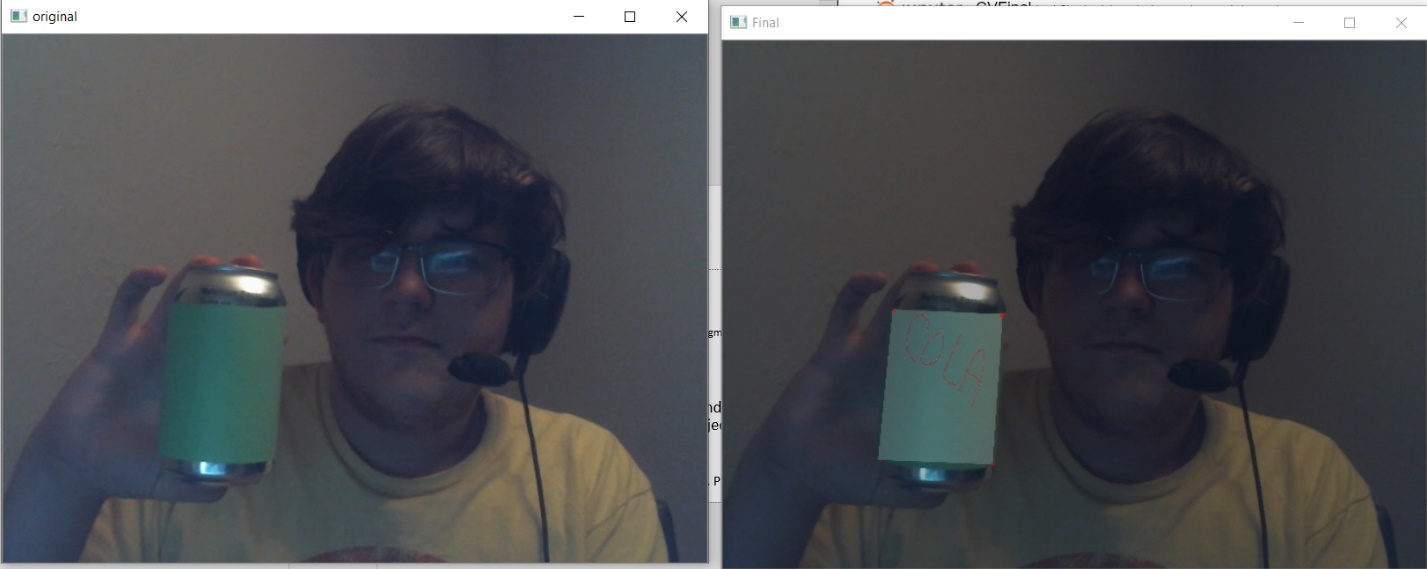
**Dynamic image overlays**

**Tyler Julian**

**Problem:**

Sometimes you do not have all your props ready to go when you are filming, so you must use a placeholder object in its place. Instead of going through each frame of your video and painstakingly replacing the placeholder object with the intended item, you can use computer vision to recognize the placeholder item and replace it automatically.

A real example could be a soda sponsorship. You know that a soda company is going to sponsor you, but you do not know which one while filming. You can have a placeholder soda can during filming and edit the sponsor into your film later. That way you can film early without any worries. Here is an example of what it would look like.

****

**Implementation Steps:**

These are the following steps that I do to replace the placeholder item.

1) The current frame is loaded into the program

2) The current frame is converted to an HSV image.

3) A mask is created that only contains green colors. Anything that is not green is black.

4) The mask is converted to a binary black and white image. Green is white, everything else is black.

5) Contours are drawn around the thresholded image. (BLUE)

6) An approximate shape is drawn that represents the contours (RED)

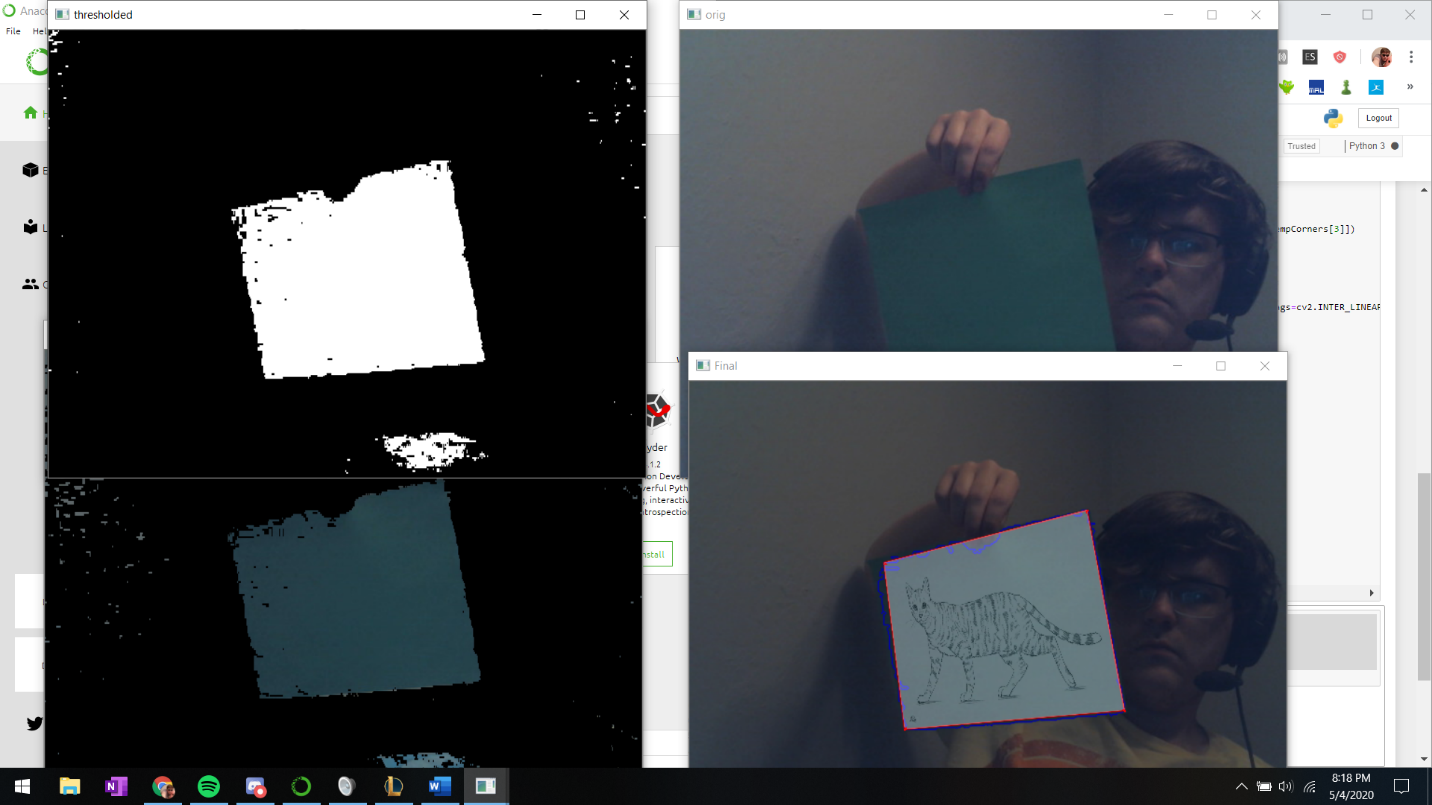
7) Corners are found using a harris-Corner detector.

8) The input image is warped to cover the placeholder image.

9) The two images are combined into the final images.

After these steps, the image can be further blended together to create a better looking picture/video.

The following image shows a frame at various steps through the process (steps 1,3,4,9).

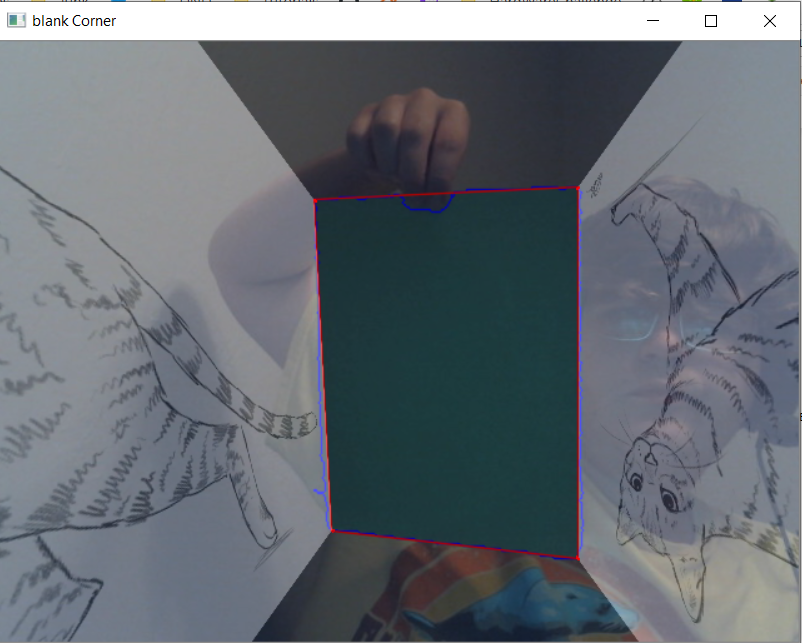


**Results:**

The software solution works decently well, and the input image is overlaid onto each frame when the program detects the placeholder image. When you input video from a webcam, you can see the output images update in real time without any loss of framerate which makes it a great tool to use when you are livestreaming from your webcam.

One of the portions that I am happy with is that my software accounts for small imperfections when finding the placeholder image. As you can see in the image above, the software ignores the fingers that cover the sheet of paper and the imperfect edges. The approximate contours create a shape that is similar to the original contours, but without noise, so the lines are smooth and straight. This allows me to easily identify the total number of corners on the paper down to 4.

One of the negative portions of the software is that It doesn’t support the placeholder being orientated in a certain way. The incorrect orientation causes the overlaid image to glitch out occasionally. See below for an example. One of the ways to correct this glitch is to sort the corners before warping the input image.



Another improvement that could be made is that the overlaid image will cover the fingers that is holding the placeholder image. Most of the time it is not noticeable unless something is blocking the placeholder item.

Overall, I believe that my implementation was an overall success, but there are several obvious fixes that can be fixed given more time. My commented software solution is listed below as well.

**Software solution:**

An attached webcam is required to run the python script. A square sheet of green paper can be used as the placeholder image. Green is a great choice of color for the placeholder image, because it is easier to filter out green hues.

**Code:** <https://github.com/TylerJulian/ComputerVisionFinal>

**import cv2**

**import numpy as np**

**cap = cv2.VideoCapture(0) # '0' is the webcam's ID. usually it is 0 or 1. 'cap' is the video object.**

**cap.set(15, 4) # '15' references video's brightness. '-4' sets the brightness.**

**while True:**

**ret, im = cap.read()**

**if im is None:**

**break**

**#cv2.imshow('orig step 0',im)**

**#convert to hsv then split into 3 values**

**hsv = cv2.cvtColor(im,cv2.COLOR\_BGR2HSV)**

**#FINDS THE GREEN VALUES**

**mask = cv2.inRange(hsv, (40,25,25),(100,255,255))**

**#filters out non green values**

**imask = mask>0**

**green = np.zeros\_like(im, np.uint8)**

**green[imask] = im[imask]**

**#thresholds the frame and convert all of the green to white and everything else to black**

**hsv = cv2.cvtColor(green, cv2.COLOR\_BGR2HSV)**

**h,s,v = cv2.split(hsv)**

**th, threshed = cv2.threshold(s,50,255,cv2.THRESH\_BINARY)**

**#finds the contours on the threshold image**

**cnts = cv2.findContours(threshed, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)[-2]**

**canvas = im.copy()**

**cnts = sorted(cnts,key = cv2.contourArea)**

**#if a green section was detected then this segments executes**

**if cnts:**

**#this finds the largest contour which should be the placeholder image.**

**#this can be modified to find the second/third/fourth largest image.**

**cnt = cnts[-1]**

**arclen = cv2.arcLength(cnt, True)**

**approx = cv2.approxPolyDP(cnt, 0.02\* arclen, True)**

**#cv2.drawContours(canvas, [cnt], -1, (255,0,0), 1, cv2.LINE\_AA)**

**#cv2.drawContours(canvas, [approx], -1, (0, 0, 255), 1, cv2.LINE\_AA)**

**#cv2.imshow('square detection final',canvas)**

**#creates a blank black image**

**blank = np.zeros\_like(im, np.uint8)**

**#draws the approximate contour onto the blank image then converts it to grayscale. This converts the red contour line to white**

**cv2.drawContours(blank, [approx], -1, (0, 0, 255), 1, cv2.LINE\_AA)**

**blank = cv2.cvtColor(blank,cv2.COLOR\_BGR2GRAY)**

**blank = np.float32(blank)**

**#detects the corners and stores them into a list of corners**

**dst = cv2.cornerHarris(blank,2,3,0.04)**

**ret, dst = cv2.threshold(dst,0.1\*dst.max(),255,0)**

**dst = np.uint8(dst)**

**ret, labels, stats, centroids = cv2.connectedComponentsWithStats(dst)**

**criteria = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 100, 0.001)**

**corners = cv2.cornerSubPix(blank,np.float32(centroids),(5,5),(-1,-1),criteria)**

**#this creates a list of corners again**

**tempCorners = []**

**for i in range(1, len(corners)):**

**tempCorners.append(corners[i])**

**#if the detected green segment is rectangular/square then we proceed to this segment**

**if len(tempCorners) == 4:**

**#print('Square:')**

**#overlay image**

**input = cv2.imread('cola.jpg')**

**#cv2.imshow('input',input)**

**#input image size, this can be determined based off the actual image size dynamically.**

**input\_pts = np.float32([[1000,0],[0,0],[0,562],[1000,562]])**

**#uses the list of corners again but in a format that works with warp perspective**

**output\_pts = np.float32([tempCorners[1],tempCorners[0],tempCorners[2],tempCorners[3]])**

**# sort to organize the output points**

**#warps the original image to that of the placeholder item**

**M = cv2.getPerspectiveTransform(input\_pts,output\_pts)**

**out = cv2.warpPerspective(input,M,(canvas.shape[1], canvas.shape[0]),flags=cv2.INTER\_LINEAR)**

**#overlays the input image.**

**canvas = cv2.addWeighted(canvas,0.7,out,0.3,0)**

**canvas[dst>0.1\*dst.max()]=[0,0,255]**

**#original image.**

**cv2.imshow('original', im)**

**#shows the final image with contours and the overlaid image. The contours can easily be removed.**

**cv2.imshow('Final', canvas)**

**k = cv2.waitKey(30) & 0xff # press [Esc] to exit.**

**if k == 27:**

**break**

**cv2.destroyAllWindows()**