Problem Set 3: Introduction to AR and Image Mosaic

September 20, 2021

ASSIGNMENT DESCRIPTION

Description

Problem Set 3 introduces basic concepts behind Augmented Reality, using the contents that you will learn in modules 3A-3D and 4A-4C: Projective geometry, Corner detection, Perspective imaging, and Homographies, respectively. Additionally, you will also learn how to insert images within images and stitch multiple images together.

Learning Objectives

- Find markers using corner detection and / or pattern recognition.
- Learn how projective geometry can be used to transform a sample image from one plane to another.
- Address the marker recognition problem when there is noise in the scene.
- Implement backwards (reverse) warping.
- Implement Harris corner detection to identify correspondence points for an image with multiple views.
- Address the presence of distortion / noise in an image.

Problem Overview

Methods to be used

In this assignment you are to use methods that work with Feature Correspondence and Corner detection. You will also apply methods that are part of Projective Geometry and Image Warping, however you will have to do these manually using linear algebra concepts.

Rules

You may use image processing functions to find color channels, load images, find edges (such as with Canny). Don't forget that those have a variety of parameters and you may need to experiment with them. There are certain functions that may not be allowed and are specified in the assignment's autograder Ed. post. Refer to this problem set's autograder post for a list of banned function calls.

Refer to this problem set's autograder post for a list of banned function calls.

Please do not use absolute paths in your submission code. All paths should be relative to the submission directory. Any submissions with absolute paths are in danger of receiving a penalty!

INSTRUCTIONS

Obtaining the Starter Files:

Obtain the starter code from the PS3 github repo.

Programming Instructions

Your main programming task is to complete the api described in the file **ps3.py**. The driver program **experiment.py** helps to illustrate the intended use and will output the files needed for the writeup.

Write-up Instructions

Create **ps3_report.pdf** - a PDF file that shows all your output for the problem set, including images labeled appropriately (by filename, e.g. ps3-1-a-1.png) so it is clear which section they are for and the small number of written responses necessary to answer some of the questions (as indicated). For a guide as to how to showcase your results, please refer to the Latex template for PS3.

How to Submit

Two assignments have been created on Gradescope: one for the report - **PS3_report**, and the other for the code - **PS3_code**.

- Report: the report (PDF only) must be submitted to the PS3_report assignment.
- Code: all files must be submitted to the PS3_code assignment. DO NOT upload zipped folders or any sub-folders, please upload each file individually. Drag and drop all files into Gradescope.

Notes

- You can only submit to the autograder **10** times in an hour. You'll receive a message like "You have exceeded the number of submissions in the last hour. Please wait for 36.0 mins before you submit again." when you exceed those 10 submissions. You'll also receive a message "You can submit 8 times in the next 53.0 mins" with each submission so that you may keep track of your submissions.

- If you wish to modify the autograder functions, create a copy of those functions and DO NOT mess with the original function call.

YOU MUST SUBMIT your report and code separately, i.e., two submissions for the code and the report, respectively. Only your last submission before the deadline will be counted for each of the code and the report.

Write-up Instructions

The assignment will be graded out of 100 points. The last submission before the time limit will only be considered. The code portion (autograder) represents **60%** of the grade and the report the remaining **40%**.
The images included in your report must be generated using experiment.py. This file should be set to be run as is to verify your results. **Your report grade will be affected if we cannot reproduce your output images.**
The report grade breakdown is shown in the question heading. As for the code grade, you will be able to see it in the console message you receive when submitting.

Assignment Overview

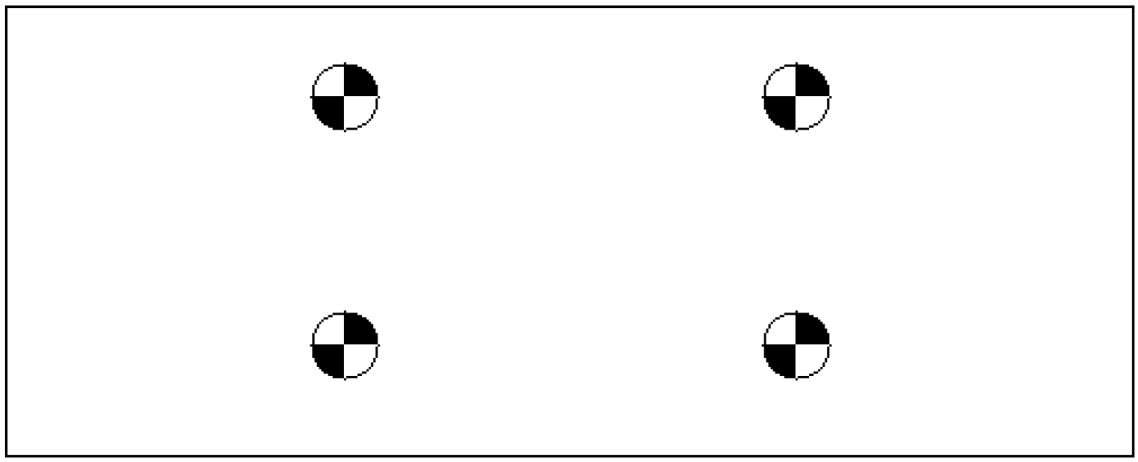
A glass/windshield manufacturer wants to develop an interactive screen that can be used in cars and eyeglasses. They have partnered with a billboard manufacturer in order to render certain marketing products according to each customer's preferences.

Their goal is to detect four points (markers) currently present in the screen's field-of-view and insert an image or video in the scene. To help with this task, the advertising company is installing blank billboards with four distinct markers, which determine the area's intended four corners. The advertising company plans to insert a target image / video into this space.

They have hired you to produce the necessary software to make this happen. They have set up their sensors so that you will receive an image / video feed and a target image / video. They expect an altered image / video that contains the target content rendered in the scene, visible in the screen.

1 MARKER DETECTION IN A SIMULATED SCENE [30]

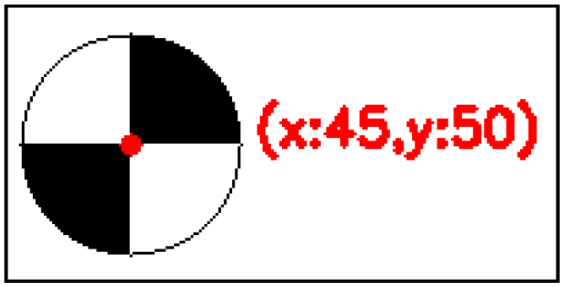
The first task is to identify the markers for this Augmented Reality exercise. In real practice, markers can be used (in the form of unique pictures) that stand out from the background of an image. Below is an image with four markers.



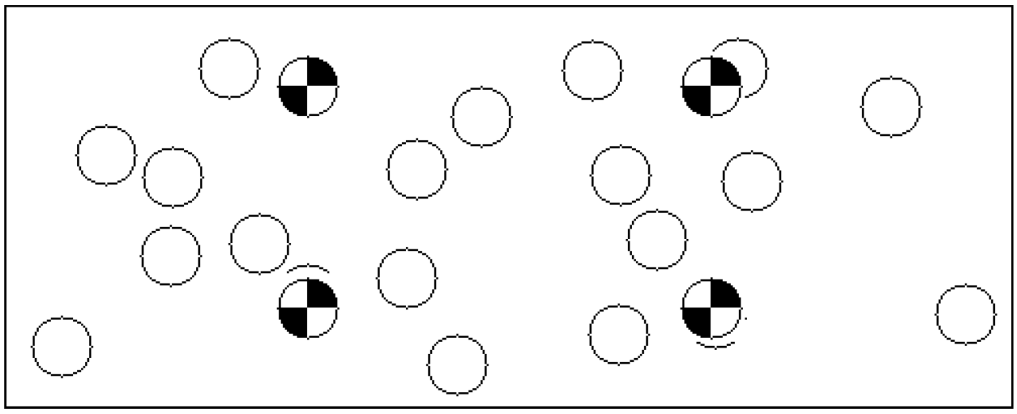
Notice that they contain a cross section bounded by a circle. The cross-section is useful in that it forms a distinguished corner. In this section you will create a function/set of functions that can the detect these markers, as shown above. You will use the images provided to detect the (x, y) center coordinates of each of these markers in the image. The position should be represented by the center of the marker (where the cross-section is). To approach this problem you should consider using techniques like detecting circles in the image, detecting corners and/or detecting a template.

Code: Complete find_markers(image)

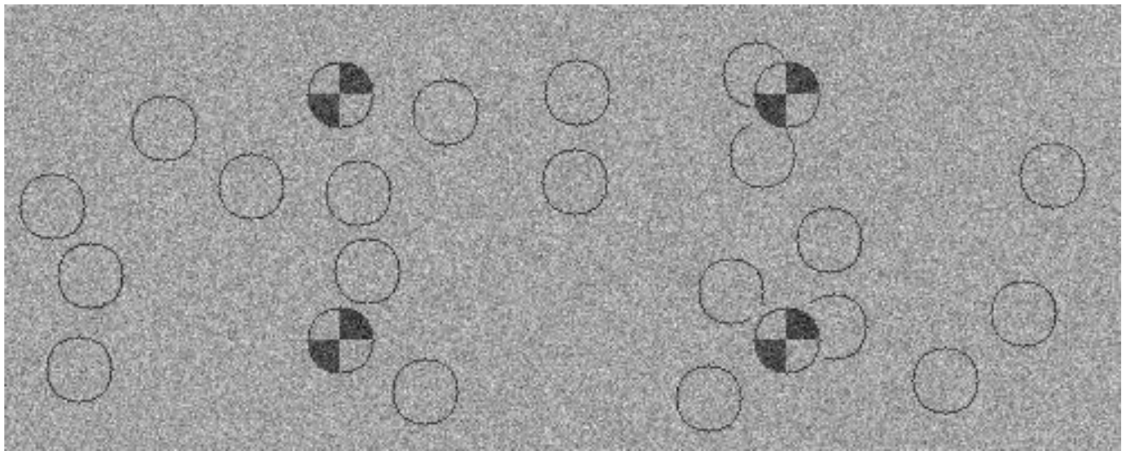
You will use the function mark_location(image, pt) in experiment.py to create a resulting image that highlights the center of each marker and overlays the marker coordinates in the image. Each marker should present their location similar to this:



Images like the one above may not be that hard to solve. However, in a real-life scene, it proves to be much more difficult. Make sure your methods are robust enough to also locate the markers in images like the one below, where there could be other objects in the scene:



Let's now assume there is "noise" in the scene (i.e. rain, fog, etc.).



Report: This part will be graded on auto-grader and there is no need to write it into report.

2 MARKER DETECTION IN A REAL SCENE [5]

Now that you have a working method to detect markers in simulated scenes, you will adapt it to identify these same markers in real scenes like the image shown below. Use the images provided to essentially repeat the task of section 1 above and draw a box (four 1-pixel wide lines, any color) where the box corners touch the marker centers.

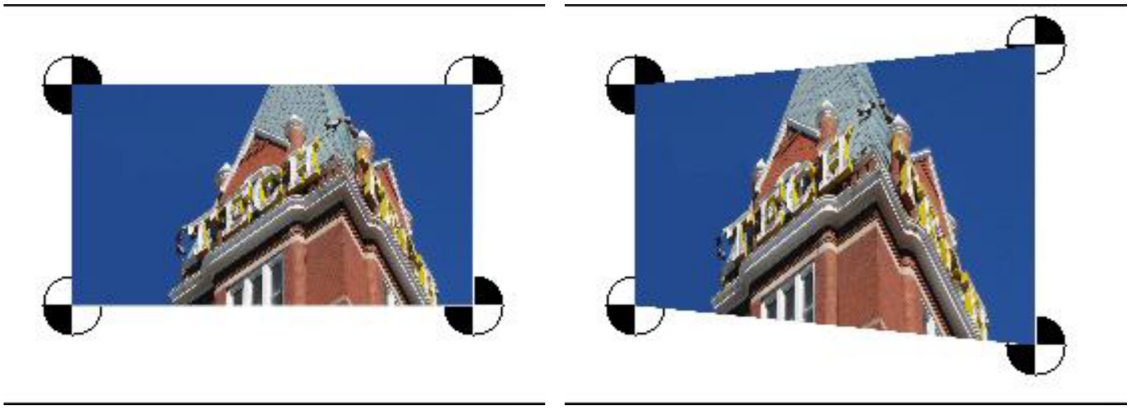
Code: Complete draw_box(image, markers)
Report: This part will be graded on auto-grader and there is no need to write it into report

3 PROJECTIVE GEOMETRY [5]

Now that you know where the billboard markers are located in the scene, we want to add the marketing image. The advertising company requires that their client's billboard image is visible from all possible angles since you are not just driving straight into the advertisements. Unphased, you know enough about computer vision to introduce projective geometry. The next task will use the information obtained in the previous section to compute a transformation matrix H. This matrix will allow you to project a set of points (x, y) to another plane represented by the points (x', y') in a 2D view. In other words we are looking at the following operation:

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

In this case, the 3x3 matrix is a *homography*, also known as a *perspective transform* or *projective transform*. There are eight unknowns, **a** through **h**, and **i** is 1. If we had four pairs of corresponding (u,v) $\Longleftrightarrow$ (u',v') points, we can solve for the homography.
The objective here is to insert an image in the rectangular area that the markers define. This insertion should be robust enough to support cases where the markers are not in an orthogonal plane from the point of view and present rotations. Here are two examples of what you should achieve:



When implementing project_imageA_onto_imageB() you will have to make the design choice of doing forward or backward wrapping. In order to make the right decision you should test both approaches and comment in the report what helped you chose a method over the other. (Note: to better see differences between the two methods you should pick a marketing image with low resolution).
Code: Complete:

- find_four_point_transform(src_points, dst_points)
- project_imageA_onto_imageB(imageA, imageB, homography)

Report: Report what wrapping technique you have used and comment on what led you to choosing this method.

4 FINDING MARKERS IN A VIDEO [5]

Static images are fine in theory, but the company wants this functional and put into practice. That means, finding markers in a moving scene.
In this part you will work with a short video sequence of a similar scene. When processing videos, you will read the input file and obtain images (frames). Once the image is obtained, you will apply the same concept as explained in the previous sections. Unlike the static image, the input video will change in translation, rotation, and perspective. Additionally there may be cases where a few markers are partially visible. Finally, you will assemble this collection of modified images into a new video. Your output must render each marker position relative to the current frame coordinates.
Besides making all the necessary modifications to make your code more robust, you will complete a function that outputs a video frame generator. This function is almost complete and it is placed so that you can learn how videos are read using OpenCV. Follow the instructions placed in ps3.py.

- First we will start with the following videos.
  - Input: **ps3-4-a.mp4**.
  - Input: **ps3-4-b.mp4**.
- Now work with noisy videos:
  - Input: **ps3-4-c.mp4**.
  - Input: **ps3-4-d.mp4**.

Code: Complete video_frame_generator(filename) **Report:** Report the 3 keyframes per video in the report.

5 Final Augmented Reality [10]

Now that you have all the pieces, insert your advertisement into the video provided. Pick an image and insert it in the provided video.

- First we will start with the following videos.
  - Input: **ps3-4-a.mp4**
  - Input: **ps3-4-b.mp4**
- Now work with noisy videos:
  - Input: **ps3-4-c.mp4**
  - Input: **ps3-4-d.mp4** - Frames to record: 207, 367, and 737 -
  - Output: **ps3-5-b-4.png, ps3-5-b-5.png, ps3- 5-b-6.png**

Report: In order to grade your implementation, you should extract a few frames from your last generated video and add them into the corresponding slide in your report.

In the next few tasks, you will be reusing the tools that you have built to stitch together 2 images of the same object from different view-points to create a combined panorama.

6 Finding Correspondence points in an image manually [5]

In this part of the project, you have to manually select correspondence points with mouse clicks from two views of the input image. The functions for this task will be provided to you in the class Mouse_Click_Correspondence(object). The points selected will have to be used to get the homography parameters. The sensitivity of the result would depend heavily on the accuracy of these correspondence points. Make sure to choose distinctive points in the image that are present in both the views. The functions in the class Mouse_Click_Correspondence(object) do not return anything and will create 2 numpy files (p1.npy and p2.npy) which will store the co-ordinates of the selected correspondence points.
Report: In order to grade your implementation, attach a screenshot of the points selected by you in the corresponding slide in the report.

7 Calculating homography parameters (Image Mosaic) [5]

In this task you will use the keypoints saved in the 2 files created in the previous section to calculate the homography parameters.

Code: Complete the following function from Image_Mosaic() class:

- get_homography_parameters()

Recall the concepts regarding homography from section 3.
The output of this task is the 3x3 homography matrix required for the perspective transformation.

8 Automatic Correspondence point detection [30]

In this task, instead of manually selecting the correspondence points, you will write code to automate this process. The inputs to this task are the 2 images and the output needs to be the

homography matrix of the required transformation. Use Harris Corner Detection to perform this task. The implemented solution must be able to work with RGB images. You should refer to module 4A_L2 to learn about Harris corners. Note that once you have found the corners for the two images we have provided a RANSAC approach to match corners from image 1 to image 2 in order to create keypoint pairs. Getting the homography matrix from the pairs of keypoints is left to you.

Code: Complete the following functions under the Automatic_Corner_Detection() class:

- gradients(self, image_bw)
- second_moments(self, image_bw, ksize=7, sigma=10)
- harris_response_map(self, image_bw, ksize=7, sigma=5, alpha=0.05)
- nms_maxpool(self, R, k, ksize)
- harris_corner(self, image_bw, k=100)

9 IMAGE STITCHING [5]

In this final task, you will be completing the code to perform the final image stitching and create the output mosaic. So far, you have calculated homography transform from one image to the other. Use perspective transformation to stitch the 2 images together.
Code: Complete the following fonction from the Image_Mosaic() class:

- image_warp_inv()

Recall concepts from section 3.
Report:

- Place in the report the two generated panorama images (one using the manually selected corners, and the other using the Harris Corner detection).
- Comment on the quality difference between the two outputs and how it relates to the importance of choosing the correct correspondence points for the image.