Term Project #1 - Digital Matting

[Github Link]

Team Members

Emma Rose Rorick 20191984 Polina Kotova 20191972 Anastasia Gordeeva 20191186

Core Algorithm

Step 1 - Preparing images

Goal - Import, load, and resize/match sizes of original video and the background photo

Import/loading image and video example code:

```
video = cv2.VideoCapture("green.mp4")
bgphoto = cv2.imread("bg.jpeg")
```

Final version:

```
video = cv2.VideoCapture('green.mp4')
image = cv2.imread('sogang.jpg')

width = int(video.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(video.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = video.get(cv2.CAP_PROP_FPS)  # this may be float such as 29.97
nframes = int(video.get(cv2.CAP_PROP_FRAME_COUNT))
image = cv2.resize(image, (width,height))

canvas = np.zeros((height, nframes, 3), dtype='uint8')

recorder = cv2.VideoWriter('chromakey_res.mp4',
```

Step 2 - Change color to HSV & define thresholds

Goal - More easily select values based on hue, then define the range of colors that would be considered 'background' in the future steps
How to do this - Load the upper and lower HSV values for the defined point's color

- In our case, from dark green → light green

```
#change color to HSV for better masking
#in HSV its easier to select green values  based on hue
video_hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

green_L = np.array([50,150,0]) #from dark green
green_H = np.array([110, 255, 255]) #to light green
```

Step 3 - Checking all pixels & create mask blur

Masking: if pixel is close to the sample then white else: black

- *Goal* - Creating a mask based on the color threshold separating background and foreground. In mask black is the active are and white is an inactive area, same as masking in photoshop

```
#create a mask based on low and high values that we set
mask = cv2.inRange(video_hsv, green_L, green_H)
#blur the mask to make edges look better
mask_blurred = cv2.GaussianBlur(mask, (3,3), 0)
```

Step 4 - Replacing defined space with another image/video black area and implement mask blur

```
masked_video = np.copy(video_bgr)
masked_video[mask_blurred != 0] = [0, 0, 0]
masked_image = np.copy(image)
masked_image[mask_blurred == 0] = [0, 0, 0]
#compile cut versions together
final = masked_video + masked_image
cv2.imshow('final', final)
```

Implementation

We implemented our first simple code with two images.

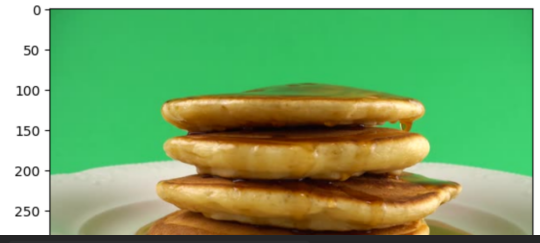
```
import cv2
import matplotlib.pyplot as plt
import numpy as np
```
✓ 0.1s

```
img = cv2.imread('pancakes.jpg')

img_copy = np.copy(img)
img_copy = cv2.cvtColor(img_copy, cv2.COLOR_BGR2RGB) #change to RGB

plt.imshow(img_copy)
print('dimensions:', img.shape)
```
✓ 0.4s

```
dimensions: (332, 590, 3)
```

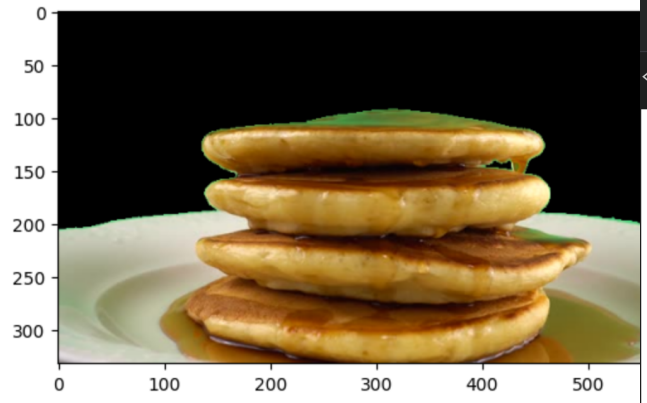



```
#color treshold
green_low = np.array([0, 160, 0])
green_up = np.array([100, 255, 120])
#masking
mask = cv2.inRange(img_copy, green_low, green_up)
plt.imshow(mask,'gray')
```
✓ 0.6s

```
<matplotlib.image.AxesImage at 0x1186d175000>
```


```
masked_img = np.copy(img_copy)
masked_img[mask != 0] = [0, 0, 0]
plt.imshow(masked_img)
```
✓ 0.5s

```
<matplotlib.image.AxesImage at 0x1186bd4e6b0>
```

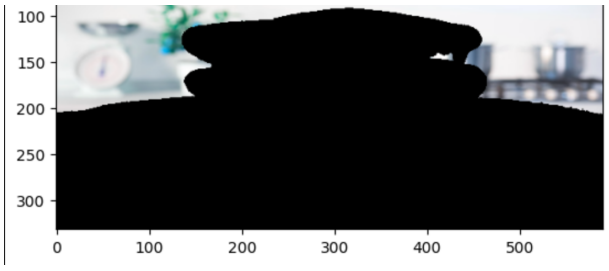



```
bckg = cv2.imread('kitchen.jpg')
bckg = cv2.cvtColor(bckg_img, cv2.COLOR_BGR2RGB)
#crop and mask backgroind

crop_bckg = cv2.resize(bckg, (590, 332))
crop_bckg[mask == 0] = [0, 0, 0]

plt.imshow(crop_bckg)
```
✓ 0.5s

```
<matplotlib.image.AxesImage at 0x1186d0d18d0>
```

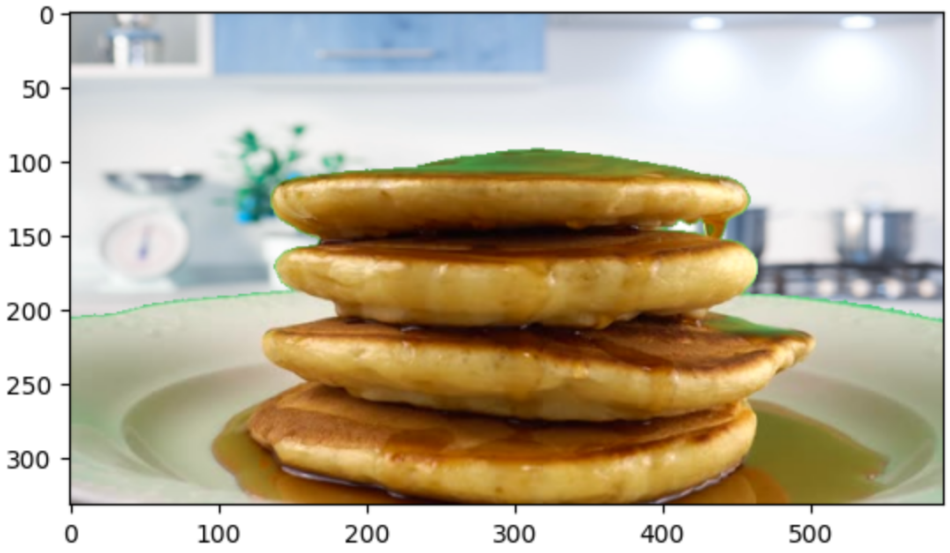


Final Image - **version 1**

```
final_img = crop_bckg + masked_img
plt.imshow(final_img)
```

✓ 0.6s

```
<matplotlib.image.AxesImage at 0x1186d24e260>
```



Result was okay, but requires a few alterations:

1. Add color sampling instead of manually inserted green values
2. Work on the edges
3. Consider green reflection when filming like we can see on top of the pancakes
4. Try inserting video rather than image

Our challenges, solutions

Color Models - simple changes
It was hard to understand which color model was used and at first colors of the video didn't process right

- ➔ We changed the model and it fixed the problem.
- ➔ We first attempted using RGB values, but manipulating hues is easier in HSV so we switched to that model.

Green outlines
Even with a good color range, an ugly green contour/outline remained so we added mask blur to soften the edges and get rid of most of the extra green edges that appeared to make the video more seamless.

Mask Blur Results - use of Gaussian Blur in **version 2**

No mask blur on
Note: Green borders

With mask blur on
Note: absence of green border

Research References

[Green Screen Lighting: How to Ensure Your Backgrounds Pop]

- Green screen lighting photo

[Blue or Green Screen Effect with OpenCV [Chroma keying] | by Teja Kummarikuntla | Fnplus Club | Medium]

[Chroma Keying with OpenCV/C++ | Alexey Smirnov (smirnov-am.github.io)]

- Vlahos formula

[Replace Green Screen using OpenCV- Python]

- Code examples