Augmented reality with the ARToolKit

FMA175

title
version 1.3

Supervisor
Petter Strandmark

By
Olle Landin
lc07ol3@student.lth.se
**Introduction**

Augmented Reality (AR) is the overlay of virtual computer graphics images on the real world. One implementation of AR is called the ARToolKit, which is a library that has the functionality to quite accurately, in real time, calculate the user’s viewpoint, so that the virtual images can be exactly aligned with real world object. An example of AR in practice is when a football game is sending on TV and the offside line (and other useful information) is rendered on top of the images from the game.

The ARToolkit uses computer vision methods, which are constrained to track specially marked cards. This report will treat some of the mathematics behind the ARToolKit as well as presenting a demo application (for iPhone) created using the ARToolkit.

**Basic Principles**

Below I have broken down what an application using the ARToolkit does into five steps.

1. The camera captures video and feeds the video into the application.
2. Each frame is thresholded to black and white, and then gets searched for any square shapes.
3. If a square is found, computer vision algorithms are used to calculate the position of the camera relative to the black square.
4. The region inside the square is transformed using a homography and is then compared by template matching with patterns in a database. Thus the marker gets identified.
5. Once the position of the camera is known and the marker identified, a computer graphics model is drawn from that same position on top of the video. [1]
Identifying the marker

This section is an elaboration of the step 2 from above. The identification process begins with the input image being thresholded to a binary image and then the connected components are grouped using a labelling function. The next step is to find the contours of the image. When the contours are found, all contours that can be fitted by four line segments are extracted. The four vertices connecting the lines are also saved. From this information a homography between the input picture and the markers in the pattern database can be obtained. To calculate a homography at least four point-pairs are needed. The marker image is then transformed into the camera screen space and compared with the patterns in the database using template matching. [2] [4]

![original image](image1)
![thresholded image](image2)
![Connected components](image3)

![contours of the image](image4)
![detected edges and corners](image5)

**Figure 1. showing the different steps of the marker square identification.** [2]
Approximating the camera matrix

To be able to render 3D-objects so that it appears as if they are positioned on top of the marker, a camera matrix has to be approximated.

Figure 2. Graphical description of the coordinate systems for the camera screen, the camera and the marker. [5]

When approximating the camera matrix, we have to take three coordinate systems into consideration; the coordinate system for the camera screen (2D), the camera (3D) and the marker (3D). The camera is positioned in the origin of the camera coordinate system, looking along the Z-axis. Two axes of the marker coordinate system are parallel to the sides of the marker square. The camera- and marker- coordinates systems differ from each other with a rotation and a translation. The relation can be described with matrices as below:
The following relationship holds between the camera screen coordinates \((x_c, y_c)\) and the camera coordinates \((X_C, Y_C, Z_C)\):

\[
(Eq. 1) \quad \begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_1 \\ R_{21} & R_{22} & R_{23} & T_2 \\ R_{31} & R_{32} & R_{33} & T_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_M \\ Y_M \\ Z_M \\ 1 \end{bmatrix} = T_{CM} * \begin{bmatrix} X_M \\ Y_M \\ Z_M \\ 1 \end{bmatrix}
\]

\(C = \text{Camera}\) \(R = \text{Rotation}\) \(X, Y, Z\) - coordinates

\(M = \text{Marker}\) \(T = \text{Translation}\) \(T_{CM} = \text{Transformation matrix from C to M}\)

The following relationship holds between the camera screen coordinates \((x_c, y_c)\) and the camera coordinates \((X_C, Y_C, Z_C)\):

\[
(Eq. 2) \quad P = \begin{bmatrix} P_{11} & P_{12} & P_{13} \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} h x_c \\ h y_c \\ h \end{bmatrix} = \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}
\]

\(P = \text{Intrinsic camera parameters}\)

\((x_c, y_c)\) – coordinates of the projection on the camera screen

\((X_C, Y_C, Z_C)\) – 3D-coordinates of points in the camera coordinates.

The equations for the lines in the camera screen are known since the line-fitting step earlier. The Intrinsic camera parameters \((P)\) are known (more on that under the section **Intrinsic camera parameters**). The equations for two parallel lines (parallel in the world, not necessarily on the screen) from the marker contour can be described as follows:

\[
(Eq. 3) \quad a_1 x + b_1 y + c_1 = 0 \]
\[
a_2 x + b_2 y + c_2 = 0
\]

All 3D-points that will be projected to these two lines can be described by combining \(Eq. 2\) and \(Eq. 3\), which gives:

\[
(Eq. 4) \quad a_1 P_{11} x_c + (a_1 P_{12} + b_1 P_{22}) y_c + (a_1 P_{13} + b_1 P_{23} + c_1) z_c = 0
\]
\[
a_2 P_{11} x_c + (a_2 P_{12} + b_2 P_{22}) y_c + (a_2 P_{13} + b_2 P_{23} + c_2) z_c = 0
\]

**Normals for the two planes:**

\[
n_1 = (a_1 P_{11}, a_1 P_{12} + b_1 P_{22}, a_1 P_{13} + b_1 P_{23} + c_1)
\]
\[
n_2 = (a_2 P_{11}, a_2 P_{12} + b_2 P_{22}, a_2 P_{13} + b_2 P_{23} + c_2)
\]
Figure 3. Graphical description of Eq. 4

The two planes in figure 3 can be interpreted as the planes containing all 3d-points that will be projected on to the two lines in the camera screen. The variables $d_1, d_2$ denotes the direction vectors of the markers sides. Since the marker has a square shape both $d_1$ and $d_2$ is the direction vector for two sides each. Since $d_1$ lie on each of the planes (see Figure 3), $d_1$ is perpendicular to $n_1$ as well as to $n_2$. This means $n_1 \times n_2 = d_1$. The direction vector for the other two sides of the marker can be calculated in the same way.

Figure 4. The direction vectors $u_1$ and $u_2$ won't be exactly perpendicular because of noise in the measurements.[4]

There will always be some noise involved in the process, so the $d_1, d_2$ won’t be exactly perpendicular; thus the $v_1, v_2$ vectors are used instead (note that the angle between $v_1, d_1$ is equal to the angle between $v_2, d_2$). From $v_1, v_2$, we get $v_3$ by taking the cross product ($v_1 \times v_2$). Together $v_1, v_2, v_3$ form
the rotation component in $T_{CM}$. The only parameters missing now is $w_1, w_2, w_3$, but these can be obtained simply by solving the system of equation obtained by the relationship between the four vertices coordinates of the marker in the marker coordinate frame and those coordinates in the camera screen coordinate frame.

To get better results the four vertices are now transformed to camera screen coordinates using the obtained transformation matrix and then compared to the measured values of the vertices projections. The transformation matrix is then refined by optimizing the sum of differences between these transformed coordinates and the coordinates measured on the screen (see eq. 5).

By iteration of this process a number of times the transformation is more accurately found. It would be possible to deal with all of the six independent variables in the optimization process. However, computational cost has to be considered. [3][4]

(Eq. 5) \[ \text{err} = \frac{1}{4} \sum_{i=1,4} \{(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \} \]

$(\hat{x}_i, \hat{y}_i)$ – Camera screen coordinates of the transformed vertices

$(x_i, y_i)$ · Camera screen coordinates of the marker corners measured in the picture

### Intrinsic camera parameters

(Eq. 6) \[ P = \begin{bmatrix} s_x f & 0 & x_0 \\ 0 & s_y f & y_0 \\ 0 & 0 & 1 \end{bmatrix} \]

$f = \text{focal length},$

$s_x = \text{the scale factor [pixel/mm] in direction of } x \text{ axis}$

$s_y = \text{the scale factor in direction of } y \text{ axis},$

$(x_0, y_0) = \text{the position that } Z \text{ axis of the camera coordinates frame passes}$

The P-matrix is camera-dependent, and if the ARToolkit is used with a new camera, the camera has to go through a calibration step so that the software can calculate the intrinsic camera parameters [4]. The ARToolkit for iOS is custom made for the iPhone, so the calibration step is done in advance.
Limitations

The tracking markers must be in view at all times in order for the virtual objects to appear. This of course comes from that the tracking algorithm won’t find any markers if they aren’t in view. This limits the area a user can move around the camera and still get the AR-effect. There is also range issues; the distance between the camera and the pattern is limited, the range is also dependent on the pattern complexity (low frequency patterns can handle bigger ranges). The marker orientation relative to camera also affects the tracking, the more tilted the marker is in respect to the camera, the more unreliable the pattern recognition gets. How good the tracking is also depends on the lighting conditions, the video shouldn’t be over- or underexposed but is also important to avoid glare effects from the markers, since it will aggravate the tracking.[1]
Demo

In addition to this report a simple demo, using the ARToolkit for iOS, has been created. The demo consists of a game in which the player’s goal is to get a table tennis ball into the middle of an archery target. The archery target is placed on top of the marker, and the ball is floating about in the air. When the user swipes across the iPhone screen the ball flies away in a direction, which depends on the direction of the swipe. A virtual wind exists in the virtual environment and the wind will affect where the ball ends up. The direction and strength of the wind is expressed by a number, where minus denotes a wind coming from the left and plus a wind coming from the right (this is left and right from the viewpoint of Figure 6). The challenge for the player is to get the ball flying away in a direction that with the help of the wind gets the ball to land in the center of the target.

Figure 5. The input from the camera on the iPhone.

Figure 6. The input from the camera on the iPhone with the AR-layer on top.
References

[1] How does ARToolKit work?. ARToolKit Documentation, 

[2] Computer Vision Algorithm. ARToolKit Documentation, 

[3] Inside ARToolKit by Hirokazu Kato, Hiroshima City University 

[4] Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System by Hirokazu Kato (Faculty of Information Sciences, Hiroshima City University) and Mark Billinghurst (Human Interface Technology Laboratory, University of Washington). 

[5] Coordinate Systems. ARToolKit Documentation, 