Homework 10: Image/Video Processing

ENGR 1050, University of Pennsylvania

April 12, 2023

This assignment consists of TWO portions. Problems in the **Warm up** section will be due **Saturday, April 15, by midnight (11:59 p.m.)**. The **Project** section will be due **Wednesday, April 19, by midnight (11:59 p.m.)**. Submit your answers on Gradescope using the instructions at the end of the handout. Late submissions will be accepted up to 48 hours after the deadline, but they will be penalized by 10% for each partial or full day late. After the late deadline, no further assignments may be submitted; post a private message on Ed Discussion to request an extension if you need one due to a special situation such as illness.

You may talk with other students about this assignment, ask the teaching team questions, use a calculator and other tools, and consult outside sources such as the Internet. However, **all submissions must be your own work, and you must write your comments by yourself.** Any submissions suspected of violating Penn's Code of Academic Integrity will be reported to the Office of Student Conduct. When you get stuck, post a question on Ed Discussion or go to office hours!

1 Install OpenCV

In the previous homework assignments, we learned how to work with matrices in NumPy. Two very important types of data that you will use matrices for are images and videos. There are many image and video-specific matrix operations and algorithms, so it is helpful to use a specialized image processing Python module. We will learn how to use OpenCV, which is built on top of both Matplotlib and NumPy. "CV" in OpenCV stands for Computer Vision, which generally refers to programming for image processing, analysis, and understanding.

First, let's install OpenCV. Open a terminal window and type the command:

```
pip install opencv-python
```

For more detailed instructions or troubleshooting, please refer to Homework 1 and previous posts on Ed Discussion about installing Numpy and Matplotlib. The same principles of troubleshooting and checking installation apply. Make a post on Ed Discussion or come to office hours if you have problems with installation!

Now, you can simply import the module any time you need to use one of its methods:

```
import cv2 as cv
```

2 Image Processing

Images are typically stored as 3D matrices in the following format:

- The rows/columns (dimensions 0 and 1) are the pixel locations in the image
- The third dimension is the color, with channel 0 = red, channel 1 = green, and channel 2 = blue; sometimes, there is a fourth channel α = transparency

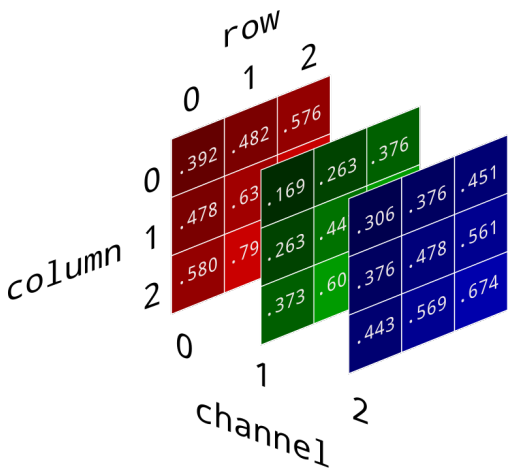


Figure 1: Visualization of an image as a 3D matrix. Image source.

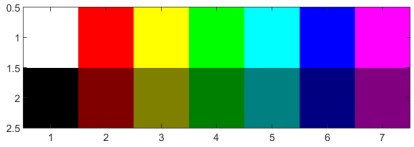
Take a look at the following code:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv

I = np.empty((2, 7, 3)) #creates an empty 3D matrix for an image of 2x7
I[:,:,0] = np.array([[1, 1, 1, 0, 0, 0, 1],
                     [0, .5, .5, 0, 0, 0, .5]]) #pixel values for the red channel
I[:,:,1] = np.array([[1, 0, 1, 1, 1, 0, 0],
                     [0, 0, .5, .5, .5, 0, 0]]) #pixel values for the green channel
I[:,:,2] = np.array([[1, 0, 0, 0, 1, 1, 1],
                     [0, 0, 0, 0, .5, .5, .5]]) #pixel values for the blue channel

#using Matplotlib to display the image
plt.imshow(I)
plt.show()
```

This code produces the following image:



Reading in the matrix, the image contains 2×7 pixels. The top left corner is the color (R=1, G=1, B=1), which corresponds to white. The top right corner is the color (R=1, G=0, B=1), which corresponds to magenta. All colors can be created from the RGB color palette in this way. Pixel values can also have the range of 0-255, which is the default in OpenCV, for which the same principles apply.

2.1 Warm-Up Problems (due April 15, 20 pts)

Complete the following exercises on manipulating images in Python.

1. (`maskPeppers.py`) A mask is a logical array that identifies particular pixels in an image and is very common in image processing. Write a script that loads in the peppers image and creates a mask (in this case, a 2D matrix), where `1` indicates all pixels with an R value greater than 190 and 0 for all other pixels. Display both the original image and the mask. Your output should look like

Hint: You might find the argument `cmap` of `imshow` useful.

2. (`smallPeppers.py`) Scaling down an image is a process that involves averaging pixel values in small blocks. Let's say we want to make an image smaller by a factor of 2. We would iterate through the image matrix and average each 2×2 block, storing that average pixel RGB value in a new matrix that is half the original size. Write a script to load in the peppers image, scale it down by a factor of 2, and display the result.

   Do not use any library (OpenCV, Matplotlib) functions for to resize or scale down the image - write your own version.

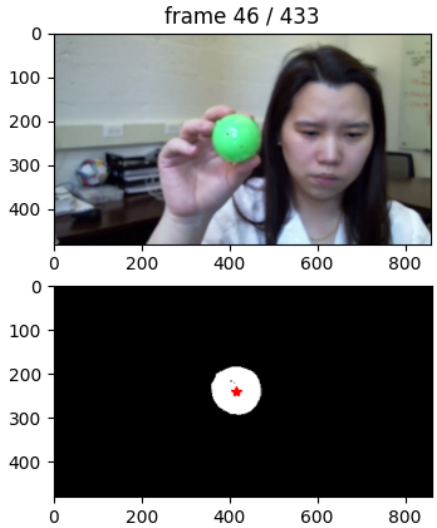
3. (`trackBall.py`) Submit pseudocode for the week's project. Write headers and function declarations for files you plan to create and outline their contents with your general approach as comments. You do not need to fill in the code.

Submit on Gradescope

1. maskPeppers.py
2. smallPeppers.py
3. trackBall.py

3 Video Processing

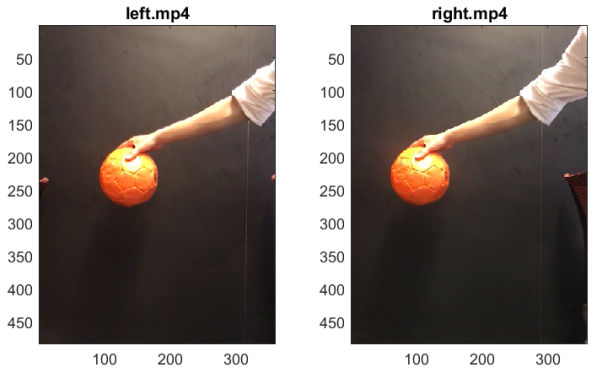
Videos are just images that are shown in sequence over time. The attached demo code shows how to load in, process, and display a video. The demo works by converting the image into an HSV (hue, saturation, value) representation to create a mask. In particular, the object we are tracking is green, so we check which pixels of the input frame are bright green with high saturation and value.



Deconstruct the demo code attached to this assignment to make sure you understand it. Try changing the values that are used to create the mask to see what happens.

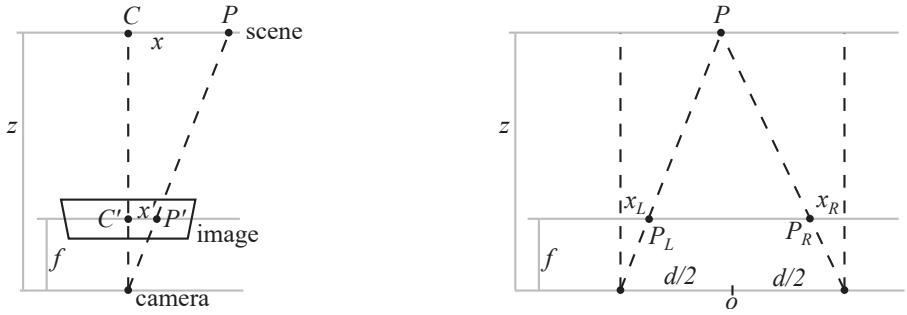
Your Mission for the Week: Stereo Vision (due April 19, 50 pts)

You will be expanding on this code to process two video feeds and extract 3D position data. The videos are synchronized to show two views of the same scene at the same time. You will be tracking an orange ball. Here are example frames from the two videos:



You will notice from the two frames that the ball is in a different position in the left vs the right view. We can use the difference in its position to figure out its distance from the camera.

Take a look at the diagram (figure 2a) below. A camera (bottom) is taking a picture of a point P in the scene in front of it. The point is projected towards the camera lens. It shows up on the camera image at the point P'.



(a) one camera (b) two cameras

Figure 2: How cameras interpret 3D scenes

If the point P were directly in front of the camera's center (e.g., the point C), then it would show up in the center of the image. If it is off-center, however, the distance of the point from the center of the image (x') is going to be proportional to the distance of P from C as

$$x' = \frac{f}{z}x \tag{1}$$

Now, normally, there's no way for us to tell what x is, so even if we measure x', we can't figure out z. But let's say we have TWO cameras looking at the same scene (figure 1b), and we know that they are distance d apart. In this case, we can use the same law of similar triangles to figure out that

$$\frac{z}{f} = \frac{d}{x_L + x_R} \tag{2}$$

Let's put the origin of our world coordinate frame halfway between the two cameras. Then, we can figure

out that the position of point P is

$$z = \frac{fd}{x_L + x_R} \tag{3}$$

$$x = \frac{z}{f} x_L - \frac{d}{2} \tag{4}$$

$$y = \frac{z}{f} y_L \tag{5}$$

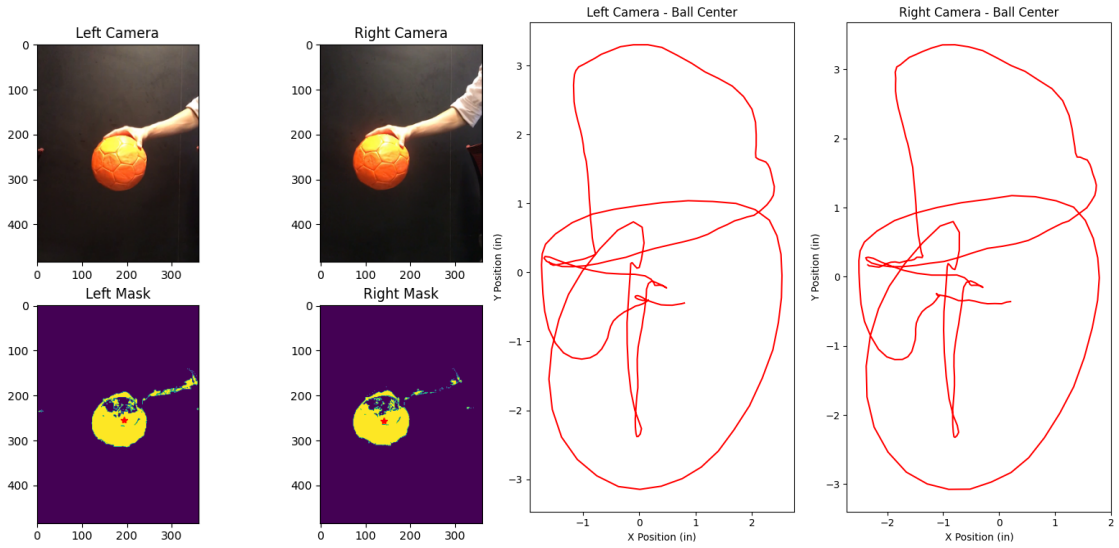
Instructions

1. Write a program `trackBall.py`, which loads in the two videos and computes the 3-D location of the ball over time.
   (a) You will need to create a mask identifying the pixels corresponding to the ball. You could operate directly on the RGB values or on the HSV representation. Experiment with your different options!
   (b) You can identify the center of the ball as the average of all coordinates of the pixels with 1s in them. The scale of the image is 0.02 in. / pixel. Use the following parameters: $d = 3$ in. and $f = 3$ in.
   Note that because images count rows of pixels down from the top of the image, you will need to flip your y axis coordinates.
2. Your script should produce the following:
   (a) An animation showing the side-by-side frames of the cameras and the output mask, formatted as shown in figure 3a.
   (b) A side-by-side plot of the 2D (x, y) positions of the center of the ball in image coordinates over time in each video. (This does not need to be animated!)
   (c) The x, y, and z coordinates of the ball over time. Use the frames per second (FPS) of the video to calculate how each coordinate corresponds to time.
   (d) A 3D plot of the trajectory of the ball. You may find the line `ax = fig.add_subplot(111, projection='3d')` helpful.
3. You should think about the structure of your code that you will need to do this task and keep your code clean. We recommend functions for the following (but you should also feel free to add your own!):
   (a) `calc_center_pixels`: This function applies a mask to the image and calculates the center of the ball in **pixel** coordinates.
   (b) `calc_center_inches`: This function calculates the center of the ball in **image** coordinates (inches). **Hint:** Make sure that the image coordinates origin matches the one shown in Fig. 2a.
   (c) `calc_3dtrajectory`: This function calculates the x, y, z positions of the ball in **scene** coordinates (inches) over time. **Hint:** Make sure that the scene coordinates origin matches the one shown in Fig. 2b and to consider Eqs. 3-5.
   (d) Additional functions for plotting as we normally see on homeworks.
4. Example plots are provided below so that you can check your plots. Put all of the plots (b, c, d) in a PDF and submit it to Gradescope. We will run your code to see (a).

Submit to Gradescope

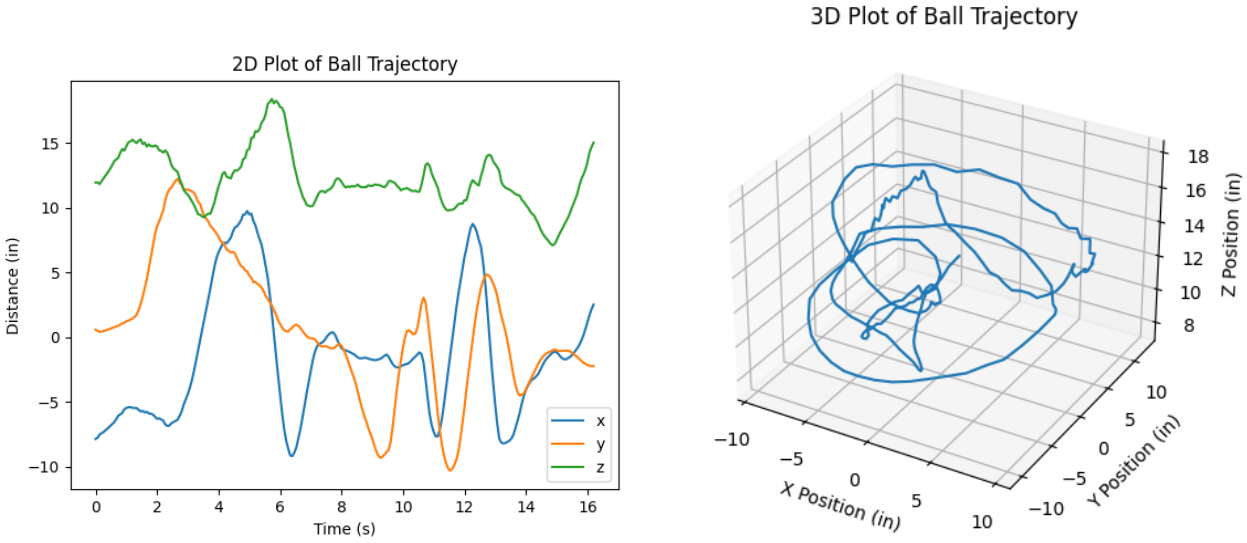
1. `trackBall.py`
2. Plots of the 2D image coordinates, trajectory, and 3D plot (Fig 3b-d) in a PDF document.

When grading, we will run your py file to check the output. You will receive feedback on Gradescope.



(a) animation

(b) 2D image coordinates



(c) trajectory

(d) 3D plot

Figure 3: Sample plots for complete video.