Raspberry Pi Security Camera Report

# Introduction

The goal of this project is to make a Security Camera that can warn you by message. For this, we will use a Raspberry Pi and a camera module.

The software will be programmed in python.

We will need to:

1. Capture the video feed
2. Compare images to see if something moved
3. Send an alert if test is positive

# Packages Used

OpenCV (package: cv2)

It is a library that is mostly used for visual computing and machine learning.  
It handles the acquisition of images, or video feed, and their manipulation/transformation.

<https://opencv-python-tutroals.readthedocs.io/>

Scikit-image (package: skimage)

This is a collection of algorithms for image processing and analysis.

<https://scikit-image.org/>

Twilio (package: twilio)

This is a platform that provide multiple communication services, such as messaging for marketing or authentication.

<https://www.twilio.com/>

# Actual Program

This is commented enough so I don’t have to explain the whole procedure here.

Have fun reading & learning.

# -\*- coding: utf-8 -\*-

"""

Created on Thu Feb  6 22:57:34 2020

@author: Valentin

"""

# --- LIBRARIES ---

#OpenCV, for video feed

import cv2

#scikit-image, for comparison, SSIM algorithm

from skimage.metrics import structural\_similarity

# Twilio, for sending messages, like SMS

from twilio.rest import Client

#import constant values, such as Twilio credentials and target phone

from env import TWILIO\_ACCOUNT\_SID,TWILIO\_AUTH\_TOKEN,TWILIO\_PHONE,PHONE\_NUMBER

# --- VARS ---

#Threshold, if the error is under it, we trigger an alert

#1 is perfect match between 2 images

#Found that we had to lower it to deal with noise and micro movements

similarityValue = 0.90

#Difference between highest and lowest value of picture

#min=0, max = 255

data\_range = 255

# --- FUNCTIONS ---

#return error between 2 images/matrix, identical if 1 is returned

def ssim(A,B):

    return structural\_similarity(A, B, data\_range=data\_range)

# --- MAIN ---

#Twilio client initialisation

client = Client(TWILIO\_ACCOUNT\_SID,TWILIO\_AUTH\_TOKEN)

#Capture the video flux with openCV (We can change 0 to 1,2,... if multiple cameras)

video\_capture = cv2.VideoCapture(0)

# First, we need to initialise the stored frame

# right now it's empty so we have nothing to compare to

# Get the actual frame

\_,buffer\_frame = video\_capture.read()

#Transform picture to GrayScale, easier to compare

buffer\_frame = cv2.cvtColor(buffer\_frame, cv2.COLOR\_BGR2GRAY) #Gray

#Store actual frame, so now we can start comparing

previous\_frame = buffer\_frame

#We do not need to compare all the frames

#let's compare every 10 frames (Approx every ~0.4s for a 24fps camera)

comparison\_frequency = 10

#Counter for video frames, we already got 1 image, so init at 1

image\_counter = 1

#Counter for printing events with number

event\_counter = 0

#Boolean so we don't spam the phone at each event, we send the message only once

#TODO: Add a reset after a certain time

is\_first\_message = True

#Main loop, runinng unless of CTRL+C or keyboard input

while True:

    try:

        #Read the next frame into buffer

        \_,buffer\_frame = video\_capture.read() # First frame

        #If we don't have images anymore, exit

        if buffer\_frame is None:

            break

        #Switch to gray

        buffer\_frame = cv2.cvtColor(buffer\_frame, cv2.COLOR\_BGR2GRAY)

        #Compare every 10 frames

        if image\_counter == comparison\_frequency:

            #Get error value

            ssim\_index = ssim(previous\_frame, buffer\_frame)

            #If error under threshold, trigger warning

            if ssim\_index < similarityValue:

                event\_counter +=1

                print('Event'+str(event\_counter)+' : Intruder Pepper Spray!')

                #Send message if first warning

                if (is\_first\_message):

                    #send text message

                    #client.messages.create(body='Intruder Alert!',from\_=TWILIO\_PHONE,to=PHONE\_NUMBER)

                    is\_first\_message = False

            #Store current frame as new reference for comparison

            previous\_frame = buffer\_frame

            #Display captured frame, so we can see what is happening

            cv2.imshow('Raspi Security Camera', buffer\_frame)

            cv2.waitKey(1) # Image not displayed if line removed

            #if comparison happened, reset frame counter

            image\_counter=0

        #Increment at each loop / captured frame

        image\_counter+=1

    #Avoid error when stopping program

    except KeyboardInterrupt:

        print('All done')

        break

#Stop video feed and close window

video\_capture.release()

cv2.destroyAllWindows()

# (Bonus) Choice of algorithm

To compare 2 images and see if they are identical, we need to use an algorithm. For this example, let’s say our test images are represented as matrix of pixels.

Their individual values are between 0 to 255, 0 is black and 255 is white.

I’ll try to explain it as simple as possible, without going too deep in the mathematics.

## First choice, Element sum

We can sum all the pixels in the matrix/pictures, and then compare the results. If results are identical, the images are.

Sum1 = 5, Sum1 = 5, identical images. Success

A B

|  |  |  |
| --- | --- | --- |
| 1 | 0 | 1 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |

|  |  |  |
| --- | --- | --- |
| 1 | 0 | 1 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |

Issue:

Here the two pictures are different, but the results are the same (5). So, we get a “success” flag, but shouldn’t because the individual pixels are different inside the matrix/images.

A B

|  |  |  |
| --- | --- | --- |
| 1 | 0 | 1 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 0 | 1 |

Next, there’s an alternative solution.

## Second, L1 norm between M1 and M2

Let’s take all pixels one by one, and subtract them (A00 – B00, A01-B01,…, Aij - Bij), then we return the result in a third matrix.

If the sum of the result matrix is null, we can say that both images are identical.

A B A B

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

|  |  |  |
| --- | --- | --- |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

Result Result

|  |  |  |
| --- | --- | --- |
| 0 | -1 | 0 |
| -1 | 0 | -1 |
| -1 | -1 | 0 |

Sum = 0, images identicals. Sum != 0, images different

We can see that we can have some issues, if one error is -1 and another is 1, we get a result as “identical”, even if the two images are not.

## Sum of squared error (SSE)

Let’s take the precedent example, but instead of subtracting each individual value, we add an extra step: we get the absolute value or square that result.

We can then get rid of negative error.

For this algorithm, squared is preferred to absolute value. There are some benefits, like the fact that differences are exaggerated, and others mathematical properties but I won’t go in details as we won’t really need it now.

Though, here we have an issue with image size.

Two 3x3 images/matrix, respectively filled with 0 (black) and 1 (white), won’t have the same error as a 100x100 matrix.

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

A (3x3) B (3x3)

Here, we get 9 as a result.

|  |  |  |
| --- | --- | --- |
| 1 | … | … |
| … | … | … |
| … | … | 1 |

|  |  |  |
| --- | --- | --- |
| 0 | … | … |
| … | … | … |
| … | … | 0 |

A (100x100) B (100x100)

Here we get 100. We can see that images in both tests are still identical, but the results are showing different output. This is not great for consistency in different environments.

## Mean Squared Error (MSE)

We take the same images as in the previous test.

We still use SSE with extra step: we average the result by number of values/pixels.

Now, in case of the previous 3x3 and 100x100 matrix comparison, we obtain the same result: 1.

The value now returned is now between 0 and 1, easier.

Still issues? Arf….

I promise that’s the last one.

## Structural Similarity SSIM

MSE go through each pixel one by one to analyse the picture. That’s not really handy in some specific scenarios.

Let’s say all the pixels in the second image shift by 1 pixel to the right. The MSE algorithm says that we have two totally different images.

Here we looks at groups of pixels, comparing luminance, contrast and structure using statistics.

It’s a more human-like analysis of the picture, how we see it from our perspective and identify differences.

Here, the error is <= 1, and 1 equals perfect match).

# Conclusion

This project has been fun and interesting for the learning part.  
It’s also been a great introduction to computer vision as I didn’t have any experience in it.

Some part of the projects can still be improved, such as multiple ways to contact customer, or some face detecting algorithms.

//TODO: Complete document