RWU Hochschule Ravensburg-Weingarten
University of Applied Sciences

Faculty for Electrical Engineering and Computer Science

Computer Vision
Project Report: Task 1

AUGMENTED REALITY

Guided by:

Prof. Dr. Stefan Elser

Submitted by:

Aditya Date - 36159

Master - Mechatronics

Gautam Kevadiya - 29112

Bachelor - Angewandte Informatik

April 28, 2023

Contents

Chapter 1

Introduction

Augmented Reality (AR) is an interactive technology which enriches the real world around us by blending or superimposing the digital content into the reality which is perceived by our eyes essentially creating an immersive experience. AR uses computer vision and image recognition to detect physical objects in the frame and then overlays virtual digital objects on it.

1.1 ArUco Markers

Numerous computer vision methods such as fiducial markers containing patterns or visual information are used to do the detection of objects and their processing in AR. A very stable version of such AR markers is developed by the university of Córdoba (UCO) thereby getting the name ArUco markers [1]. They are easily detected by the OpenCV (Open Source Computer Vision) library providing a wide range of toos for image processing.

This project proposes an algorithm to detect such a marker placed on the wall of room H238 and overlay a digital poster on it to see how it would look like before actually hanging it. The dataset 'Room with markers' uploaded by Prof. Dr. Stefan Elser on mooodle containing 11 images was used [2]. It consists of a 6X6 ArUco marker having the ID '0' photographed from various angles. The location of the marker is instrumental in executing the poster projection. This report will give a detailed description of the methodology followed to reach the desired target. A square poster of 'The Leaning Tower of Pisa' was chosen for this task shown below [3].

Figure 1.1.1: Poster for this project

Chapter 2

Marker Detection and Poster Projection Algorithm

2.1 Flowchart

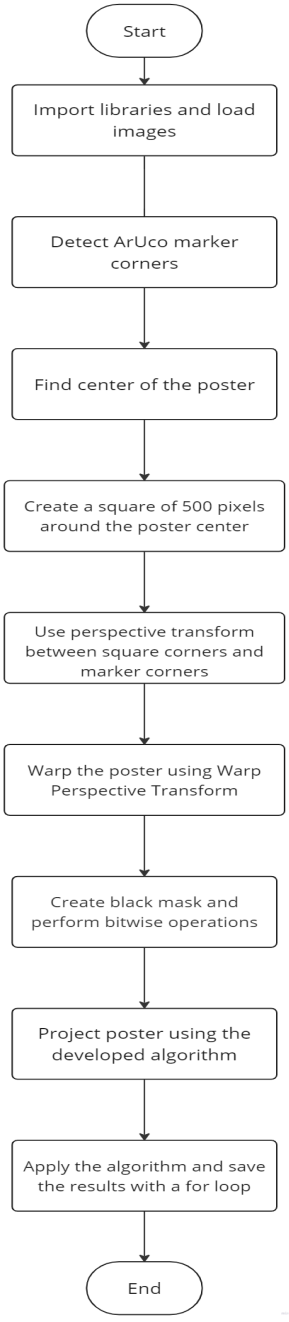

Figure 2.1.1: Algorithm Flowchart

2.2 Marker detection

The task was begun with importing necessary libraries followed by loading the dataset images and the poster using the in-built imread() function. The libraries Numpy, MatplotLib, CV2 were imported.

```
def read_image(image):
    return cv2.imread(image)
classroom = read_image('Angle  (1).jpg')
pisa = read_image('Pisa.jpg')
```

OpenCV contains an inbuilt function detectMarkers() which takes three inputs namely the source image, the dictionary which contains the marker set consisting of the matrix size and the IDs and the parameters which customize the detection process. The function finds the marker corners and then joins them with a green line to highlight them which can be seen below.

```
def detect_markers(image):
    '''Returns an image with highlighted corners and marker corner co-ordinates'''
    #Loading the dictionary that was used to generate the markers. Marker used in
                                            this task contains a 6X6 matrix.
    dictionary = cv2.aruco.Dictionary_get(cv2.aruco.DICT_6X6_1000)

    # Initializing the detector parameters using default values
    parameters =  cv2.aruco.DetectorParameters_create()

    # Detecting the marker in the image by obtaining its corners
    marker_corners, markerid, rejectedCandidates = cv2.aruco.detectMarkers(image,
                                            dictionary, parameters=parameters)

    highlighted_marker = cv2.polylines(image,[marker_corners[0][0].astype(int)],
                                            isClosed = True,(0,255,0), thickness
                                            = 3)
    return highlighted_marker
```

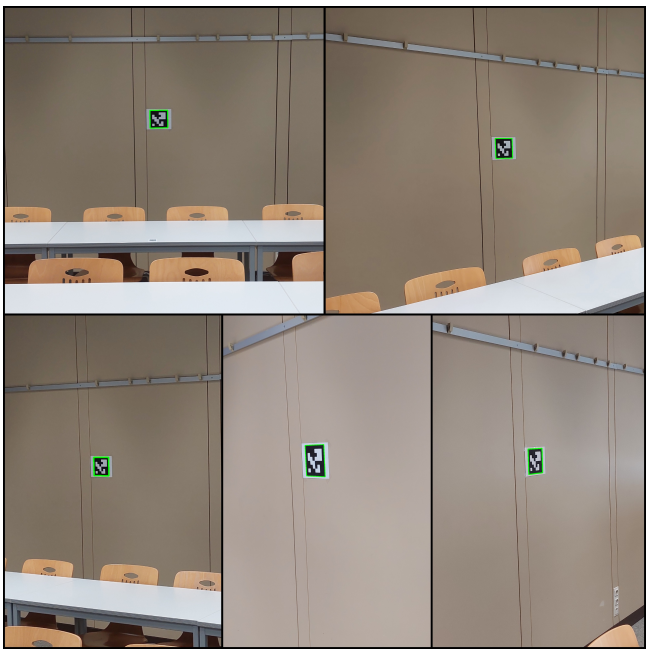

Figure 2.2.1: Highlighted markers

2.3 Selecting the corners in poster

After detecting the corner, the poster image processing was begun by finding the median and then forming a square of width 500 pixels around it. The width 500 was selected as it gave optimum projected poster size in all test images. Also, this method was followed instead of taking the end points of the frame as that would replace the ArUco marker with the poster which would not help in reaching the objective as the projection would be simply too small.

```
def grab_corners(image):
    points = np.array([
                       [0,0], #Origin
                       [(image.shape[1])-1,0], #Upper Right corner
                       [(image.shape[1])-1,(image.shape[0])-1], #Lower Right corner
                       [0,(image.shape[0])-1] #Lower Left corner
                       ])
    return points
def create_polygon_in_frame(polygon_width, polygon_height,frame_center):
    polygon_in_frame = np.array([ #upper left corner
                                  [frame_center[0]-0.5*polygon_width,
                                  frame_center[1]-0.5*polygon_height],
                                  #upper right corner
                                  [frame_center[0]+0.5*polygon_width,
                                  frame_center[1]-0.5*polygon_height],
                                  #lower right corner
                                  [frame_center[0]+0.5*polygon_width,
                                  frame_center[1]+0.5*polygon_height],
                                  #lower left corner
                                  [frame_center[0]-0.5*polygon_width,
                                  frame_center[1]+0.5*polygon_height]
                                  ])
    return polygon_in_frame.astype(int)
poster_corners = grab_corners(frame)
poster_center = np.median(frame_corners,axis = 0).astype(int)
polygon_in_poster = create_polygon_in_frame(polygon_width=500,
                                            polygon_height=500,
                                            frame_center=poster_center)
```

The following image visualizes this operation.

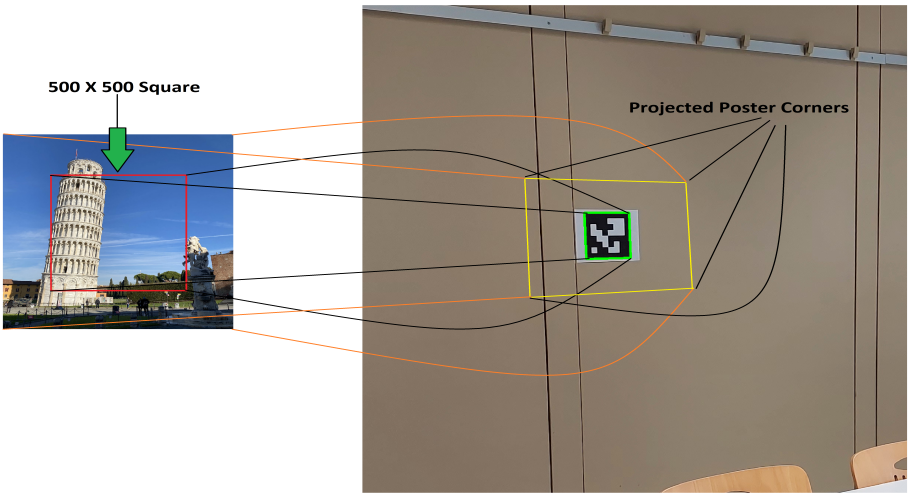

Figure 2.3.1: Warping operation visualized

2.4 Warping the poster

After finding the source and destination corners, the transformation matrix required to bring about the warping operation was found using the getPerspectiveTransform() function[4]. Then warpPerspectiveTransform() function was used along with the transformation matrix to generate the desired warping. The dsize parameter of the warping function is basically the size of the image on which the poster is projected.

```
#get transformation matrix
trans_mat = cv2.getPerspectiveTransform(polygon_in_poster.astype(np.float32),
                                        marker_corners[0][0].astype(np.float32))

#Apply the transformation matrix for warping
warped_image = cv2.warpPerspective(poster,trans_mat,dsize = (classroom.shape[1],
                                        classroom.shape[0]))
```

This produces the following output

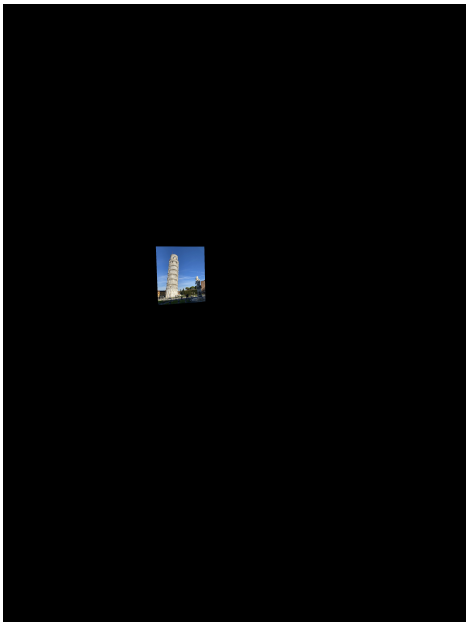

Figure 2.4.1: Warped poster

2.5 Blending the warped image onto the original photo.

Mixing the warped image with the original test image was achieved by creating a black mask and then performing three operations in series. Firstly, all the black pixels in the warped image were accessed by a for loop and the value of the corresponding pixels in the mask was changed to 255 (white). Then bitwise AND operation was performed on this changed mask and the test image to keep the poster projection area black. Lastly, the poster was projected onto this space by performing a bitwise OR operation on the warped image and the processed test image. However, this can sometimes cause problems as the warped photo can also contain black pixels which should be removed. Hence, they were removed and replaced by a value of 1 after loading the image using the following function so that it does not produce a big visual difference.

```
mask = np.zeros_like(pisa_warped) #create a black image having the same size as
                                    the warped image
for i in range(pisa_warped.shape[0]):
        for j in range(pisa_warped.shape[1]):
            if pisa_warped[i,j].any() == 0: #accessing every pixel and checking if
                                              it is pitch black
                mask[i,j] = 255             #changing the corresponding pixel
                                              value

#Bitwise AND operation
classroom_with_projection_area_black = cv2.bitwise_and(mask,read_image("Angle  (8)
                                       .jpg"))
#Bitwise OR operation to get the output
projected_poster = cv2.bitwise_or(pisa_warped,classroom_with_marker_black)

def remove_black_pixels(image):
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            if image[i,j].any() == 0 :
                image[i,j] = [1,1,1]
```

This process is demonstrated in the following images. The fourth image in the series is the output which contains the projected poster.

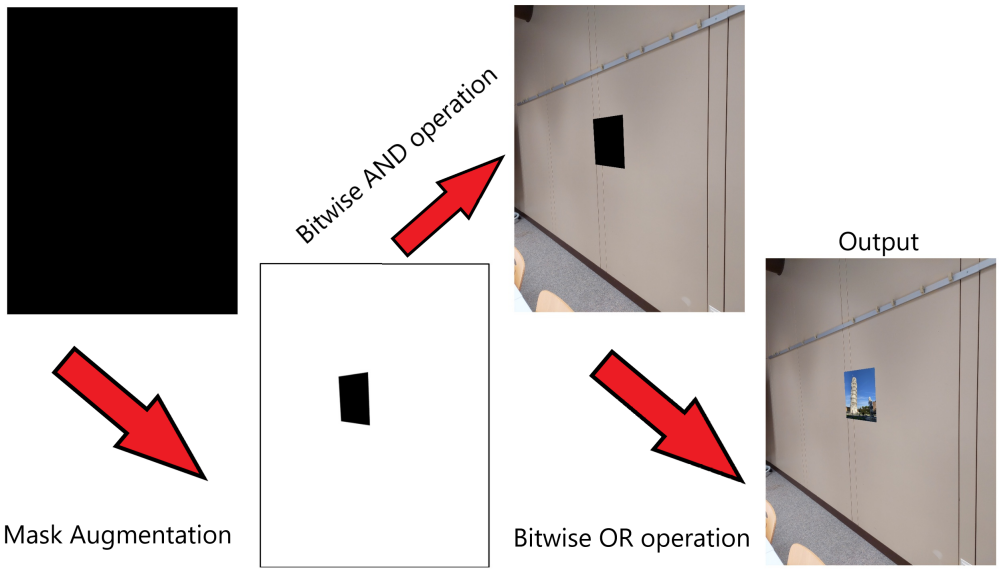

Figure 2.5.1: Masking and bitwise operations

Chapter 3

Algorithm Implementation

3.1 Summary of the algorithm.

In the above chapter it has been described how an image can be projected using an ArUco Marker and its associated OpenCV functions. The process was completed for a single image in a series of small functions for better understanding of the overall procedure. But for a group of images, this can be time consuming. Hence, a single algorithm compiling all the functions was developed. It has only two inputs, the poster and the image containing the ArUco marker.

```
def project_poster(poster, image_with_marker):
    poster = remove_black_pixels(poster) #remove all black pixels
    poster_corners = grab_corners(poster)
    poster_center = np.median(poster_corners,axis = 0)
    marker_corners = detect_markers(image_with_marker)

    #create polygon to be warped
    polygon_in_poster = create_polygon_in_frame(polygon_width=500, polygon_height=
                                                500, frame_center=frame_center)

    #calculate transformation matrix
    transf_mat = cv2.getPerspectiveTransform(polygon_in_poster.astype(np.float32),
                                             marker_corners.astype(np.float32))

    #apply image warping
    warped_image = cv2.warpPerspective(frame,trans_mat,dsize = (image_with_marker.
                                             shape[1],image_with_marker.shape[0]))
    #start poster projection using image blending
    mask = np.zeros_like(image_with_marker)
    for i in range(warped_image.shape[0]):
        for j in range(warped_image.shape[1]):
            if warped_image[i,j].any() == 0:
                mask[i,j] = 255
    image_with_marker_black = cv2.bitwise_and(mask,image_with_marker)
    augmented_frame = cv2.bitwise_or(warped_image,image_with_marker_black)
    return augmented_frame  #output image with the projected poster
```

Also, instead of manually running this algorithm for every image, the test images were accessed using a for loop and the poster was projected on them before finally saving on the disk.

```
poster = read_image('Pisa.jpg')
for i in [1,2,3,4,5]:
    classroom = read_image(f'Detected Markers\Images\Angle{i}_H.png') #load the
                                             image
    projected_frame = augment_image(poster,classroom) #project the poster on the
                                             image
    cv2.imwrite(f'Detected Markers\Projected_Poster{i}.png',projected_frame) #save
                                              the image
```

Because the last character of both the test and projected images is the same, it becomes easier to sort the results.

3.2 Evaluation

On running the above two cells, following outputs were generated. The function was applied on photographs of the classroom taken at various angles to test its performance.

Figure 3.2.1: Projected Posters

Although the frame was successfully projected, it was enlarged further for better evaluation. An enlarged preview of the poster can be seen in the following page.

Figure 3.2.2: Enlarged poster projections

Once the posters are enlarged, it becomes clearer that the projection edges are nearly parallel to the surrounding wall edges.

Chapter 4

Conclusion

With some of the inbuilt OpenCV funtions and the ArUco marker library, the markers in all images but one named '20221115_113412.jpg' could be correctly detected. For the specified image, tweaking the input parameter of the detectMarkers() function was attempted but resulted in no success. Also, there was no marker in one of the images so the detection algorithm was not applied on it.

On running the complete algorithm developed from individual functions, the poster was found to be pasted with the almost correct geometrical orientation on the test images except for the one in the top and bottom left corner on page 9 in which the top and lower poster edges do not seem to be parallel to the metal bar above and the floor edge respectively. One major reason could be the orientation of the placed marker. Apparently, the horizontal edges of the marker do not seem to be perfectly parallel to the edge of the metal bar and the floor or the white table edge respectively. Hence the projected image does not exactly match the perspective of the classroom.

All in all, the project could successfully give a preview of the poster projection on the wall to choose the most fitting one. But it can surely be improved by calculating the angle the ceiling and floor edges make with the horizontal. This data could be used along with trigonometry to rearrange the destination corners to make the poster perfectly parallel to the wall edges it will be pasted on. Such an algorithm would give an even accurate preview of the poster and help better in choosing the best poster for the purpose.

References

[1] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco Madrid-Cuevas, and Manuel Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47:2280–2292, 06 2014. Date Accessed: April 7, 2023.

[2] Stefan Elser. *Room with ArUco Markers*. `https://elearning.rwu.de/mod/folder/view.php?id=160309`, April 2023. [Dataset]. Date Accessed: April 6, 2023.

[3] Yvette. *Leaning Tower of Pisa*. `https://www.trip.com/moments/detail/pisa-1761-12132440`, January 2022. [Online]. Date Accessed: April 20, 2023.

[4] Sowmiya Narayanan. *What is Perspective Warping*. `https://pub.towardsai.net/what-is-perspective-warping-opencv-and-python-750e7a13d386`, September 2020. [Online Article]. Date Accessed: April 8, 2023.