Augmented Sword

Carlos Gomes
M.EIC
FEUP
Porto, Portugal
up201906622@up.pt

Miguel Amorim
M.EIC
FEUP
Porto, Portugal
up201907756@up.pt

Rafael Camelo
M.EIC
FEUP
Porto, Portugal
up201907729@up.pt

***Abstract*—The Virtual Reality and Image Recognition project we present aims to create an immersive and interactive experience that leverages advanced technologies to transform the way people interact with the digital and physical world. The combination of virtual reality and image recognition opens doors to a variety of exciting applications, revolutionizing the way we perceive and engage with our surroundings. This project holds great potential for industries ranging from gaming and education to healthcare and beyond, promising to redefine the boundaries of human-computer interaction.**

***Index Terms*—Augmented Reality, Sword, Reference Marker, Image Analysis, Image Processing.**

I. Introduction

This project was designed and developed for the curricular unit of Augmented and Virtual Reality, or *RVA* for short.

Nowadays, with the increasing use of cameras and mobile phones, the realm of computer vision has emerged as a powerful tool to bridge the gap between the virtual and the real world.This report delves into a fascinating journey where we harnessed the potential of Python programming and image recognition techniques to create a distinctive solution. The core objective of this project was to design an image recognition Python script capable of not only analyzing a given image but also detecting specific markers within it. Additionally, we were tasked with utilizing this marker detection to position and render a virtual sword within the image of the real world, creating a seamless transaction between the real and the virtual.

This report will provide an in-depth exploration of the methodology, experiments designed, technologies employed, challenges faced, and results achieved during the course of this project. By the end of this report, we hope to offer a comprehensive understanding of the inner workings of our image recognition Python script, its shortcomings and what would be the next steps to take this small project to the next level.

II. Implementation

In order to incorporate both marker recognition and augmented reality features we have developed a Python program using the Open CV library. We will break down the code, highlighting the key processes involved in the implementation of these functionalities.

A. Image Pre-Processing

The pre-processing steps were crucial for creating an accurate marker recognition process. These steps transform the inputs into a format that is more helpful to achieve marker isolation and consequently its identification. The first step was to transform the test image and the marker's dictionary in grayscale (Fig. 1). This process transforms the multi-channel color into a color of a single channel which reduces computational complexity and removes information related to color information that is irrelevant to marker detection.

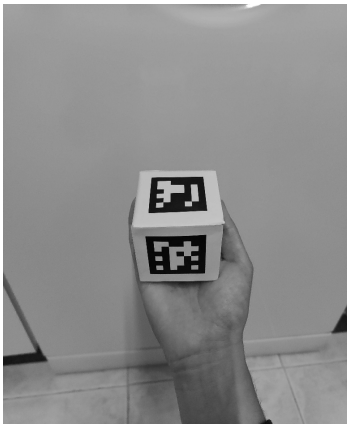

Fig. 1: Grayscale Transformation

Then we apply thresholding (Fig. 2) to the test image to convert it into a binary image. In the scope of this task, thresholding converts the grayscale image to a simple black and white image, enabling the marker to be distinguished from the background. This simplification facilitates the analysis and recognition of the marker. In our implementation, we defined that the maximum and minimum values should be 255 and 0, respectively, and use the Otsu's method. This is an algorithm that exhaustively searches for the threshold that minimizes the intra-class variance, so that we get the optimal threshold value.

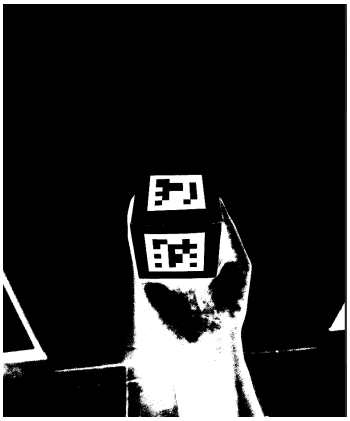

Fig. 2: Thresholding

To conclude the Pre-Processing we use Open CV *cv2.findContours*, to apply contour detection to the thresholded image. This step allow us to identify and isolate contours that could possible be a marker. To enhance robustness of this step we establish a threshold representing the percentage of area of the marker in the original image, check if the marker is a quadrilateral and if all points of the contour are located on the same side of the straight lines connecting any two points on the contour, indicating that it is convex. Following this process we have our region of interest defined and ready to apply the matching (Fig 3).

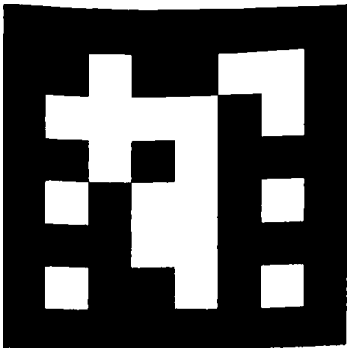

Fig. 3: Marker after contour found

B. Marker Identification

For marker identification, our initial approach was to use feature extraction methods such as SIFT, ORB, or SURF. Although it was useful in confirming whether a contour was indeed a marker, they were not accurate enough for marker identification. The features that these algorithms extract were black and white regions and the corners of the marker's pattern, which were very similar to each other. As a result, the matching process did not work as expected, even when using outlier handlers like RANSAC and the Lowe Ratio test. To address this problem, we considered about using the *cv2.matchTemplate*.This method slides the template image (marker) over the test image, and compares the template and patch of input image under the template. However, this method cannot handle rotation, meaning that the template and input need to be in the same orientation. To overcome this limitation we simply performed a match template for each marker rotation (since it is a square we only need to do it 4 times), and them save this results in a list. This process is representend in Fig. 4 and Fig. 5.

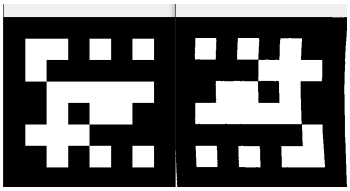

Fig. 4: Comparison between ROI and template

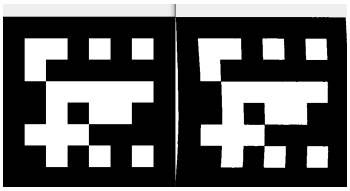

Fig. 5: Comparison between ROI and template - Match Found

The best result is then appended to another list which stores this best result for each marker. Then the index with the higher value represents the id of the matched marker. After some testing on this we also find another problem. Our image acquisition method was using a smartphone camera, but the acquired image was mirrored so the markers were also mirrored. To overcome this issue we just flipped to input image and results got better significantly. After this the marker id is displayed in the image next to it and the contour is highlighted.

C. Camera Calibration

Following the previous steps, we proceeded to calibrate the camera. This process is crucial as it allows us to accurately estimate the 3D pose of a marker in space. The calibration process involved determining the camera's intrinsic parameters, which are properties of the camera itself and do not depend on the scene being viewed. These parameters include the focal length, the optical center (also known as the principal point), and the lens distortion coefficients. We calculated the focal length in pixels using the formula:

$$\mathrm{F(px)} = (\mathrm{F(mm)} \cdot \mathrm{W(px)})/\mathrm{W(mm)} \tag{1}$$

"F represents the focal length and W represents the sensor width. For our specific case, we obtained the camera specifications from the manufacturer's website and used a common value for the sensor width in millimeters, which typically ranges between 4-7 in smartphones. Using Open CV's *cv2.solvePnP* function, we estimated the 3D pose of a marker given its 2D corners. This function returned two vectors: rvec, which represents rotation, and tvec, which represents translation. These vectors describe the transformation from the 3D model's coordinate system to the camera's coordinate system.

Finally, we projected 3D points onto the image plane using Open CV's *cv2.projectPoints* function and drew a cube using these projected points. The calibration significantly improved our plane display and allowed us to accurately draw cubes on all markers. We then selected only the marker with the largest area for further processing (Fig 6).

Fig. 6: Camera Calibration

D. Image Augmentation

In our Image Augmentation project, one fascinating aspect involves using detected markers to seamlessly incorporate a sword into an image that we've created. This process exemplifies the fusion of cutting-edge technology and creative expression.

Once the markers are detected, our augmentation algorithm takes over. It precisely analyzes the spatial data and perspective of the image, determining the most suitable location for adding the sword, being the marker with more area calculated. This ensures that the sword seamlessly integrates with the scene, appearing as if it were part of the original image all along.

The sword is a 3D draw done by our project. Firstly, it's done with the direction of the marker and its contour calculated in the pre-processing phase. After that, the algorithm does a rotation in the sword in vertical way. One of the difficulties we had was understanding the angle of rotation depending of the marker and the photo.

Here in Figure 8, we can see one example of result of this method.

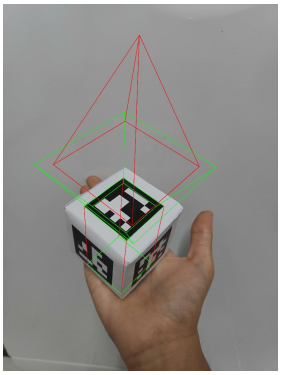

Fig. 7: Result in image2.

III. Experiments

To test our project we decide to use different images with different markers and perspectives to test both marker identification and image augmentation.

We use images as the following example:

Fig. 8: Test image number 7.

We encountered challenges during the image augmentation process, particularly when it came to rotating the sword to adapt it to a vertical position. This task was not only complex but also required a deep understanding of image manipulation techniques. Despite these challenges, our efforts ultimately yielded favorable results, as demonstrated in the appendix and, more prominently, in Figure 7 or 10.

The success in achieving these results underscores the dedication and problem-solving skills of our team, as we strived to address the unique demands of the project. This achievement also showcases the significance of our work, where our ability to overcome obstacles in image augmentation contributed to the project's overall success. Our commitment to delivering quality results and overcoming technical hurdles is a testament to the effectiveness of our approach.

IV. Results and Discussion

Our project results have indeed successfully achieved our set objectives. We implemented marker recognition and image augmentation effectively. Additionally, we made significant improvements by using a cube as a reference marker. Notably,

we accomplished this enhancement without the need for the ArUco library.

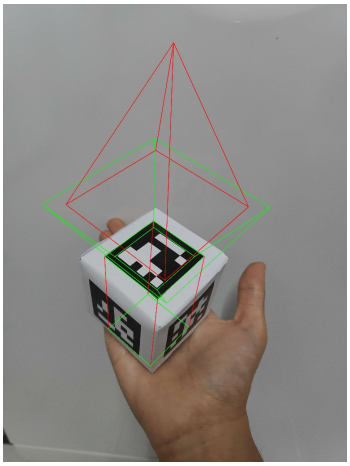

Fig. 9: Result Analysis - 1.

In Figure 9, the system only detects the top marker. As a result, the sword is displayed according to the 3D plane of that particular marker.

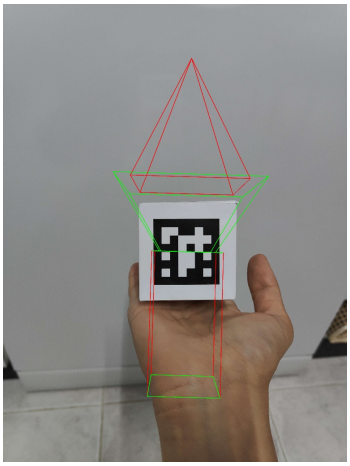

Fig. 10: Result Analysis - 2.

In the second case, as shown in Figure 10, there is again only one marker detected. However, in this instance, the orientation of the marker needed adjustment. As demonstrated, our program successfully managed to make the necessary adjustments. This highlights the system's capability to not only detect markers but also accurately adjust their orientation as required.

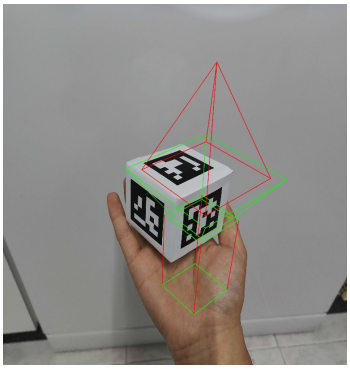

Fig. 11: Result Analysis - 3.

In the final case, as shown in Figure 11, three markers are detected: 1, 2 and 4. All of these markers have similar sizes. However, the marker on the right (4) has the largest area. As demonstrated, the sword adjusts its position based on this marker, thereby achieving the correct position. This showcases the system's ability to dynamically adjust augmented reality objects based on multiple markers, prioritizing the one with the largest area. This feature is crucial for ensuring accurate and realistic augmentation in complex scenes with multiple markers.

V. Conclusions and Future Work

Our project's results showcase not only our ability to meet the set objectives but also the innovative solutions we have brought to the fields of marker recognition and image augmentation. These achievements have the potential to influence diverse applications, from augmented reality gaming to industrial automation, where accurate and dynamic object recognition is paramount. One of the most significant accomplishments of our work is the achievement of this enhancement without relying on external libraries such as the ArUco library.

For future work, we would like to implement our approach with 3d models (glb format files for example) to create even better results. Furthermore, the entirety of this project was developed within the constraints of picture analysis, the next step is to adapt the program to be able to process video while maintaining responsiveness.

References

[1] OpenCV. (2023). Image Thresholding — OpenCV 4.8.0 documentation. Retrieved from https://docs.opencv.org/4.8.0/d7/d4d/tutorial_py_thresholding.html

[2] OpenCV. (2023). Contour Features — OpenCV 4.8.0 documentation. Retrieved from https://docs.opencv.org/4.8.0/dd/d49/tutorial_py_contour_features.html

[3] OpenCV. (2023). Template Matching — OpenCV 4.8.0 documentation. Retrieved from https://docs.opencv.org/4.8.0/d4/dc6/tutorial_py_template_matching.html

[4] OpenCV. (2023). Changing Colorspaces — OpenCV 4.8.0 documentation. Retrieved from https://docs.opencv.org/4.8.0/df/d9d/tutorial_py_colorspaces.html

[5] OpenCV. (2023). Shi-Tomasi Corner Detector & Good Features to Track — OpenCV 4.8.0 documentation. Retrieved from https://docs.opencv.org/4.8.0/d4/d8c/tutorial_py_shi_tomasi.html

[6] OpenCV.(2023). Camera Calibration and 3D Reconstruction — OpenCV 4.8.0 documentation.Retrieved from https://docs.opencv.org/4.8.0/dc/dbb/tutorial_py_calibration.html

VI. Annexes

A. Test images

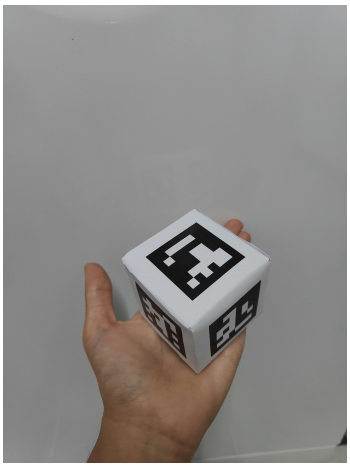

Fig. 12: Test Image - 1

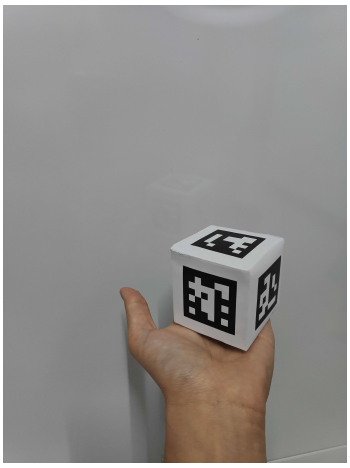

Fig. 13: Test Image - 2

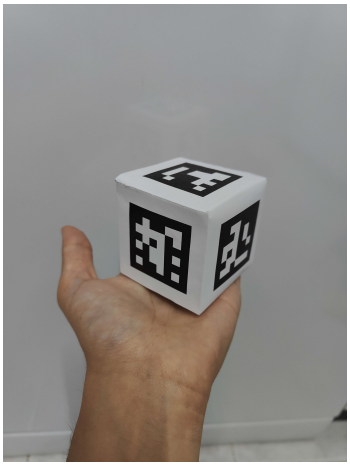

Fig. 14: Test Image - 3

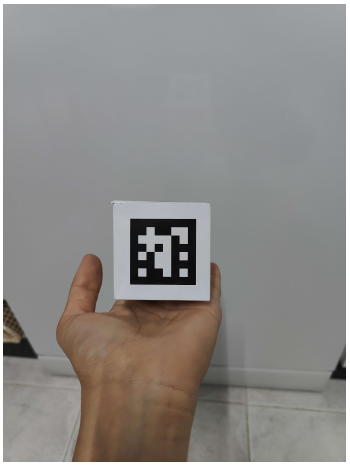

Fig. 15: Test Image - 4

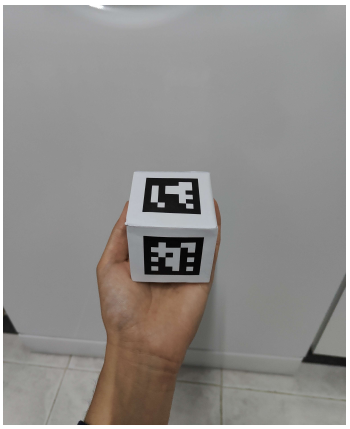

Fig. 16: Test Image - 5

B. Results

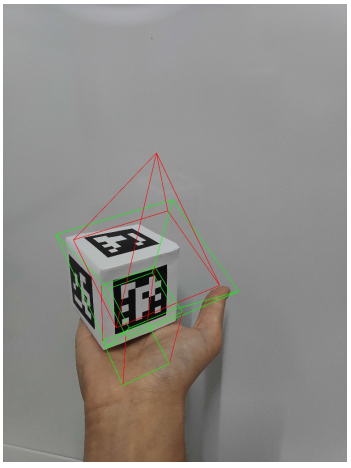

Fig. 17: Result - 4

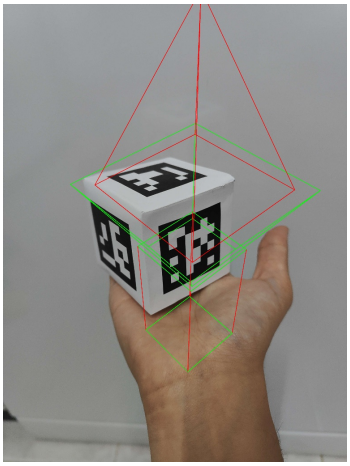

Fig. 18: Result - 5

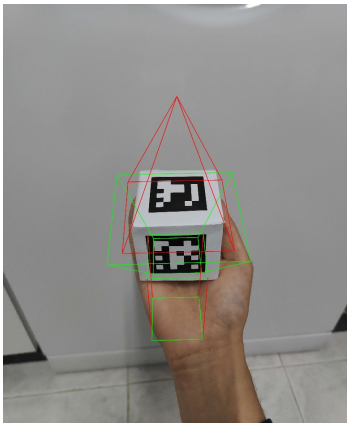

Fig. 19: Result - 6