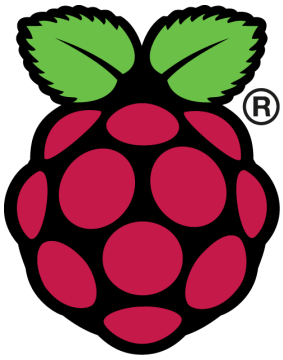
RIG PROJECT REPORT

Motion Direction Detection by Image Processing on Raspberry Pi



Harikrishna V(B180153CE)

INDEX

ABSTRACT

Computer vision is used in various fields to monitor the surroundings in real-time. It has applications in navigation of vehicles and autonomous robots, visual surveillance, medical image analysis,missile guidance, computer-human interaction, etc.
Here, we use Open Source Computer Vision Library(OpenCV) to detect the direction of motion of a moving object.

COMPONENTS USED

1. Raspberry Pi 3 A+
2. Webcam

WORKING

An image is stored as a Numpy array in OpenCV. So, by manipulating the array, we can measure the RGB values of the image. Applying various theories in Matrix theory, we can blur, smoothen, or detect the edges.

PROCEDURE

vs = cv2.VideoCapture(args["video"])

The video file supplied using the terminal command is used.

vs = VideoStream(src=0).start()

If there is no video file is supplied using terminal command, webcam is started.

frame = vs.read()

Current frame is grabbed.

frame = imutils.resize(frame, width=600)

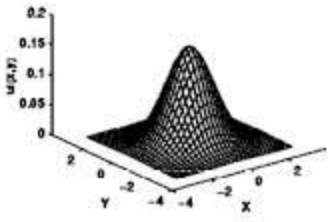
Current frame is resized to a width of 600 pixels

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

The resized image is blurred using a Gaussian function to reduce noise and details

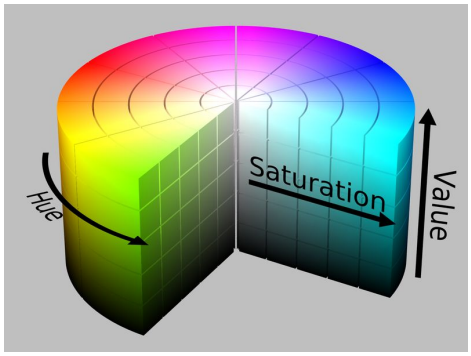
$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{x^2+y^2}{2\sigma^2}}$$

A graphical representation of the 2D Gaussian distribution with mean(0,0) and σ = 1 is shown to the right.



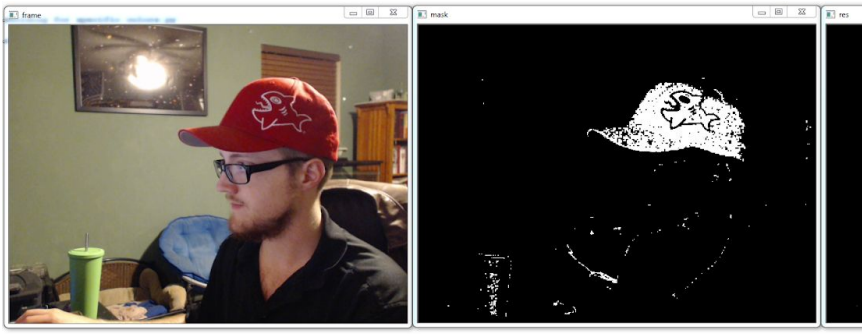
hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)

The blurred image is converted to an image in HSV colorspace



mask = cv2.inRange(hsv, colorLower, colorUpper)

Create a mask with the HSV values within the prespecified value range



mask = cv2.erode(mask, None, iterations=2)

 The foreground borders are eroded to obtain 

mask = cv2.dilate(mask, None, iterations=2)

 The foreground borders are dilated to obtain 

The process of erosion followed by dilation is known as opening.It helps in reducing noise.

cnts=cv2.findContours(mask.copy(),cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having the same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.Here, we retrieve a list of extreme outer contours and approximate it to just a few points to save memory.

c = max(cnts, key=cv2.contourArea)

The biggest contour, based on its area, is selected.

((x, y), radius) = cv2.minEnclosingCircle(c)

A parameters of the smallest circle that encloses this specific contour, is obtained

M = cv2.moments(c)

Image moments helps to calculate some features like center of mass of the object, area of the object, etc. Here, we find moments to find the center of the minimum enclosing circle.

center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))

$$C_x = \frac{M_{10}}{M_{00}} \quad C_y = \frac{M_{01}}{M_{00}}$$

cv2.circle(frame, (int(x), int(y)), int(radius),(0, 255, 255), 2)

The minimum enclosing circle is drawn to the original frame.

cv2.circle(frame, center, 5, (0, 0, 255), -1)

A red dot is drawn at the centre of the minimum enclosing circle.

pts.appendleft(center)

The center of minimum enclosing circle is added to the deque(double ended queue) of the centers of circle

dX = pts[-10][0] - pts[i][0]
dY = pts[-10][1] - pts[i][1]

The change in the coordinates of the centre of circle is calculated. The sign of these delta values is used to find the direction of motion.

SOURCE CODE

```
# USAGE
# python object_movement.py --video object_tracking_example.mp4
# python object_movement.py

# import the necessary packages
from collections import deque
from imutils.video import VideoStream
import numpy as np
import argparse
import cv2
import imutils
import time

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video", help="path to the (optional) video file")
ap.add_argument("-b", "--buffer",type=int, default=32, help="max buffer size")
args = vars(ap.parse_args())

# define the lower and upper boundaries of the ball in the HSV color space
# The values of the object is obtained using the range-detector.py script in the imutils library
colorLower = (23,109,174)
colorUpper = (39,189,255)

# initialize the list of tracked points as a deque object,so that we can easily append or pop at both the ends
pts = deque(maxlen=args["buffer"])

# the frame counter counts the number of frames
counter = 0

# the coordinate deltas to detect the direction of movement
(dX, dY) = (0, 0)

# string to store the direction of movement
direction = ""

# if a video path was not supplied, grab the reference # to the webcam using VideoStream class of 'imutils' library
if not args.get("video", False):
        vs = VideoStream(src=0,usePiCamera=False).start()

# otherwise, grab a reference to the video file using VideoCapture class of OpenCV
else:
        vs = cv2.VideoCapture(args["video"])

# allow the camera or video file to warm up
time.sleep(2.0)
```

```
while True:
        # grab the current frame
        frame = vs.read()
        # handle the frame from VideoCapture or VideoStream
        frame = frame[1] if args.get("video", False) else frame

        # if we are viewing a video and we did not grab a frame,  then we have reached the end of the video
        if frame is None:
                break
        # resize the frame, blur it, and convert it to the HSV color space
        frame = imutils.resize(frame, width=600)
        blurred = cv2.GaussianBlur(frame, (11, 11), 0)
        hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)

        # construct a mask for the color of the ball, then perform
        # a series of dilations and erosions to remove any small
        # blobs or noises left in the mask
        mask = cv2.inRange(hsv, colorLower, colorUpper)
        mask = cv2.erode(mask, None, iterations=2)
        mask = cv2.dilate(mask, None, iterations=2)

        # find the extreme outer contours in the mask and approximate it simply
        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cnts = imutils.grab_contours(cnts)

        #initialize the current center (x, y) of the ball
        center = None

        # only proceed if at least one contour was found
        if len(cnts) > 0:
                # find the largest contour in the mask using the contour areas, then use
                # it to compute the minimum enclosing circle and centroid
                c = max(cnts, key=cv2.contourArea)
                ((x, y), radius) = cv2.minEnclosingCircle(c)
                M = cv2.moments(c)
                center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))

                # only proceed if the radius meets a minimum size
                if radius > 10:
                        # draw the circle and centroid on the frame,
                        # then update the list of tracked points
                        cv2.circle(frame, (int(x), int(y)), int(radius), (0, 255, 255), 2)
                        cv2.circle(frame, center, 5, (0, 0, 255), -1)
                        pts.appendleft(center)



        # loop over the set of tracked points
        for i in np.arange(1, len(pts)):
                # if either of the tracked points are None, ignore  them
```

```
                if pts[i - 1] is None or pts[i] is None:
                        continue

                # check to see if enough points have been accumulated in the buffer
                if counter >= 10 and i == 1 and len(pts) == args["buffer"]:
                        # compute the difference between the x and y
                        # coordinates and re-initialize the direction text variables
                        dX = pts[-10][0] - pts[i][0]
                        dY = pts[-10][1] - pts[i][1]
                        (dirX, dirY) = ("", "")

                        # ensure there is significant movement in the # x-direction
                         if np.abs(dX) > 20:
                                dirX = "East" if np.sign(dX) == 1 else "West"
                        # ensure there is significant movement in the # y-direction
                        if np.abs(dY) > 20:
                                dirY = "North" if np.sign(dY) == 1 else "South"

                        # handle when both directions are non-empty
                        if dirX != "" and dirY != "":
                                direction = "{}-{}".format(dirY, dirX)

                        # otherwise, only one direction is non-empty
                        else:
                                direction = dirX if dirX != "" else dirY

                # otherwise, compute the thickness of the line and draw the connecting lines
                #thickness = int(np.sqrt(args["buffer"] / float(i + 1)) * 2.5)
                #cv2.line(frame, pts[i - 1], pts[i], (0, 0, 255), thickness)

        # show the movement deltas and the direction of movement on # the frame
        cv2.putText(frame, direction, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 0, 255), 3)
        cv2.putText(frame, "dx: {}, dy: {}".format(dX, dY), (10, frame.shape[0] - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.35, (0, 0, 255), 1)

        # show the frame to our screen and increment the frame counter
        cv2.imshow("Frame", frame) key = cv2.waitKey(1) & 0xFF counter += 1
        # if the 'q' key is pressed, stop the loop
        if key == ord("q"):
                break

# if we are not using a video file, stop the camera video stream
if not args.get("video", False):
        vs.stop()

# otherwise, release the camera
else:
        vs.release()

# close all windows
cv2.destroyAllWindows()
```

PROBLEMS FACED

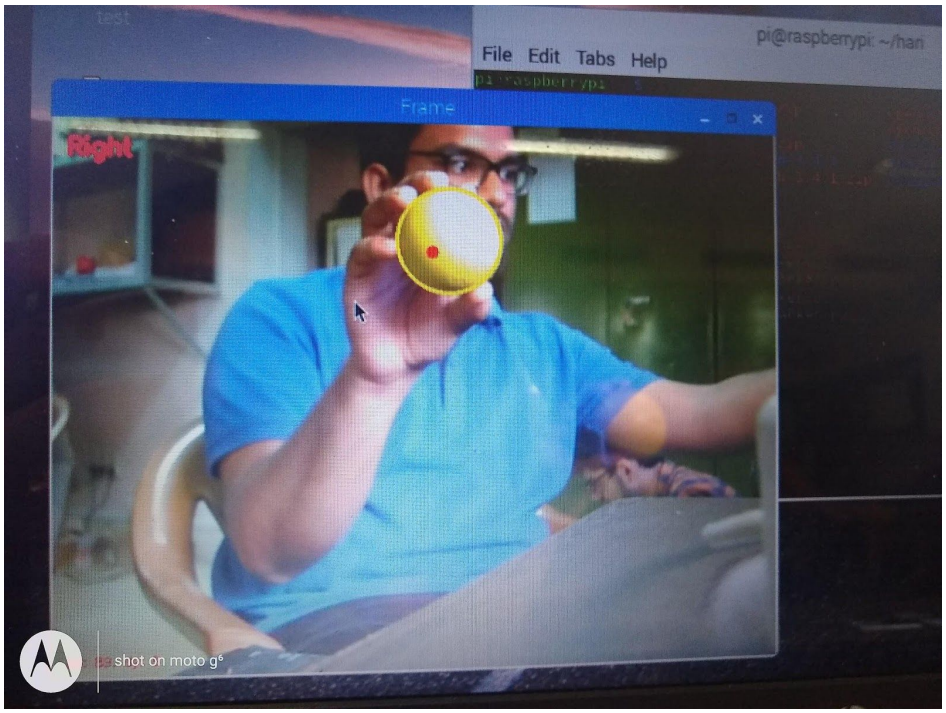
The python program was originally coded and tested in Ubuntu on a laptop and worked fine.
But, on executing the program on a Raspberry Pi, we experienced a lag in the video output. This was probably due to low processing power of Raspberry Pi compared to a laptop.
Due to this lag, the line representing the trail of the centers of the circle filled the screen making the output ugly. As of now, we just deleted the code snippet that draw those lines. We are trying to reduce the lag of the output.

This program relies on the contours obtained from a mask. This mask is made by finding the pixels with HSV values within a prespecified range. This makes it difficult to track an object of a color other than that color hard-coded within the code. For each different object, we need to find its HSV values and alter the program accordingly.
We are trying to make this easier by getting the HSV value range selected by the user using a trackbar.

Since this program tracks the color of an object, similar colored objects or background affects the program output. Also, shadows and light exposure alters the HSV values of the object. This makes it difficult to use the same HSV values in different lighting conditions. Currently, no specific solutions are found to resolve this issue.

RESULTS

A python program to detect the direction of motion of a moving object was successfully executed on a Raspberry Pi



REFERENCES

1. Blog post by Dr. Adrian Rosebrock https://www.pyimagesearch.com/2015/09/21/opencv-track-object-movement/
2. OpenCV documentation https://docs.opencv.org/2.4/doc/tutorials/tutorials.html
3. Getting Started with Videos https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_gui/py_video_display/py_video_display.html
4. Github repository of ‘imutils’ library https://github.com/jrosebr1/imutils