CS7.505 - Assignment 0
OpenCV and Chroma Keying

Avneesh Mishra
avneesh.mishra@research.iiit.ac.in *

January 14, 2022

Contents

*Roll No: 2021701032

1 Installing and Testing OpenCV

Main docs: `https://docs.opencv.org/4.x/`
Python docs: `https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html`

1.1 OpenCV Setup

The procedure to setup OpenCV using Anaconda on windows is listed below

1. Download and install Anaconda from here.
2. Setup the anaconda environment for the shell using `conda init`
3. Setup the environment

   Create the environment

   ```
   conda create -yn "cv-cs7-505"
   conda activate cv-cs7-505
   ```

   Install Python 3.9 into it

   ```
   conda install python=3.9
   ```

   Install the essential packages (before OpenCV)

   ```
   conda install numpy jupyterlab
   ```

4. Install OpenCV using `pip`

   Install using pip

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

1.2 Testing Installation of OpenCV

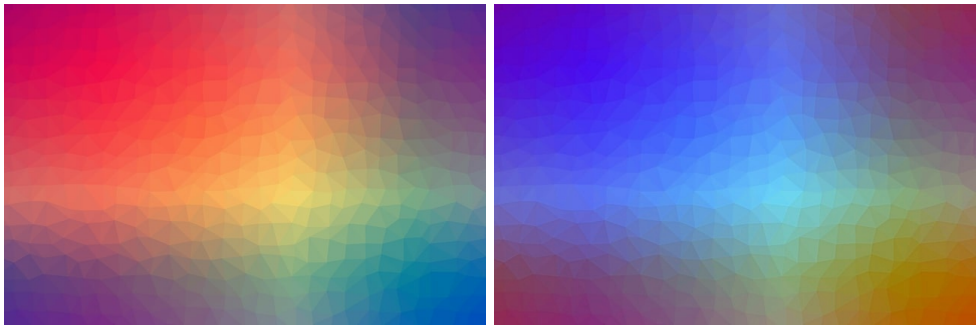
Checking Version

Check version using the script below

```python
# Check OpenCV version
# %% Import everything
import cv2 as cv
import numpy as np

# %% Main entrypoint
if __name__ == "__main__":
    print(f"OpenCV version: {cv.__version__}")
    print(f"Numpy version: {np.__version__}")

# %%
```

The output of the above script is

```
OpenCV version: 4.5.5
Numpy version: 1.21.2
```



(a) Input (b) Output

Figure 1: Test images
The red and blue channels are flipped (their layer intensities are swapped). Check code listing 1

Reading and changing color channels

Consider the images in figure 1. The code to get this output is given below

```
# Change the color channels of an image
"""
    Flips the Reg and Blue channel of an image
"""

# %% Import everything
import cv2 as cv
import numpy as np

# %% Functions
# Show image in a window
def show_img(img, win_name = "Image"):
    cv.namedWindow(win_name, cv.WINDOW_GUI_EXPANDED)
    while True:
        cv.imshow(win_name, img)
        if cv.waitKey(1) == ord('q'):
            break
    cv.destroyWindow(win_name)

# %% Main module
if __name__ == "__main__":
    # Read image
    img_in = cv.imread("./images/test.jpg")
    # Show image
    show_img(img_in, "Input")
    img_out = img_in[:,:,::-1]
    show_img(img_out, "Output")
    # Save output
    cv.imwrite("./images/output.jpg", img_out)

# %%
```

Listing 1: Transform images

2 Chroma Keying with OpenCV

2.1 Video and Images

2.1.1 Video to images

Problem Given a video, convert it into constituent images

Experiments & Learning Experiments performed are listed below

1. Giving user input to the program: Tried using the `input` built-in to read from console. Finally settled at using argparse. Learned how to parse parameters professionally using argparse.
2. Using VideoCapture and read functions to fetch images from a video file, and writing images to disk using imwrite. Some experiments were run where the output prefix directory did not exist, so added the code to create the directory first.

Solution Download video from here and store as `./videos/vtest.avi`. Then run the following

```
python .\vid_to_imgs.py -n 10
```

The code is shown in listing 2. Output is in figure 2.

```python
# Convert a video to images
"""
    Given a video, generate the images. Run as main.
"""

# %% Import everything
import cv2 as cv
import numpy as np
import argparse
import sys
import os

# %% Argument parser
parser = argparse.ArgumentParser(
    formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument('-i', '--vid-file', default="./videos/vtest.avi",
    help="Video file to read")
parser.add_argument('-n', '--num-imgs', default=0, type=int,
    help="The maximum number of images to output from video (0=all)")
parser.add_argument('-o', '--out-prefix', default="./out/img",
    type=str, help="Output prefix for images")

# %% Main entrypoint
if __name__ == "__main__":
    # Parse all (known) arguments
    args, unknown_args = parser.parse_known_args(sys.argv)
    # Check if output directory (if passed) exists
    out_dir = os.path.split(args.out_prefix)[0]
    if not os.path.isdir(out_dir):
        print(f"Folder {out_dir} is being created")
        os.makedirs(out_dir)
    # Read video from file
    cap = cv.VideoCapture(args.vid_file)
    try:
        # Read frames
        fnum = 0
        while cap.isOpened():
            ret, frame = cap.read()
            if not ret:
                print("Probably EOF reached!")
                break
            cv.imshow("Video Feed", frame)
            if cv.waitKey(100) == ord('q'):
                print(f"Break encountered")
                break
            if args.num_imgs == 0 or fnum < args.num_imgs:
                # Write to disk
```

(a) Image 1 (b) Image 5 (c) Image 10

Figure 2: Sequence of images
First, fifth, and tenth image in the 10-image sequence produced by listing 2.

```
                cv.imwrite(f"{args.out_prefix}{fnum+1}.jpg", frame)
                fnum += 1
            else:
                print(f"Reached {fnum} frames")
                break
        print(f"Wrote {fnum} frames under {args.out_prefix}*.jpg")
    finally:
        # Cleanup
        cap.release()
        cv.destroyAllWindows()

# %%
```

Listing 2: vid_to_imgs.py

The above script (listing 2) gives the following output (along with a GUI window to show the images of the video)

```
Folder ./out is being created
Reached 10 frames
Wrote 10 frames under ./out/img*.jpg
```

2.1.2 Images to Video

Problem Given a folder with images, create a video where the FPS (frame rate) can be adjusted

Experiments & Learning Experiments performed are listed below

1. Getting images into the program: It was decided that the user will place the images, labelled in a sorted order (numerically), in a dedicated folder. For this demo, the folder name is `./seq`
2. All user inputs are given through `argparse`
3. FPS will be handled by the `VideoWriter` in the saved video file. However, the preview FPS has to be manually adjusted (using `waitKey` delay)

Solution Store images in a numerical order from 1.jpg to N.jpg (where N is any number) in a dedicated folder, like `./seq`. Then run the following (for 5 FPS)

```
python .\imgs_to_vid.py -i "./seq" -f 5.0 -o "./out-5.avi"
```

The code is shown in listing 3. A snapshot of the generated video is shown in figure 3a. The following is an example for 10 FPS

```
python .\imgs_to_vid.py -i "./seq" -f 10.0 -o "./out-10.avi"
```

A snapshot of the video generated is shown in figure 3b. See figure 3 for more information.

```python
# Create video from images in a given folder
"""
    Given images (numerically sorted 1 through N) in a folder, read
    them and create a video (no audio, only video)
"""

# %% Import everthing
import cv2 as cv
import os
import glob
import argparse

# %% Argparse parser
parser = argparse.ArgumentParser(
    formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument('-i', '--imgs-folder', default="./seq",
    help="Path to folder in which images are stored (all jpg files)")
parser.add_argument('-o', '--out-file', default="./out.avi",
    help="Output file (uses AVI, XVID fourcc)")
parser.add_argument('-f', '--vid-fps', default=10.0, type=float,
    help="The desired FPS (as float) for the output file")

# %% Main entrypoint
if __name__ == "__main__":
    # Parse arguments
    args, unknown_args = parser.parse_known_args()

    # Variables
    imgs_path = f"{os.path.realpath(args.imgs_folder)}/*.jpg"
    fps = args.vid_fps
    out_file = args.out_file
    # File names (sorted numerically)
    img_fnames = sorted(glob.glob(imgs_path), key=len)
    shape_w_h = cv.imread(img_fnames[0]).shape[-2::-1]  # (W, H) of video
    # Video writer
    fourcc = cv.VideoWriter_fourcc(*"XVID")
    out_fhdlr = cv.VideoWriter(out_file, fourcc, fps, shape_w_h)

    # For ever image found, write it to the video file
    for i, img_file in enumerate(img_fnames):
        # Read file
        frame = cv.imread(img_file)
        if frame is None:
            print(f"Unable to read '{img_file}', skipping it!")
            continue
        # Write the image to video writer
        out_fhdlr.write(frame)
        # Preview
        cv.imshow("Video", frame)
        if cv.waitKey(int(1000/fps)) == ord('q'):
            print(f"Quit received after {i} frames")
            break
    # Cleanup
    out_fhdlr.release()
    cv.destroyAllWindows()

# %%
```

Listing 3: imgs_to_vid.py

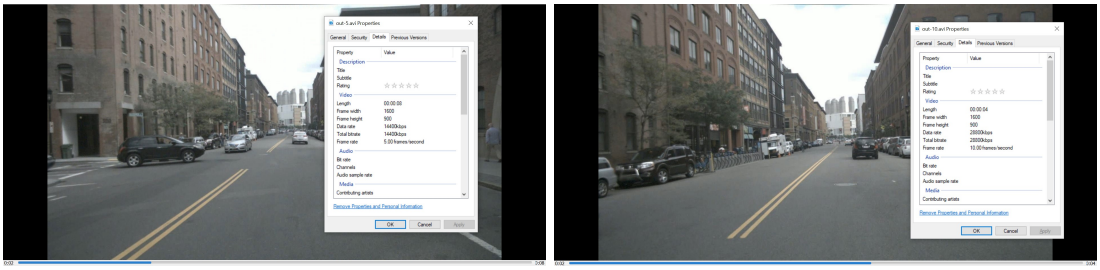
2.2 Capture Images through Webcam

Problem Use a webcam, capture images and save them to a folder

Experiments & Learning Experiments performed are listed below

1. Learned to capture images from a webcam using `VideoCapture`

Solution Run the following command



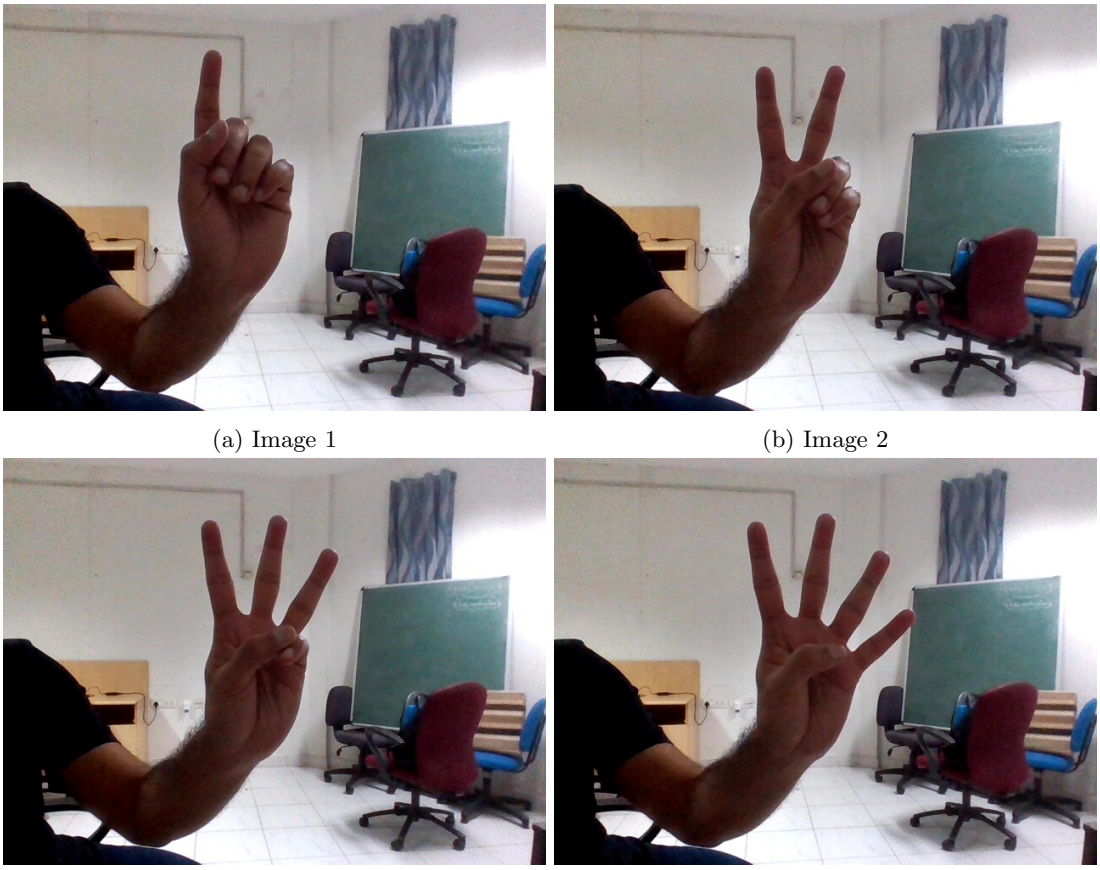
(a) 5 FPS (b) 10 FPS

Figure 3: Saved videos
Image a is a screenshot of the `5 FPS` playback (see the properties and the video frame). Image b is a screenshot of the `10 FPS` playback. Notice that the `10 FPS` version is faster, therefore a greater number of frames elapsed (for the 2 second point) compared to the `5 FPS` version.

```
python .\wbecam_capture.py -c 0 -o "./camimgs"
```

The code is included in listing 4. Output is shown in figure 4.

```python
# Capture images from a webcam
"""
    Given a webcam index and a destination folder (already existing),
    preview the feed and capture images using it. Press 'c' to capture
    an image and 'q' to quit.
"""

# %% Import everything
import cv2 as cv
import os
import argparse

# %% Argument parser
parser = argparse.ArgumentParser(
    formatter_class=argparse.ArgumentDefaultsHelpFormatter,
    description="Press 'c' to capture, 'q' to quit")
parser.add_argument('-c', '--cam', default=0, type=int,
    help="Camera index")
parser.add_argument('-o', '--out-folder', default="./camimgs",
    help="Output folder (should already exist)")

# %% Main entrypoint
if __name__ == "__main__":
    args, extra_args = parser.parse_known_args()
    webcam_id = args.cam
    out_folder = args.out_folder

    # Main camera work
    cam = cv.VideoCapture(webcam_id)
    imc = 0 # Image counter
    while True:
        ret, img = cam.read()
        if not ret:
            print("Camera did not give frames")
            break
        cv.imshow("Feed", img)
        key = cv.waitKey(1)
        if key == ord('q'):
            print("Quit command received")
            break
        elif key == ord('c'):
            imc += 1
            fname = f"{out_folder}/{imc}.jpg"
            print(f"Capturing image to '{fname}'")
            cv.imwrite(fname, img)
    cam.release()
    cv.destroyAllWindows()

# %%
```



(a) Image 1 (b) Image 2

(c) Image 3 (d) Image 4

Figure 4: Images captured through webcam
Four images captured through a webcam

Listing 4: webcam_capture.py

Four images were captured

```
Capturing image to './camimgs/1.jpg'
Capturing image to './camimgs/2.jpg'
Capturing image to './camimgs/3.jpg'
Capturing image to './camimgs/4.jpg'
Quit command received
```

2.3 Chroma Keying

Problem *Chroma Keying*: Given two videos (with one containing a green screen), create a fusion between the two, such that the subject of the green screen video is in the foreground and the green screen is replaced by the other video.

Experiments & Learning Experiments performed and things learned are listed below

1. Finding the video files was challenging. The desire was to have a foreground file where the green screen was consistent and easily extractable. Finally, a video was found on YouTube [1] and clipped. The video that would become the new background was also found on YouTube [2]. However, in the process of searching, datasets like VideoMatte240K were discovered (these weren't used because their size is too big for a single task).

[1] https://youtu.be/cqZuhJZAK-8
[2] https://youtu.be/MPUBSZYESgU

2. Comparisons were initially done *directly* on the green color channel (simple thresholding on green channel). This worked for the video, but is clearly not generalizable. There is no fixed metric that describes the *green-ness* of a color on the RGB color spectrum. Through some search and tutorials [3], a system that uses the **HSV** (Hue, Saturation, Value) colorspace was implemented. *Hue* covers the color in a single number, *saturation* covers the sharpness (presence) of the color, and *value* covers the darkness of the color.

3. Through experiments, a suitable hue threshold and margin was selected. However, the program gives provision to use custom values also. Additionally, there is also provision to capture specific images (along with masks) *while* the program is running and previewing the result.

Solution The following script (in listing 5) can be run

```
python .\chroma_keying.py -g 215 -t 10 -r 25
```

The code can be found in listing 5 and frames from the resulting video are shown in figure 5

```python
# Apply background based chroma keying
"""
    Given a foreground video with a greenscreen in background, a video
    that should be underlayed as new background, and a green value (
    along with hue threshold). Use '-h' option to know more.

    More information at [1]

    [1]: https://docs.opencv.org/4.x/df/d9d/tutorial_py_colorspaces.html
"""

# %% Import everything
import cv2 as cv
import numpy as np
import argparse
import os

# %% Argparse variables
parser = argparse.ArgumentParser(
    formatter_class=argparse.ArgumentDefaultsHelpFormatter,
    description="Use 'q' to quit and 'c' to capture")
parser.add_argument('-g', '--g-val', default=216,
    help="Green value (0-255)")
parser.add_argument('-t', '--hsv-thresh', default=5, type=float,
    help="Threshold percentage for hue (0-100)")
parser.add_argument('-w', '--img-fg', default="./videos/tiger-gs.mp4",
    help="Path to the foreground image")
parser.add_argument('-b', '--img-bg',
    default="./videos/waterfall.mp4",
    help="Path to the background image")
parser.add_argument('-o', '--out-folder', default="./chroma_out",
    help="Folder name (to store results). Must be exiting.")
parser.add_argument('-r', '--out-fps', default=30,
    help="FPS (writing)")

# %% Main entrypoint
if __name__ == "__main__":
    args, extra_args = parser.parse_known_args()
    g_val = args.g_val  # Green value
    h_thresh = args.hsv_thresh   # Threshold (percentage)
    # Foreground and background files
    fg_file = os.path.realpath(args.img_fg)
    bg_file = os.path.realpath(args.img_bg)
    # Output folder and FPS
    vid_out_file = os.path.realpath(args.out_folder)
    out_fps = int(args.out_fps)    # Output FPS
    # Convert green value to HSV
    h, s, v = cv.cvtColor(np.array([[[0, g_val, 0]]], dtype=np.uint8),
        cv.COLOR_BGR2HSV)[0,0]   # Green -> HSV
    print(f"HSV: {h}, {s}, {v}")

    # Read a frame from both streams
```

[3] https://docs.opencv.org/4.x/df/d9d/tutorial_py_colorspaces.html

```
    v1 = cv.VideoCapture(fg_file)
    ret, img_fg = v1.read()
    v1.release()
    # Main task
    i = 0
    v1 = cv.VideoCapture(fg_file)
    v2 = cv.VideoCapture(bg_file)
    fourcc = cv.VideoWriter_fourcc(*"XVID")
    vwrite = cv.VideoWriter(f"{vid_out_file}/chroma_res.avi", fourcc,
        out_fps, img_fg.shape[-2::-1])
    while True:
        ret, img_fg = v1.read()
        if not ret:
            print("Foreground video finished")
            break
        ret, img_bg = v2.read()
        if not ret:
            print("Background video finished")
            break
        # Mask through HSV
        img_fg_hsv = cv.cvtColor(img_fg, cv.COLOR_BGR2HSV)
        lower_green = np.array([h*(1-h_thresh/100), 180, 180])
        upper_green = np.array([h*(1+h_thresh/100), 255, 255])
        # img_fg_hsv channels in range (AND of each range check)
        mask = cv.inRange(img_fg_hsv, lower_green, upper_green)
        # Construct final image
        img_final = img_fg.copy()
        img_final[mask == 255] = img_bg[mask == 255]
        # Show result
        cv.imshow("Fusion", img_final)
        vwrite.write(img_final)
        key = cv.waitKey(int(1000/30))
        if key == ord('q'):
            print("Exit signal received")
            break
        elif key == ord('c'):
            i += 1  # New image
            cv.imwrite(f"{vid_out_file}/mask{i}.jpg", mask)
            cv.imwrite(f"{vid_out_file}/img_fg{i}.jpg", img_fg)
            cv.imwrite(f"{vid_out_file}/img_bg{i}.jpg", img_bg)
            cv.imwrite(f"{vid_out_file}/img_f{i}.jpg", img_final)
            print(f"Saved image '{i}' in folder '{vid_out_file}'")
    v1.release()
    v2.release()
    vwrite.release()
    cv.destroyAllWindows()
    print("Video writing completed")

# %%
```

Listing 5: chroma_keying.py

The above code (listing) generates the following output

```
HSV: 60, 255, 215
Saved image '1' in folder '****\chroma_out'
Saved image '2' in folder '****\chroma_out'
Saved image '3' in folder '****\chroma_out'
Saved image '4' in folder '****\chroma_out'
Foreground video finished
Video writing completed
```

The output files are `chroma_res.avi` which contains the main video, `img_bgN.jpg` which contains the background image (which will be separated), `img_fg1.jpg` which contains the foreground image, `maskN.jpg` which contains the mask image, and `img_fN.jpg` which contains the final image.

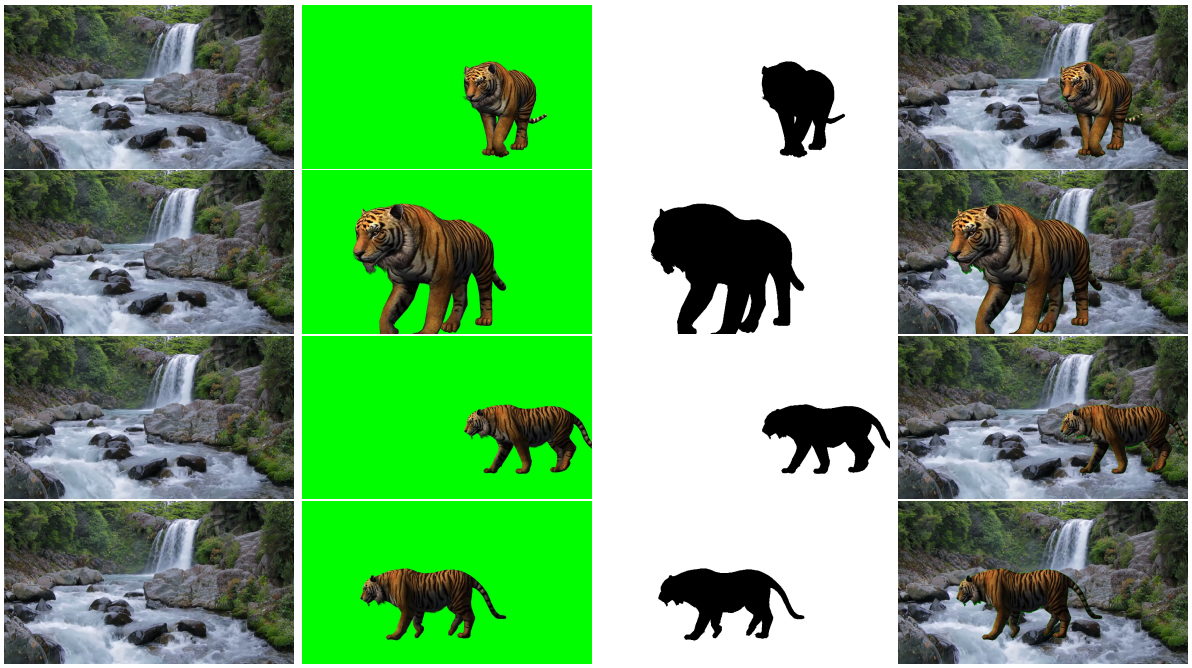


Figure 5: Chroma keying results

Snapshots and stages of the chroma keying process. The individual rows are the sequences (as time progresses). First column is the background that is to be substituted for the placeholder color. The second column is the foreground (with a placeholder color). The third column is the mask that detects the placeholder color (for substitution). The last column is the resulting image (which is put in a video file).

The video whose images are shown in the first column was taken from YouTube. The second column is also from YouTube.