16720 A HW1: An Intro to Python and Computer Vision (100 Points)

Instructor: John Galeotti
TAs: Shumian Xin, Harsh Agarwal, Rishi Madhok, Talha Siddiqui

DUE ON: 11:59 PM 12 September 2019

This homework **WILL** be graded. It is meant to provide some Python/Scipy/OpenCV practice exercises to ensure that coding will not be an obstacle for you during this course. You are required to submit the working code and a write up with the results for each of the questions asked in the assignment. We have included the names of the TA's who would be grading that particular question, so that in case of specific doubts, you could direct your questions to them. This will expedite the process of doubt clarifications!

Please follow the submission instructions at the end of the assignment. This and future assignments will assume that you are using Python 3+ and the latest Scipy libraries (numpy, matplotlib, etc). The recommended approach is to use the `Anaconda Python 3 distribution` .

NOTE: Handwritten submissions will not be accepted!

1 Color Channel alignment(25 Points, Talha Siddiqui)

You have been given the red, green, and blue channels of an image[1] that were taken separately using an old technique that captured each color on a separate piece of glass[2]. These files are named `red.npy`, `green.npy`, and `blue.npy` respectively (in the `data` folder). Because these images were taken separately, just combining them in a 3-channel matrix may not work. For instance, Figure 1 shows what happens if you simply combine the images without shifting any of the channels.

Your job is to take 3-channel RGB (red-green-blue) image and produce the correct color image (closest to the original scene) as output. Because in the future you may be hired to do this on hundreds of images, you cannot hand-align the image; you must write a function that finds the best possible displacement.

The easiest way to do this is to exhaustively search over a window of possible displacements for the different channels (you can assume the displacement will be between

[1]downloaded from `http://www.loc.gov/pictures/collection/prok/`
[2]Credit to Kris Kitani and Alyosha Efros for this problem.

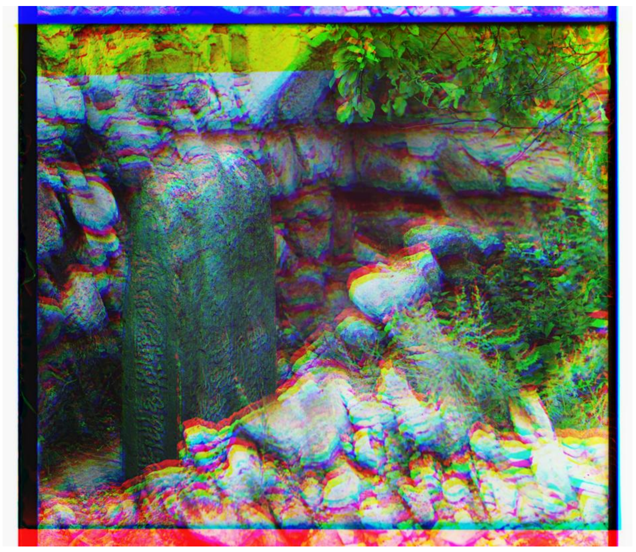


Figure 1: Combining the red, green, and blue channels without shifting

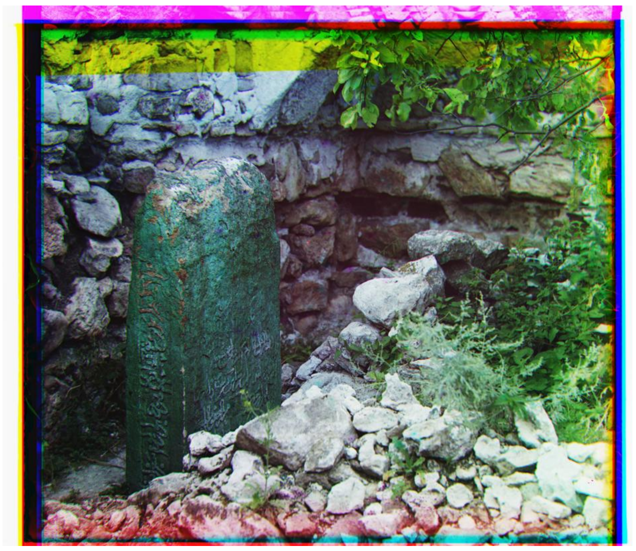


Figure 2: Aligned image

-30 and 30 pixels). Score the alignment from each possible displacement with some heuristic and choose the best alignment using these scores. You can use the following metrics:

1. Sum of Squared Differences (SSD)

   This metric tries to compare the distance between two vectors, hence we can use this to compare image pixel intensity differences. For two vectors **u**, **v** of length N, this metric is defined as

   $$SSD(\mathbf{u}, \mathbf{v}) = \sum_{i=1}^{N} (\mathbf{u}[i] - \mathbf{v}[i])^2$$

   (essentially like the Euclidean distance metric).

2. Normalized Cross Correlation (NCC)[3]

   This metric compares normalized unit vectors using a dot product, that is for vectors $\mathbf{u}, \mathbf{v}$,

   $$NCC(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}}{\|\mathbf{u}\|} \cdot \frac{\mathbf{v}}{\|\mathbf{v}\|}$$

Implement an algorithm which finds the best alignment of the three channels to produce a good RGB image; it should look something like Figure 2. Try to avoid using for loops when computing the metrics (SSD or NCC) for a specific offset. Some starter code is given in `script1.py` and `alignChannels.py`. Save your result as `rgb_output.jpg` in the `results` folder. **Also include the resulting image in the writeup**

2 Image warping (25 Points, Rishi Madhok)

We will be writing code that performs affine warping on an image. Code has been provided to get you started. In the following sections you will be adding your own code.

2.1 Example code

The file `script2.py` contains example code which demonstrates basic image loading and displaying as well as the behavior of an image warping function. Try running `script2`. You should see something like Figure 3. This script calls the function `warp` in `warpA_check.py`, which is simply a wrapper for scipy's own function `scipy.ndimage.affine_transform`.

[3] `https://en.wikipedia.org/wiki/Cross-correlation#Normalized_cross-correlation`

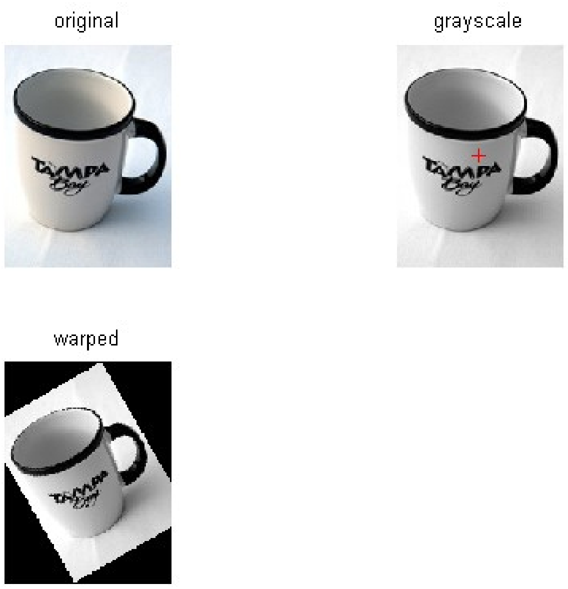


Figure 3: See script2.py

2.2 Affine warp

You will write your own function in `warpA.py`, that should give the same output as the function in `warpA_check.py`. First we'll walk through what you need to know about affine transformations.

An affine transform relates two sets of points:

$$\mathbf{p}^i_{warped} = \underbrace{\begin{pmatrix} a & b \\ c & d \end{pmatrix}}_{\mathbf{L}} \mathbf{p}^i_{source} + \mathbf{t} \tag{1}$$

where $\mathbf{p}^i_{source}$ and $\mathbf{p}^i_{warped}$ denote the 2D coordinates (e.g., $\mathbf{p}^i_s = (x^i_s, y^i_s)^T$) of the i-th point in the source space and destination (or warped) space respectively, $\mathbf{L}$ is a 2×2 matrix representing the linear component, and $\mathbf{t}$ is a 2×1 vector representing the translational component of the transform.

To more conveniently represent this transformation, we will use homogeneous coordinates, i.e., $\mathbf{p}^i_s$ and $\mathbf{p}^i_w$ will now denote the 2D homogeneous coordinates (e.g., $\mathbf{p}^i_s \equiv (x^i_s, y^i_s, 1)^T$), where "$\equiv$" denotes equivalence up to scale, and the transform becomes:

$$\mathbf{p}^i_d \equiv \underbrace{\left(\begin{array}{cc|c} \multicolumn{2}{c|}{\mathbf{L}} & \mathbf{t} \\ 0 & 0 & 1 \end{array}\right)}_{\mathbf{A}} \mathbf{p}^i_s \tag{2}$$

- Implement a function that warps image `im` using the affine transform `A`:

```
warp_im = warpA(im, A, output_shape)
```

Inputs: `im` is a grayscale `double` typed Numpy matrix of dimensions height$\times$width$\times$1 [4], `A` is a 3×3 non-singular matrix describing the transform ($\mathbf{p}^i_{warped} \equiv \mathbf{A}\mathbf{p}^i_{source}$), and `output_size=[out_height,out_width];` of the warped output image.
Outputs: `warp_im` of size `out_size(1)` $\times$ `out_size(2)` is the warped output image. The coordinates of the sampled output image points $\mathbf{p}^i_{warped}$ should be the rectangular range $(0, 0)$ to $(width - 1, height - 1)$ of integer values. The points $\mathbf{p}^i_{source}$ must be chosen such that their image, $\mathbf{A}\mathbf{p}^i_{source}$, transforms to this rectangle.

Implementation-wise, this means that we will be looking up the value of each of the destination pixels by sampling the original image at the computed $\mathbf{p}^i_{source}$. (Note that if you do it the other way round, i.e., by transforming each pixel in the source image to the destination, you could get "holes" in the destination because the mapping need not be 1 to 1). In general, the transformed values $\mathbf{p}^i_{source}$ will not lie at integer locations and you will therefore need to choose a sampling scheme; the easiest is nearest-neighbor sampling (something like, `round(`$\mathbf{p}^i_{source}$`)`). You should be able to implement this without using `for` loops (one option is to use `numpy.meshgrid` and Numpy's multidimensional indexing), although it might be easier to implement it using loops first. Save the resulting image in `results/transformed.jpg` . **Also include the resulting image in the write up.**

You should check your implementation to make sure it produces the same output as the `warp` function provided in `warpA_check.py` (for grayscale or RGB images). Obviously the purpose of this exercise is practicing Python/Scipy by implementing your own function without using anything like `scipy.ndimage.affine_transform`.

3 Hough Transform(50 Points, Harsh & Shumian)

We learned about Hough Transform in the class. It's a simple classical vision technique that can help us in finding line segments in an image. In the previous questions we have learned how to read images, perform transformations and so on. As a Computer Vision Engineer, we want you to learn how to look up for existing libraries and effectively use them to make an end to end system to perform a specific task (detecting line segments here!)

Your system takes in an image and performs a couple of operations to return you the line segments in an image using Hough Transform! We have given you a couple of images to test your implementation.

But having libraries does not mean you can skip over the math. So the first section of the assignment consists of a couple of theory questions to check your fundamental grasp on Hough Transform. The programming part would help you in understanding how to setup libraries and use the inbuilt libraries to get a working system!

[4] Images in Numpy are indexed as `im(row, col, channel)` where `row` corresponds to the y coordinate (height), and `col` to the x coordinate (width).

3.1 Theory Questions(20 points, Shumian Xin)

Type down your answers for the following questions in your write-up. Each question should only take a couple of lines. In particular, the "proofs" do not require any lengthy calculations. If you are lost in many lines of complicated algebra you are doing something much too complicated (or perhaps wrong). Each question from Q2.1 to Q2.4 are worth 5 points each.

Q2.1 Show that if you use the line equation $x \cos\theta + y \sin\theta - \rho = 0$, each image point (x, y) results in a sinusoid in (ρ, θ) Hough space. Relate the amplitude and phase of the sinusoid to the point (x, y).

Q2.2 Why do we parameterize the line in terms of (ρ, θ) instead of slope and intercept (m, c)? Express the slope and intercept in terms of ρ and θ.

Q2.3 Assuming that the image points (x, y) are in an image of width W and height H (i.e., $x \in [1, W], y \in [1, H]$), what is the maximum absolute value of ρ and what is the range of θ?

Q2.4 For points $(10, 10), (15, 15)$ and $(30, 30)$ in the image,plot the corresponding sinusoid waves in Hough space (ρ, θ) and visualize how their intersection point defines the line (what is (m, c) for this line?). Please use matplotlib in python to plot the curves and report the result in your write-up.

3.2 Getting started with OpenCV!

We already have the latest version of Anaconda and conda running on your systems. We would recommend creation of a virtual environment with the with python version 3.7 installed. Using anaconda you can use the following command to create a new virtual environment:

```
conda create -n cv_assignment python=3.7
```

The following command will help you in installing the latest build of OpenCV.

```
conda install -c conda-forge opencv
```

Check your OpenCV installation in python by typing the following command:

```
import cv2
```

If the above works without any kind of errors you are good to go. If errors persist, try googling, stack overflowing and make it work. This step is going to save you a lot of time in future!

3.3 Let's get the Hough Transform running!(10 points, Harsh Agarwal)

In the code folder you have been given a script:

```
houghTrans.py
```

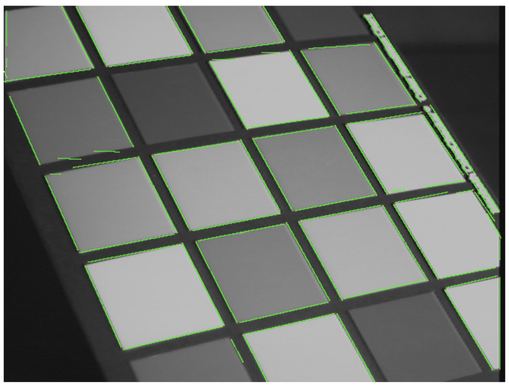
You are supposed to complete the function myHough in the provided script. To test your implementation, fill in the parameters in the main function and run the script. Make sure that you save the results obtained for each of the given test image in the results folder! So let's say your test image name is `img01.jpg` so the corresponding result should be in the `results` folder with the name `img01_hlines.jpg`

3.4 Write Up(20 points, Harsh Agarwal)

In your write-up elaborate upon the following:

1. The inbuilt functions/libraries that you have used and what arguments/parameters they take as input
2. The mathematical significance of every parameter that is being passed to the function.
3. Include results obtained on any 5 test images on various parameter settings. Justify the effect using the ideas from the previous bit!

A ideal image obtained for the `img01.jpg` would be as follows:



4 Submission and Deliverables

Please submit a zip file to Canvas/Gradescope(will be confirmed on Piazza) for this homework of the form `<hw1_andrewID>.zip`, and write up with the name `hw1_andrew_id.pdf` to Gradescope. Running the scripts in the `code` folder should generate the desired results in the `results` folder, as explained in each of the section. You should not include the `data/` folder in your submission. An example submission zip file should be:
`hw1_sigbovik.zip`:

- `code/`
  - `script1.py`
  - `script2.py`
  - `warpA.py`
  - `warpA_check.py`

  - `houghTrans.py`
  - any other files your code needs to run

To Gradescope, you will submit the writeup in the pdf form:

- `hw1_sigbovik.pdf`