Artificial Neural Networks to Recognize ARToolKit Markers

Wnêiton L. Gomes¹, Celso Camilo¹, Leonardo Araújo Lima¹
Alexandre Cardoso ¹, Edgard Lamounier Jr¹, Keiji Yamanaka¹

¹Faculdade de Engenharia Elétrica, Universidade Federal de Uberlândia.
Uberlândia – MG, Brasil.
{wneiton@yahoo.com.br, celsocamilo@gmail.com, leoaraujolima@gmail.com}
alexandre,lamounier,keiji@ufu.br

Abstract

The objective of this work is to present the implementation of an Artificial Neural Network associated to ARToolKit for the recognition of markers, providing a less complex algorithm in comparison to the traditional algorithm used in the ARToolKit system.

1. Introduction

Among several computing techniques, Artificial Neural Networks (ANNs) are based on a computational model family inspired on human biologic neurons, where its main characteristic is the learning phase [1].

There are plenty application problems where an ANN can be applied, but they are mainly used to pattern classification and optimization [1].

In Computer Graphics, models of virtual environments using Virtual Reality (VR) and Augmented Reality (AR) techniques have received special attention due to its applications into Psychology, Education, Industrial processes, movies, entertainment etc [2, 3].

Augmented Reality is defined as a combination of a vision of the real environment associated to a virtual one [4, 5, 6]. In this technique, a video image is processed in such way that it is “augmented” with graphic scenes generated by computer [7]. Most of the low cost AR projects are developed using a set of computer libraries known as ARToolKit [8].

Associated to the capacity offered by ANNs, this work has the objective to improve ARToolKit’s ability to recognize markers (patterns) in a graphic image. Since the capture of a new scene and the insertion of new virtual objects must be performed in real time for AR techniques, there is a constant need to reduce computations time.

Thus, this paper proposes an implementation of a new ARToolKit library where ANNs are used to provide a better performance during the pattern recognition process.

This work is organized as follows: Section 2 presents basic concepts of the ARToolKit system and how marker recognition takes place. Section 3 introduces ANN concepts plus its time complexity algorithms. The conclusions of this work are presented in Section 4.

2. The ARToolKit System

The ARToolKit system is a C library developed at Washington University [1]. Through computer vision techniques, the system calculates the position and orientation of markers or patterns printed in cards. This position and orientation are captures by a digital camera and when the image is reprocessed, virtual 3D objects are placed on these markers in the new scene (see Figure 1) [1].

![Figure 1. ARToolKit main flow.](image)

The first stage in Figure 1 shows the identification of the markers. In this point, the captured image by the camera is entirely investigated in order to find boundaries that compose a square. Every found square is identified as a marker and must be registered as such. However, not every marker is set as valid and may represent no virtual object. In order to be recognized as a valid marker, the figure inside the square must be previously registered (Stage 2). When a marker is considered as valid, the virtual object associated to it is shown in the new
generated scene (Stage 3). If none marker is set as valid, the process to capture new markers is re-started from scratch. Next section, presents the ARToolKit’s recognition process in details.

2.1 ARToolKit’s Valid Markers Recognition

The entire ARToolKit’s recognition process of valid markers is ranged from the identification of the picture inside the square border to the association of virtual objects with these pictures. All steps to set a marker is valid can be seen in Figure 2.

![Figure 2. Markers Recognition Process](image1)

Step 1 is a previous execution phase, where the markers are registered. This previous registration is done by an ARToolKit’s module known as Mk-Patt, which generates a file with a set of 12 transformation matrices with 16x16 dimensions. Each matrix determines a position in the marker (translation or rotation). Thus, it does not matter how the marker is captured by the camera, since many of the positioning possibilities of the marker is previously defined in this file.

The matrices are filled by numbers that vary from 0 to 255, grey scale. Next, based on a limit number, it is defined which pixel in the image is black or white, composing a binary file. For example, if the limit number is equal to 100, every matrix element which is greater or equal to 100 is converted to 1 (white) and every matrix element less than 100 is converted to 0 (black).

The matrix file is organized as follows: there are four groups of three 16x16 matrices, making 12 matrices for each registered marker. Each group has three matrices. Each matrix is associated to a translation position. For each change in the group, there is a rotation associated to the marker. As an example, Figure 3 presents one matrix of each group. Considering a limit number of 100, the picture of the marker has been highlighted by its profile, so it’s possible to follow the movement of the picture by the matrices.

![Figure 3. (a,b,c,d) Matrix files](image2)

In Step 2 (Figure 2) a 16x16 matrix is also generated from the inside region of the square which refers to the marker captured by the camera at running time. This matrix is sent to the Match function (Step 3). The main responsibility of the Match function is to identify the valid marker along with to identify the possibility of this marker be previously registered. This step is very important when two similar markers are presented – to define which marker will be chosen as reference to show the virtual object. The marker with the highest possibility is the chosen one.

The identification done by Match function (Step 3), is achieved by comparing the previously registered matrix file (Step 1) and the matrix captured by the camera (running time) in Step 2. When similarity takes place, ARToolKit considers that one reference marker has been found.

3.0 Artificial Neural Networks

ANNs are computation techniques that present an inspired model in the neural structure of intelligent organisms and that they acquire knowledge through the experience, as a human being.

ANNs have been used in a lot of knowledge areas, for example, industry, businesses, finances and medicine, and among those areas the most common problems are the classification, prediction, patterns recognition and control.

As the neurons of the human brain, ANNs also possess a neurons structure, however, mathematical and artificial. Those neurons linked by synapse connections are divided in entrance neurons, internal neurons and exit neurons, for
which the net receives external signs, process the information and communicates with the exterior.

The most typical architectures are unique-layer, with entrance and exit neurons, and multi-layer, with entrance, internal and exit neurons.

3.1 ARToolKit’s Valid Markers Recognition

As it was seen in the item 2, ARToolKit has a stage (step 2) for valid patterns recognition that was described in the item 2.1. however, this work proposes the substitution of the used algorithm, for recognition, by a ANN (Figure 4), propitiating an acting earnings.

![Figure 4. Match (ARToolKit) substitution for ANN for valid markers recognition.](image)

ANN is implemented by a training phase and another of execution, therefore, it is necessary to differentiate what will be done in the pre-execution and what will be made in the ARToolKit execution (Figure 5). The chosen approach for the training phase was the supervised, where it is setting up a training group, formed by wanted entrances and exits, which are presented for the network. With that the net has to be capable to adjust the weights (connections), as comes the training group, so that in the execution phase it is capable to emit satisfactory exits for strangers entrances.

![Figure 5. Pre-execution (ANN Training) and ARToolKit Execution (Trained ANN) phases.](image)

3.2 ANN use in the valid markers recognition

As example, a scenery was created where was used three markers: Trace, Kaje and Hiro. This way, the net has 256 entrance neurons and three exit neurons. The technique used for training was the delta rule, the $\alpha$ used is 0.0039 and the stop condition is Quadratic Error or equal to 0.1. The $\alpha$ is the learning rate used in the net, that most of the time is randomly defined. The $\alpha$ defines the step size for the net learning. The Quadratic Error is the errors addition of each exit neuron.

In the pre-execution phase the markers in the Figure 6 were presented to Mk-Patt that created a file for each marker with the 16x16 matrices. These three files formed the training base that was presented to the ANN for learning. The ANN got the Quadratic Error of 0.09998 after 1745 cycles. The follow equation displays the Quadratic Error formula [1].

$$E = \frac{1}{2} \sum (tp - yp)^2$$  \hspace{1cm} (1)

Where:

- $E$ = Quadratic error
- $tp$ = Exit target (expected exit)
- $yp$ = Network exit
In the pre-execution phase the markers in the Figure 6 were presented to Mk-Patt that created a file for each marker with the 16x16 matrices. These three files formed the training base that was presented to the ANN for learning. The ANN got the Quadratic Error of 0.09998 after 1745 cycles and a sample of the found weights are presented in the Figure 7.

3.3 Results of the algorithms complexity analysis

The algorithm analysis is an important part of the algorithm complexity theory, that provides theoretical estimates of necessary resources for any computation problem. These estimates impel an algorithms efficiency improvement [9,10,11].

It was made the complexity analyses of the Match and ANN algorithms, and the execution time analysis of each line. The Figure 8 displays the Match of ARToolKit algorithms code.

```
1: static int pattern_match(ARUint8 *data, int *code, int *dir, double 3:*cf)
4: { 
5:   int input[768]; 
6:   int i, j, k, l; 
7:   int ave, sum, res, res2; 
8:   double datapow; 
9:   sum = ave = 0; 
10:  for( i = 0; i < 768 ; i++)
11:    ave += (255-data[i]);
12:    ave /= 768;
13:  for( i = 0;i < 768; i++)
14:    input[i] = (255-data[i]) – ave;
15:    sum += input[i]*input[i];
16:    datapow = sqrt((double)sum);
17:    if(datapow == 0.0){
18:      *code = 0;
19:      *dir = 0;
20:      *cf = -1.0;
21:      return -1;
22:    }
23:    res = res2 = -1;
24:    max = 0.0;
25:    k = -1;
26:for(1 = 0; 1 < m; 1++){
27:    k++;
28:    while(patf[k] == 0 ) k++;
29:    if( patf[k] == 2 ) continue;
30:    for( j = 0; j < 4; j++ )
31:    sum = 0;
32:    for( i = 0;i < 768; i++)
33:    sum += input[i]*pat[k][j][i];
34:    sum2 = sum/patpow[k][j]/datapow;
35:    if( sum2 > max )
36:    {max = sum2; res = j; res2 = k;}
37:  }
38: }
```

The Table 1.0 displays the execution time calculations of the Match algorithm.

<table>
<thead>
<tr>
<th>mPesosRna(Entrada,Saida)</th>
<th>Valor</th>
</tr>
</thead>
<tbody>
<tr>
<td>mPesosRna(3,0)</td>
<td></td>
</tr>
<tr>
<td>mPesosRna(0,1)</td>
<td></td>
</tr>
<tr>
<td>mPesosRna(1,0)</td>
<td></td>
</tr>
<tr>
<td>mPesosRna(1,1)</td>
<td></td>
</tr>
<tr>
<td>mPesosRna(2,0)</td>
<td></td>
</tr>
<tr>
<td>mPesosRna(2,1)</td>
<td></td>
</tr>
<tr>
<td>mPesosRna(1,0)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 6. Markers used in the ANN training.

Figure 7. Some of the adjusted weights during the training.

Figure 8. ARToolKit Match algorithm [8]
Table 1. Execution time calculations of the Match algorithm. Being, the Line Cost the number of executed operations in the same and the Repetition the time amount that the line is executed.

<table>
<thead>
<tr>
<th>Line</th>
<th>Line Cost</th>
<th>Repetition</th>
<th>Total Line Time (Line Cost * Repetition)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2</td>
<td>769</td>
<td>1538</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>768</td>
<td>1536</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>769</td>
<td>1538</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>768</td>
<td>1536</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>768</td>
<td>1536</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
<td>m + 1</td>
<td>2*m + 2</td>
</tr>
<tr>
<td>27</td>
<td>1</td>
<td>1*m</td>
<td>1*m</td>
</tr>
<tr>
<td>29</td>
<td>1</td>
<td>1*m</td>
<td>1*m</td>
</tr>
<tr>
<td>30</td>
<td>2</td>
<td>5*m</td>
<td>10*m</td>
</tr>
<tr>
<td>32</td>
<td>2</td>
<td>769<em>4</em>m</td>
<td>6152*m</td>
</tr>
<tr>
<td>33</td>
<td>2</td>
<td>768<em>4</em>m</td>
<td>6144*m</td>
</tr>
<tr>
<td>34</td>
<td>2</td>
<td>4*m</td>
<td>8*m</td>
</tr>
<tr>
<td>35</td>
<td>1</td>
<td>4*m</td>
<td>4*m</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Total execution time</strong> <em>(12322</em>m) + 7689*</td>
</tr>
</tbody>
</table>

The Figure 9 display the ANN algorithm code implemented for the markers recognition.

```java
1:static void RNA(float Entradas[256],
2:float *vSaida){
3:int k,i;
4:double yin;
5:vSaida= new float[m];
6:for(k = 0; k < m; k++){
7:yin = 0;
8:for(i = 0;i <= 256;i++){
9:yin=yin+(Entradas[i] * M[i][k]);
10:}
11: if (yin >= 0.5) vSaida[k] = yin;
12: else vSaida[k] = -1;}
```

Figure 9. ANN algorithm for ARToolKit markers.

The table 2.0 displays the execution time calculations of the ANN algorithm.

Table 2. Execution time calculations of the Match algorithm. Being, the Line Cost the number of executed operations in the same and the Repetition the time amount that the line is executed.

<table>
<thead>
<tr>
<th>Line</th>
<th>Line Cost</th>
<th>Repetition</th>
<th>Total Line Time (Line Cost * Repetition)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>2</td>
<td>m + 1</td>
<td>2*m + 2</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>257 * m</td>
<td>514 * m</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>256*m</td>
<td>512*m</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>m</td>
<td>1*m</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Total execution time</strong> <em>(1029 * m) + 2</em></td>
</tr>
</tbody>
</table>

The tables 1.0 and 2.0 are formed by 4 columns:
- Line: determinates the algorithm line that is being analyzed;
- Line Cost: specifies the cost based on the executed operations number in the line. And, each operation (addition, subtraction, multiplication, division, comparison) have cost equal to one. For example, the line 11 of the Figure 9, where an addition exists (ave + =) and a subtraction (255 - data[i]), have two costs. Reminding that, the attribution doesn't have cost;
- Repetition: amount of times that the line is executed;
- Total Line Time: it is the result of the equation Line Cost * Repetition.

Conclusions and Future Work

Based on the obtained results, we concluded that the algorithm using artificial neural networks recognizes the markers just as the recognition module used by ARToolKit. However, in spite of both algorithms present a time complexity of O(m), where m represents the number of markers, the proposed algorithm is faster (execution time) than the ARToolKit’s Match algorithm. Thus, the proposed algorithm provides a better performance in the recognition process of the markers, which is very important for real time AR applications.

As future work tests involving a larger number of markers will be taking into account in order to measure...
the performance of both algorithms in these situations. Furthermore, tests where the AR markers present imperfect conditions will be considered to evaluate the algorithm quality in the recognition process.

10. References


