CV-assignment-0 Report

> Name: Vishal Reddy Mandadi
> Roll Number: 2019101119

Table of contents

Section 1 (Videos ⟷ Images)

Problem

In this section, we are supposed to convert a video file into a set of frames that constitute it and save them separately as images in a different directory. We are also supposed to convert a set of images given as input into frames and turn them into a video file with a desirable frame rate (which is also passed as an argument).

Solution

Video to images

Here, I have used OpenCV to read the video file. The cv2.VideoCapture class is used to read the video file and then cv2.VideoCapture.read is used to read and output each frame as a NumPy array. This NumPy array is then converted and saved as an image using cv2.imwrite(). OpenCV functions came out extremely handy, but great attention was paid to writing a neat and extensible code (which later came to help where I needed similar functions for chroma keying or video capturing via webcam).

Functionality:

1. Convert to images and save
2. Convert to images and return

Command-line interface options:

1. Video path (should include video file name)

2. Image directory path

How to run:

```
# For getting help
python video_to_images.py -h
# General run command
python video_to_images.py -v <video_file_path> -i <image_dir_path>
```

Salient features: Good CLI with decent options to tweak and control everything in the program, extensible code

Code

```
import cv2
import sys
import argparse

class Video:
    def __init__(self, video_file_path: str) -> None:
        self.video_file_path = video_file_path
        self.video = cv2.VideoCapture(video_file_path)

    def convert_to_images_and_save(self, image_dir_path: str):
        if image_dir_path[-1] != '/':
            image_dir_path += '/'
        counter = 0
        while(self.video.isOpened()):
            ret, frame = self.video.read()
            if ret == False:
                print("Video stream ended! (or probably an error?)")
                break

            cv2.imwrite(image_dir_path+'{}_img.jpg'.format(counter), frame)
            counter += 1

        self.video.release()
        self.video = cv2.VideoCapture(video_file_path)

    def convert_to_images_and_return(self):
        counter = 0
        img_array = []
        while(self.video.isOpened()):
            ret, frame = self.video.read()
            if ret == False:
                print("Video stream ended! (or probably an error?)")
                break

            img_array.append(frame)
            counter += 1

        self.video.release()
        self.video = cv2.VideoCapture(video_file_path)
        return img_array

if __name__=='__main__':
    parser = argparse.ArgumentParser(description='Video to Image options')
    parser.add_argument('-v', '--video_path', help='Specify the absolute path to the video file that is to be converted to images', require
    parser.add_argument('-i', '--image_dir_path', help='Specify the absolute path to the directory where the images are to be stored', requ
    args = vars(parser.parse_args())
    print(args)

    # Extracting parameters
    video_file_path = args['video_path']
    image_dir_path = args['image_dir_path']

    # Save frames as images and exit
    video = Video(video_file_path)
    video.convert_to_images_and_save(image_dir_path)

    exit()
```

Images to video

Here, I used cv2.imread to read the images from the given directory and then cv2.VideoWriter() class to convert them into a video file.

Functionality:

1. Convert images to video and save

Command-line interface options:

1. Image directory path
2. Video directory path
3. Frame rate (in fps)

How to run:

```
# For getting help about options
python images_to_video -h
# General run command
python images_to_video -i <img_dir_path> -v <video_path> -f <fps>
```

Salient features:

1. Handles image numbering very well (but we need to follow a specific format in naming the images to maintain the sequence, the format is <frame_num>_img.mp4)
2. Extensible code

Code

```
import cv2
import sys
import argparse
from os import listdir
from os.path import isfile, join
import os
import copy

class Images:
    def __init__(self, image_dir_path: str) -> None:
        self.image_dir_path = image_dir_path

    def extract_frame_num_from_file_name(self, file_name: str) -> int:
        '''
        Given a file name of form <frame_num>_img.mp4, this function extracts the frame_num and
        returns it
        '''
        num_str = ''
        for i in file_name:
            if i == '_':
                break
            num_str += i
        return int(num_str)

    def convert_to_video_and_save(self, video_file_path: str, fps: int):
        img_dir_path = self.image_dir_path
        if img_dir_path[-1] != '/':
            # print(img_dir_path+'/')
            img_dir_path = img_dir_path + '/'
            # print(img_dir_path)

        # Creating a dictionary with frame numbers as keys, file names as values
        image_files = [f for f in listdir(img_dir_path) if isfile(join(img_dir_path, f))]

        # image_file_dict is of form:
        #   image_file_dict = {
        #       <frame_num>: <file_name>
        #   }
        image_file_dict = {}
        for file_name in image_files:
            frame_num = self.extract_frame_num_from_file_name(file_name)
            image_file_dict[str(frame_num)] = file_name

        # Extracting images from files and appending them to video file
```

```
        img_array = []
        for i in range(len(image_file_dict)):
            # print(img_dir_path +image_file_dict[str(i)])
            img = cv2.imread(img_dir_path + image_file_dict[str(i)])
            # print(img!=None)

            height, width, layers = img.shape
            size = (width,height)
            img_array.append(img)

        print("Video file creation in process")
        vid_writer = cv2.VideoWriter(video_file_path, cv2.VideoWriter_fourcc(*'mp4v'), fps=int(fps), frameSize=size)
        print("Video file created and written!")
        for i in img_array:
            vid_writer.write(i)
        vid_writer.release()

if __name__=='__main__':
    parser = argparse.ArgumentParser(description='Video to Image options')
    parser.add_argument('-i', '--image_dir_path', help='Specify the absolute path to the directory where the images are to be stored', defa
    parser.add_argument('-v', '--video_path', help='Specify the absolute path to the video file that is to be converted to images', default
    parser.add_argument('-f', '--fps', help='Specify the frame rate for the generated video', default=20)
    args = vars(parser.parse_args())
    print(args)

    # Extracting parameters
    video_file_path = args['video_path']
    image_dir_path = args['image_dir_path']
    fps = args['fps']

    # Convert images to a video file and save it
    image_group = Images(image_dir_path)
    image_group.convert_to_video_and_save(video_file_path, fps)
    exit(0)
```

Challenges faced and lessons learned

One of the biggest challenges faced in this section was the proper naming of images. This is important because, while reading the images to convert them to a video file, specifying a proper sequence is important. So, to solve this, firstly, I have set a standard image naming convention:

```
<frame_num>_img.mp4
```

And, then I wrote a function, which takes care of any digit number mentioned as the frame_num in the above format and processes them accordingly. This function is given below and is defined in images_to_video.py code (Images to Videos section):

```
def extract_frame_num_from_file_name(self, file_name: str) -> int:
    '''
    Given a file name of form <frame_num>_img.mp4, this function extracts the frame_num and
    returns it
    '''
    num_str = ''
    for i in file_name:
        if i == '_':
            break
        num_str += i
    return int(num_str)
```

Since this is a basic question, I mainly learned to wield OpenCV tools and got comfortable with using them which increased my productivity in sections 2 and 3.

Section 2 (Capturing Images)

Problem

The aim of this section is to capture an image using an attached webcam or an external cam connected to a computer using Python and OpenCV. It also requires us to play the live video on the screen as we are getting ready for the picture.

Solution, challenges faced and lessons learned

I have used OpenCV's cv2.VideoCapture() class to capture the video from webcam. It is generally required to mention the camera number in the class. It is 0 for the default webcam. If there is any other camera connected to the computer, they are generally given 1, 2, ...etc. depending on how many other cameras were already connected to the computer. No big challenges were faced in this section, **however, I understood how cv2.waitKey() works and how to take input from the user via waitKey().**

Functionality:

1. Press 'c' to capture an image whenever you want to
2. Press 'q' to exit the webcam and the program subsequently

Command-line interface options:

1. Camera number (set to 0 by default) (takes values ≥ 0)
2. Image directory path

How to run:

```
# for getting help
python capture_from_webcam.py -h
# General run command
python capture_from_webcam.py -c <camera_num> -i <img_dir_path>
```

Code

```
# import the opencv library
import cv2
from datetime import datetime
import argparse

class Camera_capture:
    def __init__(self, camera_num = 0, image_dir_path = './Captured_images/') -> None:
        '''
        Parameters:
        camera_num - Each camera is assigned an integer number on your device. By default, if your
                     laptop has a webcam in-built, it takes '0' value. Other external cameras connected
                     will take values 1, 2, ...etc. If there is no webcam in-built, then the values for
                     external cameras start from 0.
        '''
        self.camera_num = camera_num
        self.image_dir_path = image_dir_path
        if self.image_dir_path[-1] != '/':
            self.image_dir_path += '/'

    def start_cam(self):
        vid = cv2.VideoCapture(self.camera_num)
        print("\nCamera {} started!\n".format(self.camera_num))
        print("Instructions:")
        print("1. To capture an image, press 'c' key on your keyboard")
        print("2. To close the webcam and exit program, press 'q' key on your keyboard")
        print("3. Captured images will be stored in {} directory, if that directory exists".format(self.image_dir_path))
        print("4. Image naming convention: img_<year>_<month>_<day>_<hours>_<mins>_<secs>.jpg\n")

        while(True):
            # Capture the video frame by frame
            ret, frame = vid.read()

            # Display the resulting frame
            cv2.imshow('frame', frame)

            # Press q to exit, c to capture an image
            wait_key = cv2.waitKey(1)

            if wait_key == ord('q'): # To quit the webcam capture
                print("Closed cam")
                break
            elif wait_key == ord('c'): # To capture and save current image frame from webcam stream
                current_time = datetime.now()
                current_time_formatted = current_time.strftime("%Y_%m_%d_%H_%M_%S")
                cv2.imwrite('{}img_{}.jpg'.format(self.image_dir_path, current_time_formatted), frame)
```

```
                print("Captured")

        # After the loop release the cap object
        vid.release()
        # Destroy all the windows
        cv2.destroyAllWindows()

if __name__=='__main__':
    parser = argparse.ArgumentParser(description='Capture images using live camera')
    parser.add_argument('-c', '--camera_num', type=int, help='Specify the camera number of the camera that will be used in this program', d
    parser.add_argument('-i', '--image_dir_path', help='Specify the absolute path to the directory where the images are to be stored', defa
    args = vars(parser.parse_args())
    # print(args)

    camera_cap = Camera_capture(camera_num=args['camera_num'], image_dir_path=args['image_dir_path'])
    camera_cap.start_cam()

    print("\nProgram execution ended!\n")
```

Section 3 (Chroma keying)

Problem

The main aim of this section is to composite two videos, one over another using a well-known technique called chroma-keying. In this technique, we record a foreground object with a green or blue matte. The background matte in this foreground video can be of any color, but we have to be careful that this color does not match any of the foreground object's colors. We then take the desired background video and create a composite of two, where the resultant video is supposed to have the foreground object with its background from the background video.

Solution

There are many methods to solve chroma-keying. The naive way is to record the color of the background matte in the foreground video, expand its limits and use it to mask the foreground video. I have used the holdout matting technique to define opacity and transparency in the form of an alpha channel. Then, I linearly combine the foreground and the background video frames using alpha-based parameters as follows:

$$I_{result} = \alpha.I_{foreground} + (1 - \alpha).I_{background}$$

The alpha_channel pixels take values from $[0, 1]$, where 0 indicates perfect transparency while 1 indicates complete opacity. An alpha channel pixel takes a value closer to one if that pixel corresponds to the foreground object, otherwise, it takes a value closer to 0 (indicating that the light has to pass through the foreground image and hit the background image, like a real holdout matte used in the 1900s). Alpha channel is an extra channel like RGB used mainly with the foreground video frames for the desired foreground object color extraction.

Challenges

1. Green spill: Often, most of the videos in the initial tryouts had a lot of green spills due to imperfect background mattes (imperfect green, shadows, ...etc.). There are five ways to solve it:
    a. Use a perfect contrasting matte (not possible to obtain one at this moment)
    b. Well lit background matte (not possible due to limited lighting facilities)
    c. Stand at a great distance from the background matte to prevent shadow spill (needs other extra setups such as different lighting, space, and ground sets)
    d. Expand the color limits of background matte (needs a lot of trial and error, but this worked well)
    e. Vlahos assumption and other advanced techniques (tuning Vlahos parameters k1 and k2 takes time and hence wasn't able to produce any tangible product from this experiment)
2. Obtaining proper videos (legally): Since the green spill problem is dominant (view the results in the Results section, point 1), I decided to use green screen videos directly from the internet. With some extended searching, I was able to finally get some good green screen foreground video
3. Speed of execution: Initially, I used to manipulate and construct the alpha_channel by iterating through all the pixels, which took more than 10 mins for each video of size 6 seconds. **Later on, I used vectorization effectively to reduce the time taken to less than a minute for**

each video of that size, now it takes about a minute for a 1 min video in the naive holdout method, which is a huge improvement. This problem still exists in Vlahos assumption parts, but will soon be solved over there too.

Functionality

1. Creates a composite of desired foreground and background video with an option to change fps

Command-line options

1. foreground video path
2. background video path
3. composite video path
4. frames rate (in fps)
5. method to use (currently, it accepts only one option - ‘naive’)
6. color limits of background matte in case naive method is used
7. The Vlahos arguments k1 and k2 (currently Vlahos is not included in the code, so, these options are for future)

How to run

```
# To get help regarding the arguments to use
python chroma_keying.py -h
# General run command
python chroma_keying.py -f <fg_path> -b <bg_path> -v <composite_path>\
 -fps <rate>\
 -m <method>\
 -c <color_limits> -k1 <vlahos_k1> -k2 <vlahos_k2>
```

Code

```
import argparse
import cv2
import numpy as np

class Video:
    def __init__(self, video_file_path: str) -> None:
        self.video_file_path = video_file_path
        self.video = cv2.VideoCapture(video_file_path)

    def convert_to_images_and_save(self, image_dir_path: str):
        if image_dir_path[-1] != '/':
            image_dir_path += '/'
        counter = 0
        while(self.video.isOpened()):
            ret, frame = self.video.read()
            if ret == False:
                print("Video stream ended! (or probably an error?)")
                break

            cv2.imwrite(image_dir_path+'{}_img.jpg'.format(counter), frame)
            counter += 1

        self.video.release()
        self.video = cv2.VideoCapture(video_file_path)

    def convert_to_images_and_return(self):
        counter = 0
        img_array = []
        while(self.video.isOpened()):
            ret, frame = self.video.read()
            if ret == False:
                print("Video stream ended! (or probably an error?)")
                break

            img_array.append(frame)
            counter += 1
```

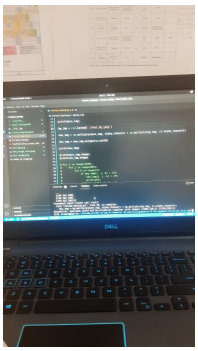
```
        self.video.release()
        self.video = cv2.VideoCapture(video_file_path)
        return img_array

class Chroma_key:
    def __init__(self, f_path: str, b_path: str, c_path: str, fps: int) -> None:
        '''
        Parameters:
        1. f_path: foreground video file path
        2. b_path: background video file path
        3. c_path: the composite video file path
        4. fps: frames per second of final composite video
        '''
        self.f_path = f_path
        self.b_path = b_path
        self.c_path = c_path
        self.fps = fps

    def create_naive_composite(self, matte_color_range: list):
        '''
        This method simply turns the matte that falls in the given color range transparent so that the
        opaque background video will take its position

        Parameters:
        1. matte_color_range: Color range of the holdout matte in foreground video

        Returns:
        None (Creates and stores the video in the specified c_path)
        '''
        # Load foreground and background as images and set matte color limits
        fg_video = Video(video_file_path=self.f_path)
        fg_img_array = fg_video.convert_to_images_and_return()
        bg_video = Video(video_file_path = self.b_path)
        bg_img_array = bg_video.convert_to_images_and_return()
        bg_img_array = bg_img_array[:1238]
        print(len(bg_img_array))
        print(len(fg_img_array))
        color_limits = matte_color_range

        # Start compositing the videos
        print("Compositing started!")
        img_res_array = []
        n_images = min(len(fg_img_array), len(bg_img_array))
        height, width, layers = fg_img_array[0].shape
        size = (width,height)
        print("Video file creation in process")
        if self.c_path == None:
            vid_writer = cv2.VideoWriter('./Joker_with_peter_result_fps={}.mp4'.format(self.fps), cv2.VideoWriter_fourcc(*'mp4v'), fps=int(
        else:
            vid_writer = cv2.VideoWriter(self.c_path, cv2.VideoWriter_fourcc(*'mp4v'), fps=int(self.fps), frameSize=size)

        for i in range(n_images):
            f_img = fg_img_array[i]
            b_img = bg_img_array[i]
            # Initialize the alpha channel and resize the images from the background video file
            alpha_channel = np.ones(shape=(f_img.shape[0], f_img.shape[1], 3))
            b_img = cv2.resize(b_img, ( f_img.shape[1], f_img.shape[0]))
            # Create masks based on holdout matte color (Vectorized and therefore very fast)
            red_ch_mask = np.logical_and(f_img[:, :, 0] >= matte_color_range[0][0], f_img[:, :, 0] <= matte_color_range[0][1])
            green_ch_mask = np.logical_and(f_img[:, :, 1] >= matte_color_range[1][0], f_img[:, :, 1] <= matte_color_range[1][1])
            blue_ch_mask = np.logical_and(f_img[:, :, 2] >= matte_color_range[2][0], f_img[:, :, 2] <= matte_color_range[2][1])
            # Update the alpha channel based on the masks created
            for i in range(3):
                alpha_channel[:, :, i] = np.logical_not(np.logical_and(np.logical_and(red_ch_mask, green_ch_mask), blue_ch_mask))
            # Add background image to the alpha multiplied foreground
            img_res = np.multiply(f_img, alpha_channel) + np.multiply(b_img, (1 - alpha_channel))
            img_res = img_res.astype(np.uint8)
            # cv2.imshow("image", img_res)
            # cv2.waitKey(0)
            # img_res_array.append(img_res)

            vid_writer.write(img_res)
        vid_writer.release()

        print("Saved!")

if __name__=='__main__':
    parser = argparse.ArgumentParser(description='Chroma keying (Compositing one video over another)')
    parser.add_argument('-f', '--foreground_video_path', help='Specify the absolute path to the foreground video file that is to be used',
    parser.add_argument('-b', '--background_video_path', help='Specify the absolute path to the background video file that is to be used',
    parser.add_argument('-v', '--composite_video_path', help='Specify the absolute path (with filename included) where the composite video
```

```
    parser.add_argument('-fps', '--frames_per_second', help='Specify the frames per second (integer > 0) of the composite video (by default
    parser.add_argument('-m', '--method', help='This variable sets what method to use to composite the two videos. Valid options include: \
    parser.add_argument('-c', '--naive_method_matte_color_limits', help='If naive method is chosen, please input the color limits of the ba
    parser.add_argument('-k1', '--vlahos_method_k1', help='If vlahos method is chosen, please input the value of k1', default=0)
    parser.add_argument('-k2', '--vlahos_method_k2', help='If vlahos method is chosen, please input the value of k2', default=1)
    args = vars(parser.parse_args())
    print(args)

    # Create an instance of the chroma key class
    chroma_key = Chroma_key(f_path=args['foreground_video_path'], b_path=args['background_video_path'], c_path=args['composite_video_path']
                            fps=args['frames_per_second'])

    # check for the method and apply
    if args['method']=='naive':
        chroma_key.create_naive_composite(matte_color_range=args['naive_method_matte_color_limits'])
    elif args['method']=='vlahos':
        pass
    else:
        print("Invalid method name. Please use a valid method name. For more information, run 'python chroma_keying.py -h'")

    exit(0)
```

Results

1. Green spill problem in a near-green real background matte (taken in my hostel room)

Original Foreground

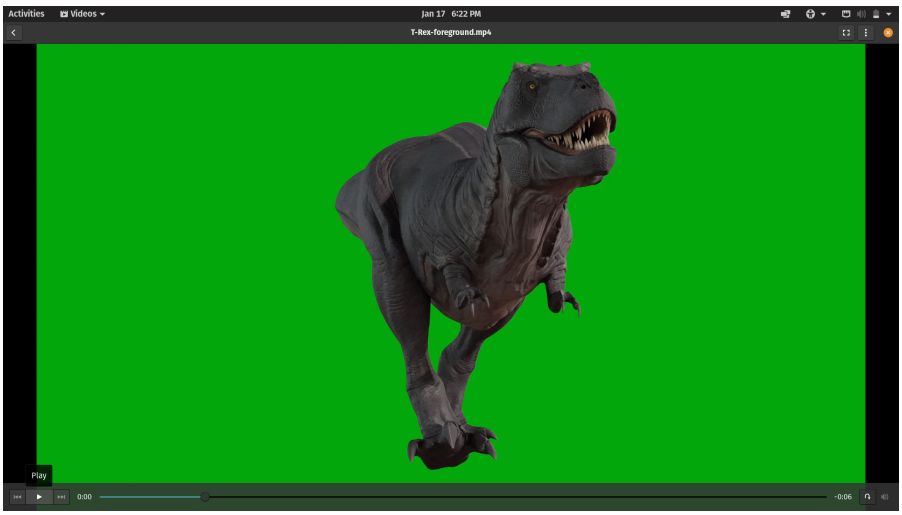


Original Background

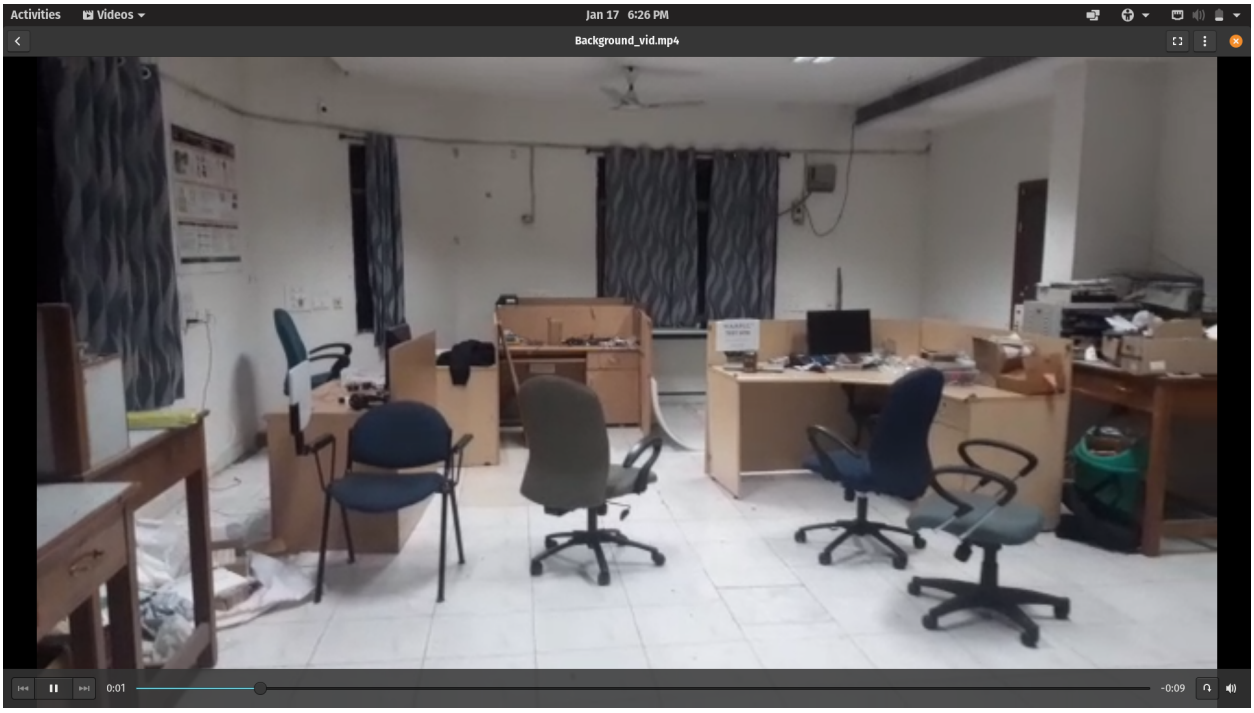
Result (low color limits)

Result with expanded color limits

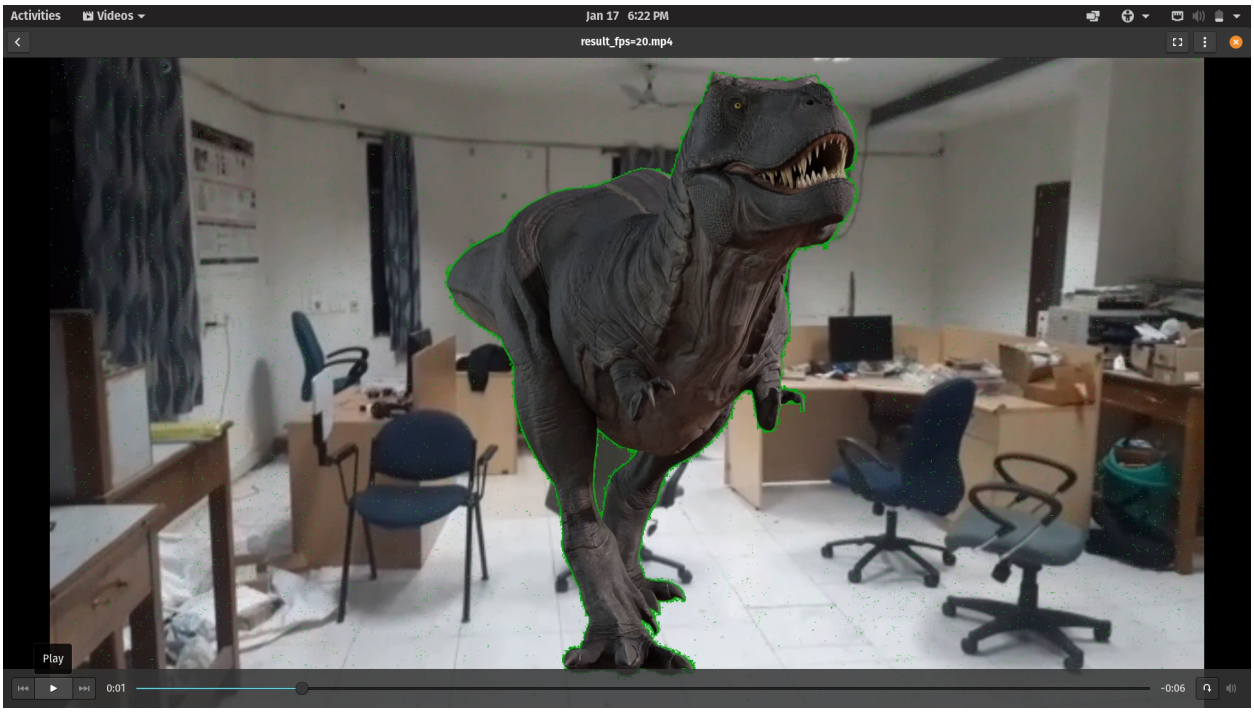
2. Following are the images from the resulting video generated using the holdout matting method



Dynamic foreground video (from Pixabay - https://pixabay.com/videos/search/green screen/)



Dynamic background video (RRC main lab taken using my phone's camera)



Resulting video obtained on compositing the above foreground and background videos

Other results:

Please use this link to find more results.

Conclusion

This assignment helped me learn, and wield OpenCV's strong set of tools freely. Personally, I loved it as I learned a lot from it. The Stanford paper mentioned in the assignment pdf was extremely useful in building the concepts (though it was initially hard to grasp). Some great concepts learned among other many include:

1. effects of Pre-multiplied alpha
2. Vlahos assumption
3. Holdout matting and alpha channel

Thank you!