Shri Vile Parle Kelvani Mandal's

DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING

Approved by AICTE and Affiliated to the University of Mumbai

Department of Electronics & Telecommunication Engineering

Mini Project Report On

Title: **Patient Fall Detection using Haar Cascades**

SUBMITTD BY:

Sr No.	Name of Students	SAP ID
1.	Akshay Bhogan	60002160007
2.	Yogesh Deshpande	60002160021
3.	Atulya Kumar	60002160052

Teacher's Name: Dr. Sunil Karamchandani

Shri Vile Parle Kelvani Mandal's

DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING

Approved by AICTE and Affiliated to the University of Mumbai

Department of Electronics & Telecommunication Engineering

CERTIFICATE

This is to certify that M/S. _____,

SAP ID_____ of TE EXTC 1: has submitted their

Case Study for Image Processing and Machine Vision for the Academic Year 2018-2019.

Guide

Examiner

Head of Department

EXTC Department

Index

Sr No.	Topic	Page No.
1	Introduction	1
2	Theory	1
3	Implementation	2
4	Code	3
5	Observation	18
6	Software Design	19
7	Conclusion	20
8	Reference	20

Introduction:

Falls represent a major public health risk worldwide for the elderly people. A fall not assisted in time can cause functional impairment in an elder and a significant decrease in his mobility, independence and life quality. Our modern societies are suffering the increase of elderly population while at the same time social security and health costs must be cut down. In order to avoid the need of special care centres, the actual trend is to encourage elderly to stay living autonomously in their own homes as long as possible. This thesis proposes an automatic monitoring system for formal and informal home care patients and care centres. The proposed system will provide security and a feeling of safety by detecting when a resident suffers from a fall. This feeling of safety can increase the person's ability to perform daily routines at home. After the detection the system will be able to alert professional personnel or family.

Theory:

The proposed system is affordable. This system uses computer vision to detect persons and their actions. The patient does not need to remember to put the detector on, charge it or anything else, which is convenient for the target group that suffers from dementia. This solution also does not need costly installations or costly hardware. On the other hand, the system provides a service leveraging this data to create a new machine learning model each time a fall is detected. In most of our countries, elderly people represent the fastest growing segment of the population, and this trend will increase over the next years. Indeed, by the year 2035, one third of the European population will be more than 65 years old. At the same time, the Public Health Services institutions have to face budget restrictions and increasing pressure to limit costs. Together with the lack of rooms in the care centres, these evolutions lead to encourage elderly to stay living longer at home instead of being admitted in care centres. For the elderly population, which represents a large part of Social Health Services expenditures, it means most of the time living alone and independently in their homes, with all the risks it involves. One of the major risks incurred by the fragile population (elderly, illness, people in adaptation time after a chirurgical intervention, etc…) is to fall. Indeed, 30% of elderly people fall once a year at least, representing

75% of the victims of falls. The fall event is responsible for 70% of accidental deaths in persons aged 75+, and for increasing the level of fear, anxiety or depression leading to the reduction of the day to day activity. These observations have encouraged the development of fall detection devices to detect or even prevent a fall event and to ensure a rapid and efficient help when such an event occur.

Implementation:

Object detection is commonly referred to as a method that is responsible for discovering and identifying the existence of objects of a certain class. An extension of this can be considered as a method of image processing to identify objects from digital images. The cascade classifier consists of a list of stages, where each stage consists of a list of weak learners. The system detects objects in question by moving a window over the image. Each stage of the classifier labels the specific region defined by the current location of the window as either positive or negative –positive meaning that an object was found or negative means that the specified object was not found in the image. A Haar Cascade is basically a classifier which is used to detect the object for which it has been trained for, from the source. The Haar Cascade is trained by superimposing the positive image over a set of negative images.
For our usage we have used a combination of upper, lower and full body haar cascades.

The video is analysed frame by frame and the motion of the detected person is tracked. Changes are recorded between the previous frame and current frame. A sudden fall results in calculation of Δy of upper and lower body cascade.

If the change in Δy is negative i.e. upper body moves downwards rapidly and the full body haar cascade detects "horizontality" of the body, then after a few seconds an alarm is triggered.

Code:

Main.py:

```
import video
import time
import sys
import numpy as np
import cv2
import time
video = video.Video()
time.sleep(1.0) # let camera autofocus + autosaturation settle
video.nextFrame()
video.testBackgroundFrame()
while 1:
        video.nextFrame()
        video.testBackgroundFrame()
        video.updateBackground()
        video.compare()
        video.showFrame()
        video.testSettings()
        if video.testDestroy():
                sys.exit()
```

bs.py

```
import cv2
import settings
class Bs:
        def __init__(self):
```

```
        self.settings = settings.Settings()
        self.method = self.settings.bsMethod
        if self.method == 0:
            self.fgbg =
cv2.BackgroundSubtractorMOG2(self.settings.MOG2history,
self.settings.MOG2thresh, self.settings.MOG2shadow)
            self.foregroundMask = None

        if self.method == 1:
            self.backgroundFrame = None
            self.frameCount = 1

    def updateBackground(self, frame):
        if self.method == 0:
            self.foregroundMask = self.fgbg.apply(frame,
self.foregroundMask, self.settings.MOG2learningRate)

        if self.method == 1:
            alpha = (1.0/self.frameCount)
            if self.backgroundFrame is None:
                self.backgroundFrame = frame
            self.backgroundFrame = cv2.addWeighted(frame, alpha,
self.backgroundFrame, 1.0-alpha, 0)
            self.frameCount += 1

    def compareBackground(self, frame):
        if self.method == 0:
            return self.foregroundMask
```

```
            if self.method == 1:
                self.frameDelta = cv2.absdiff(self.backgroundFrame, frame)
                self.foregroundMask = cv2.threshold(self.frameDelta,
self.settings.thresholdLimit, 255, cv2.THRESH_BINARY)[1]
                return self.foregroundMask

        def deleteBackground(self):
            if self.method == 0:
                self.foregroundMask = None

            if self.method == 1:
                self.backgroundFrame = None

        def resetBackgroundIfNeeded(self, frame):
            if self.method == 0:
                if self.foregroundMask is None:
                    self.foregroundMask = self.fgbg.apply(frame)

            if self.method == 1:
                if self.backgroundFrame is None:
                    self.updateBackground(frame)
                    self.frameCount = 1
```

person.py

```
class Person(object):
    """Person"""
    amount = 0

    def __init__(self, x, y, w, h, movementMaximum, movementMinimum,
movementTime):
        self.x = x
        self.y = y
        self.w = w
        self.h = h
        self.movementTime = movementTime
        self.movementMaximum = movementMaximum
        self.movementMinimum = movementMinimum
        self.lastmoveTime = 0
        self.alert = 0
        self.alarmReported = 0
        self.lastseenTime = 0
        self.remove = 0
        Person.amount += 1
        if Person.amount > 1000:
            Person.amount = 0
        self.id = Person.amount

    def samePerson(self, x, y, w, h):
        same = 0
```

```
            if x+self.movementMaximum > self.x and x-
self.movementMaximum < self.x:
                if y+self.movementMaximum > self.y and y-
self.movementMaximum < self.y:
                    same = 1
            return same


    def editPerson(self, x, y, w, h):
            if abs(x-self.x) > self.movementMinimum or abs(y-self.y) >
self.movementMinimum or abs(w-self.w) > self.movementMinimum or abs(h-
self.h) > self.movementMinimum:
                self.lastmoveTime = 0
            self.x = x
            self.y = y
            self.w = w
            self.h = h
            self.lastseenTime = 0


    def getId(self):
            return self.id


    def tick(self):
            self.lastmoveTime += 1
            self.lastseenTime += 1
            if self.lastmoveTime > self.movementTime:
                self.alert = 1
            if self.lastseenTime > 4: # how many frames ago last seen
                self.remove = 1
```

```
    def getAlert(self):
        return self.alert

    def getRemove(self):
        return self.remove
class Persons:
    def __init__(self, movementMaximum, movementMinimum,
movementTime):
        self.persons = []
        self.movementMaximum = movementMaximum
        self.movementMinimum = movementMinimum
        self.movementTime = movementTime
        Person.amount = 0

    def addPerson(self, x, y, w, h):
        person = self.familiarPerson(x, y, w, h)
        if person:
            person.editPerson(x, y, w, h)
            return person
        else:
            person = Person(x ,y ,w ,h , self.movementMaximum,
self.movementMinimum, self.movementTime)
            self.persons.append(person)
            return person

    def familiarPerson(self, x, y, w, h):
```

```
            for person in self.persons:
                if person.samePerson(x, y, w, h):
                    return person
            return None

        def tick(self):
            for person in self.persons:
                person.tick()
                if person.getRemove():
                    self.persons.remove(person)
```

setings.py

```
class Settings(object):

    def __init__(self):
        self.debug = 1 # boolean
        self.source = 0 # camera source
        self.bsMethod = 1 # listed in bs.py
        self.MOG2learningRate = 0.001
        self.MOG2shadow = 0
        self.MOG2history = 100
        self.MOG2thresh = 20
        self.minArea = 50*50 # minimum area to be considered as a person
        self.thresholdLimit = 50
        self.dilationPixels = 30
        self.useGaussian = 0 # boolean
        self.useBw = 1 # boolean
```

```
        self.useResize = 1 # boolean
        self.gaussianPixels = 31
        self.movementMaximum = 75 # amount to move to still be the same person
        self.movementMinimum = 3 # minimum amount to move to not trigger alarm
        self.movementTime = 50 # number of frames after the alarm is triggered
        self.location = 'Viikintie 1'
        self.phone = '01010101010'
```

video.py

```
import time
import cv2
import person
import settings
import webservice
import bs

class Video:
    def __init__(self):
        self.settings = settings.Settings()
        self.camera = cv2.VideoCapture(self.settings.source)
        self.bs = bs.Bs()
        self.persons = person.Persons(self.settings.movementMaximum, self.settings.movementMinimum, self.settings.movementTime)
        self.start = time.time()
```

```
            self.webservice = webservice.Webservice(self.settings.location,
self.settings.phone)
            self.errorcount = 0
            self.alertLog = []
            self.frameCount = 1


        def nextFrame(self):
            grabbed, self.frame = self.camera.read()
            if not grabbed: # eof
                self.destroyNow()
            self.convertFrame()


        def showFrame(self):
            if self.settings.debug:
                cv2.imshow("Thresh", self.thresh)
                #if self.settings.bsMethod == 1:
                    #cv2.imshow("backgroundFrame",
self.backgroundFrame)
                    #cv2.imshow("frameDelta", self.frameDelta)
            cv2.imshow("Feed", self.frame)



        def destroyNow(self):
            self.camera.release()
            cv2.destroyAllWindows()


        def testDestroy(self):
```

```
            key = cv2.waitKey(1) & 0xFF
            if key == ord("q"):
                    self.destroyNow()
                    return 1
            else:
                    return 0


    def resetBackgroundFrame(self):
            grabbed, self.frame = self.camera.read()
            self.convertFrame()
            self.bs.resetBackgroundIfNeeded(self.frame)
            self.persons = person.Persons(self.settings.movementMaximum,
self.settings.movementMinimum, self.settings.movementTime)
            #self.frameCount = 1
            #print 'resetbackgroundFrame'


    def testBackgroundFrame(self):
            key = cv2.waitKey(1) & 0xFF
            if key == ord("n"):
                    self.bs.deleteBackground()
            #self.resetBackgroundFrame()


    def updateBackground(self):
            self.bs.updateBackground(self.frame)


    def testSettings(self):
            key = cv2.waitKey(1) & 0xFF
```

```
            if key == ord("0"):
                self.settings.minArea += 50
                print "minArea: " , self.settings.minArea
            if key == ord("9"):
                self.settings.minArea -= 50
                print "minArea: " , self.settings.minArea
            if key == ord("8"):
                self.settings.dilationPixels += 1
                print "dilationPixels: " , self.settings.dilationPixels
            if key == ord("7"):
                self.settings.dilationPixels -= 1
                print "dilationPixels: " , self.settings.dilationPixels
            if key == ord("6"):
                self.settings.thresholdLimit += 1
                print "thresholdLimit: " , self.settings.thresholdLimit
            if key == ord("5"):
                self.settings.thresholdLimit -= 1
                print "thresholdLimit: " , self.settings.thresholdLimit
            if key == ord("4"):
                self.settings.movementMaximum += 1
                print "movementMaximum: " ,
self.settings.movementMaximum
            if key == ord("3"):
                self.settings.movementMaximum -= 1
                print "movementMaximum: " ,
self.settings.movementMaximum
            if key == ord("2"):
```

```
                self.settings.movementMinimum += 1
                print "movementMinimum: " ,
self.settings.movementMinimum
        if key == ord("1"):
                self.settings.movementMinimum  -= 1
                print "movementMinimum: " ,
self.settings.movementMinimum
        if key == ord("o"):
                if self.settings.useGaussian:
                        self.settings.useGaussian = 0
                        print "useGaussian: off"
                        self.resetbackgroundFrame()
                else:
                        self.settings.useGaussian = 1
                        print "useGaussian: on"
                        self.resetbackgroundFrame()
        if key == ord("+"):
                self.settings.movementTime += 1
                print "movementTime: " , self.settings.movementTime
        if key == ord("p"):
                self.settings.movementTime -= 1
                print "movementTime : " , self.settings.movementTime



    def convertFrame (self):
        # resize current frame, make it gray scale and blur it
```

```
        if self.settings.useResize:
            r = 750.0 / self.frame.shape[1]
            dim = (750, int(self.frame.shape[0] * r))
            self.frame = cv2.resize(self.frame, dim, interpolation =
cv2.INTER_AREA)
        if self.settings.useBw:
            self.frame = cv2.cvtColor(self.frame,
cv2.COLOR_BGR2GRAY)
        if self.settings.useGaussian:
            self.frame = cv2.GaussianBlur(self.frame,
(self.settings.gaussianPixels, self.settings.gaussianPixels), 0)


    def compare(self):
        # difference between the current frame and backgroundFrame
        self.thresh = self.bs.compareBackground(self.frame)
        self.thresh = cv2.dilate(self.thresh, None,
iterations=self.settings.dilationPixels) # dilate thresh
        _, contours, _ = cv2.findContours(self.thresh.copy(),
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE) #find contours


        self.persons.tick()


        detectStatus = "idle"



        for contour in contours:
            if cv2.contourArea(contour) < self.settings.minArea:
                continue
```

```
                (x, y, w, h) = cv2.boundingRect(contour)

                #if self.thresh.shape[1] < w+50 and self.thresh.shape[0] <
h+50:
                        #self.newLightconditions()
                #       continue

                person = self.persons.addPerson(x, y, w, h)
                color = (0, 255, 0)
                if person.alert:
                        color = (0, 0, 255)
                        cv2.line(self.frame, (x, y), (x + w, y + h), color, 2)
                        cv2.line(self.frame, (x + w, y), (x , y + h), color, 2)
                        detectStatus = "Alarm, not moving"
                        if not person.alarmReported:
                                self.webservice.alarm("not moving", person.id)
                                person.alarmReported = 1

                cv2.rectangle(self.frame, (x, y), (x + w, y + h), color, 2)
                cv2.putText(self.frame,
"{}".format(cv2.contourArea(contour)), (x, y+h+20),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 1)
                cv2.putText(self.frame, "{} : {}".format(person.id,
person.lastmoveTime), (x, y+20), cv2.FONT_HERSHEY_SIMPLEX, 0.8,
color, 1)

        # Hud + fps
```

```
            if self.settings.debug:
                self.end = time.time()
                seconds = self.end - self.start
                fps  = round((1 / seconds), 1)
                self.start = time.time()


                cv2.putText(self.frame, "Status: {}".format(detectStatus),
(10, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 140, 255), 1)
                cv2.putText(self.frame, "FPS: {}".format(fps), (400, 20),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 140, 255), 1)


        def newLightconditions(self):
            self.errorcount += 1
            if self.errorcount > 10:
                time.sleep(1.0)
                self.resetBackgroundFrame()
                self.errorcount = 0
```

webservice.py

```
import requests

class Webservice(object):

        def __init__(self, place, phone):
            #self.url =
'http://tunn.us/tools/healthservice/add.php?place='+place+'&phone='+phone
```

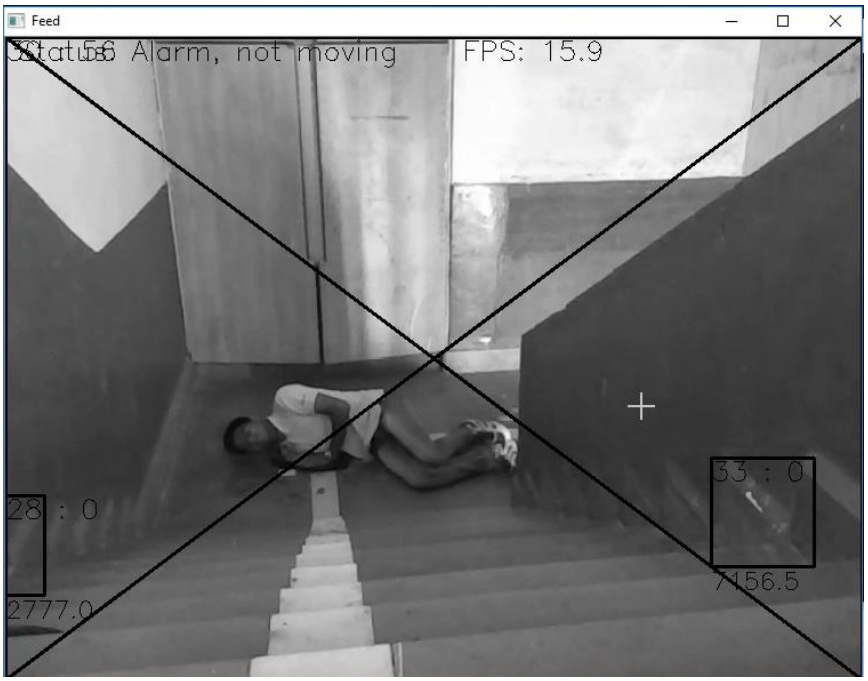
```
        self.url =
'http://salmi.pro/ject/fall/add.php?place='+place+'&phone='+phone
        self.data = ''


    def alarm(self, detectiontype, personid):
        tempurl = self.url
        tempurl =
tempurl+'&type=+'+detectiontype+'&personid='+str(personid)
        response = requests.get(tempurl, data=self.data)
        print response
```

Observation:

Standing Person (Fall not detected)

Person sitting at ground level (Fall not detected)



Person falling towards the floor and then lying unresponsive **(Fall Detected)**

Software Design:

The software is written in Python using the OpenCV library. The code is tested with Python version 2.7.11 and OpenCV version 2.4.13. Code should work in Python 2.7.x and OpenCV 2.4.x. The code is licensed under GPLv3.

main.py - is the main loop. This main loop starts the software and keeps it running while calling different functions in the video class.

video.py - Video class - utilizes the video feed. It is able to capture frames from the feed, call different background subtraction features, take keyboard inputs for changing settings (these can be found in testSettings()), raise an alarm, downscale frames for rapider future use, show frames for debugging reasons and release the camera and cleanup if quitting.

bs.py - Bs class - is handling the background subtraction and currently supports MOG2 and the dynamic approach.

person.py - Person and Persons class - is called by Video class. It includes two classes Person and Persons. Each time a frame is analyzed and a person is found

it will try to analyze if the person has been in the previous frames and is it the same person as earlier. Other features are data of how much the person moved during last frames, has the person raised an alarm and has it been sent to the webservice, where was the person in the last frame and should the person be removed (exited the scene). Person class also contains the alarm counter.

webservice.py - Webservice class - is able to send alarms to a webservice via http-requests.

settings.py - Settings class - includes all the settings that can be modified in the system. Some of these settings can be changed on the fly but some are static when the program runs. This file includes settings for the following things:

Conclusion:

We have successfully implemented a fall detection application using python, opencv and haar cascades. The application raises an alarm whenever a person falls or faints in the video feed.

Reference:

1. Docs, OpenCV. "Face Detection Using Haar Cascades." OpenCV: Face Detection Using Haar Cascades, 4 Aug. 2017, Face Detection using Haar Cascades

2. Haar Cascades
http://alereimondo.no-ip.org/OpenCV/34

3. Gonzales and Woods, "Digital Image Processing", Pearson Education, India

4. OpenCV, "Cascade Classifier Training —OpenCV 2.4.9.0 documentation," [Online].:-
http://docs.opencv.org/doc/user_guide/ug_traincascade.html.