Augmented Museum - CS 5330 Final Project

1st Kevin Heleodoro
CS 5330 - Pattern Recognition & Computer Vision
Northeastern University
Boston, USA
heleodoro.k@northeastern.edu

***Abstract*—Technological advances have provided many benefits to the field of Augmented Reality (AR) which, in turn, provide ground-breaking new techniques to many fields; art, medicine, and gaming to name a few. This project focused on the field of art and how AR can bring beloved creations into everyday life. The program was built using C++ and the OpenCV library to recognize ArUco markers within a screen and calculate the pose of the marker relative to the camera. After that, a selected image, resized to fit a 560x720px frame was used as an overlay on the marker which used the marker's rotation and translation vectors to impose a "life-like" 3-dimensional effect. This allows the user to move around the room and observe the image from different angles and distances.**

***Index Terms*—augmented reality, marker-based AR, ArUco, art**

I. Introduction

The purpose of this project was to experiment with AR in a domestic setting where ArUco Markers are used as placeholders for images (famous paintings for the examples in this paper). ArUco markers were created to serve as the central point where images would be placed. A calibrated camera was then used to detect the corners of the marker and calculate the estimated pose in relation to the camera. Multiple markers can be easily detected using this approach, which will serve us later when we overlay more than one image simultaneously within the screen.

Once a marker has been detected the next step is to calculate the size of the overlay image and find the homography between it and the marker. A mask is then created and used as an overlay for the marker which displays the requested image, in full-size, centered at the marker's location. The challenge here was correctly manipulating the size of the overlay and maintaining the relationship to the marker's rotation and translation vectors. In the end, the program was able to effectively recognize an ArUco marker within its screen and place the image on top of the marker.

Multiple markers were also used to test parallel execution of overlays within one screen. This was achieved with relative success as the processing power needed to handle multiple markers can grow at a rapid pace.

II. Related Work

A. The Use of Marker-Based Augmented Reality in Space Measurement [1]

Boonbrahm et al. uses ArUco markers to determine the measurement of different size rooms. In part of their research they noted that attempts to use a Markerless AR were unsuccessful due to the its difficulty with detecting uniform colored surfaces. They instead relied on the ArUco markers for their reliability and "speedy" recognition capabilities. Similar to this project, their research consisted of detecting multiple markers in a camera view and calculating their pose. This technique allowed them to find the direction which the markers were facing compared to the others and derive the measurements of the room.

B. Mobile Augmented Reality (AR) Marker-based for Indoor Library Navigation [2]

Rusnida Romli et al. conducted an experiment using virtual markers to display navigational information within a library. Instead of ArUco markers, the team defined virtual points within their program to be recognized as "markers". Once the markers were detected on the screen, a message was overlaid that included some text with simple directions to other parts of the library. They were also able to use these same markers to overlay multiple info-graphic images onto the display screen.

C. ArUco marker-based displacement measurement technique: uncertainty analysis [3]

Tocci et al. performed research on using industrial cameras to measure displacement of a structure using the detection of ArUco markers. For this experiment the authors compared the commonly used *Squared Planar Marker* with ArUco markers and determined that the ArUco markers, in cases of non-uniform light conditions and the desire to detect multiple markers simultaneously, were better suited to their research. Physical ArUco markers were printed on standard printing-paper and placed on a vibrating structure to measure the amplitude displacements.

III. Methodology

A. Camera Calibration

Calibrating a camera is a normal task that needs to often be conducted in order to execute a range of computer vision programs. Since the purpose of this project was to accurately detect ArUco markers in a frame, estimate its position, and then overlay an image onto it, it was especially important that this step was conducted in the appropriate manner. To perform the calibration a (9,6) chessboard was printed onto standard printing-paper and multiple images were captured of it from different angles and distances. With the use of OpenCV's

cv :: $calibrateCamera$ function [4] we can calculate the **Camera Matrix** and the **Distortion Coefficients**. Both of these parameters are essential in estimating the pose of the ArUco marker.

Fig. 1: Example calibration image. Multiple images taken from different angles are required to effectively calibrate the camera.

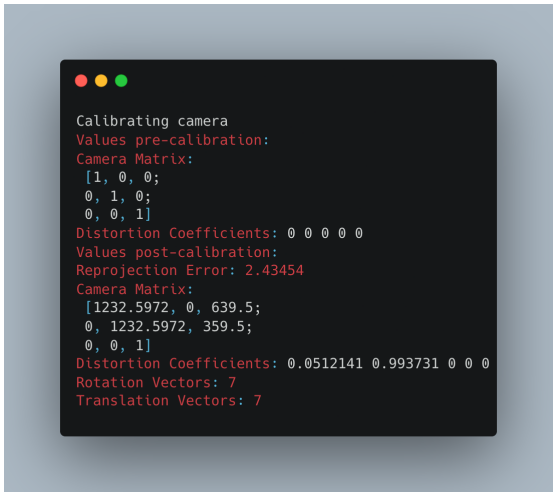

Fig. 2: Camera Matrix and Distortion Coefficients that were calculated using the chessboard images. These parameters will be necessary to find the pose of the ArUco markers in later steps.

B. ArUco Markers

The ArUco marker size chosen for this program consisted of a **8 x 8** grid of blocks. The border, 1 block thick, served as the boundary for the arbitrarily dispersed **6 x 6** grid contained within. When OpenCV references ArUco markers they choose to ignore the border, since those blocks will always be black, and focus solely on the blocks that can be either black or white. With this in mind the application uses a predefined library of **6 x 6** markers with a total of 250 unique combinations of blocks.

Figure (3) contains an example of the ArUco marker first used to conduct detection testing. This marker is a member of OpenCV's predefined library of **6 x 6** ArUco markers.

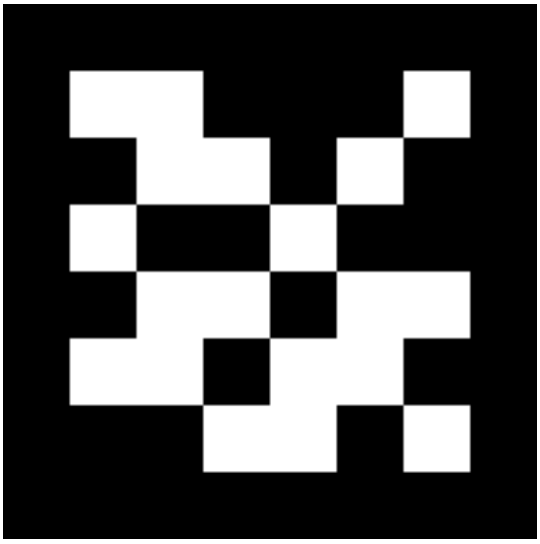

Fig. 3: Aruco_marker_58. A variety of these markers were created in later stages of testing to overlay multiple images.

C. ArUco Marker Detection

The task of marker detection begins with capturing frames from a video feed. Each frame is processed to search for markers that match the predefined library. If no markers are detected within a frame then the program moves on to the next frame. We utilize OpenCV's cv :: $aruco$ :: $ArucoDetector$ :: $detectMarkers$ function [5] to search each frame for any markers. If a marker is found the function returns a list of the corners positions within the frame for the marker. Once we have the marker corners we can begin to display where it is within the frame and then estimate its pose in relation to the camera.

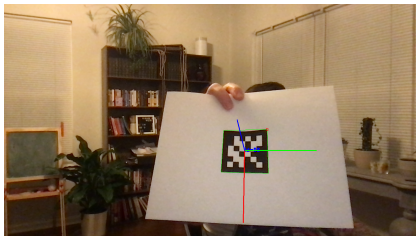

Fig. 4: ArUco Marker after detection and pose estimation calculated.

Figure (4) shows the results of successfully detecting the ArUco marker and using the camera calibration parameters to estimate its pose in relation to the camera. From here we can begin the process of overlaying an image to the marker.

D. Image Overlay

The most challenging portion of this program consists of taking a chosen image, resizing it and then placing it over the marker. For the test images I utilized the *Best Artwork of All Time* dataset [6] which consisted of over 8000 different artworks from well-known artists throughout history. For a

reference on overlaying images on ArUco markers I used the work of Rahul Kedia [7] as a guide to calculating homography and successfully overlay content over a marker. Although his implementation used four markers placed in a rectangular shape on a wall, I was able to use the logic as a baseline for how to approach this problem.

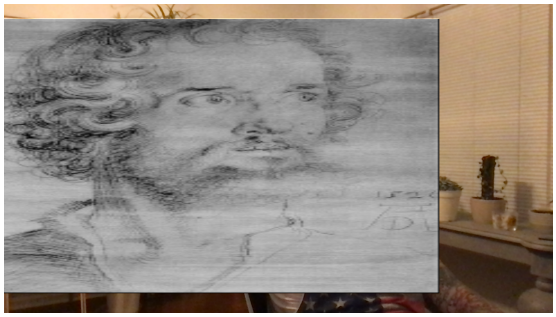

Fig. 5: After trying to resize the image and incorrectly calculating the marker corners the overlay was blown out of proportion and its left side was fixed to the left border of the frame.

The workflow for detecting the marker and overlaying the selected image consists of the following steps:

- Load the images and resize them to (560, 720)px.
- Set the coordinates system for the ArUco marker based on the predefined length of 200px.
- Detect the ArUco marker within a frame.
- Find the markers pose from 3D to 2D point correspondence.
- Calculate the proposed 3D corners for the newly resized overlay image.
- Project those 3D points onto a 2D plane using the camera matrix and distortion coefficients.
- Compute the homography from the overlay image to the projected points.
- Warp the overlay image using the homography matrix.
- Create a mask where the overlay image is positioned.
- Create the inverse of that mask for the background.
- Prepare background image by masking out overlay area.
- Combine the background and warped overlay.

IV. Results

The following figures represent the progression of testing that I went through to get the image resized image to correctly overlay onto the marker.

After getting to a satisfactory result from detecting a single marker, the next challenge was to perform the same workflow on multiple markers within the same screen. Because of the initial design of the program, the logic loop was not built to handle overlaying multiple markers. When more than one marker was presented within the frame, the program would select one of the markers present and use it as the marker which to overlay the image, essentially ignoring all other markers. Any disturbance or movement to the marker that was currently being overlaid would cause the program to immediately jump to the next available marker within the screen and overlay the image to it instead.

Fig. 6: Images: (a) The second overlay attempt resulted in the image directly replacing the ArUco marker; (b) Another attempt was made at resizing the overlay image to fit the ArUco marker. The error here was in assigning the top-left corner of the overlay to the top-left corner of the marker. The translation and rotation vectors did not maintain the relationship because I wasn't accounting for the 3rd dimension vectors.; (c) On this iteration I was able to get the translation and rotation vectors to maintain the relationship between overlay and marker. However, the sizing wasn't quite right and the overlay was extremely unstable at most angles.; (d) Example of overlay instability.; (e) After resizing the image before attempting to overlay I was able to get the overlay to behave as expected. Here is the ArUco marker placed on a shelf approximately 10 feet from the camera.; The overlay now reflected the intended size proportion. This also showcases the easy recognition capabilities of the ArUco detection system despite the marker being placed in a "busy" context.

Fig. 7: Multiple overlay images on multiple ArUco markers

This issue stemmed from how I was originally processing the individual overlays, where I was unable to directly process all the markers at the same time. By manipulating the detection logic to directly modify the source frame I was able to place

individual images over each marker detected. I was unable to perfect the direct manipulation technique to arrive output multiple full-size images, but in Fig. (7) you can see that all the markers detected have been overlaid with a different image.

REFERENCES

[1] S. Boonbrahm, P. Boonbrahm, and C. Kaewrat, "The use of marker-based augmented reality in space measurement," *Procedia Manufacturing*, vol. 42, pp. 337–343, 01 2020.

[2] R. Romli *et al.*, "Mobile augmented reality (ar) marker-based for indoor library navigation," in *IOP Conference Series: Materials Science and Engineering*, vol. 767, 2020, p. 012062.

[3] T. Tocci *et al.*, "Aruco marker-based displacement measurement technique: uncertainty analysis," *Engineering Research Express*, vol. 3, p. 035032, 2021.

[4] OpenCV Contributors, "Camera Calibration and 3D Reconstruction," https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html, 2024, accessed: 2024-04-23.

[5] ——, "ArucoDetector Class Reference," https://docs.opencv.org/4.x/d2/d1a/classcv_1_1aruco_1_1ArucoDetector.html, 2024, accessed: 2024-04-23.

[6] Ikarus777, "Best Artworks of All Time," https://www.kaggle.com/datasets/ikarus777/best-artworks-of-all-time?resource=download-directoryselect=resized, 2024, accessed: 2024-04-23.

[7] R. Kedia, "Augmented Reality Television with ArUco Markers: OpenCV, Python, C++," https://kediarahul.medium.com/augmented-reality-television-with-aruco-markers-opencv-python-c-c81823fbff54, 2021, accessed: 2024-04-23.