CSC320 — Introduction to Visual Computing, Spring 2023

Assignment 1: Homographies

Posted: Tuesday, January 17, 2023
Due: 11:59am, Tuesday February 7, 2023
Late policy: 15% marks deduction per 24hrs, 0 marks if > 5 days late.
Submission website: https://markus.teach.cs.toronto.edu/2023-01/courses/18

In this assignment you will implement and experiment with a document scanner tool that can let you unwarp images based on keypoints. The functionality is similar to that of Microsoft's Office Lens or Adobe Scan. This tool is based on homographies, which is a matrix transform that relates two planar surfaces in space as discussed in Lectures 1 and 2. Your specific task is to complete the technique's implementation in the starter code. The code has an optional app that you can use to test your algorithm and play around with the document scanner interactively.

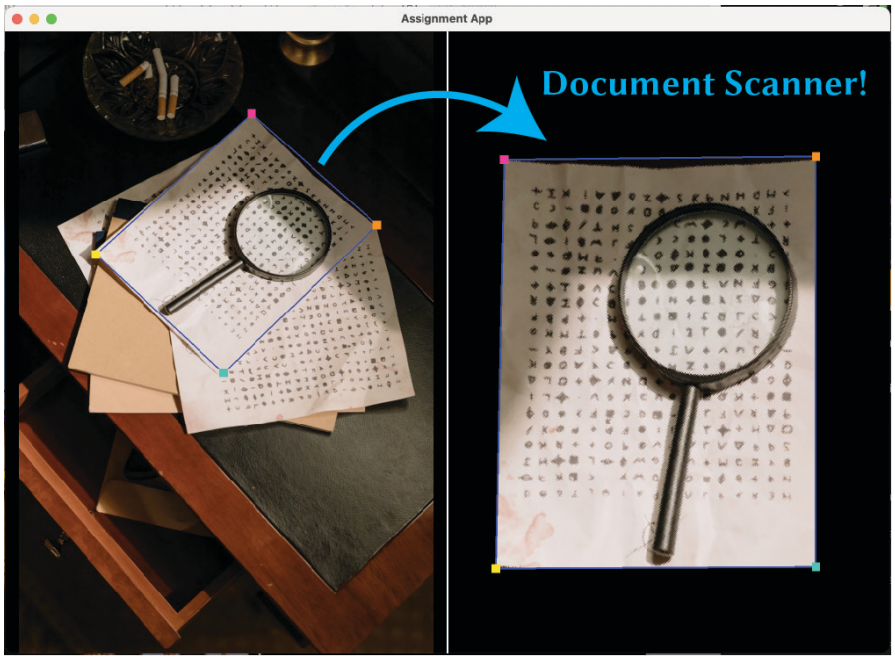


Figure 1: You will make a cool document scanner!

Goals: The goals of this assignment are to (1) get you familiar with working with images, Python, and NumPy; (2) learn how to implement basic geometrical operations like transformations and mapping; and (3) learn some insights of what kind of difficulties arise when you geometrically manipulate images.

Bonus: We suggest that you implement the transformations in the most naive way possible using for loops; there are ways to implement it *much* faster by taking advantage of NumPy and vectorization. Although there are no explicit bonus marks for efficient code in this assignment, you should discuss any improvements you made in your report. Exceptional efforts in this direction may be rewarded with a bonus on a case-by-case basis.

Important: You should start by getting a high level idea of what the code does by reading this document. Then, try to run the reference solution by following the instructions in the section below. Once you get the high level idea of what the code does, take a look at `app/a1/a1_headless.py` (where the app is implemented) and `viscomp/algos/a1.py` (what you need to complete) to understand the overall flow of things. Everything you need to implement is in `viscomp/algos/a1.py`. You do not need to make any other changes. **What you should not do:** you may **not** rely on OpenCV or any external computer vision or image processing library to implement the assignment. You should be able to implement everything just purely with NumPy and Python.

Testing your implementation: Your implementation will be tested on the teaching labs Linux machines, by running `app/a1/a1_tests.py`. Make sure you can run these tests by following the instructions below and check that your output looks fine.

Part A. Homography Algorithm (85 Marks)

In this section, you will implement the two algorithms:

1. **Homography Matrix**: Calculating a homography matrix from 2 quadrilaterals.
2. **Backward Mapping**: Looping through every destination pixel and using the homography to find the corresponding source pixel.

Both of these functions and their skeletons can be found in `viscomp/algos/a1.py`. Your implementation will wholly reside in this file. You will not need to make modification to anything else in the codebase.

Part A.1: Homography Matrix: The `calculate_homography()` function (45 marks)

We can estimate a homography H given sufficiently many point correspondences. Usually, many correspondences are used to compute H. However, for this assignment, we will use the minimum number of correspondences needed for a unique solution: **four**, under the condition that the four points are distinct, and define a (convex) quadrilateral. You will be implementing the code for determining H in `calculate_homography()`.

Let's start by writing down the equations for a single point p_i in the source image and its correspondence p'_i in the destination image, shown in the Figure 2 below:

Points p_i and p'_i have known coordinates, either provided by the values in the `.csv` files for the reference solution or extracted from the user's clicks in the interactive app. We can write them as:

$$p_i = \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}, \quad p'_i = \begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix}$$

and are related by the unknown homography matrix H

Figure 2: The goal of this algorithm is to find the mapping from the destination to source using the correspondences p_i and p_i'.

$$H = \begin{bmatrix} a & b & c \\ d & e & f \\ h & k & 1 \end{bmatrix}$$

such that $p_i' = Hp_i$. The eight values a, b, c, d, e, f, h, k are unknown and are what we want to solve for. For our point p_i, if we were to perform the multiplication $p_i' = Hp_i$, we would find

$$\begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} = \begin{bmatrix} (ax_i + by_i + c) \,/\, (hx_i + ky_i + 1) \\ (dx_i + ey_i + f) \,/\, (hx_i + ky_i + 1) \\ 1 \end{bmatrix}$$

Therefore, we find that that one point correspondence p_i and p_i' can give us two equations. Four point correspondences will then provide us with eight equations, exactly what we need for our eight unknowns.

Thus, your work for `calculate_homography()` is to take in as input the two arguments

- `source`: an `np.ndarray` of size 4x2 containing the 2D coordinates (x_i, y_i) of 4 points p_i in the source image;
- `destination`: an `np.ndarray` of size 4x2 containing the 2D coordinates (x_i', y_i') of 4 points p_i' in the destination image;

and return as output the 3x3 homography matrix H by solving for a, b, c, d, e, f, h, k.

Part A.2: Backward Mapping: The `backward_mapping()` function (40 marks)

Now that we have our homography matrix H, we can apply it to portion of the source image in the convex region outlined by the four source correspondence points. This is the task for `backward_mapping()`, where you will pass in as input

- `transform`: your 3x3 homography matrix solved for in `calculate_homography`
- `source_image`: the H_s x W_s x 4 image that is used as the source to read from.
- `destination_image`: the H_d x W_d x 4 image that is used as the destination to write to.
- `destination_coords`: the 2D locations of the 4 corners on the destination.

and return the backward mapped image. This task will require looping through every destination pixel, finding which points are inside the convex polygon, using the homography H to find the corresponding source pixel, and writing to that pixel with the destination pixel. Make sure to check your matrix dimensions when implementing the multiplication!

Part B: Experimentation (15 Marks)

Part B.1: Trying it on your own data! (5 Marks)

Your task is to run the algorithm on your *own* data. Specifically:

1. Take your own pictures using a smartphone camera or equivalent of some document with interesting patterns, at an oblique angle (meaning the document should not be perpendicular to your line of sight). Run your code on at least five different images of different surfaces and imaging conditions (i.e., different viewing angles, different plans, different patterns being imaged.
2. Make a new directory: `app/a1/report/`
3. Resize your images to something reasonable, like 720px on one side of the image. Save them to directory `app/a1/report/`. (What is the problem with using an image that is too large?)
4. Using some image viewing tool (like Preview on OSX) to find 4 corners (remember the order!) of the quadrilateral on the image. The output quadrilateral should be a simple rectangle of some sort. Save these as `app/a1/report/image_source.csv` and `app/a1/report/image_destination.csv`.
5. Use `app/a1/a1_headless.py` to run your algorithm using these as input images. Save the output as `app/a1/report/output.jpg`.

Part B.2: Write a lab report (10 Marks)

Your report should include the following: (1) Your name, student number, and student username; (2) images of the input and results; (3) the images used, and (4) a qualitative discussion of the results. This discussion should cover the following: (1) what artifacts do you see and what causes them; (2) what limitations does the algorithm have (try to be as analytic as possible); and (3) your ideas about

how you might improve the results. Your report should be in PDF format. LaTeX, Word or even powerpoint can be used to create the submitted PDF. Bear in mind, however, that the report should be in a readable form, not just a bunch of photos with annotations and/or a couple of sentences. Assume the marker knows the context of all questions, so do not spend time repeating material from this handout or from the lecture slides.

Place your report as a file `app/a1/report/report.pdf` (**Important:** Save this as a *PDF*, **not** `.docx` or other!).

What to turn in

You will be submitting the completed CHECKLIST form, your code, your written report and images. Use the following sequence of commands to pack everything up:

```
cd CSC320
tar cvfz assign1.tar.gz viscomp
```

Important: After uploading the tarfile to MarkUS, download it from MarkUS, unpack it and **verify that your code and all other assignment components are there!** Students in previous instances of the course have accidentally submitted the starter tarfile instead of their own code and results, leading to major issues with marking—and wasting a lot of time for all involved. Because of this, there will be **no accommodation** for such errors this term: if you submit the starter code instead of your own code and/or submit no results, you will receive a zero on those parts of the assignment; if you discover this error a day later and want to re-submit, the standard late submission policy will apply. As stated on the course website, the course's late policy includes a brief grace period to allow you to quickly resubmit your tarfile a few minutes after the submission deadline without any penalty if you discover omissions in your submitted tarfile.

Setting up your environment

`conda` is a great tool for helping us handle Python packages and their dependencies library dependencies in a stable and sandboxed environment. If you are unfamiliar with conda, please head over to their website for more info. Otherwise, let's get started with the installs needed for this assignment. Make sure your `PATH` variable is updated.

Important: Although not recommended, it is possible to skip `conda` installation by skipping straight to the `pip install` command. However, we highly recommend that you get familiar with `conda` because this is a very common way of setting up Python environments that don't collide with each other that is standard in both industry and academia.

First, create a conda environment and install dependencies with the following commands. The first command will create a new environment where all of your packages for the assignment will live. In conda, you can switch between environments with `conda activate`. Make sure you are in the correct environment whenever you run Python code.

```
conda create -n csc320a1 python=3.9
conda activate csc320a1
```

Download the assignment from Dropbox, and use these commands to unpack:

```
tar xvfz homography.tar.gz
rm homography.tar.gz
cd CSC320/viscomp/
```

Then, install all of the dependencies. Make sure you run a command like `which pip` to make sure that the `pip` being used is the one installed through conda.

```
pip install --upgrade pip
pip install -r requirements.txt
```

Install `viscomp`. This following command will make the code inside `viscomp` available externally as a package that you can import in Python.

```
python setup.py develop
```

Installing the interactive app (Optional)

If you want to use the interactive parts of the code and the assignment, you need additional requirements:

```
pip install -r requirements_app.txt
```

Windows-specific installs

For Windows users, you may have to go through the following additional steps in order to install `glumpy` in `requirements.txt`:

- Microsoft Visual C++ 14.0 or greater is required. Get it with Microsoft C++ Build Tools
- Download and place `freetype.dll` in the top level of this assignment. Instructions and download links can be found here

OSX-specific installs

For OSX users, you may have to also specifically install freetype with:

```
conda install freetype
```

Tour of the codebase

Assignment implementation for students

`viscomp/algos/a1.py` contains the meat of the assignment that you will be implementing. This is the **only** file you will need to modify for your assignment. Future assignments will also be sandboxed in the `viscomp/algos/` directory, building off of our existing codebase.

Scripts for executing implementation

`app/a1/a1_app.py` contains code for launching interactive app. See section below for running instructions.

`app/a1/a1_headless.py` contains code for running homography and warping in a headless way. See section below for running instructions.

`app/a1/a1_tests.py` contains code for running batch of tests on teaching lab machines. See section below for running instructions.

Reference solution images and 2D coordinates for testing

`app/a1/data/pexels-publicdomain/` contains images for reference solutions.

`app/a1/tests/` contains `.csv` files of four distinct 2D coordinate pairs for each image in `data/pexels-publicdomai`

Miscellaneous files and directories

Remaining scripts and directories such as `viscomp/ops`, `viscomp/base_app.py`, `viscomp/config_parser.py` are tools for launching and running the app/implementation.

Starter Code & Reference Solution

Running the reference solution

First, download the reference solutions and unpack them. You can then run the reference solutions with:

```
cd app/a1
python a1_headless.py --image-path data/pexels-publicdomain/building1.jpg \
                      --source-path tests/building1_source.csv \
                      --destination-path tests/building1_destination.csv \
                      --output-path results/building1 \
                      --reference-solution
```

This will output an file called `building1_output.png` to `results`. This image should be more or less identical to the image located in `test_results/building1_output.png`.

A closer look at the code for running the reference solution

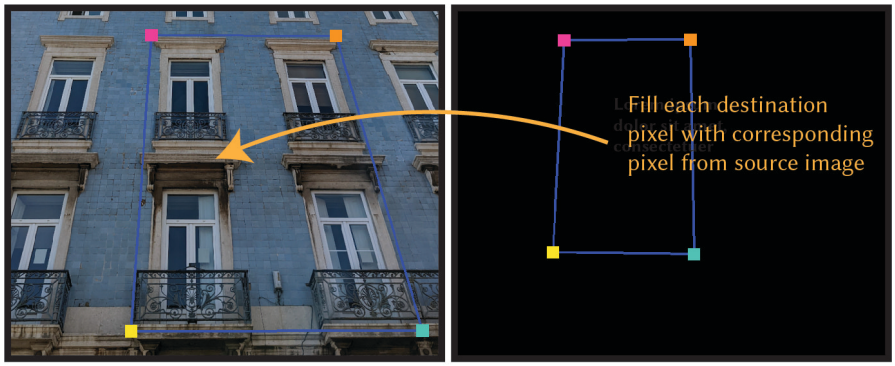
First, the code takes in an input image, which you specify with the command line (CLI) argument `--image-path`. The image should be a `.jpg` or `.png`. Next, we pass in two `.csv` files labelled with the arguments `--source-path` and `--destination-path`. The two `.csv` files contain the four distinct 2D coordinates that define a (convex) quadrilateral (a convex 4-sided polygon).

- The `--source-path` input defines a quadrilateral on the input image, in integer pixel coordinates where `0, 0` is the bottom left corner of the image, and `height, width` is the top right corner of the image.
- The `--destination-path` defines a quadrilateral on an **empty** image, also in integer pixel coordinates space.

These two quadrilaterals define 2 planes, for which their geometrical relationship is defined via the homography matrix.

Important: The 2D coordinates are in the order of: top left, top right, bottom right, bottom left. In other words, they are in **clockwise** order. See the image below for an example.

The goal of the algorithm that you will be implementing in this assignment is to find a way to take that empty quadrilateral (the destination quadrilateral), and fill it in with the contents in the quadrilateral on the input image (the source quadrilateral).



Source Image Destination Image

Please read the checklist in `app/a1/CHECKLIST.txt` carefully. It includes details on the distribution of marks in the assignment. You will need to complete this form prior to submission, and it will be the first file TAs look at when marking your assignment.

Running the interactive app

You can invoke an interactive app in which you will be able to interactively click on corners of your quadrilateral in clockwise order to run the algorithm. This app can also be useful for debugging your algorithm, since you'll get much more immediate feedback. It can be launched on your personal machines.

To run the interactive app on one of the provided reference images:

```
python a1_app.py --input-image data/pexels-publicdomain/tokyo0.jpg
```

Note that you have be in the `app/a1` folder to run this command.

Running on all provided sample images

In addition to `app/a1/a1_headless.py` which runs the algorithm on the specified input and `app/a1/a1_app.py` which launches the interactive app, we provide the option to run the algorithm on all provided sample images:

```
python a1_tests.py
```

This is a nice way to run your algorithm implementation on the batch of reference inputs. You can use this make sure all the outputs look good! Note that you have to be in the `app/a1` folder to run this command.

Other useful tips: Use a debugger! You can easily debug Python code by injecting `import pdb; pdb.set_trace()` wherever you want a breakpoint to enter the interactive Python REPL to print variables and run code.

Freely-available resources on NumPy, OpenCV, Computer Vision Programming, etc:

1. Short NumPy tutorial by Olessia Karpova (CSC420, 2014):
   https://github.com/olessia/tutorials/blob/master/numpy_tutorial.ipynb
2. Jan Erik Solem, *Programming Computer Vision with Python*, O'Reilly Media, 2012 (preprint):
   http://programmingcomputervision.com/ (look at Chapters 1 and 10)
3. *OpenCV-Python documentation* (Intro to OpenCV, Core Operations, Image Processing in OpenCV):
   https://docs.opencv.org/3.4.13/d6/d00/tutorial_py_root.html#gsc.tab=0
4. *Matplotlib User Guide* (especially Chapter 2.1.4):
   https://matplotlib.org/3.2.2/Matplotlib.pdf
5. *NumPy Reference Guide* (especially Chapters 1.4, 1.5, 4.17):
   https://numpy.org/doc/stable/numpy-ref.pdf

Academic Honesty. Cheating on assignments has very serious repercussions for the students involved, far beyond simply getting a zero on their assignment. I am very saddened by the fact that isolated incidents of academic dishonesty occur almost every year in courses I've taught. These resulted in very serious consequences for the students that took part in them. Note that academic offences may be discovered and handled retroactively, even after the semester in which the course was taken for credit. The bottomline is this: you are not off the hook if you managed to cheat and not be discovered until the semester is over!

You should never hand down code to students taking the course in later years, or post it on sites such as GitHub. This will likely cause a lot of trouble both to you and to the other students!

Use of chat-gpt for coding help or reports is also expressly forbidden.

Each assignment will have a written component and most will also have a programming component. The course policy is as follows:

- **Written components:** All reports submitted as part of your assignments in CSC320 are strictly individual work. No part of these reports should be shared with others, or taken from others. This includes verbatim text, paraphrased text, and/or images used. You are, however, allowed to discuss these components with others at the level of ideas, and indeed you are welcome to brainstorm together.
- **Programming components (if any):** Collaboration on a programming component by individuals (whether or not they are taking the class) is encouraged at the level of ideas. Feel free to ask each other questions, brainstorm on algorithms, or work together on a (virtual or real) whiteboard. Be careful, however, about copying the actual code for programming assignments or merely adapting others' code. This sort of collaboration at the level of artifacts is permitted if explicitly acknowledged, but this is usually self-defeating. Specifically, you will get zero points for any portion of an artifact that you did not transform from concept into substance by yourself. If you neglect to label, clearly and prominently, any code that isn't your own or that you adapted from someone else's code, that's academic dishonesty for the purpose of this course and will be treated accordingly.

  There are some circumstances under which you may want to collaborate with someone else on the programming component of an assignment. You and a friend, for example, might create independent parts of an assignment, in which case you would each get the points pertaining to your portion, and you'd have the satisfaction of seeing the whole thing work. Or you might get totally stuck and copy one subroutine from someone else, in which case you could still get the points for the rest of the assignment (and the satisfaction of seeing the whole thing work). But if you want all the points, you have to write everything yourself. These collaborations must be explicitly acknowledged in the `CHECKLIST.txt` file you submit.

The principle behind the above policies is simple: I want you to learn as much as possible. I don't care if you learn from me or from each other. The goal of artifacts (programming assignments) is simply to demonstrate what you have learned. So I'm happy to have you share ideas, but if you want your own points you have to internalize the ideas and them craft them into an artifact by yourself, without any direct assistance from anyone else, and without relying on any code taken from others (whether at this university or from the web).

A final note of caution: The course's assignments cover widely-used techniques, some of which have been used in prior installments of this course. Code for some of the assignments—and even answers to some written questions—may be just a mouse click (or a google search) away. You must resist the temptation to search for, download and/or adapt someone else's solutions and submit them as your own without proper attribution. Accidentally stumbling upon a solution is no excuse either: the minute you suspect a webpage describes even a partial solution to an assignment, either stop reading it or cite it in the checklist you submit. Simply put: if an existing solution is easy for you to find, it is just as easy for us to find it as well. In fact, you can be sure we already know about it!!