ARToolKit

version 2.33

November 2000

Hirokazu Kato[α]
Mark Billinghurst[β]
Ivan Poupyrev[γ]

[α]Hiroshima City University
kato@sys.im.hiroshima-cu.ac.jp

[β]Human Interface Technology Laboratory
University of Washington
Box 352-142
Seattle WA 98195
grof@hitl.washington.edu

[γ]ATR MIC Labs
MIC Research Labs
ATR International
Hikaridai, Seika, Souraku-gun
Kyoto 619-02, Japan
poup@mic.atr.co.jp

Table of Contents

1 Introduction

ARToolKit is a C language software library that lets programmers easily develop Augmented Reality applications. Augmented Reality (AR) is the overlay of virtual computer graphics images on the real world, and has many potential applications in industrial and academic research.

One of the most difficult parts of developing an Augmented Reality application is precisely calculating the user's viewpoint in real time so that the virtual images are exactly aligned with real world objects. ARToolKit uses computer vision techniques to calculate the real camera position and orientation relative to marked cards, allowing the programmer to overlay virtual objects onto these cards. The fast, precise tracking provided by ARToolKit should enable the rapid development of many new and interesting AR applications.

This document contains a complete description of the ARToolKit library, how to install it, and how to use its functionality in AR applications. Several simple sample applications are provided with ARToolKit to enable the programmer to get started immediately. ARToolKit includes the tracking libraries and complete source code for these libraries enabling programming to port the code to a variety of platforms or customize it for their own applications. If you have installed and used previous versions of ARToolKit you may wish to start by reading section XX, What's New, otherwise it is recommended that you read the first XX sections before trying to develop your own applications.

ARToolKit currently runs the SGI IRIX, PC Linux and PC Windows 95/98/NT/2000 platforms. There are separate versions of ARToolKit for each. The functionality of each version of the toolkit is the same, but the performance may vary depending on the different hardware configurations. On the SGI platform, ARToolKit has only been tested and run on the SGI O2 computers, however it should also run on other SGI computers with video input capabilities.

The current version of ARToolKit supports both video and optical see-through augmented reality. Video see-through AR is where virtual images are overlaid on live video of the real world. The alternative is optical see-through augmented reality, where computer graphics are overlaid directly on a view of the real world. Optical see-through augmented reality typically requires a see-through head mounted display and has more complicated camera calibration and registration requirements.

Comments and questions about ARToolKit and any bug reports should be sent to Hirokazu Kato (kato@sys.im.hiroshima-cu.ac.jp) or Mark Billinghurst (grof@hitl.washington.edu). There is also an electronic mailing list which is used to broadcast news about new releases of ARToolKit, bug fixes and applications people are working on. To subscribe, send email to **majordomo@hitl.washington.edu** with the words **"subscribe artoolkit"** in the body of the email. After you're subscribed if you send email to **artoolkit@hitl.washington.edu** it will be broadcast to all members of the mailing list.

NOTE: ARToolKit is free for use in non-commercial applications and is distributed as open-source under the GPL license (see Appendix B). Those interested in using ARToolKit in commercial applications should contact either Hirokazu Kato or Mark Billinghurst for licensing details.

2 Installation

The PC, SGI and LINUX versions of the ARToolKit software are available for free download from http://www.hitl.washington.edu/resarch/shared_space/download/

PC Installation

ARToolKit for the PC is distributed as a single compressed file, `ARToolKit2.32.win32.zip`. Once you receive this file copy it into the directory where you want ARToolKit to be located and uncompress it using WinZip or a similar decompression utility.

WinZip should create the following files and directory structure:

[PUT MORE HERE]

The bin directory contains some sample executable programs that you can execute right away, as described in the next section. Source code for all these programs is found in the `examples` directory. The compiled ARToolKit libraries are found in the `lib` directory and complete source code for the libraries found in the `lib/Src` directory.

ARToolkit has been compiled and tested on PCs running Windows 95/98 or NT equipped with a Connectix Color QuickCam and on Windows 95/98 with USB cameras. ARToolKit uses Microsoft's Vision SDK so there is no reason why it should not run with any camera and frame grabber board supported by the Microsoft Vision SDK and Video for Windows.

The sample programs included with the ARToolKit all make use of the GLUT OpenGL interface library. In order for these programs to compile, GLUT 3.6 or higher must be installed. The GLUT libraries are available for free download from http://reality.sgi.com/opengl/glut3/.

SGI IRIX/ PC Linux Installation

ARToolKit for IRIX/Linux is distributed as the file, `ARToolKit2.33.tar.gz`. Once you receive this file copy it into the directory where you want ARToolKit to be located and uncompress it using `gunzip` and then `tar cvf ARToolKit2.33.tar` or a similar decompression utility.

```
$gunzip ARToolKit2.33.tar.gz
$tar cvf ARToolKit2.33.tar
```

…

After decompressing ARToolKit you will have the following directories:

```
$ls
bin examples include lib patterns util
```

The `bin` directory contains some sample executable programs that you can execute right away, as described in the next section. Source code for all these programs is found in the `examples` directory. The compiled ARToolKit libraries are found in the `lib` directory and complete source code for the libraries found in the `lib/Src` directory, while the `include` directory contains the necessary header files. The `patterns` directory contains sample tracking patterns and in `util`

there are various utility programs for training new tracking patterns, and performing camera calibration.

The sample programs included with the ARToolKit all make use of the GLUT OpenGL interface library. In order for these programs to compile, GLUT 3.6 or higher must be installed. The GLUT libraries are available for free download from http://reality.sgi.com/opengl/glut3/.

3 Getting Started

Once ARToolKit has been installed there is a sample program, `simpleTest`, in the `bin` directory that can be run to show the capabilities of ARToolKit. In order to run the code you need to print out the paper fiduccial marks that are contained in the directory `patterns` - these `are sampPatt1.pdf, sampPatt2.pdf, kanjiPatt.pdf` and `hiroPatt.pdf.` Best performance is achieved if these marks are glued to pieces of cardboard to keep them flat.

The next sections first describe the hardware requirements to run ARToolKit and then describe how to run ARToolKit on a Windows PC, a Linux PC or an SGI IRIX system. In each case the `simpleTest` program output is the same, so the final section describes how the `simpleTest` program looks when it is running.

Hardware Requirements

The basic hardware requirements in order to develop and run ARToolKit applications are a video camera and video capture capabilities. On a Windows PC (95/98/2000) video capture can be achieved by using a USB camera, a frame grabber card or graphics card with video input. The device used needs to have Video For Windows drivers. Suitable frame grabber cards and USB cameras are listed in Section 10 - Resources. For Linux machines a frame grabber card with Video 4 Linux support is required. Suitable hardware is listed in Section 10, as well as on the Video 4 Linux website: http://[INSERT URL HERE]. After installing the Linux capture card the correct software driver for the card must also be installed. Driver software is also available from the Video 4 Linux website. Finally, for SGI computers, ARToolKit runs on O2 machines with the Audio Video option installed. This is a standard hardware card available from SGI or SGI hardware resellers.

Running ARToolKit on a Windows PC

[INSERT MORE HERE]

Running ARToolKit on an SGI

Running ARToolKit on an Linux PC

simpleTest Output

A video window will appear on the screen. When you point the camera at either or both of the paper markers you should see a virtual red cone or blue cube appear attached to the markers. Figure 1 shows a screen snapshot of the program running. As the real markers are moved the virtual objects should move with them and appear exactly aligned with the real markers.

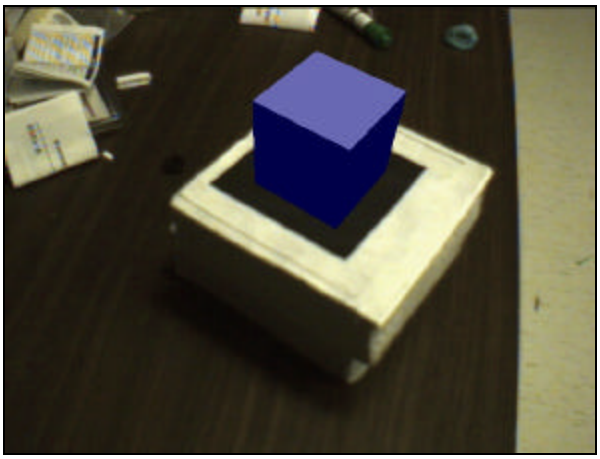


Fig 1: the Simple Program Running – a virtual block aligned with a real mark

In order for the virtual object to appear the entire black square border of the marker must be in view and also the pattern contained within the square border must be visible at all times. If the virtual images do not appear, or flicker in and out of view it may be because of lighting conditions. This can often be fixed by changing the lighting threshold value used by the image processing routines. [IS THIS STILL CORRECT] If you hit the 't' key on the keyboard you will be prompted to enter a new threshold value. This should be a value between 0 and 255; the default is 100. Hitting the 'd' key will show the thresholded video image below the main video window. Patterns found in the input image are marked by a red square in the thresholded image. This will help you check the effect of lighting and threshold values on the video input. Figure 2 shows `simpleTest` with the threshold image on.

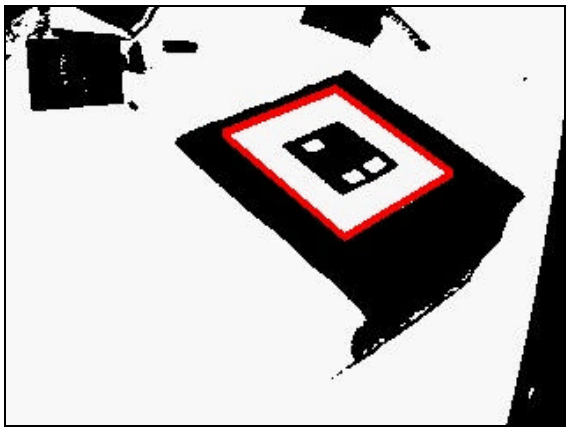


Fig 2: Thresholded Video Image with Identified Pattern

The 'Esc' quits the program and prints out frame rate information. **NOTE:** If the program crashes or is quit by any method other than hitting the Escape key you may need to reboot your computer before using any video applications again.

The `simpleTest` program shows how an application based on the ARToolKit software is able to calculate the camera viewpoint in real time and use this information to precisely overlay virtual images on a real world object. In the next section we explain how this works.

4 How ARToolKit Works

ARToolKit uses computer vision techniques to calculate the real camera viewpoint relative to a real world marker. There are several steps as shown in figures 3-5. First the live video image (figure 3) is turned into a binary (black or white) image based on a lighting threshold value (figure 4). This image is then searched for square regions. ARToolKit finds all the squares in the binary image, many of which are not the tracking markers. For each square, the pattern inside the square is captured and matched again some pre-trained pattern templates. If there is a match, then ARToolKit has found one of the AR tracking markers. ARToolKit then uses the known square size and pattern orientation to calculate the position of the real video camera relative to the physical marker. A 3x4 matrix is filled in with the video camera real world coordinates relative to the card. This matrix is then used to set the position of the virtual camera coordinates. Since the virtual and real camera coordinates are the same, the computer graphics that are drawn precisely overlay the real marker (figure 5). The OpenGL API is used for setting the virtual camera coordinates and drawing the virtual images.

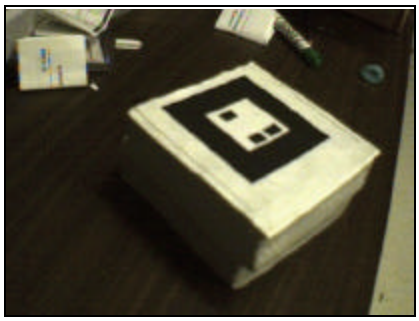


Fig 3: Input Video

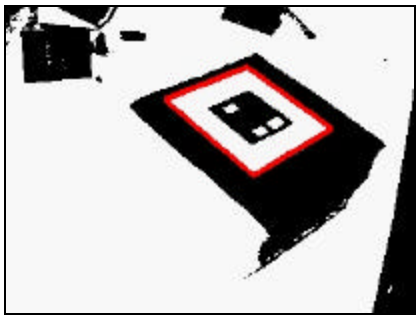


Fig 4: Thresholded Video

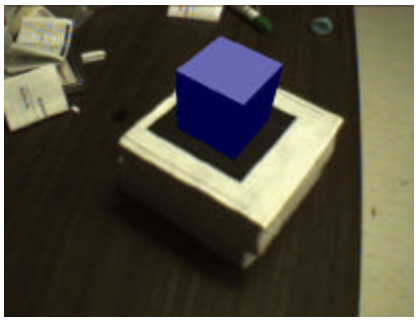
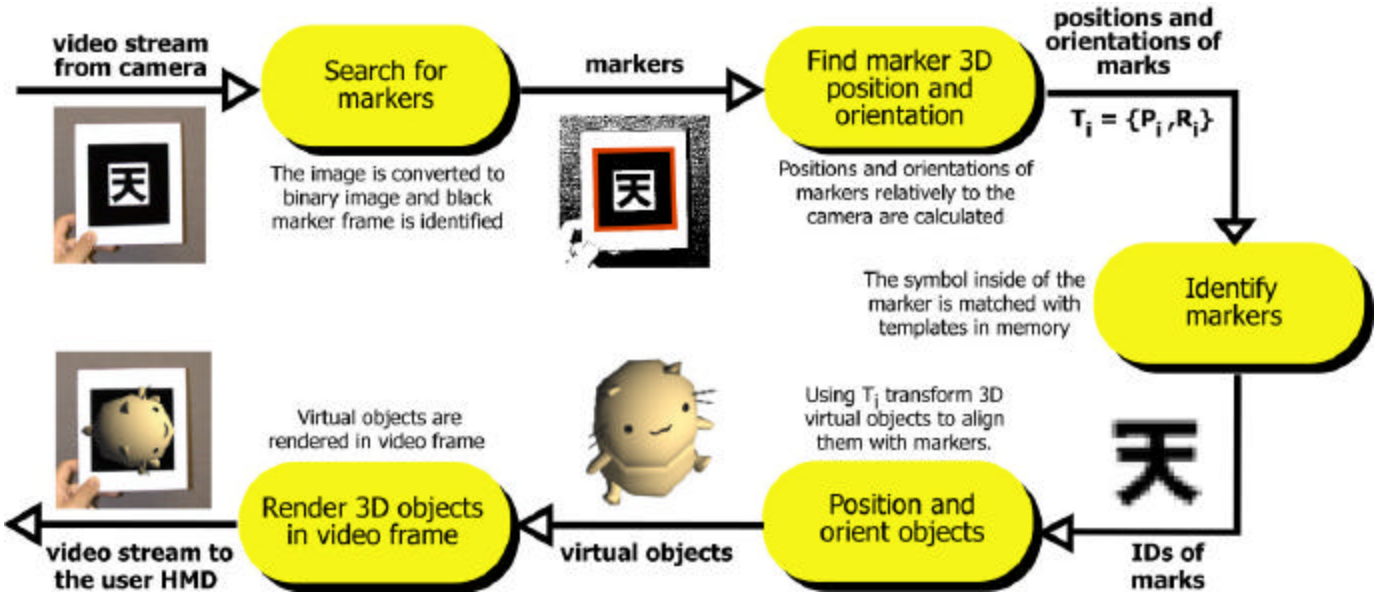


Fig 5: Virtual Overlay

The diagram below shows the image processing used in ARToolKit in more detail.



5 What's New

The previous version of ARToolKit available for download was version 2.11. There are the following differences between this and version 2.33.

[INSERT DIFFERENCES HERE]

6 Using ARToolKit to Develop AR Applications

There are two parts to developing applications that use ARToolKit; writing the application, and training image-processing routines on the real world markers that will be used in the application.

6.1 Developing the application

In writing an ARToolKit application the following steps must be taken:

1. Initialize the video path and read in the marker pattern files and camera parameters.
2. Grab a video input frame.
3. Detect the markers and recognized patterns in the video input frame.
4. Calculate the camera transformation relative to the detected patterns.
5. Draw the virtual objects on the detected patterns.
6. Close the video path down.

Steps 2 through 5 are repeated continuously until the application quits, while steps 1 and 6 are just performed on initialization and shutdown of the application respectively. In addition to these steps the application may need to respond to mouse, keyboard or other application specific events.

To show in detail how to develop an application we will step through the source code for the `simpleTest` program. This is found in the directory `examples/simple/`

The file we will be looking at is `simpleTest.c`. This program simply consists of a `main` routine and several graphics drawing routines. The main routine is shown below:

```
main(int argc, char **argv)
{
  init();

   arVideoCapStart();
   argMainLoop( NULL, keyEvent, mainLoop );
}
```

This routine calls an `init` initialization routine that contains code for initialization of the video path, reading in the marker and camera parameters and setup of the graphics window. This corresponds to step 1 above. Next, the function arVideoCapStart() starts video image capture. Finally, the `argMainLoop` function is called which starts the main program loop and associates the function `keyEvent` with any keyboard events and `mainLoop` with the main graphics

rendering loop. The definition of `argMainLoop` is contained in the file gsub.c that can be found in the directory `lib/Src/Gl/`

In `simpleTest.c` the functions which correspond to the six application steps above are shown in table 1.0. The functions corresponding to steps 2 through 5 are called within the `mainLoop` function. In the remainder of this section we will explain these function calls in more detail.

ARToolKit Step	Function
1. Initialize the application	*init*
2. Grab a video input frame.	*arVideoGetImage*
3. Detect the markers	*arDetectMarker*
4. Calculate camera transformation	*arGetTransMat*
5. Draw the virtual objects	*draw*
6. Close the video path down.	*cleanup*

Table 1.0 Function calls and code that corresponds to the ARToolKit applications steps.

init( )

The `init` routine is called from the main routine and is used to open the video path and read in the initial ARToolKit application parameters. The key parameters for an ARToolKit application are:

- the patterns that will be used for the pattern template matching and the virtual objects these patterns correspond to.
- the camera characteristics of the video camera being used.

These are both read in from file names that can either be specified on the command line or by using default hard-coded file names. In the `init` routine the default camera parameter file name is `Data/camera_para.dat`, while the default object file name is `Data/object_data.` The file containing the pattern names and virtual objects is read in with the function call:

```
/* load in the object data - trained markers and associated bitmap files */
if( (object=read_objectdata(odataname,&objectnum)) == NULL ) exit(0);
```

In the function `read_objectdata`, all of the trained patterns corresponding to the pattern names are read into AR library.

After these have been read in then the video path is opened and the video image size found:

```
/* open the video path */
if( arVideoOpen( vconf ) < 0 ) exit(0);

 /* find the size of the window */
 if( arVideoInqSize(&xsize, &ysize) < 0 ) exit(0);
 printf("Image size (x,y) = (%d,%d)\n", xsize, ysize);
```

The variable vconf contains the initial video configuration and is defined at the top of simpleTest.c.

Then the camera parameters are read in:

```
/* set the initial camera parameters */
```

```
if( arParamLoad(cparaname, 1, &wparam) < 0 ) {
        printf("Camera parameter load error !!\n");
        exit(0);
}
```

Next, the parameters are transformed for the current image size, because camera parameters change depending on the image size, even if same camera is used.

```
arParamChangeSize( &wparam, xsize, ysize, &cparam );
```

The camera parameters are set to those read in, the camera parameters printed to the screen and a graphics window opened:

```
if( xsize < 400 ) arResampleFlag = 0;
   else           arResampleFlag = 1;
   fullWindow            = 0;
   arDistortedFittingFlag  = 0;
   arDebug               = 0;
   arInitCparam( &cparam );
   printf("*** Camera Parameter ***\n");
   arParamDisp( &cparam );

   /* open the graphics window */
   argInit( &cparam, 1.0, fullWindow, 2, 1, 0 );
```

If fullWindow = 1 [PUT MORE HERE]

If arDistortedFittingFlag = 1, the video shown in the video output window is warp to correct for the distortions present in most camera lenses. Section XX describes the ARToolKit camera calibration utilities that can be used to collect camera lens parameters.

If arDebug = 1, thresholded images are generated in the image processing step and shown on-screen to the user. This additional step slows down the image processing. Finally, the local variable fullWindow is used for the setup of graphics window. If fullWindow is 1, the graphics are not drawn in a window, but full screen.

mainLoop

This is the routine where the bulk of the ARToolKit function calls are made and it contains code corresponding to steps 2 through 5 of the required application steps.

First a video frame is captured using the function arVideoGetImage:

```
/* grab a video frame */
   if( (dataPtr = (ARUint8 *)arVideoGetImage()) == NULL ) {
      arUtilSleep(2);
      return;
   }
```

The video image is then displayed on screen. This can either be an unwarped image, or an image warped to correct for camera distortions. Warping the image produces a more normal image, but can result in a significant reduction in video frame rate.

```
/* display the video image */
   if( dispmode ) {
      /* unwarped image */
      arDistortedFittingFlag  = 0;
      argDispImage2( dataPtr );
   }
   else {
      /* warped video image */
      arDistortedFittingFlag  = 1;
      argDispImage( dataPtr, 0, 0 );
   }
```

Then the function arDetectMarker is used to search the video image is searched for squares that have the correct marker patterns:

```
/* detect the markers in the video frame */
   if( arDetectMarker(dataPtr, thresh, &marker_info, &marker_num) < 0 ) {
      cleanup();
      exit(0);
   }
```

The number of markers found is contained in the variable `marker_num`, while `marker_info` is a pointer to a list of marker structures containing the coordinate information and recognition confidence values and object id numbers for each of the markers. The `marker_info` structure will be explained in more detail in section XX.

Next, all the confidence values of the detected markers are compared to associate the correct marker id number with the highest confidence value:

```
for( j = 0; j < marker_num; j++ ) {
        if( object[i].id == marker_info[j].id ) {
                if( k == -1 ) k = j;
                else {
                if( marker_info[k].cf < marker_info[j].cf ) k = j;
                }
        }
}
```

The transformation between the marker cards and camera can then be found by using the `arGetTransMat` function:

```
/* get the transformation between the marker and the real camera */
if( arGetTransMat(&marker_info[k],
```

```
            object_center, object[i].marker_width, object[i].trans) < 0 ) {
    object[i].visible = 0;
}
else {
    object[i].visible = 1;
}
```

The real camera position and orientation relative to the marker object `i` are contained in the 3x4 matrix, `object[i].trans`.

Finally, the virtual objects can be drawn on the card using the `draw` function:

```
/* draw the virtual objects attached to the tracking patterns */
glClearDepth( 1.0 );
glClear(GL_DEPTH_BUFFER_BIT);
draw( object, objectnum );
```

The `draw` function and associated OpenGL graphics routines are contained in the file `draw_object.c.` In the `draw` function the 3x4 matrix contained in object[k].trans is converted to an array of 16 values, `glpara`, using the function call `argConvGlpara.` The glpara array is then passed to the draw_object function. These sixteen values are the position and orientation values of the real camera, so using them to set the position of the virtual camera causes any graphical objects to be drawn to appear exactly aligned with the corresponding physical marker.

In the `draw_object` function the virtual camera position is set using the OpenGL function `glLoadMatrixd(gl_para)`. Different graphical objects are then drawn depending on which marker card is in view, such as a cube for the pattern named “cube” and a cone for the pattern named “cone”. The relationship between the patterns and the virtual objects shown on the patterns is determined in the `object_data` file in the `bin/Data` directory.

The steps mentioned above occur every time through the main rendering loop. While the program is running mouse events are handled by the mouseEvent function and keyboard events by the `keyEvent` function.

cleanup

The cleanup function is called to stop the video processing and close down the video path to free it up for other applications. This is accomplished by using the `arVideoCapStop(), arVideoClose()` and `argCleanup()` routines.

5.2 Recognizing different patterns

The `simpleTest` program uses template matching to recognize the different patterns inside the marker squares. Squares in the video input stream are matched against pre-trained patterns. These patterns are loaded at run time and are contained in the `Data` directory of the `bin` directory. In

this directory, the text file `object_data` specifies which marker objects are to be recognized and the patterns associated with each object. The `object_data` file begins with the number of objects to be specified and then a text data structure for each object. Each of the markers in the `object_data` file are specified by the following structure:

Name
Pattern Recognition File Name
Width of tracking marker

For example the structure corresponding to the marker with the virtual cube is:

#pattern 1
cone
Data/hiroPatt
80.0

Note that lines beginning with a # character are comment lines and are ignored by the file reader.

In order to change the patterns that are recognized the `sampPatt1` filename must be replaced with a different template file name. These template files are simply a set of sample images of the desired pattern. The program to create these template files is called `mk_patt` and is contained in the `bin` directory. The source code for `mk_patt` is in the `mk_patt.c` file in the `util` directory.

To create a new template pattern, first print out the file blankPatt.gif found in the `patterns` directory. This is just a black square with an empty white square in the middle. Then create a black and white or color image of the desired pattern that fits in the middle of this square and print it out. The best patterns are those that are asymmetric and do not have fine detail on them. Figure 6 shows some possible sample patterns. Attach the new pattern to the center of the blank square.

Fig 6: Possible Sample Patterns.

Once the new pattern has been made, change to the `bin` directory and run the `mk_patt` program. You will be prompted to enter a camera parameter filename. Enter the filename camera_para.dat. This is the default camera parameter file.

```
ARToolKit2.32/bin/mk_patt
Enter camera parameter filename: camera_para.dat
```

The program will then open up a video window as shown in figure 7.

Fig 7: mk_patt Video Window

Place the pattern to be trained on a flat surface in similar lighting conditions as will exist when the recognition application will be running. Then hold the video camera above the pattern, pointing directly down at the pattern, and turn it until a red and green square appears around the pattern (figure 7). This indicates that the `mk_patt` software has found the square around the test pattern. The camera should be rotated until the red corner of the highlighted square is the top left hand corner of the square in the video image, as shown in Figure 7. Once the square has been found and oriented correctly hit the left mouse button. You will then be prompted for a pattern filename. Once a filename has been entered a bitmap image of the pattern is created and copied into this file. This is then used for the ARToolKit pattern matching. Once one pattern has been trained, others can be trained simply by pointing the camera at new patterns and repeating the process, or the right mouse button can be hit to quit the application. In order to use the new pattern files they need to be copied to the `bin/Data` directory and the `object_data` file edited to associate the patterns with virtual objects.

5.3 Other Sample Programs

The `simpleTest` program showed how the ARToolKit could be used to develop applications that overlay virtual images on real world objects. In the `bin` directory there are three other sample programs; `exview, opticalTest` and `simplest`. These programs use the same marker patterns as the `simpleTest` application.

Exview

The exview application allows you to see an external view of the camera as it is being tracked. This is useful to check for tracking accuracy. This application also shows how the ARToolKit libraries can be used to find a quaternion representation of the viewpoint orientation. In the simpleTest example the camera viewpoint was set using matrix transformations. This is often difficult to do correctly so many 3D graphics applications use quaternion representations. ARToolKit also

supports quaternion output, enabling the tracking routines to be integrated into other rendering packages.

To run exview … [PUT MORE HERE]

OpticalTest

[PUT MORE HERE]

Simplest

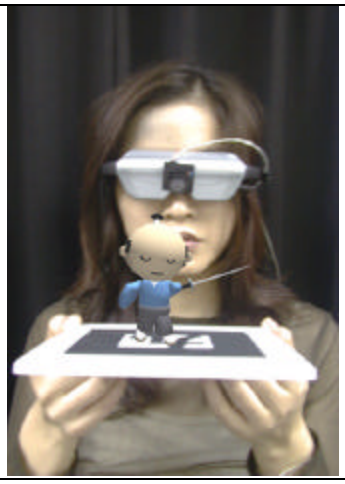
[PUT MORE HERE]

6 Head Mounted and Optical See-through Augmented Reality

ARToolKit uses computer vision techniques for image-based recognition and tracking. Since it uses only a single camera, a self-contained head tracking system can be developed if this camera is fixed to a head mounted display (HMD). In this way AR applications can be developed which use head mounted displays. Figure 8 shows two different HMDs with cameras attached, the Virtual i-O i-glasses and the Olympus EyeTrek displays. A more extensive list of HMD manufacturers and their websites in included in Appendix XX.

It is not necessary to have a head mounted display to use the ARToolKit, a camera connected to a computer is the only requirement. Without an HMD ARToolKit applications can be viewed on a computer monitor, with an HMD a more immersive experience can be created.

Virtual i-O i-glasses with camera



Olympus EyeTrek with camera

Figure 8 Different HMDs with Cameras Attached.

The ARToolKit programs shown so far have used video see-through augmented reality where the computer graphics are overlaid on video of the real world. Both the i-glasses and EyeTrek support NTSC video input, so a simple way to achieve video see-through AR is to connect the head mounted camera to the computer running the ARToolKit application, display the AR application on-screen and then use a VGA-NTSC converter to convert the monitor output to a signal that can be displayed back in the HMDs. The SGI O2 computer has NTSC input and output so these HMDs can be connected directly to the computer without the need of a converter.

ARToolKit also supports optical see-through augmented reality. In this case an optical see-through HMD is used and only the virtual images are shown in the HMD. Since the HMD is see-through the effect is that the virtual images are overlaid directly on the real world. The i-glasses are semi-transparent and so can be used for optical see-through AR, however the EyeTrek displays are fully occlusive.

There are advantages and disadvantages of optical versus video see-through AR. Optical see-through AR allows you to see virtual objects stereoscopically, rather than with bioccular viewing as

with a single camera video see-through system. The real world is also perceived at the resolution of the eyes rather than the resolution of the display. The main drawback with optical see-through AR is the lag in the system. In a video see-through application the video frame of the real world that is shown in the HMD is the same video frame that is used to calculate the head position and orientation so the virtual image will appear exactly overlaid on the real world scene. This is achieved by not showing the video frame until the image processing is completed. However in an optical see-through application the view of the real world cannot be delayed, so the delay introduced into the system by the graphics and image processing is perceived by the user. This results in virtual images that may not appear attached to the real objects they are supposed to be registered with, or that "swim" around. These lag effects are particularly noticeable with rapid head motions or moving the object the virtual image is supposed to be attached to.

For a full discussion of the advantages and disadvantages of optical and video see-through AR, please see Ron Azuma's Siggraph 95 course notes on Augmented Reality; available on the web at http://epsilon.cs.unc.edu/~azuma/azuma_AR.html

7.1 Optical See-through Calibration

In order to achieve good stereoscopic viewing in the optical see-through configuration it is necessary to measure the exact position of both the user's eyes relative to the head worn camera. This is achieved by a simple self-calibration method in which the user holds a calibration pattern in front of each eye in turn and lines up the pattern with a virtual cross hair shown in the see-through display. To achieve good stereo viewing eye position calibration should be done each time a user starts the AR application.

The calibration pattern used is shown in figure 9, and can be found in the pdf file clibPatt.pdf in ARToolKit2.32/patterns/calibPatt.pdf. This should be printed out and mounted on a piece of cardboard with enough white space around it so that it can be easily held in one hand. Note that the length of each side of black square of the calibration pattern must be 80mm.

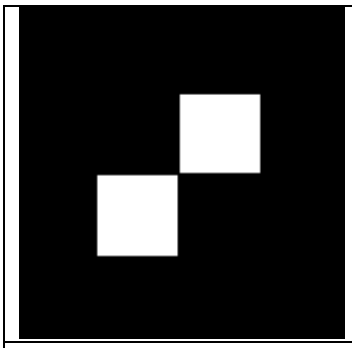


Figure 9 The Calibration Pattern

Figure 10 The User Calibrating Their Eye Position

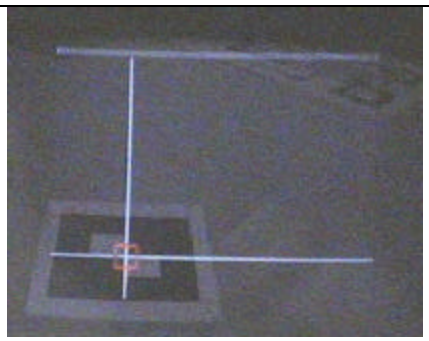


Figure 11 Lining up the Calibration Pattern

The `opticalTest` program contains code for calibration so using this as an example the calibration steps are as follows:

1. Place the HMD firmly on your head.
2. While `opticalTest` is running hit the right mouse button. A white cross should appear on-screen.
3. Shut your right eye, hold the calibration card in your left hand at arms length and use your right hand to work the mouse (figure 10).
4. Move the calibration card until a red or green square appears at the intersection of the white cross hairs. This means that the calibration card is in view of the head mounted camera and it's pattern has been recognized.
5. Keeping the right eye shut and the red or green square active move the calibration card and line up the white cross hairs with the intersection of the squares on the card (figure 11).
6. Click the left mouse button. If the calibration parameters are captured the red or green square will change to the opposite color.
7. Keeping the right eye shut move the calibration card as close as possible to your viewpoint which keeping the red or green square active and repeat steps 5 and 6.
8. After capturing a calibration point at arms length and close to the body, the cross hairs should change position – keeping your right eye shut repeat steps 3 through 7. There are 5 calibration positions so ten measurements are taken for each eye.

After taking ten measurements for the left eye, the white cross hairs should be back in the original position. Now shut your left eye and repeat steps 3 through 8 to capture ten measurements for the right eye.

After all 20 measurements have been taken the calibrated eye position will be printed out, looking something like this:

[INSERT MORE HERE]

```
71 grof@nekilstas: ~/ARToolKit/ARToolKit2.11.sgi/bin > simpleTest
No.1:                  cube
No.2:                  cone
No.3:                 torus
No.4:                sphere
Image size (x,y) = (720,486)
*** Camera Parameter ***
-------------------------------------
SIZE = 720, 486
Distotion factor = 360.000000 292.500000 86.913580
595.25376 0.03134 363.93197 0.00000
0.00000 598.38436 290.73261 0.00000
0.00000 0.00000 1.00000 0.00000
-------------------------------------
72 grof@nekilstas: ~/ARToolKit/ARToolKit2.11.sgi/bin > simpleTest
No.1:                  cube
No.2:                  cone
```

```
No.3:                torus
No.4:               sphere
Image size (x,y) = (720,486)
*** Camera Parameter ***
-------------------------------------
SIZE = 720, 486
Distotion factor = 360.000000 292.500000 86.913580
595.25376 0.03134 363.93197 0.00000
0.00000 598.38436 290.73261 0.00000
0.00000 0.00000 1.00000 0.00000
-------------------------------------
*** HMD Parameter ***
LEFT
-------------------------------------
SIZE = 640, 480
Distotion factor = 320.000000 240.000000 0.000000
838.82820 -16.62518 253.09900 126950.60781
23.87198 -844.98218 6.10683 8732.17217
0.15221 0.05533 0.50958 100.00000
-------------------------------------
Right
-------------------------------------
SIZE = 640, 480
Distotion factor = 320.000000 240.000000 0.000000
1038.90204 135.22789 347.46085 86058.02704
54.04275 -919.99055 30.86466 -3385.23435
0.42853 0.37057 0.68346 100.00000
-------------------------------------

73 grof@nekilstas: ~/ARToolKit/ARToolKit2.11.sgi/bin >
```

In the application window the white cross hairs disappear and simple.exe will continue running in optical-see through mode. The application can be switched between optical and video-see through modes by hitting the left mouse button. In the optical see-through mode the screen will appear black with just the virtual images being shown. The Virtual I-O I-glasses use a field sequential stereo signal that can be produced in OpenGL by interleaving the right and left viewpoints in the image plane. This produces an on-screen image like that shown in figure 12. In the HMD this image would be seen as a 3D red sphere. Other rendering techniques may be needed for other types of see-through HMDs. For a field sequential HMD you should do the optical see-through calibration with the right eye first and then the left eye second. If you can see a hair cursor in only one eye at a time, then the calibration may be carried out normally.

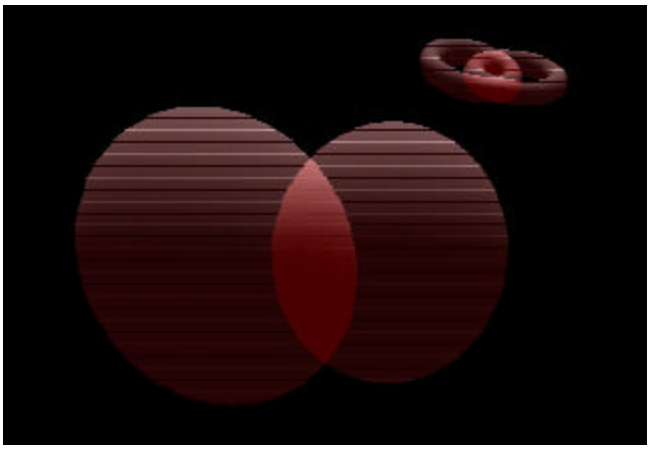


Figure 12. Optical See-through Stereo Image

If the calibration has been performed well then the 3D virtual objects should be exactly aligned with the markers used in the `opticalTest` application. However, if there was a calibration error, or if someone is using the application when it has been calibrated for another person's eyes then the 3D objects may appear to float above or below the cards or may be distorted.

Although the calibration process may appear quite time consuming, with practice it can be completed in just a couple of minutes and should produce good stereo viewing results.

In addition to calibrating for the user's eye position, it may be necessary to calibrate for the optical properties of the camera and HMD. The method for doing this is detailed in the next section.

The actual code for calibration the eye position is found in the `mouseEvent` function in `opticalTest.c`. When the user hits the right mouse button the following code is executed:

```
/* turn on and off the debug mode with middle mouse */
if( button == GLUT_RIGHT_BUTTON  && state == GLUT_DOWN ) {
  argCalibHMD( target_id, thresh, calibPostFunc );
}
```

The `argCalibHMD` function runs the entire calibration process and takes as arguments:

target_id: the id of the calibration pattern being used
thresh: the current image threshold value
calibPostFunc: a pointer to a function that is executed after the calibration is completed.

The function `calibPostFunc` is defined in `opticalTest.c` and merely checks to make sure the calibration procedure hasn't returned incorrect values and prints out the left and right eye HMD parameters (*lpara, rpara*). The full definition of `argCalibHMD` and the calibration function it calls can be found in the `gsub.c` file in the `ARToolKit2.11.win32/lib/Src/Gl` directory.

8 Camera Calibration

In the current ARToolKit software default camera properties are contained in the camera parameter file, `camera_para.dat,` that is read in each time an application is started. The parameters should be sufficient for a wide range of different cameras. However using a relatively simple camera calibration technique it is possible to generate a separate parameter file for the specific cameras that are being used. In a video-see through AR interface, if the camera parameters are known then the video image can be warped to remove camera distortions.

In this section we describe how to use the ARToolKit camera calibration routines. In order to use these routines the pattern files calib_cpara.pdf and calib_dist.pdf should be printed out. These are found in the `ARToolKit2.33/patterns` directory. The calib_cpara.pdf file is a grid of lines and should be scaled so that the lines are exactly 40 mm apart. The calib_dist.pdf file contains a 6 x 4 dot pattern and should be scaled so the dots are 40 mm apart. A copy of these images is also contained in Appendix XX. Once the files are printed out they should be stuck to pieces of cardboard so they are flat and rigid. The figures below show these patterns as seen through the camera lens.

Fig 7.1: The calib_dist pattern

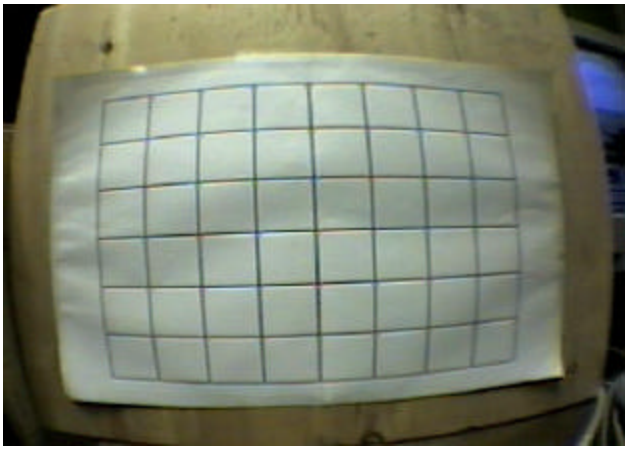


Fig 7.2: The calib_cpara pattern

The important camera properties that must be measured include the center point of the camera image, the lens distortion and the camera focal length. The program `calib_dist` is used to measure the image center point and lens distortion, while `calib_param` produces the other camera properties. Both of these programs can be found in the `bin` directory and their source is in the `utils/calib_dist` and `utils/calib_cparam` directories.

The calib_dist should be run first and them calib_cparam, since calib_cparam uses the output of calib_dist. In the remainder of this section we explain who to run each of these programs.

Running calib_dist

Calib_dist uses the calib_dist.pdf image of a pattern of 6x4 dots spaced equally apart. When viewed through the camera lens, lens distortion causes a pin cushion effect that produces uneven spacing

between the dots in the camera image (see figure 7.1). The `calib_dist` program measures the spacing between the dots and uses this to calculate the lens distortion.

To run the software, plug the camera you want to calibrate into the video-in port on the SGI O2 then type `calib_dist` at the command prompt:

```
grof@nekilstas: ~/ARToolKit/ARToolKit2.32a/bin > calib_dist
Image size (x,y) = (720,486)

-----------
Mouse Button
 Left   : Grab image.
 Right  : Quit.
-----------
```

A window will appear showing live video. Point the camera at the calibration pattern so that all the dots are in view and click the left mouse button. This freezes the video image, as shown in figure 7.1. Now click with the left mouse button on the image and drag (holding the mouse button down) a black rectangle over each dot in the image in turn. Starting with the dot closest to the top left hand corner of the image and continue until all the dots are found. The dots must be covered in the following order:

```
 1  2  3  4  5  6
 7  8  9 10 11 12
13 14 15 16 17 18
19 20 21 22 23 24
```

After each rectangle is drawn image processing software will find the dot enclosed by the rectangle and place a red cross at its center. If a red cross does not appear redraw the rectangle until the dot is found. Figure 7.2 shows a user drawing a rectangle over one of the last dots.

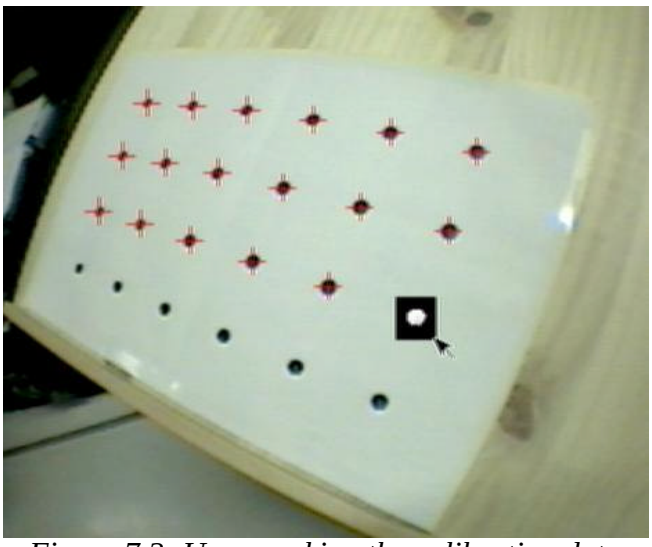


Figure 7.2: User marking the calibration dots.

Once an image is captured, while each dot is being found the following will appear on-screen:

```
-----------
Mouse Button
 Left   : Rubber-bounding of feature. (6 x 4)
 Right  : Cansel rubber-bounding & Retry grabbing.
-----------
 # 1/24
 # 2/24
 # 3/24
 # 4/24
 # 5/24
 # 6/24
 # 7/24
 # 8/24
 # 9/24
 # 10/24
 # 11/24
 # 12/24
 # 13/24
 # 14/24
 # 15/24
 # 16/24
 # 17/24
 # 18/24
 # 19/24
 # 20/24
 # 21/24
 # 22/24
 # 23/24
 # 24/24
```

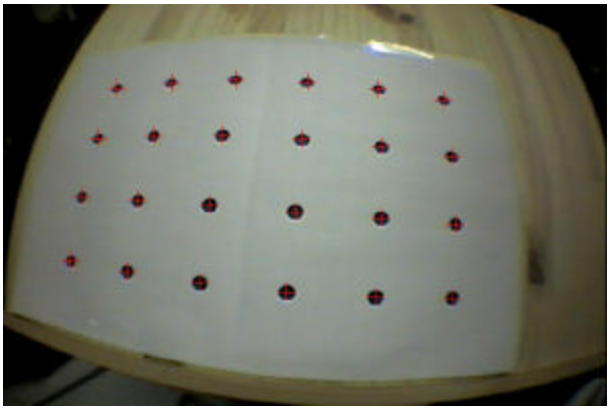
Once all 24 dots in the image have been found click the left mouse button again. This will store the position of the dots and unfreeze the video image.

```
-----------
Mouse Button
 Left   : Save feature position.
 Right  : Discard & Retry grabbing.
-----------

### No.1 ###
 1,  1: 125.01, 102.84
 2,  1: 198.73,  96.19
 3,  1: 283.00,  94.30
 4,  1: 369.78,  99.93
 5,  1: 448.78, 110.33
```

```
 6,  1: 514.39, 123.37
 1,  2: 118.84, 173.96
 2,  2: 192.13, 171.33
 3,  2: 277.61, 171.27
 4,  2: 366.40, 175.28
 5,  2: 446.74, 181.88
 6,  2: 512.50, 189.64
 1,  3: 119.86, 246.72
 2,  3: 191.37, 248.83
 3,  3: 274.59, 251.42
 4,  3: 361.36, 253.61
 5,  3: 440.32, 255.61
 6,  3: 505.38, 257.05
 1,  4: 127.78, 313.80
 2,  4: 196.05, 319.71
 3,  4: 272.48, 327.11
 4,  4: 355.03, 325.72
 5,  4: 430.25, 324.01
 6,  4: 493.18, 320.03
```

You should now take another image and repeat the process for 5-10 images from various angles and positions. The more images taken the more accurate the calibration. The figures below show typical sample images.



Once you have taken 5-10 images, click the right mouse button to stop the image capture and start calculating the camera distortion values.

```
-----------
Mouse Button
 Left   : Grab next image.
 Right  : Calc parameter.
-----------
[360.0, 243.0, 174.0] 596.018289
[360.0, 243.0, 174.0] 596.018289
[360.0, 243.0, 174.0] 596.018289
```

```
[360.0, 243.0, 174.0] 596.018289
[360.0, 243.0, 174.0] 596.018289
[360.0, 243.0, 174.0] 596.018289
[360.0, 243.0, 174.0] 596.018289
[330.0, 223.0, 201.0] 590.288659
[330.0, 228.0, 201.0] 486.692482
[330.0, 233.0, 201.0] 400.390511
[325.0, 238.0, 201.0] 330.137494
[325.0, 243.0, 201.0] 276.447160
[325.0, 248.0, 201.0] 241.422442
[325.0, 253.0, 201.0] 227.895132
[325.0, 253.0, 201.0] 227.895132
[325.0, 253.0, 201.0] 227.895132
[325.0, 253.0, 201.0] 227.895132
[325.0, 253.0, 201.0] 227.895132
[325.0, 253.0, 201.0] 227.895132
[325.0, 253.0, 201.0] 227.895132
[325.0, 253.0, 201.0] 227.895132
[325.0, 253.0, 201.0] 227.895132
[325.0, 253.0, 201.0] 227.895132
[325.0, 253.0, 201.0] 227.895132
[325.0, 253.0, 201.0] 227.895132
[324.0, 253.5, 201.0] 227.334239
--------------
Center X: 324.000000
       Y: 253.500000
Dist Factor: 201.000000
--------------
-----------
Mouse Button
 Left   : Check fittness.
 Right  :
   1/10.
-----------
```

It may take over a minute on an SGI O2 to calculate these parameters, please be patient. The center x and y values and distortion factor are the final key values generated by the calib_dist code. These values will be different for every camera and should be written down for use in the `calib_cparam` program.

In order to check that these parameters are correct click the left mouse button again. This will show the first image grabbed with red lines drawn through the calibration dots. These lines should pass through the center of each of these dots (see figure 7.3). Each time the left mouse is clicked the next image grabbed will be shown. Figure 7.4 shows another example of a final image.

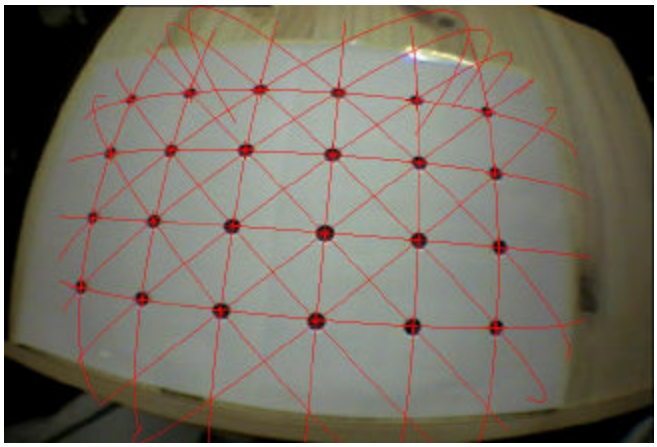


Figure 7.3: Sample Calibrated Image

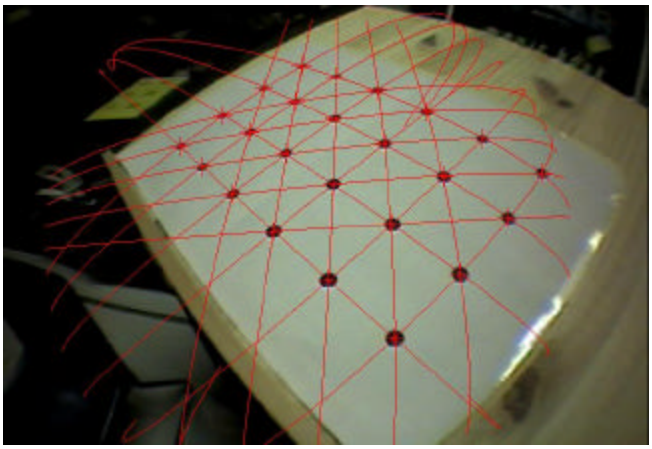


Figure 7.4 Sample Calibrated Image

Once you are satisfied with the results from `calib_dist` hit the right mouse button to quit and run the `calib_cparam` code.

Running calib_cparam

Calib_cparam is used to find the camera focal length and other parameters. It uses the pattern contained in calib_cparam.pdf, a grid pattern of 7 horizontal lines and 9 vertical lines (see figure 7.2). This pattern should be printed out and glued to a piece of cardboard or other rigid board.

Calib_cparam is run as follows:

1) Type `calib_cparam` at the command prompt and input the center coordinates and distortion ratio found from `calib_dist`:

```
grof@nekilstas: ~/ARToolKit/ARToolKit2.32a/bin > ./calib_cparam
Input center coordinates: X = 324
                        : Y = 253
Input distotion retio: F = 201
Image size (x,y) = (720,486)
```

A live video window will appear.

2) Put the calibration board in front of the camera so the board is perpendicular to the camera, all of the grid lines are visible and the grid is as large as possible (see figure 7.2).

3) Click the left mouse button to grab the image. A white horizontal line will appear overlaid on the image.

4) Move the white line to overlay the top black grid line as close as possible. The line is moved up and down using the up and down arrow keys, while it is rotated clockwise and anticlockwise using the left and right arrow keys. Once the white line is aligned with the top grid line hit the enter key. This line will now turn blue and another white line will appear (see figure 7.4). This process should then be repeated for all the horizontal lines.

Once the last horizontal line has been placed a vertical white line will appear and the process should be repeated for the vertical lines. The first vertical white line should be placed over the left most grid lines and the other lines placed from left to right (see figure 7.5).

The ordering of lines is very important. They should be placed from top to bottom and then from left to right until all 16 lines have been drawn on the screen.

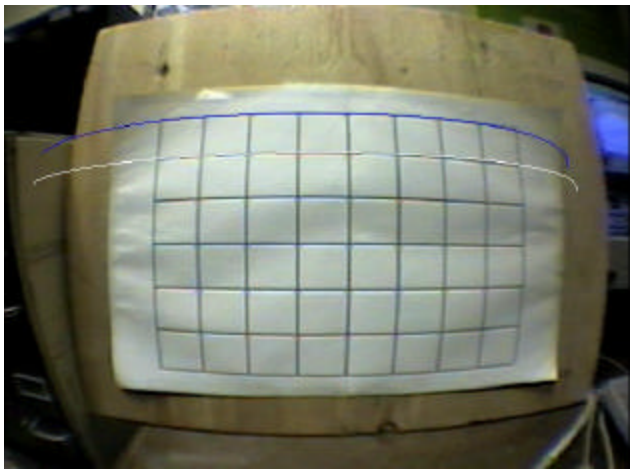


Figure 7.4: Horizontal Line Placement

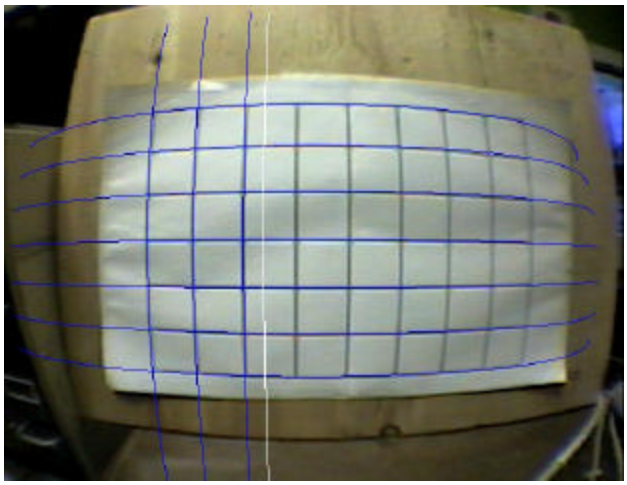


Figure 7.5: Vertical Line Placement

5) Once this process has been completed for one image, the grid pattern should be moved 100mm away from the camera (keeping the camera perpendicular to the pattern) and the process repeated again. Figure 7.6 shows the last line placement for one of these captured images.

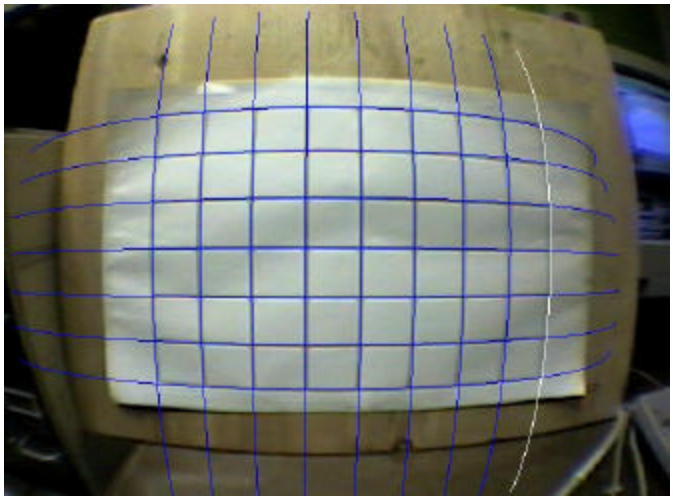


Figure 7.6: Final Line Placement

6) Repeat the process five times, moving the calibration pattern a total of 500mm away from the camera. After the fifth calibration step the program will automatically calculate the camera parameters. You will be prompted for a file name to store these parameters in:

```
point_num = 315
```

```
-------------------------------------
SIZE = 720, 486
Distotion factor = 324.000000 253.000000 201.000000
372.97979 -24.50134 248.86941 0.00000
0.00000 327.03507 122.42421 0.00000
0.00000 0.00000 1.00000 0.00000
-------------------------------------
Input filename: wideAngleKeyence.dat
```

Once the values are stored in a camera data file hit the right mouse button to quit.

By changing the name of this camera data file to camera_para.dat and placing it in the bin/Data directory it can be used immediately in the ARToolKit sample programs. Calibrating your camera should produce improved tracking results.

The distance between the grid lines in the calib_cparam.pdf pattern is exactly 40mm, while the pattern has to be moved back 100mm for each measurement and the measurements have to be repeated five times. These values are all fixed in the source code `calib_cparam_sub.c` in the `util/calib_cparam` directory.

If you want to change the distance between grid lines the following source code should be modified:

```
inter_coord[k][j][i+7][0] =   40.0*i;
inter_coord[k][j][i+7][1] =   40.0*j;
inter_coord[k][j][i+7][2] =  100.0*k;
```

where 40.0 is the current distance, and the 100.0 is the distance the pattern should be moved back from the camera each time. The number of measurements that need to be taken can be modified by changing the variable:

```
*loop_num = 5;
```

8 Limitations of Computer Vision-Based AR

There are some limitations to purely computer vision based AR systems. Naturally the virtual objects will only appear when the tracking marks are in view. This may limit the size or movement of the virtual objects. It also means that if users cover up part of the pattern with their hands or other objects the virtual object will disappear.

There are also range issues. The larger the physical pattern the further away the pattern can be detected and so the great volume the user can be tracked in. Table 1 shows some typical maximum ranges for square markers of different sizes. These results were gathered by making maker patterns of a range of different sizes (length on a side), placing them perpendicular to the camera and moving the camera back until the virtual objects on the squares disappeared.

Pattern Size (inches)	**Usable Range (inches)**
2.75	*16*
3.50	*25*
4.25	*34*
7.37	*50*

Table 1: Tracking range for different sized patterns

This range is also affected somewhat by pattern complexity. The simpler the pattern the better. Patterns with large black and white regions (i.e. low frequency patterns) are the most effective. Replacing the 4.25 inch square pattern used above, with a pattern of the same size but much more complexity, reduced the tracking range from 34 to 15 inches.

Tracking is also affected by the marker orientation relative to the camera. As the markers become more tilted and horizontal, less and less of the center patterns are visible and so the recognition becomes more unreliable.

Finally, the tracking results are also affected by lighting conditions. Overhead lights may create reflections and glare spots on a paper marker and so make it more difficult to find the marker square. To reduce the glare patterns can be made from more non-reflective material. For example, by gluing black velvet fabric to a white base. The 'fuzzy' velvet paper available at craft shops also works very well.

8 Potential Applications Using ARToolKit

ARToolKit provides the basic tracking capabilities that can be used to develop a wide range of AR applications. The Human Interface Technology Laboratory (HIT Lab) has been using ARToolKit to explore how augmented reality can be used to enhance face to face and remote conferencing. Figures 13 and 14 show two views of using AR for face to face collaboration. In this case several users gather around a table and each user is wearing a head mounted display with camera attached. On the table is a set of marker patterns with virtual objects attached. When the users look at the patterns they see these three dimensional virtual objects at the same time as their real environment and the other collaborations around the table. This seamless blend of real and virtual objects makes it very to easy for them to collaborate with their partners.

Fig 13: Face to face AR Collaboration

Fig 14: The View from Inside the HMD

The ARToolKit has also been used to support remote augmented reality conferencing. In this case a live virtual video image of a remote collaborator is shown on one of the marker cards. The enables video conferencing to move from the desktop computer out into the real world. Figure 15 shows the view of someone using the remote AR conferencing application.

Fig 15: Remote AR Conferencing using ARToolKit.

More information can be found out about these applications at the following web site:
http://www.hitl.washington.edu/research/shared_space/

9 Resources

A good introduction to Augmented Reality can be found in Ron Azuma's Siggraph 95 course notes; Course Notes #9: Developing Advanced Virtual Reality Applications, ACM SIGGRAPH (Los Angeles, CA, 6-11 August 1995), 20-1 to 20-38. These are available on the web at http://epsilon.cs.unc.edu/~azuma/azuma_AR.html

The i-glasses head mounted display is one of the most cost effective head mounted displays that supports both video and optical see-through AR. They are available from i-O display systems at http://www.i-glasses.com

Information about the Olympus Eye Trek displays can be found at http://www.eye-trek.com/

Small cameras suitable for mounting on the i-glasses display or other head mounted displays can be purchased from Marshall Electronics – http://www.mars-cam.com/

10 ARToolKit Library Functions

This section provides a partial listing of the external functions provided by ARToolKit. In this document we do not review the camera and HMD calibration functions because they are not needed in video-see through applications. However in the next release of ARToolKit these functions will be described.

The ARToolkit library consists of four packages found in the `lib` directory:

AR32.lib: the bulk of the ARToolKit functions, including routines for marker tracking, calibration and parameter collection.

ARvideoWin32.lib: a collection of video routines for capturing the video input frames. This is a wrapper around the Microsoft Vision SDK video capture routines.

ARgsub32.lib: a collection of graphic routines based on the OpenGL and GLUT libraries.

Strings32.lib: a collection of string processing routines.

Fig 16 shows the hierarchical structure of libraries. The exception to this structure is the argCalibHMD function in Argsub32.lib which uses both AR32.lib and ArvideoWin32.lib.

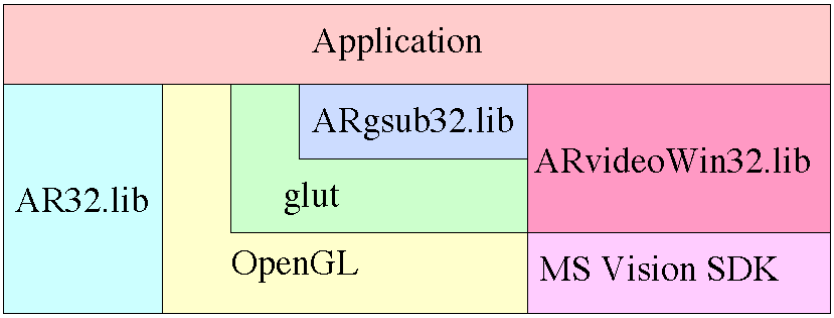


Fig 16:Hierarchical structure of libraries.

Basic Structures

Information about the detected markers is contained in the `ARMarkerInfo` structure defined in ar.h in the include directory.

```
typedef struct {
    int     area;
    int     id;
    int     dir;
    double  cf;
    double  pos[2];
    double  line[4][3];
    double  vertex[4][2];
} ARMarkerInfo;
```

In this structure `id` is the marker identity number, `cf` the confidence value that the marker has been correctly identified (between 0.0-1.0), `pos[2]` the center of the marker in ideal screen coordinates, `line[4][3]` the line equation for the four sides of the marker in ideal screen coordinates and `vertex[4][2]` the ideal screen coordinates of the marker vertices. For the line

equation `line[4][3]`, the three values `line[X][0]`, `line[X][1]`, and `line[X][2]` are the a,b,c values in the line equation ax+by+c = 0.

Augmented Reality Functions

In the library AR32.lib the following functions are commonly used in video-see through applications.

Functions for loading in initial parameters and trained patterns:

```
int arInitCparam( ARParam *param );
int arLoadPatt( char *filename );
```

Functions for detecting markers and camera position:

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

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

int arGetTransMat( ARMarkerInfo *marker_info,
              double pos3d[4][2], double trans[3][4] );

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

The remainder of this section describes these functions in a little more detail.

`int arInitCparam( ARParam *param );`

Function: to set the camera parameters specified in the camera parameter 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.
Variables: **param* – the camera parameter structure.
Returns: Always 0

`int arLoadPatt( char *filename );`

Function: to load the bitmap pattern specified in the file `filename` into the pattern matching array for later use by the marker detection routines.
Variables: **filename* – the 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.

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

Function: to detect the square markers in the video input frame.
Variables:
*ARUint8 *dataPtr* - a pointer to the color image which is to be searched for square markers. The pixel format is 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.
int thresh – specifies the threshold value (between 0-255) to be used to convert the input image into a binary image.
*ARMarkerInfo **marker_info* – a pointer to an array of ARMarkerInfo structures returned which contain all the information about the detected squares in the image.
*int *square_num* - the number of detected markers in the image.
Returns: 0 when the function completes normally, -1 otherwise.

`arDetectMarkerLite(..)` 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.

```
int arGetTransMat( ARMarkerInfo *marker_info,
                   double pos3d[4][2], double trans[3][4] );
```

Function: to 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.
Variables:
*ARMarkerInfo *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`.
double pos3d[4][2] - the physical vertex positions 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. For example, for a marker with sides 50mm long, and with the origin of marker coordinates in the center of the marker:

```
pos3d[4][2] = { {-25.0, -25.0},
                { 25.0, -25.0},
                     { 25.0,  25.0},
                {-25.0,  25.0} };
```

pos3d[4][2] is specified in the `object_data` parameter file read in at the program initialization.
double conv[3][4] - 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.

```
int arSavePatt( ARUint8 *image, ARMarkerInfo *marker_info,
                char *filename );
```
Function: used in `mk_patt` to save a bitmap of the pattern of the currently detected marker.
Variables:
*ARUint8 *image* – a pointer to the image containing the marker pattern to be trained.
*ARMarkerInfo *marker_info* – a pointer to the ARMarkerInfo structure of the pattern to be trained.
*char *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.

Video Input Functions

In the library ArvideoWin32.lib the following functions are commonly used:

```
int  arVideoOpen( void );
int  arVideoClose( void );

int  arVideoInqSize( int *x, int *y );
unsigned char *arVideoGetImage( void );
```

The video functions are defined in more detail below.

```
int  arVideoOpen( void );
```

Function: to open a video path by trying all the registered Windows video devices.
Returns: 0 if successful, -1 if a video path couldn't be opened.

```
int  arVideoClose( void );
```

Function: a function that needs to be called in order to shut down the video capture.
Variables: None
Returns: 0 if shut down successfully, otherwise –1.

```
int arVideoInqSize( int *x, int *y );
```

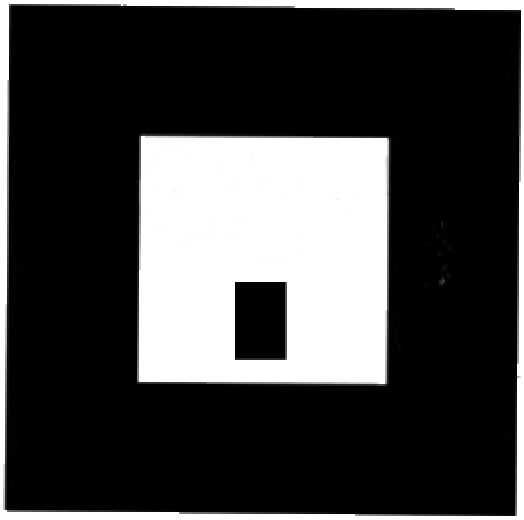
Function: a function that returns the size of the captured video frame.
Variables:
*int *x, *y* – a pointer to the width (*x*) and height (*y*) of the captured image.
Returns: 0 if the dimensions are found successfully, otherwise –1.

```
unsigned char *arVideoGetImage( void );
```

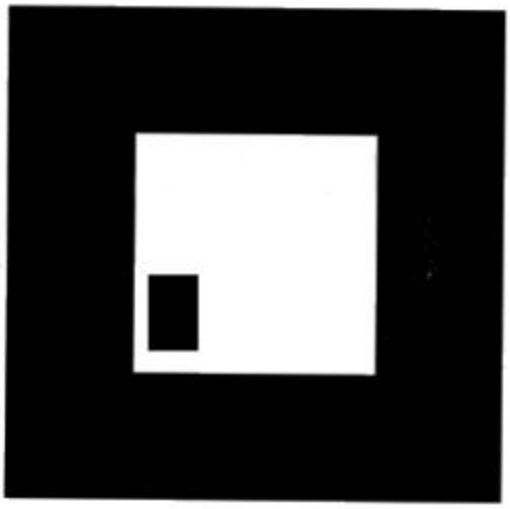
Function: to capture a single video frame.
Variables: None
Returns: A pointer to the captured video frame, or NULL if a frame isn't captured.

Sample Patterns

SampPatt1



SampPatt2



hiroPatt

kanjiPatt

Appendix A: GNU General Public License

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

NOTES

Some AR applications may use optical see-through head mounted displays. If this is the case, parameters representing camera-eye-HMD relationship are required for both of eyes. These parameters should be calibrated every time using a particular target pattern. The following lines of code reads the file of containing the pre-trained target pattern for this calibration. If optical see-through augmented reality is not used, these lines of code are not needed.

```
    if( (target_id = arLoadPatt(TARGET_PATT_FILE)) < 0 ) {
printf("Target pattern load error!!\n");
exit(0);
    }
```

The program `collide.exe` shows how interactions between AR objects can be performed. In this case the relative positions of the marked cards are tracked and when two cards are within a certain distance of each other there is an interaction between the virtual objects. The distance between the cards is calculated in the `draw` function in `collideTest.c.` When the two marks are within collision distance of each other the virtual objects on the marks change color to green.

Similarly `range.exe` shows an interaction between the camera and the virtual object. As the camera is brought closer to the physical marker the virtual object grows in size. The distance between the camera and object is simply found from the position and orientation matrix. The source code for these programs is found in the `examples/collide` and `examples/range` directories.

In the current ARToolKit software default camera properties are contained in the file `camera_para.dat` which is read in each time an application is started. The parameters should be sufficient for a wide range of different cameras. In the next version of ARToolKit we will distribute detailed information about how to measure camera and HMD optical properties for see-through applications.

Camera and HMD calibration is not necessary with video-see through AR. Here the user perceives the world through the same lens as the camera, so the camera distortions usually do not affect the AR tracking. The existing `camera_para.dat` data file should produce good tracking results for any camera and display.