Acknowledgment

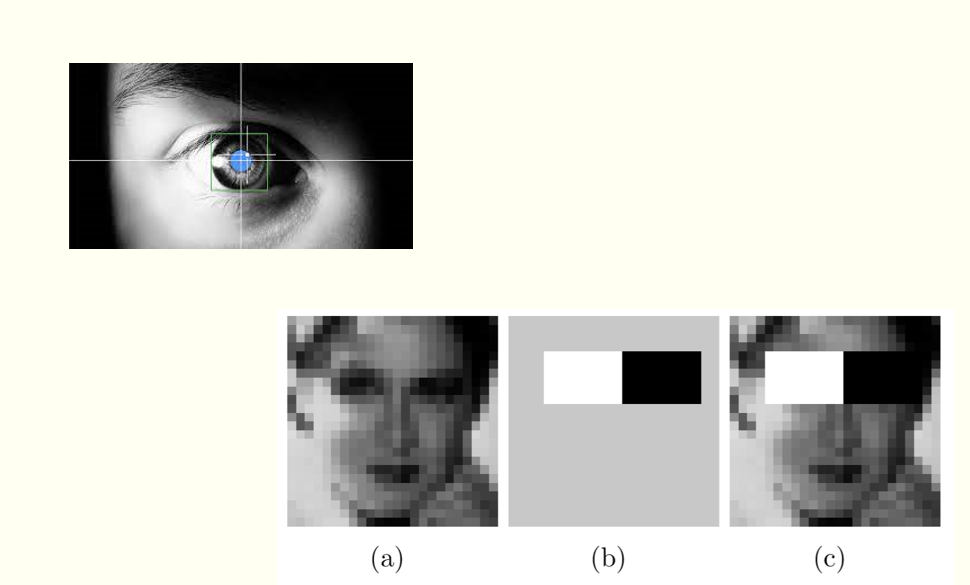
I would like to thank Prof. Dr. Ugur HALICI for her valuable support throughout the development of this thesis. This thesis really would not have been reach to an end without her guidance and patience. I would like to thank to Ilkay ULUSOY for her guidance at the beginning of this work. I also would like to thank to ASELSAN A.S. especially to my department Test Engineering for letting me to involve in this thesis work. Finally I am grateful to my parents and my colleagues at work for their continuous encouragement.

abstract

The problem of eye detection in face images is very important for a large number of applications ranging from face recognition to gaze tracking. In this paper, we propose a new algorithm for eyes detection that uses iris geometrical information for determining in the whole image the region candidate to contain an eye, and then the symmetry for selecting the couple of eyes. The novelty of this work is that the algorithm works on complex images without constraints on the background, skin color segmentation and so on. Different experiments, carried out on images of subjects with different eyes colors, some of them wearing glasses, demonstrate the effectiveness and robustness of the proposed algorithm.

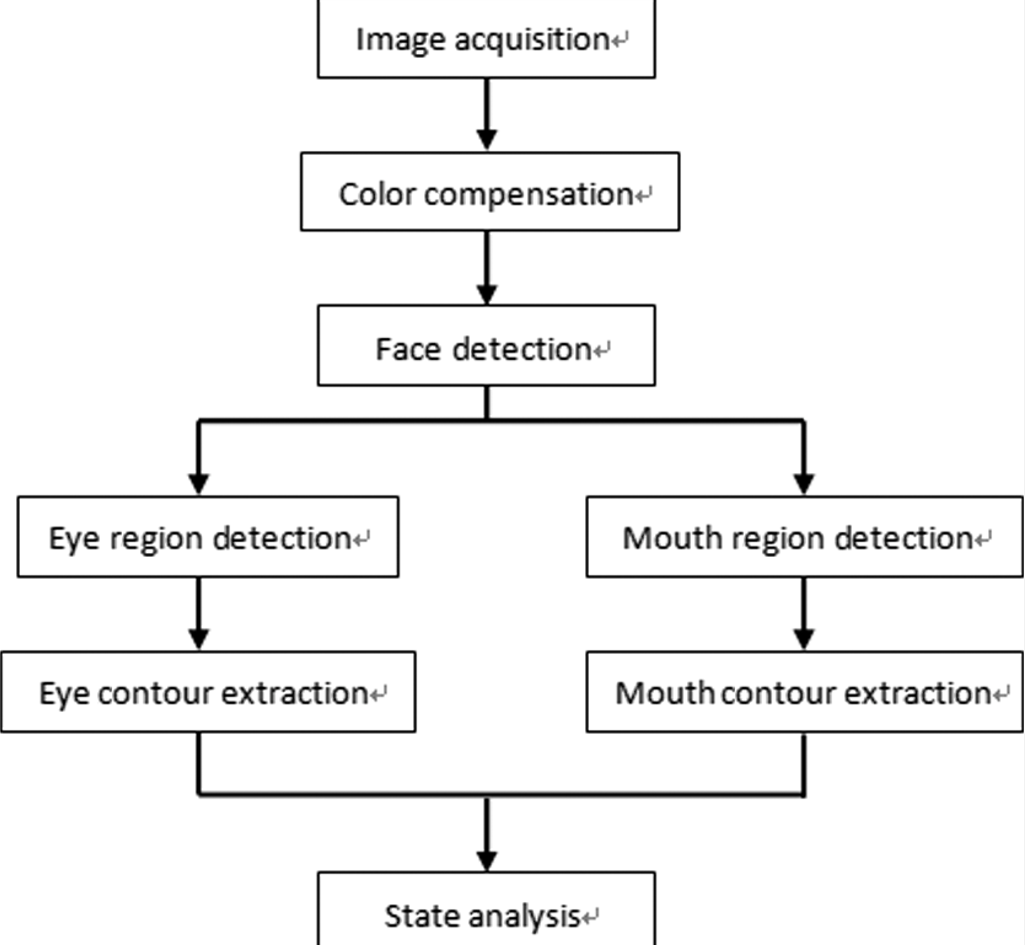
Intro

Reliable object detection is very important in computer vision and robotics applications. The histogram of oriented gradients (HOG) is established as one of the most popular hand-crafted features, which along with support vector machine (SVM) classification provides excellent performance for object recognition. We investigate dimensionality deduction on HOG features in combination with SVM classifiers to obtain efficient feature representation and improved classification performance. In addition to lean HOG features, we explore descriptors resulting from dimensionality reduction on histograms of binary descriptors. We consider three-dimensionality reduction techniques: standard principal component analysis, random projections, a computationally efficient linear mapping that is data independent, and locality preserving projections (LPP), which learns the manifold structure of the data. Our methods focus on the application of eye detection and were tested on an eye database created using the BioID and FERET face databases. Our results indicate that manifold learning is beneficial to classification utilizing HOG features. To demonstrate the broader usefulness of lean HOG features for object class recognition, we evaluated our system's classification performance on the CalTech-101 dataset with favorable outcomes.



Existing system:Current most of the objection detection model just detect the region of the eye using Haar cascade detector which use filters to detect. This is basically a machine learning based approach where a cascade function is trained from a lot of images both positive and negative. Based on the training it is then used to detect the objects in the other images.

So how this works is they are huge individual .xml files with a lot of feature sets and each xml corresponds to a very specific type of use case.



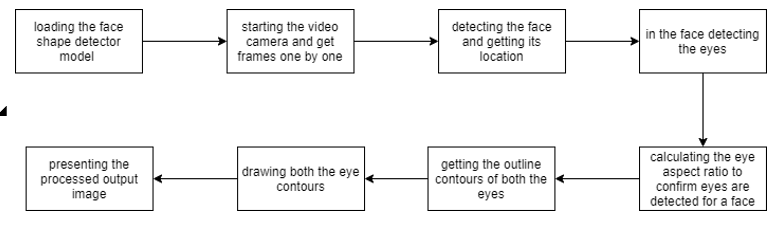
Disadvantages:

It detects the region with bounding box which contains the eye but not the eye curve, pinpoint of the eye using HOG and SVM algorithms.

3. propsed system:

We are using features extraction and feature recognition to recognize the region which contains the eye then advance image processing techniques to get the boundary of the eye which are used in all modern face recognition model.

Architecture:



Advantages:

It uses advanced features extraction techniques used by all modern face recognition projects by companies like google, apple. It gives us the exact eye instead of a box containing the eye.

conclusion

In this study, a real-time system capable of face tracking, eye area tracking and extracting eye features was developed. The system’s face tracking algorithm is based on CAMSHIFT algorithm developed by [4] which is a feature-based face tracking algorithm searching for the color of the human skin in images. Detection of eye areas after face localization is based on Principal Components Analysis technique which was previously used for face recognition. In order to detect eyes with different sizes without so much computational work and without using databases of different sizes for each image, we have developed an eigenface based method called adaptive eigeneye method. After detection of the eye areas, features of the eyes, region#1 which is the area containing eye pupil and iris and region#2 which is the visible eyeball area, are determined. The extraction of detailed location and shape information of eye features is performed by applying different image preprocessings and region segmentation based on boundary estimation using edge features. Boundary of the final regions are found by circle detection algorithms or by an active contour (Snake) The system works well with cheap cameras and does not require camera lens calibration. Although the system performance is highly dependent on the system settings done by the user before the tracking process, it is very easy to make these adjustment by the help of friendly user interface.

From the obtained results it is clear that histogram of gradient provides better result in cataract detection as it directly identify the white layer responsible for cataract in an eye. It is less complexity and less time consuming when compared to exiting methods like support machine vector. This work can also further applied for traffic sign detection.

**CHAPTER-4 SYSTEM REQUIREMENT SPECIFICATIONS**

**Technology**

* **Software Requirement Specification**

Front End : HTML

Back End : JAVA

Operating System : Windows-10

Frame Work : JAVA

* **Hardware Requirements**

Processor : Intel Pentium

Hard disk : Minimum 500MB

RAM : Minimum 2GB

**SAMPLE CODE:**

##Importing libraries

from imutils import face\_utils as Futility

import cv2 as ComputerVision

import systemcheck

import imutils as ImageUtilities

import dlib as digitalLibrary

from scipy.spatial import distance as length

import cv2

ColortoBWconverter = ComputerVision.COLOR\_BGR2GRAY

THICKNESS = 2

COLOR = (0,0,255)

sensitivity = 0.19  # Threshold to consider eye as closed

ww = 450

det = 0

countFrm = 15  # increase eye close time

check = digitalLibrary.get\_frontal\_face\_detector()

forecast = digitalLibrary.shape\_predictor("faceShape.dat")

(LeftEyeStart, LeftEyeEnd) = Futility.FACIAL\_LANDMARKS\_68\_IDXS["left\_eye"]

(RightEyeStart, RightEyeEnd) = Futility.FACIAL\_LANDMARKS\_68\_IDXS["right\_eye"]

Internalcamera = ComputerVision.VideoCapture(0)

# Initialize the Internal Camera

def getRatio(gridEye):

    global length

    ratio1 = length.euclidean(gridEye[1], gridEye[5])

    ratio2 = length.euclidean(gridEye[2], gridEye[4])

    ratio3 = length.euclidean(gridEye[0], gridEye[3])

    ratio = (ratio1 + ratio2) / (2.0 \* ratio3)

    return ratio

def drawEyeCurve(eyeCurve):

    global ComputerVision

    global RawImage

    ComputerVision.drawContours(RawImage, [eyeCurve], -1, COLOR, thickness=THICKNESS)

def getLeftEyeCurveDetails(Lrat):

    global ComputerVision

    x = ComputerVision.convexHull(Lrat)

    return x

def getRightEyeCurveDetails(Rrat):

    global ComputerVision

    x = ComputerVision.convexHull(Rrat)

    return x

while True:

    done,RawImage =Internalcamera.read()

    # Read the Image from Internal Camera

    if (done == 0):

        print("Unable to get Image data")

        continue

    RawImage = ImageUtilities.resize(RawImage, width= ww)

    #Resizing the Image in suitable size

    BWimg = ComputerVision.cvtColor(RawImage, ComputerVision.COLOR\_BGR2GRAY)

    # Making the Color Black&White for size Reduction and faster Processing

    BWimg = ImageUtilities.resize(BWimg, width= ww)

    #Resizing the B&W Image in suitable size

    multiFaceData = check(BWimg, 0)

    # Get data of all the multiple faces found

    det = 0

    #print(det)

    for singleFaceData in multiFaceData:

        # getting the detected face locations

        x1,y1,x2,y2 = singleFaceData.left(),singleFaceData.top(),singleFaceData.right(),singleFaceData.bottom()

        # Take all the faces one by one from image

        det = 0

        geometry = forecast(BWimg, singleFaceData)

        # Try Predicting Location of the face

        geometry = Futility.shape\_to\_np(geometry)

        # Change the face data into Numpy array for numerical processing

        LEye = geometry[LeftEyeStart:LeftEyeEnd]

        #Get the Geometrical details of LeftEye

        if(LEye.any()):

            det = 1

        Lratio = getRatio(LEye)

        #Get the Aspect ratio of the left eye

        if(Lratio < 0):

            continue

        REye = geometry[RightEyeStart:RightEyeEnd]

        #Get the Geometrical details of RightEye

        if(REye.any()):

            det = 1

        Rratio = getRatio(REye)

        #Get the Aspect ratio of the right eye

        if(Rratio < 0):

            continue

        finalRatio = (Lratio + Rratio )/ 2.0

        # Get the arithmetic mean of both ratios

        LEyeCurve = getLeftEyeCurveDetails(LEye)

        drawEyeCurve(LEyeCurve)

        #Get the curve definition of Left Eye and Draw it

        REyeCurve = getRightEyeCurveDetails(REye)

        drawEyeCurve(REyeCurve)

        # drawing rectangle around the detected face

        ComputerVision.rectangle(RawImage, (x1,y1), (x2,y2), (255,0,255), 3)

        #Get the curve definition of Right Eye and draw it

    ComputerVision.imshow("Eye Detection", RawImage)

    #Show thw Update RawImage on Screen

    got = ComputerVision.waitKey(125) & 0xFF

    if got == 27:

        break

cv2.destroyAllWindows()

Internalcamera.release()