Computer Vision

Project Report:

AUGMENTED REALITY

Project Author: Nirbhay Borikar
Date:26/11/2024

Contents

1 **Chapter 1** **1-2**
1.1 Introduction..... **1**

2 **Chapter 2** **3-8**
2.1 Marker Detection..... **3**
2.2 Selecting the corners in the poster..... **6**
2.3 Warping the Poster..... **6**
2.4 Execution..... **8**

3 **Chapter 3** **9-11**
3.1 Result..... **9**
3.2 Evaluation..... **10**
3.3 Task 1.2..... **11**

4 **Chapter 4** **12**
4.1 Conclusion..... **12**

Chapter 1

Abstract

In this study, we analyze and detect ArUco markers, specifically focusing on the 6x6 marker format. For experimentation, we utilized a 5x5 marker with ID 7 and a 6x6 marker with ID 0, both set to a size of 100 mm. After loading the images and detecting the markers, we optimized the process to integrate a poster into the image. The poster was placed such that its center was aligned with the center of the ArUco marker. Additionally, the placement ensured that the poster remained undistorted and symmetrically aligned within the image.

Introduction

An ArUco marker is a type of fiducial marker extensively used in computer vision applications to detect and estimate the spatial pose of objects within a scene. These markers find significant utility in various domains, including Augmented Reality (AR), Robot Localization, and other fields requiring precise determination of spatial orientation and positioning [1]. This study aims to develop an algorithm to detect an ArUco marker positioned on the wall of Room H238 and overlay a digital poster to visualize its appearance before physically installing it. The dataset titled *Room with ArUco Markers*, uploaded by Prof. Dr. Stefan Elser on Moodle, comprises 11 images for analysis [2]. The dataset includes photographs of a 6x6 ArUco marker with the ID '0', captured from multiple perspectives. Identifying the precise location of this marker is critical for successfully projecting the digital poster. A poster featuring the Ancient Temple of Kedarnath was chosen for this task [3].

Figure 1: Poster for this project.

Figure 2: ArUco Marker is on the wall and the Poster Image Corner is outside the wall. This image is copy pasted.

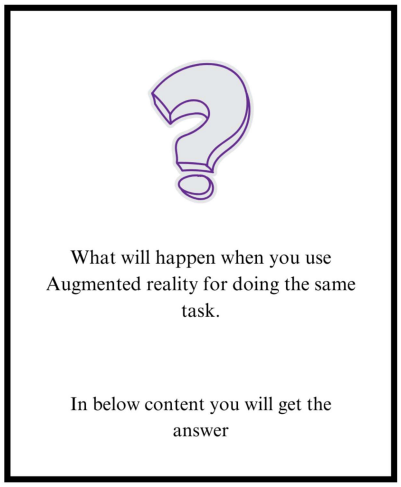

Figure 3: How to achieve target ?

Chapter 2

Marker Detection and Poster Projection Algorithm

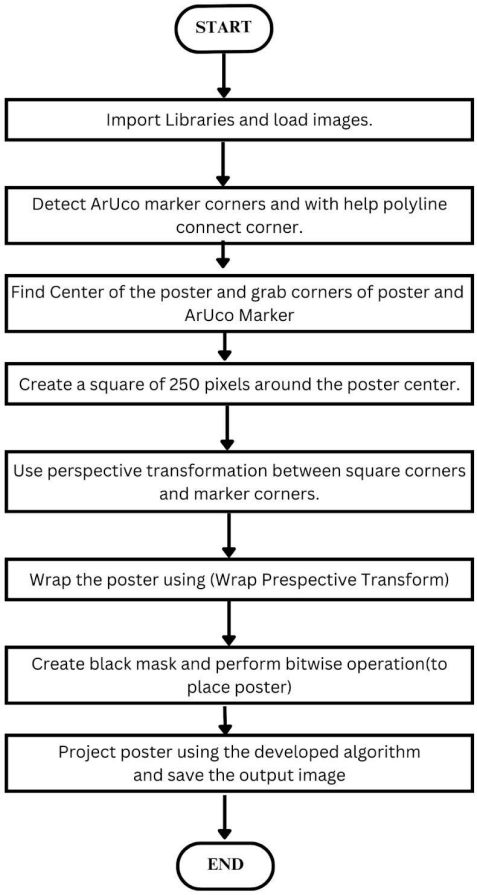

Figure 4: Algorithm Flowchart.

2.1 Marker Detection

To utilize a specific function in our task, we have imported several essential libraries, including cv2, NumPy, and Matplotlib. The cv2 library provides various functions such as ‘cv2.imread()‘, ‘cv2.imshow()‘, and ‘cv2.waitKey(0)‘, along with several other task-specific functions, all of which are identified by the ‘cv2‘ prefix. The NumPy library is integral for image processing since images in OpenCV are represented as multi-dimensional arrays. NumPy enables efficient manipulation and optimization of these arrays, making it a critical component for image-related computations. Finally, Matplotlib serves as a powerful tool for visualizing images. It is particularly useful for displaying output images

directly within notebooks or scripts, providing an intuitive way to analyze results. This combination of libraries streamlines the process of working with images and enhances the efficiency of various image processing tasks.

Function 1

After that, we implemented our first function to read images, which essentially loads the image into the OpenCV environment.

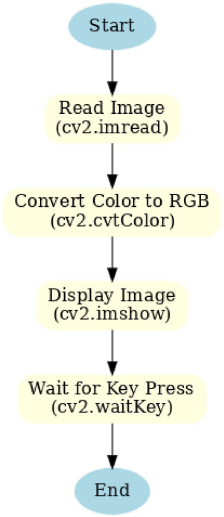

Figure 5: Flowchart of Image to Read, Color Conversion, and Display Result.

Function 2

The OpenCV library includes an inbuilt function, ‘detectMarkers()‘, which requires three primary inputs: the source image, a dictionary containing the marker set (specifying the matrix size and IDs), and detection parameters. One such parameter, for instance, controls the perimeter of the square ArUco marker and allows customization of the detection process. To access the marker dictionary, the ‘cv2.getPredefinedDictionary()‘ function is commonly used as an alternative to ‘dictionary_get()‘, both of which retrieve marker dictionaries. Regarding the ArUco marker’s matrix size, the ‘detectMarkers()‘ function identifies marker corners in the source image and highlights them by connecting these corners with green lines. This visualization is achieved using the ‘polylines()‘ function, which enables the joining of polygon points.

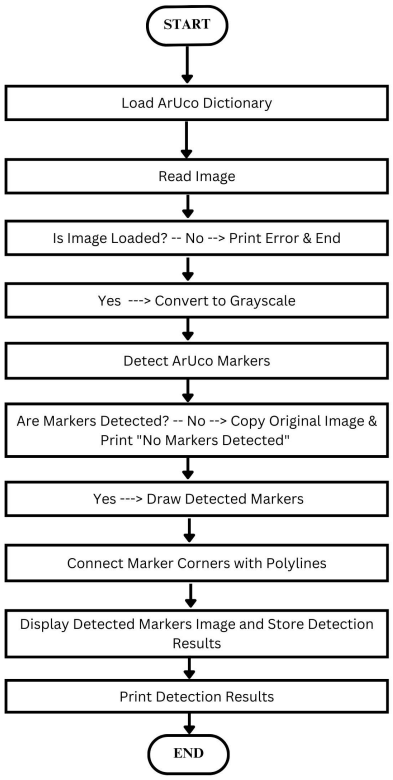

Figure 6: To detect ArUco Marker, return coordinates and also connect corners using Polylines().

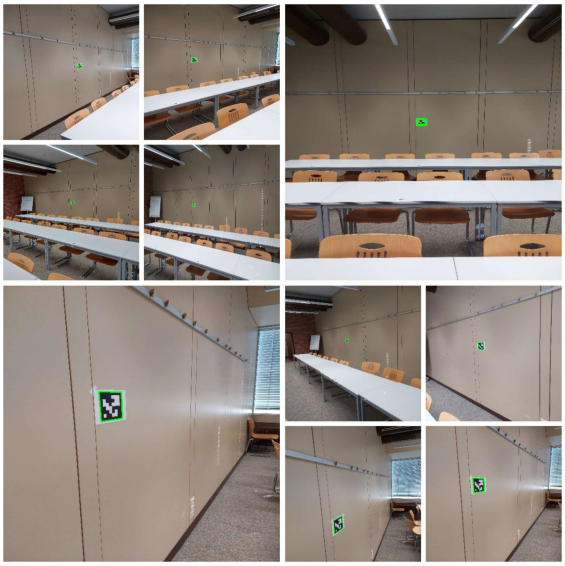

Figure 7: ArUco Markers successfully detected in all the ten images out of eleven.

The marker detection algorithm successfully identified the marker with a green square in all images containing the marker. However, one image in the dataset lacks the marker, resulting in the algorithm failing to detect it, which causes an error.

2.2 Selecting the Corners in Poster

The presented approach utilizes a series of image processing functions to prepare and manipulate key geometric features for augmenting images effectively. First, the grab_corners function identifies the corner points of an image, providing a rectangular boundary that serves as the basis for further processing. Next, the get_polygon_width_height function calculates the width and height of a polygon based on marker corner coordinates, ensuring accurate representation and scaling of the target region. To facilitate a consistent and centered augmentation process, the create_polygon_in_frame function constructs a polygon with specified dimensions (width and height) centered at a given frame's midpoint. These functions collectively enable precise geometric transformations, allowing for seamless alignment of augmented content onto markers while preserving spatial integrity and avoiding distortions. This methodology ensures flexibility and accuracy in applications requiring image augmentation or marker-based projection.

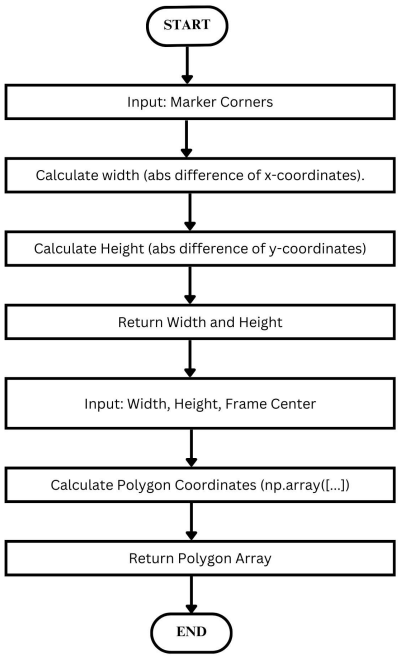

Figure 8: Corner Selection for Poster.

2.3 Warping the Poster

The warping process ensures that the poster image (frame) is seamlessly overlaid onto the detected marker in the target image. The transformation begins with the identification of a fixed polygon within the frame, created using the 'create_polygon_in_frame' function. This polygon is then mapped onto the detected marker's corner points using a perspective transformation matrix ('cv2.getPerspectiveTransform'). This matrix facilitates the warping of the frame to match the dimensions, orientation, and perspective of the marker in the target image, ensuring an accurate alignment.

To handle black pixels in the frame, the ‘remove_black_pixels‘ function replaces zero-intensity pixels with small non-zero values (e.g., [1,1,1]). This step prevents errors during processing and ensures a clean transformation by avoiding gaps or unintended black spots in the final warped image.
The masking process complements the warping step by addressing the remaining black spaces. A mask is created to identify regions corresponding to the marker in the original image. Black pixels in the warped frame are isolated using this mask, and the corresponding marker area in the original image is cleared using bitwise operations (‘cv2.bitwise_and‘). This ensures that the marker is replaced cleanly by the warped poster without residual artifacts. Finally, the warped image is blended onto the cleared marker area using bitwise OR operations (‘cv2.bitwise_or‘), producing a seamless augmented image. This combined approach of warping and masking ensures precise overlaying of the poster, eliminates unwanted black spaces, and maintains the visual integrity of the augmented image.

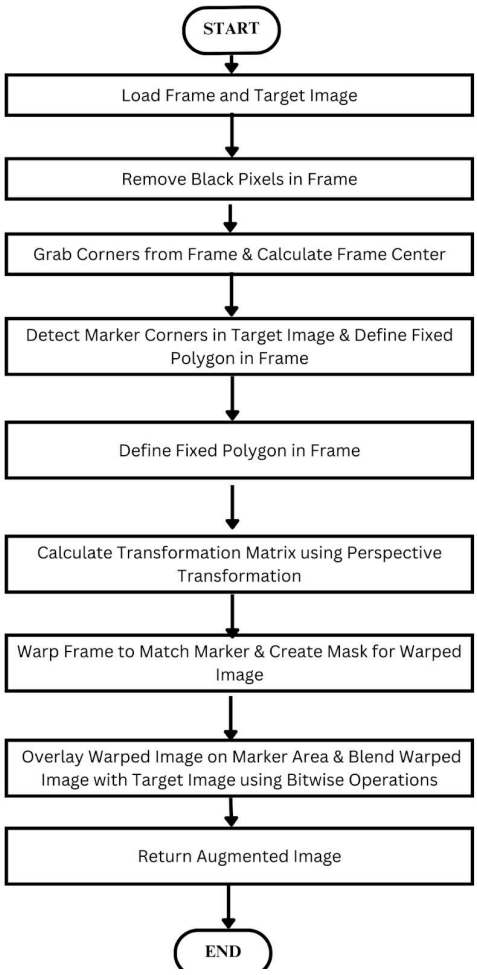

Figure 9: Flowchart of Warping the Poster

2.4 Execution

In this section, we provide the path to the classroom image, the poster, and other essential parameters or functions required for the process. Additionally, the resulting image is displayed using Matplotlib, the output image is saved to a specified location, and the output path is printed to the console.

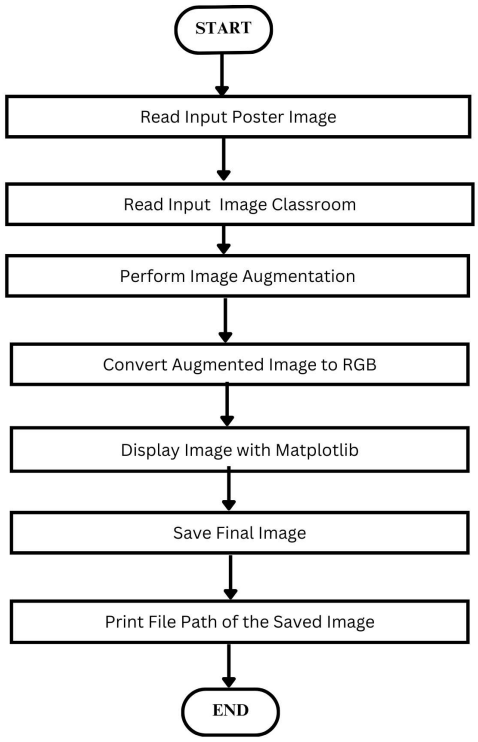

Figure 10: Main Execution

Chapter 3

3.1 Result

On running the above algorithms, following outputs were generated. The function was applied on photographs of the classroom taken at various angles to test its performance.Images are zoomed in for better.Here the poster dimension taken is 250 x 250 pixels.

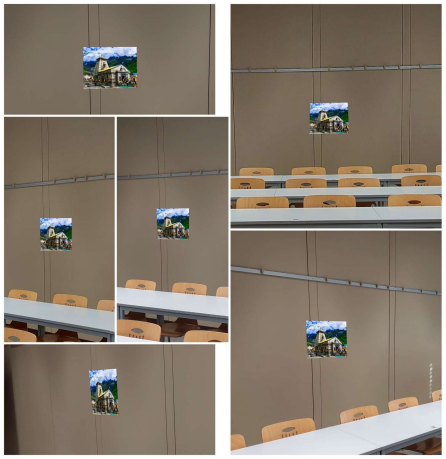

Figure 11: Final_Output1.

Figure 12: Final_Output2.

The below image is taken just to show that we can increase and decrease size of the poster inside the image to any extent.

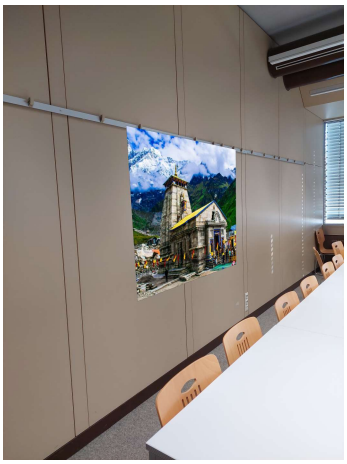

Figure 13: Example Image of higher dimension

3.2 Evaluation

For evaluation, we used both a perspective-based and a mathematical approach. Specifically, we analyzed the relationship between the edges of the poster and the edges of the wall within the classroom image. If any two sides of the poster were found to be parallel to the edges of the wall in the image, we could confidently conclude that the poster was aligned. In below image representation, the relation between edges are mentioned.

Figure 14: Final Evaluation

Task 1.2

We also applied the algorithm to our custom dataset, selecting a 5x5 ArUco marker with ID 7, which was placed on the door of a dormitory room in Germany, commonly used by students. Photographs of the marker were taken from various angles, and the detection algorithm was used to identify the marker in each image. Below, we present the output images generated after applying the algorithm to this dataset.

Figure 15: Another dataset with comparable output.

Chapter 4

Conclusion

In this project, we successfully utilized OpenCV's built-in functions and a predefined ArUco marker library to achieve our objectives. Out of the provided dataset, the algorithm failed to produce an output for only one image due to the absence of an ArUco marker. However, in the remaining ten images, the ArUco markers were successfully detected, and the algorithm effectively placed the poster geometrically and accurately on the wall where the marker was located. The evaluation process, detailed in the evaluation section, assessed the alignment of the poster using the edges present in the images. By identifying parallel relationships between the edges, we ensured proper alignment and positioning. To further validate the algorithm, we tested it on additional datasets. For instance, we captured images of a dormitory room door from different angles. The algorithm consistently delivered comparable results, confirming its reliability and robustness across varying scenarios. Overall, the project demonstrated the efficiency and adaptability of the algorithm in accurately detecting ArUco markers and aligning posters in diverse environments.

Gitlab Repository:
https://github.com/nirbhayborikar/OpenCV_and_augmented_reality-3d_object_detection_and_opeartion-