Yoseph Ayele Kebede

UID: 114196729

Dr. Mitchell

ENPM701 Assignment 4

Robotic Control Command Identification

**Assignment #4**

Reading Direction from Green Arrow in an image/sign and analyzing raspberry pi camera performance over time

**Question #2**

**# SODAR: Reading distance from Ultrasonic sensor**

import numpy as np

import cv2

import imutils

import RPi.GPIO as gpio

import time

import os

# Define pin allocations

trig = 16

echo = 18

def distance():

    gpio.setmode(gpio.BOARD)

    gpio.setup(trig, gpio.OUT)

    gpio.setup(echo, gpio.IN)

    # Ensure output has no value

    gpio.output(trig, False)

    time.sleep(0.01)

    # Generate trigger pulse

    gpio.output(trig, True)

    time.sleep(0.00001)

    gpio.output(trig, False)

    # Generate echo time signal

    while gpio.input(echo) == 0:

        pulse\_start = time.time()

    while gpio.input(echo) == 1:

        pulse\_end = time.time()

    pulse\_duration = pulse\_end - pulse\_start

    # Convert time to distance

    distance = pulse\_duration \* 17150

    distance = round(distance, 2)

    # Cleanup gpio pins & return distance estimate

    gpio.cleanup()

    return distance

if \_\_name\_\_ == "\_\_main\_\_":

    # Record image using Raspistill

    name = "lecture4inclass.jpg"

    os.system('raspistill -w 640 -h 480 -o /home/pi/ENPM701-class-files/' + name)

    print("Hi!")

    print("Ultrasonic sensor will print 10 successive range estimates per 1Hz")

    img = cv2.imread('lecture4inclass.jpg')

    img = imutils.resize(img, width=400)

    # show image

#     cv2.imshow("Ultrasonic on image", img)

#     cv2.waitKey(0)

    # Start printing range values

    start = time.time()

    idx = 1

    measurements = []

    while idx <= 10:

        if (round((time.time() - start), 2) == 1):

            dist = distance()

            print ("Counter: ", idx, "| Distance: ", dist, " cm")

            idx += 1

            start = time.time()

            measurements.append(dist)

    # Take the average of the measurements

    ave = round(np.average(measurements), 2)

    # Choose font style and color

    font = cv2.FONT\_HERSHEY\_COMPLEX\_SMALL

    red = (0, 0, 255)

    # Append measurement on image

    cv2.putText(img, str(ave)+" cm", (100, 200), font, 1, red, 2)

    # show image

    cv2.imshow("Ultrasonic on image", img)

    # Save appended image

    cv2.imwrite('lecture4inclass\_US\_txt.jpg', img)

    print("Done")

    cv2.waitKey(0)

    cv2.destroyAllWindows()

**Output:**



**Fig-1:** 0.5m water bottle distance measured using Pi Camera

**Question #3**

**# Detect arrow from picture and read its direction**

**First Part: HSV Masking and Direction Detection**

import cv2

import os

import imutils

import numpy as np

import matplotlib.pyplot as plt

def img\_masking(gnArrow):

    # Save the shape of the image array

    (height, width, c)  = gnArrow.shape

    # Convert image from BGR to HSV space

    gnArrowHSV = cv2.cvtColor(gnArrow, cv2.COLOR\_BGR2HSV)

    # Display HSV converted image

    cv2.imwrite("grnArrowHSV.png", gnArrowHSV)

    # HSV bounds

    minHSV = np.array([49, 103, 61])

    maxHSV = np.array([93, 255, 248])

    # Create a function that will search through every

    # pixel and mask

    # Need cv2.inRange() or custom function for HSV masking

    maskHSV = cv2.inRange(gnArrowHSV, minHSV, maxHSV)

    cv2.imwrite("gnArrow\_MaskOnlyHSV.png",maskHSV)

    # Read all three differnet pictures for ease of display

    grn\_HSV = cv2.imread("grnArrowHSV.png")

    grn\_Mask = cv2.imread("gnArrow\_MaskOnlyHSV.png")

    # Now stack images horizontally for ease of display

    grn\_all = np.hstack([gnArrow,grn\_HSV,grn\_Mask])

    return maskHSV, grn\_all

def blur\_img(maskHSV):

    blurred = cv2.GaussianBlur(maskHSV,(11,11), 0)

    return blurred

def corner\_detect(img,orig\_img):

    # Detect corners from image

    corners = cv2.goodFeaturesToTrack(img,5,0.01,10)

    corners = np.int0(corners)

    # Create a list to store the x,y location of points

    pts\_loc = []

    # identify location of corners in image

    for i in corners:

        # Extract x,y coordinate of points

        x,y = i.ravel()

        # Draw circle of corners on image

        cv2.circle(img,(x,y),3,255,-1)

        cv2.circle(orig\_img,(x,y),3,(255,0,0),-1)

        # Store image coordinate of corners in a list

        pts\_loc.append([x,y])

    # Create a column vector from pts list

    pts\_loc = np.array(pts\_loc)

    return img, pts\_loc, orig\_img

def def\_det(pt\_list):

    # Extract x,y points from pt\_list

    x = pt\_list[:,0]

    y = pt\_list[:,1]

    # Determine the min and max width & height values

    # of the points, as if to drow rectangle around arrow

    x\_min = x.min()

    y\_min = y.min()

    x\_max = x.max()

    y\_max = y.max()

    # Store height of bounding box

    vert\_dst = y\_max - y\_min

    # Store width of bounding box

    horz\_dst = x\_max - x\_min

    # Compute and store half dimensions of

    # box, will come later when determining

    # arrow direction

    y\_half = vert\_dst/2 + y\_min

    x\_half = horz\_dst/2 + x\_min

    # Initializing the direction of the arrow

    direction = ""

    # Initialize counter to track the number of

    # points below the half vertical distance.

    count = 0

    # When the arrow vertical (up/down) or (left/right)

    if vert\_dst > horz\_dst:

        # Since the head has 3 corners identified,

        # if the arrow is pointing up (noting origin of image

        # is at the left top corner), there will be three

        # points below the middle of the arrow vertically

        # loop through all vertical points of corner ordered

        # points

        for i in y:

            if (i < y\_half):

                count+=1

        # Determine direction

        if count >= 3:

            direction = "Up"

        else:

            direction = "Down"

    else:

        # Similar to above if logic

        # Since the head has 3 corners identified,

        # if the arrow is pointing left (noting origin of image

        # is at the left top corner), there will be three

        # points to the left of the middle of the arrow horizontally

        # loop through all vertical points of corner ordered

        # points

        for i in x:

            if (i < x\_half):

                count+=1

        # Determine direction

        if count >= 3:

            direction = "Left"

        else:

            direction = "Right"

    return direction

def img\_dir\_txt(img,img\_crn,direction):

    (cx,cy) = (img.shape[1]/4.5, img.shape[0]/8)

    # Drawing white background

    cv2.rectangle(img,(0,0),(int(cx),int(cy)),(255,255,255),-1)

    # Placing text over white background

    font = cv2.FONT\_HERSHEY\_COMPLEX\_SMALL

    green = (0,255,0)

    cv2.putText(img,direction,(0,int(cy/2)),font,2,green,2)

    return img

def main():

    print("Arrow detection program started!")

    # Read an image from library

    gnArrow = cv2.imread("greenArrow01.png")

    # Apply HSV masking to image read

    img\_msk, grn\_all = img\_masking(gnArrow)

    # Apply Gaussian bluring on image

    img\_blurred = blur\_img(img\_msk)

    cv2.imwrite("arw\_blur.png",img\_blurred)

    # Detect corners from image

    img\_crnr, pts\_loc, org\_img = corner\_detect(img\_blurred,gnArrow)

    cv2.imwrite("arw\_crnr\_det.png",img\_crnr)

    # Identify direction of arrow

    direction = def\_det(pts\_loc)

    print("The arrow is pointing " + direction)

    # Display original image, corner, and text together

    fin\_img = img\_dir\_txt(org\_img,img\_crnr,direction)

    # Read masked images for display purposes

    blur = cv2.imread("arw\_blur.png")

    img\_crnr = cv2.imread("arw\_crnr\_det.png")

    # Stack the arrow detection steps

    arr\_det = np.hstack([blur,img\_crnr,fin\_img])

    # Stack the whole process from original to final

    fin\_disp = np.vstack([grn\_all,arr\_det])

    # Display full output of process

    cv2.imshow("Green Arrow Masked",fin\_disp)

    cv2.imwrite("gnArrow\_det\_full.png",fin\_disp)

    cv2.waitKey(0)

    cv2.destroyAllWindows()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

A screenshot of a video game

Description automatically generated**Fig-2:** Original image (top left), Converted to HSV scale (top middle), HSV Masking (top right), Application of Gaussian Blur (bottom left), Shi-Tomasi Corner Detection (bottom middle), Arrow Direction Detection Full (bottom right)

**Second Part: Application of Arrow Detection to Live Video Stream**

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

def img\_mask\_blur(img):

    # Save the shape of the image array

    (height, width, c)  = img.shape

    # Convert image from BGR to HSV space

    imgHSV = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

    # HSV bounds

    minHSV = np.array([49, 103, 61])

    maxHSV = np.array([93, 255, 248])

    # Create a function that will search through every

    # pixel and mask

    # Need cv2.inRange() or custom function for HSV masking

    maskHSV = cv2.inRange(imgHSV, minHSV, maxHSV)

    # Apply Gaussian blur

    blurred = cv2.GaussianBlur(maskHSV,(11,11), 0)

    return blurred

def corner\_detect(img,orig\_img):

    # Detect corners from image

    corners = cv2.goodFeaturesToTrack(img,5,0.01,10)

    # Check if there are corners

    if corners is None:

        return orig\_img, []

    else:

        corners = np.int0(corners)

    # Create a list to store the x,y location of points

    pts\_loc = []

    # identify location of corners in image

    for i in corners:

        # Extract x,y coordinate of points

        x,y = i.ravel()

        # Draw circle of corners on image

        cv2.circle(orig\_img,(x,y),3,(255,0,0),-1)

        # Store image coordinate of corners in a list

        pts\_loc.append([x,y])

    # Create a column vector from pts list

    pts\_loc = np.array(pts\_loc)

    return orig\_img, pts\_loc

def def\_det(img,pt\_list):

    # Extract x,y points from pt\_list

    x = pt\_list[:,0]

    y = pt\_list[:,1]

    # Determine the min and max width & height values

    # of the points, as if to drow rectangle around arrow

    x\_min = x.min()

    y\_min = y.min()

    x\_max = x.max()

    y\_max = y.max()

    # Store height of bounding box

    vert\_dst = y\_max - y\_min

    # Store width of bounding box

    horz\_dst = x\_max - x\_min

    # Compute and store half dimensions of

    # box, will come later when determining

    # arrow direction

    y\_half = vert\_dst/2 + y\_min

    x\_half = horz\_dst/2 + x\_min

    # Find the percentage difference between vertical

    # and horizonal max separtaion distances of points

    ver\_pnt = 100 \* (vert\_dst/(vert\_dst+horz\_dst))

    hor\_pnt = 100 \* (horz\_dst/(vert\_dst+horz\_dst))

    # Initializing the direction of the arrow

    direction = ""

    # Initialize color to display direction on

    # image frame

    color = (0,0,0)

    # Initialize counter to track the number of

    # points below the half vertical distance.

    count = 0

    # When the arrow vertical (up/down) or (left/right)

    if (vert\_dst > horz\_dst) and (ver\_pnt > 60):

        # Since the head has 3 corners identified,

        # if the arrow is pointing up (noting origin of image

        # is at the left top corner), there will be three

        # points below the middle of the arrow vertically

        # loop through all vertical points of corner ordered

        # points

        for i in y:

            if (i < y\_half):

                count+=1

        # Determine direction

        if count > 2:

            direction = "Up"

            # default color

        else:

            direction = "Down"

            color = (0,0,255)

    elif (horz\_dst > vert\_dst) and (hor\_pnt > 60):

        # Similar to above if logic

        # Since the head has 3 corners identified,

# if the arrow is pointing left (noting origin of image

        # is at the left top corner), there will be three

        # points to the left of the middle of the arrow horizontally

        # loop through all vertical points of corner ordered

        # points

        for i in x:

            if (i < x\_half):

                count+=1

        # Determine direction

        if count > 2:

            direction = "Left"

            color = (255,0,0)

        else:

            direction = "Right"

            color = (0,255,0)

    else:

        direction = "---"

        color = (10,10,10)

    return direction, color

def img\_dir\_txt(img,direction,color):

    # Place direction text on image

    (cx,cy) = (img.shape[1]/4.5, img.shape[0]/8)

    # Drawing white background

    cv2.rectangle(img,(0,0),(int(cx),int(cy)),(255,255,255),-1)

    # Placing text over white background

    font = cv2.FONT\_HERSHEY\_COMPLEX\_SMALL

    # Print direction of arrow on image

    cv2.putText(img,direction,(0,int(cy/2)),font,2,color,2)

    return img

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)

    # Initialize time count for iteration

    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))

    # Initialize performance collection variables

    # (total number of frames, duration of each iteration )

    frm\_cnt = 0

    duration = 0

    # Open .txt file to save data

    f = open('hw4data.txt','a')

    # keep looping

    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

        # Flig image vertically and horizontally

        # to accomodate for pi camera mount

        image = cv2.flip(image, -1)

        # Apply HSV masking and Gaussian Blur to image read

        img\_blurred = img\_mask\_blur(image)

        # Detect corners from image

        image, pts\_loc = corner\_detect(img\_blurred,image)

        # Go to the next iteration if on arrow detected

        if len(pts\_loc) == 0:

            # show the frame to our screen

            cv2.imshow("No Arrow Detection", image)

        else:

            # Identify direction of arrow

            direction, color = def\_det(image,pts\_loc)

            # Image of direction detected

            image = img\_dir\_txt(image,direction,color)

            # show the frame to our screen

            cv2.imshow("Direction Arrow Detection", image)

        key = cv2.waitKey(1) & 0xFF

        # write frame into file

        out.write(image)

        frm\_cnt += 1

        # clear the stream in preparation for the next frame

        rawCapture.truncate(0)

        duration = time.time() - start

        # End time count for iteration

        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/bDCpjN02aJo>

**Third Phase – Arrow Detection Performance Time**

import numpy as np

import matplotlib.pyplot as plt

# 1) Load imu data as string

picamdata = np.genfromtxt("hw4data.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 perf angle with respect to horizontal step 0 - imusize

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

# n, bins, patches = ax.hist(perf,num\_bins, density=True)

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()

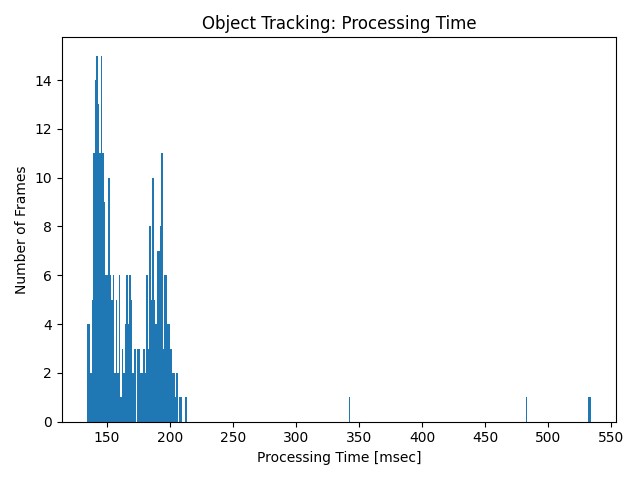
fig.savefig('hist-obj\_trck\_prcs\_time.png')

plt.close()

**Plots from Performance Time analysis**

A graph showing a red line

Description automatically generated



**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.