Security Camera

By

Student: DO VIET HOANG
Student ID: 10329935
Tutor: Anisur Rahman
Class: Thu 2pm – 4pm GP-S635

STATEMENT OF CONTRIBUTION

By completing this form, we agree to the following:

1. We declare that all of the work submitted for this assignment is our own original work except for material that is explicitly referenced and for which we have permission, or which is freely available (and also referenced).

2. We agree that QUT may archive this assignment for an indefinite period of time, and use it in the future for educational purposes including, but not limited to: as an example of previous work; as the basis for assignments, lectures or tutorials; for comparison when scanning for plagiarism, etc.

3. We agree to indemnify QUT and hold it blameless if copyright infringements are found in this work and the copyright owner takes action against QUT that is not covered by the normal terms of Educational Use.

Student's name	Do Viet Hoang
Contribution	100%
Signature	Hoang

PROJECT OBJECTIVES

- The project is an internet-connected security camera utilizing Raspberry Pi and Pi camera to detect human motion, stream live video over network and warn users about security risks via email.

- The implementations of the project are various such as capturing video of sneaky fridge thieves, tracking your pets or simply observing your own room and detecting suspicious activities.

TECHNOLOGIES: REVIEW AND DISCUSSION

1. Python

The security camera is a Python project. Python programming language is used in this project because it is simple and easy to learn syntax for the beginners. Moreover, it is also a high-level programming language that I have basic background and has been taught in one of my subjects at QUT. Since Python utilizes Interpretation to execute and compile code, it can fasten the edit-test-debug cycle significantly. Python is a free software and pre-installed in every Linux distribution including Raspbian Stretch – a computer operating system for Raspberry Pi built on top of Linux Kernel.

2. Python libraries

a. OpenCV 3.3

OpenCV (Open source computer vision) is one of the most optimal tools supporting real-time computer vision. It includes over 2500 optimized algorithms (both computer vision and machine learning algorithms) to help developers in various tasks related to processing images such as detect objects, track movement, enhance image resolution, etc (OpenCV, n.d.).

The reasons why I implement OpenCV in my project are not only because of the great number of useful built-in modules it provides but also there is a huge user community where I can easily seek help from professionals and experienced developer. OpenCV has Python interfaces and supports Raspbian. Even though the installation of OpenCV 3 in Raspberry Pi is quite challenging and time-consuming, it works perfectly in Raspbian environment and yields desirable outcome.

My project is building a security camera, which mainly focuses on processing images. In more detail, it will accumulate the weighted average between the current frame and previous frames, then compute the difference between the current frame and average frame to produce a frame called delta. Then, my program will threshold the delta image to find 'motion' region in the video stream. By offering many convenience functions such as cv2.accumulateWeighted( ), cv2.absdiff( ), cv2.threshold( ), cv2.dilate( ) …, OpenCV minimized the workload to several lines of code.

b. Imutils

Imutils is a set of handy functions to support operating image processing such as translation, resizing, rotation, skeletonization and displaying images with OpenCV and Python (jrosebr1, n.d.). In my project, Imutils is utilized to resize the frames by imutils.resize(), and find contours of motion area and grab them by imutils.grab_contours( ).

c. Flask

Flask is a microframework written in Python. It offers a set of libraries and technologies that facilitate the building of a web application such as web pages, a blog, a wiki, etc.

My security camera project implements Flask web server in streaming live video. The video is rendered by Flask and displayed in a web page based on a HTML template. Thanks to Flask, it turns my project into a cloud technology that allows users to access and connect at any time, in anywhere and provides greater security. Flask does not stream a video, instead, it transfers a sequence of independent JPEG pictures, which is also called Motion JPEG. The benefit of this method is that it can reduce the latency when transmitting data over network. However, the quality of images is not at its best since JPEG format is not optimized for motion video.

3. HTML

HTML is the standard markup language, which is used to build the Web. In my application, I used HTML and some of its elements that I have learnt at QUT to define the structure of my webpage. My webpage includes a title of the app, a heading and a image tag for rendering live video.

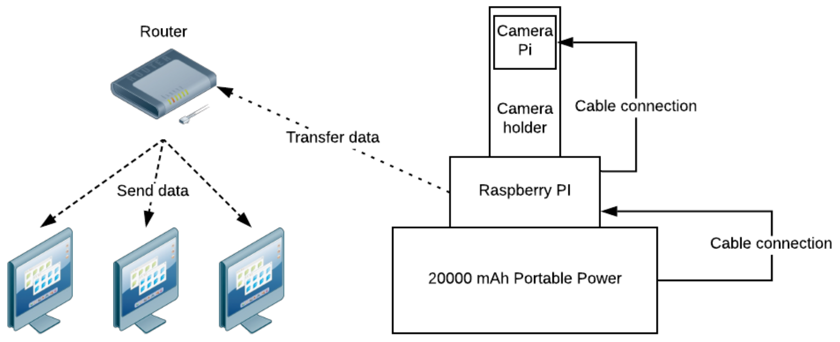
DESIGN AND IMPLEMENTATION

OVERVIEW

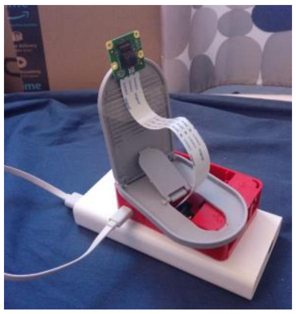
The hardware components of project include following parts:

- Raspberry Pi 3 Model B+
- Raspberry Pi Enclosure
- Pi NoIR Camera to capture images
- Power Supply to make the system portable
- MicroSD Card to store Raspbian OS
- An adjustable camera holder to change angle of camera

The diagram below shows the associations of the components:



The picture below shows how my security camera looks:



The link below is the example of output video that the security camera will send via email when suspicious activity is detected: https://youtu.be/UkQK8tghOUQ

The source code of program: https://www.dropbox.com/s/80v95bnrqwzd3hw/security_cam.py?dl=0

INSTALLING IMUTILS AND OPENCV 3 (ADRIAN ROSEBOCK, 2015)

Before starting building application, I need to install OpenCV 3 and Imutils with Python bindings on Raspbian Stretch. The setup process requires many dependencies and pre-requisites that have to be installed, therefore, it is quite time-consuming (it takes about 3 hours to install OpenCV 3 in my Raspberry 3). The first step is to update and upgrade any existing packages by following command: `$ sudo apt-get update && sudo apt-get upgrade`. Then I need to install some development tools to configure the OpenCV build process, image and video I/O packages to load various image and video files, GTK development library to compile the *highgui* module, some Python modules and dependencies to support matrix operations:

```
$ sudo apt-get install build-essential cmake pkg-config
$ sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-
$ sudo apt-get install libxvidcore-dev libx264-dev
$ sudo apt-get install libgtk2.0-dev libgtk-3-dev
$ sudo apt-get install libatlas-base-dev gfortran
$ pip install numpy imultils
```

The OpenCV source code can be downloaded directly from Github by following commands:

```
$ wget -O opencv.zip https://github.com/Itseez/opencv/archive/3.3.0.zip
$ unzip opencv.zip
$ wget -O opencv_contrib.zip
https://github.com/Itseez/opencv_contrib/archive/3.3.0.zip   $ unzip
opencv_contrib.zip
```

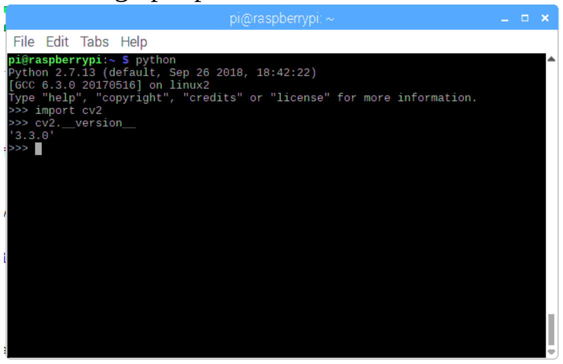
Once I have OpenCV source code, I can build OpenCV in *build* directory using CMake:

```
 $ cd ~/opencv-3.3.0/
 $ mkdir build
 $ cd build
 $ cmake -D CMAKE_BUILD_TYPE=RELEASE \
     -D CMAKE_INSTALL_PREFIX=/usr/local \
     -D INSTALL_PYTHON_EXAMPLES=ON \
     -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-3.3.0/modules \
     -D BUILD_EXAMPLES=ON ..
```

Finally, OpenCV is ready to be compiled by `$ make -j4` (this step takes up to 2 hours). Once OpenCV 3 has finished compiling, I can install it on my Raspberry Pi by:

```
$ sudo make install
$ sudo ldconfig
```

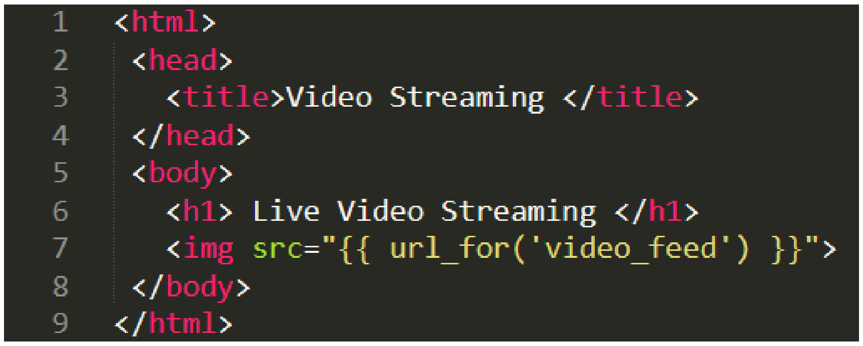
This is my result after successful setting up OpenCV:



```
pi@raspberrypi:~ $ python
Python 2.7.13 (default, Sep 26 2018, 18:42:22)
[GCC 6.3.0 20170516] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
'3.3.0'
>>> 
```

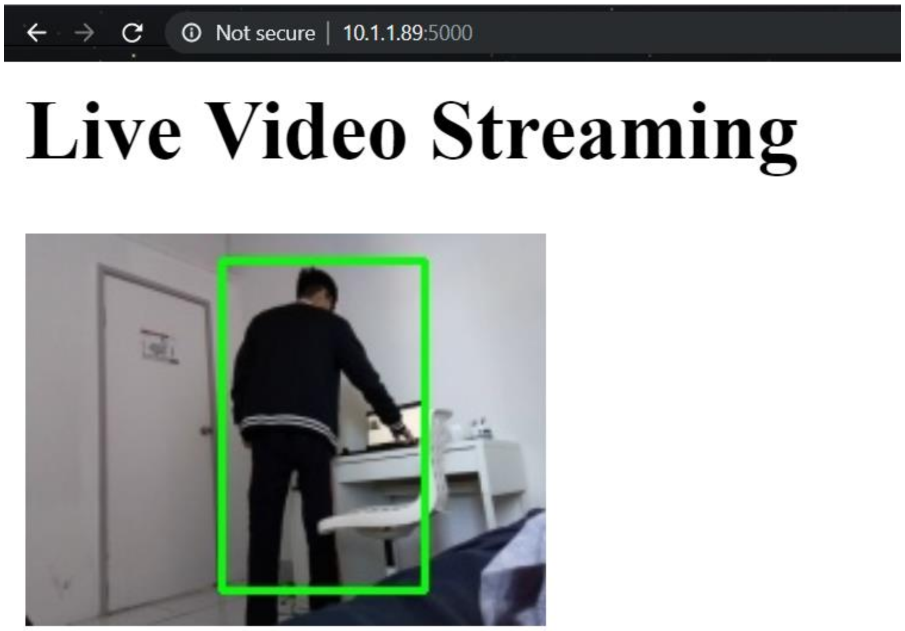
CREATE A WEB PAGE TO DISPLAY SECURITY FEED

The idea is to stream live video over the network; therefore, we need a web page that allows users to connect and see what is happening with their place. To achieve that, I use a simple HTML template showing motion security images. The HTML file, which is named *index.html*, only defines some fundamental parts of the website such as *a title* of the page, *a heading* and *motion images*:



```
<html>
 <head>
   <title>Video Streaming </title>
 </head>
 <body>
   <h1> Live Video Streaming </h1>
   <img src="{{ url_for('video_feed') }}">
 </body>
</html>
```

This is what the web page looks like:

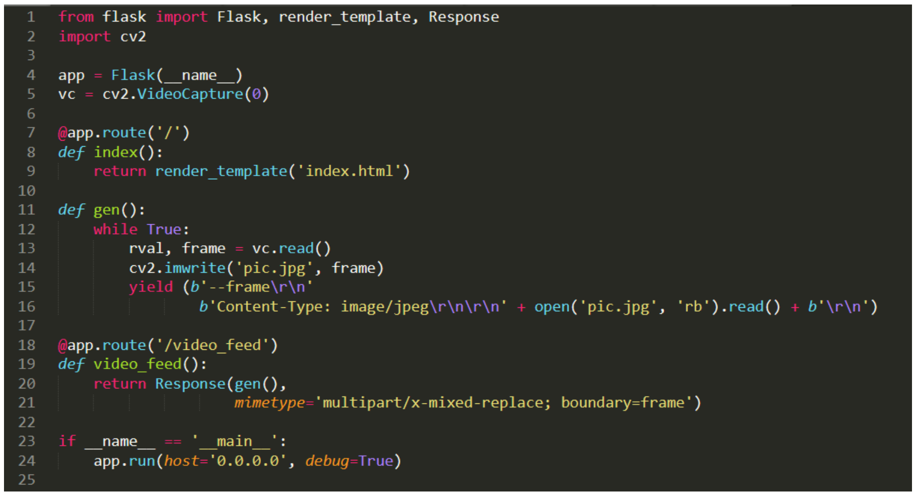


BUILDING A LIVE VIDEO STREAM SERVER USING FLASK (MIGUEL GRINBERG, 2014)

Flask can be easily installed by going to Terminal and entering:

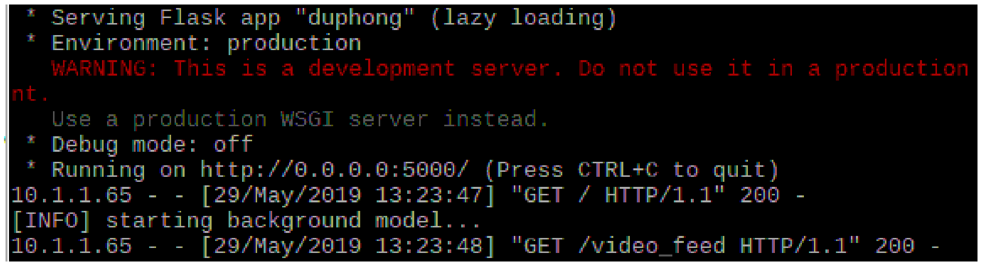
```
$ sudo apt-get install python-flask
```

Bellow you can see the lines of code to build a web application that can display Motion JPEG images:



```python
from flask import Flask, render_template, Response
import cv2

app = Flask(__name__)
vc = cv2.VideoCapture(0)

@app.route('/')
def index():
    return render_template('index.html')

def gen():
    while True:
        rval, frame = vc.read()
        cv2.imwrite('pic.jpg', frame)
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + open('pic.jpg', 'rb').read() + b'\r\n')

@app.route('/video_feed')
def video_feed():
    return Response(gen(),
                    mimetype='multipart/x-mixed-replace; boundary=frame')

if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=True)

```

The cv2.VideoCapture(0) (an OpenCV function) is responsible for obtaining a set of frames taken from Pi Camera. There are two routes in the application namely '/' and '/video_feed'. The '/' serves the main page, which is defined in the *index.html* file. The '/video_feed' route is in charge of returning the streaming response, then the src attribute of the image tag in *index.html* will include the URL to this route. The '/video_feed' route uses the generator function called gen( ), which enters a while loop where it continuously returns frames from the camera as response chunks and yields this frame with a content type of image/jpeg as shown above. This is the result when the connection is established successful and server starts streaming video over port 5000:



```
 * Serving Flask app "duphong" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production
nt.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
10.1.1.65 - - [29/May/2019 13:23:47] "GET / HTTP/1.1" 200 -
[INFO] starting background model...
10.1.1.65 - - [29/May/2019 13:23:48] "GET /video_feed HTTP/1.1" 200 -
```

PROCESSING IMAGES WITH OPENCV AND IMUTILS (ADRIAN ROSEBOCK, 2015)

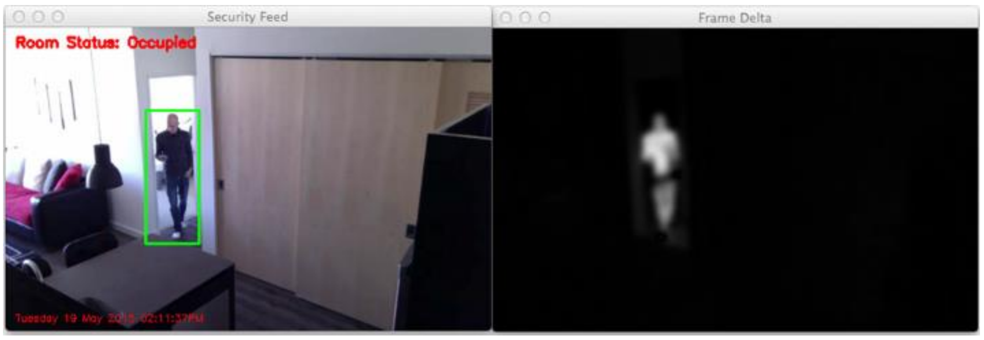
Once the frame is grabbed by Pi Camera, I resize it to have a width of 500 pixels and convert it to grayscale. The Gaussian blur method is applied here to ensure the high frequency noise would be removed and the structure of objects can be seen more clearly.

```
# resize the frame and convert the frame to grayscale
frame = imutils.resize(frame, width=500)
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray, (21, 21), 0)
```

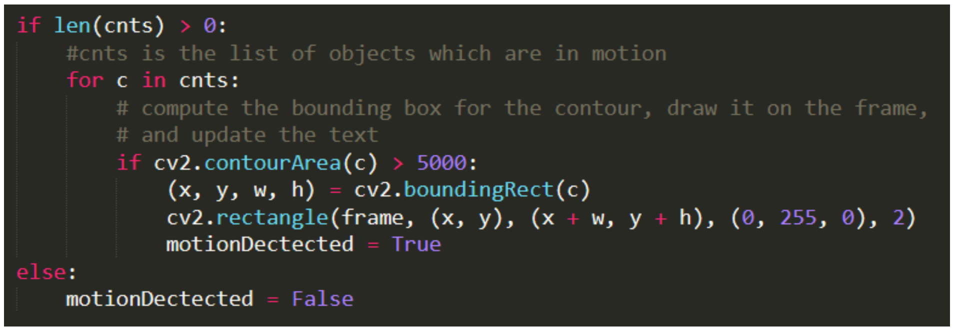
In the next stage, OpenCV will accumulate the weighted average between the current frame and previous frames, then compute the difference between the current frame and the averaged frames to produce a frame called delta. Then, my program will threshold the delta image and apply contour detection to find 'motion' region in the video stream:

```
# if the average frame is None, initialize it
if avg is None:
    print("[INFO] starting background model...")
    avg = gray.copy().astype("float")

    continue

# accumulate the weighted average between the current frame and
# previous frames, then compute the difference between the current
# frame and running average
cv2.accumulateWeighted(gray, avg, 0.5)
frameDelta = cv2.absdiff(gray, cv2.convertScaleAbs(avg))

# threshold the delta image, dilate the thresholded image to fill
# in holes, then find contours on thresholded image
thresh = cv2.threshold(frameDelta, 5, 255,
    cv2.THRESH_BINARY)[1]
thresh = cv2.dilate(thresh, None, iterations=2)
cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
```

Bellow you can see an example of the current frame with bounded 'motion' region and the delta frame:



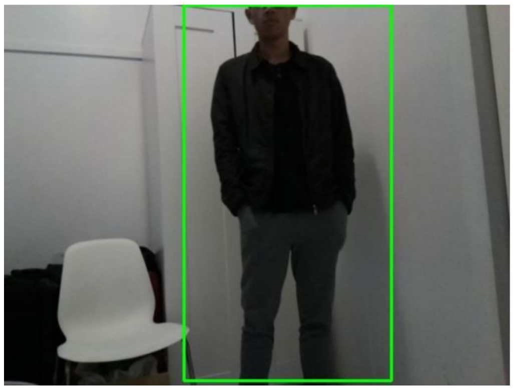
Now, the program has found the motion areas. However, the next problem I need to tackle is that how to make program only focus on detecting motion of person but not other objects. To achieve this result simultaneously maintain the FPS of video, I simply add constraint to the contour's area of objects. If the area is larger than 5000 pixel², the program will classify that object as a person and start recording. However, the figure for the contour area used to detect human motion is determined based on the location of the camera. The camera will work perfectly in the environment with not much background motion.



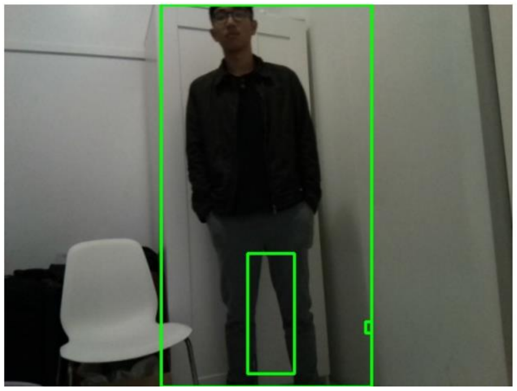
```
if len(cnts) > 0:
    #cnts is the list of objects which are in motion
    for c in cnts:
        # compute the bounding box for the contour, draw it on the frame,
        # and update the text
        if cv2.contourArea(c) > 5000:
            (x, y, w, h) = cv2.boundingRect(c)
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
            motionDectected = True
else:
    motionDectected = False
```

This is the result:

After limiting the contour area to detect motion

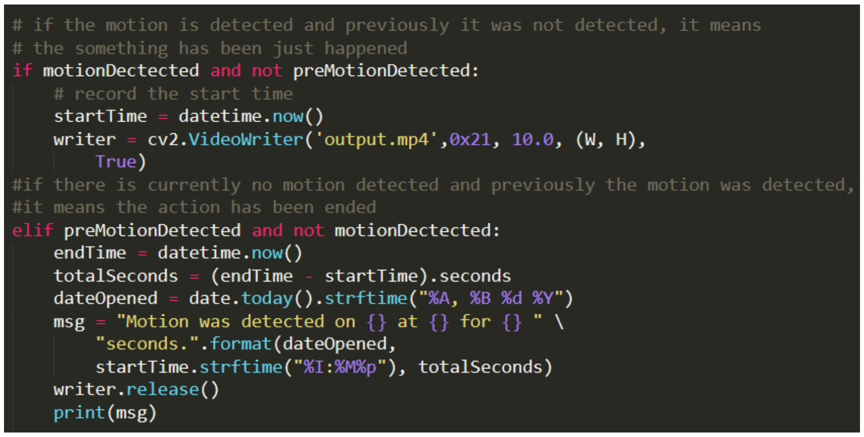


Before limiting the contour area to detect motion



GRAB VIDEO AND SEND IT VIA EMAIL USING PYTHON

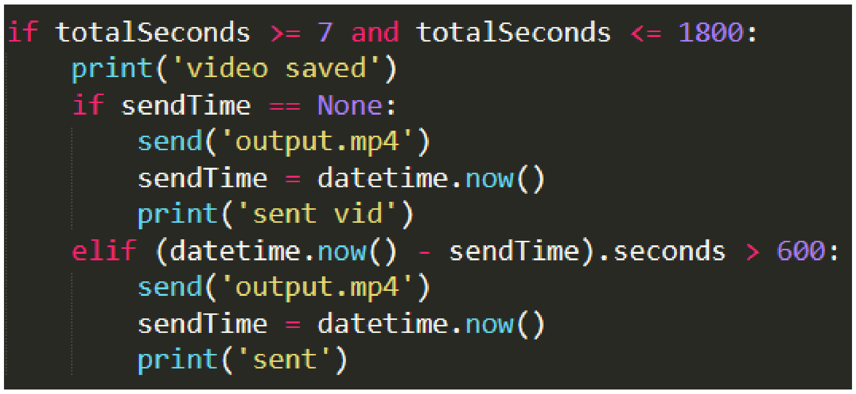
The application will automatically start writing the sequence of frames to a video in mp4 format when human motion is detected (in this program, human is defined as an object that has contour area greater than 5000). It will stop recording and print a notification if there is no motion appears.



```
# if the motion is detected and previously it was not detected, it means
# the something has been just happened
if motionDectected and not preMotionDetected:
    # record the start time
    startTime = datetime.now()
    writer = cv2.VideoWriter('output.mp4',0x21, 10.0, (W, H),
        True)
#if there is currently no motion detected and previously the motion was detected,
#it means the action has been ended
elif preMotionDetected and not motionDectected:
    endTime = datetime.now()
    totalSeconds = (endTime - startTime).seconds
    dateOpened = date.today().strftime("%A, %B %d %Y")
    msg = "Motion was detected on {} at {} for {} " \
        "seconds.".format(dateOpened,
        startTime.strftime("%I:%M%p"), totalSeconds)
    writer.release()
    print(msg)
```

Once the video is generated, it has to satisfy the below conditions before being sent via email:

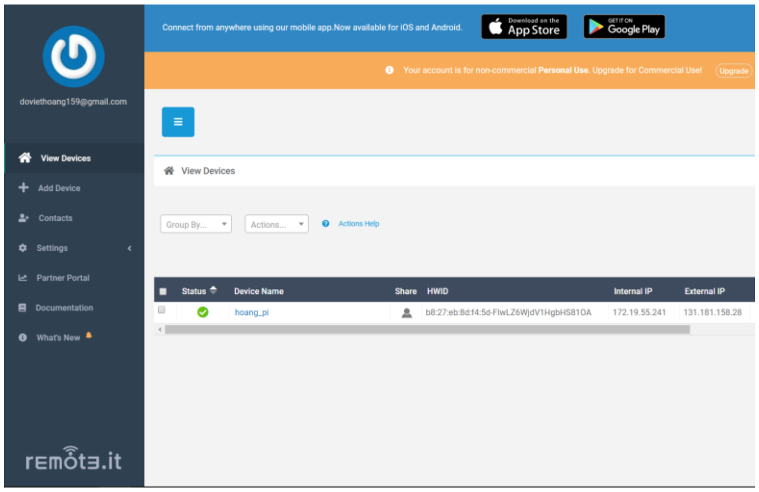
- The length of the video must be greater than 7 seconds and lower than 30 minutes. Therefore, I can reduce the risk that video contains non-sense content and avoid error that attachment size exceeds the allowable limit.
- The video will only be sent if there is no previous sending taken 10 minutes beforehand. It prevents the program from spamming email too frequently.



```
if totalSeconds >= 7 and totalSeconds <= 1800:
    print('video saved')
    if sendTime == None:
        send('output.mp4')
        sendTime = datetime.now()
        print('sent vid')
    elif (datetime.now() - sendTime).seconds > 600:
        send('output.mp4')
        sendTime = datetime.now()
        print('sent')
```

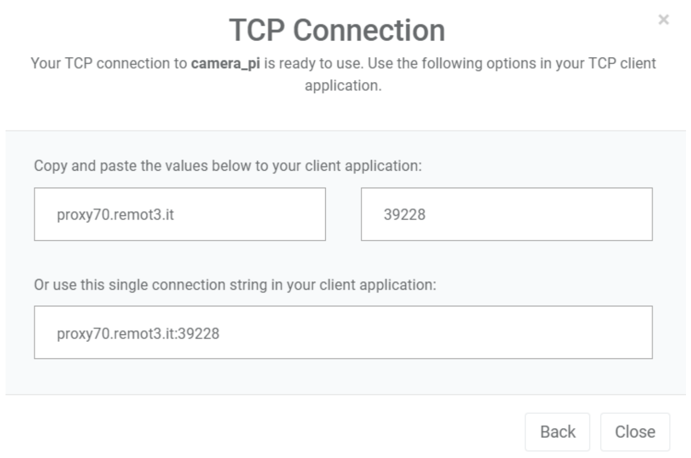
SET-UP PORT FORWARDING

At this point, my application only offers access to the live stream video webpage to users sharing the same network with the Raspberry Pi over the port 5000 (you can access by typing *raspberry_IP:5000* in your browser). To turn this application into a truly IoT device, the port forwarding technique is required to enable the access from anywhere. As mentioned in the lecture, there are some way around this such as creating a proxy server or making a hole in the firewall and set-up port forwarding. However, it is impossible to take these options in QUT, where the server security is protected by many experienced computer professionals. Therefore, using an alternative third-party service, Remote.it, is my choice. After installing Remote.it in Raspberry Pi by command `sudo apt-get install connectd`, I can easily establish TCP connection to the port 5000 by typing command `sudo connectd_installer` and following the app's instructions. Now I can see my Raspberry Pi in Remote.it web page:



	Status	Device Name	Share	HWID	Internal IP	External IP
	✓	hoang_pi		b8:27:eb:8d:f4:5d-FlwLZ6WjdV1HgbHS81OA	172.19.55.241	131.181.158.28

By choosing custom TCP service in Raspberry Pi and clicking the name of my device in the web page, an URL is provided by Remote.it (e.g. *proxy72.rt3.io:39288*):



TCP Connection

Your TCP connection to **camera_pi** is ready to use. Use the following options in your TCP client application.

Copy and paste the values below to your client application:

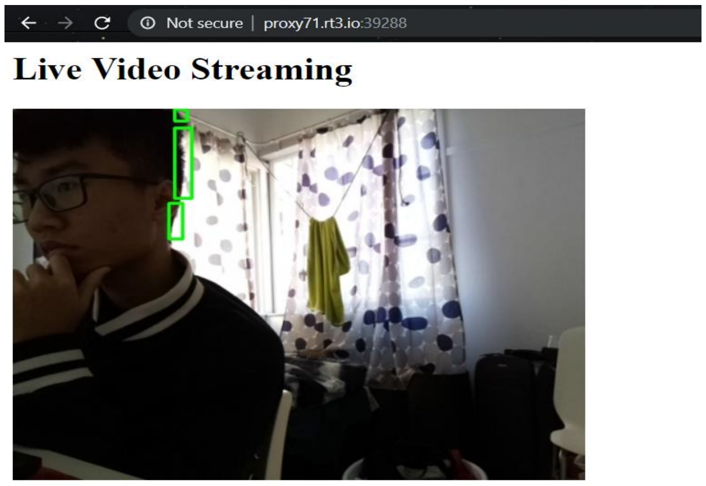
proxy70.remot3.it

39228

Or use this single connection string in your client application:

proxy70.remot3.it:39228

This is the result when I access this URL in Google Chrome from different network:



Live Video Streaming

FUTURE IMPROVEMENTS

Implementing IR LEDs to enable Pi camera to capture images from darkness

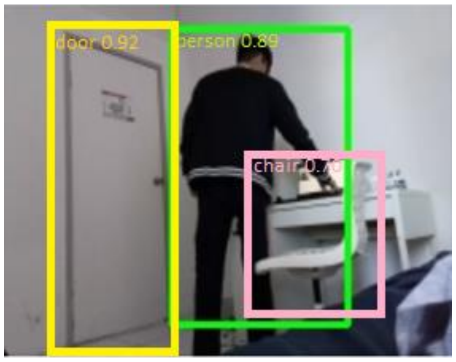
The project uses NoIR filter camera, which can pick up infrared light that's invisible to humans. However, due to the lack of a source of infrared illumination, I cannot take full advantage of this camera. The IR LEDs can install next to the camera and provide illumination to help the camera see in seemingly complete darkness (Tony DiCola, 2015). In this way, the protection of the security camera will be enhanced significantly.

Making a web application

The application generates a web page to display the real-time video that is captured by Pi Camera. It is true that its function is still not versatile. In the future, I want to upgrade this web page to a real web application that can serve multiple purposes and allow users to interact such as: taking picture from the website, saving recorded videos for better management, tracking the history timeline or setting up the schedule for camera activity. Moreover, the website also should have more attractive design and layout to improve user experience. To carry out these proposed improvements, a significant amount of knowledge about HTML, CSS and JavaScript and the design of back end server is required.

Using DNN technique for real-time object detection

One of the reasons I installed OpenCV 3.3 instead of OpenCV 2 is that the dnn module has been part of the main repository in version 3.3.0. This module supports deep learning by loading pre-trained object detection network and returning the output with bounding box, name and coordinates of each objects in the image (Adrian Rosebrock, 2017). This is the result when I used deep neutral network method to detect object:



The advantage of this method is that it can prevent the camera from recording non-human motion and make the system more solid. On the other hand, processing images using dnn is a time-consuming and resource-intensive task. It is impossible for a minicomputer like Raspberry Pi to perform it smoothly. In fact, when detecting objects in real-time video, my Pi computer can only return the video with 0.8 frames per second – which is inadequate for a security camera project. One solution for the performance problem is building a Raspberry Pi cluster – making a powerful computer from multiple minicomputers. Therefore, it may take less time to process frames and produce more desirable outcome.

Reference list

OpenCV (n.d.) Introduction to OpenCV. Retrieved from: https://opencv.org/about/

Jrosebr1 (n.d.) Introduction to Imutils. Retrieved from https://github.com/jrosebr1/imutils

Adrian Rosebrock (2017). Deep learning with OpenCV. Retrieved from: https://www.pyimagesearch.com/2017/08/21/deep-learning-with-opencv/

Tony DiCola (2015). Cloud camera. Retrieved from: https://learn.adafruit.com/cloud-cam-connected-raspberry-pi-security-camera/overview

Adrian Rosebrock (2015). Home surveillance and motion detection with the Raspberry Pi, Python, OpenCV, and Dropbox. Retrieved from: https://www.pyimagesearch.com/2015/06/01/home-surveillance-and-motion-detection-with-the-raspberry-pi-python-and-opencv/

Miguel Grinberg (2014). Video streaming with Flask. Retrieved from: https://blog.miguelgrinberg.com/post/video-streaming-with-flask