# Augmented Sword

David Preda
*FEUP*
Porto, Portugal
up201904726@fe.up.pt

Eduardo Correia
*FEUP*
Porto, Portugal
up201806433@fe.up.pt

Mário Travassos
*FEUP*
Porto, Portugal
up201905871@fe.up.pt

## I. Introduction

Augmented reality (AR) has rapidly evolved as a transformative technology with applications spanning various fields, from gaming and education to industrial use cases. In this report, we delve into the development of an **augmented reality application** aimed at enhancing the interaction between a physical reference marker and a virtual element, using a combination of computer vision and augmented reality techniques. The primary objective of this project is to merge the physical world with the virtual by **rendering a virtual sword associated with a cube-shaped reference marker**.

The application was developed in the Python programming language, and uses OpenCV [1] as the computer vision library of choice.

In the following sections, we present the methodology, results, and analysis of our augmented reality application, shedding light on the technical intricacies of marker detection, transformation, and the generation of virtual content. The project emphasizes the importance of marker recognition, and implementing custom marker identification methods for the reference marker.

This report serves as a comprehensive overview of the project, detailing the technical approach and outcomes of our augmented reality endeavour.

## II. Camera Calibration

To calibrate the camera, a script was created following the steps present in OpenCV's tutorial, which allows for an automatic calibration of the camera. In order to perform the calibration, several images of a 9 by 6 chessboard are taken at different angles, which are used to calculate the **intrinsic parameters** of the camera being used. These parameters are then saved in a *.npz* file for subsequent use in our application.

## III. Marker Recognition

This section provides an overview of the methodology used for marker recognition. We also include Figure 1 to illustrate an example as it goes through each major step in our marker recognition pipeline.

### A. Marker Detection

*1) Thresholding and Preprocessing:* The process starts with the conversion of the input image into **grayscale**, a step that simplifies subsequent analysis. Grayscale simplification reduces the complexity of the image, making it more amenable to further processing. Following this, a **Gaussian blur** is applied to the image, reducing image noise and enhancing the visibility of marker features. This noise reduction step is vital for ensuring that markers stand out distinctly against the background, leading to more precise detection.

Fig. 1. Pre-processing steps and marker homography. From left to right: original image; post-threshold image; detected contours; shape after homography.

To further improve marker detection, **Contrast Limited Adaptive Histogram Equalization** (CLAHE) is utilized. This technique enhances image contrast, bringing out subtle patterns within the markers. The result is an image that is better prepared for the subsequent thresholding step. **Otsu's method** is then employed for thresholding, converting the image into a binary format. This binary image effectively highlights the shapes of the markers against the background.

*2) Corner Detection and Contour Analysis:* Identification of marker corners is done through a **contour analysis**. This process involves extracting image contours, which serve as the foundational elements for identifying marker corners. To ensure precision, contours with **lengths smaller than a specific threshold are filtered out**, leaving behind the contours of interest. These retained contours undergo a **polygonal approximation**, to retrieve shapes with four sides.

*3) Corner Sorting and Refinement:* To guarantee an orderly and **consistent arrangement of marker corners**, a requirement for accurate marker identification, a sorting process is employed, so that markers are ordered anti-clockwise. This sorting operation hinges on the calculation of angles concerning a reference point based on the centroid of the marker.

Additionally, **close rectangles are removed** after sorting. This step aims to eliminate potential sources of confusion by preserving distinct markers. Further filtration is applied to the remaining rectangles, targeting those with a substantial length-to-width ratio. This selective filtering ensures that **only square-shaped markers are retained** for subsequent stages of analysis.

*4) Selection of the Largest Marker:* When multiple markers are detected, the code relies on a selection criterion based on the **contour area**. This criterion favours the marker with the **largest contour area**, as it is typically the most visually prominent and recognizable within the scene. This approach ensures that the augmented reality system prioritizes the most visually engaging reference point for accurate content alignment.

### B. Marker Identification

After detecting a possible marker, it is then identified by comparing it to a set of target markers until a match is found.

For this purpose, we attempted to implement two algorithms, a difference-based one, and a grid-based one.

*1) Difference-based Algorithm:* The *identify* method is the heart of marker detection. It receives an image and the detected corners of a potential marker as input. This method involves two primary subfunctions:

- *get_marker_from_image* employs **homography** to transform the detected marker's perspective into a standardized marker image. This homography transformation is crucial for ensuring that the detected marker aligns accurately with the predefined markers stored in the dictionary.
- *get_marker_from_dict* is responsible for retrieving a marker image from the dictionary based on the desired marker ID. This function facilitates a direct comparison between the detected marker and predefined markers.

The detected marker is then rotated in four orientations (0°, 90°, 180°, and 270°) to **match each orientation** of the predefined markers, allowing for comprehensive comparison.

The code systematically compares the detected marker with each predefined marker, rotating the detected marker in tandem. This comparison involves **a bitwise XOR operation** between the detected marker from the predefined marker. The resulting difference highlights the **discrepancies between the two markers**. The **number of white pixels** (pixel intensity equal to 255) in the subtraction image is counted to assess the **level of similarity** between the markers. Fewer white pixels signify a closer match. We showcase an example of this algorithm on Figure 3.

The code also takes into account the orientation of the marker's corners, ensuring that they align correctly with the predefined markers in all four orientations. If a match is found (i.e., a small number of white pixels), the code **returns the rotated marker's corners and the corresponding marker ID**. In the absence of a match, the code returns `None` and a marker ID of -1.

One potential critique of this algorithm lies in the possibility of incorrect marker orientations yielding a **high degree of similarity with an incorrect match**. To address this concern, we have curated a **limited subset of ArUco markers**, each possessing distinct visual characteristics, even when considering all possible orientations.

Moreover, even without the aforementioned mitigation, the likelihood of one marker exhibiting similarity to the homography greater than that of the correct match is exceedingly low. Collectively, these measures significantly **minimize the chances of an incorrect orientation resulting in an erroneous marker match**.

Fig. 2. Difference algorithm performed on an ArUco marker. From left to right: homography result; target marker; algorithm result.

*2) Grid-based Algorithm:* In an exploration of alternative marker detection methods, we investigated the feasibility and suitability of a **grid-based solution**. This approach entailed dividing the detected marker into a grid, where **each grid square was represented by its most prevalent colour**. The grid of colours was subsequently **compared to the predefined markers** in an attempt to identify potential matches.

However, our investigation revealed a notable drawback of the grid-based approach — it exhibited **significant computational slowness** when compared to the difference algorithm, which raised concerns about its real-time suitability for marker identification. As a result, we made the decision to **opt for the subtraction algorithm** due to its **superior speed and efficiency**, ensuring a more responsive and streamlined marker detection process within our augmented reality system.

## IV. Rendering

As a reference marker, we opted to use a cube-shaped object with an AruCo marker on each of their faces. The markers are 4 by 4 and their ids range from 94 to 99. [2]

This enabled to render the sword in all possible orientations, with each face indicating a side of the sword and with the handle positioned at the centre of the cube.

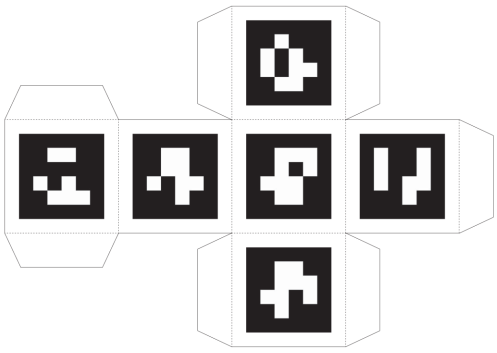

Fig. 3. Net of reference marker cube

After detecting the AruCo marker, the rotation and translation vectors, required to be able to project points in 3D space, are obtained using the OpenCV `solvePnP` function. This function uses the camera matrix and distortion coefficients

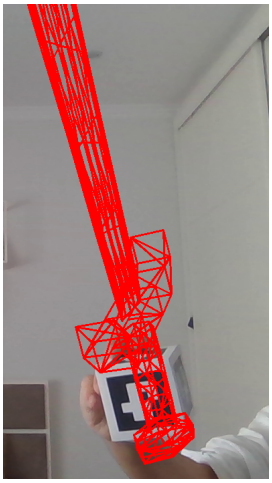

Fig. 4. Sword wireframe rendering on top of reference marker.

acquired when the camera is calibrated. We have six different sets of model points, one for each side of the cube, and depending on the marker detected, a different set is used. This is so that the origin coordinate of our virtual environment stands at the centre of the cube and the sword is rotated accordingly.

Then, a Wavefront object [3] is loaded, containing the vertices and faces wireframe information of our sword model. To render the sword, the 3D vertices are scaled up and projected onto the 2D image plane, using `projectPoints` and contours are drawn based on specified faces. This produces the result shown in Figure 4

## V. Conclusion

The main objectives of the project were concluded successfully. We were able to render a 3D sword on a reference marker, as well as accompany said marker's movement.

Having said that, however, there are still some setbacks which provide room for future improvements. Some optimizations in the marker recognition process would greatly improve the alignment of the objects when overlaid upon the markers, and rendering the objects in a different manner (drawing the full geometry of the sword, for example) would aid in making the augmentation feel more realistic. Furthermore, some other improvements, which are out of scope of this project, such as implementing occlusion, namely occluding part of the sword behind the marker, would also aid in providing the user with a more authentic and complete AR experience.

## References


[1] G. Bradski. The OpenCV Library. Dr. Dobb's Journal of Software Tools, 2000

[2] Gwyll. Fologram Community. https://community.fologram.com/t/aruco-marker-download/147, 2020

[3] Skyllet. OpenGameArt. https://opengameart.org/content/low-poly-sword-0, 2015