Alec Lowi
Nicholas Karamyan
Isaac Forrest
Cynthia Wu

Tracing Targeted Color Pixels in Video

I. Abstract

Digital image processing in mathematics does not have any limitations to manipulating pixels in a digital photo. Think of a digital video as just a bunch of slightly adjusted images (known as video frames) that can each be processed or modified in any way that is desired, ultimately leading to video processing. In this research project, we created our own video processing program that traces certain assigned colored pixels in any given video. For our project, we used a weightlifting video to present the methods we used. Using digital image processing techniques such as image smoothing with a 2D Gaussian filter, converting from RGB to HSV, along with a variety of functions in python's OpenCV and the imutils library, we found a way to track the barbell of a weightlifting video and extract data. This data can then be used to find barbell speed, barbell angle/path, and other information about weightlifting techniques that can be useful to all weightlifting athletes around the world.

II. Introduction

Our project consists primarily of video data analysis. We want to be able to extract certain trends in video as data and analyze the trends using graphical analysis or basic grouping. We will use a weight lifting video (.mp4) for basic demonstration. When a barbell lift is executed on a digital video (of type mp4), there exists desired color pixels from each video frame that can be extracted and converted into coordinates on an xy-plane. These coordinates can be useful sets of data for athletes, law-enforcement, entertainment professionals, large corporate businesses, and many more. As it pertains to our project, the coordinate data is used to find the speed of the bar, the angle at a time (t), and/or path of the bar.

III. More Detailed Mathematical Background

Methods from class that were used in this project included mathematical image processing using image smoothing and converting from RGB color spaces to HSV color spaces. A Gaussian Blur, a form of image smoothing, was used in the video. The OpenCV function used in the code convolves the image with a Gaussian kernel of the form:

$$w(s,t) = G(s,t) = Ke^{-\frac{s^2+t^2}{2\sigma^2}}$$

When the proper parameters of the Gaussian kernel have been chosen, convolution is then completed in order to smooth the image:

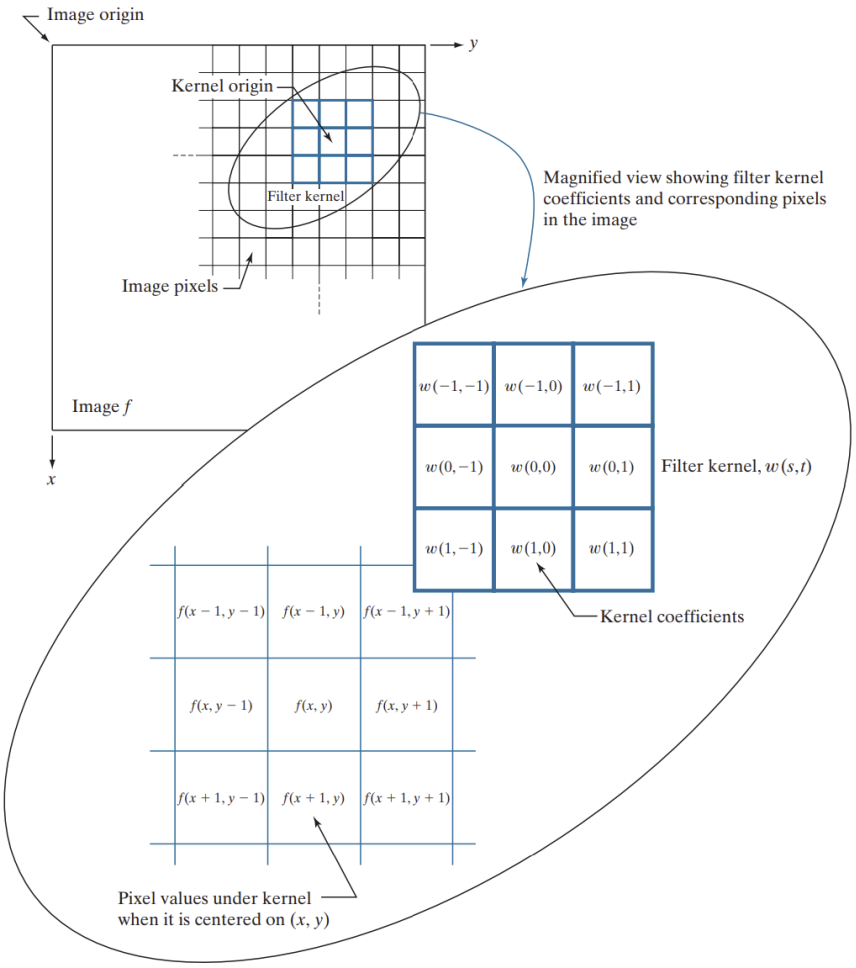

TABLE 3.6 Mean and standard deviation of the product (×) and convolution (★) of two 1-D Gaussian functions, f and g. These results generalize directly to the product and convolution of more than two 1-D Gaussian functions (see Problem 3.25).

	f	g	$f \times g$	$f \star g$
Mean	m_f	m_g	$m_{f\times g} = \dfrac{m_f\sigma_g^2 + m_g\sigma_f^2}{\sigma_f^2 + \sigma_g^2}$	$m_{f\star g} = m_f + m_g$
Standard deviation	σ_f	σ_g	$\sigma_{f\times g} = \sqrt{\dfrac{\sigma_f^2\sigma_g^2}{\sigma_f^2 + \sigma_g^2}}$	$\sigma_{f\star g} = \sqrt{\sigma_f^2 + \sigma_g^2}$

The next mathematical imaging method used in the project from class was converting the video frames from the RGB color space to the HSV color space. In class we learned how to convert from RGB to HSI, which is done using the following mathematical method:

$$H = \begin{cases} \theta & \text{if } B \le G \\ 360 - \theta & \text{if } B > G \end{cases} \tag{6-16}$$

with[†]

$$\theta = \cos^{-1}\left\{ \frac{\frac{1}{2}\left[(R-G)+(R-B)\right]}{\left[(R-G)^2+(R-B)(G-B)\right]^{1/2}} \right\} \tag{6-17}$$

The saturation component is given by

$$S = 1 - \frac{3}{(R+G+B)}\left[\min(R,G,B)\right] \tag{6-18}$$

Finally, the intensity component is obtained from the equation

$$I = \frac{1}{3}(R+G+B) \tag{6-19}$$

The OpenCV function RGB_2_HSV was used in the project. HSV is very similar to the HSI color space. The only difference is that the value component is used instead of the intensity component, where V = max(R, G, B) with the subsequent calculations being the range of 0 to 1.

IV. Methods

A weightlifting video will be used in the program to demonstrate the program's purpose. A video was recorded of a basic, heavy barbell squat with a lemon peel taped to the end of the barbell. This was to provide brighter pixels for the program to recognize and to grab onto. We also used a tripod to record the video to avoid any discrepancies in the extracted video data. The tripod allowed us to trust that the coordinates we extracted from the video were as accurate as possible. Once the video was taken and clipped into the portion we wanted, the video was captured into our program. See the following pseudo-coding summary to gain a further understanding of our program:

1. Import all necessary python libraries and packages (see acknowledgements)
2. Initialize upper and lower boundaries of the desired color in the HSV color space
    a. See the Range-detector link in acknowledgements
3. Initialize data structure *pts* using *deque()*
4. Grab video reference pointer *camera* using the built in *cv2.VideoCapture*
5. Utilize *time.sleep(2.0)* to allow the video file to warm up
6. *While* loop of type boolean (true) to continue until the video runs out of frames
    a. Call the *read()* method of the camera pointer. Also initialize *grabbed* of type boolean to indicate the frame being successfully read

b. If not grabbed
    i. Break
c. Initialize variable *frame* to a bit using *imutils.resize()* (each frame) *width=SET WIDTH*
d. Initialize filter *blurred*. Set to a *cv2.GaussianBlur()* to reduce high frequency noise. This will help focus on objects we want inside the *frame*
e. Initialize pointer *hsv* with cv2.*COLOR_BGR2HSV* which will convert frames to HSV color space
f. Create *mask*. Make call to *cv2.inRange* to handle localization of yellow. Output will be of type binary mask
g. Create 2 more *masks* of calls to *cv2.erode()* and *cv2.dilate()* to remove any discoloration or "blobs" left on the binary mask.
h. Initialize list() *cnts* using *imutils.grab_contours(cnts)* to compute the contours of the objects we are tracking on the image. Made compatible with all versions of OpenCV
i. Initialize the *center* which will be used as the (x,y)-coordinates of the object we are tracking in the frame
j. *if len(cnts) > 0:* to check that at least one contour was found on the mask.
    i. Initialize *c* to compute largest contour on the *mask*
    ii. Compute minimum enclosing circle
    iii. Compute the *center* xy-coordinates
    iv. *if radius > 10:* to check if the radius is sufficiently large
        1. Then we draw two circles using *cv2.circle(),* one surrounding the ball itself and another to indicate the center of the ball
k. Append the *center* coordinates to the *pts* list (deque())
l. Use for loop to loop over the set of tracked points in the *pts* deque()
    i. If the points tracked are of NoneType, ignore them
    ii. Make call to *cv2.line()* in the *frame* and the *pts[i-1]* to trace the red line in the video along the object we are tracking
m. Make call to *cv2.imshow()* to show the mask on the video
n. Make call to *cv2.imshow()* to show the modified frame on the video
o. Call to *cv2.waitKey()* to destroy the window when finished
p. If statement for the velocity deque() < 2
    i. Then append the points left

7. For loop through the list of appended points (displacement)
    a. Use math.sqrt((math.sqrt(veloc[i][0]*veloc[i][0]+veloc[i][1]*veloc[i][1]))
8. Initialize all the maximum and minimum speed variables
9. Initialize the set of speed variables sorted (to make the colored line easier)
10. Create variable "cmap" to gather colors for the line plt.get_cmap('cool_r', type)
11. Re-create the aforementioned while loop
    a. After the "frame" variable, loop through the pts deque() to find the proper speed at each point, and set the colorOFLine variable to the following:

b. colorofline = tuple((math.floor(255*c_rgba[2]),math.floor(255*c_rgba[1]),math.floor(255*c_rgba[0])))
c. Use cv2.imshow() to display the velocity line through every frame
d. cv2.waitKey(1) to allow the video to play and complete once all the frames have passed through with the velocity line

12. Initialize *f* to open and save the data points to .txt file
13. *f.write(str(list(pts)))* to create strings in the .txt file (these are the xy-coordinates in writing!!!)
14. *camera.release()* to close the video file
15. *cv2.destroyAllWindows()* to allow user to close the window and open a new video file with a fresh slate

Once the code was complete, the weightlifting video was opened and used in the code to closely analyze the data.

V. Results

When the weightlifting video was used in our code, we had two resulting videos. The first resulting video (figure 1) was the demonstration for simple tracking of the lemon peel. A plain red line followed the lemon peel throughout the duration of the video, tracking the path of the barbell, as seen below:

Figure 1

The second resulting video (Figure 2) had the velocity line implemented. Here it can be seen that the line has a variety of pink and blue colors. The part of the line that is blue represents where the path of the barbell was slow, with the pink areas of the line representing where the bar was moving fast, as seen below:

Figure 2

Downsizing the video frames, using image smoothing with a gaussian blur to focus on the desired yellow pixels of the lemon peel, and converting to the HSV color space altered the video output, but made the tracking line extremely accurate along the barbell. Using these methods ensured that the line would not jump to other pixels in the video and negatively impact the data we were analyzing.

VI. Conclusions

As previously mentioned, this tracking technique is extremely useful in weightlifting analysis. If you look closely at the purple line in figure 2, notice a shade of blue in the line halfway up from the squat. This tells the lifter where the sticking point of the executed lift is, which can be useful when trying to implement better technique. This method of tracking can be improved with some forms of user input. If a desired video is chosen and the user can choose which area of space in the video they want to track, then the program can be useful for many other forms of video analysis. It is possible that this program can be processed for iOS implementation and used for a smartphone app.

VII. Acknowledgements

The following libraries were used to create the program:

A. OpenCV Image Processing

1. Smoothing Images
2. Image Moments
3. Color Space Conversions
4. Eroding and Dilating
5. Contours

B. OpenCV Drawing Functions

C. Python Imutils
    1. Range-Detector
D. Python Collections - Container Datatypes
E. Python DataTime

VIII. References

- Rosebrock, Adrian. “Ball Tracking with Opencv.” *PyImageSearch*, 17 Apr. 2021 **-** Ball Tracking using OpenCV
- Gonzalez, Rafael C., and Richard E. Woods. *Digital Image Processing*. 4th ed., Pearson, 2018.

IX. Author Contributions

Alec Lowi completed the pseudo-code, project-writeup, and project proposal.

Nicholas Karamyan wrote a majority of the project code with resulting video #1 (figure 1) and got the weightlifting video to work in the tracking software.

Isaac Forrest wrote the project code for the velocity implementation with resulting video #2 (figure 2) and got the velocity line working with the weightlifting video.

Cynthia Wu created all of the project presentation slides for the group.

All group members equally contributed to the project.