Yoseph Ayele Kebede

Dr. Mitchell

ENPM701 Assignment 3

Object Tracking Algorithm

Assignment #3

Tracking Green Color from a traffic light image and analyzing raspberry pi camera performance over time

Question #1

 # Steps 1 – 4: Traffic Green Light Tracking

```
import tkinter
import cv2
import os
import imutils
import numpy as np
import matplotlib.pyplot as plot
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
from datetime import datetime, timedelta


def img_show(name, img):
        cv2.imshow(name,img)
        cv2.waitKey(0)
        cv2.destroyAllWindows()


def mask_color(image, imageHSV):
        # HSV bounds
        minHSV = np.array([50,100, 100])
        maxHSV = np.array([70, 255, 255])
```

```
    # mask HSV
    maskHSV = cv2.inRange(imageHSV, minHSV, maxHSV)
    return maskHSV
def main():
    # Purpose: Continuosly locate and identify green light from video feed
    # Pull in code from Assignment 2 and adjust
    # initialize the Raspberry Pi camera
    camera = PiCamera()
    camera.resolution = (640, 480)
    camera.framerate = 25
    rawCapture = PiRGBArray(camera, size=(640,480))

    # Pull in code from steps 1 and 2
    # allow the camera to warmup
    time.sleep(0.1)

    start = time.time()

    # create object to read camera
    video = cv2.VideoCapture(0)

    if (video.isOpened() == False):
        print("Error reading video")

    # define the codec and create VideoWriter object
    fourcc = cv2.VideoWriter_fourcc(*'XVID')
    out = cv2.VideoWriter('videonameNew.avi', fourcc, 3, (640, 480))
```

```
	frm_cnt = 0
	duration = 0

	# Open .txt file to save data
	f = open('hw3data_4.txt','a')

	# Inirtialize circle variables
	center = 0
	radius = 0.0
	x = 0.0
	y = 0.0

	# keep looping over video
	for frame in camera.capture_continuous(rawCapture, format="bgr",
use_video_port=False):
		# Record iteration start time
		startR = datetime.now() #.microsecond / 1000000
		# grab the current frame
		image = frame.array

		# Convert image from BGR to HSV space
		imageHSV = cv2.cvtColor(image,cv2.COLOR_BGR2HSV)

		# mask the green light from HSV and convert to grayscale
		mask = mask_color(image, imageHSV)

		# Apply contour function to find edges
		img, contours, hierarchy =
cv2.findContours(mask,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
```

```
# Contour detection
if (contours is not None) and (len(contours) >= 1) :
        # Take the first contour
        cnt = contours[0]

        # compute the moments of contours
        momnt = cv2.moments(cnt)
        print(momnt)

        # min Enclosing circle
        (x,y),radius = cv2.minEnclosingCircle(cnt)
        # Save circle radius and center as int
        center = int(x),int(y)
        radius = int(radius)

        # Draw circle on top of original image
        cv2.circle(image, center, radius, (0,255,255),2)
        cv2.circle(image, center,0,(0,0,255),5)
else:
        print("Countour not found")

# show the frame to our screen
cv2.imshow("Frame", image)
key = cv2.waitKey(1) & 0xFF
# write frame into file
out.write(image)
# increase frame counter
frm_cnt += 1
```

```
            # clear the stream in preparation for the next frame
            rawCapture.truncate(0)
            duration = time.time() - start

            stopR = datetime.now()
            now = stopR - startR
            outstring = str(now.total_seconds()) + '\n'
            f.write(outstring)
            print(now)

            # press the 'q' key to stop the video stream
            if (key == ord("q") or (duration >= 40)) and (frm_cnt > 110):
                  break

      # Release video capture and video object
      video.release()
      out.release()

      # Close all windows
      cv2.destroyAllWindows()

if __name__ == "__main__":
      main()
```

Video Demonstration: https://youtu.be/6PzuW9L8anA

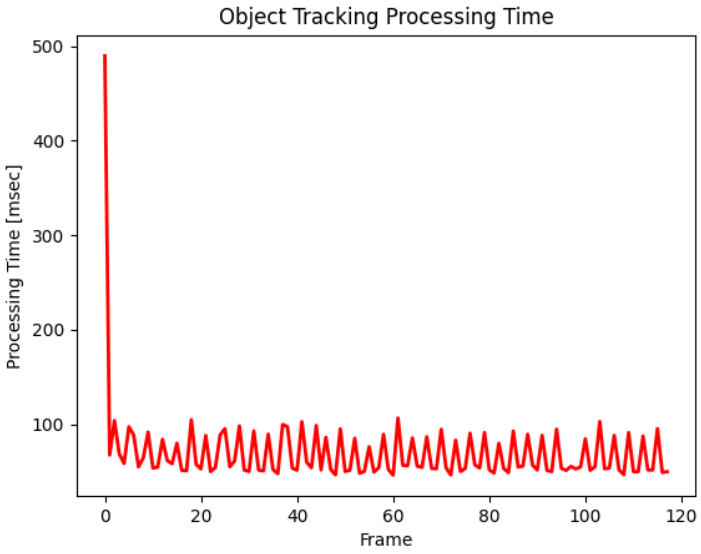
Step 5 – Object Tracking Performance Time

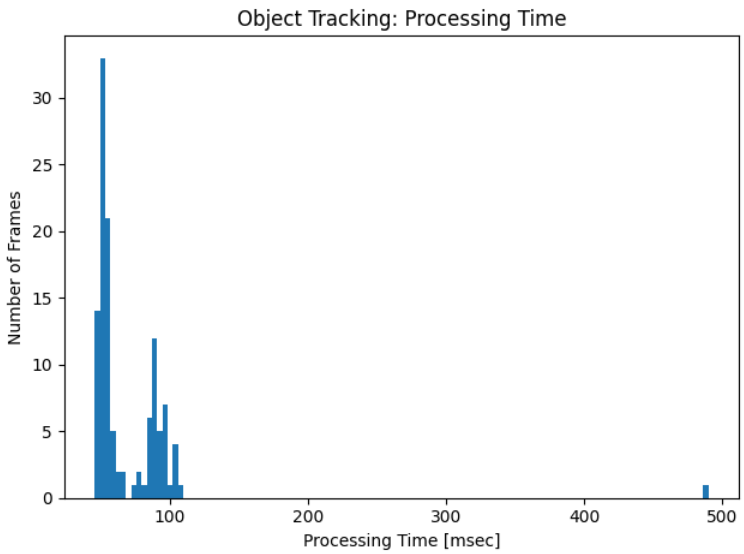
```
import numpy as np
import matplotlib.pyplot as plt
```

```
# 1) Load imu data as string
picamdata = np.genfromtxt("hw3data_4.txt", dtype=str)

# Obtain shape of data/array
size = picamdata.shape

# Extract raw data and convert to milliseconds for better resolution
perf = picamdata[:].astype(np.float64) * 1000

# Create horizontal x axis for plot
x = np.linspace(0,size[0],num=size[0],endpoint=False,dtype=int)

# Plot performance time in msec
fig, ax1 = plt.subplots()
ax1.plot(x,perf,ls='solid', color='red',linewidth=2, label='picam-img-det-raw-data')

# 3) Label the axes, title, legend
ax1.set(title="Object Tracking Processing Time",
        ylabel="Processing Time [msec]",
        xlabel="Frame")
plt.show()
fig.savefig('obj_trck_prcs_time.png')
plt.close()

# Plot histogram of data
num_bins = size[0]
```

```
fig, ax = plt.subplots()

# The histogram of the data
ax.hist(perf,bins=num_bins)
ax.set_xlabel('Processing Time [msec]')
ax.set_ylabel('Number of Frames')
ax.set_title('Object Tracking: Processing Time')

# Tweak spacing to prevent clipping of ylabel
fig.tight_layout()
plt.show()
fig.savefig('hist-obj_trck_prcs_time.png')
plt.close()
```

Plots from Performance Time analysis





PiCamera Performance Discussion

It is apparent from the plots generated that the piCamera performance is heavily affected at the start of the program execution which is evident by the amount of time it takes to loop through a single iteration in order to perform the required task (in this case detecting edges of a masked image of green light). However, as the video frame iteration continues its performance improves gradually and, thus, begins to perform decently.

Therefore, one can learn from this exercise that he or she should disregard the results from the first few seconds, due to abundance of caution in ensuring a stable result from his or her algorithm being executed. As a result, “warming up” the PiCamera for few seconds seems to be the optimal decision to undertake when using this device along with the Raspberry Pi.