CS 6475 Final Project: Video Stabilization

Thanh Le

Fall 2020

This project is based on "Auto-directed Video Stabilization with Robust L1 Optimal Camera Paths",

by Matthias Grundmann, Vivek Kwatra, and Irfan Essa

Project Goal

1. What was your original project scope?

The scope of this project is to develop an algorithm to that stabilize videos by removing undesired motions. Post-process video stabilization has three main steps:

- Camera Motion – Estimate the original (shaky) camera path of the video
- Path Stabilization – Estimate a new smooth camera path via Linear Programming
- Cropping – Synthesize the stabilized video using the estimated smooth camera path

2. What motivated you to choose this project?

I chose this project because I felt I could leverage code from previous assignments such as A3 – Panoramas and A5 – Video Texture. There is also a module on this topic, 06-03 Video Stabilization, on Canvas. This made the research paper easier to digest compared to the other research papers.

Scope Changes

Did you run into issues that required you to change project scope? If yes, give a detailed explanation of what changed. If no, say so.

I ran out of time before I could investigate adding a saliency constraint for the optimal camera path. This would perform directed video stabilization. This feature requires a significant amount of changes to the base stabilization code and there is no guarantee I would be able to successfully implement the feature. Thus I didn't include it in my scope.

Demonstration of Original Result Sets

1. Demonstrate **at least three complete result sets**. If your results can easily be shown here in your slides, please do. Explain in detail what you are showing.
    - Label each input/result set (ex: Set 1)

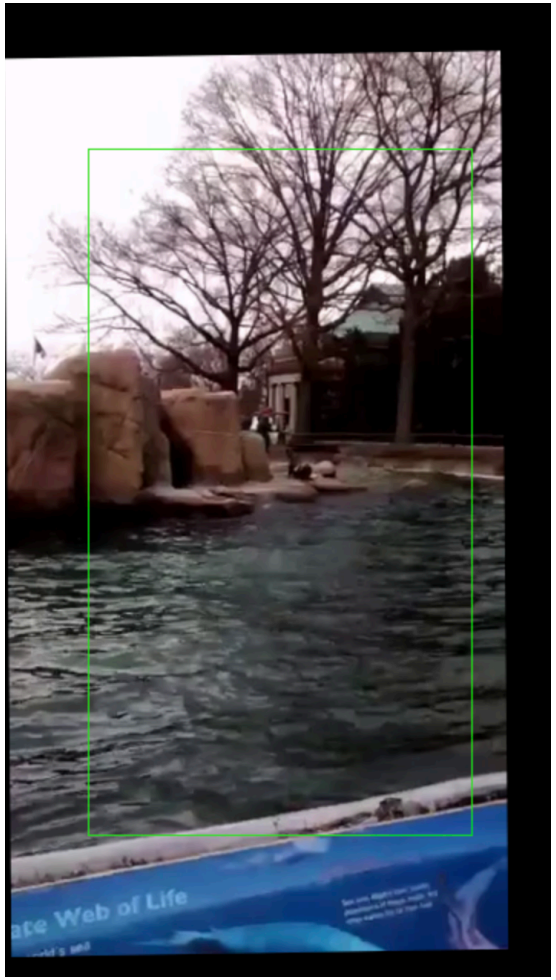
2. **Provide a WORKING link to your main directory here in the report.** All resources (individual videos, gifs, image sets, etc.) must be hosted under one main directory on GT Box. Detailed instructions are in the project's README on GitHub. If we cannot access your resources separately, there will be deductions.

GT Box: https://gatech.box.com/s/h97a2tm7d2ycz19plw9wsedqmkan8fy7

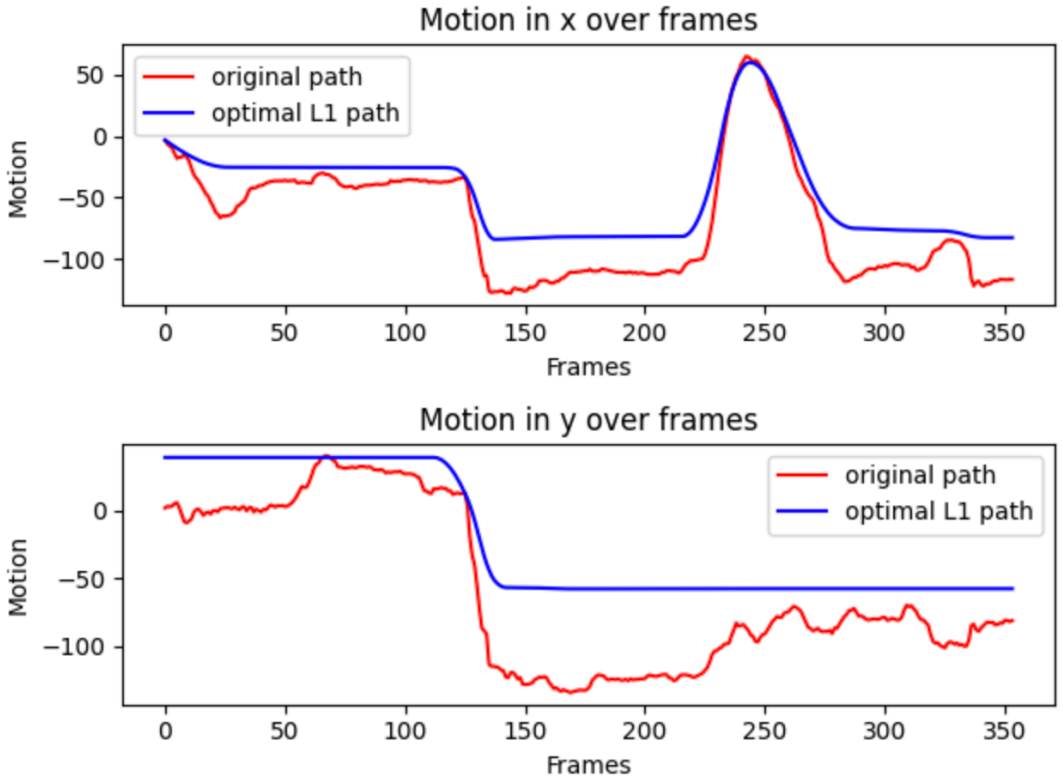
- Readme.txt
- Input 1, Input 2, and Input 3 (original shaky videos)
- Result 1, Result 2, and Result 3 (stabilized videos, motion plots, and frames)
- Skater original, and Skater stabilized videos

Result 1: Seal

- Original video: https://gatech.box.com/s/3m5tj151cuyz5ezviqv3cu2bq87khvv0
- Output video: https://gatech.box.com/s/3kvhexf3jlke56mawnlk4tmsvokoglbh
- Description: This video was taken a few years ago at the Central Park Zoo. The focus of this video are the seals.
- Comment: I am satisfied with the final output of this video. It is much more stabilize than the original video. The optimal path is very close and smooth compare to the original path of the camera.



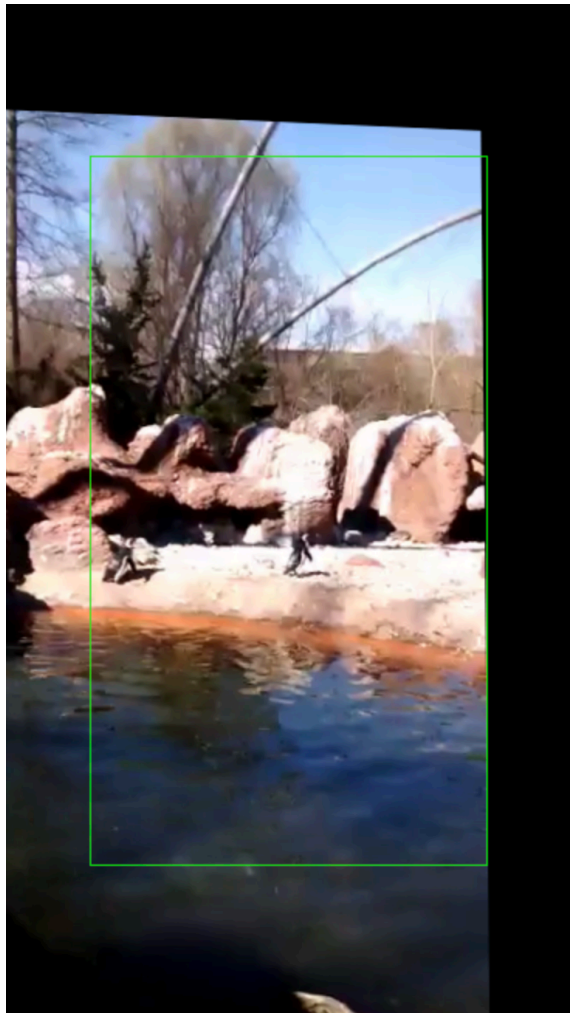
Frame0001.png - The first frame of the video after transformation was applied



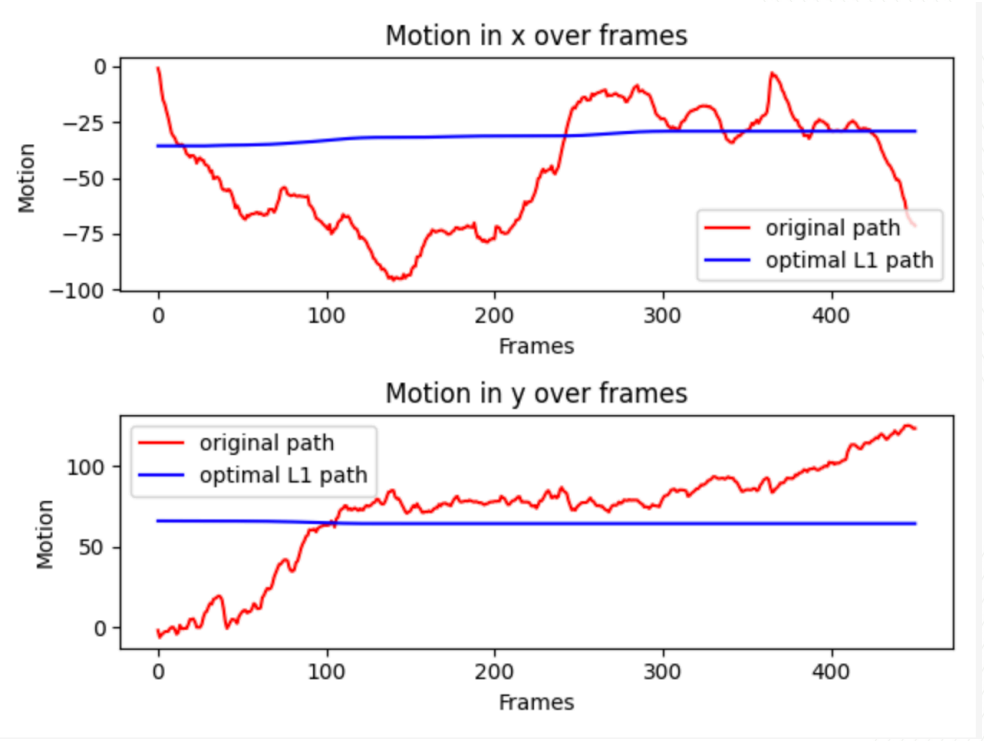
Motion plots for Result 1 video

Result 2: Penguins

- Original video: https://gatech.box.com/s/ia40q5hp6kyf24tl9k9uwz9ahm1q924v
- Output video: https://gatech.box.com/s/ujptbb8jgt1bzgvr3y434ca78vwv7xl8
- Description: This video was taken a few years ago at the Central Park Zoo. The focus of this video are the penguins.
- Comment: The final video is less shaky than the original video, however, the background is still a bit wobbly. The optimal path is not the best, it doesn't closely follow original camera path.



Frame0001.png - The first frame of the video after transformation was applied



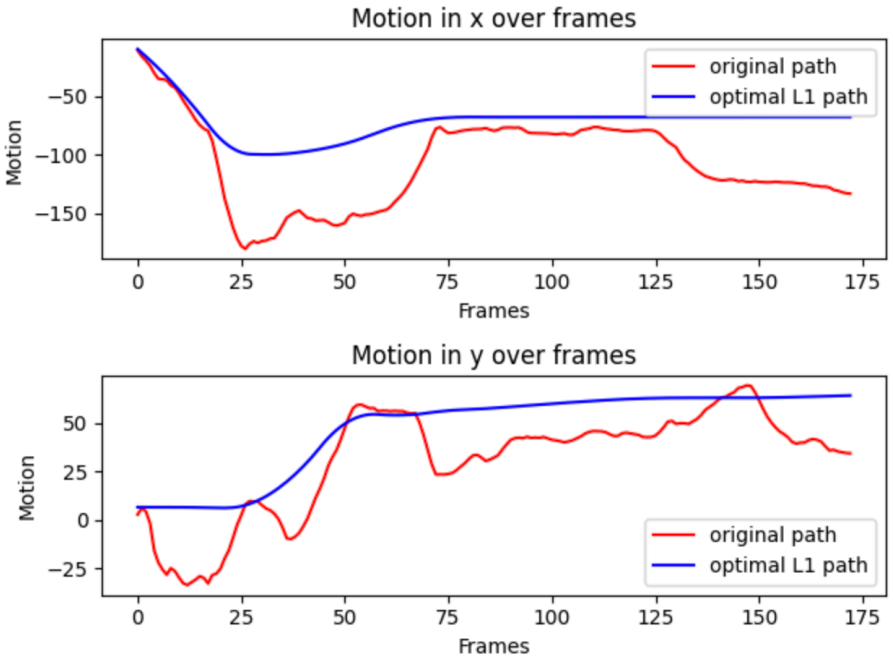
Motion plots for Result 2 video

Georgia Tech
CREATING THE NEXT

Result 3: Birds

- Original video: https://gatech.box.com/s/r2exes3pbplzl6hd5set1x6jlff2ccej
- Output video: https://gatech.box.com/s/j2sb24fxbat8tr442tpns1ryq7rzq5rr
- Description: This video was taken a few years ago at the Central Park Zoo. The focus of this video are the pink birds.
- Comment: I am satisfied with the final output of this video. It is much more stabilize than the original video. The optimal path is very close and smooth compare to the original path of the camera.

Frame0001.png - The first frame of the video after transformation was applied

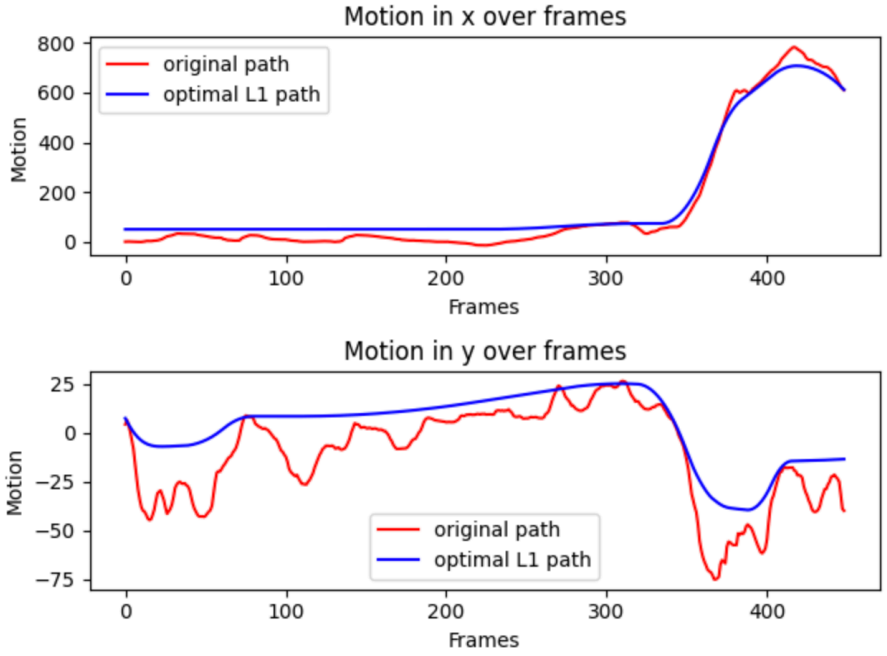


Motion plots for Result 3 video

YouTube: Skater

- Original video: https://gatech.box.com/s/zgr81aqhe1wzp3mhkrdbz3z6hezwcn9v
- Output video: https://gatech.box.com/s/9xlr59gbg0qmnsla1utop6ujv8nwu4ad
- Description: This video is from YouTube. The focus of of video is the skater in the center.
- Comment: This video was used in the paper. I am satisfied with the final output of this video. It is much more stabilize than the original video. The optimal path is very close and smooth compare to the original path of the camera.

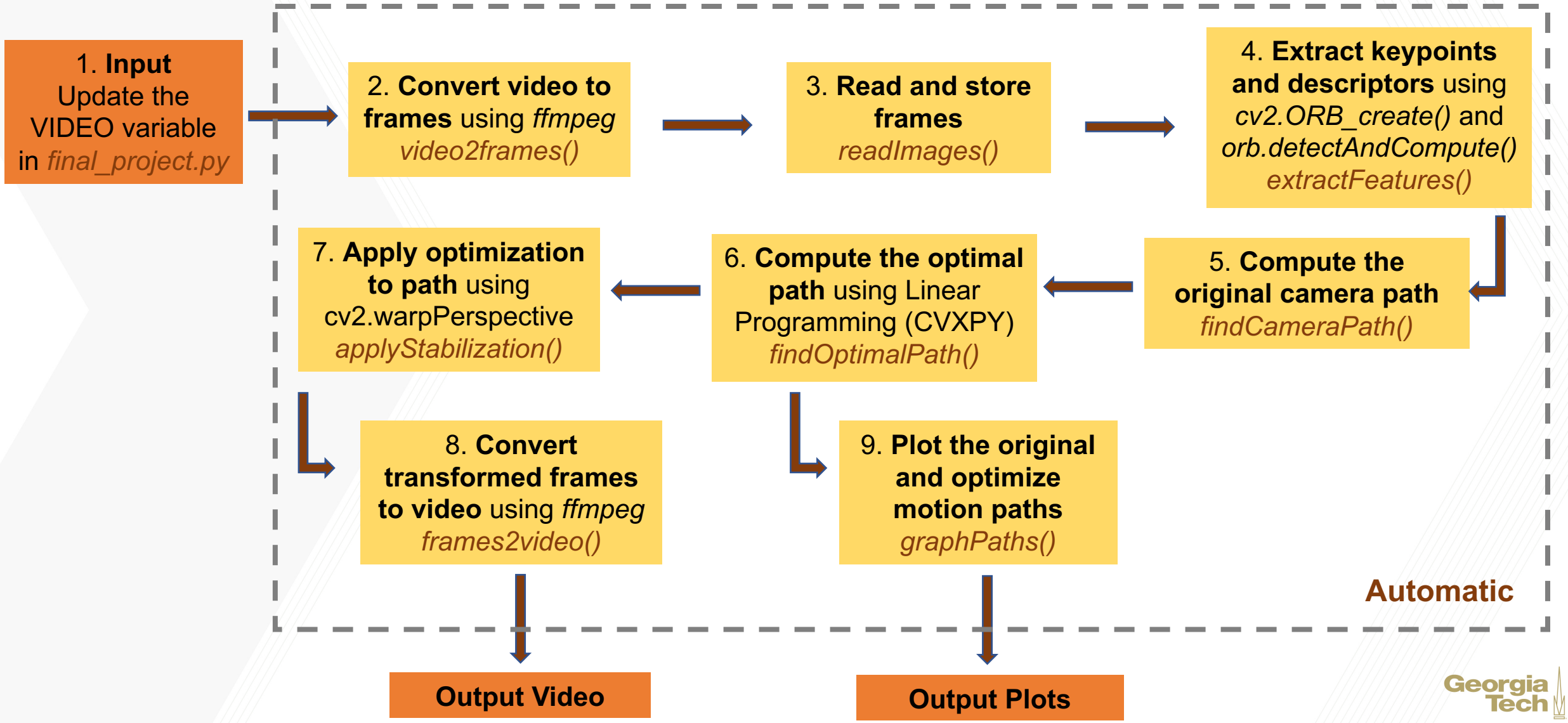
Frame0001.png - The first frame of the video after transformation was applied



Motion plots for the Skater video

Georgia Tech
CREATING THE NEXT

Computational Pipeline



Georgia Tech
CREATING THE NEXT

Project Development

Setup:

1. **Input video** – During development of the code I simply used the skater video. I didn't start thinking about what videos to use for input until after I had gotten *applyStabilization()* to somewhat work. I know I have shaky camera video in my archive, so I didn't worry too much about creating original input video. I made the decision to prioritize speed over output quality, so all my input videos are less 480p resolution and about 10-15 second long.

2. ***video2frames() and frames2video()*** - I leveraged the information from A5-Video Texture assignment to write these functions. These functions main purpose is to convert video into frames and vice versa frames to video. The functions will automatically create a source directory and place the output there. From my experience in A5, I made this function a part of the main script so I wouldn't have to go back and forth between the IDE and terminal to use *ffmpeg*. Thanks to this I saved a lot of time.

3. ***readImages()*** – The purpose of this function is to reads in input images from an image directory. This code is taken from A5-Video Textures assignment in the main.py file. I assume this is okay to do since this code was given for student to use in the previous assignment.

Project Development

Camera Motion:

1. ***extractFeatures()*** – Leveraged from Module 4 and 5 Demo Code – feature detection and feature matching. This function find the key points and descriptors with ORB detector for each frame. The demo code are for an outdated version of python and OpenCV, so I had a ton of errors when I tried to run it. Nonetheless, since the framework is already there it wasn't difficult to find equivalent updated function. The demo code suggested the *cv2.SIFT()* function however this wasn't working and caused the bunch of error. I ended up using *cv2.ORB_create()* function since it was much faster than the alterative *cv2.SIFT_create()* detector.

2. ***findCameraPath()*** – The paper didn't explicitly explain how to do this but after looking at a few OpenCV tutorial on feature matching it was clear I needed to look for Affine Transformations between every two consecutive frames. Again, I leverage the Demo code and a few OpenCV tutorial to complete this function.

Project Development

Path Stabilization and Cropping

1. ***findOptimalPath()*** – Needless to say, this is the most important part of the project. Figuring out what to do from the research paper was not an easy task. I had to reread the paper more than 5 times and constantly referencing back for what to do next. I also looked at other research and reference material on Video Stabilization and L1 Optimal Camera Path to understand the equations and variables presented in the paper. I spend roughly 2 weeks on this section of the code alone. I didn't know about Linear Programming, so I spend a few days trying to get a rough idea of what it is and how to solve optimization problem using it. Luckily, it wasn't too bad and other students shared on Piazza and Slack that they used CVXPY library to accomplished this LP programming. Finally, I was able to follow the equations from section 2.1 and Algorithm 1 from the paper to set up the objective and constraints to calculate the optimal camera path.
2. ***graphPaths()*** – I simply extract the x and y values from the transformation matrix and plot them over the number of frames. I used Matplotlib library to plot my charts.
3. ***applyStabilization()*** – I have less than 2 days to work on this and thus I was not able to produce the best-looking results. The output videos are all less unsteady than their original counterpart, but they are still a bit wobbly. I suspect that I was not able to apply the translation properly onto the original frames. This function output two set of transformed frames. One set has a green rectangle on top and other set is a cropped version without the black border.

2. Both good and failed interim results (from various parts of your work and pipeline).

The good interim result would be the skater video shown in previous slide. The code produced a very good optimal path, and the video is indeed less shaky than the original video. The *graphPaths*() function was written before the *applyStabilization*() function to help me visualize the optional path. I was ecstatic when I got the plots working.

I mixed up the frame's width and height multiple time throughout the code and this caused a lot of errors. It took me a long time to debug the code to figure out that was the issue. One of the time was in the Inclusion Constraints in *findOptimalPath*() and another time was when I was working on the *applyStabilization*() function

I also kept getting "error: (-215:Assertion failed) (type == CV_8U && dtype == CV_32S) || dtype == CV_32F in function 'batchDistance' match". In the beginning it was due to my environment not properly setup in Pycharm IDE. I also got a few other OpenCV errors due to this. Later, this turn out to be coming from v2.BFMatcher.match() function. The reason is because cv2.orb.detectAndCompute() was not able to find any key point or descriptor on a frame. After I got the skater video working, I tried to run the code with a video that I took from a concert. The video was apparently too dark, and the orb function was not able to find any features. I didn't have time to further investigate. But I used this as a baseline and looked for videos I have that have lots of feature so I wouldn't run into this situation again.

Project Development

3. What did you finish? What is not finished?

I finished all the steps in my computational pipeline. I still need more time to polish the *extractFeatures(), findCameraPath(), and applyStabilization()* functions. Currently the code will not work for video with few details. The extract features function will find no key points and descriptors and the code will fail at the find camera path function since no match will be found for that frame and its adjacent frame. I didn't write any code to handle this and just output an error. The apply stabilization function is not 100% complete. I applied the transformation by calling cv2.warpPerspective but the final video is cropped based on the center points and crop ratio.

4. What would you do differently? Explain.

I didn't have time to create shaky videos for my inputs. So I just looked through my phone history to find shaky video. The videos I used are from 2016 when I have a Motorola phone with no built-in stabilization video feature like my current Samsung Galaxy Note. Given more time I would had went outside and capture better input videos.

I would also like to make my code command line compatible. Currently user must go into the main file, final_project.py, to update the VIDEO variable which stores the name of the input video of interest before running the code. Making the code command line compatible would make it easier to run the code and take in user arguments.

Computation: Discussion of Code Functions

Walk through your code functions. You MUST:

- Show and discuss code snippets. Explain the purpose of the code that you are presenting.

- Explain the major algorithms. If you used them, show that you understand them in your own words. Do **NOT** copy from the technical papers.

I ran out of time, so I am not able to properly discuss my algorithms. I placed my code snippets with comments over the next few slides. I hope this will at least give me partial credits.

Find Original Camera Path

```python
# Module 4 and 5 Demo Code - feature_detection and feature matching
# https://docs.opencv.org/master/dc/dc3/tutorial_py_matcher.html
def extractFeatures(frames):
    n = len(frames)
    features = []

    for i in range(n):
        # Initiate SIFT detector
        # orb = cv2.SIFT_create()   # slow
        orb = cv2.ORB_create()
        img_gray = cv2.cvtColor(frames[i], cv2.COLOR_BGR2GRAY)
        # find the keypoints and descriptors with SIFT
        kp, des = orb.detectAndCompute(img_gray, None)

        if np.array(kp).shape[0] == 0:
            print("frame: " + str(i) + " has no keypoints")

        features.append((kp, des))

    return features
```

```python
# panorama.py (A4), feature_matching.py and warp_affine.py (Module 4 and 5 Demo Code)
# https://docs.opencv.org/master/d1/de0/tutorial_py_feature_homography.html
def findCameraPath(frames, features):
    n = len(frames)
    path = []

    # Create BFMatcher (Brute Force Matcher) object
    bf = cv2.BFMatcher(normType=cv2.NORM_HAMMING, crossCheck=False)
    # bf = cv2.BFMatcher(normType=cv2.NORM_L2, crossCheck=False)

    for i in range(n - 1):
        kp1, des1 = features[i][0], features[i][1]
        kp2, des2 = features[i + 1][0], features[i + 1][1]
        # Match descriptors
        matches = bf.match(des1, des2)
        # get the matching key points for each of the frames
        pts1 = np.float32([kp1[m.queryIdx].pt for m in matches]).reshape(-1, 1, 2)
        pst2 = np.float32([kp2[m.trainIdx].pt for m in matches]).reshape(-1, 1, 2)

        # Computes an optimal affine transformation between two point sets
        mat = cv2.estimateAffine2D(pts1, pst2, method=cv2.RANSAC)[0]
        # Affine => | a b c |
        # (3 x 3)   | d e f |
        #           | 0 0 1 |
        mat = np.vstack([mat, [0, 0, 1]])
        path.append(mat)

    return path
```

Find Optimal Path - Set and Objective

```python
# https://publik.tuwien.ac.at/files/publik_274527.pdf (pg 16 - 19)
# [Grundmann, Kwatra, and Essa (2011)] Algorithm: Summarized LP for the optimal camera path
# https://www.cc.gatech.edu/cpl/projects/videostabilization/stabilization.pdf
def findOptimalPath(original_path, frame_size, crop_ratio=1):
    n = len(original_path)

    # Weights of the objective eq (see Figure 8(d) from paper)
    w1, w2, w3 = 10, 1, 100

    # Use weighting of 100:1 for affine to translational parts
    affine_weights = np.transpose([1, 1, 100, 100, 100, 100])

    # slack variables and constraint list
    pt = cp.Variable((n, 6))    # optimized path
    e1 = cp.Variable((n, 6))
    e2 = cp.Variable((n, 6))
    e3 = cp.Variable((n, 6))
    constraints = []

    # Objective Eq (3)
    objective = cp.Minimize(cp.sum(w1 * e1 @ affine_weights + w2 * e2 @ affine_weights + w3 * e3 @ affine_weights))
```

Find Optimal Path – Smoothness constraints

```
# 1. Smoothness constraints
for i in range(n - 3):
    # Pt is the optimized path
    # pt = (dx, dy, a, b, c, d).T
    #       0    1  2  3  4  5
    # Bt is the transformation between old and new path
    # Bt = | a  c  0 |
    #      | b  d  0 |
    #      | dx dy 1 |
    Bt = np.array([[pt[i, 2], pt[i, 4], 0],
                   [pt[i, 3], pt[i, 5], 0],
                   [pt[i, 0], pt[i, 1], 1]])

    Bt1 = np.array([[pt[i + 1, 2], pt[i + 1, 4], 0],
                    [pt[i + 1, 3], pt[i + 1, 5], 0],
                    [pt[i + 1, 0], pt[i + 1, 1], 1]])

    Bt2 = np.array([[pt[i + 2, 2], pt[i + 2, 4], 0],
                    [pt[i + 2, 3], pt[i + 2, 5], 0],
                    [pt[i + 2, 0], pt[i + 2, 1], 1]])

    Bt3 = np.array([[pt[i + 3, 2], pt[i + 3, 4], 0],
                    [pt[i + 3, 3], pt[i + 3, 5], 0],
                    [pt[i + 3, 0], pt[i + 3, 1], 1]])
```

Find Optimal Path – Smoothness constraints

```
        # Eq (4) Rt = Ft+1 @ Bt+1 - Bt
        # original_path = | a b dx |     original_path.T = | a  c  0 |
        #                 | c d dy |                       | b  d  0 |
        #                 | 0 0  1 |                       | dx dy 1 |
        Rt = np.transpose(original_path[i + 1]) @ Bt1 - Bt
        Rt1 = np.transpose(original_path[i + 2]) @ Bt2 - Bt1
        Rt2 = np.transpose(original_path[i + 3]) @ Bt3 - Bt2

        # rearrange Residuals [dx, dy, a, b, c, d]
        Rt = np.array([Rt[2, 0], Rt[2, 1], Rt[0, 0], Rt[1, 0], Rt[0, 1], Rt[1, 1]])
        Rt1 = np.array([Rt1[2, 0], Rt1[2, 1], Rt1[0, 0], Rt1[1, 0], Rt1[0, 1], Rt1[1, 1]])
        Rt2 = np.array([Rt2[2, 0], Rt2[2, 1], Rt2[0, 0], Rt2[1, 0], Rt2[0, 1], Rt2[1, 1]])

        # -e1 <= Rt(p) <= e1
        # -e2 <= Rt1(p) - Rt(p) <= e2
        # -e3 <= Rt2(p) -2*Rt1(p) - Rt(p) <= e3
        for j in range(6):
            constraints.append(-e1[i, j] <= Rt[j])
            constraints.append(Rt[j] <= e1[i, j])

            constraints.append(-e2[i, j] <= Rt1[j] - Rt[j])
            constraints.append(Rt1[j] - Rt[j] <= e2[i, j])

            constraints.append(-e3[i, j] <= Rt2[j] - 2 * Rt1[j] + Rt[j])
            constraints.append(Rt2[j] - 2 * Rt1[j] + Rt[j] <= e3[i, j])
```

Georgia Tech
CREATING THE NEXT

Find Optimal Path – Proximity constraints

```python
# e >= 0
constraints.append(e1 >= 0)
constraints.append(e2 >= 0)
constraints.append(e3 >= 0)

# 2. Proximity Constraints
# 0.9 <= a, d <= 1.1
# -0.1 <= b, c <= 0.1
# -0.05 <= b + c <= 0.05
# -0.1 <= a - d <= 0.1
lb = np.array([0.9, -0.1, -0.1, 0.9, -0.05, -0.1])
ub = np.array([1.1, 0.1, 0.1, 1.1, 0.05, 0.1])
# U =        a b c d b+c a-d
#       tx | 0 0 0 0  0   0 |
#       ty | 0 0 0 0  0   0 |
#       a  | 1 0 0 0  0   1 |
#       b  | 0 1 0 0  1   0 |
#       c  | 0 0 1 0  1   0 |
#       d  | 0 0 0 1  0  -1 |
U = np.array([[0, 0, 0, 0, 0, 0],
              [0, 0, 0, 0, 0, 0],
              [1, 0, 0, 0, 0, 1],
              [0, 1, 0, 0, 1, 0],
              [0, 0, 1, 0, 1, 0],
              [0, 0, 0, 1, 0, -1]])
# Eq (7) lb <= U @ pt <= ub
for i in range(n):
    constraints.append(lb <= pt[i, :] @ U)
    constraints.append(pt[i, :] @ U <= ub)
```

Georgia Tech
CREATING THE NEXT

Find Optimal Path – Inclusion constraints

```
    # 3. Inclusion Constraints
    # Eq (8)
    # | 0 | <= | 1 0 cx cy 0  0  | <= | w |
    # | 0 |    | 0 1 0  0  cx cy |    | h |
    h, w = frame_size[:2]
    center_w, center_h = w // 2, h // 2
    dw, dh = crop_ratio * center_w, crop_ratio * center_h

    # corners
    top_left = (center_w - dw, center_h - dh)
    top_right = (center_w + dw, center_h - dh)
    bottom_left = (center_w - dw, center_h + dh)
    bottom_right = (center_w + dw, center_h + dh)

    crop_frame = [top_left, top_right, bottom_right, bottom_left]
    for i in range(len(crop_frame)):
        cx, cy = crop_frame[i]
        constraints.append(0 <= pt @ np.transpose([1, 0, cx, cy, 0, 0]))
        constraints.append(pt @ np.transpose([1, 0, cx, cy, 0, 0]) <= w)

        constraints.append(0 <= pt @ np.transpose([0, 1, 0, 0, cx, cy]))
        constraints.append(pt @ np.transpose([0, 1, 0, 0, cx, cy]) <= h)
```

Georgia Tech
CREATING THE NEXT

Find Optimal Path – Running CVXPY Solver

```
    print('Running ECOS solver ...')
    problem = cp.Problem(objective, constraints)
    problem.solve(solver=cp.ECOS)
    # print("optimal value", problem.value)
    # print(pt.value)

    # Convert from parametric to matrix form
    # | dx dy a b c d | => | a b dx |
    #   0  1  2 3 4 5      | c d dy |
    #                      | 0 0 1  |
    optimal_path = []
    for i in range(n):
        mat = np.array([[pt.value[i, 2], pt.value[i, 3], pt.value[i, 0]],
                        [pt.value[i, 4], pt.value[i, 5], pt.value[i, 1]],
                        [0, 0, 1]])
        optimal_path.append(mat)

    return optimal_path
```

Georgia Tech
CREATING THE NEXT

Apply Stabilization

```python
# panorama.py (A4)
# https://www.pyimagesearch.com/2014/05/05/building-pokedex-python-opencv-perspective-warping-step-5-6/
def applyStabilization(filename, original_frames, optimal_path, crop_ratio=1):
    n = len(original_frames)
    h, w = original_frames[0].shape[:2]

    # Create folder for the output frames
    rect_output_path = os.path.join("output", filename, "rect")
    crop_output_path = os.path.join("output", filename, "crop")

    exist = os.path.exists(rect_output_path)
    if not exist:
        os.makedirs(rect_output_path)
        print("Directory '% s' created" % rect_output_path)

    exist = os.path.exists(crop_output_path)
    if not exist:
        os.makedirs(crop_output_path)
        print("Directory '% s' created" % crop_output_path)
```

```python
    # rect_frames = []
    # crop_frames = []
    for i in range(1, n):
        # apply transformation by calling cv2.warpPerspective
        warp_frame = cv2.warpPerspective(original_frames[i], optimal_path[i - 1], (w, h))
        rect, crop = drawRectangle(warp_frame, crop_ratio)
        frame = 'frame{0:04d}.png'.format(i)
        cv2.imwrite(rect_output_path + '/' + frame, rect)
        cv2.imwrite(crop_output_path + '/' + frame, crop)

        # rect_frames.append(rect)
        # crop_frames.append(crop)

    return rect_output_path, crop_output_path
```

Georgia Tech
CREATING THE NEXT

Main

```python
import stabilization as st

# user inputs
VIDEO = "penguins.mp4"
CROP_RATIO = 0.8


def main():
    video_name = VIDEO.replace('.', '_')
    # print("Converted video to frames ...")
    frames_path = st.video2frames(VIDEO)
    print("Reading images ...")
    images_list = st.readImages(frames_path)
    print('Total frames: ', len(images_list))
    image_size = images_list[0].shape
    print("Extracting features ...")
    features = st.extractFeatures(images_list)
    print("Calculating original camera path ...")
    original_path = st.findCameraPath(images_list, features)
    print("Calculating optimal camera path ...")
    optimal_path = st.findOptimalPath(original_path, image_size, CROP_RATIO)
    rect_output_path, crop_output_path = st.applyStabilization(video_name, images_list, optimal_path, 0.6)
    print("Converting frames to video ...")
    st.frames2video(rect_output_path)
    st.frames2video(crop_output_path)
    print("Plotting camera paths ...")
    st.graphPaths(video_name, original_path, optimal_path)
```

Descriptions of libraries/packages

- OS - The OS module in python provides functions for interacting with the operating system. I used this module to run the command line code in video2frames() and frames2video() function in stabilization.py
- Numpy - NumPy is a Python library used for working with arrays and matrices. NumPy stands for Numerical Python.
- CV2 – OpenCV is a library of Python bindings designed to solve computer vision problems. I used to find matching features and computing affine transform.
- Glob - This module is useful in any situation where the program needs to look for a list of files on the filesystem with names matching a pattern.
- Matplotlib - This is a comprehensive library for creating static, animated, and interactive visualizations in Python. I used to plot the motions over frames in the *graphPaths() function*.
- CVXPY - Python-embedded modeling language for convex optimization problems. This library is heavily leverage in the *findOptimalPath()* to calculate the optimal path based on the algorithm presented In the paper.

List of Code Files

- final_project.py (required)
  - This is the main file. Open this file to update user input, i.e. video filename, and run the code.

- stabilization.py
  - This file contains all of the function definitions related to video stabilization.

Resources

Grundmann, Kwatra, and Essa (2011)

Algorithm: Summarized LP for the optimal camera path
https://www.cc.gatech.edu/cpl/projects/videostabilization/stabilization.pdf

https://www.youtube.com/watch?v=QdXugkXBTbc&ab_channel=GoogleDevelopers

https://patentimages.storage.googleapis.com/a0/d1/f0/130ef429e3c974/US8531535.pdf

How to create a video from images with FFmpeg?

https://stackoverflow.com/questions/24961127/how-to-create-a-video-from-images-with-ffmpeg

OpenCV - Feature Matching, Feature Matching + Homography to find Objects

https://docs.opencv.org/master/dc/dc3/tutorial_py_matcher.html

https://docs.opencv.org/master/d1/de0/tutorial_py_feature_homography.html

Module 4 and 5 Demo Code – feature matching, feature detection, and image_transforms

A3 - Panorama

A5- Video Textures

Piazza @ 1072 Video Stabilization Discussion Thread

https://piazza.com/class/kdkh6c8vx7z5tf?cid=1072

Resources

Techniques for Improving Mobile Video Creation (pg 16 - 19)
https://publik.tuwien.ac.at/files/publik_274527.pdf

Python OpenCV How to draw ractangle center of image and crop image inside rectangle?
https://stackoverflow.com/questions/46795669/python-opencv-how-to-draw-ractangle-center-of-image-and-crop-image-inside-rectan/46803516

Building a Pokedex in Python: OpenCV and Perspective Warping
https://www.pyimagesearch.com/2014/05/05/building-pokedex-python-opencv-perspective-warping-step-5-6/

Homography Examples using OpenCV

https://www.learnopencv.com/homography-examples-using-opencv-python-c/

Graph Plotting in Python
https://www.geeksforgeeks.org/graph-plotting-in-python-set-1/

Matplotlib - Nan Test
https://matplotlib.org/3.1.0/gallery/lines_bars_and_markers/nan_test.html#sphx-glr-gallery-lines-bars-and-markers-nan-test-py

Resources

The Youtube video of the skater

https://www.youtube.com/watch?v=2S_U1pnLE-M

Slack Variables

https://www.youtube.com/watch?v=ENEJnPTetBo&t=326s&ab_channel=MathFAQ

CVXPY - Robust Kalman filtering for vehicle tracking

https://www.cvxpy.org/examples/applications/robust_kalman.html

CVXPY - Linear program

https://www.cvxpy.org/examples/basic/linear_program.html

How can I safely create a nested directory?

https://stackoverflow.com/questions/273192/how-can-i-safely-create-a-nested-directory

Image Segmentation using OpenCV - Extracting specific Areas of an image

https://circuitdigest.com/tutorial/image-segmentation-using-opencv