Integrated Project: Vision controlled camera

Ana Catarina Macedo — a54773@alunos.uminho.pt
Miguel Palhas — pg19808@alunos.uminho.pt
Pedro Costa — pg19830@alunos.uminho.pt

University of Minho
Department of Informatics

Categories and Subject Descriptors:

Additional Key Words and Phrases: Computer vision, modelation and visualization, augmented reality

1. INTRODUCTION

In this document is presented the second stage of the Integrated Project module of the Computer Graphics Specialization Unit of the Master's Degree in Informatics Engineering.

In the previous stage, the `ARToolKit` library was used to identify a pattern in the image captured by a usual webcam. Using the same library, one could also retrieve the required transformation matrix to position an object in the same position of the pattern, thus achieving a basic augmented reality system.

For that stage, the goal was to extract the translation and rotation data from the transformation matrix, so the model could be manually positioned in the pattern.

The translation data did not present any obstacle, as it is very straight forward to extract.

The rotation data though, consisted in extracting the Euler angles from the 3×3 top-left submatrix. This was hard because of three things:

—Rotations and scales both take effect in this 3×3 submatrix. The problem was simplified by ignoring scales, as no method to remove them was found;

—Even assuming that every rotation is performed only around the coordinate system axes (to simplify), the final result depends on the order by which the rotations are applied. The `ARToolKit` library does not mention this in the documentation. This order was assumed to be the standard Heading-Pitch-Roll (Y-X-Z);

—Given the angles, the rotation matrix values are generated with trigonometric functions. The inverse functions return a smaller limited ranges of values for the angles, which results in a very unstable positioning of the model in the pattern.

Despite existing a function the `ARToolKit` API to retrieve the angles, this function was not documented. The angles retrieved from this function showed the same stability problems of the inverse trigonometric approach.

For this stage, given a camera in a virtual 3-D world, the goal is to capture the movement of the pattern and use it to change the camera's perspective accordingly.

2. IMPLEMENTATION

The implemented system consists on two distinct parts: the capture system, controlled by the `ARToolKit` library, and the virtual world where the camera to be controlled exists.

The capture system searches for the pattern, and then updates a matrix shared[1] by both parts. If a pattern is found, the transformation matrix is retrieved from the library and saved in the shared container. Otherwise, it is set to be the identity matrix. This sets the direction of the camera to the default when no pattern is found by the camera, effectively resetting the perspective to the initial state. An alternative would be to leave the matrix untouched in this case, leaving the camera in the same direction as was defined by the last found pattern.

The camera starts at the world coordinate system origin. It also has a direction vector (negative in the z axis, by default), which defines the point to look at and an "up" direction (positive in the y axis, by default). The direction vector allows the camera to follow the angles of the pattern both horizontally and vertically. The "up" vector, on the other hand, allows the camera to follow the rotation of the pattern in its own plane (if the pattern is upside down, the camera will reflect that). To note that due to the relation between the coordinate systems of the camera and the pattern (see Figure 1), the camera will have the correct up direction when the pattern is in fact upside down.

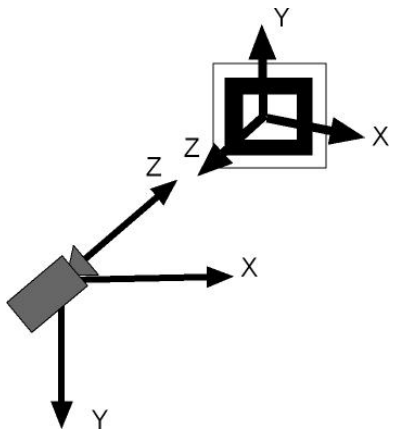

Fig. 1. The `ARToolKit` coordinate systems for the camera and the identified pattern.

To achieve the rotation of the camera, at the beginning of each frame the transformation matrix (T) is multiplied by the direction vector (d) and the up vector (u), generating the computed versions

[1]The concurrent accesses are controlled using *mutexes* from the `Boost` library.

(d' and u', respectively).[2]. Any translation of Δs space units to be performed since the computation of the last frame is applied to the position vector p along the computed direction d'. This same vector d' is then used to define the target position t at a distance of z space units from the camera. These computed values are the ones passed to the `lookAt` function.

$$
\begin{aligned}
d' &= T \times d \\
u' &= T \times u \\
t &= p + d' * z \\
p &= p + d' * \Delta s
\end{aligned}
$$

The camera control is tested in a scene with various teapots in a cube shape around the origin of the world coordinate system. To draw the teapots, different shaders are used to change the colors. Each teapot receives a color related to its position in the cube:

(1) **Red** for the teapots in the faces intersected by the x axis;
(2) **Green** for those in the faces intersected by the y axis;
(3) **Blue** for the ones in the faces intersected by the z axis.

The teapots in the edges receive the color from both faces they belong to, and the ones in the corners are drawn in white. All the shaders use Phong's algorithm lit from a point in space. The light and placement of the teapots help in recognizing the camera's movement, while the color system allows for some orientation while testing, so the viewer is able to distinguish the various possible movements.

3. CONCLUSION

This document presented the implementation of a 3-D world scene with a vision controlled perspective. The implemented system uses the `ARToolKit` library to recognize the pattern and applies its orientation to the virtual camera, successfully allowing the viewer to "look around" in the 3-D world.

Although not implemented in the current version, this system supports a navigation system which may or may not be independent of the pattern orientation. Using abstract vectors instead of only points, it becomes trivial to implement movement along the direction of the camera for example. Such features were not implemented only due to time constraints.

Before reaching the current stage, two distinct solutions were prepared to answer this stage of the Integrated Project, but each implemented a system different than the one intended. The reason for this to happen was the lack of a guiding example, or a text describing the assignment. Although the simplicity of the project allowed to keep correcting, the misinterpretations which arised from this resulted in a heavy time consumption preparing code and studying approaches which will be ignored in any evaluation process.

Any future work on this system will start by implementing the mentioned navigation system, as the only challenge is in deciding if and how the implemented vision system would interact with the navigation. The system can move in the direction the camera is looking, or both systems can be made completely apart allowing the viewer to change the camera's position while "looking around".

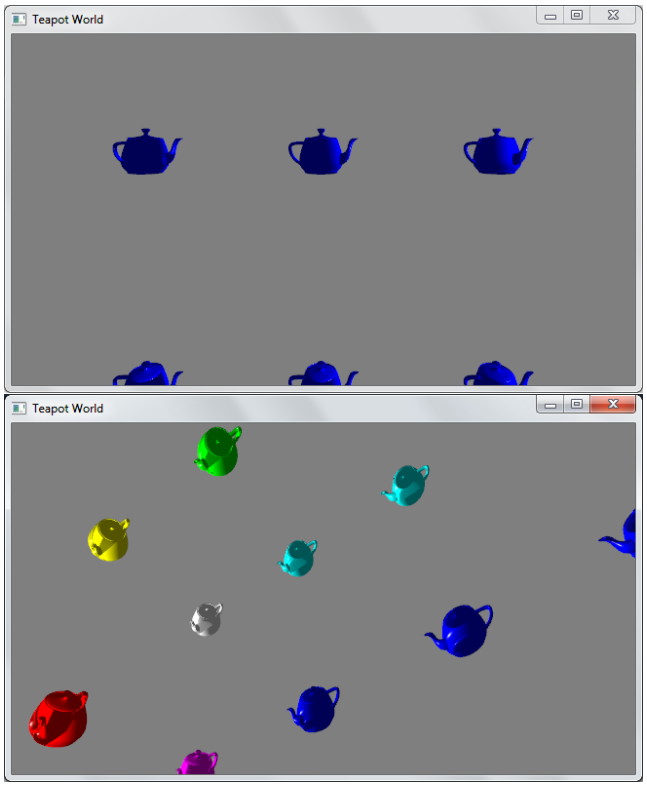

Fig. 2. Screenshots of the working application. On top, the default view (camera looking in the negative z direction); below, the view with the pattern pointing to the frontal top left corner of the cube (negative x, positive y, negative z).

[2]The default values are used in the multiplication, otherwise the pattern would act like a steering wheel in a continuous camera movement. This would imply adding much complexity to the solution to allow some control over the velocity, and would most probably bring back the difficulties found in the previous stage with the Euler angles.