A Guide and tool for perspective warping

By Anthony Caccese

Introduction:

Warping the perspective of a skewed image to create a flat image is an interesting computer vision problem. First, to perform such a task, you first need to understand that images are just large matrices. Each pixel on your monitor corresponds to a value in a matrix. Therefore, an image that has a resolution of 1920x1080 consists of at least three matrices of dimensions 1920x1080 (the reason for three instead of one is for the red, green, and blue channels). This fact allows us to perform amazing things with images using simple matrix multiplication. For example, to apply a simple transformation to a pixel, you can use the following:

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

Credit: Yay-Photogrammetry by Dr. Yoo

So similarly, you can create a matrix that you multiply to rotate, scale, or change the perspective of the image. For a project such as changing the perspective of a square of grass to become straight on view, a homography can be used.

A homography can project a planar surface in one image to a plane in another. This is done by taking four points in the image one image and four points in another and then projecting one onto another (Figure 2). The four points that are selected in each image cannot be co-linear. A single point in these images can be represented as x' and y' for the image you want to map to and x and y for the image that is being mapped. In order to do a projection from one plane to another, the four-point algorithm (figure 3) is used to calculate the homography matrix (figure 4), which is used to do the projection between images.

For this project, by selecting the four corners of the squares in the image, you can map them to another virtual camera that is square to give the effect of straightening out the square.

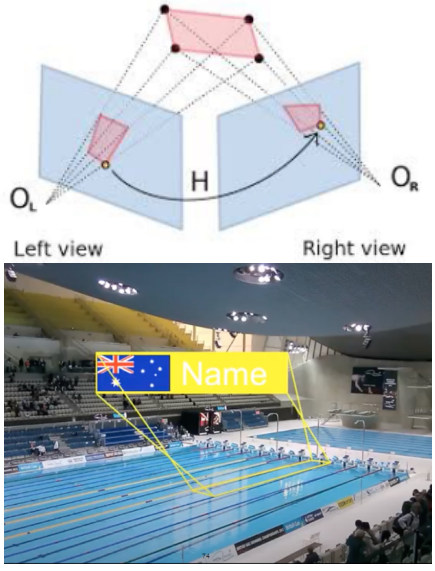


Figure 2: Use case of a homography

The Four-Point Algorithm

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x'_2x_2 & -x'_2y_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -y'_2x_2 & -y'_2y_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x'_3x_3 & -x'_3y_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -y'_3x_3 & -y'_3y_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x'_4x_4 & -x'_4y_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -y'_4x_4 & -y'_4y_4 \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{bmatrix}$$

Figure 3: Four point Algorithm

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \sim \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Figure 4: Homography Matrix

Guide:

Here are the instructions on how to set up and use the program to create the homography.

Setup:

Requirements:

To begin you need to have python installed on your computer. You can do so by downloading the latest version and following their installation instructions - I used version 3.8.

https://www.python.org/downloads/

For ease of use, you may want to download Visual Studio Code so that you can easily run the program.

https://code.visualstudio.com/download

Once python is installed you check to see if things are working by opening your command prompt on windows or terminal if on mac, then type the following.

```
pip install opencv-contrib-python
pip install pillow
```

*Note some versions use different syntax for installing packages if this doesn't work try using these instead:

```
python -m pip install opencv-contrib-python
python -m pip install pillow
```

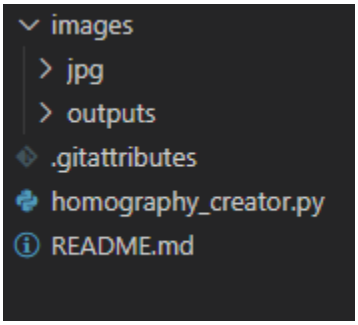
Environment Setup:

If you have my zip file called caccese_cv_homography_creator you can unzip it in any location desired.

Can download the files here:
https://drive.google.com/file/d/1StOT6j5GKZ9bSa0GxP4wMzUaXnbi7ahr/view?usp=share_link

In this folder, you will have the following structure:

homography_creator.py is the python program
images folder containing all images, this will be where your input images will be stored and output images.
jpg folder is where any images you want to use as inputs will be stored.
outputs folder is created whenever a homography is computed.



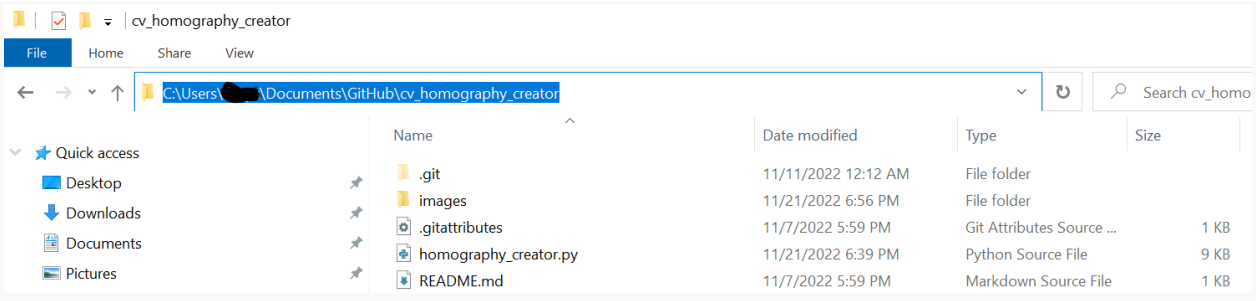
Running the program:

Before running the program, I would suggest adding images to the jpg folder. This program will only work for jpg images therefore, you will need to convert from other file formats. For this dataset, you can use https://heictojpg.com/ to convert from HEIC to jpg.

Two options for running the program
Option 1: Use the command prompt/terminal to run the program.
Option 2: Use vs code to run the program.

Option 1:

To use the command prompt/terminal to run the program, you need to find the file path of the program. This can be done by opening the file explorer and finding where the program is located. Then you can copy the file path as shown below.



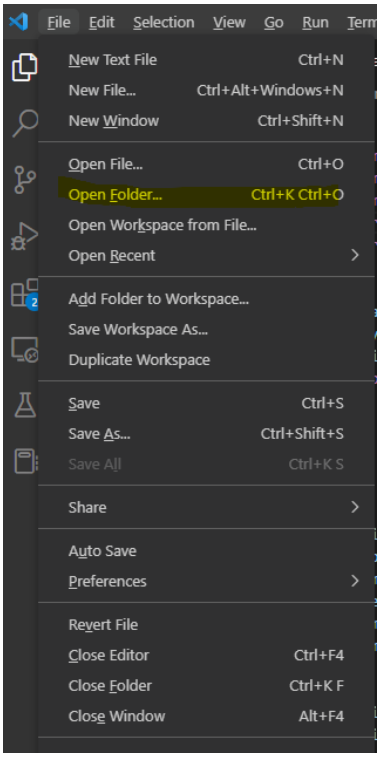
Then open the command prompt/terminal and type the following commands

```
cd (paste the file path here)
py homography_creator.py
```

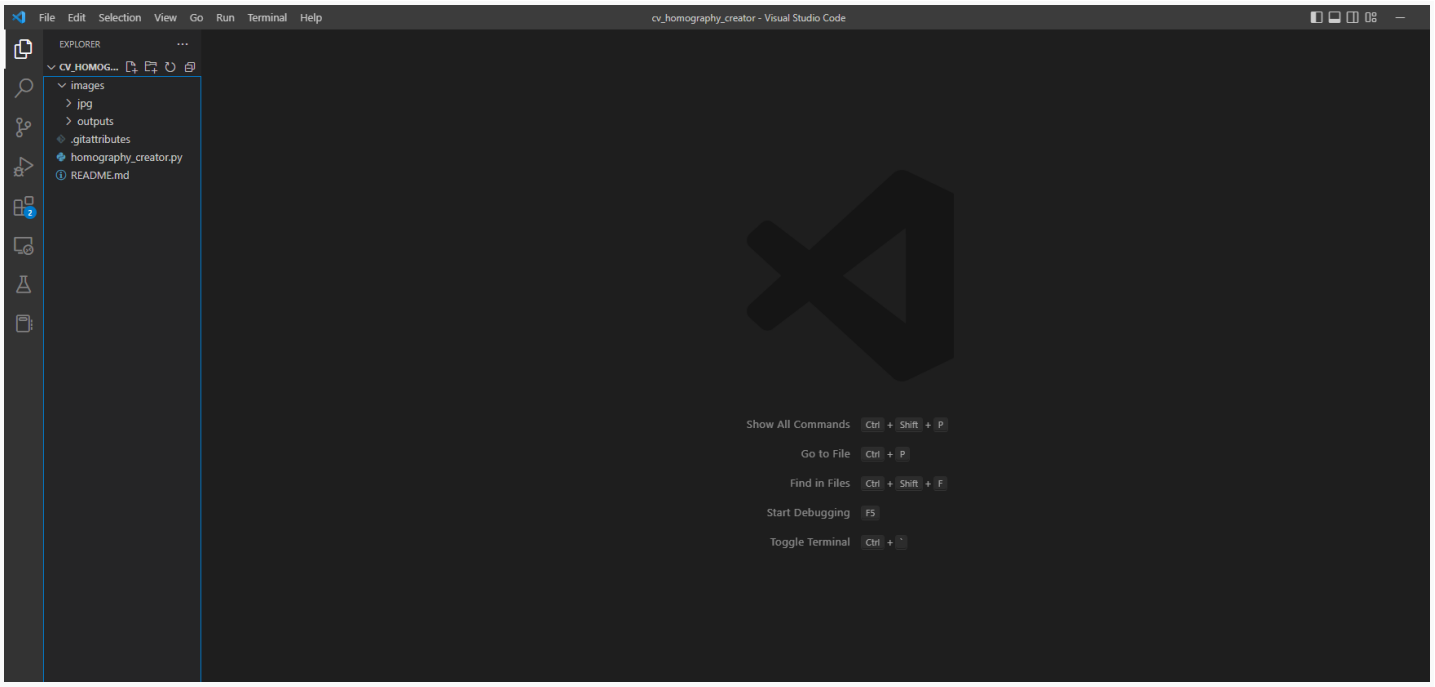
This should run the python program.

Option 2:

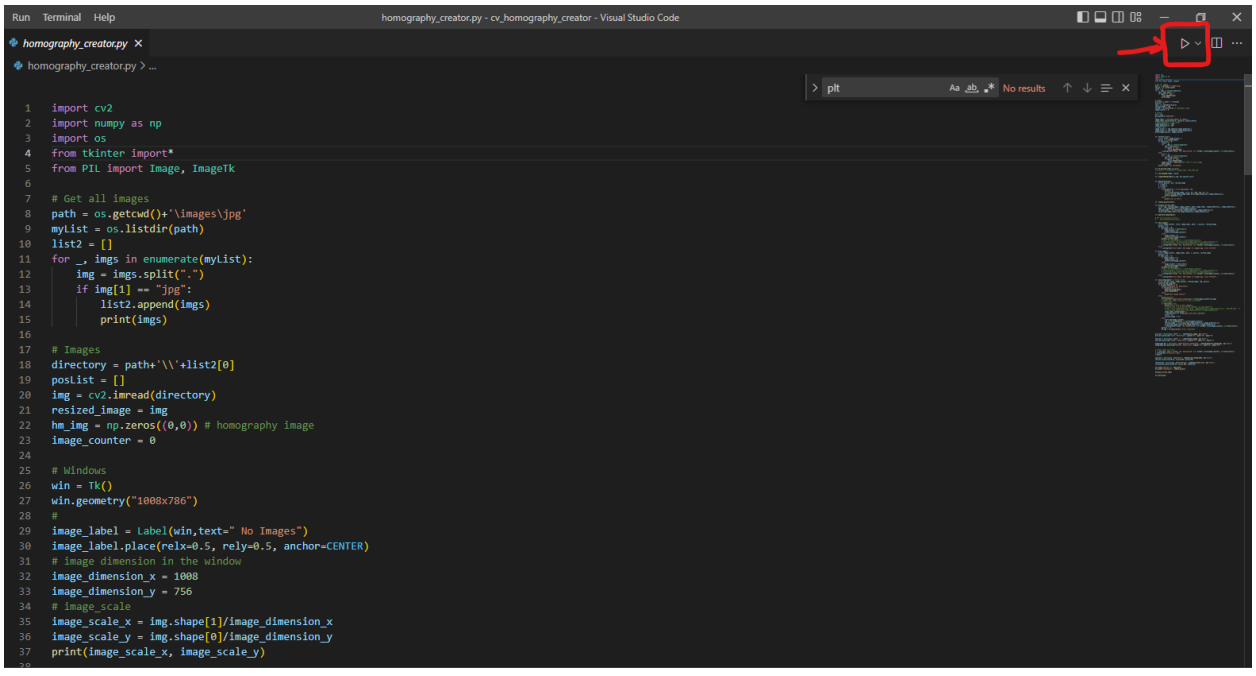
Open Visual Studio Code, and in the top left corner, click file open folder.



Then Navigate to the folder where the program is located. Once in the folder, you should see a similar file structure on the left.



Select the file called homography_creator.py. Once you click on it, you can then click on the run-in terminal button in the top right-hand corner.

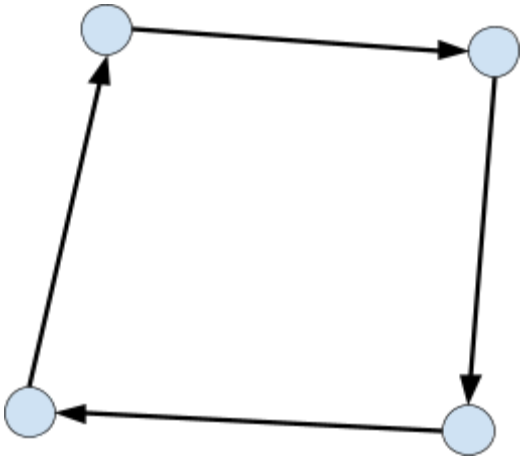


How to use the Program:

When the program starts, you will be presented with the current images in the jpg folder. The top bar consists of a save button to save the homography, the name of the current image displayed, the number of images in the jpg folder that have not been processed, and a refresh button to reload the images in the jpg folder.

On the side, there are two buttons a left and right arrow allowing you to scroll through the current unprocessed images.

To generate the flat image, use the left mouse button to add points. Select the corners of the square in the image. In order to properly generate a homography you need to click the points in a clockwise or counterclockwise sequence.



Once four points are selected, you can click “generate homography”. This will show you a preview of the homography in a new window. You can save the image by clicking the save button in the top left corner. This will save the output in the outputs folder. If you need to load more images, you can add more to the jpg folder and then click the refresh button.

https://youtu.be/xn8In6Oi9Uk - video shows how to generate the perspective fixed images.

Controls:

Left-Mouse-Click - adds points to the image

Backspace - delete all current points plotted on the image.

Appendix:

Source Code:

```
import cv2

import numpy as np

import os

from tkinter import*

from PIL import Image, ImageTk

# Get all images

path = os.getcwd()+'\images\jpg'

myList = os.listdir(path)

list2 = []

for _, imgs in enumerate(myList):

    img = imgs.split(".")

    if img[1] == "jpg":

        list2.append(imgs)

        print(imgs)



# Images

directory = path+'\\'+list2[0]

posList = []

img = cv2.imread(directory)

resized_image = img

hm_img = np.zeros((0,0)) # homography image

image_counter = 0



# Windows

win = Tk()

win.geometry("1008x1058")

#
```

```
image_label = Label(win,text=" No Images")

image_label.place(relx=0.5, rely=0.5, anchor=CENTER)

# image dimension in the window

image_dimension_x = 1008

image_dimension_y = 1008

# image_scale

image_scale_x = img.shape[1]/image_dimension_x

image_scale_y = img.shape[0]/image_dimension_y

print(image_scale_x, image_scale_y)




def refresh_list():

    global list2, image_counter, l

    myList = os.listdir(path)

    if len(list2) > 0:

        list2 = []

        for _, imgs in enumerate(myList):

            img = imgs.split(".")

            if img[1] == "jpg":

                list2.append(imgs)

        l.config(text="Image: {0}; Unprocessed: {1}".format( list2[image_counter], str(len(list2))))

    else:

        list2 = []

        for _, imgs in enumerate(myList):

            img = imgs.split(".")

            if img[1] == "jpg":

                list2.append(imgs)

        image_counter = len(list2)-1 # reset to first image

        next_image()

    print("Image list refreshed")
```

```
def to_pil(img,label,x,y,w,h):

    global canvas, image_container

    img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

    img = cv2.resize(img,(w,h))

    # img = cv2.flip(img,1)

    image = Image.fromarray(img)

    pic = ImageTk.PhotoImage(image)

    print(pic)

    label.configure(image=pic)

    label.image = pic

    label.place(x=x, y=y)



def resizeImage(image, scale):

    scale_percent = scale # percent of original size

    width = int(image.shape[1] * scale_percent / 100)

    height = int(image.shape[0] * scale_percent / 100)

    dim = (width, height)



    # resize image

    return cv2.resize(image, dim, interpolation = cv2.INTER_AREA)



def computeHomography(src_img, dst_img,pts_list):

    buf = 0

    pts_src = np.array(pts_list)

    pts_dst = np.array([[buf, buf],[dst_img.shape[1]-buf, buf],[dst_img.shape[1]-buf,
dst_img.shape[0]-buf],[buf, dst_img.shape[0]-buf]])

    print(pts_src)

    # Calculate Homography

    h, status = cv2.findHomography(pts_src, pts_dst)
```

```
    # Warp source image to destination based on homography
    return cv2.warpPerspective(src_img, h, (dst_img.shape[1],dst_img.shape[0]))



def add_point(event):
    global posList, win, resized_image
    x = event.x
    y = event.y
    if y >15:
        if(len(posList) < 4 and len(list2) > 0):
            print(x, y)
            cv2.circle(resized_image, (x,y), 10, (255, 255, 0), -1)
            to_pil(resized_image,image_label,0,30,image_dimension_x,image_dimension_y)
            posList.append([x, y])
        else:
            print("List is full")


def remove_points(event):
    global posList
    posList = []
    display_current_img()
    print("Points cleared")


def display_current_img():
    global img, resized_image, image_counter, path, image_label, image_dimension_x, image_dimension_y
    img = cv2.imread(path+'\\'+list2[image_counter])
    resized_image = cv2.resize(img,(image_dimension_x,image_dimension_y))
    to_pil(img,image_label,0,30,image_dimension_x,image_dimension_y)


def generate_homography():
```

```
    global hm_img

    if len(posList) == 4:

        print("Generating Homography")

        img = cv2.imread(path+'\\'+list2[image_counter])

        canvas_size = np.zeros((img.shape[0], img.shape[1]))

        x = int(image_dimension_x*image_scale_x)

        y = int(image_dimension_x*image_scale_x)

        print(x, y)

        scaled_list = []

        for point in posList: # apply scale so no resolution is lost

            scaled_list.append([point[0]*image_scale_x,point[1]*image_scale_y])

        hm_image = computeHomography(img,canvas_size,scaled_list)



        hm_img = cv2.resize(hm_image,(x,y)) # save homography

        resized_hm_img = cv2.resize(hm_img,(image_dimension_x,image_dimension_y)) # resize to window

        print(resized_hm_img.shape[0],resized_hm_img.shape[1])

        cv2.imshow('Homography Computed', resized_hm_img)

    else:

        print("Not enought points")



def next_image():

    global image_counter, list2, image_label, path, l, posList, resized_image

    posList = []

    if len(list2) > 0:

        if image_counter < len(list2)-1:

            image_counter +=1

            print(list2[image_counter])

        else:

            image_counter = 0

            print(list2[image_counter])
```

```
        display_current_img()
        l.config(text="Image: {0}; Unprocessed: {1}".format( list2[image_counter], str(len(list2))))
    else:
        l.config(text="List empty: Add images to /images/jpg, click refresh")


def prev_image():
    global image_counter, image_label, path, l, posList, resized_image
    posList = []
    if len(list2) > 0:
        if image_counter > 0:
            image_counter -=1
            print(list2[image_counter])
        else:
            image_counter = len(list2)-1
            print(list2[image_counter])
        display_current_img()
        l.config(text="Image: {0}; Unprocessed: {1}".format( list2[image_counter], str(len(list2))))
    else:
        l.config(text="List empty: Add images to /images/jpg, click refresh")


def save_homography(): # doesn't work
    global hm_img, list2, image_counter, resized_image, img, posList
    print(hm_img.shape[0])
    if hm_img.shape[0] == 0:
        print("Homography not generated")
        if len(posList) == 4:
            generate_homography()
            save_homography()
        else:
            print("Not enough points")
```

```
    else:

        print("Saving")

        cv2.imwrite("images\outputs\Homography_"+list2[image_counter],hm_img)

        # remove this image from list or mark as finished

        # next_image()

        if len(list2) <= 1:

            print("end of list no more images")

            image_label.config(image='')

            l.config(text="All homographys have been computed")

            list2 = []

            resized_image = None

        else:

            list2.pop(image_counter)

            img = cv2.imread(path+'\\'+list2[image_counter])

            resized_image = cv2.resize(img,(image_dimension_x,image_dimension_y))

            to_pil(img,image_label,0,30,image_dimension_x,image_dimension_y)

            l.config(text="Image: {0}; Unprocessed: {1}".format( list2[image_counter],
str(len(list2))))

        posList = []

        hm_img = np.zeros((0,0)) #clear homography

prev_btn = Button(win, text=" < ", command=prev_image, bg='white')

prev_btn.pack(side="left", fill="none", expand="no", padx="10", pady="0")

next_btn = Button(win, text=" > ", command=next_image, bg='white')

next_btn.pack(side="right", fill="none", expand="no", padx="10", pady="0")



homography_btn = Button(win, text="Generate Homography", command=generate_homography, bg='white')

homography_btn.pack(side="bottom", fill="none", expand="no", padx="10", pady="10")

# Create label for title

l = Label(win, text = "Image: {0}; Unprocessed: {1}".format( list2[image_counter], str(len(list2))))

l.config(font =("Courier", 14))
```

```
l.pack()

save_btn = Button(win, text="Save", command=save_homography, bg='white')

save_btn.place(relx=0.01, rely=0.005, anchor=NW)



refresh_btn = Button(win, text="Refresh", command=refresh_list, bg='white')

refresh_btn.place(relx=0.99, rely=0.005, anchor=NE)



win.bind("<Button 1>", add_point)

win.bind("<BackSpace>", remove_points)



display_current_img()



win.mainloop()
```