COEN 243 (Internet of Things) Project Report

Pok Lai Alexander Law
Student
Santa Clara University
Santa Clara, California
Plaw@scu.edu

Jingqi Elita Liu
Student
Santa Clara University
Santa Clara, California
Jliu12@scu.edu

***Abstract*— A front-door security camera costs us a lot of money, both in installation and maintenance cost. A Raspberry Pi operating system was installed and configured to allow remote access and allow for the camera to be used. The camera is set up, so it can see outside the front door. The program takes two photos in a loop and constantly checks the difference in pixels between a configured version of the two images. If the difference in pixel colors passes a certain threshold, the camera is deemed to have seen a change in the door. A video file will be generated in addition to the images. A text message will be sent to the cell phone user to notify them someone has been at the door, and a face image of this person will be uploaded to AWS for the user to manually verify the suspected entry.**

I. Setup

1. The first step is to set up the Raspberry Pi board by installing all the equipment pieces such as the heat sink and the camera module [1]. Figure 1 and 2 below shows the Raspberry Pi board and how it was installed:

Figure 1: A Raspberry Pi with a camera model attached

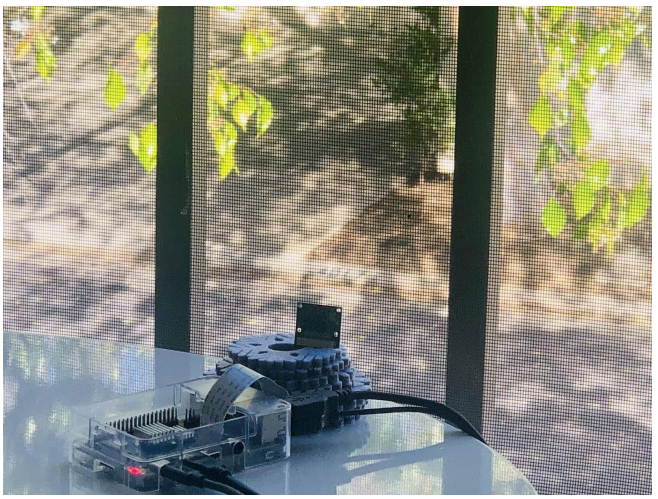

Figure 2: The Camera Facing the front of the house

The device specifications are as follows:

- Raspberry Pi 4 Model B
- 1.5GHz 64-bit quad-core CPU (2GB RAM)
- Camera Module
- Sensor type: Mini Vision OV5647 Color CMOS QSXGA

The Raspberry Pi operating system is then downloaded from the official website and saved to an SD card [2]. The SD card is placed into the Raspberry Pi and is run for the first time. Next, the VNC viewer was installed so the Raspberry Pi can be run remotely on the desktop. Anaconda, Anaconda Prompt, and Filezilla are also installed. The Raspberry Pi settings are set to open to SSH file transfer and VNC remote desktop. The IP address is noted so files can be transferred via Filezilla and the remote desktop can work via VNC. Finally, the camera is set to face the outside of the home to detect changes in movement.

II. Taking Photos

1.

Given the limited capacity of the Raspberry Pi (2GB of memory), space storage is essential. This means files should have a mixture between quality and space, as well as files being overwritten to prevent duplicate files [3]. The following command takes a photo:

raspistill -w 1280 -h 720 -vf -hf -o test%0d.jpg

Figure 3 below shows a sample photo being taken.

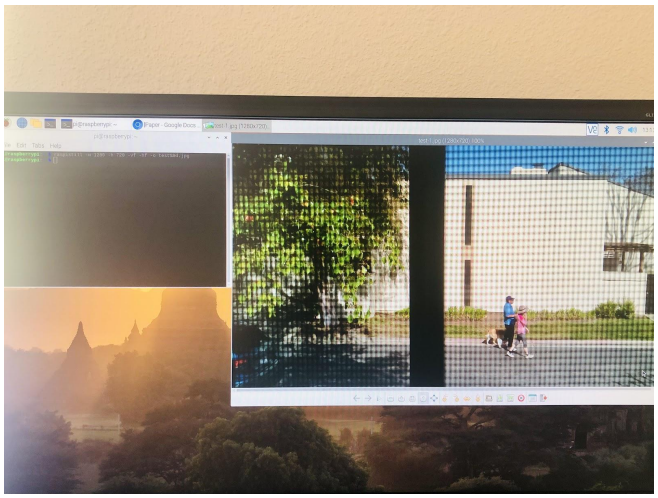

Figure 3: A sample photo being taken

In this example, the width contains 1280 pixels, and the height contains 720 pixels. Vf and hf indicate a vertical flip and horizontal flip, respectively. The file is saved as test0.jpg. The file can be read and accessed in the home folder of the Raspberry Pi. The file can be read by the following function:

test1=cv2.imread("test.jpg")

III. Configuring Images

Now, we need to set up our Raspberry Pi with OpenCV - a computer vision library [4] - to enable the image recognition features, by installing the opencv-python package:

pip3 install opencv-python

The image is first configured using taking a specific area of the photos, which are more likely to be entered to save space. In this case, we are taking the area pavement from the door to the sidewalk. The following function locates the masking points, which we can use to determine our boundary.

points=cv2.selectROI(img,False)

print(points)

The points that are selected are saved as an array. A mask of zeros are created and used to combine with the main image to filter out the points using a bitwise AND so that we can save space. The following functions show the process below.

mask=np.zeros((img.shape[0],img.shape[1]),dtype="uint8")

pts=np.array([[1100,500],[1100,700],[100,700],[100,500]],dtype=np.int32)

cv2.fillConvexPoly(mask,pts,255)

masked=cv2.bitwise_and(img,img,mask=mask)

Figure 4 and 5 below show a bitwise masked photo.

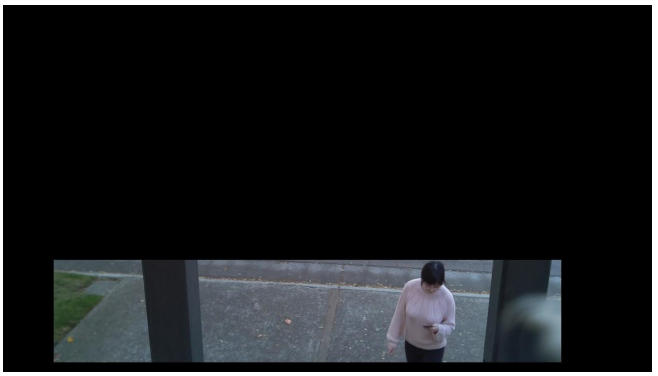

Figure 4: First Bit-Mask Image

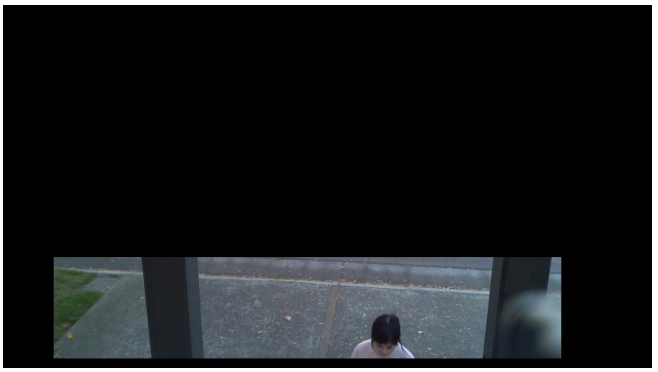

Figure 5: Second Bit-Mask Image

After this is done, the file can be resized to a smaller scale to further save space. The masked image is then converted to gray and a Gaussian Blur is conducted to reduce the impact of noise. The following code snippet does this.

gray=imutils.resize(masked,width=200)

gray=cv2.cvtColor(gray,cv2.COLOR_BGR2GRAY)

gray=cv2.GaussianBlur(gray,(11,11),0)

Figures 6 and 7 below show the images which are gray-scaled and with the Gaussian Blur applied to them.

Figure 6: First Bit-Mask with Gaussian Blur

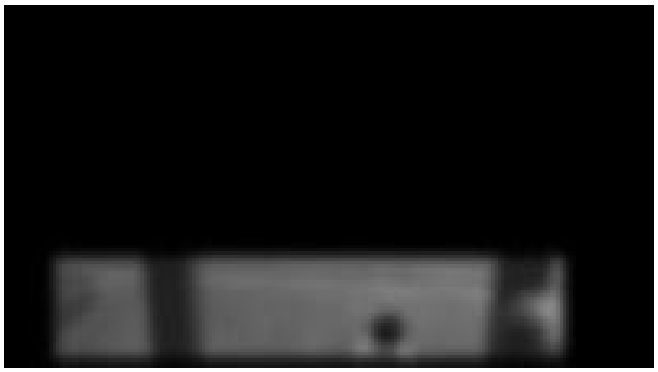

Figure 7: Second Bit-Mask with Gaussian Blur

After these steps, the image that we want to use is ready to be compared.

IV. Main Loop

The main loop uses a while loop with a counter which ends when the user decides to stop running the program. A counter is used to check the number of times pictures are taken. Two images are taken in a cycle and saved in two locations. Between the first and second image, there is a time delay (tl) of 1000ms, and there is another 1000ms from the second image before another cycle begins [5]. The following code snippet shows the above process.

```
counter= 0
while True:
counter=counter+1
command='raspistill -w 1280 -h 720 -t 1000 -tl 1000 -o test%0d.jpg'
os.system(command)
```

Figure 8 below shows the detector loop.

Figure 8: Main Detector Loop

The next step is to set up an array of differences called a detector. For each pixel, we compare the first and second images. If the difference between the pixels surpasses a certain threshold (in this case set to be 50), we will assume the pixel is different and set a value of 255 in the detector array. The sum of the detector array is placed into a variable called detector_total.

```
pixel_threshold=50
detector_total=np.uint64(0)

detector=np.zeros((gray2.shape[0],gray2.shape[1]),dtype="uint8")
for i in range(0,gray2.shape[0]):
for j in range(0,gray2.shape[1]):
if abs(int(gray2[i,j])-int(gray1[i,j]))>pixel_threshold:
detector[i,j]=255
detector_total = np.uint64(np.sum(detector))
print("detector_total= ", detector_total)
print(" ")
```

Finally, the detector_total variable is compared with a threshold value, and if the threshold value is surpassed, the program will alert us that there is enough difference between the two images such that someone/something has entered the walkway. A 20-second video is recorded automatically in this case. This is converted into a .mp4 format.

```
if detector_total>10000:
command2='raspivid -t 2000 -w 1280 -h 720 -fps 30 -o ' + timestr + '.h264'
os.system(command2)
command3='MP4Box -fps 30 -add ' + timestr + '.h264 ' + timestr +'.mp4'
os.system(command3)
```

Figure 9 below shows the video is saved in the home folder.

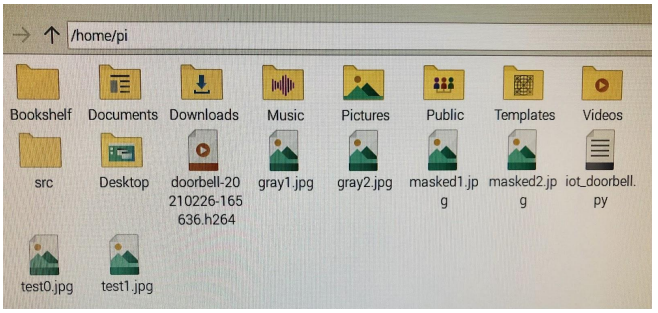

Figure 9: Videos being Uploaded

V. Amazon Web Services Video and Image Upload

Amazon Web Services (AWS) provides pay-as-you-go cloud computing platforms and APIs to individuals and companies, including object storage services such as AWS S3. To connect with AWS server. First, we registered an

AWS account, signed into the AWS S3 service, and created a bucket [6]. Figure 10 below shows this.

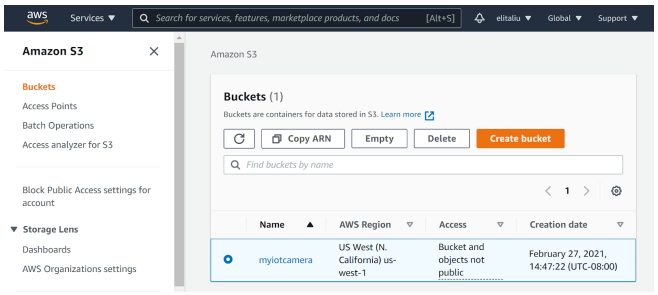

Figure 10: AWS Bucket Creation

Second, we installed the boto3 package on our Raspberry Pi, using the command line:

Pip3 install boto3

Third, we wrote a new function in our program by using the AWS boto3 API, for uploading a picture of the front-door person when triggered:

```
def upload_to_aws (local_file_location, aws_bucket, aws_s3file):
    s3 = boto3.client('s3', aws_access_key_id = 'access_key', aws_secret_access_key = 'secret_access_key')
    try:
      s3.upload_file(local_file_location,
                     aws_bucket, aws_s3file)
      print("Success!")
      return True
```

Figure 11 below the videos uploaded to AWS.

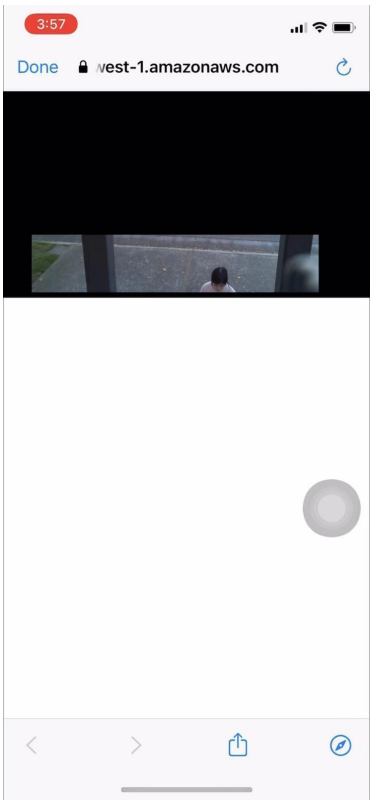

Figure 11: AWS Image Upload

VI. TWILIO UPLOAD

Twilio is a method to send text messages to groups automatically [7]. To install the Twilio package on our Raspberry Pi, we used the command line:

pip3 install twilio

After an account is created, a phone number, the access token is generated. Similarly, the phone number, access token, and the user's phone number are used to connect the Raspberry Pi to Twilio. A text message to the user's phone number is sent from the Twilio number to notify the user immediately there is a possible intruder. The user can then go to the AWS app and check out the captured image for confirmation. The following code shows the uploading process in a dynamic fashion.

```
account_sid="xxx"
auth_token="xxx"
client=Client(account_sid,auth_token
message=client.api.account.messages.create(
        to="+xxx",
        from_="+xxx",
        body="Smart Doorbell Detection")
```

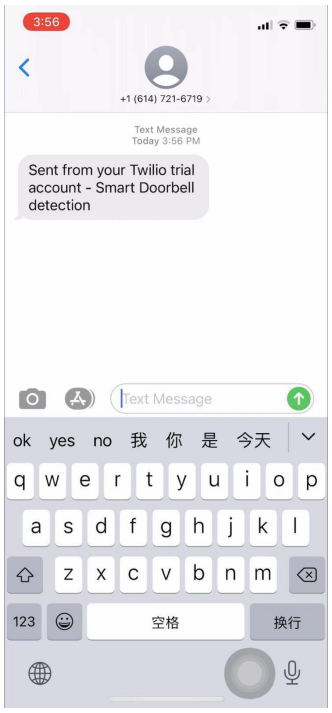

Figure 12: Twilio Upload

REFERENCES

[1] Kumar, KN Karthick, H. Natraj, and T. Prem Jacob. "Motion activated security camera using Raspberry Pi." *2017 International Conference on Communication and Signal Processing (ICCSP)*. IEEE, 2017.

[2] Yamato, Yoji, Yoshifumi Fukumoto, and Hiroki Kumazaki. "Security camera movie and ERP data matching system to prevent theft." *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2017.

[3] Zhao, Cheah Wai, Jayanand Jegatheesan, and Son Chee Loon. "Exploring iot application using raspberry pi." *International Journal of Computer Networks and Applications* 2.1 (2015): 27-34.

[4] Brock, J. Dean, Rebecca F. Bruce, and Marietta E. Cameron. "Changing the world with a Raspberry Pi." *Journal of Computing Sciences in Colleges* 29.2 (2013): 151-153.

[5] Kulkarni, Prachi H., Pratik D. Kute, and V. N. More. "IoT based data processing for automated industrial meter reader using Raspberry Pi." *2016 International Conference on Internet of Things and Applications (IOTA)*. IEEE, 2016.

[6] Muller, Lisa, et al. "A Traffic Analysis on Serverless Computing Based on the Example of a File Upload Stream on AWS Lambda." *Big Data and Cognitive Computing* 4.4 (2020): 38.

[7] Venkatesan, Supreeta, et al. "Design and implementation of an automated security system using Twilio messaging service." *2017 International Conference on Smart Cities, Automation & Intelligent Computing Systems (ICON-SONICS)*. IEEE, 2017.