Security is handled by the Pi Camera and OpenCV.OpenCV helps us to detect motion in the video feed. This can be done in 2 different ways

* Background subtraction Technique

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 (object recognition etc.). Generally an image's regions of interest are objects (humans, cars, text etc.) in its foregroundWe’ll make the assumption that the first frame of the video stream contains no motion and is a good Example of what our background looks like. Now that we have our background modelled viathe firstFrame variable, we can utilize it to compute the difference between the initial frame and subsequent new frames from the video stream. Computing the difference between two frames is a simple Subtraction, where we take the absolute value of their corresponding pixel intensity differences.

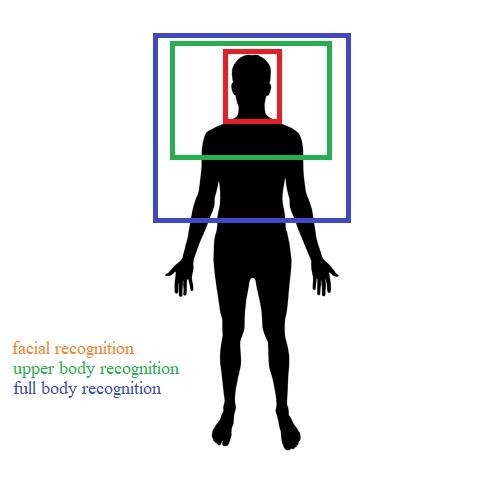
* Differential Images Technique

We will take and compare 3 different picture frames of the video feed. All the frames must be within a certain time interval from each other. Then we need to compare those images to see that there is some motion in the video or not. Differential techniques that reduces redundancy prior to efficient variable length encoding and the exploitation of statistical image characterisation, are clearly attractive. Both simple running and more complex predictive differentiation have found commercial use.

In our project we have also run our motion sensing code on the different Human body recognition models i.e. object detectors and compared those results to find out which model works best with rudimentary camera sensor like the one we have got in our project.

The Human Body Recognition models that we have used are:

* + Facial Recognition Model
  + Full Body Recognition Model
  + Upper Body Recognition Model



**Figure 8: Human body showing silhouette for different body recognition**

To make this part of the project we need to follow the following steps.

Step 1: At first we need to install OpenCV and imutils (image utilities) in our system.

**Code:** sudo apt-get install python-opencv

**Code:** sudo apt-get install imutils

Step 2: We need to check that the camera module is working or not by running this code in terminal Code: raspistill –o cam.jpg

This will create a image file named cam.jpg in root folder of PI. If image file is not created then there must be one of two errors

* + Camera not connected properly
  + CSI port not enabled

Step 3: Write the python codes and save the files in your root folder:

**Code: main.py**

import cv2 import sys

from mail import sendEmail

from flask import Flask, render\_template, Response

from camera import VideoCamera

import time import threading

email\_update\_interval = 600 # sends an email only once in this time interval video\_camera = VideoCamera(flip=True) # creates a camera object, flip vertically

object\_classifier = cv2.CascadeClassifier("models/fullbody\_recognition\_model.xml") # an opencv classifier

# App Globals (do not edit) app = Flask(\_\_name\_\_) last\_epoch = 0

defcheck\_for\_objects():

globallast\_epoch while True: try:

frame, found\_obj = video\_camera.get\_object(object\_classifier) if found\_obj and (time.time() - last\_epoch) >email\_update\_interval: last\_epoch = time.time() print "Sending email..." sendEmail(frame) print "done!"

print "group12 code!"

except: print "Error sending email: ", sys.exc\_info()[0]

@app.route('/') def index():

returnrender\_template('index.html')

def gen(camera): while True: frame = camera.get\_frame() yield (b'--frame\r\n'

b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')

@app.route('/video\_feed') defvideo\_feed():

return Response(gen(video\_camera),

mimetype='multipart/x-mixed-replace; boundary=frame')

if \_\_name\_\_ == '\_\_main\_\_':

t = threading.Thread(target=check\_for\_objects, args=())

t.daemon = True

t.start()

app.run(host='0.0.0.0', debug=False)

**Code: mail.py**

importsmtplib from email.MIMEMultipart import MIMEMultipart from email.MIMEText import MIMEText from email.MIMEImage import MIMEImage

# Email you want to send the update from (only works with gmail) fromEmail = 'group12tict@gmail.com'

# You can generate an app password here to avoid storing your password in plain text

# https://support.google.com/accounts/answer/185833?hl=en fromEmailPassword = 'group12tict'

# Email you want to send the update to toEmail = 'group12tict@gmail.com'

defsendEmail(image):

msgRoot = MIMEMultipart('related') msgRoot['Subject'] = 'Security Update' msgRoot['From'] = fromEmail msgRoot['To'] = toEmail

msgRoot.preamble = 'Raspberry pi security camera update'

msgAlternative = MIMEMultipart('alternative') msgRoot.attach(msgAlternative)

msgText = MIMEText('Smart security cam found object') msgAlternative.attach(msgText)

msgText = MIMEText('<imgsrc="cid:image1">', 'html') msgAlternative.attach(msgText)

msgImage = MIMEImage(image) msgImage.add\_header('Content-ID', '<image1>')

msgRoot.attach(msgImage)

smtp = smtplib.SMTP('smtp.gmail.com', 587) smtp.starttls()

smtp.login(fromEmail, fromEmailPassword) smtp.sendmail(fromEmail, toEmail, msgRoot.as\_string()) smtp.quit()

**Code: camera.py**

import cv2

fromimutils.video.pivideostream import PiVideoStream import imutils

import time import numpy as np

classVideoCamera(object): def \_\_init\_\_(self, flip = False): self.vs = PiVideoStream().start()

self.flip = flip

time.sleep(2.0)

def \_\_del\_\_(self): self.vs.stop()

defflip\_if\_needed(self, frame): if self.flip:

returnnp.flip(frame, 0) return frame

defget\_frame(self):

frame = self.flip\_if\_needed(self.vs.read()) ret, jpeg = cv2.imencode('.jpg', frame) return jpeg.tobytes()

defget\_object(self, classifier):

found\_objects = False

frame = self.flip\_if\_needed(self.vs.read()).copy() gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

objects = classifier.detectMultiScale( gray, scaleFactor=1.1, minNeighbors=5,

minSize=(30, 30),

flags=cv2.CASCADE\_SCALE\_IMAGE

)

iflen(objects) > 0: found\_objects = True

# Draw a rectangle around the objects for (x, y, w, h) in objects:

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

ret, jpeg = cv2.imencode('.jpg', frame)

return (jpeg.tobytes(), found\_objects)

Step 4: Run the python code’s main file by by running this code in terminal

**Code:** python main.py

In the Security part the desired outputs must be clearly showing the target human and making a boundary surrounding the desired part of the target as described in the object recognition model used during the test.



**Figure 9: Camera Module shows TargetHuman**

:

There can be some random objects placed right in front of the target or in the frame. This was done to check how the code works and mimics a normal household i.e. indoor situation where there might be some objects like jars placed in random locations.

The lighting situation was simple fluorescent tube and pictures were taken sometimes by keeping the target more towards light and in other times the target was kept relatively shadowed.



**Figure 10: Full body Recognition with camera module inversed**

In the picture above we have tested the using full body recognition model and the outcome was such that it will work even if the camera module is inversed.

In some cases we also checked the code by placing objects that has a shape that might resemble the silhouette of a human figure.

We also checked code by moving objects like books or paper balls to mimic birds or flies and as we expected there was no output for that.



**Figure 11: Upper Body Recognition Model**

We found that the upper body recognition model works best in indoor situation. The above picture is a example of upper body recognition model.

Limitations found in the Security part i.e. object detector modules are as follows:

* Upper Body and Full Body recognition model deal with frontal or backside view on a human target but lacks the sophistication of detecting side views on a Human.
* We noticed that successful detections containing the target do not sit tightly on the body but also include some of the background left and right. On further searching on this issue we found that this is not a bug but accurately reflects the employed training data of OpenCV which also includes portions of the background to ensure proper silhouette representation.
* We did not get good result with facial recognition model in indoor situation may be because we were using a mediocre camera module and operating on very low resolution i.e. 1920\*1080 to be exact. In such low resolution and indoor light conditions the performance of facial recognition model was very bad. But according to the internet in other scenarios that are not fitted for our project, many people find that facial recognition model is probably the best.
* A frontal face as a pattern is pretty distinct with respect to other patterns. This is not so for upper and especially full bodies, because they have to rely on fragile silhouette information rather than internal (facial) features. Still, we found especially the upper body detector to perform amazingly well. In contrast to a face detector these detectors will also work at very low image resolutions