Population Traffic tracking IOT system

Team members:

JiaLun Bao Cooper Union 2018’ EE
Rafi Mueen Cooper Union 2019’ BSE
ZheKai Jin   Cooper Union 2019’ EE

(by 2017 Summer)

Author: ZheKai Jin   Cooper Union 2019’ EE

Table of Contents:

Abstract

This project involves tracking the population in regards of number of occupants in a building as well as the population flow. The goal is to locate and pinpoint occupants' position and the test is firstly conducted in cooper and will be extended to train station as a larger application. This has been done by collecting data about WiFi MAC Addresses, Bluetooth addresses, video and picture captured by attached cameras. The information about humidity, temperature, and ambient lighting level is also collected for data visualization as well as later application of energy saving. Through the sensor system, back-end processing and front-end visualization, this project demonstrated a fully functionable and a complete Internet-of-Things system.

Mission statement

The primary goal of the project is to be able to process the acquired data in a real time base to give a precise information about the population in a limited area and to keep record of the changing the number of occupants by adding the number of people entering into the data stored as well as subtracting the number of people leaving in a real time base. The Mac Addresses can be interpolated together with occupancy data to provide a pinpoint of occupants' exact location. After all the data is collected, the front-end website should be able to retrieve them from the database which the sensor system upload into and then process them to a presentable level with viewer options. The sensor system should be compact, easy-to-install, and allowed to work without wall plug in power in special location for a considerable amount of time. Putting all these pieces together seamlessly, the IOT system should be able to reflect the background and occupancy information about the building in real time.

Smartphone Detection
Bluetooth
Wifi
SHOP
DEPARTMENT STORE

Team distribution

Sensor network, Database communication and real time Processing:

-To acquire all kinds of data with real-time processing and store them into the database

JiaLun Bao Cooper Union 2018’ EE

ZheKai Jin   Cooper Union 2019’ EE

Data visualization:

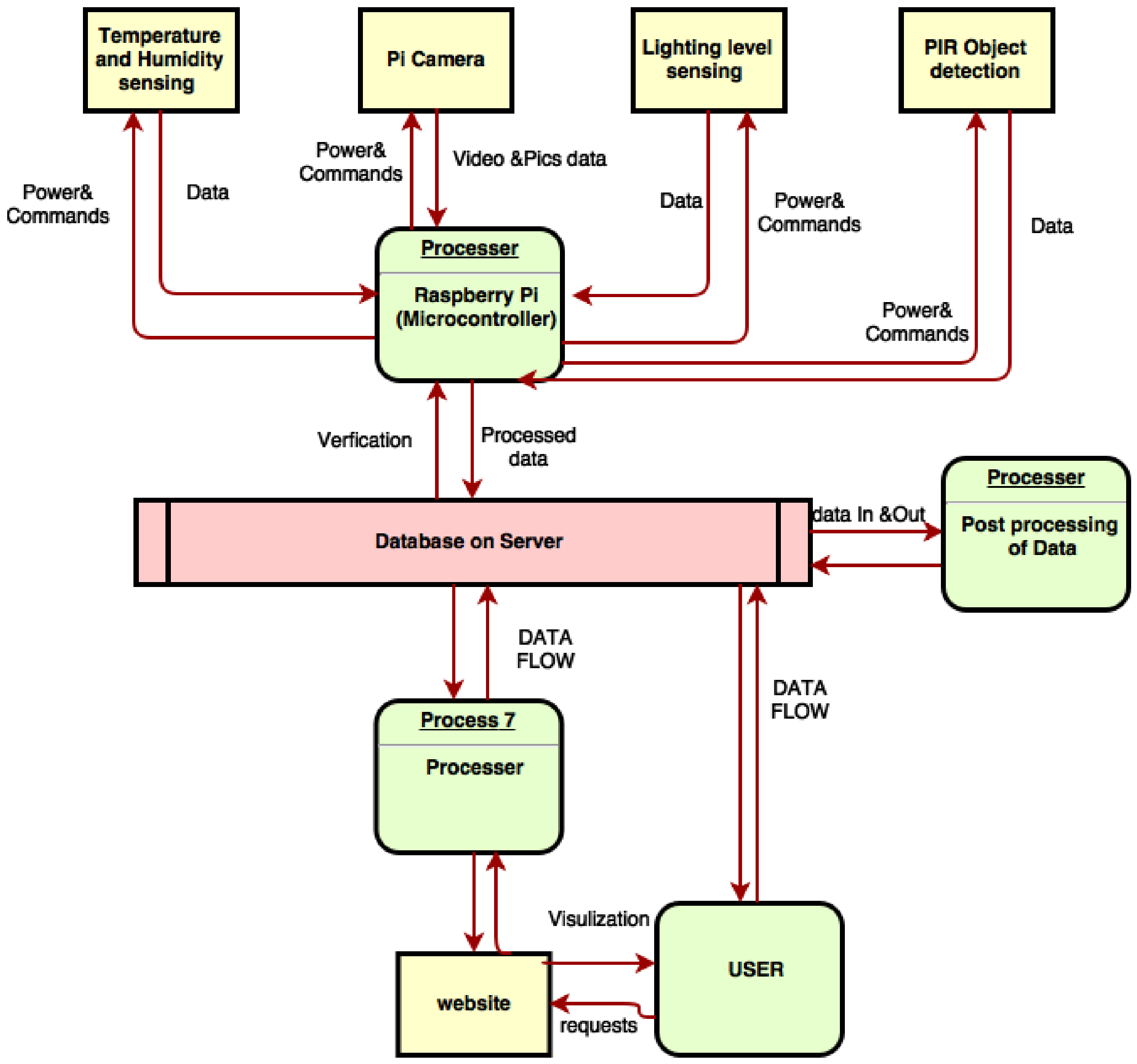
- To present data in the database onto the website and cope with various view options

Rafi Mueen Cooper Union 2019’ BSE

Data Post-processing:

-To process the data to show trends and patterns.

Flow diagram



Setup & Application

As target testing, we first install the box in the ground level of Cooper and try to track the population flow with verification of Ground Truth Data from security and then finetune the precision of the sensors as well as the accuracy level of the algorithm used. In the same time during the summer, the Lab is also installed with the sensor box on the ceiling to collect all the data available but with a limited population flow. The data is verified with a registration table at the entry and video typing.

The difference between the data and the real value can be firstly minimized with algorithm choosing from one frame differentiating to multi frame analysing. Moreover, if the devices and the algorithms have been fine-tuned to their best, machine learning can take its role here by collecting a large data sample and push the data to real value.

Later, the devices are meant to be set up anywhere with its range considered to monitor the data set and then monitored on the website with the contour map given. For example, in a train station, we can have one setup per terminal to monitor the population flow between the terminal , and also fine tune the lighting level of the station with the occupancy number, time and the ambient lighting level by accessing the firmware controlling the train station lighting. Also, one can clearly identify numbers of Wifi devices and Bluetooth devices within each terminal.

Wifi & Bluetooth Address Acquisition

Wifi acquisition->Summary:

In order to realize the wifi address acquisition, we take the advantage of Linux-base system on the Raspberry Pi, and utilize the monitor mode of the Wifi card to capture all the information floating in the air.

Method plan out:

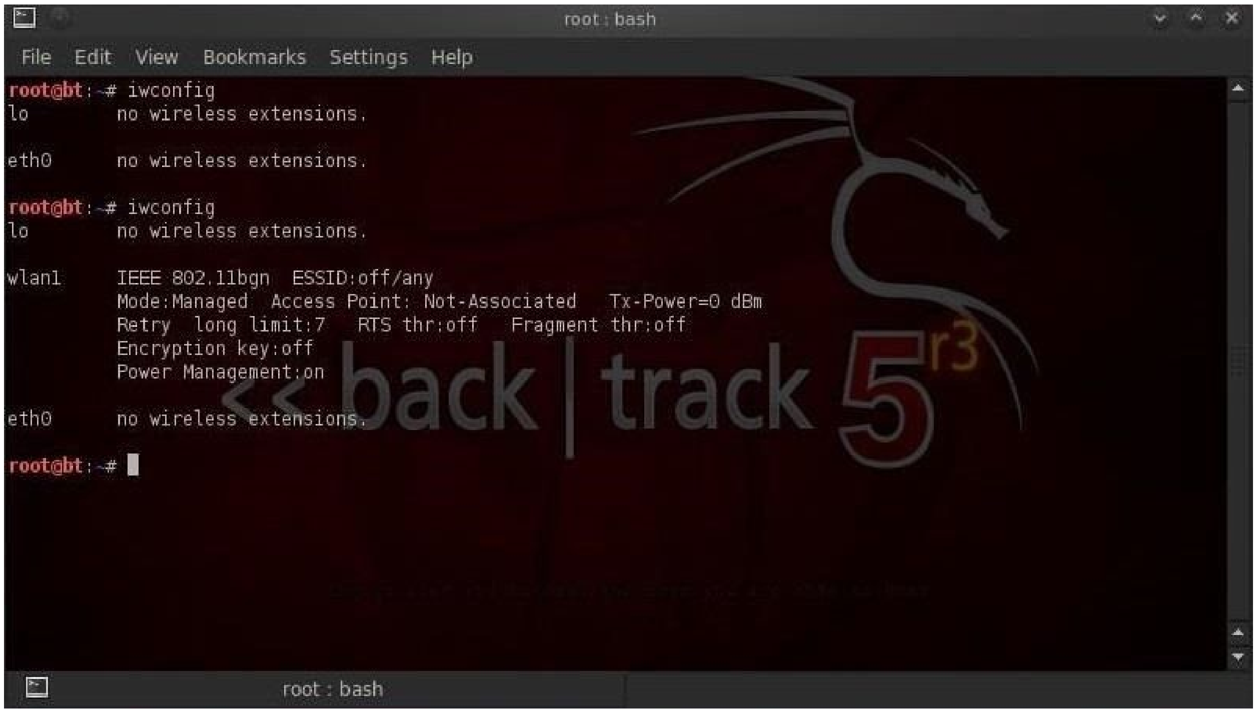
The aircrack-ng is a suite of tools for manipulating and cracking Wi-Fi networks. The ng stands for a new generation with the older one no longer supported anymore. Within this suite, a tool called **aircrack** for cracking passwords. Aircrack-ng is capable of doing DOS attacks as well rogue access points, caffe latte, evil twin, and many others. But we are only monitoring the population traffic, so not any of this offensive hacking is used.

Equipment needed:

Raspberry Pi (any version is supported) installed with aircrack-ng

Wifi adaptor with monitor mode supported and extended firmware support on Linux. (Raspberry Pi 3's built in wifi adaptor does not support **monitor mode**)

Step one -> iwconfig : ensure that system has recognized external Wifi card



```
root@bt:~# iwconfig
lo        no wireless extensions.

eth0      no wireless extensions.

root@bt:~# iwconfig
lo        no wireless extensions.

wlan1     IEEE 802.11bgn  ESSID:off/any
          Mode:Managed  Access Point: Not-Associated   Tx-Power=0 dBm
          Retry  long limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:on

eth0      no wireless extensions.

root@bt:~#
```

From the information on the screenshot, the wifi card is capable of 802.11bgn mode of Wifi standard , and its ESSID is off, with its the mode managed, etc.

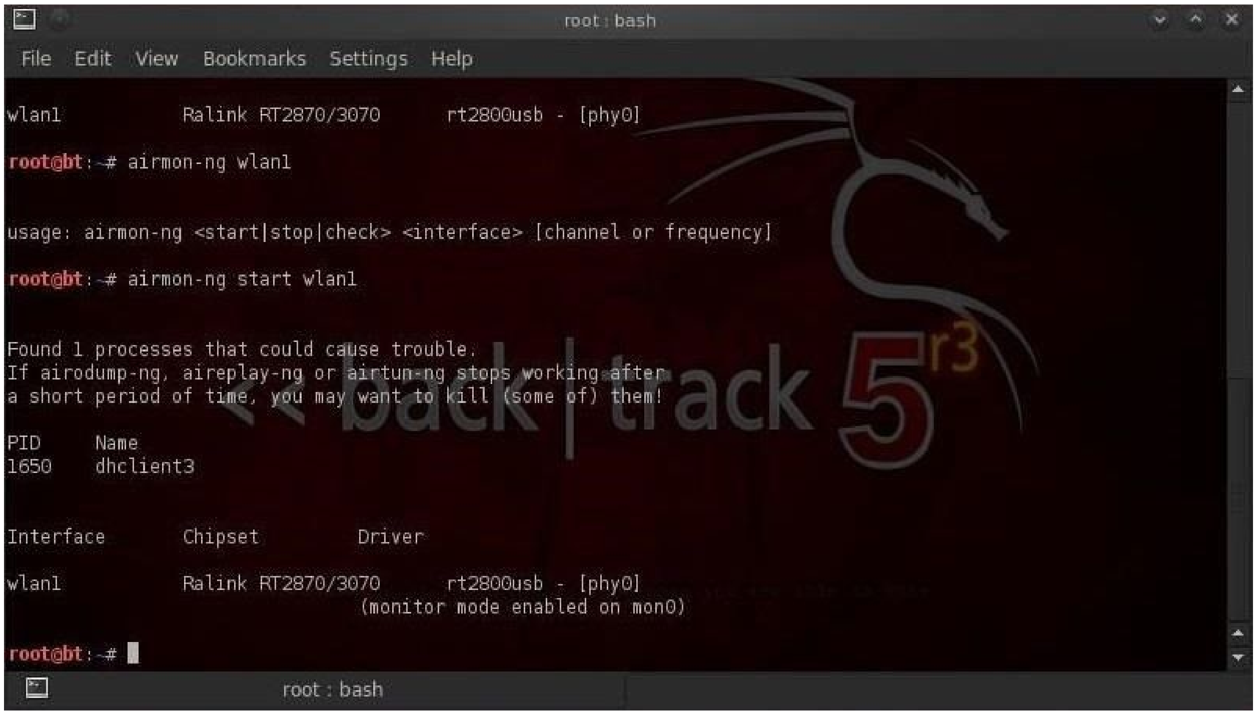
Noted: A standard 802.11 wireless card support 7 modes: Master (acting as an access point), Managed (client, also known as station), Ad hoc, Mesh, Repeater, Promiscuous, and Monitor mode.

Step Two -> Airmon-ng : converts the wifi card into monitor mode

Airmon-ng turns the wifi card into monitor mode, basically will turn a computer with a wireless network interface controller (WNIC) to monitor all traffic received from the wireless network. Unlike promiscuous mode, which is also used for packet sniffing, monitor mode allows packets to be captured without having to associate with an access point or ad hoc network first.

The command form: **bt > airmon-ng start wlan1**

airmon-ng <start|stop> <interface> [channel] **airmon-ng** <check> [kill]



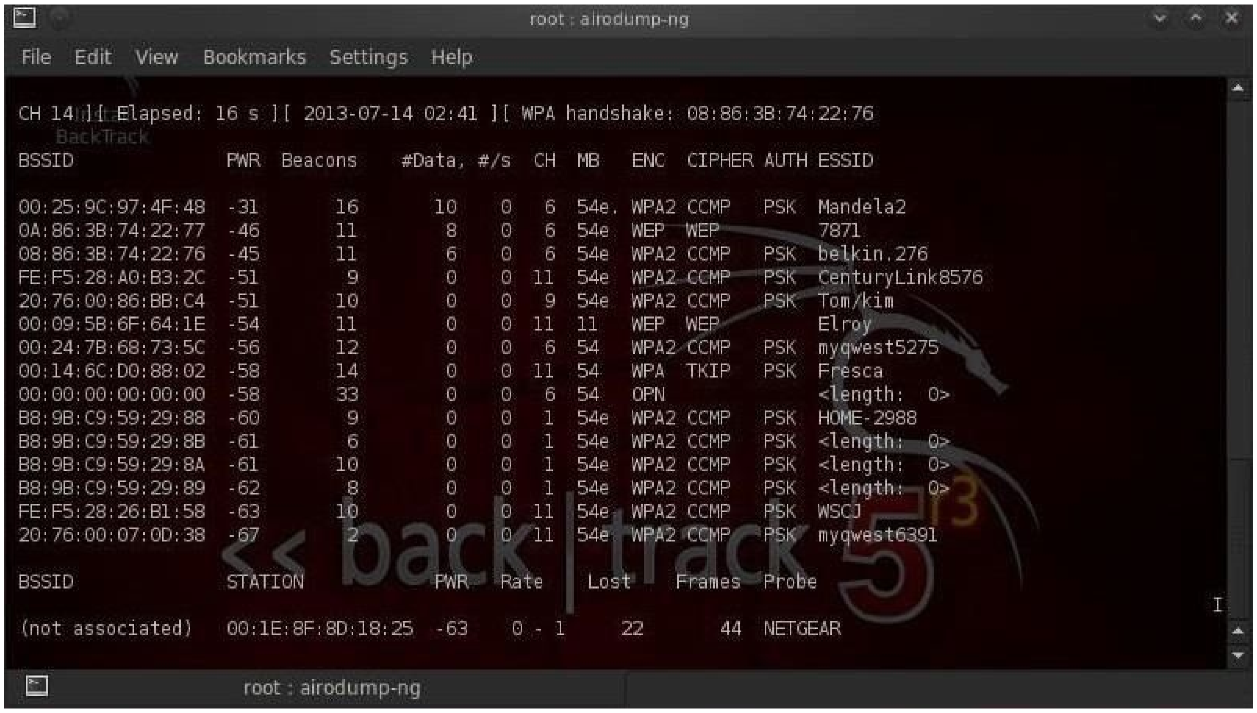
```
wlan1           Ralink RT2870/3070      rt2800usb - [phy0]

root@bt:~# airmon-ng wlan1

usage: airmon-ng <start|stop|check> <interface> [channel or frequency]

root@bt:~# airmon-ng start wlan1

Found 1 processes that could cause trouble.
If airodump-ng, aireplay-ng or airtun-ng stops working after
a short period of time, you may want to kill (some of) them!

PID     Name
1650    dhclient3

Interface       Chipset         Driver

wlan1           Ralink RT2870/3070      rt2800usb - [phy0]
                                (monitor mode enabled on mon0)

root@bt:~# 
```

Now the monitor mode is enabled, from the screen information and with the monitor mode the wifi card is capturing data from the air.

Step Three -> Airodump-ng : capture all the available connected devices

The next tool in the aircrack-ng suite that we used is **airodump-ng**, which enables us to capture packets of our specification. We activate this tool by typing the **airodump-ng** command and specify the **renamed monitor interface** (mon0):

- **bt >airodump-ng mon0**



```
CH 14 ][ Elapsed: 16 s ][ 2013-07-14 02:41 ][ WPA handshake: 08:86:3B:74:22:76

BSSID              PWR  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

00:25:9C:97:4F:48  -31       16       10    0   6  54e. WPA2 CCMP   PSK  Mandela2
0A:86:3B:74:22:77  -46       11        8    0   6  54e  WEP  WEP         7871
08:86:3B:74:22:76  -45       11        6    0   6  54e  WPA2 CCMP   PSK  belkin.276
FE:F5:28:A0:B3:2C  -51        9        0    0  11  54e  WPA2 CCMP   PSK  CenturyLink8576
20:76:00:86:BB:C4  -51       10        0    0   9  54e  WPA2 CCMP   PSK  Tom/kim
00:09:5B:6F:64:1E  -54       11        0    0  11  11   WEP  WEP         Elroy
00:24:7B:68:73:5C  -56       12        0    0   6  54   WPA2 CCMP   PSK  myqwest5275
00:14:6C:D0:88:02  -58       14        0    0  11  54   WPA  TKIP   PSK  Fresca
00:00:00:00:00:00  -58       33        0    0   6  54   OPN              <length:  0>
B8:9B:C9:59:29:88  -60        9        0    0   1  54e  WPA2 CCMP   PSK  HOME-2988
B8:9B:C9:59:29:8B  -61        6        0    0   1  54e  WPA2 CCMP   PSK  <length:  0>
B8:9B:C9:59:29:8A  -61       10        0    0   1  54e  WPA2 CCMP   PSK  <length:  0>
B8:9B:C9:59:29:89  -62        8        0    0   1  54e  WPA2 CCMP   PSK  <length:  0>
FE:F5:28:26:B1:58  -63       10        0    0  11  54e  WPA2 CCMP   PSK  WSCJ
20:76:00:07:0D:38  -67        2        0    0  11  54e  WPA2 CCMP   PSK  myqwest6391

BSSID              STATION            PWR   Rate    Lost    Frames  Probe

(not associated)   00:1E:8F:8D:18:25  -63    0 - 1     22       44  NETGEAR
```

As we can see in the screenshot above, airodump-ng displays all of the APs (access points) within range with their BSSID (MAC address), their power, the number of beacon frames, the number of data packets, the channel, the speed, the encryption method, the type of cipher used, the authentication method used, and finally, the ESSID.

Now we acquire the needed information about all the devices that are connected to the nearby wifi and we can even look into the station by typing

```
bt>airdump-ng --bssid <any station ssid you see on the screenshot> -c <channel name#> -w <filename> mon0
```

But that's not enough for our purpose yet since we want all the information about devices with just Wifi feature on them even that are not connected to any of the base station around(routers): Thus we need tshark to capture all the **probe requests**, which will include all the connected devices :

A **probe request** is a special frame sent by a client station requesting information from either a specific access point, specified by SSID, or all access points in the area, specified with the broadcast SSID. That being said, it means when you turn on your wifi device and the wifi adaptor on your device will automatically send a probe requests to the base stations to see if any of them is free to join (without WPA2 security required) or connected before. This also explains the reason that your phone will automatically connected to used Wifi before.

To acquire all the Wifi information, we use the command line tool tshark, which is the command line version of ethical hacking tool wireshark.

```
sudo tshark -a duration:30 -S -l -i mon0 -Y 'wlan.fc.type_subtype eq 4' -T fields -e frame.time -e wlan.sa_resolved -e wlan_mgt.ssid >> probe.csv
```

This command will capture all the required information we need to probe.csv and saved to raspberry Pi.

The display filter is selected to just show the probe request. Other information can also be captured by change the filter. Reference of the filter info can be traced here[1].

sudo airmon-ng stop mon0 (stop monitor mode)

At the end , Airmon-ng is stopped to turn off monitor mode.

And the installation of tshark and aircrack-ng can be found at these sites:

http://www.linux-magazine.com/Issues/2015/174/Tshark

https://www.aircrack-ng.org/doku.php?id=install_aircrack#installing_on_mac_osx

Bluetooth(BT) Data

For the BT data, we use a tool called **hcitool** for capturing.

hcitool is the swiss army knife for Bluetooth in Linux. It is aptly named hcitool as it communicates via a common HCI (Host Controller Interface) port to your bluetooth devices. You can utilize the utility to scan for devices and send commands data for standard Bluetooth and Bluetooth Low Energy.

bt> sudo hcitool -i hci0 scan

Will return all the BT-turned-on devices in the range (whether connected or not)

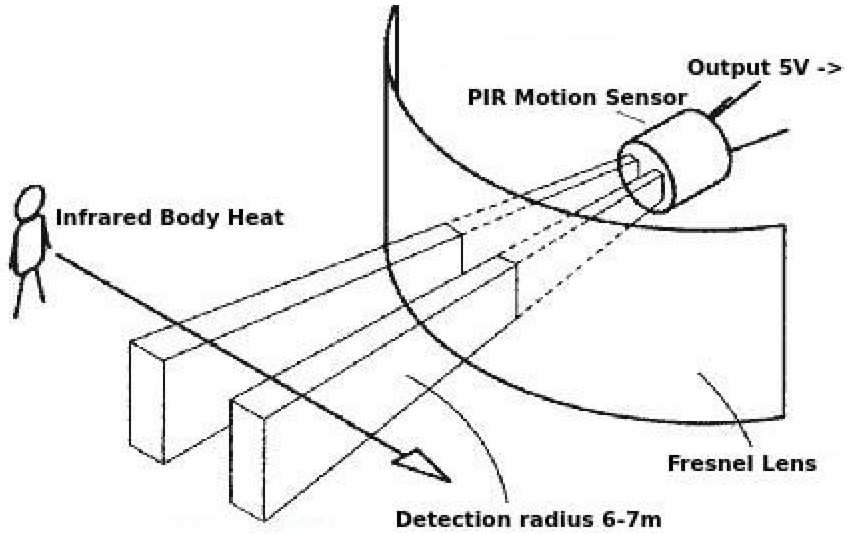
[1] http://www.lovemytool.com/blog/2010/07/wireshark-wireless-display-and-capture-filters-samples-part-2-by-joke-snelders.html

Object detection by PIR sensors

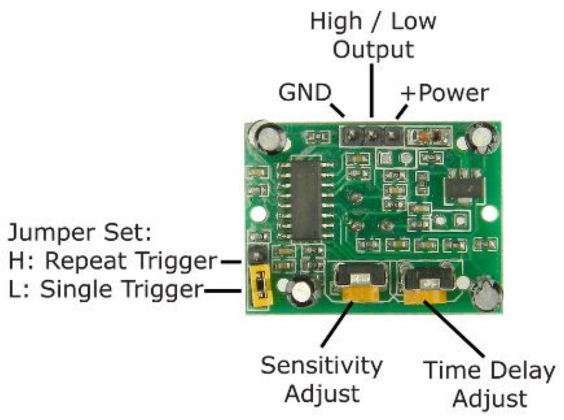
Summary:
use PIR sensor which return digital data back into raspberry pi to sense if there is object presence.

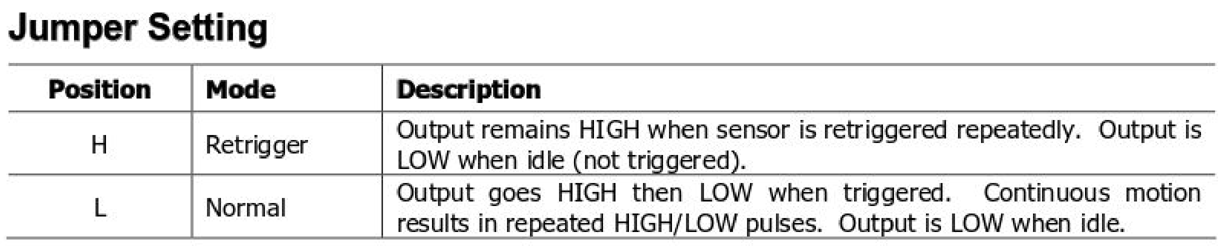
Theoretical Method planned out:

PIR stands for Passive InfraRed. This motion sensor consists of a fresnel lens, an infrared detector, and supporting detection circuitry. The lens on the sensor focuses any infrared radiation present around it toward the infrared detector. Our bodies generate infrared heat, and as a result, this heat is picked up by the motion sensor. The sensor outputs a 5V signal for a period of one minute as soon as it detects the presence of a person. It offers a tentative range of detection of about 6-7 meters and is highly sensitive. When the PIR motion sensor detects a person, it outputs a 5V signal to the Raspberry Pi through its GPIO and we define what the Raspberry Pi should do as it detects an intruder through the python coding. Then we can define function in the program to print out : “Intruder detected”. The drawback is that it only works for motion detection and works for a limited range. In another word, if there is a person already in the range before the PIR started, the presense will not be detected, but for general purpose detection, it will suffice our needs.



Practical problem:
The PIR motion sensor has two operation modes, which can be toggled on the circuity:





Jumper Setting

Position	Mode	Description
H	Retrigger	Output remains HIGH when sensor is retriggered repeatedly. Output is LOW when idle (not triggered).
L	Normal	Output goes HIGH then LOW when triggered. Continuous motion results in repeated HIGH/LOW pulses. Output is LOW when idle.

Jumper to L position the sensor will 'toggle' (change state) every time motion is detected - probably providing the on-board timer has timed out. This is unlikely to be of much use in a practical application. (Watch the comments for corrections!)

Moving the jumper to the H position will result in the more usual sensor logic. The sensor will turn on when motion is detected and turn off some time after the last motion is detected. This sensor will reset the timer (which would otherwise turn the output off) every time motion is detected and would be suitable, for example, for room occupancy lighting control where you don't want the lights to blink off while the unit resets.

To be noted: Infrared sensors has been tested, but with the analog circuit on them and digital input on the raspberry Pi, ADC (analog to digital convertor) circuit is needed in the box, to avoid the extra weight, we use the PIR motion sensor at the end.

Traffic tracking with Opencv library

Setting up the camera:
https://www.raspberrypi.org/learning/getting-started-with-picamera/worksheet/
The methodology to set up the camera can be followed on this site since the Raspberry Pi has an peripheral software built with camera already installed in the Raspbian Operating system, no the backtrack used for Wifi capturing demo.

By using the python and Picamera, one can use the camera for taking still pictures and recording videos, and to manipulate effects.
For example, to take 5 photos in a row:

```
camera.start_preview()
for i in range(5):
    sleep(5)
    camera.capture('/home/pi/Desktop/image%s.jpg' % i)
camera.stop_preview()
```

And save it on the current working directory.

Moreover, the video recording features:

```
camera.start_preview()
camera.start_recording('/home/pi/video.h264')
sleep(10)
camera.stop_recording()
camera.stop_preview()
```

Then, to display the video, just do

omxplayer video.h264

Use opencv library for population flow :

OpenCV (*Open Source Computer Vision*) is a library of programming functions mainly aimed at real-time computer vision. Here we use the opencv python library to allow the processing of video that Pi camera captured to monitor the In & Out flow of population.

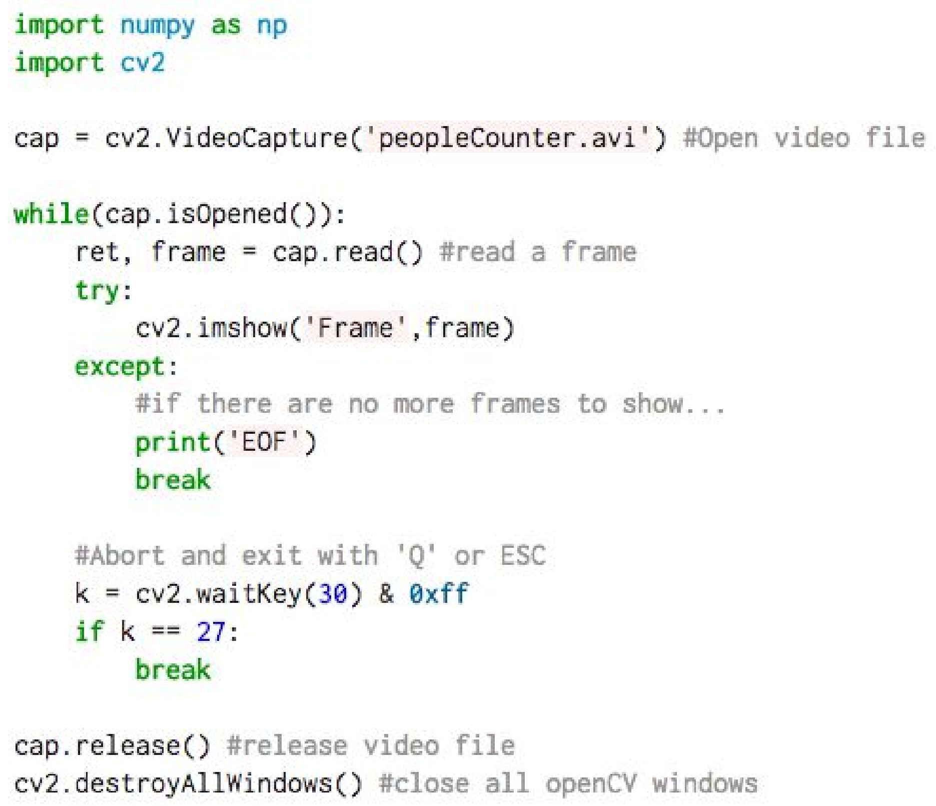
To install opencv:

http://www.pyimagesearch.com/2016/04/18/install-guide-raspberry-pi-3-raspbian-jessie-opencv-3/

The procedures can be followed here, but the modified version can be found in the appendix.

To realize population trraffic monitoring, we follow a guide on this blog: http://www.femb.com.mx

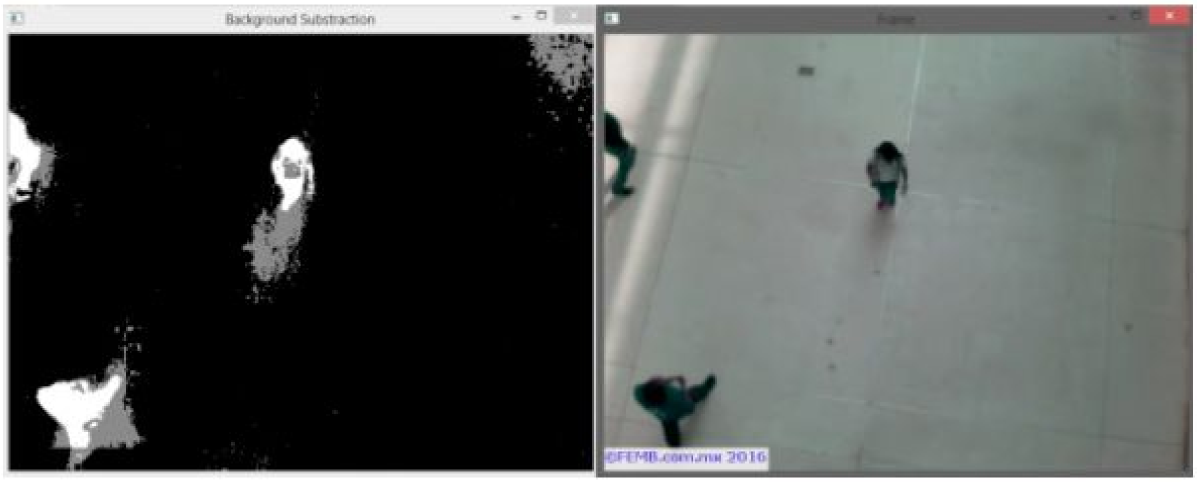
- Firstly ,we need to open the Opencv video stream with opencv's function to capture video:



```
import numpy as np
import cv2

cap = cv2.VideoCapture('peopleCounter.avi') #Open video file

while(cap.isOpened()):
    ret, frame = cap.read() #read a frame
    try:
        cv2.imshow('Frame',frame)
    except:
        #if there are no more frames to show...
        print('EOF')
        break

    #Abort and exit with 'Q' or ESC
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release() #release video file
cv2.destroyAllWindows() #close all openCV windows
```

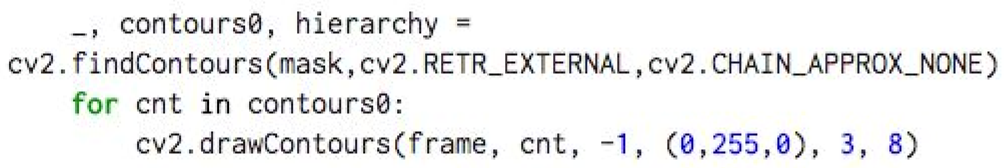
- Then, we add a background subtractor to reduce the unnecessary parts in our video,

```
import numpy as np
import cv2

cap = cv2.VideoCapture('peopleCounter.avi') #Open video file

while(cap.isOpened()):
    ret, frame = cap.read() #read a frame
    try:
        cv2.imshow('Frame',frame)
    except:
        #if there are no more frames to show...
        print('EOF')
        break

    #Abort and exit with 'Q' or ESC
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release() #release video file
cv2.destroyAllWindows() #close all openCV windows
```



The left image with the background removed is much easier for processing and will be used for people counting.

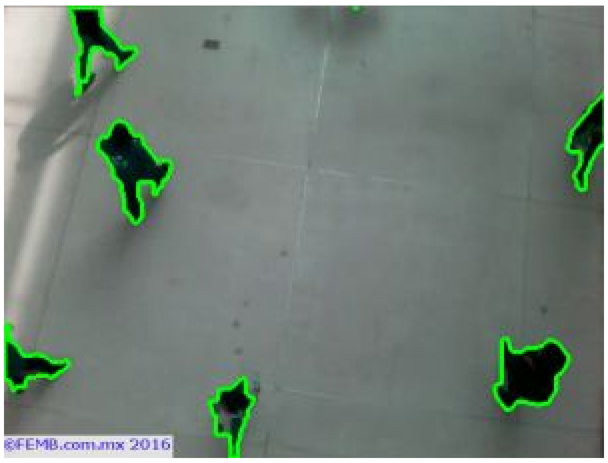
- Next, we conclude the frame with contour shape use the findcontours function in openCV



```
    _, contours0, hierarchy =
cv2.findContours(mask,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_NONE)
    for cnt in contours0:
        cv2.drawContours(frame, cnt, -1, (0,255,0), 3, 8)
```

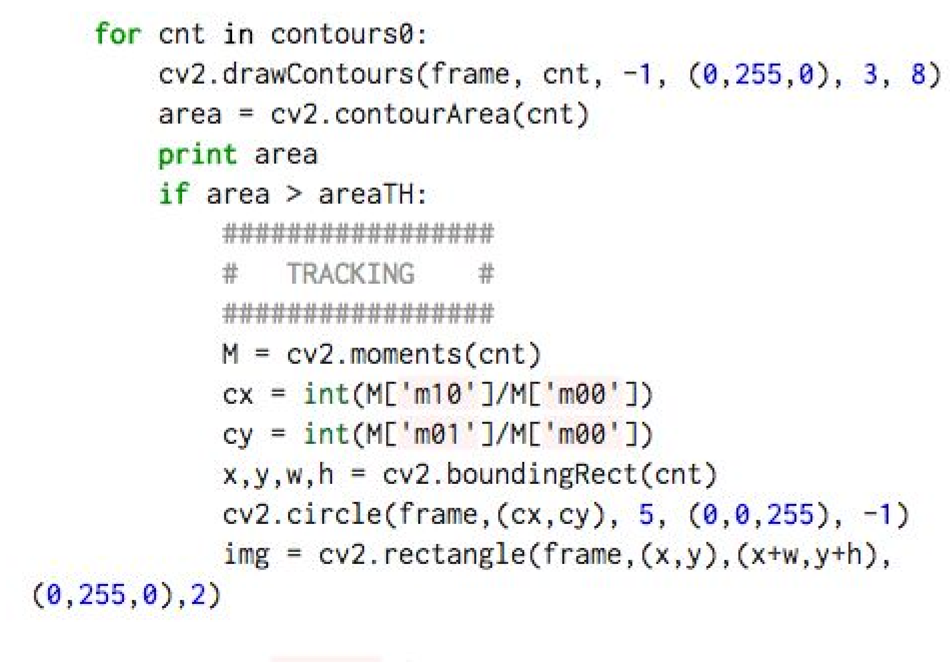
We give the function our mask, cv2.RETR_EXTERNAL means we only care about external contours (contours within contours will not be detected), and cv2.CHAIN_APPROX_NONE is the algorithm used to “make” the contour (you can change it to another one as a test).

The drawcontour gives the visual representation :



- Next, we want to define the function person to actually differentiate the person contour:

  defining a minimum area the contour must have:



```
    for cnt in contours0:
        cv2.drawContours(frame, cnt, -1, (0,255,0), 3, 8)
        area = cv2.contourArea(cnt)
        print area
        if area > areaTH:
            #################
            #   TRACKING    #
            #################
            M = cv2.moments(cnt)
            cx = int(M['m10']/M['m00'])
            cy = int(M['m01']/M['m00'])
            x,y,w,h = cv2.boundingRect(cnt)
            cv2.circle(frame,(cx,cy), 5, (0,0,255), -1)
            img = cv2.rectangle(frame,(x,y),(x+w,y+h),
  (0,255,0),2)
```

For now, we already get if the person is in the image, and we need to track its motion, or the direction they are moving to(left/right :: up /down) depends on the direction of the camera setup .

In the first frame if we detect someone we give that person an ID and store its initial position in the image.

Then, on the following frames, we want to keep track of that person, we match the person's contour in the following frames to the ID we set when it first appeared, as well as keep storing that person's coordinates.

Then, after the person crosses a limit (or a certain amount limits) in the image, we evaluate, using all of the stored positions, if he/she is moving left or right.(up or down)

To handle all of this IDing and storing of coordinates we have a class called Person.

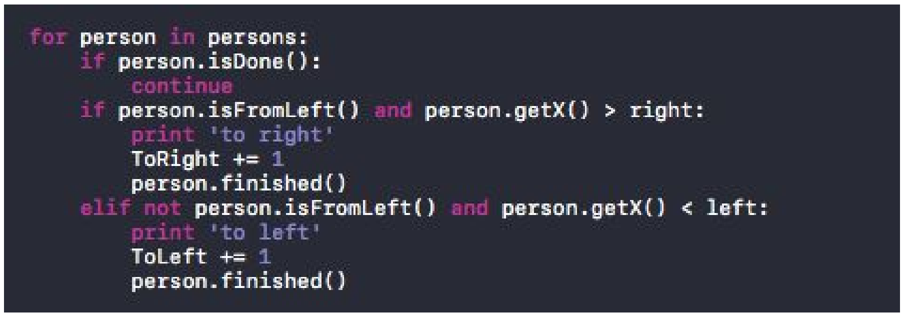
```
for i in persons:
```

```
                if abs(x-i.getX()) <= w and abs(y-i.getY()) <= h:
                    # boundary condition determination
                    new = False
                    i.updateCoords(cx,cy)   #update coordinates
                    break
            if new == True:
                p = Person.MyPerson(pid,cx,cy, max_p_age)
                persons.append(p)
                pid += 1
```

And use this part to track the person’s movement.

The actual and complete code can be referred in the appendix part.

- Now that the person’s movement is followed, we can build left bound and right bound: which here we **isFromLeft** and right position to evaluate if the object has crossed line_L or line_R in the correct direction. If so, then a counter is incremented.. and we’re counting people.



```
for person in persons:
    if person.isDone():
        continue
    if person.isFromLeft() and person.getX() > right:
        print 'to right'
        ToRight += 1
        person.finished()
    elif not person.isFromLeft() and person.getX() < left:
        print 'to left'
        ToLeft += 1
        person.finished()
```

Finally, we construct the counter to count people from left to right or from right to left.

Ambient lighting sensing

```python
# VL6180X integrates both distance sensor
# and light intensity sensor.

# 1-100k lux with 16 bits output precision.
# 0-100 mm in turns of range. (really short)
# the default I2C address for it is 0x29

import time
from rpisensors import VL6180X

sensor = VL6180X(1)

while True:
        dist = sensor.read_distance()
        lux = sensor.read_lux()
        print "%d mm| %d lux" % (dist,lux)
        time.sleep(1)
```

The VL6180X is the latest product based on ST’s patented FlightSense™technology. This is a ground-breaking technology allowing absolute distance to be measured independent of target reflectance. Instead of estimating the distance by measuring the amount of light reflected back from the object (which is significantly influenced by color and surface), the VL6180X precisely measures the time the light takes to travel to the nearest object and reflect back to the sensor (Time-of-Flight).

Combining an IR emitter, a range sensor and an ambient light sensor in a three-in-one ready-to-use reflowable package, the VL6180X is easy to integrate and saves the end-product maker long and costly optical and mechanical design optimizations. And it uses I2C bus for the result reading.

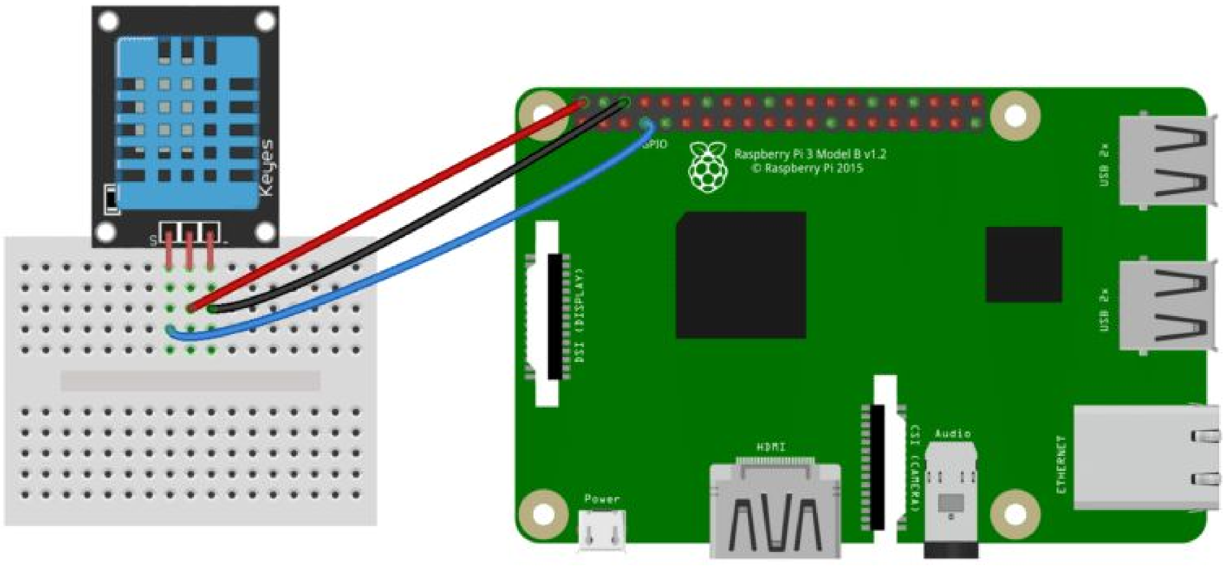
Thus, we can get the lux level and read the distance by using this sensor, the I2c bus hook up can be referred here.

https://pypi.python.org/pypi/rpisensors/0.2.1

Humidity & Temperature Sensing

```
import Adafruit_DHT as dht
import time


time.sleep(2)
h,t = dht.read_retry(dht.DHT22,4)

# convert temperature to Fahrenheit
t = t*9/5.0 + 32
print 'Temp={0:0.1f}F Humidity={1:0.1f}%'.format(t,h)
```

This feature is realized with DHT temperature & humidity sensors. These sensors are very basic and slow, but are great for basic data logging. The DHT sensors are made of two parts, a capacitive humidity sensor and a thermistor. There is also a very basic chip inside that does some analog to digital conversion and spits out a digital signal with the temperature and humidity. The digital signal is fairly easy to read using any microcontroller, like the Raspberry PI we are using . So just by linking it to the GPIO pin as input and then connect the power , the temperature and humidity can be read from the data bus.



SSH communication with Cooper network

In order to communicate with the Pi while it is on operation or set it up after installation,
We need to build connection to the Pi. There are a few ways to realize that.

- One way would be directly connecting the Pi through the ethernet connection, which could be realized if we know the IP address of the Pi.
  Examples:

```
ssh pi@169.254.180.143  (pi1)
ssh pi@169.254.152.101  (pi1)
ssh pi@169.254.180.144  (pi2)
```

- Another way would be connect the Pi to the ethernet connection of the school network and log into the Pi via school network. That gives more flexibility to the first method since multiple interfaces of ethernet connection can be found at school and we can reach the Pi at anywhere in school.

```
ssh -p2222 pi@pi.eng.cooper.edu
ssh pi@pi2.eng.cooper.edu
```

- The third way, which is the current setting of the Pi, is to use the school server as a moderator, and setting the connection to that server on the startup of the Pi, thus bundling the Pi and the server together, and the server can redirect us to the Pi when we log into the server through SSH and then SSH into the Pi on the server. The bundling step is similar to the wired connection in the second step but through wireless, which can be referred as **reverse tunnelling.**

Setting on the Pi

```
#!/bin/sh

autossh -N -o 'PubkeyAuthentication=yes' -o 'PasswordAuthentication=no' -i
/home/pi/.ssh/nopwd -R 3329:localhost:22 bao2@gwip.cooper.edu -p 8199 &
```

Logging in on PC

```
ssh -t -p 8199 bao2@gwip.cooper.edu ssh -t -p3329 pi@localhost
```

Data logging with the local server

The CPU of the Raspberry Pi and Storage sometimes will not be powerful enough for our requirements, and we need a server to store data and sometimes post-processing them. Thus, we need a server-end script to listen on the socket which is created on the Pi end and the connection is established in between the Pi and the server.

Following code demos a communication of the Pi to server using the socket communication in a PIR motion detector triggered image capture action.

Pi side:

```
from picamera import PiCamera
from time import sleep
import RPi.GPIO as GPIO
import io
import socket
import struct
import time



# start a socket
client_socket = socket.socket()
client_socket.connect(("199.98.20.87",8001))
connection = client_socket.makefile('wb')
stream = io.BytesIO()


# setup pin mode
GPIO.setmode(GPIO.BOARD)
GPIO.setup(11,GPIO.IN)

# setup camera
camera = PiCamera()
camera.rotation = 180
camera.resolution = (640,480)
# camera.framerate = 15
# camera.brightness = 70
# camera.annotate_text_size = 50

# capture background image

sleep(3)
camera.capture(stream,'jpeg')
```

```
connection.write(struct.pack('<L',stream.tell()))
connection.flush()
stream.seek(0)
connection.write(stream.read())
stream.seek(0)
stream.truncate()


# initialize state
prev_state = 0
start = time.time()
prev = start


while True:
        # PIR sensor code
        current_state = GPIO.input(11)
        if current_state == 1 and prev_state == 0:
                #print "%d %s"% (current_state, prev_state)
                #camera.start_preview()
                sleep(0.2)
                camera.capture(stream, 'jpeg')
                #camera.stop_preview()
                #print "captured"

                # tell the length of image
                connection.write(struct.pack('<L',stream.tell()))
                connection.flush()
                stream.seek(0)

                # send the image data to the socket
                connection.write(stream.read())

                # time out
                #if time.time() - current > 30:
                #       stream.seek(0)
                #       stream.truncate()
                #       continue
                # reset the stream
                stream.seek(0)
                stream.truncate()

        #else:
                #print "NO MOTION DETECTED"

        #print "%d %d" %(prev_state, current_state)
        prev_state = current_state

##############################################################

        # run time is 48 hours or 2 days
        if current - start > 172800:
                # clean up
```

```
            file.close()
            # send 0 to signal end of connection
            connection.write(struct.pack('<L',0))
            connection.close()
            client_socket.close()
            break


      sleep(1)
```

And on the local server, we need to run a script that listens to the socket and do post-processing with data with Database structure like MongoDB and Mysql, or just store the copy of images to save the memory room in the Pi.

Here is an example where the server read and save images the Picamera take.

```
import io
import socket
import struct
import time
from PIL import Image

# Start a socket listening for connections on 0.0.0.0:8000 (0.0.0.0 means
# all interfaces)
server_socket = socket.socket()
server_socket.bind(('0.0.0.0', 8000))
server_socket.listen(0)

root = './img/'
ext = 'png'

#print socket.gethostname()

# Accept a single connection and make a file-like object out of it
connection = server_socket.accept()[0].makefile('rb')
try:
    while True:
        # Read the length of the image as a 32-bit unsigned int. If the
        # length is zero, quit the loop
            image_len = struct.unpack('<L',
connection.read(struct.calcsize('<L')))[0]
```

```
        if not image_len:
            break
        # Construct a stream to hold the image data and read the image
        # data from the connection
        image_stream = io.BytesIO()
        image_stream.write(connection.read(image_len))
        # Rewind the stream, open it as an image with PIL and do some
        # processing on it
        image_stream.seek(0)
        image = Image.open(image_stream)
        #image.show()
        print('Image is %dx%d' % image.size)
        image.verify()
        print('Image is verified')

        for x in range(image.size[0]):
                for y in range(image.size[1]):
                        image.putpixel((x,y), (x,x,x))

        t = time.strftime('%b_%d_%H_%M_%S', time.localtime())
        image.save(root + t + ext)

finally:
    connection.close()
    server_socket.close()
```

Data Visualization & Website Design

The goal of the data visualization website is to provide the means to easily select and display data that the sensors have gathered in the past (historical data) or data currently being gathered (real time data). To accomplish this, several design choices have been made to streamline the process for both the sensor domain and the end user. It is important to note that the data is perhaps the most important part of the project, thus the project was built around it to accommodate its needs.

CURSORY GLANCE

Back End DBMS

In order to build around the data, a suitable database management software (DBMS) must be established to store it. The data collected by the sensors can theoretically can be managed by any number of DBMS’s. However, this project had many requirements that heavily swayed towards the use of MongoDB as the DBMS of choice. Specifically, the project required:

- Scalability: While the project initially was developed to collect data for one sensor, this is only to verify and validate. The project was developed in mind with the ability to handle thousands of sensors that each handle thousands of pieces of data per hour in addition to a few sensors each handling minimal loads.

- Server Side Querying: The ability to select complex attributes and generating information from them is an important aspect of this project; a database manager with a robust querying system is thus a must.
- Dynamic typing: The sensors the system is handling can be widely differing; for example, a sensor may pick up population data, while another may pick up only temperature data. As such, typing is difficult to address in a “schema-ful” database.

MongoDB, a NoSQL DBMS, satisfies all our requirements and allows us to manipulate our data as desired.

Web Server

Once the DBMS was chosen, the backend of the website was built around it, continuing with the “data rules” paradigm. Typically, the LAMP (Linux, Apache, MySQL, PHP), or more recently LEMP (Linux, NginX, MySQL, PHP) software stacks dominate the web landscape. At first, this conventional approach was taken to build the website, substituting “MySQL” with “MongoDB” in the LAMP stack. However, it soon became apparent that the LAMP stack was incompatible with MongoDB, specifically the Apache and PHP components. The decision to switch over to a different software stack was made and the substantial progress made on the LAMP platform was soon rewritten to a new platform more accommodating to MongoDB, MERN (MongoDB, Express, React.js, Node.js). This switch also had several advantages and streamlined development, significantly reducing overhead from the switch. Since all softwares on the MERN stack were written in the same language (Javascript), there was very little “mindset switching” and code conversion issues when decisions made to move components from one domain in the stack to another were made.

Front End UI

The sensor network architecture is very complex and sizeable, which made the task of providing a intuitive yet powerful interface a challenge.

Future: Sensor Interfacing with DBMS

DETAILED GLANCE AND CODE ANALYSIS

Database Querying

The decision to process data in the cloud vs. on device was integral to a smooth client experience and data security. There were many advantages to processing in either domain, namely for cloud processing:

- Security: Less user access to unauthorized data
- Usability: Less reliance on client hardware for processing, freeing up more resources for other tasks or power conservation
- Scalability: Less reliance on user for up to date software for more consistent user experience across multiple devices

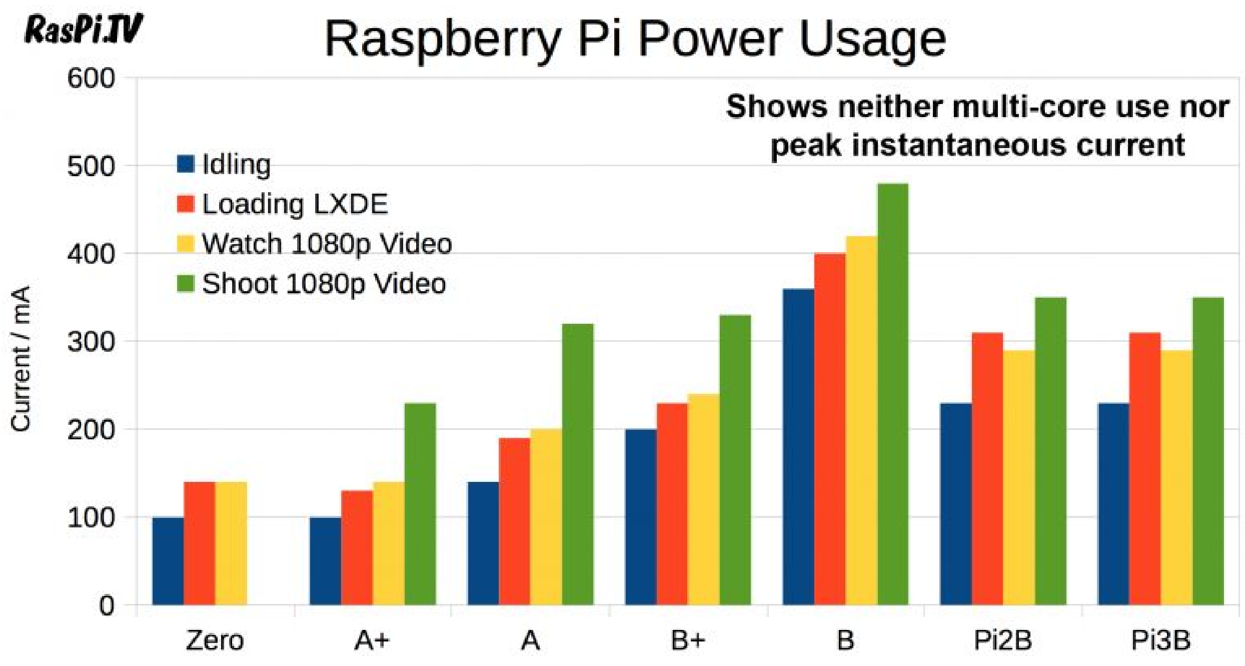
On the other hand, on device processing also had the following perks:

- Usability: More access to data for manipulation in the event of disconnection from server
- Resource utilization: More utility gained from distributed data processing on client hardware

The decision was made to process data in the cloud for the smoothness and security reasons. Realistically, client hardware would not justify distributed data processing for an on device processing approach and the security concern with unauthorized data reaching the client was too great to consider the usability gains for on device processing approach relevant.
This meant a number of database query requests have been programmed into the web server to accumulate the appropriate data.

Energy Harvesting research

As for energy harvesting, according to our research, the Pi with all the accessories and WIFI as well as multi-threading will probably take 0.5 Amps . And if we use a standard power bank on the market, with 20000mAh, which is related to 3.7V. After the calculation

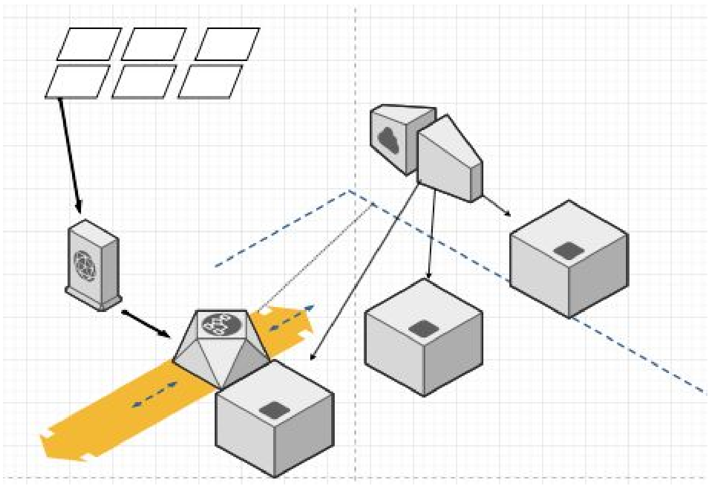


$Capacity = \frac{StatedCapacity * 3.7V}{5V}$ we can see that only after about 24 hours, the power will be dry out in the power bank. It’s not ideal to use such a power bank if you have to refill it everyday and we are now still using a wall plug to support the power for the raspberry Pi.

However, research at energy harvesting has been explored. The wireless charging to supply constant 0.5A with 5V load is totally feasible with a receiver setup that only weighs about 40 grams. The power for the transmitter of the wireless charging setup, can be from the wall plug, but that just solves the cord trouble without essentially saving the power for the device. On the other hand, the power can be supplied from the solar energy panel. With transmission efficiency tradeoff in mind, a 20w should be enough for the 2.5W output at the Pi end. But the setup is designed to be indoor, so the setup's idea is to use the solar energy panel to collect energy outside, converting it to a dc power transmitter indoor, and then wirelessly transferring to the device hanged in the ceiling.

Detailed wireless technology research can be found in this similar research https://docs.google.com/presentation/d/1X8ElVUhhut9i08JShnHOG0T47P6EflJJdBkAeUSc1nc/edit?usp=sharing

And an illustration graph can be drawn as following:



Where the panels receives the power and converts it to dc then the transmitter wirelessly transfer the energy to the receiver and then to the three boxes there.

Future work

For the future work,

- we are going to run more tests on the setup to see if it is accurate enough and maybe incorporate machine learning to improve the accuracy.

- We might want to incorporate the energy harvesting idea into the project to make this environmentally-friendly.

- We might want to put the box onto a robot and access it remotely by Augmented Reality and allow users to easily access the data not only with the website data visualization.

- We might want to reprogram the device to allow the device into motion and therefore allow a greater range of detection.

Reference

http://www.lovemytool.com/blog/2010/07/wireshark-wireless-display-and-capture-filters-samples-part-2-by-joke-snelders.html

http://www.lovemytool.com/blog/2010/07/wireshark-wireless-display-and-capture-filters-samples-part-2-by-joke-snelders.html

https://www.dexterindustries.com/howto/run-a-program-on-your-raspberry-pi-at-startup/

http://docs.opencv.org/2.4/doc/tutorials/tutorials.html

More can be found at source.txt

Appendix:Code Listed

https://github.com/ZhekaiJin/Cooper_IOT-