- Main Page
- Data Structures
- Files

- File List
- Globals

ar.h File Reference

ARToolKit subroutines. More...

```
#include <stdio.h>
#include <malloc.h>
#include <AR/config.h>
#include <AR/param.h>
```

Data Structures

struct	**ARMarkerInfo** *main structure for detected marker.* More...
struct	**ARMarkerInfo2** *internal structure use for marker detection.* More...
struct	**arPrevInfo** *structure for temporal continuity of tracking* More...

Defines

#define	**arMalloc**(V, T, S) *allocation macro function*

Typedefs

typedef char	**ARInt8**
typedef short	**ARInt16**
typedef int	**ARInt32**
typedef unsigned char	**ARUint8**
typedef unsigned short	**ARUint16**
typedef unsigned int	**ARUint32**
typedef int	**AR_PIXEL_FORMAT** *ARToolKit pixel-format specifiers.*

Functions

ARUint32	**arGetVersion** (char **versionStringRef) *Get the ARToolKit version information in numberic and string format.*
int	**arInitCparam** (**ARParam** *param) *initialize camera parameters.*
int	**arLoadPatt** (const char *filename) *load markers description from a file*
int	**arDetectMarker** (ARUint8 *dataPtr, int thresh, **ARMarkerInfo** **marker_info, int *marker_num) *main function to detect the square markers in the video input frame.*
int	**arDetectMarkerLite** (ARUint8 *dataPtr, int thresh, **ARMarkerInfo** **marker_info, int *marker_num) *main function to detect rapidly the square markers in the video input frame.*
double	**arGetTransMat** (**ARMarkerInfo** *marker_info, double center[2], double width, double conv[3][4]) *compute camera position in function of detected markers.*
double	**arGetTransMatCont** (**ARMarkerInfo** *marker_info, double prev_conv[3][4], double center[2], double width, double conv[3][4]) *compute camera position in function of detected marker with an history function.*
double	**arGetTransMat2** (double rot[3][3], double pos2d[][2], double pos3d[][2], int num, double conv[3][4])
double	**arGetTransMat3** (double rot[3][3], double ppos2d[][2], double ppos3d[][2], int num, double conv[3][4], double *dist_factor, double cpara[3][4])
double	**arGetTransMat4** (double rot[3][3], double ppos2d[][2], double ppos3d[][3], int num, double conv[3][4])
double	**arGetTransMat5** (double rot[3][3], double ppos2d[][2], double ppos3d[][3], int num, double conv[3][4], double *dist_factor, double cpara[3][4])
int	**arFreePatt** (int patt_no) *remove a pattern from memory.*
int	**arActivatePatt** (int pat_no) *activate a pattern on the recognition procedure.*
int	**arDeactivatePatt** (int pat_no) *desactivate a pattern on the recognition procedure.*
int	**arSavePatt** (ARUint8 *image, **ARMarkerInfo** *marker_info, char *filename) *save a marker.*
int	**arUtilMatInv** (double s[3][4], double d[3][4]) *Inverse a non-square matrix.*
int	**arUtilMatMul** (double s1[3][4], double s2[3][4], double d[3][4]) *Multiplication of two matrix.*
int	**arUtilMat2QuatPos** (double m[3][4], double q[4], double p[3]) *extract a quaternion/position of matrix.*

int	**arUtilQuatPos2Mat** (double q[4], double p[3], double m[3][4]) *create a matrix with a quaternion/position.*
double	**arUtilTimer** (void) *get the time with the ARToolkit timer.*
void	**arUtilTimerReset** (void) *reset the internal timer of ARToolkit.*
void	**arUtilSleep** (int msec) *sleep the actual thread.*
ARInt16 *	**arLabeling** (ARUint8 *image, int thresh, int *label_num, int **area, double **pos, int **clip, int **label_ref) *extract connected components from image.*
void	**arLabelingCleanup** (void) *clean up static data allocated by arLabeling.*
void	**arGetImgFeature** (int *num, int **area, int **clip, double **pos) *XXXBK.*
ARMarkerInfo2 *	**arDetectMarker2** (ARInt16 *limage, int label_num, int *label_ref, int *warea, double *wpos, int *wclip, int area_max, int area_min, double factor, int *marker_num) *XXXBK.*
ARMarkerInfo *	**arGetMarkerInfo** (ARUint8 *image, **ARMarkerInfo2** *marker_info2, int *marker_num) *information on*
int	**arGetCode** (ARUint8 *image, int *x_coord, int *y_coord, int *vertex, int *code, int *dir, double *cf) *XXXBK.*
int	**arGetPatt** (ARUint8 *image, int *x_coord, int *y_coord, int *vertex, ARUint8 ext_pat[AR_PATT_SIZE_Y][AR_PATT_SIZE_X][3]) *Get a normalized pattern from a video image.*
int	**arGetLine** (int x_coord[], int y_coord[], int coord_num, int vertex[], double line[4][3], double v[4][2]) *estimate a line from a list of point.*
int	**arGetContour** (ARInt16 *limage, int *label_ref, int label, int clip[4], **ARMarkerInfo2** *marker_info2) *XXXBK.*
double	**arModifyMatrix** (double rot[3][3], double trans[3], double cpara[3][4], double vertex[][3], double pos2d[][2], int num) *XXXBK.*
int	**arGetAngle** (double rot[3][3], double *wa, double *wb, double *wc) *extract euler angle from a rotation matrix.*
int	**arGetRot** (double a, double b, double c, double rot[3][3]) *create a rotation matrix with euler angle.*
int	**arGetNewMatrix** (double a, double b, double c, double trans[3], double trans2[3][4], double cpara[3][4], double ret[3][4]) *XXXBK.*
int	**arGetInitRot** (**ARMarkerInfo** *marker_info, double cpara[3][4], double rot[3][3]) *XXXBK.*
int	**arsInitCparam** (ARSParam *sparam)
void	**arsGetImgFeature** (int *num, int **area, int **clip, double **pos, int LorR)
ARInt16 *	**arsLabeling** (ARUint8 *image, int thresh, int *label_num, int **area, double **pos, int **clip, int **label_ref, int LorR)
int	**arsGetLine** (int x_coord[], int y_coord[], int coord_num, int vertex[], double line[4][3], double v[4][2], int LorR)
ARMarkerInfo *	**arsGetMarkerInfo** (ARUint8 *image, **ARMarkerInfo2** *marker_info2, int *marker_num, int LorR)
int	**arsDetectMarker** (ARUint8 *dataPtr, int thresh, **ARMarkerInfo** **marker_info, int *marker_num, int LorR)
int	**arsDetectMarkerLite** (ARUint8 *dataPtr, int thresh, **ARMarkerInfo** **marker_info, int *marker_num, int LorR)
double	**arsGetTransMat** (**ARMarkerInfo** *marker_infoL, **ARMarkerInfo** *marker_infoR, double center[2], double width, double transL[3][4], double transR[3][4])
double	**arsGetTransMatCont** (**ARMarkerInfo** *marker_infoL, **ARMarkerInfo** *marker_infoR, double prev_conv[3][4], double center[2], double width, double transL[3][4], double transR[3][4])
double	**arsGetTransMat2** (double rot[3][3], double ppos2dL[][2], double ppos3dL[][3], int numL, double ppos2dR[][2], double ppos3dR[][3], int numR, double transL[3][4], double transR[3][4])
double	**arsGetPosErr** (double pos2dL[2], double pos2dR[2])
int	**arsCheckPosition** (double pos2dL[2], double pos2dR[2], double thresh)
int	**arsCheckMarkerPosition** (**ARMarkerInfo** *marker_infoL, **ARMarkerInfo** *marker_infoR, double thresh)
double	**arsModifyMatrix** (double rot[3][3], double trans[3], ARSParam *arsParam, double pos3dL[][3], double pos2dL[][2], int numL, double pos3dR[][3], double pos2dR[][2], int numR)

Variables

int	**arDebug** *activate artoolkit debug mode*
ARUint8 *	**arImage** *internal image*
int	**arFittingMode** *fitting display mode use by ARToolkit.*
int	**arImageProcMode** *define the image size mode for marker detection.*
ARParam	**arParam** *internal intrinsic camera parameter*
int	**arImXsize** *internal image size in width.*
int	**arImYsize** *internal image size in heigth*
int	**arTemplateMatchingMode** *XXXBK.*
int	**arMatchingPCAMode** *XXXBK.*
ARUint8 *	**arImageL**
ARUint8 *	**arImageR**

ARSParam	**arsParam**
double	**arsMatR2L** [3][4]

Detailed Description

ARToolKit subroutines.

Core of the ARToolKit Library. This file provides image analysis and marker detection routines. Differents routines give access to camera and marker configurations. Other routines manipulate marker info structures for deliver 3D transformation of markers (more precisely the position of the camera in function of the marker coordinate system).

Remarks:

History :

Author:
Hirokazu Kato kato@sys.im.hiroshima-cu.ac.jp

Version:
3.1

Date:
01/12/07

Define Documentation

#define arMalloc(V, T, S)

Value:

```
{ if( ((V) = (T *)malloc( sizeof(T) * (S) )) == 0 ) \
{printf("malloc error!!\n"); exit(1);} }
```

allocation macro function

allocate S elements of type T.

Parameters:
V returned allocated area pointer
T type of element
S number of elements

Typedef Documentation

AR_PIXEL_FORMAT

ARToolKit pixel-format specifiers.

ARToolKit functions can accept pixel data in a variety of formats. This enumerations provides a set of constants you can use to request data in a particular pixel format from an ARToolKit function that returns data to you, or to specify that data you are providing to an ARToolKit function is in a particular pixel format.

AR_PIXEL_FORMAT_RGB Each pixel is represented by 24 bits. Eight bits per each Red, Green, and Blue component. This is the native 24 bit format for the Mac platform.

AR_PIXEL_FORMAT_BGR Each pixel is represented by 24 bits. Eight bits per each Blue, Red, and Green component. This is the native 24 bit format for the Win32 platform.

AR_PIXEL_FORMAT_RGBA Each pixel is represented by 32 bits. Eight bits per each Red, Green, Blue, and Alpha component.

AR_PIXEL_FORMAT_BGRA Each pixel is represented by 32 bits. Eight bits per each Blue, Green, Red, and Alpha component. This is the native 32 bit format for the Win32 platform.

AR_PIXEL_FORMAT_ABGR Each pixel is represented by 32 bits. Eight bits per each Alpha, Blue, Green, and Red component. This is the native 32 bit format for the SGI platform.

AR_PIXEL_FORMAT_ARGB Each pixel is represented by 32 bits. Eight bits per each Alpha, Red, Green, and Blue component. This is the native 32 bit format for the Mac platform.

AR_PIXEL_FORMAT_MONO Each pixel is represented by 8 bits of luminance information.

AR_PIXEL_FORMAT_2vuy 8-bit 4:2:2 Component Y'CbCr format. Each 16 bit pixel is represented by an unsigned eight bit luminance component and two unsigned eight bit chroma components. Each pair of pixels shares a common set of chroma values. The components are ordered in memory; Cb, Y0, Cr, Y1. The luminance components have a range of [16, 235], while the chroma value has a range of [16, 240]. This is consistent with the CCIR601 spec. This format is fairly prevalent on both Mac and Win32 platforms. '2vuy' is the Apple QuickTime four-character code for this pixel format. The equivalent Microsoft fourCC is 'UYVY'.

AR_PIXEL_FORMAT_yuvs 8-bit 4:2:2 Component Y'CbCr format. Identical to the AR_PIXEL_FORMAT_2vuy except each 16 bit word has been byte swapped. This results in a component ordering of; Y0, Cb, Y1, Cr. This is most prevalent yuv 4:2:2 format on both Mac and Win32 platforms. 'yuvs' is the Apple QuickTime four-character code for this pixel format. The equivalent Microsoft fourCC is 'YUY2'.

Function Documentation

int arActivatePatt(int *pat_no*)

activate a pattern on the recognition procedure.

Activate a pattern to be check during the template matching operation.

Parameters:

patt_no number of pattern to activate

Returns:

return 1 in success, -1 if error

int arDeactivatePatt(int *pat_no*)

desactivate a pattern on the recognition procedure.

Desactivate a pattern for not be check during the template matching operation.

Parameters:

patt_no number of pattern to desactivate

Returns:

return 1 in success, -1 if error

```
int arDetectMarker( ARUint8 *       dataPtr,
                    int             thresh,
                    ARMarkerInfo ** marker_info,
                    int *           marker_num
                  )
```

main function to detect the square markers in the video input frame.

This function proceeds to thresholding, labeling, contour extraction and line corner estimation (and maintains an history). It's one of the main function of the detection routine with arGetTransMat.

Parameters:

dataPtr	a pointer to the color image which is to be searched for square markers. The pixel format depend of your architecture. Generally ABGR, but the images are treated as a gray scale, so the order of BGR components does not matter. However the ordering of the alpha comp, A, is important.
thresh	specifies the threshold value (between 0-255) to be used to convert the input image into a binary image.
marker_info	a pointer to an array of **ARMarkerInfo** structures returned which contain all the information about the detected squares in the image
marker_num	the number of detected markers in the image.

Returns:

0 when the function completes normally, -1 otherwise

```
ARMarkerInfo2* arDetectMarker2( ARInt16 * limage,
                                int       label_num,
                                int *     label_ref,
                                int *     warea,
                                double *  wpos,
                                int *     wclip,
                                int       area_max,
                                int       area_min,
                                double    factor,
                                int *     marker_num
                              )
```

XXXBK.

XXXBK

Parameters:

limage	XXXBK
label_num	XXXBK
label_ref	XXXBK
warea	XXXBK
wpos	XXXBK
wclip	XXXBK
area_max	XXXBK
area_min	XXXBK
factor	XXXBK
marker_num	XXXBK

Returns:

XXXBK XXXBK

```
int arDetectMarkerLite( ARUint8 *       dataPtr,
                        int             thresh,
                        ARMarkerInfo ** marker_info,
                        int *           marker_num
                      )
```

main function to detect rapidly the square markers in the video input frame.

this function is a simpler version of arDetectMarker that does not have the same error correction functions and so runs a little faster, but is more error prone

Parameters:

dataPtr	a pointer to the color image which is to be searched for square markers. The pixel format depend of your architecture. Generally ABGR, but the images are treated as a gray scale, so the order of BGR components does not matter. However the ordering of the alpha component, A, is important.
thresh	specifies the threshold value (between 0-255) to be used to convert the input image into a binary image.
marker_info	a pointer to an array of **ARMarkerInfo** structures returned which contain all the information about the detected squares in the image
marker_num	the number of detected markers in the image.

Returns:

0 when the function completes normally, -1 otherwise

```
int arFreePatt( int patt_no )
```

remove a pattern from memory.

desactivate a pattern and remove from memory. post-condition of this function is unavailability of the pattern.

Parameters:

patt_no	number of pattern to free

Returns:

return 1 in success, -1 if error

```
int arGetAngle( double   rot[3][3],
                double * wa,
                double * wb,
                double * wc
              )
```

extract euler angle from a rotation matrix.

Based on a matrix rotation representation, furnish the cprresponding euler angles.

Parameters:

rot	the initial rotation matrix
wa	XXXBK:which element ?
wb	XXXBK:which element ?
wc	XXXBK:which element ?

Returns:

XXXBK

```
int arGetCode( ARUint8 * image,
               int *     x_coord,
               int *     y_coord,
               int *     vertex,
               int *     code,
               int *     dir,
               double *  cf
             )
```

XXXBK.

XXXBK

Parameters:

image	XXXBK
x_coord	XXXBK
y_coord	XXXBK
vertex	XXXBK
code	XXXBK
dir	XXXBK
cf	XXXBK

Returns:

XXXBK

```
int arGetContour( ARInt16 *          limage,
                  int *              label_ref,
                  int                label,
                  int                clip[4],
                  ARMarkerInfo2 *    marker_info2
                )
```

XXXBK.

XXXBK

Parameters:

limage	XXXBK
label_ref	XXXBK
label	XXXBK
clip	XXXBK
marker_info2	XXXBK

Returns:

XXXBK

```
void arGetImgFeature( int *       num,
                      int **      area,
                      int **      clip,
                      double **   pos
                    )
```

XXXBK.

XXXBK

Parameters:

num	XXXBK
area	XXXBK
clip	XXXBK
pos	XXXBK

```
int arGetInitRot( ARMarkerInfo * marker_info,
                  double         cpara[3][4],
                  double         rot[3][3]
                )
```

XXXBK.

XXXBK:initial of what ?

Parameters:

marker_info	XXXBK
cpara	XXXBK
rot	XXXBK

Returns:

XXXBK

```
int arGetLine( int      x_coord[],
               int      y_coord[],
               int      coord_num,
               int      vertex[],
               double   line[4][3],
               double   v[4][2]
             )
```

estimate a line from a list of point.

Compute a linear regression from a list of point.

Parameters:

x_coord	X coordinate of points
y_coord	Y coordinate of points
coord_num	number of points
vertex	XXXBK
line	XXXBK
v	XXXBK

Returns:

XXXBK

```
ARMarkerInfo* arGetMarkerInfo( ARUint8 *          image,
                               ARMarkerInfo2 *    marker_info2,
                               int *              marker_num
                             )
```

information on

XXXBK

Parameters:

image	XXXBK
marker_info2	XXXBK
marker_num	XXXBK

Returns:

XXXBK

```
int arGetNewMatrix( double a,
                    double b,
                    double c,
                    double trans[3],
                    double trans2[3][4],
                    double cpara[3][4],
                    double ret[3][4]
                  )
```

XXXBK.

XXXBK

Parameters:

a	XXXBK
b	XXXBK
c	XXXBK
trans	XXXBK
trans2	XXXBK
cpara	XXXBK
ret	XXXBK

Returns:

XXXBK

```
int arGetPatt( ARUint8 * image,
               int *     x_coord,
               int *     y_coord,
               int *     vertex,
               ARUint8   ext_pat[AR_PATT_SIZE_Y][AR_PATT_SIZE_X][3]
             )
```

Get a normalized pattern from a video image.

This function returns a normalized pattern from a video image. The format is a table with AR_PATT_SIZE_X by AR_PATT_SIZE_Y

Parameters:

image	video input image
x_coord	XXXBK
y_coord	XXXBK
vertex	XXXBK
ext_pat	detected pattern.

Returns:

XXXBK

```
int arGetRot( double a,
              double b,
              double c,
              double rot[3][3]
            )
```

create a rotation matrix with euler angle.

Based on a euler description, furnish a rotation matrix.

Parameters:

a	XXXBK:which element ?
b	XXXBK:which element ?
c	XXXBK:which element ?
rot	the resulted rotation matrix

Returns:
XXXBK

```
double arGetTransMat( ARMarkerInfo * marker_info,
                      double         center[2],
                      double         width,
                      double         conv[3][4]
                    )
```

compute camera position in function of detected markers.

calculate the transformation between a detected marker and the real camera, i.e. the position and orientation of the camera relative to the tracking mark.

Parameters:

marker_info	the structure containing the parameters for the marker for which the camera position and orientation is to be found relative to. This structure is found using arDetectMarker.
center	the physical center of the marker. arGetTransMat assumes that the marker is in x-y plane, and z axis is pointing downwards from marker plane. So vertex positions can be represented in 2D coordinates by ignoring the z axis information. The marker vertices are specified in order of clockwise.
width	the size of the marker (in mm).
conv	the transformation matrix from the marker coordinates to camera coordinate frame, that is the relative position of real camera to the real marker

Returns:
always 0.

```
double arGetTransMatCont( ARMarkerInfo * marker_info,
                          double         prev_conv[3][4],
                          double         center[2],
                          double         width,
                          double         conv[3][4]
                        )
```

compute camera position in function of detected marker with an history function.

calculate the transformation between a detected marker and the real camera, i.e. the position and orientation of the camera relative to the tracking mark. Since this routine operate on previous values, the result are more stable (less jittering).

Parameters:

marker_info	the structure containing the parameters for the marker for which the camera position and orientation is to be found relative to. This structure is found using arDetectMarker.
prev_conv	the previous transformation matrix obtain.
center	the physical center of the marker. arGetTransMat assumes that the marker is in x-y plane, and z axis is pointing downwards from marker plane. So vertex positions can be represented in 2D coordinates by ignoring the z axis information. The marker vertices are specified in order of clockwise.
width	the size of the marker (in mm).
conv	the transformation matrix from the marker coordinates to camera coordinate frame, that is the relative position of real camera to the real marker

Returns:
always 0.

```
ARUint32 arGetVersion( char ** versionStringRef )
```

Get the ARToolKit version information in numberic and string format.

As of version 2.72, ARToolKit now allows querying of the version number of the toolkit available at runtime. It is highly recommended that any calling program that depends on features in a certain ARToolKit version, check at runtime that it is linked to a version of ARToolKit that can supply those features. It is NOT sufficient to check the ARToolKit SDK header versions, since with ARToolKit implemented in dynamically-loaded libraries, there is no guarantee that the version of ARToolKit installed on the machine at run-time will as recent as the version of the ARToolKit SDK which the host program was compiled against. The version information is reported in binary-coded decimal format, and optionally in an ASCII string. See the config.h header for more discussion of the definition of major, minor, tiny and build version numbers.

Parameters:

versionStringRef	If non-NULL, the location pointed to will be filled with a pointer to a string containing the version information. Fields in the version string are separated by spaces. As of version 2.72.0, there is only one field implemented, and this field contains the major, minor and tiny version numbers in dotted-decimal format. The string is guaranteed to contain at least this field in all future versions of the toolkit. Later versions of the toolkit may add other fields to this string to report other types of version information. The storage for the string is malloc'ed inside the function. The caller is responsible for free'ing the string.

Returns:
Returns the full version number of the ARToolKit in binary coded decimal (BCD) format. BCD format allows simple tests of version number in the caller e.g. if ((arGetVersion(NULL) >> 16) > 0x0272) printf("This release is later than 2.72\n"); The major version number is encoded in the most-significant byte (bits 31-24), the minor version number in the second-most-significant byte (bits 23-16), the tiny version number in the third-most-significant byte (bits 15-8), and the build version number in the least-significant byte (bits 7-0).

int arInitCparam(ARParam * *param*)

initialize camera parameters.

set the camera parameters specified in the camera parameters structure *param to static memory in the AR library. These camera parameters are typically read from a data file at program startup. In the video-see through AR applications, the default camera parameters are sufficient, no camera calibration is needed.

Parameters:

param the camera parameter structure

Returns:

always 0

ARInt16* arLabeling(ARUint8 * *image*,
int *thresh*,
int * *label_num*,
int ** *area*,
double ** *pos*,
int ** *clip*,
int ** *label_ref*
)

extract connected components from image.

Label the input image, i.e. extract connected components from the input video image.

Parameters:

image	input image, as returned by **arVideoGetImage()**
thresh	lighting threshold
label_num	Ouput- number of detected components
area	On return, if label_num > 0, points to an array of ints, one for each detected component.
pos	On return, if label_num > 0, points to an array of doubles, one for each detected component.
clip	On return, if label_num > 0, points to an array of ints, one for each detected component.
label_ref	On return, if label_num > 0, points to an array of ints, one for each detected component.

Returns:

returns a pointer to the labeled output image, ready for passing onto the next stage of processing.

void arLabelingCleanup(void)

clean up static data allocated by arLabeling.

In debug mode, arLabeling may allocate and use static storage. This function deallocates this storage.

int arLoadPatt(const char * *filename*)

load markers description from a file

load the bitmap pattern specified in the file filename into the pattern matching array for later use by the marker detection routines.

Parameters:

filename name of the file containing the pattern bitmap to be loaded

Returns:

the identity number of the pattern loaded or –1 if the pattern load failed.

double arModifyMatrix(double *rot*[3][3],
double *trans*[3],
double *cpara*[3][4],
double *vertex*[][3],
double *pos2d*[][2],
int *num*
)

XXXBK.

XXXBK

Parameters:

rot	XXXBK
trans	XXXBK
cpara	XXXBK
vertex	XXXBK
pos2d	XXXBK
num	XXXBK

Returns:

XXXBK

```
int arSavePatt( ARUint8 *       image,
                ARMarkerInfo * marker_info,
                char *         filename
              )
```

save a marker.

used in mk_patt to save a bitmap of the pattern of the currently detected marker. The saved image is a table of the normalized viewed pattern.

Parameters:

image	a pointer to the image containing the marker pattern to be trained.
marker_info	a pointer to the **ARMarkerInfo** structure of the pattern to be trained.
filename	The name of the file where the bitmap image is to be saved.

Returns:

0 if the bitmap image is successfully saved, -1 otherwise.

```
int arUtilMat2QuatPos( double m[3][4],
                       double q[4],
                       double p[3]
                     )
```

extract a quaternion/position of matrix.

Extract a rotation (quaternion format) and a position (vector format) from a transformation matrix. The precondition is an euclidian matrix.

Parameters:

m	source matrix
q	a rotation represented by a quaternion.
p	a translation represented by a vector.

Returns:

0 if the extraction success, -1 otherwise (quaternion not normalize)

```
int arUtilMatInv( double s[3][4],
                  double d[3][4]
                )
```

Inverse a non-square matrix.

Inverse a matrix in a non homogeneous format. The matrix need to be euclidian.

Parameters:

s	matrix input
d	resulted inverse matrix.

Returns:

0 if the inversion success, -1 otherwise

Remarks:

input matrix can be also output matrix

```
int arUtilMatMul( double s1[3][4],
                  double s2[3][4],
                  double d[3][4]
                )
```

Multiplication of two matrix.

This procedure do a multiplication matrix between s1 and s2 and return the result in d : d=s1*s2. The precondition is the output matrix need to be different of input matrix. The precondition is euclidian matrix.

Parameters:

s1	first matrix.
s2	second matrix.
d	resulted multiplication matrix.

Returns:

0 if the multiplication success, -1 otherwise

```
int arUtilQuatPos2Mat( double q[4],
                       double p[3],
                       double m[3][4]
                     )
```

create a matrix with a quaternion/position.

Create a transformation matrix from a quaternion rotation and a vector translation.

Parameters:

q a rotation represented by a quaternion.
p a translation represented by a vector.
m destination matrix

Returns:
always 0

void arUtilSleep(int *msec*)

sleep the actual thread.

Sleep the actual thread.

Parameters:
msec time to sleep (in millisecond)

double arUtilTimer(void)

get the time with the ARToolkit timer.

Give the time elapsed since the reset of the timer.

Returns:
elapsed time (in milliseconds)

void arUtilTimerReset(void)

reset the internal timer of ARToolkit.

Reset the internal timer used by ARToolKit. timer measurement (with **arUtilTimer()**).

Variable Documentation

int arDebug

activate artoolkit debug mode

control debug informations in ARToolKit. the possible values are:

- 0: not in debug mode
- 1: in debug mode by default: 0

int arFittingMode

fitting display mode use by ARToolkit.

Correction mode for the distorsion of the camera. You can enable a correction with a texture mapping. the possible values are:

- AR_FITTING_TO_INPUT: input image
- AR_FITTING_TO_IDEAL: compensated image by default: DEFAULT_FITTING_MODE in config.h

ARUint8 * arImage

internal image

internal image used. (access only for debugging ARToolKit) by default: NULL

int arImageProcMode

define the image size mode for marker detection.

Video image size for marker detection. This control if all the image is analyzed the possible values are :

- AR_IMAGE_PROC_IN_FULL: full image uses.
- AR_IMAGE_PROC_IN_HALF: half image uses. by default: DEFAULT_IMAGE_PROC_MODE in config.h

int arImXsize

internal image size in width.

internal image size in width (generally initialize in arInitCparam)

int arImYsize

internal image size in heigth

internal image size in heigth (generally initialize in arInitCparam)

int arMatchingPCAMode

XXXBK.

XXXBK the possible values are : -AR_MATCHING_WITHOUT_PCA: without PCA -AR_MATCHING_WITH_PCA: with PCA by default: DEFAULT_MATCHING_PCA_MODE in config.h

ARParam arParam

internal intrinsic camera parameter

internal variable for camera intrinsic parameters

int arTemplateMatchingMode

XXXBK.

XXXBK the possible values are : AR_TEMPLATE_MATCHING_COLOR: Color Template AR_TEMPLATE_MATCHING_BW: BW Template by default: DEFAULT_TEMPLATE_MATCHING_MODE in config.h