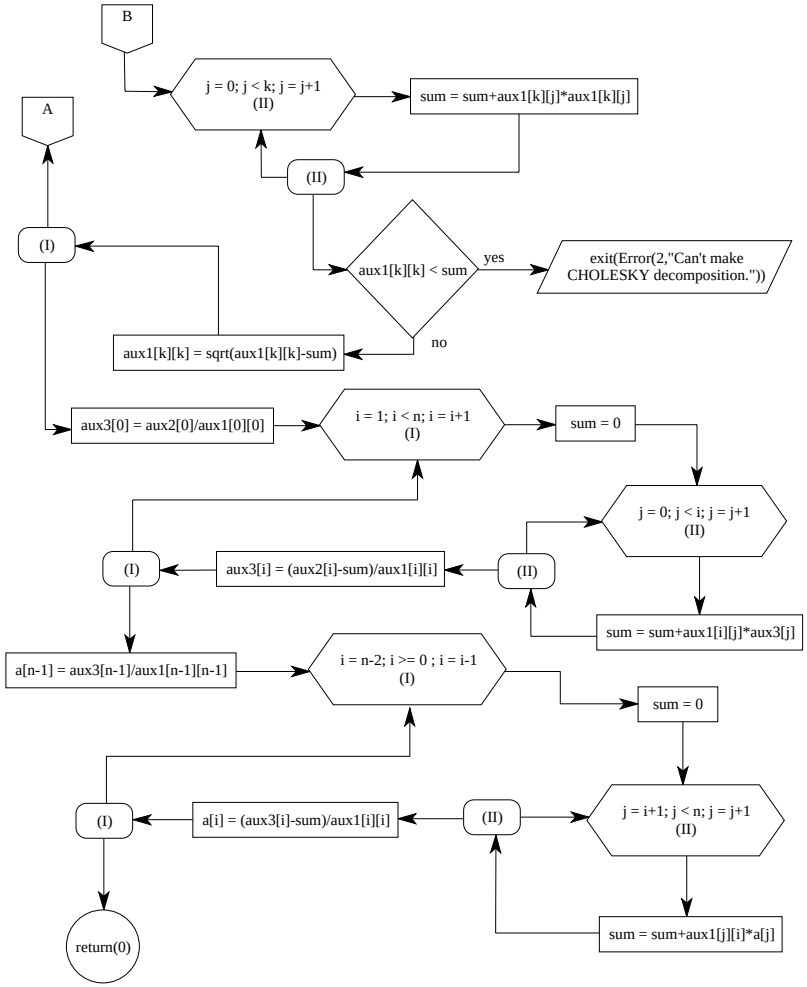

B
r[0] = a[0]*ty
r[1] = a[1]*ty
r[3] = a[3]*ty
r[4] = a[4]*ty
tx = a[2]*ty
aux = r[0]*point[auxin].xw+r[1]*point[auxin].yw+tx
aux1 = r[3]*point[auxin].xw+r[4]*point[auxin].yw+ty
aux3 = point[auxin].xd/aux
aux4 = point[auxin].yd/aux1
aux3 < 0 or aux4 < 0
no
yes
r[0] = -(r[0])
r[1] = -(r[1])
r[3] = -(r[3])
r[4] = -(r[4])
tx = -(tx)
ty = -(ty)
r[2] = sqrt(1-(r[0]*r[0])-(r[1]*r[1]))
r[5] = sqrt(1-(r[3]*r[3])-(r[4]*r[4]))
aux = r[0]*r[3]+r[1]*r[4]
aux = aux/|(aux)|
aux = -(aux)
r[5] = r[5]*aux
r[6] = r[1]*r[5]-r[2]*r[4]
r[7] = r[2]*r[3]-r[0]*r[5]
r[8] = r[0]*r[4]-r[1]*r[3]
aux1 = r[3]*point[i].xw+r[4]*point[i].yw+ty
aux2 = r[6]*point[i].xw+r[7]*point[i].yw
z[i][0] = aux1
z[i][1] = -(point[i].yd)
y[i] = aux2*point[i].yd
i = 0; i < n; i = i+1
(I)
(I)
linleastsquar(n, 2, z, a, y)
f = a[0]
tz = a[1]
f < 0
yes
no
r[2] = -(r[2])
r[5] = -(r[5])
r[6] = -(r[6])
r[7] = -(r[7])
f = -(f)
tz = -(tz)
k1 = 0
lambda = 0.001
quiold = 0
i = 0; i < n; i = i+1
(I)
aux = r[3]*point[i].xw+r[4]*point[i].yw+ty
aux1 = r[6]*point[i].xw+r[7]*point[i].yw+tz
aux3 = point[i].yd-f*aux/aux1
quiold = quiold+aux3*aux3
(I)
C


C
(I)
beta[0] = beta[1] = beta[2] = 0
alfa[0][0] = alfa[0][1] = alfa[0][2] = 0
alfa[1][0] = alfa[1][1] = alfa[1][2] = 0
alfa[2][0] = alfa[2][1] = alfa[2][2] = 0
i = 0; i < n; i = i+1
(II)
alfa[1][0] = alfa[0][1]
alfa[2][0] = alfa[0][2]
alfa[2][1] = alfa[1][2]
alfa[0][0] = alfa[0][0]*(1+lambda)
alfa[1][1] = alfa[1][1]*(1+lambda)
alfa[2][2] = alfa[2][2]*(1+lambda)
(II)
aux = r[3]*point[i].xw+r[4]*point[i].yw+ty
aux1 = r[6]*point[i].xw+r[7]*point[i].yw
aux2 = point[i].yd*(point[i].yd*point[i].yd+point[i].xd*point[i].xd)
aux3 = aux1+tz
aux4 = aux/aux3
aux3 = -f*aux/(aux3*aux3)
aux5 = point[i].yd-f*aux4+k1*aux2
beta[0] = beta[0]+aux5*aux4
beta[1] = beta[1]+aux5*aux3
beta[2] = beta[2]+aux5*(-aux2)
alfa[0][0] = alfa[0][0]+aux4*aux4
alfa[0][1] = alfa[0][1]+aux4*aux3
alfa[0][2] = alfa[0][2]+aux4*(-aux2)
alfa[1][1] = alfa[1][1]+aux3*aux3
alfa[1][2] = alfa[1][2]+(-aux2)*aux3
alfa[2][2] = alfa[2][2]+aux2*aux2
solve(alfa, delta, beta)
quinew = 0
i = 0; i < n; i = i+1
(II)
(II)
aux = r[3]*point[i].xw+r[4]*point[i].yw+ty
aux1 = r[6]*point[i].xw+r[7]*point[i].yw+(tz+delta[1])
aux2 = point[i].yd*(k1+delta[2])*(point[i].yd*point[i].yd+point[i].xd*point[i].xd)
aux3 = (f+delta[0])*aux/aux1
aux4 = point[i].yd-aux3+aux2
quinew = quinew+aux4*aux4
quinew >= quiold
yes
no
lambda = lambda*0.1
f = f+delta[0]
tz = tz+delta[1]
k1 = k1+delta[2]
quiold-quinew < nlstop
yes
no
lambda = lambda*10
quiold = quinew
1
(I)
quiold = quinew
D

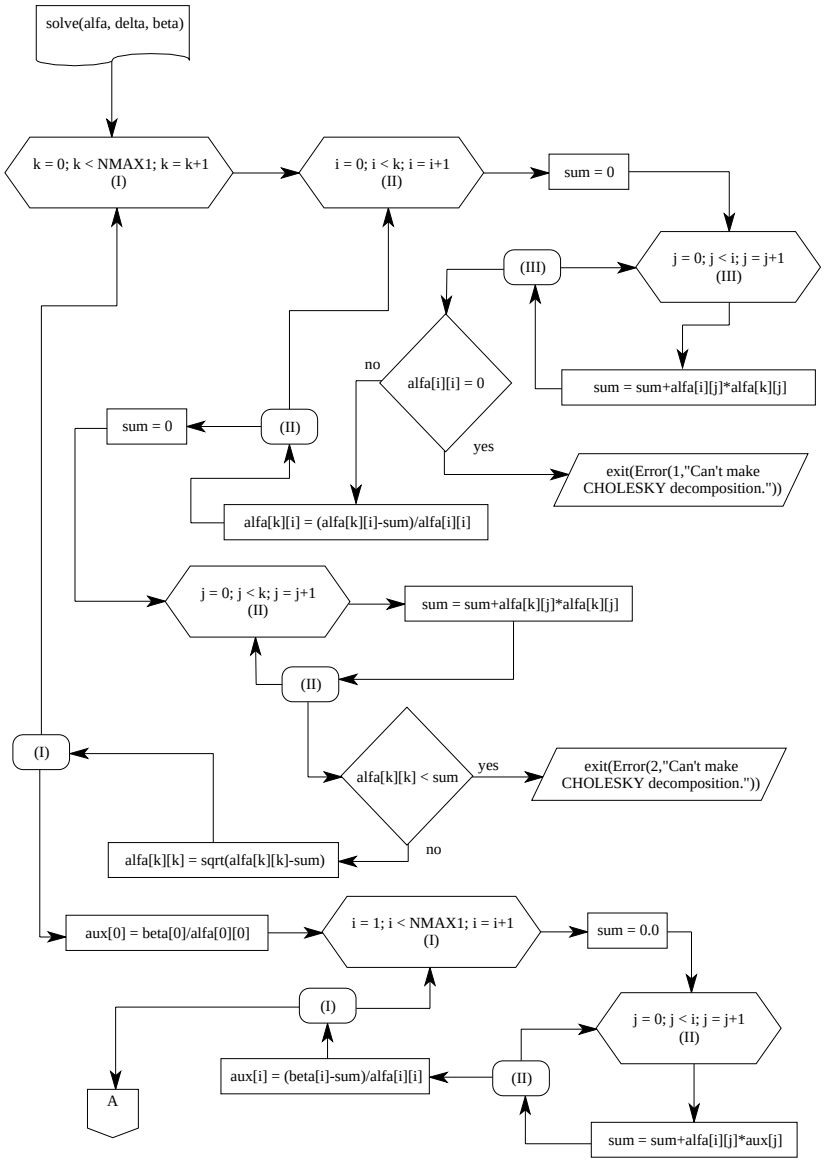


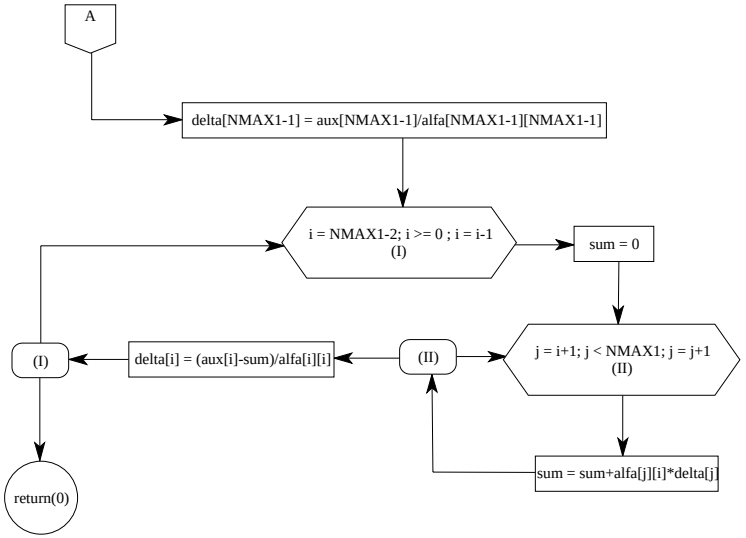
2.2.2 - Módulo linleastsquar



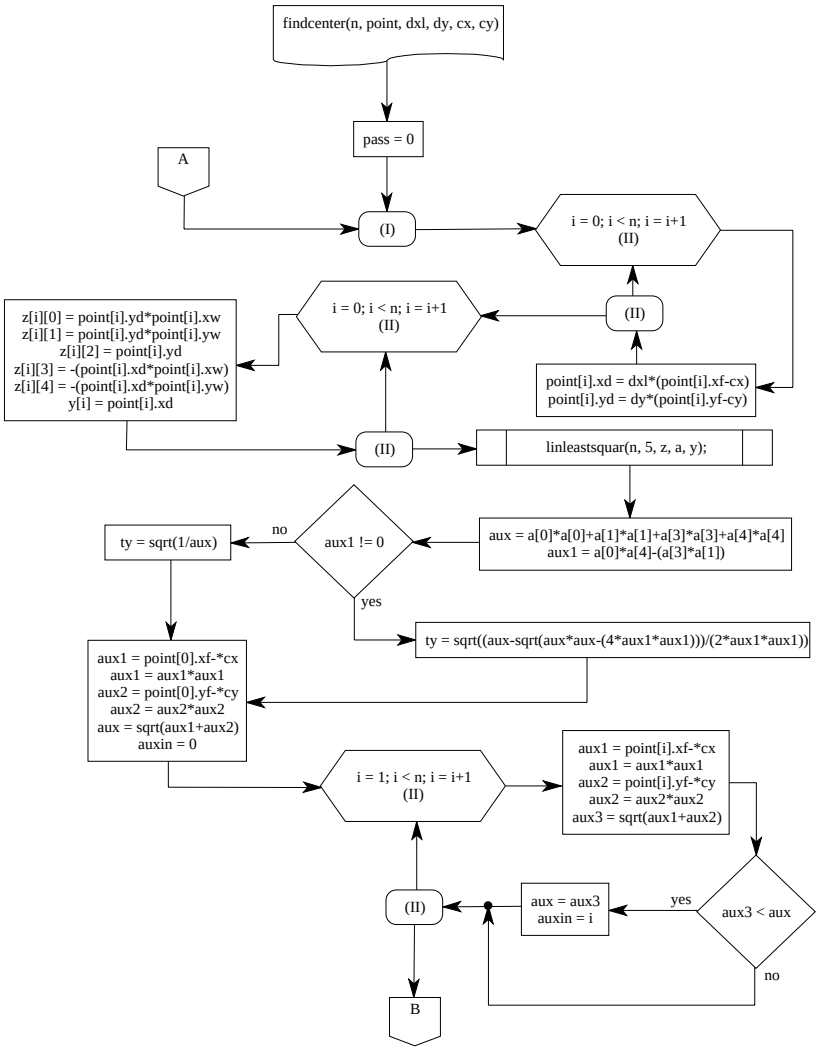


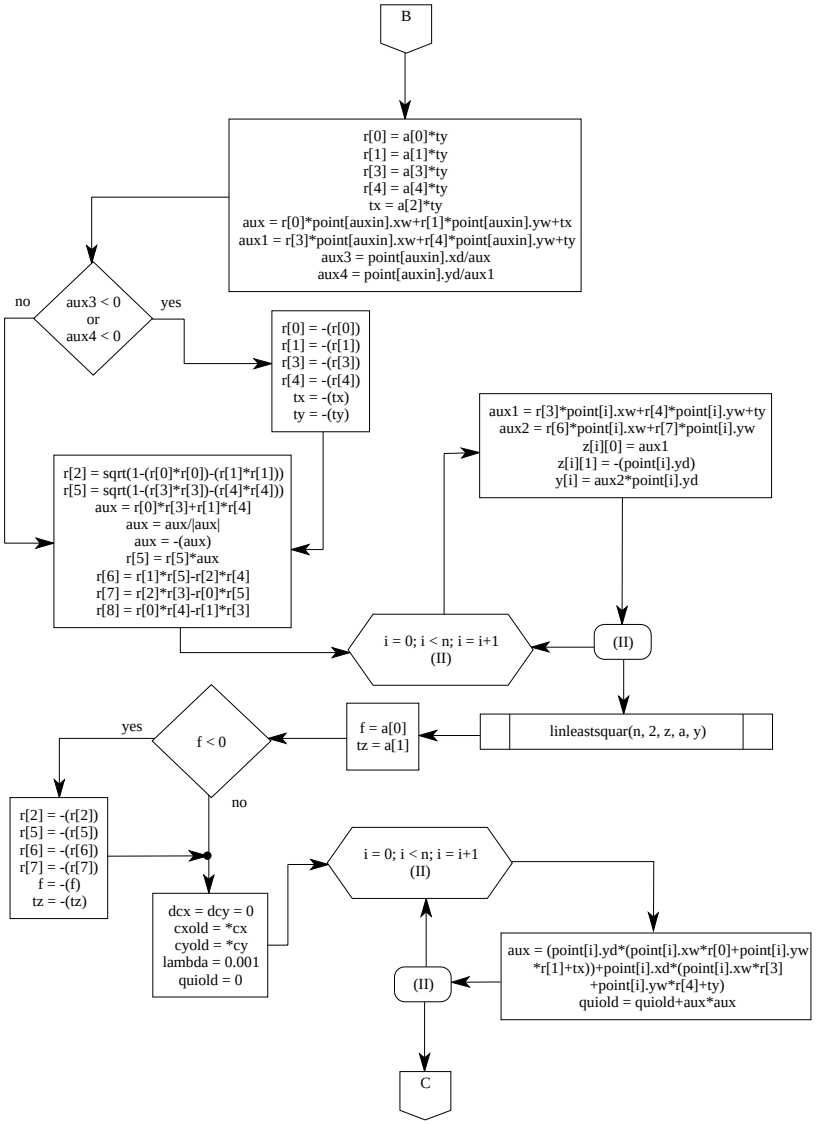
2.2.3 - Módulo solve

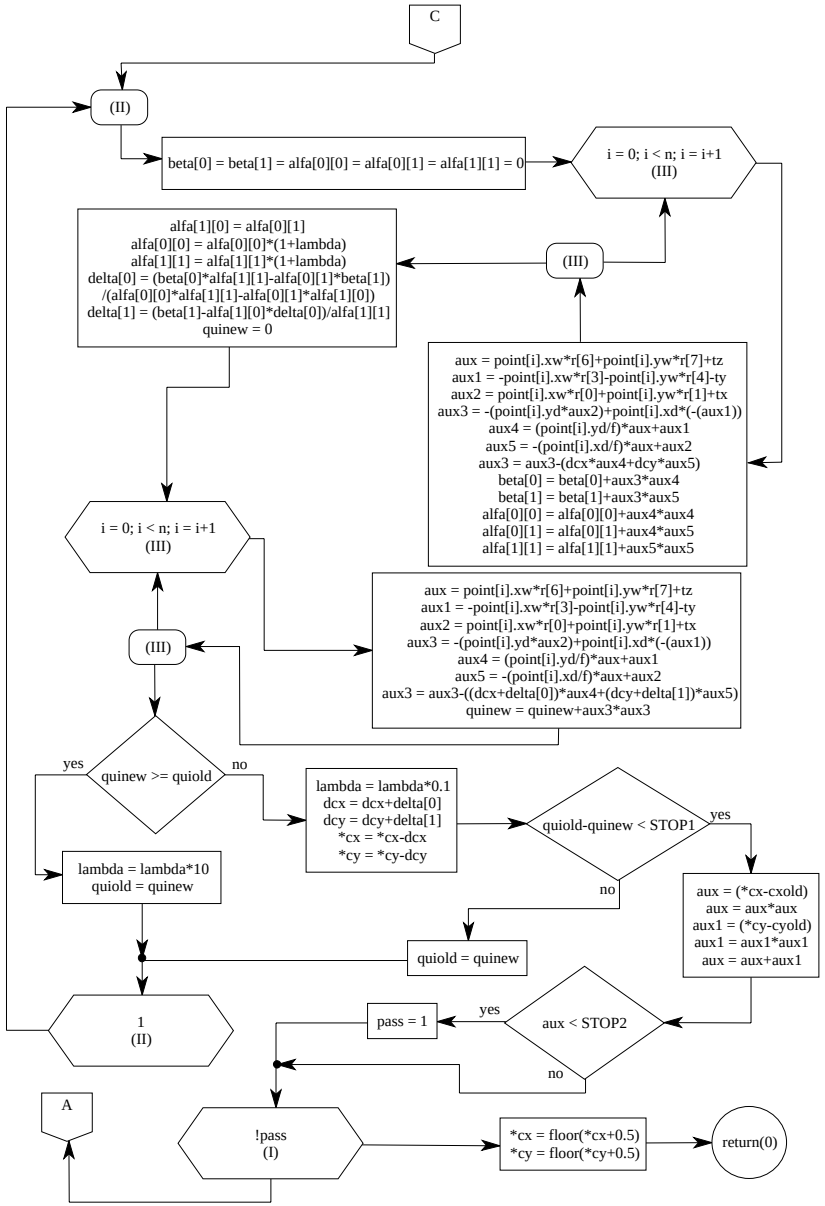




2.2.4 - Módulo findcenter




B
r[0] = a[0]*ty
r[1] = a[1]*ty
r[3] = a[3]*ty
r[4] = a[4]*ty
tx = a[2]*ty
aux = r[0]*point[auxin].xw+r[1]*point[auxin].yw+tx
aux1 = r[3]*point[auxin].xw+r[4]*point[auxin].yw+ty
aux3 = point[auxin].xd/aux
aux4 = point[auxin].yd/aux1
aux3 < 0 or aux4 < 0
no
yes
r[0] = -(r[0])
r[1] = -(r[1])
r[3] = -(r[3])
r[4] = -(r[4])
tx = -(tx)
ty = -(ty)
r[2] = sqrt(1-(r[0]*r[0])-(r[1]*r[1]))
r[5] = sqrt(1-(r[3]*r[3])-(r[4]*r[4]))
aux = r[0]*r[3]+r[1]*r[4]
aux = aux/|aux|
aux = -(aux)
r[5] = r[5]*aux
r[6] = r[1]*r[5]-r[2]*r[4]
r[7] = r[2]*r[3]-r[0]*r[5]
r[8] = r[0]*r[4]-r[1]*r[3]
aux1 = r[3]*point[i].xw+r[4]*point[i].yw+ty
aux2 = r[6]*point[i].xw+r[7]*point[i].yw
z[i][0] = aux1
z[i][1] = -(point[i].yd)
y[i] = aux2*point[i].yd
i = 0; i < n; i = i+1
(II)
(II)
linleastsquar(n, 2, z, a, y)
f = a[0]
tz = a[1]
f < 0
yes
no
r[2] = -(r[2])
r[5] = -(r[5])
r[6] = -(r[6])
r[7] = -(r[7])
f = -(f)
tz = -(tz)
dcx = dcy = 0
cxold = *cx
cyold = *cy
lambda = 0.001
quiold = 0
i = 0; i < n; i = i+1
(II)
(II)
aux = (point[i].yd*(point[i].xw*r[0]+point[i].yw
r[1]+tx))+point[i].xd(point[i].xw*r[3]
+point[i].yw*r[4]+ty)
quiold = quiold+aux*aux
C


C
(II)
beta[0] = beta[1] = alfa[0][0] = alfa[0][1] = alfa[1][1] = 0
i = 0; i < n; i = i+1
(III)
alfa[1][0] = alfa[0][1]
alfa[0][0] = alfa[0][0]*(1+lambda)
alfa[1][1] = alfa[1][1]*(1+lambda)
delta[0] = (beta[0]*alfa[1][1]-alfa[0][1]*beta[1])
/(alfa[0][0]*alfa[1][1]-alfa[0][1]*alfa[1][0])
delta[1] = (beta[1]-alfa[1][0]*delta[0])/alfa[1][1]
quinew = 0
(III)
aux = point[i].xw*r[6]+point[i].yw*r[7]+tz
aux1 = -point[i].xw*r[3]-point[i].yw*r[4]-ty
aux2 = point[i].xw*r[0]+point[i].yw*r[1]+tx
aux3 = -(point[i].yd*aux2)+point[i].xd*(-(aux1))
aux4 = (point[i].yd/f)*aux+aux1
aux5 = -(point[i].xd/f)*aux+aux2
aux3 = aux3-(dcx*aux4+dcy*aux5)
beta[0] = beta[0]+aux3*aux4
beta[1] = beta[1]+aux3*aux5
alfa[0][0] = alfa[0][0]+aux4*aux4
alfa[0][1] = alfa[0][1]+aux4*aux5
alfa[1][1] = alfa[1][1]+aux5*aux5
i = 0; i < n; i = i+1
(III)
(III)
aux = point[i].xw*r[6]+point[i].yw*r[7]+tz
aux1 = -point[i].xw*r[3]-point[i].yw*r[4]-ty
aux2 = point[i].xw*r[0]+point[i].yw*r[1]+tx
aux3 = -(point[i].yd*aux2)+point[i].xd*(-(aux1))
aux4 = (point[i].yd/f)*aux+aux1
aux5 = -(point[i].xd/f)*aux+aux2
aux3 = aux3-((dcx+delta[0])*aux4+(dcy+delta[1])*aux5)
quinew = quinew+aux3*aux3
quinew >= quiold
yes
no
lambda = lambda*0.1
dcx = dcx+delta[0]
dcy = dcy+delta[1]
*cx = *cx-dcx
*cy = *cy-dcy
quiold-quinew < STOP1
yes
no
lambda = lambda*10
quiold = quinew
aux = (*cx-cxold)
aux = aux*aux
aux1 = (*cy-cyold)
aux1 = aux1*aux1
aux = aux+aux1
quiold = quinew
1
(II)
pass = 1
aux < STOP2
yes
no
A
!pass
(I)
*cx = floor(*cx+0.5)
*cy = floor(*cy+0.5)
return(0)

2.3 - Listagem da implementação

Apresenta-se, de seguida, a listagem da implementação ***calcamera*** desenvolvida em linguagem *C* em ambiente *UNIX* numa estação de trabalho *SunSparc* e que se destina ao ambiente de processamento de imagem *XITE*[5], do qual utiliza algumas funções e tipos de variáveis já previamente definidas. Esta implementação contém comentários para auxílio à sua compreensão, assim como informação sobre os diferentes módulos, para sua posterior extracção através da ferramenta *CDOC(1)* existente no *XITE*.

```
/*
_____________________________________________________________________

  calcamera.c
  @(#)calcamera.c   1.00 93/12/17, Copyright 1993, DEEC, FEUP
  Departamento de Engenharia Electrotecnica e de Computadores
  Faculdade de Engenharia
  Rua dos Bragas
  4099 PORTO CODEX
  PORTUGAL
  E-mail: gpai@obelix.fe.up.pt
_____________________________________________________________________

  readarg.h
  @(#)readarg.h   1.3 92/01/15, Copyright 1990, Blab, UiO
  Image processing lab, Department of Informatics
  University of Oslo
  E-mail: blab@ifi.uio.no
_____________________________________________________________________

  message.h
  @(#)message.h   1.3 92/01/15, Copyright 1990, Blab, UiO
  Image processing lab, Department of Informatics
  University of Oslo
  E-mail: blab@ifi.uio.no
_____________________________________________________________________

  Permission to use, copy, modify and distribute this software and its documentation for any purpose
and without fee is hereby granted, provided that this copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in supporting documentation and that the
name of B-lab, Department of Informatics or University of Oslo not be used in advertising or publicity
pertaining to distribution of the software without specific, written prior permission.

  B-LAB DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT
SHALL B-LAB BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL
DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA
OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER
TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
*/

/*********************************************************************/

static char *SccsId = "@(#)calcamera.c 1.00 93/12/17, DEEC, FEUP";

/*********************************************************************/

/* INCLUDES */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <blab/readarg.h>
#include <blab/message.h>

/*********************************************************************/

/* DEFINES */

#define NPOINT 150       /* maximum number of points to be considered */
#define NMAX 5           /* number of variables in the model for the least-squares */
#define NMAX1 3          /* dimension square matrix of coefficients to be solved by solve function
*/
#define STOP1 0.000001 /* control variable used in the non linear search of the buffer's center */
#define STOP2 0.000001     /* control variable used in the non linear search of the buffer's center
*/

/*********************************************************************/

/* GLOBAL VARIABLES */

typedef struct {

  float xw;         /* point's 3D world coordinate x */
```

[5] *X-based Image processing Tools an Environmemt*, [Lønnestad, 1992].

```
  float yw;        /* point's 3D world coordinate y */
  float xf;        /* point's 2D frame buffer coordinate x */
  float yf;        /* point's 2D frame buffer coordinate y */
  double xd;       /* point's 2D distorted image coordinate x */
  double yd;       /* point's 2D distorted image coordinate y */

} calpoint;

/*************************************************************************/

/*F:linleastsquar*
_________________________________________________________________

  linleastsquar

_________________________________________________________________

Name:       linleastsquar - make the linear least-squares regression by normal
                            equations and Cholesky's decomposition

Syntax:          | int linleastsquar(m, n, z, a, y)
                 | int m, n;
                 | double z[][NMAX], a[], y[];

Description:   'z[][]' is the matrix of the observed values for the independent
                 variable.
                 'n' is the number of variables in the model and 'm' is the number
                 of data points, 'NMAX' is the maximum number of variables in
                 the model.
                 The vector 'y[]' contains the observed values of the dependent
                 variable.
                 The vector 'a[]' contains the unknown coefficients.

Return value:  | 0 => Ok.
                 | 1, 2 => Can't make CHOLESKY decomposition.

Example:         | Try yourself.

Restrictions:    The matrix [(z[[]][]transposed)*z[][]] must be positive definite.

Author:          Joao Tavares

Id:              @(#)linleastsquar.c   1.00   93/12/18

_________________________________________________________________
*/

int linleastsquar(m, n, z, a, y)
int m, n;
double z[][NMAX], a[], y[];

{

 register int i, j, k;
 double aux1[NMAX][NMAX], aux2[NMAX], aux3[NMAX], sum;

 /* make aux1[][]=(z[][]transposed)*z[][] */

 for (i = 0; i < n; ++i) {

  for (j = 0; j < n; ++j) {

    aux1[i][j] = 0.0;
    for (k = 0; k < m; ++k) aux1[i][j] += z[k][i]*z[k][j];

  }

 }

 /* make aux2[][]=(z[][]transposed)*y[][] */

 for (i = 0; i < n; ++i) {

  aux2[i] = 0.0;
  for (j = 0; j < m; ++j) aux2[i] += z[j][i]*y[j];

 }

 /*  cholesky decomposition - aux1[][]=l[][]*(l[][]transposed) */

 for (k = 0; k < n; ++k) {

  for (i = 0; i < k; ++i) {

    sum = 0.0;
    for (j = 0; j < i; ++j) sum += aux1[i][j]*aux1[k][j];
    if (aux1[i][i] == 0.0) exit(Error(1,"\nCan't make CHOLESKY decomposition.\n"));
    aux1[k][i] = (aux1[k][i]-sum)/aux1[i][i];

  }
  sum = 0.0;
  for (j = 0; j < k; ++j) sum += aux1[k][j]*aux1[k][j];
```

```
  if (aux1[k][k] < sum) exit(Error(2,"\nCan't make CHOLESKY decomposition.\n"));
  aux1[k][k] = sqrt(aux1[k][k]-sum);

 }

 /* forward substitution */

 aux3[0] = aux2[0]/aux1[0][0];
 for (i = 1; i < n; ++i) {

  sum = 0.0;
  for (j = 0; j < i; ++j) sum += aux1[i][j]*aux3[j];
  aux3[i] = (aux2[i]-sum)/aux1[i][i];

 }

 /* backward substitution */

 a[n-1] = aux3[n-1]/aux1[n-1][n-1];
 for (i = n-2; i >= 0 ; --i) {

  sum = 0.0;
  for (j = i+1; j < n; ++j) sum += aux1[j][i]*a[j];
  a[i] = (aux3[i]-sum)/aux1[i][i];

 }

 return(0);

}

/*****************************************************************************/

/*F:solve*
_______________________________________________________________________

 solve
_______________________________________________________________________

Name:          solve - solve a system of linear algebraic equations by Cholesky's
                       decomposition

Syntax:        | int solve(alfa, delta, beta)
               | double alfa[][NMAX1], delta[], beta[];

Description:   'alfa[][]' is the 'NMAX1'-by-'NMAX1' square matrix of coefficients.
               'delta[]' is the vector of unknowns.
               'beta[]' is the vector of constants.

Return value:  | 0 => Ok.
               | 1, 2 => Can't make CHOLESKY decomposition.

Example:       | Try yourself.

Restrictions:  The matrix 'alfa[][]' must be positive definite.

Author:        Joao Tavares

Id:            @(#)solve.c   1.00   93/12/18
_______________________________________________________________________
*/

int solve(alfa, delta, beta)
double alfa[][NMAX1], delta[], beta[];

{

 register int i, j, k;
 double aux[NMAX1], sum;

 /* cholesky decomposition - alfa[][]=l[][]*(l[][]transposed) */

 for (k = 0; k < NMAX1; ++k) {

  for (i = 0; i < k; ++i) {

   sum = 0.0;
   for (j = 0; j < i; ++j) sum += alfa[i][j]*alfa[k][j];
   if (alfa[i][i] == 0.0) exit(Error(1,"\nCan't make CHOLESKY decomposition.\n"));
   alfa[k][i] = (alfa[k][i]-sum)/alfa[i][i];

  }
  sum = 0.0;
  for (j = 0; j < k; ++j) sum += alfa[k][j]*alfa[k][j];
  if (alfa[k][k] < sum) exit(Error(2,"\nCan't make CHOLESKY decomposition.\n"));
  alfa[k][k] = sqrt(alfa[k][k]-sum);

 }

 /* forward substitution */
```

```
  aux[0] = beta[0]/alfa[0][0];
  for (i = 1; i < NMAX1; ++i) {

    sum = 0.0;
    for (j = 0; j < i; ++j) sum += alfa[i][j]*aux[j];
    aux[i] = (beta[i]-sum)/alfa[i][i];

  }

  /* backward substitution */

  delta[NMAX1-1] = aux[NMAX1-1]/alfa[NMAX1-1][NMAX1-1];
  for (i = NMAX1-2; i >= 0 ; --i) {

    sum = 0.0;
    for (j = i+1; j < NMAX1; ++j) sum += alfa[j][i]*delta[j];
    delta[i] = (aux[i]-sum)/alfa[i][i];

  }

  return(0);

}

/*****************************************************************************/

/*F:findcenter*
_______________________________________________________________________

  findcenter
_______________________________________________________________________

Name:           findcenter - compute the image center in the frame buffer using the
                             radial alignment constraint

Syntax:         | int findcenter(n, point, dxl, dy, cx, cy)
                | double dxl, dy, *cx, *cy;
                | int n;
                | calpoint point[];

Description:    'n' is the number of calibration points.
                'point' is typedef struct { float xw; float yw; float xf; float yf;
                double xd; double yd; } calpoint. Where xw, yw are the world
                3D coordinate; xd, yd are the distorted image coordinate and xf,
                yf are the computer image coordinate in frame buffer.
                'dxl' is sx*dx where dx is the distance between adjacent sensor
                elements in x (scan line) direction and sx is the horizontal uncertainty
                factor.
                'dy' is the center to center distance between adjacent CCD sensor
                in the y direction.
                'cx' and 'cy' are the row and column numbers of the center of frame
                buffer, the input values are half of image's size.

Return value:   | 0 => Ok.

Example:        | Try yourself.

Restrictions:   The set of calibration points must be coplanar with the world 3D
                coordinate system being chosen so that z=0 for all points.
                The number of calibration points must be larger than five.

Author:         Joao Tavares

Id:             @(#)findcenter.c   1.00   94/01/10
_______________________________________________________________________
*/

int findcenter(n, point, dxl, dy, cx, cy)
double dxl, dy, *cx, *cy;
int n;
calpoint point[];

{

  int auxin, pass;
  double tx, ty, tz, f, k1, r[9], z[NPOINT][NMAX], a[NMAX], y[NPOINT], aux, aux1, aux2, aux3,
aux4, aux5,  beta[2], alfa[2][2], delta[2], quiold, quinew, lambda, dcx, dcy, cxold, cyold;
  register int i;

  pass = 0;

  do {

    /* define xd, yd for all calibration points */

    for (i = 0; i < n; ++i) {

      point[i].xd = dxl*(point[i].xf-*cx);
      point[i].yd = dy*(point[i].yf-*cy);
```

```
}

/* define z[NPOINT][NMAX], y[NPOINT] of linear equations */

for (i = 0; i < n; ++i) {

  z[i][0] = point[i].yd*point[i].xw;
  z[i][1] = point[i].yd*point[i].yw;
  z[i][2] = point[i].yd;
  z[i][3] = -(point[i].xd*point[i].xw);
  z[i][4] = -(point[i].xd*point[i].yw);
  y[i] = point[i].xd;

}

/* solve the linear regression by linear least squares */

linleastsquar(n, 5, z, a, y);

/* define r[], tx, ty */

aux = a[0]*a[0]+a[1]*a[1]+a[3]*a[3]+a[4]*a[4];
aux1 = a[0]*a[4]-(a[3]*a[1]);

if (aux1 != 0.0) ty = sqrt((aux-sqrt(aux*aux-(4*aux1*aux1)))/(2*aux1*aux1));
else ty = sqrt(1.0/aux);

/* define ty sign */

aux1 = point[0].xf-*cx;
aux1 *= aux1;
aux2 = point[0].yf-*cy;
aux2 *= aux2;
aux = sqrt(aux1+aux2);
auxin = 0;

/* find the point more distant of *cx, *cy */

for (i = 1; i < n; ++i) {

  aux1 = point[i].xf-*cx;
  aux1 *= aux1;
  aux2 = point[i].yf-*cy;
  aux2 *= aux2;
  aux3 = sqrt(aux1+aux2);
  if (aux3 < aux) {

    aux = aux3;
    auxin = i;

  }

}

r[0] = a[0]*ty;
r[1] = a[1]*ty;
r[3] = a[3]*ty;
r[4] = a[4]*ty;
tx = a[2]*ty;

aux = r[0]*point[auxin].xw+r[1]*point[auxin].yw+tx;
aux1 = r[3]*point[auxin].xw+r[4]*point[auxin].yw+ty;
aux3 = point[auxin].xd/aux;
aux4 = point[auxin].yd/aux1;

if (aux3 < 0.0 || aux4 < 0.0) {

  /* ty sign is wrong */

  r[0] = -(r[0]);
  r[1] = -(r[1]);
  r[3] = -(r[3]);
  r[4] = -(r[4]);
  tx = -(tx);
  ty = -(ty);

}

/* define R */

r[2] = sqrt(1.0-(r[0]*r[0])-(r[1]*r[1]));
r[5] = sqrt(1.0-(r[3]*r[3])-(r[4]*r[4]));
aux = r[0]*r[3]+r[1]*r[4];
aux /= fabs(aux);
aux = -(aux);
r[5] *= aux;
r[6] = r[1]*r[5]-r[2]*r[4];
r[7] = r[2]*r[3]-r[0]*r[5];
r[8] = r[0]*r[4]-r[1]*r[3];
```

```
  /* compute an approximation of f and tz by ignoring lens distorcion */

  for (i = 0; i < n; ++i) {

    aux1 = r[3]*point[i].xw+r[4]*point[i].yw+ty;
    aux2 = r[6]*point[i].xw+r[7]*point[i].yw;
    z[i][0] = aux1;
    z[i][1] = -(point[i].yd);
    y[i] = aux2*point[i].yd;

  }

  /* solve the linear regression by linear least squares */

  linleastsquar(n, 2, z, a, y);

  f = a[0];
  tz = a[1];

  if (f < 0.0) {

    r[2] = -(r[2]);
    r[5] = -(r[5]);
    r[6] = -(r[6]);
    r[7] = -(r[7]);
    f = -(f);
    tz = -(tz);

  }

  /* compute the *cx, *cy by Levenberg-Marquardt method */

  dcx = dcy = 0.0;
  cxold = *cx;
  cyold = *cy;
  lambda = 0.001;
  quiold = 0.0;

  /* compute merit function for the first solution */

  for (i = 0; i < n; ++i) {

    aux = -(point[i].yd*(point[i].xw*r[0]+point[i].yw*r[1]+tx))+point[i].xd*(point[i].xw*
r[3]+point[i].yw*r[4]+ty);
    quiold += aux*aux;

  }

  do {

    beta[0] = beta[1] = alfa[0][0] = alfa[0][1] = alfa[1][1] = 0.0;

    /* compute beta[] and alfa[][] */

    for (i = 0; i < n; ++i) {

      aux = point[i].xw*r[6]+point[i].yw*r[7]+tz;
      aux1 = -point[i].xw*r[3]-point[i].yw*r[4]-ty;
      aux2 = point[i].xw*r[0]+point[i].yw*r[1]+tx;
      aux3 = -(point[i].yd*aux2)+point[i].xd*(-(aux1));
      aux4 = (point[i].yd/f)*aux+aux1;
      aux5 = -(point[i].xd/f)*aux+aux2;
      aux3 -= dcx*aux4+dcy*aux5;
      beta[0] += aux3*aux4;
      beta[1] += aux3*aux5;
      alfa[0][0] += aux4*aux4;
      alfa[0][1] += aux4*aux5;
      alfa[1][1] += aux5*aux5;

    }

    alfa[1][0] = alfa[0][1];
    alfa[0][0] *= (1.0+lambda);
    alfa[1][1] *= (1.0+lambda);

    /* solve the linear equations */

    delta[0] = (beta[0]*alfa[1][1]-alfa[0][1]*beta[1])/(alfa[0][0]*alfa[1][1]-alfa[0][1]*alfa[1][0]);
    delta[1] = (beta[1]-alfa[1][0]*delta[0])/alfa[1][1];

    /* compute merit function for the new solution */

    quinew = 0.0;
    for (i = 0; i < n; ++i) {

      aux = point[i].xw*r[6]+point[i].yw*r[7]+tz;
      aux1 = -point[i].xw*r[3]-point[i].yw*r[4]-ty;
      aux2 = point[i].xw*r[0]+point[i].yw*r[1]+tx;
      aux3 = -(point[i].yd*aux2)+point[i].xd*(-(aux1));
```

```
      aux4 = (point[i].yd/f)*aux+aux1;
      aux5 = -(point[i].xd/f)*aux+aux2;
      aux3 -= (dcx+delta[0])*aux4+(dcy+delta[1])*aux5;
      quinew += aux3*aux3;

    }

    if (quinew >= quiold) {

      lambda *= 10.0;
      quiold = quinew;

    }
    else {

      lambda *= 0.1;
      dcx += delta[0];
      dcy += delta[1];
      *cx -= dcx;
      *cy -= dcy;
      if (quiold-quinew < STOP1) {

        aux = (*cx-cxold);
        aux *= aux;
        aux1 = (*cy-cyold);
        aux1 *= aux1;
        aux += aux1;
        if (aux < STOP2) pass = 1;
        break;

      }
      quiold = quinew;

    }

  } while(1);

 } while(!pass);

 *cx = floor(*cx+0.5);
 *cy = floor(*cy+0.5);

 return(0);

}
```

```
/*****************************************************************/

/*P:calcamera*
_________________________________________________________________

 calcamera
_________________________________________________________________

Name:        calcamera - performs a camera calibration in the context of 3D
                         machine vision using a coplanar set of calibration points

Syntax:    | [-f <outfilename>] [-n <nlstop>] [-s <sx>] [-c <findcenter> ] <dx>
           | <dy> <Cx> <Cy> <infilename>

Description:  Determining the internal camera geometric and optical
              characteristics (the intrinsic parameters: the effective focal length
              f and the lens radial distortion coefficient k1), and the 3D position
              and orientation of the camera relative to a certain world coordinate
              system (extrinsic parameters: the rotation matrix R and the
              translation vector T).
              The output results can be writing in the file 'outfilename' if the flag
              '-f' is on, by default this flag is off.
              The error tolerance for the exact solution of f, tz and k1, by
              Levenberg-Marquardt method can be indicated through flag '-n', by
             default 'nlstop' is 0.0000001.
              The horizontal uncertainty factor 'sx' can be indicated through flag '-s',
              and by default 'sx' is 0.710935.
              If through the flag '-c' is indicated that 'findcenter' is 1 then the image
              center in the frame buffer is computed, by default isn't compute.
              'dx' is the distance between adjacent sensor elements in x
              (scan line) direction.
              'dy' is the center to center distance between adjacent CCD sensor in
              the y direction.
              'Cx' and 'Cy' are the row and column numbers of the image center in the
              frame buffer. If the correct values are unknown, then the input values
              are half of image's size.
              'infilename' is the name of input file which have the world and frame
              buffer coordinate of all the calibration points.

Return value: | 1 => Can't open file <infilename>.
              | 2 => The point(n) doesn't have the world coordinate z equal to zero.
              | 3 => The number of calibration points must be larger than 5.
              | 4 => The horizontal uncertainty factor (sx) must be different from zero.
              | 5 => Can't open file <outfilename>.
```

```
Examples:     | Try yourself.

Restrictions: The set of calibration points must be coplanar with the world 3D
              coordinate system being chosen so that z=0 for all points.
              The number of calibration points must be larger than five.
              The horizontal uncertainty factor 'sx' must be different from zero.

Author:       Joao Tavares

Id:           @(#)calcamera.c   1.0   93/12/18
_____________________________________________________________________
*/

main(argc, argv)
int argc;
char **argv;

{

  int n, auxin, find;
  double sx, nlstop, dx, dxl, dy, tx, ty, tz, f, k1, r[9], z[NPOINT][NMAX], a[NMAX], y[NPOINT],
aux, aux1, aux2, aux3, aux4, aux5,  beta[3], cx, cy, alfa[3][3], delta[3], quiold, quinew, lambda;
  float auxf;
  char *outfilename, arg[200], arg1[200], arg2[80], arg3[40], arg4[60];
  register int i;
  FILE *infpt, *outfpt;
  calpoint point[NPOINT];

  InitMessage(&argc, argv, "Usage: %s\n [-f <outfilename>] [-n <nlstop>] [-s <sx>] [-c
<findcenter> ] <dx> <dy> <Cx> <Cy> <infilename>\n");
  Iset_message(1);
  Iset_abort(1);

  /* read switches */

  outfilename = read_switch(&argc, argv, "-f", 1, NULL);
  nlstop = atof(read_switch(&argc, argv, "-n", 1, "0.0000001"));
  sx = atof(read_switch(&argc, argv, "-s", 1, "0.710935"));
  find = atoi(read_switch(&argc, argv, "-c", 1, "0"));

  /* read argv[] */

  if (argc == 1) Usage(1, (char*)0);
  if (argc != 6) Usage(2, "\nBad number of arguments.\n");

  dx = atof(argv[1]);
  dy = atof(argv[2]);
  cx = atoi(argv[3]);
  cy = atoi(argv[4]);

  /* open input file, read world 3D coordinates, compute the image 2D coordinates of all calibration
points, and close */

  infpt = fopen(argv[5], "r");
  if (infpt == NULL) exit(Error(1, "\nCan't open file %s.\n", argv[5]));
  n = 0;
  fscanf(infpt, "%g %g %g %g %g", &point[n].xw, &point[n].yw, &auxf, &point[n].xf,
&point[n].yf);

  while (!feof(infpt)) {

    if (auxf != 0.0) exit(Error(2, "\nThe point(%i) doesn't have the world coordinate z equal to
zero.\n", n));
    n += 1;
    fscanf(infpt, "%g %g %g %g %g", &point[n].xw, &point[n].yw, &auxf, &point[n].xf,
&point[n].yf);

  }

  fclose(infpt);
  if (n < 5) exit(Error(3, "\nThe number of calibration points must be larger than 5.\n"));

  /* define dxl */

  if (sx == 0.0) exit(Error(4, "\nThe horizontal uncertainty factor (sx) must be different from
zero.\n"));

  dxl = dx*sx;

  /* if was desire find the image center in the frame buffer */

  if (find == 1) findcenter (n, point, dxl, dy, &cx, &cy);

  /* define xd, yd for all calibration points */

  for (i = 0; i < n; ++i) {

    point[i].xd = dxl*(point[i].xf-cx);
    point[i].yd = dy*(point[i].yf-cy);
```

```
}

/* define z[NPOINT][NMAX], y[NPOINT] of linear equations */

for (i = 0; i < n; ++i) {

  z[i][0] = point[i].yd*point[i].xw;
  z[i][1] = point[i].yd*point[i].yw;
  z[i][2] = point[i].yd;
  z[i][3] = -(point[i].xd*point[i].xw);
  z[i][4] = -(point[i].xd*point[i].yw);
  y[i] = point[i].xd;

}

/* solve the linear regression by linear least squares */

linleastsquar(n, 5, z, a, y);

/* define r[], tx, ty */

aux = a[0]*a[0]+a[1]*a[1]+a[3]*a[3]+a[4]*a[4];
aux1 = a[0]*a[4]-(a[3]*a[1]);

if (aux1 != 0.0) ty = sqrt((aux-sqrt(aux*aux-(4*aux1*aux1)))/(2*aux1*aux1));
else ty = sqrt(1.0/aux);

/* define ty sign */

aux1 = point[0].xf-cx;
aux1 *= aux1;
aux2 = point[0].yf-cy;
aux2 *= aux2;
aux = sqrt(aux1+aux2);
auxin = 0;

/* find the point more distant of cx, cy */

for (i = 1; i < n; ++i) {

  aux1 = point[i].xf-cx;
  aux1 *= aux1;
  aux2 = point[i].yf-cy;
  aux2 *= aux2;
  aux3 = sqrt(aux1+aux2);
  if (aux3 < aux) {

    aux = aux3;
    auxin = i;

  }

}

r[0] = a[0]*ty;
r[1] = a[1]*ty;
r[3] = a[3]*ty;
r[4] = a[4]*ty;
tx = a[2]*ty;

aux = r[0]*point[auxin].xw+r[1]*point[auxin].yw+tx;
aux1 = r[3]*point[auxin].xw+r[4]*point[auxin].yw+ty;
aux3 = point[auxin].xd/aux;
aux4 = point[auxin].yd/aux1;

if (aux3 < 0.0 || aux4 < 0.0) {

  /* ty sign is wrong */

  r[0] = -(r[0]);
  r[1] = -(r[1]);
  r[3] = -(r[3]);
  r[4] = -(r[4]);
  tx = -(tx);
  ty = -(ty);

}

/* define R */

r[2] = sqrt(1.0-(r[0]*r[0])-(r[1]*r[1]));
r[5] = sqrt(1.0-(r[3]*r[3])-(r[4]*r[4]));
aux = r[0]*r[3]+r[1]*r[4];
aux /= fabs(aux);
aux = -(aux);
r[5] *= aux;
r[6] = r[1]*r[5]-r[2]*r[4];
r[7] = r[2]*r[3]-r[0]*r[5];
r[8] = r[0]*r[4]-r[1]*r[3];
```

```
/* compute an approximation of f and tz by ignoring lens distortion */

for (i = 0; i < n; ++i) {

  aux1 = r[3]*point[i].xw+r[4]*point[i].yw+ty;
  aux2 = r[6]*point[i].xw+r[7]*point[i].yw;
  z[i][0] = aux1;
  z[i][1] = -(point[i].yd);
  y[i] = aux2*point[i].yd;

}

/* solve the linear regression by linear least squares */

linleastsquar(n, 2, z, a, y);

f = a[0];
tz = a[1];

if (f < 0.0) {

  r[2] = -(r[2]);
  r[5] = -(r[5]);
  r[6] = -(r[6]);
  r[7] = -(r[7]);
  f = -(f);
  tz = -(tz);

}

/* compute the exact solution for f, tz and k1, by Levenberg-Marquardt method */

k1 = 0.0;
lambda = 0.001;
quiold = 0.0;

/* compute merit function for the first solution */

for (i = 0; i < n; ++i) {

  aux = r[3]*point[i].xw+r[4]*point[i].yw+ty;
  aux1 = r[6]*point[i].xw+r[7]*point[i].yw+tz;
  aux3 = point[i].yd-f*aux/aux1;
  quiold += aux3*aux3;

}

do {

  beta[0] = beta[1] = beta[2] = 0.0;
  alfa[0][0] = alfa[0][1] = alfa[0][2] = 0.0;
  alfa[1][0] = alfa[1][1] = alfa[1][2] = 0.0;
  alfa[2][0] = alfa[2][1] = alfa[2][2] = 0.0;

  /* compute beta[] and alfa[][] */

  for (i = 0; i < n; ++i) {

    aux = r[3]*point[i].xw+r[4]*point[i].yw+ty;
    aux1 = r[6]*point[i].xw+r[7]*point[i].yw;
    aux2 = point[i].yd*(point[i].yd*point[i].yd+point[i].xd*point[i].xd);
    aux3 = aux1+tz;
    aux4 = aux/aux3;
    aux3 = -f*aux/(aux3*aux3);
    aux5 = point[i].yd-f*aux4+k1*aux2;
    beta[0] += aux5*aux4;
    beta[1] += aux5*aux3;
    beta[2] += aux5*(-aux2);
    alfa[0][0] += aux4*aux4;
    alfa[0][1] += aux4*aux3;
    alfa[0][2] += aux4*(-aux2);
    alfa[1][1] += aux3*aux3;
    alfa[1][2] += (-aux2)*aux3;
    alfa[2][2] += aux2*aux2;

  }

  alfa[1][0] = alfa[0][1];
  alfa[2][0] = alfa[0][2];
  alfa[2][1] = alfa[1][2];
  alfa[0][0] *= (1.0+lambda);
  alfa[1][1] *= (1.0+lambda);
  alfa[2][2] *= (1.0+lambda);

  /* solve the linear equations */

  solve(alfa, delta, beta);

  /* compute merit function for the new solution */
```

```
    quinew = 0.0;
    for (i = 0; i < n; ++i) {

      aux = r[3]*point[i].xw+r[4]*point[i].yw+ty;
      aux1 = r[6]*point[i].xw+r[7]*point[i].yw+(tz+delta[1]);
      aux2 = point[i].yd*(k1+delta[2])*(point[i].yd*point[i].yd+point[i].xd*point[i].xd);
      aux3 = (f+delta[0])*aux/aux1;
      aux4 = point[i].yd-aux3+aux2;
      quinew += aux4*aux4;

    }

    if (quinew >= quiold) {

      lambda *= 10.0;
      quiold = quinew;

    }
    else {

      lambda *= 0.1;
      f += delta[0];
      tz += delta[1];
      k1 += delta[2];

      if (quiold-quinew < nlstop) break;
      quiold = quinew;

      }

  } while(1);

  /* output results */

  sprintf(arg, "\nResults for %s with:\n\n\tStop for nonlinear equations=%g, Find center=%d,
sx=%g, dx=%g,\n\tdy=%g, Cx=%g, Cy=%g, Input file: %s with %i calibration points.\n", argv[0],
nlstop, find, sx, dx, dy, cx, cy, argv[5], n);
  sprintf(arg1, "\nRotation matrix R:\n\n\t\t  %+#g\t%+#g\t%+#g\n\t\t  %+#g\t%+#g\t%+#g\n\t\t
%+#g\t%+#g\t%+#g\n", r[0], r[1], r[2], r[3], r[4], r[5], r[6], r[7], r[8]);
  sprintf(arg2, "\nTranslation vector T: (%+#g, %+#g, %+#g)\n", tx, ty, tz);
  sprintf(arg3, "\nFocal length f: %+#g\n", f);
  sprintf(arg4, "\nRadial lens distortion k1: %+#g\n\n", k1);

  printf(arg);
  printf(arg1);
  printf(arg2);
  printf(arg3);
  printf(arg4);

  /* open file if it was desire and write arg, ..., arg4, and close */

  if (outfilename != NULL) {

    outfpt = fopen(outfilename, "a");
    if (outfpt == NULL) exit(Error(5, "\nCan't open file %s.\n", outfilename));

    fprintf(outfpt, arg);
    fprintf(outfpt, arg1);
    fprintf(outfpt, arg2);
    fprintf(outfpt, arg3);
    fprintf(outfpt, arg4);
    fclose(outfpt);

  }

}

/*************************************************************************/
```

3 - Determinação das coordenadas na memória frame dos pontos de calibração: calpoint

Nesta secção é apresentada a implementação de um método descrito em [Tavares, 1995] para determinação das coordenadas na memória *frame* (X_f, Y_f) dos pontos de calibração. Como principais características deste método salientam-se:

- ✓ Precisão ao nível do *subpixel*.
- ✓ Os pontos de calibração são definidos como vértices de figuras geométricas com quatro lados paralelos dois a dois, como, por exemplo, quadrados, rectângulos, losângulos, etc.
- ✓ Não consideração de falsas linhas.
- ✓ As coordenadas dos pontos de calibração são determinadas pela intersecção de rectas que são definidas analiticamente por regressão linear, pelo método dos mínimos quadrados, dos *pixels* pertencentes às respectivas rectas.
- ✓ A distância máxima para um dado *pixel* pertencer a uma dada recta é especificada pelo utilizador; este valor deverá estar de acordo com as dimensões das figuras geométricas utilizadas e do espaçamento entre estas.
- ✓ Determinação automática das duas direcções principais existentes na imagem de entrada, sendo a diferença mínima aceitável entre estas especificada pelo utilizador, assim como a tolerância de erro de classificação quanto à direcção de um dado *pixel*.
- ✓ Saída das coordenadas determinadas em ficheiro.
- ✓ Criação de uma imagem de saída com três bandas: a primeira, contém todos os *pixels* identificados com amplitude superior a um dado limiar especificado pelo utilizador e direcção aproximadamente igual a uma direcção principal; a segunda, contém todos os *pixels* identificados com amplitude superior a um dado limiar especificado pelo utilizador e direcção aproximadamente igual à outra direcção principal; na terceira, são identificados os pontos de calibração com uma cruz.

Como principais requisitos da imagem a ser sujeita a esta implementação têm-se:

- ⌦ A imagem de entrada deverá conter figuras geométricas perfeitamente iguais e paralelas entre si.
- ⌦ A imagem de entrada deverá ser sujeita previamente a um detector de orlas de intensidade, como, por exemplo, o detector de Deriche[6].
- ⌦ A imagem de entrada deverá ser constituída por duas bandas do tipo *"UNS_BYTE"*. Na primeira banda deverão ser registados os valores de amplitude e, na segunda, os valores de direcção entre *0* e π.

3.1 - Descrição dos diferentes módulos

A implementação é constituída pelos módulos: ***principal***, ***lregression***, ***mark***, ***findlines*** e ***findpoints***, *Fig. 3*. Apresenta-se de seguida a descrição de cada um destes módulos.

3.1.1 - Módulo principal

Este módulo é responsável pela interface implementação/utilizador. A seu cargo está:

- ✓ Leitura dos argumentos opcionais ("*switches*"):

[6] Ver, por exemplo [Tavares, 1995].

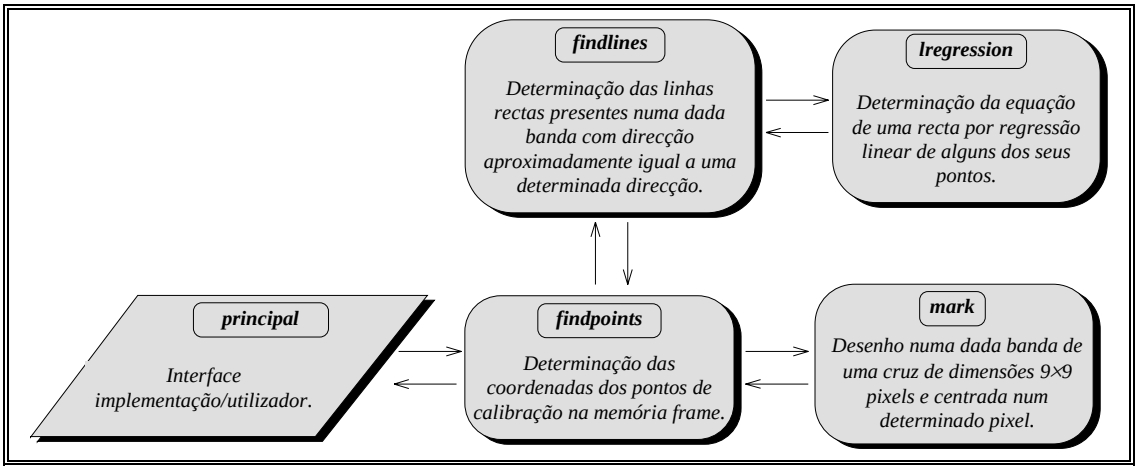


*Fig. 3 - Módulos integrantes da implementação **calpoint** e suas relações.*

- *title*, (["-ti <title>"]), título da imagem de saída. Por defeito, é igual a "*Calibration points*".
- *erthresang*, ("[-e <erthresang>]"), tolerância de erro na classificação de um dado *pixel* quanto à sua direcção. Por defeito, é igual a *2*.
- *difangmax*, ("[-d <difangmax>]"), diferença mínima entre as duas direcções principais. Por defeito, é igual a *32*.
- *maxdist*, ("[-m <maxdist>]"), máxima distância de um dado *pixel* a uma recta para ser classificado como pertencente a esta. Por defeito, é igual a *10.0*.
- *thresamp*, ("[-t <thresamp>"]), valor mínimo de amplitude que um dado *pixel* deverá apresentar para não ser classificado como ruído. Por defeito, é igual a *50*.

✓ Leitura dos argumentos passados à implementação pela linha de comando:

- *inimage*, ("<inimage>"), nome da imagem de entrada.
- *outimage*, ("<outimage>"), nome da imagem de saída.
- *outfile*, ("<outfile>"), nome do ficheiro para escrita dos resultados obtidos, nomeadamente as coordenadas na memória *frame* (X_f, Y_f) dos pontos de calibração determinados. Na abertura, o ficheiro, se ainda não existir, é criado; caso contrário, os resultados são acrescentados ao ficheiro já existente.

✓ Leitura da imagem de entrada.

✓ Criação da imagem de saída e das respectivas bandas.

✓ Abertura do ficheiro de saída.

✓ Chamada do módulo ***findpoints***.

✓ Fecho do ficheiro de saída.

✓ Apresentação do número de pontos de calibração determinados.

✓ Escrita em disco da imagem de saída com a respectiva descrição das condições de obtenção.

✓ Comunicação da existência de erros em argumentos de entrada ou ao longo do decurso da execução:

- Através de valores de retorno:
  - 1, ("Can't read image <inimage>."), o módulo não consegue realizar a leitura da imagem especificada.

- 2, ("The <inimage> doesn't have two bands."), a imagem especificada não é constituída por duas bandas.
- 3, ("Bad input pixel type for band[1]."), a primeira banda da imagem de entrada não é do tipo *"UNS_BYTE"*.
- 4, ("Bad input pixel type for band[2]."), a segunda banda da imagem de entrada não é do tipo *"UNS_BYTE"*.
- 5, ("The input bands have different size."), as bandas da imagem de entrada têm tamanhos diferentes.
- 6, ("Can't make image <outimage>"), o módulo não consegue criar a imagem de saída especificada.
- 7, ("Can't make band[1] of output image <outimage>."), o módulo não consegue criar a primeira banda da imagem de saída especificada.
- 8, ("Can't make band[2] of output image <outimage>."), o módulo não consegue criar a segunda banda da imagem de saída especificada.
- 9, ("Can't make band[3] of output image <outimage>."), o módulo não consegue criar a terceira banda da imagem de saída especificada.
- 10, ("Can't open file <outfile>."), o módulo não consegue abrir o ficheiro de saída especificado.

- Através de mensagens:
  - 1, (*InitMessage*), quando o número de argumentos é igual a um, isto é, apenas o nome da implementação.
  - 2, ("Bad number of arguments."), quando o número de argumentos é diferente de nove.

Como restrições na utilização deste módulo têm-se:

- A imagem de entrada tem de ser do tipo *"UNS_BYTE"*.
- A imagem de entrada deverá ser constituída por duas bandas de igual tamanho, resultantes de um detector de orlas de intensidade como, por exemplo, o detector de Deriche. Este detector deverá classificar cada *pixel* na primeira banda quanto à amplitude e, na segunda banda, quanto à direcção, entre *0* e π.

3.1.2 - Módulo lregression

Este módulo é responsável pela determinação da equação de uma recta que melhor se ajusta a um dado conjunto de pontos. Esta determinação é realizada utilizando a regressão linear pelo método dos mínimos quadrados[7].

Como parâmetros de entrada para este módulo têm-se:

- *linepoint[ ]*, do tipo estrutura *{double x; double y;}*, onde *x* e *y* são as coordenadas de um dado ponto.
- *nline*, do tipo inteiro, número de identificação da recta cuja equação deve ser determinada.
- *n*, do tipo inteiro, número de pontos que deverão ser considerados.
- *calline[ ]*, do tipo estrutura *{double m; double b;}*, onde *m* e *b* são os parâmetros da equação $y = mx + b$ da recta a ser determinada.

Como valores de retorno deste módulo têm-se:

[7] Ver, por exemplo [Chapra, 1988].

- 0, a execução do módulo decorreu com normalidade,
- 1, ("Can't make the linear regression."), a regressão linear não pode ser executada, pois a expressão $n\sum_i x_i^2 - \left(\sum_i x_i\right)^2$ é igual a zero para os pontos a ajustar.

Como restrições na utilização deste módulo têm-se que a expressão $n\sum_i x_i^2 - \left(\sum_i x_i\right)^2$ deve ser diferente de zero para os pontos a ajustar.

3.1.3 - Módulo mark

Este módulo é responsável pelo desenho, numa dada banda de entrada, de uma cruz de dimensões 9×9 centrada num *pixel* cujas coordenadas são passadas como parâmetros de entrada. O valor de cinzento com que esta cruz é desenhada é também indicado por um parâmetro de entrada.

Como parâmetros de entrada para este módulo têm-se:

- *band*, do tipo "*IBAND*", banda na qual deverá ser desenhada a cruz,
- *y*, do tipo inteiro, coordenada *y* do *pixel* central da cruz a ser desenhada,
- *x*, do tipo inteiro, coordenada *x* do *pixel* central da cruz a ser desenhada,
- *color*, do tipo inteiro, valor de cinzento que deverá assumir cada *pixel* pertencente à cruz a ser desenhada.

Este módulo não tem qualquer restrição na sua utilização, e retorna *0* como indicação de que a sua execução decorreu em normalidade.

3.1.4 - Módulo findlines

Este módulo é responsável pela determinação das equações das rectas que são definidos por um conjunto de *pixels*, numa dada banda de entrada, que têm mais ou menos a mesma direcção. O declive médio das rectas que deverão ser determinadas é passado a este módulo como parâmetro de chamada. Durante a execução, o módulo opta automaticamente pela melhor regressão linear, isto é, para rectas com declives entre *-45º* e *45º* a regressão linear é do tipo $y = mx + b$, e para os restantes declives é do tipo $x = m'y + b'$. Para determinar se um *pixel* pertence a uma dada recta, a sua distância a esta recta é calculada e comparada com a máxima distância admissível, que é passada ao módulo como parâmetro de chamada. Para eliminação de falsas linhas, o módulo só considera que se trata de uma recta se o número de *pixels* que esta contém é maior do que a variável *GOODLINE* definida globalmente. Para a determinação da equação de cada recta, este módulo utiliza o módulo ***lregression*** já descrito anteriormente.

Como parâmetros de entrada para este módulo têm-se:

- *band*, do tipo "*IBAND*", banda na qual se pretende determinar as rectas definidas pelos *pixels* que as constituem,
- *m*, do tipo real, declive médio das rectas a determinar,
- *maxdist*, do tipo real, máxima distância que um *pixel* pode ter de uma dada recta para ser considerado como integrante da mesma,
- *calline[]*, do tipo estrutura *{double m; double b}*, onde *m* e *b* são os parâmetros da equação $y = mx + b$ (ou $x = m'y + b'$) da recta a ser determinada.
- *v*, do tipo inteiro, igual a *1* se a direcção média das rectas a determinar for vertical.

Este módulo retorna o número de rectas encontradas, na banda de entrada, com a direcção média *m*, e não tem nenhuma restrição quanto à sua utilização.

3.1.5 - Módulo findpoints

Este módulo é responsável pela determinação das coordenadas na memória *frame* (X_f, Y_f) dos pontos de calibração, segundo o método descrito em [Tavares, 1995].

A seu cargo está:

- ✓ Determinação das duas direcções principais presentes na imagem de entrada; a mínima diferença entre estas direcções é passada como parâmetro de entrada.
- ✓ Classificação dos *pixels*, na primeira banda da imagem de saída, que têm amplitude superior a um dado limiar e que têm uma direcção próxima de uma das direcções principais.
- ✓ Classificação dos *pixels*, na segunda banda da imagem de saída, que têm amplitude superior a um dado limiar e que têm uma direcção próxima da outra direcção principal.
- ✓ Determinação das rectas que existem em cada uma destas bandas; para tal, passa cada banda como argumento de chamada ao módulo ***findlines*** descrito anteriormente.
- ✓ Determinação das coordenadas dos pontos de calibração através da intersecção das várias rectas definidas analiticamente.
- ✓ Registo dos pontos de calibração na terceira banda de saída; para tal, utiliza o módulo ***mark*** descrito anteriormente.
- ✓ Escrita das coordenadas dos pontos de calibração no ficheiro de saída de resultados.

Como parâmetros de entrada para este módulo têm-se:

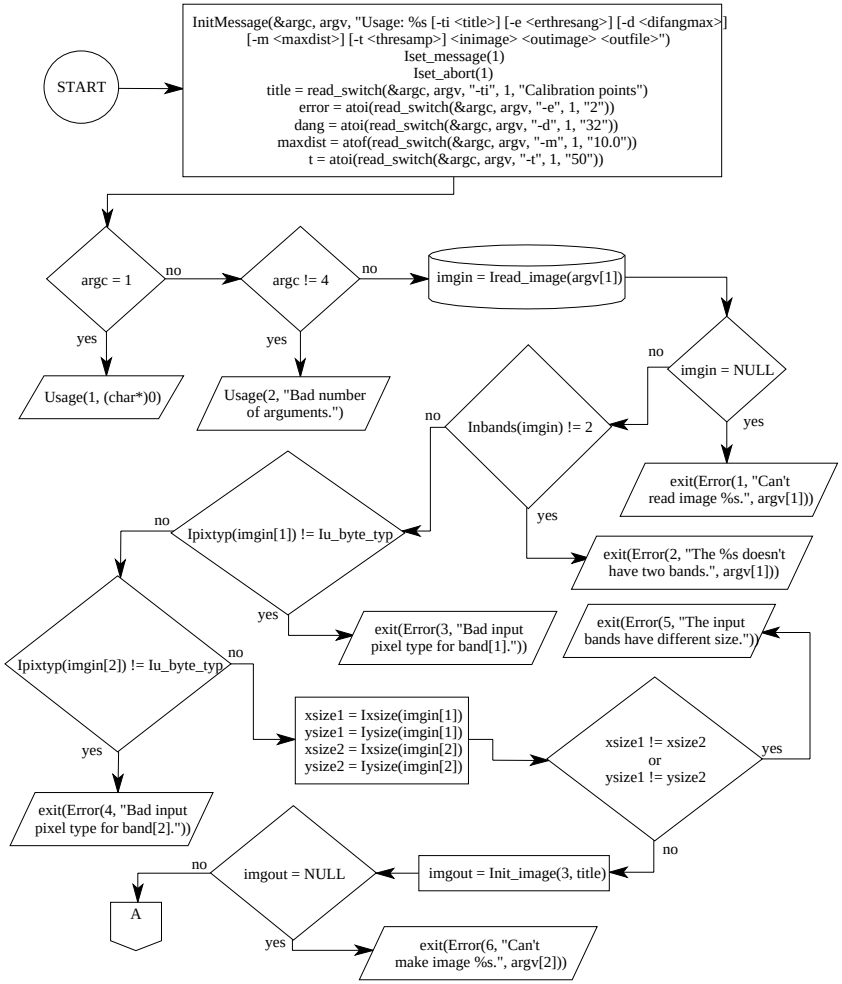
- ➲ *bin1*, do tipo "*IBAND*", primeira banda da imagem de entrada, obtida por um detector de orlas de intensidade, com os valores de amplitude para cada *pixel*.
- ➲ *bin2*, do tipo "*IBAND*", segunda banda da imagem de entrada, obtida por um detector de orlas de intensidade, com os valores da direcção para cada *pixel*.
- ➲ *bout1*, do tipo "*IBAND*", primeira banda da imagem de saída onde devem ser registados os *pixels* com amplitude superior a um dado limiar e com direcção próxima de uma das direcções principais.
- ➲ *bout2*, do tipo "*IBAND*", segunda banda da imagem de saída onde devem ser registados os *pixels* com amplitude superior a um dado limiar e com direcção próxima da outra direcção principal.
- ➲ *bout3*, do tipo "*IBAND*", terceira banda da imagem de saída onde deverão ser assinaladas, por meio de cruzes, as localizações dos vários pontos de calibração.
- ➲ *error*, do tipo inteiro, tolerância de erro na classificação de um *pixel* como tendo direcção igual a uma das direcções principais.
- ➲ *dang*, do tipo inteiro, mínima diferença admissível entre as duas direcções principais.
- ➲ *maxdist*, do tipo real, máxima distância de um dado *pixel* a uma recta para que seja classificado como pertencente a esta.
- ➲ *t*, do tipo inteiro, valor do limiar de amplitude para decidir se um dado *pixel* tem amplitude suficiente para não ser considerado como ruído.

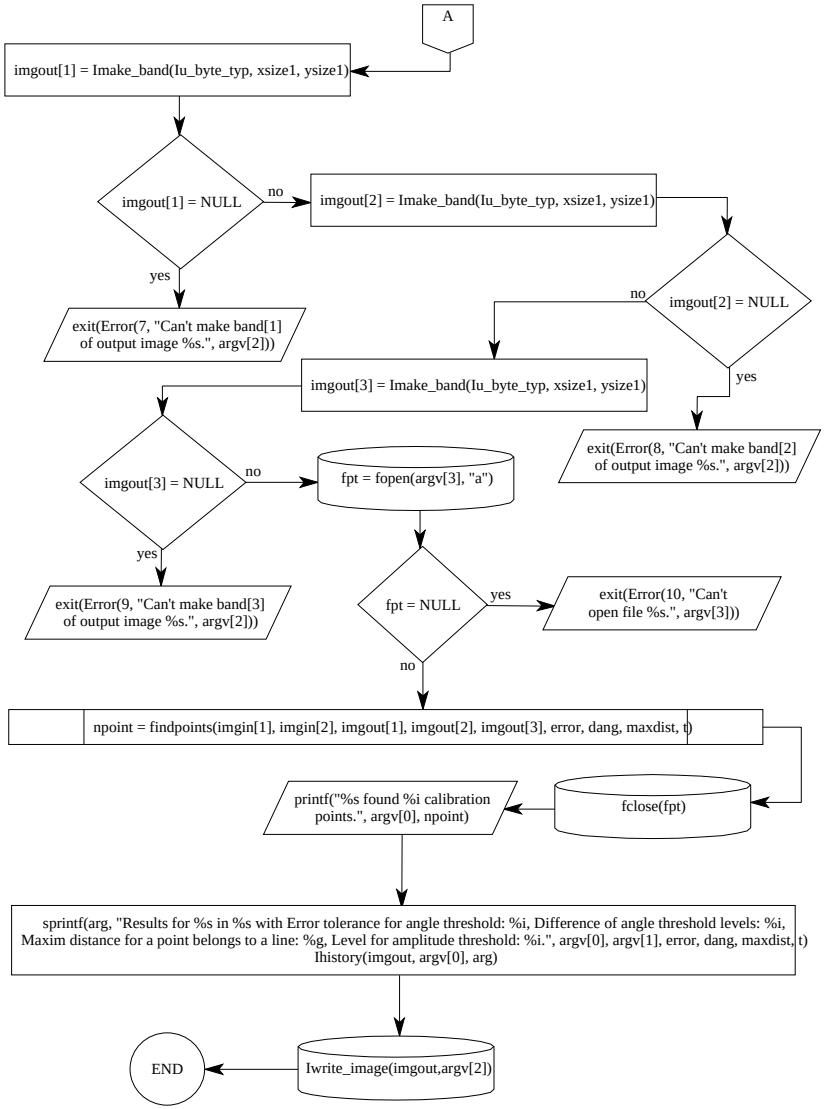
Este módulo retorna o número de pontos de calibração determinados e não apresenta quaisquer restrições quanto à sua utilização.

3.2 - Fluxogramas dos diferentes módulos

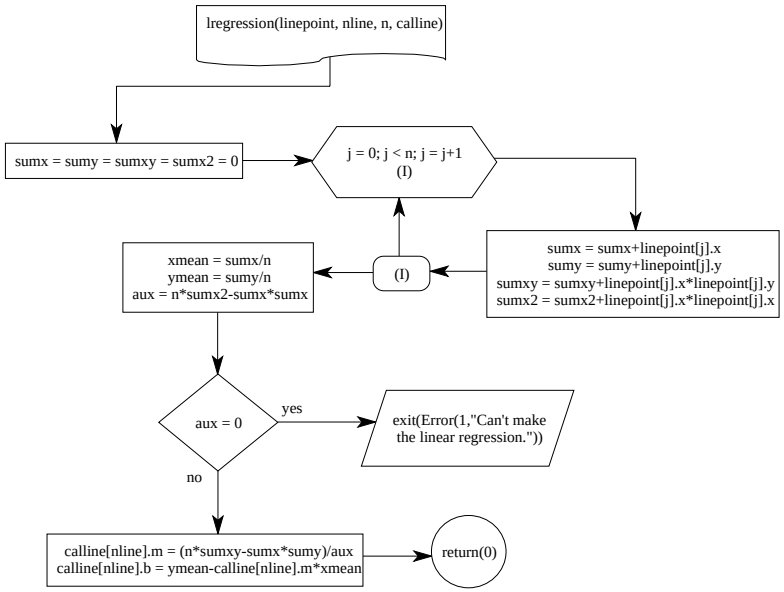
Apresentam-se, de seguida, os fluxogramas dos módulos que constituem a implementação ***calpoint***.

3.2.1 - Módulo principal

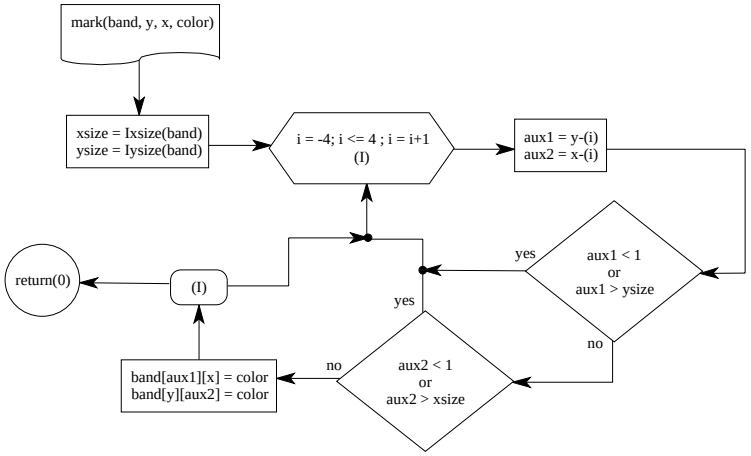




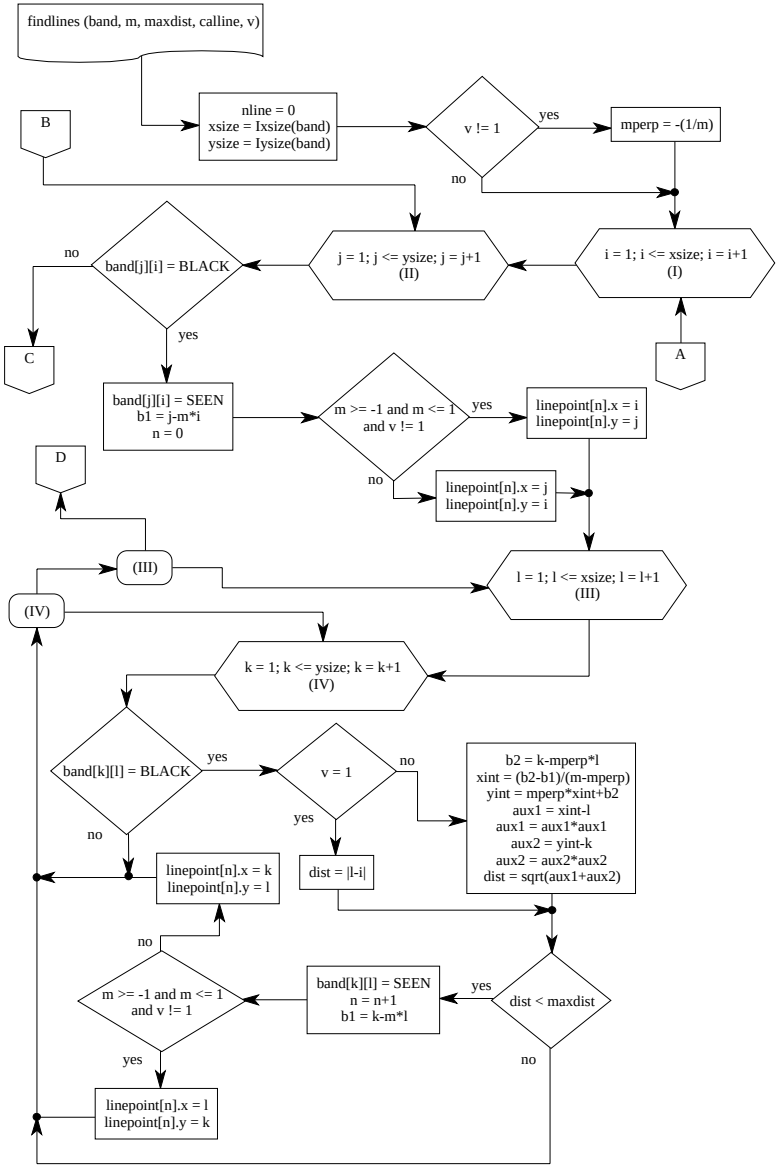
3.2.2 - Módulo lregression

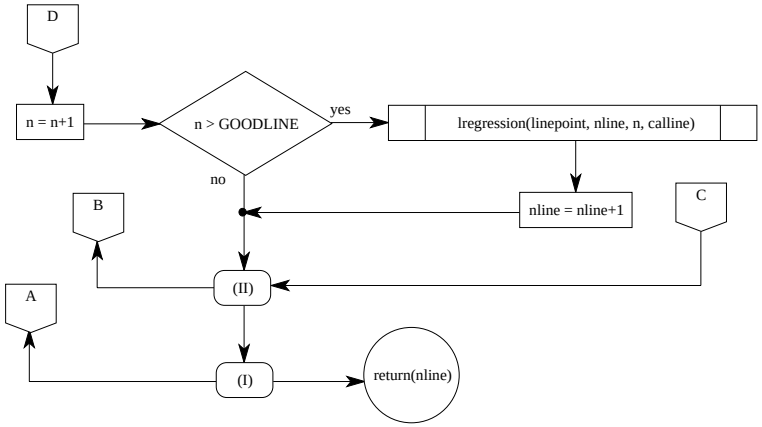


3.2.3 - Módulo mark

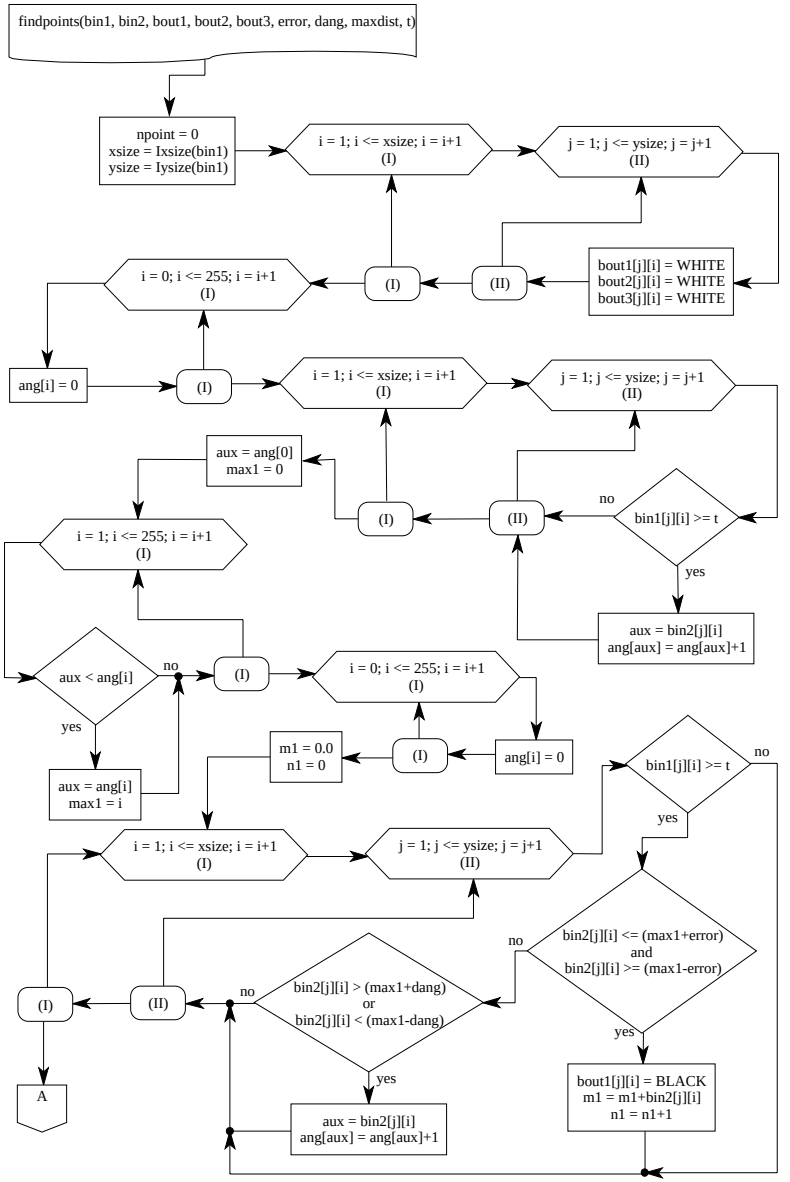


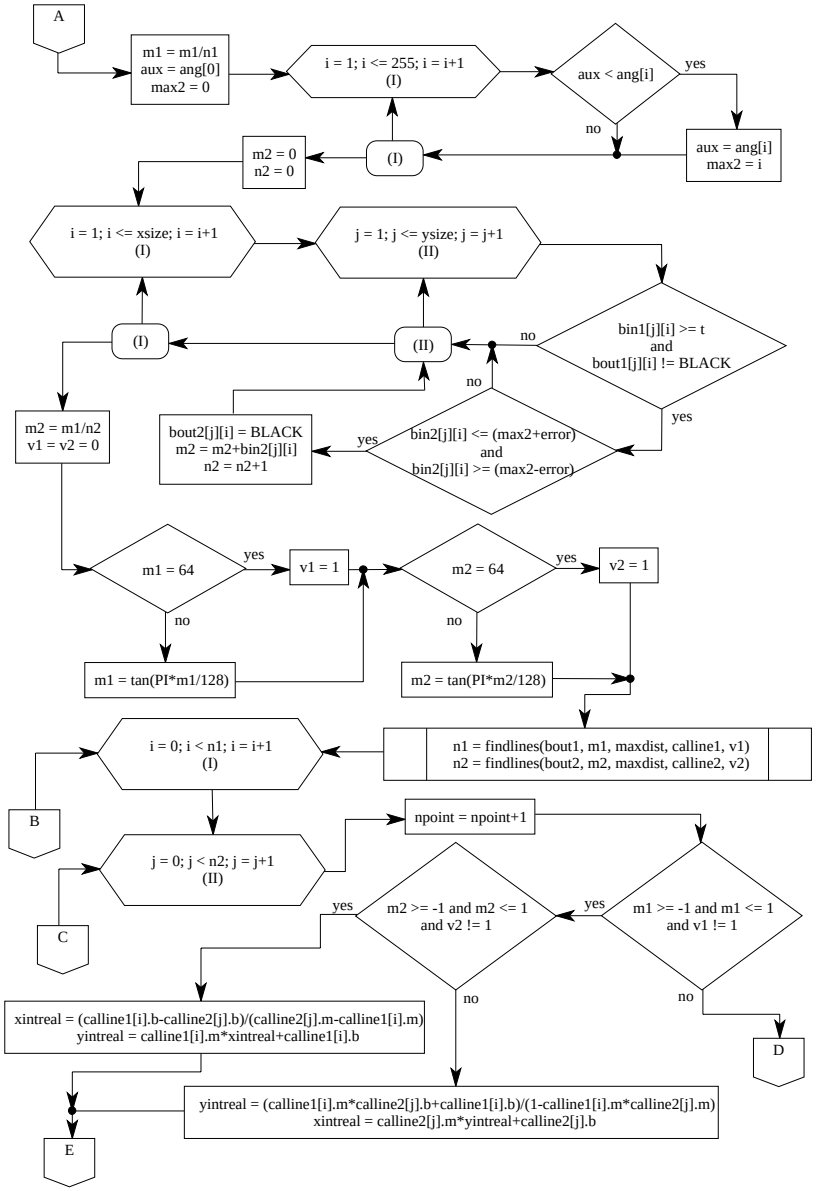
3.2.4 - Módulo findlines

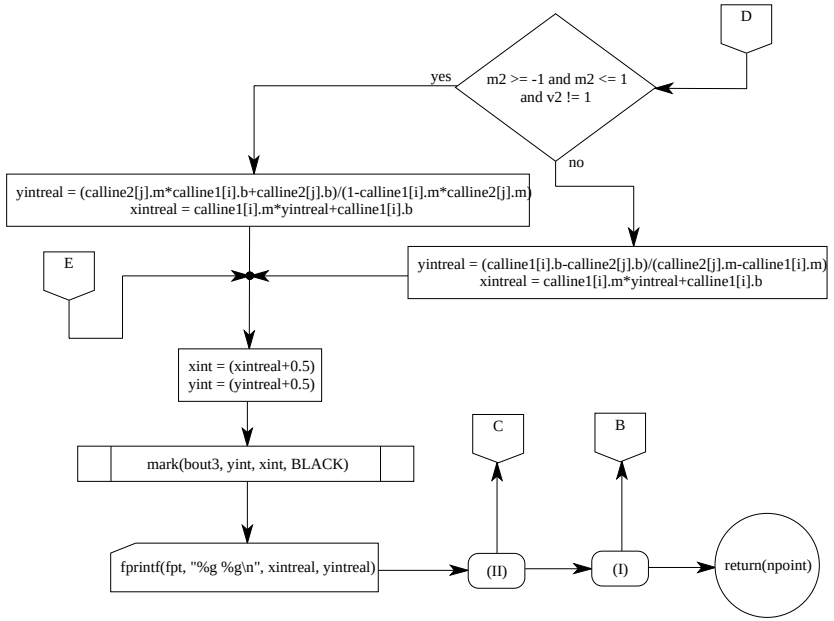




3.2.5 - Módulo findpoints







3.3 - Listagem da implementação

Apresenta-se, de seguida, a listagem da implementação ***calpoint*** desenvolvida em linguagem *C* em ambiente *UNIX* numa estação de trabalho *SunSparc* e que se destina ao ambiente de processamento de imagem *XITE*[8], do qual utiliza algumas funções e tipos de variáveis já previamente definidas. Esta implementação contém comentários para auxílio à sua compreensão, assim como informação sobre os diferentes módulos, para sua posterior extracção através da ferramenta *CDOC(1)* existente no *XITE*.

```
/*
_____________________________________________________________________

  calpoint.c
  @(#)calpoint.c   1.00 93/12/18, Copyright 1993, DEEC, FEUP
  Departamento de Engenharia Electrotecnica e de Computadores
```

[8] *X-based Image processing Tools an Environmemt*, [Lønnestad, 1992].

Faculdade de Engenharia
Rua dos Bragas
4099 PORTO CODEX
PORTUGAL
E-mail: gpai@obelix.fe.up.pt

---

biff.h
@(#)biff.h 1.23 92/04/10, Copyright 1990, Blab, UiO
Image processing lab, Department of Informatics
University of Oslo
E-mail: blab@ifi.uio.no

---

readarg.h
@(#)readarg.h 1.3 92/01/15, Copyright 1990, Blab, UiO
Image processing lab, Department of Informatics
University of Oslo
E-mail: blab@ifi.uio.no

---

message.h
@(#)message.h 1.3 92/01/15, Copyright 1990, Blab, UiO
Image processing lab, Department of Informatics
University of Oslo
E-mail: blab@ifi.uio.no

```
/*****************************************************************/

static char *SccsId = "@(#)calpoint.c 1.00 93/12/18, DEEC, FEUP";

/*****************************************************************/

/* INCLUDES */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <blab/biff.h>
#include <blab/readarg.h>
#include <blab/message.h>

/*****************************************************************/

/* DEFINES */

#define WHITE 255            /* white's value */
#define BLACK 0              /* black's value */
#define SEEN 156             /* tag used for indication that some pixel was seen already */
#define NLINES 30            /* maximum number of lines to be considered */
#define NPOINTLINE 1500  /* maximum number of points that any line could have */
#define GOODLINE 40        /* minimum number of line's points for a line could be considered */
#define MPI 3.141592654    /* pi's value */

/*****************************************************************/

/* GLOBAL VARIABLES */

FILE *fpt;
typedef struct{

  double x;    /* point's coordinate x */
  double y;    /* point's coordinate y */

  } point;

typedef struct{

  double m;   /* line's equation y=m*x+b */
  double b;

} line;

/*****************************************************************/
```

```
/*F:lregression*
_____________________________________________________________________

  lregression
_____________________________________________________________________

Name:          lregression - make the linear regression by least-squares fit of a
                             straight line

Syntax:        | lregression(linepoint, nline, n, calline)
               | point *linepoint;
               | line *calline;
               | int nline, n;

Description:   'n' is the number of points to be considered.
               'nline' is the tag of the line to be considered.
               'linepoint' is a typedef struct{double x; double y;} point; where x
               and y are the point's coordinate.
               'calline' is a typedef struct{double m; double b;} line; where m
               and b are coefficients representing the slope and the intercept,
               respectively.

Return value:  | 0 => Ok.
               | 1 => Can't make the linear regression.

Example:       | Try yourself.

Restrictions:  (n*sum(x^2)-(sum(x))^2) must be different from zero.

Author:        Joao Tavares

Id:            @(#)lregression.c   1.00   93/12/18
_____________________________________________________________________
*/

lregression(linepoint, nline, n, calline)
point *linepoint;
line *calline;
int nline, n;

{

  register int j;
  double sumx, sumy, sumxy, sumx2, xmean, ymean, aux;

  sumx = sumy = sumxy = sumx2 = 0.0;
  for (j = 0; j < n; ++j) {

    sumx += linepoint[j].x;
    sumy += linepoint[j].y;
    sumxy += linepoint[j].x*linepoint[j].y;
    sumx2 += linepoint[j].x*linepoint[j].x;

  }

  xmean = sumx/n;
  ymean = sumy/n;

  aux = n*sumx2-sumx*sumx;
  if (aux == 0.0) exit(Error(1,"\nCan't make the linear regression.\n"));

  /* save m and b of this line */

  calline[nline].m = (n*sumxy-sumx*sumy)/aux;
  calline[nline].b = ymean-calline[nline].m*xmean;

  return(0);

}

/*********************************************************************/

/*F:mark*
_____________________________________________________________________

  mark
_____________________________________________________________________

Name:          mark - mark one cross

Syntax:        | int mark(band, y, x, color)
               | IBAND band;
               | int x, y, color;

Description:   'band' is the image's band.
               'x' and 'y' are the coordinate of the cross's center.
               'color' is the value for all the cross's pixels.

Return value:  | 0 => Ok.
```

```
Example:        | Try yourself.

Restrictions:   None.

Author:         Joao Tavares

Id:             @(#)mark.c   1.00   93/12/18
_____________________________________________________________________
*/

int mark(band, y, x, color)
IBAND band;
int x, y, color;

{

  int xsize, ysize, aux1, aux2;
  register int i;
  xsize = Ixsize(band);
  ysize = Iysize(band);

  for (i = -4; i <= 4 ; ++i) {

    aux1 = y-(i);
    aux2 = x-(i);
    if (aux1 < 1 || aux1 > ysize) continue;
    if (aux2 < 1 || aux2 > xsize) continue;

    /* make all pixel's in cross at y, x as color */

    band[aux1][x] = color;
    band[y][aux2] = color;

  }

  return(0);

}

/*********************************************************************/

/*F:findlines*
_____________________________________________________________________
```

```
findlines
_____________________________________________________________________

Name:           findlines - find all the parallel lines in an input band

Syntax:         | int findlines(band, m, maxdist, calline, v)
                | IBAND band;
                | double m, maxdist;
                | int v;
                | line *calline;

Description:    'band' is the input band with all the pixels which have more or less
                the same direction.
                'm' is the slope of the median direction in the input band.
                The maximum distance to a point belongs to some line is 'maxdist'.
                'calline' is a typedef struct{double m; double b;} line; where m
                and b are coefficients.
                If the median direction in the input band is vertical then 'v' is 1.

Return value:   | The number of lines.

Example:        | Try yourself.

Restrictions:   None.

Author:         Joao Tavares

Id:             @(#)findlines.c   1.00   93/12/18
_____________________________________________________________________
*/

int findlines (band, m, maxdist, calline, v)
IBAND band;
double m, maxdist;
int v;
line *calline;

{

  point linepoint[NPOINTLINE];
  double mperp, b1, b2, xint, yint, dist, aux1, aux2;
  int xsize, ysize, n, nline;
  register int i, j, l, k;

  nline = 0;
```

```
xsize = Ixsize(band);
ysize = Iysize(band);
if (v != 1) mperp = -(1.0/m);

/* find the first point of a line */

for (i = 1; i <= xsize; ++i) {

  for (j = 1; j <= ysize; ++j) {

    if (band[j][i] == BLACK) {

      band[j][i] = SEEN;
      b1 = j-m*i;
      n = 0;
      if (m >= -1 && m <= 1 && v != 1) {

        linepoint[n].x = i;
        linepoint[n].y = j;

      }
      else {

        linepoint[n].x = j;
        linepoint[n].y = i;

      }

      /* find all the line's points */

      for (l = 1; l <= xsize; ++l) {

        for (k = 1; k <= ysize; ++k) {

          if (band[k][l] == BLACK) {

            if (v == 1) dist = abs(l-i);
            else {

              b2 = (double)k-mperp*(double)l;
              xint = (b2-b1)/(m-mperp);
              yint = mperp*xint+b2;
              aux1 = xint-(double)l;
              aux1 *= aux1;
              aux2 = yint-(double)k;
              aux2 *= aux2;
              dist = sqrt(aux1+aux2);

            }
            if (dist < maxdist) {

              /* this point belong to the line */

              band[k][l] = SEEN;
              n += 1;
              b1 = k-m*l;
              if (m >= -1 && m <= 1 && v != 1) {

                linepoint[n].x = l;
                linepoint[n].y = k;

              }
              else {

                linepoint[n].x = k;
                linepoint[n].y = l;

              }

            }

          }

        }

      n += 1;
      if (n > GOODLINE) {

        /* it's a line, make the linear regression of line's points */

        lregression(linepoint, nline, n, calline);
        nline += 1;

      }

    }

  }
```

```
  }

  return(nline);

}

/*****************************************************************************/

/*F:findpoints*
_____________________________________________________________________________

  findpoints
_____________________________________________________________________________

Name:        findpoints - find the corner's points of parallel squares or rectangles
                          with same dimension

Syntax:    | int findpoints(bin1, bin2, bout1, bout2, bout3, error, dang, maxdist, t)
           | IBAND bin1, bin2, bout1, bout2, bout3;
           | int error, dang, t;
           | double maxdist;

Description:  The band 'bin1' have the amplitude and the band 'bin2' have the
              directions [0-PI] of the output results of an edges detector, for
              example Deriche.
              The error tolerance for the direction threshold is 'error'.
              The minimum difference between the two levels of direction
              threshold is 'dang'.
              The maximum distance to a point belongs to some line is 'maxdist'.
              The amplitude threshold level is 't'.
              The output band 'bout1' have all the points which belongs to lines
              with one direction.
              The output band 'bout2' have all the points which belongs to lines
              with the other direction.
              The output band 'bout3' have all the corners found mark with one
              cross.

Return value: | The number of points found.

Example:      | Try yourself.

Restrictions: None.

Author:       Joao Tavares

Id:           @(#)findpoints.c   1.00   93/12/18
_____________________________________________________________________________
*/

int findpoints(bin1, bin2, bout1, bout2, bout3, error, dang, maxdist, t)
IBAND bin1, bin2, bout1, bout2, bout3;
int error, dang, t;
double maxdist;

{

 register int i, j;
 int xsize, ysize, ang[256], xint, yint, aux, max1, max2, n1, n2, npoint, v1, v2;
 float xintreal, yintreal;
 double m1, m2;
 line calline1[NLINES], calline2[NLINES];

 npoint = 0;
 xsize = Ixsize(bin1);
 ysize = Iysize(bin1);

 /* make all pixels in bout1, bout2 and bout3, as WHITE */

 for (i = 1; i <= xsize; ++i) {

  for (j = 1; j <= ysize; ++j) {

    bout1[j][i] = WHITE;
    bout2[j][i] = WHITE;
    bout3[j][i] = WHITE;

  }

 }

 /* angle threshold */

 for (i = 0; i <= 255; ++i) ang[i] = 0;
 for (i = 1; i <= xsize; ++i) {

  for (j = 1; j <= ysize; ++j) {

    if (bin1[j][i] >= t) {
```

```
      aux = bin2[j][i];
      ang[aux] += 1;

    }

  }

}
aux = ang[0];
max1 = 0;
for (i = 1; i <= 255; ++i) {

  if (aux < ang[i]) {

    aux = ang[i];
    max1 = i;

  }

}
for (i = 0; i <= 255; ++i) ang[i] = 0;
m1 = 0.0;
n1 = 0;
for (i = 1; i <= xsize; ++i) {

  for (j = 1; j <= ysize; ++j) {

    if (bin1[j][i] >= t) {

      if (bin2[j][i] <= (max1+error) && bin2[j][i] >= (max1-error))
      {

        /* make the pixel in bout1 as BLACK  and compute the average direction */

        bout1[j][i] = BLACK;
        m1 += bin2[j][i];
        n1 += 1;

      }
      else if (bin2[j][i] > (max1+dang) || bin2[j][i] < (max1-dang))
        {

        aux = bin2[j][i];
        ang[aux] += 1;
```



```
        }

    }

  }

}
m1 /= n1;
aux = ang[0];
max2 = 0;
for (i = 1; i <= 255; ++i) {

  if (aux < ang[i]) {

    aux = ang[i];
    max2 = i;

  }

}
m2 = 0.0;
n2 = 0;
for (i = 1; i <= xsize; ++i) {

  for (j = 1; j <= ysize; ++j) {

    if (bin1[j][i] >= t && bout1[j][i] != BLACK) {

      if (bin2[j][i] <= (max2+error) && bin2[j][i] >= (max2-error)) {

        /* make the pixel in bout2 as BLACK and compute the average direction */

        bout2[j][i] = BLACK;
        m2 += bin2[j][i];
        n2 += 1;

      }

    }

  }

}
m2 /= n2;
```

```
/* compute the line's angles */

v1 = v2 = 0;
if (m1 == 64.0) v1 = 1;
else m1 = tan(MPI*m1/128.0);
if (m2 == 64.0) v2 = 1;
else m2 = tan(MPI*m2/128.0);

/* find lines in bout1 */

n1 = findlines(bout1, m1, maxdist, calline1, v1);

/* find lines in bout2 */

n2 = findlines(bout2, m2, maxdist, calline2, v2);

/* find the intersection points */

for (i = 0; i < n1; ++i) {

  for (j = 0; j < n2; ++j) {

    npoint += 1;
    if (m1 >= -1.0 && m1 <= 1.0 && v1 != 1) {

      if (m2 >= -1.0 && m2 <= 1.0 && v2 != 1) {

        xintreal = (calline1[i].b-calline2[j].b)/(calline2[j].m-calline1[i].m);
        yintreal = calline1[i].m*xintreal+calline1[i].b;

      }
      else {

        yintreal = (calline1[i].m*calline2[j].b+calline1[i].b)/(1.0-calline1[i].m*calline2[j].m);
        xintreal = calline2[j].m*yintreal+calline2[j].b;

      }

    }
    else {

      if(m2 >= -1.0 && m2 <= 1.0 && v2 != 1) {

        yintreal = (calline2[j].m*calline1[i].b+calline2[j].b)/(1.0-calline1[i].m*calline2[j].m);
        xintreal = calline1[i].m*yintreal+calline1[i].b;

      }
      else {

        yintreal = (calline1[i].b-calline2[j].b)/(calline2[j].m-calline1[i].m);
        xintreal = calline1[i].m*yintreal+calline1[i].b;

      }

    }

    /* make a cross in bout3 at yint, xint with color BLACK */

    xint = (xintreal+0.5);
    yint = (yintreal+0.5);
    mark(bout3, yint, xint, BLACK);

    /* write in output file the intersection point */

    fprintf(fpt, "%g %g\n", xintreal, yintreal);

  }

 }

 return(npoint);

}

/*****************************************************************************/

/*P:calpoint*
_____________________________________________________________________________

 calpoint
_____________________________________________________________________________

Name:          calpoint - find the corners of parallel squares or rectangles with
                          same dimension in an image

Syntax:     | [-ti <title>] [-e <erthresang>] [-d <difangmax>] [-m <maxdist>]
            | [-t <thresamp>] <inimage> <outimage> <outfile>

Description:  'calpoint' find the corners of parallel squares or rectangles with
               same dimension in an input image 'inimage'. This image must
```

```
                have two bands, the first with the amplitude and the second with
                the directions of the output results of an edges detector, for
                example Deriche.
                The error tolerance for the direction threshold is indicate trough
                flag '-e', by default 'erthresang' is 2.
                The minimum difference between the two levels of direction
                threshold is indicate through flag 'd', by default 'difangmax' is 32.
                The maximum distance to a point belongs to some line is indicate
                through flag '-m', by default 'maxdist' is 10.0.
                The amplitude threshold level is indicate through flag '-t', by
                default 'thresamp' is 50.
                The 'outimage' image will have the 'title' which is indicate through
                flag '-ti', by default 'title' is "Calibration points". This image will
                have three bands. The first with all the points which belongs to lines
                with one direction, the second with all the points which belongs to
                lines with the other direction, and the last with all the corners found.
                The output 'outfile' file contain the image coordinate for all the
                corners found.

Return value:   | 1 => Can't read image <inimage>.
                | 2 => The <inimage> doesn't have two bands.
                | 3 => Bad input pixel type for band[1].
                | 4 => Bad input pixel type for band[2].
                | 5 => The input bands have different size.
                | 6 => Can't make image <outimage>.
                | 7 => Can't make band[1] of output image <outimage>.
                | 8 => Can't make band[2] of output image <outimage>.
                | 9 => Can't make band[3] of output image <outimage>.
                | 10 => Can't open file <outfile>.

Examples:       | Try yourself.

Restrictions:   Accepts only UNS_BYTE pixels.
                Images with only two bands, the first one must have the amplitude
                values and the second must have the directions values [0-PI], of
                the output results of an edges detector.

Author:         Joao Tavares

Id:             @(#)calpoint.c   1.0   94/12/18
_____________________________________________________________________
*/

#ifdef MAIN

main(argc, argv)
int argc;
char **argv;

{

  IMAGE imgin, imgout;
  int xsize1, ysize1, xsize2, ysize2, t, error, dang, npoint;
  double maxdist;
  char *title, arg[200];

  InitMessage(&argc, argv, "Usage: %s\n [-ti <title>] [-e <erthresang>] [-d <difangmax>] [-m
<maxdist>] [-t <thresamp>] <inimage> <outimage> <outfile>\n");
  Iset_message(1);
  Iset_abort(1);

  /* read switches */

  title = read_switch(&argc, argv, "-ti", 1, "Calibration points");
  error = atoi(read_switch(&argc, argv, "-e", 1, "2"));
  dang = atoi(read_switch(&argc, argv, "-d", 1, "32"));
  maxdist = atof(read_switch(&argc, argv, "-m", 1, "10.0"));
  t = atoi(read_switch(&argc, argv, "-t", 1, "50"));

  if (argc == 1) Usage(1, (char*)0);
  if (argc != 4) Usage(2, "\nBad number of arguments.\n");

  /* read input image */

  imgin = Iread_image(argv[1]);
  if (imgin == NULL) exit(Error(1, "\nCan't read image %s.\n", argv[1]));
  if (Inbands(imgin) != 2) exit(Error(2, "\nThe %s doesn't have two bands.\n", argv[1]));
  if (Ipixtyp(imgin[1]) != Iu_byte_typ) exit(Error(3, "\nBad input pixel type for band[1].\n"));
  if (Ipixtyp(imgin[2]) != Iu_byte_typ) exit(Error(4, "\nBad input pixel type for band[2].\n"));
  xsize1 = Ixsize(imgin[1]);
  ysize1 = Iysize(imgin[1]);
  xsize2 = Ixsize(imgin[2]);
  ysize2 = Iysize(imgin[2]);
  if (xsize1 != xsize2 || ysize1 != ysize2) exit(Error(5, "\nThe input bands have different size.\n"));

  /* make output image */

  imgout = Init_image(3, title);
  if (imgout == NULL) exit(Error(6, "\nCan't make image %s.\n", argv[2]));
```

```
  /* make bands of output image */

  imgout[1] = Imake_band(Iu_byte_typ, xsize1, ysize1);
  if (imgout[1] == NULL) exit(Error(7, "\nCan't make band[1] of output image %s.\n", argv[2]));
  imgout[2] = Imake_band(Iu_byte_typ, xsize1, ysize1);
  if (imgout[2] == NULL) exit(Error(8, "\nCan't make band[2] of output image %s.\n", argv[2]));
  imgout[3] = Imake_band(Iu_byte_typ, xsize1, ysize1);
  if (imgout[3] == NULL) exit(Error(9, "\nCan't make band[3] of output image %s.\n", argv[2]));

  /* open output file */

  fpt = fopen(argv[3], "a");
  if (fpt == NULL) exit(Error(10, "\nCan't open file %s.\n", argv[3]));

  /* call function findpoints */

  npoint = findpoints(imgin[1], imgin[2], imgout[1], imgout[2], imgout[3], error, dang, maxdist, t);

  /* close output file */

  fclose(fpt);

  /* output results */

  printf("\n%s found %i calibration points.\n", argv[0], npoint);

  /* write output image with history */

  sprintf(arg, "\nResults for %s in %s with Error tolerance for angle threshold: %i, Difference of
angle threshold levels: %i, Maxim distance for a point belongs to a line: %g, Level for amplitude
threshold: %i.", argv[0], argv[1], error, dang, maxdist, t);

  Ihistory(imgout, argv[0], arg);
  Iwrite_image(imgout,argv[2]);

}

#endif


/*****************************************************************
```

4 - Simulador de uma dada câmara: simcamera

Nesta secção é apresentada uma implementação de um simulador de uma dada câmara, segundo o método descrito em [Tavares, 1995]. Neste método, é utilizado o modelamento teórico do processo de aquisição de imagem proposto em [Tsai, 1987] e apresentado em [Tavares, 1995].

Esta implementação determina automaticamente as coordenadas 3D mundo (x_w, y_w, z_w) para cada ponto de calibração, de modo que o conjunto coplanar dos pontos de calibração fique uniformemente distribuído no plano de calibração.

Uma possível utilização desta implementação é o fornecimento de dados simulados para teste da implementação ***calcamera*** descrita anteriormente.

4.1 - Descrição da implementação

Esta implementação é constituída apenas pelo módulo ***principal***. Este módulo é responsável por:

- ✓ Leitura dos argumentos opcionais ("*switches*"):
  - *npoint*, ("[-n <npoint>]"), número de pontos de calibração pretendido. Por defeito, é igual a *36*.
  - *zwplane*, ("[-z <zwplane>]"), coordenada 3D mundo *z* para o plano de calibração. Por defeito, é igual a *0.0*.
  - *xcenterbuf*, ("[-cx <xcenterbuf>]"), coordenada do centro da imagem na memória *frame* segundo a direcção *x*, C_x. Por defeito, é igual a metade da dimensão da memória *frame* segundo a mesma direcção.
  - *ycenterbuf*, ("[-cy <ycenterbuf>]"), coordenada do centro da imagem na memória *frame* segundo a direcção *y*, C_y. Por defeito, é igual a metade da dimensão da memória *frame* segundo a mesma direcção.
- ✓ Leitura dos argumentos passados à implementação pela linha de comando:
  - *xsizebuf*, ("<xsizebuf>"), dimensão da memória *frame* segundo a direcção *x*.
  - *ysizebuf*, ("<ysizebuf>"), dimensão da memória *frame* segundo a direcção *y*.
  - *rotx*, ("<rotx>"), valor em grau da rotação segundo o eixo *x*.
  - *roty*, ("<roty>"), valor em grau da rotação segundo o eixo *y*.
  - *rotz*, ("<rotz>"), valor em grau da rotação segundo o eixo *z*.
  - *tx*, ("<tx>"), translação segundo o eixo *x*.
  - *ty*, ("<ty>"), translação segundo o eixo *y*.
  - *tz*, ("<tz>"), translação segundo o eixo *z*.
  - *f*, ("<f>"), distância focal efectiva, *f*.
  - *dx*, ("<dx>"), distância entre centros dos elementos sensores vizinhos na direcção *x*, d_x.
  - *dy*, ("<dy>"), distância entre centros dos sensores CCD vizinhos na direcção *y*, d_y.
  - *sx*, ("<sx>"), factor de incerteza horizontal, s_x.
  - *k1*, ("<k1>"), factor de distorção radial da lente, k_1.

- *filename*, (“<filename>”), nome do ficheiro para escrita dos resultados obtidos, nomeadamente as coordenadas 3D mundo (x_w, y_w, z_w) e as coordenadas na memória *frame* (X_f, Y_f), para cada ponto de calibração.

✓ Abertura do ficheiro de saída dos resultados.

✓ Apresentação das componentes determinadas da matriz de rotação 3D do sistema de coordenadas mundo para o sistema câmara.

✓ Execução da simulação da câmara propriamente dita.

✓ Escrita na ficheiro de saída das coordenadas 3D mundo (x_w, y_w, z_w) e das coordenadas na memória *frame* (X_f, Y_f), para todos os pontos de calibração.

✓ Fecho do ficheiro de saída de resultados.

✓ Comunicação da existência de erros em argumentos de entrada ou ao longo do decurso da execução:

- Através de valores de retorno:
  - 1, (“Wrong value for number of points <npoint>, the sqrt of npoint must be integer.”), a raiz quadrada do valor especificado para o número de pontos de calibração não é inteira.
  - 2, (“Wrong value for rotation about x axis <rotx>; it must be greater than -360 and smaller than 360.”), o valor especificado para *rotx* não está compreendido entre *-360* e *360* grau.
  - 3, (“Wrong value for rotation about y axis <roty>; it must be greater than -360 and smaller than 360.”), o valor especificado para *roty* não está compreendido entre *-360* e *360* grau.
  - 4, (“Wrong value for rotation about z axis <rotz>; it must be greater than -360 and smaller than 360.”), o valor especificado para *rotz* não está compreendido entre *-360* e *360* grau.
  - 5, (“Can’t open file <filename>.”), o módulo não consegue abrir o ficheiro de saída especificado.
- Através de mensagens:
  - 1, (*InitMessage*), quando o número de argumentos é igual a um, isto é, apenas o nome da implementação.
  - 2, (“Bad number of arguments.”), quando o número de argumentos é diferente de quinze.

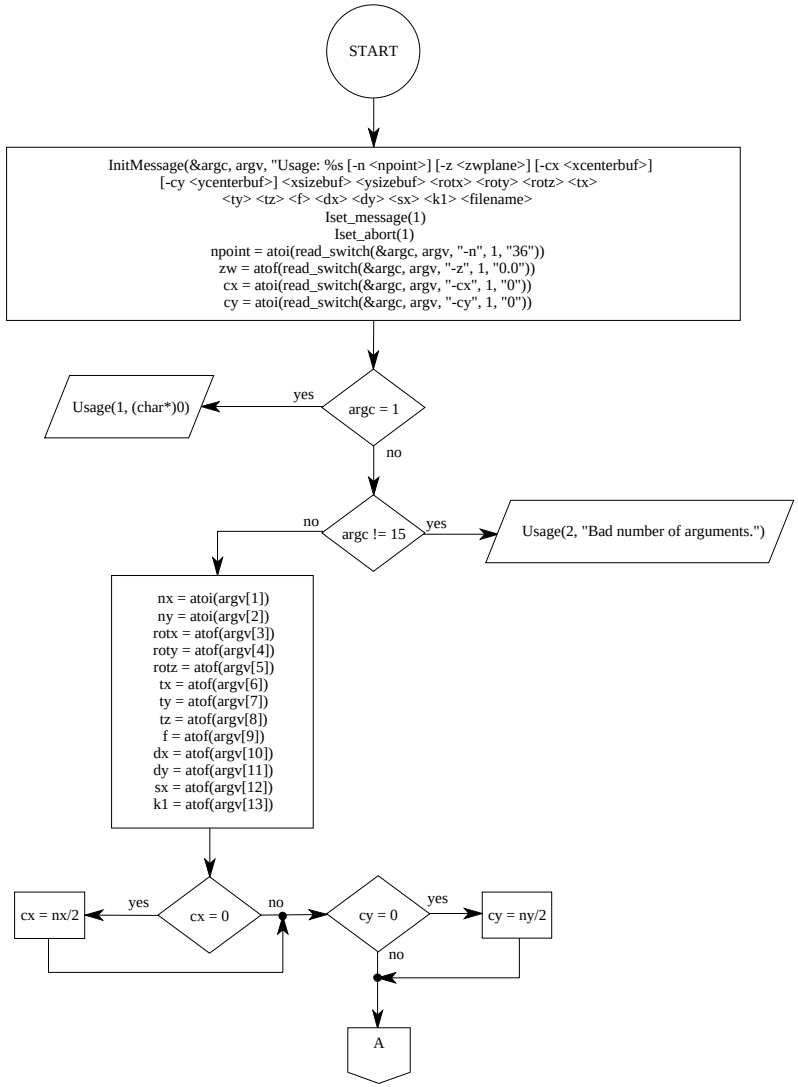
Como restrições na utilização desta implementação têm-se:

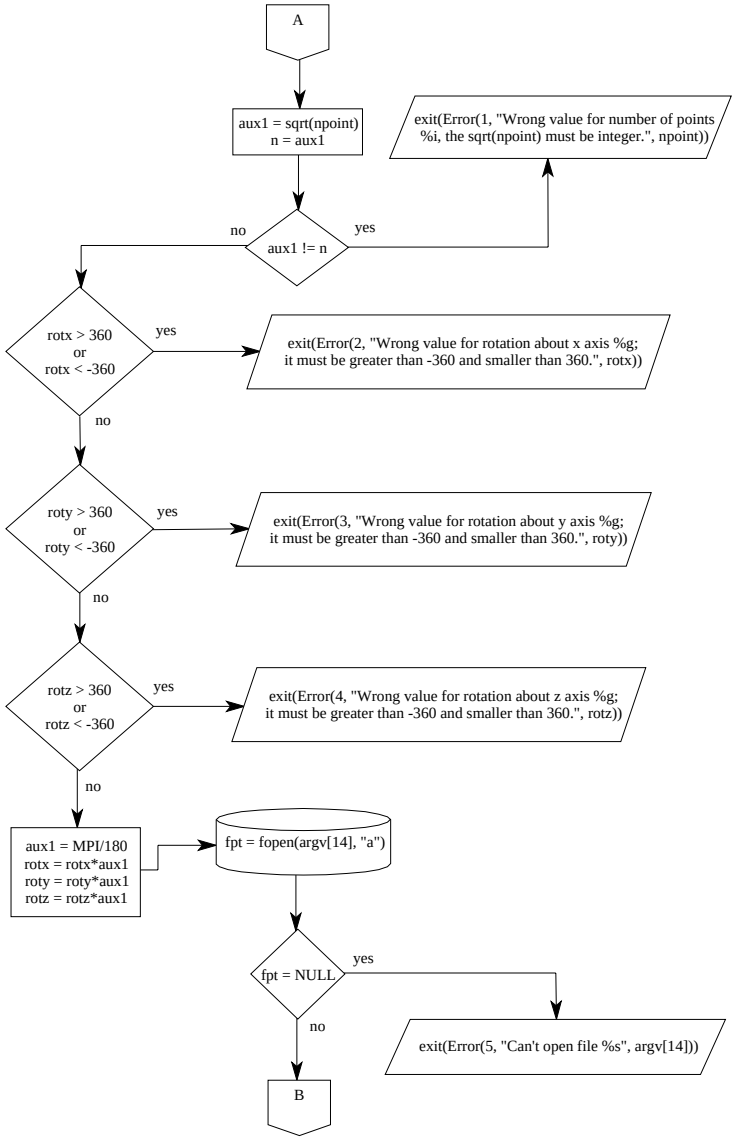
⌧ O valor especificado para o número de pontos de calibração pretendidos, *npoint*, deve ter raiz quadrada inteira, como por exemplo *16*, *36*, *49*, etc.

⌧ Os valores especificados para as rotações segundo os três eixos principais: *rotx*, *roty* e *rotz*, deverão estar definidos em grau e compreendidos entre *-360* e *360* grau.

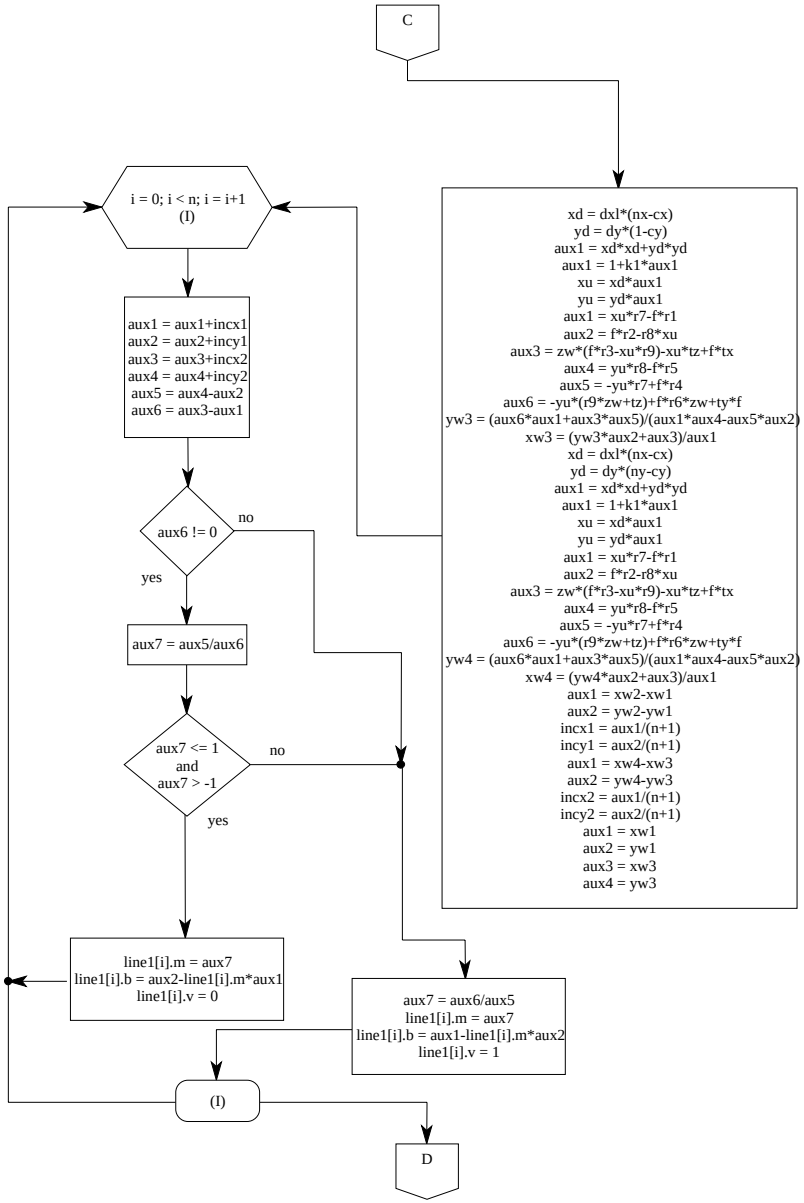
4.2 - Fluxograma da implementação

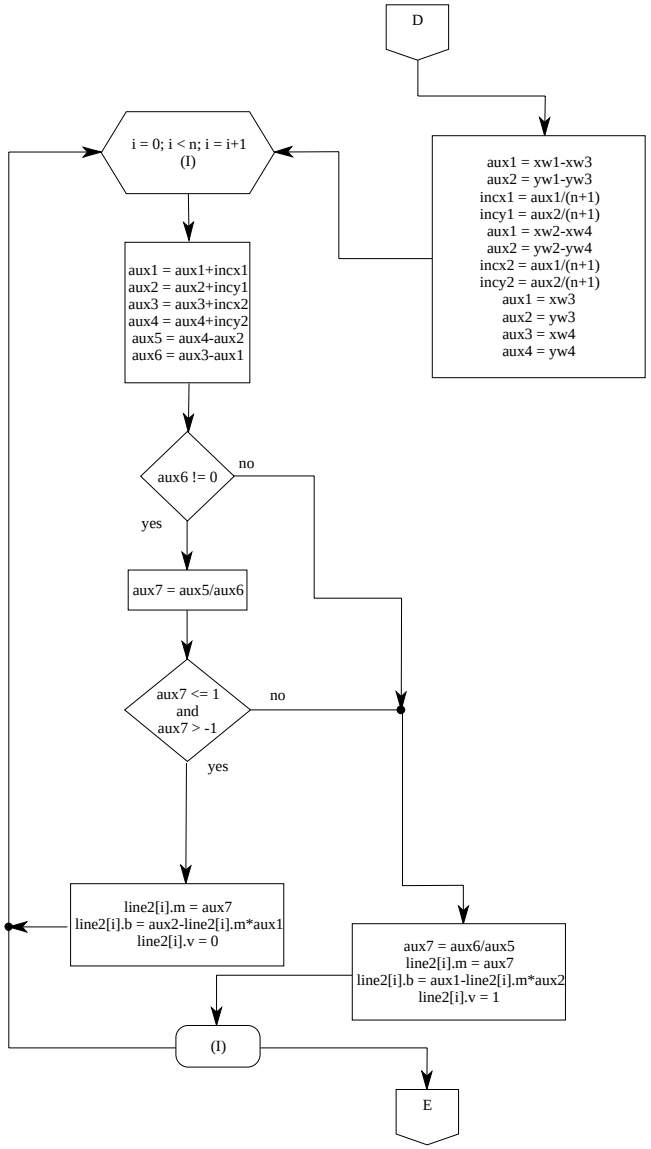
Apresenta-se, de seguida, o fluxograma da implementação ***simcamera***.

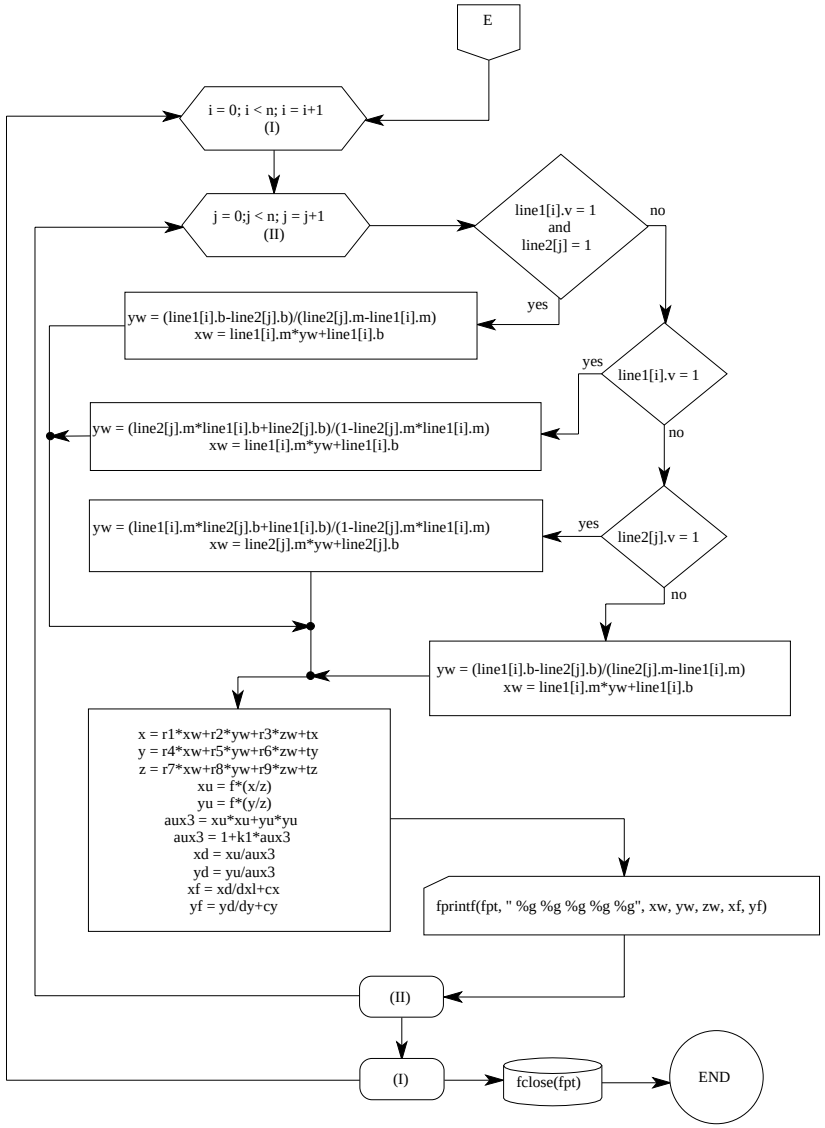





B
dxl = sx*dx
aux1 = cos(rotx)
aux2 = sin(rotx)
aux3 = cos(roty)
aux4 = sin(roty)
aux5 = cos(rotz)
aux6 = sin(rotz)
r1 = aux3*aux5
r2 = aux3*aux6
r3 = -aux4
r4 = aux5*aux2*aux4-aux1*aux6
r5 = aux6*aux2*aux4+aux1*aux5
r6 = aux2*aux3
r7 = aux5*aux4*aux1+aux2*aux6
r8 = -aux2*aux5+aux6*aux4*aux1
r9 = aux3*aux1
printf("Matrix R: r1=%g r2=%g r3=%g r4=%g r5=%g r6=%g r7=%g r8=%g r9=%g", r1, r2, r3, r4, r5, r6, r7, r8, r9)
xd = dxl*(1-cx)
yd = dy*(1-cy)
aux1 = xd*xd+yd*yd
aux1 = 1+k1*aux1
xu = xd*aux1
yu = yd*aux1
aux1 = xu*r7-f*r1
aux2 = f*r2-r8*xu
aux3 = zw*(f*r3-xu*r9)-xu*tz+f*tx
aux4 = yu*r8-f*r5
aux5 = -yu*r7+f*r4
aux6 = -yu*(r9*zw+tz)+f*r6*zw+ty*f
yw1 = (aux6*aux1+aux3*aux5)/(aux1*aux4-aux5*aux2)
xw1 = (yw1*aux2+aux3)/aux1
xd = dxl*(1-cx)
yd = dy*(ny-cy)
aux1 = xd*xd+yd*yd
aux1 = 1+k1*aux1
xu = xd*aux1
yu = yd*aux1
aux1 = xu*r7-f*r1
aux2 = f*r2-r8*xu
aux3 = zw*(f*r3-xu*r9)-xu*tz+f*tx
aux4 = yu*r8-f*r5
aux5 = -yu*r7+f*r4
aux6 = -yu*(r9*zw+tz)+f*r6*zw+ty*f
yw2 = (aux6*aux1+aux3*aux5)/(aux1*aux4-aux5*aux2)
xw2 = (yw2*aux2+aux3)/aux1
C


C
i = 0; i < n; i = i+1
(I)
aux1 = aux1+incx1
aux2 = aux2+incy1
aux3 = aux3+incx2
aux4 = aux4+incy2
aux5 = aux4-aux2
aux6 = aux3-aux1
aux6 != 0
no
yes
aux7 = aux5/aux6
aux7 <= 1
and
aux7 > -1
no
yes
line1[i].m = aux7
line1[i].b = aux2-line1[i].m*aux1
line1[i].v = 0
aux7 = aux6/aux5
line1[i].m = aux7
line1[i].b = aux1-line1[i].m*aux2
line1[i].v = 1
(I)
D
xd = dxl*(nx-cx)
yd = dy*(1-cy)
aux1 = xd*xd+yd*yd
aux1 = 1+k1*aux1
xu = xd*aux1
yu = yd*aux1
aux1 = xu*r7-f*r1
aux2 = f*r2-r8*xu
aux3 = zw*(f*r3-xu*r9)-xu*tz+f*tx
aux4 = yu*r8-f*r5
aux5 = -yu*r7+f*r4
aux6 = -yu*(r9*zw+tz)+f*r6*zw+ty*f
yw3 = (aux6*aux1+aux3*aux5)/(aux1*aux4-aux5*aux2)
xw3 = (yw3*aux2+aux3)/aux1
xd = dxl*(nx-cx)
yd = dy*(ny-cy)
aux1 = xd*xd+yd*yd
aux1 = 1+k1*aux1
xu = xd*aux1
yu = yd*aux1
aux1 = xu*r7-f*r1
aux2 = f*r2-r8*xu
aux3 = zw*(f*r3-xu*r9)-xu*tz+f*tx
aux4 = yu*r8-f*r5
aux5 = -yu*r7+f*r4
aux6 = -yu*(r9*zw+tz)+f*r6*zw+ty*f
yw4 = (aux6*aux1+aux3*aux5)/(aux1*aux4-aux5*aux2)
xw4 = (yw4*aux2+aux3)/aux1
aux1 = xw2-xw1
aux2 = yw2-yw1
incx1 = aux1/(n+1)
incy1 = aux2/(n+1)
aux1 = xw4-xw3
aux2 = yw4-yw3
incx2 = aux1/(n+1)
incy2 = aux2/(n+1)
aux1 = xw1
aux2 = yw1
aux3 = xw3
aux4 = yw3


D
i = 0; i < n; i = i+1
(I)
aux1 = xw1-xw3
aux2 = yw1-yw3
incx1 = aux1/(n+1)
incy1 = aux2/(n+1)
aux1 = xw2-xw4
aux2 = yw2-yw4
incx2 = aux1/(n+1)
incy2 = aux2/(n+1)
aux1 = xw3
aux2 = yw3
aux3 = xw4
aux4 = yw4
aux1 = aux1+incx1
aux2 = aux2+incy1
aux3 = aux3+incx2
aux4 = aux4+incy2
aux5 = aux4-aux2
aux6 = aux3-aux1
aux6 != 0
no
yes
aux7 = aux5/aux6
aux7 <= 1
and
aux7 > -1
no
yes
line2[i].m = aux7
line2[i].b = aux2-line2[i].m*aux1
line2[i].v = 0
aux7 = aux6/aux5
line2[i].m = aux7
line2[i].b = aux1-line2[i].m*aux2
line2[i].v = 1
(I)
E


E
i = 0; i < n; i = i+1
(I)
j = 0;j < n; j = j+1
(II)
line1[i].v = 1
and
line2[j] = 1
no
yes
yw = (line1[i].b-line2[j].b)/(line2[j].m-line1[i].m)
xw = line1[i].m*yw+line1[i].b
line1[i].v = 1
yes
no
yw = (line2[j].m*line1[i].b+line2[j].b)/(1-line2[j].m*line1[i].m)
xw = line1[i].m*yw+line1[i].b
yw = (line1[i].m*line2[j].b+line1[i].b)/(1-line2[j].m*line1[i].m)
xw = line2[j].m*yw+line2[j].b
line2[j].v = 1
yes
no
yw = (line1[i].b-line2[j].b)/(line2[j].m-line1[i].m)
xw = line1[i].m*yw+line1[i].b
x = r1*xw+r2*yw+r3*zw+tx
y = r4*xw+r5*yw+r6*zw+ty
z = r7*xw+r8*yw+r9*zw+tz
xu = f*(x/z)
yu = f*(y/z)
aux3 = xu*xu+yu*yu
aux3 = 1+k1*aux3
xd = xu/aux3
yd = yu/aux3
xf = xd/dxl+cx
yf = yd/dy+cy
fprintf(fpt, " %g %g %g %g %g", xw, yw, zw, xf, yf)
(II)
(I)
fclose(fpt)
END

4.3 - Listagem da implementação

Apresenta-se, de seguida, a listagem da implementação ***simcamera*** desenvolvida em linguagem *C* em ambiente *UNIX* numa estação de trabalho *SunSparc* e que se destina ao ambiente de processamento de imagem *XITE*[9], do qual utiliza algumas funções e tipos de variáveis já previamente definidas. Esta implementação contém comentários para auxílio à sua compreensão, assim como informação sobre os diferentes módulos, para sua posterior extracção através da ferramenta *CDOC(1)* existente no *XITE*.

```
/*
_____________________________________________________________________

  simcamera.c
  @(#)simcamera.c   1.00 93/12/30, Copyright 1993, DEEC, FEUP
  Departamento de Engenharia Electrotecnica e de Computadores
  Faculdade de Engenharia
  Rua dos Bragas
  4099 PORTO CODEX
  PORTUGAL
  E-mail: gpai@garfield.fe.up.pt
_____________________________________________________________________

  readarg.h
  @(#)readarg.h   1.3 92/01/15, Copyright 1990, Blab, UiO
  Image processing lab, Department of Informatics
  University of Oslo
  E-mail: blab@ifi.uio.no
_____________________________________________________________________

  message.h
  @(#)message.h   1.3 92/01/15, Copyright 1990, Blab, UiO
  Image processing lab, Department of Informatics
  University of Oslo
  E-mail: blab@ifi.uio.no
_____________________________________________________________________

  Permission to use, copy, modify and distribute this software and its documentation for any purpose
and without fee is hereby granted, provided that this copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in supporting documentation and that the
name of B-lab, Department of Informatics or University of Oslo not be used in advertising or publicity
pertaining to distribution of the software without specific, written prior permission.

  B-LAB DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT
SHALL B-LAB BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL
DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA
OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER
TORTIOUS ACTION, ARISING  OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
*/

/*********************************************************************/

static char *SccsId = "@(#)simcamera.c 1.00 93/12/30, DEEC, FEUP";

/*********************************************************************/

/* INCLUDES */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <blab/readarg.h>
#include <blab/message.h>

/*********************************************************************/

/* DEFINES */

#define MPI 3.141592654    /* pi's value */
#define NLINES 10          /* maximum number of lines to be considered in one direction */

/*********************************************************************/

/* GLOBAL VARIABLES */

typedef struct{

  double m;      /* variables for the line's equation y='m'*x+'b' */
  double b;
  int v;         /* line's flag for indication that the line is vertical */

} line;
```

[9] *X-based Image processing Tools an Environmemt*, [Lønnestad, 1992].

```
/*********************************************************************/

/*P:simcamera*
_____________________________________________________________________

  simcamera
_____________________________________________________________________

Name:       simcamera - simulation of a camera using the TSAI's model and a
                        coplanar set of points.

Syntax:     | simcamera [-n <npoint>] [-z <zwplane>] [-cx <xcenterbuf>]
            | [-cy <ycenterbuf>] <xsizebuf> <ysizebuf> <rotx> <roty> <rotz>
            | <tx> <ty> <tz> <f> <dx> <dy> <sx> <k1> <filename>

Description:  'simcamera' performs the simulation of a camera using the TSAI's
              model.
              The world 3D coordinates and frame buffer coordinates of all
              points are automatically generate.
              The number of points to be considered is indicate through flag
              '-n', by default the number of points to be considered is 36.
              The world 3D coordinate z of all the points is indicate through
              flag '-z', by default 'zwplane' is zero.
              The image center in the frame buffer for direction x is indicate
              through flag '-cx', by default 'xcenterbuf' is (xsizebuf/2).
              The image center in the frame buffer for direction y is indicate
              through flag '-cy', by default 'ycenterbuf' is (ysizebuf/2).
              'rotx', 'roty' and 'rotz', are the angles (in degree) of rotation from
              object world coordinate system to the 3D camera coordinate
              system about the x, y and z, axis.
              'tx', 'ty' and 'tz', are the translations from object world coordinate
              system to the 3D camera coordinate system by the x, y and
              z axis.
              'f' is the effective focal length.
              'dx' is the center to center distance between adjacent sensor
              elements in x (scanline) direction.
              'dy' is the center to center distance between adjacent CCD sensor
              in the y direction.
              'sx' is the horizontal uncertainty factor.
              'k1' is the lens's radial distortion.
              'filename' is the file's name to write the output results; the world
              3D coordinates and the frame buffer 2D coordinates of all points.

Return value: | 1 => Wrong value for number of points <npoint>, the sqrt of
              |      npoint must be integer.
              | 2 => Wrong value for rotation about x axis <rotx>; it must be
              |      greater than -360 and smaller than 360.
              | 3 => Wrong value for rotation about y axis <roty>; it must be
              |      greater than -360 and smaller than 360.
              | 4 => Wrong value for rotation about z axis <rotz>; it must be
              |      greater than -360 and smaller than 360.
              | 5 => Can't open file <filename>.

Examples:     | Try yourself.

Restrictions: The sqrt of <npoint> must be integer.
              The values for <rotx>, <roty> and  <rotz> must be smaller than
              360 and greater than -360 degree.

Author:       Joao Tavares

Id:           @(#)simcamera.c   1.0   93/12/30
_____________________________________________________________________
*/

main(argc, argv)
int argc;
char **argv;

{

  FILE *fpt;
  register int i, j;
  int npoint, nx, ny, n, cx, cy;
  double xw, yw, zw, rotx, roty, rotz, tx, ty, tz, f, dx, dy, dxl, sx, k1, r1, r2, r3, r4, r5, r6, r7, r8, r9,
incx1, incy1, incx2, incy2, x, y, z, xu, yu, xd, yd, xf, yf, xw1, xw2, xw3, xw4, yw1, yw2, yw3, yw4,
aux1, aux2, aux3, aux4, aux5, aux6, aux7;
  line line1[NLINES], line2[NLINES];

  InitMessage(&argc, argv, "Usage: %s\n [-n <npoint>] [-z <zwplane>] [-cx <xcenterbuf>] [-cy
<ycenterbuf>] <xsizebuf> <ysizebuf> <rotx> <roty> <rotz> <tx> <ty> <tz> <f> <dx> <dy> <sx>
<k1> <filename>\n");
  Iset_message(1);
  Iset_abort(1);

  /* read switches */

  npoint = atoi(read_switch(&argc, argv, "-n", 1, "36"));
  zw = atof(read_switch(&argc, argv, "-z", 1, "0.0"));
  cx = atoi(read_switch(&argc, argv, "-cx", 1, "0"));
```

```
  cy = atoi(read_switch(&argc, argv, "-cy", 1, "0"));

  if (argc == 1) Usage(1, (char*)0);
  if (argc != 15) Usage(2, "\nBad number of arguments.\n");

  /* read argv[] */

  nx = atoi(argv[1]);
  ny = atoi(argv[2]);
  rotx = atof(argv[3]);
  roty = atof(argv[4]);
  rotz = atof(argv[5]);
  tx = atof(argv[6]);
  ty = atof(argv[7]);
  tz = atof(argv[8]);
  f = atof(argv[9]);
  dx = atof(argv[10]);
  dy = atof(argv[11]);
  sx = atof(argv[12]);
  k1 = atof(argv[13]);

  /* if cx and cy aren't given then compute cx and cy by the size of the frame buffer */

  if (cx == 0) cx = nx/2;
  if (cy == 0) cy = ny/2;

  /* npoint is correct ? */

  aux1 = sqrt((double)npoint);
  n = aux1;
  if (aux1 != n)  exit(Error(1, "\nWrong value for number of points %i, the sqrt(npoint) must be
integer.\n", npoint));

  /* rotx, roty and rotz are correct ? */

if (rotx > 360.0 || rotx < -360.0) exit(Error(2, "\nWrong value for rotation about x axis %g; it must
be greater than -360 and smaller than 360.\n", rotx));
  if (roty > 360.0 || roty < -360.0) exit(Error(3, "\nWrong value for rotation about y axis %g; it must
be greater than -360 and smaller than 360.\n", roty));
  if (rotz > 360.0 || rotz < -360.0) exit(Error(4, "\nWrong value for rotation about z axis %g; it must
be greater than -360 and smaller than 360.\n", rotz));

  aux1 = MPI/180.0;
  rotx = rotx*aux1;
  roty = roty*aux1;
  rotz = rotz*aux1;

  /* open file */

  fpt = fopen(argv[14], "a");
  if (fpt == NULL) exit(Error(5, "\nCan't open file %s.\n", argv[14]));

  dxl = sx*dx;

  /* compute the matrix R */

  aux1 = cos(rotx);
  aux2 = sin(rotx);
  aux3 = cos(roty);
  aux4 = sin(roty);
  aux5 = cos(rotz);
  aux6 = sin(rotz);

  r1 = aux3*aux5;
  r2 = aux3*aux6;
  r3 = -aux4;
  r4 = aux5*aux2*aux4-aux1*aux6;
  r5 = aux6*aux2*aux4+aux1*aux5;
  r6 = aux2*aux3;
  r7 = aux5*aux4*aux1+aux2*aux6;
  r8 = -aux2*aux5+aux6*aux4*aux1;
  r9 = aux3*aux1;

  printf("\n Matrix R: r1=%g r2=%g r3=%g r4=%g r5=%g r6=%g\n r7=%g r8=%g r9=%g\n", r1,
r2, r3, r4, r5, r6, r7, r8, r9);

  /* compute xw1, yw1, xw2, yw2, xw3, yw3, xw4, yw4 */

  xd = dxl*(1-cx);
  yd = dy*(1-cy);
  aux1 = xd*xd+yd*yd;
  aux1 = 1.0+k1*aux1;
  xu = xd*aux1;
  yu = yd*aux1;
  aux1 = xu*r7-f*r1;
  aux2 = f*r2-r8*xu;
  aux3 = zw*(f*r3-xu*r9)-xu*tz+f*tx;
  aux4 = yu*r8-f*r5;
  aux5 = -yu*r7+f*r4;
  aux6 = -yu*(r9*zw+tz)+f*r6*zw+ty*f;
```

```
yw1 = (aux6*aux1+aux3*aux5)/(aux1*aux4-aux5*aux2);
xw1 = (yw1*aux2+aux3)/aux1;

xd = dxl*(1-cx);
yd = dy*(ny-cy);
aux1 = xd*xd+yd*yd;
aux1 = 1.0+k1*aux1;
xu = xd*aux1;
yu = yd*aux1;
aux1 = xu*r7-f*r1;
aux2 = f*r2-r8*xu;
aux3 = zw*(f*r3-xu*r9)-xu*tz+f*tx;
aux4 = yu*r8-f*r5;
aux5 = -yu*r7+f*r4;
aux6 = -yu*(r9*zw+tz)+f*r6*zw+ty*f;
yw2 = (aux6*aux1+aux3*aux5)/(aux1*aux4-aux5*aux2);
xw2 = (yw2*aux2+aux3)/aux1;

xd = dxl*(nx-cx);
yd = dy*(1-cy);
aux1 = xd*xd+yd*yd;
aux1 = 1.0+k1*aux1;
xu = xd*aux1;
yu = yd*aux1;
aux1 = xu*r7-f*r1;
aux2 = f*r2-r8*xu;
aux3 = zw*(f*r3-xu*r9)-xu*tz+f*tx;
aux4 = yu*r8-f*r5;
aux5 = -yu*r7+f*r4;
aux6 = -yu*(r9*zw+tz)+f*r6*zw+ty*f;
yw3 = (aux6*aux1+aux3*aux5)/(aux1*aux4-aux5*aux2);
xw3 = (yw3*aux2+aux3)/aux1;

xd = dxl*(nx-cx);
yd = dy*(ny-cy);
aux1 = xd*xd+yd*yd;
aux1 = 1.0+k1*aux1;
xu = xd*aux1;
yu = yd*aux1;
aux1 = xu*r7-f*r1;
aux2 = f*r2-r8*xu;
aux3 = zw*(f*r3-xu*r9)-xu*tz+f*tx;
aux4 = yu*r8-f*r5;
aux5 = -yu*r7+f*r4;
aux6 = -yu*(r9*zw+tz)+f*r6*zw+ty*f;
yw4 = (aux6*aux1+aux3*aux5)/(aux1*aux4-aux5*aux2);
xw4 = (yw4*aux2+aux3)/aux1;

/* compute xw, yw, xf and yf for all points */

aux1 = xw2-xw1;
aux2 = yw2-yw1;
incx1 = aux1/(n+1);
incy1 = aux2/(n+1);
aux1 = xw4-xw3;
aux2 = yw4-yw3;
incx2 = aux1/(n+1);
incy2 = aux2/(n+1);
aux1 = xw1;
aux2 = yw1;
aux3 = xw3;
aux4 = yw3;

/* define line1[] */

for (i = 0; i < n; ++i) {

  aux1 += incx1;
  aux2 += incy1;
  aux3 += incx2;
  aux4 += incy2;
  aux5 = aux4-aux2;
  aux6 = aux3-aux1;

  if (aux6 != 0.0) {

    aux7 = aux5/aux6;
    if (aux7 <= 1.0 && aux7 >= -1.0) {

      line1[i].m = aux7;
      line1[i].b = aux2-line1[i].m*aux1;
      line1[i].v = 0;
      continue;

    }

  }

  aux7 = aux6/aux5;
  line1[i].m = aux7;
```

```
  line1[i].b = aux1-line1[i].m*aux2;
  line1[i].v = 1;

}

aux1 = xw1-xw3;
aux2 = yw1-yw3;
incx1 = aux1/(n+1);
incy1 = aux2/(n+1);
aux1 = xw2-xw4;
aux2 = yw2-yw4;
incx2 = aux1/(n+1);
incy2 = aux2/(n+1);
aux1 = xw3;
aux2 = yw3;
aux3 = xw4;
aux4 = yw4;

/* define line2[] */

for (i = 0; i < n; ++i) {

  aux1 += incx1;
  aux2 += incy1;
  aux3 += incx2;
  aux4 += incy2;
  aux5 = aux4-aux2;
  aux6 = aux3-aux1;

  if (aux6 != 0.0) {

    aux7 = aux5/aux6;
    if (aux7 <= 1.0 && aux7 >= -1.0) {

      line2[i].m = aux7;
      line2[i].b = aux2-line2[i].m*aux1;
      line2[i].v = 0;
      continue;

    }

  }
  aux7 = aux6/aux5;
  line2[i].m = aux7;
  line2[i].b = aux1-line2[i].m*aux2;
  line2[i].v = 1;

}

for (i = 0; i < n; ++i) {

  for (j = 0; j < n; ++j) {

    if (line1[i].v == 1 && line2[j].v == 1) {

      yw = (line1[i].b-line2[j].b)/(line2[j].m-line1[i].m);
      xw = line1[i].m*yw+line1[i].b;

    }
    else {

      if (line1[i].v == 1) {

        yw = (line2[j].m*line1[i].b+line2[j].b)/(1.0-line2[j].m*line1[i].m);
        xw = line1[i].m*yw+line1[i].b;

      }
      else {

        if (line2[j].v == 1) {

          yw = (line1[i].m*line2[j].b+line1[i].b)/(1.0-line1[i].m*line2[j].m);
          xw = line2[j].m*yw+line2[j].b;

        }
        else {

          xw = (line1[i].b-line2[j].b)/(line2[j].m-line1[i].m);
          yw = line1[i].m*xw+line1[i].b;

        }

      }

    }
    x = r1*xw+r2*yw+r3*zw+tx;
    y = r4*xw+r5*yw+r6*zw+ty;
    z = r7*xw+r8*yw+r9*zw+tz;
    xu = f*(x/z);
    yu = f*(y/z);
```

```
      aux3 = xu*xu+yu*yu;
      aux3 = 1.0+k1*aux3;
      xd = xu/aux3;
      yd = yu/aux3;
      xf = xd/dxl+cx;
      yf = yd/dy+cy;
      fprintf(fpt, " %g %g %g %g %g\n", xw, yw, zw, xf, yf);

    }

  }

  /* close file */

  fclose(fpt);

}

/*************************************************************************/
```

5 - Formatação dos dados de entrada para a implementação calcamera: input

Nesta secção é apresentada uma implementação para formatação dos dados de entrada para a implementação ***calcamera*** descrita anteriormente.

5.1 - Descrição da implementação

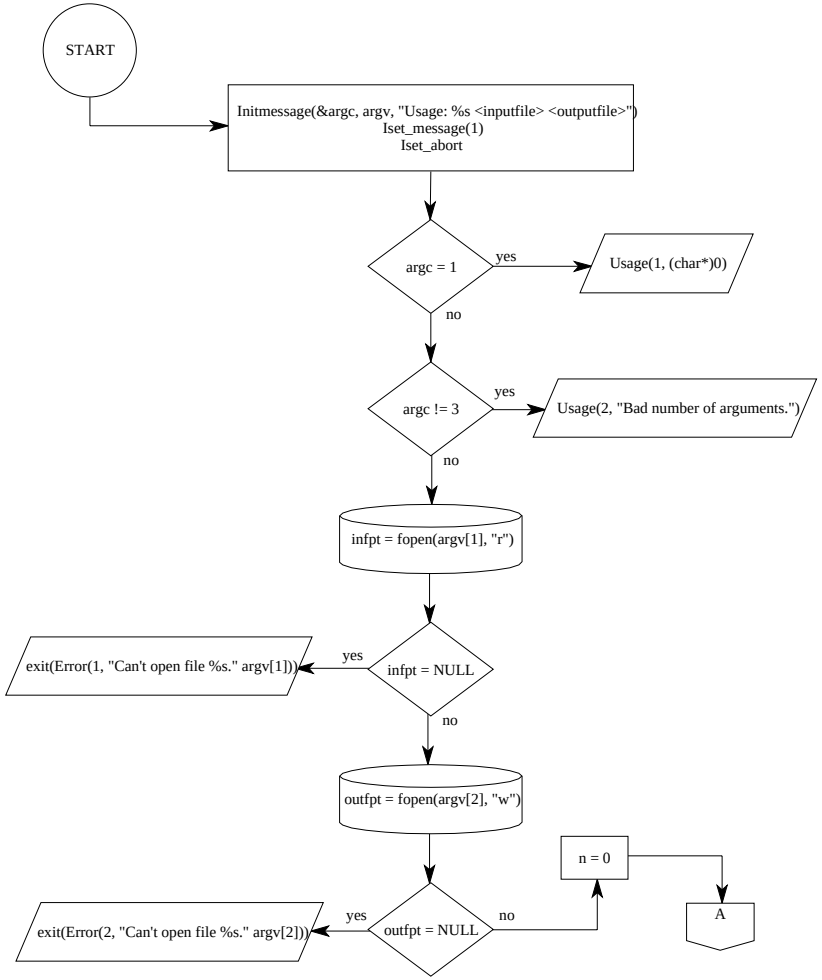
Esta implementação é constituída apenas pelo módulo ***principal***. Este módulo é responsável por:

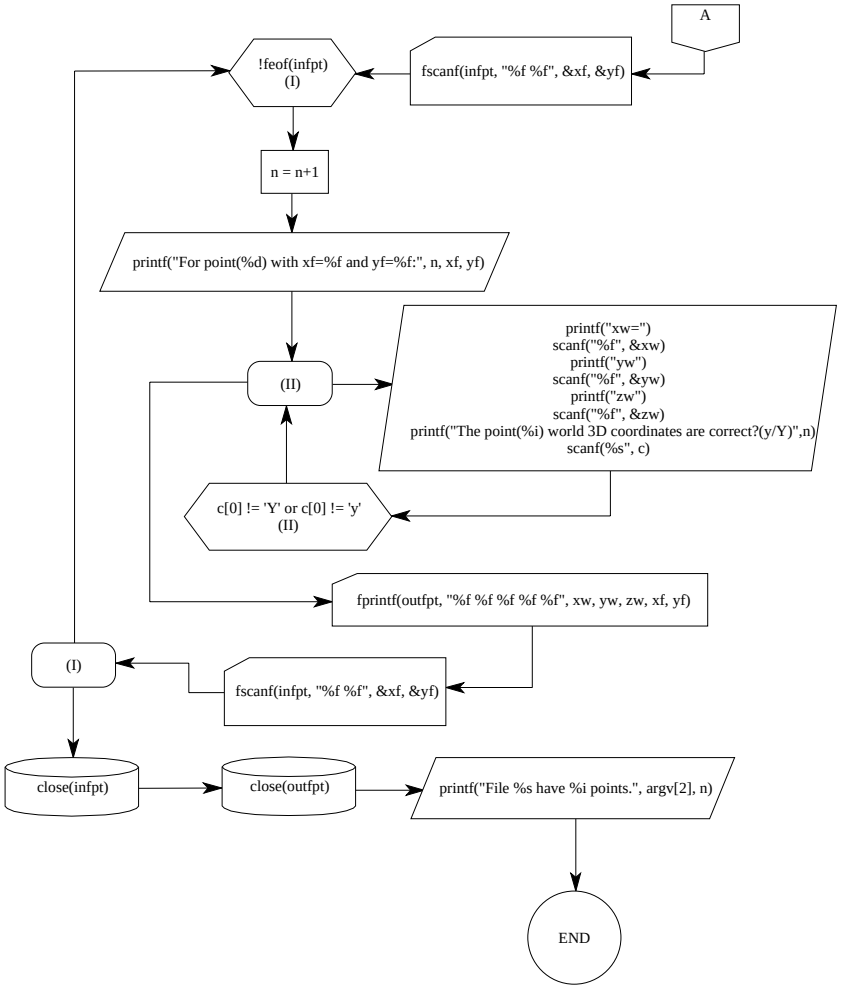
- ✓ Leitura dos argumentos passados à implementação pela linha de comando:
  - *inputfile*, ("<inputfile>"), nome do ficheiro de entrada que contém as coordenadas na memória *frame* (X_f, Y_f), para cada ponto de calibração.
  - *outputfile*, ("<outputfile>"), nome do ficheiro de saída que deverá conter, devidamente formatadas para a aplicação ***calcamera***, as coordenadas 3D mundo (x_w, y_w, z_w) e as coordenadas 2D na memória *frame* (X_f, Y_f), para cada ponto de calibração.
- ✓ Abertura do ficheiro de entrada.
- ✓ Abertura do ficheiro de saída dos resultados.
- ✓ Leitura a partir do ficheiro de entrada das coordenadas na memória *frame* (X_f, Y_f) para todos os pontos de calibração.
- ✓ Interrogação ao utilizador das coordenadas 3D mundo (x_w, y_w, z_w) para todos os pontos de calibração.
- ✓ Escrita devidamente formatada, para a implementação ***calcamera***, no ficheiro de saída das coordenadas 3D mundo (x_w, y_w, z_w) e das coordenadas na memória *frame* (X_f, Y_f), para todos os pontos de calibração.
- ✓ Fecho do ficheiro de entrada.
- ✓ Fecho do ficheiro de saída.
- ✓ Comunicação da existência de erros em argumentos de entrada ou ao longo do decurso da execução:
  - Através de valores de retorno:
    - 1, ("Can't open file <inputfile>."), o módulo não consegue abrir o ficheiro de entrada especificado.
    - 2, ("Can't open file <outputfile>."), o módulo não consegue abrir o ficheiro de saída especificado.
  - Através de mensagens:
    - 1, (*InitMessage*), quando o número de argumentos é igual a um, isto é, apenas o nome da implementação.
    - 2, ("Bad number of arguments."), quando o número de argumentos é diferente de três.

Este módulo não apresenta qualquer tipo de restrições quanto à sua utilização.

5.2 - Fluxograma da implementação

Apresenta-se, de seguida, o fluxograma da implementação ***input***.





5.3 - Listagem da implementação

Apresenta-se, de seguida, a listagem da implementação ***input*** desenvolvida em linguagem *C* em ambiente *UNIX* numa estação de trabalho *SunSparc* e que se destina ao ambiente de processamento de imagem *XITE*[10], do qual utiliza algumas funções e tipos de variáveis já previamente definidas. Esta implementação contém comentários para auxílio à sua compreensão, assim como informação sobre os diferentes módulos, para sua posterior extracção através da ferramenta *CDOC(1)* existente no *XITE*.

```
/*
_____________________________________________________________________

  input.c
  @(#)input.c   1.00 93/12/18, Copyright 1993, DEEC, FEUP
  Departamento de Engenharia Electrotecnica e de Computadores
  Faculdade de Engenharia
  Rua dos Bragas
  4099 PORTO CODEX
  PORTUGAL
  E-mail: gpai@obelix.fe.up.pt
_____________________________________________________________________

  message.h
  @(#)message.h   1.3 92/01/15, Copyright 1990, Blab, UiO
  Image processing lab, Department of Informatics
  University of Oslo
  E-mail: blab@ifi.uio.no
_____________________________________________________________________

  Permission to use, copy, modify and distribute this software and its documentation for any purpose
and without fee is hereby granted, provided that this copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in supporting documentation and that the
name of B-lab, Department of Informatics or  University of Oslo not be used in advertising or
publicity pertaining to distribution of the software without specific, written prior permission.

  B-LAB DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT
SHALL B-LAB BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL
DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA
OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER
TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
*/

/************************************************************************/

static char *SccsId = "@(#)input.c 1.00 93/12/18, DEEC, FEUP";

/************************************************************************/

/* INCLUDES */

#include <stdio.h>
#include <blab/message.h>

/************************************************************************/

/*P:input*
_____________________________________________________________________

  input
_____________________________________________________________________

Name:          input - join the world 3D and the frame buffer coordinates for all
                       the calibration points

Syntax:        | input <inputfile> <outputfile>

Description:   'input' read in input file 'inputfile' the frame buffer coordinates, ask
               to the user the world 3D coordinate and write in the output file
               'outputfile' the world 3D and the frame buffer coordinates.

Return value:  | 1 => Can't open file <inputfile>.
               | 2 => Can't open file <outputfile>.

Examples:      | Try yourself.

Restrictions:  None.

Author:        Joao Tavares

Id:            @(#)input.c   1.0   93/12/18
_____________________________________________________________________
*/
```

[10] *X-based Image processing Tools an Environmemt*, [Lønnestad, 1992].

```
#ifdef MAIN

main(argc, argv)
int argc;
char **argv;

{

  int n;
  float xw, yw, zw, xf, yf;
  char c[2];
  FILE *infpt, *outfpt;

  InitMessage(&argc, argv, "Usage: %s\n <inputfile> <outputfile>\n");
  Iset_message(1);
  Iset_abort(1);

  if (argc == 1) Usage(1, (char*)0);
  if (argc != 3) Usage(2, "\nBad number of arguments.\n");

  /* open input  and output files */

  infpt = fopen(argv[1], "r");
  if (infpt == NULL) exit(Error(1, "\nCan't open file %s.\n", argv[1]));
  outfpt = fopen(argv[2], "w");
  if (outfpt == NULL) exit(Error(2, "\nCan't open file %s.\n", argv[2]));

  /* read in input file the frame buffer 2D coordinates, input to the user the world 3D coordinates,
and write in the output file */

  n = 0;
  fscanf(infpt,"%f %f", &xf, &yf);
  while(!feof(infpt)) {

    n += 1;
    printf("\nFor point(%d) with xf=%f and yf=%f:\n", n, xf, yf);
    do {

      printf("\n\txw=");
      scanf("%f", &xw);
      printf("\n\tyw=");
      scanf("%f", &yw);
      printf("\n\tzw=");
      scanf("%f", &zw);

      printf("\nThe point(%i) world 3D coordinates are correct?(y/Y) ", n);
      scanf("%s", c);

    } while (c[0] != 'Y' && c[0] != 'y');

    /* write in output file the coordinates of point */

    fprintf(outfpt, "%f %f %f %f %f\n", xw, yw, zw, xf, yf);
    fscanf(infpt,"%f %f", &xf, &yf);

  }

  /* close input file and output file */

  fclose(infpt);
  fclose(outfpt);

  /* output the number of points */

  printf("\nFile %s have %i points.\n", argv[2], n);

}

#endif

/*****************************************************************************/
```

Capítulo III

Seguimento de Linhas e Consequente Aproximação Poligonal Utilizando Faixas Dinâmicas

Introdução

Neste capítulo é apresentada uma implementação para seguimento de linhas e consequente aproximação poligonal utilizando faixas dinâmicas, segundo o método descrito em [Tavares, 1995].

Algumas características desta implementação são:

- ✓ O seguimento de linhas é baseado nos valores da amplitude e da direcção das orlas de intensidade obtidas por um detector como, por exemplo, o detector de Deriche[1].
- ✓ A utilização no seguimento de dois níveis de limiar para a amplitude (histerese).
- ✓ A possibilidade de aproximação de linhas abertas através da prévia detecção de um dos seus extremos.
- ✓ A possibilidade de aproximação de linhas fechadas, fazendo a ligação de todos os seus segmentos de recta de aproximação; isto é, após o seguimento e aproximação poligonal, as linhas continuam fechadas.
- ✓ A possibilidade de fecho de linhas abertas, se tal for pretendido pelo utilizador.
- ✓ A possibilidade de seguimento e aproximação poligonal de todas as linhas existentes na mesma imagem de entrada.
- ✓ A possibilidade de seguimento e aproximação poligonal de linhas de espessura não unitária.
- ✓ A implementação de um módulo responsável pelo desenho dos segmentos de recta resultantes da aproximação poligonal na imagem de saída.
- ✓ A possibilidade de saída dos resultados obtidos (as coordenadas dos pontos inicial e final que definem cada segmento de recta resultante) num ficheiro, para futura utilização.

1 - Descrição dos diferentes módulos

A implementação é constituída pelos módulos: ***principal***, ***strip***, ***find***, ***finddir***, ***closing*** e ***dda***, *Fig. 1*. Apresenta-se de seguida a descrição de cada um destes módulos.

1.1 - Módulo *principal*

Este módulo é responsável pela interface implementação/utilizador. A seu cargo está:

- ✓ Leitura dos argumentos opcionais ("*switches*"):
  - *title*, ("[-t <title>]"), título da imagem de saída. Por defeito, é "*Dynamic Strip*".
  - *filename*, ("[-f <filename>]"), nome do ficheiro para escrita dos resultados obtidos (coordenadas dos pontos que definem cada segmento de linha resultante). Por defeito é "*NULL*", o que implica a não abertura do ficheiro. Quando especificado, o ficheiro, se ainda não existir, é criado; caso contrário, os resultados são acrescentados ao ficheiro já existente.
  - *tsure*, ("[-ts <tsure>]"), valor mínimo de amplitude para que um determinado *pixel* seja classificado como seguramente pertencente a uma dada linha. Por defeito, o seu valor é *100*.
  - *tprobable*, ("[-tp <tprobable>]"), valor mínimo de amplitude para que um determinado *pixel* seja classificado como provavelmente pertencente a uma dada linha. Por defeito, o seu valor é *50*.

[1] Ver, por exemplo [Tavares, 1995].

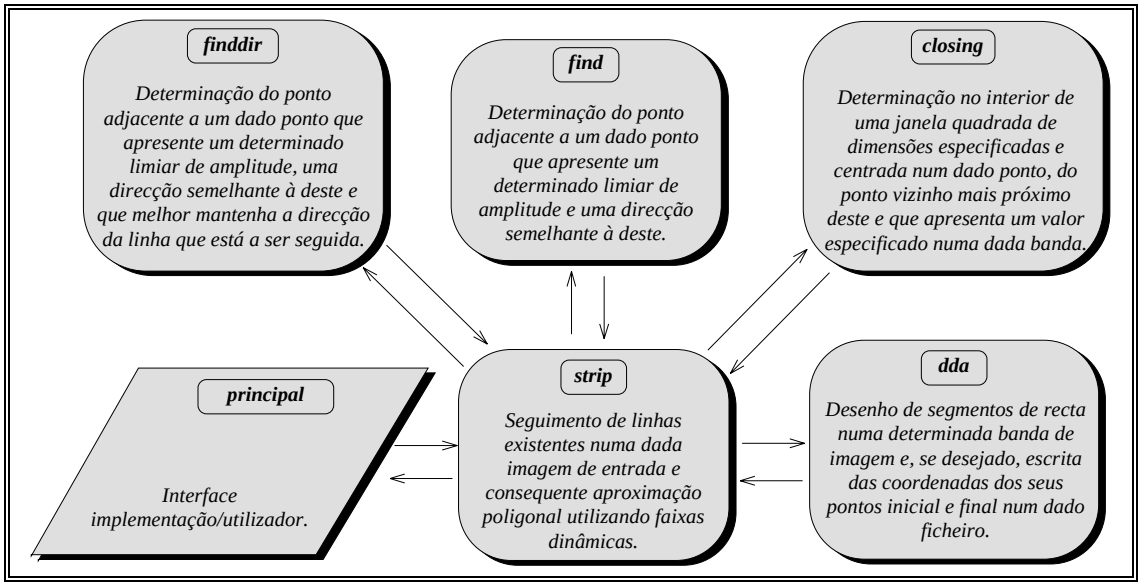


*Fig. 1- Módulos integrantes da implementação **dstrip** e suas relações.*

- *erdirection*, ("[-e <erdirection>]"), diferença máxima de direcção (em grau) para um determinado *pixel* ser considerado como pertencente a uma dada linha. Por defeito, o seu valor é *5*.
- *dadm*, ("[-d <dadm>]"), metade da largura da faixa utilizada para a aproximação poligonal, isto é, a tolerância de erro. Por defeito, o seu valor é *1.0*.
- *wmclose*, ("[-w <wmclose>]"), largura da janela quadrada no interior da qual deverá ser detectado o vizinho de um determinado *pixel*, para fecho de uma linha que seja constituída por mais de (*wmclose + 2*) *pixels*. Por defeito, é igual a *3*.
- *1PixWid*, ("[-p <1PixWid >]"), quando especificado como igual a *1* a implementação considera que a imagem de entrada contém linhas de espessura unitária. Por defeito, a implementação considera que a imagem de entrada contém linhas de espessura não unitária.
- *linecolor*, ("[-l <linecolor>]"), valor que deverá ser atribuído aos *pixels* pertencentes aos segmentos de recta resultantes da aproximação, na imagem de saída. Por defeito, o seu valor é *0*, correspondente ao preto.
- *backcolor*, ("[-b <backcolor>]"), valor que deverão ter todos os *pixels* na imagem de saída que não pertençam aos segmentos de recta resultantes da aproximação, isto é, os *pixels* que constituem o fundo da imagem. Por defeito, o seu valor é *255*, correspondente ao branco.

✓ Leitura dos argumentos passados à implementação pela linha de comando:

- input, ("<input>"), nome da imagem de entrada.
- output, (" <output>"), nome da imagem de saída.

✓ Leitura da imagem de entrada.

✓ Criação da imagem de saída.

✓ Criação da banda da imagem de saída.

✓ Abertura, se pretendido pelo utilizador, do ficheiro para escrita dos resultados obtidos com descrição de como estes foram obtidos.

✓ Chamada do módulo ***strip***.

✓ Fecho do ficheiro de saída de resultados, se pretendido pelo utilizador.

- ✓ Escrita da imagem de saída em disco, com descrição de como foi obtida.
- ✓ Comunicação da existência de erros em argumentos de entrada ou ao longo do decurso da execução:
  - Através de valores de retorno:
    - 1, ("Wrong width of mask for closing line <wmclose>; it must be odd"), o valor especificado para *wmclose* não é ímpar.
    - 2, ("Wrong value for tprobable <tprobable>; it must be greater than or equal to 2 and smaller than or equal to 255."), o valor especificado para *tprobable* não está compreendido entre *2* e *255*.
    - 3, ("Wrong value for tsure <tsure>; it must be greater than or equal to tprobable <tprobable> and smaller than or equal to 255."), o valor especificado para *tsure* não está compreendido entre *tprobable* e *255*.
    - 4, ("Wrong value for linecolor <linecolor>; it must be greater than or equal to *0* and smaller than or equal to 255."), o valor especificado para *linecolor* não está compreendido entre *0* e *255*.
    - 5, ("Wrong value for backcolor <backcolor>; it must be greater than or equal to *0* and smaller than or equal to 255."), o valor especificado para *backcolor* não está compreendido entre *0* e *255*.
    - 6, ("Can't read image <input>."), o módulo não consegue ler a imagem de entrada especificada.
    - 7, ("The <input> image doesn't have two bands."), a imagem de entrada especificada não contém duas bandas.
    - 8, ("Bad input pixel type for the first band of <input>."), a primeira banda da imagem de entrada especificada não contém *pixels* do tipo "*UNS_BYTE*".
    - 9, ("Bad input pixel type for the second band of <input>."), a segunda banda da imagem de entrada especificada não contém *pixels* do tipo "*UNS_BYTE*".
    - 10, ("The two bands of input image <input> have different size."), as bandas da imagem de entrada especificada têm dimensões diferentes.
    - 11, ("Can't make ouput image <output>."), o módulo não consegue criar a imagem de saída especificada.
    - 12, ("Can't make band of output image <output>"), o módulo não consegue criar a banda da imagem de saída especificada.
    - 13, ("Can't open output file <filename>"), o módulo não consegue abrir o ficheiro especificado para escrita de resultados obtidos.
  - Através de mensagens:
    - 1, (*InitMessage*), quando o número de argumentos é igual a um, isto é, apenas o nome da implementação.
    - 2, ("Bad number of arguments."), quando o número de argumentos é diferente de três.

Como restrições na utilização deste módulo têm-se:

- ⌧ A imagem de entrada tem de ser do tipo "*UNS_BYTE*", com duas bandas de igual tamanho resultantes de um detector de orlas de intensidade. Este detector deverá classificar cada *pixel* na primeira banda quanto à amplitude e, na segunda banda, quanto à direcção entre *0* e 2π.
- ⌧ O valor para *wmclose* deverá ser ímpar.

- *linecolor* e *backcolor* deverão ter valores compreendidos entre *0* e *255*.
- *tprobable* deverá ter um valor compreendido entre *2* e *255*.
- *tsure* deverá ter um valor igual ou superior ao valor de *tprobable* e menor ou igual a *255*.

1.2 - Módulo *strip*

Este módulo é responsável pelo seguimento de linhas e consequente aproximação poligonal utilizando faixas dinâmicas, pelo método descrito em [Tavares, 1995].

Como parâmetros de entrada para este módulo têm-se:

- *bandin1*, do tipo "*IBAND*", primeira banda da imagem de entrada, que contém os valores de amplitude obtidos por um detector de orlas de intensidade.
- *bandin2*, do tipo "*IBAND*", segunda banda da imagem de entrada, que contém os valores de direcção, entre *0* e 2π, obtidos por um detector de orlas de intensidade.
- *bandout*, do tipo "*IBAND*", banda de saída, com o desenho dos segmentos de recta obtidos pela implementação.
- *lcolor*, do tipo inteiro, valor que deverá ser atribuído, na imagem de saída, aos *pixels* pertencentes aos segmentos de recta resultantes da aproximação poligonal.
- *bcolor*, do tipo inteiro, valor que deverão ter todos os *pixels*, na imagem de saída, que não pertençam aos segmentos de recta resultantes da aproximação poligonal, isto é, os *pixels* que constituem o fundo da imagem.
- *ts*, do tipo inteiro, valor mínimo de amplitude para que um determinado *pixel* seja classificado como seguramente pertencente a uma dada linha.
- *tp*, do tipo inteiro, valor mínimo de amplitude para que um determinado *pixel* seja classificado como provavelmente pertencente a uma dada linha.
- *ed*, do tipo inteiro, diferença máxima de direcção para um determinado *pixel* ser considerado como pertencente a uma dada linha.
- *dadm*, do tipo real, metade da largura da faixa de aproximação poligonal.
- *wm*, do tipo inteiro, largura da janela onde poderá ser localizado o primeiro *pixel* determinado na linha que está a ser aproximada, para realizar o seu fecho, quando esta é constituída por mais de (*wm + 2*) *pixels*.
- *mark*, do tipo inteiro; quando igual a *0*, o módulo executa o seguimento e aproximação poligonal considerando que a imagem de entrada contém linhas de espessura unitária.
- *fpt*, do tipo "*FILE*", apontador para o ficheiro de escrita dos resultados obtidos, isto é, as coordenadas dos pontos que definem cada segmento de recta resultante da aproximação poligonal.

No início da sua tarefa, este módulo atribui a cada *pixel* da banda de saída *bandout* o valor correspondente a *bcolor*, e a cada *pixel* da primeira banda de entrada *bandin1* que apresente um valor inferior a *tp* atribui o valor correspondente a *WHITE*[2]. De seguida, faz um varrimento da banda de entrada *bandin1* à procura de possíveis linhas abertas através da prévia identificação de um dos seus pontos extremos; este ponto só tem um vizinho com valor de amplitude igual ou superior ao valor correspondente a *ts* e direcção aproximadamente igual a este; para tal, utiliza o módulo ***find***. Sempre que uma linha aberta é detectada, o módulo procede ao seu seguimento e aproximação poligonal;

[2] Variável definida globalmente.

todos os *pixels* desta linha ficam, na banda de entrada *bandin1*, com valor igual ao valor correspondente a *WHITE*, excepto o primeiro *pixel* detectado que fica com o valor igual ao valor correspondente a *FIRST*[3]. No fim do varrimento de toda a banda de entrada *bandin1*, o módulo executa um novo varrimento à procura de possíveis linhas fechadas, através da identificação de um dos seus *pixels*; o valor de amplitude do *pixel* detectado deverá ser igual ou superior a *ts*; para tal, utiliza o módulo ***find***. Sempre que uma linha fechada é detectada, o módulo procede ao seu seguimento e aproximação poligonal; todos os *pixels* desta linha ficam, na banda de entrada *bandin1*, com o valor *WHITE* excepto o primeiro *pixel* detectado, que fica com o valor *FIRST*. No fim deste segundo varrimento, o módulo termina e retorna *0* se todo o processo decorreu normalmente.

Após a determinação do segundo *pixel* de uma dada linha e até ao fim da mesma, o módulo utiliza o módulo ***finddir*** para a determinação do próximo *pixel*; o *pixel* a determinar deverá apresentar um valor de amplitude superior a *ts*, direcção aproximadamente igual à do *pixel* anterior e manter o mais possível a direcção da respectiva linha. Sempre que a determinação anterior não é conseguida, o módulo tenta determinar um próximo *pixel* que apresente um valor de amplitude superior a *tp*, direcção aproximadamente igual ao *pixel* anterior e manter o mais possível a direcção da respectiva linha, utilizando novamente o módulo ***finddir***. Se, mais uma vez, a determinação não for conseguida, o módulo tenta conseguir o fecho da linha. Para tal, utiliza o módulo ***closing*** para tentar localizar dentro de uma janela de dimensões $wm \times wm$, centrada no *pixel* anterior, o primeiro *pixel* determinado na linha, através do seu valor que deverá ser igual a *FIRST*. Esta última operação só é realizada para linhas que sejam constituídas por mais de $(wm + 2)$ *pixels*; esta restrição impede que linhas de pequeno comprimento fiquem reduzidas a um *pixel*. De notar que *pixels* isolados na banda de entrada *bandin1* são ignorados; assim, este módulo só considera para aproximação linhas que sejam constituídas pelo menos por dois *pixels*, que tenham direcções semelhantes e amplitudes superiores ao valor de *ts*. Quando *mark* é igual a *1*, isto é, quando a imagem de entrada contém linhas de espessura não unitária, todos os candidatos a próximo *pixel* da linha que satisfaçam os critérios de seguimento passam a ter o valor correspondente a *WHITE*, na banda de entrada *bandin1*.

Para o desenho dos segmentos de recta na banda de saída, o módulo utiliza o módulo ***dda***.

Como restrições à utilização deste módulo têm-se:

- Os *pixels* nas bandas de entrada deverão ser do tipo "*UNS_BYTE*".
- O valor de *wm* deverá ser ímpar.
- O valor de *tp* deverá ser superior ou igual a *2* e inferior ou igual a *255*.
- O valor de *ts* deverá ser igual ou superior ao valor de *tp* e inferior ou igual a *255*.

1.3 - Módulo *find*

Este módulo é responsável pela determinação de um *pixel* vizinho de um dado *pixel* que apresente um valor de amplitude superior a um determinado limiar e direcção aproximadamente igual.

Como parâmetros de entrada deste módulo têm-se:

- *band1*, do tipo "*IBAND*", banda onde a determinação do *pixel* vizinho é efectuada em termos de amplitude.
- *band2*, do tipo "*IBAND*", banda onde a determinação do *pixel* vizinho é efectuada em termos de direcção.
- *t*, do tipo inteiro, valor mínimo de amplitude que um *pixel* vizinho pode apresentar para ser considerado como candidato a melhor *pixel* vizinho.
- *ed*, do tipo inteiro, diferença máxima de direcção admissível entre o *pixel* dado e um *pixel* vizinho para este ser considerado como candidato a melhor *pixel* vizinho.

[3] Variável definida globalmente.

- *xc*, *yc*, do tipo inteiro, coordenadas do *pixel* de que se pretende determinar o melhor *pixel* vizinho.
- *xn*, *yn*, do tipo inteiro, passados por endereço, coordenadas do melhor *pixel* vizinho determinado pelo módulo.
- *color*, do tipo inteiro, valor que deverá ser atribuído aos *pixels* considerados como candidatos a melhor *pixel* vizinho do dado *pixel* na banda *band1*, se *mark* for igual a *1*.
- *mark*, do tipo inteiro; se igual a *1*, os *pixels* considerados como candidatos a melhor *pixel* vizinho do dado *pixel* assumem o valor correspondente a *color* na banda *band1*.

Esta determinação decorre da seguinte maneira: o módulo determina todos os *pixels* adjacentes que satisfaçam as condições exigidas ao nível de amplitude e de direcção, guardando as coordenadas do *pixel* que apresenta a direcção mais semelhante à do *pixel* dado.

Este módulo retorna o número de *pixels* determinados como candidatos a melhor *pixel* vizinho, não apresentando qualquer tipo de restrições na sua utilização.

1.4 - Módulo *finddir*

Este módulo é responsável pela determinação de um *pixel* vizinho de um dado *pixel* que apresente um valor de amplitude superior a um determinado limiar, direcção aproximadamente igual, e que melhor mantenha a direcção da linha que está a ser seguida.

Como parâmetros de entrada deste módulo têm-se:

- *band1*, do tipo "*IBAND*", banda onde a determinação do *pixel* vizinho é efectuada em termos de amplitude.
- *band2*, do tipo "*IBAND*", banda onde a determinação do *pixel* vizinho é efectuada em termos de direcção.
- *t*, do tipo inteiro, valor mínimo de amplitude que um *pixel* vizinho pode apresentar para ser considerado como candidato a melhor *pixel* vizinho.
- *ed*, do tipo inteiro, diferença máxima de direcção admissível entre o *pixel* dado e um *pixel* vizinho para este ser considerado como candidato a melhor *pixel* vizinho.
- *xl*, *yl*, do tipo inteiro, coordenadas do *pixel* que teve como melhor *pixel* vizinho o *pixel* de coordenadas *xc*, *yc*.
- *xc*, *yc*, do tipo inteiro, coordenadas do *pixel* de que se pretende determinar o vizinho.
- *xn*, *yn*, do tipo inteiro, passados por endereço, coordenadas do melhor *pixel* vizinho determinado pelo módulo.
- *color*, do tipo inteiro, valor que deverá ser atribuído aos *pixels* considerados como candidatos a melhor *pixel* vizinho do dado *pixel* na banda *band1*, se *mark* for igual a *1*.
- *mark*, do tipo inteiro; se igual a *1*, os *pixels* considerados como candidatos a melhor *pixel* vizinho do dado *pixel* assumem o valor correspondente a *color* na banda *band1*.

Este módulo, no início da sua tarefa, calcula a direcção α definida pelo *pixel* dado e pelo que teve este como melhor *pixel* vizinho. Após este cálculo, para todos *pixels* vizinhos que satisfaçam as condições exigidas ao nível de amplitude e de direcção, o módulo calcula a direcção φ_i definida pelo *pixel* dado e pelo respectivo *pixel* vizinho que está a ser considerado. O módulo define como melhor *pixel* vizinho aquele que apresenta a menor diferença entre as direcções α e φ_i. Deste modo, o módulo determina como melhor *pixel* vizinho aquele que melhor mantém a direcção da linha que está a ser seguida.

Este módulo retorna o número de *pixels* determinados como candidatos a melhor *pixel* vizinho, não apresentando qualquer tipo de restrições na sua utilização.

1.5 - Módulo *closing*

Este módulo é responsável pela localização no interior de uma janela quadrada de um *pixel*, vizinho de um dado *pixel*, que deverá apresentar um determinado valor.

Como parâmetros de entrada deste módulo têm-se:

- *band*, do tipo "*IBAND*", banda onde a detecção é efectuada.
- *xc*, *yc*, do tipo inteiro, coordenadas do *pixel* do qual se pretende determinar o vizinho, isto é, o centro da janela.
- *xn*, *yn*, do tipo inteiro, passados ao módulo por endereço, coordenadas do *pixel* vizinho determinado.
- *wm*, do tipo inteiro, largura da janela quadrada no interior da qual se realiza a determinação.
- *color*, do tipo inteiro, valor que o *pixel* determinado deverá apresentar.

A determinação do *pixel* vizinho decorre da seguinte maneira: primeiro, o módulo tenta determinar numa janela 3×3; quando determina um *pixel* vizinho guarda as suas coordenadas e retorna; caso não determine nenhum *pixel*, procura na janela de dimensão seguinte (5×5,7×7, ...), até determinar algum *pixel* ou até a dimensão da máscara ser superior a *wm*.

Este módulo apresenta como única restrição quanto à sua utilização que o valor para *wm* deverá ser ímpar e retorna:

- 0, como indicação de que a determinação não foi conseguida.
- 1, como indicação de que a determinação foi conseguida.

1.6 - Módulo *dda*

Este módulo é responsável pelo desenho, na imagem de saída, dos segmentos de recta resultantes do seguimento e aproximação poligonal efectuada pelo módulo ***strip***.

Trata-se de uma implementação do algoritmo de Bresenham (também designado por versão meio ponto do segmento), que tem como grande vantagem a não utilização de multiplicações, de divisões e de funções de arredondamento. Pode-se consultar este algoritmo em [Foley, 1991] e, para uma sua implementação em linguagem C [Schildt, 1991].

Neste módulo, também é efectuada a escrita das coordenadas dos *pixels* que definem cada segmento de recta no ficheiro de resultados, caso seja pretendido pelo utilizador.

Como parâmetros de entrada para este módulo têm-se:

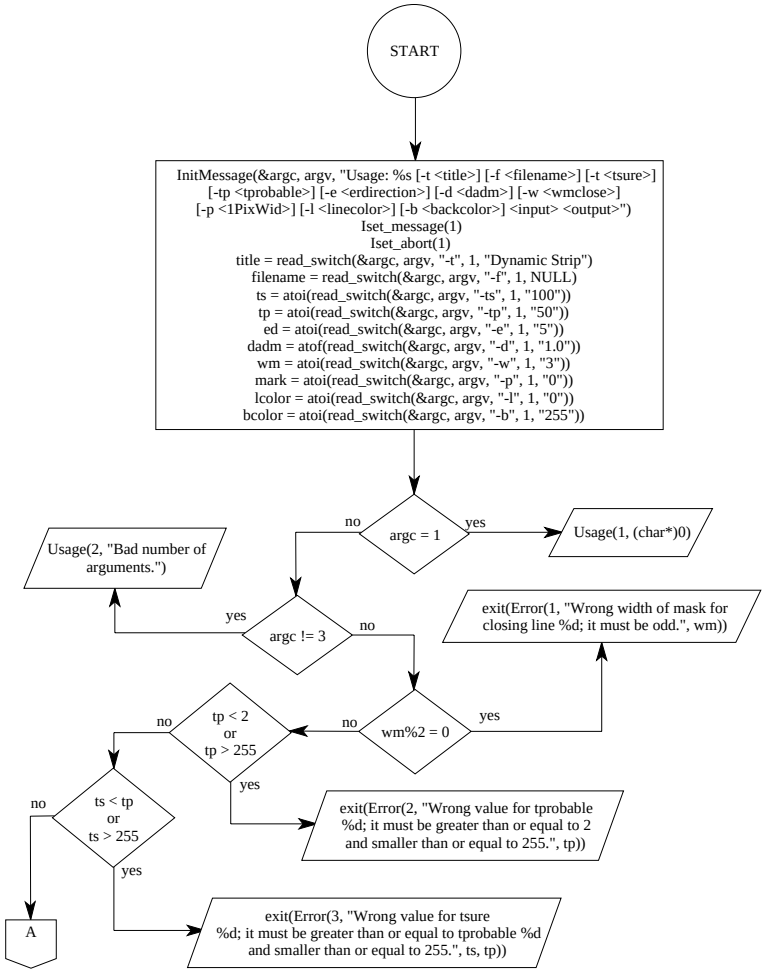
- *band*, do tipo "*IBAND*", banda da imagem de saída na qual é atribuído aos *pixels* pertencentes ao segmento de recta o valor *color*.
- *xi*, *yi*, do tipo inteiro, coordenadas do *pixel* inicial do segmento de recta.
- *xf*, *yf*, do tipo inteiro, coordenadas do *pixel* final do segmento de recta.
- color, do tipo inteiro, valor que é atribuído aos *pixels* pertencentes ao segmento de recta.
- *fpt*, do tipo "*FILE*", apontador para o ficheiro de escrita das coordenadas dos *pixels* inicial e final do segmento de recta.

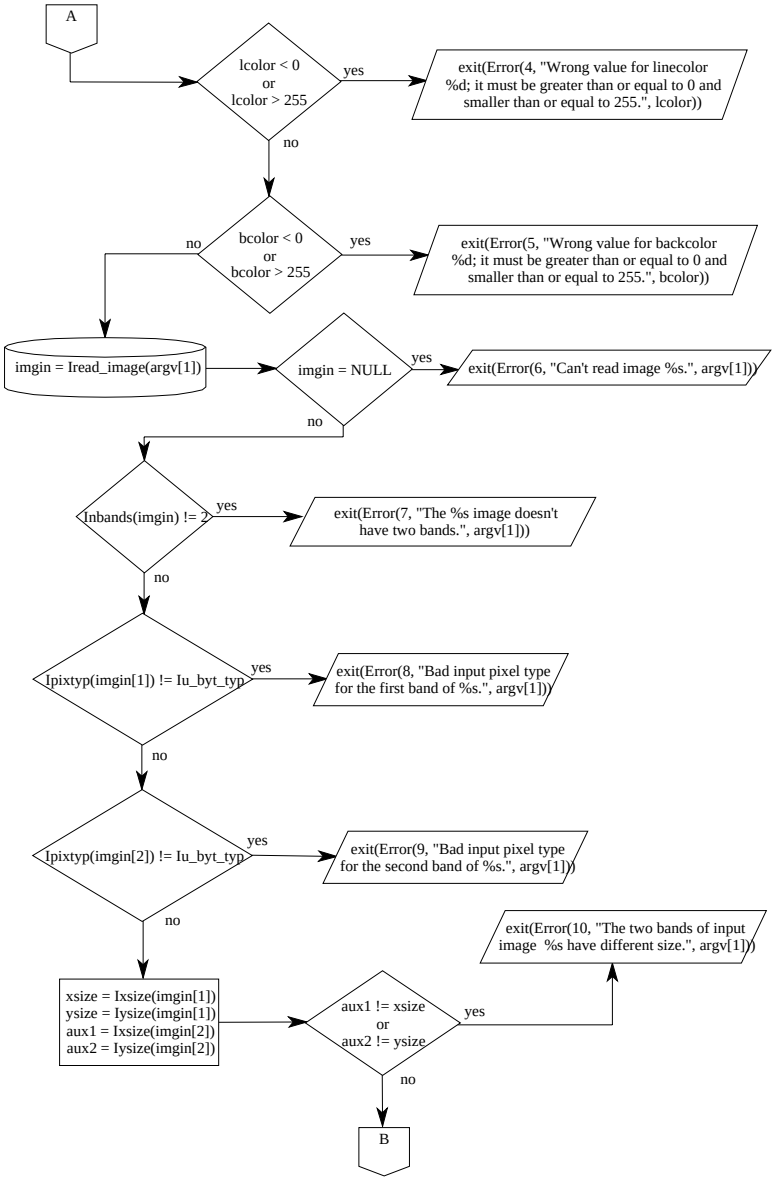
Este módulo retorna *0*, como indicação de que o processo decorreu com normalidade, e não tem qualquer tipo de restrições quanto à sua utilização.

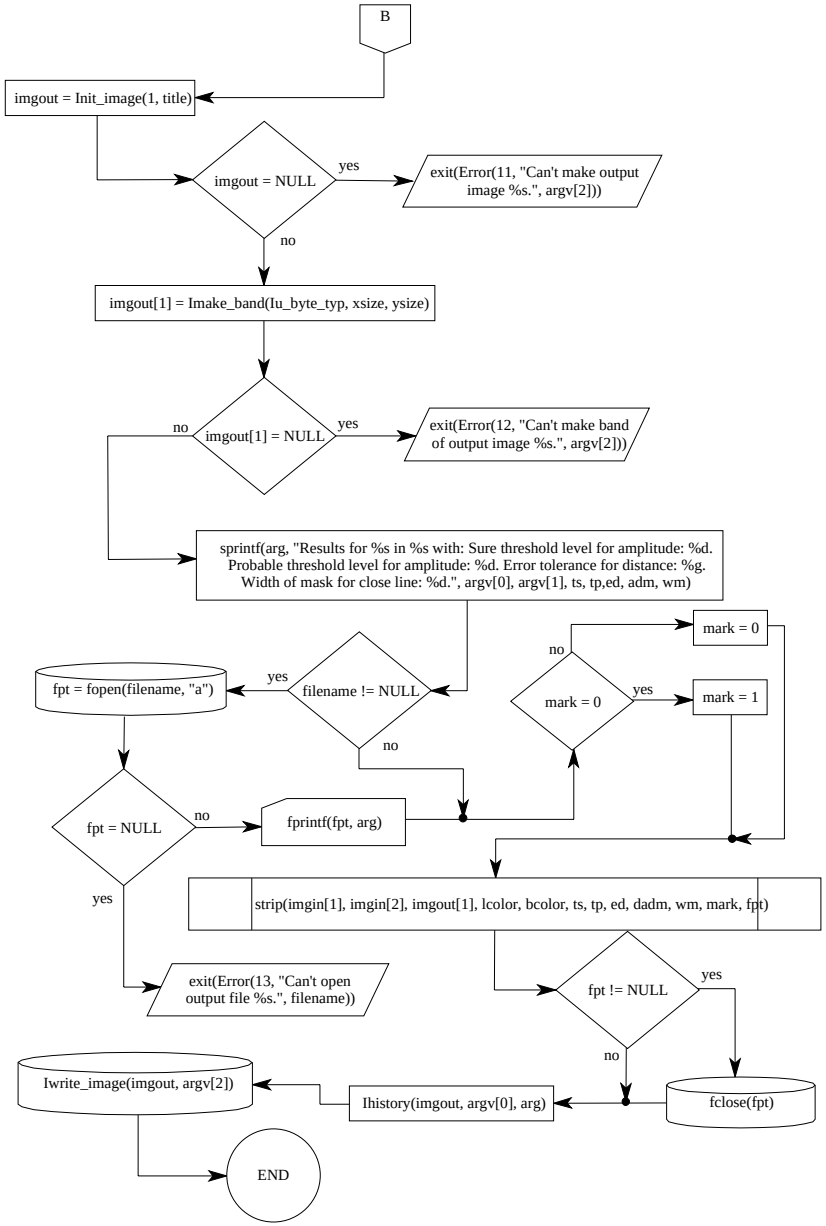
2 - Fluxogramas dos diferentes módulos

Apresentam-se, de seguida, os fluxogramas dos diferentes módulos que constituem a implementação ***dstrip***.

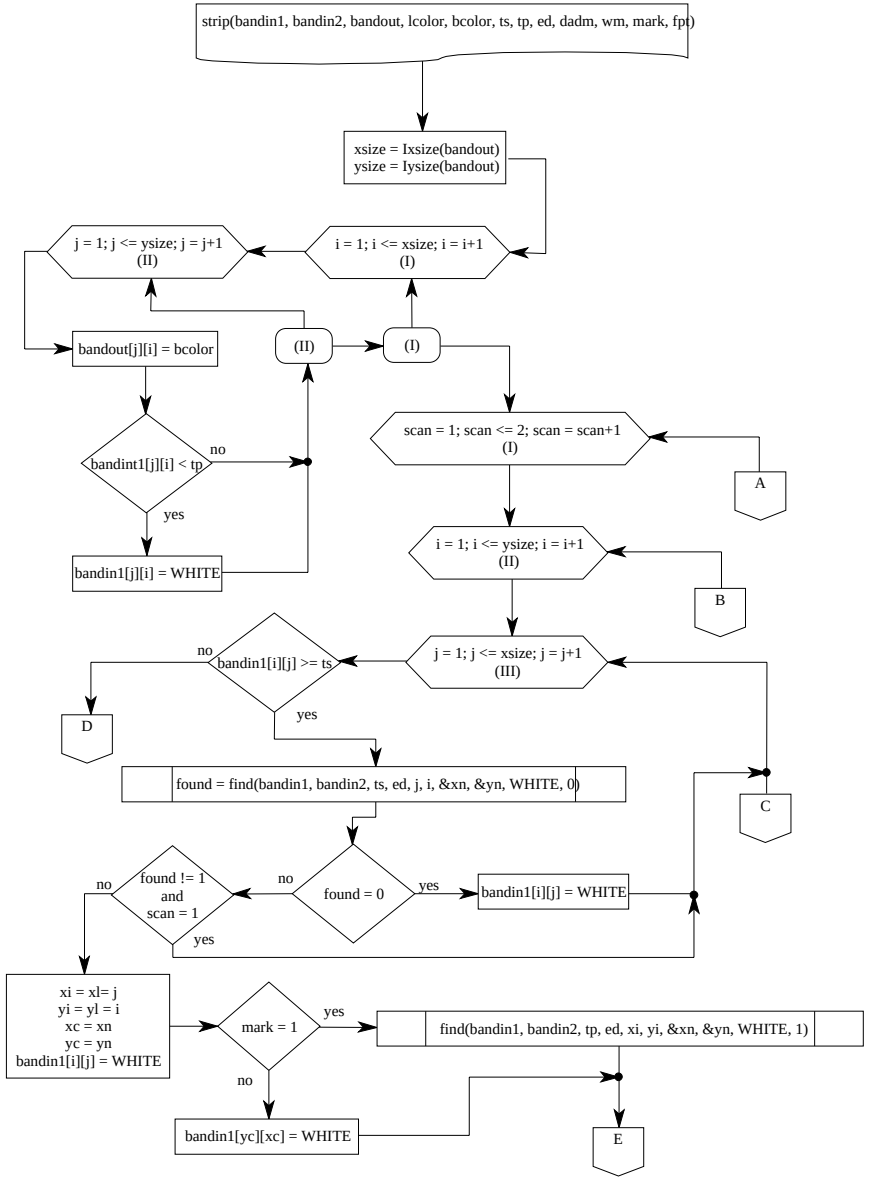
2.1 - Módulo *principal*

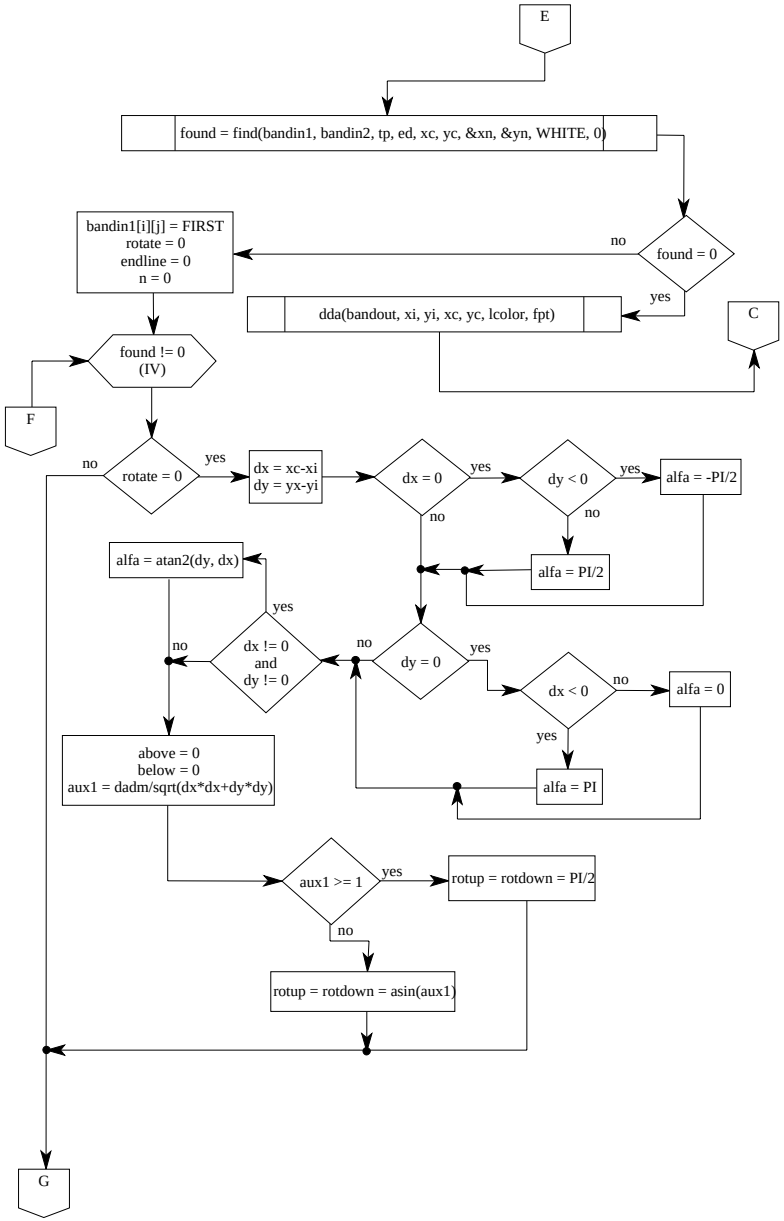


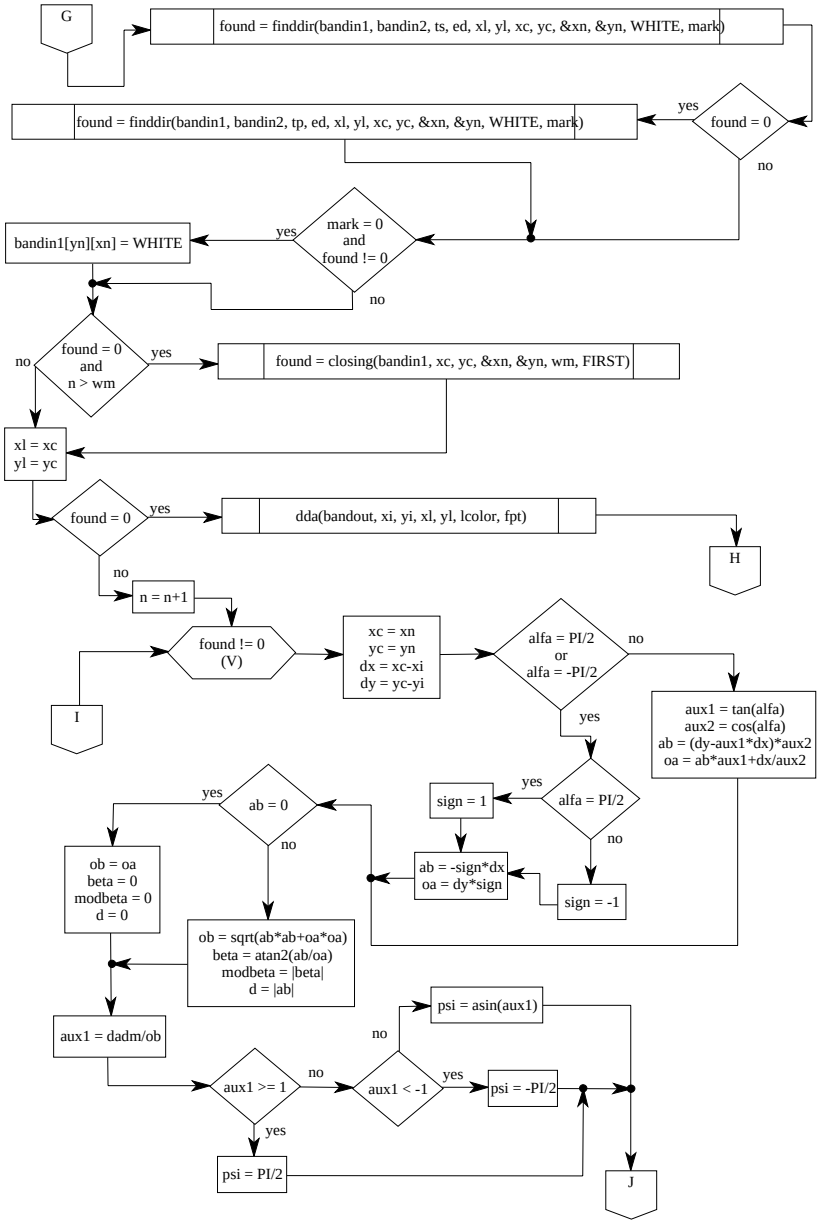


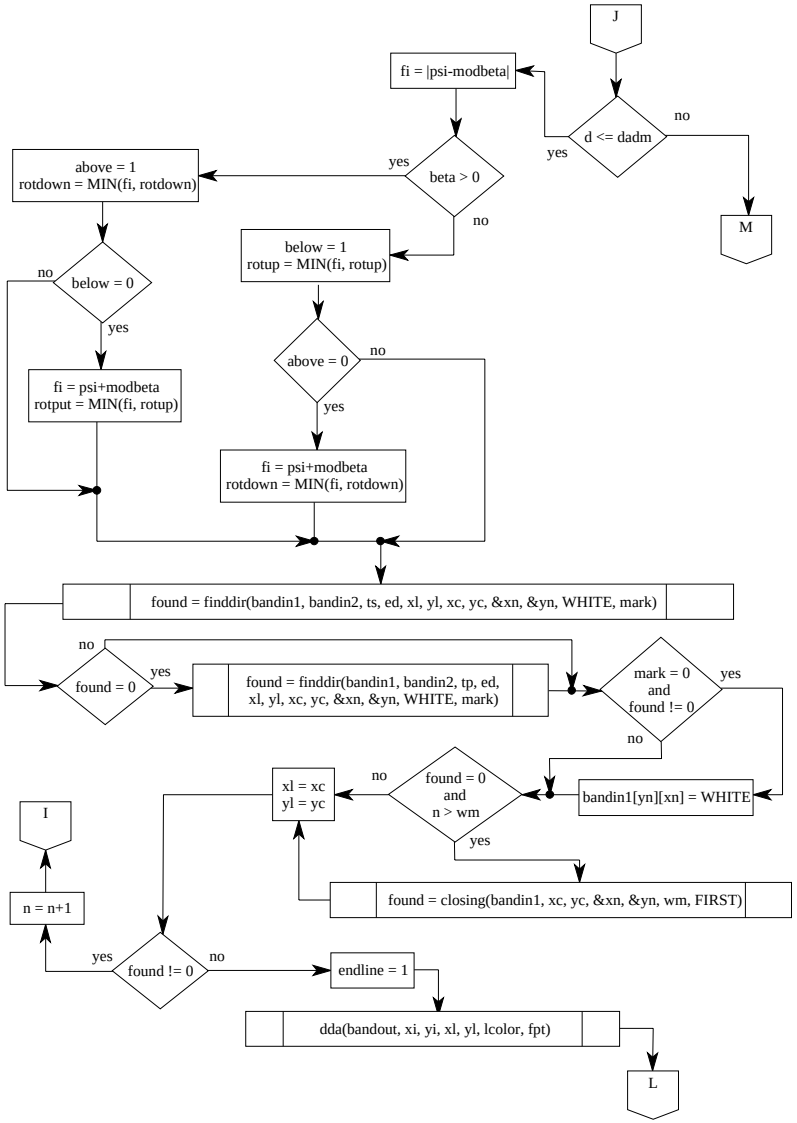


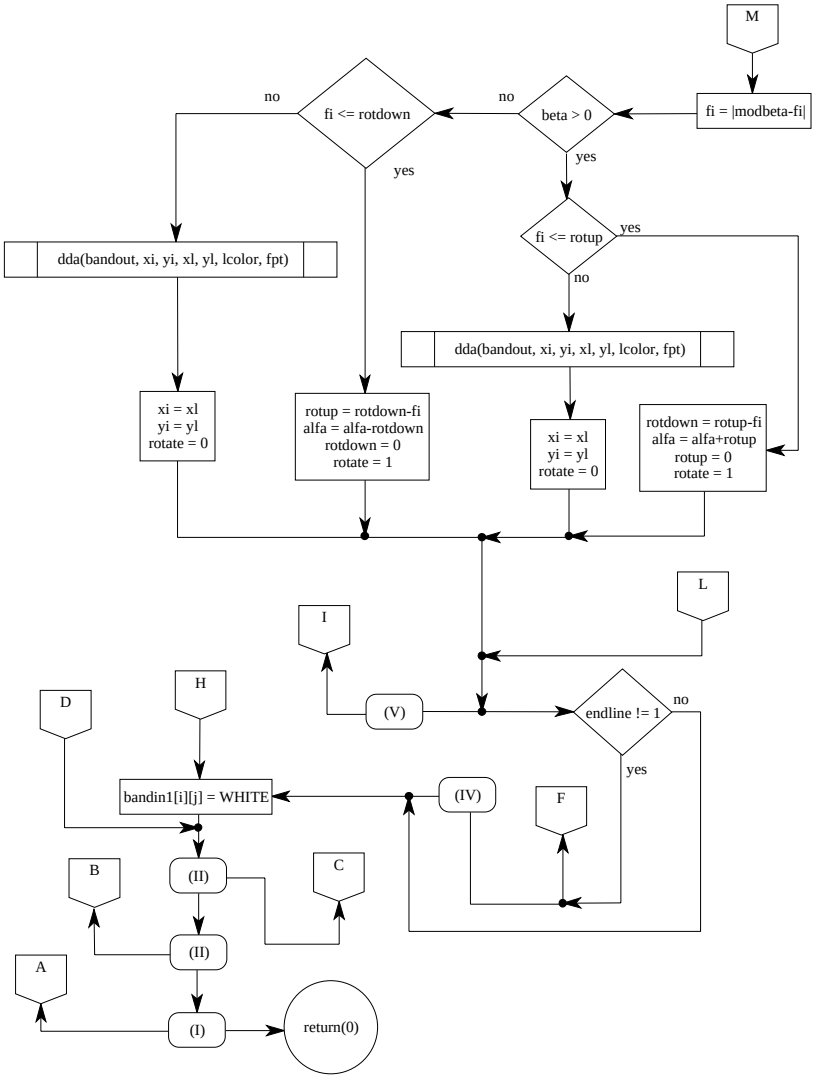
2.2 - Módulo *strip*



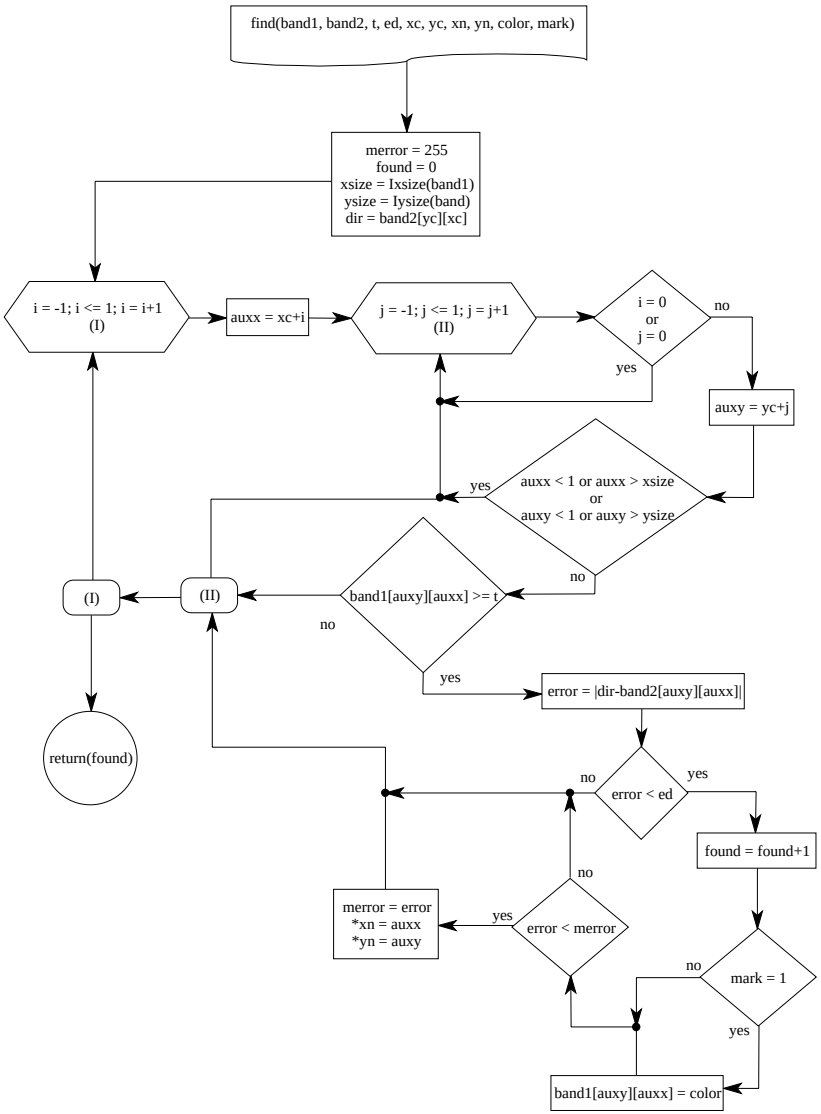

E
found = find(bandin1, bandin2, tp, ed, xc, yc, &xn, &yn, WHITE, 0)
found = 0
no
yes
bandin1[i][j] = FIRST
rotate = 0
endline = 0
n = 0
dda(bandout, xi, yi, xc, yc, lcolor, fpt)
C
found != 0
(IV)
F
rotate = 0
no
yes
dx = xc-xi
dy = yx-yi
dx = 0
dy < 0
alfa = -PI/2
alfa = PI/2
alfa = atan2(dy, dx)
dx != 0
and
dy != 0
dy = 0
dx < 0
alfa = 0
alfa = PI
above = 0
below = 0
aux1 = dadm/sqrt(dx*dx+dy*dy)
aux1 >= 1
rotup = rotdown = PI/2
rotup = rotdown = asin(aux1)
G


G
found = finddir(bandin1, bandin2, ts, ed, xl, yl, xc, yc, &xn, &yn, WHITE, mark)
found = finddir(bandin1, bandin2, tp, ed, xl, yl, xc, yc, &xn, &yn, WHITE, mark)
found = 0
yes
no
bandin1[yn][xn] = WHITE
mark = 0
and
found != 0
found = 0
and
n > wm
found = closing(bandin1, xc, yc, &xn, &yn, wm, FIRST)
xl = xc
yl = yc
found = 0
dda(bandout, xi, yi, xl, yl, lcolor, fpt)
H
n = n+1
found != 0
(V)
I
xc = xn
yc = yn
dx = xc-xi
dy = yc-yi
alfa = PI/2
or
alfa = -PI/2
aux1 = tan(alfa)
aux2 = cos(alfa)
ab = (dy-aux1*dx)*aux2
oa = ab*aux1+dx/aux2
ab = 0
sign = 1
alfa = PI/2
ob = oa
beta = 0
modbeta = 0
d = 0
ab = -sign*dx
oa = dy*sign
sign = -1
ob = sqrt(ab*ab+oa*oa)
beta = atan2(ab/oa)
modbeta = |beta|
d = |ab|
aux1 = dadm/ob
psi = asin(aux1)
aux1 >= 1
aux1 < -1
psi = -PI/2
psi = PI/2
J

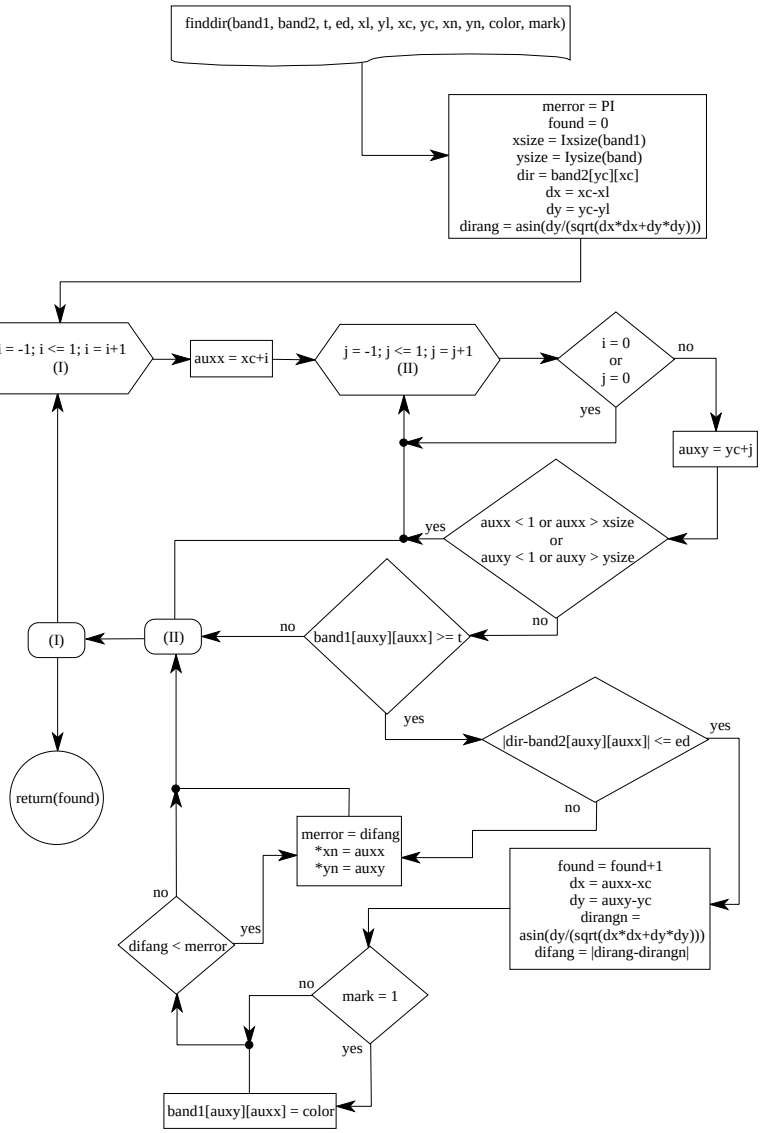

J
fi = |psi-modbeta|
d <= dadm
yes
no
M
beta > 0
above = 1
rotdown = MIN(fi, rotdown)
below = 1
rotup = MIN(fi, rotup)
below = 0
fi = psi+modbeta
rotput = MIN(fi, rotup)
above = 0
fi = psi+modbeta
rotdown = MIN(fi, rotdown)
found = finddir(bandin1, bandin2, ts, ed, xl, yl, xc, yc, &xn, &yn, WHITE, mark)
found = 0
found = finddir(bandin1, bandin2, tp, ed, xl, yl, xc, yc, &xn, &yn, WHITE, mark)
mark = 0 and found != 0
bandin1[yn][xn] = WHITE
found = 0 and n > wm
xl = xc
yl = yc
found = closing(bandin1, xc, yc, &xn, &yn, wm, FIRST)
I
n = n+1
found != 0
endline = 1
dda(bandout, xi, yi, xl, yl, lcolor, fpt)
L


M
fi = |modbeta-fi|
beta > 0
fi <= rotdown
fi <= rotup
dda(bandout, xi, yi, xl, yl, lcolor, fpt)
dda(bandout, xi, yi, xl, yl, lcolor, fpt)
xi = xl
yi = yl
rotate = 0
rotup = rotdown-fi
alfa = alfa-rotdown
rotdown = 0
rotate = 1
xi = xl
yi = yl
rotate = 0
rotdown = rotup-fi
alfa = alfa+rotup
rotup = 0
rotate = 1
L
I
(V)
endline != 1
H
D
bandin1[i][j] = WHITE
(IV)
F
B
(II)
C
(II)
A
(I)
return(0)

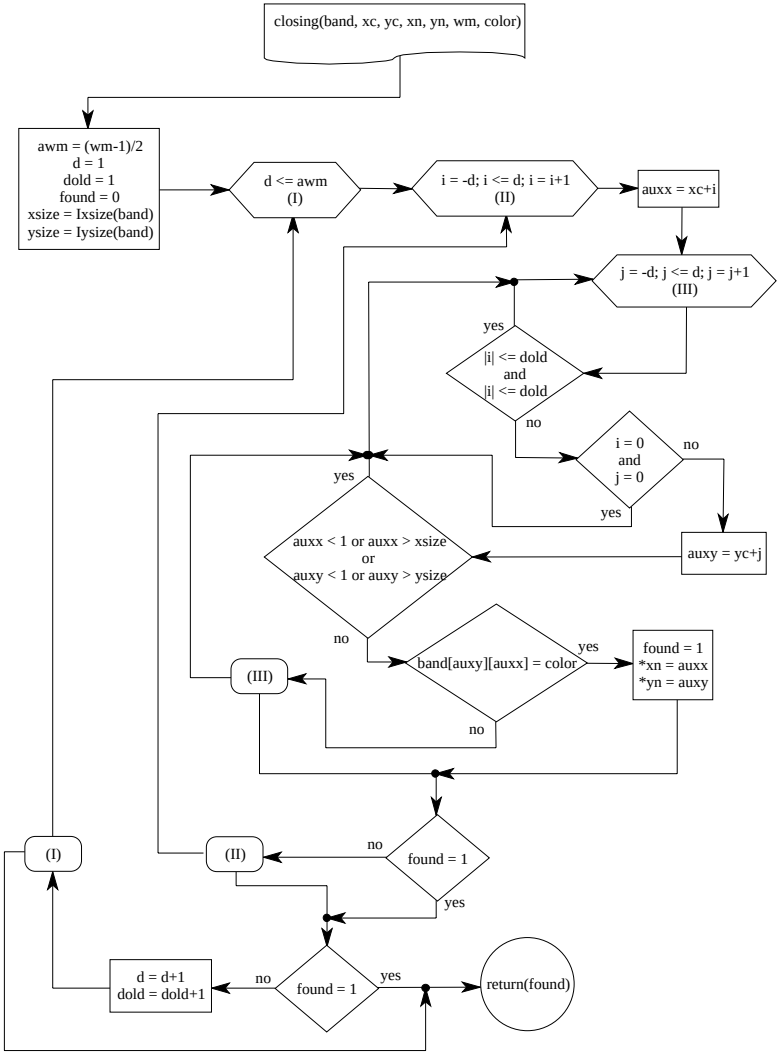
2.3 - Módulo *find*



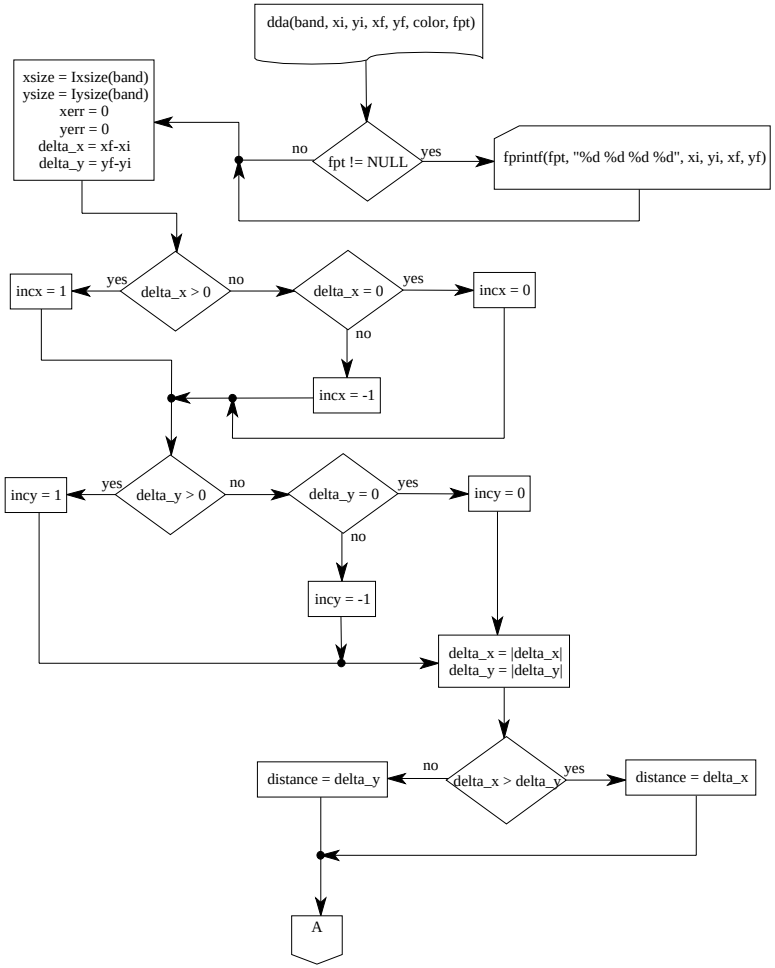
2.4 - Módulo *finddir*

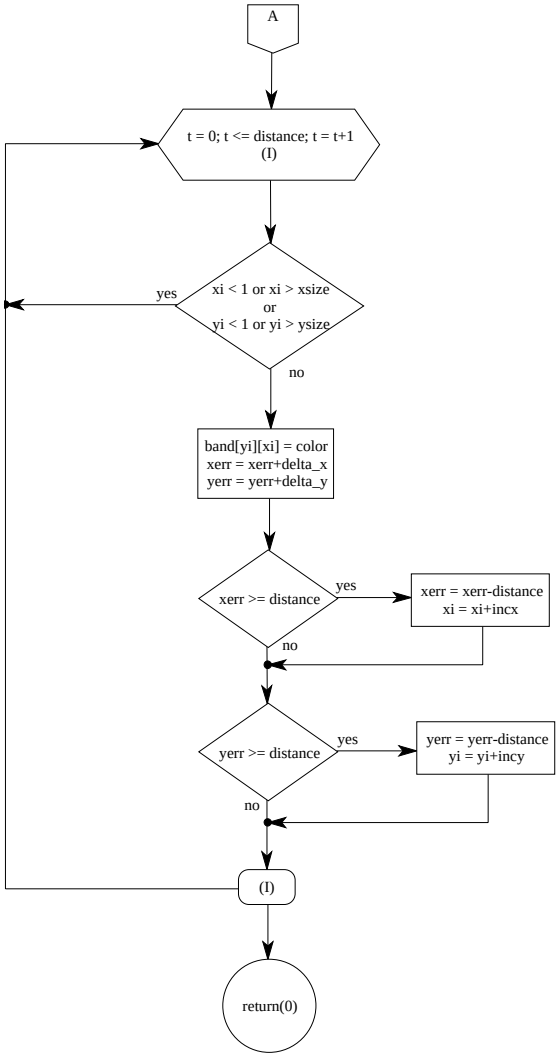


2.5 - Módulo *closing*



2.6 - Módulo *dda*





3 - Listagem da implementação

Apresenta-se, de seguida, a listagem da implementação ***dstrip*** desenvolvida em linguagem *C* em ambiente *UNIX* numa estação de trabalho *SunSparc* e que se destina ao ambiente de processamento de imagem *XITE*[4], do qual utiliza algumas funções e tipos de variáveis já previamente definidas. Esta implementação contém comentários para auxílio à sua compreensão, assim como informação sobre os diferentes módulos, para sua posterior extracção através da ferramenta *CDOC(1)* existente no *XITE*.

```
/*
_____________________________________________________________________

dstrip.c
@(#)dstrip.c   2.00 94/03/05, Copyright 1994, DEEC, FEUP
Departamento de Engenharia Electrotecnica e de Computadores
Faculdade de Engenharia
Rua dos Bragas
4099 PORTO CODEX
PORTUGAL
E-mail: gpai@obelix.fe.up.pt
_____________________________________________________________________

biff.h
@(#)biff.h   1.23 92/04/10, Copyright 1990, Blab, UiO
Image processing lab, Department of Informatics
University of Oslo
E-mail: blab@ifi.uio.no
_____________________________________________________________________

readarg.h
@(#)readarg.h   1.3 92/01/15, Copyright 1990, Blab, UiO
Image processing lab, Department of Informatics
University of Oslo
E-mail: blab@ifi.uio.no
_____________________________________________________________________

message.h
@(#)message.h   1.3 92/01/15, Copyright 1990, Blab, UiO
Image processing lab, Department of Informatics
University of Oslo
E-mail: blab@ifi.uio.no
_____________________________________________________________________
```

[4] *X-based Image processing Tools an Environmemt*, [Lønnestad, 1992].

```
/*****************************************************************************/

static char *SccsId="@(#)dstrip.c 2.00 94/03/05, DEEC, FEUP";

/*****************************************************************************/

/* INCLUDES */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <blab/biff.h>
#include <blab/readarg.h>
#include <blab/message.h>

/*****************************************************************************/

/* DEFINES */

#define WHITE 0                     /* white's value */
#define FIRST 1                     /* tag for the first point of a line */
#define MPI 3.141592654             /* pi's value */
#define AMPI 1.570796327            /* half pi's value */
#define MIN(a,b) (((a)<(b)) ? (a) : (b))   /* minimum of two numbers */

/*****************************************************************************/

/*F:strip*
```

strip

Name: strip - curve fitting using the Dynamic Strip modified algorithm

Syntax:
```
| int strip(bandin1, bandin2, bandout, lcolor, bcolor, ts, tp, ed,
| dadm, wm, mark, fpt)
| IBAND bandin1, bandin2, bandout;
| float dadm;
| int lcolor, bcolor, ts, tp, ed, wm, mark;
| FILE *fpt;
```

Description: 'strip' performs a polygonal approximation of the lines in 'bandin1' and 'bandin2' and write the output in 'bandout' and 'fpt' using dynamic strip's which can rotate around their first point's.
The band 'bandin1' have the amplitude and the band 'bandin2' have the directions of the output results of an edges detector, for example Deriche.
'bcolor' is the background colour and 'lcolor' is the colour's of output lines in 'bandout'.
'ts' is the minimum value for a pixel in 'bandin1' could be considered has sure belonging to a line.
'tp' is the minimum value for a pixel in 'bandin1' could be considered has probably belonging to a line.
'ed' is the maximum direction error tolerance for a pixel could be considered as belongs to a line.
'dadm' is half of the error tolerance for distance.
'wm' is the width of mask in which can be the first pixel found of a line with more than 'wm+2' pixels that will be considered for closing that line.
If 'mark' is zero then the function works like the input line's width is one pixel.

Return value: | 0 => Ok.

Example: | Try yourself.

Restrictions: Only UNS_BYTE pixels.
'tp' must be greater than or equal to 2, smaller than or equal to 255 and smaller than 'ts'.
'ts' must be greater than or equal to tp and smaller than or equal to 255.
'wm' must be odd.

Author: Joao Tavares

Id: @(#)strip.c 2.00 94/03/05

```
_____________________________________________________________________
*/

int strip(bandin1, bandin2, bandout, lcolor, bcolor, ts, tp, ed, dadm, wm, mark, fpt)
IBAND bandin1, bandin2, bandout;
float dadm;
int lcolor, bcolor, ts, tp, ed, wm, mark;
FILE *fpt;

{

 register int i, j, scan;
 int xsize, ysize, rotate, above, below, found, xn, yn, xc, yc, xl, yl, sign, xi, yi, dx, dy, endline, n;
 float beta, modbeta, fi, psi, ob, d, oa, ab, rotup, rotdown, aux1, aux2;
 double alfa;

 xsize = Ixsize(bandout);
 ysize = Iysize(bandout);

 /* make all pixel's in imgout as bcolor and make a previous amplitude threshold in bandin1 */

 for (i = 1; i <= xsize; ++i) {

  for (j = 1; j <= ysize; ++j) {

   bandout[j][i] = bcolor;
   if (bandin1[j][i] < tp) bandin1[j][i] = WHITE;

  }

 }

 /* finding lines in imgin */

 for (scan = 1; scan <= 2; ++scan) {

  for (i = 1; i <= ysize; ++i) {

   for (j = 1; j <= xsize; ++j) {

    if (bandin1[i][j] >= ts) {

     /* find the second pixel of this line and how many neighbours this pixel has */

     found = find(bandin1, bandin2, ts, ed, j, i, &xn, &yn, WHITE, 0);

     if (found == 0) {

      /* this pixel is only noise */

      bandin1[i][j] = WHITE;
      continue;

     }
     if (found != 1 && scan == 1) continue;

     xi = xl = j;
     yi = yl = i;
     xc = xn;
     yc = yn;

    /* make the first pixel found as WHITE */

    bandin1[i][j] = WHITE;

     /* mark the meighbours of the first pixel */

     if (mark == 1) find(bandin1, bandin2, tp, ed, xi, yi, &xn, &yn, WHITE, 1);
     else bandin1[yc][xc] = WHITE;

     /* verification that this line have more than two pixels */

     found = find(bandin1, bandin2, tp, ed, xc, yc, &xn, &yn, WHITE, 0);
     if (found == 0) {

      /* this line only has 2 pixels, call dda with the initial and final point of this segment */

      dda(bandout, xi, yi, xc, yc, lcolor, fpt);

continue;

     }

     /* mark the first pixel as FIRST */

     bandin1[i][j] = FIRST;

     rotate = 0;
     endline = 0;
     n = 0;
```

```
        while (found != 0) {

          if (rotate == 0) {

            /* define alfa for this strip */

            dx = xc-xi;
            dy = yc-yi;

            if (dx == 0) {

              if (dy < 0) alfa = -(AMPI);
              else alfa = AMPI;

            }
            if (dy == 0) {

              if (dx < 0) alfa = MPI;
              else alfa = 0.0;

            }
            if (dx != 0 && dy != 0) alfa = atan2((float)(dy),(float)(dx));

            /* define some variables for this strip */

            above = 0;
            below = 0;
            aux1 = dadm/(sqrt((float)(dx*dx)+(float)(dy*dy)));
            if (aux1 >= 1.0) rotup = rotdown = AMPI;
            else rotup = rotdown = asin(aux1);

          }

          /* finding the next pixel of the line in this strip */

          found = finddir(bandin1, bandin2, ts, ed, xl, yl, xc, yc, &xn, &yn, WHITE, mark);
          if (found == 0) found = finddir(bandin1, bandin2, tp, ed, xl, yl, xc, yc, &xn, &yn, WHITE,
mark);
          if (mark == 0 && found != 0) bandin1[yn][xn]=WHITE;
          if (found == 0 && n > wm) found = closing(bandin1, xc, yc, &xn, &yn, wm, FIRST);

          xl = xc;
          yl = yc;
          if (found == 0) {

            /* call dda with the initial and final pixel of the segment */

            dda(bandout, xi, yi, xl, yl, lcolor, fpt);
            break;

          }

          /* if found the next pixel */

          n += 1;
          while (found != 0) {

            xc = xn;
            yc = yn;

            /* define ab, oa, ob, beta, modbeta, d */

            dx = xc-xi;
            dy = yc-yi;
            if (alfa == AMPI || alfa == -(AMPI)) {

              if (alfa == AMPI) sign = 1;
              else sign = (-1);
              ab = (float)(-sign*dx);
              oa = (float)(dy*sign);

            }
            else {

              aux1 = tan(alfa);
              aux2 = cos(alfa);
              ab = ((float)(dy)-aux1*((float)(dx)))*aux2;
              oa = ab*aux1+((float)(dx))/aux2;

            }
            if (ab == 0.0) {

              ob = oa;
              beta = 0.0;
              modbeta = 0.0;
              d = 0.0;

            }
            else {
```

```
  ob = sqrt(ab*ab+oa*oa);
  beta = atan2(ab, oa);
  modbeta = fabs(beta);
  d = fabs(ab);

}
aux1 = dadm/ob;
if (aux1 >= 1.0) psi = AMPI;
else {

  if (aux1 <= -1) psi = -AMPI;
  else psi = asin(aux1);

}

/* verifying the current pixel */

if (d <= dadm) {

  fi = fabs(psi-modbeta);
  if (beta > 0.0) {

    /* the current pixel is above the critical line */

    above = 1;
    rotdown = MIN(fi, rotdown);
    if (below == 0) {

      /* there isn't yet any pixel below the critical line */

      fi = psi+modbeta;
      rotup = MIN(fi, rotup);

    }

  }
  else {

    /* the current pixel is below the critical line */

    below = 1;
    rotup = MIN(fi, rotup);
    if (above == 0) {

      /* there isn't yet any pixel above the critical line */

      fi = psi+modbeta;
      rotdown = MIN(fi, rotdown);

    }

  }

  /* finding another pixel of this line */

  found = finddir(bandin1, bandin2, ts, ed, xl, yl, xc, yc, &xn, &yn, WHITE, mark);
  if (found == 0) found = finddir(bandin1, bandin2, tp, ed, xl, yl, xc, yc, &xn, &yn,
WHITE, mark);
  if (mark == 0 && found != 0) bandin1[yn][xn] = WHITE;
  if (found == 0 && n > wm) found = closing(bandin1, xc, yc, &xn, &yn, wm, FIRST);

  xl = xc;
  yl = yc;
  if (found != 0) {

    n += 1;
    continue;

  }
  else {

    endline = 1;

    /* call dda with the initial and final point of this strip */

    dda(bandout, xi, yi, xl, yl, lcolor, fpt);

    break;

  }

}
else {

  fi = fabs(modbeta-psi);
  if (beta > 0.0) {

    /* the current pixel is above the critical line and out of this strip */

    if (fi <= rotup) {
```

```
            /* the current pixel can be in this strip if it rotate */

            rotdown = rotup-fi;
            alfa += rotup;
            rotup = 0.0;
            rotate = 1;

          }
          else {

            /* the current pixel can't be in this strip, call dda with the initial and final pixel of this
segment */

            dda(bandout, xi, yi, xl, yl, lcolor, fpt);
            xi = xl;
            yi = yl;
            rotate = 0;

          }
          break;

        }
        else {

          /* the current pixel is below the critical line and out of this strip */

          if (fi <= rotdown) {

            /* the current pixel can be in this strip if it rotate */

            rotup = rotdown-fi;
            alfa -= rotdown;
            rotdown = 0.0;
            rotate = 1;

          }
          else {

            /* the current pixel can't be in this strip, call dda with the initial and final pixel of this
segment */

            dda(bandout, xi, yi, xl, yl, lcolor, fpt);
            xi = xl;
            yi = yl;
            rotate = 0;

          }
          break;

        }

      }

      if (endline != 1) continue;
      else break;

    }

    /* make the first pixel found as WHITE */

    bandin1[i][j] = WHITE;

  }

 }

}

return(0);

}
```

```
/*****************************************************************************/

/*F:find*
_____________________________________________________________________________

 find
_____________________________________________________________________________

Name:        find - find the best neighbour of a pixel

Syntax:      | int find(band1, band2, t, ed, xc, yc, xn, yn, color, mark)
             | IBAND band1, band2;
             | int t, ed, xc, yc, *xn, *yn, color, mark;
```

```
Description:    'find' looks in 'band1' and 'band2' for the best neighbour of a pixel
                at 'xc', 'yc'.
                The band 'band1' have the amplitude and the band 'band2' have
                the directions of the output results of an edges detector, for
                example Deriche.
                't' is the minimum amplitude value for a pixel could be considered
                as a possible neighbour.
                'ed' is the error tolerance for direction for a pixel could be
                considered as a possible neighbour.
                'xn' and 'yn' are the coordinates of the found pixel.
                If 'mark' is one, all the possible good neighbours will have the value
                of 'color' in 'band1'.

Return value:   | The number of good neighbours.

Example:        | Try yourself.

Restrictions:   None.

Author:         Joao Tavares

Id:             @(#)find.c   1.00   94/03/05
_____________________________________________________________________
*/

int find(band1, band2, t, ed, xc, yc, xn, yn, color, mark)
IBAND band1, band2;
int t, ed, xc, yc, *xn, *yn, color, mark;

{

  register int i, j;
  int error, found, xsize, ysize, dir, merror, auxx, auxy;

  merror = 255;
  found = 0;
  xsize = Ixsize(band1);
  ysize = Iysize(band1);
  dir = band2[yc][xc];

  for ( i = -1; i <= 1; ++i) {

    auxx = xc+i;
    for (j = -1; j <= 1; ++j) {

      /* finding the best neighbour of this pixel */

      if (i == 0 && j == 0) continue;
      auxy = yc+j;
      if (auxx < 1 || auxx > xsize || auxy < 1 || auxy > ysize) continue;
      if (band1[auxy][auxx] >= t) {

        error = abs(dir-band2[auxy][auxx]);
        if (error <= ed) {

          found += 1;

          /* if is desire make the found pixel as color */

          if (mark == 1) band1[auxy][auxx] = color;
          if (error <= merror) {

            merror = error;

            /* save the coordinates of the found pixel */

            *xn = auxx;
            *yn = auxy;

          }

        }

      }

    }

  return(found);

}

/*****************************************************************************/

/*F:finddir*
_____________________________________________________________________

  finddir
_____________________________________________________________________
```

```
Name:         finddir - find the best neighbour of a pixel

Syntax:       | int finddir(band1, band2, t, ed, xl, yl, xc, yc, xn, yn, color, mark)
              | IBAND band1, band2;
              | int t, ed, xc, yc, *xn, *yn, color, mark;

Description:  'finddir' looks in 'band1' and 'band2' for the best neighbour of a
              pixel at 'xc', 'yc' which have it's last good neighbour at 'xl', 'yl'.
              The band 'band1' have the amplitude and the band 'band2' have
              the directions of the output results of an edges detector, for
              example Deriche.
              't' is the minimum amplitude value for a pixel could be considered
              as a possible neighbour.
              'ed' is the error tolerance for direction for a pixel could be
              considered as a possible neighbour.
              'xn' and 'yn' are the coordinates of the found pixel.
              If 'mark' is one, all the possible good neighbours will have the
              value of 'color' in 'band1'.

Return value: | The number of good neighbours.

Example:      | Try yourself.

Restrictions: None.

Author:       Joao Tavares

Id:           @(#)finddir.c   1.00   94/03/05
-------------------------------------------------------------------------------
*/

int finddir(band1, band2, t, ed, xl, yl, xc, yc, xn, yn, color, mark)
IBAND band1, band2;
int t, ed, xl, yl, xc, yc, *xn, *yn, color, mark;

{

  register int i, j;
  int found, xsize, ysize, auxx, auxy, dir;
  double error, dirang, dirangn, difang, merror, dx, dy;

  merror = MPI;
  found = 0;
  xsize = Ixsize(band1);
  ysize = Iysize(band1);
  dir = band2[yc][xc];

  /* compute the angle between the x axis and the line defined by the current pixel and the previous
one */

  dx = (float)(xc-xl);
  dy = (float)(yc-yl);
  dirang = asin(dy/(sqrt((float)(dx*dx)+(float)(dy*dy))));

  for (i = -1; i <= 1; ++i) {

    auxx = xc+i;
    for (j = -1; j <= 1; ++j) {

      /* finding the best neighbour of this pixel */

      if (i == 0 && j == 0) continue;
      auxy = yc+j;
      if (auxx < 1 || auxx > xsize || auxy < 1 || auxy > ysize) continue;
      if (band1[auxy][auxx] >= t) {

        if (abs(dir-band2[auxy][auxx]) <= ed) {

          found += 1;
          dx = (float)(auxx-xc);
          dy = (float)(auxy-yc);

          /* compute the angle between the x axis and the line defined by the current pixel and it's
possible neighbour */

          dirangn = asin(dy/(sqrt((float)(dx*dx)+(float)(dy*dy))));
          difang = fabs(dirang-dirangn);

          /* if desire mark the found pixel in band1 as color */

          if (mark == 1) band1[auxy][auxx] = color;
          if (difang <= merror) {

            merror = difang;

            /* save the coordinates of the found pixel */

            *xn = auxx;
            *yn = auxy;
```

```
          }
        }
      }
    }
  }
  return(found);
}
/*************************************************************************/
/*F:closing*
_________________________________________________________________

  closing
_________________________________________________________________

Name:        closing - find one pixel which is neighbour of a given pixel

Syntax:      | int closing(band, xc, yc, xn, yn, wm, color)
             | IBAND band;
             | int xc, yc, *xn, *yn, wm, color;

Description:  'closing' find the pixel '*xn, *yn' which is neighbour of the given
               pixel 'xc, yc' in 'band', has a value as 'color' and is the closest.
               'wm' is the width of the square mask in which the neighbour can be.

Return value:  | 0 => If not found the neighbour.
               | 1 => If found the neighbour.

Example:       | Try yourself.

Restrictions:  Width of mask 'wm' must be odd.

Author:        Joao Tavares

Id:            @(#)closing.c   1.00   94/03/05
_________________________________________________________________
*/
int closing(band, xc, yc, xn, yn, wm, color)
IBAND band;
int xc, yc, *xn, *yn, wm, color;

{

 register int i, j;
 int  auxx, auxy, awm, d, dold, xsize, ysize, found;

 awm = (wm-1)/2;
 d = 1;
 dold = 0;
 found = 0;
 xsize = Ixsize(band);
 ysize = Iysize(band);

 while (d <= awm) {

  /* finding the pixel with color which are d distance from the current pixel */

  for (i = -(d); i <= d; ++i) {

   auxx = xc+i;
   for (j = -(d); j <= d ; ++j) {

     if (abs(i) <= dold && abs(j) <= dold) continue;
     if (i == 0 && j == 0) continue;
     auxy = yc+j;
     if (auxx < 1 || auxx > xsize || auxy < 1 || auxy > ysize ) continue;
     if (band[auxy][auxx] == color) {

       found = 1;

       /* save the coordinates of the found pixel */

       *xn = auxx;
       *yn = auxy;
       break;

     }

   }
   if (found == 1) break;
```

```
   }
   if (found == 1) break;
   ++d;
   ++dold;

  }

  return(found);

}

/*************************************************************************/

/*F:dda*
_____________________________________________________________________

  dda
_____________________________________________________________________

Name:          dda - drawing a segment of line using the algorithm of Bresenham

Syntax:        | dda(xi, yi, xf, yf, band, color, fpt)
               | IBAND band;
               | int xi, yi, xf, yf, color;
               | FILE *fpt;

Description:   'dda' draw the segment of line between the initial pixel 'xi, yi' and
               the final pixel 'xf, yf' in the band 'band' with color as 'color'.
               If 'fpt' is different of NULL the initial and final points of the
               segment will be writing in file 'fpt'.

Return value:  | 0 => Ok.

Example:       | Try yourself.

Restrictions:  None.

Author:        Joao Tavares

Id:            @(#)dda.c   1.00   93/05/05
_____________________________________________________________________
*/

int dda(band, xi, yi, xf, yf, color, fpt)
int xi, yi, xf, yf, color;
FILE *fpt;
IBAND band;

{

 register int t, distance;
 int xsize, ysize, xerr, yerr, delta_x, delta_y, incx, incy;

 /* write in file, if it was desire, the coordinates of the initial and final pixel of the segment */

 if (fpt != NULL) fprintf(fpt, "\t%d %d\t%d %d\n", xi, yi, xf, yf);

 xsize = Ixsize(band);
 ysize = Iysize(band);
 xerr = 0;
 yerr = 0;

 /* calculation of delta x and delta y */

 delta_x = xf-xi;
 delta_y = yf-yi;

 /* determination of increment x and increment y */

 if (delta_x > 0) incx = 1;
 else if (delta_x == 0) incx = 0;
 else incx = -1;
 if (delta_y > 0) incy = 1;
 else if (delta_y == 0) incy = 0;
 else incy = -1;

 /* determination of the distance */

 delta_x = abs(delta_x);
 delta_y = abs(delta_y);

 if (delta_x > delta_y) distance = delta_x;
 else distance = delta_y;

 /* mark the closest point to the line with color */

 for (t = 0; t <= distance; t++) {

  if (xi < 1 || xi > xsize || yi < 1 || yi > ysize) continue;
  band[yi][xi] = color;
```

```
   xerr += delta_x;
   yerr += delta_y;
   if (xerr >= distance) {

     xerr -= distance;
     xi += incx;

   }
   if (yerr >= distance) {

     yerr -= distance;
     yi += incy;

   }

 }
 return(0);

}

/*****************************************************************************/

/*P:dstrip*
```

---

dstrip

Name: dstrip - curve fitting using the Dynamic Strip modified algorithm

Syntax: | dstrip [-t <title>] [-f <filename>] [-ts <tsure>] [-tp <tprobable>]
| [-e <erdirection>] [-d <dadm>] [-w <wmclose>] [-p <1PixWid>]
| [-l <linecolor>] [-b <backcolor>] <input> <output>

Description: 'dstrip' perform the polygonal approximation of the lines in image 'input' and write the output in image 'output' using the Dynamic Strip modified algorithm.
The 'input' image must have two bands, the first with the amplitude and the second with the directions of the output results of an edges detector, for example Deriche.
The 'output' image will have the 'title' which is indicate through flag '-t', by default the 'title' is "Dynamic strip".
The initial and final point's of each segment of line can be writing in the file 'filename' if the flag '-f' is on, by default this flag is off.
The minimum amplitude value for a pixel could be considered as sure belonging to a line can be indicated through flag '-ts'. By default 'tsure' is 100.
The minimum amplitude value for a pixel could be considered as probably belonging to a line can be indicated through flag '-tp'. By default 'tprobable' is 50.
The maximum direction error tolerance to a pixel could be considered as belonging to a line can be indicated trough flag '-ed'. By default 'erdirection' is 1.
The error tolerance for distance '-dadm' by default is 1.0 and can be indicated through flag 'd'.
The width of mask 'wmclose' in which the closing point of a line that have more than 'wmclose+2' pixels can be is by default 5 and can be indicated through flag 'w'.
If the width of the input lines is one pixel then '1PixWid' must be indicate through flag '-p' as 1. By default '1PixWid' is 0.
The background colour of 'output' image can be indicated trough flag '-b'. By default 'backcolor' is 255 (white).
The line’s color in 'output' image can be indicated trough flag '-l'. By default 'linecolor' is 0 (black).

Return value: | 1 => Wrong width of mask for closing line <wmclose>; it must be odd.
| 2 => Wrong value for tprobable <tprobable>; it must be greater than or equal to 2 and smaller than or equal to 255.
| 3 => Wrong value for tsure <tsure>; it must be greater than or equal to tprobable <tprobable> and smaller than or equal to 255.
| 4 => Wrong value for linecolor <linecolor>; it must be greater than or equal to 0 and smaller than or equal to 255.
| 5 => Wrong value for backcolor <backcolor>; it must be greater than or equal to 0 and smaller than or equal to 255.
| 6 => Can't read image <input>.
| 7 => The <input> image doesn't have two bands.
| 8 => Bad input pixel type for the first band of <input>.
| 9 => Bad input pixel type for the second band of <input>.
| 10 => The two bands of input image <input> have different size.
| 11 => Can't make image <output>.
| 12 => Can't make band of output image <output>.
| 13 => Can't open file <filename>.

Examples: | Try yourself.

Restrictions: Images only UNS_BYTE pixels, with two bands with equal size. The first band have the amplitude and the second band have the directions of the output results of an edges detector, for example Deriche.

```
                The value for tprobable must be greater than or equal to 2, smaller
                than or equal to 255 and smaller than tsure.
                The value for tsure must be greater than or equal to tprobable and
                smaller than or equal to 255.
                The value for linecolor must be greater than or equal to 0 and smaller
                than or equal to 255.
                The value for backcolor must be greater than or equal to 0 and smaller
                than or equal to 255.
                Width of mask 'wmclose' must be odd.

Author:         Joao Tavares

Id:             @(#)dstrip.c   2.0   94/03/5
_____________________________________________________________________
*/

#ifdef MAIN

main(argc, argv)
int argc;
char **argv;

{

  IMAGE imgin, imgout;
  int ts, tp, ed, wm, lcolor, bcolor, xsize, ysize, mark, aux1, aux2;
  float dadm;
  char *filename, *title, arg[250];
  FILE *fpt;

  InitMessage(&argc, argv, "Usage: %s\n [-t <title>] [-f <filename>] [-ts <tsure>] [-tp <tprobable>]
[-e <erdirection>] [-d <dadm>] [-w <wmclose>] [-p <1PixWid>] [-l <linecolor>] [-b <backcolor>]
<input> <output>\n");
  Iset_message(1);
  Iset_abort(1);

  /* read switches */

  title = read_switch(&argc, argv, "-t", 1, "Dynamic Strip");
  filename = read_switch(&argc, argv, "-f", 1, NULL);
  ts = atoi(read_switch(&argc, argv, "-ts", 1, "100"));
  tp = atoi(read_switch(&argc, argv, "-tp", 1, "50"));
  ed = atoi(read_switch(&argc, argv, "-e", 1, "5"));
  dadm = atof(read_switch(&argc, argv, "-d", 1, "1.0"));
  wm = atoi(read_switch(&argc, argv, "-w", 1, "5"));
  mark = atoi(read_switch(&argc, argv, "-p", 1, "0"));
  lcolor = atoi(read_switch(&argc, argv, "-l", 1, "0"));
  bcolor = atoi(read_switch(&argc, argv, "-b", 1, "255"));

  if (argc == 1) Usage(1, (char*)0);
  if (argc != 3) Usage(2, "\nBad number of arguments.\n");
  if (wm%2 == 0)  exit(Error(1, "\nWrong width of mask for closing line %d; it must be odd.\n",
wm));
  if (tp < 2 || tp > 255) exit(Error(2, "\nWrong value for tproblable %d; it must be greater than or
equal to 2 and smaller than or equal to 255.\n", tp));
  if (ts < tp || ts > 255) exit(Error(3, "\nWrong value for tsure %d; it must be greater than or equal to
tprobable %d and smaller than or equal to 255.\n", ts, tp));
  if (lcolor < 0 || lcolor > 255) exit(Error(4, "\nWrong value for linecolor %d; it must be greater than
or equal to 0 and smaller than or equal to 255.\n", lcolor));
  if (bcolor < 0 || bcolor > 255) exit(Error(5, "\nWrong value for backcolor %d; it must be greater
than or equal to 0 and smaller than or equal to 255.\n", bcolor));

  /* read the input image */

  imgin = Iread_image(argv[1]);
  if (imgin == NULL) exit(Error(6, "\nCan't read image %s.\n", argv[1]));
  if (Inbands(imgin) != 2) exit(Error(7, "\nThe %s image doesn't have two bands.\n", argv[1]));
  if (Ipixtyp(imgin[1]) != Iu_byte_typ) exit(Error(8, "\nBad input pixel type for the first band of
%s.\n", argv[1]));
  if (Ipixtyp(imgin[2]) != Iu_byte_typ) exit(Error(9, "\nBad input pixel type for the second band of
%s.\n", argv[1]));

  xsize = Ixsize(imgin[1]);
  ysize = Iysize(imgin[1]);
  aux1 = Ixsize(imgin[2]);
  aux2 = Iysize(imgin[2]);
  if (aux1 != xsize || aux2 != ysize) exit(Error(10, "\nThe two bands of input image %s have
different size.\n"));

  /* make the output image */

  imgout = Init_image(1, title);
  if (imgout == NULL) exit(Error(11, "\nCan't make image %s.\n", argv[2]));

  /* make band of the output image */

  imgout[1] = Imake_band(Iu_byte_typ, xsize, ysize);
  if (imgout[1] == NULL) exit(Error(12, "\nCan't make band of output image %s.\n", argv[2]));
```

```
  sprintf(arg, "Results for %s in %s with:\n\tSure threshold level for amplitude: %d.\n\tProbable
threshold level for amplitude: %d.\n\tError tolerance for direction: %d.\n\tError tolerance for
distance: %g.\n\tWidth of mask for close line: %d.\n\n", argv[0], argv[1], ts, tp, ed, dadm, wm);

  /* open file, if it was desire, and write arg */

  if (filename != NULL) {

    fpt = fopen(filename, "a");
    if (fpt == NULL) exit(Error(13, "\nCan't open file %s.\n", filename));
    fprintf(fpt, arg);

  }
  if (mark == 0 ) mark = 1;
  else mark = 0;

  /* call strip function */

  strip(imgin[1], imgin[2], imgout[1], lcolor, bcolor, ts, tp, ed, dadm, wm, mark, fpt);

  /* close file if it was desire */

  if (fpt != NULL) fclose(fpt);

  /* write output image with history */

  Ihistory(imgout, argv[0], arg);
  Iwrite_image(imgout,argv[2]);

}

#endif

/*************************************************************************/
```

Capítulo IV

Fusão de Segmentos de Recta Próximos e de Direcções Similares

Introdução

Neste capítulo são apresentadas duas implementações com interesse no domínio da fusão de segmentos de recta que pertençam a uma mesma entidade. Assim, são apresentadas as seguintes implementações:

- ✓ Fusão de segmentos de recta próximos e de direcções similares: ***linelink***.
- ✓ Desenho, numa imagem de saída, dos segmentos de recta definidos num ficheiro de entrada: ***showlines***.

O desenvolvimento da implementação ***showlines*** prende-se com o facto de os segmentos de recta a considerar pela implementação ***linelink*** serem introduzidos na forma dum ficheiro. Deste modo, a sua utilização torna-se desejável quando se pretende comparar a imagem obtida pela implementação com a imagem constituída pelo conjunto dos segmentos de recta que possuem mais do que um certo número de *pixels* e cujas coordenadas iniciais e finais estão definidas num ficheiro de entrada.

1 - Fusão de segmentos de recta próximos e de direcções similares: linelink

Nesta secção é apresentada uma implementação capaz de proceder à fusão de segmentos de recta que pertencem a uma mesma entidade, segundo o método descrito em [Tavares, 1995]. Resumidamente, este método consiste em:

i) Definir as coordenadas do centróide (x_G, y_G) do conjunto dos quatro pontos extremos dos dois segmentos de recta a fundir, sendo os respectivos comprimentos l_i e l_j as massas pontuais a considerar. Por este ponto deverá passar o segmento de fusão de menor erro.

ii) Definir a direcção do segmento de recta de fusão θ_r, como a média pesada das direcções dos segmentos a fundir, sendo os comprimentos destes os respectivos pesos.

iii) Definir um referencial X_G, Y_G, centrado no centróide determinado em ***i)*** e com o eixo X_G paralelo à direcção do segmento de fusão θ_r definida em ***ii)***.

iv) Definir as coordenadas dos pontos extremos *a*, *b*, *c* e *d*, dos dois segmentos a fundir *i* e *j*, no referencial X_G, Y_G, definido em ***iii)***.

v) As duas projecções mais afastadas no eixo X_G dos quatro pontos extremos *a*, *b*, *c* e *d*, dos dois segmentos de recta a fundir *i* e *j*, definem os pontos extremos do segmento resultante da fusão.

Algumas características desta implementação são:

- ✓ A possibilidade de fusão de segmentos de recta que sejam não sobrepostos; neste caso, os critérios de fusão são: direcção aproximadamente igual e distâncias de proximidade segundo os eixos X_G e Y_G inferiores aos respectivos máximos admissíveis, definidos pelo utilizador.
- ✓ A possibilidade de fusão de segmentos de recta que sejam totalmente ou parcialmente sobrepostos; neste caso, os critérios de fusão são: direcção aproximadamente igual, distância de proximidade segundo o eixo Y_G inferior a um dado máximo admissível, definido pelo utilizador.
- ✓ A implementação procede em primeiro lugar à fusão dos segmentos de recta que sejam não sobrepostos, de seguida procede à fusão dos segmentos que sejam parcialmente sobrepostos e, por último, procede à fusão dos segmentos que sejam totalmente sobrepostos, utilizando em cada fase os respectivos critérios de fusão.

- ✓ A ordem de fusão é imposta pelo comprimento dos segmentos de recta, sendo fundidos em primeiro lugar os segmentos de maior comprimento.
- ✓ A fusão de dois segmentos de recta só é efectuada se nenhum dos segmentos foi fundido com outro.
- ✓ A direcção do segmento de recta resultante é definida como a média pesada das direcções dos dois segmentos a fundir, sendo os comprimentos os respectivos pesos.
- ✓ A localização do segmento de recta resultante é devidamente influenciada pelos comprimentos dos segmentos a fundir.
- ✓ A possibilidade de eliminação de falsos segmentos de recta, isto é, segmentos que sejam constituídos por um número de *pixels* inferior a um mínimo admissível, definido pelo utilizador, não são considerados pela implementação.
- ✓ A implementação de um módulo responsável pelo desenho dos segmentos de recta resultantes, na imagem de saída.
- ✓ A possibilidade de saída dos resultados obtidos (as coordenadas dos *pixels* inicial e final de cada segmento de recta resultante da fusão) num ficheiro, para futura utilização.
- ✓ A possibilidade de utilização da implementação iterativamente; isto é, aplicar novamente a implementação a um conjunto de segmentos que já sofreu uma prévia fusão. Assim, é possível controlar mais refinadamente todo o processo de fusão.

1.1 - Descrição dos diferentes módulos

A implementação é constituída pelos módulos: ***principal***, ***linking*** e ***dda***, *Fig.* 1. Apresenta-se, de seguida, a descrição de cada um destes módulos.

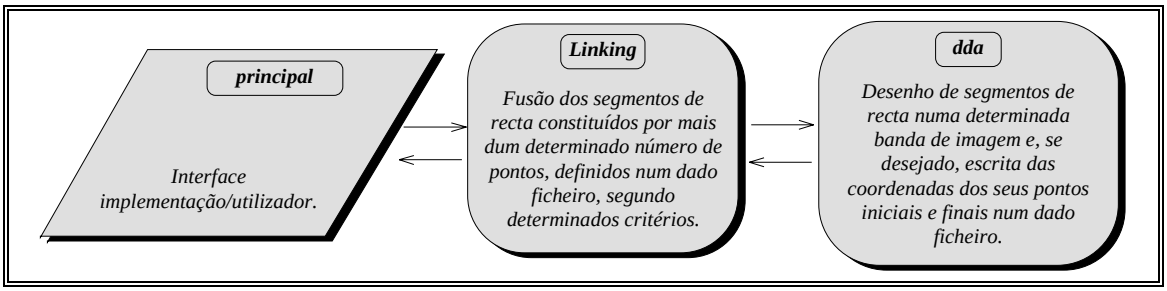


Fig. 1- Módulos integrantes da implementação ***linelink*** *e suas relações.*

1.1.1 - Módulo principal

Este módulo é responsável pela interface implementação/utilizador. A seu cargo está:

- ✓ Leitura dos argumentos opcionais ("*switches*"):
  - *title*, ("[-t <title>]"), título da imagem de saída. Por defeito, é "*Lines after linking*".
  - *lineColor*, ("[-l <lineColor>]"), valor que deverá ser atribuído aos *pixels* pertencentes aos segmentos de recta resultantes da fusão, na imagem de saída. Por defeito, o seu valor é *0*, correspondente ao preto.
  - *backColor*, ("[-b <backColor>]"), valor que deverão ter todos os *pixels* na imagem de saída que não pertençam aos segmentos de recta resultantes da fusão, isto é, os *pixels* que constituem o fundo da imagem. Por defeito, o seu valor é *255*, correspondente ao branco.
  - *xsize*, ("[-x <xsize>]"), dimensão segundo *x* da imagem de saída. Por defeito, o seu valor é *256*.

- *ysize*, ("[-y <ysize>]"), dimensão segundo *y* da imagem de saída. Por defeito, o seu valor é *256*.
- *maxDirDif*, ("[-d <maxDirDif>]"), máxima diferença de direcção entre dois segmentos de recta para que seja possível proceder à sua fusão. Por defeito, o seu valor é *5* grau.
- *maxDist*, ("[-d1 <maxDist>]"), máxima distância segundo o eixo X_G entre dois segmentos de recta que sejam não sobrepostos para que seja possível proceder à sua fusão. Por defeito, o seu valor é *8*.
- *maxWstrip*, ("[-w <maxWstrip>]"), máxima distância segundo o eixo Y_G entre dois segmentos de recta que sejam não sobrepostos para que seja possível proceder à sua fusão. Por defeito, o seu valor é *6*.
- *maxWstripParOverlap*, ("[-w1 <maxWstripParOverlap>]"), máxima distância segundo o eixo Y_G entre dois segmentos de recta que sejam parcialmente sobrepostos para que seja possível proceder à sua fusão. Por defeito, o seu valor é *4*.
- *maxWstripToOverlap*, ("[-w2 <maxWstripToOverlap>]"), máxima distância segundo o eixo Y_G entre dois segmentos de recta que sejam totalmente sobrepostos para que seja possível proceder à sua fusão. Por defeito, o seu valor é *2*.
- *nPixGoLine*, ("[-n <nPixGoLine>]"), mínimo número de *pixels* que devem constituir um segmento de recta, para não ser considerado como ruído. Por defeito, o seu valor é *20*.
- *outputFilename*, ("[-f <outputFilename>]"), nome do ficheiro para escrita dos resultados obtidos (coordenadas dos *pixels* que definem cada segmento de recta resultante) e respectiva descrição do modo como foram obtidos. Por defeito é "*NULL*", o que implica a não abertura do ficheiro. Quando especificado, o ficheiro, se ainda não existir, é criado; caso contrário, os resultados são acrescentados ao ficheiro já existente.

✓ Leitura dos argumentos passados à implementação pela linha de comando:

- *inputFile*, ("<inputFile>"), nome do ficheiro de entrada que contém as coordenadas dos *pixels* inicial e final de cada segmento de recta a ser considerado para efeitos de fusão.
- *outputImage*, ("<outputImage>"), nome da imagem de saída que apresentará os segmentos de recta obtidos após a fusão.

✓ Abertura do ficheiro de entrada.

✓ Criação da imagem de saída.

✓ Criação da banda da imagem de saída.

✓ Abertura, se pretendido pelo utilizador, do ficheiro para escrita dos resultados obtidos, com descrição do modo como estes foram obtidos.

✓ Chamada do módulo ***linking***.

✓ Fecho do ficheiro de saída de resultados, se pretendido pelo utilizador.

✓ Escrita da imagem de saída em disco, com a descrição de como foi obtida.

✓ Comunicação da existência de erros em argumentos de entrada ou ao longo do decurso da execução:

- Através de valores de retorno:
  - 1, ("Wrong value for line color <lineColor>; it must be greater than or equal to 0 and smaller than or equal to 255."), o valor especificado para *lineColor* não está compreendido entre *0* a *255*.

- 2, ("Wrong value for background color <backColor>; it must be greater than or equal to 0 and smaller than or equal to 255."), o valor especificado para *backColor* não está compreendido entre *0* a *255*.
- 3, ("Can't open input file <inputFilename>."), o módulo não consegue abrir o ficheiro de entrada especificado.
- 4, ("Can't make output image <outputImage>."), o módulo não consegue criar a imagem de saída especificada.
- 5, ("Can't make band of output image <outputImage>"), o módulo não consegue criar a banda da imagem de saída especificada.
- 7, ("Can't open output file <outputFilename>"), o módulo não consegue abrir o ficheiro de saída especificado para escrita de resultados obtidos.

- Através de mensagens:
  - 1, (*InitMessage*), quando o número de argumentos é igual a um, isto é, apenas o nome da implementação.
  - 2, ("Bad number of arguments."), quando o número de argumentos é diferente de três.

Como restrições na utilização deste módulo têm-se: *lineColor* e *backColor* deverão ter valores compreendidos entre *0* a *255*.

1.1.2 - Módulo linking

Este módulo é responsável pela fusão de segmentos de recta que pertencem a uma mesma entidade, segundo o método descrito em [Tavares, 1995]. Como parâmetros de entrada para este módulo têm-se:

- *band*, do tipo "*IBAND*", banda de saída com desenho dos segmentos de recta resultantes da implementação.
- *lcolor*, do tipo inteiro, valor que deverá ser atribuído, na imagem de saída, aos *pixels* pertencentes aos segmentos de recta resultantes da implementação.
- *bcolor*, do tipo inteiro, valor que deverão ter todos os *pixels*, na imagem de saída, que não pertençam aos segmentos de recta resultantes da implementação, isto é, os *pixels* que constituem o fundo da imagem.
- *mdir*, do tipo real, máxima diferença de direcção, em grau, entre dois segmentos de recta para que seja possível proceder à sua fusão.
- *mdist*, do tipo real, máxima distância segundo o eixo X_G entre dois segmentos de recta que não sejam sobrepostos, para que seja possível proceder à sua fusão.
- *wstrip*, do tipo real, máxima distância segundo o eixo Y_G entre dois segmentos de recta que não sejam sobrepostos, para que seja possível proceder à sua fusão.
- *wstrippo*, do tipo real, máxima distância segundo o eixo Y_G entre dois segmentos de recta que sejam parcialmente sobrepostos, para que seja possível proceder à sua fusão.
- *wstripto*, do tipo real, máxima distância segundo o eixo Y_G entre dois segmentos de recta que sejam totalmente sobrepostos, para que seja possível proceder à sua fusão.
- *nline*, do tipo inteiro, mínimo número de *pixels* que um dado segmento de recta deverá ter, para não ser considerado como ruído.

O módulo ***linking***, no início da sua tarefa, determina quantos segmentos de recta, constituídos por mais de *nline pixels*, existem no ficheiro de entrada. Após esta determinação, o módulo reserva a

quantidade de memória necessária. Seguidamente, este módulo atribui a cada *pixel* da banda da imagem de saída o valor correspondente a *bcolor*. Após esta atribuição, o módulo determina todas as características necessárias para cada segmento de recta que seja constituído por mais de *nline pixels*, nomeadamente: coordenadas dos *pixels* inicial e final, comprimento e orientação. Após esta fase, o módulo reordena os segmentos de recta pelo seu comprimento de forma decrescente. Na fase seguinte, o módulo passa a fundir os segmentos de recta que satisfaçam o critério de pertencerem a uma mesma entidade e de serem não sobrepostos. Após esta fusão, o módulo passa à fase de fusão dos segmentos de recta que satisfaçam o critério de pertencerem a uma mesma entidade e que sejam parcialmente sobrepostos. De seguida, este módulo procede à fusão dos segmentos de recta que satisfaçam o critério de pertencerem a uma mesma entidade e que sejam totalmente sobrepostos.

Para desenho na imagem de saída e para escrita no ficheiro de saída (quando desejado pelo utilizador) das coordenadas dos *pixels* iniciais e finais dos segmentos de recta resultantes (segmentos não fundidos e fundidos), este módulo utiliza o módulo ***dda***.

De notar que a fusão de dois segmentos de recta só se efectua se nenhum deles tiver sido fundido antes.

Este módulo tem como única restrição quanto à sua utilização a existência de memória suficiente e retorna:

- 0, como indicação de que o processo decorreu com normalidade,
- 1, ("Not enough memory."), como indicação de inexistência de memória suficiente.

1.1.3 - Módulo dda

Este módulo é responsável por:

- ✓ Desenho dos segmentos de recta obtidos pelo módulo ***linking***, na imagem de saída.
- ✓ Escrita das coordenadas dos *pixels* inicial e final de cada segmento de recta resultante, no ficheiro de saída, se desejado pelo utilizador (quando o apontador global *fptout* é diferente de "*NULL*").

Trata-se de uma implementação do algoritmo de Bresenham (também designado por versão meio ponto do segmento), que tem como grande vantagem a não utilização de multiplicações, de divisões e de funções de arredondamento. Pode-se consultar sobre este algoritmo [Foley, 1991] e, sobre uma sua implementação em linguagem C [Schildt, 1991].

Como parâmetros de entrada para este módulo têm-se:

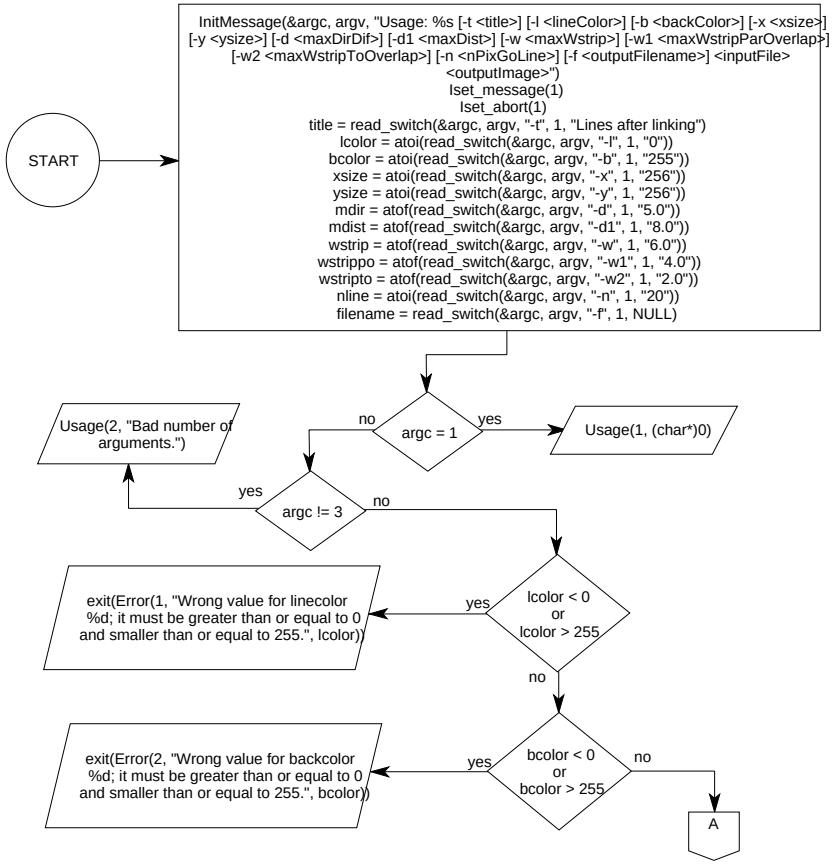
- *band*, do tipo "*IBAND*", banda da imagem de saída na qual é atribuído ao valor dos *pixels* pertencentes ao segmento de recta o valor *color*.
- *xi*, *yi*, do tipo inteiro, coordenadas do *pixel* inicial do segmento de recta.
- *xf*, *yf*, do tipo inteiro, coordenadas do *pixel* final do segmento de recta.
- *color*, do tipo inteiro, valor que é atribuído aos *pixels* pertencentes ao segmento de recta.

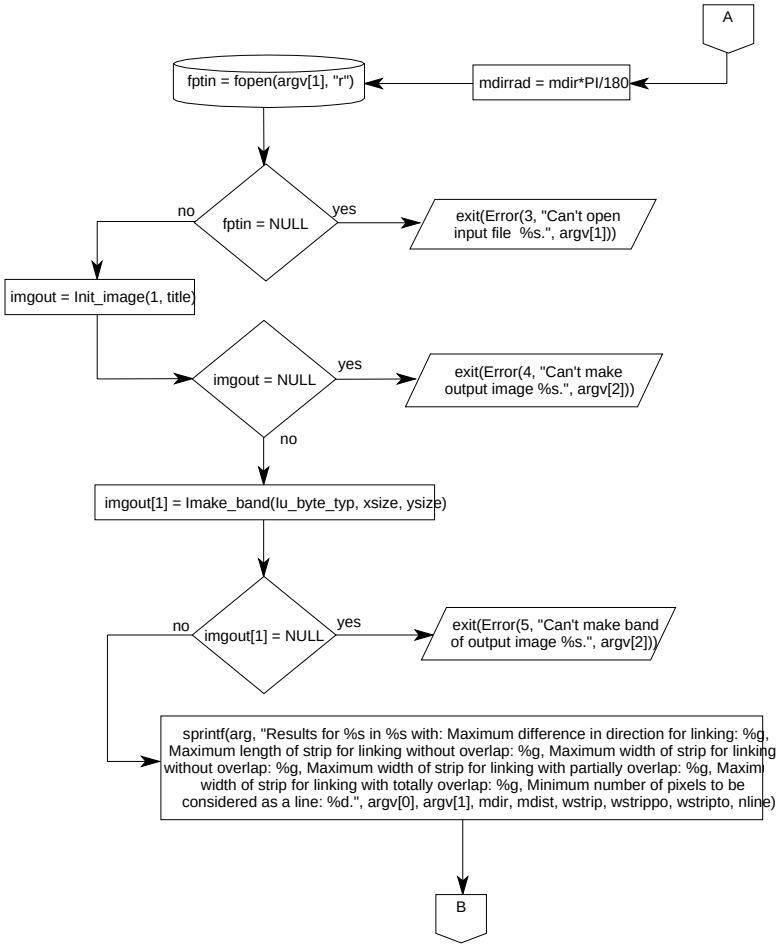
Este módulo retorna *0* como indicação de que o processo decorreu com normalidade, e não tem qualquer tipo de restrições para a sua utilização.

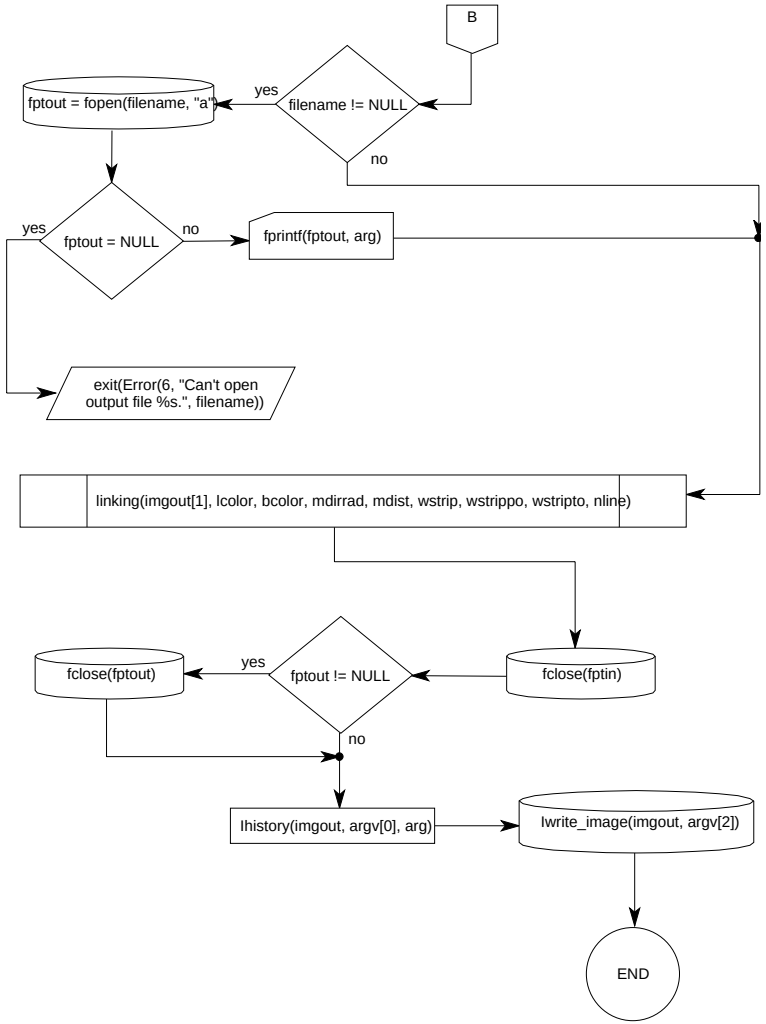
1.2 - Fluxogramas dos diferentes módulos

Apresentam-se, de seguida, os fluxogramas dos diferentes módulos que constituem a implementação ***linelink***.

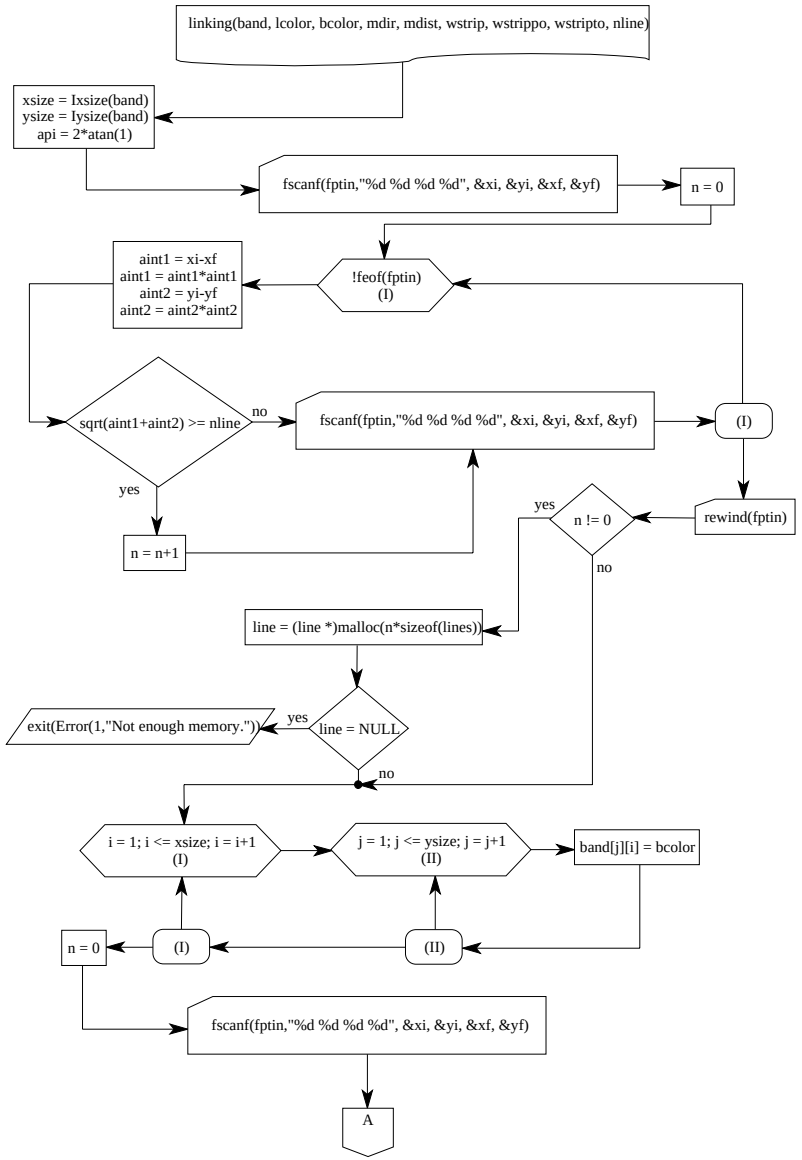
1.2.1 - Módulo principal

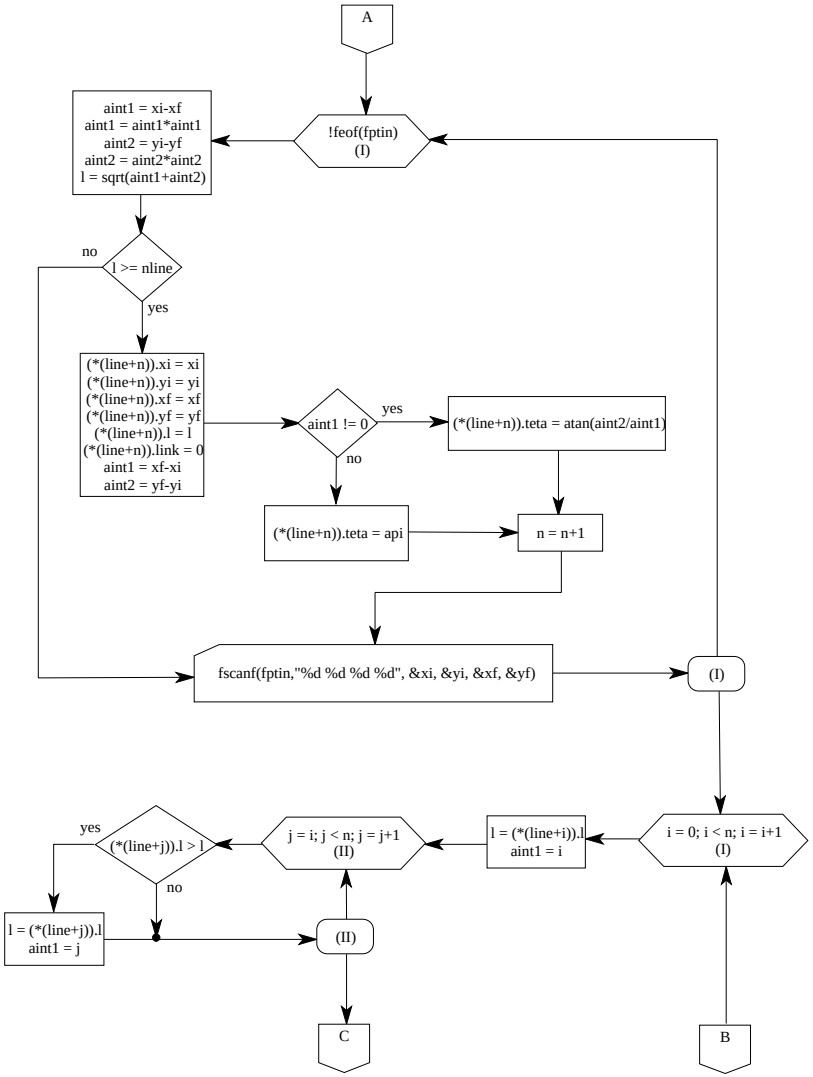


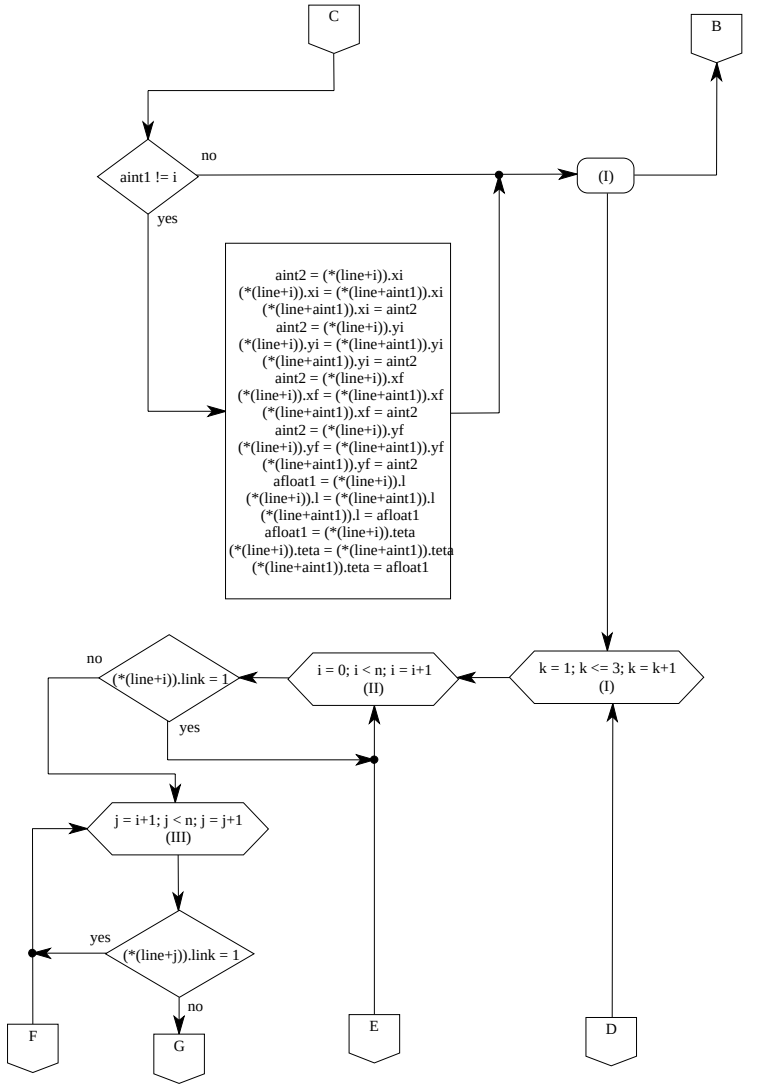


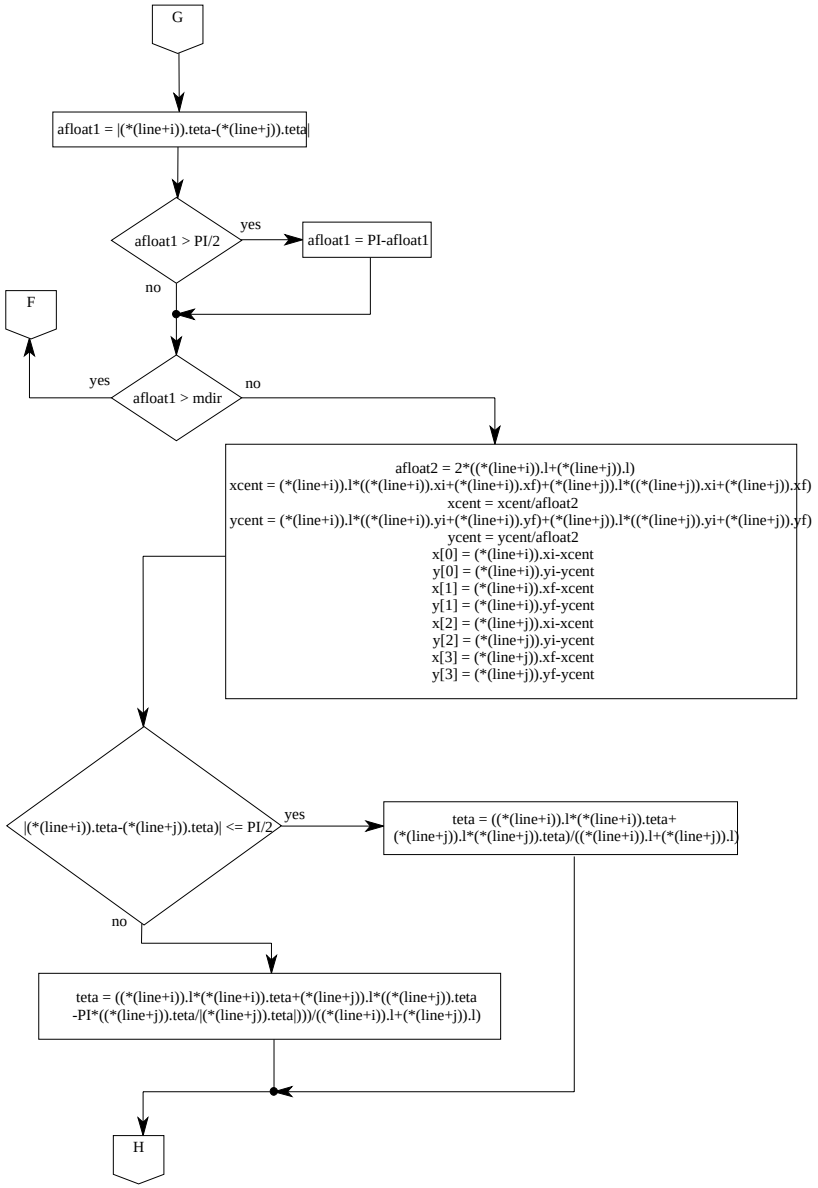


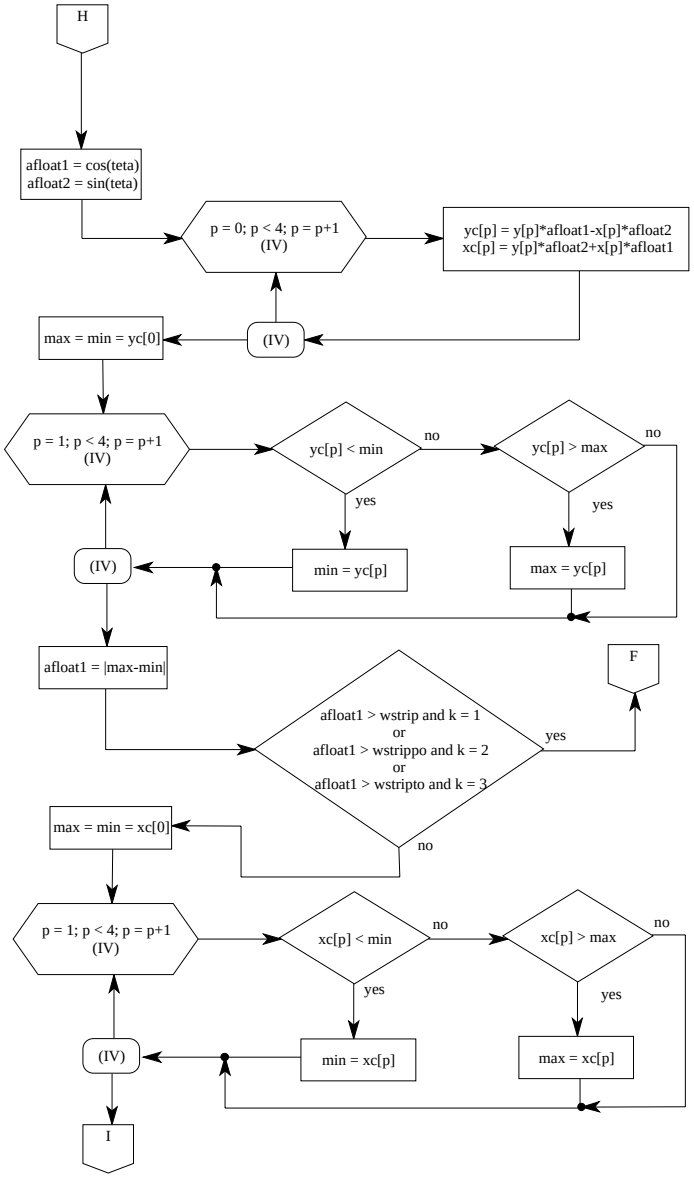
1.2.2 - Módulo linking

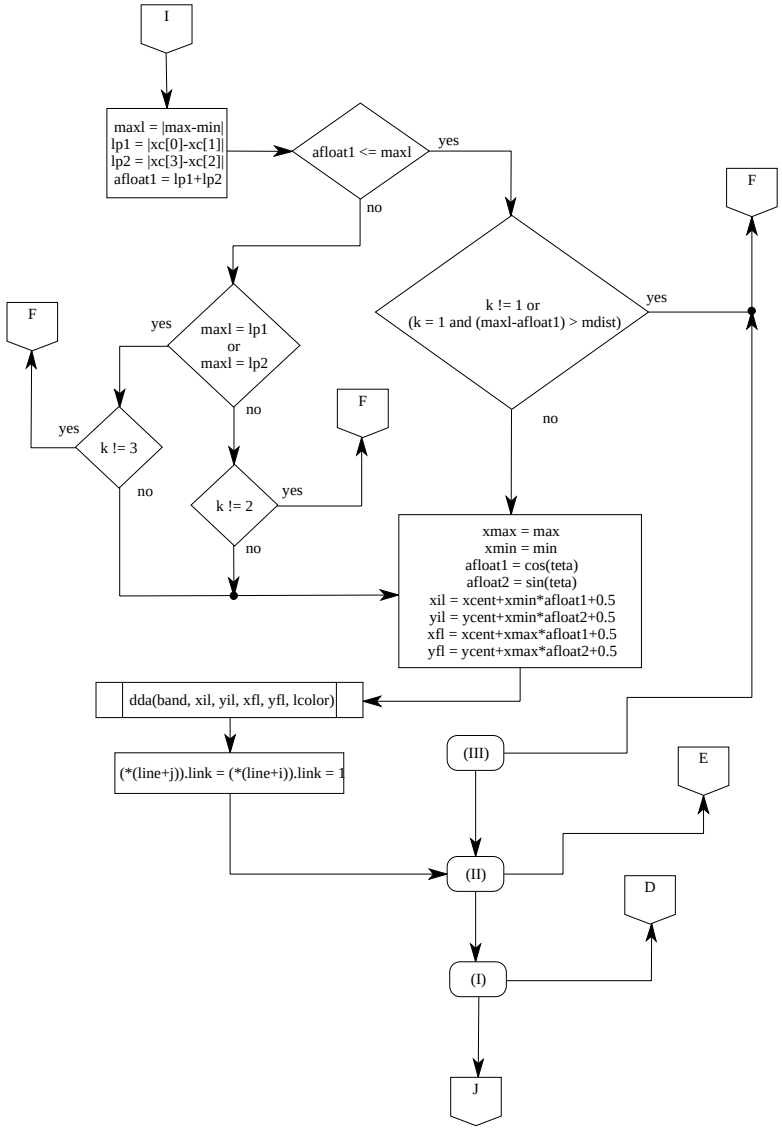


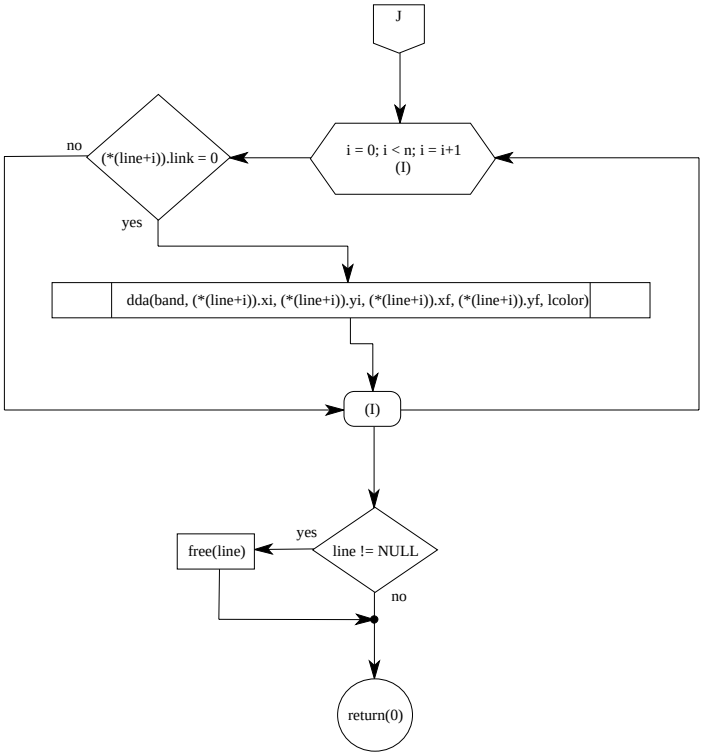

A
!feof(fptin)
(I)
aint1 = xi-xf
aint1 = aint1*aint1
aint2 = yi-yf
aint2 = aint2*aint2
l = sqrt(aint1+aint2)
l >= nline
no
yes
(*(line+n)).xi = xi
(*(line+n)).yi = yi
(*(line+n)).xf = xf
(*(line+n)).yf = yf
(*(line+n)).l = l
(*(line+n)).link = 0
aint1 = xf-xi
aint2 = yf-yi
aint1 != 0
yes
no
(*(line+n)).teta = atan(aint2/aint1)
(*(line+n)).teta = api
n = n+1
fscanf(fptin,"%d %d %d", &xi, &yi, &xf, &yf)
(I)
i = 0; i < n; i = i+1
(I)
l = (*(line+i)).l
aint1 = i
j = i; j < n; j = j+1
(II)
(*(line+j)).l > l
yes
no
l = (*(line+j)).l
aint1 = j
(II)
C
B


C
B
aint1 != i
no
yes
(I)
aint2 = (*(line+i)).xi
(*(line+i)).xi = (*(line+aint1)).xi
(*(line+aint1)).xi = aint2
aint2 = (*(line+i)).yi
(*(line+i)).yi = (*(line+aint1)).yi
(*(line+aint1)).yi = aint2
aint2 = (*(line+i)).xf
(*(line+i)).xf = (*(line+aint1)).xf
(*(line+aint1)).xf = aint2
aint2 = (*(line+i)).yf
(*(line+i)).yf = (*(line+aint1)).yf
(*(line+aint1)).yf = aint2
afloat1 = (*(line+i)).l
(*(line+i)).l = (*(line+aint1)).l
(*(line+aint1)).l = afloat1
afloat1 = (*(line+i)).teta
(*(line+i)).teta = (*(line+aint1)).teta
(*(line+aint1)).teta = afloat1
k = 1; k <= 3; k = k+1
(I)
i = 0; i < n; i = i+1
(II)
(*(line+i)).link = 1
no
yes
j = i+1; j < n; j = j+1
(III)
(*(line+j)).link = 1
yes
no
F
G
E
D

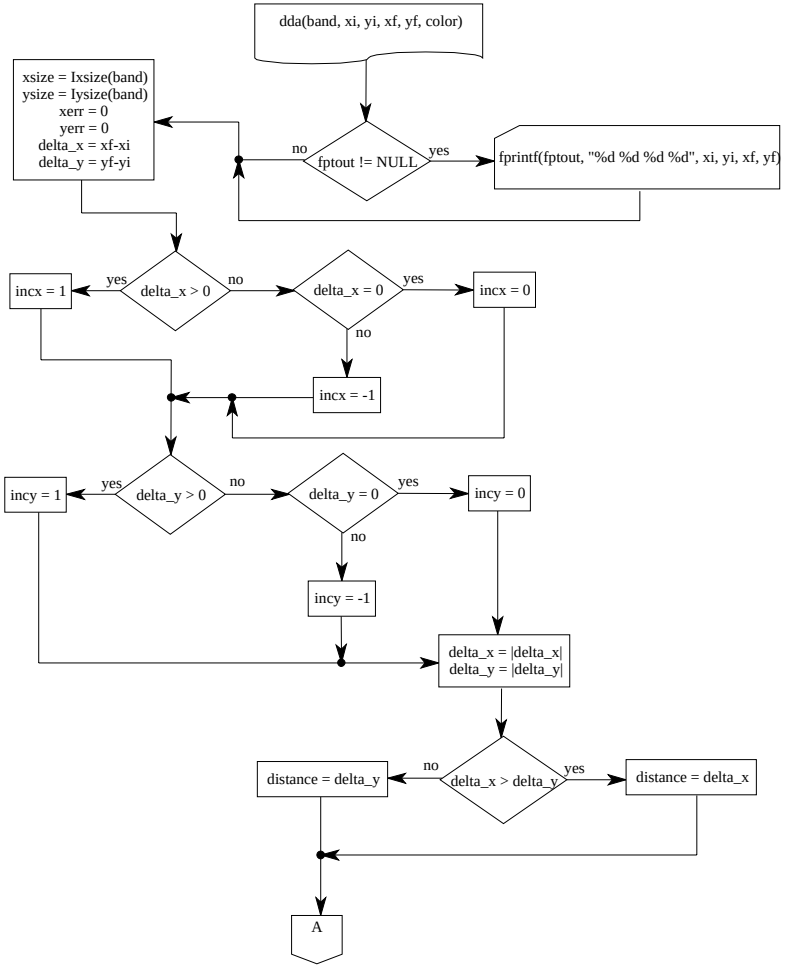

G
afloat1 = |(*(line+i)).teta-(*(line+j)).teta|
afloat1 > PI/2
yes
afloat1 = PI-afloat1
no
F
yes
afloat1 > mdir
no
afloat2 = 2*((*(line+i)).l+(*(line+j)).l)
xcent = (*(line+i)).l*((*(line+i)).xi+(*(line+i)).xf)+(*(line+j)).l*((*(line+j)).xi+(*(line+j)).xf)
xcent = xcent/afloat2
ycent = (*(line+i)).l*((*(line+i)).yi+(*(line+i)).yf)+(*(line+j)).l*((*(line+j)).yi+(*(line+j)).yf)
ycent = ycent/afloat2
x[0] = (*(line+i)).xi-xcent
y[0] = (*(line+i)).yi-ycent
x[1] = (*(line+i)).xf-xcent
y[1] = (*(line+i)).yf-ycent
x[2] = (*(line+j)).xi-xcent
y[2] = (*(line+j)).yi-ycent
x[3] = (*(line+j)).xf-xcent
y[3] = (*(line+j)).yf-ycent
|(*(line+i)).teta-(*(line+j)).teta)| <= PI/2
yes
teta = ((*(line+i)).l*(*(line+i)).teta+
(*(line+j)).l*(*(line+j)).teta)/((*(line+i)).l+(*(line+j)).l)
no
teta = ((*(line+i)).l*(*(line+i)).teta+(*(line+j)).l*((*(line+j)).teta
-PI*((*(line+j)).teta/|(*(line+j)).teta|)))/((*(line+i)).l+(*(line+j)).l)
H

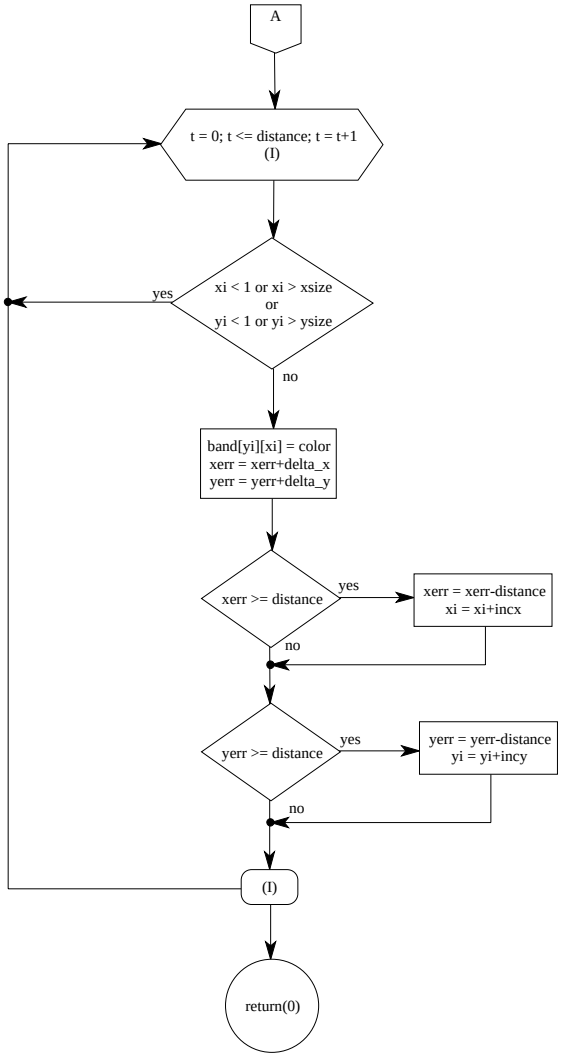

H
afloat1 = cos(teta)
afloat2 = sin(teta)
p = 0; p < 4; p = p+1
(IV)
yc[p] = y[p]*afloat1-x[p]*afloat2
xc[p] = y[p]*afloat2+x[p]*afloat1
(IV)
max = min = yc[0]
p = 1; p < 4; p = p+1
(IV)
yc[p] < min
no
yc[p] > max
no
yes
yes
(IV)
min = yc[p]
max = yc[p]
afloat1 = |max-min|
F
afloat1 > wstrip and k = 1
or
afloat1 > wstrippo and k = 2
or
afloat1 > wstripto and k = 3
yes
no
max = min = xc[0]
p = 1; p < 4; p = p+1
(IV)
xc[p] < min
no
xc[p] > max
no
yes
yes
(IV)
min = xc[p]
max = xc[p]
I


I
maxl = |max-min|
lp1 = |xc[0]-xc[1]|
lp2 = |xc[3]-xc[2]|
afloat1 = lp1+lp2
afloat1 <= maxl
yes
no
F
k != 1 or
(k = 1 and (maxl-afloat1) > mdist)
yes
no
F
maxl = lp1
or
maxl = lp2
yes
no
F
k != 3
yes
no
k != 2
yes
no
xmax = max
xmin = min
afloat1 = cos(teta)
afloat2 = sin(teta)
xil = xcent+xmin*afloat1+0.5
yil = ycent+xmin*afloat2+0.5
xfl = xcent+xmax*afloat1+0.5
yfl = ycent+xmax*afloat2+0.5
dda(band, xil, yil, xfl, yfl, lcolor)
(*(line+j)).link = (*(line+i)).link = 1
(III)
E
(II)
D
(I)
J


J
i = 0; i < n; i = i+1
(I)
no
(*(line+i)).link = 0
yes
dda(band, (*(line+i)).xi, (*(line+i)).yi, (*(line+i)).xf, (*(line+i)).yf, lcolor)
(I)
yes
free(line)
line != NULL
no
return(0)

1.2.3 - Módulo dda





1.3 - Listagem da implementação

Apresenta-se, de seguida, a listagem da implementação ***linelink*** desenvolvida em linguagem *C* em ambiente *UNIX* numa estação de trabalho *SunSparc* e que se destina ao ambiente de processamento de imagem *XITE*[1], do qual utiliza algumas funções e tipos de variáveis já previamente definidas. Esta implementação contém comentários para auxílio à sua compreensão, assim como informação sobre os diferentes módulos, para sua posterior extracção através da ferramenta *CDOC(1)* existente no *XITE*.

```
/*
_____________________________________________________________________

  linelink.c
  @(#)linelink.c   1.00 94/03/11, Copyright 1994, DEEC, FEUP
  Departamento de Engenharia Electrotecnica e de Computadores
  Faculdade de Engenharia
  Rua dos Bragas
  4099 PORTO CODEX
  PORTUGAL
  E-mail: gpai@obelix.fe.up.pt
_____________________________________________________________________

  biff.h
  @(#)biff.h   1.23 92/04/10, Copyright 1990, Blab, UiO
  Image processing lab, Department of Informatics
  University of Oslo
  E-mail: blab@ifi.uio.no
_____________________________________________________________________

  readarg.h
  @(#)readarg.h   1.3 92/01/15, Copyright 1990, Blab, UiO
  Image processing lab, Department of Informatics
  University of Oslo
  E-mail: blab@ifi.uio.no
_____________________________________________________________________

  message.h
  @(#)message.h   1.3 92/01/15, Copyright 1990, Blab, UiO
  Image processing lab, Department of Informatics
  University of Oslo
  E-mail: blab@ifi.uio.no
_____________________________________________________________________

  Permission to use, copy, modify and distribute this software and its documentation for any purpose
and without fee is hereby granted, provided that this copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in supporting documentation and that the
name of B-lab, Department of Informatics or University of Oslo not be used in advertising or publicity
pertaining to distribution of the software without specific, written prior permission.

  B-LAB DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT
SHALL B-LAB BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL
DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA
OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER
TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
*/

/************************************************************************/

static char *SccsId = "@(#)linelink.c 1.00 94/03/01, DEEC, FEUP";

/************************************************************************/

/* INCLUDES */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <blab/biff.h>
#include <blab/readarg.h>

/************************************************************************/

/* DEFINES */

#define MPI 3.141592654   /* pi's value */

/************************************************************************/

/* GLOBAL VARIABLES */

typedef struct {
```

[1] *X-based Image processing Tools an Environmemt*, [Lønnestad, 1992].

```
  int xi;          /* coordinate x of line's initial point */
  int yi;          /* coordinate y of line's initial point */
  int xf;          /* coordinate x of line's final point */
  int yf;          /* coordinate y of line's final point */
  float l;         /* line's length */
  float teta;      /* line's direction */
  int link;        /* line's flag for indication that the line is already linked */

} lines;

FILE *fptin, *fptout; /* files's pointers */

/*************************************************************************/

/*F:linking*
_________________________________________________________________________

  linking

_________________________________________________________________________

Name:          linking - link/merge segments that belongs to the same line

Syntax:     | int linking(band, lcolor, bcolor, xsize, ysize, mdir, mdist, wstrip,
            | wstrippo, wstripto, nline)
            | int lcolor, bcolor, xsize, ysize, nline;
            | float mdir, mdist, wstrip, wstrippo, wstripto;
            | IBAND band;

Description:  'linking' performs the link of segments that belongs to the same line.
              The output lines will be in 'band' with color 'lcolor' in a background
              'bcolor'.
              'xsize' and 'ysize' are the size of the output band 'band'.
              'mdir' is the maximum difference in directions for two segments
              could be considered as belongs to the same line.
              'mdist' is the maximum distance between the two segments without
              overlap in the center of "mass" system so they could be considered
              as belongs to the same line.
              'wstrip' is the maximum width of the strip for linking two segments
              without overlap so they could be considered as belongs to the same
              line.
              'wstrippo' is the maximum width of the strip for linking two segments
              partially overlap so they could be considered as belongs to the same
              line.
              'wstripto' is the maximum width of the strip for linking two segments
              totally overlap so they could be considered as belongs to the same line.
              'nline' it's the minimum number of pixel's of a segment so it could be
              considered as a good segment.
              'fptin' it's a global pointer to the file with the initial and final
              coordinates of all the segments to be considered.
              'fptout' it's a global pointer to the file in which the initial and final
              coordinates of all lines founded by 'linking' could be writing
              (if fptout!=NULL).

Return value: | 0 => Ok.
              | 1 => Not enough memory.

Example:      | Try yourself.

Restrictions: None.

Author:       Joao Tavares

Id:           @(#)linking.c   1.00   94/03/11
_________________________________________________________________________
*/

int linking(band, lcolor, bcolor, mdir, mdist, wstrip, wstrippo, wstripto, nline)
int lcolor, bcolor, nline;
float mdir, mdist, wstrip, wstrippo, wstripto;
IBAND band;

{

  register int i, j, k, p;
  int xsize, ysize, n, xi, yi, xf, yf, aint1, aint2, xil, yil, xfl, yfl;
  float teta, l, lp1, lp2, api, xcent, ycent, x[4], y[4], xc[4], yc[4], maxl, max, min, xmax, xmin,
afloat1, afloat2;
  lines *line;

  xsize = Ixsize(band);
  ysize = Iysize(band);
  api = 2.0*atan(1.0);

  /* find how many good lines are in the input file */

  fscanf(fptin, "%d %d %d %d", &xi, &yi, &xf, &yf);
  n = 0;
  while (!feof(fptin)) {

    aint1 = xi-xf;
```

```
  aint1 *= aint1;
  aint2 = yi-yf;
  aint2 *= aint2;
  if (sqrt((float)(aint1+aint2)) >= nline) n += 1;
  fscanf(fptin, "%d %d %d %d", &xi, &yi, &xf, &yf);

}
rewind(fptin);
if (n != 0) {

  /* allocation of memory */

  line = (lines *)malloc(n*sizeof(lines));
  if(line == NULL) exit(Error(1,"\nNot enough memory.\n"));

}

/* make all the pixels in band as bcolor */

for (i = 1; i <= xsize; ++i) {

  for (j = 1; j <= ysize; ++j) band[j][i]=bcolor;

}

/* read and compute some features of all the good lines in the input file */

n = 0;
fscanf(fptin, "%d %d %d %d", &xi, &yi, &xf, &yf);
while (!feof(fptin)) {

  aint1 = xi-xf;
  aint1 *= aint1;
  aint2 = yi-yf;
  aint2 *= aint2;
  l = sqrt((float)(aint1+aint2));
  if (l >= nline) {

    (*(line+n)).xi = xi;
    (*(line+n)).yi = yi;
    (*(line+n)).xf = xf;
    (*(line+n)).yf = yf;
    (*(line+n)).l = l;
    (*(line+n)).link = 0;
    aint1 = xf-xi;
    aint2 = yf-yi;
    if(aint1 != 0) (*(line+n)).teta = atan((float)(aint2)/(float)(aint1));
    else (*(line+n)).teta = api;
    n += 1;

  }
  fscanf(fptin, "%d %d %d %d", &xi, &yi, &xf, &yf);

}

/* order the lines by their length */

for (i = 0; i < n; ++i) {

  l = (*(line+i)).l;
  aint1 = i;
  for (j = i; j < n; ++j) {

    if ((*(line+j)).l > l) {

      l = (*(line+j)).l;
      aint1 = j;

    }

  }
  if (aint1 != i) {

    /* change the features of this lines */

    aint2 = (*(line+i)).xi;
    (*(line+i)).xi = (*(line+aint1)).xi;
    (*(line+aint1)).xi = aint2;
    aint2 = (*(line+i)).yi;
    (*(line+i)).yi = (*(line+aint1)).yi;
    (*(line+aint1)).yi = aint2;
    aint2 = (*(line+i)).xf;
    (*(line+i)).xf = (*(line+aint1)).xf;
    (*(line+aint1)).xf = aint2;
    aint2 = (*(line+i)).yf;
    (*(line+i)).yf = (*(line+aint1)).yf;
    (*(line+aint1)).yf = aint2;
    afloat1 = (*(line+i)).l;
    (*(line+i)).l = (*(line+aint1)).l;
    (*(line+aint1)).l = afloat1;
```

```
    afloat1 = (*(line+i)).teta;
    (*(line+i)).teta = (*(line+aint1)).teta;
    (*(line+aint1)).teta = afloat1;

  }

}

/* linking lines */

for (k = 1; k <= 3; ++k) {

  for (i = 0; i < n; ++i) {

    if ((*(line+i)).link == 1) continue;
    for (j = i+1; j < n; ++j) {

      if ((*(line+j)).link == 1) continue;
      afloat1 = fabs((*(line+i)).teta-(*(line+j)).teta);
      if (afloat1 > api) afloat1 = MPI-afloat1;
      if (afloat1 > mdir) continue;

      /* find the center of "mass" */

      afloat2 = 2.0*((*(line+i)).l+(*(line+j)).l);
      xcent = (*(line+i)).l*((*(line+i)).xi+(*(line+i)).xf)+(*(line+j)).l*((*(line+j)).xi+ (*(line+j)).xf);
      xcent /= afloat2;
      ycent = (*(line+i)).l*((*(line+i)).yi+(*(line+i)).yf)+(*(line+j)).l*((*(line+j)).yi+ (*(line+j)).yf);
      ycent /= afloat2;
      x[0] = (*(line+i)).xi-xcent;
      y[0] = (*(line+i)).yi-ycent;
      x[1] = (*(line+i)).xf-xcent;
      y[1] = (*(line+i)).yf-ycent;
      x[2] = (*(line+j)).xi-xcent;
      y[2] = (*(line+j)).yi-ycent;
      x[3] = (*(line+j)).xf-xcent;
      y[3] = (*(line+j)).yf-ycent;

      /* compute the direction of the linking line */

      if (fabs((*(line+i)).teta-(*(line+j)).teta) <= api)

        teta = ((*(line+i)).l*(*(line+i)).teta+(*(line+j)).l*(*(line+j)).teta)/((*(line+i)).l+ (*(line+j)).l);

      else

        teta = ((*(line+i)).l*(*(line+i)).teta+(*(line+j)).l*((*(line+j)).teta-
MPI*((*(line+j)).teta/fabs((*(line+j)).teta))))/((*(line+i)).l+(*(line+j)).l);

      /* compute the initial and final point of each line in the center of "mass" system */

      afloat1 = cos(teta);
      afloat2 = sin(teta);
      for (p = 0; p < 4; ++p) {

        yc[p] = y[p]*afloat1-x[p]*afloat2;
        xc[p] = y[p]*afloat2+x[p]*afloat1;

      }

      /* compute the width of the strip for link this lines */

      max = min = yc[0];
      for (p = 1; p < 4; ++p) {

        if (yc[p] < min) min = yc[p];
        else if (yc[p] > max) max = yc[p];

      }
      afloat1 = fabs(max-min);
      if ((afloat1 > wstrip && k == 1) || (afloat1 > wstrippo && k == 2) || (afloat1 > wstripto && k
== 3)) continue;

      /* compute the length of the strip for link this lines */

      max = min = xc[0];
      for (p = 1; p < 4; ++p) {

        if (xc[p] < min) min = xc[p];
        else if (xc[p] > max) max = xc[p];

      }
      maxl = fabs(max-min);

      /* find if this lines are overlap */

      lp1 = fabs(xc[0]-xc[1]);
      lp2 = fabs(xc[2]-xc[3]);
      afloat1 = lp1+lp2;
      if (afloat1 <= maxl){
```

```
  /* this lines aren't overlap */

  if (k != 1 || (k == 1 && (maxl-afloat1) > mdist)) continue;

}
else {

  /* this lines are overlap */

  if (maxl == lp1 || maxl == lp2) {

    /* this lines are totally overlap */

    if (k != 3) continue;

  }
  else {

    /* this lines are partially overlap */

    if (k != 2) continue;

  }

}

/* compute the initial and final point of the linking line */

xmax = max;
xmin = min;
afloat1 = cos(teta);
afloat2 = sin(teta);
xil = xcent+xmin*afloat1+0.5;
yil = ycent+xmin*afloat2+0.5;
xfl = xcent+xmax*afloat1+0.5;
yfl = ycent+xmax*afloat2+0.5;

/* call dda function */

dda(band, xil, yil, xfl, yfl, lcolor);

/* mark this lines as link */

(*(line+j)).link = (*(line+i)).link = 1;

        break;

      }

    }

  }

  /* mark all the others lines */

  for (i = 0; i < n; ++i) {

    if ((*(line+i)).link == 0) dda(band, (*(line+i)).xi, (*(line+i)).yi, (*(line+i)).xf, (*(line+i)).yf,
lcolor);

  }

  /* free memory */

  if (line != NULL) free(line);

  return(0);

}

/*****************************************************************************/

/*F:dda*
_____________________________________________________________________________

  dda
_____________________________________________________________________________

Name:          dda - drawing a segment of line using the algorithm of Bresenham

Syntax:        | dda(xi, yi, xf, yf, band, color)
               | int xi, yi, xf, yf, color;
               | IBAND band;

Description:   'dda' draw the segment of line between the initial pixel 'xi, yi' and
                the final pixel 'xf, yf' in the band 'band' with color as 'color'.
                If the global pointer 'fptout' is different of NULL the initial and
                final points of all segments will be writing in 'fptout''s file.
```

```
Return value:   | 0 => Ok.

Example:        | Try yourself.

Restrictions:   None.

Author:         Joao Tavares

Id:             @(#)dda.c   1.00   93/05/05
_____________________________________________________________________
*/

int dda(band, xi, yi, xf, yf, color)
int xi, yi, xf, yf, color;
IBAND band;

{

  register int t, distance;
  int xsize, ysize, xerr, yerr, delta_x, delta_y, incx, incy;

  /* write in file, if it was desire, the coordinates of the initial and final pixels of the segment */

  if (fptout != NULL) fprintf(fptout, "\t%d %d\t%d %d\n", xi, yi, xf, yf);

  xsize = Ixsize(band);
  ysize = Iysize(band);
  xerr = 0;
  yerr = 0;

  /* calculation of delta x and delta y */

  delta_x = xf-xi;
  delta_y = yf-yi;

  /* determination of increment x and increment y */

  if (delta_x > 0) incx = 1;
  else if (delta_x == 0) incx = 0;
  else incx = -1;
  if (delta_y > 0) incy = 1;
  else if (delta_y == 0) incy = 0;
  else incy = -1;

  /* determination of the distance */

  delta_x = abs(delta_x);
  delta_y = abs(delta_y);

  if (delta_x > delta_y) distance = delta_x;
  else distance = delta_y;

  /* mark the closest point to the line with color */

  for (t = 0; t <= distance; t++) {

    if (xi < 1 || xi > xsize || yi < 1 || yi > ysize) continue;
    band[yi][xi] = color;
    xerr += delta_x;
    yerr += delta_y;
    if (xerr >= distance) {

      xerr -= distance;
      xi += incx;

    }
    if (yerr >= distance) {

      yerr -= distance;
      yi += incy;

    }

  }
  return(0);

}

/*********************************************************************/

/*P:linelink*
_____________________________________________________________________

  linelink
_____________________________________________________________________

Name:      linelink - link/merge segments that belongs to the same line

Syntax:   | [-t <title>] [-l <lineColor>] [-b <backColor>] [-x <xsize>]
          | [-y <ysize>] [-d <maxDirDif>] [-d1 <maxDist>] [-w <maxWstrip>]
```

```
        | [-w1 <maxWstripParOverlap>] [-w2 <maxWstripToOverlap>]
        | [-n <nPixGoLine>] [-f <outputFilename] <inputFile> <outputImage>

Description:  'linelink' perform the linking of segments that belongs to the same
              line.
              The output image will have the 'title' which is indicate through flag
              '-t', by default the 'title' is "Lines after linking".
              The output lines will be draw in 'outputImage' image with color
              'lineColor' in a 'backColor' background.
              The color of the output lines 'outputImage' image can be indicate
              trough flag '-l'. By default 'lineColor' is 0 (black).
              The color of 'outputImage' image background can be indicate
              trough flag '-b'. By default 'backColor' is 255 (white).
              The size of output image in x direction can be indicate trough flag
              '-x'. By default 'xsize' is 256.
              The size of output image in y direction can be indicate trough flag
              '-y'. By default 'ysize' is 256.
              The maximum difference in directions for two segments could be
              considered as belongs to the same line can be indicate through flag
              '-d'. By default 'maxDirDif' is 5 degree.
              The maximum distance between two segments without overlap in
              the center of "mass" system for they could be considered as belongs
              to the same line can be indicate through flag '-d1'. By default
              'maxDist' is 8.
              The maximum width of the strip for linking two segments without
              overlap so they could be considered as belongs to the same line can
              be indicate through flag '-w'. By default 'maxWstrip' is 6.
              The maximum width of the strip for linking two segments partially
              overlap so they could be considered as belongs to the same line can
              be indicate through flag '-w1'. By default 'maxWstripParOverlap' is 4.
              The maximum width of the strip for linking two segments totally
              overlap so they could be considered as belongs to the same line can
              be indicate through flag '-w2'. By default 'maxWstripToOverlap' is 2.
              The minimum number of segment's pixels to a segment could be
              considered as a good segment can be indicate through flag '-n'. By
              default 'nPixGoLine' is 20.
              The initial and final point's of each output line can be writing in the
              file 'outputFilename' if the flag '-f' is on, by default this flag is off.
              The input file 'inputFile' have the coordinates of the initial and final
              pixel of all the segments to be considered for linking.

Return value: | 1 => Wrong value for line color <lineColor>; it must be greater than
              |      or equal to 0 and smaller than or equal to 255.
              | 2 => Wrong value for background color <backColor>; it must be
              |      greater than or equal to 0 and smaller than or equal to 255.
              | 3 => Can't open input file <inputFile>.
              | 4 => Can't make output image <outputImage>.
              | 5 => Can't make band of output image <outputImage>.
              | 6 => Can't open output file <ouputFilename>.

Examples:     | Try yourself.

Restrictions: The value for line color and for background color must be greater
              than or equal to 0 and smaller than or equal to 255.

Author:       Joao Tavares

Id:           @(#)linelink.c   1.0   94/03/11
_____________________________________________________________________________
*/

#ifdef MAIN

main(argc, argv)
int argc;
char **argv;

{

  int lcolor, bcolor, xsize, ysize, nline;
  float mdir, mdirrad, mdist, wstrip, wstrippo, wstripto;
  char *filename, *title, arg[400];
  IMAGE imgout;

  InitMessage(&argc, argv, "Usage: %s\n [-t <title>] [-l <lineColor>] [-b <backColor>] [-x
<xsize>] [-y <ysize>] [-d <maxDirDif>] [-d1 <maxDist>] [-w <maxWstrip>] [-w1
<maxWstripParOverlap>] [-w2 <maxWstripToOverlap] [-n <nPixGoLine>] [-f <outputFilename]
<inputFile> <outputImage>\n");
  Iset_message(1);
  Iset_abort(1);

  /* read switches */

  title = read_switch(&argc, argv, "-t", 1, "Lines after linking");
  lcolor = atoi(read_switch(&argc, argv, "-l", 1, "0"));
  bcolor = atoi(read_switch(&argc, argv, "-b", 1, "255"));
  xsize = atoi(read_switch(&argc, argv, "-x", 1, "256"));
  ysize = atoi(read_switch(&argc, argv, "-y", 1, "256"));
  mdir = atof(read_switch(&argc, argv, "-d", 1, "5.0"));
  mdist = atof(read_switch(&argc, argv, "-d1", 1, "8.0"));
```

```
  wstrip = atof(read_switch(&argc, argv, "-w", 1, "6.0"));
  wstrippo = atof(read_switch(&argc, argv, "-w1", 1, "4.0"));
  wstripto = atof(read_switch(&argc, argv, "-w2", 1, "2.0"));
  nline = atoi(read_switch(&argc, argv, "-n", 1, "20"));
  filename = read_switch(&argc, argv, "-f", 1, NULL);

  if (argc == 1) Usage(1, (char*)0);
  if (argc != 3) Usage(2, "\nBad number of arguments.\n");
  if (lcolor < 0 || lcolor > 255) exit(Error(1, "\nWrong value for linecolor %d; it must be greater than
or equal to 0 and smaller than or equal to 255.\n", lcolor));
  if (bcolor < 0 || bcolor > 255) exit(Error(2, "\nWrong value for background color %d; it must be
greater than or equal to 0 and smaller than or equal to 255.\n", bcolor));

  mdirrad = mdir*MPI/180.0;

  /* open input file */

  fptin = fopen(argv[1], "r");
  if (fptin == NULL) exit(Error(3, "\nCan't open input file %s.\n", argv[1]));

  /* make output image */

  imgout = Init_image(1, title);
  if (imgout == NULL) exit(Error(4, "\nCan't make output image %s.\n", argv[2]));

  /* make band of output image */

  imgout[1] = Imake_band(Iu_byte_typ, xsize, ysize);

  if (imgout[1] == NULL) exit(Error(5, "\nCan't make band of output image %s.\n", argv[2]));

  sprintf(arg, "Results for %s in %s with:\n\n\tMaximum difference in direction for linking:
%g\n\tMaximum length of strip for linking without overlap: %g\n\tMaximum width of strip for
linking without overlap: %g\n\tMaximum width of strip for linking with partially overlap:
%g\n\tMaximum width of strip for linking with total overlap: %g\n\tMinimum number of pixels to
be considered as a line: %d.\n\n", argv[0], argv[1], mdir, mdist, wstrip, wstrippo, wstripto, nline);

  /* open output file if it was desire and write arg */

  if (filename != NULL) {

    fptout = fopen(filename, "a");
    if (fptout == NULL) exit(Error(6, "\nCan't open output file %s.\n", filename));
    fprintf(fptout, arg);
  }

  /* call linking function */

  linking(imgout[1], lcolor, bcolor, mdirrad, mdist, wstrip, wstrippo, wstripto, nline);

  /* close input file */

  fclose(fptin);

  /* close output file if it was desire */

  if (fptout != NULL) fclose(fptout);

  /* write output image with history */

  Ihistory(imgout, argv[0], arg);
  Iwrite_image(imgout,argv[2]);

}

#endif

/*****************************************************************************/
```

2 - Implementação para o desenho de segmentos de recta: showlines

Nesta secção é apresentada uma implementação que tem como principal objectivo o desenho, numa imagem de saída, dos segmentos de recta definidos num ficheiro. Algumas características desta implementação são:

- ✓ A possibilidade de eliminação de falsos segmentos de recta; isto é, segmentos que sejam constituídos por um número de *pixels* inferior a um dado valor, definido pelo utilizador, não são considerados.
- ✓ A possibilidade de saída dos resultados obtidos (as coordenadas dos *pixels* inicial e final que definem cada segmento de recta) num ficheiro, para futura utilização.

2.1 - Descrição dos diferentes módulos

A implementação é constituída pelos módulos: ***principal***, ***showl*** e ***dda***, este último já apresentado na secção anterior, *Fig. 2*. Apresenta-se de seguida a descrição dos dois primeiros módulos.

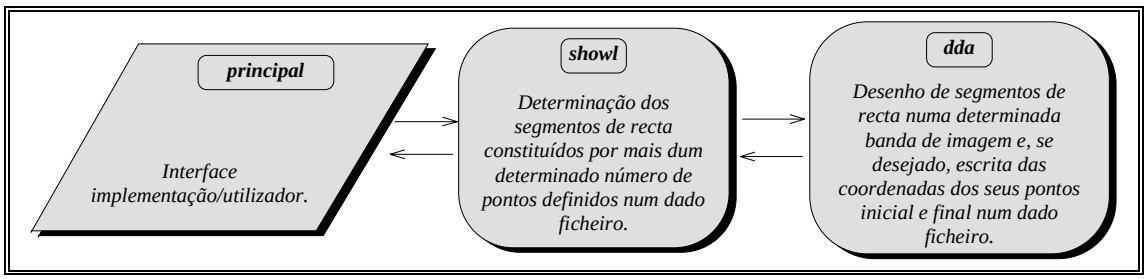


*Fig. 2 - Módulos integrantes da implementação **showlines** e suas relações.*

2.1.1 - Módulo principal

Este módulo é responsável pela interface implementação/utilizador. A seu cargo está:

- ✓ Leitura dos argumentos opcionais ("*switches*"):
  - *title*, ("[-t <title>]"), título da imagem de saída. Por defeito, é "*Lines in file*".
  - *lineColor*, ("[-l <lineColor>]"), valor que deverá ser atribuído aos *pixels* pertencentes aos segmentos de recta determinados pela implementação. Por defeito, o seu valor é *0*, correspondente ao preto.
  - *backColor*, ("[-b <backColor>]"), valor que deverão ter todos os *pixels* na imagem de saída que não pertençam aos segmentos de recta determinados pela implementação, isto é, os *pixels* que constituem o fundo da imagem. Por defeito, o seu valor é *255*, correspondente ao branco.
  - *xsize*, ("[-x <xsize>]"), dimensão segundo *x* da imagem de saída. Por defeito, o seu valor é *256*.
  - *ysize*, ("[-y <ysize>]"), dimensão segundo *y* da imagem de saída. Por defeito, o seu valor é *256*.
  - *nPixGoLine*, ("[-n <nPixGoLine>]"), mínimo número de *pixels* que um dado segmento de recta deve ter, para não ser considerado como ruído. Por defeito, o seu valor é *20*.
  - *outputFilename*, ("[-f <outputFilename>]"), nome do ficheiro para escrita dos resultados obtidos (coordenadas dos *pixels* que definem cada segmento de recta determinado). Por defeito é "*NULL*", o que implica a não abertura do ficheiro. Quando especificado, o

ficheiro, se ainda não existir, é criado; caso contrário, os resultados são acrescentados ao ficheiro já existente.

- ✓ Leitura dos argumentos passados à implementação pela linha de comando:
  - *inputFile*, (“<inputFile>”), nome do ficheiro de entrada que contém as coordenadas dos *pixels* iniciais e finais de cada segmento de recta a ser considerado.
  - *outputImage*, (“<outputImage>”), nome da imagem de saída que apresentará os segmentos de recta determinados pela implementação.
- ✓ Abertura do ficheiro de entrada.
- ✓ Criação da imagem de saída.
- ✓ Criação da banda da imagem de saída.
- ✓ Abertura, se pretendido pelo utilizador, do ficheiro para escrita dos resultados obtidos com descrição de como estes foram obtidos.
- ✓ Chamada do módulo ***showl***.
- ✓ Fecho do ficheiro de saída de resultados, se pretendido pelo utilizador.
- ✓ Escrita da imagem de saída em disco, com a descrição de como foi obtida.
- ✓ Comunicação da existência de erros em argumentos de entrada ou ao longo do decurso da execução:
  - Através de valores de retorno:
    - 1, (“Wrong value for line color <lineColor>; it must be greater than or equal to 0 and smaller than or equal to 255.”), o valor especificado para *lineColor* não está compreendido entre *0* a *255*.
    - 2, (“Wrong value for background color <backColor>; it must be greater than or equal to 0 and smaller than or equal to 255.”), o valor especificado para *backColor* não está compreendido entre *0* a *255*.
    - 3, (“Can’t open input file <inputFilename>.”), o módulo não consegue abrir o ficheiro de entrada especificado.
    - 4, (“Can’t make output image <outputImage>.”), o módulo não consegue criar a imagem de saída especificada.
    - 5, (“Can’t make band of output image <outputImage>”), o módulo não consegue criar a banda da imagem de saída especificada.
    - 6, (“Can’t open output file <outputFilename>”), o módulo não consegue abrir o ficheiro de saída especificado para escrita de resultados obtidos.
  - Através de mensagens:
    - 1, (*InitMessage*), quando o número de argumentos é igual a um, isto é, apenas o nome da implementação.
    - 2, (“Bad number of arguments.”), quando o número de argumentos é diferente de três.

Como restrições na utilização deste módulo têm-se: *lineColor* e *backColor* deverão ter valores compreendidos entre *0* e *255*.

2.1.2 - Módulo showl

Este módulo é responsável pela determinação dos segmentos de recta, constituídos por mais de um certo número de *pixels*, definidos no ficheiro de entrada e pela consequente passagem das coordenadas iniciais e finais ao módulo ***dda***.

Como parâmetros de entrada para este módulo têm-se:

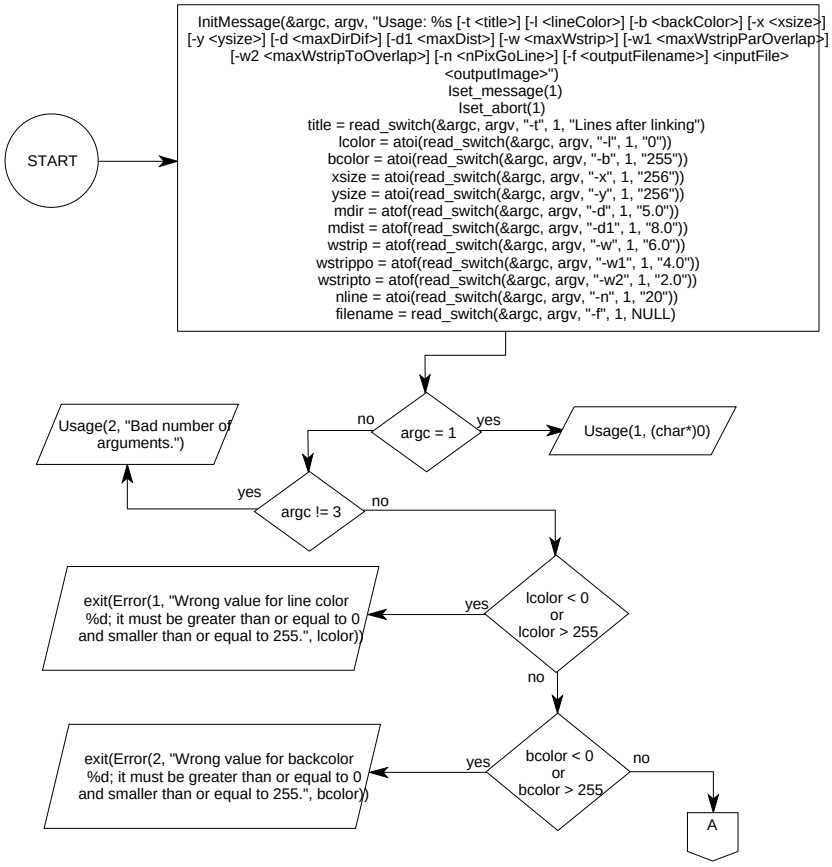
- *band*, do tipo "*IBAND*", banda de saída para o desenho dos segmentos de recta determinados pela implementação.
- *lcolor*, do tipo inteiro, valor que deverá ser atribuído, na imagem de saída, aos *pixels* pertencentes aos segmentos de recta determinados pela implementação.
- *bcolor*, do tipo inteiro, valor que deverão ter todos os *pixels* na imagem de saída que não pertençam aos segmentos de recta determinados pela implementação.
- *nline*, do tipo inteiro, mínimo número de *pixels* que um dado segmento de recta deverá ter, para não ser considerado como ruído.

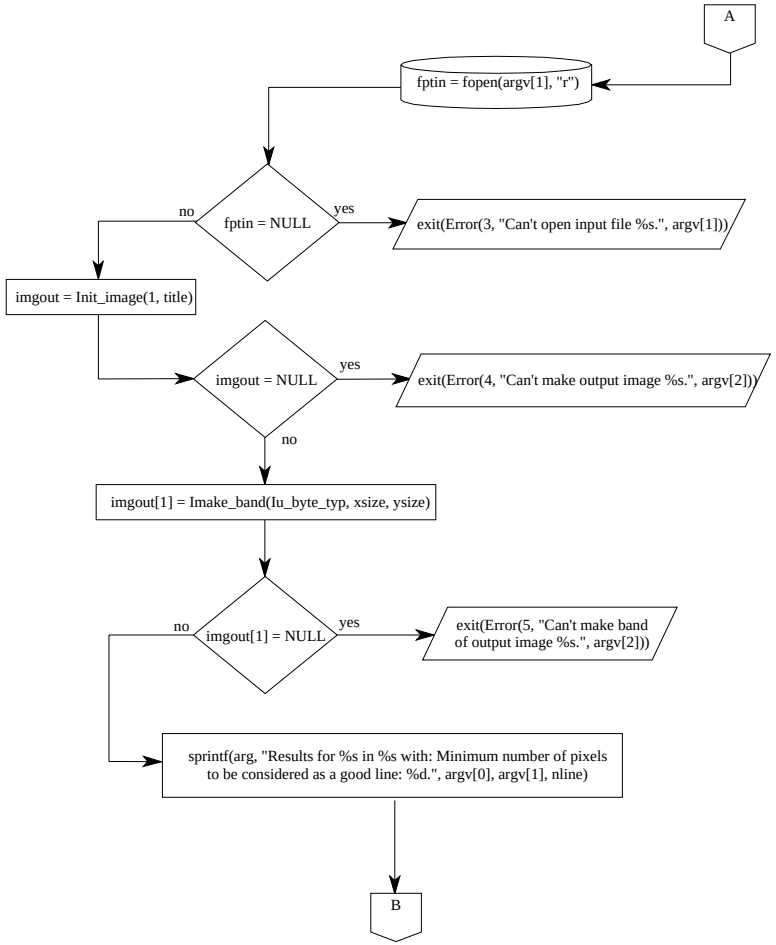
Este módulo retorna *0* como indicação de que o processo decorreu com normalidade, e não tem qualquer tipo de restrições quanto à sua utilização.

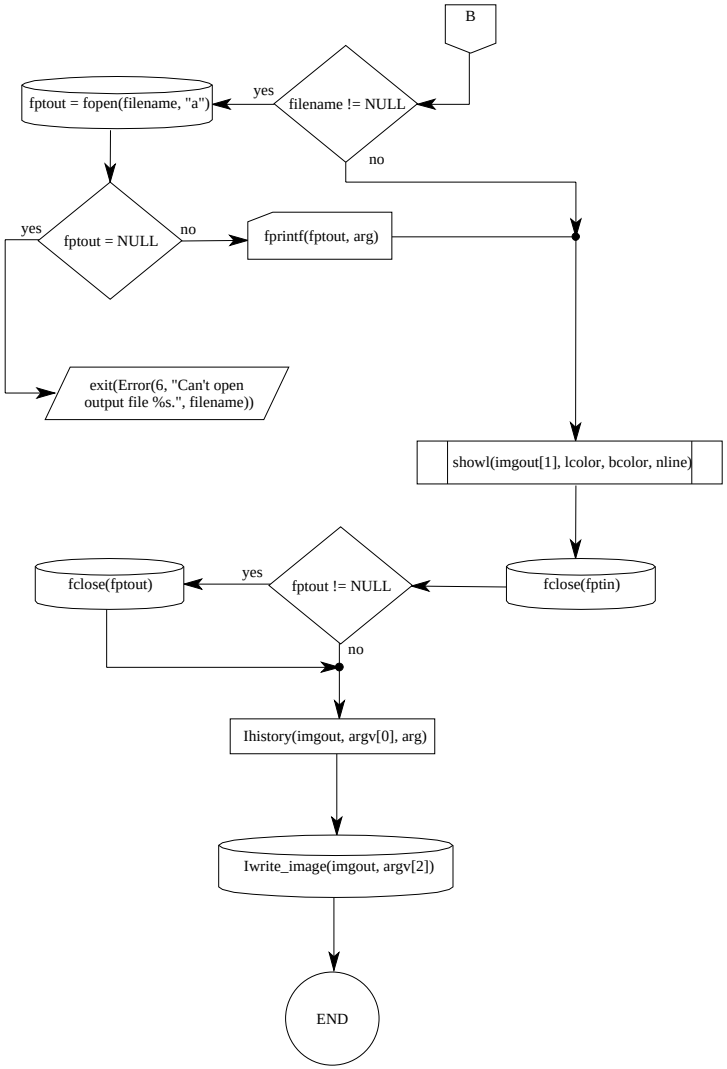
2.2 - Fluxogramas dos diferentes módulos

Apresentam-se, de seguida, os fluxogramas dos módulos que constituem a implementação ***showline***, exceptuando o fluxograma do módulo ***dda***, que foi previamente apresentado na secção anterior.

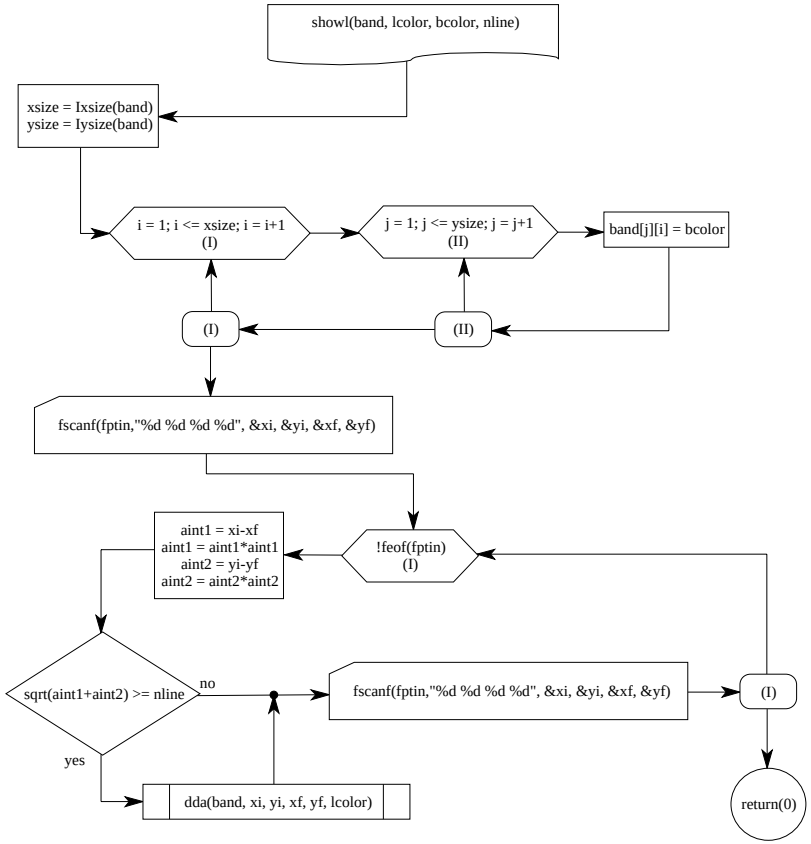
2.2.1 - Módulo principal




A
fptin = fopen(argv[1], "r")
fptin = NULL
no
yes
exit(Error(3, "Can't open input file %s.", argv[1]))
imgout = Init_image(1, title)
imgout = NULL
yes
exit(Error(4, "Can't make output image %s.", argv[2]))
no
imgout[1] = Imake_band(Iu_byte_typ, xsize, ysize)
imgout[1] = NULL
no
yes
exit(Error(5, "Can't make band
of output image %s.", argv[2]))
sprintf(arg, "Results for %s in %s with: Minimum number of pixels
to be considered as a good line: %d.", argv[0], argv[1], nline)
B


B
filename != NULL
yes
fptout = fopen(filename, "a")
no
fptout = NULL
yes
no
fprintf(fptout, arg)
exit(Error(6, "Can't open
output file %s.", filename))
showl(imgout[1], lcolor, bcolor, nline)
fptout != NULL
yes
fclose(fptout)
fclose(fptin)
no
Ihistory(imgout, argv[0], arg)
Iwrite_image(imgout, argv[2])
END

2.2.2 - Módulo showl



2.3 - Listagem da implementação

Apresenta-se, de seguida, a listagem da implementação ***showlines*** desenvolvida em linguagem *C* em ambiente *UNIX* numa estação de trabalho *SunSparc* e que se destina ao ambiente de processamento de imagem *XITE*[2], do qual utiliza algumas funções e tipos de variáveis já previamente definidas. Esta implementação contém comentários para auxílio à sua compreensão, assim como informação sobre os diferentes módulos, para sua posterior extracção através da ferramenta *CDOC(1)* existente no *XITE*.

```
/*
_____________________________________________________________________

  showlines.c
  @(#)lshowlines.c   1.00 94/03/30, Copyright 1994, DEEC, FEUP
  Departamento de Engenharia Electrotecnica e de Computadores
  Faculdade de Engenharia
  Rua dos Bragas
  4099 PORTO CODEX
  PORTUGAL
  E-mail: gpai@obelix.fe.up.pt
_____________________________________________________________________

  biff.h
  @(#)biff.h   1.23 92/04/10, Copyright 1990, Blab, UiO
  Image processing lab, Department of Informatics
  University of Oslo
  E-mail: blab@ifi.uio.no
_____________________________________________________________________

  readarg.h
  @(#)readarg.h   1.3 92/01/15, Copyright 1990, Blab, UiO
  Image processing lab, Department of Informatics
  University of Oslo
  E-mail: blab@ifi.uio.no
_____________________________________________________________________

  message.h
  @(#)message.h   1.3 92/01/15, Copyright 1990, Blab, UiO
  Image processing lab, Department of Informatics
  University of Oslo
  E-mail: blab@ifi.uio.no
_____________________________________________________________________

  Permission to use, copy, modify and distribute this software and its documentation for any purpose
and without fee is hereby granted, provided that this copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in supporting documentation and that the
name of B-lab, Department of Informatics or University of Oslo not be used in advertising or publicity
pertaining to distribution of the software without specific, written prior permission.
```

[2] *X-based Image processing Tools an Environmemt*, [Lønnestad, 1992].

```
  B-LAB DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT
SHALL B-LAB BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL
DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA
OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER
TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
*/

/************************************************************************/

static char *SccsId = "@(#)showlines.c 1.00 94/03/30, DEEC, FEUP";

/************************************************************************/

/* INCLUDES */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <blab/biff.h>
#include <blab/readarg.h>
#include <blab/message.h>

/************************************************************************/

/* GLOBAL VARIABLES */

FILE *fptin, *fptout;       /* file's pointers */

/************************************************************************/

/*F:showl*
_____________________________________________________________________

  showl
_____________________________________________________________________

Name:          showl - drawing input lines in an output band

Syntax:      | int showl(band, lcolor, bcolor, nline)
             | IBAND band;
             | int lcolor, bcolor, nline;

Description:  'showl' performs the drawing of input lines in an output 'band'.
              The output lines will be drawing in 'band' with color 'lcolor' in a
              background 'bcolor'.
              'nline' is the minimum number of line's pixels to a line could be
              considered as a good line.
              '*fptin' it's a global pointer to the file with the initial and final
              coordinates of all the lines to be considered.
              '*fptout' it's a global pointer to the file in which the initial and final
              coordinates of all lines could be writing (if fptout!=NULL).

Return value:  | 0 => Ok.

Example:       | Try yourself.

Restrictions:  None.

Author:        Joao Tavares

Id:            @(#)showl.c   1.00   94/03/11
_____________________________________________________________________
*/

int showl(band, lcolor, bcolor, nline)
IBAND band;
int lcolor, bcolor, nline;

{

 register int i, j;
 int xsize, ysize, xi, yi, xf, yf, aint1, aint2;

 /* make all the pixels in band as bcolor */

 xsize = Ixsize(band);
 ysize = Iysize(band);

 for (i = 1; i <= xsize; ++i) {

  for (j = 1; j <= ysize; ++j) band[j][i] = bcolor;

 }

 /* find good lines in the input file */

 fscanf(fptin, "%d %d %d %d", &xi, &yi, &xf, &yf);
 while (!feof(fptin)) {
```

```
    aint1 = xi-xf;
    aint1 *= aint1;
    aint2 = yi-yf;
    aint2 *= aint2;
    if (sqrt((float)(aint1+aint2)) >= nline) dda(band, xi, yi, xf, yf, lcolor);
    fscanf(fptin, "%d %d %d %d", &xi, &yi, &xf, &yf);

  }

  return(0);

}

/*************************************************************************/

/*F:dda*
_________________________________________________________________________

  dda
_________________________________________________________________________

Name:          dda - drawing a segment of line using the algorithm of Bresenham

Syntax:        | dda(xi, yi, xf, yf, band, color)
               | IBAND band;
               | int xi, yi, xf, yf, color;

Description:   'dda' draw the segment of line between the initial pixel 'xi, yi' and
               the final pixel 'xf, yf' in the band 'band' with color as 'color'.
               If the global pointer 'fptout' is different of NULL the initial and
               final points of all segments will be writing in 'fptout"s file.

Return value:  | 0 => Ok.

Example:       | Try yourself.

Restrictions:  None.

Author:        Joao Tavares

Id:            @(#)dda.c   1.00   93/05/05
_________________________________________________________________________
*/

int dda(band, xi, yi, xf, yf, color)
IBAND band;
int xi, yi, xf, yf, color;

{

 register int t, distance;
 int xsize, ysize, xerr, yerr, delta_x, delta_y, incx, incy;

 /* write in file, if it was desire, the coordinates of the initial and final pixels of the segment */

 if (fptout != NULL) fprintf(fptout, "\t%d %d\t%d %d\n", xi, yi, xf, yf);

 xsize = Ixsize(band);
 ysize = Iysize(band);
 xerr = 0;
 yerr = 0;

 /* calculation of delta x and delta y */

 delta_x = xf-xi;
 delta_y = yf-yi;

 /* determination of increment x and increment y */

 if (delta_x > 0) incx = 1;
 else if (delta_x == 0) incx = 0;
 else incx = -1;
 if (delta_y > 0) incy = 1;
 else if (delta_y == 0) incy = 0;
 else incy = -1;

 /* determination of the distance */

 delta_x = abs(delta_x);
 delta_y = abs(delta_y);

 if (delta_x > delta_y) distance = delta_x;
 else distance = delta_y;

 /* mark the closest point to the line with color */

 for (t = 0; t <= distance; t++) {

   if (xi < 1 || xi > xsize || yi < 1 || yi > ysize) continue;
```

```
   band[yi][xi] = color;
   xerr += delta_x;
   yerr += delta_y;
   if (xerr >= distance) {

     xerr -= distance;
     xi += incx;

   }
   if (yerr >= distance) {

     yerr -= distance;
     yi += incy;

   }

  }

  return(0);

}

/*****************************************************************************/

/*P:showlines*
_____________________________________________________________________________

  showlines
_____________________________________________________________________________

Name:      showlines - drawing input lines in an output image

Syntax:    | [-t <title>] [-l <lineColor>] [-b <backColor>] [-x <xsize>]
           | [-y <ysize>] [-n <nPixGoLine>] [-f <outputFilename] <inputFile>
           | <outputImage>

Description:  'showlines' performs the drawing of input lines in the output image
              'outputImage'.
              The output image will have the 'title' which is indicate through flag
              '-t', by default the 'title' is "Lines in file".
              The output lines will be design in 'outputImage' image with color
              'lineColor' in a 'backColor' background.
              The color of the output lines 'outputImage' image can be indicate
              trough flag '-l'. By default 'lineColor' is 0 (black).
              The color of 'outputImage' image background can be indicate
              trough flag '-b'. By default 'backColor' is 255 (white).
              The size of output image in x direction can be indicate trough flag
              '-x'. By default 'xsize' is 256.
              The size of output image in y direction can be indicate trough flag
              '-y'. By default 'ysize' is 256.
              The minimum number of line's pixels to a line could be considered
              as a good line can be indicate through flag '-n'. By default
              'nPixGoLine' is 20.
              The initial and final point's of each output line can be writing in the
              file 'outputFilename' if the flag '-f' is on, by default this flag is off.
              The input file 'inputFile' have the coordinates of the initial and final
              pixel of all the lines.

Return value:  | 1 => Wrong value for line color %d; it must be greater than or
               |      equal to 0 and smaller than or equal to 255.
               | 2 => Wrong value for background color %d; it must be greater than
               |      or equal to 0 and smaller than or equal to 255.
               | 3 => Can't open input file <inputFile>.
               | 4 => Can't make output image <outputImage>.
               | 5 => Can't make band of output image <outputImage>.
               | 6 => Can't open output file <ouputFilename>.

Examples:      | Try yourself.

Restrictions:  The value for line color and for background color must be greater
               than or equal to 0 and smaller than or equal to 255.

Author:        Joao Tavares

Id:            @(#)showlines.c   1.0   94/03/11
_____________________________________________________________________________
*/

#ifdef MAIN

main(argc, argv)
int argc;
char **argv;

{

  IMAGE imgout;
  int lcolor, bcolor, xsize, ysize, nline;
  char *filename, *title, arg[300];
```

```
  InitMessage(&argc, argv, "Usage: %s\n [-t <title>] [-l <lineColor>] [-b <backColor>] [-x
<xsize>] [-y <ysize>] [-n <nPixGoLine>] [-f <outputFilename] <inputFile> <outputImage>\n");
  Iset_message(1);
  Iset_abort(1);

  /* read switches */

  title = read_switch(&argc, argv, "-t", 1, "Lines in file");
  lcolor = atoi(read_switch(&argc, argv, "-l", 1, "0"));
  bcolor = atoi(read_switch(&argc, argv, "-b", 1, "255"));
  xsize = atoi(read_switch(&argc, argv, "-x", 1, "256"));
  ysize = atoi(read_switch(&argc, argv, "-y", 1, "256"));
  nline = atoi(read_switch(&argc, argv, "-n", 1, "10"));
  filename = read_switch(&argc, argv, "-f", 1, NULL);

  if (argc == 1) Usage(1, (char*)0);
  if (argc != 3) Usage(2, "\nBad number of arguments.\n");
  if (lcolor < 0 || lcolor > 255) exit(Error(1, "\nWrong value for line color %d; it must be greater
than or equal to 0 and smaller than or equal to 255.\n", lcolor));
  if (bcolor < 0 || bcolor > 255) exit(Error(2, "\nWrong value for background color %d; it must be
greater than or equal to 0 and smaller than or equal to 255.\n", bcolor));

  /* open the input file */

  fptin = fopen(argv[1], "r");
  if (fptin == NULL) exit(Error(3, "\nCan't open input file %s.\n", argv[1]));

  /* make the output image */

  imgout = Init_image(1, title);
  if (imgout == NULL) exit(Error(4, "\nCan't make output image %s.\n", argv[2]));

  /* make band of the output image */

  imgout[1] = Imake_band(Iu_byte_typ, xsize, ysize);

  if (imgout[1] == NULL) exit(Error(5, "\nCan't make band of output image %s.\n", argv[2]));

  sprintf(arg, "Results for %s in %s with:\n\n\tMinimum number of pixels to be considered as a line:
%d.\n\n", argv[0], argv[1], nline);

  /* open the output, file if it was desire, and write arg */

  if (filename != NULL) {
    fptout = fopen(filename, "a");
    if (fptout == NULL) exit(Error(6, "\nCan't open output file %s.\n", filename));
    fprintf(fptout, arg);

  }

  /* call the showl function */

  showl(imgout[1], lcolor, bcolor, nline, fptin, fptout);

  /* close the input file */

  fclose(fptin);

  /* close output file if it was desire */

  if (fptout != NULL) fclose(fptout);

  /* write the output image with history */

  Ihistory(imgout, argv[0], arg);
  Iwrite_image(imgout,argv[2]);

}

#endif

/*****************************************************************************/
```

Capítulo V

Seguimento de Segmentos de Recta ao Longo de Sequências de Imagens e Obtenção de Coordenadas 3D

Introdução

Neste capítulo são apresentadas quatro implementações com interesse nos domínios do seguimento de segmentos de recta ao longo de uma sequência de imagem e da obtenção de coordenadas 3D. As quatro implementações apresentadas são:

- ✓ Determinação da área visível por uma dada câmara, sobre um plano de coordenadas especificadas pelo utilizador: ***simspace***.
- ✓ Simulador de uma dada câmara móvel para segmentos de recta no espaço tridimensional, cujas coordenadas 3D dos pontos iniciais e finais são especificadas pelo utilizador: ***simoca***.
- ✓ Seguimento de segmentos de recta ao longo de uma dada sequência de imagens, e posterior obtenção de coordenadas 3D dos pontos inicial e final de cada entidade emparelhada, quando o movimento da respectiva câmara é conhecido: ***deep***.
- ✓ Obtenção de coordenadas 3D de pontos cujas coordenadas 2D, em duas imagens obtidas por uma mesma câmara para duas posições e orientações distintas, são especificadas pelo utilizador: ***compdeep***.

As duas primeiras implementações, ***simspace*** e ***simoca***, têm como principal interesse a simulação de uma câmara móvel de características conhecidas, quando as entidades a considerar são segmentos de recta no espaço tridimensional. Deste modo, torna-se possível obter dados simulados para posterior utilização com a implementação ***deep***.

A quarta implementação, ***compdeeep***, tem como principal interesse confrontar e investigar o processo de obtenção de coordenadas 3D segundo o método da triangulação estereoscópica[1].

Em todas as implementações apresentadas neste capítulo, o modelo utilizado para a câmara foi o apresentado resumidamente no capítulo II, e detalhadamente em [Tavares, 1995].

1 - Determinação da área visível por uma dada câmara sobre um determinado plano: simspace

Nesta secção é apresentada uma implementação do método, descrito em [Tavares, 1995], para a determinação da área visível por uma dada câmara sobre um plano de coordenadas especificadas. Esta implementação determina os quatro vértices da área visível por uma dada câmara sobre um plano de coordenadas especificadas.

1.1 - Descrição dos diferentes módulos

Esta implementação é constituída pelos módulos: ***principal***, ***compute_mr*** e ***frame_world***, *Fig. 1*. Apresenta-se de seguida a descrição de cada um destes módulos.

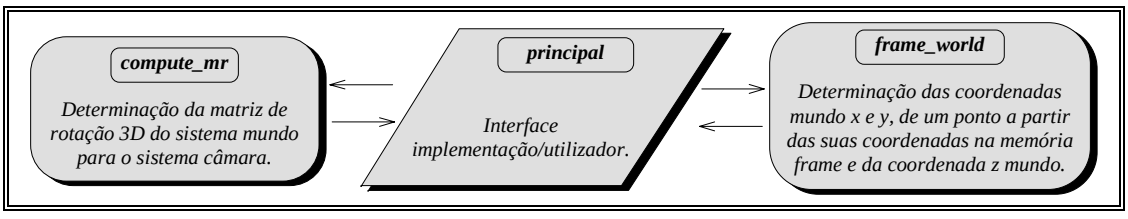


*Fig. 1 - Módulos integrantes da implementação **simspace** e suas relações.*

[1] Ver, por exemplo [Tavares, 1995].

1.1.1 - Módulo principal

Este módulo é responsável pela interface implementação/utilizador. A seu cargo está:

- ✓ Leitura dos argumentos opcionais ("*switches*"):
  - *xcenterbuf*, ("[-cx <xcenterbuf>]"), coordenada do centro da imagem na memória *frame* segundo a direcção *x*, C_x. Por defeito, é igual a metade da dimensão da memória *frame* segundo a mesma direcção.
  - *ycenterbuf*, ("[-cy <ycenterbuf>]"), coordenada do centro da imagem na memória *frame* segundo a direcção *y*, C_y. Por defeito, é igual a metade da dimensão da memória *frame* segundo a mesma direcção.
  - *xsizebuf*, ("[-xs <xsizebuf>]"), dimensão da memória *frame* segundo a direcção *x*. Por defeito, é igual a *512*.
  - *ysizebuf*, ("[-ys <ysizebuf>]"), dimensão da memória *frame* segundo a direcção *y*. Por defeito, é igual a *512*.
  - *rotationx*, ("[-rx <rotationx>]"), valor em grau da rotação, segundo o eixo *x*, da transformação geométrica do sistema de coordenadas mundo para o sistema câmara. Por defeito, é igual a *0.0* grau.
  - *rotationy*, ("[-ry <rotationy>]"), valor em grau da rotação, segundo o eixo *y*, da transformação geométrica do sistema de coordenadas mundo para o sistema câmara. Por defeito, é igual a *0.0* grau.
  - *rotationz*, ("[-rz <rotationz>]"), valor em grau da rotação, segundo o eixo *z*, da transformação geométrica do sistema de coordenadas mundo para o sistema câmara. Por defeito, é igual a *0.0* grau.
  - *translationx*, ("[-tx <translationx>]"), translação, segundo o eixo *x*, da transformação geométrica do sistema de coordenadas mundo para o sistema câmara. Por defeito, é igual a *0.0*.
  - *translationy*, ("[-ty <translationy>]"), translação, segundo o eixo *y*, da transformação geométrica do sistema de coordenadas mundo para o sistema câmara. Por defeito, é igual a *0.0*.
  - *translationz*, ("[-tz <translationz>]"), translação, segundo o eixo *z*, da transformação geométrica do sistema de coordenadas mundo para o sistema câmara. Por defeito, é igual a *0.0*.
- ✓ Leitura dos argumentos passados à implementação pela linha de comando:
  - *f*, ("<f>"), distância focal efectiva *f*.
  - *dx*, ("<dx>"), distância entre centros dos elementos sensores vizinhos na direcção *x*, d_x.
  - *dy*, ("<dy>"), distância entre centros dos sensores CCD vizinhos na direcção *y*, d_y.
  - *sx*, ("<sx>"), factor de incerteza horizontal s_x.
  - *k*, ("<k>"), factor de distorção radial da lente k_1.
  - *zwplane*, ("<zwplane>"), coordenada 3D mundo *z* para o plano sobre o qual se pretende determinar a área visível pela câmara.
- ✓ Chamada do módulo ***compute_mr***.
- ✓ Apresentação das componentes determinadas da matriz de rotação 3D da transformação geométrica do sistema de coordenadas mundo para o sistema câmara.

- ✓ Chamada do módulo ***frame_world***.
- ✓ Apresentação das coordenadas dos quatro vértices que definem a área visível pela câmara sobre o plano especificado.
- ✓ Comunicação da existência de erros em argumentos de entrada ou ao longo do decurso da execução:
  - Através de valores de retorno:
    - 1, ("The value for sx must be different from zero."), o valor especificado para o factor de incerteza horizontal deve ser diferente de zero.
    - 2, ("The value for dx must be greater than zero."), o valor especificado para a distância entre centros dos elementos sensores vizinhos na direcção *x*, deve ser maior do que zero.
    - 3, ("The value for dy must be greater than zero."), o valor especificado para a distância entre centros dos sensores CCD vizinhos na direcção *y*, deve ser maior do que zero.
    - 4, ("Wrong value for rotation about x <rotationx>; it must be greater than -360 and smaller than 360 degree."), o valor especificado para a rotação segundo o eixo *x* não está compreendido entre -*360* e *360* grau.
    - 5, ("Wrong value for rotation about y <rotationy>; it must be greater than -360 and smaller than 360 degree."), o valor especificado para a rotação segundo o eixo *y* não está compreendido entre -*360* e *360* grau.
    - 6, ("Wrong value for rotation about z <rotationz>; it must be greater than -360 and smaller than 360 degree."), o valor especificado para a rotação segundo o eixo *z* não está compreendido entre -*360* e *360* grau.
  - Através de mensagens:
    - 1, (*InitMessage*), quando o número de argumentos é igual a um, isto é, apenas o nome da implementação.
    - 2, ("Bad number of arguments."), quando o número de argumentos é diferente de sete.

Este módulo apresenta como restrições quanto à sua utilização:

- ⊠ O valor especificado para o factor de incerteza horizontal deve ser diferente de zero.
- ⊠ Os valores especificados para a distância entre centros dos elementos sensores vizinhos na direcção *x* e para a distância entre centros dos sensores CCD vizinhos na direcção *y*, devem ser maiores do que zero.
- ⊠ Os valores especificados para as rotações segundo os três eixos principais: *rotationx*, *rotationy* e *rotationz*, deverão estar definidos em grau e compreendidos entre -*360* e *360*.

1.1.2 - Módulo compute_mr

Este módulo é responsável pela determinação dos elementos da matriz de rotação 3D da transformação geométrica do sistema de coordenadas mundo para o sistema câmara.

Como parâmetros de entrada para este módulo têm-se:

- ➲ *mr[ ][3]*, do tipo real, matriz de rotação 3D pretendida.
- ➲ *rot[ ]*, do tipo real, vector que contém os valores das rotações segundo os três eixos principais.
- ➲ *c*, do tipo inteiro; quando igual a *1*, os valores das rotações segundo os três eixos principais são convertidos de grau para radiano.

Este módulo retorna *0* como indicação que a sua execução decorreu em normalidade, e não apresenta qualquer tipo de restrições quanto à sua utilização.

1.1.3 - Módulo frame_world

Este módulo é responsável pela determinação das coordenadas 3D mundo *x* e *y* de um ponto, a partir das suas coordenadas na memória *frame* e da sua coordenada *z* no mesmo sistema mundo.

Como parâmetros de entrada para este módulo têm-se:

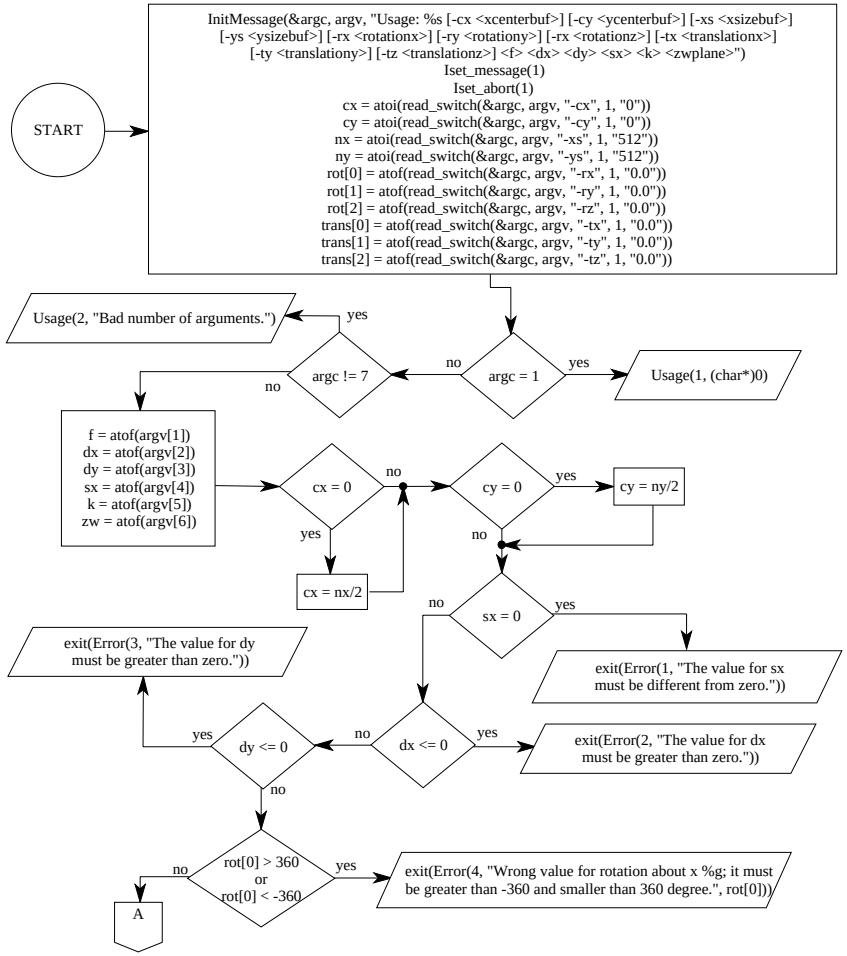
- *mr[ ][3]*, do tipo real, matriz de rotação 3D do sistema de coordenadas mundo para o sistema câmara.
- *trans[ ]*, do tipo real, vector de translação 3D do sistema de coordenadas mundo para o sistema câmara.
- *xf*, do tipo inteiro, coordenada *x* no sistema 2D da memória *frame* do respectivo ponto.
- *yf*, do tipo inteiro, coordenada *y* no sistema 2D da memória *frame* do respectivo ponto.
- *xw*, do tipo real, passado por endereço, coordenada *x* no sistema 3D mundo obtida pelo módulo para o respectivo ponto.
- *yw*, do tipo real, passado por endereço, coordenada *y* no sistema 3D mundo obtida pelo módulo para o respectivo ponto.
- *zw*, do tipo real, coordenada *z* no sistema 3D mundo do respectivo ponto.
- *dxl*, do tipo real, igual a $s_x d_x$ onde s_x é o factor de incerteza horizontal e d_x é a distância entre centros dos elementos sensores vizinhos na direcção *x*.
- *dy*, do tipo real, distância entre centros dos sensores CCD vizinhos na direcção *y*.
- *cx*, do tipo inteiro, coordenada do centro da imagem na memória *frame* segundo a direcção *x*.
- *cy*, do tipo inteiro, coordenada do centro da imagem na memória *frame* segundo a direcção *y*.
- *f*, do tipo real, distância focal efectiva.
- *k*, do tipo real, factor de distorção radial da lente.

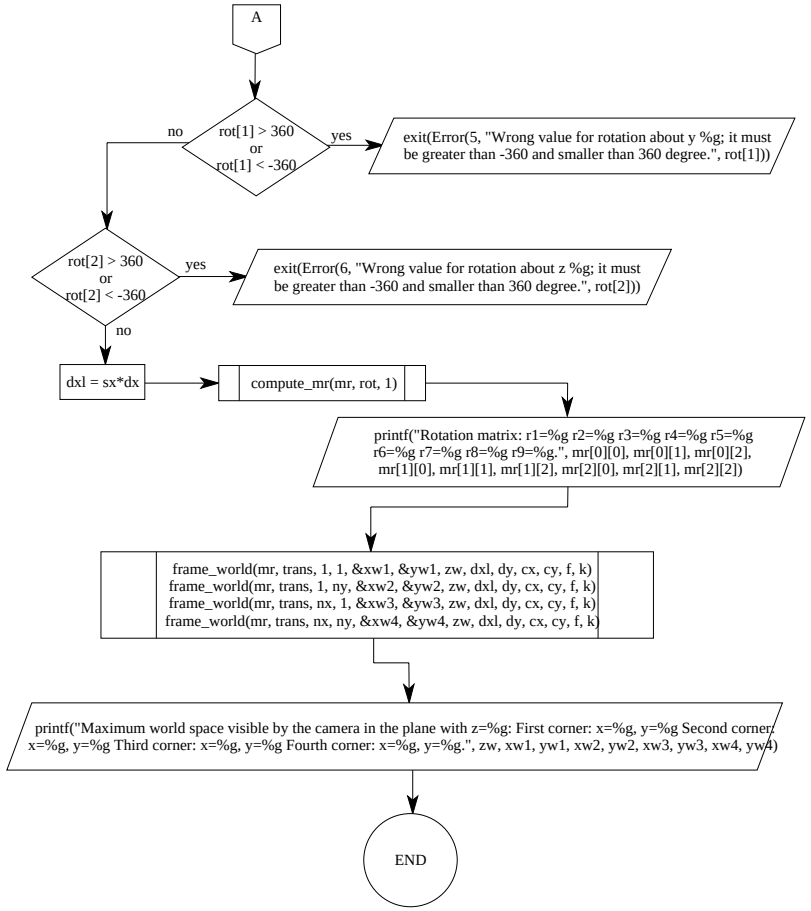
Este módulo retorna *0* como indicação que a sua execução decorreu em normalidade, e não apresenta qualquer tipo de restrições quanto à sua utilização.

1.2 - Fluxogramas dos diferentes módulos

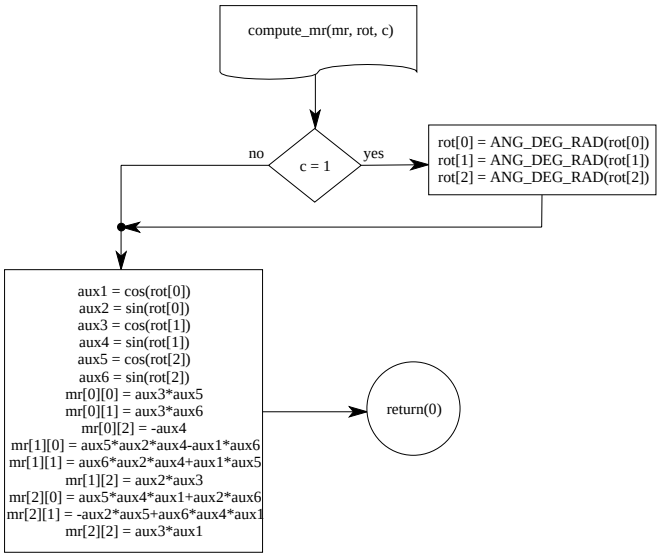
Apresentam-se, de seguida, os fluxogramas dos módulos que constituem a implementação ***simspace***.

1.2.1 - Módulo principal

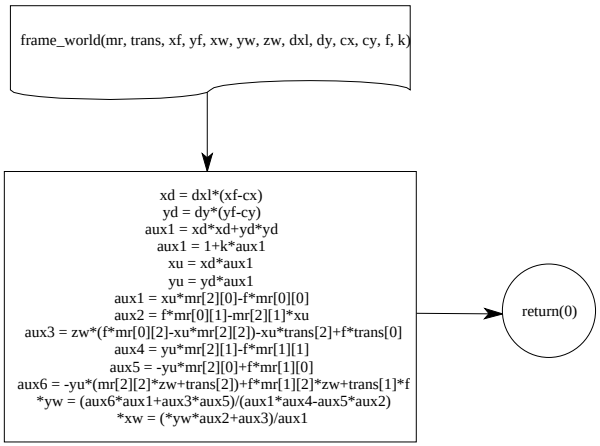




1.2.2 - Módulo compute_mr



1.2.3 - Módulo frame_world



1.3 - Listagem da implementação

Apresenta-se, de seguida, a listagem da implementação ***simspace*** desenvolvida em linguagem *C* em ambiente *UNIX* numa estação de trabalho *SunSparc* e que se destina ao ambiente de processamento de imagem *XITE*[2], do qual utiliza algumas funções e tipos de variáveis já previamente definidas. Esta implementação contém comentários para auxílio à sua compreensão, assim como informação sobre os diferentes módulos, para sua posterior extracção através da ferramenta *CDOC(1)* existente no *XITE*.

```
/*
_____________________________________________________________________


simspace.c
@(#)simspace.c   1.00 94/06/19, Copyright 1994, DEEC, FEUP
Departamento de Engenharia Electrotecnica e de Computadores
Faculdade de Engenharia
4099 PORTO CODEX
PORTUGAL
E-mail: gpai@obelix.fe.up.pt
_____________________________________________________________________


readarg.h
@(#)readarg.h   1.3 92/01/15, Copyright 1990, Blab, UiO
Image processing lab, Department of Informatics
University of Oslo
E-mail: blab@ifi.uio.no
_____________________________________________________________________


message.h
@(#)message.h   1.3 92/01/15, Copyright 1990, Blab, UiO
Image processing lab, Department of Informatics
University of Oslo
E-mail: blab@ifi.uio.no
_____________________________________________________________________


  Permission to use, copy, modify and distribute this software and its documentation for any purpose
and without fee is hereby granted, provided that this copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in supporting documentation and that the
```

[2] *X-based Image processing Tools an Environmemt*, [Lønnestad, 1992].

```
name of B-lab, Department of Informatics or University of Oslo not be used in advertising or publicity
pertaining to distribution of the software without specific, written prior permission.

  B-LAB DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT
SHALL B-LAB BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL
DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA
OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER
TORTIOUS ACTION, ARISING OUT OF OR IN  CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
*/
/*****************************************************************************/

/* INCLUDES */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <blab/readarg.h>
#include <blab/message.h>
#include "gdefs.h"

/*****************************************************************************/

/*F:compute_mr*
_____________________________________________________________________________

  compute_mr
_____________________________________________________________________________

Name:       compute_mr - compute the rotation matrix 3D from object world
                         coordinate system to the camera coordinate system

Syntax:     | int compute_mr(mr, rot, c)
            | int c;
            | double mr[][3], rot[];

Description:  'compute_mr' performs the determination of the rotation matrix 3D
              from object world coordinate system to the camera coordinate
              system.
              'mr[][3]' is the rotation matrix 3D from object world coordinate
              system to the camera coordinate system.
              'rot[]' is the angles of the rotation 3D from object world coordinate
              system to the camera coordinate system.
              If 'c' is one the angles are convert to rad.

Return value:  | 0 => Ok.

Example:       | Try yourself.

Restrictions:  None.

Author:        Joao Tavares

Id:            @(#)compute_mr.c   1.00   94/06/10
_____________________________________________________________________________
*/

int compute_mr(mr, rot, c)
int c;
double mr[][3], rot[];

{

 double aux1, aux2, aux3, aux4, aux5, aux6;

 if (c == 1) {

  /* covertion deg/rad */

  rot[0] = ANG_DEG_RAD(rot[0]);
  rot[1] = ANG_DEG_RAD(rot[1]);
  rot[2] = ANG_DEG_RAD(rot[2]);

 }

 /* compute the rotation matrix */

 aux1 = cos(rot[0]);
 aux2 = sin(rot[0]);
 aux3 = cos(rot[1]);
 aux4 = sin(rot[1]);
 aux5 = cos(rot[2]);
 aux6 = sin(rot[2]);

 mr[0][0] = aux3*aux5;
 mr[0][1] = aux3*aux6;
 mr[0][2] = -aux4;
 mr[1][0] = aux5*aux2*aux4-aux1*aux6;
 mr[1][1] = aux6*aux2*aux4+aux1*aux5;
```

```
  mr[1][2] = aux2*aux3;
  mr[2][0] = aux5*aux4*aux1+aux2*aux6;
  mr[2][1] = -aux2*aux5+aux6*aux4*aux1;
  mr[2][2] = aux3*aux1;

  return(0);

}

/*************************************************************************/

/*F:frame_world*
_______________________________________________________________________

  frame_world
_______________________________________________________________________

Name:     frame_world - compute the world's coordinates of a point from its
                        frame buffer's coordinates and z world coordinate

Syntax:   | int frame_world(mr, trans, xf, yf, xw, yw, zw, dxl, dy, cx, cy, f, k)
          | double mr[][3], trans[], *xw, *yw, zw, dxl, dy, f, k;
          | int xf, yf, cx, cy;

Description:  'frame_world' performs the determination of the world coordinates
              of a point from its frame buffer's coordinates and z world coordinate.
              'mr[][3]' is the rotation matrix 3D from object world coordinate
              system to the camera coordinate system.
              'trans[]' is the translation vector 3D from object world coordinate
              system to the camera coordinate system.
              'xf' and 'yf' are the frame buffer's coordinates.
              'xw' and 'yw' passed by reference are the world's coordinates.
              'zw' is the z world's coordinate.
              'dxl' is (sx*dx) where dx is the distance between adjacent sensor
              elements in x (scan line) direction and sx is the horizontal
              uncertainty factor.
              'dy' is the center to center distance between adjacent CCD sensor
              in the y direction.
              'cx' and 'cy' are the row and column numbers of the center of frame
              buffer.
              'f' is the effective focal length.
              'k' is the lens's radial distortion.

Return value: | 0 => Ok.

Example:      | Try yourself.

Restrictions: None.

Author:       Joao Tavares

Id:           @(#)frame_world.c   1.00   94/06/10
_______________________________________________________________________

*/

int frame_world(mr, trans, xf, yf, xw, yw, zw, dxl, dy, cx, cy, f, k)
double mr[][3], trans[], *xw, *yw, zw, dxl, dy, f, k;
int xf, yf, cx, cy;

{

  double xd, yd, xu, yu, aux1, aux2, aux3, aux4, aux5, aux6;

  xd = dxl*(xf-cx);
  yd = dy*(yf-cy);
  aux1 = xd*xd+yd*yd;
  aux1 = 1.0+k*aux1;
  xu = xd*aux1;
  yu = yd*aux1;
  aux1 = xu*mr[2][0]-f*mr[0][0];
  aux2 = f*mr[0][1]-mr[2][1]*xu;
  aux3 = zw*(f*mr[0][2]-xu*mr[2][2])-xu*trans[2]+f*trans[0];
  aux4 = yu*mr[2][1]-f*mr[1][1];
  aux5 = -yu*mr[2][0]+f*mr[1][0];
  aux6 = -yu*(mr[2][2]*zw+trans[2])+f*mr[1][2]*zw+trans[1]*f;
  *yw = (aux6*aux1+aux3*aux5)/(aux1*aux4-aux5*aux2);
  *xw = (*yw*aux2+aux3)/aux1;

  return(0);

}

/*************************************************************************/

/*P:simspace*
_______________________________________________________________________

  simspace
_______________________________________________________________________
```

Name: simspace - determination of the area in a plane visible by a camera

Syntax: | simspace [-cx <xcenterbuf>] [-cy <ycenterbuf>] [-xs <xsizebuf>]
| [-ys <ysizebuf>] [-rx <rotationx>] [-ry <rotationy>] [-rx <rotationz>]
| [-tx <translationx>] [-ty <translationy>] [-tz <translationz>] <f>
| <dx> <dy> <sx> <k> <zwplane>

Description: 'simspace' performs the determination of the area in a plane visible by a camera using the TSAI's model.
The center of frame buffer in direction x is indicate through flag '-cx', by default 'xcenterbuf' is (xsizebuf/2).
The center of frame buffer in direction y is indicate through flag '-cy', by default 'ycenterbuf' is (ysizebuf/2).
The frame buffer's dimensions in directions x and y are indicate trough flags '-xs' and '-ys'. By default the frame buffer's dimensions are 512 by 512.
The angles (in degree) of rotation 3D from object world coordinate system to the camera coordinate system about the x, y and z axis, are indicate through the flags '-rx', '-ry' and '-rz'. By default they are all zero.
The translation 3D from object world coordinate system to the camera coordinate system along the x, y and z axis, are indicate through the flags '-tx', '-ty' and '-tz'. By default they are all zero.
'f' is the effective focal length.
'dx' is the center to center distance between adjacent sensor elements in x (scanline) direction.
'dy' is the center to center distance between adjacent CCD sensor in the y direction.
'sx' is the horizontal uncertainty factor.
'k' is the lens's radial distortion.
'zwplane' is the world 3D coordinate z of the plane.

Return value: | 1 => The value for sx must be different from zero.
| 2 => The value for dx must be greater than zero.
| 3 => The value for dy must be greater than zero.
| 4 => Wrong value for rotation about x <rotationx>; it must be
| greater than -360.0 and smaller than 360.0 degree.
| 5 => Wrong value for rotation about y <rotationy>; it must be
| greater than -360.0 and smaller than 360.0 degree.
| 6 => Wrong value for rotation about z <rotationz>; it must be
| greater than -360.0 and smaller than 360.0 degree.

Examples: | Try yourself.

Restrictions: The value for sx must be different from zero.
The values for dx and dy must be greater than zero.
The values for <rotationx>, <rotationy> and <rotationz>, must be smaller than 360 and greater than -360.

Author: Joao Tavares

Id: @(#)simspace.c 1.0 94/06/10

---

*/

```
#ifdef MAIN

static char *SccsId = "@(#)simspace.c 1.00 94/06/10, DEEC, FEUP";

void main(argc, argv)
int argc;
char **argv;

{

  int nx, ny, cx, cy;
  double zw, mr[3][3], rot[3], trans[3], f, dx, dy, dxl, sx, k, xw1, xw2, xw3, xw4, yw1, yw2, yw3,
yw4;
  int compute_mr();
  int frame_world();

  InitMessage(&argc, argv, "Usage: %s\n [-cx <xcenterbuf>] [-cy <ycenterbuf>] [-xs <xsizebuf>] [-
ys <ysizebuf>] [-rx <rotationx>] [-ry <rotationy>] [-rx <rotationz>] [-tx <translationx>] [-ty
<translationy>] [-tz <translationz>] <f> <dx> <dy> <sx> <k> <zwplane>\n");
  Iset_message(1);
  Iset_abort(1);

  /* read switches */

  cx = atoi(read_switch(&argc, argv, "-cx", 1, "0"));
  cy = atoi(read_switch(&argc, argv, "-cy", 1, "0"));
  nx = atoi(read_switch(&argc, argv, "-xs", 1, "512"));
  ny = atoi(read_switch(&argc, argv, "-ys", 1, "512"));
  rot[0] = atof(read_switch(&argc, argv, "-rx", 1, "0.0"));
  rot[1] = atof(read_switch(&argc, argv, "-ry", 1, "0.0"));
  rot[2] = atof(read_switch(&argc, argv, "-rz", 1, "0.0"));
  trans[0] = atof(read_switch(&argc, argv, "-tx", 1, "0.0"));
  trans[1] = atof(read_switch(&argc, argv, "-ty", 1, "0.0"));
  trans[2] = atof(read_switch(&argc, argv, "-tz", 1, "0.0"));
```

```
  if (argc == 1) Usage(1, (char*)0);
  if (argc != 7) Usage(2, "\nBad number of arguments.\n");

  /* read argv[] */

  f = atof(argv[1]);
  dx = atof(argv[2]);
  dy = atof(argv[3]);
  sx = atof(argv[4]);
  k = atof(argv[5]);
  zw = atof(argv[6]);

  if (cx == 0) cx = nx/2;
  if (cy == 0) cy = ny/2;

  if ( sx == 0.0) exit(Error(1, "\nThe value for sx must be different from zero.\n"));
  if ( dx <= 0.0) exit(Error(2, "\nThe value for dx must be greater than zero.\n"));
  if ( dy <= 0.0) exit(Error(3, "\nThe value for dy must be greater than zero.\n"));

  if (rot[0] > 360.0 || rot[0] < -360.0) exit(Error(4, "\nWrong value for rotation about x %g; it must
be greater than -360 and smaller than 360 degree.\n", rot[0]));
  if (rot[1] > 360.0 || rot[1] < -360.0) exit(Error(5, "\nWrong value for rotation about y %g; it must
be greater than -360 and smaller than 360 degree.\n", rot[1]));
  if (rot[2] > 360.0 || rot[2] < -360.0) exit(Error(6, "\nWrong value for rotation about z %g; it must
be greater than -360 and smaller than 360 degree.\n", rot[2]));

  dxl = sx*dx;

  /* compute the matrix R */

  compute_mr(mr, rot, 1);

  printf("\nRotation matrix:\n\n\tr1=%g\tr2=%g\tr3=%g\n\tr4=%g\tr5=%g\tr6=%g\n\tr7=
%g\tr8=%g\tr9=%g.\n", mr[0][0], mr[0][1], mr[0][2], mr[1][0], mr[1][1], mr[1][2], mr[2][0],
mr[2][1], mr[2][2]);

  /* compute xw1, yw1, xw2, yw2, xw3, yw3, xw4, yw4 */

  frame_world(mr, trans, 1, 1, &xw1, &yw1, zw, dxl, dy, cx, cy, f, k);
  frame_world(mr, trans, 1, ny, &xw2, &yw2, zw, dxl, dy, cx, cy, f, k);
  frame_world(mr, trans, nx, 1, &xw3, &yw3, zw, dxl, dy, cx, cy, f, k);
  frame_world(mr, trans, nx, ny, &xw4, &yw4, zw, dxl, dy, cx, cy, f, k);

  printf("\nMaximum world space visible by the camera in the plane with z=%g:\n\n\tFirst corner:
x=%g, y=%g\n\tSecond corner: x=%g, y=%g\n\tThird corner: x=%g, y=%g\n\tFourth corner:
x=%g, y=%g.\n\n", zw, xw1, yw1, xw2, yw2, xw3, yw3, xw4, yw4);

}

#endif

/*****************************************************************************/
```

2 - Simulador de uma dada câmara móvel: simoca

Nesta secção é apresentada uma implementação do método, descrito em [Tavares, 1995], para simulação de uma dada câmara móvel para segmentos de recta no espaço tridimensional, cujas coordenadas 3D dos pontos iniciais e finais são especificadas pelo utilizador. Com esta implementação torna-se possível obter as coordenadas imagem dos pontos inicial e final de cada segmento de recta no espaço tridimensional, para uma dada câmara e para sucessivas posições e orientações. Internamente, esta implementação inclui um algoritmo para a determinação das partes visíveis de cada segmento de recta pela câmara, para cada uma das suas posições e orientações.

2.1 - Descrição dos diferentes módulos

Esta implementação é constituída pelos módulos: ***principal***, ***world_frame***, ***compute_mr***, ***clip2d***, ***clipt***, ***double_int*** e ***simulation***, *Fig. 2*. Apresenta-se de seguida a descrição de cada um destes módulos.

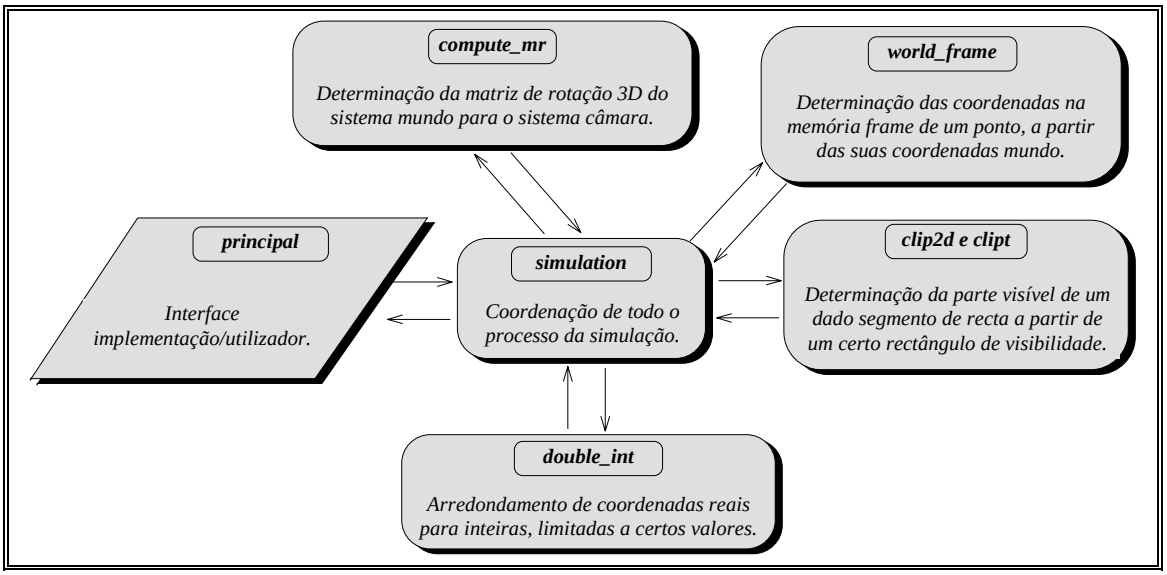


*Fig. 2 - Módulos integrantes da implementação **simoca** e suas relações.*

2.1.1 - Módulo principal

Este módulo é responsável pela interface implementação/utilizador. A seu cargo está:

- ✓ Leitura dos argumentos opcionais ("*switches*"):
  - *xcenterbuf*, ("[-cx <xcenterbuf>]"), coordenada do centro da imagem na memória *frame* segundo a direcção *x*, C_x. Por defeito, é igual a metade da dimensão da memória *frame* segundo a mesma direcção.
  - *ycenterbuf*, ("[-cy <ycenterbuf>]"), coordenada do centro da imagem na memória *frame* segundo a direcção *y*, C_y. Por defeito, é igual a metade da dimensão da memória *frame* segundo a mesma direcção.
  - *xsizebuf*, ("[-xs <xsizebuf>]"), dimensão da memória *frame* segundo a direcção *x*. Por defeito, é igual a *512*.
  - *ysizebuf*, ("[-ys <ysizebuf>]"), dimensão da memória *frame* segundo a direcção *y*. Por defeito, é igual a *512*.

- *verbose*, ("[-v <verbose>]"); quando é especificado como igual a *1*, durante a simulação o utilizador é informado dos resultados obtidos. Por defeito, é igual a *0*.

✓ Leitura dos argumentos passados à implementação pela linha de comando:

- *f*, ("<f>"), distância focal efectiva *f*.
- *dx*, ("<dx>"), distância entre centros dos elementos sensores vizinhos na direcção *x*, d_x.
- *dy*, ("<dy>"), distância entre centros dos sensores CCD vizinhos na direcção *y*, d_y.
- *sx*, ("<sx>"), factor de incerteza horizontal s_x.
- *k*, ("<k>"), factor de distorção radial da lente k_1.
- *filepoints*, ("<filepoints>"), nome do ficheiro de texto que contém as coordenadas 3D dos pontos inicial e final de cada segmento de recta a considerar para a simulação.
- *filesimulation*, ("<filesimulation>"), nome do ficheiro de texto que contém a posição e a orientação do referencial câmara relativamente ao referencial mundo e o nome do ficheiro de texto para escrita dos resultados obtidos, para cada posição e orientação da câmara a considerar.

✓ Abertura dos ficheiros *filepoints* e *filesimulation*.

✓ Chamada do módulo ***simulation***.

✓ Fecho dos ficheiros *filepoints* e *filesimulation*.

✓ Comunicação da existência de erros em argumentos de entrada ou ao longo do decurso da execução:

- Através de valores de retorno:
  - 1, ("The value for sx must be different from zero."), o valor especificado para o factor de incerteza horizontal deve ser diferente de zero.
  - 2, ("The value for dx must be greater than zero."), o valor especificado para a distância entre centros dos elementos sensores vizinhos na direcção *x* deve ser maior do que zero.
  - 3, ("The value for dy must be greater than zero."), o valor especificado para a distância entre centros dos sensores CCD vizinhos na direcção *y* deve ser maior do que zero.
  - 4, ("Can't open input point's file <filepoints>."), o módulo não consegue abrir o ficheiro especificado.
  - 5, ("Can't open input simulation's file <filesimulation>."), o módulo não consegue abrir o ficheiro especificado.
- Através de mensagens:
  - 1, (*InitMessage*), quando o número de argumentos é igual a um, isto é, apenas o nome da implementação.
  - 2, ("Bad number of arguments."), quando o número de argumentos é diferente de oito.

O ficheiro de entrada *filepoints* contém as coordenadas 3D dos pontos inicial e final de cada segmento de recta a considerar e deverá ter a forma: $x_{w_k}{}^i\ y_{w_k}{}^i\ z_{w_k}{}^i\ x_{w_k}{}^f\ y_{w_k}{}^f\ z_{w_k}{}^f\ x_{w_j}{}^i\ y_{w_j}{}^i\ z_{w_j}{}^i x_{w_j}{}^f\ y_{w_j}{}^f\ z_{w_j}{}^f$ ....

O ficheiro de entrada *filesimulation* deverá conter:

- as três rotações (em grau) e as três translações da transformação geométrica do sistema de coordenadas mundo para o sistema câmara;
- o nome do ficheiro de texto para escrita dos resultados correspondentes a cada posição e orientação da câmara a considerar.

A forma deste ficheiro de entrada deverá ser: $R^i_{(x,\phi)}\ R^i_{(y,\theta)}\ R^i_{(z,\psi)}\ t_x^i\ t_y^i\ t_z^i$ *ficheiro*^{i} $R^j_{(x,\phi)}\ R^j_{(y,\theta)}\ R^j_{(z,\psi)}\ t_x^j\ t_y^j\ t_z^j$ *ficheiro*^{j}....

A forma do ficheiro *ficheiro*^{i} de saída dos resultados respectivos é: $x_f^i\ y_f^i\ x_f^f\ y_f^f$....

Como restrições na utilização deste módulo têm-se:

- O valor especificado para o factor de incerteza horizontal deve ser diferente de zero.
- Os valores especificados para a distância entre centros dos elementos sensores vizinhos na direcção *x* e para a distância entre centros dos sensores CCD vizinhos na direcção *y*, devem ser maiores do que zero.

2.1.2 - Módulo world_frame

Este módulo é responsável pela determinação das coordenadas 2D na memória *frame* de um ponto, a partir das suas coordenadas 3D mundo.

Como parâmetros de entrada para este módulo têm-se:

- *mr[ ][3]*, do tipo real, matriz de rotação 3D do sistema de coordenadas mundo para o sistema câmara.
- *trans[ ]*, do tipo real, vector de translação 3D do sistema de coordenadas mundo para o sistema câmara.
- *world[ ]*, do tipo real, vector das coordenadas 3D do respectivo ponto.
- *frame[ ]*, do tipo real, vector que conterá as coordenadas 2D na memória *frame* do respectivo ponto.
- *f*, do tipo real, distância focal efectiva.
- *k*, do tipo real, factor de distorção radial da lente.
- *dxl*, do tipo real, igual a $s_x d_x$ onde s_x é o factor de incerteza horizontal e d_x é a distância entre centros dos elementos sensores vizinhos na direcção *x*.
- *dy*, do tipo real, distância entre centros dos sensores CCD vizinhos na direcção *y*.
- *cx*, do tipo inteiro, coordenada do centro da imagem na memória *frame* segundo a direcção *x*.
- *cy*, do tipo inteiro, coordenada do centro da imagem na memória *frame* segundo a direcção *y*.

Este módulo retorna *0* como indicação de que a sua execução decorreu em normalidade, e apresenta como restrições quanto à sua utilização: a distância entre centros dos elementos sensores vizinhos na direcção *x* e a distância entre centros dos sensores CCD vizinhos na direcção *y*, deverão ser diferentes de zero.

2.1.3 - Módulo compute_mr

Este módulo é responsável pela determinação dos elementos da matriz de rotação 3D da transformação geométrica do sistema de coordenadas mundo para o sistema câmara.

Como parâmetros de entrada para este módulo têm-se:

- *mr[ ][3]*, do tipo real, matriz de rotação 3D pretendida.
- *rot[ ]*, do tipo real, vector que contém os valores das rotações segundo os três eixos principais da transformação geométrica do sistema de coordenadas mundo para o sistema câmara.
- *c*, do tipo inteiro; quando igual a *1*, os valores das rotações segundo os três eixos principais são convertidos de grau para radiano.

Este módulo retorna *0* como indicação que a sua execução decorreu em normalidade, e não apresenta qualquer tipo de restrições quanto à sua utilização.

2.1.4 - Módulo clip2d

Este módulo é responsável pela determinação das coordenadas 2D dos pontos iniciais e finais da parte visível de um segmento de recta, segundo o algoritmo paramétrico de Liang-Barsky[3]. Este algoritmo foi o adoptado devido a sua elevada eficiência.

Como parâmetros de entrada para este módulo têm-se:

- *x0*, do tipo real, passado por endereço, coordenada do ponto inicial do segmento de recta segundo a direcção *x*; após a execução da implementação conterá a coordenada do ponto inicial da parte visível segundo a mesma direcção.
- *y0*, do tipo real, passado por endereço, coordenada do ponto inicial do segmento de recta segundo a direcção *y*; após a execução da implementação conterá a coordenada do ponto inicial da parte visível segundo a mesma direcção.
- *x1*, do tipo real, passado por endereço, coordenada do ponto final do segmento de recta segundo a direcção *x*; após a execução da implementação conterá a coordenada do ponto final da parte visível segundo a mesma direcção.
- *y1*, do tipo real, passado por endereço, coordenada do ponto final do segmento de recta segundo a direcção *y*; após a execução da implementação conterá a coordenada do ponto final da parte visível segundo a mesma direcção.
- *xmin*, do tipo real, valor mínimo possível para as coordenadas visíveis segundo a direcção *x*.
- *xmax*, do tipo real, valor máximo possível para as coordenadas visíveis segundo a direcção *x*.
- *ymin*, do tipo real, valor mínimo possível para as coordenadas visíveis segundo a direcção *y*.
- *ymax*, do tipo real, valor máximo possível para as coordenadas visíveis segundo a direcção *y*.

Este módulo retorna:

- 0, como indicação de que o segmento de recta não é visível.
- 1, como indicação de que o segmento de recta é visível, sendo as coordenadas dos pontos inicial e final da parte visível: *x0*, *y0* e *x1*, *y1*.

Este módulo, durante a sua execução, utiliza o módulo ***clipt*** e não apresenta qualquer tipo de restrições quanto à sua utilização.

2.1.5 - Módulo clipt

Este módulo é responsável pela determinação dos parâmetros *tL* ou *tE* para a intersecção de um segmento de recta com um segmento do rectângulo de visibilidade, segundo o algoritmo paramétrico de Liang-Barsky.

Como parâmetros de entrada para este módulo têm-se:

- *denom*, do tipo real, corresponde a *denom* no algoritmo paramétrico de Liang-Barsky.
- *num*, do tipo real, corresponde a *num* no algoritmo paramétrico de Liang-Barsky.
- *tl*, do tipo real, passado por endereço, corresponde a *tL* no algoritmo paramétrico de Liang-Barsky.

[3] Ver, por exemplo [Foley, 1991].

- *te*, do tipo real, passado por endereço, corresponde a *tE* no algoritmo paramétrico de Liang-Barsky.

Este módulo retorna:

- 0, como indicação de que o segmento de recta não é interior ao respectivo segmento de recta do rectângulo de visibilidade.
- 1, como indicação de que o segmento de recta é interior ao respectivo segmento de recta do rectângulo de visibilidade.

Este módulo não apresenta qualquer tipo de restrições quanto à sua utilização.

2.1.6 - Módulo double_int

Este módulo é responsável pela conversão de coordenadas reais para inteiras de um dado ponto, estando estas coordenadas inteiras limitadas a um dado intervalo.

Como parâmetros de entrada para este módulo têm-se:

- *x*, do tipo real, passado por endereço, coordenada do ponto segundo a direcção *x*; após a execução da implementação conterá o respectivo valor após conversão.
- *y*, do tipo real, passado por endereço, coordenada do ponto segundo a direcção *y*; após a execução da implementação conterá o respectivo valor após conversão.
- *xmin*, do tipo inteiro, valor mínimo possível para a coordenada inteira segundo a direcção *x*.
- *xmax*, do tipo inteiro, valor máximo possível para a coordenada inteira segundo a direcção *x*.
- *ymin*, do tipo inteiro, valor mínimo possível para a coordenada inteira segundo a direcção *y*.
- *ymax*, do tipo inteiro, valor máximo possível para a coordenada inteira segundo a direcção *y*.

Este módulo retorna *0* como indicação de que a execução decorreu com normalidade, e não apresenta qualquer tipo de restrições quanto à sua utilização.

2.1.7 - Módulo simulation

Este módulo é responsável pela simulação propriamente dita.

Como parâmetros de entrada para este módulo têm-se:

- *xsize*, do tipo inteiro, dimensão da memória *frame* segundo a direcção *x*.
- *ysize*, do tipo inteiro, dimensão da memória *frame* segundo a direcção *y*.
- *dxl*, do tipo real, igual a $s_x d_x$ onde s_x é o factor de incerteza horizontal e d_x é a distância entre centros dos elementos sensores vizinhos na direcção *x*.
- *dy*, do tipo real, distância entre centros dos sensores CCD vizinhos na direcção *y*.
- *cx*, do tipo inteiro, coordenada do centro da imagem na memória *frame* segundo a direcção *x*.
- *cy*, do tipo inteiro, coordenada do centro da imagem na memória *frame* segundo a direcção *y*.
- *k*, do tipo real, factor de distorção radial da lente.
- *f*, do tipo real, distância focal efectiva.
- *verb*, do tipo inteiro; quando igual a *1*, o utilizador é informado dos resultados obtidos durante a execução da implementação.

Este módulo retorna:

- 0, como indicação de que a execução da implementação decorreu com normalidade.
- 1, ("Can't open output file <filename>."), o módulo não consegue abrir o ficheiro de saída especificado.

Este módulo, para cada posição da câmara desejada, executa:

I) Leitura, a partir do ficheiro *filesimulation*, da posição e orientação do referencial câmara relativamente ao referencial mundo, e do nome do ficheiro de saída respectivo.

II) Determinação da matriz de rotação envolvida; para tal utiliza o módulo ***compute_mr***.

III) Para cada segmento de recta no espaço 3D executa:

a) Leitura, a partir do ficheiro *filepoints*, das coordenadas 3D dos pontos inicial e final do segmento de recta respectivo.

b) Determinação das coordenadas, na memória *frame*, dos pontos inicial e final do segmento de recta respectivo; para tal utiliza o módulo ***world_frame***.

c) Utiliza o módulo ***clip2d*** para a determinação das coordenadas dos pontos inicial e final da parte visível do segmento de recta respectivo.

d) Executa a conversão das coordenadas dos pontos inicial e final do segmento de recta respectivo para coordenadas inteiras; para tal utiliza o módulo ***double_int***.

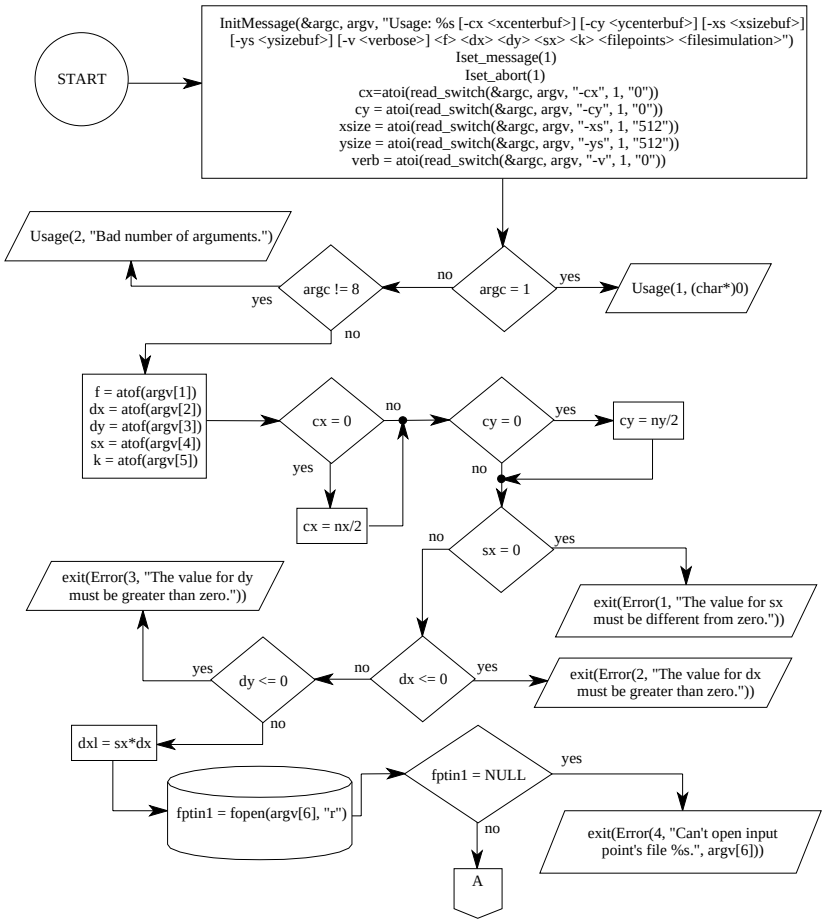
e) Escrita das coordenadas dos pontos inicial e final do segmento de recta em causa no ficheiro de saída respectivo.

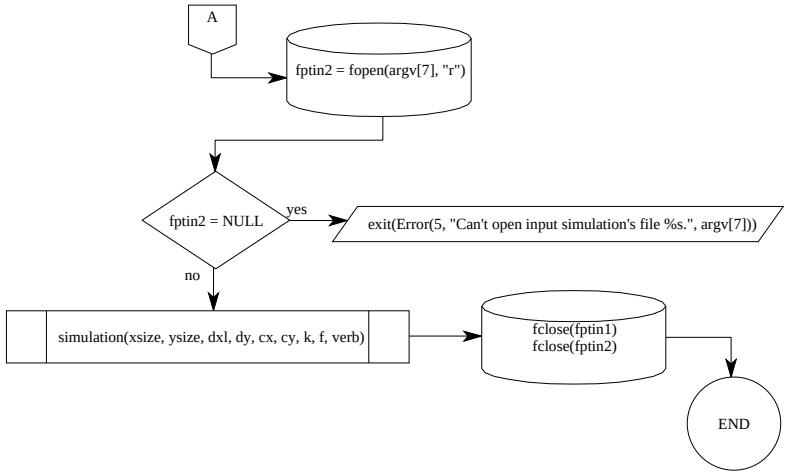
Este módulo não apresenta qualquer tipo de restrições quanto à sua utilização.

2.2 - Fluxogramas dos diferentes módulos

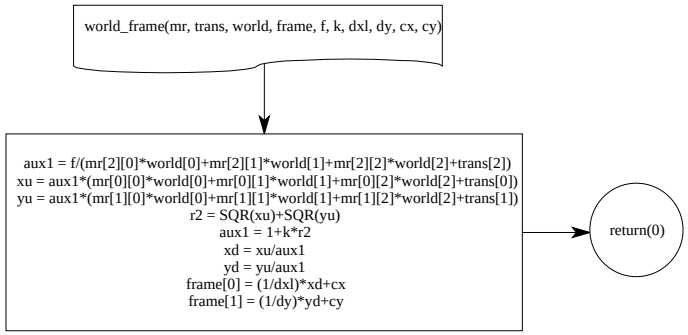
Apresentam-se, de seguida, os fluxogramas dos módulos que constituem a implementação ***simoca***, exceptuando o do módulo ***compute_mr***, que foi apresentado na secção anterior.

2.2.1 - Módulo principal

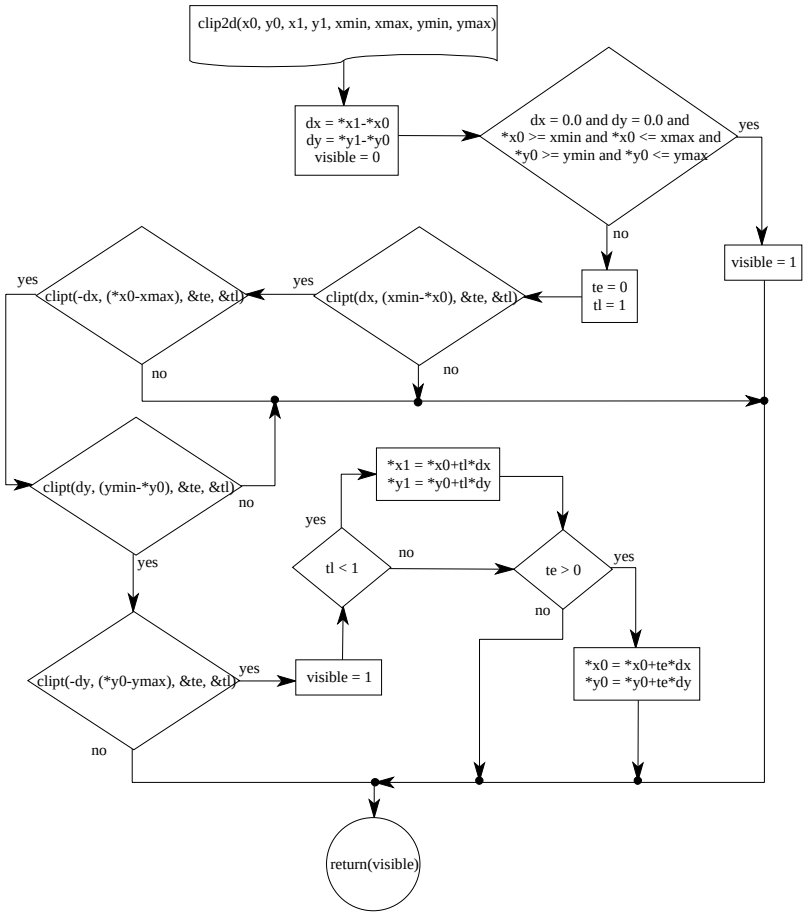




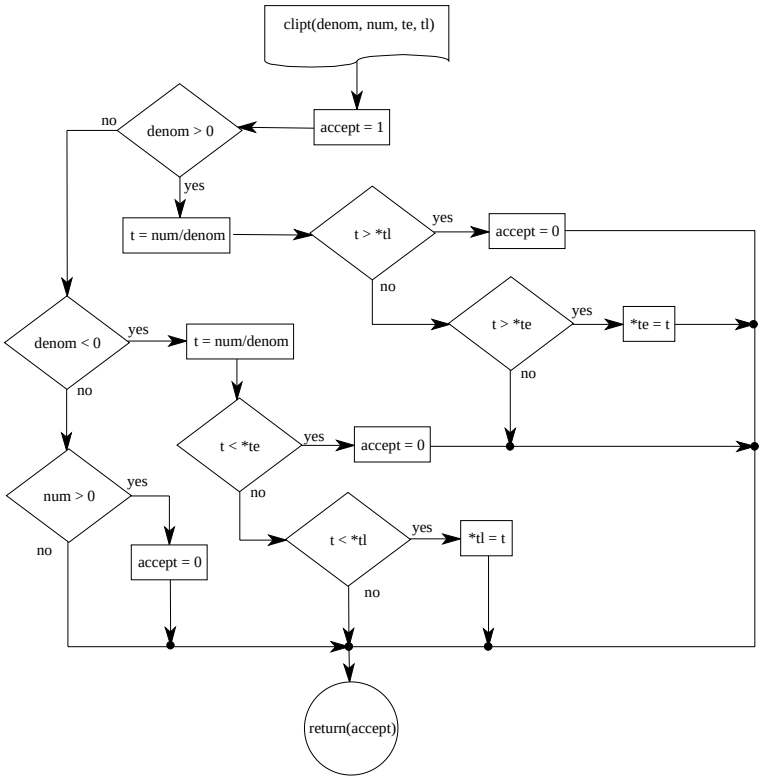
2.2.2 - Módulo world_frame



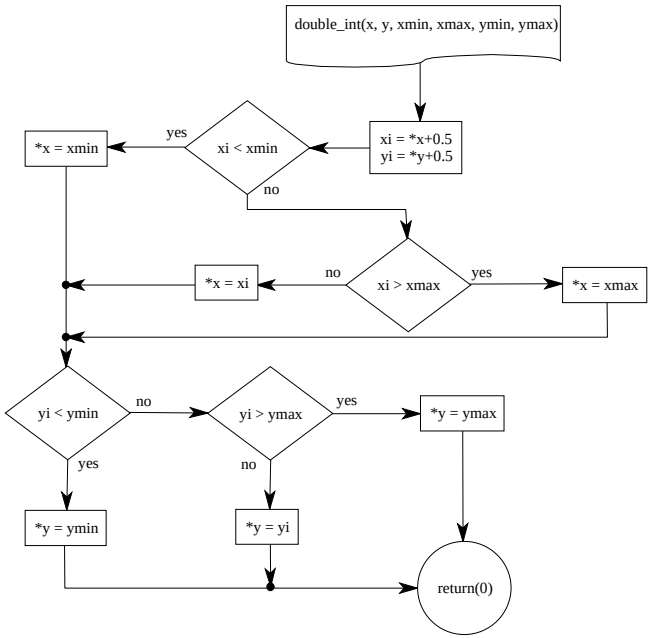
2.2.3 - Módulo clip2d



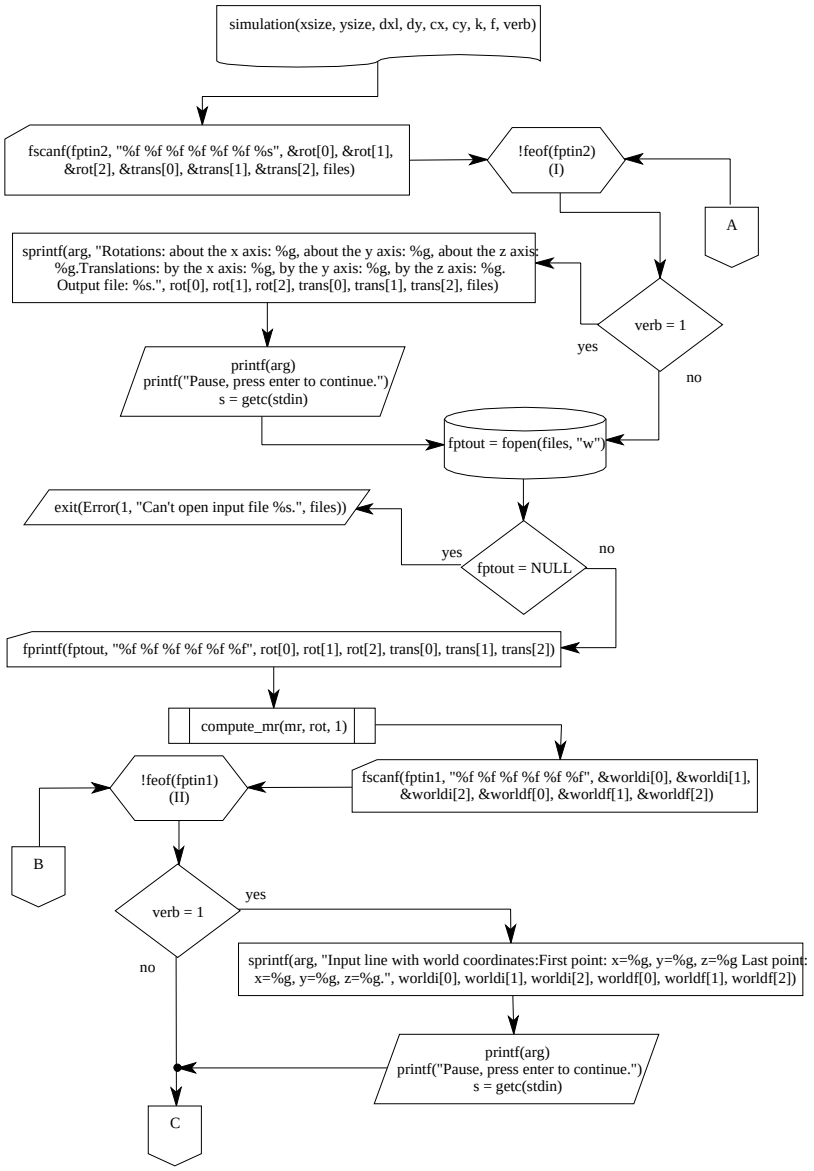
2.2.4 - Módulo clipt

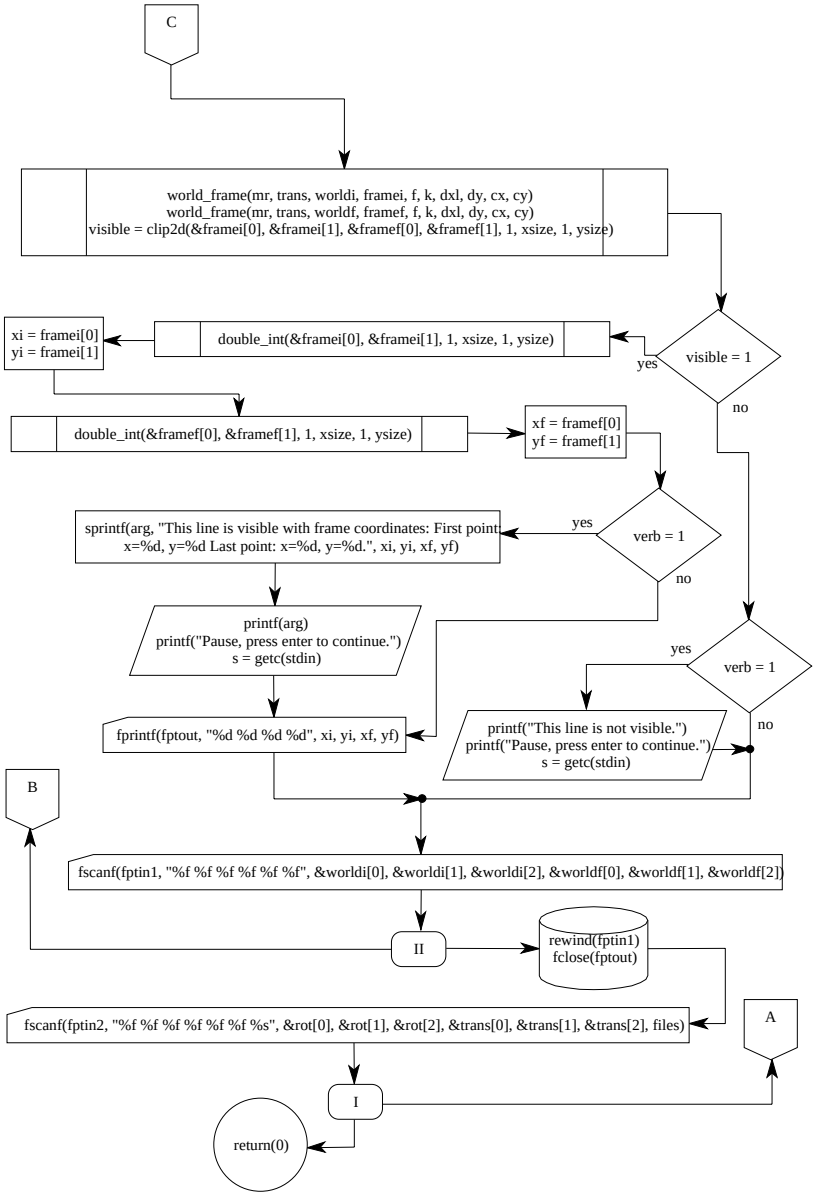


2.2.5 - Módulo double_int



2.2.6 - Módulo simulation





2.3 - Listagem da implementação

Apresenta-se, de seguida, a listagem da implementação ***simoca*** desenvolvida em linguagem *C* em ambiente *UNIX* numa estação de trabalho *SunSparc* e que se destina ao ambiente de processamento de imagem *XITE*[4], do qual utiliza algumas funções e tipos de variáveis já previamente definidas. Esta implementação contém comentários para auxílio à sua compreensão, assim como informação sobre os diferentes módulos, para sua posterior extracção através da ferramenta *CDOC(1)* existente no *XITE*.

```
/*
_____________________________________________________________________

simoca.c
@(#)simoca.c   1.00 94/06/08, Copyright 1994, DEEC, FEUP
Departamento de Engenharia Electrotecnica e de Computadores
Faculdade de Engenharia
4099 PORTO CODEX
PORTUGAL
E-mail: gpai@obelix.fe.up.pt
_____________________________________________________________________

readarg.h
@(#)readarg.h   1.3 92/01/15, Copyright 1990, Blab, UiO
Image processing lab, Department of Informatics
University of Oslo
E-mail: blab@ifi.uio.no
_____________________________________________________________________

message.h
@(#)message.h   1.3 92/01/15, Copyright 1990, Blab, UiO
Image processing lab, Department of Informatics
University of Oslo
E-mail: blab@ifi.uio.no
_____________________________________________________________________

  Permission to use, copy, modify and distribute this software and its documentation for any purpose
and without fee is hereby granted, provided that this copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in supporting documentation and that the
```

[4] *X-based Image processing Tools an Environmemt*, [Lønnestad, 1992].

```
/************************************************************************/

/* INCLUDES */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <blab/readarg.h>
#include <blab/message.h>
#include "gdefs.h"

/************************************************************************/

/* GLOBAL VARIABLES */

FILE *fptin1, *fptin2, *fptout;       /* file's pointer */

/************************************************************************/

/*F:world_frame*
_______________________________________________________________________

  world_frame
_______________________________________________________________________

Name:    world_frame - compute the frame buffer's coordinates of a point from
                       its world's coordinates

Syntax:    | int world_frame(mr, trans, world, frame, f, k, dxl, dy, cx, cy)
           | double mr[][3], frame[], f, k, dxl, dy;
           | float trans[], world[];

Description:   'world_frame' performs the determination of the frame buffer's
               coordinates of a point from its world's coordinates.
               'mr[][3]' is the rotation matrix 3D from object world coordinate
               system to the camera coordinate system.
               'trans[]' is the translation vector 3D from object world coordinate
               system to the camera coordinate system.
               'world[]' is the world's coordinates's vector.
               'frame[]' is the frame buffer's coordinates's vector.
               'dxl' is (sx*dx) where dx is the distance between adjacent sensor
               elements in x (scan line) direction and sx is the horizontal
               uncertainty factor.
               'dy' is the center to center distance between adjacent CCD sensor
               in the y direction.
               'cx' and 'cy' are the row and column numbers of the image's center
                in the frame buffer.
               'f' is the effective focal length.
               'k' is the lens's radial distortion.

Return value:  | 0 => Ok.

Example:       | Try yourself.

Restrictions:  'dx' and 'dy' must be different from zero.

Author:        Joao Tavares

Id:            @(#)world_frame.c   1.00   94/06/08
_______________________________________________________________________

*/

int world_frame(mr, trans, world, frame, f, k, dxl, dy, cx, cy)
double mr[][3], frame[], f, k, dxl, dy;
float trans[], world[];
int cx, cy;

{

 double aux1, xu, yu, r2, xd, yd;

 aux1 = f/(mr[2][0]*world[0]+mr[2][1]*world[1]+mr[2][2]*world[2]+trans[2]);

 /* compute the undistorted coordinates */

 xu = aux1*(mr[0][0]*world[0]+mr[0][1]*world[1]+mr[0][2]*world[2]+trans[0]);
 yu = aux1*(mr[1][0]*world[0]+mr[1][1]*world[1]+mr[1][2]*world[2]+trans[1]);
```

```
  /* compute the distorted coordinates */

  r2 = SQR(xu)+SQR(yu);
  aux1 = 1.0+k*r2;
  xd = xu/aux1;
  yd = yu/aux1;

  /* compute the frame coordinates */

  frame[0] = (1.0/dxl)*xd+(double)(cx);
  frame[1] = (1.0/dy)*yd+(double)(cy);

  return(0);

}

/*************************************************************************/

/*F:compute_mr*
_________________________________________________________________

  compute_mr
_________________________________________________________________

Name:       compute_mr - compute the rotation matrix 3D from object world
                         coordinate system to the camera coordinate system

Syntax:     | int compute_mr(mr, rot, c)
            | int c;
            | double mr[][3];
            | float rot[];

Description:  'compute_mr' performs the determination of the rotation matrix
              from object world coordinate system to the camera 3D coordinate
              system.
              'mr[][3]' is the rotation matrix 3D from object world coordinate
              system to the camera coordinate system.
              'rot[]' is the rotation's vector 3D from object world coordinate
              system to the camera coordinate system.
              If 'c' is 1 the angles are convert to rad.

Return value: | 0 => Ok.

Example:      | Try yourself.

Restrictions:  None.

Author:        Joao Tavares

Id:            @(#)compute_mr.c   1.00   94/06/10
_________________________________________________________________
*/

int compute_mr(mr, rot, c)
int c;
float rot[];
double mr[][3];

{

  double aux1, aux2, aux3, aux4, aux5, aux6;

  if (c == 1) {

    /* conversion deg/rad */

    rot[0] = ANG_DEG_RAD(rot[0]);
    rot[1] = ANG_DEG_RAD(rot[1]);
    rot[2] = ANG_DEG_RAD(rot[2]);

  }

  /* compute the rotation matrix */

  aux1 = cos(rot[0]);
  aux2 = sin(rot[0]);
  aux3 = cos(rot[1]);
  aux4 = sin(rot[1]);
  aux5 = cos(rot[2]);
  aux6 = sin(rot[2]);

  mr[0][0] = aux3*aux5;
  mr[0][1] = aux3*aux6;
  mr[0][2] = -aux4;
  mr[1][0] = aux5*aux2*aux4-aux1*aux6;
  mr[1][1] = aux6*aux2*aux4+aux1*aux5;
  mr[1][2] = aux2*aux3;
  mr[2][0] = aux5*aux4*aux1+aux2*aux6;
  mr[2][1] = -aux2*aux5+aux6*aux4*aux1;
```

```
  mr[2][2] = aux3*aux1;

  return(0);

}

/*********************************************************************/

/*F:clip2d*
_____________________________________________________________________

  clip2d
_____________________________________________________________________

Name:        clip2d - line clipping in the 2D space

Syntax:     | int clip2d(x0, y0, x1, y1, xmin, xmax, ymin, ymax)
            | double *x0, *y0, *x1, *y1, xmin, xmax, ymin, ymax;

Description:  'clip2d' performs the line clipping in the 2D space using the
              Liang-Barsky parametric algorithm.
              'x0' and 'y0' passed by reference are the coordinates of the first
              line's point.
              'x1' and 'y1' passed by reference are the coordinates of the last
              line's point.
              'xmin' and 'xmax' are the minimum and maximum values that the
              visible coordinates x can have.
              'ymin' and 'ymax' are the minimum and maximum values that the
              visible coordinates y can have.

Return value: | 0 => Line not visible.
              | 1 => Line visible.

Example:      | Try yourself.

Restrictions:  None.

Author:        Joao Tavares

Id:            @(#)clip2d.c   1.00   94/06/10
_____________________________________________________________________
*/

int clip2d(x0, y0, x1, y1, xmin, xmax, ymin, ymax)
double *x0, *y0, *x1, *y1, xmin, xmax, ymin, ymax;
{

  double dx, dy, te, tl;
  int visible;
  int clipt();

  dx = *x1-*x0;
  dy = *y1-*y0;
  visible = 0;
  if (dx == 0.0 && dy == 0.0 && *x0 >= xmin && *x0 <= xmax && *y0 >= ymin && *y0 <=
ymax) visible = 1;
  else {

    te = 0.0;
    tl = 1;
    if (clipt(dx, (xmin-*x0), &te, &tl)) {

      if (clipt(-dx, (*x0-xmax), &te, &tl)) {

        if (clipt(dy, (ymin-*y0), &te, &tl)) {

          if (clipt(-dy, (*y0-ymax), &te, &tl)) {

            visible = 1;
            if (tl < 1.0) {

              *x1 = *x0+tl*dx;
              *y1 = *y0+tl*dy;

            }
            if (te > 0.0) {

              *x0 = *x0+te*dx;
              *y0 = *y0+te*dy;

            }

          }

        }

      }

    }
```

```
  }

  return(visible);

}

/*********************************************************************/

/*F:clipt*
_____________________________________________________________________

  clipt
_____________________________________________________________________

Name:       clipt - computing the intersection of a line and an edge

Syntax:     |int clipt(denom, num, te, tl)
            | double denom, num, *te, *tl;

Description:  'clipt' computes a new value of 'te' or 'tl' for an interior
              intersection of a line segment and an edge for the Liang-Barsky
              parametric algorithm.
              'denom' is -(Ni.D) as defined in the Liang-Barsky parametric
              algorithm.
              'num' is Ni.(P0-Pei) as defined in the Liang-Barsky parametric
              algorithm.

Return value: | 0 => Not accept, the line is not inside the edge.
              | 1 => Accept.

Example:    | Try yourself.

Restrictions:  None.

Author:       Joao Tavares

Id:           @(#)clipt.c   1.00   94/06/10
_____________________________________________________________________
*/

int clipt(denom, num, te, tl)
double denom, num, *te, *tl;

{
  double t;
  int accept;

  accept = 1;
  if (denom > 0.0 ) {

    t = num/denom;
    if (t > *tl) accept = 0;
    else if(t > *te) *te = t;

  }
  else {

    if (denom < 0.0) {

      t = num/denom;
      if ( t < *te) accept = 0;
      else if (t < *tl) *tl = t;

    }
    else if (num > 0.0) accept = 0;

  }

  return(accept);

}

/*********************************************************************/

/*F:double_int*
_____________________________________________________________________

  double_int
_____________________________________________________________________

Name:       double_int - round double 2D coordinates to integer 2D coordinates

Syntax:     | int double_int(x, y, xmin, xmax, ymin, ymax)
            | double *x, *y;
            | int xmin, xmax, ymin, ymax;

Description:  'double_int' performs the rounding of double 2D coordinates to
              integer 2D coordinates.
```

```
                'x' and 'y' passed by reference are the coordinates.
                'xmin' and 'xmax' are the minimum and maximum values that the
                integer coordinate x can have.
                'ymin' and 'ymax' are the minimum and maximum values that the
                integer coordinate y can have.

Return value:  | 0 => Ok.

Example:       | Try yourself.

Restrictions:  None.

Author:        Joao Tavares

Id:            @(#)double_int.c   1.00   94/06/08
_____________________________________________________________________
*/

int double_int(x, y, xmin, xmax, ymin, ymax)
double *x, *y;
int xmin, xmax, ymin, ymax;

{

  int xi, yi;

  xi = *x+0.5;
  yi = *y+0.5;
  if (xi < xmin) *x = (double)(xmin);
  else {

    if(xi > xmax) *x = (double)(xmax);
    else *x = (double)(xi);

  }
  if (yi < ymin) *y = (double)(ymin);
  else {

    if(yi > ymax) *y = (double)(ymax);
    else *y = (double)(yi);

  }

  return(0);
}

/*********************************************************************/

/*F:simulation*
_____________________________________________________________________

  simulation
_____________________________________________________________________

Name:        simulation - camera simulation for lines in 3D world space

Syntax:      | int simulation(xsize, ysize, dxl, dy, cx, cy, k, f, verb)
             | int xsize, ysize, cx, cy, verb;
             | double dxl, dy, k, f;

Description:  'simulation' performs a camera simulation for lines in 3D world
              space using the TSAI's model.
              'xsize' and 'ysize' are the frame buffer's dimensions in directions x
              and y.
              'cx' and 'cy' are the image's center in the frame buffer for directions
              x and y.
              'dxl' is dx*sx where dx is the center to center distance between
              adjacent sensor elements in x (scanline) direction and sx is the
              horizontal uncertainty factor.
              'dy' is the center to center distance between adjacent CCD sensor
              in the y direction.
              'k' is the lens's radial distortion.
              'f' is the effective focal length.
              If 'verb' is 1 the user can see the output's results.

Return value:  | 0 => Ok.
               | 1 => Can't open input file <filename>.

Example:       | Try yourself.

Restrictions:  None.

Author:        Joao Tavares

Id:            @(#)simulation.c   1.00   94/06/08
_____________________________________________________________________
*/

int simulation(xsize, ysize, dxl, dy, cx, cy, k, f, verb)
```

```
int xsize, ysize, cx, cy, verb;
double dxl, dy, k, f;

{

  double framei[2], framef[2], mr[3][3];
  float rot[3], trans[3], worldi[3], worldf[3];
  int xi, yi, xf, yf, visible;
  char files[10], s, arg[350];
  int compute_mr();
  int clip2D();
  int double_int();
  int world_frame();

  fscanf(fptin2, "%f %f %f %f %f %f %s", &rot[0], &rot[1], &rot[2], &trans[0], &trans[1],
&trans[2], files);
  while (!feof(fptin2)) {

    if (verb == 1) {

      sprintf(arg, "\nRotations: about the x axis: %g, about the y axis: %g, about the z axis:
%g.\nTranslations: by the x axis: %g, by the y axis: %g, by the z axis: %g.\nOutput file: %s.\n",
rot[0], rot[1], rot[2], trans[0], trans[1], trans[2], files);
      printf(arg);
      printf("\nPause, press enter to continue.\n");
      s = getc(stdin);

    }

    /* open the ouput file */

    fptout = fopen(files, "w");

    if (fptout == NULL) exit(Error(1, "\nCan't open input file %s.\n", files));
    fprintf(fptout, "%f %f %f %f %f %f\n", rot[0], rot[1], rot[2], trans[0], trans[1], trans[2]);

    /* compute the rotation matrix */

    compute_mr(mr, rot, 1);

    fscanf(fptin1, "%f %f %f %f %f %f", &worldi[0], &worldi[1], &worldi[2], &worldf[0],
&worldf[1], &worldf[2]);
    while (!feof(fptin1)) {

      if  (verb == 1) {

        sprintf(arg, "\nInput line with world coordinates:\n\n\tFirst point: x=%g, y=%g, z=%g\t\Last
point: x=%g, y=%g, z=%g.\n\n", worldi[0], worldi[1], worldi[2], worldf[0], worldf[1], worldf[2]);
        printf(arg);
        printf("\nPause, press enter to continue.\n");
        s = getc(stdin);

      }

      /* compute the frame coordinates of the first and last points */

      world_frame(mr, trans, worldi, framei, f, k, dxl, dy, cx, cy);
      world_frame(mr, trans, worldf, framef, f, k, dxl, dy, cx, cy);

      /* make the clip of this line */

      visible = clip2d(&framei[0], &framei[1], &framef[0], &framef[1], 1.0, ((double)(xsize)), 1.0,
((double)(ysize)));

      /* if this line is visible write the first and last points in the output file */

      if (visible == 1) {

        double_int(&framei[0], &framei[1], 1, xsize, 1, ysize);
        xi = framei[0];
        yi = framei[1];
        double_int(&framef[0], &framef[1], 1, xsize, 1, ysize);
        xf = framef[0];
        yf = framef[1];
        if (verb == 1) {

          sprintf(arg, "\nThis line is visible with frame coordinates:\n\n\tFirst point: x=%d, y=%d\tLast
point: x=%d, y=%d.\n", xi, yi, xf, yf);
          printf(arg);
          printf("\nPause, press enter to continue.\n");
          s = getc(stdin);

        }

        fprintf(fptout, "%d %d %d %d\n", xi, yi, xf, yf);

      }
      else {

        if (verb == 1) {
```

```
        printf("\nThis line is not visible.\n");
        printf("\nPause, press enter to continue.\n");
        s = getc(stdin);

       }

     }
     fscanf(fptin1, "%f %f %f %f %f %f", &worldi[0], &worldi[1], &worldi[2], &worldf[0],
&worldf[1], &worldf[2]);

   }
   rewind(fptin1);

   /* close the output file */

   fclose(fptout);

   fscanf(fptin2, "%f %f %f %f %f %f %s", &rot[0], &rot[1], &rot[2], &trans[0], &trans[1],
&trans[2], files);

  }

  return(0);

}

/*************************************************************************/

/*P:simoca*
_________________________________________________________________

  simoca
_________________________________________________________________

Name:        simoca - camera simulation for lines in 3D world space

Syntax:     | simoca [-cx <xcenterbuf>] [-cy <ycenterbuf>] [-xs <xsizebuf>]
            | [-ys <ysizebuf>] [-v <verbose>] <f> <dx> <dy> <sx> <k>
            | <filespoints> <filesimulation>

Description:  'simoca' performs a camera simulation for lines in 3D world space
              using the TSAI's model.
              The center of frame buffer in direction x is indicate through flag
              '-cx', by default 'xcenterbuf' is (xsizebuf/2).
              The image's center in the frame buffer for direction y is indicate
              through flag '-cy', by default 'ycenterbuf' is (ysizebuf/2).
              The frame buffer's dimensions in directions x and y are indicate
              trough flags '-xs' and '-ys'. By default the frame buffer's dimensions
              are 512 by 512.
              The user can see the output's results if the flag '-v' is 1. By default
              'verbose' is 0.
              'f' is the effective focal length.
              'dx' is the center to center distance between adjacent sensor
              elements in x (scanline) direction.
              'dy' is the center to center distance between adjacent CCD sensor in
              the y direction.
              'sx' is the horizontal uncertainty factor.
              'k' is the lens's radial distortion.
              The line's end point's world's coordinates are in the file 'filepoints'.
              (for example:
              0.0 0.0 0.0 0.0 100.0 1000.0
              10.0 20.0 100.0 30.0 50.0 1000.0
              )
              The file 'filesimulation' contain the angles (in degree) of rotation 3D and
              the translation 3D from object world coordinate system to the camera
              coordinate system for the x, y and z axis, and the output file's name for
              each simulation's step. (for example:
              0.0 0.0 0.0 0.0 0.0 0.0
              out1.txt
              10.0 10.0 20.0 0.0 0.0 40.0
              out2.txt
              )

Return value: | 1 => The value for sx must be different from zero.
              | 2 => The value for dx must be greater than zero.
              | 3 => The value for dy must be greater than zero.
              | 4 => Can't open input point's file 'filepoints'.
              | 5 => Can't open input simulation's file 'filesimulation'.

Examples:     | Try yourself.

Restrictions: The value for sx must be different from zero.
              The values for dx and dy must be greater than zero.

Author:       Joao Tavares

Id:           @(#)simoca.c   1.0   94/06/18
_________________________________________________________________
*/
```

```
#ifdef MAIN

static char *SccsId = "@(#)simoca.c 1.00 94/06/08, DEEC, FEUP";

void main(argc, argv)
int argc;
char **argv;

{

  int xsize, ysize, cx, cy, verb;
  double f, dx, dy, dxl, sx, k, aux;

  InitMessage(&argc, argv, "Usage: %s\n [-cx <xcenterbuf>] [-cy <ycenterbuf>] [-xs <xsizebuf>] [-
ys <ysizebuf>] [-v <verbose>]\n <f> <dx> <dy> <sx> <k> <filepoints> <filesimulation>\n");
  Iset_message(1);
  Iset_abort(1);

  /* read switches */

  cx = atoi(read_switch(&argc, argv, "-cx", 1, "0"));
  cy = atoi(read_switch(&argc, argv, "-cy", 1, "0"));
  xsize = atoi(read_switch(&argc, argv, "-xs", 1, "512"));
  ysize = atoi(read_switch(&argc, argv, "-ys", 1, "512"));
  verb = atoi(read_switch(&argc, argv, "-v", 1, "0"));

  if (argc == 1) Usage(1, (char*)0);
  if (argc != 8) Usage(2, "\nBad number of arguments.\n");

  /* read argv[] */

  f = atof(argv[1]);
  dx = atof(argv[2]);
  dy = atof(argv[3]);
  sx = atof(argv[4]);
  k = atof(argv[5]);

  /* if cx and cy aren't given compute cx and cy by the frame buffer's size */

  if (cx == 0) cx = xsize/2;
  if (cy == 0) cy = ysize/2;

  if ( sx == 0.0) exit(Error(1, "\nThe value for sx must be different from zero.\n"));
  if ( dx <= 0.0) exit(Error(2, "\nThe value for dx must be greater than zero.\n"));
  if ( dy <= 0.0) exit(Error(3, "\nThe value for dy must be greater than zero.\n"));

  dxl = sx*dx;

  /* open the input point's file */

  fptin1 = fopen(argv[6], "r");
  if (fptin1 == NULL) exit(Error(4, "\nCan't open input point's file %s.\n", argv[6]));

  /* open the input simulation's file */

  fptin2 = fopen(argv[7], "r");
  if (fptin2 == NULL) exit(Error(5, "\nCan't open input simulation's file %s.\n", argv[7]));

  /* call simulation function */

  simulation(xsize, ysize, dxl, dy, cx, cy, k, f, verb);

  /* close the input files */

  fclose(fptin1);
  fclose(fptin2);

}

#endif

/*************************************************************************/
```

3 - Obtenção de informação tridimensional a partir do movimento de uma câmara: deep

Nesta secção é apresentada a implementação da abordagem, descrita em [Tavares, 1995], para a obtenção de informação tridimensional a partir do movimento de uma dada câmara. Como na abordagem proposta, as entidades consideradas são segmentos de recta, sendo os resultados apresentados na forma de uma imagem constituída por uma banda para cada intervalo, de largura definida pelo utilizador, de posições e orientações sucessivas da câmara. Em cada uma destas bandas aparecem desenhadas as diferentes entidades emparelhadas, com os pontos que as constituem com um nível de cinzento relacionado com a sua profundidade. Existe, do mesmo modo, a possibilidade de os resultados obtidos serem escritos num ficheiro de texto; neste caso, serão escritas as coordenadas memória *frame* e as coordenadas no espaço 3D dos pontos inicial e final de cada entidade emparelhada em duas imagens da sequência.

Para determinação da profundidade dos pontos intermédios de cada entidade utiliza-se, nesta implementação, uma aproximação baseada na interpolação linear das profundidades dos pontos inicial e final da mesma entidade.

Como é referido em [Tavares, 1995] durante a apresentação da abordagem seguida, para o seguimento ser obtido com maior êxito as imagens sucessivas não devem estar muito espaçadas; no entanto, para o erro resultante da obtenção das coordenadas tridimensionais ser reduzido a disparidade estereoscópica das duas posições e orientações consideradas deve ser razoável; assim, para ultrapassar esta contradição, optou-se por considerar todas as imagens para o seguimento, e determinar as coordenadas tridimensionais após um certo número de imagens de intervalo, sendo este número definido pelo utilizador.

Na fase de emparelhamento, esta implementação realiza uma primeira tentativa de emparelhamento de todas as entidades, utilizando filtros de Kalman[5] e distâncias normalizadas de Mahalanobis[6]; depois desta tentativa, as entidades ainda não emparelhadas sofrem uma segunda tentativa de emparelhamento, utilizando para tal filtros de Kalman e restrições geométricas[6].

3.1 - Descrição dos diferentes módulos

Esta implementação é constituída pelos módulos referidos na *Tab. I*. Cada um dos grupos definidos é constituído por módulos que são comuns a um dado contexto de aplicação. Assim, o grupo ***I*** contém os módulos mais particulares a esta implementação, o grupo ***II*** os relacionados com funções de visão tridimensional, o grupo ***III*** com resolução de sistemas lineares sobredeterminados, o grupo ***IV*** com as entidades utilizadas, o grupo ***V*** com os estados das entidades utilizados pelos filtros de Kalman, o grupo ***VI*** com operações de matrizes, o Grupo ***VII*** com listas ligadas, e o grupo ***VIII*** com gestão de memória, *Fig. 3*.

Apresenta-se de seguida a descrição de cada um destes módulos.

3.1.1 - Módulo principal

Este módulo é responsável pela interface implementação/utilizador. A seu cargo está:

- ✓ Leitura dos argumentos opcionais ("*switches*"):
  - *title*, ("[-t <title>]"), título da imagem de saída para representação dos resultados obtidos pela implementação. Por defeito, é "*Deep extracted*".

[5] Ver, por exemplo [Tavares, 1995] e [Maybeck, 1979].

[6] Ver, por exemplo [Tavares, 1995].

Grupo I	*Grupo II*	*Grupo III*	*Grupo IV*
principal	***compute_mpp***	***svdfit***	***create_line***
macthing	***frame_undistorted***	***svbksb***	***delete_line***
mahalanobis	***match_line***	***svdcmp***	***print_line***
measure	***stereo_eq***		***create_mpp***
elipse			***delete_mpp***
compute_deep			
dda_deep			
Grupo V	***Grupo VI***	***Grupo VII***	***Grupo VIII***
InitStateSpace	***create_matrix***	***create_list***	***memory_get***
RemoveStateSpace	***delete_matrix***	***delete_list***	***vector***
statePredictCovrpos	***print_matrix***	***insert_cell_list***	***matrix***
statePredict	***matrix_opr***	***delete_cell_list***	***free_vector***
stateUpdate	***matrix_mult***		***free_matrix***
stateDistance	***matrix_wsqr***		
copyState	***matrix_invert***		
	matrix_eig		

*Tab. I - Grupos que constituem a implementação **deep**.*

- *maxDeep*, ("[-m <maxDeep>]"), máxima profundidade a considerar na representação da profundidade dos pontos que constituem as diferentes entidades emparelhadas nas bandas da imagem de saída. Por defeito, é igual a *2800.0*.
- *minDeep*, ("[-n <minDeep>]"), mínima profundidade a considerar na representação da profundidade dos pontos que constituem as diferentes entidades emparelhadas nas bandas da imagem de saída. Por defeito, é igual a *0.0*.
- *nFrameDeep*, ("[-nf <nFrameDeep>]"), número de imagens de intervalo a considerar para a determinação das coordenadas 3D dos pontos inicial e final de cada entidade emparelhada. Por defeito, é igual a *0*.
- *xSize*, ("[-x <xSize>]"), dimensão da memória *frame* segundo a direcção *x*. Por defeito, é igual a *512*.
- *ySize*, ("[-y <ySize>]"), dimensão da memória *frame* segundo a direcção *y*. Por defeito, é igual a *512*.
- *sx*, ("[-s <sx>]"), factor de incerteza horizontal s_x. Por defeito, é igual a *0.710935*.
- *focalDist*, ("[-f <focalDist>]"), distância focal efectiva *f*. Por defeito, é igual a *50.0*.
- *radialLensDist*, ("[-k <radialLensDist>]"), factor de distorção radial da lente k_1. Por defeito, é igual a *0.001*.
- *dx*, ("[-dx <dx>]"), distância entre centros dos elementos sensores vizinhos na direcção *x*, d_x. Por defeito, é igual a *8.37765957e-3* mm.
- *dy*, ("[-dy <dy>]"), distância entre centros dos sensores CCD vizinhos na direcção *y*, d_y. Por defeito, é igual a *8.07560136e-3* mm.

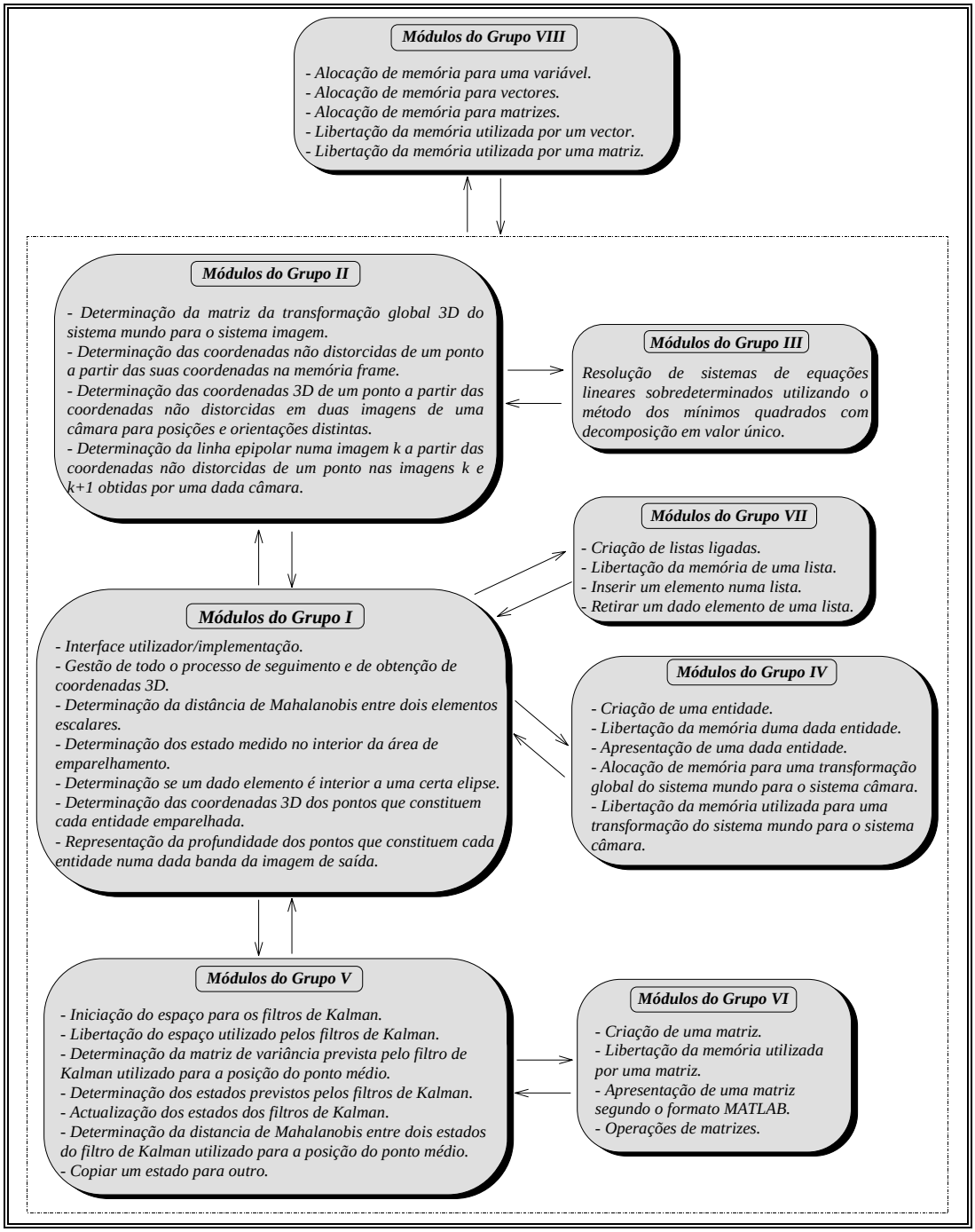


*Fig. 3 - Grupos integrantes da implementação **deep** e suas relações.*

- *xCenterBuf*, ("[-cx <xCenterBuf>]"), coordenada do centro da imagem na memória *frame* segundo a direcção *x*, C_x. Por defeito, é igual a metade da dimensão da memória *frame* segundo a mesma direcção.
- *yCenterBuf*, ("[-cy <yCenterBuf>]"), coordenada do centro da imagem na memória *frame* segundo a direcção *y*, C_y. Por defeito, é igual a metade da dimensão da memória *frame* segundo a mesma direcção.
- *maxDistMedP*, ("[-d1 <maxDistMedP>]"), máxima distância possível entre a posição do ponto médio de uma entidade e a prevista pelo filtro de Kalman respectivo, para a entidade em consideração ser declarada como candidata ao emparelhamento. Por defeito, é igual a *100.0*.

- *maxDifLength*, ("[-d2 <maxDifLength>]"), máxima diferença possível entre o comprimento de uma entidade e o previsto pelo filtro de Kalman respectivo, para a entidade em consideração ser declarada como candidata ao emparelhamento. Por defeito, é igual a *20.0*.
- *maxDifAng*, ("[-d3 <maxDifAng>]"), máxima diferença possível entre a direcção de uma entidade e a prevista pelo filtro de Kalman respectivo, para a entidade em consideração ser declarada como candidata ao emparelhamento. Por defeito, é igual a *10.0*.
- *outputFile*, ("[-fi <outputFile>]"), nome do ficheiro de saída para escrita dos resultados obtidos. Por defeito é "*NULL*", o que implica a não abertura do ficheiro. Quando especificado, o ficheiro, se ainda não existir, é criado; caso contrário, os resultados são acrescentados ao ficheiro já existente.
- *verbose*, ("[-v <verbose>]"); quando é especificado como igual a *1*, durante a execução da implementação o utilizador é informado de como esta está a decorrer, assim como dos resultados obtidos. Por defeito, o utilizador não é informado.

✓ Leitura dos argumentos passados à implementação pela linha de comando:

- *inputFile*, ("<inputFile>"), nome do ficheiro de texto que contém os nomes dos ficheiros, para cada posição e orientação da câmara, a considerar pela implementação. Cada um destes ficheiros contém a posição e a orientação do referencial câmara relativamente ao referencial mundo e as coordenadas memória *frame* dos pontos inicial e final de cada segmento de recta.
- *outputImage*, ("<outputImage>"), nome da imagem de saída para representação dos resultados obtidos pela implementação.

✓ Abertura do ficheiro de entrada *inputFile*.

✓ Determinação de quantos nomes de ficheiros o ficheiro de entrada *inputFile* contém.

✓ Determinação de quantas bandas serão utilizadas durante a execução da implementação.

✓ Criação da imagem de saída *ouputImage*, se o número de bandas a utilizar for diferente de zero.

✓ Abertura do ficheiro de saída *outputFile*, se desejado pelo utilizador, com a escrita da descrição de como os resultados foram obtidos.

✓ Chamada da função ***matching***.

✓ Fecho do ficheiro de entrada *inputFile*.

✓ Fecho do ficheiro de saída *outputFile*, se este foi desejado pelo utilizador.

✓ Escrita da imagem de saída *ouputImage*, com a descrição de como foi obtida, se o número de bandas utilizadas for diferente de zero.

✓ Comunicação da existência de erros em argumentos de entrada ou ao longo do decurso da execução:

- Através de valores de retorno:
  - 1, ("Wrong value for xsize <xSize>; it must be greater than zero."), o valor especificado para a dimensão da memória *frame* segundo a direcção *x* deve ser maior do que zero.
  - 2, ("Wrong value for ysize <ySize>; it must be greater than zero."), o valor especificado para a dimensão da memória *frame* segundo a direcção *y* deve ser maior do que zero.

- 3, (“Wrong value for f; it must be different from zero.”), o valor especificado para a distância focal efectiva deve ser diferente de zero.
- 4, (“Wrong value for sx; it must be different from zero.”), o valor especificado para o factor de incerteza horizontal deve ser diferente de zero.
- 5, (“Wrong value for dx <dx>; it must be greater than zero.”), o valor especificado para a distância entre centros dos elementos sensores vizinhos na direcção *x* deve ser maior do que zero.
- 6, (“Wrong value for dy <dy>; it must be greater than zero.”), o valor especificado para a distância entre centros dos sensores CCD vizinhos na direcção *y* deve ser maior do que zero.
- 7, (“Can't open input file <inputFile>.”), o módulo não consegue abrir o ficheiro especificado.
- 8, (“The input file <inputFile> contains only one name.”), o ficheiro especificado só contém um nome; deve, pelo menos, conter dois nomes.
- 9, (“Can't make output image <ouputImage>.”), o módulo não consegue criar a imagem de saída especificada.
- 10, (“Can't open output file <ouputFile>.”), o módulo não consegue abrir o ficheiro de saída especificado.

- Através de mensagens:
  - 1, (*InitMessage*), quando o número de argumentos é igual a um, isto é, apenas o nome da implementação.
  - 2, (“Bad number of arguments.”), quando o número de argumentos é diferente de três.

O ficheiro de entrada *inputFile* deverá ser constituído pelos nomes dos ficheiros que contêm a posição e orientação do referencial mundo relativamente ao referencial câmara e as coordenadas na memória *frame* dos pontos inicial e final de cada segmento de recta, para cada posição e orientação da câmara a considerar. Este ficheiro deverá ter a forma: $ficheiro^i\ ficheiro^j$.... Cada posição e orientação da câmara terá deste modo um ficheiro associado. Cada um destes ficheiros contém as três rotações e as três translações da transformação geométrica 3D do sistema de coordenadas mundo para o sistema câmara e as coordenadas na memória *frame* dos pontos inicial e final de cada segmento de recta a considerar. A forma destes ficheiros deverá ser então: $R_{(x,\phi)}\ R_{(y,\theta)}\ R_{(z,\psi)}\ t_x\ t_y\ t_z\ x_{f_k}^{\ i}\ y_{f_k}^{\ i}\ x_{f_k}^{\ f}\ y_{f_k}^{\ f}\ x_{f_l}^{\ i}\ y_{f_l}^{\ i}\ x_{f_l}^{\ f}\ y_{f_l}^{\ f}$ ....

O ficheiro de saída dos resultados obtidos pela implementação, quando desejado pelo utilizador, terá a forma: $x_{f_k}^{\ i}\ y_{f_k}^{\ i}\ x_{w_k}^{\ i}\ y_{w_k}^{\ i}\ z_{w_k}^{\ i}\ x_{f_k}^{\ f}\ y_{f_k}^{\ f}\ x_{w_k}^{\ f}\ y_{w_k}^{\ f}\ z_{w_k}^{\ f}\ x_{f_l}^{\ i}\ y_{f_l}^{\ i}\ x_{w_l}^{\ i}\ y_{w_l}^{\ i}\ z_{w_l}^{\ i}\ x_{f_l}^{\ f}\ y_{f_l}^{\ f}\ x_{w_l}^{\ f}\ y_{w_l}^{\ f}\ z_{w_l}^{\ f}$ .... Sempre que o ficheiro especificado já exista, os resultados obtidos pela implementação serão acrescentados.

Como restrições quanto à utilização deste módulo têm-se:

- Os valores especificados para o factor de incerteza horizontal e para a distância focal efectiva devem ser diferentes de zero.
- Os valores especificados para a distância entre centros dos elementos sensores vizinhos na direcção *x* e para a distância entre centros dos sensores CCD vizinhos na direcção *y* devem ser maiores do que zero.
- Os valores especificados para as dimensões da memória *frame* segundo as direcções *x* e *y* devem ser maiores do que zero.
- O ficheiro de entrada *inputFile* deve conter mais do que um nome.

3.1.2 - Módulo matching

Este módulo é responsável por toda a gestão dos restantes módulos que constituem esta implementação; assim, este módulo coordena todo o processo de seguimento 2D e obtenção de coordenadas 3D.

Como parâmetros de entrada para este módulo têm-se:

- *imgout*, do tipo "*IMAGE*", imagem de saída na qual deverão ser representadas as entidades devidamente emparelhadas, numa relação profundidade/nível de cinzento.
- *xsize*, do tipo inteiro, dimensão da memória *frame* segundo a direcção *x*.
- *ysize*, do tipo inteiro, dimensão da memória *frame* segundo a direcção *y*.
- *maxdeep*, do tipo real, máxima profundidade a considerar na representação da profundidade dos pontos que constituem as diferentes entidades emparelhadas nas bandas da imagem de saída.
- *mindeep*, do tipo real, mínima profundidade a considerar na representação da profundidade dos pontos que constituem as diferentes entidades emparelhadas nas bandas da imagem de saída.
- *nfdeep*, do tipo inteiro, número de posições e orientações de intervalo a considerar para a determinação das coordenadas 3D dos pontos inicial e final de cada entidade emparelhada.
- *f*, do tipo real, distância focal efectiva.
- *k*, do tipo real, factor de distorção radial da lente.
- *cxf*, do tipo inteiro, coordenada do centro da imagem na memória *frame* segundo a direcção *x*.
- *cyf*, do tipo inteiro, coordenada do centro da imagem na memória *frame* segundo a direcção *y*.
- *dxl*, do tipo real, igual a $s_x d_x$ onde s_x é o factor de incerteza horizontal e d_x é a distância entre centros dos elementos sensores vizinhos na direcção *x*.
- *dy*, do tipo real, distância entre centros dos sensores CCD vizinhos na direcção *y*.
- *d*, do tipo real, passado por endereço, vector que contém os valores das restrições geométricas respectivas. Assim:
  - *d[0]* é a máxima distância possível entre a posição do ponto médio de uma entidade e a prevista pelo filtro de Kalman respectivo, para a entidade em consideração ser declarada como candidata ao emparelhamento.
  - *d[1]* é a máxima diferença possível entre o comprimento de uma entidade e o previsto pelo filtro de Kalman respectivo, para a entidade em consideração ser declarada como candidata ao emparelhamento.
  - *d[2]* é a máxima diferença possível entre a direcção de uma entidade e a prevista pelo filtro de Kalman respectivo, para a entidade em consideração ser declarada como candidata ao emparelhamento.
- *nfiles*, do tipo inteiro, número total de posições e orientações da câmara a considerar.
- *verb*, do tipo inteiro; quando é igual a *1*, durante a execução do módulo o utilizador é informado de como esta está a decorrer, assim como dos resultados obtidos.

Este módulo executa as seguintes tarefas:

I) Criação de uma lista ligada linear para o modelo de entidades 2D e outra para as matrizes das transformações geométricas 3D do sistema de coordenadas mundo para o sistema câmara,

para as sucessivas posições e orientações da câmara a considerar. Para o efeito, utiliza o módulo ***create_list***.

II) Criação das matrizes necessárias para a utilização dos filtros de Kalman no seguimento. Para o efeito, utiliza a função ***InitStateSpace***.

III) Cálculo da área de incerteza a ser utilizada na fase de inicialização das entidades. Para tal, utiliza: para a criação das matrizes auxiliares necessárias, o módulo ***create_matrix***; para a previsão da variância[7] de posição do ponto médio, o módulo ***statePredictCovrpos***; para o cálculo dos vectores e valores próprios da matriz de incerteza, o módulo ***matrix_eig***.

IV) Para todas as posições e orientações sucessivas da câmara a considerar, executa:

a) Abertura do ficheiro de entrada respectivo, determinação de quantos segmentos de recta contém, reserva de memória necessária para a estrutura que irá conter estas entidades, leitura, a partir do ficheiro respectivo, das coordenadas imagem distorcida de cada entidade e cálculo das suas características utilizadas. Quando desejado pelo utilizador, este é informado das coordenadas imagem distorcida, do comprimento e da direcção de cada uma das entidades que o módulo está a ler do ficheiro de entrada.

b) Determinação da matriz da transformação geométrica 3D do sistema de coordenadas mundo para o sistema câmara, para a posição e orientação da câmara que está a ser considerada. Para o efeito, utiliza: o módulo ***create_mpp***, para criar um elemento do tipo *mpp*, e o módulo ***compute_mpp*** para determinação da matriz de transformação respectiva.

c) Caso não seja o primeiro ficheiro de entrada a ser considerado:

i) Determina se deve ou não extrair as coordenadas 3D. Se sim, (isto é, se o número de ficheiros de entrada já considerados for múltiplo de *nfdeep*) cria uma banda na imagem de saída, assumindo todos os seus *pixels* o valor equivalente a branco.

ii) Tenta obter o emparelhamento de cada elemento que constitui a modelo de entidades 2D com um elemento que constitui a estrutura de entidades que ainda não foram emparelhadas, utilizando filtros de Kalman e distâncias normalizadas de Mahalanobis. Assim, executa a previsão do estado da respectiva entidade, determina os valores e vectores próprios da matriz de incerteza de posição do ponto médio e determina o respectivo estado medido. Caso exista alguma entidade da estrutura cujo ponto médio é interior à respectiva elipse de emparelhamento, o módulo determina qual a melhor das candidatas ao emparelhamento com a entidade. Esta candidata deverá apresentar:

- Distâncias normalizadas de Mahalanobis para a direcção, o comprimento e a posição do ponto médio, inferiores aos respectivos valores definidos globalmente. De todas as candidatas ao emparelhamento que obedecem à condição anterior, selecciona a que possui o menor produto das distâncias anteriores[8].

Se o emparelhamento é conseguido com êxito, o módulo executa a actualização da entidade no modelo de entidades 2D, determina se os pontos inicial e final da imagem da entidade nas imagens obtidas pela câmara nas duas posições e orientações sucessivas sofreram uma inversão na sua ordem, actualizando a respectiva etiqueta da entidade. Quando desejado, o utilizador é informado que foi obtido emparelhamento com êxito e do modo como este decorreu. De seguida, o módulo executa a actualização do estado da respectiva entidade no modelo de entidades 2D.

[7] Os valores de inicialização da matriz de variância estão definidos globalmente.

[8] Para resultados no emparelhamento o mais correctos possível, deve-se ter especial atenção ao valor que se atribui às variáveis definidas globalmente para os valores máximos admissíveis das respectivas distâncias normalizadas. Doutro modo, podem ocorrer erros no emparelhamento devido à propriedade absorvente da multiplicação.

iii) Tenta novamente obter o emparelhamento de cada entidade ainda não emparelhada que constitui o modelo de entidades 2D, com um elemento que constitui a estrutura de entidades e que também ainda não foi emparelhado, utilizando filtros de Kalman e restrições geométricas. Assim, executa a previsão do estado da respectiva entidade e considera como candidatas ao emparelhamento com a respectiva entidade, todas as entidades da estrutura que ainda não foram emparelhadas e que verifiquem as seguintes condições:

- A diferença entre a posição do seu ponto médio e a prevista para a entidade em causa pelo respectivo filtro de Kalman, deve ser inferior à máxima distância admissível definida pelo utilizador.
- A diferença entre o seu comprimento e o previsto para a entidade em causa pelo respectivo filtro de Kalman, deve ser inferior ao valor máximo admissível definido pelo utilizador.
- A diferença entre a sua direcção e a prevista para a entidade em causa pelo respectivo filtro de Kalman, deve ser inferior ao valor máximo admissível definido pelo utilizador.
- De todas as entidades candidatas ao emparelhamento que obedeçam às condições anteriores, é determinada como melhor candidata ao emparelhamento com a entidade em causa, aquela que possui o menor produto das diferenças anteriores[9].

Após esta fase, calcula os valores e vectores próprios da matriz de incerteza de posição. De seguida, determina o respectivo estado medido, transladando previamente o centro da elipse de emparelhamento de modo a coincidir com o ponto médio da entidade determinada como melhor candidata ao emparelhamento. Caso exista alguma entidade da estrutura de entidades que seja realmente uma boa candidata ao respectivo emparelhamento (o estado medido é devidamente coerente), o mesmo é conseguido com êxito. Neste caso, o módulo executa a actualização da entidade no respectivo modelo, determina se os pontos inicial e final da imagem da entidade nas imagens obtidas pela câmara nas duas posições e orientações sucessivas sofreram uma inversão na sua ordem, actualizando a respectiva etiqueta da entidade. Quando desejado, o utilizador é informado de que foi obtido emparelhamento com êxito e do modo como este decorreu.

iv) Caso, após esta nova tentativa de emparelhamento, a entidade não seja emparelhada com êxito, o utilizador, se desejado, é disso informado, passando o módulo a actualizar a entidade em causa. Assim, esta entidade assume o estado médio medido como igual ao estado médio previsto e o estado da variância medido como igual ao estado da variância de inicialização.

v) Após as fases de emparelhamento, o módulo actualiza o factor de confiança de todas as entidades do respectivo modelo. Esta actualização é realizada do seguinte modo: se este factor é inferior a cinco e a entidade foi emparelhada com êxito, então é incrementado uma unidade; caso a entidade em causa não tenha sido emparelhada com êxito, então o mesmo factor é decrementado uma unidade; se este factor de confiança passar a ser menor do que zero, a entidade é retirada do modelo.

Caso o número de ficheiros de entrada já considerados seja múltiplo de *nfdeep*, também nesta fase o módulo determina as coordenadas tridimensionais das entidades do modelo 2D que foram emparelhadas com êxito, cujo factor de confiança seja superior ou igual a três e cuja matriz da transformação geométrica do sistema de coordenadas mundo para

[9] Para resultados no emparelhamento o mais correctos possível deve-se ter especial atenção ao valor que se atribui aos valores máximos admissíveis das respectivas distâncias. Doutro modo, podem ocorrer erros no emparelhamento devido à propriedade absorvente da multiplicação.

o sistema câmara anteriormente considerada para esta entidade difira da actual pelo menos de *nfdeep* posições e orientações da câmara.

d) Todos os elementos que constituem a estrutura de entidades que não foram emparelhados, são inseridos, após a respectiva inicialização, no modelo de entidades 2D. Esta inicialização é realizada do seguinte modo: a entidade é inicializada com os respectivos valores por defeito; após esta inicialização, procede-se à fase de medição considerando a elipse de incerteza de inicialização como centrada no respectivo ponto médio; de seguida, executa-se a actualização da respectiva entidade.

e) Após esta actualização do modelo 2D, o módulo liberta a memória utilizada com a estrutura de entidades e, se desejado pelo utilizador, apresenta o actual estado do modelo.

V) No final da sua execução liberta a memória utilizada: pelas matrizes necessárias à utilização dos filtros de Kalman no seguimento (para tal, utiliza o módulo ***removeStateSpace***); pelas matrizes auxiliares (utilizando para o efeito o módulo ***delete-matrix***); pelas listas ligadas lineares do modelo 2D e das transformações geométricas 3D do sistema de coordenadas mundo para o sistema câmara (utilizando para o efeito o módulo ***delete_list***).

Este módulo, além dos módulos já anteriormente citados, utiliza os seguintes módulos:

- ***frame_undistorted***, para o cálculo das coordenadas imagem não distorcida a partir das correspondentes coordenadas na memória *frame*.
- ***mahalanobis***, para o cálculo da distância normalizada de Mahalanobis para uma única característica.
- ***_lnext***, para se mover de um elemento para outro numa lista linear ligada.
- ***_pinfo***, para aceder à informação de cada entidade numa dada lista.
- ***delete_cell_list***, para retirar de uma lista um dado elemento.
- ***statePredict***, para executar a previsão do estado de cada entidade que constitui o modelo de entidades 2D.
- ***measure***, para a determinação do estado medido.
- ***stateUpdate***, para a actualização das entidades no modelo de entidades 2D.
- ***compute_deep***, para o cálculo das coordenadas tridimensionais para a entidade em causa, e escrita no ficheiro de saída dos resultados obtidos, se desejado, assim como a representação dos resultados obtidos na imagem de saída.

Como valores de retorno, este módulo apresenta:

- *1*, (“Can’t open input file <filename>.”), o módulo não consegue abrir o ficheiro de entrada especificado.
- *2*, (“The input file <filename>, only has one row; it must have at least two”), o ficheiro de entrada apenas contém uma linha; deverá pelo menos apresentar duas.
- *3*, (“Not enough memory.”), não existe memória suficiente para a execução do módulo.

Este módulo retorna *0* como indicação de que a sua execução decorreu em normalidade, e apresenta as seguintes restrições quanto à sua utilização:

- Os ficheiros de entrada devem conter pelos menos duas linhas, isto é, uma constituída pelos valores das três rotações e das três translações da transformação geométrica 3D do sistema de coordenadas mundo para o sistema câmara, e outra constituída pelas coordenadas 2D na memória *frame* dos pontos inicial e final de pelo menos uma entidade.
- A existência de memória suficiente.

3.1.3 - Módulo mahalanobis

Este módulo é responsável pela determinação da distância normalizada de Mahalanobis entre dois elementos *m* e *p*, para uma única característica.

Como parâmetros de entrada para este módulo têm-se:

- *am*, do tipo real, característica do elemento *m*.
- *vm*, do tipo real, variância associada à característica do elemento *m*.
- *ap*, do tipo real, característica do elemento *p*.
- *vp*, do tipo real, variância associada à característica do elemento *p*.

Caso *vm + vp = 0*, então o módulo calcula metade da distância Euclidiana, pois, neste caso, não é possível calcular a distância de normalizada de Mahalanobis respectiva.

Este módulo retorna a distância calculada e não apresenta qualquer tipo de restrições quanto à sua utilização.

3.1.4 - Módulo measure

Este módulo é responsável por efectuar a medição das variâncias de posição, de comprimento e de direcção, no interior da elipse de emparelhamento. Do mesmo modo, este módulo é responsável pela determinação do número de entidades cujo ponto médio é interior à referida elipse.

Como parâmetros de entrada para este módulo têm-se:

- *line*, do tipo *lines*, passado por endereço, estrutura que contém as entidades que existem no ficheiro de entrada para a posição e orientação da câmara que está a ser considerada.
- *n*, do tipo inteiro, número de entidades que constituem a estrutura *line*.
- *ms*, do tipo *State*, passado por endereço, estado medido pretendido da entidade que se pretende emparelhar.
- *ps*, do tipo *State*, passado por endereço, estado previsto da entidade que se pretende emparelhar.
- *vx*, do tipo real, componente segundo a direcção *x* de um vector paralelo ao maior eixo da elipse de emparelhamento.
- *vy*, do tipo real, componente segundo a direcção *y* de um vector paralelo ao maior eixo da elipse emparelhamento.
- *cx*, do tipo real, coordenada do centro da elipse de emparelhamento segundo a direcção *x*.
- *cy*, do tipo real, coordenada do centro da elipse de emparelhamento segundo a direcção *y*.
- *a*, do tipo real, dimensão do maior raio da elipse de emparelhamento.
- *b*, do tipo real, dimensão do menor raio da elipse de emparelhamento.
- *mark*, do tipo inteiro; quando igual a *1*, a etiqueta de cada entidade para identificação que o seu ponto médio é interior à dada elipse de emparelhamento assume no fim da execução o valor *0*.

Para o cálculo das variâncias de posição, este módulo determina os momentos no interior de elipse de emparelhamento especificada, utilizando o exponencial negativo da soma das distâncias normalizadas de Mahalanobis para a direcção e comprimento, como função de peso.

Este módulo utiliza o módulo ***elipse*** para determinar se uma dada entidade tem o respectivo ponto médio interior à elipse de emparelhamento, e o módulo ***mahalanobis*** para o cálculo da distância normalizada de Mahalanobis para uma única característica.

Não existe qualquer tipo de restrições quanto à utilização deste módulo, sendo o valor de retorno, no fim da execução do mesmo, o número de entidades da estrutura *line* cujo ponto médio é interior à elipse de emparelhamento.

3.1.5 - Módulo elipse

Este módulo é responsável por determinar se um determinado ponto é ou não interior a uma dada elipse.

Como parâmetros de entrada para este módulo têm-se:

- *vx*, do tipo real, componente segundo a direcção *x* de um vector paralelo ao maior eixo da elipse.
- *vy*, do tipo real, componente segundo a direcção *y* de um vector paralelo ao maior eixo da elipse.
- *cx*, do tipo real, coordenada do centro da elipse segundo a direcção *x*.
- *cy*, do tipo real, coordenada do centro da elipse segundo a direcção *y*.
- *a*, do tipo real, dimensão do maior raio da elipse.
- *b*, do tipo real, dimensão do menor raio da elipse.
- *px*, do tipo real, coordenada do ponto segundo a direcção *x*.
- *py*. do tipo real, coordenada do ponto segundo a direcção *y*.

Para este módulo determinar se o ponto dado é ou não interior à elipse, roda o ponto em torno do centro da elipse de maneira que o eixo maior desta fique horizontal, e depois, aplica a equação da elipse para verificar se o ponto rodado é ou não interior a esta.

Este módulo não apresenta qualquer tipo de restrições quanto à sua utilização e retorna:

- 0, caso o ponto não seja interior à elipse.
- 1, caso o ponto seja interior à elipse.

3.1.6 - Módulo compute_deep

Este módulo é responsável pela determinação das coordenadas 3D dos pontos inicial e final de uma dada entidade, após a determinação da verdadeira localização dos pontos inicial e final do segmento de recta, visíveis numa dada imagem, no segmento visível na imagem anterior a esta, e com o qual foi devidamente emparelhado.

Como parâmetros de entrada para este módulo têm-se:

- *band*, do tipo "*BAND*", banda da imagem de saída na qual o módulo ***dda_deep*** deverá representar a respectiva entidade.
- *ep*, do tipo *Line*, passado por endereço, entidade pertencente à lista ligada que define o modelo de entidades 2D, cujas coordenadas 3D dos pontos inicial e final se pretende determinar.
- *mppnew[ ][4]*, do tipo real, matriz da transformação geométrica 3D do sistema de coordenadas mundo para o sistema câmara, correspondente à actual posição e orientação da câmara.

- *mppold[ ][4]*, do tipo real, matriz da transformação geométrica 3D do sistema de coordenadas mundo para o sistema câmara, correspondente à anterior posição e orientação da câmara considerada para a determinação das coordenadas 3D da respectiva entidade.

- *maxdeep*, do tipo real, máxima profundidade a considerar na representação da profundidade. Utilizado na definição da relação profundidade/cinzento no módulo ***dda_deep***.

- *mindeep*, do tipo real, mínima profundidade a considerar na representação da profundidade. Utilizado na definição da relação profundidade/cinzento no módulo ***dda_deep***.

- *verb*, do tipo inteiro; quando igual a *1*, o utilizador é informado das coordenadas 3D, determinadas pelo módulo, para os pontos inicial e final da respectiva entidade.

Este módulo, antes de determinar as coordenadas 3D dos pontos extremos da entidade, determina a verdadeira localização dos pontos inicial e final do segmento de recta, visíveis numa dada imagem, no segmento visível na imagem anterior a esta, e com o qual foi devidamente emparelhado. Assim:

- Este módulo determina a linha epipolar do ponto inicial da entidade na imagem obtida pela câmara para a actual posição e orientação, no plano imagem da câmara para a anterior posição e orientação considerada para a referida entidade. Se esta linha epipolar não for paralela à imagem da entidade obtida pela câmara para a anterior posição e orientação, calcula a intersecção destas linhas. De seguida, determina a linha epipolar do ponto final da entidade na imagem obtida pela câmara para a actual posição e orientação, no plano imagem da câmara para a anterior posição e orientação. Novamente, se esta linha epipolar não for paralela à imagem da entidade obtida pela câmara para a anterior posição e orientação, calcula a intersecção destas linhas.

- Cada um dos dois pontos obtidos pela intersecção da imagem da entidade, obtida pela câmara para a anterior posição e orientação, com a respectiva linha epipolar, só é considerado como correspondente à verdadeira localização se o ângulo (em grau) formado por estas linhas for superior ao valor da variável definida globalmente *NEARPAR*. Quando a condição anterior é verificada apenas para um só ponto, então o módulo associa o ponto de intersecção como correspondente à verdadeira localização do referido ponto, e o extremo da imagem da entidade, obtida pela câmara para a anterior posição e orientação, que lhe é mais afastado como correspondente à localização verdadeira do outro ponto. Quando a mesma condição não é verificada para nenhum dos dois pontos, então o módulo verifica se a entidade sofreu alguma inversão da correspondência entre os pontos iniciais e finais nas imagens obtidas pela câmara. Caso não se tenha verificado inversão, o módulo associa uma correspondência directa entre os pontos iniciais e finais de cada imagem da entidade, obtidas pela câmara para as duas posições e orientações. No caso contrário, o módulo associa uma correspondência invertida entre os pontos iniciais e finais de cada imagem da entidade. De notar que este procedimento só é correcto quando a entidade não sofre grandes variações de posição nas sucessivas imagens da sequência.

Durante a sua execução, este módulo utiliza outros módulos, nomeadamente: para a determinação das linhas epipolares, o módulo ***match_line***; para a resolução do sistema de equações estéreo, o módulo ***stereo_eq*** e, para a representação da respectiva entidade na banda da imagem de saída, o módulo ***dda_deep***.

Quando é desejado pelo utilizador, este módulo é também responsável pela escrita das coordenadas na memória *frame* e das coordenadas 3D dos pontos inicial e final da entidade no ficheiro de saída.

Este módulo retorna *0* como indicação de que a sua execução decorreu com normalidade, e não apresenta qualquer tipo de restrições quanto à sua utilização.

3.1.7 - Módulo dda_deep

Este módulo é responsável pelo desenho de segmentos de recta numa banda da imagem de saída. Trata-se de uma implementação do algoritmo de Bresenham[10], também designado por versão meio ponto do segmento. Este algoritmo tem como grande vantagem a não utilização de multiplicações, de divisões e de funções de arredondamento. A cada ponto que pertence ao segmento de recta, é atribuído, na banda da imagem de saída, o valor de cinzento correspondente à sua profundidade, sendo esta determinada aproximadamente por interpolação linear das profundidades dos pontos inicial e final do respectivo segmento. Para uma melhor visualização, a gama de cinzento disponível não é toda utilizada; assim, um certo número de elementos inferiores e superiores não são utilizados, sendo este número definido por um parâmetro de entrada.

Como parâmetros de entrada para este módulo têm-se:

- *band*, do tipo "*IBAND*", banda na qual se pretende desenhar o segmento de recta.
- *xi*, do tipo inteiro, coordenada do ponto inicial do segmento de recta, segundo a direcção *x*.
- *yi*, do tipo inteiro, coordenada do ponto inicial do segmento de recta, segundo a direcção *y*.
- *deepi*, do tipo real, profundidade (coordenada *z*) do ponto inicial do segmento de recta.
- *xf*, do tipo inteiro, coordenada do ponto final do segmento de recta, segundo a direcção *x*.
- *yf*, do tipo inteiro, coordenada do ponto final do segmento de recta, segundo a direcção *y*.
- *deepf*, do tipo real, profundidade (coordenada *z*) do ponto final do segmento de recta.
- *maxdeep*, do tipo real, máxima profundidade a considerar na representação da profundidade. Este parâmetro é utilizado na definição da relação profundidade/cinzento.
- *mindeep*, do tipo real, mínima profundidade a considerar na representação da profundidade. Este parâmetro é utilizado na definição da relação profundidade/cinzento.
- *lowlevel*, do tipo inteiro, número de elementos inferiores e superiores que não são utilizados na gama de cinzento disponível.

Este módulo retorna *0* como indicação de que a sua execução decorreu em normalidade, e não apresenta qualquer tipo de restrições quanto à sua utilização.

3.1.8 - Módulo compute_mpp

Este módulo é responsável pela determinação da matriz homogénea da transformação geométrica 3D do sistema de coordenadas mundo para o sistema câmara. Esta transformação geométrica é a composição das transformações 3D: translação, rotação e perspectiva, sendo esta a respectiva ordem.

Como parâmetros de entrada para este módulo têm-se:

- *mpp[ ][4]*, do tipo real, matriz homogénea da transformação geométrica 3D do sistema de coordenadas mundo para o sistema câmara.
- *rot[ ]*, do tipo real, vector que contém os valores dos ângulos de rotação segundo os três eixos principais.
- *trans[ ]*, do tipo real, vector que contém os valores das translações segundo os três eixos principais.
- *f*, do tipo real, distância focal efectiva.

[10] Ver, por exemplo [Foley, 1991].

- *c*, do tipo inteiro; quando igual a *1*, os valores das rotações segundo os três eixos principais são convertidos de grau para radiano.

Este módulo retorna *0* como indicação de que a sua execução decorreu em normalidade, e não apresenta qualquer tipo de restrições quanto à sua utilização.

3.1.9 - Módulo frame_undistorted

Este módulo é responsável pela determinação das coordenadas imagem não distorcida de um dado ponto, a partir das respectivas coordenadas na memória *frame*.

Como parâmetros de entrada para este módulo têm-se:

- *xf*, do tipo inteiro, coordenada na memória *frame* do ponto, segundo a direcção *x*.
- *yf*, do tipo inteiro, coordenada na memória *frame* do ponto, segundo a direcção *y*.
- *xu*, do tipo real, passado por endereço, coordenada imagem não distorcida do ponto, segundo a direcção *x*.
- *yu*, do tipo real, passado por endereço, coordenada imagem não distorcida do ponto, segundo a direcção *y*.
- *k*, do tipo real, factor de distorção radial da lente.
- *dxl*, do tipo real, igual a $s_x d_x$, onde s_x é o factor de incerteza horizontal e d_x é a distância entre centros dos elementos sensores vizinhos na direcção *x*.
- *dy*, do tipo real, distância entre centros dos sensores CCD vizinhos na direcção *y*.
- *cx*, do tipo inteiro, coordenada do centro da imagem na memória *frame*, segundo a direcção *x*.
- *cy*, do tipo inteiro, coordenada do centro da imagem na memória *frame*, segundo a direcção *y*.

Este módulo retorna *0* como indicação de que a sua execução decorreu em normalidade, e não apresenta qualquer tipo de restrições quanto à sua utilização.

3.1.10 - Módulo match_line

Este módulo é responsável pela determinação da linha epipolar num plano imagem de um determinado ponto, a partir das duas matrizes das transformações do sistema de coordenadas mundo para o sistema câmara respectivo, e das suas coordenadas imagem não distorcida no outro plano imagem. O método utilizado é descrito em [Tavares, 1995] e foi inicialmente apresentado em [Mendonça, 1990].

Como parâmetros de entrada para este módulo têm-se:

- *mpp1[ ][4]*, do tipo real, matriz homogénea da transformação geométrica 3D do sistema de coordenadas mundo para o sistema câmara *1*.
- *mpp2[ ][4]*, do tipo real, matriz homogénea da transformação geométrica 3D do sistema de coordenadas mundo para o sistema câmara *2*.
- *px*, do tipo real, coordenada imagem não distorcida do ponto segundo a direcção *x*, no plano imagem da câmara *1*.
- *py*, do tipo real, coordenada imagem não distorcida do ponto segundo a direcção y, no plano imagem da câmara *1*.
- *a1*, do tipo real, passado por endereço, variável de parametrização $y = a1.x + b1$ da linha epipolar, no plano imagem da câmara *2*.

- *b1*, do tipo real, passado por endereço, variável de parametrização $y = a1.x + b1$ da linha epipolar, no plano imagem da câmara *2*.

Resumidamente, o método utilizado consiste em determinar as coordenadas imagem não distorcida, no plano imagem da câmara *2*, de dois pontos determinados sobre a linha de projecção definida pelo centro óptico da câmara *1* e pela imagem não distorcida do ponto dado no plano imagem da mesma câmara. As imagens destes dois pontos, no plano imagem da câmara *2*, definem a linha epipolar pretendida.

Este módulo retorna o valor: *1*, ("Can´t compute the matching line."), o módulo não consegue determinar a linha epipolar; e *0*, como indicação de que a sua execução decorreu em normalidade.

As restrições quanto à sua utilização são as existentes na definição e conceito físico da linha epipolar.

3.1.11 - Módulo stereo_eq

Este módulo é responsável pela determinação das coordenadas 3D de um determinado ponto, a partir das duas matrizes das transformações do sistema de coordenadas mundo para o sistema câmara respectivo, e das suas coordenadas imagem não distorcida nos respectivos planos imagem. O método utilizado é o da triangulação estéreo[11].

Como parâmetros de entrada para este módulo têm-se:

- *mpp1[ ][4]*, do tipo real, matriz homogénea da transformação geométrica 3D do sistema de coordenadas mundo para o sistema câmara *1*.
- *mpp2[ ][4]*, do tipo real, matriz homogénea da transformação geométrica 3D do sistema de coordenadas mundo para o sistema câmara *2*.
- *xu1*, do tipo real, coordenada imagem não distorcida do ponto segundo a direcção *x*, no plano imagem da câmara *1*.
- *yu1*, do tipo real, coordenada imagem não distorcida do ponto segundo a direcção *y*, no plano imagem da câmara *1*.
- *xu2*, do tipo real, coordenada imagem não distorcida do ponto segundo a direcção *x*, no plano imagem da câmara *2*.
- *yu2*, do tipo real, coordenada imagem não distorcida do ponto segundo a direcção *y*, no plano imagem da câmara *2*.
- *a*, do tipo real, passado por endereço, vector das coordenadas 3D obtidas pela implementação.

Durante a sua execução este módulo utiliza os módulos: ***matrix***, ***vector***, ***free_matrix***, ***free_vector*** e ***svdfit***.

Este módulo retorna *0* como indicação de que a sua execução decorreu em normalidade, e tem como restrições quanto à sua utilização as existentes na definição e conceito físico do princípio da triangulação estéreo.

3.1.12 - Módulo svdfit

Este módulo é responsável pela execução da regressão linear de um vector de variáveis independentes a um dado modelo. O método utilizado[12] é o dos mínimos quadrados, utilizando a decomposição em valor único da matriz dos valores observados para as variáveis independentes no modelo. Esta decomposição pode ser consideravelmente mais lenta do que a resolução das equações normais equivalentes; no entanto, tem como grande vantagem, pelo menos teórica, a de corresponder a

[11] Ver, por exemplo [Tavares, 1995].

[12] Ver, por exemplo [Press, 1992].

soluções de aproximação, para as variáveis independentes do modelo, de melhor qualidade (isto é, de menor erro) [Press, 1992].

Como parâmetros de entrada para este módulo têm-se:

- *xin*, do tipo real, passado por endereço, matriz dos valores observados para as variáveis independentes no modelo.
- *yin*, do tipo real, passado por endereço, vector dos valores observados para as variáveis dependentes no modelo.
- *ndata*, do tipo inteiro, número de valores observados para as variáveis independentes no modelo; isto é, o número de linhas da matriz *xin*.
- *ain*, do tipo real, passado por endereço, vector das variáveis independentes do modelo a determinar.
- *ma*, do tipo inteiro, número de variáveis independentes no modelo; isto é, o número de colunas da matriz *xin* e de linhas do vector *yin*.

Durante a sua execução este módulo utiliza os módulos: ***matrix***, ***vector***, ***svbksb***, ***svdcmp***, ***free_matrix*** e ***free_vector***.

Este módulo não tem qualquer tipo de restrições quanto à sua utilização.

3.1.13 - Módulo svbksb

Este módulo é responsável pela resolução de um sistema linear $A.X = B$.

Como parâmetros de entrada para este módulo têm-se:

- *u*, do tipo real, passado por endereço, matriz $U[1...m][1...n]$ resultante da decomposição em valor único da matriz *A*, efectuada pelo módulo ***svdcmp***.
- *w[ ]*, do tipo real, vector dos valores da diagonal da matriz $W[1...n][1...n]$; isto é, dos valores únicos resultantes da decomposição efectuada pelo módulo ***svdcmp***.
- *v*, do tipo real, passado por endereço, matriz $V[1...n][1...n]$ resultante da decomposição em valor único da matriz *A*, efectuada pelo módulo ***svdcmp***.
- *m*, do tipo inteiro, número de linhas da matriz *A*.
- *n*, do tipo inteiro, número de colunas da matriz *A*.
- *b[ ]*, do tipo real, vector *B*.
- *x[ ]*, do tipo real, vector das soluções a determinar.

Durante a sua execução, este módulo utiliza os módulos: ***vector*** e ***free_vector***, e não tem qualquer tipo de restrições quanto à sua utilização.

3.1.14 - Módulo svdcmp

Este módulo é responsável pela decomposição em valor único $U.W.V^T$ de uma matriz *A* de dimensões $m \times n$, sendo *U* uma matriz de colunas ortogonais, de dimensões $m \times n$, *W* uma matriz diagonal de dimensões $n \times n$ cujos elementos são nulos ou maiores do que zero (os valores únicos), e *V* uma matriz ortogonal de dimensões $n \times n$. Em problemas numéricos, esta decomposição apresenta vantagens sobre a eliminação de Gauss e sobre a decomposição *LU* [Press, 1992], sendo a decomposição de utilização preferencial para a resolução de problemas de aproximação linear pelo método dos mínimos quadrados.

Como parâmetros de entrada para este módulo têm-se:

- *a*, do tipo real, passado por endereço, matriz $A[1...m][1...n]$, a decompor pelo módulo.
- *m*, do tipo inteiro, número de linhas da matriz *A*.
- *n*, do tipo inteiro, número de colunas da matriz *A*.
- *w[ ]*, do tipo real, vector dos valores da diagonal da matriz $W[1...n][1...n]$; isto é, os valores únicos resultantes da decomposição.
- *v*, passado por endereço, do tipo inteiro, matriz $V[1...n][1...n]$ resultante da decomposição.

Após a execução deste módulo, a matriz *A* passa a corresponder à matriz $U[1...m][1...n]$.

Durante a sua execução este módulo utiliza os módulos: ***vector*** e ***free_vector***, e retorna *1*, ("No convergence in 30 svdcmp iterations."), como indicação de que o módulo não consegue convergir com êxito após trinta iterações.

Como restrições à sua utilização este módulo apresenta o número de iterações necessárias para a convergência ser obtida com êxito.

3.1.15 - Módulo create_line

Este módulo é responsável pela alocação da memória necessária para um dado elemento do modelo de entidades 2D; assim, para o elemento pretendido, executa a alocação de memória para uma estrutura que contém várias características da respectivo segmento de recta e para as diversas matrizes utilizadas para definir o estado do referido elemento. Este estado será utilizado pelos filtros de Kalman no seguimento.

Durante a sua execução, este módulo utiliza os módulos: ***memory_get*** e ***create_matrix***.

Este módulo não apresenta qualquer tipo de restrições à sua utilização, e retorna um apontador para o respectivo elemento.

3.1.16 - Módulo delete_line

Este módulo é responsável pela libertação da memória utilizada por um dado elemento do modelo de entidades 2D.

Como parâmetro de entrada para este módulo tem-se:

- *el*, do tipo *Line*, passado por endereço, apontador do elemento do modelo de entidades 2D, cuja memória se pretende libertar.

Este módulo, durante a sua execução, utiliza o módulo ***delete_matrix***, e não apresenta qualquer tipo de restrições quanto à sua utilização.

3.1.17 - Módulo print_line

Este módulo é responsável pela apresentação das características de um dado elemento do modelo de entidades 2D.

Como parâmetro de entrada para este módulo tem-se:

- *el*, do tipo *Line*, passado por endereço, apontador do elemento do modelo de entidades 2D cujas características se pretende que sejam apresentadas.

Este módulo não apresenta qualquer tipo de restrições quanto à sua utilização.

3.1.18 - Módulo create_mpp

Este módulo é responsável pela alocação de memória necessária para um dado elemento da lista ligada linear que contém as matrizes das transformações do sistema de coordenadas mundo para o sistema câmara, para cada posição e orientação da câmara a considerar.

Este módulo, durante a sua execução, utiliza o módulo ***memory_get***; não apresenta qualquer tipo de restrições à sua utilização, e retorna um apontador para o respectivo elemento.

3.1.19 - Módulo delete_mpp

Este módulo é responsável pela libertação da memória utilizada por um dado elemento da lista ligada linear que contém as matrizes das transformações do sistema de coordenadas mundo para o sistema câmara, para cada posição e orientação da câmara a considerar.

Como parâmetro de entrada para este módulo tem-se:

- *el*, do tipo *Mpp*, passado por endereço, apontador do elemento da lista ligada linear que contém as matrizes das transformações do sistema de coordenadas mundo para o sistema câmara, para cada posição e orientação da câmara a considerar, cuja memória utilizada se pretende libertar.

Este módulo não apresenta qualquer tipo de restrições quanto à sua utilização.

3.1.20 - Módulo InitStateSpace

Este módulo é responsável pela reserva de memória para as matrizes utilizadas no seguimento pelos filtros de Kalman.

Durante a sua execução, este módulo utiliza o módulo ***create_matrix***. Para a sua utilização, este módulo não apresenta qualquer tipo de restrições.

3.1.21 - Módulo RemoveStateSpace

Este módulo é responsável pela libertação da memória utilizada pelas matrizes necessárias no seguimento pelos filtros de Kalman.

Durante a sua execução, este módulo utiliza o módulo ***delete_matrix***. Para a sua utilização, este módulo não apresenta qualquer tipo de restrições.

3.1.22 - Módulo statePredictCovrpos

Este módulo é responsável pela determinação da variância da posição do ponto médio prevista pelo respectivo filtro de Kalman, para um dado elemento do modelo de entidades 2D.

Como parâmetros de entrada para este módulo têm-se:

- *cstate*, do tipo *State*, passado por endereço, apontador para o estado actual da entidade em causa.
- *pstate*, do tipo *State*, passado por endereço, apontador para o estado previsto da entidade em causa.

Este módulo, durante a sua execução, utiliza os módulos: ***matrix_wsqr*** e ***matrix_opr***, e não apresenta qualquer tipo de restrições quanto à sua utilização.

3.1.23 - Módulo statePredict

Este módulo é responsável pela determinação do estado previsto para um dado elemento do modelo de entidades 2D, utilizando para o efeito os respectivos filtros de Kalman.

Como parâmetros de entrada para este módulo têm-se:

- *cstate*, do tipo *State*, passado por endereço, apontador para o estado actual da entidade em causa.

- *pstate*, do tipo *State*, passado por endereço, apontador para o estado previsto da entidade em causa, a determinar.

Este módulo, durante a sua execução, utiliza os módulos: ***matrix_mult***, ***matrix_wsqr*** e ***matrix_opr***, e não apresenta qualquer tipo de restrições quanto à sua utilização.

3.1.24 - Módulo stateUpdate

Este módulo é responsável pela determinação do estado actualizado para um dado elemento do modelo de entidades 2D, utilizando para o efeito os respectivos filtros de Kalman.

Como parâmetros de entrada para este módulo têm-se:

- *pstate*, do tipo *State*, passado por endereço, apontador para o estado previsto da entidade em causa.
- *mstate*, do tipo *State*, passado por endereço, apontador para o estado medido da entidade em causa.
- *ustate*, do tipo *State*, passado por endereço, apontador para o estado actualizado da entidade em causa a determinar.

Este módulo durante a sua execução utiliza os módulos: ***matrix_mult***, ***matrix_invert***, ***matrix_wsqr***, e ***matrix_opr***, e não apresenta qualquer tipo de restrições quanto à sua utilização.

3.1.25 - Módulo stateDistance

Este módulo é responsável pela determinação da distância normalizada de Mahalanobis para a posição do ponto médio entre dois estados de um dado elemento do modelo de entidades 2D.

Como parâmetros de entrada para este módulo têm-se:

- *s1*, do tipo *State*, passado por endereço, apontador para um dos estados da entidade em causa.
- *s2*, do tipo *State*, passado por endereço, apontador para o outro estado da entidade em causa.

Este módulo durante a sua execução utiliza os módulos: ***matrix_opr***, ***matrix_invert*** e ***matrix_wsqr***, e não apresenta qualquer tipo de restrições quanto à sua utilização.

3.1.26 - Módulo copyState

Este módulo é responsável pela execução da cópia do conteúdo de um estado para um outro estado.

Como parâmetros de entrada para este módulo têm-se:

- *s1*, do tipo *State*, passado por endereço, apontador para o estado cujo conteúdo se pretende copiar.
- *s2*, do tipo *State*, passado por endereço, apontador para o estado que deverá ficar com o conteúdo copiado.

Este módulo, durante a sua execução, utiliza o módulo ***matrix_opr***, e não apresenta qualquer tipo de restrições quanto à sua utilização.

3.1.27 - Módulo create_matrix

Este módulo é responsável pela alocação de memória para uma dada matriz do tipo real.

Como parâmetros de entrada para este módulo têm-se:

- *nr*, do tipo inteiro, número de linhas da matriz pretendida.

➲ *nc*, do tipo inteiro, número de colunas da matriz pretendida.

Este módulo, durante a sua execução, utiliza o módulo ***memory_get***, não apresenta qualquer tipo de restrições quanto à sua utilização, e retorna o apontador para a matriz pretendida.

3.1.28 - Módulo delete_matrix

Este módulo é responsável pela libertação da memória utilizada para uma dada matriz do tipo real.

Como parâmetro de entrada para este módulo tem-se:

➲ *mx*, do tipo *Matrix*, passado por endereço, apontador para a matriz em causa.

Este módulo não apresenta qualquer tipo de restrições quanto à sua utilização.

3.1.29 - Módulo print_matrix

Este módulo é responsável pela apresentação de uma dada matriz do tipo real, segundo o formato da aplicação MATLAB da MathWorks Inc.

Como parâmetros de entrada para este módulo têm-se:

➲ *mx*, do tipo *Matrix*, passado por endereço, apontador para a matriz em causa.

➲ *name*, do tipo caracter, designação que se pretende para a matriz.

Este módulo não apresenta qualquer tipo de restrições quanto à sua utilização.

3.1.30 - Módulo matrix_opr

Este módulo é responsável pela execução da operação de matrizes $M_{RES} = sign * M_1 + fact * M_2$, onde:

- M_{RES}, M_1 e M_2 são matrizes.
- *sign* e *fact* são escalares, com valores diferentes consoante a operação desejada.

Como parâmetros de entrada para este módulo têm-se:

➲ *m1*, do tipo *Matrix*, passado por endereço, apontador para a matriz M_1.

➲ *m2*, do tipo *Matrix*, passado por endereço, apontador para a matriz M_2.

➲ *mres*, do tipo *Matrix*, passado por endereço, apontador para a matriz M_{RES}.

➲ *d*, do tipo real, valor do escalar a considerar para a operação pretendida.

➲ *op*, do tipo *Mopr*, tipo de operação pretendida; assim, se *op* é igual a:

- *EQU*, então: *sign* é igual a 1 e *fact* é igual a 0.
- *ADD*, então: *sign* é igual a 1 e *fact* é igual a 1.
- *DIF*, então: *sign* é igual a 1 e *fact* é igual a -1.
- *NEG*, então: *sign* é igual a -1 e *fact* é igual a 0.
- *MSC*, então: *sign* é igual a *d* e *fact* é igual a 0.

➲ *tr*, do tipo *Mopr*; quando igual a *TRP*, a operação é realizada considerando a matriz M_1 transposta.

O módulo determina automaticamente as dimensões das matrizes M_1 e M_2 que deve considerar.

Este módulo não apresenta qualquer tipo de restrições quanto à sua utilização e retorna:

- 0, como indicação de que a sua execução decorreu em normalidade.
- 1, como indicação de que pelo menos uma das matrizes especificadas não está definida.

3.1.31 - Módulo matrix_mult

Este módulo é responsável pela execução da multiplicação de duas matrizes: M_1 e M_2.

Como parâmetros de entrada para este módulo têm-se:

- *m1*, do tipo *Matrix*, passado por endereço, apontador para a matriz M_1.
- *m2*, do tipo *Matrix*, passado por endereço, apontador para a matriz M_2.
- *mmult*, do tipo *Matrix*, passado por endereço, apontador para a matriz resultante M_{MMULT}.
- *op*, do tipo *Mopr*, tipo de operação pretendida; assim, se *op* é igual a:
  - *EQU*, então a operação de matrizes executada pelo módulo é $M_{MMULT} = M_1 * M_2$.
  - *ADD*, então a operação de matrizes executada pelo módulo é $M_{MMULT} = M_{MMULT} + M_1 * M_2$.
  - *DIF*, então a operação de matrizes executada pelo módulo é $M_{MMULT} = M_{MMULT} - M_1 * M_2$.
- *tr*, do tipo *Mopr*; quando igual a *TRP*, a operação é realizada considerando a matriz M_2 transposta.

O módulo determina automaticamente as dimensões das matrizes M_1 e M_2 que deve considerar.

Este módulo não apresenta qualquer tipo de restrições quanto à sua utilização e retorna:

- 0, como indicação de que a sua execução decorreu em normalidade.
- 1, como indicação de que pelo menos uma das matrizes especificadas não está definida.

3.1.32 - Módulo matrix_wsqr

Este módulo é responsável pela execução da multiplicação de duas matrizes: M_1 e M_2^T.

Como parâmetros de entrada para este módulo têm-se:

- *m1*, do tipo *Matrix*, passado por endereço, apontador para a matriz M_1.
- *m2*, do tipo *Matrix*, passado por endereço, apontador para a matriz M_2.
- *msqr*, do tipo *Matrix*, passado por endereço, apontador para a matriz resultante M_{MSQR}.
- *op*, do tipo *Mopr*, tipo de operação pretendida; assim, se *op* é igual a:
  - *EQU*, então a operação de matrizes executada pelo módulo é $M_{MMULT} = M_1 * M_2^T$.
  - *ADD*, então a operação de matrizes executada pelo módulo é $M_{MMULT} = M_{MMULT} + M_1 * M_2^T$.
  - *DIF*, então a operação de matrizes executada pelo módulo é $M_{MMULT} = M_{MMULT} - M_1 * M_2^T$.

O módulo determina automaticamente as dimensões das matrizes M_1 e M_2 que deve considerar.

Este módulo não apresenta qualquer tipo de restrições quanto à sua utilização e retorna:

- 0, como indicação de que a sua execução decorreu em normalidade.
- 1, como indicação de que pelo menos uma das matrizes especificadas não está definida.

3.1.33 - Módulo matrix_invert

Este módulo é responsável pela determinação da matriz inversa de uma dada matriz.

Como parâmetro de entrada para este módulo tem-se:

- *mx*, do tipo *Matrix*, passado por endereço, apontador para a matriz que se pretende inverter.

O módulo só considera a submatriz de dimensões *2×2* superior esquerda da matriz de entrada *mx*. No final da execução, a submatriz de dimensões *2×2* superior esquerda da matriz de entrada *mx* corresponde à matriz invertida determinada.

Este módulo apresenta como restrição quanto à sua utilização o facto de a matriz que se pretende inverter deve ter, necessariamente, determinante não nulo, retornando *0*, como indicação de que a sua execução decorreu em normalidade, e *1* ("Can't make the matrix inversion."), como indicação de que o módulo não pode determinar a matriz invertida desejada.

3.1.34 - Módulo matrix_eig

Este módulo é responsável pela determinação dos valores e vectores próprios de uma dada matriz.

Como parâmetros de entrada para este módulo têm-se:

- *mx*, do tipo *Matrix*, passado por endereço, apontador para a matriz da qual se pretende determinar os valores e vectores próprios.
- *vec*, do tipo *Matrix*, passado por endereço, apontador para a matriz dos vectores próprios determinados pelo módulo.
- *val*, do tipo *Matrix*, passado por endereço, apontador para a matriz cujos elementos diagonais são os valores próprios determinados pelo módulo.

O módulo só considera a submatriz de dimensões *2×2* superior esquerda da matriz de entrada *mx*, apresentando, como única restrição quanto à sua utilização, o facto de a matriz de que se pretende determinar os valores e vectores próprios dever ser, necessariamente, definida positivamente.

Este módulo retorna *0*, como indicação de que a sua execução decorreu em normalidade, e *1* ("Can't compute the matrix eigenvalues and eigenvectors."), como indicação de que o módulo não pode determinar os valores e vectores próprios da matriz em causa.

3.1.35 - Módulo create_list

Este módulo é responsável pela reserva de memória para uma dada lista ligada.

Como parâmetro de entrada para este módulo tem-se:

- *lt*, do tipo *Ltype*, tipo de lista pretendida. Deve ser igual a:
  - *LINEAR*, para lista ligada do tipo linear.
  - *CIRCULAR*, para lista ligada do tipo circular.

Este módulo durante a sua execução, utiliza o módulo ***memory_get***; não apresenta qualquer tipo de restrições quanto à sua utilização, e retorna o apontador do tipo *list* para a lista pretendida.

3.1.36 - Módulo delete_list

Este módulo é responsável pela libertação da memória utilizada com uma dada lista ligada.

Como parâmetros de entrada para este módulo têm-se:

- *l*, do tipo *List*, lista cuja memória se pretende libertar.
- *func*, do tipo função, passado por endereço, função para o módulo utilizar para libertar a memória reservada para o campo de informação de cada elemento da lista ligada.

Este módulo não apresenta qualquer tipo de restrições quanto à sua utilização.

3.1.37 - Módulo insert_cell_list

Este módulo é responsável pela inserção de uma dado elemento numa dada lista ligada.

Como parâmetros de entrada para este módulo têm-se:

- *info*, do tipo *caracter*, passado por endereço, campo de informação do elemento que se pretende inserir na lista ligada.
- *lpos*, do tipo *List*, referência ao elemento da lista ligada, depois ou antes do qual se deve inserir o elemento pretendido.
- *sw*, do tipo *Lpos*, posição do novo elemento na lista ligada, relativamente ao elemento referenciado por *lpos* na respectiva lista. Se *lpos* for igual a *AFTER*, o novo elemento é inserido depois do elemento referenciado por *lpos*; se for igual a *BEFORE*, o novo elemento é inserido antes do elemento referenciado por *lpos*.

Este módulo não apresenta qualquer tipo de restrições quanto à sua utilização, e retorna um apontador do tipo *list* para a respectiva lista.

3.1.38 - Módulo delete_cell_list

Este módulo é responsável pela remoção de uma dado elemento numa dada lista ligada.

Como parâmetros de entrada para este módulo têm-se:

- *c*, do tipo *cell*, passado por endereço, elemento que se pretende remover da respectiva lista ligada.
- *l*, do tipo *List*, lista ligada da qual se deve remover o elemento pretendido.
- *func*, do tipo função, passado por endereço, função para o módulo utilizar para libertar a memória reservada para o campo de informação de cada elemento da lista ligada.

Este módulo não apresenta qualquer tipo de restrições quanto à sua utilização, e retorna um apontador do tipo *list* para a respectiva lista.

3.1.39 - Módulo memory_get

Este módulo é responsável pela reserva de uma dada quantidade de memória.

Como parâmetros de entrada para este módulo tem-se:

- *size, do tipo inteiro*, quantidade de memória pretendida.

Este módulo apresenta como restrição quanto à sua utilização a existência de memória disponível, e retorna:

- Um apontador do tipo *caracter* para a memória desejada.
- Uma mensagem de erro, com identificação do módulo chamador, quando não existe memória disponível.

3.1.40 - Módulo vector

Este módulo é responsável pela reserva de memória para um dado vector.

Como parâmetros de entrada para este módulo têm-se:

- *nl*, do tipo inteiro, índice para o primeiro elemento do vector.
- *nh*, do tipo inteiro, índice para o último elemento do vector.

Este módulo apresenta como restrição quanto à sua utilização a existência de memória disponível, e retorna:

- Um apontador, do tipo real, para o vector.
- 1, ("Allocation failure in vector( )."), quando o módulo não consegue reservar a quantidade de memória necessária.

3.1.41 - Módulo matrix

Este módulo é responsável pela reserva de memória para uma dada matriz.

Como parâmetros de entrada para este módulo têm-se:

- *nrl*, do tipo inteiro, índice para a primeira linha da matriz.
- *nrh*, do tipo inteiro, índice para a última linha da matriz.
- *ncl*, do tipo inteiro, índice para a primeira coluna da matriz.
- *nch*, do tipo inteiro, índice para a última coluna da matriz.

Este módulo apresenta como restrição quanto à sua utilização a existência de memória disponível, e retorna:

- Um apontador, do tipo real, para a matriz.
- Os valores:
  - 1, ("Allocation failure 1 in matrix( )."), o módulo não consegue reservar a quantidade de memória necessária.
  - 2, ("Allocation failure 2 in matrix( )."), o módulo não consegue reservar a quantidade de memória necessária.

3.1.42 - Módulo free_vector

Este módulo é responsável pela libertação da memória utilizada por um dado vector.

Como parâmetros de entrada para este módulo têm-se:

- *v*, do tipo real, passado por endereço, apontador do vector.
- *nl*, do tipo inteiro, índice para o primeiro elemento do vector.
- *nh*, do tipo inteiro, índice para o último elemento do vector.

Este módulo não apresenta qualquer tipo de restrições quanto à sua utilização.

3.1.43 - Módulo free_matrix

Este módulo é responsável pela libertação da memória utilizada por uma dada matriz.

Como parâmetros de entrada para este módulo têm-se:

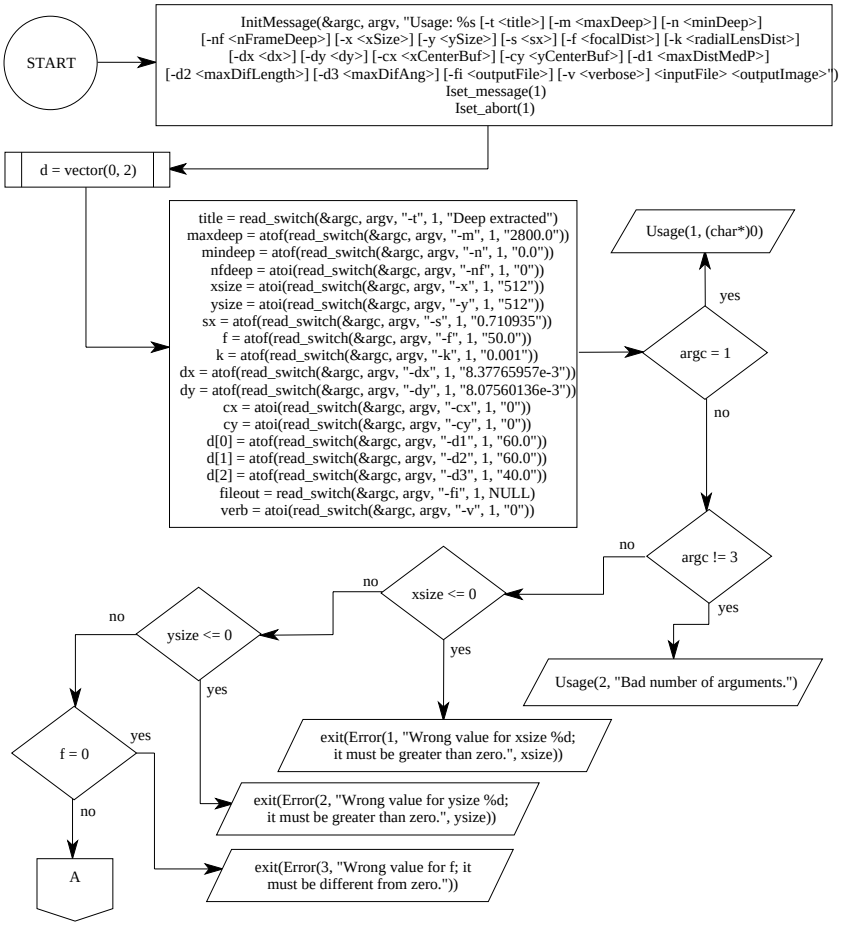
- *m*, do tipo real, passado por endereço, apontador da matriz.
- *nrl*, do tipo inteiro, índice para a primeira linha da matriz.
- *nrh*, do tipo inteiro, índice para a última linha da matriz.
- *ncl*, do tipo inteiro, índice para a primeira coluna da matriz.
- *nch*, do tipo inteiro, índice para a última coluna da matriz.

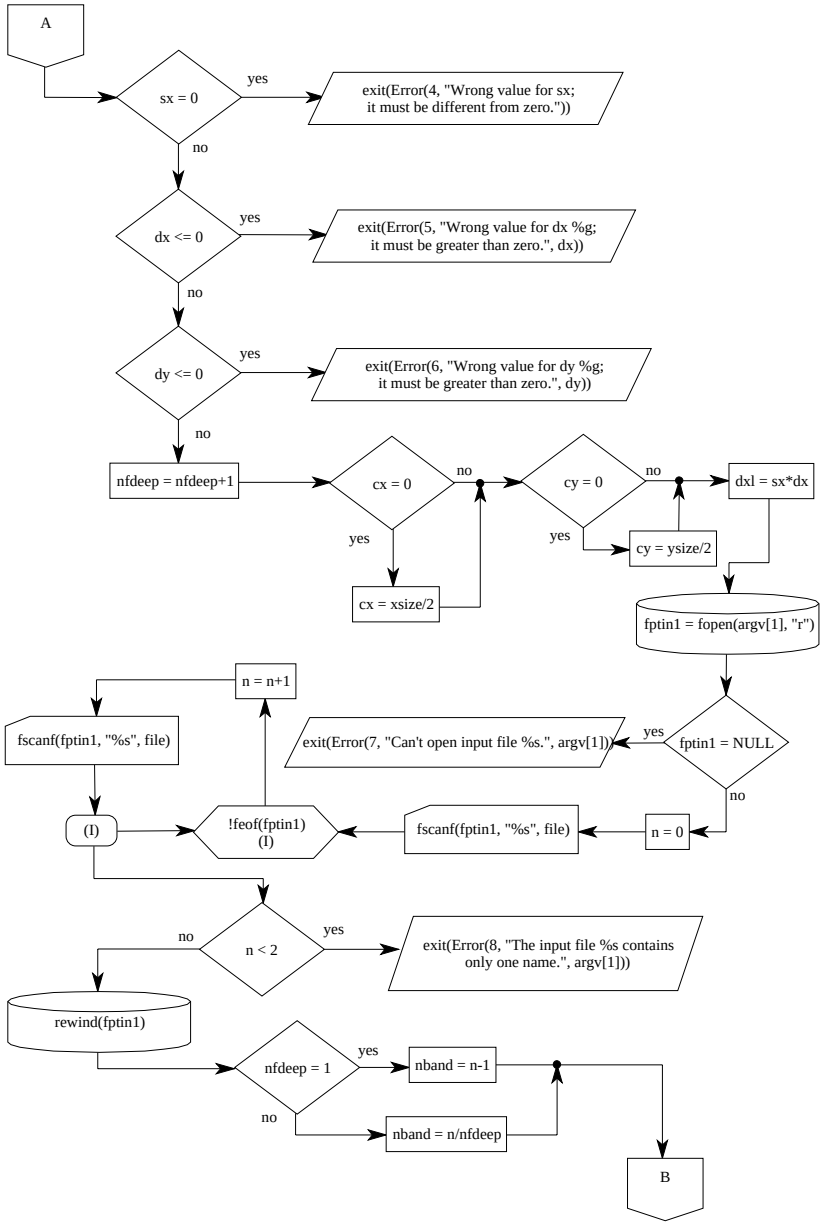
Este módulo não apresenta qualquer tipo de restrições quanto à sua utilização.

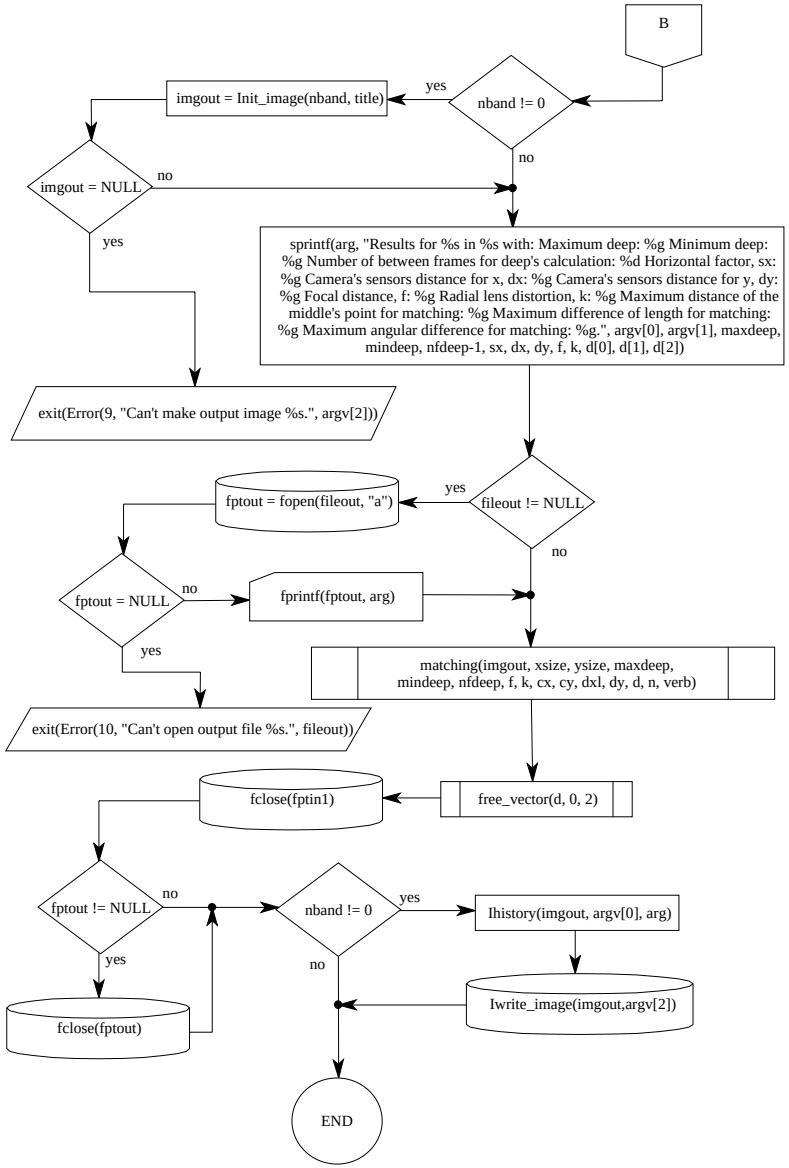
3.2 - Fluxogramas dos diferentes módulos

Apresentam-se, de seguida, os fluxogramas dos módulos que constituem a implementação ***deep***.

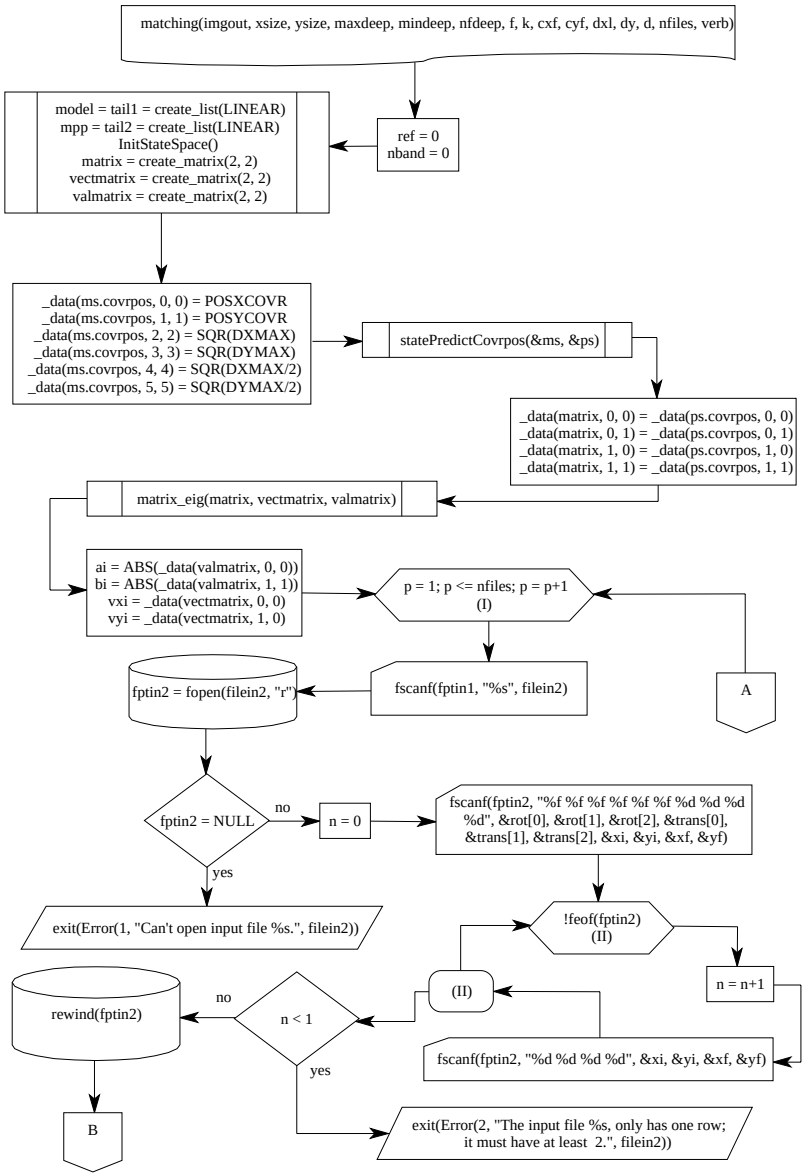
3.2.1 - Módulo principal

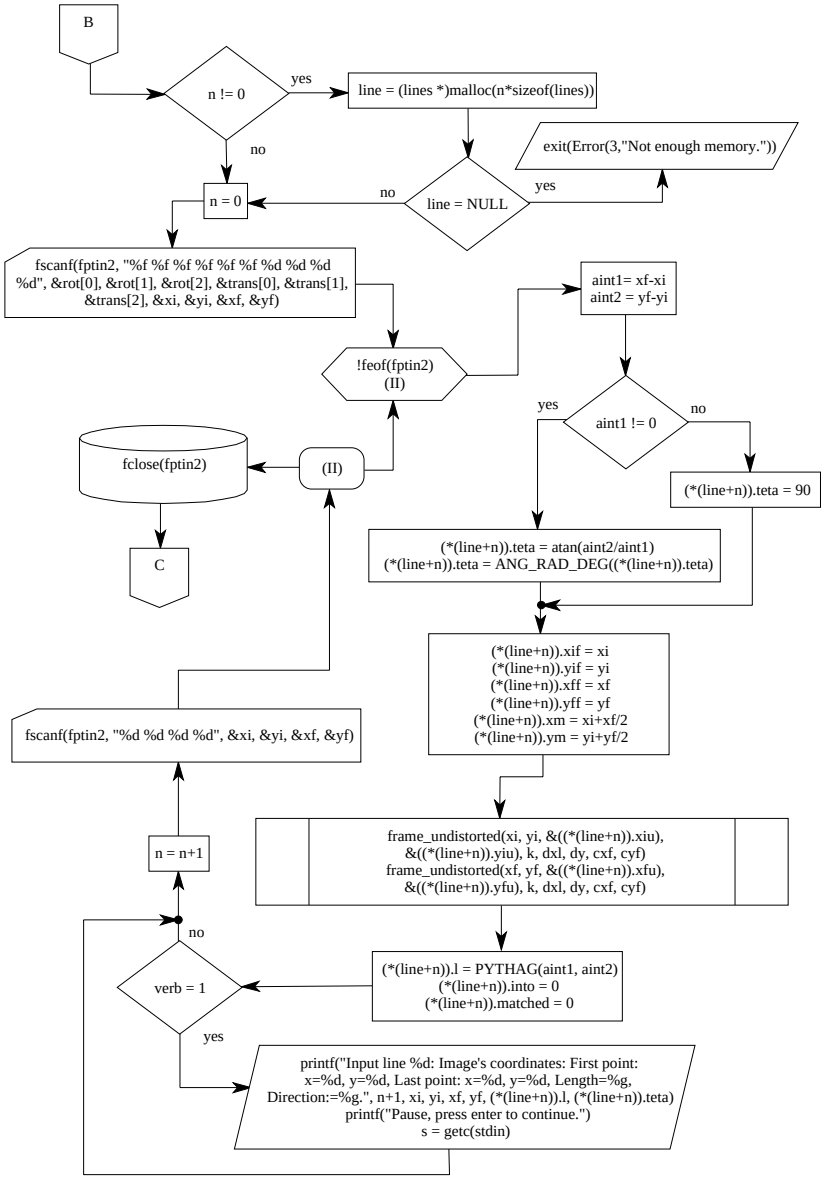


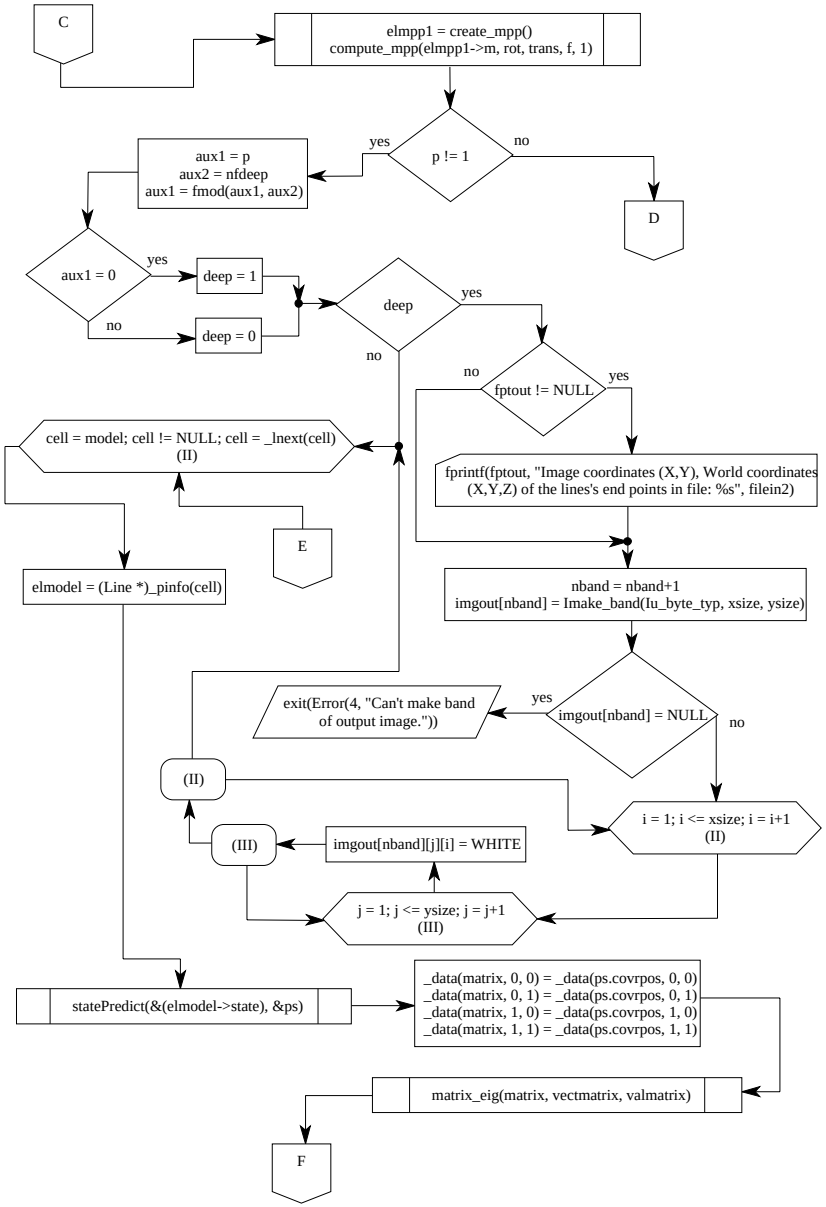

A
sx = 0
yes
exit(Error(4, "Wrong value for sx; it must be different from zero."))
no
dx <= 0
yes
exit(Error(5, "Wrong value for dx %g; it must be greater than zero.", dx))
no
dy <= 0
yes
exit(Error(6, "Wrong value for dy %g; it must be greater than zero.", dy))
no
nfdeep = nfdeep+1
cx = 0
no
yes
cx = xsize/2
cy = 0
no
yes
cy = ysize/2
dxl = sx*dx
fptin1 = fopen(argv[1], "r")
fptin1 = NULL
yes
exit(Error(7, "Can't open input file %s.", argv[1]))
no
n = 0
fscanf(fptin1, "%s", file)
!feof(fptin1)
(I)
n = n+1
(I)
n < 2
no
yes
exit(Error(8, "The input file %s contains only one name.", argv[1]))
rewind(fptin1)
nfdeep = 1
yes
nband = n-1
no
nband = n/nfdeep
B

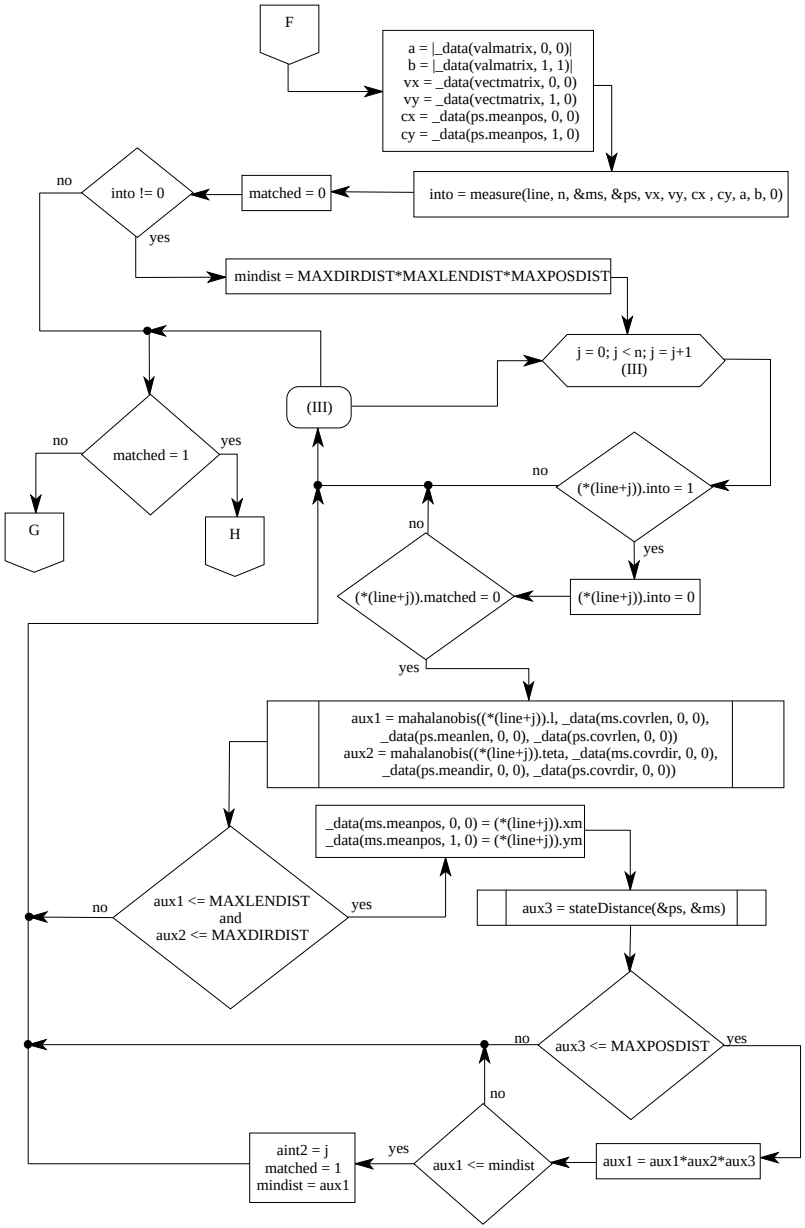

B
nband != 0
yes
imgout = Init_image(nband, title)
no
imgout = NULL
no
yes
exit(Error(9, "Can't make output image %s.", argv[2]))
sprintf(arg, "Results for %s in %s with: Maximum deep: %g Minimum deep: %g Number of between frames for deep's calculation: %d Horizontal factor, sx: %g Camera's sensors distance for x, dx: %g Camera's sensors distance for y, dy: %g Focal distance, f: %g Radial lens distortion, k: %g Maximum distance of the middle's point for matching: %g Maximum difference of length for matching: %g Maximum angular difference for matching: %g.", argv[0], argv[1], maxdeep, mindeep, nfdeep-1, sx, dx, dy, f, k, d[0], d[1], d[2])
fileout != NULL
yes
fptout = fopen(fileout, "a")
no
fptout = NULL
no
fprintf(fptout, arg)
yes
exit(Error(10, "Can't open output file %s.", fileout))
matching(imgout, xsize, ysize, maxdeep, mindeep, nfdeep, f, k, cx, cy, dxl, dy, d, n, verb)
free_vector(d, 0, 2)
fclose(fptin1)
fptout != NULL
no
yes
fclose(fptout)
nband != 0
yes
no
Ihistory(imgout, argv[0], arg)
Iwrite_image(imgout,argv[2])
END

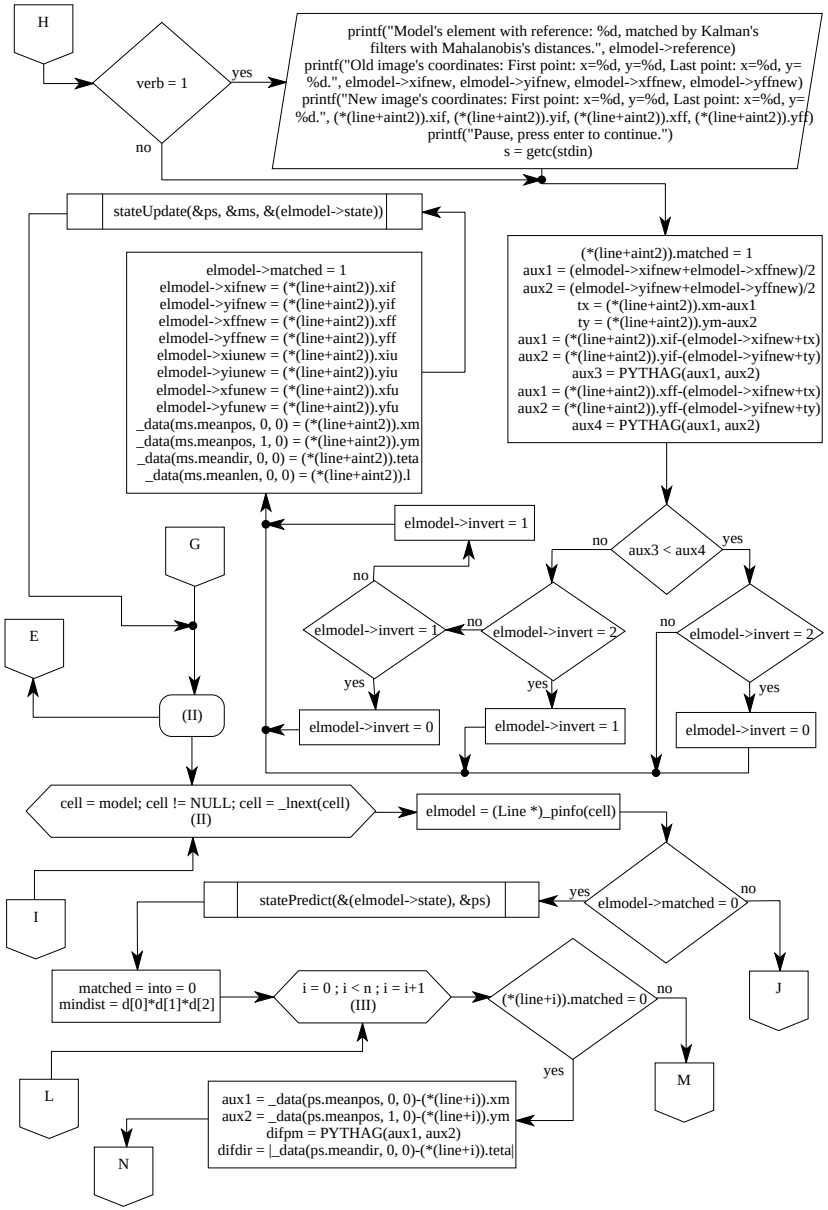
3.2.2 - Módulo matching

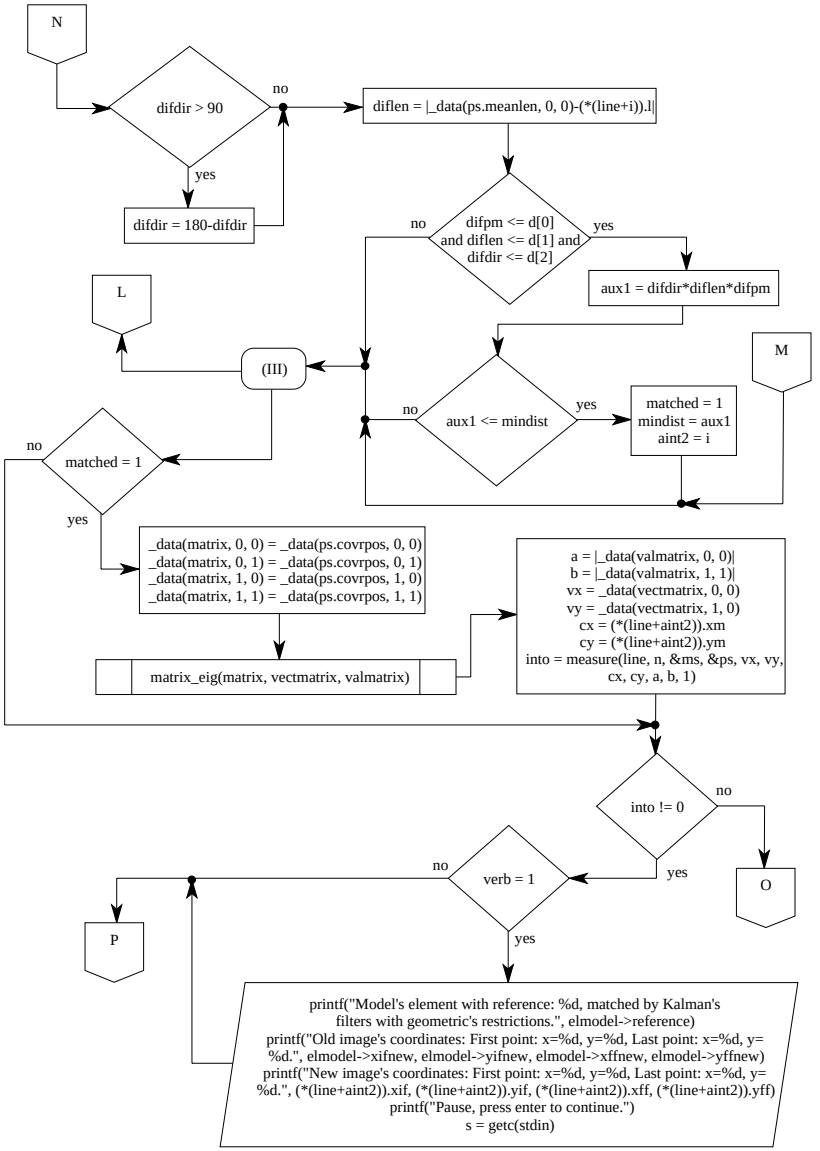


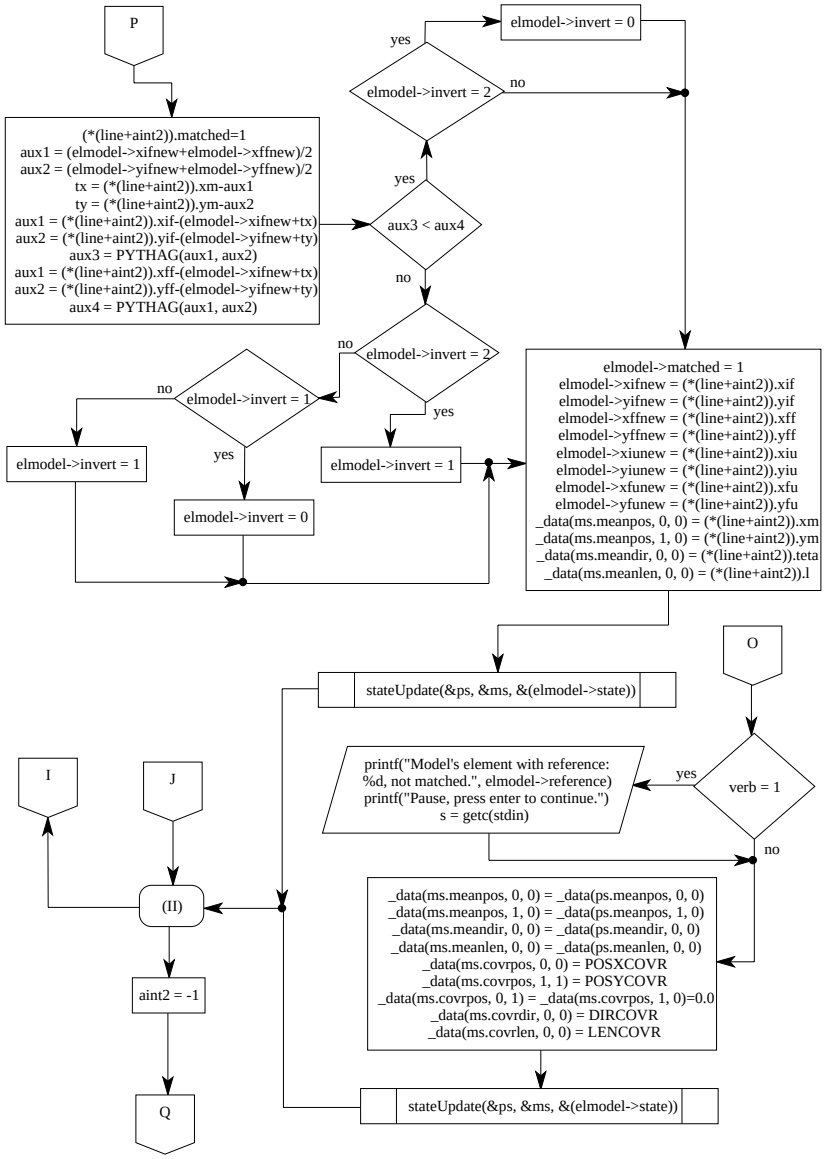


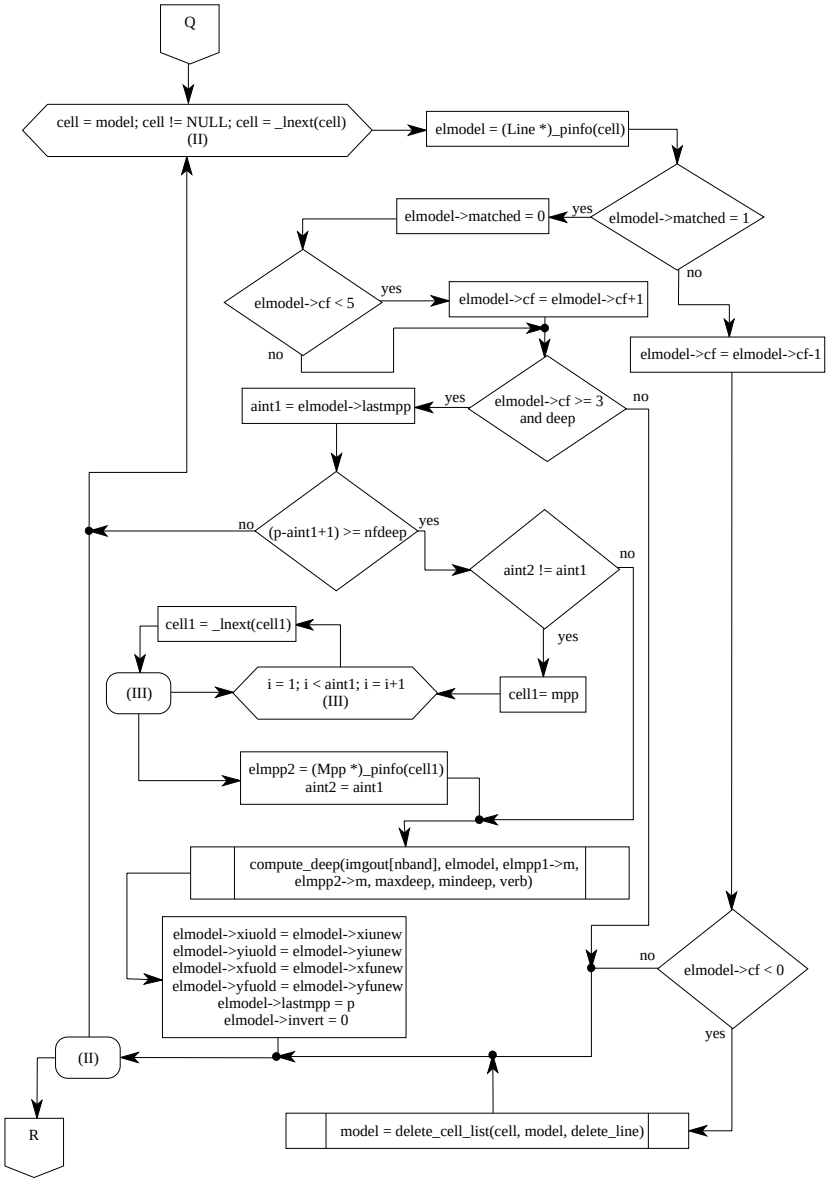

C
elmpp1 = create_mpp()
compute_mpp(elmpp1->m, rot, trans, f, 1)
p != 1
yes
no
aux1 = p
aux2 = nfdeep
aux1 = fmod(aux1, aux2)
D
aux1 = 0
yes
no
deep = 1
deep = 0
deep
yes
no
fptout != NULL
no
yes
cell = model; cell != NULL; cell = _lnext(cell)
(II)
E
fprintf(fptout, "Image coordinates (X,Y), World coordinates (X,Y,Z) of the lines's end points in file: %s", filein2)
elmodel = (Line *)_pinfo(cell)
nband = nband+1
imgout[nband] = Imake_band(Iu_byte_typ, xsize, ysize)
imgout[nband] = NULL
yes
no
exit(Error(4, "Can't make band of output image."))
(II)
i = 1; i <= xsize; i = i+1
(II)
(III)
imgout[nband][j][i] = WHITE
j = 1; j <= ysize; j = j+1
(III)
statePredict(&(elmodel->state), &ps)
_data(matrix, 0, 0) = _data(ps.covrpos, 0, 0)
_data(matrix, 0, 1) = _data(ps.covrpos, 0, 1)
_data(matrix, 1, 0) = _data(ps.covrpos, 1, 0)
_data(matrix, 1, 1) = _data(ps.covrpos, 1, 1)
matrix_eig(matrix, vectmatrix, valmatrix)
F

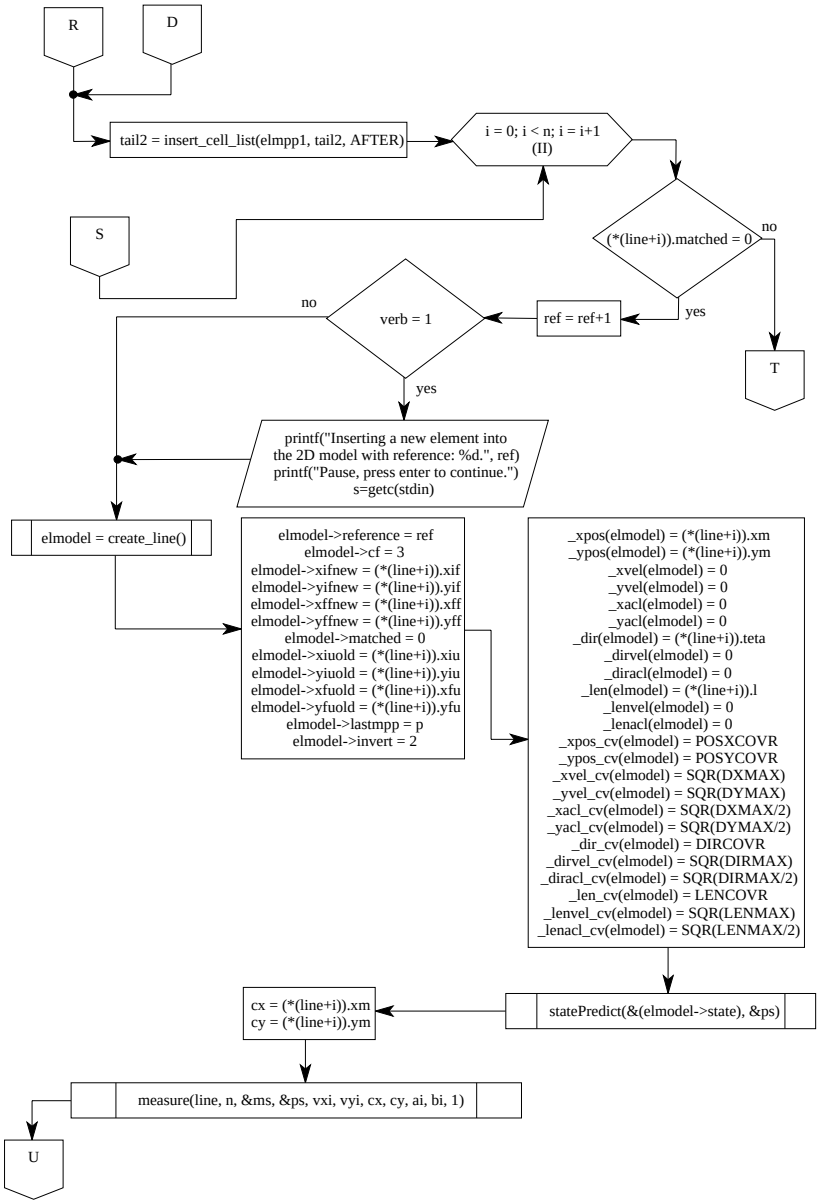

F
a = |_data(valmatrix, 0, 0)|
b = |_data(valmatrix, 1, 1)|
vx = _data(vectmatrix, 0, 0)
vy = _data(vectmatrix, 1, 0)
cx = _data(ps.meanpos, 0, 0)
cy = _data(ps.meanpos, 1, 0)
into = measure(line, n, &ms, &ps, vx, vy, cx , cy, a, b, 0)
matched = 0
into != 0
no
yes
mindist = MAXDIRDIST*MAXLENDIST*MAXPOSDIST
j = 0; j < n; j = j+1
(III)
(III)
matched = 1
no
yes
G
H
(*(line+j)).into = 1
no
yes
(*(line+j)).into = 0
(*(line+j)).matched = 0
no
yes
aux1 = mahalanobis((*(line+j)).l, _data(ms.covrlen, 0, 0), _data(ps.meanlen, 0, 0), _data(ps.covrlen, 0, 0))
aux2 = mahalanobis((*(line+j)).teta, _data(ms.covrdir, 0, 0), _data(ps.meandir, 0, 0), _data(ps.covrdir, 0, 0))
_data(ms.meanpos, 0, 0) = (*(line+j)).xm
_data(ms.meanpos, 1, 0) = (*(line+j)).ym
aux1 <= MAXLENDIST
and
aux2 <= MAXDIRDIST
no
yes
aux3 = stateDistance(&ps, &ms)
aux3 <= MAXPOSDIST
no
yes
no
aint2 = j
matched = 1
mindist = aux1
yes
aux1 <= mindist
aux1 = aux1*aux2*aux3

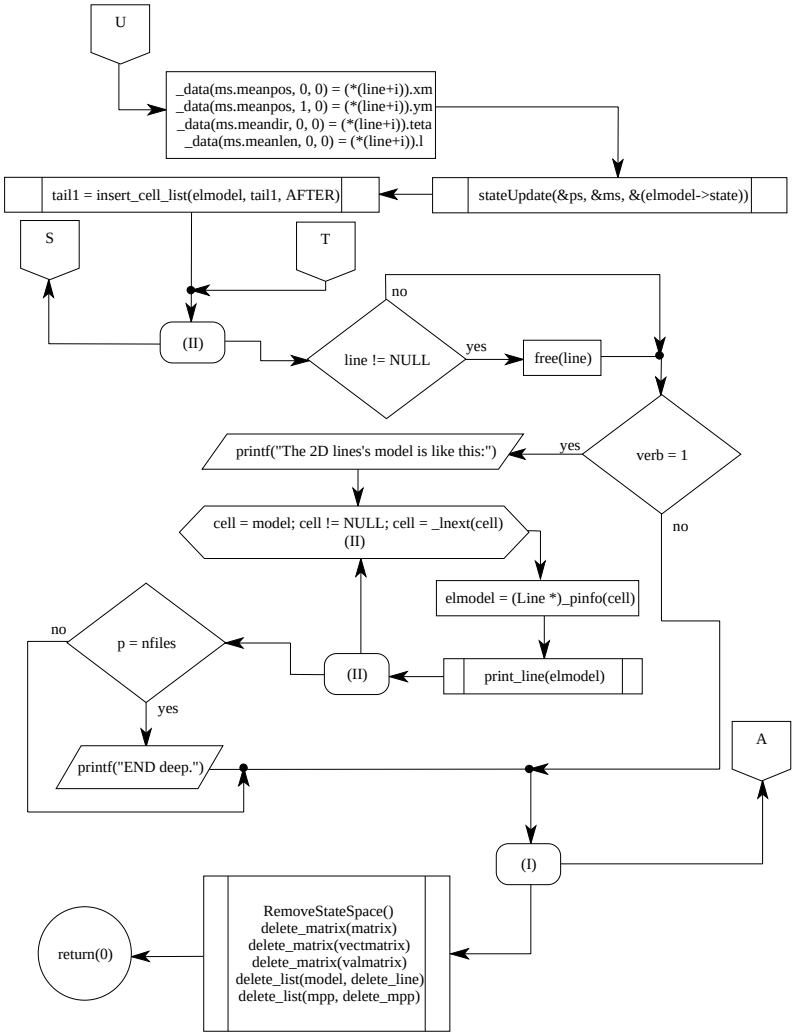

H
verb = 1
yes
no
printf("Model's element with reference: %d, matched by Kalman's filters with Mahalanobis's distances.", elmodel->reference)
printf("Old image's coordinates: First point: x=%d, y=%d, Last point: x=%d, y=%d.", elmodel->xifnew, elmodel->yifnew, elmodel->xffnew, elmodel->yffnew)
printf("New image's coordinates: First point: x=%d, y=%d, Last point: x=%d, y=%d.", (*(line+aint2)).xif, (*(line+aint2)).yif, (*(line+aint2)).xff, (*(line+aint2)).yff)
printf("Pause, press enter to continue.")
s = getc(stdin)
stateUpdate(&ps, &ms, &(elmodel->state))
elmodel->matched = 1
elmodel->xifnew = (*(line+aint2)).xif
elmodel->yifnew = (*(line+aint2)).yif
elmodel->xffnew = (*(line+aint2)).xff
elmodel->yffnew = (*(line+aint2)).yff
elmodel->xiunew = (*(line+aint2)).xiu
elmodel->yiunew = (*(line+aint2)).yiu
elmodel->xfunew = (*(line+aint2)).xfu
elmodel->yfunew = (*(line+aint2)).yfu
_data(ms.meanpos, 0, 0) = (*(line+aint2)).xm
_data(ms.meanpos, 1, 0) = (*(line+aint2)).ym
_data(ms.meandir, 0, 0) = (*(line+aint2)).teta
_data(ms.meanlen, 0, 0) = (*(line+aint2)).l
(*(line+aint2)).matched = 1
aux1 = (elmodel->xifnew+elmodel->xffnew)/2
aux2 = (elmodel->yifnew+elmodel->yffnew)/2
tx = (*(line+aint2)).xm-aux1
ty = (*(line+aint2)).ym-aux2
aux1 = (*(line+aint2)).xif-(elmodel->xifnew+tx)
aux2 = (*(line+aint2)).yif-(elmodel->yifnew+ty)
aux3 = PYTHAG(aux1, aux2)
aux1 = (*(line+aint2)).xff-(elmodel->xifnew+tx)
aux2 = (*(line+aint2)).yff-(elmodel->yifnew+ty)
aux4 = PYTHAG(aux1, aux2)
aux3 < aux4
elmodel->invert = 1
elmodel->invert = 1
elmodel->invert = 2
elmodel->invert = 2
elmodel->invert = 0
elmodel->invert = 1
elmodel->invert = 0
G
E
(II)
cell = model; cell != NULL; cell = _lnext(cell)
(II)
elmodel = (Line *)_pinfo(cell)
elmodel->matched = 0
statePredict(&(elmodel->state), &ps)
I
matched = into = 0
mindist = d[0]*d[1]*d[2]
i = 0 ; i < n ; i = i+1
(III)
(*(line+i)).matched = 0
J
M
L
N
aux1 = _data(ps.meanpos, 0, 0)-(*(line+i)).xm
aux2 = _data(ps.meanpos, 1, 0)-(*(line+i)).ym
difpm = PYTHAG(aux1, aux2)
difdir = |_data(ps.meandir, 0, 0)-(*(line+i)).teta|


N
difdir > 90
no
yes
difdir = 180-difdir
diflen = |_data(ps.meanlen, 0, 0)-(*(line+i)).l|
difpm <= d[0] and diflen <= d[1] and difdir <= d[2]
no
yes
aux1 = difdir*diflen*difpm
L
(III)
M
aux1 <= mindist
no
yes
matched = 1
mindist = aux1
aint2 = i
matched = 1
no
yes
_data(matrix, 0, 0) = _data(ps.covrpos, 0, 0)
_data(matrix, 0, 1) = _data(ps.covrpos, 0, 1)
_data(matrix, 1, 0) = _data(ps.covrpos, 1, 0)
_data(matrix, 1, 1) = _data(ps.covrpos, 1, 1)
matrix_eig(matrix, vectmatrix, valmatrix)
a = |_data(valmatrix, 0, 0)|
b = |_data(valmatrix, 1, 1)|
vx = _data(vectmatrix, 0, 0)
vy = _data(vectmatrix, 1, 0)
cx = (*(line+aint2)).xm
cy = (*(line+aint2)).ym
into = measure(line, n, &ms, &ps, vx, vy, cx, cy, a, b, 1)
into != 0
no
yes
O
verb = 1
no
yes
P
printf("Model's element with reference: %d, matched by Kalman's filters with geometric's restrictions.", elmodel->reference)
printf("Old image's coordinates: First point: x=%d, y=%d, Last point: x=%d, y=%d.", elmodel->xifnew, elmodel->yifnew, elmodel->xffnew, elmodel->yffnew)
printf("New image's coordinates: First point: x=%d, y=%d, Last point: x=%d, y=%d.", (*(line+aint2)).xif, (*(line+aint2)).yif, (*(line+aint2)).xff, (*(line+aint2)).yff)
printf("Pause, press enter to continue.")
s = getc(stdin)

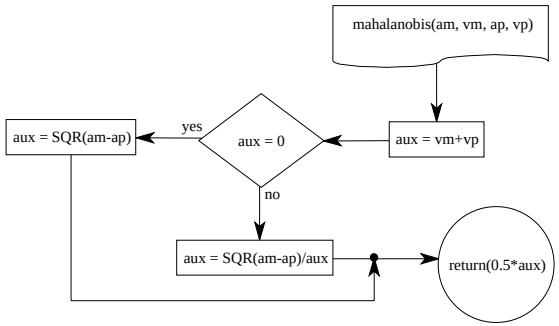

P
(*(line+aint2)).matched=1
aux1 = (elmodel->xifnew+elmodel->xffnew)/2
aux2 = (elmodel->yifnew+elmodel->yffnew)/2
tx = (*(line+aint2)).xm-aux1
ty = (*(line+aint2)).ym-aux2
aux1 = (*(line+aint2)).xif-(elmodel->xifnew+tx)
aux2 = (*(line+aint2)).yif-(elmodel->yifnew+ty)
aux3 = PYTHAG(aux1, aux2)
aux1 = (*(line+aint2)).xff-(elmodel->xifnew+tx)
aux2 = (*(line+aint2)).yff-(elmodel->yifnew+ty)
aux4 = PYTHAG(aux1, aux2)
aux3 < aux4
yes
no
elmodel->invert = 2
yes
elmodel->invert = 0
no
elmodel->invert = 2
no
elmodel->invert = 1
yes
elmodel->invert = 1
yes
elmodel->invert = 0
elmodel->invert = 1
elmodel->matched = 1
elmodel->xifnew = (*(line+aint2)).xif
elmodel->yifnew = (*(line+aint2)).yif
elmodel->xffnew = (*(line+aint2)).xff
elmodel->yffnew = (*(line+aint2)).yff
elmodel->xiunew = (*(line+aint2)).xiu
elmodel->yiunew = (*(line+aint2)).yiu
elmodel->xfunew = (*(line+aint2)).xfu
elmodel->yfunew = (*(line+aint2)).yfu
_data(ms.meanpos, 0, 0) = (*(line+aint2)).xm
_data(ms.meanpos, 1, 0) = (*(line+aint2)).ym
_data(ms.meandir, 0, 0) = (*(line+aint2)).teta
_data(ms.meanlen, 0, 0) = (*(line+aint2)).l
stateUpdate(&ps, &ms, &(elmodel->state))
O
verb = 1
yes
printf("Model's element with reference:
%d, not matched.", elmodel->reference)
printf("Pause, press enter to continue.")
s = getc(stdin)
no
I
J
(II)
aint2 = -1
Q
_data(ms.meanpos, 0, 0) = _data(ps.meanpos, 0, 0)
_data(ms.meanpos, 1, 0) = _data(ps.meanpos, 1, 0)
_data(ms.meandir, 0, 0) = _data(ps.meandir, 0, 0)
_data(ms.meanlen, 0, 0) = _data(ps.meanlen, 0, 0)
_data(ms.covrpos, 0, 0) = POSXCOVR
_data(ms.covrpos, 1, 1) = POSYCOVR
_data(ms.covrpos, 0, 1) = _data(ms.covrpos, 1, 0)=0.0
_data(ms.covrdir, 0, 0) = DIRCOVR
_data(ms.covrlen, 0, 0) = LENCOVR
stateUpdate(&ps, &ms, &(elmodel->state))


Q
cell = model; cell != NULL; cell = _lnext(cell)
(II)
elmodel = (Line *)_pinfo(cell)
elmodel->matched = 1
yes
elmodel->matched = 0
no
elmodel->cf < 5
yes
elmodel->cf = elmodel->cf+1
no
elmodel->cf = elmodel->cf-1
elmodel->cf >= 3
and deep
yes
aint1 = elmodel->lastmpp
no
(p-aint1+1) >= nfdeep
yes
no
aint2 != aint1
no
yes
cell1 = _lnext(cell1)
(III)
i = 1; i < aint1; i = i+1
(III)
cell1= mpp
elmpp2 = (Mpp *)_pinfo(cell1)
aint2 = aint1
compute_deep(imgout[nband], elmodel, elmpp1->m,
elmpp2->m, maxdeep, mindeep, verb)
elmodel->xiuold = elmodel->xiunew
elmodel->yiuold = elmodel->yiunew
elmodel->xfuold = elmodel->xfunew
elmodel->yfuold = elmodel->yfunew
elmodel->lastmpp = p
elmodel->invert = 0
elmodel->cf < 0
no
yes
(II)
R
model = delete_cell_list(cell, model, delete_line)

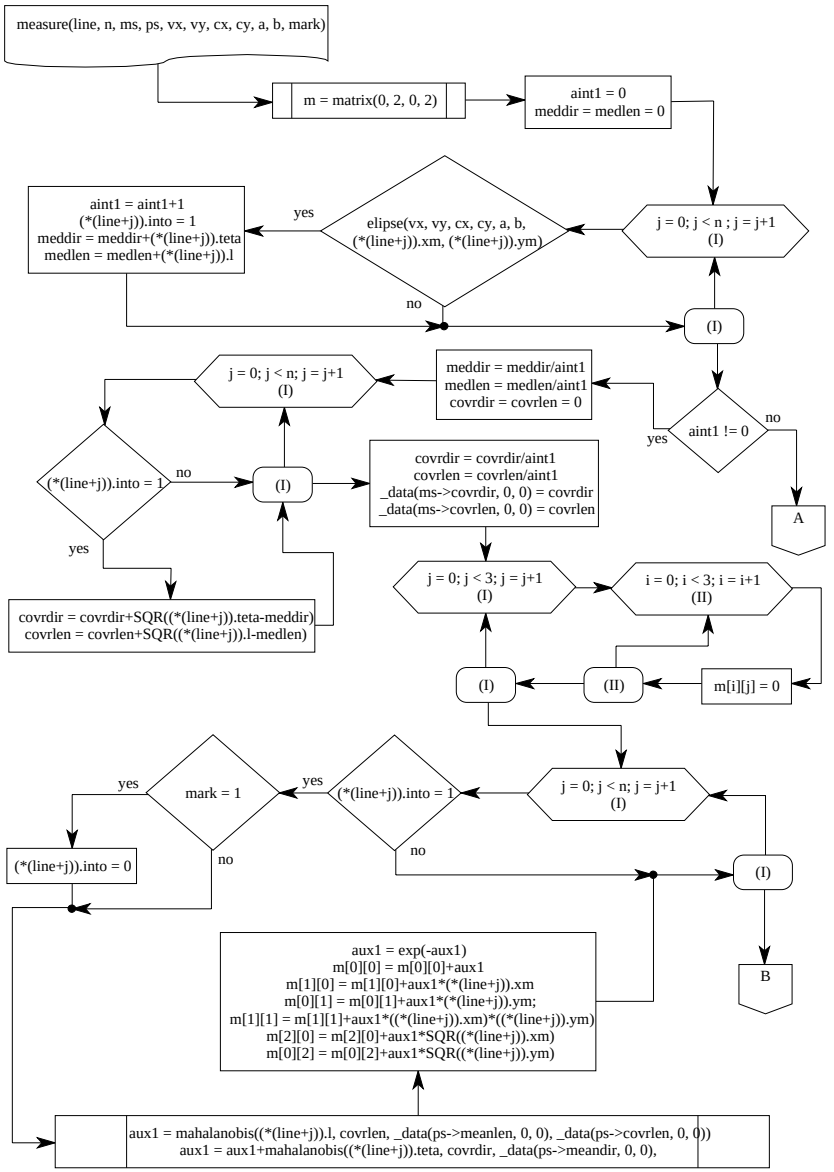

R
D
tail2 = insert_cell_list(elmpp1, tail2, AFTER)
i = 0; i < n; i = i+1
(II)
S
(*(line+i)).matched = 0
no
yes
ref = ref+1
no
verb = 1
yes
T
printf("Inserting a new element into
the 2D model with reference: %d.", ref)
printf("Pause, press enter to continue.")
s=getc(stdin)
elmodel = create_line()
elmodel->reference = ref
elmodel->cf = 3
elmodel->xifnew = (*(line+i)).xif
elmodel->yifnew = (*(line+i)).yif
elmodel->xffnew = (*(line+i)).xff
elmodel->yffnew = (*(line+i)).yff
elmodel->matched = 0
elmodel->xiuold = (*(line+i)).xiu
elmodel->yiuold = (*(line+i)).yiu
elmodel->xfuold = (*(line+i)).xfu
elmodel->yfuold = (*(line+i)).yfu
elmodel->lastmpp = p
elmodel->invert = 2
_xpos(elmodel) = (*(line+i)).xm
_ypos(elmodel) = (*(line+i)).ym
_xvel(elmodel) = 0
_yvel(elmodel) = 0
_xacl(elmodel) = 0
_yacl(elmodel) = 0
_dir(elmodel) = (*(line+i)).teta
_dirvel(elmodel) = 0
_diracl(elmodel) = 0
_len(elmodel) = (*(line+i)).l
_lenvel(elmodel) = 0
_lenacl(elmodel) = 0
_xpos_cv(elmodel) = POSXCOVR
_ypos_cv(elmodel) = POSYCOVR
_xvel_cv(elmodel) = SQR(DXMAX)
_yvel_cv(elmodel) = SQR(DYMAX)
_xacl_cv(elmodel) = SQR(DXMAX/2)
_yacl_cv(elmodel) = SQR(DYMAX/2)
_dir_cv(elmodel) = DIRCOVR
_dirvel_cv(elmodel) = SQR(DIRMAX)
_diracl_cv(elmodel) = SQR(DIRMAX/2)
_len_cv(elmodel) = LENCOVR
_lenvel_cv(elmodel) = SQR(LENMAX)
_lenacl_cv(elmodel) = SQR(LENMAX/2)
statePredict(&(elmodel->state), &ps)
cx = (*(line+i)).xm
cy = (*(line+i)).ym
measure(line, n, &ms, &ps, vxi, vyi, cx, cy, ai, bi, 1)
U

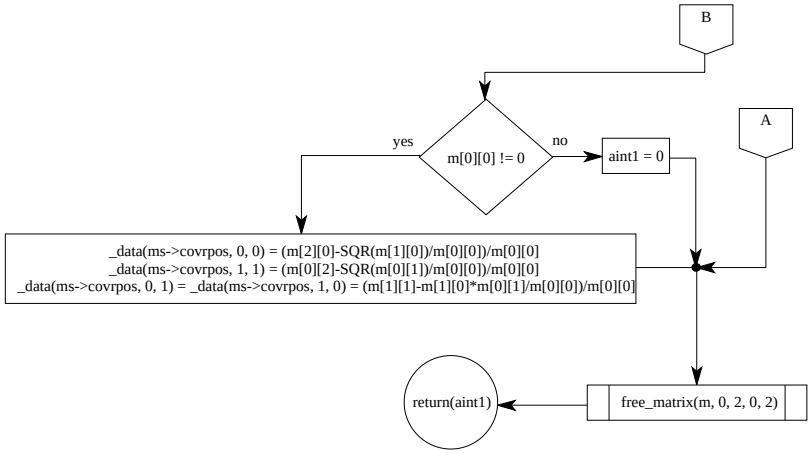

U
_data(ms.meanpos, 0, 0) = (*(line+i)).xm
_data(ms.meanpos, 1, 0) = (*(line+i)).ym
_data(ms.meandir, 0, 0) = (*(line+i)).teta
_data(ms.meanlen, 0, 0) = (*(line+i)).l
tail1 = insert_cell_list(elmodel, tail1, AFTER)
stateUpdate(&ps, &ms, &(elmodel->state))
S
T
(II)
no
line != NULL
yes
free(line)
verb = 1
yes
printf("The 2D lines's model is like this:")
no
cell = model; cell != NULL; cell = _lnext(cell)
(II)
elmodel = (Line *)_pinfo(cell)
print_line(elmodel)
(II)
no
p = nfiles
yes
printf("END deep.")
A
(I)
RemoveStateSpace()
delete_matrix(matrix)
delete_matrix(vectmatrix)
delete_matrix(valmatrix)
delete_list(model, delete_line)
delete_list(mpp, delete_mpp)
return(0)

3.2.3 - Módulo mahalanobis

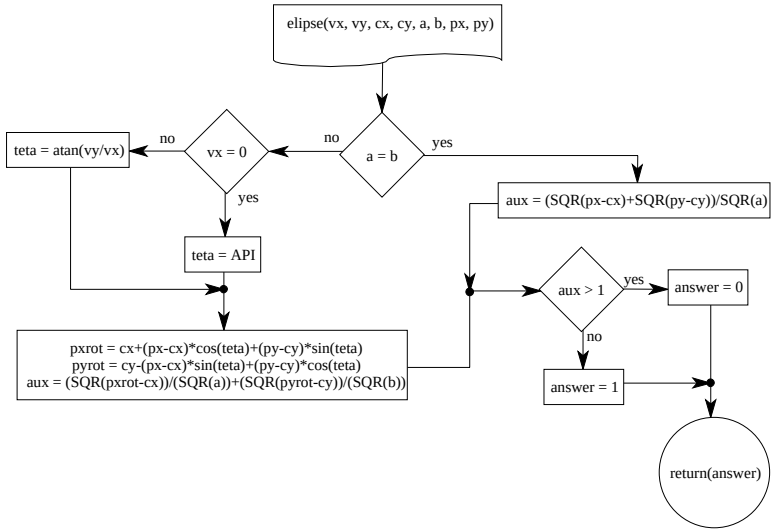


3.2.4 - Módulo measure

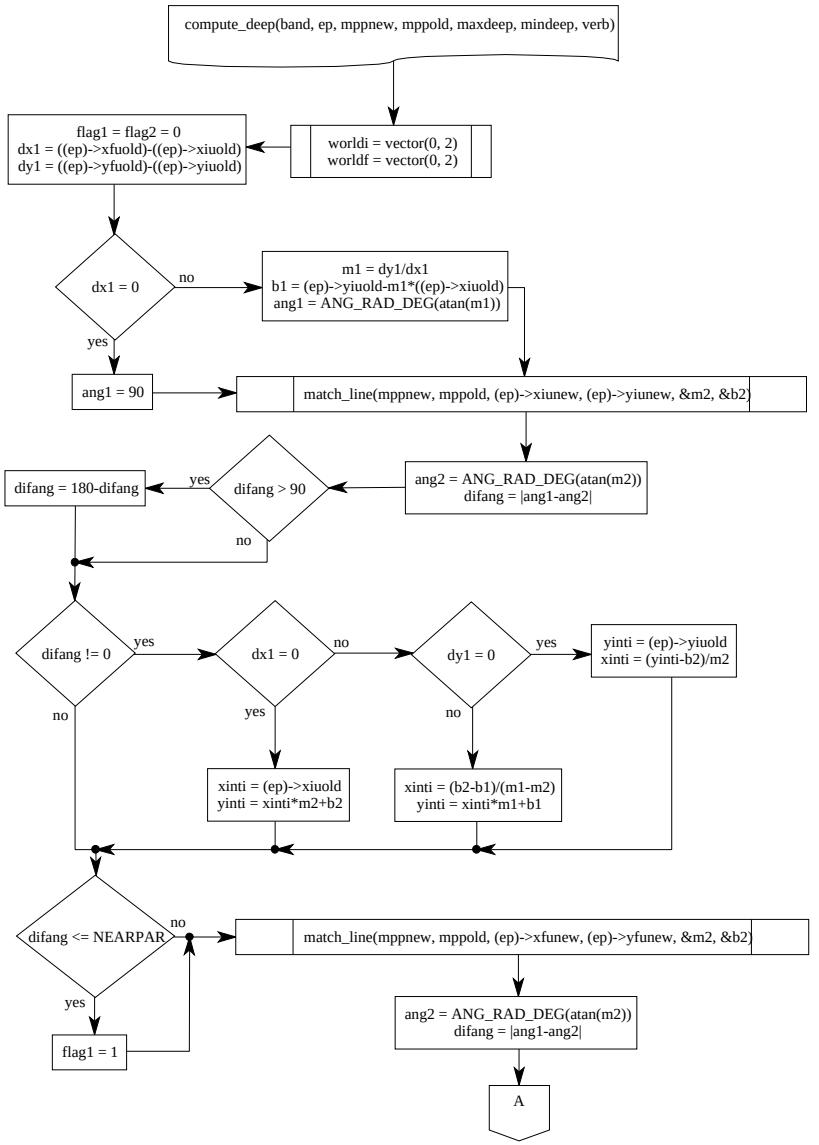


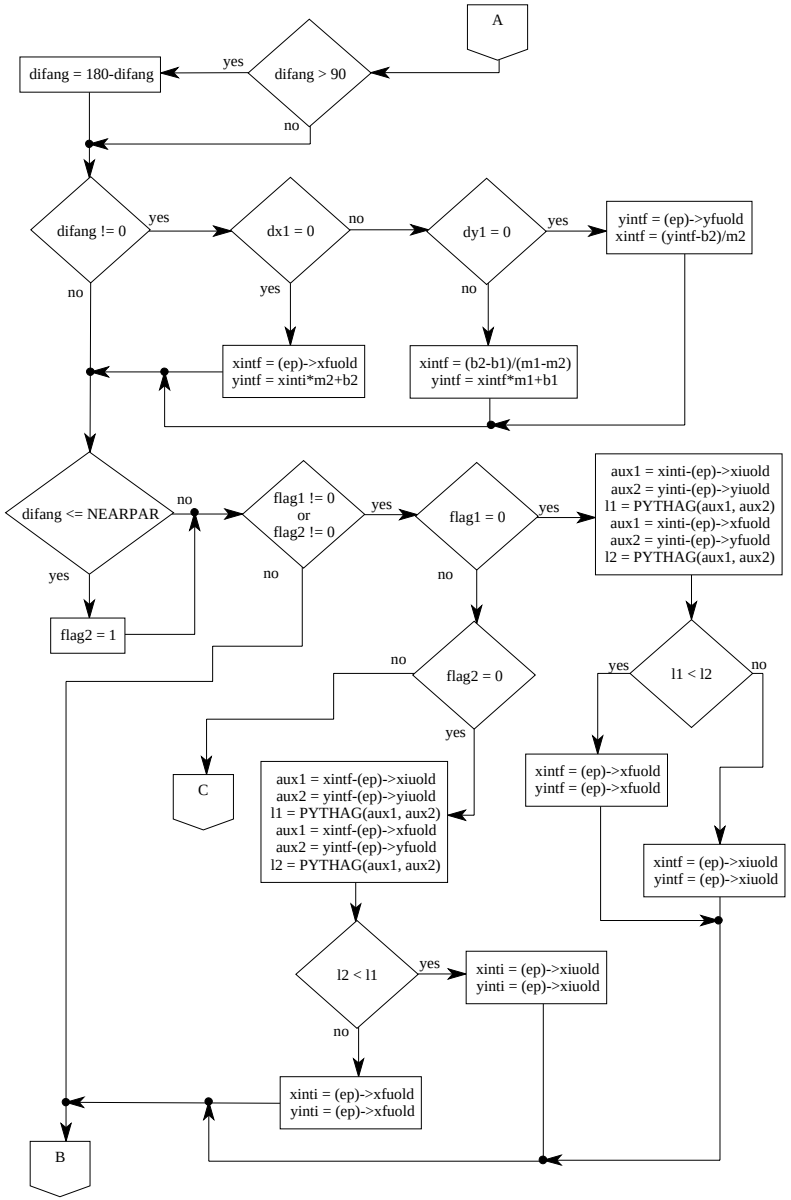


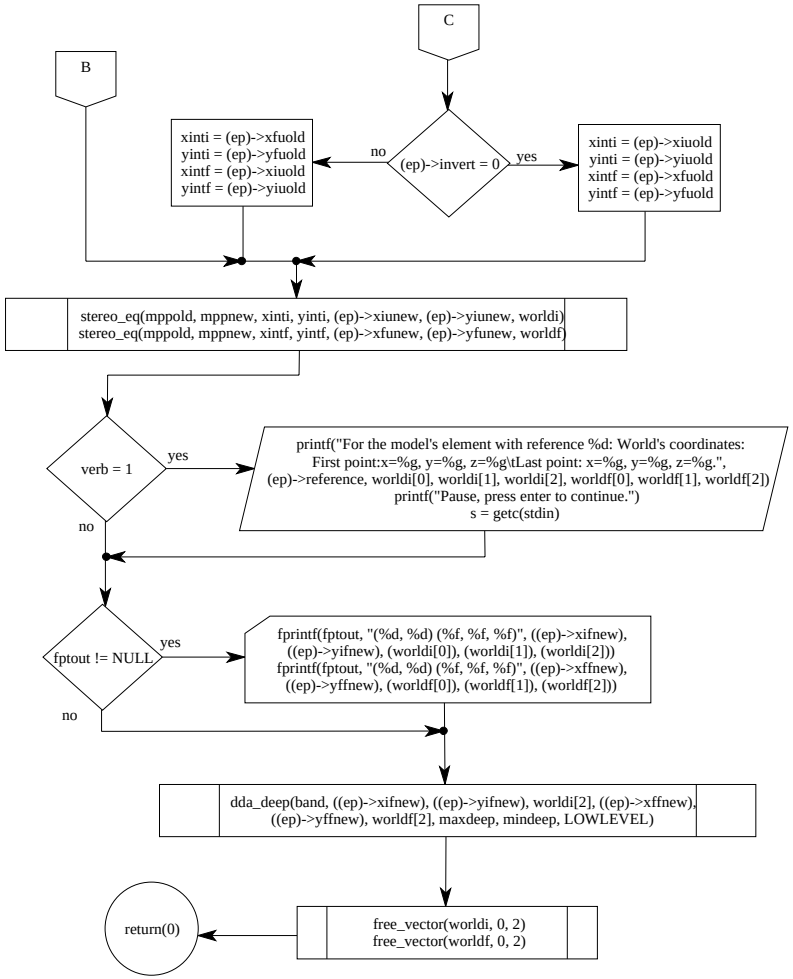
3.2.5 - Módulo elipse



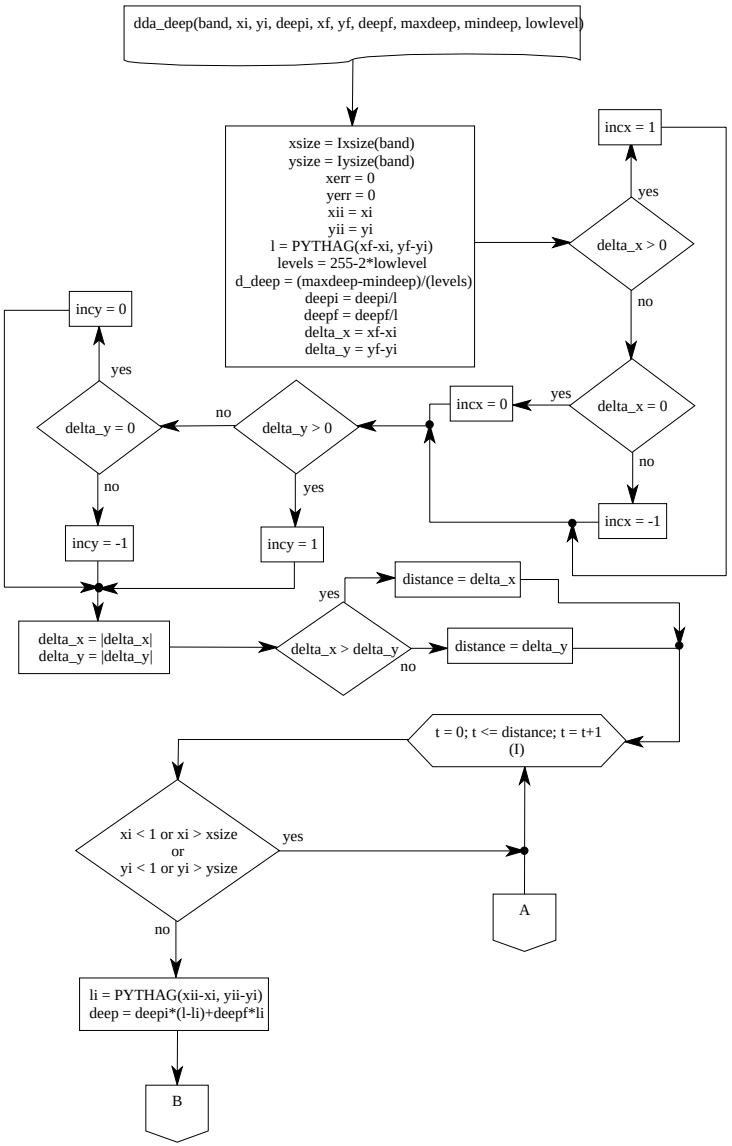
3.2.6 - Módulo compute_deep

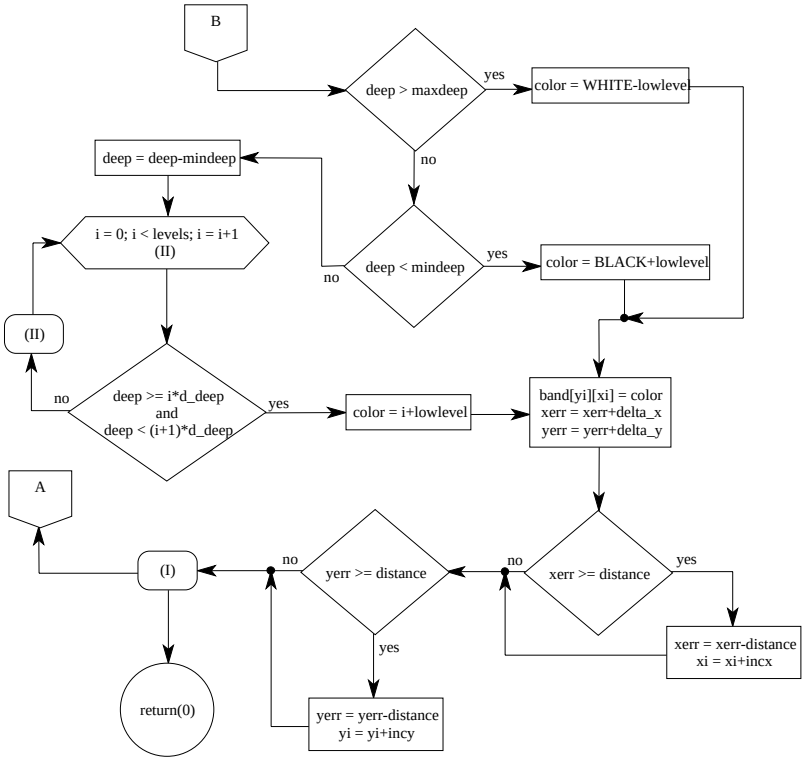



A
difang = 180-difang
yes
difang > 90
no
difang != 0
yes
dx1 = 0
no
dy1 = 0
yes
yintf = (ep)->yfuold
xintf = (yintf-b2)/m2
no
yes
no
xintf = (ep)->xfuold
yintf = xinti*m2+b2
xintf = (b2-b1)/(m1-m2)
yintf = xintf*m1+b1
difang <= NEARPAR
no
flag1 != 0
or
flag2 != 0
yes
flag1 = 0
yes
aux1 = xinti-(ep)->xiuold
aux2 = yinti-(ep)->yiuold
l1 = PYTHAG(aux1, aux2)
aux1 = xinti-(ep)->xfuold
aux2 = yinti-(ep)->yfuold
l2 = PYTHAG(aux1, aux2)
yes
flag2 = 1
no
no
flag2 = 0
yes
l1 < l2
yes
no
C
aux1 = xintf-(ep)->xiuold
aux2 = yintf-(ep)->yiuold
l1 = PYTHAG(aux1, aux2)
aux1 = xintf-(ep)->xfuold
aux2 = yintf-(ep)->yfuold
l2 = PYTHAG(aux1, aux2)
xintf = (ep)->xfuold
yintf = (ep)->xfuold
xintf = (ep)->xiuold
yintf = (ep)->xiuold
l2 < l1
yes
xinti = (ep)->xiuold
yinti = (ep)->xiuold
no
xinti = (ep)->xfuold
yinti = (ep)->xfuold
B

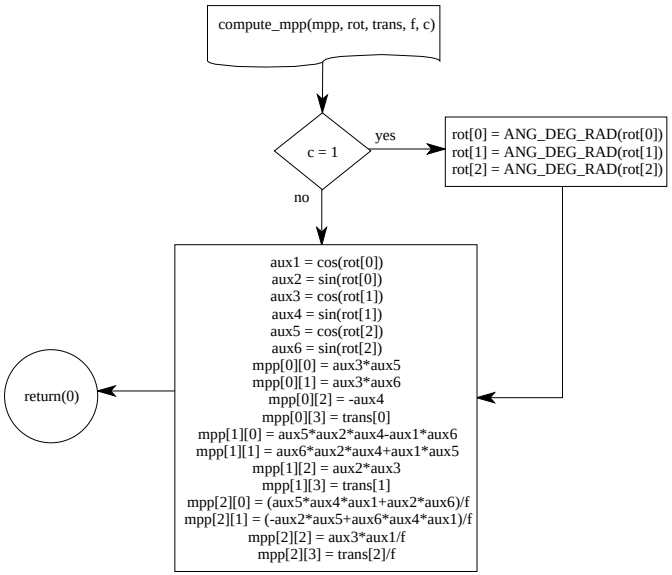

C
B
xinti = (ep)->xfuold
yinti = (ep)->yfuold
xintf = (ep)->xiuold
yintf = (ep)->yiuold
no
(ep)->invert = 0
yes
xinti = (ep)->xiuold
yinti = (ep)->yiuold
xintf = (ep)->xfuold
yintf = (ep)->yfuold
stereo_eq(mppold, mppnew, xinti, yinti, (ep)->xiunew, (ep)->yiunew, worldi)
stereo_eq(mppold, mppnew, xintf, yintf, (ep)->xfunew, (ep)->yfunew, worldf)
verb = 1
yes
printf("For the model's element with reference %d: World's coordinates:
First point:x=%g, y=%g, z=%g\tLast point: x=%g, y=%g, z=%g.",
(ep)->reference, worldi[0], worldi[1], worldi[2], worldf[0], worldf[1], worldf[2])
printf("Pause, press enter to continue.")
s = getc(stdin)
no
fptout != NULL
yes
fprintf(fptout, "(%d, %d) (%f, %f, %f)", ((ep)->xifnew),
((ep)->yifnew), (worldi[0]), (worldi[1]), (worldi[2]))
fprintf(fptout, "(%d, %d) (%f, %f, %f)", ((ep)->xffnew),
((ep)->yffnew), (worldf[0]), (worldf[1]), (worldf[2]))
no
dda_deep(band, ((ep)->xifnew), ((ep)->yifnew), worldi[2], ((ep)->xffnew),
((ep)->yffnew), worldf[2], maxdeep, mindeep, LOWLEVEL)
return(0)
free_vector(worldi, 0, 2)
free_vector(worldf, 0, 2)

3.2.7 - Módulo dda_deep

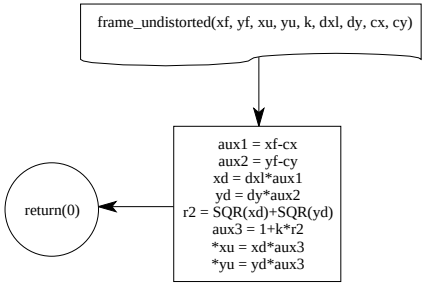




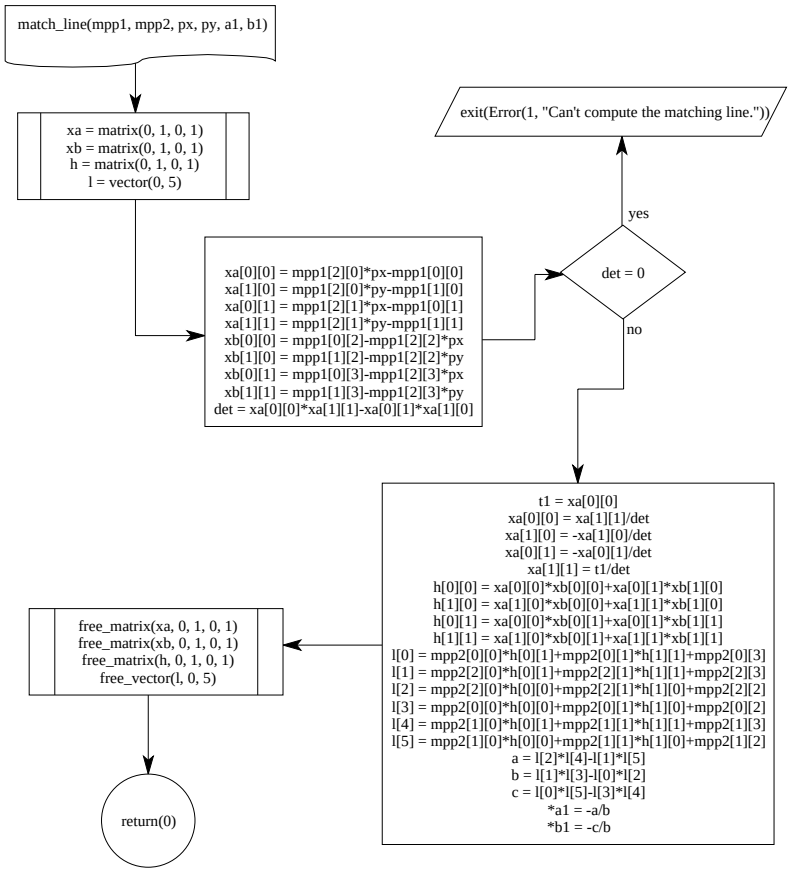
3.2.8 - Módulo compute_mpp



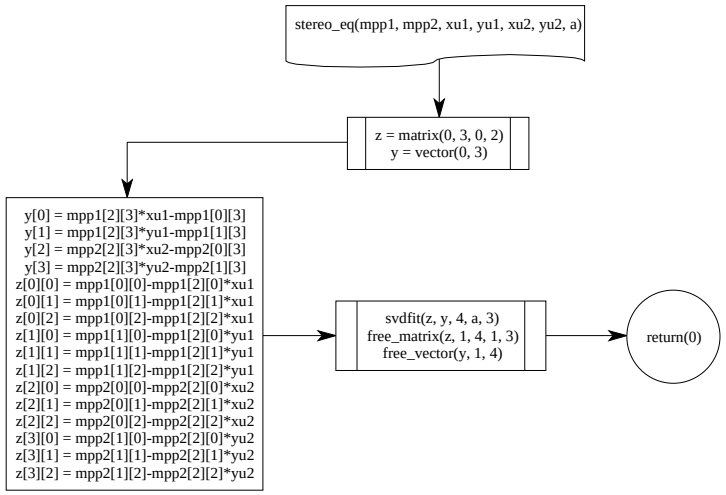
3.2.9 - Módulo frame_undistorted



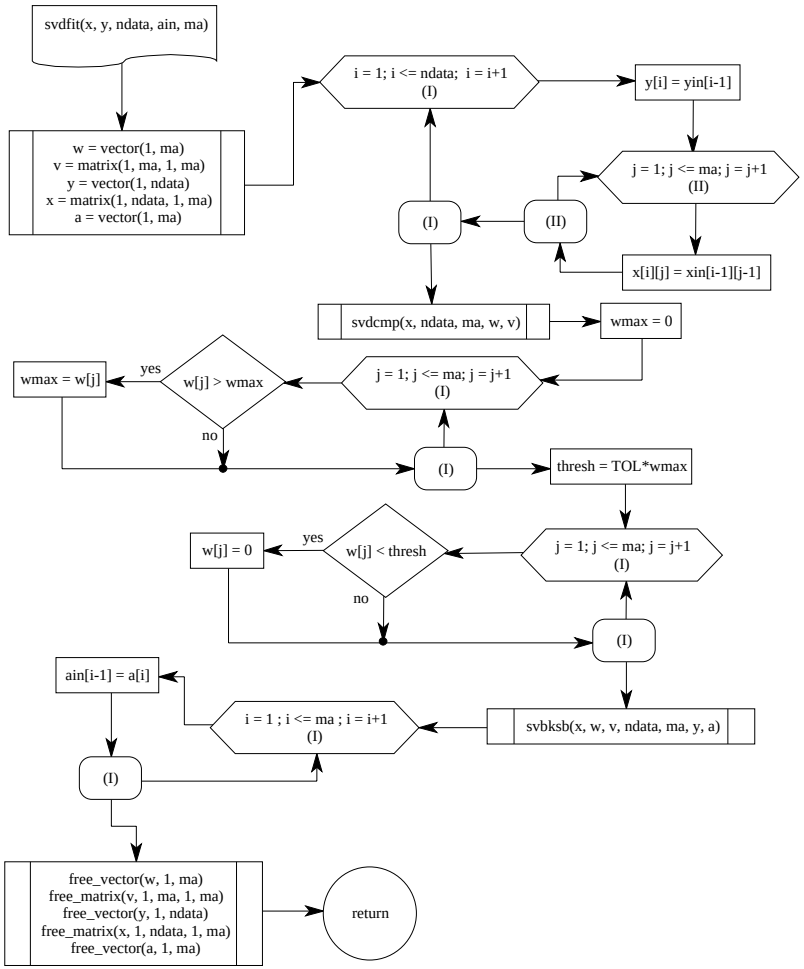
3.2.10 - Módulo match_line



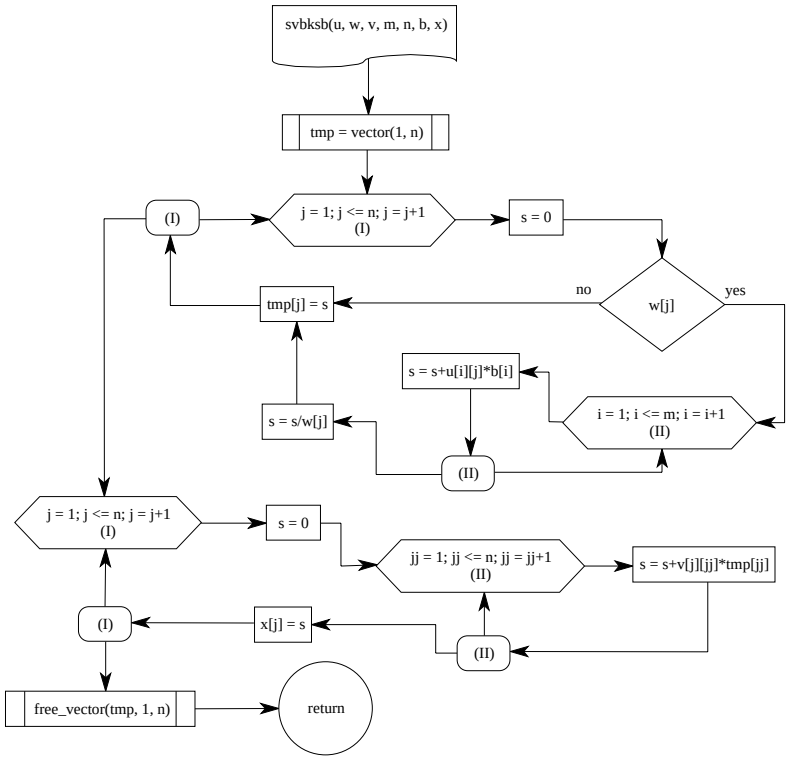
3.2.11 - Módulo stereo_eq



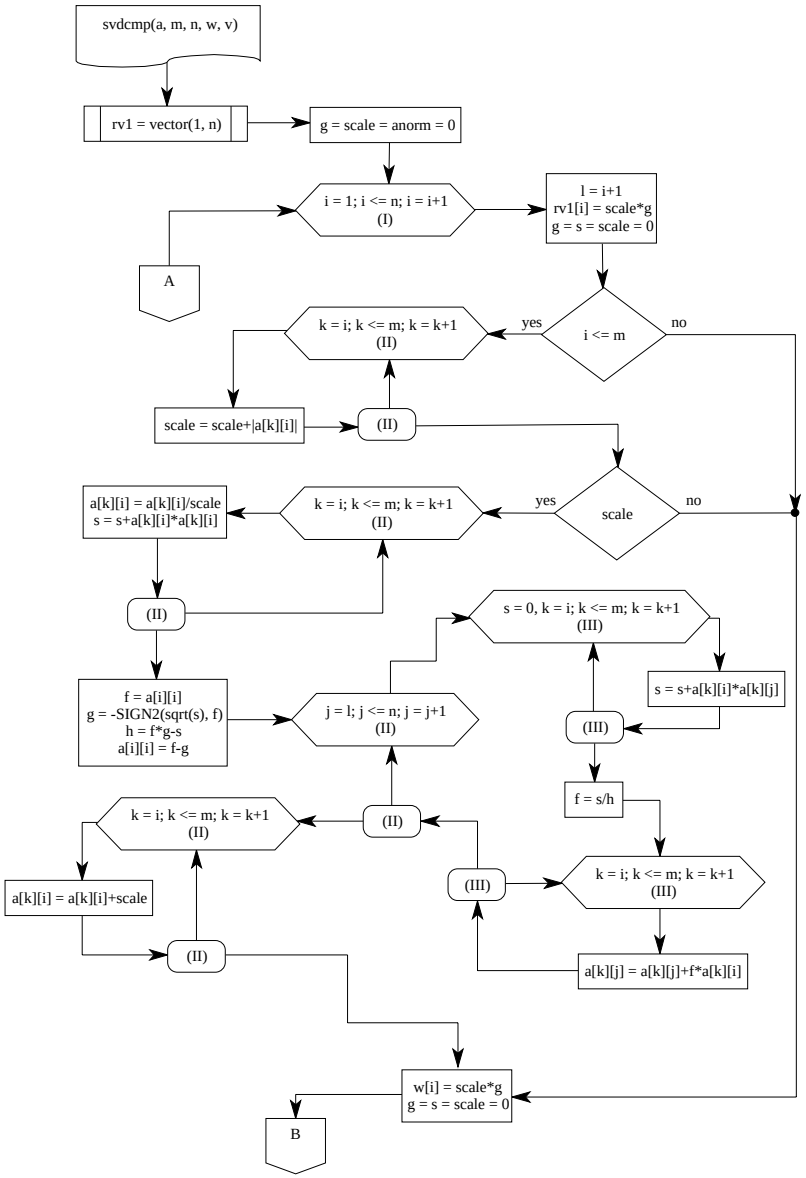
3.2.12 - Módulo svdfit

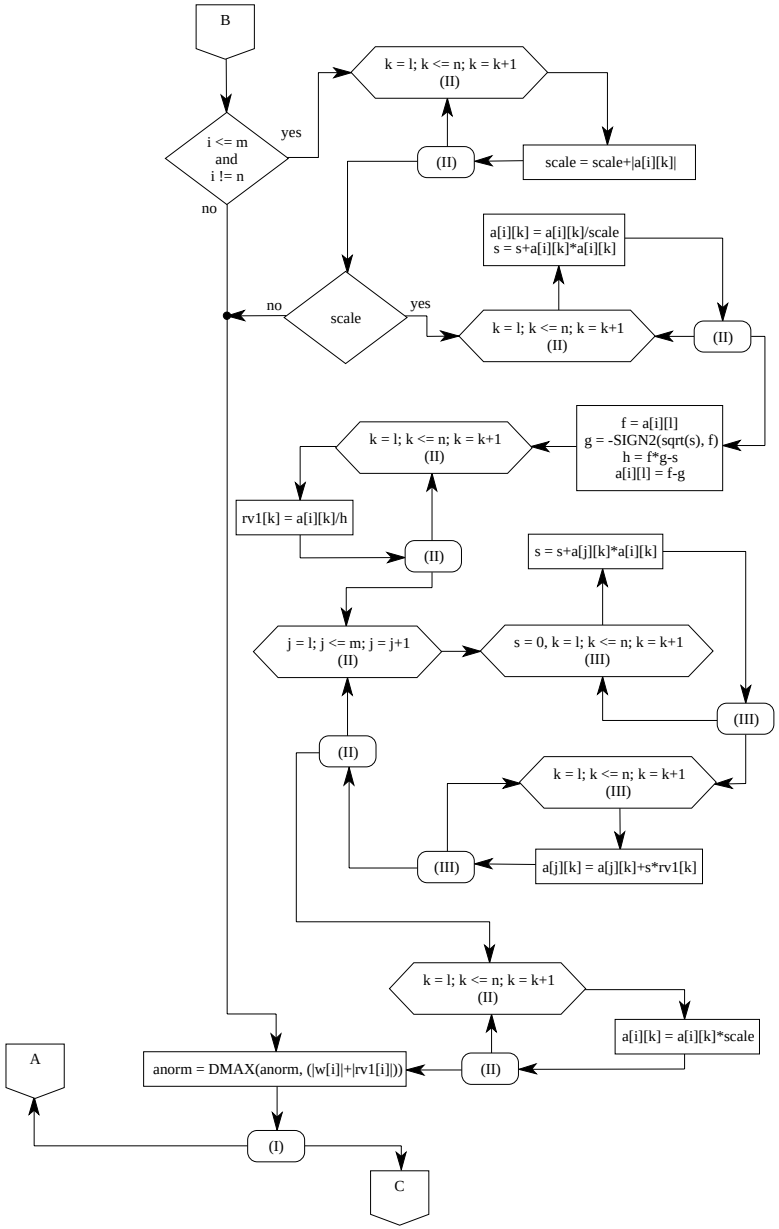


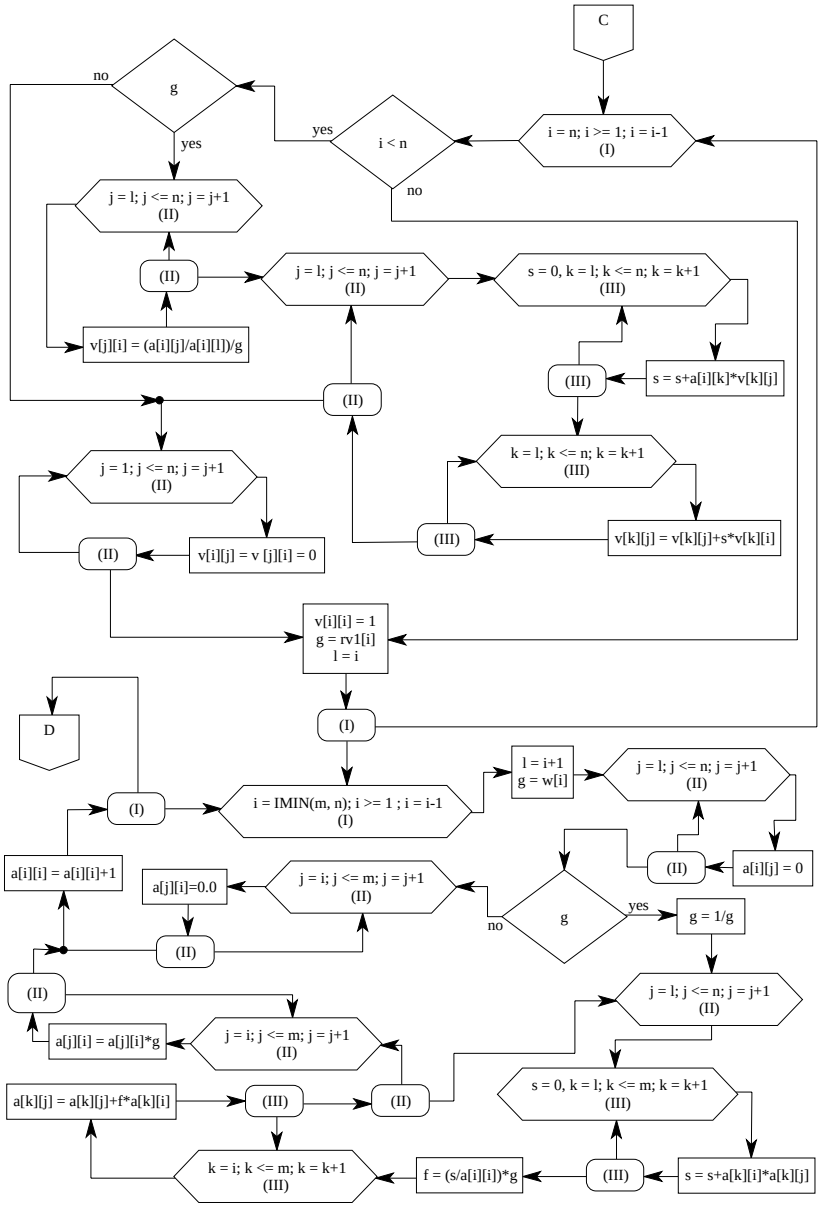
3.2.13 - Módulo svbksb

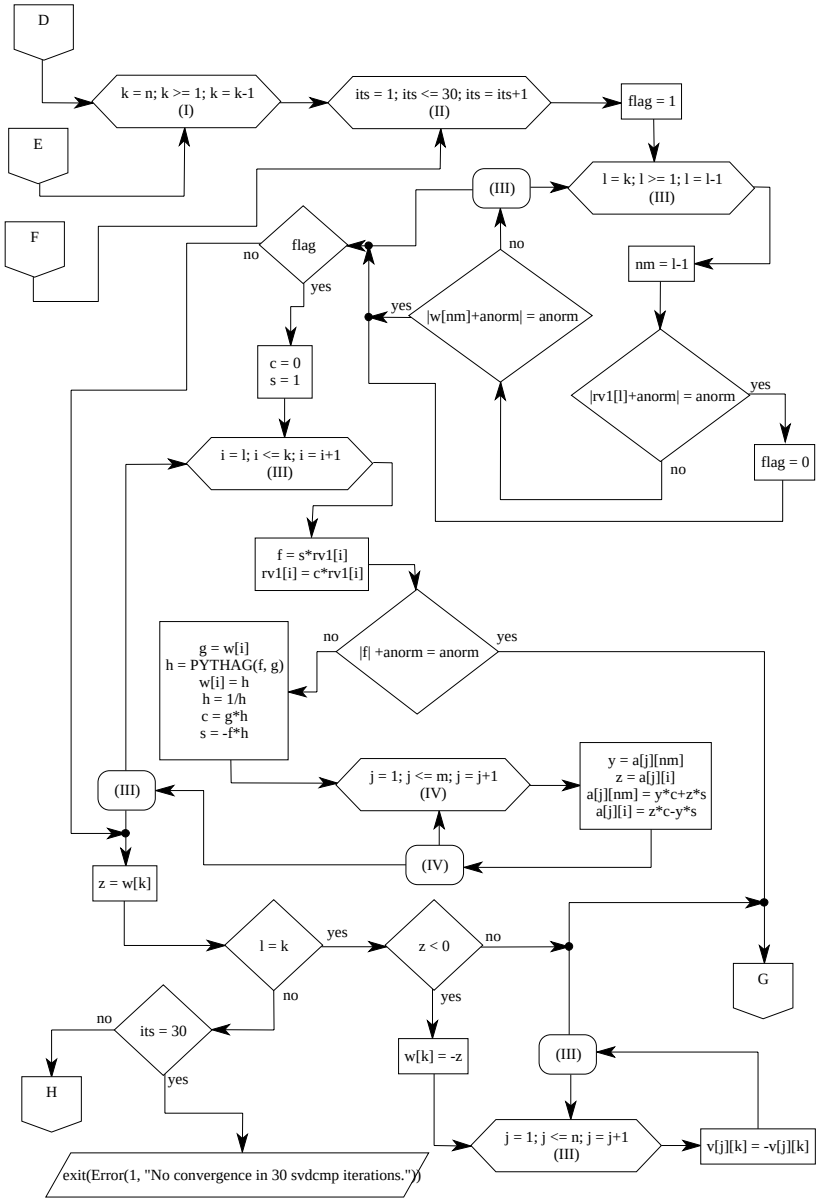


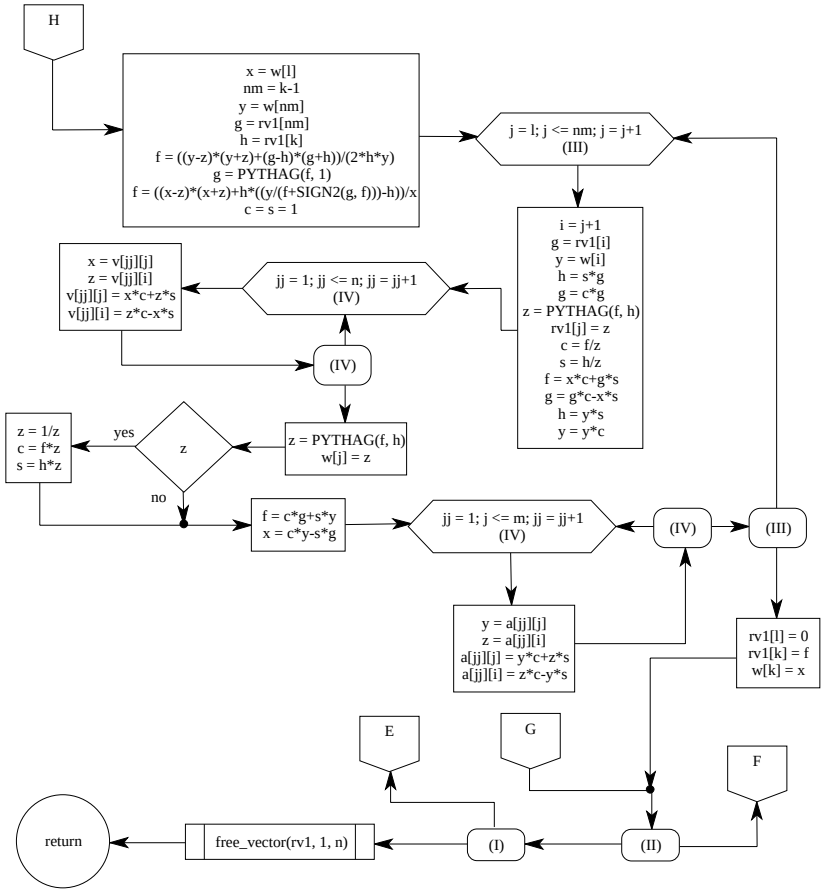
3.2.14 - Módulo svdcmp



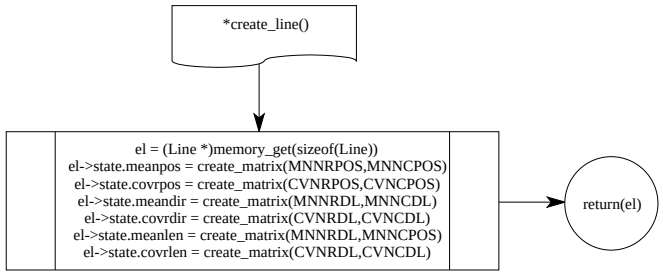

B
i <= m and i != n
yes
no
k = l; k <= n; k = k+1 (II)
(II)
scale = scale+|a[i][k]|
a[i][k] = a[i][k]/scale
s = s+a[i][k]*a[i][k]
scale
no
yes
k = l; k <= n; k = k+1 (II)
(II)
f = a[i][l]
g = -SIGN2(sqrt(s), f)
h = f*g-s
a[i][l] = f-g
k = l; k <= n; k = k+1 (II)
rv1[k] = a[i][k]/h
(II)
s = s+a[j][k]*a[i][k]
j = l; j <= m; j = j+1 (II)
s = 0, k = l; k <= n; k = k+1 (III)
(III)
(II)
k = l; k <= n; k = k+1 (III)
(III)
a[j][k] = a[j][k]+s*rv1[k]
k = l; k <= n; k = k+1 (II)
a[i][k] = a[i][k]*scale
A
anorm = DMAX(anorm, (|w[i]|+|rv1[i]|))
(II)
(I)
C


C
i = n; i >= 1; i = i-1 (I)
i < n
yes
no
g
no
yes
j = l; j <= n; j = j+1 (II)
(II)
j = l; j <= n; j = j+1 (II)
s = 0, k = l; k <= n; k = k+1 (III)
v[j][i] = (a[i][j]/a[i][l])/g
(III)
s = s+a[i][k]*v[k][j]
(II)
k = l; k <= n; k = k+1 (III)
j = 1; j <= n; j = j+1 (II)
v[k][j] = v[k][j]+s*v[k][i]
(III)
(II)
v[i][j] = v [j][i] = 0
v[i][i] = 1
g = rv1[i]
l = i
D
(I)
(I)
i = IMIN(m, n); i >= 1 ; i = i-1 (I)
l = i+1
g = w[i]
j = l; j <= n; j = j+1 (II)
a[i][i] = a[i][i]+1
a[j][i]=0
j = i; j <= m; j = j+1 (II)
g
yes
no
(II)
a[i][j] = 0
g = 1/g
(II)
(II)
j = l; j <= n; j = j+1 (II)
a[j][i] = a[j][i]*g
j = i; j <= m; j = j+1 (II)
a[k][j] = a[k][j]+f*a[k][i]
(III)
(II)
s = 0, k = l; k <= m; k = k+1 (III)
k = i; k <= m; k = k+1 (III)
f = (s/a[i][i])*g
(III)
s = s+a[k][i]*a[k][j]

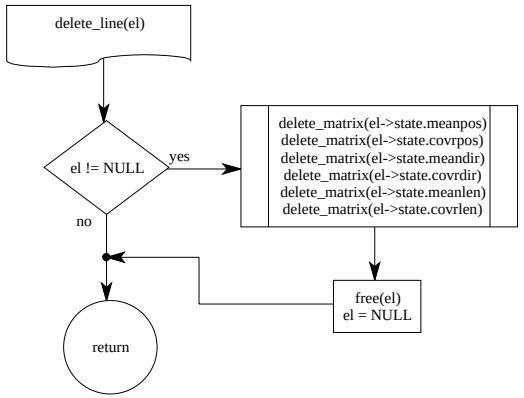

D
k = n; k >= 1; k = k-1
(I)
its = 1; its <= 30; its = its+1
(II)
flag = 1
E
(III)
l = k; l >= 1; l = l-1
(III)
F
flag
no
yes
|w[nm]+anorm| = anorm
yes
no
nm = l-1
c = 0
s = 1
|rv1[l]+anorm| = anorm
yes
no
flag = 0
i = l; i <= k; i = i+1
(III)
f = s*rv1[i]
rv1[i] = c*rv1[i]
|f| +anorm = anorm
no
yes
g = w[i]
h = PYTHAG(f, g)
w[i] = h
h = 1/h
c = g*h
s = -f*h
(III)
j = 1; j <= m; j = j+1
(IV)
y = a[j][nm]
z = a[j][i]
a[j][nm] = y*c+z*s
a[j][i] = z*c-y*s
(IV)
z = w[k]
l = k
yes
no
z < 0
no
yes
G
its = 30
no
yes
H
w[k] = -z
(III)
j = 1; j <= n; j = j+1
(III)
v[j][k] = -v[j][k]
exit(Error(1, "No convergence in 30 svdcmp iterations."))


H
x = w[l]
nm = k-1
y = w[nm]
g = rv1[nm]
h = rv1[k]
f = ((y-z)*(y+z)+(g-h)*(g+h))/(2*h*y)
g = PYTHAG(f, 1)
f = ((x-z)*(x+z)+h*((y/(f+SIGN2(g, f)))-h))/x
c = s = 1
j = l; j <= nm; j = j+1
(III)
i = j+1
g = rv1[i]
y = w[i]
h = s*g
g = c*g
z = PYTHAG(f, h)
rv1[j] = z
c = f/z
s = h/z
f = x*c+g*s
g = g*c-x*s
h = y*s
y = y*c
x = v[jj][j]
z = v[jj][i]
v[jj][j] = x*c+z*s
v[jj][i] = z*c-x*s
jj = 1; jj <= n; jj = jj+1
(IV)
(IV)
z = PYTHAG(f, h)
w[j] = z
z
yes
no
z = 1/z
c = f*z
s = h*z
f = c*g+s*y
x = c*y-s*g
jj = 1; j <= m; jj = jj+1
(IV)
(IV)
(III)
y = a[jj][j]
z = a[jj][i]
a[jj][j] = y*c+z*s
a[jj][i] = z*c-y*s
rv1[l] = 0
rv1[k] = f
w[k] = x
E
G
F
return
free_vector(rv1, 1, n)
(I)
(II)

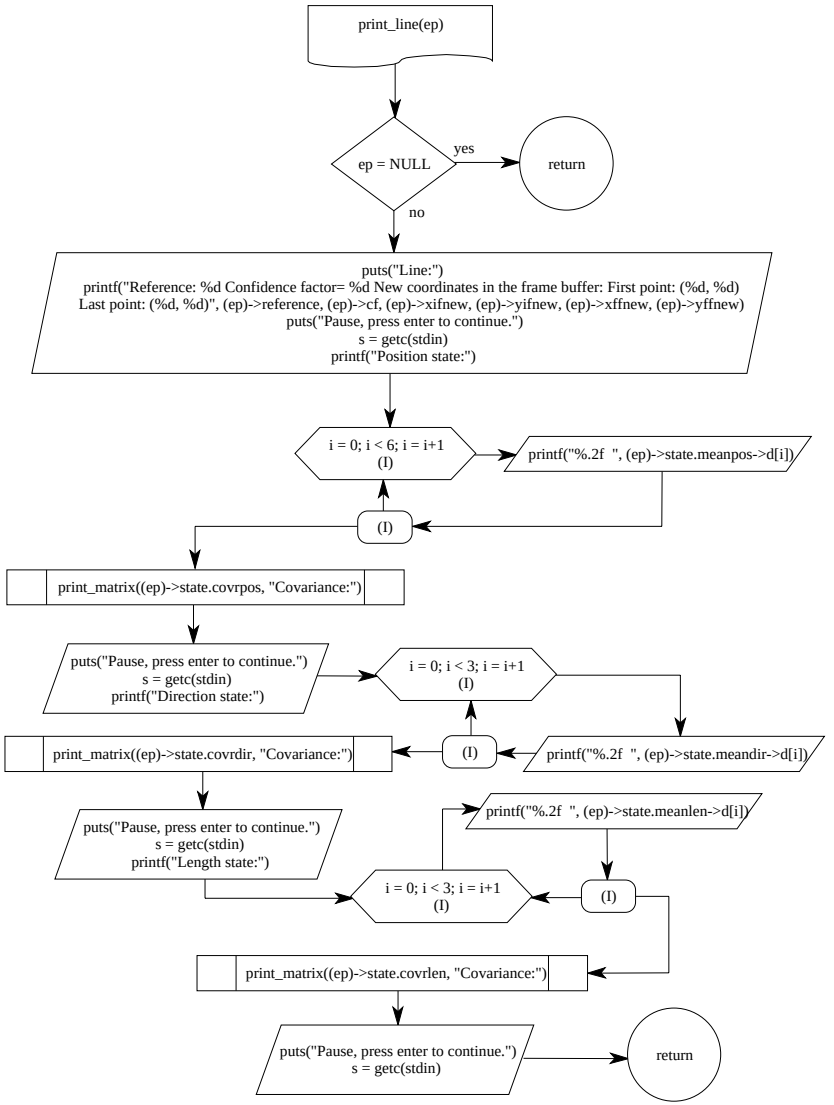
3.2.15 - Módulo create_line



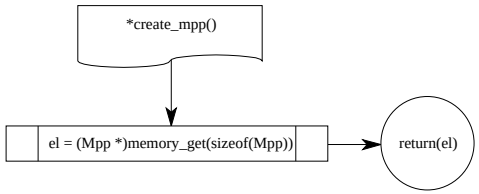
3.2.16 - Módulo delete_line



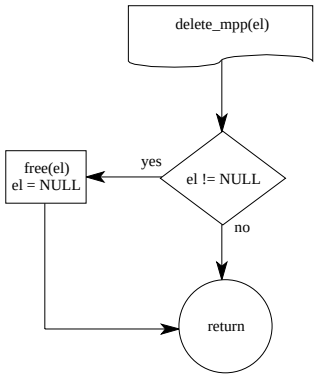
3.2.17 - Módulo print_line



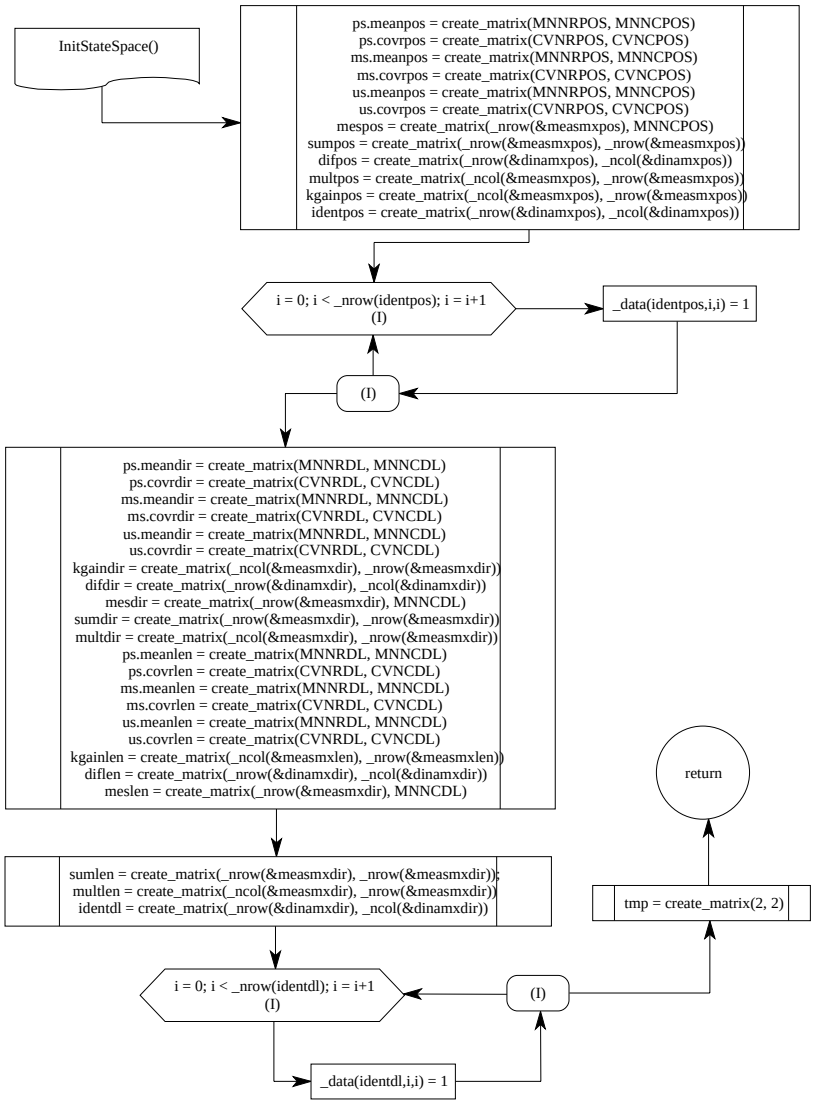
3.2.18 - Módulo create_mpp



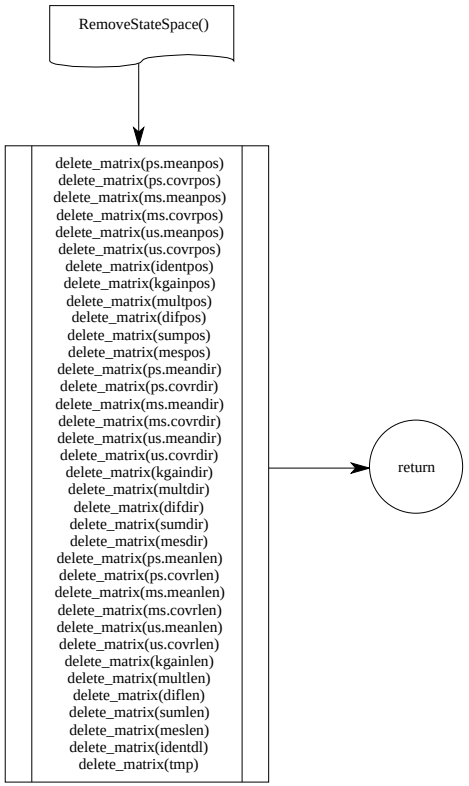
3.2.19 - Módulo delete_mpp



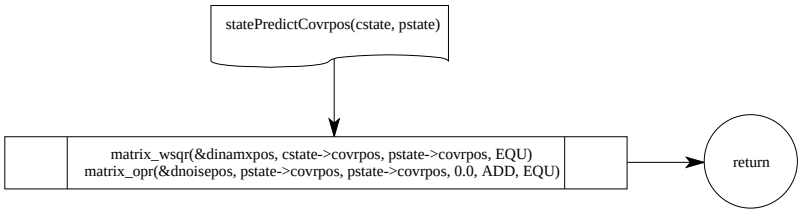
3.2.20 - Módulo InitStateSpace



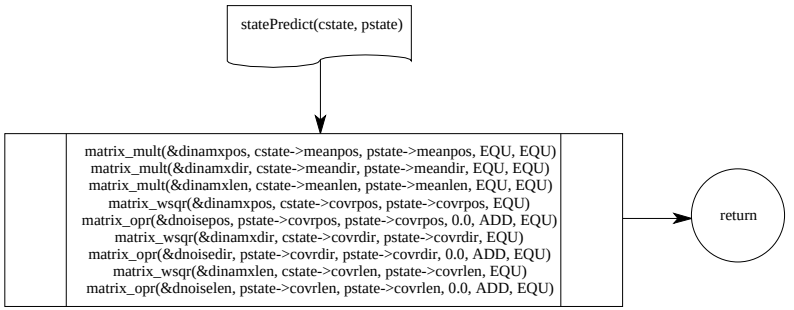
3.2.21 - Módulo RemoveStateSpace



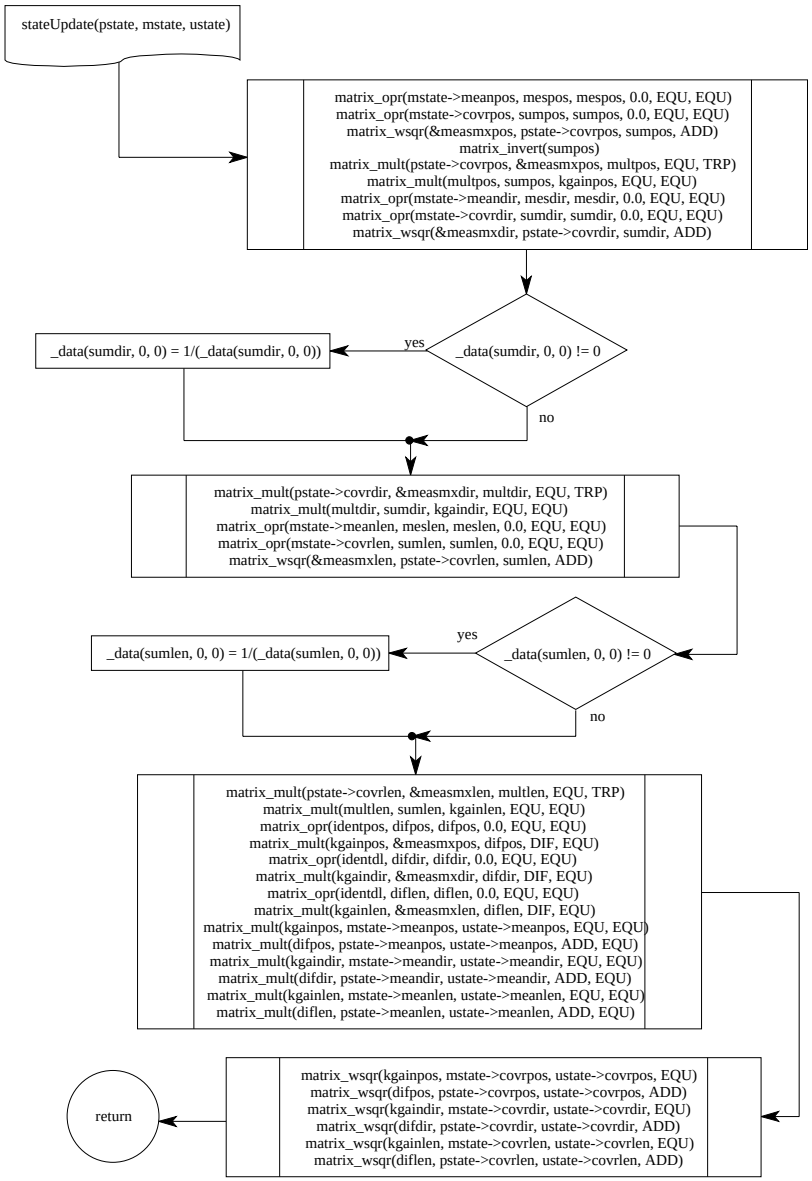
3.2.22 - Módulo statePredictCovrpos



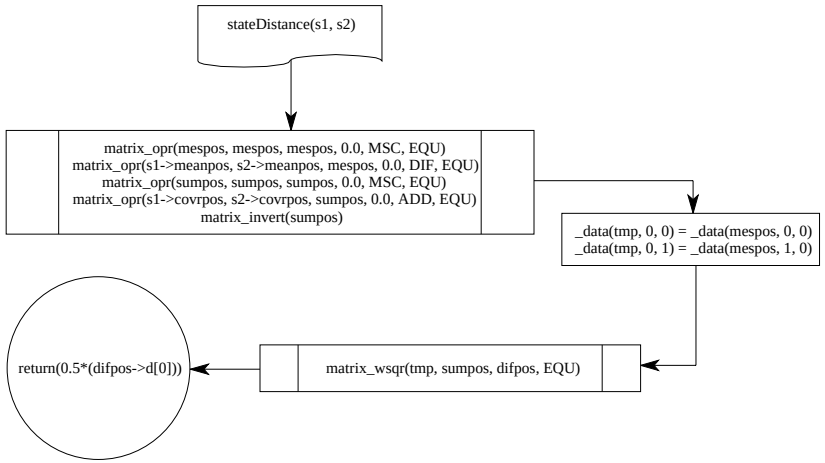
3.2.23 - Módulo statePredict



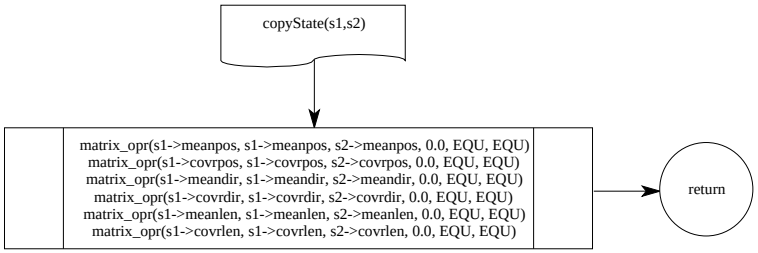
3.2.24 - Módulo stateUpdate



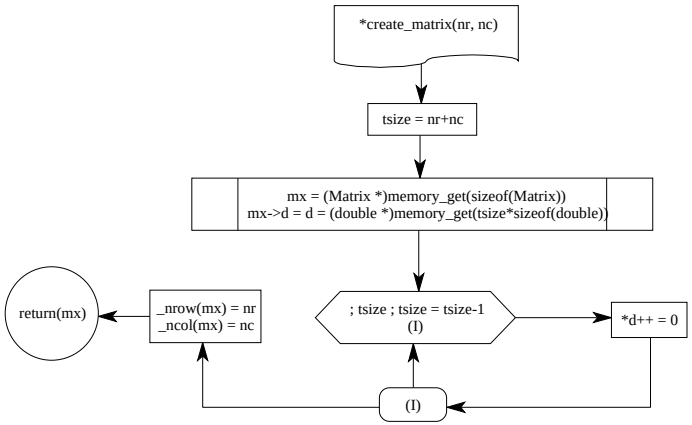
3.2.25 - Módulo stateDistance



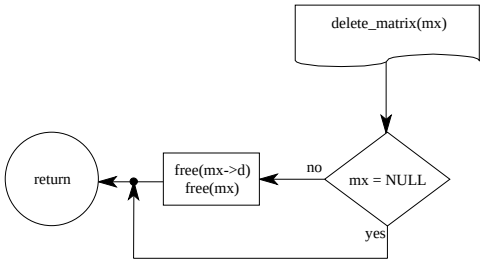
3.2.26 - Módulo copyState



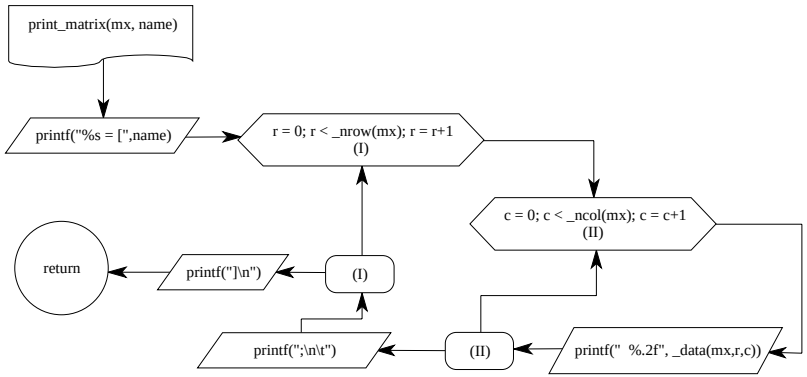
3.2.27 - Módulo create_matrix



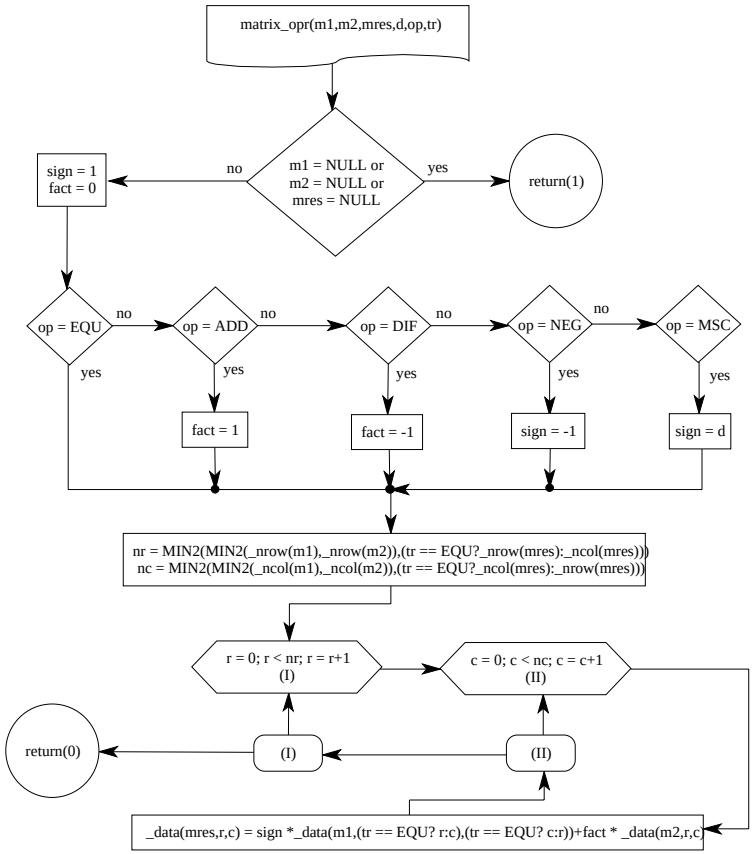
3.2.28 - Módulo delete_matrix



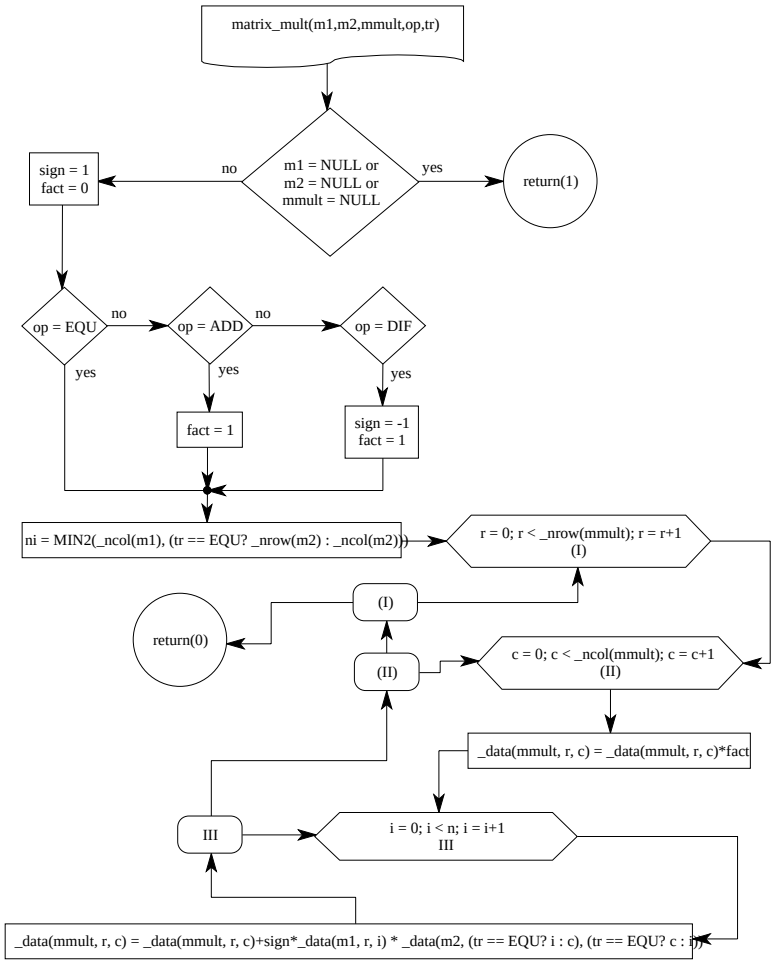
3.2.29 - Módulo print_matrix



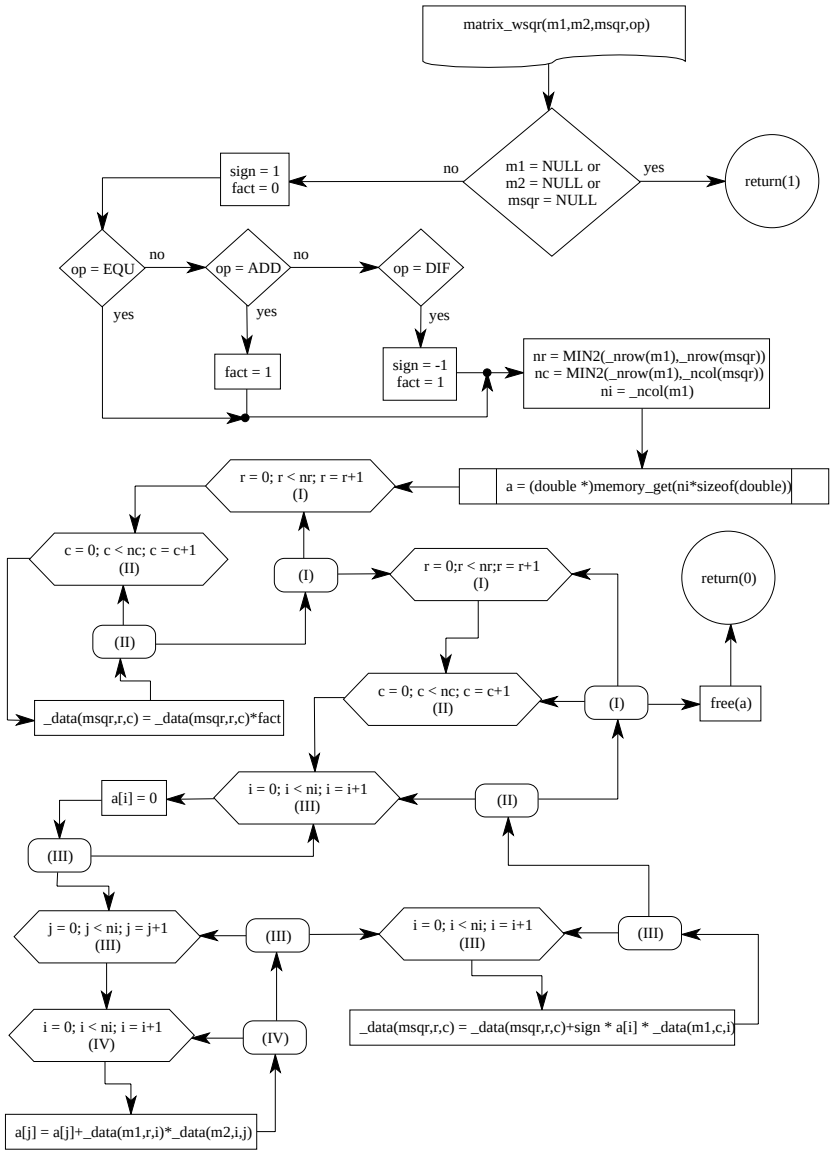
3.2.30 - Módulo matrix_opr



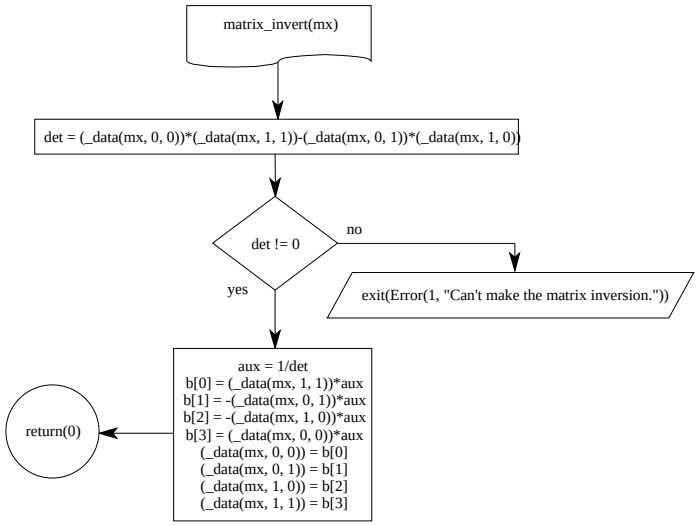
3.2.31 - Módulo matrix_mult



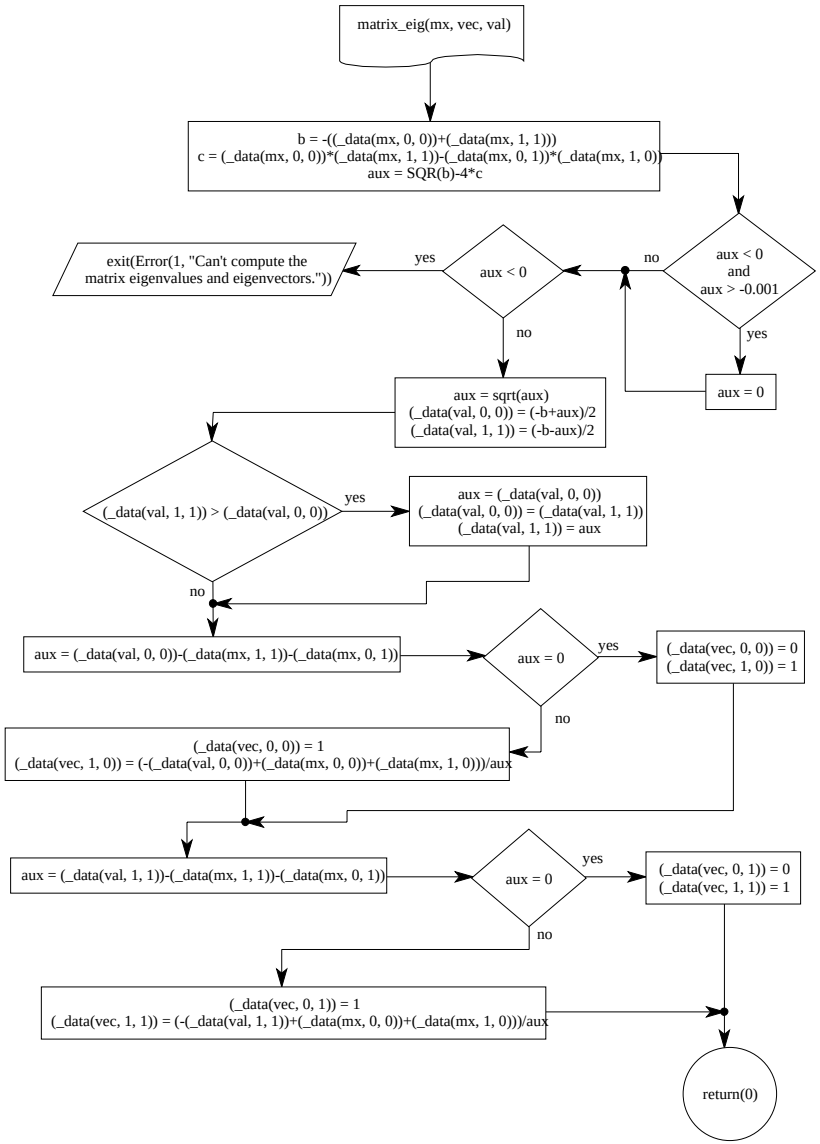
3.2.32 - Módulo matrix_wsqr



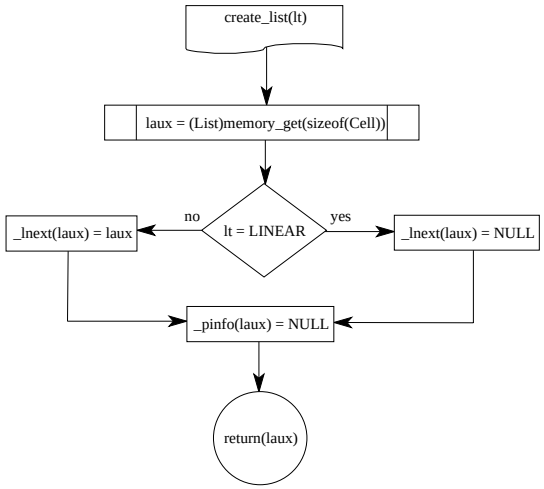
3.2.33 - Módulo matrix_invert



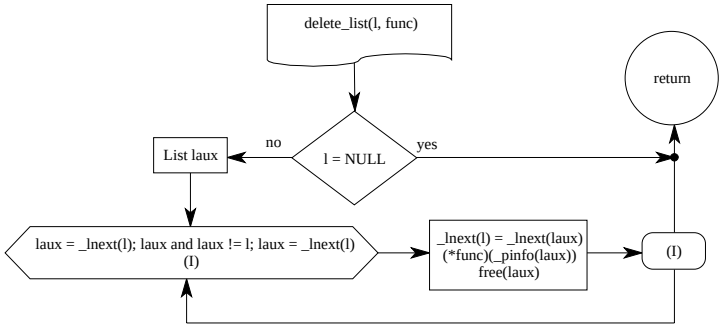
3.2.34 - Módulo matrix_eig



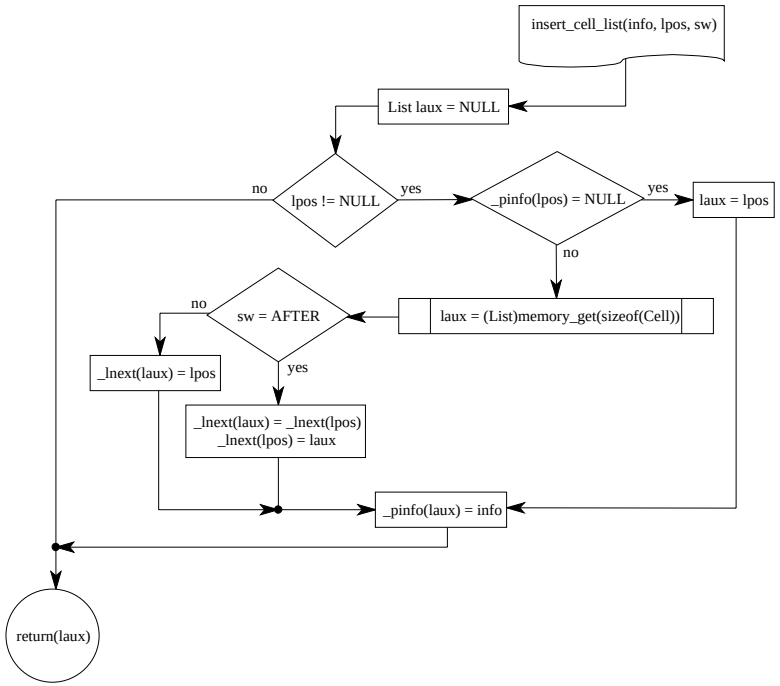
3.2.35 - Módulo create_list



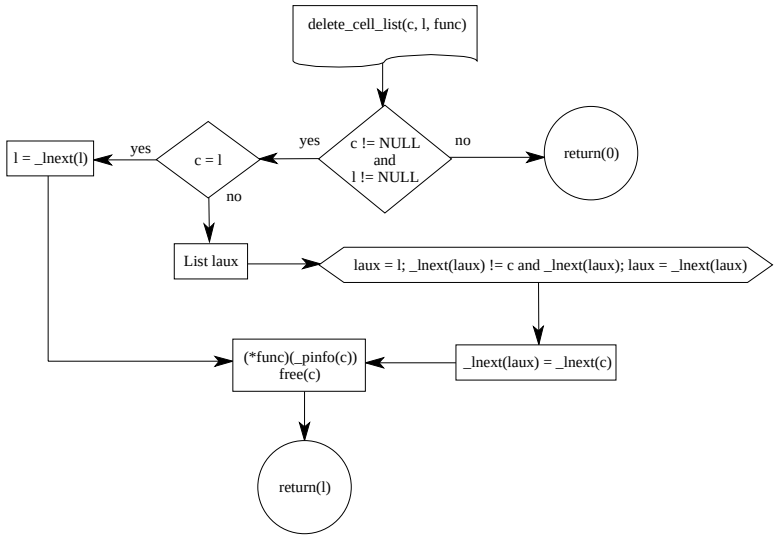
3.2.36 - Módulo delete_list



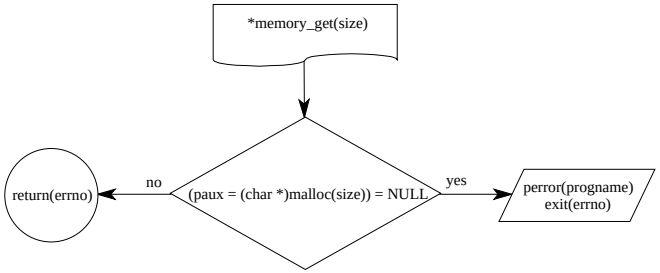
3.2.37 - Módulo insert_cell_list



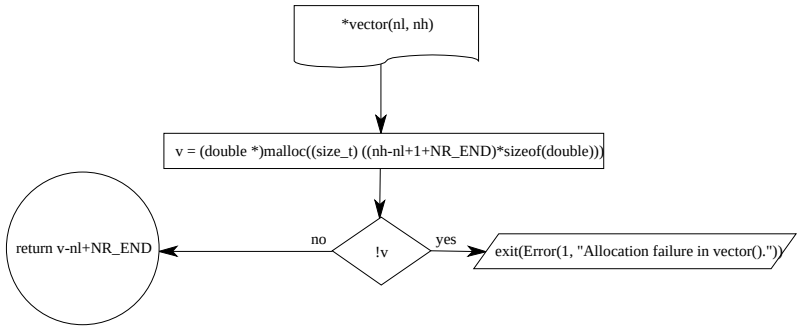
3.2.38 - Módulo delete_cell_list



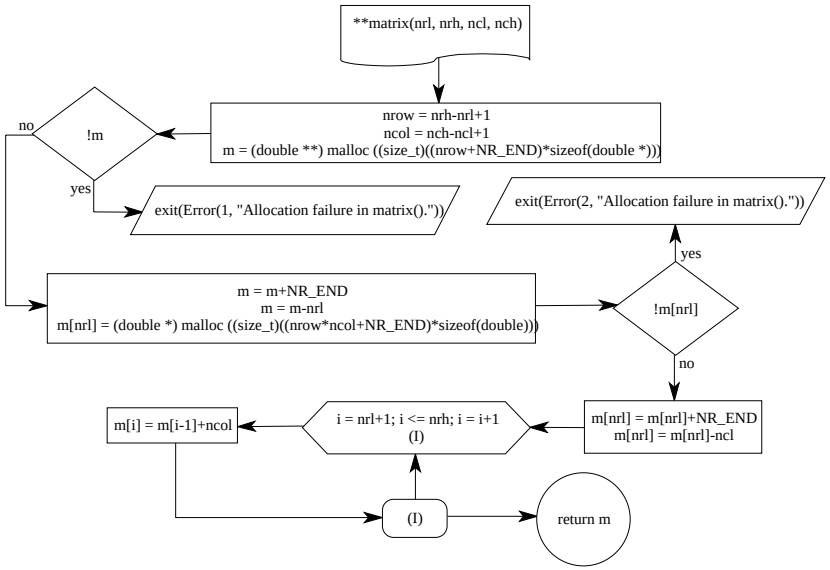
3.2.39 - Módulo memory_get



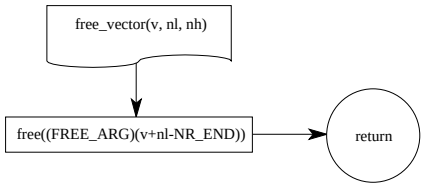
3.2.40 - Módulo vector



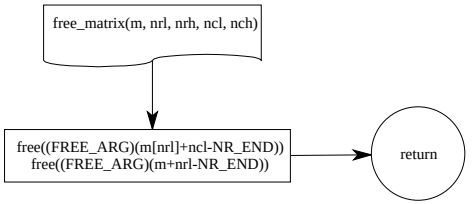
3.2.41 - Módulo matrix



3.2.42 - Módulo free_vector



3.2.43 - Módulo free_matrix



3.3 - Listagem da implementação

Apresenta-se, de seguida, a listagem da implementação ***deep*** desenvolvida em linguagem *C* em ambiente *UNIX* numa estação de trabalho *SunSparc* e que se destina ao ambiente de processamento de imagem *XITE*[13], do qual utiliza algumas funções e tipos de variáveis já previamente definidas. Esta implementação contém comentários para auxílio à sua compreensão, assim como informação sobre os diferentes módulos, para sua posterior extracção através da ferramenta *CDOC(1)* existente no *XITE*.

Com o objectivo de tornar esta implementação mais simples, facilmente entendível e o mais reutilizável possível, optou-se por dividi-la em vários ficheiros. Assim, cada grupo de módulos que constitui a implementação constitui um ficheiro. Além destes ficheiros, existem os ficheiros associados, que contêm os protótipos dos módulos. Este ficheiros são utilizados para incluir os módulos num qualquer ficheiro que os utilize. Existe também um ficheiro para definição de macros e de algumas variáveis de uso global, ***gdefs.h***. Deste modo, esta

[13] *X-based Image processing Tools an Environmemt*, [Lønnestad, 1992].

implementação é constituída pelos seguintes ficheiros: ***deep.c***, ***func_3d.h***, ***func_3d.c***, ***linleasq.h***, ***linleasq.c***, ***states.h***, ***states.c***, ***model.h***, ***model.c***, ***matrix.h***, ***matrix.c***, ***list.h***, ***list.c***, ***memory.h***, ***memoy.c*** e ***gdefs.h***.

3.2.1 - Ficheiro deep.c

Este ficheiro contém a listagem relativa aos módulos que pertencem ao grupo ***I*** da implementação ***deep***.

```
/*
_____________________________________________________________________

  deep.c
  @(#)deep.c   1.00 94/05/22, Copyright 1994, DEEC, FEUP
  Image processing lab, Department of Electrical and Computer
  Engineering
  University of Porto
  E-mail: gpai@obelix.fe.up.pt
_____________________________________________________________________

  readarg.h
  @(#)readarg.h   1.3 92/01/15, Copyright 1990, Blab, UiO
  Image processing lab, Department of Informatics
  University of Oslo
  E-mail: blab@ifi.uio.no
_____________________________________________________________________

  message.h
  @(#)messahe.h   1.3 92/01/15, Copyright 1990, Blab, UiO
  Image processing lab, Department of Informatics
  University of Oslo
  E-mail: blab@ifi.uio.no
_____________________________________________________________________

  biff.h
  @(#)biff.h   1.3 92/01/15, Copyright 1990, Blab, UiO
  Image processing lab, Department of Informatics
  University of Oslo
  E-mail: blab@ifi.uio.no
_____________________________________________________________________

  Permission to use, copy, modify and distribute this software and its documentation for any purpose
and without fee is hereby granted, provided that this copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in supporting documentation and that the
name of B-lab, Department of Informatics or University of Oslo not be used in advertising or publicity
pertaining to distribution of the software without specific, written prior permission.

  B-LAB DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT
SHALL B-LAB BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL
DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA
OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER
TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
*/

/************************************************************************/

/* INCLUDES */

#include <stdio.h>
#include <math.h>
#include <blab/biff.h>
#include <blab/readarg.h>
#include <blab/message.h>
#include "gdefs.h"
#include "memory.h"
#include "matrix.h"
#include "list.h"
#include "states.h"
#include "model.h"
#include "func_3d.h"

/************************************************************************/

/* DEFINES */

#define DXMAX 7.0              /* maximum displacement for x position */
#define DYMAX 2.0              /* maximum displacement for y position */
#define DIRMAX 1.5             /* maximum displacement for direction */
#define LENMAX 2.5             /* maximum displacement for length */
#define POSXCOVR 10.0          /* covariance for x position */
#define POSYCOVR 10.0          /* covariance for y position */
#define DIRCOVR 2.5            /* covariance for direction */
#define LENCOVR 10.0           /* covariance for length */
#define MAXLENDIST 7.8794      /* maximum length difference for matching */
```

```
#define MAXDIRDIST  6.6349   /* maximum direction difference for matching */
#define MAXPOSDIST 9.2103   /* maximum position distance for matching */
#define LOWLEVEL 35             /* not used grey levels for deep */
#define NEARPAR 20              /* maximum difference in direction (in degree) between a line and
any ends point's matching lines for considered the lines near parallel in the computing of the 3D
world coordinates */

/*********************************************************************/

/* GLOBAL VARIABLES */

typedef struct {

  unsigned int xif;          /* frame buffer's coordinate x of the line's first point */
  unsigned int yif;          /* frame buffer's coordinate y of the line's first point */
  double xiu;                /* undistorted's coordinate x of the line's first point */
  double yiu;                /* undistorted's coordinate y of the line's first point */
  double xm;                 /* frame buffer's coordinate x of the line's middle point */
  double ym;                 /* frame buffer's coordinate y of the line's middle point */
  unsigned int xff;          /* frame buffer's coordinate x of the line's last point */
  unsigned int yff;          /* frame buffer's coordinate y of the line's last point */
  double xfu;                /* undistorted's coordinate x of the line's last point */
  double yfu;                /* undistorted's coordinate y of the line's last point */
  double l;                  /* line's length */
  double teta;               /* line's direction */
  unsigned int matched;    /* line's flag for indication that the line is matched */
  unsigned int into;         /* line's flag for indication that the line's middle point is inside of one
ellipse */

} lines;

FILE *fptin1, *fptin2, *fptout;          /* file's pointers */

/*********************************************************************/

/*f:matching*
_____________________________________________________________________

  matching
_____________________________________________________________________

Name:    matching - computing the world coordinates of line's end points from
                    the image's coordinates in succesive files (frames)

Syntax:     | int matching(imgout, xsize, ysize, maxdeep, mindeep, nfdeep, f, k,
            | cxf, cyf, dxl, dy, d, nfiles, verb)
            | int xsize, ysize, nfdeep, cxf, cyf, nfiles, verb;
            | double maxdeep, mindeep, f, k, dxl, dy, *d;
            | IMAGE imgout;

Description:  'matching' performs the computing of the world coordinates of
              line's end points from the image's coordinates in successive frames
              (files). The camera's model is the TSAI's model and in the matching
              phase are used Kalman's filters with Mahalanobis's distances or
              geometric's restrictions.
              For the computing of the world's coordinates is used the
              correspondent matching line and the solution of the stereo
              projection's equations by the linear least-squares regression using
              SVD decomposition.
              'imgout' is the output image. This image will have one band for every
              'nfdeep' where the matched lines are drawing with the relation
              deep/grey.
              'xsize' and 'ysize' are the frame buffer's dimensions in directions x
              and y.
              'maxdeep' and 'mindeep' are the deep's maximum and minimum
              values to consider in the line's deep drawing.
              'nfdeep' is the gap between frames (files) for the calculation of the
              world's coordinates.
              'f' is the effective focal length.
              'k' is the lens's radial distortion.
              'cxf' and 'cyf' are the image's center in the frame buffer for directions x and y.
              'dxl' is (dx*sx) where dx is the center to center distance between adjacent
              sensor elements in x (scanline) direction and sx is the horizontal
              uncertainty factor.
              'dy' is the center to center distance between adjacent CCD sensor in the
              y direction.
              'd' is the vector of the parameters for the matching by Kalman's filters with
              geometric's restrictions, d[0] is the maximum distance for line's middle point,
              d[1] is the maximum difference for line's length and d[2] is the maximum
              difference for line's directions.
              'nfiles' is the number of file's names in the input file, passed by it's
              global pointer.
              If 'verb' is 1 the user can see the output's results.
              The output results are writing after every 'nfeep' in the output file if
              it's global pointer is not NULL. The writing's format is: image's
              coordinates, world's coordinates.

Return value:  | 0 => Ok.
               | 1 => Can't open input file <filename>.
               | 2 => The input file <filename>, only has one row; it must have
```

```
                |       at least two.
                | 3 => Not enough memory.

Examples:       | Try yourself.

Restrictions:   Input file must contain more than one name (frame).
                Enough memory.

Author:         Joao Tavares

Id:             @(#)matching.c   1.0    94/05/22
_____________________________________________________________________
*/

int matching(imgout, xsize, ysize, maxdeep, mindeep, nfdeep, f, k, cxf, cyf, dxl, dy, d, nfiles, verb)
int xsize, ysize, nfdeep, cxf, cyf, nfiles, verb;
double maxdeep, mindeep, f, k, dxl, dy, *d;
IMAGE imgout;

{

  register int i, j, p, q;
  int n, xi, yi, xf, yf, aint1, aint2, into, deep;
  double mindist, a, b, vx, vy, cx, cy, ai, bi, vxi, vyi, difpm, difdir, diflen, tx, ty, aux1, aux2, aux3,
aux4;
  float rot[3], trans[3];
  lines *line;
  char filein2[10], s;
  List model, tail1, mpp, tail2;
  Cell *cell, *cell1;
  Line *elmodel;
  Mpp *elmpp1, *elmpp2;
  unsigned int ref, matched, nband;
  Matrix *matrix, *vectmatrix, *valmatrix;
  extern State ps, ms;
  double mahalanobis();
  int measure();
  int compute_deep();

  ref = 0;
  nband = 0;

  /* make 2D model of lines */

  model = tail1 = create_list(LINEAR);

  /* make list of perspective projection matrix */

  mpp = tail2 = create_list(LINEAR);

  /* get memory resources */

  InitStateSpace();

  /* make some auxiliary matrices */

  matrix = create_matrix(2, 2);
  vectmatrix = create_matrix(2, 2);
  valmatrix = create_matrix(2, 2);

  /* compute the initial area of matching */

  /* assign initial covariance's values */

  _data(ms.covrpos, 0, 0) = POSXCOVR;
  _data(ms.covrpos, 1, 1) = POSYCOVR;
  _data(ms.covrpos, 2, 2) = SQR(DXMAX);
  _data(ms.covrpos, 3, 3) = SQR(DYMAX);
  _data(ms.covrpos, 4, 4) = SQR(DXMAX/2.0);
  _data(ms.covrpos, 5, 5) = SQR(DYMAX/2.0);

  /* predict the new state's covariance */

  statePredictCovrpos(&ms, &ps);

  /* compute the area of matching */

  _data(matrix, 0, 0) = _data(ps.covrpos, 0, 0);
  _data(matrix, 0, 1) = _data(ps.covrpos, 0, 1);
  _data(matrix, 1, 0) = _data(ps.covrpos, 1, 0);
  _data(matrix, 1, 1) = _data(ps.covrpos, 1, 1);
  matrix_eig(matrix, vectmatrix, valmatrix);
  ai = ABS(_data(valmatrix, 0, 0));
  bi = ABS(_data(valmatrix, 1, 1));
  vxi = _data(vectmatrix, 0, 0);
  vyi = _data(vectmatrix, 1, 0);

  for (p = 1; p <= nfiles; p++) {

    /* read the name of the input file */
```

```
   fscanf(fptin1, "%s", filein2);

   /* open the input file */

   fptin2 = fopen(filein2, "r");

   if (fptin2 == NULL) exit(Error(1, "\nCan't open input file %s.\n", filein2));

   n = 0;

   /* find the number of lines in the input file */

   fscanf(fptin2, "%f %f %f %f %f %f %d %d %d %d", &rot[0], &rot[1], &rot[2], &trans[0],
&trans[1], &trans[2], &xi, &yi, &xf, &yf);
   while (!feof(fptin2)) {

     n += 1;
     fscanf(fptin2, "%d %d %d %d", &xi, &yi, &xf, &yf);

   }
   if (n < 1) exit(Error(2, "\nThe input file %s, only has one row; it must have at least two.\n",
filein2));

   rewind(fptin2);

   if (n != 0) {

     /* allocation of memory for the line's structure */

     line = (lines *)malloc(n*sizeof(lines));
     if(line == NULL) exit(Error(3,"\nNot enough memory.\n"));

   }

   /* read and compute some features of all the lines in the input file */

   n = 0;
   fscanf(fptin2, "%f %f %f %f %f %f %d %d %d %d", &rot[0], &rot[1], &rot[2], &trans[0],
&trans[1], &trans[2], &xi, &yi, &xf, &yf);
   while (!feof(fptin2)) {

     aint1 = xf-xi;
     aint2 = yf-yi;

     /* compute the line's angle */

     if(aint1 != 0) {

       (*(line+n)).teta = atan((double)(aint2)/(double)(aint1));
       (*(line+n)).teta = ANG_RAD_DEG((*(line+n)).teta);

     }
     else (*(line+n)).teta = 90.0;

     (*(line+n)).xif = xi;
     (*(line+n)).yif = yi;
     (*(line+n)).xff = xf;
     (*(line+n)).yff = yf;

     /* compute the line's middle point */

     (*(line+n)).xm = (double)(xi+xf)/2.0;
     (*(line+n)).ym = (double)(yi+yf)/2.0;

     /* compute the undistorted coordinates of the line's end points */

     frame_undistorted(xi, yi, &((*(line+n)).xiu), &((*(line+n)).yiu), k, dxl, dy, cxf, cyf);
     frame_undistorted(xf, yf, &((*(line+n)).xfu), &((*(line+n)).yfu), k, dxl, dy, cxf, cyf);

     /* compute the line's length */

     (*(line+n)).l = PYTHAG((double)(aint1), (double)(aint2));
     (*(line+n)).into = 0;
     (*(line+n)).matched = 0;

     if (verb == 1) {

        printf("\nInput line %d:\n\n\tImage's coordinates:\tFirst point: x=%d, y=%d,\tLast point:
x=%d, y=%d,\n\tLength=%g,\tDirection:=%g.\n", n+1, xi, yi, xf, yf, (*(line+n)).l, (*(line+n)).teta);
        printf("\nPause, press enter to continue.\n");
        s = getc(stdin);

     }
     n += 1;
     fscanf(fptin2, "%d %d %d %d", &xi, &yi, &xf, &yf);

   }

   /* close this input file */
```

```
  fclose(fptin2);

  /* compute the new perspective projection matrix */

  elmpp1 = create_mpp();
  compute_mpp(elmpp1->m, rot, trans, f, 1);

  if (p != 1) {

    aux1 = p;
    aux2 = nfdeep;
    aux1 = fmod(aux1, aux2);
    if (aux1 == 0.0) deep = 1;
    else deep = 0;
    if (deep) {

      /* write in the output file if desired */

      if (fptout != NULL) fprintf(fptout, "\nImage coordinates (X,Y), World coordinates (X,Y,Z) of
the lines's end points in file: %s\n\n", filein2);

      /* make one band of output image */

      nband += 1;
      imgout[nband] = Imake_band(Iu_byte_typ, xsize, ysize);
      if (imgout[nband] == NULL) exit(Error(4, "\nCan't make band of output image.\n"));

      /* make all the pixels in output band as WHITE */

      for (i = 1; i <= xsize; i++) {

        for (j = 1; j <= ysize; j++) imgout[nband][j][i] = WHITE;

      }

    }

    /* matching the model's elements with the struture's elements by Kalman's filters and
Mahalanobis's distances */

    for (cell = model; cell != NULL; cell = _lnext(cell)) {

      elmodel = (Line *)_pinfo(cell);

      /* predict the new state */

      statePredict(&(elmodel->state), &ps);

      /* compute the area of matching */

      _data(matrix, 0, 0) = _data(ps.covrpos, 0, 0);
      _data(matrix, 0, 1) = _data(ps.covrpos, 0, 1);
      _data(matrix, 1, 0) = _data(ps.covrpos, 1, 0);
      _data(matrix, 1, 1) = _data(ps.covrpos, 1, 1);
      matrix_eig(matrix, vectmatrix, valmatrix);
      a = ABS(_data(valmatrix, 0, 0));
      b = ABS(_data(valmatrix, 1, 1));
      vx = _data(vectmatrix, 0, 0);
      vy = _data(vectmatrix, 1, 0);
      cx = _data(ps.meanpos, 0, 0);
      cy = _data(ps.meanpos, 1, 0);

      /* compute the measure state */

      into = measure(line, n, &ms, &ps, vx, vy, cx , cy, a, b, 0);

      /* try the matching */

      matched = 0;
      if (into != 0) {

        mindist = MAXDIRDIST*MAXLENDIST*MAXPOSDIST;
        for (j = 0; j < n; j++) {

          if ((*(line+j)).into == 1) {

            (*(line+j)).into = 0;
            if ((*(line+j)).matched == 0) {

              /* compute the MAHALANOBIS distance for length and direction between the line and
the element */

              aux1 = mahalanobis((*(line+j)).l, _data(ms.covrlen, 0, 0), _data(ps.meanlen, 0, 0),
_data(ps.covrlen, 0, 0));
              aux2 = mahalanobis((*(line+j)).teta, _data(ms.covrdir, 0, 0), _data(ps.meandir, 0, 0),
_data(ps.covrdir, 0, 0));
              if (aux1 <= MAXLENDIST && aux2 <= MAXDIRDIST ) {
```

```
            /* compute the MAHALANOBIS distance of the line's state and the element's predict
state */

            _data(ms.meanpos, 0, 0) = (*(line+j)).xm;
            _data(ms.meanpos, 1, 0) = (*(line+j)).ym;
            aux3 = stateDistance(&ps, &ms);
            if (aux3 <= MAXPOSDIST) {

              aux1 = aux1*aux2*aux3;

              if (aux1 <= mindist) {

                aint2 = j;
                matched = 1;
                mindist = aux1;

              }

            }

          }

        }

      }

    }
    if (matched == 1) {

      /* update the element of the model */

      if (verb == 1) {

        printf("\nModel's element with reference: %d, matched by Kalman's filters with
Mahalanobis's distances.\n", elmodel->reference);
        printf("\n\n\tOld image's coordinates:\tFirst point: x=%d, y=%d,\tLast point: x=%d,
y=%d.\n", elmodel->xifnew, elmodel->yifnew, elmodel->xffnew, elmodel->yffnew);
        printf("\n\tNew image's coordinates:\tFirst point: x=%d, y=%d,\tLast point: x=%d,
y=%d.\n", (*(line+aint2)).xif, (*(line+aint2)).yif, (*(line+aint2)).xff, (*(line+aint2)).yff);
        printf("\nPause, press enter to continue.\n");
        s = getc(stdin);

      }

      (*(line+aint2)).matched = 1;

      /* find if the end's points are inverted */

      aux1 = ((double)(elmodel->xifnew+elmodel->xffnew))/2.0;
      aux2 = ((double)(elmodel->yifnew+elmodel->yffnew))/2.0;
      tx = (*(line+aint2)).xm-aux1;
      ty = (*(line+aint2)).ym-aux2;
      aux1 = (double)((*(line+aint2)).xif-(elmodel->xifnew+tx));
      aux2 = (double)((*(line+aint2)).yif-(elmodel->yifnew+ty));
      aux3 = PYTHAG(aux1, aux2);
      aux1 = (double)((*(line+aint2)).xff-(elmodel->xifnew+tx));
      aux2 = (double)((*(line+aint2)).yff-(elmodel->yifnew+ty));
      aux4 = PYTHAG(aux1, aux2);
      if (aux3 < aux4) {

        if (elmodel->invert == 2) elmodel->invert = 0;

      }
      else {

        if (elmodel->invert == 2) elmodel->invert = 1;
        else {

          if (elmodel->invert == 1) elmodel->invert = 0;
          else elmodel->invert = 1;

        }

      }

      /* update the element */

      elmodel->matched = 1;
      elmodel->xifnew = (*(line+aint2)).xif;
      elmodel->yifnew = (*(line+aint2)).yif;
      elmodel->xffnew = (*(line+aint2)).xff;
      elmodel->yffnew = (*(line+aint2)).yff;
      elmodel->xiunew = (*(line+aint2)).xiu;
      elmodel->yiunew = (*(line+aint2)).yiu;
      elmodel->xfunew = (*(line+aint2)).xfu;
      elmodel->yfunew = (*(line+aint2)).yfu;
      _data(ms.meanpos, 0, 0) = (*(line+aint2)).xm;
      _data(ms.meanpos, 1, 0) = (*(line+aint2)).ym;
      _data(ms.meandir, 0, 0) = (*(line+aint2)).teta;
```

```
        _data(ms.meanlen, 0, 0) = (*(line+aint2)).l;

        /* update the 2D model */

        stateUpdate(&ps, &ms, &(elmodel->state));

       }

      }

     /* try the matching of the 2D model's elements not yet matched by Kalman's filters with
geometric restrictions */

     for (cell = model; cell != NULL; cell = _lnext(cell)) {

       elmodel = (Line *)_pinfo(cell);

       if (elmodel->matched == 0) {

         /* try the matching of this element */

         /* predict the new state */

         statePredict(&(elmodel->state), &ps);

         matched = into = 0;
         mindist = d[0]*d[1]*d[2];
         for (i = 0 ; i < n ; i++) {

           if ((*(line+i)).matched == 0) {

             /* compute the distance between the line's middle point and the predict element's middle
point */

             aux1 = _data(ps.meanpos, 0, 0)-(*(line+i)).xm;
             aux2 = _data(ps.meanpos, 1, 0)-(*(line+i)).ym;
             difpm = PYTHAG(aux1, aux2);

             /* compute the difference between the line's angle and the predict element's angle */

             difdir = ABS(_data(ps.meandir, 0, 0)-(*(line+i)).teta);
             if (difdir > 90.0) difdir = 180.0-difdir;

             /* compute the difference between the line's length and the predict element's length */

             diflen = ABS(_data(ps.meanlen, 0, 0)-(*(line+i)).l);

             /* try the matching */

             if (difpm <= d[0] && diflen <= d[1] && difdir <= d[2]) {

               aux1 = difdir*diflen*difpm;
               if (aux1 <= mindist) {

                 matched = 1;
                 mindist = aux1;
                 aint2 = i;

               }

             }

           }

         }
         if (matched == 1) {

           /* compute the area of matching */

           _data(matrix, 0, 0) = _data(ps.covrpos, 0, 0);
           _data(matrix, 0, 1) = _data(ps.covrpos, 0, 1);
           _data(matrix, 1, 0) = _data(ps.covrpos, 1, 0);
           _data(matrix, 1, 1) = _data(ps.covrpos, 1, 1);
           matrix_eig(matrix, vectmatrix, valmatrix);
           a = ABS(_data(valmatrix, 0, 0));
           b = ABS(_data(valmatrix, 1, 1));
           vx = _data(vectmatrix, 0, 0);
           vy = _data(vectmatrix, 1, 0);
           cx = (*(line+aint2)).xm;
           cy = (*(line+aint2)).ym;

           /* compute the measure state */

           into = measure(line, n, &ms, &ps, vx, vy, cx, cy, a, b, 1);

         }
         if (into != 0) {

           /* update the element of the model */
```

```
        if (verb == 1) {

          printf("\nModel's element with reference: %d, matched by Kalman's filters with
geometric's restrictions.\n", elmodel->reference);
          printf("\n\n\tOld image's coordinates:\tFirst point: x=%d, y=%d,\tLast point: x=%d,
y=%d.\n", elmodel->xifnew, elmodel->yifnew, elmodel->xffnew, elmodel->yffnew);
          printf("\n\tNew image's coordinates:\tFirst point: x=%d, y=%d,\tLast point: x=%d,
y=%d.\n", (*(line+aint2)).xif, (*(line+aint2)).yif, (*(line+aint2)).xff, (*(line+aint2)).yff);
          printf("\nPause, press enter to continue.\n");
          s = getc(stdin);

        }

        (*(line+aint2)).matched = 1;

        /* find if the end's points are inverted */

        aux1 = ((double)(elmodel->xifnew+elmodel->xffnew))/2.0;
        aux2 = ((double)(elmodel->yifnew+elmodel->yffnew))/2.0;
        tx = (*(line+aint2)).xm-aux1;
        ty = (*(line+aint2)).ym-aux2;
        aux1 = (double)((*(line+aint2)).xif-(elmodel->xifnew+tx));
        aux2 = (double)((*(line+aint2)).yif-(elmodel->yifnew+ty));
        aux3 = PYTHAG(aux1, aux2);
        aux1 = (double)((*(line+aint2)).xff-(elmodel->xifnew+tx));
        aux2 = (double)((*(line+aint2)).yff-(elmodel->yifnew+ty));
        aux4 = PYTHAG(aux1, aux2);
        if (aux3 < aux4) {

          if (elmodel->invert == 2) elmodel->invert = 0;

        }
        else {

          if (elmodel->invert == 2) elmodel->invert = 1;
          else {

            if (elmodel->invert == 1) elmodel->invert = 0;
            else elmodel->invert = 1;

          }

        }

        /* update the element */

          elmodel->matched = 1;
          elmodel->xifnew = (*(line+aint2)).xif;
          elmodel->yifnew = (*(line+aint2)).yif;
          elmodel->xffnew = (*(line+aint2)).xff;
          elmodel->yffnew = (*(line+aint2)).yff;
          elmodel->xiunew = (*(line+aint2)).xiu;
          elmodel->yiunew = (*(line+aint2)).yiu;
          elmodel->xfunew = (*(line+aint2)).xfu;
          elmodel->yfunew = (*(line+aint2)).yfu;
          _data(ms.meanpos, 0, 0) = (*(line+aint2)).xm;
          _data(ms.meanpos, 1, 0) = (*(line+aint2)).ym;
          _data(ms.meandir, 0, 0) = (*(line+aint2)).teta;
          _data(ms.meanlen, 0, 0) = (*(line+aint2)).l;

          /* update the 2D model */

          stateUpdate(&ps, &ms, &(elmodel->state));

        }
        else {

          /* assume large values for the covariance's and the predicted mean values */

          if (verb == 1) {

            printf("\nModel's element with reference: %d, not matched.\n", elmodel->reference);
            printf("\nPause, press enter to continue.\n");
            s = getc(stdin);

          }

          _data(ms.meanpos, 0, 0) = _data(ps.meanpos, 0, 0);
          _data(ms.meanpos, 1, 0) = _data(ps.meanpos, 1, 0);
          _data(ms.meandir, 0, 0) = _data(ps.meandir, 0, 0);
          _data(ms.meanlen, 0, 0) = _data(ps.meanlen, 0, 0);
          _data(ms.covrpos, 0, 0) = POSXCOVR;
          _data(ms.covrpos, 1, 1) = POSYCOVR;
          _data(ms.covrpos, 0, 1) = _data(ms.covrpos, 1, 0) = 0.0;
          _data(ms.covrdir, 0, 0) = DIRCOVR;
          _data(ms.covrlen, 0, 0) = LENCOVR;

          /* update the element of the model */

          stateUpdate(&ps, &ms, &(elmodel->state));
```

```
        }

       }

      }

     /* update the confidence factor of the 2D model's elements and computer the world coordinates
of the line's end points */

     aint2 = -1;
     for (cell = model; cell != NULL; cell = _lnext(cell)) {

       elmodel = (Line *)_pinfo(cell);

       if (elmodel->matched == 1) {

         elmodel->matched = 0;
         if ((elmodel->cf) < 5) (elmodel->cf) += 1;

         if (elmodel->cf >= 3 && deep) {

           aint1 = elmodel->lastmpp;
           if ((p-aint1+1) >= nfdeep) {

             /* compute the deep */

             if (aint2 != aint1) {

               /* find the element's last perspective projection matrix */

               cell1 = mpp;
               for (i = 1; i < aint1; i++) cell1 = _lnext(cell1);
               elmpp2 = (Mpp *)_pinfo(cell1);
               aint2 = aint1;

             }

             /* call compute_deep function */

             compute_deep(imgout[nband], elmodel, elmpp1->m, elmpp2->m, maxdeep, mindeep,
verb);

             /* update some variables of this element */

             elmodel->xiuold = elmodel->xiunew;
             elmodel->yiuold = elmodel->yiunew;
             elmodel->xfuold = elmodel->xfunew;
             elmodel->yfuold = elmodel->yfunew;
             elmodel->lastmpp = p;
             elmodel->invert = 0;

           }

         }
         else {

           (elmodel->cf) -= 1;
           if ((elmodel->cf) < 0) model = delete_cell_list(cell, model, delete_line);

         }

       }

     }

     /* insert the last perspective projection matrix into the mpp list */

     tail2 = insert_cell_list(elmpp1, tail2, AFTER);

     /* update the 2D model with new elements */

     for (i = 0; i < n; i++) {

       if ((*(line+i)).matched == 0) {

         ref += 1;

         if (verb == 1) {

           printf("\nInserting a new element into the 2d model with reference: %d.\n", ref);
           printf("\nPause, press enter to continue.\n");
           s = getc(stdin);

         }

         /* create one element of the 2D model */
```

```
elmodel = create_line();

elmodel->reference = ref;
elmodel->cf = 3;
elmodel->xifnew = (*(line+i)).xif;
elmodel->yifnew = (*(line+i)).yif;
elmodel->xffnew = (*(line+i)).xff;
elmodel->yffnew = (*(line+i)).yff;
elmodel->matched = 0;
elmodel->xiuold = (*(line+i)).xiu;
elmodel->yiuold = (*(line+i)).yiu;
elmodel->xfuold = (*(line+i)).xfu;
elmodel->yfuold = (*(line+i)).yfu;
elmodel->lastmpp = p;
elmodel->invert = 2;

/* assign initial mean values */

_xpos(elmodel) = (*(line+i)).xm;
_ypos(elmodel) = (*(line+i)).ym;
_xvel(elmodel) = 0.0;
_yvel(elmodel) = 0.0;
_xacl(elmodel) = 0.0;
_yacl(elmodel) = 0.0;
_dir(elmodel) = (*(line+i)).teta;
_dirvel(elmodel) = 0.0;
_diracl(elmodel) = 0.0;
_len(elmodel) = (*(line+i)).l;
_lenvel(elmodel) = 0.0;
_lenacl(elmodel) = 0.0;

/* assign initial variances values */

_xpos_cv(elmodel) = POSXCOVR;
_ypos_cv(elmodel) = POSYCOVR;
_xvel_cv(elmodel) = SQR(DXMAX);
_yvel_cv(elmodel) = SQR(DYMAX);
_xacl_cv(elmodel) = SQR(DXMAX/2.0);
_yacl_cv(elmodel) = SQR(DYMAX/2.0);
_dir_cv(elmodel) = DIRCOVR;
_dirvel_cv(elmodel) = SQR(DIRMAX);
_diracl_cv(elmodel) = SQR(DIRMAX/2.0);
_len_cv(elmodel) = LENCOVR;
_lenvel_cv(elmodel) = SQR(LENMAX);
_lenacl_cv(elmodel) = SQR(LENMAX/2.0);

   /* predict the new state */

   statePredict(&(elmodel->state), &ps);

   /* compute the measure state for this element */

   cx = (*(line+i)).xm;
   cy = (*(line+i)).ym;

   measure(line, n, &ms, &ps, vxi, vyi, cx, cy, ai, bi, 1);

   /* update the state of this element */

   _data(ms.meanpos, 0, 0) = (*(line+i)).xm;
   _data(ms.meanpos, 1, 0) = (*(line+i)).ym;
   _data(ms.meandir, 0, 0) = (*(line+i)).teta;
   _data(ms.meanlen, 0, 0) = (*(line+i)).l;
   stateUpdate(&ps, &ms, &(elmodel->state));

   /* into the 2D model list after the last one */

   tail1 = insert_cell_list(elmodel, tail1, AFTER);

  }

}

/* free memory of line's struture */

if (line != NULL) free(line);

/* if desired print state of the 2D model */

if (verb == 1) {

  printf("\nThe 2D lines's model is like this:\n\n");
  for (cell = model; cell != NULL; cell = _lnext(cell)) {

    elmodel = (Line *)_pinfo(cell);
    print_line(elmodel);

  }
  if (p == nfiles) printf("\nEND deep.\n");
```

```
    }

  }

  /* free memory resources */

  RemoveStateSpace();

  /* delete auxiliary matrices */

  delete_matrix(matrix);
  delete_matrix(vectmatrix);
  delete_matrix(valmatrix);

  /* free 2D model list of lines */

  delete_list(model, delete_line);

  /* free mpp list */

  delete_list(mpp, delete_mpp);

  return(0);

}

/*************************************************************************/

/*F:mahalanobis*
_______________________________________________________________________

  mahalanobis
_______________________________________________________________________

Name:         mahalanobis - computes the Mahalanobis's distance for one variable

Syntax:       | double mahalanobis(am, vm, ap, vp)
              | double am, vm, ap, vp;

Description:  'mahalanobis' performs the calculation of the Mahalanobis's
              distance between two states for only one variable.
              The variable have the mean value 'am' and the variance 'vm' in one
              state and the mean value 'ap' and the variance 'vp' in the other state.

Return value: | The Mahalanobis's distance.

Example:      | Try yourself.

Restrictions: None.

Author:       Joao Tavares

Id:           @(#)mahalanobis.c   1.00   94/05/22
_______________________________________________________________________
*/

double mahalanobis(am, vm, ap, vp)
double am, vm, ap, vp;

{

  double aux;

  /* compute the MAHALANOBIS's distance */

  aux = vm+vp;
  if (aux == 0.0) aux = SQR(am-ap);
  else aux = SQR(am-ap)/aux;

  return(0.5*aux);

}

/*************************************************************************/

/*F:measure*
_______________________________________________________________________

  measure
_______________________________________________________________________

Name:         measure - computes the measure state

Syntax:       | int measure(line, n, ms, ps, vx, vy, cx, cy, a, b, mark)
              | int n, mark;
              | double vx, vy, cx, cy, a, b;
              | lines *line;
              | State *ms, *ps;

Description:  'measure' performs the calculation of the 'ms' measure state.
```

```
                'line' is the pointer to the line's strutur with 'n' lines.
                'ps' is the predict state.
                'vx' and 'vy' are the x and y components of the biggest axis's vector
                of the matching's ellipse.
                'cx' and 'cy' are the coordinates x and y of the matching ellipse's
                center.
                'a' and 'b' are the dimensions of the bigger and smallest matching
                ellipse's radius.
                If 'mark' is 1 the line's into flag is clear.

Return value:   | The number of lines of which the middle point's are inside the
                | ellipse.

Example:        | Try yourself.

Restrictions:   None.

Author:         Joao Tavares

Id:             @(#)measure.c   1.00   94/05/22
_____________________________________________________________________
*/

int measure(line, n, ms, ps, vx, vy, cx, cy, a, b, mark)
int n, mark;
double vx, vy, cx, cy, a, b;
lines *line;
State *ms, *ps;

{

  int aint1;
  register int i, j;
  double aux1, meddir, medlen, covrdir, covrlen, **m;
  double mahalanobis();
  int elipse();

  /* allocation of memory to the moment's matrix */

  m = matrix(0, 2, 0, 2);

  /* compute the direction and length average */

  aint1 = 0;
  meddir = medlen = 0.0;

  for (j = 0; j < n ; j++) {

   if (elipse(vx, vy, cx, cy, a, b, (*(line+j)).xm, (*(line+j)).ym)) {

     aint1 += 1;
     (*(line+j)).into = 1;
     meddir += (*(line+j)).teta;
     medlen += (*(line+j)).l;

   }

  }
  if (aint1 != 0) {

   meddir /= aint1;
   medlen /= aint1;

   /* compute the direction and length variance */

   covrdir = covrlen = 0.0;
   for (j = 0; j < n; j++) {

     if ((*(line+j)).into == 1) {

       covrdir += SQR((*(line+j)).teta-meddir);
       covrlen += SQR((*(line+j)).l-medlen);

     }

   }
   covrdir /= aint1;
   covrlen /= aint1;
   _data(ms->covrdir, 0, 0) = covrdir;
   _data(ms->covrlen, 0, 0) = covrlen;

   /* compute the moments inside the area of matching */

   for (j = 0; j < 3; j++) {

     for(i = 0; i < 3; i++) m[i][j] = 0.0;

   }
   for (j = 0; j < n; j++) {

     if ((*(line+j)).into == 1) {
```

```
      if (mark == 1) (*(line+j)).into = 0;
      aux1 = mahalanobis((*(line+j)).l, covrlen, _data(ps->meanlen, 0, 0), _data(ps->covrlen, 0, 0));
      aux1 += mahalanobis((*(line+j)).teta, covrdir, _data(ps->meandir, 0, 0), _data(ps->covrdir, 0,
0));
      aux1 = exp((double)(-aux1));
      m[0][0] += aux1;
      m[1][0] += aux1*(*(line+j)).xm;
      m[0][1] += aux1*(*(line+j)).ym;
      m[1][1] += aux1*((*(line+j)).xm)*((*(line+j)).ym);
      m[2][0] += aux1*SQR((*(line+j)).xm);
      m[0][2] += aux1*SQR((*(line+j)).ym);

    }

   }

   if (m[0][0] != 0.0) {

     /* compute the measure position variances */

    _data(ms->covrpos, 0, 0) = (m[2][0]-SQR(m[1][0])/m[0][0])/m[0][0];
    _data(ms->covrpos, 1, 1) = (m[0][2]-SQR(m[0][1])/m[0][0])/m[0][0];
    _data(ms->covrpos, 0, 1) = _data(ms->covrpos, 1, 0) = (m[1][1]-m[1][0]*m[0][1]/
m[0][0])/m[0][0];

   }
   else aint1 = 0;

 }

 /* free the memory of the moment's matrix */

 free_matrix(m, 0, 2, 0, 2);

 return(aint1);

}

/*****************************************************************************/

/*F:elipse*
_____________________________________________________________________

 elipse
```

_____

Name: elipse - determination if one point is inside of an ellipse

Syntax: | int elipse(vx, vy, cx, cy, a, b, px, py)
| double vx, vy, cx, cy, a, b, px, py;

Description: 'elipse' performs the determination if the point with coordinates x='px' and y='py' is inside the ellipse with center point's coordinates x='cx' and y='cy'.
'a' and 'b' are the dimensions of the biggest and smallest ellipse's radius.
'vx' and 'vy' are the x and y components of the biggest axis's vector.

Return value: | 0 => The point is outside.
| 1 => The point is inside.

Example: | Try yourself.

Restrictions: None.

Author: Joao Tavares

Id: @(#)elipse.c 1.00 94/05/22

_____

```
*/

int elipse(vx, vy, cx, cy, a, b, px, py)
double vx, vy, cx, cy, a, b, px, py;

{

 double teta, pxrot, pyrot, aux;
 int answer;

 /* find if the point (px, py) is inside the ellipse */

 if (a == b) aux = (SQR(px-cx)+SQR(py-cy))/SQR(a);
 else {

  if (vx == 0.0) teta = API;
  else teta = atan(vy/vx);
  pxrot = cx+(px-cx)*cos(teta)+(py-cy)*sin(teta);
  pyrot = cy-(px-cx)*sin(teta)+(py-cy)*cos(teta);
```

```
    aux = (SQR(pxrot-cx))/(SQR(a))+(SQR(pyrot-cy))/(SQR(b));

  }
  if (aux > 1.0) answer = 0;
  else answer = 1;

  return(answer);

}

/*************************************************************************/

/*F:compute_deep*
_____________________________________________________________________

  compute_deep
_____________________________________________________________________

Name:  compute_deep - determination of the 3D world's coordinates of line's
                      end points

Syntax:       | int compute_deep(band, ep, mppnew, mppold, maxdeep,
              | mindeep, verb)
              | Line *ep;
              | double mppnew[][4], mppold[][4], maxdeep, mindeep;
              | int verb;
              | IBAND band;

Description:  'compute_deep' performs the determination of the 3D world's
              coordinates of the 'ep' element's end points.
              'band' is the output band with the draw of the lines. The line's
              points are draw with greys correspondent to it's deep.
              'mppnew' is the new perspective projection matrix.
              'mppold' is the old perspective projection matrix.
              'maxdeep' and 'mindeep' are the maximum and minimum deep to
              consider for the drawing.
              If 'verb' is 1 the 3D world's coordinates of the element's end points
              are print.

Return value: | 0 => Ok.

Example:      | Try yourself.

Restrictions: None.

Author:       Joao Tavares

Id:           @(#)compute_deep.c   1.00   94/05/22
_____________________________________________________________________
*/

int compute_deep(band, ep, mppnew, mppold, maxdeep, mindeep, verb)
Line *ep;
double mppnew[][4], mppold[][4], maxdeep, mindeep;
int verb;
IBAND band;

{

  double m2, b2, m1, b1, xinti, yinti, xintf, yintf, dx1, dy1, *worldi, *worldf, aux1, aux2, l1, l2,
ang1, ang2, difang;
  int flag1, flag2;
  char s;
  int dda_deep();

  /* allocation of memory to the world's coordinates vectors */

  worldi = vector(0, 2);
  worldf = vector(0, 2);

  flag1 = flag2 = 0;
  dx1 = ((ep)->xfuold)-((ep)->xiuold);
  dy1 = ((ep)->yfuold)-((ep)->yiuold);

  if (dx1 == 0.0) ang1 = 90.0;
  else {

    m1 = dy1/dx1;
    b1 = (ep)->yiuold-m1*((ep)->xiuold);
    ang1 = ANG_RAD_DEG(atan(m1));

  }

  /* find the correspondence between end's points */

  /* compute the matching line of the first point */

  match_line(mppnew, mppold, (ep)->xiunew, (ep)->yiunew, &m2, &b2);
  ang2 = ANG_RAD_DEG(atan(m2));
```

```
difang = fabs(ang1-ang2);
if (difang > 90.0) difang = 180.0-difang;

if (difang != 0.0) {

  /* find the intersection point */

  if (dx1 == 0.0) {

    xinti = (ep)->xiuold;
    yinti = xinti*m2+b2;

  }
  else {

    if (dy1 == 0.0) {

      yinti = (ep)->yiuold;
      xinti = (yinti-b2)/m2;

    }
    else {

      xinti = (b2-b1)/(m1-m2);
      yinti = xinti*m1+b1;

    }

  }

}
if (difang <= NEARPAR) flag1 = 1;

/* compute the matching line of the second point */

match_line(mppnew, mppold, (ep)->xfunew, (ep)->yfunew, &m2, &b2);
ang2 = ANG_RAD_DEG(atan(m2));

difang = fabs(ang1-ang2);
if (difang > 90.0) difang = 180.0-difang;

if (difang != 0.0) {

  /* find the intersection point */

  if (dx1 == 0.0) {

    xintf = (ep)->xfuold;
    yintf = xintf*m2+b2;

  }
  else {

    if (dy1 == 0.0) {

      yintf = (ep)->yfuold;
      xintf = (yintf-b2)/m2;

    }
    else {

      xintf = (b2-b1)/(m1-m2);
      yintf = m1*xintf+b1;

    }

  }

}
if (difang <= NEARPAR) flag2 = 1;

if (flag1 != 0 || flag2 != 0) {

  /* find the true correspondence between end's points */

  if (flag1 == 0) {

    aux1 = xinti-(ep)->xiuold;
    aux2 = yinti-(ep)->yiuold;
    l1 = PYTHAG(aux1, aux2);
    aux1 = xinti-(ep)->xfuold;
    aux2 = yinti-(ep)->yfuold;
    l2 = PYTHAG(aux1, aux2);
    if (l1 < l2) {

      xintf = (ep)->xfuold;
      yintf = (ep)->xfuold;

    }
    else {
```

```
    xintf = (ep)->xiuold;
    yintf = (ep)->xiuold;

   }

  }
  else {

   if (flag2 == 0) {

    aux1 = xintf-(ep)->xiuold;
    aux2 = yintf-(ep)->yiuold;
    l1 = PYTHAG(aux1, aux2);
    aux1 = xintf-(ep)->xfuold;
    aux2 = yintf-(ep)->yfuold;
    l2 = PYTHAG(aux1, aux2);
    if (l2 < l1) {

      xinti = (ep)->xiuold;
      yinti = (ep)->xiuold;

    }
    else {

      xinti = (ep)->xfuold;
      yinti = (ep)->xfuold;

    }

   }
   else {

    if ((ep)->invert == 0) {

      xinti = (ep)->xiuold;
      yinti = (ep)->yiuold;
      xintf = (ep)->xfuold;
      yintf = (ep)->yfuold;

    }
    else {

      xinti = (ep)->xfuold;
      yinti = (ep)->yfuold;
      xintf = (ep)->xiuold;
      yintf = (ep)->yiuold;

     }

    }

   }

  }

  /* compute the world coordinates of the first point */

  stereo_eq(mppold, mppnew, xinti, yinti, (ep)->xiunew, (ep)->yiunew, worldi);

  /* compute the world coordinates of the second point */

  stereo_eq(mppold, mppnew, xintf, yintf, (ep)->xfunew, (ep)->yfunew, worldf);

  /* if desired, output results to the user */

  if (verb == 1) {

    printf("\nFor the model's element with reference %d:\n\n\tWorld's coordinates:\tFirst point:
x=%g, y=%g, z=%g\tLast point: x=%g, y=%g, z=%g.\n", (ep)->reference, worldi[0], worldi[1],
worldi[2], worldf[0], worldf[1], worldf[2]);
    printf("\nPause, press enter to continue.\n");
    s = getc(stdin);

  }

  /* print in the output file the results if desired */

  if (fptout != NULL) {

    fprintf(fptout, "\n\t(%d, %d)\t(%f, %f, %f)", ((ep)->xifnew), ((ep)->yifnew), ((float)(worldi[0])),
((float)(worldi[1])), ((float)(worldi[2])));
    fprintf(fptout, "\n\t(%d, %d)\t(%f, %f, %f)\n", ((ep)->xffnew), ((ep)->yffnew),
((float)(worldf[0])), ((float)(worldf[1])), ((float)(worldf[2])));

  }

  /* write in the output band the results */
```

```
  dda_deep(band, ((ep)->xifnew), ((ep)->yifnew), worldi[2], ((ep)->xffnew), ((ep)->yffnew),
worldf[2], maxdeep, mindeep, LOWLEVEL);

  /* free the memory of the world's coordinates vectors */

  free_vector(worldi, 0, 2);
  free_vector(worldf, 0, 2);

  return(0);

}

/*************************************************************************/

/*F:dda_deep*
_____________________________________________________________________

  dda_deep
_____________________________________________________________________

Name:     dda_deep - drawing line's segment using the algorithm of Bresenham
                     and a grey's level/deep representation.

Syntax:   | int dda_deep(band, xi, yi, deepi, xf, yf, deepf, maxdeep, mindeep,
          | lowlevel)
          | int xi, yi, xf, yf, lowlevel;
          | double deepi, deepf, maxdeep, mindeep;
          | IBAND band;

Description: 'dda_deep' draw the line's segment between the initial point 'xi, yi'
             and the final point 'xf, yf' in 'band'.
             'deepi' and 'deepf' are the deep's values for the initial and final
             line's points. The deep's value of the others line's points are
             computing by linear interpolation of 'deepi' and 'deepf'.
             'lowlevel' are the lowest and highest grey's values not used for the
             point's deep representation.
             'maxdeep' and 'mindeep' are the maximum and minimum deep to
             consider for the drawing.

Return value: | 0 => Ok.

Example:  | Try yourself.

Restrictions: None.

Author:      Joao Tavares

Id:          @(#)dda_deep.c   1.00   94/05/22
_____________________________________________________________________
*/

int dda_deep(band, xi, yi, deepi, xf, yf, deepf, maxdeep, mindeep, lowlevel)
int xi, yi, xf, yf, lowlevel;
double deepi, deepf, maxdeep, mindeep;
IBAND band;

{

  register int t, distance;
  int xsize, ysize, xerr, yerr, delta_x, delta_y, incx, incy, xii, yii, i, levels, color;
  double l, li, d_deep, deep;

  xsize = Ixsize(band);
  ysize = Iysize(band);

  xerr = 0;
  yerr = 0;

  /* save the initial coordinates of this segment */

  xii = xi;
  yii = yi;

  /* compute the length of this segment */

  l = PYTHAG((double)(xf-xi), (double)(yf-yi));

  /* find the available levels of deep */

  levels = 255-2*lowlevel;

  /* compute the relation deep/grey */

  d_deep = (maxdeep-mindeep)/((double)(levels));

  deepi = deepi/l;
  deepf = deepf/l;

  /* calculation of delta x and delta y */
```

```
delta_x = xf-xi;
delta_y = yf-yi;

/* determination of increment x and increment y */

if (delta_x > 0) incx = 1;
else if (delta_x == 0) incx = 0;
else incx = -1;
if (delta_y > 0) incy = 1;
else if (delta_y == 0) incy = 0;
else incy = -1;

/* determination of the distance */

delta_x = abs(delta_x);
delta_y = abs(delta_y);

if (delta_x > delta_y) distance = delta_x;
else distance = delta_y;

/* mark the closest point to the line with color */

for (t = 0; t <= distance; t++) {

  if (xi < 1 || xi > xsize || yi < 1 || yi > ysize) continue;

  li = PYTHAG((double)(xii-xi), (double)(yii-yi));

  /* find the deep of this pixel */

  deep = deepi*(l-li)+deepf*li;

  /* find the color of this pixel */

  if (deep > maxdeep) color = WHITE-lowlevel;
  else {

    if (deep < mindeep) color = BLACK+lowlevel;
    else {

      deep -= mindeep;
      for (i = 0; i < levels; i++) {

        if (deep >= ((double)(i))*d_deep && deep < ((double)(i+1))*d_deep) {
          color = i+lowlevel;
          break;

        }

      }

    }

  }

  /* mark this line's pixel as color */

  band[yi][xi] = color;

  xerr += delta_x;
  yerr += delta_y;
  if (xerr >= distance) {

    xerr -= distance;
    xi += incx;

  }
  if (yerr >= distance) {

    yerr -= distance;
    yi += incy;

  }

 }

 return(0);

}

/*************************************************************************/

/*P:deep*
_________________________________________________________________________

 deep
_________________________________________________________________________

Name:    deep - computing the world coordinates of line's end points from the
```

image's coordinates in successive frames (files)

Syntax: | deep [-t <title>] [-m <maxDeep>] [-n <minDeep>] [-nf <nFrameDeep>]
| [-x <xSize>] [-y <ySize>] [-s <sx>] [-f <focalDist>]
| [-k <radialLensDist>] [-dx <dx>] [-dy <dy>] [-cx <xCenterBuf>]
| [-cy <yCenterBuf>] [-d1 <maxDistMedP>] [-d2 <maxDifLength>]
| [-d3 <maxDifAng>] [-fi <outputFile>] [-v <verbose>] <inputFile>
| <outputImage>

Description: 'deep' performs the computing of the world coordinates of line's end points from the image's coordinates in successive frames (files). The camera's model is the TSAI's model and in the matching phase are used Kalman's filters with Mahalanobis's distances or with geometric's restrictions. For the computing of the world's coordinates is used the correspondent matching line and the solution of the stereo projection's equations by the linear least-squares regression using SVD decomposition.
The image's title is indicate trough flag '-t', by default 'title' is Deep extracted.
The deep's maximum and minimum values to consider in the line's deep drawing are indicate trough flags '-m' and 'n'. By default 'maxDeep' is 2800.0 and 'mindeep' is 0.0.
The gap between frames (files) for the calculation of the world's coordinates is indicate trough flag '-nf'. By default 'nFrameDeep' is 0.
The frame buffer's dimensions in directions x and y are indicate trough flags '-x' and '-y'. By default the frame buffer's dimensions are 512 by 512.
The horizontal uncertainty factor is indicate trough flag '-s'. By default 'sx' is 0.710935.
The effective focal length is indicate trough flag '-f'. By default 'focalDist' is 50.0.
The lens's radial distortion is indicate trough flag '-k'. By default 'radialLensDist' is 0.001.
The center to center distance between adjacent sensor elements in x (scanline) direction is indicate trough flag '-dx'. By default 'dx' is 8.37765957e-3 mm.
The center to center distance between adjacent CCD sensor in the y direction is indicate trough flag '-dy'. By default 'dy' is 8.07560136e-3 mm.
The image's center in the frame buffer for directions x and y are indicate through flags '-cx' and '-cy'. By default 'xcenterbuf' is (xsizebuf/2) and 'ycenterbuf' is (ysizebuf/2).
The parameters for the matching by Kalman's filters with geometric's restrictions are indicate through flags '-d1' the maximum distance for line's middle point, '-d2' the maximum difference for line's length and '-d3' the maximum difference for line's directions. By default 'maxDistMedP' is 100.0, 'maxDifLength' is 20.0 and 'maxDifAng' is 10.0.
The user can see the output's results if the flag '-v' is 1. By default 'verbose' is 0.
The results after every 'nFramedeep' are writing in the output file 'ouputFile' if it's name is indicate through flag 'fi'. The writing's format is: image's coordinates, world's coordinates.
'inputFile' is the name of the input file which contain the file's names for each frame to consider. All of this files must contain the camera's position (three rotations (in degree) and three translations) and the image's coordinates for the line's end points. 'outputImage' is the name of the output image. This image will have one band for every 'nFrameDeep' where the matched lines are drawing with the relation deep/grey.

Return value: | 1 => Wrong value for xsize <xSize>; it must be greater than zero.
| 2 => Wrong value for ysize <ySize>; it must be greater than zero.
| 3 => Wrong value for f; it must be different from zero.
| 4 => Wrong value for sx; it must be different from zero.
| 5 => Wrong value for dx <dx>; it must be greater than zero.
| 6 => Wrong value for dy <dy>; it must be greater than zero.
| 7 => Can't open input file <inputFile>.
| 8 => The input file <inputFile> contain only one name.
| 9 => Can't make output image <ouputImage>.
| 10 => Can't open output file <ouputFile>.

Examples: | Try yourself.

Restrictions: The values for f and sx must be different from zero. The values for xSize, ySize, dx and dy must be greater than zero. The input file must contain more than one name (file).

Author: Joao Tavares

Id: @(#)deep.c 1.0 94/05/22

---

*/

#ifdef MAIN

static char *SccsId = "@(#)deep.c 1.00 94/05/22, DEEC, FEUP";
char progname[]="deep";

```
void main(argc, argv)
int argc;
char **argv;

{

  int xsize, ysize, cx, cy, verb, n, nfdeep, nband;
  double sx, f, k, dx, dxl, dy, *d, maxdeep, mindeep;
  char file[10], *fileout, *title, arg[500];
  IMAGE imgout;
  int matching();

  InitMessage(&argc, argv, "Usage: %s\n [-t <title>] [-m <maxDeep>] [-n <minDe ep>] [-nf
<nFrameDeep>] [-x <xSize>] [-y <ySize>] [-s <sx>] [-f <focalDist>] [-k <radialLensDist>] [-dx
<dx>] [-dy <dy>] [-cx <xCenterBuf>] [-cy <yCenterBuf>] [-d1 <maxDistMedP>] [-d2
<maxDifLength>] [-d3 <maxDifAng>] [-fi <outputFil e>] [-v <verbose>] <inputFile>
<outputImage>\n");
  Iset_message(1);
  Iset_abort(1);

  /* allocation of memory to the vector of geometric's restrictions */

  d = vector(0, 2);

  /* read switches */

  title = read_switch(&argc, argv, "-t", 1, "Deep extracted");
  maxdeep = atof(read_switch(&argc, argv, "-m", 1, "2800.0"));
  mindeep = atof(read_switch(&argc, argv, "-n", 1, "0.0"));
  nfdeep = atoi(read_switch(&argc, argv, "-nf", 1, "0"));
  xsize = atoi(read_switch(&argc, argv, "-x", 1, "512"));
  ysize = atoi(read_switch(&argc, argv, "-y", 1, "512"));
  sx = atof(read_switch(&argc, argv, "-s", 1, "0.710935"));
  f = atof(read_switch(&argc, argv, "-f", 1, "50.0"));
  k = atof(read_switch(&argc, argv, "-k", 1, "0.001"));
  dx = atof(read_switch(&argc, argv, "-dx", 1, "8.37765957e-3"));
  dy = atof(read_switch(&argc, argv, "-dy", 1, "8.07560136e-3"));
  cx = atoi(read_switch(&argc, argv, "-cx", 1, "0"));
  cy = atoi(read_switch(&argc, argv, "-cy", 1, "0"));
  d[0] = atof(read_switch(&argc, argv, "-d1", 1, "60.0"));
  d[1] = atof(read_switch(&argc, argv, "-d2", 1, "60.0"));
  d[2] = atof(read_switch(&argc, argv, "-d3", 1, "40.0"));
  fileout = read_switch(&argc, argv, "-fi", 1, NULL);
  verb = atoi(read_switch(&argc, argv, "-v", 1, "0"));

  if (argc == 1) Usage(1, (char*)0);
  if (argc != 3) Usage(2, "\nBad number of arguments.\n");
  if (xsize <= 0) exit(Error(1, "\nWrong value for xsize %d; it must be greater than zero.\n", xsize));
  if (ysize <= 0) exit(Error(2, "\nWrong value for ysize %d; it must be greater than zero.\n", ysize));
  if (f == 0.0) exit(Error(3, "\nWrong value for f; it must be different from zero.\n"));
  if (sx == 0.0) exit(Error(4, "\nWrong value for sx; it must be different from zero.\n"));
  if (dx <= 0.0) exit(Error(5, "\nWrong value for dx %g; it must be greater than zero.\n", dx));
  if (dy <= 0.0) exit(Error(6, "\nWrong value for dy %g; it must be greater than zero.\n", dy));

  nfdeep += 1;
  if (cx == 0) cx = xsize/2;
  if (cy == 0) cy = ysize/2;
  dxl = sx*dx;

  /* open input file */

  fptin1 = fopen(argv[1], "r");
  if (fptin1 == NULL) exit(Error(7, "\nCan't open input file %s.\n", argv[1]));

  /* find how many names the input file contain */

  n = 0;
  fscanf(fptin1, "%s", file);
  while (!feof(fptin1)) {

    n += 1;
    fscanf(fptin1, "%s", file);

  }
  if (n < 2) exit(Error(8, "\nThe input file %s, contain only one name.\n", argv[1]));
  rewind(fptin1);

  /* find how many bands will be used */

  if(nfdeep == 1) nband = n-1;
  else nband = n/nfdeep;

  /* make output image if nband is not null */

  if (nband != 0) {

    imgout = Init_image(nband, title);
    if (imgout == NULL) exit(Error(9, "\nCan't make output image %s.\n", argv[2]));

  }
```

```
  sprintf(arg, "Results for %s in %s with:\n\n\tMaximum deep: %g\n\tMinimum deep:
%g\n\tNumber of between frames for deep's calculation: %d\n\tHorizontal factor, sx:
%g\n\tCamera's sensors distance for x, dx: %g\n\tCamera's sensors distance for y, dy: %g\n\tFocal
distance, f: %g\n\tRadial lens distortion, k: %g\n\tMaximum distance of the middle's point for
matching: %g\n\tMaximum difference of length for matching: %g\n\tMaximum angular difference
for matching: %g.\n\n", argv[0], argv[1], maxdeep, mindeep, nfdeep-1, sx, dx, dy, f, k, d[0], d[1],
d[2]);

  /* open output file if it was desire and write arg */

  if (fileout != NULL) {

    fptout = fopen(fileout, "a");
    if (fptout == NULL) exit(Error(10, "\nCan't open output file %s.\n", fileout));
    fprintf(fptout, arg);

  }

  /* call matching function */

  matching(imgout, xsize, ysize, maxdeep, mindeep, nfdeep, f, k, cx, cy, dxl, dy, d, n, verb);

  /* free the memory of the vector of geometric's restrictions */

  free_vector(d, 0, 2);

  /* close input file */

  fclose(fptin1);

  /* close output file if it was desire */

  if (fptout != NULL) fclose(fptout);

  /* write output image with history if nband is not null */

  if (nband != 0) {

    Ihistory(imgout, argv[0], arg);
    Iwrite_image(imgout,argv[2]);

  }

}
```

```
#endif

/*****************************************************************************/
```

3.2.2 - Ficheiro func_3d.h

Este ficheiro contém as definições de variáveis e de protótipos relativamente aos módulos que pertencem ao grupo ***II*** da implementação ***deep***.

```
/*
_____________________________________________________________________________

  func_3d.h
  @(#)func_3d.h   1.0 94/06/15, Copyright 1994, DEEC, UoP
  Image processing lab,
  Department of Electrical and Computer Engineering
  University of Porto
  E-mail: gpai@obelix.fe.up.pt
_____________________________________________________________________________

  File Name: func_3d.h
  Purpose  : 3D functions.
  Author   : Joao Tavares, DEEC, UoP
  Version  : 1.0
  Date     : 15/06/94
_____________________________________________________________________________
*/

/*****************************************************************************/

/* FUNCTIONS PROTOTYPES */

int compute_mpp();
int frame_undistorted();
int match_line();
int stereo_eq();

/*****************************************************************************/
```

3.2.3 - Ficheiro func_3d.c

Este ficheiro contém a listagem relativa aos módulos que pertencem ao grupo ***II*** da implementação ***deep***.

```
/*
_____________________________________________________________________

  func_3d.c
  @(#)func_3d.c   1.0 94/06/15, Copyright 1994, DEEC, UoP
  Image processing lab, Department of Electrical and Computer
  Engineering
  University of Porto
  E-mail: gpai@obelix.fe.up.pt
_____________________________________________________________________

  message.h
  @(#)message.h   1.3 92/01/15, Copyright 1990, Blab, UiO
  Image processing lab, Department of Informatics
  University of Oslo
  E-mail: blab@ifi.uio.no
_____________________________________________________________________

  These software routines were developed for the XITE system under the written
  permission of B-lab, Department of Informatics, University of Oslo.
_____________________________________________________________________

  Permission to use, copy, modify and distribute this software and its documentation for any purpose
and without fee is hereby granted, provided that this copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in supporting documentation and that the
name of B-lab, Department of Informatics or University of Oslo not be used in advertising or publicity
pertaining to distribution of the software without specific, written prior permission.

  B-LAB DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT
SHALL B-LAB BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL
DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA
OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER
TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
*/
/*********************************************************************/

/* INCLUDES */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <blab/message.h>
#include "gdefs.h"
#include "memory.h"
#include "linleasq.h"
#include "func_3d.h"

/*********************************************************************/

#ifndef lint
static char *SccsId = "@(#)func_3d.c 1.0 94/06/15, DEEC, UoP";
#endif

/*********************************************************************/

/*F:compute_mpp*
_____________________________________________________________________

  compute_mpp
_____________________________________________________________________

Name:      compute_mpp - compute the perspective projection matrix

Syntax:    | int compute_mpp(mpp, rot, trans, f, c)
           | int c;
           | float rot[], trans[];
           | double mpp[][4], f;

Description: 'compute_mpp' performs the determination of the perspective
             projection matrix (of the transformation 3D from the world's
             coordinate system to the camera's coordinate system).
             'mpp[][4]' is the perspective projection matrix.
             'rot[]' is the rotation's vector 3D from object world coordinate
             system to the camera coordinate system.
             'trans[]' is the translation's vector 3D from object world
             coordinate system to the camera coordinate system.
             'f' is the effective focal distance.
             If 'c' is 1 the angles are convert to rad.

Return value: | 0 => Ok.

Example:   | Try yourself.
```

```
Restrictions:   None.

Author:         Joao Tavares

Id:             @(#)compute_mpp.c   1.00   94/06/15
_____________________________________________________________________
*/

int compute_mpp(mpp, rot, trans, f, c)
int c;
float rot[], trans[];
double mpp[][4], f;

{

  double aux1, aux2, aux3, aux4, aux5, aux6;

  if (c == 1) {

    /* conversion deg/rad */

    rot[0] = ANG_DEG_RAD(rot[0]);
    rot[1] = ANG_DEG_RAD(rot[1]);
    rot[2] = ANG_DEG_RAD(rot[2]);

  }

  /* compute the matrix perspective projection */

  aux1 = cos(rot[0]);
  aux2 = sin(rot[0]);
  aux3 = cos(rot[1]);
  aux4 = sin(rot[1]);
  aux5 = cos(rot[2]);
  aux6 = sin(rot[2]);

  mpp[0][0] = aux3*aux5;
  mpp[0][1] = aux3*aux6;
  mpp[0][2] = -aux4;
  mpp[0][3] = trans[0];
  mpp[1][0] = aux5*aux2*aux4-aux1*aux6;
  mpp[1][1] = aux6*aux2*aux4+aux1*aux5;
  mpp[1][2] = aux2*aux3;
  mpp[1][3] = trans[1];
  mpp[2][0] = (aux5*aux4*aux1+aux2*aux6)/f;
  mpp[2][1] = (-aux2*aux5+aux6*aux4*aux1)/f;
  mpp[2][2] = aux3*aux1/f;
  mpp[2][3] = trans[2]/f;

  return(0);

}

/*************************************************************************/

/*F:frame_undistorted*
_____________________________________________________________________

  frame_undistorted
_____________________________________________________________________

Name:    frame_undistorted - compute the undistorted coordinates from the
                             frame buffer's coordinates

Syntax:     | int frame_undistorted(xf, yf, xu, yu, k, dxl, dy, cx, cy)
            | int xf, yf, cx, cy;
            | double *xu, *yu, k, dxl, dy;

Description:  'frame_undistorted' performs the determination of the undistorted
              coordinates of a point from the frame buffer's coordinates by
              TSAI's model.
              'xf' and 'yf' are the frame buffer's coordinates's.
              'xu' and 'yu' passed by reference are the undistorted's
              coordinates's.
              'k' is the lens's radial distortion.
              'dxl' is sx*dx where dx is the distance between adjacent sensor
              elements in x (scan line) direction and sx is the horizontal
              uncertainty factor.
              'dy' is the center to center distance between adjacent CCD sensor
              in the y direction.
              'cx' and 'cy' are the row and column numbers of the image's center
              in the frame buffer.

Return value:  | 0 => Ok.

Example:       | Try yourself.

Restrictions:  'dx' and 'dy' must be different from zero.
```

```
Author:          Joao Tavares

Id:              @(#)frame_undistorted.c   1.00   94/06/08
_____________________________________________________________________
*/

int frame_undistorted(xf, yf, xu, yu, k, dxl, dy, cx, cy)
int xf, yf, cx, cy;
double *xu, *yu, k, dxl, dy;

{

  /* compute from frame coordinates the undistorted coordinates */

  double r2, xd, yd, aux1, aux2, aux3;

  aux1 = (double)(xf-cx);
  aux2 = (double)(yf-cy);
  xd = dxl*aux1;
  yd = dy*aux2;

  r2 = SQR(xd)+SQR(yd);
  aux3 = 1.0+k*r2;

  *xu = xd*aux3;
  *yu = yd*aux3;

  return(0);

}

/*********************************************************************/

/*F:match_line*
_____________________________________________________________________

  match_line
_____________________________________________________________________

Name:       match_line - compute the matching line

Syntax:     | int match_line(mpp1, mpp2, px, py, a1, b1)
            | double mpp1[][4], mpp2[][4], px, py, *a1, *b1;

Description:  'match_line' performs the determination of the matching line in the
              image's plane 2.
              'mpp1[][4]' is the perspective projection matrix for the image's
              plane 1.
              'mpp2[][4]' is the perspective projection matrix for image's plane 2.
              'px' and 'py' are the point's coordinates in the image's plane 1.
              'a1' and 'b1' are the matching line's parameters in the image's plane 1.
              The equation of this line is: y='a1'*x+'b1'.

Return value: | 0 => Ok.
              | 1 => Can´t compute the matching line.

Example:      | Try yourself.

Restrictions: None.

Author:       Joao Tavares

Id:           @(#)match_line.c   1.00   94/06/15
_____________________________________________________________________
*/

int match_line(mpp1, mpp2, px, py, a1, b1)
double mpp1[][4], mpp2[][4], px, py, *a1, *b1;

{

  double **xa, **xb, **h, *l, det, t1, a, b, c;

  xa = matrix(0, 1, 0, 1);
  xb = matrix(0, 1, 0, 1);
  h = matrix(0, 1, 0, 1);
  l = vector(0, 5);

  /* compute the xa matrix */

  xa[0][0] = mpp1[2][0]*px-mpp1[0][0];
  xa[1][0] = mpp1[2][0]*py-mpp1[1][0];
  xa[0][1] = mpp1[2][1]*px-mpp1[0][1];
  xa[1][1] = mpp1[2][1]*py-mpp1[1][1];

  /* compute the xb matrix */

  xb[0][0] = mpp1[0][2]-mpp1[2][2]*px;
  xb[1][0] = mpp1[1][2]-mpp1[2][2]*py;
  xb[0][1] = mpp1[0][3]-mpp1[2][3]*px;
```

```
  xb[1][1] = mpp1[1][3]-mpp1[2][3]*py;

  /* compute the inverse matrix (xa) of xa */

  det = xa[0][0]*xa[1][1]-xa[0][1]*xa[1][0];
  if (det == 0.0) exit(Error(1, "\nCan't compute the matching line.\n"));
  t1 = xa[0][0];
  xa[0][0] = xa[1][1]/det;
  xa[1][0] = -xa[1][0]/det;
  xa[0][1] = -xa[0][1]/det;
  xa[1][1] = t1/det;

  /* compute the h matrix */

  h[0][0] = xa[0][0]*xb[0][0]+xa[0][1]*xb[1][0];
  h[1][0] = xa[1][0]*xb[0][0]+xa[1][1]*xb[1][0];
  h[0][1] = xa[0][0]*xb[0][1]+xa[0][1]*xb[1][1];
  h[1][1] = xa[1][0]*xb[0][1]+xa[1][1]*xb[1][1];

  /* compute l[0]...l[6] */

  l[0] = mpp2[0][0]*h[0][1]+mpp2[0][1]*h[1][1]+mpp2[0][3];
  l[1] = mpp2[2][0]*h[0][1]+mpp2[2][1]*h[1][1]+mpp2[2][3];
  l[2] = mpp2[2][0]*h[0][0]+mpp2[2][1]*h[1][0]+mpp2[2][2];
  l[3] = mpp2[0][0]*h[0][0]+mpp2[0][1]*h[1][0]+mpp2[0][2];
  l[4] = mpp2[1][0]*h[0][1]+mpp2[1][1]*h[1][1]+mpp2[1][3];
  l[5] = mpp2[1][0]*h[0][0]+mpp2[1][1]*h[1][0]+mpp2[1][2];

  /* compute the line's parameters */

  a = l[2]*l[4]-l[1]*l[5];
  b = l[1]*l[3]-l[0]*l[2];
  c = l[0]*l[5]-l[3]*l[4];
  *a1 = -a/b;
  *b1 = -c/b;

  free_matrix(xa, 0, 1, 0, 1);
  free_matrix(xb, 0, 1, 0, 1);
  free_matrix(h, 0, 1, 0, 1);
  free_vector(l, 0, 5);

  return(0);

}
```

```
/*****************************************************************/

/*F:stereo_eq*
_________________________________________________________________

  stereo_eq
_________________________________________________________________

Name:       stereo_eq - compute the world's coordinates of a point

Syntax:     | int stereo_eq(mpp1, mpp2, xu1, yu1, xu2, yu2, a)
            | double mpp1[][4], mpp2[][4], xu1, yu1, xu2, yu2, *a;

Description: 'stereo_eq' performs the determination of the world's coordinates of
             a point using the projection stereo equations.
             'mpp1[][4]' is the perspective projection matrix for the image's plane 1.
             'mpp2[][4]' is the perspective projection matrix for the image's plane 2.
             'xu1' and 'yu1' are the point's coordinates in the image's plane 1.
             'xu2' and 'yu2' are the point's coordinates in the image's plane 2.
             'a' passed by reference is the vector of the world's coordinates.

Return value: | 0 => Ok.

Example:    | Try yourself.

Restrictions: None.

Author:     Joao Tavares

Id:         @(#)stereo_eq.c   1.00   94/06/15
_________________________________________________________________
*/

int stereo_eq(mpp1, mpp2, xu1, yu1, xu2, yu2, a)
double mpp1[][4], mpp2[][4], xu1, yu1, xu2, yu2, *a;

{

  double **z, *y;

  z = matrix(0, 3, 0, 2);
  y = vector(0, 3);

  /* compute the vector of the observed values */
```

```
  y[0] = mpp1[2][3]*xu1-mpp1[0][3];
  y[1] = mpp1[2][3]*yu1-mpp1[1][3];
  y[2] = mpp2[2][3]*xu2-mpp2[0][3];
  y[3] = mpp2[2][3]*yu2-mpp2[1][3];

  /* compute the matrix of the observed values */

  z[0][0] = mpp1[0][0]-mpp1[2][0]*xu1;
  z[0][1] = mpp1[0][1]-mpp1[2][1]*xu1;
  z[0][2] = mpp1[0][2]-mpp1[2][2]*xu1;
  z[1][0] = mpp1[1][0]-mpp1[2][0]*yu1;
  z[1][1] = mpp1[1][1]-mpp1[2][1]*yu1;
  z[1][2] = mpp1[1][2]-mpp1[2][2]*yu1;
  z[2][0] = mpp2[0][0]-mpp2[2][0]*xu2;
  z[2][1] = mpp2[0][1]-mpp2[2][1]*xu2;
  z[2][2] = mpp2[0][2]-mpp2[2][2]*xu2;
  z[3][0] = mpp2[1][0]-mpp2[2][0]*yu2;
  z[3][1] = mpp2[1][1]-mpp2[2][1]*yu2;
  z[3][2] = mpp2[1][2]-mpp2[2][2]*yu2;

  /* call svdfit function */

  svdfit(z, y, 4, a, 3);

  free_matrix(z, 0, 3, 0, 2);
  free_vector(y, 0, 3);

  return(0);

}

/*********************************************************************/
```

3.2.4 - Ficheiro linleasq.h

Este ficheiro contém as definições de variáveis e de protótipos relativamente aos módulos que pertencem ao grupo ***III*** da implementação ***deep***.

```
/*
_____________________________________________________________________

  linleasq.h
  @(#)linleasq.h   1.0 94/06/15, Copyright 1994, DEEC, UoP
  Image processing lab,
  Department of Electrical and Computer Engineering
  University of Porto
  E-mail: gpai@obelix.fe.up.pt
_____________________________________________________________________

  File Name: linleasq.h
  Purpose  : Definitions for handling linear systems by linear least squares.
  Author   : Joao Tavares, DEEC, UoP
  Version  : 1.0
  Date     : 17/06/94
_____________________________________________________________________
*/

/*********************************************************************/

/* FUNCTIONS PROTOTYPES */

void svdfit();
void svbksb();
void svdcmp();

/*********************************************************************/
```

3.2.5 - Ficheiro linleasq.c

Este ficheiro contém a listagem relativa aos módulos que pertencem ao grupo ***III*** da implementação ***deep***.

```
/*
_____________________________________________________________________

  linleasq.c
  @(#)linleasq.c   1.0 94/06/15, Copyright 1994, DEEC, UoP
  Image processing lab, Department of Electrical and Computer
  Engineering
  University of Porto
  E-mail: gpai@obelix.fe.up.pt
_____________________________________________________________________

  message.h
  @(#)message.h   1.3 92/01/15, Copyright 1990, Blab, UiO
  Image processing lab, Department of Informatics
```

University of Oslo
E-mail: blab@ifi.uio.no

```
*/
/************************************************************************/

/* INCLUDES */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <blab/message.h>
#include "memory.h"
#include "gdefs.h"
#include "linleasq.h"

/************************************************************************/

#ifndef lint
static char *SccsId = "@(#)linleasq.c 1.0 94/06/15, DEEC, UoP";
#endif

/************************************************************************/

/*F:svdfit*
```

---

svdfit

Name: svdfit - make the linear least-squares regression using SVD decomposition

Syntax: | void svdfit(xin, yin, ndata, ain, ma)
| double **xin, *yin, *ain;
| int ndata, ma;

Description: 'svdfit' performs the linear least-squares regression using the singular value decomposition.
'xin' is the matrix of the observed values for the independent variables in the model.
'ma' is the number of variables in the model and 'ndata' is the number of data points in the model.
The vector 'yin' contains the observed values of the dependent variables in the model.
The vector 'ain' contains the unknown coefficients of the independent variables in the model.

Return value: | None.

Example: | Try yourself.

Restrictions: None.

Author: Joao Tavares

Id: @(#)svdfit.c 1.00 94/06/15

```
*/

void svdfit(xin, yin, ndata, ain, ma)
double **xin, *yin, *ain;
int ndata, ma;

{

  register int i, j;
  double wmax, thresh, **v, *w, **x, *y, *a;

  w = vector(1, ma);
  v = matrix(1, ma, 1, ma);
  y = vector(1, ndata);
```

```
  x = matrix(1, ndata, 1, ma);
  a = vector(1, ma);

  for (i = 1; i <= ndata; i++) {

    y[i] = yin[i-1];
    for (j = 1; j <= ma; j++) x[i][j] = xin[i-1][j-1];

  }

  /* Singular value decomposition */

  svdcmp(x, ndata, ma, w, v);

  /* Edit the singular values, given TOL from the #define statement, between here */

  wmax = 0.0;
  for (j = 1; j <= ma; j++) if (w[j] > wmax) wmax = w[j];
  thresh = TOL*wmax;
  for (j = 1; j <= ma; j++) if (w[j] < thresh) w[j] = 0.0;

  /* ... and here. */

  svbksb(x, w, v, ndata, ma, y, a);

  for (i = 1 ; i <= ma ; i++) ain[i-1] = a[i];

  free_vector(w, 1, ma);
  free_matrix(v, 1, ma, 1, ma);
  free_vector(y, 1, ndata);
  free_matrix(x, 1, ndata, 1, ma);
  free_vector(a, 1, ma);

  return;

}

/*****************************************************************************/

/*F:svbksb*
_____________________________________________________________________________

  svbksb
_____________________________________________________________________________

Name:          svbksb - compute A.X=B

Syntax:        | void svbksb(u, w, v, m, n, b, x)
               | double **u, w[], **v, b[], x[];
               | int m, n;

Description:   'svbksb' solves A.X=B for a vector X, where A is specified by the
               arrays 'u[1..m][1..n]', 'w[1..n]', 'v[1..n][1..n]' as returned by svdcmp.
               'm' and 'n' are the dimensions of 'a', and will be equal for squares
               matrices. 'b[1..m]' is the right-hand side. 'x[1..n]' is the output
               solution vector.

Return value:  | None.

Example:       | Try yourself.

Restrictions:  None.

Author:        Joao Tavares

Id:            @(#)svbksb.c   1.00   94/06/15
_____________________________________________________________________________
*/

void svbksb(u, w, v, m, n, b, x)
double **u, w[], **v, b[], x[];
int m, n;

{

  int jj, j, i;
  double s, *tmp;

  tmp = vector(1, n);

  /* Calculate (trans[U])*[B] */

  for (j = 1; j <= n; j++) {

    s = 0.0;

    /* Nonzero result only if w(j) is non zero */

    if (w[j]) {
```

```
    for (i = 1; i <= m; i++) s += u[i][j]*b[i];

    /* This is the divide by w(j) */

    s /= w[j];

   }
   tmp[j] = s;

 }

 /* Matrix multiply by [V] to get answer */

 for (j = 1; j <= n; j++) {

   s = 0.0;
   for (jj = 1; jj <= n; jj++) s += v[j][jj]*tmp[jj];
   x[j] = s;

 }
 free_vector(tmp, 1, n);

 return;

}

/******************************************************************************/

/*F:svdcmp*
_______________________________________________________________________________

 svdcmp
_______________________________________________________________________________

Name:       svdcmp - construct the singular value decomposition

Syntax:     | void svdcmp(a, m, n, w, v)
            | double **a, w[], **v;
            | int m, n;

Description:  Given a matrix 'a[1..m][1..n]' 'svdcmp' computes its singular
              decomposition, A=U.W.(transpose)V.
              The matrix U replaces 'a' on output. The diagonal matrix of
              singular values W is output as a vector 'w[1..n]'. The matrix V
              (not the transpose) is output as 'v[1..n][1..n]'.

Return value:  | 1 => No convergence in 30 svdcmp iterations.

Example:       | Try yourself.

Restrictions:  The iteration's number.

Author:        Joao Tavares

Id:            @(#)svdcmp.c   1.00   94/06/15
_______________________________________________________________________________
*/

void svdcmp(a, m, n, w, v)
double **a, w[], **v;
int m, n;

{

 int flag, i, its, j, jj, l, k, nm;
 double anorm, c, f, g, h, s, scale, x, y, z, *rv1;

 rv1 = vector(1, n);
 g = scale = anorm = 0.0;

 /* Householder reduction to bidiagonal form */

 for (i = 1; i <= n; i++) {

  l = i+1;
  rv1[i] = scale*g;
  g = s = scale = 0.0;
  if (i <= m) {

    for (k = i; k <= m; k++) scale += fabs(a[k][i]);
    if (scale) {

      for (k = i; k <= m; k++) {

        a[k][i] /= scale;
        s += a[k][i]*a[k][i];

      }
      f = a[i][i];
      g = -SIGN2(sqrt(s), f);
```

```
    h = f*g-s;
    a[i][i] = f-g;
    for (j = l; j <= n; j++) {

      for (s = 0.0, k = i; k <= m; k++) s += a[k][i]*a[k][j];
      f = s/h;
      for ( k = i; k <= m; k++) a[k][j] += f*a[k][i];

    }
    for (k = i; k <= m; k++) a[k][i] *= scale;

  }

}
w[i] = scale*g;
g = s = scale = 0.0;
if (i <= m && i != n) {

  for (k = l; k <= n; k++) scale += fabs(a[i][k]);
  if (scale) {

    for (k = l; k <= n; k++) {

      a[i][k] /= scale;
      s += a[i][k]*a[i][k];

    }
    f = a[i][l];
    g = - SIGN2(sqrt(s), f);
    h = f*g-s;
    a[i][l] = f-g;
    for (k = l; k <= n; k++) rv1[k] = a[i][k]/h;
    for (j = l; j <= m; j++) {

      for (s = 0.0, k = l; k <= n; k++) s += a[j][k]*a[i][k];
      for (k = l; k <= n; k++) a[j][k] += s*rv1[k];

    }
    for (k = l; k <= n; k++) a[i][k] *= scale;

  }

}
anorm = DMAX(anorm, (fabs(w[i])+fabs(rv1[i])));
}

/* Accumulation of right-hand transformations */

for (i = n; i >= 1; i--) {

  if (i < n) {

    if (g) {

      /* Double division to avoid possible underflow */

      for (j = l; j <= n; j++) v[j][i] = (a[i][j]/a[i][l])/g;
      for (j = l; j <= n; j++) {

        for (s = 0.0, k = l; k <= n; k++) s += a[i][k]*v[k][j];
        for (k = l; k <= n; k++) v[k][j] += s*v[k][i];

      }

    }
    for (j = 1; j <= n; j++) v[i][j] = v [j][i] = 0.0;

  }
  v[i][i] = 1.0;
  g = rv1[i];
  l = i;

}

/* Accumulation of left-hand transformations */

for(i = IMIN(m, n); i >=1 ; i--) {

  l = i+1;
  g = w[i];
  for (j = l; j <= n; j++) a[i][j] = 0.0;
  if (g) {

    g = 1.0/g;
    for (j = l; j <= n; j++) {

      for (s = 0.0, k = l; k <= m; k++) s += a[k][i]*a[k][j];
      f = (s/a[i][i])*g;
      for (k = i; k <= m; k++) a[k][j] += f*a[k][i];
```

```
    }
    for (j = i; j <= m; j++) a[j][i] *= g;

  } else for (j = i; j <= m; j++) a[j][i] = 0.0;
  ++a[i][i];

}

 /* Diagonalization of the bidiagonal form: Loop over singular values, and over allowed iterations
*/

 for (k = n; k >= 1; k--) {

  for (its = 1; its <= 30; its++) {

    flag = 1;

    /* Test for splitting */

    for ( l = k; l >= 1; l--) {

      nm = l-1;

      /* Note that rv1[1] is always zero */

      if ((double)(fabs(rv1[l]+anorm)) == anorm) {

        flag = 0;
        break;

      }
      if ((double)(fabs(w[nm]+anorm)) == anorm) break;

    }
    if (flag) {

      /* Cancellation of rv1[l], if l>1 */

      c = 0.0;
      s = 1.0;
      for (i = l; i <= k; i++) {

        f = s*rv1[i];
        rv1[i] = c*rv1[i];
        if ((double)(fabs(f)+anorm) == anorm) break;
        g = w[i];
        h = PYTHAG(f, g);
        w[i] = h;
        h = 1.0/h;
        c = g*h;
        s = -f*h;
        for (j = 1; j <= m; j++) {

          y = a[j][nm];
          z = a[j][i];
          a[j][nm] = y*c+z*s;
          a[j][i] = z*c-y*s;

        }

      }

    }
    z = w[k];
    if (l == k) {

      /* Convergence */

      if (z < 0.0) {

        /* Singular value is made nonnegative */

        w[k] = -z;
        for (j = 1; j <= n; j++) v[j][k] = -v[j][k];

      }
      break;

    }
    if (its == 30) exit(Error(1, "\nNo convergence in 30 svdcmp iterations.\n"));

    /* Shift from bottom 2-by-2 minor */

    x = w[l];
    nm = k-1;
    y = w[nm];
    g = rv1[nm];
    h = rv1[k];
    f = ((y-z)*(y+z)+(g-h)*(g+h))/(2.0*h*y);
```

```
g = PYTHAG(f, 1.0);
f = ((x-z)*(x+z)+h*((y/(f+SIGN2(g, f)))-h))/x;
c = s = 1.0;

/* Next QR transformation */

for (j = l; j <= nm; j++) {

 i = j+1;
 g = rv1[i];
 y = w[i];
 h = s*g;
 g = c*g;
 z = PYTHAG(f, h);
 rv1[j] = z;
 c = f/z;
 s = h/z;
 f = x*c+g*s;
 g = g*c-x*s;
 h = y*s;
 y *= c;
 for (jj = 1; jj <= n; jj++) {

  x = v[jj][j];
  z = v[jj][i];
  v[jj][j] = x*c+z*s;
  v[jj][i] = z*c-x*s;

 }
 z = PYTHAG(f, h);
 w[j] = z;

 /* Rotation can be arbitrary if z=0 */

 if (z) {

  z = 1.0/z;
  c = f*z;
  s = h*z;

 }
 f = c*g+s*y;
 x = c*y-s*g;
 for (jj = 1; jj <= m; jj++) {
      y = a[jj][j];
      z = a[jj][i];
      a[jj][j] = y*c+z*s;
      a[jj][i] = z*c-y*s;

     }

   }
   rv1[l] = 0.0;
   rv1[k] = f;
   w[k] = x;

  }

 }
 free_vector(rv1, 1, n);

 return;

}

/*************************************************************************/
```

3.2.6 - Ficheiro model.h

Este ficheiro contém as definições de variáveis e de protótipos relativamente aos módulos que pertencem ao grupo ***IV*** da implementação ***deep***.

```
/*
-------------------------------------------------------------------------

 model.h
 @(#)model.h   1.0 94/05/19, Copyright 1994, DEEC, UoP
 Image processing lab,
 Department of Electrical and Computer Engineering
 University of Porto
 E-mail: gpai@obelix.fe.up.pt
-------------------------------------------------------------------------

 File Name: model.h
 Purpose  : Definitions for lines data structures.
 Author   : Joao Tavares, DEEC, UoP
 Version  : 1.0
```

```
  Date    : 19/05/94
_____________________________________________________________________
*/

/*********************************************************************/

/* MACROS DEFINITIONS */

/* ... FOR POSITION'S MEAN */

#define _xpos(ep) ((ep)->state.meanpos->d[0])       /* x position */
#define _ypos(ep) ((ep)->state.meanpos->d[1])       /* y position */
#define _xvel(ep) ((ep)->state.meanpos->d[2])       /* x position's velocity */
#define _yvel(ep) ((ep)->state.meanpos->d[3])       /* y position's velocity */
#define _xacl(ep) ((ep)->state.meanpos->d[4])       /* x position's acceleration */
#define _yacl(ep) ((ep)->state.meanpos->d[5])       /* y position's acceleration */

/* ... FOR POSITION'S VARIANCE */

#define _xpos_cv(ep) ((ep)->state.covrpos->d[0])    /* x position */
#define _ypos_cv(ep) ((ep)->state.covrpos->d[7])    /* y position */
#define _xvel_cv(ep) ((ep)->state.covrpos->d[14])   /* x position's velocity */
#define _yvel_cv(ep) ((ep)->state.covrpos->d[21])   /* y position's velocity */
#define _xacl_cv(ep) ((ep)->state.covrpos->d[28])   /* x position's acceleration */
#define _yacl_cv(ep) ((ep)->state.covrpos->d[35])   /* y position's acceleration */

/* ... FOR DIRECTION'S MEAN */

#define _dir(ep) ((ep)->state.meandir->d[0])        /* direction */
#define _dirvel(ep) ((ep)->state.meandir->d[1])     /* direction's velocity  */
#define _diracl(ep) ((ep)->state.meandir->d[2])     /* direction's acceleration */

/* ... FOR DIRECTION'S VARIANCE */

#define _dir_cv(ep) ((ep)->state.covrdir->d[0])     /* direction */
#define _dirvel_cv(ep) ((ep)->state.covrdir->d[4])  /* direction's velocity */
#define _diracl_cv(ep) ((ep)->state.covrdir->d[8])  /* direction's acceleration */

/* ... FOR LENGTH'S MEAN */

#define _len(ep) ((ep)->state.meanlen->d[0])        /* length */
#define _lenvel(ep) ((ep)->state.meanlen->d[1])     /* length's velocity */
#define _lenacl(ep) ((ep)->state.meanlen->d[2])     /* length's acceleration */

/* ... FOR LENGTH'S VARIANCE */

#define _len_cv(ep) ((ep)->state.covrlen->d[0])     /* length */
#define _lenvel_cv(ep) ((ep)->state.covrlen->d[4])  /* length's velocity */
#define _lenacl_cv(ep) ((ep)->state.covrlen->d[8])  /* length's acceleration */

/*********************************************************************/

/* TYPES DEFINITIONS */

typedef struct _line
{

  unsigned int reference;   /* reference */
  int cf;                   /* confidence factor */
  unsigned int xifnew;      /* frame buffer's coordinate x of the first point */
  unsigned int yifnew;      /* frame buffer's coordinate y of the first point */
  unsigned int xffnew;      /* frame buffer's coordinate x of the last point */
  unsigned int yffnew;      /* frame buffer's coordinate y of the last point */
  unsigned int matched;     /* matched's flag */
  unsigned int lastmpp;     /* last perspective projection matrix's reference */
  unsigned int invert;      /* flag for indication that the end's points are inverted */
  double xiuold;            /* old undistorted's coordinate x of the first point */
  double yiuold;            /* old undistorted's coordinate y of the first point */
  double xfuold;            /* old undistorted's coordinate x of the last point */
  double yfuold;            /* old undistorted's coordinate y of the last point */
  double xiunew;            /* new undistorted's coordinate x of the first point */
  double yiunew;            /* new undistorted's coordinate y of the first point */
  double xfunew;            /* new undistorted's coordinate x of the last point */
  double yfunew;            /* new undistorted's coordinate y of the last point */
  State state;              /* global state */

} Line;

typedef struct _mpp
{

  double m[3][4];           /* perspective projection's matrix */

} Mpp;

/*********************************************************************/

/* FUNCTIONS PROTOTYPES */

Line *create_line();
```

```
void delete_line();
void print_line();
Mpp *create_mpp();
void delete_mpp();

/*****************************************************************************/
```

3.2.7 - Ficheiro model.c

Este ficheiro contém a listagem relativa aos módulos que pertencem ao grupo ***IV*** da implementação ***deep***.

```
/*
_____________________________________________________________________

  model.c
  @(#)model.c   1.0 94/05/19, Copyright 1994, DEEC, UoP
  Image Processing Lab, Department of Electrical and Computer
  Engineering
  University of Porto
  E-mail:gpai@obelix.fe.up.pt
_____________________________________________________________________

  These software routines were developed for the XITE system under the written
  permission of B-lab, Department of Informatics, University of Oslo.
_____________________________________________________________________
*/
/*****************************************************************************/

/* INCLUDES */

#include <stdio.h>
#include <math.h>
#include "memory.h"
#include "matrix.h"
#include "states.h"
#include "model.h"

/*****************************************************************************/

#ifndef lint
static char *SccsId = "@(#)model.c 1.0 94/05/19, DEEC, UoP";
#endif
```

```
/*****************************************************************************/

/*F:create_line*
_____________________________________________________________________

  create_line
_____________________________________________________________________

Name:        create_line - create one Line's element

Syntax:     | Line *create_line()

Description:    'create_line' performs the creation of one Line's element.

Return value:  | The element's pointer.

Example:       | Try yourself.

Restrictions:   None.

Author:          Joao Tavares

Id:              @(#)create_line.c   1.00   94/05/19
_____________________________________________________________________
*/

Line *create_line()

{

  Line *el;

  el = (Line *)memory_get(sizeof(Line));
  el->state.meanpos = create_matrix(MNNRPOS,MNNCPOS);
  el->state.covrpos = create_matrix(CVNRPOS,CVNCPOS);
  el->state.meandir = create_matrix(MNNRDL,MNNCDL);
  el->state.covrdir = create_matrix(CVNRDL,CVNCDL);
  el->state.meanlen = create_matrix(MNNRDL,MNNCPOS);
  el->state.covrlen = create_matrix(CVNRDL,CVNCDL);

  return(el);

}
```

```
/*********************************************************************/

/*F:delete_line*
_____________________________________________________________________

 delete_line
_____________________________________________________________________

Name:       delete_line - delete one Line's element

Syntax:     | void delete_line(el)
            | Line *el;

Description:  'delete_line' delete one Line's element.
              'el' Line's element to delete.

Return value: | None.

Example:      | Try yourself.

Restrictions:  None.

Author:        Joao Tavares

Id:            @(#)delete_line.c   1.00   94/05/19
_____________________________________________________________________
*/

void delete_line(el)
Line *el;

{

 if (el != NULL)
 {

   delete_matrix(el->state.meanpos);
   delete_matrix(el->state.covrpos);
   delete_matrix(el->state.meandir);
   delete_matrix(el->state.covrdir);
   delete_matrix(el->state.meanlen);
   delete_matrix(el->state.covrlen);
   free(el);
   el = NULL;
 }

 return;

}

/*********************************************************************/

/*F:print_line*
_____________________________________________________________________

 print_line
_____________________________________________________________________

Name:       print_line - print one Line's element

Syntax:     | void print_line(ep)
            | Line *ep;

Description:  'print_line' print one Line's element.
              'el' Line's element to print.

Return value: | None.

Example:      | Try yourself.

Restrictions:  None.

Author:        Joao Tavares

Id:            @(#)print_line.c   1.00   94/05/19
_____________________________________________________________________
*/

void print_line(ep)
Line *ep;

{

 int i;
 char s;

 if (ep == NULL) return;
 puts("Line:\n");
```

```
  printf("\nReference: %d\nConfidence factor= %d\n\nNew coordinates in the frame buffer:\n\tFirst
point: (%d, %d)\tLast point: (%d, %d)\n", (ep)->reference, (ep)->cf, (ep)->xifnew, (ep)->yifnew,
(ep)->xffnew, (ep)->yffnew);
  puts("\nPause, press enter to continue.\n");
  s = getc(stdin);
  printf("\n\nPosition state:\n\n");
  for (i = 0; i < 6; ++i)
  printf("%.2f  ", (ep)->state.meanpos->d[i]);
  print_matrix((ep)->state.covrpos, "\n\nCovariance:\n\n");
  puts("\nPause, press enter to continue.\n");
  s = getc(stdin);
  printf("\n\nDirection state:\n\n");
  for (i = 0; i < 3; ++i)
  printf("%.2f  ", (ep)->state.meandir->d[i]);
  print_matrix((ep)->state.covrdir, "\n\nCovariance:\n\n");
  puts("\nPause, press enter to continue.\n");
  s = getc(stdin);
  printf("\n\nLength state:\n\n");
  for (i = 0; i < 3; ++i)
  printf("%.2f  ", (ep)->state.meanlen->d[i]);
  print_matrix((ep)->state.covrlen, "\n\nCovariance:\n\n");
  puts("\nPause, press enter to continue.\n");
  s = getc(stdin);

  return;

}

/*************************************************************************/

/*F:create_mpp*
_________________________________________________________________

  create_mpp
_________________________________________________________________

Name:        create_mpp - create one Mpp's element

Syntax:      | Mpp *create_mpp()

Description:     'create_mpp' performs the creation of one Mpp's element.

Return value:   | The element's pointer.

Example:         | Try yourself.

Restrictions:   None.

Author:          Joao Tavares

Id:              @(#)create_mpp.c   1.00   94/05/19
_________________________________________________________________
*/

Mpp *create_mpp()

{

  Mpp *el;

  el = (Mpp *)memory_get(sizeof(Mpp));

  return(el);

}

/*************************************************************************/

/*F:delete_mpp*
_________________________________________________________________

  delete_mpp
_________________________________________________________________

Name:        delete_mpp - delete one Mpp's element

Syntax:      | void delete_mpp(el)
             | Mpp *el;

Description:   'delete_mpp' delete one Mpp's element.
               'el' is the Mpp's element to delete.

Return value:  | None.

Example:         | Try yourself.

Restrictions:   None.

Author:          Joao Tavares
```

```
Id:             @(#)delete_mpp.c   1.00   94/05/19
_____________________________________________________________________
*/

void delete_mpp(el)
Mpp *el;

{

  if (el != NULL)
  {

    free(el);
    el = NULL;

  }

  return;

}

/*********************************************************************/
```

3.2.8 - Ficheiro states.h

Este ficheiro contém as definições de variáveis e de protótipos relativamente aos módulos que pertencem ao grupo ***V*** da implementação ***deep***.

```
/*
_____________________________________________________________________

  states.h
  @(#)states.h   1.0 94/05/19, Copyright 1994, DEEC, UoP
  Image processing lab,
  Department of Electrical and Computer Engineering
  University of Porto
  E-mail: gpai@obelix.fe.up.pt
_____________________________________________________________________

  File Name: states.h
  Purpose  : Definitions for state data structures.
  Author   : Joao Tavares, DEEC, UoP
  Version  : 1.0
  Date     : 19/05/94
_____________________________________________________________________
*/

/*********************************************************************/

/* CONSTANTS DEFINITIONS */

#define MNNRPOS (6)     /* number of rows in the state mean position matrix */
#define MNNCPOS (1)     /* number of columns in the state mean position matrix */
#define CVNRPOS (6)     /* number of rows in the state covariance position matrix */
#define CVNCPOS (6)     /* number of columns in the state covariance position matrix */
#define MNNRDL (3)      /* number of rows in the state mean direction/length matrix */
#define MNNCDL (1)      /* number of columns in the state mean direction/length matrix */
#define CVNRDL (3)      /* number of rows in the state covariance direction/length matrix */
#define CVNCDL (3)      /* number of columns in the state covariance direction/length matrix */

/*********************************************************************/

/* TYPES DEFINITIONS */

typedef struct _state
{

  Matrix *meanpos;        /* position mean's vector */
  Matrix *covrpos;        /* position variance's matrix */
  Matrix *meandir;        /* direction mean's vector */
  Matrix *covrdir;        /* direction variance's vector */
  Matrix *meanlen;        /* length mean's vector */
  Matrix *covrlen;        /* length variance's matrix */

} State;

/*********************************************************************/

/* FUNCTIONS PROTOTYPES */

void InitStateSpace();
void RemoveStateSpace();
void statePredivtCovrpos();
void statePredict();
void stateUpdate();
double stateDistance();
void copyState();
```

```
/*************************************************************************/
```

3.2.9 - Ficheiro states.c

Este ficheiro contém a listagem relativa aos módulos que pertencem ao grupo ***V*** da implementação ***deep***.

```
/*
-------------------------------------------------------------------------

  states.c
  @(#)states.c   1.0 94/05/19, Copyright 1994, DEEC, UoP
  Image processing lab, Department of Electrical and Computer
  Engineering
  University of Porto
  E-mail: gpai@obelix.fe.up.pt

-------------------------------------------------------------------------

  These software routines were developed for the XITE system under the written
  permission of B-lab, Department of Informatics, University of Oslo.

-------------------------------------------------------------------------
*/
/*************************************************************************/

/* INCLUDES */

#include <stdio.h>
#include <math.h>
#include "matrix.h"
#include "states.h"

/*************************************************************************/

#ifndef lint
static char *SccsId = "@(#)states.c 1.0 94/05/19, DEEC, UoP";
#endif

/*************************************************************************/

/* CONSTANTS DEFINITIONS */

#define T (1.0)                 /* time elapse between each two frames */
#define APOS (0.85)             /* reduction factor for position's acceleration */
#define VAR_XPOS (8.0)    /* variance of dynamic system noise for x position */
#define VAR_YPOS (4.0)    /* variance of dynamic system noise for y position */
#define VAR_XVEL (2.0)    /* variance of dynamic system noise for x velocity */
#define VAR_YVEL (2.0)    /* variance of dynamic system noise for y velocity */
#define VAR_APOS (8.0)    /* variance of dynamic system noise for position acceleration */
#define VAR_DIR  (1.5)    /* variance of dynamic system noise for direction */
#define VAR_VDIR (0.5)    /* variance of dynamic system noise for direction's velocity */
#define ADIR (0.75)           /* reduction factor for direction's acceleration */
#define VAR_ADIR (1.0)    /* variance of dynamic system noise for direction's acceleration */
#define VAR_LEN  (5.5)    /* variance of dynamic system noise for length */
#define VAR_VLEN (0.5)    /* variance of dynamic system noise for length's velocity */
#define ALEN (0.75)           /* reduction factor for length's acceleration */
#define VAR_ALEN (1.0)    /* variance of dynamic system noise for length's acceleration */

/* position's dynamic matrix's data */

static double dinadatapos[] = { (1.0), (0.0), T, (0.0), (T*T)/2.0, (0.0),
                                (0.0), (1.0), (0.0), T, (0.0), (T*T)/2.0,
                                (0.0), (0.0), (1.0), (0.0), T, (0.0),
                                (0.0), (0.0), (0.0), (1.0), (0.0), T,
                                (0.0), (0.0), (0.0), (0.0), APOS, (0.0),
                                (0.0), (0.0), (0.0), (0.0), (0.0), APOS };

/* position's dynamic noise matrix's data */

static double dnoidatapos[] = { VAR_XPOS, (0.0), (0.0), (0.0), (0.0), (0.0),
                                (0.0), VAR_YPOS, (0.0), (0.0), (0.0), (0.0),
                                (0.0), (0.0), VAR_XVEL, (0.0), (0.0), (0.0),
                                (0.0), (0.0), (0.0), VAR_YVEL, (0.0), (0.0),
                                (0.0), (0.0), (0.0), (0.0), VAR_APOS, (0.0),
                                (0.0), (0.0), (0.0), (0.0), (0.0), VAR_APOS };

/* position's measure matrix's data */

static double measdatapos[] = { (1.0), (0.0), (0.0), (0.0), (0.0), (0.0),
                                (0.0), (1.0), (0.0), (0.0), (0.0), (0.0) };

/* direction's dynamic matrix's data */

static double dinadatadir[] = { (1.0), T, (T*T)/2.0,
                                (0.0), (1.0), T,
                                (0.0), (0.0), ADIR };

/* direction's dynamic noise matrix's data */
```

```
static double dnoidatadir[] = { VAR_DIR, (0.0), (0.0),
                                (0.0), VAR_VDIR, (0.0),
                                (0.0), (0.0), VAR_ADIR };

/* direction's measure vector's data */

static double measdatadir[] = { (1.0), (0.0), (0.0) };

/* length's dynamic matrix's data */

static double dinadatalen[] = { (1.0), T, (T*T)/2.0,
                                (0.0), (1.0), T,
                                (0.0), (0.0), ALEN };

/* length's dynamic noise matrix's data */

static double dnoidatalen[] = { VAR_LEN, (0.0), (0.0),
                                (0.0), VAR_VLEN, (0.0),
                                (0.0), (0.0), VAR_ALEN };

/* length's measure vector's data */

static double measdatalen[] = { (1.0), (0.0), (0.0) };

/* matrix declarations */

static Matrix dinamxpos = { MNNRPOS, MNNRPOS, dinadatapos};

static Matrix measmxpos = { 2*MNNCPOS, MNNRPOS, measdatapos};

static Matrix dnoisepos = { MNNRPOS, MNNRPOS, dnoidatapos};

static Matrix dinamxdir = { MNNRDL, MNNRDL, dinadatadir};

static Matrix measmxdir = { MNNCDL, MNNRDL, measdatadir};

static Matrix dnoisedir = { CVNCDL, CVNRDL, dnoidatadir};

static Matrix dinamxlen = { MNNRDL, MNNRDL, dinadatalen};

static Matrix measmxlen = { MNNCDL, MNNRDL, measdatalen};

static Matrix dnoiselen = { CVNCDL, CVNRDL, dnoidatalen};

/*****************************************************************/

/* COMMON VARIABLES DEFINITIONS */

static Matrix *mespos, *sumpos, *difpos, *multpos, *kgainpos, *mesdir, *sumdir,
 *difdir, *multdir, *kgaindir, *meslen, *sumlen, *diflen, *multlen, *kgainlen, *tmp;

Matrix *identpos, *identdl;
State ps, ms, us;

/*****************************************************************/

/*F:InitSpace*
_________________________________________________________________

 InitSpace
_________________________________________________________________

Name:           InitStateSpace - allocation of the necessary state's space

Syntax:         | void InitStateSpace()

Description:    'InitSpace' performs the allocation of the necessary state's space.

Return value:   | None.

Example:        | Try yourself.

Restrictions:   None.

Author:         Joao Tavares

Id:             @(#)InitSpace.c     1.00 94/05/19
_________________________________________________________________
*/

void InitStateSpace()

{

  register int i;

  ps.meanpos = create_matrix(MNNRPOS, MNNCPOS);
  ps.covrpos = create_matrix(CVNRPOS, CVNCPOS);
  ms.meanpos = create_matrix(MNNRPOS, MNNCPOS);
  ms.covrpos = create_matrix(CVNRPOS, CVNCPOS);
```

```
  us.meanpos = create_matrix(MNNRPOS, MNNCPOS);
  us.covrpos = create_matrix(CVNRPOS, CVNCPOS);
  mespos = create_matrix(_nrow(&measmxpos), MNNCPOS);
  sumpos = create_matrix(_nrow(&measmxpos), _nrow(&measmxpos));
  difpos = create_matrix(_nrow(&dinamxpos), _ncol(&dinamxpos));
  multpos = create_matrix(_ncol(&measmxpos), _nrow(&measmxpos));
  kgainpos = create_matrix(_ncol(&measmxpos), _nrow(&measmxpos));
  identpos = create_matrix(_nrow(&dinamxpos), _ncol(&dinamxpos));
  for (i = 0; i < _nrow(identpos); i++) _data(identpos,i,i) = (1.0);
  ps.meandir = create_matrix(MNNRDL, MNNCDL);
  ps.covrdir = create_matrix(CVNRDL, CVNCDL);
  ms.meandir = create_matrix(MNNRDL, MNNCDL);
  ms.covrdir = create_matrix(CVNRDL, CVNCDL);
  us.meandir = create_matrix(MNNRDL, MNNCDL);
  us.covrdir = create_matrix(CVNRDL, CVNCDL);
  kgaindir = create_matrix(_ncol(&measmxdir), _nrow(&measmxdir));
  difdir = create_matrix(_nrow(&dinamxdir), _ncol(&dinamxdir));
  mesdir = create_matrix(_nrow(&measmxdir), MNNCDL);
  sumdir = create_matrix(_nrow(&measmxdir), _nrow(&measmxdir));
  multdir = create_matrix(_ncol(&measmxdir), _nrow(&measmxdir));
  ps.meanlen = create_matrix(MNNRDL, MNNCDL);
  ps.covrlen = create_matrix(CVNRDL, CVNCDL);
  ms.meanlen = create_matrix(MNNRDL, MNNCDL);
  ms.covrlen = create_matrix(CVNRDL, CVNCDL);
  us.meanlen = create_matrix(MNNRDL, MNNCDL);
  us.covrlen = create_matrix(CVNRDL, CVNCDL);
  kgainlen = create_matrix(_ncol(&measmxlen), _nrow(&measmxlen));
  diflen = create_matrix(_nrow(&dinamxdir), _ncol(&dinamxdir));
  meslen = create_matrix(_nrow(&measmxdir), MNNCDL);
  sumlen = create_matrix(_nrow(&measmxdir), _nrow(&measmxdir));
  multlen = create_matrix(_ncol(&measmxdir), _nrow(&measmxdir));
  identdl = create_matrix(_nrow(&dinamxdir), _ncol(&dinamxdir));
  for (i = 0; i < _nrow(identdl); i++) _data(identdl,i,i) = (1.0);
  tmp = create_matrix(2, 2);

  return;

}

/*****************************************************************************/

/*F:RemoveStateSpace*
_____________________________________________________________

  RemoveStateSpace
_____________________________________________________________

Name:     RemoveStateSpace - desallocation of the state's space

Syntax:     | void RemoveStateSpace()

Description:   'RemoveStateSpace' performs the desallocation of the state's space.

Return value:   | None.

Example:        | Try yourself.

Restrictions:    None.

Author:          Joao Tavares

Id:              @(#)RemoveStateSpace.c   1.00   94/05/19
_____________________________________________________________
*/

void RemoveStateSpace()

{

  delete_matrix(ps.meanpos);
  delete_matrix(ps.covrpos);
  delete_matrix(ms.meanpos);
  delete_matrix(ms.covrpos);
  delete_matrix(us.meanpos);
  delete_matrix(us.covrpos);
  delete_matrix(identpos);
  delete_matrix(kgainpos);
  delete_matrix(multpos);
  delete_matrix(difpos);
  delete_matrix(sumpos);
  delete_matrix(mespos);
  delete_matrix(ps.meandir);
  delete_matrix(ps.covrdir);
  delete_matrix(ms.meandir);
  delete_matrix(ms.covrdir);
  delete_matrix(us.meandir);
  delete_matrix(us.covrdir);
  delete_matrix(kgaindir);
  delete_matrix(multdir);
  delete_matrix(difdir);
```

```
  delete_matrix(sumdir);
  delete_matrix(mesdir);
  delete_matrix(ps.meanlen);
  delete_matrix(ps.covrlen);
  delete_matrix(ms.meanlen);
  delete_matrix(ms.covrlen);
  delete_matrix(us.meanlen);
  delete_matrix(us.covrlen);
  delete_matrix(kgainlen);
  delete_matrix(multlen);
  delete_matrix(diflen);
  delete_matrix(sumlen);
  delete_matrix(meslen);
  delete_matrix(identdl);
  delete_matrix(tmp);

  return;

}

/*************************************************************************/

/*F:statePredictCovrpos*
_____________________________________________________________________

  statePredictCovrpos
_____________________________________________________________________

Name:        statePredictCovrpos - state's position's variance predict by Kalman's
                                   equation

Syntax:     | void statePredictCovrpos(cstate, pstate)
            | State *cstate, *pstate;

Description:  'statePredict' predict the new state's position's variance by the
              Kalman's equation.
              'cstate' is correspondent current state.
              'pstate' is the output predict state.

Return value: | None.

Example:      | Try yourself.

Restrictions:  None.

Author:        Joao Tavares

Id:            @(#)statePredictCovrpos.c   1.00   94/05/19
_____________________________________________________________________
*/

void statePredictCovrpos(cstate, pstate)
State *cstate, *pstate;

{

  /*---------- predict state covariance ------------------*/

  matrix_wsqr(&dinamxpos, cstate->covrpos, pstate->covrpos, EQU);
  matrix_opr(&dnoisepos, pstate->covrpos, pstate->covrpos, 0.0, ADD, EQU);

  return;

}

/*************************************************************************/

/*F:statePredict*
_____________________________________________________________________

  statePredict
_____________________________________________________________________

Name:        statePredict - state's predict by Kalman's equation

Syntax:     | void statePredict(cstate, pstate)
            | State *cstate, *pstate;

Description:  'statePredict' predict the new state by the Kalman's equation.
              'cstate' is correspondent current state.
              'pstate' is the output predict state.

Return value: | None.

Example:      | Try yourself.

Restrictions:  None.

Author:        Joao Tavares
```

```
Id:             @(#)statePredict.c   1.00   94/05/19
_____________________________________________________________________
*/

void statePredict(cstate, pstate)
State *cstate, *pstate;

{

  /*------------------- predict state mean --------------------*/

  matrix_mult(&dinamxpos, cstate->meanpos, pstate->meanpos, EQU, EQU);
  matrix_mult(&dinamxdir, cstate->meandir, pstate->meandir, EQU, EQU);
  matrix_mult(&dinamxlen, cstate->meanlen, pstate->meanlen, EQU, EQU);

  /*--------------- predict state covariance ------------------*/

  matrix_wsqr(&dinamxpos, cstate->covrpos, pstate->covrpos, EQU);
  matrix_opr(&dnoisepos, pstate->covrpos, pstate->covrpos, 0.0, ADD, EQU);
  matrix_wsqr(&dinamxdir, cstate->covrdir, pstate->covrdir, EQU);
  matrix_opr(&dnoisedir, pstate->covrdir, pstate->covrdir, 0.0, ADD, EQU);
  matrix_wsqr(&dinamxlen, cstate->covrlen, pstate->covrlen, EQU);
  matrix_opr(&dnoiselen, pstate->covrlen, pstate->covrlen, 0.0, ADD, EQU);

  return;

}

/*************************************************************************/

/*F:stateUpdate*
_____________________________________________________________________

  stateUpdate
_____________________________________________________________________

Name:       stateUpdate - state's update by Kalman's equation

Syntax:     | void stateUpdate(pstate, mstate, ustate)
            | State *pstate, *mstate, *ustate;

Description:  'stateUpdate' performs the updating of one state by the Kalman's
              equation.
              'pstate' is correspondent Kalman's predicted state.
              'mstate' is correspondent measured state.
              'ustate' is the output Kalman's update state.

Return value: | None.

Example:      | Try yourself.

Restrictions:  None.

Author:        Joao Tavares

Id:            @(#)stateUpdate.c   1.00   94/05/19
_____________________________________________________________________
*/

void stateUpdate(pstate, mstate, ustate)
State *pstate, *mstate, *ustate;

{

  /*----------------- calculate kalman gain -------------------*/

  matrix_opr(mstate->meanpos, mespos, mespos, 0.0, EQU, EQU);
  matrix_opr(mstate->covrpos, sumpos, sumpos, 0.0, EQU, EQU);
  matrix_wsqr(&measmxpos, pstate->covrpos, sumpos, ADD);
  matrix_invert(sumpos);
  matrix_mult(pstate->covrpos, &measmxpos, multpos, EQU, TRP);
  matrix_mult(multpos, sumpos, kgainpos, EQU, EQU);
  matrix_opr(mstate->meandir, mesdir, mesdir, 0.0, EQU, EQU);
  matrix_opr(mstate->covrdir, sumdir, sumdir, 0.0, EQU, EQU);
  matrix_wsqr(&measmxdir, pstate->covrdir, sumdir, ADD);
  if (_data(sumdir, 0, 0) != 0.0) _data(sumdir, 0, 0) = 1.0/(_data(sumdir, 0, 0));
  matrix_mult(pstate->covrdir, &measmxdir, multdir, EQU, TRP);
  matrix_mult(multdir, sumdir, kgaindir, EQU, EQU);
  matrix_opr(mstate->meanlen, meslen, meslen, 0.0, EQU, EQU);
  matrix_opr(mstate->covrlen, sumlen, sumlen, 0.0, EQU, EQU);
  matrix_wsqr(&measmxlen, pstate->covrlen, sumlen, ADD);
  if (_data(sumlen, 0, 0) != 0.0) _data(sumlen, 0, 0) = 1.0/(_data(sumlen, 0, 0));
  matrix_mult(pstate->covrlen, &measmxlen, multlen, EQU, TRP);
  matrix_mult(multlen, sumlen, kgainlen, EQU, EQU);

  /*----------------- calculate [[I]-[K]*[H]] -----------------*/

  matrix_opr(identpos, difpos, difpos, 0.0, EQU, EQU);
  matrix_mult(kgainpos, &measmxpos, difpos, DIF, EQU);
  matrix_opr(identdl, difdir, difdir, 0.0, EQU, EQU);
```

```
  matrix_mult(kgaindir, &measmxdir, difdir, DIF, EQU);
  matrix_opr(identdl, diflen, diflen, 0.0, EQU, EQU);
  matrix_mult(kgainlen, &measmxlen, diflen, DIF, EQU);

  /*-------------------- update state mean --------------------*/

  matrix_mult(kgainpos, mstate->meanpos, ustate->meanpos, EQU, EQU);
  matrix_mult(difpos, pstate->meanpos, ustate->meanpos, ADD, EQU);
  matrix_mult(kgaindir, mstate->meandir, ustate->meandir, EQU, EQU);
  matrix_mult(difdir, pstate->meandir, ustate->meandir, ADD, EQU);
  matrix_mult(kgainlen, mstate->meanlen, ustate->meanlen, EQU, EQU);
  matrix_mult(diflen, pstate->meanlen, ustate->meanlen, ADD, EQU);

  /*----------------- update state covariance -----------------*/

  matrix_wsqr(kgainpos, mstate->covrpos, ustate->covrpos, EQU);
  matrix_wsqr(difpos, pstate->covrpos, ustate->covrpos, ADD);
  matrix_wsqr(kgaindir, mstate->covrdir, ustate->covrdir, EQU);
  matrix_wsqr(difdir, pstate->covrdir, ustate->covrdir, ADD);
  matrix_wsqr(kgainlen, mstate->covrlen, ustate->covrlen, EQU);
  matrix_wsqr(diflen, pstate->covrlen, ustate->covrlen, ADD);

  return;

}

/*****************************************************************************/

/*F:stateDistance*
_____________________________________________________________________________

  stateDistance
_____________________________________________________________________________

Name:     stateDistance - compute the Mahalanobis's distance between two states

Syntax:     | double stateDistance(s1, s2)
            | State *s1, *s2;

Description:  'stateDistance' performs the calculation of the Mahalanobis's
              distance for position between the states 's1' and 's2'.

Return value:  | The Mahalanobis's distance.

Example:       | Try yourself.

Restrictions:   None.

Author:         Joao Tavares

Id:             @(#)stateDistance.c   1.00   94/05/19
_____________________________________________________________________________
*/

double stateDistance(s1, s2)
State *s1, *s2;

{

  matrix_opr(mespos, mespos, mespos, 0.0, MSC, EQU);
  matrix_opr(s1->meanpos, s2->meanpos, mespos, 0.0, DIF, EQU);
  matrix_opr(sumpos, sumpos, sumpos, 0.0, MSC, EQU);
  matrix_opr(s1->covrpos, s2->covrpos, sumpos, 0.0, ADD, EQU);
  matrix_invert(sumpos);
  _data(tmp, 0, 0) = _data(mespos, 0, 0);
  _data(tmp, 0, 1) = _data(mespos, 1, 0);
  matrix_wsqr(tmp, sumpos, difpos, EQU);

  return(0.5*(difpos->d[0]));

}

/*****************************************************************************/

/*F:copyState*
_____________________________________________________________________________

  copyState
_____________________________________________________________________________

Name:       copyState - copy one state to an other

Syntax:     | void copyState(s1,s2)
            | State *s1, *s2;

Description:     'copyState' performs the copy of the state 's1' to the state 's2'.

Return value:   | None.

Example:        | Try yourself.
```

```
Restrictions:    None.

Author:          Joao Tavares

Id:              @(#)copyState.c   1.00   94/05/19
_____________________________________________________________________
*/

void copyState(s1,s2)
State *s1, *s2;

{

  matrix_opr(s1->meanpos, s1->meanpos, s2->meanpos, 0.0, EQU, EQU);
  matrix_opr(s1->covrpos, s1->covrpos, s2->covrpos, 0.0, EQU, EQU);
  matrix_opr(s1->meandir, s1->meandir, s2->meandir, 0.0, EQU, EQU);
  matrix_opr(s1->covrdir, s1->covrdir, s2->covrdir, 0.0, EQU, EQU);
  matrix_opr(s1->meanlen, s1->meanlen, s2->meanlen, 0.0, EQU, EQU);
  matrix_opr(s1->covrlen, s1->covrlen, s2->covrlen, 0.0, EQU, EQU);

  return;

}

/*********************************************************************/
```

3.2.10 - Ficheiro matrix.h

Este ficheiro contém as definições de variáveis e de protótipos relativamente aos módulos que pertencem ao grupo ***VI*** da implementação ***deep***.

```
/*
_____________________________________________________________________

  matrix.h
  @(#)matrix.h   1.1 94/05/21, Copyright 1994, DEEC, UoP
  Image processing lab,
  Department of Electrical and Computer Engineering
  University of Porto
  E-mail: gpai@obelix.fe.up.pt
_____________________________________________________________________

  File Name: matrix.h
  Purpose  : Definitions for handling matrix structures and perform matrix
             calculations.
  Author   : Miguel Correia, DEEC, UoP
             Revised by Joao Tavares, DEEC, UoP
  Version  : 1.1
  Date     : 21/05/94
_____________________________________________________________________
*/

/*********************************************************************/

/* TYPES DEFINITIONS */

typedef enum {EQU, ADD, DIF, NEG, MSC, TRP} Mopr;  /* matrix's operations */

typedef struct _mxsize
{

  int nr, nc;    /* matrix's size */

} Mxsize;

typedef struct _matrix
{

  int nr, nc;     /* matrix's dimension */
  double *d;    /* pointer to the matrix's data */

} Matrix;

/*********************************************************************/

/* MACROS DEFINITIONS */

#define _nrow(mx)     ((mx)->nr)                          /* number of rows of a matrix */
#define _ncol(mx)     ((mx)->nc)                          /* number of columns of a matrix */
#define _data(mx,r,c) *((mx)->d+(r)*_ncol(mx)+(c))    /* one element of the matrix */

/*********************************************************************/

/* FUNCTIONS PROTOTYPES */

Matrix *create_matrix();
void delete_matrix();
```

```
void print_matrix();
int matrix_opr();
int matrix_mult();
int matrix_wsqr();
int matrix_invert();
int matrix_eig();

/*****************************************************************************/
```

3.2.11 - Ficheiro matrix.c

Este ficheiro contém a listagem relativa aos módulos que pertencem ao grupo ***VI*** da implementação ***deep***.

```
/*
_____________________________________________________________________

  matrix.c
  @(#)matrix.c   1.1 94/05/21, Copyright 1994, DEEC, UoP
  Image processing lab,
  Department of Electrical and Computer Engineering
  University of Porto
  E-mail: gpai@obelix.fe.up.pt
_____________________________________________________________________

  File Name: matrix.c
  Purpose  : Source code of matrix handling and calculations routines.
  Author   : Miguel Correia, DEEC, UoP
             Revised by Joao Tavares, DEEC, UoP
  Version  : 1.1
  Date     : 21/05/94
_____________________________________________________________________
*/

/*****************************************************************************/

/* INCLUDES */

#include <stdio.h>
#include <malloc.h>
#include <math.h>
#include <blab/message.h>
#include "gdefs.h"
#include "memory.h"
#include "matrix.h"

/*****************************************************************************/

#ifndef lint
static char *SccsId = "@(#)matrix.c 1.0 94/05/21, DEEC, UoP";
#endif

/*****************************************************************************/

/*F:create_matrix*
_____________________________________________________________________

  create_matrix
_____________________________________________________________________

Name:        create_matrix - create one matrix

Syntax:     | Matrix *create_matrix(nr, nc)
            | int nr, nc;

Description:  'create_matrix' performs the creation of one matrix with 'nr' rows
              and 'nc' columns.

Return value:  | Matrix's pointer.

Example:       | Try yourself.

Restrictions:  Enough memory.

Author:        Miguel Correia

Id:            @(#)create_matrix.c   1.00   94/05/21
_____________________________________________________________________
*/

Matrix *create_matrix(nr, nc)
int nr, nc;

{

  int tsize = nr*nc;
  double *d;
  Matrix *mx;
```

```
  mx = (Matrix *)memory_get(sizeof(Matrix));
  mx->d = d = (double *)memory_get(tsize*sizeof(double));
  for (; tsize; tsize--) *d++ = (double) 0.0;
  _nrow(mx) = nr;
  _ncol(mx) = nc;

  return(mx);

}

/*****************************************************************/

/*F:delete_matrix*
_________________________________________________________________

  delete_matrix
_________________________________________________________________

Name:        delete_matrix - delete one matrix

Syntax:     | void delete_matrix(mx)
            | Matrix *mx;

Description:     'delete_matrix' delete the 'mx' matrix.

Return value:   | None.

Example:         | Try yourself.

Restrictions:    None.

Author:          Miguel Correia

Id:              @(#)delete_matrix.c   1.00   94/05/21
_________________________________________________________________
*/

void delete_matrix(mx)
Matrix *mx;

{

  if(mx == NULL) return;
  else
  {

    free(mx->d);
    free(mx);

  }

  return;

}

/*****************************************************************/

/*F:print_matrix*
_________________________________________________________________

  print_matrix
_________________________________________________________________

Name:        print_matrix - print one matrix

Syntax:     | void print_matrix(mx, name)
            | Matrix *mx;
            | char *name;

Description:   'print_matrix' print the 'mx' matrix in MATLAB's format.
               'name' is the matrix's title.

Return value:  | None.

Example:         | Try yourself.

Restrictions:    None.

Author:          Miguel Correia

Id:              @(#)print_matrix.c   1.00   94/05/21
_________________________________________________________________
*/

void print_matrix(mx, name)
Matrix *mx;
char *name;

{
```

```
  int r, c;

  printf("%s = [",name);
  for (r = 0; r < _nrow(mx); r++)
  {

    for(c = 0; c < _ncol(mx); c++) printf("  %.2f", _data(mx,r,c));
    printf(";\n\t");

  }
  printf("]\n");

  return;

}

/*************************************************************************/

/*F:matrix_opr*
_____________________________________________________________________

  matrix_opr
_____________________________________________________________________

Name:       matrix_opr - matrices's add

Syntax:     | matrix_opr(m1,m2,mres,d,op,tr)
            | Matrix *m1, *m2, *mres;
            | double d;
            | Mopr op,tr;

Description:  'matrix_opr' performs the calculations
              '[mres] = sign*[m1]+fact[m2]'. If 'tr' is 'EQU' and if 'op' is:

                  - 'EQU' then sign=1.0 and fact=0,
                  - 'ADD' then sign=1.0 and fact=1,
                  - 'DIF' then sign=1.0 and fact=-1,
                  - 'NEG' then sign=-1.0 and fact=0,
                  - 'MSC' then sign='d' and fact=0.

              If 'tr' is 'TRP' is like above but with 'm1' transposed.

Return value: | 0 => Ok.
              | 1 => If any input matrix is NULL.

Example:      | Try yourself.

Restrictions: None.

Author:       Miguel Correia

Id:           @(#)matrix_opr.c   1.00   94/05/21
_____________________________________________________________________
*/

matrix_opr(m1,m2,mres,d,op,tr)
Matrix *m1, *m2, *mres;
double d;
Mopr op,tr;

{

  int r, c, nr, nc;
  double sign, fact;

  if (m1 == NULL || m2 == NULL || mres == NULL) return(1);
  sign = (double)(1.0);
  fact = (double)(0.0);
  switch (op)
  {

    case EQU:
      break;
    case ADD:
      fact = (double)(1.0);
      break;
    case DIF:
      fact = (double)(-1.0);
      break;
    case NEG:
      sign = (double)(-1.0);
      break;
    case MSC:
      sign = d;
      break;

  }
  nr = MIN2(MIN2(_nrow(m1),_nrow(m2)),(tr == EQU?_nrow(mres):_ncol(mres)));
  nc = MIN2(MIN2(_ncol(m1),_ncol(m2)),(tr == EQU?_ncol(mres):_nrow(mres)));
```

```
  for (r = 0; r < nr; r++)
    for (c = 0; c < nc; c++)
      _data(mres,r,c) = sign *_data(m1,(tr == EQU? r:c),(tr == EQU? c:r)) + fact * _data(m2,r,c);

  return(0);

}

/*****************************************************************************/

/*F:matrix_mult*
_____________________________________________________________________________

  matrix_mult
_____________________________________________________________________________

Name:       matrix_mult - matrices's multiplication

Syntax:     | matrix_mult(m1,m2,mmult,op,tr)
            | Matrix *m1, *m2, *mmult;
            | Mopr op, tr;

Description: 'matrix_mult' performs the calculations:
                 If 'tr' is EQU:
                    - '[mmult]=[m1]*[m2]' for 'op'=EQU,
                    - '[mmult]-=[m1]*[m2]' for 'op'=DIF,
                    - '[mmult]+=[m1]*[m2]' for 'op'=ADD.
                 If 'tr' is TRP then the calculations are like above but with the matrix
                 '[m2]' transposed.

Return value:  | 0 => Ok.
               | 1 => If any input matrix is NULL.

Example:       | Try yourself.

Restrictions:   None.

Author:        Miguel Correia

Id:            @(#)matrix_mult.c   1.00   94/05/21
_____________________________________________________________________________
*/

matrix_mult(m1,m2,mmult,op,tr)
Matrix *m1, *m2, *mmult;
Mopr op, tr;

{

 int i, ni, r, c;
 double sign, fact;

 if (m1 == NULL || m2 == NULL || mmult == NULL) return(1);
 sign = (double)(1.0);
 fact = (double)(0.0);
 switch (op)
 {

  case EQU:
    break;
  case ADD:
    fact = (double)(1.0);
    break;
  case DIF:
    sign = (double)(-1.0);
    fact = (double)(1.0);
    break;

 }
 ni = MIN2(_ncol(m1),(tr==EQU? _nrow(m2): _ncol(m2)));
 for (r = 0; r < _nrow(mmult); r++)
  for (c = 0; c < _ncol(mmult); c++)
  {

    _data(mmult,r,c) *= fact;
    for (i = 0; i < ni; i++)
      _data(mmult,r,c) += sign *
        _data(m1,r,i) * _data(m2,(tr == EQU? i:c), (tr == EQU? c:i));

  }

 return(0);

}

/*****************************************************************************/

/*F:matrix_wsqr*
_____________________________________________________________________________
```

```
  matrix_wsqr
_____________________________________________________________________

Name:       matrix_wsqr - compute [m1]*[m2]*(transposed[m1])

Syntax:     | matrix_wsqr(m1,m2,msqr,op)
            | Matrix *m1, *m2, *msqr;
            | Mopr op;

Description: 'matrix_wsqr' performs the calculations:
                - '[msqr]=[m1]*[m1]*(transposed[m1])' for 'op'=EQU,
                - '[msqr]-=[m1]*[m1]*(transposed[m1])' for 'op'=DIF,
                - '[msqr]+=[m1]*[m1]*(transposed[m1])' for 'op'=ADD.

Return value:  | 0 => Ok.
               | 1 => If any input matrix is NULL.

Example:       | Try yourself.

Restrictions:   None.

Author:        Miguel Correia

Id:            @(#)matrix_wsqr.c   1.00   94/05/21
_____________________________________________________________________
*/

matrix_wsqr(m1,m2,msqr,op)
Matrix *m1, *m2, *msqr;
Mopr op;

{

  int i, j, ni, nr, nc, r, c;
  double sign, fact, *a;

  if (m1 == NULL || m2 == NULL || msqr == NULL) return(1);
  sign = (double)(1.0);
  fact = (double)(0.0);
  switch (op)
  {

    case EQU:
      break;
    case ADD:
      fact = (double)(1.0);
      break;
    case DIF:
      sign = (double)(-1.0);
      fact = (double)(1.0);
      break;

  }
  nr = MIN2(_nrow(m1),_nrow(msqr));
  nc = MIN2(_nrow(m1),_ncol(msqr));
  ni =_ncol(m1);
  a = (double *)memory_get(ni*sizeof(double));
  for (r = 0; r < nr; r++)
    for (c = 0; c < nc; c++)
      _data(msqr,r,c) *= fact;
  for (r = 0; r < nr; r++)
    for (c = 0; c < nc; c++)
    {

      for (i = 0;i < ni; i++)
        a[i] = (double) (0.0);
      for (j =0; j < ni; j++)
        for (i = 0;i < ni; i++)
          a[j] += _data(m1,r,i)*_data(m2,i,j);
      for (i = 0; i < ni; i++)
        _data(msqr,r,c) += sign * a[i] * _data(m1,c,i);

    }
  free(a);

  return(0);

}

/*********************************************************************/

/*F:matrix_invert*
_____________________________________________________________________

  matrix_invert
_____________________________________________________________________

Name:       matrix_invert - compute matrix inversion

Syntax:     | int matrix_invert(mx)
```

```
            | Matrix *mx;

Description:  'matrix_invert' performs the inversion of the 2x2 'mx's' upper left
              submatrix.
              The 'mx's' submatrix is erased with the output result.

Return value: | 0 => Ok.
              | 1 => Can't make the matrix inversion.

Example:      | Try yourself.

Restrictions: Only for matrices with determinant non zero.

Author:       Joao Tavares

Id:           @(#)matrix_invert.c   1.00   94/05/21
_____________________________________________________________________
*/

int matrix_invert(mx)
Matrix *mx;

{

  double b[4], det, aux;

  /* only for the 2x2 upper left submatrix */

  det = (_data(mx, 0, 0))*(_data(mx, 1, 1))-(_data(mx, 0, 1))*(_data(mx, 1, 0));
  if (det != 0.0) {

    aux = 1.0/det;
    b[0] = (_data(mx, 1, 1))*aux;
    b[1] = -(_data(mx, 0, 1))*aux;
    b[2] = -(_data(mx, 1, 0))*aux;
    b[3] = (_data(mx, 0, 0))*aux;
    (_data(mx, 0, 0)) = b[0];
    (_data(mx, 0, 1)) = b[1];
    (_data(mx, 1, 0)) = b[2];
    (_data(mx, 1, 1)) = b[3];

  }
  else exit(Error(1, "\nCan't make the matrix inversion.\n"));

  return(0);

}

/*********************************************************************/

/*F:matrix_eig*
_____________________________________________________________________

  matrix_eig
_____________________________________________________________________

Name:     matrix_eig - compute matrix eigenvalues and eigenvectors

Syntax:   | int matrix_eig(mx, vec, val)
          | Matrix *mx, *vec, *val;

Description:  'matrix_eig' performs the calculation of the eigenvalues and
              eigenvectors of the 2x2 'mx's' upper left submatrix.
              'vec' is the eigenvectors matrix. They are in column form.
              'val' is the eigenvalues matrix. They are in diagonal and grow
              less form.

Return value: | 0 => Ok.
              | 1 => Can't compute the matrix eigenvalues and eigenvectors.

Example:      | Try yourself.

Restrictions: Only for positive defined matrices.

Author:       Joao Tavares

Id:           @(#)matrix_eig.c   1.00   94/05/21
_____________________________________________________________________
*/

int matrix_eig(mx, vec, val)
Matrix *mx, *vec, *val;

{

  double b, c, aux;

  /* only for 2x2 upper left submatrix */

  b = -((_data(mx, 0, 0))+(_data(mx, 1, 1)));
```

```
c = (_data(mx, 0, 0))*(_data(mx, 1, 1))-(_data(mx, 0, 1))*(_data(mx, 1, 0));
aux = SQR(b)-4.0*c;
if (aux < 0.0 && aux > -0.001) aux = 0.0;
if (aux < 0.0) exit(Error(1, "\nCan't compute the matrix eigenvalues and eigenvectors.\n"));
else {

  aux = sqrt(aux);
  (_data(val, 0, 0)) = (-b+aux)/2.0;
  (_data(val, 1, 1)) = (-b-aux)/2.0;
  if ((_data(val, 1, 1)) > (_data(val, 0, 0))) {

    aux = (_data(val, 0, 0));
    (_data(val, 0, 0)) = (_data(val, 1, 1));
    (_data(val, 1, 1)) = aux;

  }
  aux = (_data(val, 0, 0))-(_data(mx, 1, 1))-(_data(mx, 0, 1));
  if (aux == 0.0) {

    (_data(vec, 0, 0)) = 0.0;
    (_data(vec, 1, 0)) = 1.0;

  }
  else {

    (_data(vec, 0, 0)) = 1.0;
    (_data(vec, 1, 0)) = (-(_data(val, 0, 0))+(_data(mx, 0, 0))+(_data(mx, 1, 0)))/aux;

  }
  aux = (_data(val, 1, 1))-(_data(mx, 1, 1))-(_data(mx, 0, 1));
  if (aux == 0.0) {

    (_data(vec, 0, 1)) = 0.0;
    (_data(vec, 1, 1)) = 1.0;

  }
  else {

    (_data(vec, 0, 1)) = 1.0;
    (_data(vec, 1, 1)) = (-(_data(val, 1, 1))+(_data(mx, 0, 0))+(_data(mx, 1, 0)))/aux;

  }

}
```

```
  return(0);

}

/*****************************************************************************/
```

3.2.12 - Ficheiro list.h

Este ficheiro contém as definições de variáveis e de protótipos relativamente aos módulos que pertencem ao grupo ***VII*** da implementação ***deep***.

```
/*
_____________________________________________________________________________

  list.h
  @(#)list.h   1.0 93/12/07, Copyright 1993, DEEC, UoP
  Image processing lab,
  Department of Electrical and Computer Engineering
  University of Porto
  E-mail: gpai@obelix.fe.up.pt
_____________________________________________________________________________

  File Name: list.h
  Purpose  : Definitions for handling list structures.
  Author   : Miguel Correia, DEEC, UoP
  Version  : 1.0
  Date     : 07/12/93
_____________________________________________________________________________
*/

/*****************************************************************************/

/* TYPES DEFINITIONS */

typedef enum {LINEAR, CIRCULAR} Ltype;      /* list's type */
typedef enum {BEFORE, AFTER} Lpos;          /* position in the list */

typedef struct _list
{

  struct _list *lnext;       /* pointer to the next list's element */
  char *pinfo;               /* pointer to the element's information field */
```

```
}Cell,*List;

/*************************************************************************/

/* MACROS DEFINITIONS */

#define _lnext(l)     ((l)->lnext)          /* next list's element */
#define _pinfo(l)     ((l)->pinfo)          /* element's information field */

/*************************************************************************/

/* FUNCTIONS PROTOTYPES */

List create_list();
void delete_list();
List insert_cell_list();
List delete_cell_list();

/*************************************************************************/
```

3.2.13 - Ficheiro list.c

Este ficheiro contém a listagem relativa aos módulos que pertencem ao grupo ***VII*** da implementação ***deep***.

```
/*
_____________________________________________________________________

  list.c
  @(#)list.c   1.0 93/12/07, Copyright 1993, DEEC, UoP
  Image processing lab,
  Department of Electrical and Computer Engineering
  University of Porto
  E-mail: gpai@obelix.fe.up.pt
_____________________________________________________________________

  File Name: list.c
  Purpose  : Source code of list handling routines.
  Author   : Miguel Correia, DEEC, UoP
  Version  : 1.0
  Date     : 07/12/93
_____________________________________________________________________

  These software routines were developed for the XITE system under the written
  permission of B-lab, Department of Informatics, University of Oslo.
_____________________________________________________________________
*/

/*************************************************************************/

/* INCLUDES */

#include <stdio.h>
#include <malloc.h>
#include "memory.h"
#include "list.h"

/*************************************************************************/

#ifndef lint
static char *SccsId = "@(#)list.c 1.0 93/12/07, DEEC, UoP";
#endif

/*************************************************************************/

/*F:create_list*
_____________________________________________________________________

  create_list
_____________________________________________________________________

Name:        create_list - create a list

Syntax:      | List create_list(lt)
             | Ltype lt;

Description:  'create_list' performs the creation of a list.
              'lt' is the list type: LINEAR or CIRCULAR.

Return value: | A pointer to the list's header.

Example:     | Try yourself.

Restrictions:  None.

Author:        Miguel Correia

Id:            @(#)create_list.c   1.00   93/12/07
```

```
_____________________________________________________________________

*/

List create_list(lt)
Ltype lt;    /* list type: LINEAR or CIRCULAR */

{

  List laux;

  laux = (List)memory_get(sizeof(Cell));
  _lnext(laux) = ((lt == LINEAR) ? NULL: laux);
  _pinfo(laux) = NULL;

  return(laux);

}

/*************************************************************************/

/*F:delete_list*
_____________________________________________________________________

  delete_list
_____________________________________________________________________

Name:        delete_list - delete a list

Syntax:     | void delete_list(l,func)
            | List l;
            | void (*func)();

Description:  'delete_list' performs the delete of a list.
              'l' is the list's pointer.
              'func' is an user function to remove the information field.

Return value:  None.

Example:       | Try yourself.

Restrictions:  None.

Author:        Miguel Correia

Id:            @(#)delete_list.c   1.00   93/12/07
_____________________________________________________________________

*/

void delete_list(l,func)
List l;
void (*func)();      /* user function to remove the information field */

{

 if(l == NULL) return;
 else
 {
   List laux;
   for(laux = _lnext(l);laux && laux != l; laux = _lnext(l))
   {

     _lnext(l) = _lnext(laux);
     (*func)(_pinfo(laux));
     free(laux);

   }

 }

}

/*************************************************************************/

/*F:insert_cell_list*
_____________________________________________________________________

  insert_cell_list
_____________________________________________________________________

Name:        insert_cell_list - insert an information field into a list

Syntax:     | List insert_cell_list(info,lpos,sw)
            | char *info;
            | List lpos;
            | Lpos sw;

Description:  'insert_cell_list' performs the insertion of an information field
              into a list.
```

```
                'info' is a pointer to the information field.
                'lpos' is the list's pointer.
                'sw' is the desired position of insert: AFTER or BEFORE.

Return value:  A list's pointer.

Example:       | Try yourself.

Restrictions:   None.

Author:        Miguel Correia

Id:            @(#)insert_cell_list.c   1.00   93/12/07
_____________________________________________________________________
*/

List insert_cell_list(info,lpos,sw)
char *info;
List lpos;
Lpos sw;

{

 List laux = NULL;

 if (lpos != NULL)
 {

  if (_pinfo(lpos) == NULL)       /* info field is empty */
  {

   laux = lpos;       /* insert at position */

  }
  else
  {

   laux = (List)memory_get(sizeof(Cell));
   if(sw == AFTER)         /* insert after position */
   {

    _lnext(laux) = _lnext(lpos);
    _lnext(lpos) = laux;

   }
   else         /* insert before position */
   {

    _lnext(laux) = lpos;

   }

  }
  _pinfo(laux) = info;

 }

 return(laux);

}

/*************************************************************************/

/*F:delete_cell_list*
_____________________________________________________________________

 delete_cell_list
_____________________________________________________________________

Name:       delete_cell_list - delete a list's cell

Syntax:     | List delete_cell_list(c,l,func)
            | Cell *c;
            | List l;
            | void (*func)();

Description:   'delete_cell_list' performs the delete of a list's cell.
               'c' is a pointer of the cell to remove.
               'l' is the list's pointer.
               'func' is an user function to remove the information field

Return value:  | A pointer to the list.

Example:       | Try yourself.

Restrictions:   None.

Author:        Miguel Correia

Id:            @(#)delete_cell_list.c   1.00   93/12/07
```

```
_____________________________________________________________________
*/

List delete_cell_list(c,l,func)
Cell *c;
List l;
void (*func)();

{

  if(c != NULL && l != NULL)
  {

    if (c == l) l = _lnext(l);    /* delete the first cell */
    else
    {

      List laux;

      for (laux = l; _lnext(laux) != c && _lnext(laux); laux = _lnext(laux));
      _lnext(laux) = _lnext(c);

    }
    (*func)(_pinfo(c));
    free(c);
    return(l);

  }

  return(0);

}

/*********************************************************************/
```

3.2.14 - Ficheiro memory.h

Este ficheiro contém as definições de variáveis e de protótipos relativamente aos módulos que pertencem ao grupo ***VIII*** da implementação ***deep***.

```
/*
_____________________________________________________________________
  memory.h
  @(#)memory.h   1.0 94/06/15, Copyright 1994, DEEC, UoP
  Image processing lab,
  Department of Electrical and Computer Engineering
  University of Porto
  E-mail: gpai@obelix.fe.up.pt
_____________________________________________________________________

  File Name: memory.h
  Purpose  : Definitions for handling memory resources.
  Author   : Joao Tavares, DEEC, UoP
  Version  : 1.0
  Date     : 15/06/94
_____________________________________________________________________
*/

/*********************************************************************/

/* DEFINES */

#define NR_END 1
#define FREE_ARG char *

/*********************************************************************/

/* FUNCTIONS PROTOTYPES */

char *memory_get();
double *vector();
double **matrix();
void free_vector();
void free_matrix();

/*********************************************************************/
```

3.2.15 - Ficheiro memory.c

Este ficheiro contém a listagem relativa aos módulos que pertencem ao grupo ***VIII*** da implementação ***deep***.

```
/*
_____________________________________________________________________
```

```
memory.c
@(#)memory.c   1.0 94/06/15, Copyright 1994, DEEC, UoP
Image processing lab,
Department of Electrical and Computer Engineering
University of Porto
E-mail: gpai@obelix.fe.up.pt
_____________________________________________________________________

message.h
@(#)message.h   1.3 92/01/15, Copyright 1990, Blab, UiO
Image processing lab, Department of Informatics
University of Oslo
E-mail: blab@ifi.uio.no
_____________________________________________________________________

These software routines were developed for the XITE system under the written
permission of B-lab, Department of Informatics, University of Oslo.
_____________________________________________________________________

 Permission to use, copy, modify and distribute this software and its documentation for any purpose
and without fee is hereby granted, provided that this copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in supporting documentation and that the
name of B-lab, Department of Informatics or University of Oslo not be used in advertising or publicity
pertaining to distribution of the software without specific, written prior permission.

 B-LAB DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING
ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT
SHALL B-LAB BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL
DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA
OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER
TORTIOUS ACTION, ARISING OUT OF OR IN  CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
*/
/*****************************************************************/

/* INCLUDES */

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <errno.h>
#include <blab/message.h>
#include "memory.h"

/*****************************************************************/

/* EXTERN VARIABLE */

extern char *progname; /* progname's name */

/*****************************************************************/

#ifndef lint
static char *SccsId = "@(#)memory.c 1.0 96/06/15, DEEC, UoP";
#endif

/*****************************************************************/

/*F:memory_get*
_____________________________________________________________________

 memory_get
_____________________________________________________________________

Name:        memory_get - allocation of memory

Syntax:     | char *memory_get(size)
            | int size;

Description:   'memory_get' performs the allocation of memory.
               'size' is the desire size.

Return value:  | A pointer.

Example:       | Try yourself.

Restrictions:  Enough memory.

Author:         Joao Tavares

Id:            @(#)memory_get.c   1.00   94/06/15
_____________________________________________________________________
*/

char *memory_get(size)
int size;

{

 char * paux;
```

```
  if((paux = (char *)malloc(size)) == NULL)
  {

    perror(progname);
    exit(errno);

  }

  return(paux);

}

/*****************************************************************************/

/*F:vector*
______________________________________________________________________________

  vector
______________________________________________________________________________

Name:        vector - allocation of memory to a vector

Syntax:     | double *vector(nl, nh)
            | int nl, nh;

Description:   'vector' performs the allocation of memory to a vector.
               'nl' and 'nh' are the vector's first and last index.

Return value:  | A pointer to the vector.
               | 1 => Allocation failure in vector().

Example:       | Try yourself.

Restrictions:   Enough memory.

Author:         Joao Tavares

Id:            @(#)vector.c   1.00   94/06/15
______________________________________________________________________________
*/

double *vector(nl, nh)
int nl, nh;
{

  double *v;

  v = (double *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(double)));
  if (!v) exit(Error(1, "\nAllocation failure in vector().\n"));

  return v-nl+NR_END;

}

/*****************************************************************************/

/*F:matrix*
______________________________________________________________________________

  matrix
______________________________________________________________________________

Name:        matrix - allocation of memory to a matrix

Syntax:     | double **matrix(nrl, nrh, ncl, nch)
            | int nrl, nrh, ncl, nch;

Description:   'matrix' performs the allocation of memory to a matrix.
               'nrl' and 'nrh' are the first and last index to the matrix's rows.
               'ncl' and 'nch' are the first and last index to the matrix's columns.

Return value:  | A pointer to the matrix.
               | 1 => Allocation failure 1 in matrix().
               | 2 => Allocation failure 2 in matrix().

Example:       | Try yourself.

Restrictions:   Enough memory.

Author:         Joao Tavares

Id:            @(#)matrix.c   1.00   94/06/15
______________________________________________________________________________
*/

double **matrix(nrl, nrh, ncl, nch)
int nrl, nrh, ncl, nch;
```

```
{

  int i, nrow = nrh-nrl+1, ncol = nch-ncl+1;
  double **m;

  m = (double **) malloc ((size_t)((nrow+NR_END)*sizeof(double *)));
  if (!m) exit(Error(1, "\nAllocation failure in matrix().\n"));

  m += NR_END;
  m -= nrl;
  m[nrl] = (double *) malloc ((size_t)((nrow*ncol+NR_END)*sizeof(double)));
  if (!m[nrl]) exit(Error(2, "\nAllocation failure in matrix().\n"));
  m[nrl] += NR_END;
  m[nrl] -= ncl;

  for (i = nrl+1; i <= nrh; i++) m[i] = m[i-1]+ncol;

  return m;

}

/*****************************************************************************/


/*F:free_vector*
_____________________________________________________________________________

  free_vector
_____________________________________________________________________________

Name:       free_vector - free the memory reserved for a vector

Syntax:     | void free_vector(v, nl, nh)
            | double *v;
            | int nl, nh;;

Description:  'free_vector' performs the desalocation of the memory reseved to a
              vector.
              'v' is the vector's pointer.
              'nl' and 'nh' are the vector's first and last index.

Return value: | None.

Example:      | Try yourself.

Restrictions:  None.

Author:        Joao Tavares

Id:            @(#)free_vector.c   1.00   94/06/15
_____________________________________________________________________________
*/

void free_vector(v, nl, nh)
double *v;
int nl, nh;

{

  free((FREE_ARG)(v+nl-NR_END));

}

/*****************************************************************************/

/*F:free_matrix*
_____________________________________________________________________________

  free_matrix
_____________________________________________________________________________

Name:       free_matrix - free the memory reserved to a matrix

Syntax:     | void free_matrix(m, nrl, nrh, ncl, nch)
            | double **m;
            | int nrl, nrh, ncl, nch;

Description:  'free_matrix' performs the desallocation of the memory reserved to
              a matrix.
              'm' is the matrix's pointer.
              'nrl' and 'nrh' are the first and last index to the matrix's rows.
              'ncl' and 'nch' are the first and last index to the matrix's columns.

Return value: | None.

Example:      | Try yourself.

Restrictions:  None.

Author:        Joao Tavares
```

```
Id:             @(#)free_matrix.c   1.00   94/06/15
_____________________________________________________________________
*/

void free_matrix(m, nrl, nrh, ncl, nch)
double **m;
int nrl, nrh, ncl, nch;

{

  free((FREE_ARG)(m[nrl]+ncl-NR_END));
  free((FREE_ARG)(m+nrl-NR_END));

}

/*********************************************************************/
```

3.2.16 - Ficheiro gdefs.h

Este ficheiro contém definições de macros e de algumas variáveis de uso global.

```
/*
_____________________________________________________________________

  gdefs.h
  @(#)gdefs.h   1.0 94/05/19, Copyright 1993, DEEC, UoP
  Image processing lab,
  Department of Electrical and Computer Engineering
  University of Porto
  E-mail: gpai@obelix.fe.up.pt
_____________________________________________________________________

  File Name: gdefs.h
  Purpose  : General Definitions for all programs.
  Author   : Joao Tavares, DEEC, UoP
  Version  : 1.0
  Date     : 19/05/94
_____________________________________________________________________
*/

/*********************************************************************/

/* GLOBAL CONSTANTS DEFINES */

#define FALSE 0                 /* false's value */
#define TRUE  1                 /* true's value */
#define BLACK 0                 /* black's value */
#define WHITE 255               /* white's value */
#define MPI 3.141592654         /* pi's value */
#define API 1.570796327         /* alf pi's value */
#define TOL 1.0e-5              /* math's tolerance */

/*********************************************************************/

/* GLOBAL VARIABLES */

/* error messages */

static char *errors[] = {

  " Ok!\n",
  " \n",
  " Bad pixel type!\n",
  " Band size error!\n",
  " Not enough memory!\n",

};

/*********************************************************************/

/* GLOBAL MACROS DEFINES */

/* maximum of two */
#define MAX2(a,b) ((a)<(b) ? (b) : (a))

/* minimum of two */
#define MIN2(a,b) ((a)<(b) ? (a) : (b))

/* abs */
#define ABS(a) ((a)>0.0 ? (a) : (-(a)))

/* sign */
#define SIGN(a) ((a)>0.0 ? 1 : ((a)<0.0 ? (-1) : 0))

/* square */
#define SQR(a) ((a)*(a))
```

```
/* degree to rad */
#define ANG_DEG_RAD(a) ((a)*MPI/180.0)

/* rad to degree */
#define ANG_RAD_DEG(a) ((a)*180.0/MPI)

/* a if b>=0 or -a if b<0 */
#define SIGN2(a, b) ((b)>=0.0 ? fabs(a) : -fabs(a))

/* square for double */
static double sqrarg;
#define DSQR(a) ((sqrarg=(a)) == 0.0 ? 0.0 : sqrarg*sqrarg)

/* maximum of two doubles */
static double maxarg1, maxarg2;
#define DMAX(a, b) (maxarg1=(a), maxarg2=(b), (maxarg1) > (maxarg2) ?\
(maxarg1) : (maxarg2))

/* minimum of two integers */
static int iminarg1, iminarg2;
#define IMIN(a, b) (iminarg1=(a), iminarg2=(b), (iminarg1) < (iminarg2) ?\
(iminarg1) : (iminarg2))

/* pythag's function */
static double absa, absb;
#define PYTHAG(a, b) (absa=fabs(a), absb=fabs(b), absa > absb ?\
(absa*sqrt(1.0+DSQR(absb/absa))) : (absb==0.0 ? 0.0 : (absb*sqrt(1.0+\
DSQR(absa/absb)))))

/*************************************************************************/
```

4 - Obtenção de coordenadas 3D de pontos: compdeep

Nesta secção é apresentada uma implementação para obtenção de coordenadas 3D de pontos cujas coordenadas 2D em duas imagens obtidas por uma mesma câmara, para duas posições e orientações distintas, são especificadas pelo utilizador: ***compdeep***. A técnica para a obtenção de coordenadas 3D utilizada nesta implementação é baseada no método da triangulação estereoscópica[14].

4.1 - Descrição dos diferentes módulos

Esta implementação é constituída pelos módulos referidos na *Tab. II*.

Grupo I	*Grupo II*	*Grupo III*	*Grupo IV*
principal	***compute_mpp***	***svdfit***	***vector***
	frame_undistorted	***svbksb***	***matrix***
	stereo_eq	***svdcmp***	***free_vector***
			free_matrix

*Tab. II - Grupos que constituem a implementação **compdeep**.*

Cada um dos grupos definidos é constituído por módulos que são comuns a uma dado contexto. Assim, o grupo ***I*** contém o módulo mais particular a esta implementação, o grupo ***II*** os módulos relacionados com funções de visão tridimensional, o grupo ***III*** os módulos para resolução de sistemas lineares sobredeterminados e o grupo ***IV*** os módulos para gestão de memória, *Fig. 4*.

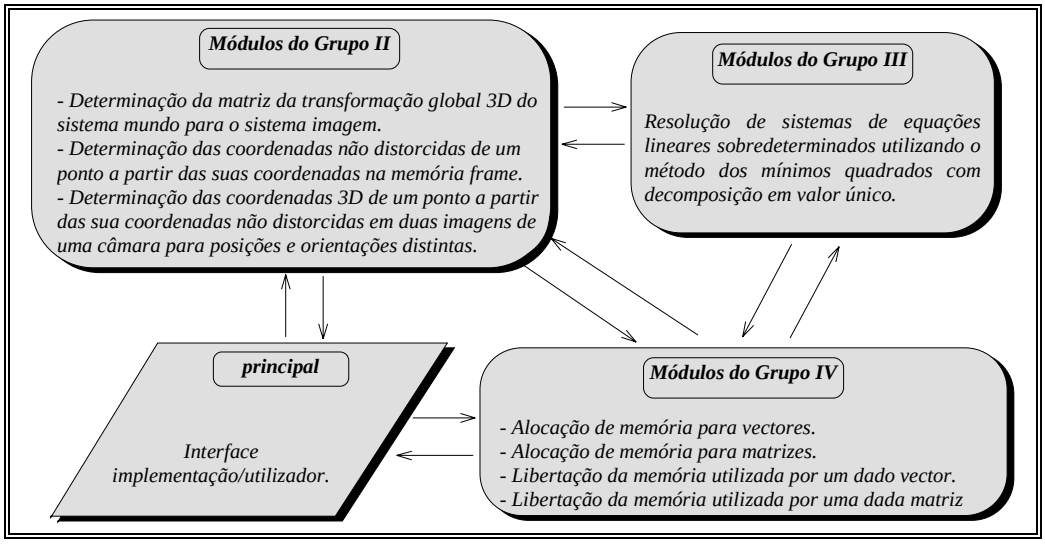


*Fig. 4 - Módulos integrantes da implementação **compdeep** e suas relações.*

Apresenta-se, de seguida, apenas a descrição do módulo ***principal***, pois todos os restantes foram já apresentados na secção anterior.

4.1.1 - Módulo principal

Este módulo é responsável pela interface implementação/utilizador. A seu cargo está:

- ✓ Leitura dos argumentos opcionais ("*switches*"):

[14] Ver, por exemplo [Tavares, 1995].

- *sx*, ("[-s <sx>]"), factor de incerteza horizontal s_x. Por defeito, é igual a *0.710935*.
- *dx*, ("[-dx <dx>]"), distância entre centros dos elementos sensores vizinhos na direcção *x*, d_x. Por defeito, é igual a *8.37765957e-3* mm.
- *dy*, ("[-dy <dy>]"), distância entre centros dos sensores CCD vizinhos na direcção *y*, d_y. Por defeito, é igual a *8.07560136e-3* mm.
- *xCenterBuf*, ("[-cx <xCenterBuf>]"), coordenada do centro da imagem na memória *frame* segundo a direcção *x*, C_x. Por defeito, é igual a *256*.
- *yCenterBuf*, ("[-cy <yCenterBuf>]"), coordenada do centro da imagem na memória *frame* segundo a direcção *y*, C_y. Por defeito, é igual a *256*.

✓ Leitura dos argumentos passados à implementação pela linha de comando:

- *focalDist*, ("<focalDist>"), distância focal efectiva *f*.
- *radialLensDist*, ("<radialLensDist>"), factor de distorção radial da lente k_1.

✓ Interrogação ao utilizador dos valores (em grau) das três rotações 3D e dos valores das três translações 3D, segundo os três eixos principais da transformação geométrica do sistema de coordenadas mundo para o sistema câmara, para a primeira posição e orientação da câmara a considerar.

✓ Interrogação ao utilizador dos valores (em grau) das três rotações 3D e dos valores das três translações 3D, segundo os três eixos principais da transformação geométrica do sistema de coordenadas mundo para o sistema câmara, para a segunda posição e orientação da câmara a considerar.

✓ Determinação da transformação geométrica do sistema de coordenadas mundo para o sistema câmara para a primeira posição e orientação da câmara a considerar, utilizando para o efeito o módulo ***compute_mpp***.

✓ Apresentação da matriz da transformação geométrica do sistema de coordenadas mundo para o sistema câmara obtida para a primeira posição e orientação da câmara a considerar.

✓ Determinação da transformação geométrica do sistema de coordenadas mundo para o sistema câmara para a segunda posição e orientação da câmara a considerar, utilizando para o efeito o módulo ***compute_mpp***.

✓ Apresentação da matriz da transformação geométrica do sistema de coordenadas mundo para o sistema câmara obtida para a segunda posição e orientação da câmara a considerar.

✓ Interrogação ao utilizador das coordenadas na memória *frame* para o ponto, segundo as direcções *x* e *y*, para a primeira posição e orientação da câmara a considerar.

✓ Determinação das coordenadas imagem não distorcida para o ponto, segundo as direcções *x* e *y*, para a primeira posição e orientação da câmara a considerar. Utiliza para o efeito o módulo ***frame_undistorted***.

✓ Interrogação ao utilizador das coordenadas na memória *frame* para o ponto, segundo as direcções *x* e *y*, para a segunda posição e orientação da câmara a considerar.

✓ Determinação das coordenadas imagem não distorcida para o ponto, segundo as direcções *x* e *y*, para a segunda posição e orientação da câmara a considerar. Utiliza para o efeito o módulo ***frame_undistorted***.

✓ Apresentação das coordenadas imagem não distorcida para o ponto, segundo as direcções *x* e *y*, nos planos imagem da câmara, para a primeira e a segunda posição e orientação a considerar.

- ✓ Determinação das coordenadas 3D para o ponto, utilizando para o efeito o módulo ***stereo_eq***.
- ✓ Apresentação das coordenadas 3D para o ponto, obtidas pela implementação.
- ✓ Comunicação da existência de erros em argumentos de entrada ou ao longo do decurso da execução:
  - Através de valores de retorno:
    - 1, ("Wrong value for dx <dx>; it must be greater than zero."), o valor especificado para a distância entre centros dos elementos sensores vizinhos na direcção *x* deve ser maior do que zero.
    - 2, ("Wrong value for dy <dy>; it must be greater than zero."), o valor especificado para a distância entre centros dos sensores CCD vizinhos na direcção *y* deve ser maior do que zero.
    - 3, ("Wrong value for f; it must be different from zero."), o valor especificado para a distância focal efectiva deve ser diferente de zero.
    - 4, ("Wrong value for sx; it must be different from zero."), o valor especificado para o factor de incerteza horizontal deve ser diferente de zero.
  - Através de mensagens:
    - 1, (*InitMessage*), quando o número de argumentos é igual a um, isto é, apenas o nome da implementação.
    - 2, ("Bad number of arguments."), quando o número de argumentos é diferente de três.

Até o utilizador especificar o valor *9999* para a coordenada na memória *frame* segundo a direcção *x* para uma dado ponto e para a primeira posição e orientação da câmara a considerar, esta implementação executa ciclicamente as fases:

I) Interrogação ao utilizador das coordenadas na memória *frame* para o ponto, segundo as direcções *x* e *y*, para a primeira posição e orientação da câmara a considerar.

II) Determinação das coordenadas imagem não distorcidas para o ponto, segundo as direcções *x* e *y*, para a primeira posição e orientação da câmara a considerar.

III) Interrogação ao utilizador das coordenadas na memória *frame* para o ponto, segundo as direcções *x* e *y*, para a segunda posição e orientação da câmara a considerar.

IV) Determinação das coordenadas imagem não distorcidas para o ponto, segundo as direcções *x* e *y*, para a segunda posição e orientação da câmara a considerar.

V) Apresentação das coordenadas imagem não distorcidas para o ponto, segundo as direcções *x* e *y*, nos planos imagem da câmara para a primeira e segunda posição e orientação a considerar.

VI) Determinação das coordenadas 3D para o ponto.

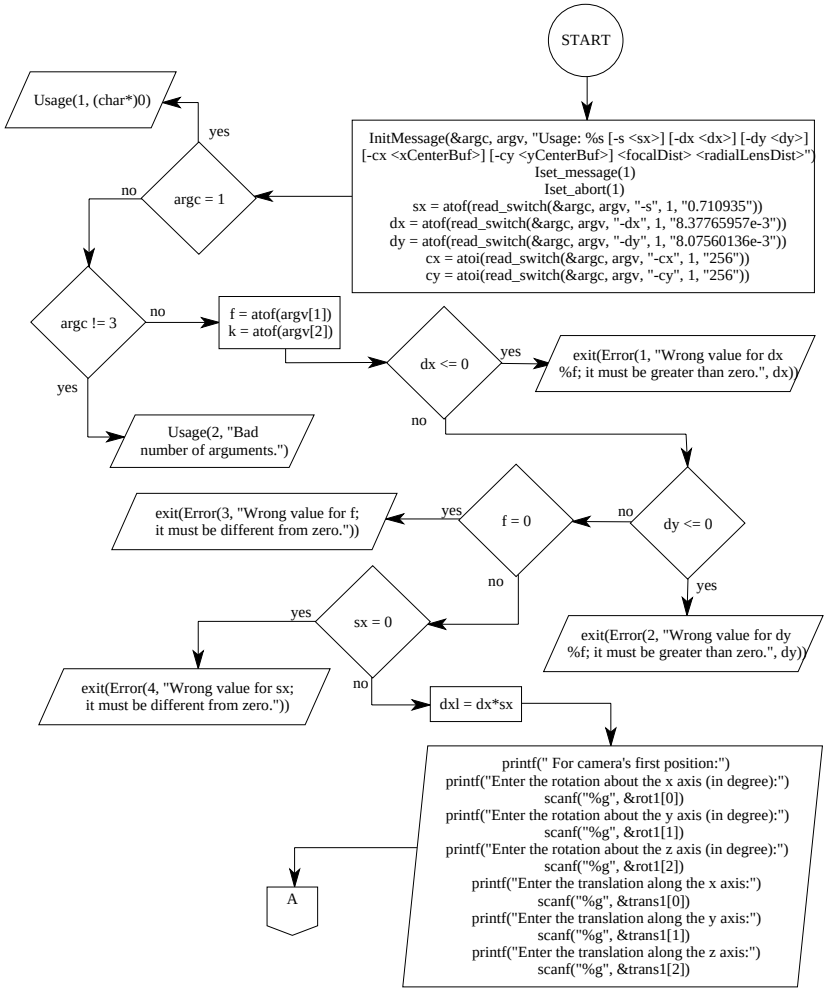
VII) Apresentação das coordenadas 3D para o ponto, obtidas pela implementação.

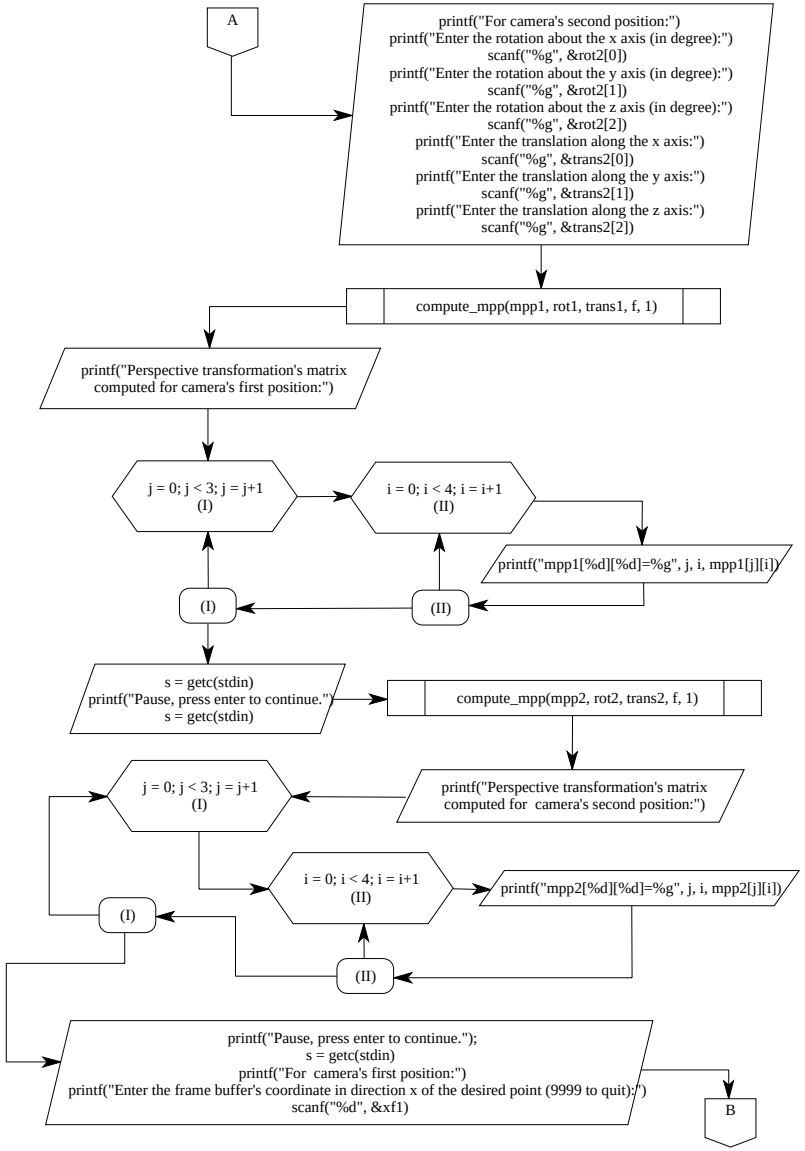
Como restrições quanto à utilização deste módulo têm-se:

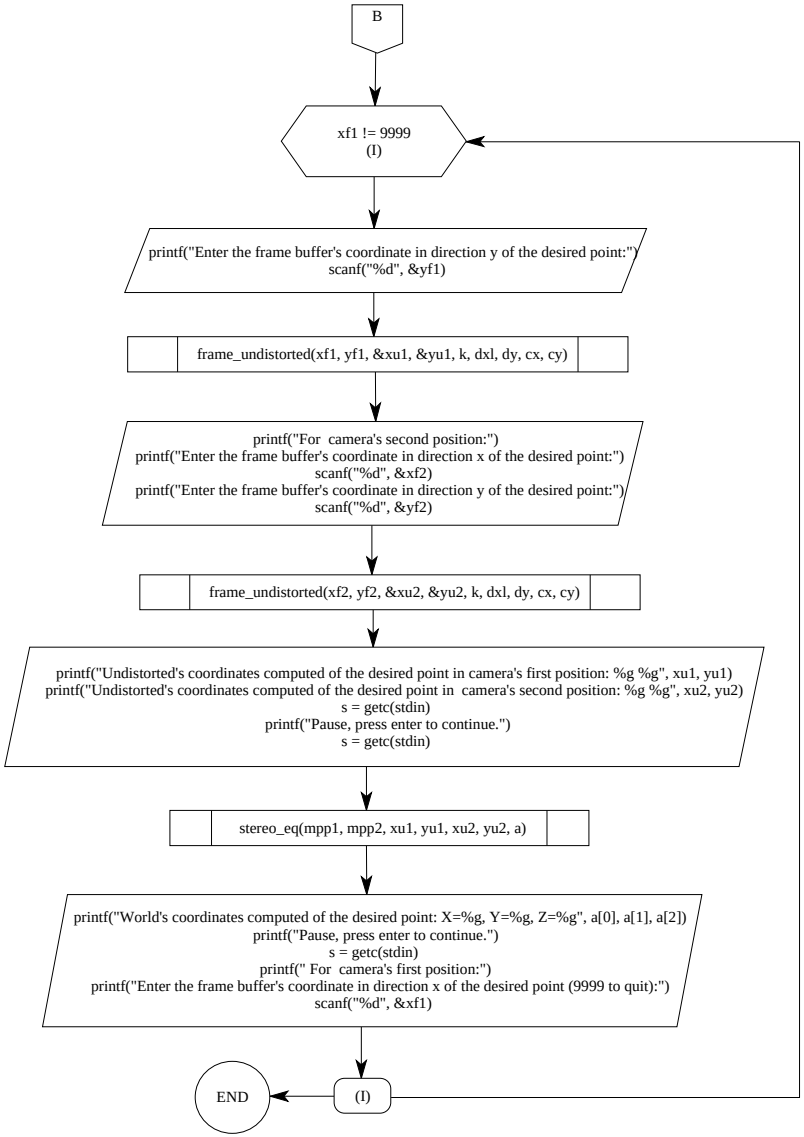
- ⌧ Os valores especificados para a distância entre centros dos elementos sensores vizinhos na direcção *x* e para a distância entre centros dos sensores CCD vizinhos na direcção *y*, devem ser maiores do que zero.
- ⌧ Os valores especificados para a distância focal efectiva e para o factor de incerteza horizontal, devem ser diferentes de zero.

4.2 - Fluxogramas dos diferentes módulos

Apresenta-se, de seguida, apenas o fluxograma do módulo ***principal***, já que os fluxogramas dos restantes módulos da implementação ***compdeep*** já foram apresentados na secção anterior.







4.3 - Listagem da implementação

Apresenta-se, de seguida, a listagem da implementação ***compdeep*** desenvolvida em linguagem *C* em ambiente *UNIX* numa estação de trabalho *SunSparc* e que se destina ao ambiente de processamento de imagem *XITE*[15], do qual utiliza algumas funções e tipos de variáveis já previamente definidas. Esta implementação contém comentários para auxílio à sua compreensão, assim como informação sobre os diferentes módulos, para sua posterior extracção através da ferramenta *CDOC(1)* existente no *XITE*.

Com o objectivo de tornar esta implementação mais simples, facilmente entendível e o mais reutilizável possível, optou-se por dividi-la em vários ficheiros. Assim, cada grupo de módulos que constitui a implementação constitui um ficheiro. Além destes ficheiros, existem os ficheiros associados, que contêm os protótipos dos módulos. Este ficheiros são utilizados para incluir os módulos num qualquer ficheiro que os utilize. Existe também um ficheiro para definição de macros e de algumas variáveis de uso global, ***gdefs.h***. Deste modo, esta implementação é constituída pelos seguintes ficheiros: ***compdeep.c***, ***func_3d.h***, ***func_3d.c***, ***linleasq.h***, ***linleasq.c***, ***memory.h***, ***memoy.c*** e ***gdefs.h***.

Seguidamente apenas é apresentado o ficheiro ***compdeep.c***, pois todos os restantes foram apresentados na secção anterior. Contudo, alguns destes ficheiros incluem módulos não utilizados nesta implementação.

4.3.1 - Ficheiro compdeep.c

Este ficheiro contém a listagem relativa ao módulo que constitui o grupo ***I*** da implementação ***compdeep***.

```
/*
_____________________________________________________________________________


 compdeep.c
 @(#)compdeep.c   1.0 94/07/25, Copyright 1994, DEEC, UoP
 Image processing lab, Department of Electrical and Computer
 Engineering
 University of Porto
 E-mail: gpai@obelix.fe.up.pt
```

[15] *X-based Image processing Tools an Environmemt*, [Lønnestad, 1992].

---

readarg.h
@(#)readarg.h 1.3 92/01/15, Copyright 1990, Blab, UiO
Image processing lab, Department of Informatics
University of Oslo
E-mail: blab@ifi.uio.no

---

message.h
@(#)messahe.h 1.3 92/01/15, Copyright 1990, Blab, UiO
Image processing lab, Department of Informatics
University of Oslo
E-mail: blab@ifi.uio.no

/*

/**/

/* INCLUDES */

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <blab/message.h>
#include <blab/readarg.h>
#include "memory.h"
#include "gdefs.h"
#include "func_3d.h"
```

/**/

```
static char *SccsId = "@(#)compdeep.c 1.0 94/07/25, DEEC, UoP";
char progname[]="compdeep";
```

/**/

/*P:compdeep*

---

compdeep

Name: compdeep - computing the world coordinates of points from the buffer's coordinates

Syntax: | compdeep [-s <sx>] [-dx <dx>] [-dy <dy>] [-cx <xCenterBuf>] | [-cy <yCenterBuf>] <focalDist> <radialLensDist>

Description: 'deep' performs the computing of the world coordinates of points from the buffer's coordinates.
The camera's model is the TSAI's model.
For the computing of the world's coordinates is used the correspondent solution of the stereo projection's equations by the linear least-squares regression using SVD decomposition.
The horizontal uncertainty factor is indicate trough flag '-s'. By default 'sx' is 0.710935.
The center to center distance between adjacent sensor elements in x (scanline) direction is indicate trough flag '-dx'. By default 'dx' is 8.37765957e-3 mm.
The center to center distance between adjacent CCD sensor in the y direction is indicate trough flag '-dy'. By default 'dy' is 8.07560136e-3 mm.
The center of frame buffer in directions x and y are indicate through flags '-cx' and '-cy'. By default 'xcenterbuf' is 256 and 'ycenterbuf' is 256.
'focalDist' is the effective focal length.
'radialLensDist' is the lens's radial distortion.
To quit the user must enter 9999 for the point's frame buffer coordinate x for camera's first position.

Return value: | 1 => Wrong value for dx <dx>; it must be greater than zero.
| 2 => Wrong value for dy <dy>; it must be greater than zero.
| 3 => Wrong value for f; it must be different from zero.
| 4 => Wrong value for sx; it must be different from zero.

```
Examples:     | Try yourself.

Restrictions:  The values for f and sx must be different from zero.
               The values for dx and dy must be greater than zero.

Author:        Joao Tavares

Id:            @(#)deep.c 1.0   94/07/25
_____________________________________________________________________
*/

void main(argc, argv)
int argc;
char **argv;

{

  register i, j;
  double mpp1[3][4], mpp2[3][4], a[3], xu1, xu2, yu1, yu2, dxl, f, dx, sx, dy, k;
  float rot1[3], trans1[3], rot2[3], trans2[3];
  char s;
  int xf1, yf1, xf2, yf2, cx, cy;

  InitMessage(&argc, argv, "Usage: %s\n [-s <sx>] [-dx <dx>] [-dy <dy>] [-cx <xCenterBuf>] [-cy
<yCenterBuf>] <focalDist> <radialLensDist>\n");
  Iset_message(1);
  Iset_abort(1);

  /* read switches */

  sx = atof(read_switch(&argc, argv, "-s", 1, "0.710935"));
  dx = atof(read_switch(&argc, argv, "-dx", 1, "8.37765957e-3"));
  dy = atof(read_switch(&argc, argv, "-dy", 1, "8.07560136e-3"));
  cx = atoi(read_switch(&argc, argv, "-cx", 1, "256"));
  cy = atoi(read_switch(&argc, argv, "-cy", 1, "256"));

  if (argc == 1) Usage(1, (char*)0);
  if (argc != 3) Usage(2, "\nBad number of arguments.\n");

  /* read argv[] */

  f = atof(argv[1]);
  k = atof(argv[2]);

  if (dx <= 0.0) exit(Error(1, "\nWrong value for dx %f; it must be greater than zero.\n", dx));
  if (dy <= 0.0) exit(Error(2, "\nWrong value for dy %f; it must be greater than zero.\n", dy));
  if (f == 0.0) exit(Error(3, "\nWrong value for f; it must be different from zero.\n"));
  if (sx == 0.0) exit(Error(4, "\nWrong value for sx; it must be different from zero.\n"));

  dxl = dx*sx;

  /* input the rotations and translations for camera's first position */

  printf("\n\n For camera's first position:\n");

  printf("\n\t Enter the rotation about the x axis (in degree):");
  scanf("%g", &rot1[0]);
  printf("\n\t Enter the rotation about the y axis (in degree):");
  scanf("%g", &rot1[1]);
  printf("\n\t Enter the rotation about the z axis (in degree):");
  scanf("%g", &rot1[2]);

  printf("\n\t Enter the translation along the x axis:");
  scanf("%g", &trans1[0]);
  printf("\n\t Enter the translation along the y axis:");
  scanf("%g", &trans1[1]);
  printf("\n\t Enter the translation along the z axis:");
  scanf("%g", &trans1[2]);

  /* input the rotations and translations for camera's second position */

  printf("\n\n For camera's second position:\n");

  printf("\n\t Enter the rotation about the x axis (in degree):");
  scanf("%g", &rot2[0]);
  printf("\n\t Enter the rotation about the y axis (in degree):");
  scanf("%g", &rot2[1]);
  printf("\n\t Enter the rotation about the z axis (in degree):");
  scanf("%g", &rot2[2]);

  printf("\n\t Enter the translation along the x axis:");
  scanf("%g", &trans2[0]);
  printf("\n\t Enter the translation along the y axis:");
  scanf("%g", &trans2[1]);
  printf("\n\t Enter the translation along the z axis:");
  scanf("%g", &trans2[2]);

  /* compute the transformation perspective's matrix for camera's first position */

  compute_mpp(mpp1, rot1, trans1, f, 1);
```

```
printf("\n\n Perspective transformation's matrix computed for camera's first position:\n\n");
for (j = 0; j < 3; ++j) {

  for (i = 0; i < 4; ++i) printf("  mpp1[%d][%d]=%g", j, i, mpp1[j][i]);
  printf("\n");

}

s = getc(stdin);
printf("\nPause, press enter to continue.\n");
s = getc(stdin);

/* compute the transformation perspective's matrix for camera's second position */

compute_mpp(mpp2, rot2, trans2, f, 1);

printf("\n\n Perspective transformation's matrix computed for camera's second position:\n\n");
for (j = 0; j < 3; ++j) {

  for (i = 0; i < 4; ++i) printf("  mpp2[%d][%d]=%g", j, i, mpp2[j][i]);
  printf("\n");

}

printf("\nPause, press enter to continue.\n");
s = getc(stdin);

/* input the frame buffer's coordinates of the desired point in camera's first position */

printf("\n For camera's first position:\n");

printf("\n\t Enter the frame buffer's coordinate in direction x of the desired point (9999 to quit):");
scanf("%d", &xf1);

while (xf1 != 9999) {

  printf("\n\t Enter the frame buffer's coordinate in direction y of the desired point:");
  scanf("%d", &yf1);

  /* compute the undistorted's coordinates of the desired point in camera's first position */

  frame_undistorted(xf1, yf1, &xu1, &yu1, k, dxl, dy, cx, cy);

  /* input the frame buffer's coordinates of the desired point in camera's second position */

    printf("\n For camera's second position:\n");

    printf("\n\t Enter the frame buffer's coordinate in direction x of the desired point:");
    scanf("%d", &xf2);
    printf("\n\t Enter the frame buffer's coordinate in direction y of the desired point:");
    scanf("%d", &yf2);

    /* compute the undistorted's coordinates of the desired point in camera's second position */

    frame_undistorted(xf2, yf2, &xu2, &yu2, k, dxl, dy, cx, cy);

    printf("\n\nUndistorted's coordinates computed of the desired point in camera's first position: %g
%g\n", xu1, yu1);
    printf("\n\nUndistorted's coordinates computed of the desired point in camera's second position:
%g %g\n", xu2, yu2);

    s = getc(stdin);
    printf("\nPause, press enter to continue.\n");
    s = getc(stdin);

    /* compute the world's coordinates of the desired point */

    stereo_eq(mpp1, mpp2, xu1, yu1, xu2, yu2, a);

    printf("\nWorld's coordinates computed of the desired point:\tX=%g, Y=%g, Z=%g\n", a[0],
a[1], a[2]);
    printf("\nPause, press enter to continue.\n");
    s = getc(stdin);

    /* input the frame buffer's coordinates of the desired point in camera's first position */

    printf("\n For camera's first position:\n");
    printf("\n\t Enter the frame buffer's coordinate in direction x of the desired point (9999 to
quit):");
    scanf("%d", &xf1);

  }

}

/*************************************************************************/
```

Bibliografia

[Chapra, 1988] - *Steven C. Chapra, Raymond P. Canale*
Numerical Methods for Engineers
MCGRAW-HILL - 1988

[Foley, 1991] - *Foley, vanDam, Feiner, Hughes*
Computer Graphics Principles and Practice
ADDISON WESLEY 12110 - 1991 SECOND EDITION

[Hall, 1993] - *Ernest L. Hall*
Fundamental Principles of Robot Vision
SPIE VOL. 2056 INTELLIGENT ROBOTS AND COMPUTER VISION XII (1993) - 321/333

[Lenz, 1988] - *Reimar K. Lenz and Roger Y. Tsai*
Techniques for Calibration of the Scale Factor and Image Center for High Accuracy 3-D Machine Vision Metrology
IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 10, NO. 5, SEPTEMBER 1988 - 713/720

[Lønnestad, 1992] - *Tor Lønnestad, Otto Milvang*
XITE User's Manual
October 21, 1992

[Maybeck, 1979] - *Peter S. Maybeck*
Stochastic Models, Estimation, and Control
Volume I
Mathematics In Science and Engineering - Volume 141
ACADEMIC PRESS -1979

[Mendonça, 1990] - *Ana Maria Mendonça, Jorge Alves da Silva*
Comunicação Interna
FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO - 1990

[Press, 1992] - *William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery*
Numerical Recipes - The Art of Scientific Computing
CAMBRIDGE UNIVERSITY PRESS - 1992 SECOND EDITION

[Tavares, 1995] - *João Manuel Ribeiro da Silva Tavares*
Obtenção de Estrutura Tridimensional a Partir de Movimento de Câmara
Dissertação submetida ao Departamento de Engenharia Electrotécnica e de Computadores em 1995, para satisfação parcial dos requisitos do Mestrado em Engenharia Electrotécnica e de Computadores.
Orientador: *A. Jorge Padilha*
FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO - 1995

[Tsai, 1987] - *Roger Y. Tsai*
A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses
IEEE JOURNAL OF ROBOTICS AND AUTOMATION, VOL. RA-3, NO. 4, AUGUST 1987 - 323/344

[Schalkoff, 1989] - *Robert J. Schalkoff*
Digital Image Processing and Computer Vision
JOHN WILEY & SONS, INC. - 1989

[Schildt, 1991] - *Herbert Schildt*
C Completo e Total
OSBORNE MCGRAW-HILL - 1991