PROJECT REPORT

ON

IMAGE PROCESSING USING OPENCV

Submitted by:

ADITYA AGARWAL (9915102091)

AYUSH DAS (9915102099)

ADITYA CHANDWANI (9915102114)

Under the guidance of:

MR. ABHAY KUMAR

DEPARTMENT OF ELECTFRONICS AND COMMUNICATION ENGINEERING

JAYPEE INSTITUE OF INFORMATION TECHNOLOGY, NOIDA(U.P.)

MAY, 2018

CERTIFICATE

This is to certify that the thesis entitled “Image processing using OpenCv” submitted by Aditya Agarwal, Ayush Das and Aditya Chandwani in partial fulfillment of the requirements for the award of Bachelor of Technology Degree in Electronics and Communication Engineering of the Jaypee Institute of Information Technology, Noida is an authentic work carried out by them under any supervision and guidance. The matter embodied in this report is original and has not been submitted for the award of any other degree.

Signature of Supervisor:

Name of supervisor: Abhay Kumar

ECE Department,

JIIT, Sec-128

Noida-20301

Dated:

DECLARATION

We hereby declare that this written submission represents our own ideas in our own words and where other's ideas or words have been included, have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated any idea in our submission .

Place: Noida

Date:

Name: Aditya Agrawal

Enrollment no: 9915102091

Name: Ayush Das

Enrollment no: 9915102099

Name: Aditya Chandwani

Enrollment no: 9915102114

ABSTRACT

This project is about digital image processing using OpenCv. **Digital image processing** is the use of computer algorithms to perform image processing on digital images .OpenCV, which is an image and video processing library with bindings in C++, C, Python, and Java. OpenCV is used for all sorts of image and video analysis, like facial recognition and detection, license plate reading, photo editing, advanced robotic vision, optical character recognition, and a whole lot more.

The software used here is Pycharm. **PyCharm** is an Integrated Development Environment (IDE) used in computer programming, specifically for the Python language. It is developed by the Czech company Jet Brains. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes),

The following procedure is done:-

1. Download Python 2.7.x version, numpy and Opencv 2.7.x version. Check if your Windows either 32 bit or 64 bit is compatible and install accordingly.
2. Make sure that numpy is running in your python then try to install opencv.
3. Run the programs according to their xml files.

<u>ACKNOWLEDGEMENT</u>

Project making is an integral part of any graduate program and for that purpose we are in an institution which cannot get any better, one of the best. We take the opportunity to express our gratitude to all of them who in some or other way helped us to accomplish this challenging project on “Image processing using Open Cv”. No amount of written expression is sufficient to show our deepest sense of gratitude to them. We thank our institute who has given us an opportunity to show our skills. We also thank our nearer and dearer ones without their support this project would have not been completed.

We would like to thank **Mr. Abhay Kumar**, our mentor, who allowed who allowed us to do the project and gave his everlasting support and guidance on the ground of which we have acquired anew field of knowledge. It was he who gave us the golden opportunity to do this wonderful project on the above mentioned topic, which also helped us in a lot of learning and we came to know about so many new things due to which we are really thankful to him.

TABLE OF CONTENTS

ABSTRACT iv

ACKNOWLEDGEMENT...v

LIST OF FIGURES :- vii

CHAPTER 1 : IMAGE PROCESSING..1

1.1 Literature surveys related to image processing :- 2

1.1.1 Image-based Face Detection and Recognition: based Face Detection and Recognition (Faizan Ahmad , Aaima Najam and Zeeshan Ahmed):-...2

1.1.2 Tracking a tennis ball using image processing (Jnizi Mao):-.......................................2

1.1.3 Obstacle Avoidance and Color Detection using Image Segmentation (Kamruzzaman Ronny (kr334) and Norris Xu (nx26)) 2

CHAPTER 2 : BLOCK DIAGRAM...4

CHAPTER 3 : INTRODUCTION TO OPENCV..5

CHAPTER 4 : METHODOLOGIES AND ALGORITHMS USED...............................7

CHAPTER 5 : CONCLUSION AND FUTURE WORK 15

CHAPTER 6 : REFERENCES...16

CHAPTER 7 : APPENDIX...17

LIST OF FIGURES :-

Fig 5.1 still canny edge detection 8

Fig 5.2 webcam canny edge detection..8

Fig 5.5 face detection...10

Fig 5.4 red ball tracking...11

Fig 5.6 green ball path trace 12

Fig 5.7 distance between centroids of two balls……………………………………………….13

CHAPTER 1: IMAGE PROCESSING

Introduction to image processing**:**

- **Image processing** is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it
- It is a type of signal dispensation in which input is image, like video frame or photograph and output may be image or characteristics associated with that image.
- Usually **Image Processing** system includes treating images as two dimensional signals while applying already set signal processing methods to them.
- It is among rapidly growing technologies today, with its applications in various aspects of a business.

Image processing basically includes the following three steps.

- Importing the image with optical scanner or by digital photography.
- Analysing and manipulating the image which includes data compression and image enhancement and spotting patterns that are not to human eyes like satellite photographs.
- Output is the last stage in which result can be altered image or report that is based on image analysis.

1.1 Literature surveys related to image processing:-

1.1.1 Image-based Face Detection and Recognition: based Face Detection and Recognition (Faizan Ahmad , Aaima Najam and Zeeshan Ahmed):-

The goal of this paper is to evaluate various face detection and recognition methods, provide complete solution for image based face detection and recognition.

- Higher accuracy, better response rate as an initial step for video surveillance.
- Solution is proposed based on performed tests on various face rich databases in terms of subjects, pose, emotions, race and light.
- The actual advantages of face based identification over other biometrics are uniqueness and acceptance.

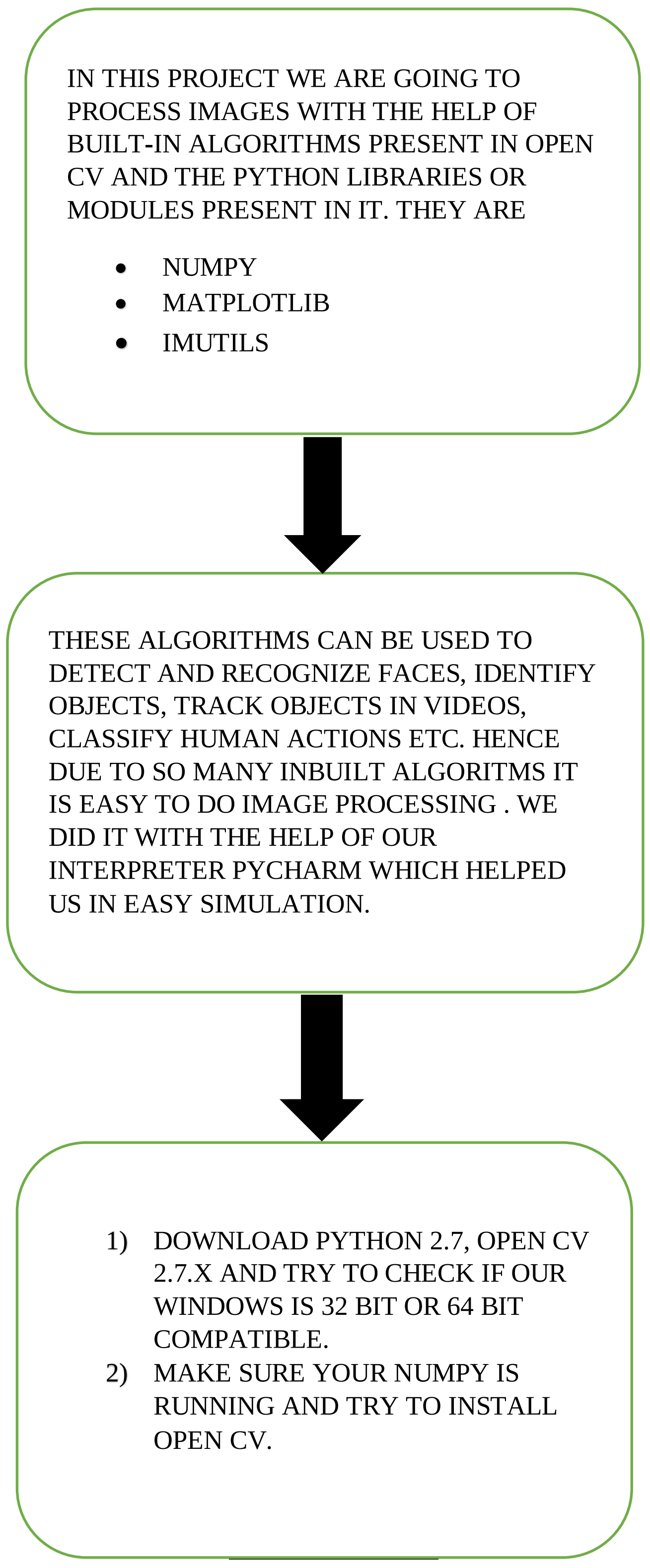
1.1.2 Tracking a tennis ball using image processing (Jnizi Mao):-

- In this paper, we first present the solution of background subtraction with color and shape segmentation.
- We compared it with haar’s classifier.
- With so many limitations still it is a feasible method for object tracking.
- Background subtraction, also known as foreground detection, is a technique in the fields of image processing and computer vision wherein an image's foreground is extracted for further processing.

1.1.3 Obstacle Avoidance and Color Detection using Image Segmentation (Kamruzzaman Ronny (kr334) and Norris Xu (nx26))

- Rovio, produced by WowWee, is a small robot equipped with a webcam, IR sensor, base station localization signal, and a unique wheel setup allowing it to move sideways and diagonally as well as forward and backward.
- The goal of this project is to program Rovio to successfully avoid any obstacles while navigating an indoor environment and simultaneously searching for a color-coded target object. For this project, we used only Rovio's webcam.
- We used some median filters to smooth the image and then applied image segmentation on the smoothed image; we then decided which of the segments was the floor and used the position of other segments to see where the obstacles were and avoid them.
- We used a simple color filter to find the target object and programmed Rovio to move sideways to avoid obstacles while keeping the target in view. We tested our code in a variety of different situations and the success rate was very high: in almost all trials, Rovio was able to successfully avoid obstacles and find the target.

CHAPTER 2 : BLOCK DIAGRAM



IN THIS PROJECT WE ARE GOING TO PROCESS IMAGES WITH THE HELP OF BUILT-IN ALGORITHMS PRESENT IN OPEN CV AND THE PYTHON LIBRARIES OR MODULES PRESENT IN IT. THEY ARE

- NUMPY
- MATPLOTLIB
- IMUTILS

THESE ALGORITHMS CAN BE USED TO DETECT AND RECOGNIZE FACES, IDENTIFY OBJECTS, TRACK OBJECTS IN VIDEOS, CLASSIFY HUMAN ACTIONS ETC. HENCE DUE TO SO MANY INBUILT ALGORITMS IT IS EASY TO DO IMAGE PROCESSING . WE DID IT WITH THE HELP OF OUR INTERPRETER PYCHARM WHICH HELPED US IN EASY SIMULATION.

1) DOWNLOAD PYTHON 2.7, OPEN CV 2.7.X AND TRY TO CHECK IF OUR WINDOWS IS 32 BIT OR 64 BIT COMPATIBLE.
2) MAKE SURE YOUR NUMPY IS RUNNING AND TRY TO INSTALL OPEN CV.

CHAPTER 3 : INTRODUCTION TO OPENCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. The library is used extensively in companies, research groups and by governmental bodies.

Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many startups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV. OpenCV's deployed uses span the range from stitching street view images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan.

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDA_and OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a template interface that works seamlessly with STL containers.

OpenCV (Open Source Computer Vision Library) is released under a BSD license and hence it's free for both academic and commercial use. It has C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform.

Adopted all around the world, OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. Usage ranges from interactive art, to mines inspection, stitching maps on the web or through advanced robotics.

CHAPTER 4 : METHODOLOGIES AND ALGORITHMS USED

Canny Edge Detection

Canny Edge Detection is a popular edge detection algorithm. It was developed by John F. Canny in

1. It is a multi-stage algorithm and we will go through each stages.
2. Noise Reduction

   Since edge detection is susceptible to noise in the image, first step is to remove the noise in the image with a 5x5 Gaussian filter.

3. Finding Intensity Gradient of the Image

   Smoothened image is then filtered with a Sobel kernel in both horizontal and vertical direction to get first derivative in horizontal direction (Gx) and vertical direction (Gy). From these two images, we can find edge gradient and direction for each pixel as follows:

   Edge_Gradient(G)=G2x+G2y−−−−−−−√Angle(θ)=tan−1(GyGx)

   Gradient direction is always perpendicular to edges. It is rounded to one of four angles representing vertical, horizontal and two diagonal directions.

4. Non-maximum Suppression

   After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient.

5. Hysteresis Thresholding

   This stage decides which are all edges are really edges and which are not. For this, we need two threshold values, minVal and maxVal. Any edges with intensity gradient more than maxVal are sure to be edges and those below minVal are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to "sure-edge" pixels, they are considered to be part of edges. Otherwise, they are also discarded. See the image below:

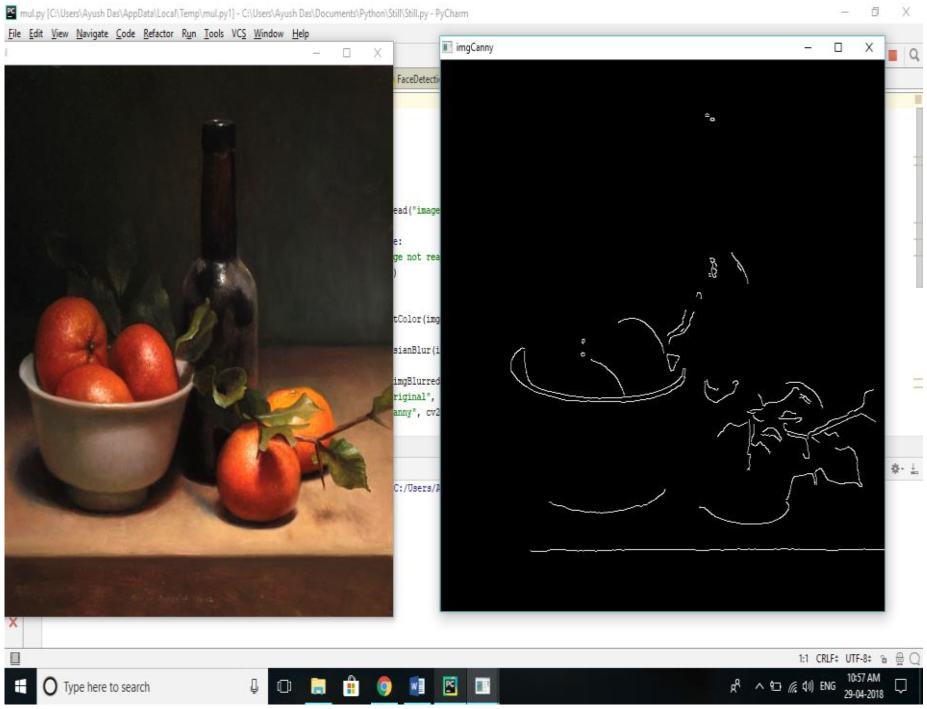

mul.py [C:\Users\Ayush Das\AppData\Local\Temp\mul.py1] - C:\Users\Ayush Das\Documents\Python\Still\Still.py - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
imgCanny
Type here to search
10:57 AM
29-04-2018

Fig 5.1 Still Canny Edge Detection

.

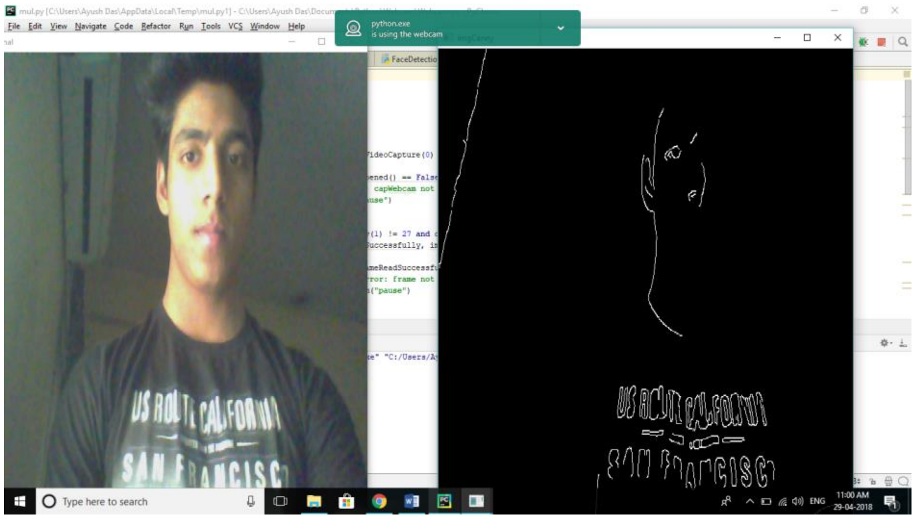

mul.py [C:\Users\Ayush Das\AppData\Local\Temp\mul.py1]
File Edit View Navigate Code Refactor Run Tools VCS Window Help
python.exe
is using the webcam
Type here to search
11:00 AM
29-04-2018

Fig 5.2 Webcam Canny Edge Detection

Face Detection using Haar Cascades

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

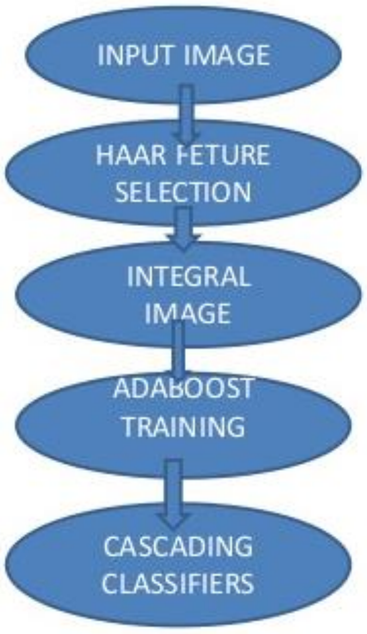
Here we will work with face detection. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, Haar features shown in the below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle.

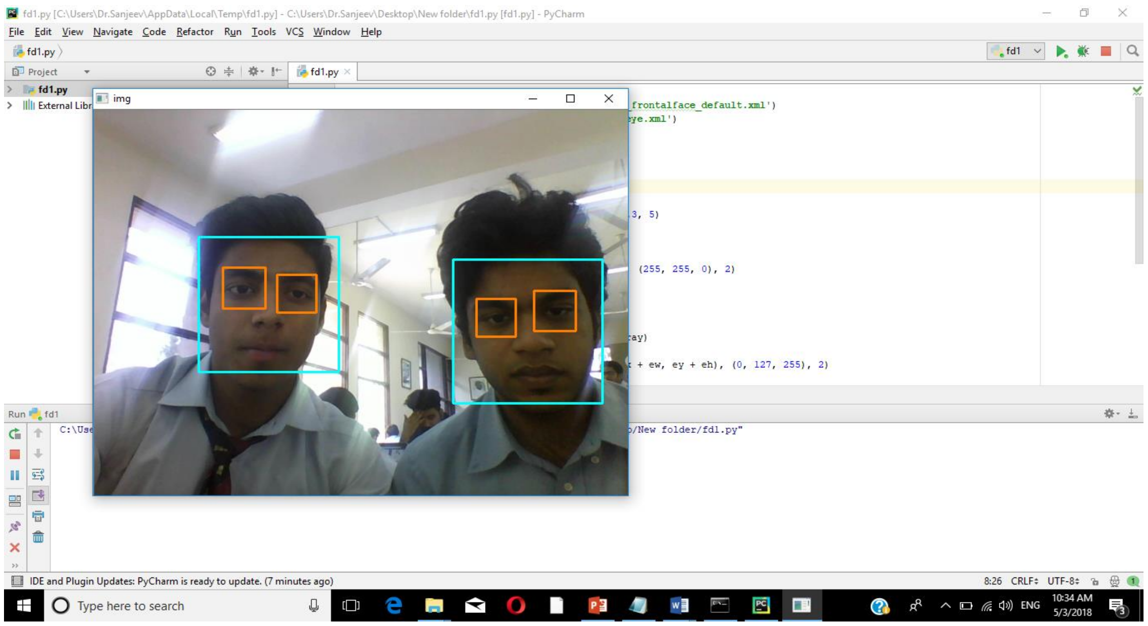
Now, all possible sizes and locations of each kernel are used to calculate lots of features. For each feature calculation, we need to find the sum of the pixels under white and black rectangles.

In an image, most of the image is non-face region. So it is a better idea to have a simple method to check if a window is not a face region. If it is not, discard it in a single shot, and don't process it again. Instead, focus on regions where there can be a face. This way, we spend more time checking possible face regions.

For this they introduced the concept of **Cascade of Classifiers**. Instead of applying all 6000 features on a window, the features are grouped into different stages of classifiers and applied one-by-one. (Normally the first few stages will contain very many fewer features). If a window fails the first stage, discard it. We don't consider the remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region.

Flowchart for face detection:




fd1.py [C:\Users\Dr.Sanjeev\AppData\Local\Temp\fd1.py] - C:\Users\Dr.Sanjeev\Desktop\New folder\fd1.py [fd1.py] - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
fd1.py
Project
fd1.py
External Libr
img
Run fd1
IDE and Plugin Updates: PyCharm is ready to update. (7 minutes ago)
8:26 CRLF UTF-8
Type here to search
ENG
10:34 AM
5/3/2018

```
_frontalface_default.xml')
eye.xml')

3, 5)

 (255, 255, 0), 2)

ray)
 + ew, ey + eh), (0, 127, 255), 2)
```

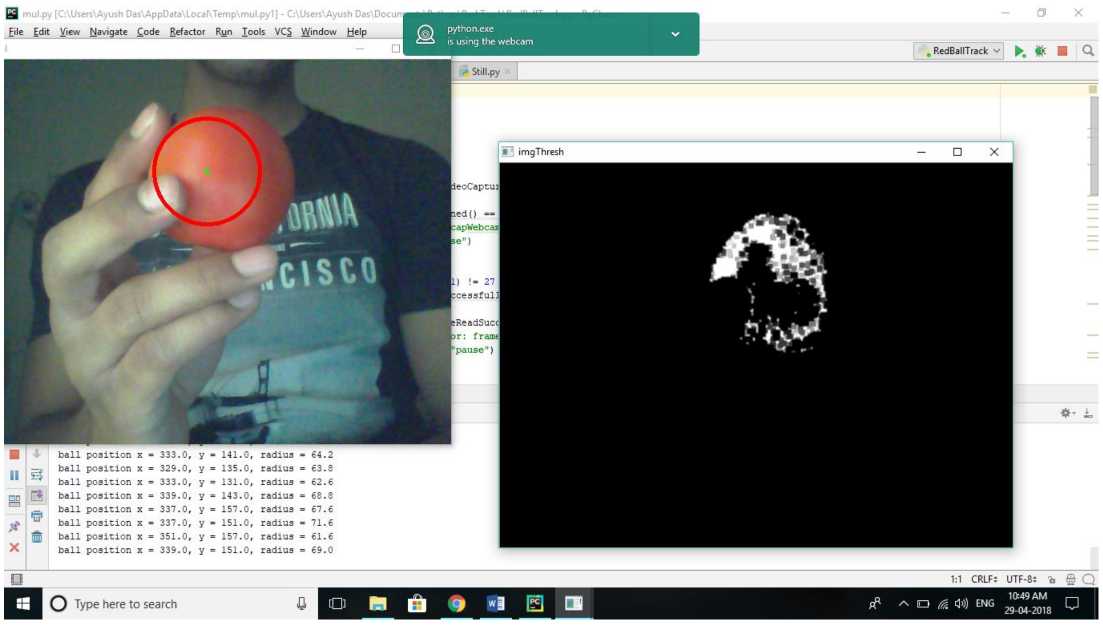
```
C:\Use
/New folder/fd1.py"
```

Fig 5.3 Face Detection

Object Tracking:-

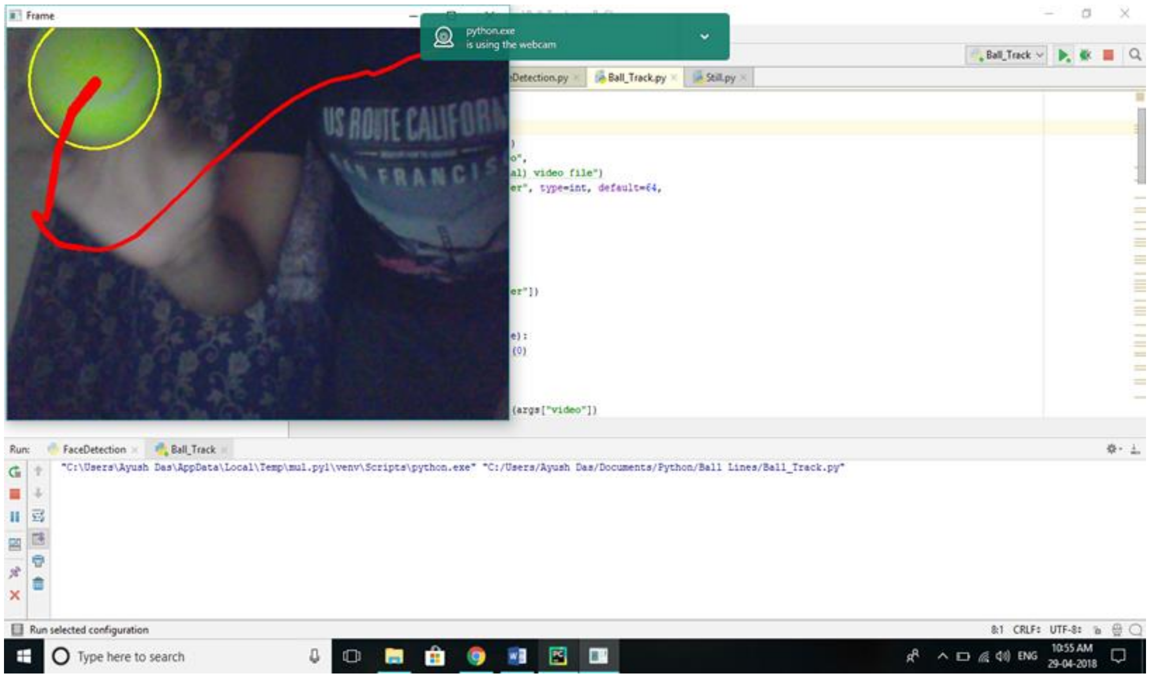
Now we know how to convert BGR image to HSV, we can use this to extract a colored object. In HSV, it is easier to represent a color than RGB color-space. In our application, we will try to extract a blue colored object. So here is the method:

- Take each frame of the video
- Convert from BGR to HSV color-space
- We threshold the HSV image for a range of blue color
- Now extract the blue object alone, we can do whatever on that image we want.



```
ball position x = 333.0, y = 141.0, radius = 64.2
ball position x = 329.0, y = 135.0, radius = 63.8
ball position x = 333.0, y = 131.0, radius = 62.6
ball position x = 339.0, y = 143.0, radius = 68.8
ball position x = 337.0, y = 157.0, radius = 67.6
ball position x = 337.0, y = 151.0, radius = 71.6
ball position x = 351.0, y = 157.0, radius = 61.6
ball position x = 339.0, y = 151.0, radius = 69.0
```

Fig 5.4 Red Ball Tracking



```
)
o",
al) video file")
er", type=int, default=64,

er"])

e):
(0)

(args["video"])
```

```
"C:\Users\Ayush Das\AppData\Local\Temp\mul.pyl\venv\Scripts\python.exe" "C:/Users/Ayush Das/Documents/Python/Ball Lines/Ball_Track.py"
```

Fig 5.6 Green Ball Path Trace

Distance between the Centroid of the Two Balls:-

- We used hsv (hue saturation value) to find the color range of the balls and after finding the color range.

- With the help of the boundary points we would form a contour to show the shape of the balls clearly.

- Then we will clear the noise using morphological transformation and find the centroid of the balls.

- After finding the centroid we will use the Euclidean algorithm to find the distance between the centroid of the two balls.

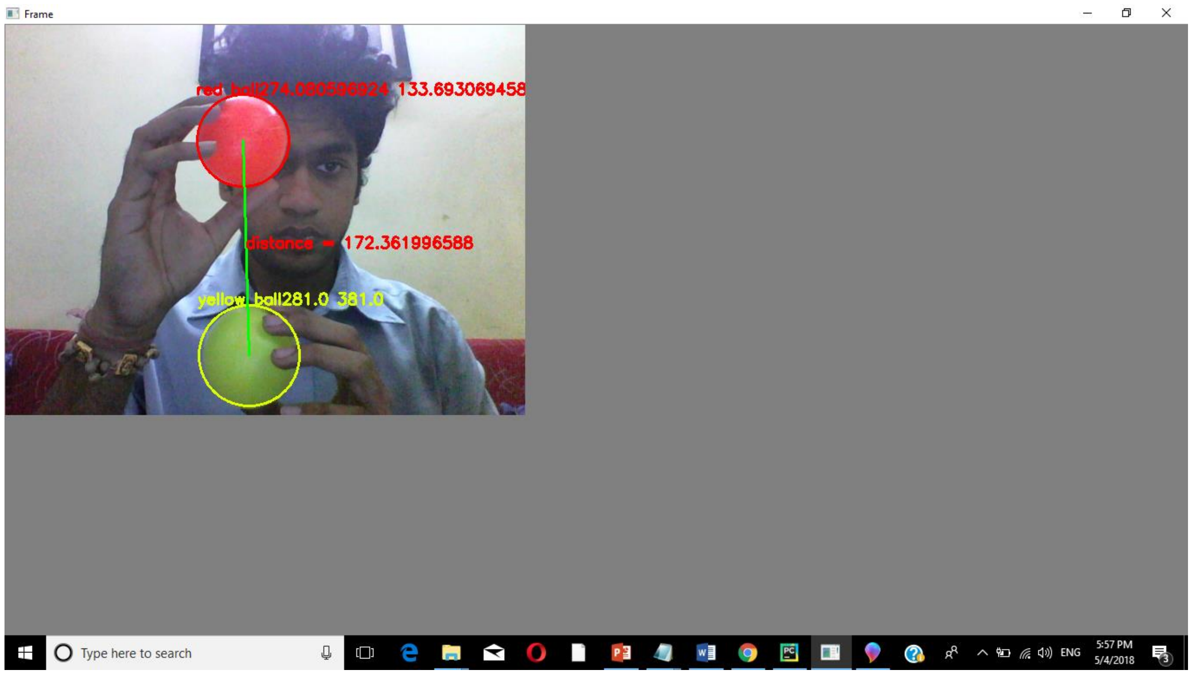


Fig 5.7 Distance Between Centroids Of Two Balls

Some Important Concepts:-

- **Image thresholding** is a simple, yet effective, way of partitioning an **image** into a foreground and background. This **image** analysis technique is a type of **image segmentation** that isolates objects by converting grayscale **images** into binary **images**.

- **HSL** (**hue, saturation, lightness**) and **HSV** (**hue, saturation, value**) are two alternative representations of the RGB color model, designed in the 1970s by computer graphics researchers to more closely align with the way human vision perceives color-making attributes. In these models, colors of each hue are arranged in a radial slice, around a central axis of neutral colors which ranges from black at the bottom to white at the top.

- **Morphological transformations**

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, second one is called **structuring element** or **kernel** which decides the nature of operation. Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing, Gradient etc also comes into play. We will see them one-by-one with help of following image:

1. Erosion

The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of foreground object (Always try to keep foreground in white). So what it does? The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).

So what happens is that, all the pixels near boundary will be discarded depending upon the size of kernel. So the thickness or size of the foreground object decreases or simply white region decreases in the image. It is useful for removing small white noises (as we have seen in colorspace chapter), detach two connected objects etc.

2. Dilation

It is just opposite of erosion. Here, a pixel element is '1' if atleast one pixel under the kernel is '1'. So it increases the white region in the image or size of foreground object increases. Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases. It is also useful in joining broken parts of an object.

3. Opening

Opening is just another name of **erosion followed by dilation**. It is useful in removing noise, as we explained above. Here we use the function

4. Closing

Closing is reverse of Opening, **Dilation followed by Erosion**. It is useful in closing small holes inside the foreground objects, or small black points on the object.

- **Median filter** is a nonlinear digital filtering technique, often used to remove noise from an image or signal. Such noise reduction is a typical pre-processing step to improve the results of later processing (for example, edge detection on an image).

CHAPTER 5 : CONCLUSION AND FUTURE WORK

- There is bio medical engineering which uses a lot of image processing techniques. Also there are research institutes which require people skilled in OpenCV. OpenCV can also be used very widely in Robotics applications, and as everyone knows, automation using robots is the future!

- The Vision community witnessed a paradigm shift towards learning representations (what is popularly known as Deep Learning, or DL), instead of using hand-crafted features (LBP, SIFT, SURF, HoG etc). The support for DL in OpenCV is still in its formative stages (this was the last time I checked, things might have improved a bit now). But that shouldn't be a point of concern because OpenCV has always been more of a Computer Vision library than a ML one. Also, the Python API for OpenCV allows it to be used quite comfortably with other Python-based DL libraries such as Tensorflow or PyTorch. Similarly, there are OpenCV bindings for Lua if you are a Torch user

CHAPTER 6 : REFERENCES

[1] A.N.a.Z.A. Faizan Ahmad, “face detection and recognition,” ieee, vol.4, no.1, p. 44, 2005.

[2] A.P.A.K.S.R. Telrandhe, “Brain Tumor Detection using Object Labeling Algorithm & SVM”, ieee, vol. 2, no. pp. 2-8, p. 28, Nov 2015.

[3] j. mao, “tracking a tennis ball,” neuma, vol. 2, no. 4, p.34, 1997.

CHAPTER 7 : APPENDIX

Still Canny Edge Detection

```
import cv2
import numpy as np
import os



def main():
    imgOriginal = cv2.imread("image.jpg")

    if imgOriginal is None:
        print "error: image not read from file \n\n"
        os.system("pause")
        return

    imgGrayscale = cv2.cvtColor(imgOriginal, cv2.COLOR_BGR2GRAY)

    imgBlurred = cv2.GaussianBlur(imgGrayscale, (5, 5), 0)

    imgCanny = cv2.Canny(imgBlurred, 100, 200)
    cv2.namedWindow("imgOriginal", cv2.WINDOW_AUTOSIZE)
    cv2.namedWindow("imgCanny", cv2.WINDOW_AUTOSIZE)

    cv2.imshow("imgOriginal", imgOriginal)
    cv2.imshow("imgCanny", imgCanny)

    cv2.waitKey()

    cv2.destroyAllWindows()
  return
if __name__ == "__main__":
    main()
```

Webcam Canny Edge Detection

```
import cv2
import numpy as np
import os


def main():

    capWebcam = cv2.VideoCapture(0)

    if capWebcam.isOpened() == False:
        print "error: capWebcam not accessed successfully\n\n"
        os.system("pause")
        return

    while cv2.waitKey(1) != 27 and capWebcam.isOpened():
        blnFrameReadSuccessfully, imgOriginal = capWebcam.read()

        if not blnFrameReadSuccessfully or imgOriginal is None:
            print "error: frame not read from webcam\n"
            os.system("pause")
            break

        imgGrayscale = cv2.cvtColor(imgOriginal, cv2.COLOR_BGR2GRAY)

        imgBlurred = cv2.GaussianBlur(imgGrayscale, (5, 5), 0)

        imgCanny = cv2.Canny(imgBlurred, 100, 200)

        cv2.namedWindow("imgOriginal", cv2.WINDOW_NORMAL)
        cv2.namedWindow("imgCanny", cv2.WINDOW_NORMAL)

        cv2.imshow("imgOriginal", imgOriginal)
        cv2.imshow("imgCanny", imgCanny)


    cv2.destroyAllWindows()
```

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

Scaling

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('messi5.jpg')

rows,cols,ch = img.shape

pts1 = np.float32([[50,50],[200,50],[50,200]])
pts2 = np.float32([[10,100],[200,50],[100,250]])

M = cv2.getAffineTransform(pts1,pts2)

dst = cv2.warpAffine(img,M,(cols,rows))

plt.subplot(121),plt.imshow(img),plt.title('Input')

plt.subplot(122),plt.imshow(dst),plt.title('Output')

plt.show()
```

Red Ball Track

```
import cv2
import numpy as np
import os


def main():

    capWebcam = cv2.VideoCapture(0)

    if capWebcam.isOpened() == False:
        print "error: capWebcam not accessed successfully\n\n"
        os.system("pause")
        return

    while cv2.waitKey(1) != 27 and capWebcam.isOpened():
        blnFrameReadSuccessfully, imgOriginal = capWebcam.read()

        if not blnFrameReadSuccessfully or imgOriginal is None:
            print "error: frame not read from webcam\n"
            os.system("pause")
            break

        imgHSV = cv2.cvtColor(imgOriginal, cv2.COLOR_BGR2HSV)

        imgThreshLow = cv2.inRange(imgHSV, (0, 155, 155), (18, 255, 255))
        imgThreshHigh = cv2.inRange(imgHSV, (165, 155, 155), (179, 255, 255))

        imgThresh = cv2.add(imgThreshLow, imgThreshHigh)

        imgThresh = cv2.GaussianBlur(imgThresh, (3, 3), 2)

        imgThresh = cv2.dilate(imgThresh, np.ones((5,5),np.uint8))
        imgThresh = cv2.erode(imgThresh, np.ones((5,5),np.uint8))

        intRows, intColumns = imgThresh.shape

        circles = cv2.HoughCircles(imgThresh, cv2.HOUGH_GRADIENT, 2, intRows / 4)
        if circles is not None:
            for circle in circles[0]:
                x, y, radius = circle
                print "ball position x = " + str(x) + ", y = " + str(y) + ", radius = " + str(radius)
                cv2.circle(imgOriginal, (x, y), 3, (0, 255, 0), cv2.FILLED)
                cv2.circle(imgOriginal, (x, y), radius, (0, 0, 255), 3)


        cv2.namedWindow("imgOriginal", cv2.WINDOW_AUTOSIZE)
        cv2.namedWindow("imgThresh", cv2.WINDOW_AUTOSIZE)

        cv2.imshow("imgOriginal", imgOriginal)
        cv2.imshow("imgThresh", imgThresh)


    cv2.destroyAllWindows()

    return

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

Face Detection

```
import cv2
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
cap = cv2.VideoCapture(0)

while 1:

    ret, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    for (x, y, w, h) in faces:

        cv2.rectangle(img, (x, y), (x + w, y + h), (255, 255, 0), 2)

        roi_gray = gray[y:y + h, x:x + w]
        roi_color = img[y:y + h, x:x + w]

        eyes = eye_cascade.detectMultiScale(roi_gray)
        for (ex, ey, ew, eh) in eyes:
            cv2.rectangle(roi_color, (ex, ey), (ex + ew, ey + eh), (0, 127, 255), 2)

    cv2.imshow('img', img)

    k = cv2.waitKey(30) & 0xff

    if k == 27:
        break

cap.release()

cv2.destroyAllWindows()
```

Green Ball Path Trace

```
from collections import deque
import numpy as np
import argparse
import imutils
import cv2


ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video",
  help="path to the (optional) video file")
ap.add_argument("-b", "--buffer", type=int, default=64,
  help="max buffer size")
args = vars(ap.parse_args())


greenLower = (29, 86, 6)
greenUpper = (64, 255, 255)
pts = deque(maxlen=args["buffer"])


if not args.get("video", False):
  camera = cv2.VideoCapture(0)


else:
  camera = cv2.VideoCapture(args["video"])


while True:

  (grabbed, frame) = camera.read()


  if args.get("video") and not grabbed:
    break


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

```
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)


mask = cv2.inRange(hsv, greenLower, greenUpper)
mask = cv2.erode(mask, None, iterations=2)
mask = cv2.dilate(mask, None, iterations=2)


cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
  cv2.CHAIN_APPROX_SIMPLE)[-2]
center = None


if len(cnts) > 0:

  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"]))


  if radius > 10:

    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)


for i in xrange(1, len(pts)):

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

  thickness = int(np.sqrt(args["buffer"] / float(i + 1)) * 2.5)
  cv2.line(frame, pts[i - 1], pts[i], (0, 0, 255), thickness)
```

```
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF


    if key == ord("q"):
      break


  camera.release()
  cv2.destroyAllWindows()
```

Identifying The Distance Between The Centroid of Two Balls

```
from collections import deque
import numpy as np
import argparse
import imutils
import cv2
import urllib


ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video",
                help="path to the (optional) video file")
ap.add_argument("-b", "--buffer", type=int, default=64,
                help="max buffer size")
args = vars(ap.parse_args())

lower = {'red':(166, 84, 141),'yellow': (23, 59, 119)}
upper = {'red':(186,255,255),'yellow': (54, 255, 255)}

colors = {'red':(0,0,255),'yellow': (0, 255, 217)}

if not args.get("video", False):
    camera = cv2.VideoCapture(0)



else:
    camera = cv2.VideoCapture(args["video"])

while True:

    (grabbed, frame) = camera.read()

    if args.get("video") and not grabbed:
        break


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

    blurred = cv2.GaussianBlur(frame, (11, 11), 0)
    hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
    pts = []
    for key, value in upper.items():
        kernel = np.ones((9, 9), np.uint8)
        mask = cv2.inRange(hsv, lower[key], upper[key])
        mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
        mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)


        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
                                cv2.CHAIN_APPROX_SIMPLE)[-2]



        center = None

        if len(cnts) > 0:

            c = max(cnts,key=cv2.contourArea)
            ((x, y), radius) = cv2.minEnclosingCircle(c)


            if radius > 0.5:

                cv2.circle(frame, (int(x), int(y)), int(radius), colors[key], 2)
                cv2.putText(frame, key + " ball" + str(x) + " " + str(y), (int(x - radius), int(y -
radius)), cv2.FONT_HERSHEY_SIMPLEX, 0.6,
                            colors[key], 2)
                pts.append((x,y))

    if(len(pts) == 2):
        lineThickness = 2
        x1,y1 = pts[0]
        x2,y2 = pts[1]
        d = ((x1-y1)**2 + (x2-y2)**2)**0.5
        cv2.line(frame, (int(x1), int(y1)), (int(x2), int(y2)), (0, 255, 0), lineThickness)
        cv2.putText(frame, 'distance = ' + str(d), (int(0.5*(x1 + x2)), int(0.5*(y1 + y2))),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, colors[key], 2)
    cv2.imshow("Frame", frame)

    key = cv2.waitKey(1) & 0xFF4
    if key == ord("q"):
        break


camera.release()
cv2.destroyAllWindows()
```