Naaman Trumbull
ASN3: How does a Robot See the World?

Notes: Ignore the "Data" folder that appears next to my written code files. This folder contains the media that was provided to us for this assignment.

The folder "Outputs" contains miscellaneous outputs I created that showcase my progression through the assignment. Files such as my template, a grayscale Video, and a single frame with box detection can be found here.
My final output, "Output.mp4," can be found here.

The folder "VideoOutputs" contains six different video outputs with six different methods of template matching. You will be able to see a direct comparison between all 6 of my template matching methods here.

The folder "StepTwo" contains all my python code files.

Step 2:

Part 1: I did not write any code to create my template of the barrel. I used MSPaint to crop the cone manually. MSPaint has a feature where you can crop objects within certain shapes, and a cone happens to be one of the objects. I then converted the image to grayscale. My final template is to the right.

Part 2: For this part, I created the video by separating it into frames. I then converted each frame to grayscale using cv2.cvtColor() and then wrote it to a video writer. I then released the video with the line result.release() which saves my video as an mp4 file. Here is the code:

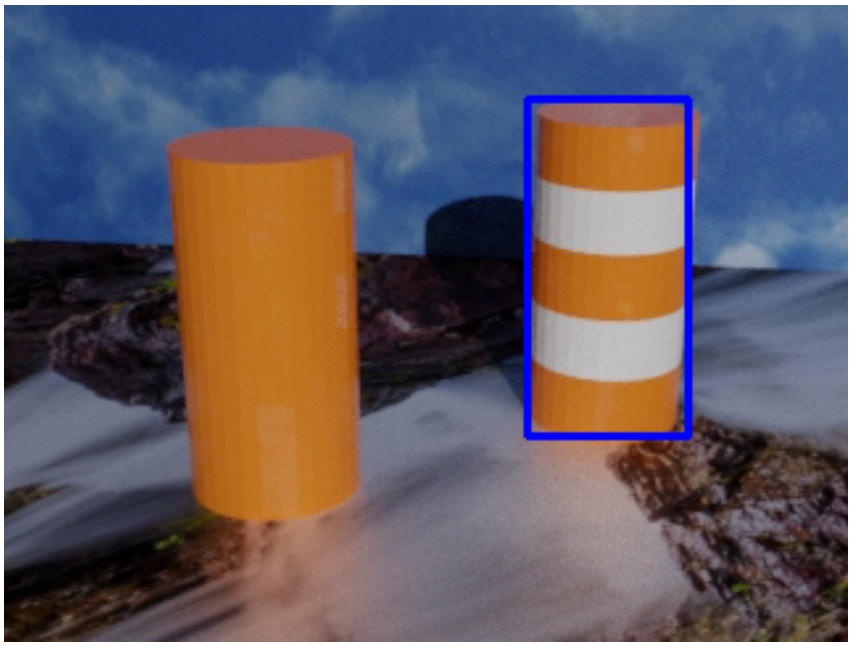
```
1   import numpy as np
2   import cv2
3
4   cap = cv2.VideoCapture('ASN4_trumbull/Data/Vid.mp4')
5
6   width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
7   height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
8
9   result = cv2.VideoWriter("ASN4_trumbull/Outputs/VidGrayScale.mp4",
10                           cv2.VideoWriter_fourcc(*"MPG4"),
11                           30.0, (width, height))
12
13  while(cap.isOpened()):
14      ret, frame = cap.read()
15
16      grayScale = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
17      result.write(grayScale)
18
19      if ret == True:
20          if cv2.waitKey(1) & 0xFF == ord('q'):
21              break
22      else:
23          break
24
25  # Release everything if job is finished
26  cap.release()
27  result.release()
28  cv2.destroyAllWindows()
```

Part 3: In order to create a barrel detector, I started by detecting the barrel with one single frame. I used code from OpenCV's Template Matching tutorial (source below) to implement this. I modified the code slightly to reflect my data.

Input:

```python
import cv2 as cv
import numpy as np

# Creating a barrel detector for one single image. See part 4 for the full video

img_rgb = cv.imread("ASN4_trumbull/Outputs/FrameOneUncropped.png", 1)
img_gray = cv.imread("ASN4_trumbull/Outputs/FrameOneUncropped.png", 0)
template = cv.imread("ASN4_trumbull/Outputs/CroppedCone.png", 0)
w, h = template.shape[::-1]

# All the 6 methods for comparison in a list
methods = ['cv.TM_CCOEFF', 'cv.TM_CCOEFF_NORMED', 'cv.TM_CCORR',
            'cv.TM_CCORR_NORMED', 'cv.TM_SQDIFF', 'cv.TM_SQDIFF_NORMED']

img = img_gray.copy()
method = eval(methods[1])
# Apply template Matching
res = cv.matchTemplate(img, template, method)
min_val, max_val, min_loc, max_loc = cv.minMaxLoc(res)
# If the method is TM_SQDIFF or TM_SQDIFF_NORMED, take minimum
if method in [cv.TM_SQDIFF, cv.TM_SQDIFF_NORMED]:
    top_left = min_loc
else:
    top_left = max_loc
bottom_right = (top_left[0] + w, top_left[1] + h)
cv.rectangle(img_rgb, top_left, bottom_right, 255, 2)

cv.imwrite("ASN4_trumbull/Outputs/InitialFrame.png", img_rgb)
```

Output Frame:



Part 4: Using my code from part 2 and 3, as well as my template from part 1, I was able to create a barrel detector for an entire video. This code creates and writes my final output video titled “Output.mp4” which can be found in the Outputs folder. The code iterates through the color video frame by frame, converting each frame to grayscale. It uses the grayscale frame to run the cv.matchTemplate() method but applies the rectangle results to the color image, giving us a full color output.

Input:

```python
import cv2 as cv
import numpy as np

# Creating a barrel detector for one single image. See part 4 for the full video

cap_rgb = cv.VideoCapture('ASN4_trumbull/Data/Vid.mp4')

template = cv.imread("ASN4_trumbull/Outputs/CroppedCone.png", 0)
methods = ['cv.TM_CCOEFF', 'cv.TM_CCOEFF_NORMED', 'cv.TM_CCORR',
           'cv.TM_CCORR_NORMED', 'cv.TM_SQDIFF', 'cv.TM_SQDIFF_NORMED']
w, h = template.shape[::-1]

width = int(cap_rgb.get(cv.CAP_PROP_FRAME_WIDTH))
height = int(cap_rgb.get(cv.CAP_PROP_FRAME_HEIGHT))

result = cv.VideoWriter("ASN4_trumbull/Outputs/Output.mp4",
                        cv.VideoWriter_fourcc(*"MPG4"),
                        30.0, (width, height))

while(cap_rgb.isOpened()):
    ret, frame = cap_rgb.read()

    frame_gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)

    method = eval(methods[5])

    res = cv.matchTemplate(frame_gray, template, method)
    min_val, max_val, min_loc, max_loc = cv.minMaxLoc(res)

    if method in [cv.TM_SQDIFF, cv.TM_SQDIFF_NORMED]:
        top_left = min_loc
    else:
        top_left = max_loc
    bottom_right = (top_left[0] + w, top_left[1] + h)
    cv.rectangle(frame, top_left, bottom_right, 255, 2)

    if ret == True:
        if cv.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        break

    result.write(frame)

cap_rgb.release()
result.release()
cv.destroyAllWindows()
```

Part 5: This part uses the same code from part 4. I tested the 6 different methods of template matching by changing the parameter in line 25 to a different index in the methods array. I placed all six of these videos in the "VideoOutputsComparison" folder so that I could watch all of them back to back. The three with the best results are "TM_CCOEFF," "TM_CCORR," and "TM_SQDIFF_NORMED."

The reason these three methods worked the best for me can be explained by OpenCV's Template Matching tutorial. When you look at the "Matching result" photos (which my code stores under the variable name "res") you can see that these methods have distinct white (or black) points where the template matches. This makes it easy to detect where the coordinates of the rectangles should be placed and this is why these three perform better on my data. Some of the methods that do not perform as well have points that are less distinct, making it harder to place the rectangle.

Part 6: When using color in a template matching method, you need to start by considering how color can appear different based on a variety of factors. Colors can appear different due to different amounts of daylight for example. So, you first have to implement some sort of color thresholder that accounts for a range of closely related colors when scanning an image with your template. Another thing to consider with color images is that you do not have to rely on just the color of the template to detect an object. You can also consider other factors of the template, such as the shape, to detect an object. Combine the shape with edge detection and this is how one could begin to implement this type of template matching.

Source:

OpenCV's Template Matching Tutorial:
https://docs.opencv.org/3.4/d4/dc6/tutorial_py_template_matching.html

- Used the code provided on this page to implement my barrel detector (Parts 3 and 4).

Challenges: The most challenging part of this assignment was keeping track of several files. I learned how to keep track of files by organizing them into folders and giving them identifying names.

Feedback: Having the lab time to work on this assignment was extremely helpful. Being in the lab really forced me to sit down and work on my code. Unfortunately, I was unable to meet with my PRM during the week before spring break so I did not get much help from them as I have in the past. I liked this assignment. It was challenging but in a fair way.