##### A MINOR PROJECT REPORT

ON

MOTION DETECTION SYSTEM

***In partial fulfilment for the award of the degree***

***Of***

**BACHELOR OF COMPUTER APPLICATIONS**



MAHARAJA SURAJMAL INSTITUTE

##### C-4, JANAK PURI,

**NEW DELHI - 110058**

##### December 2017

Submitted to:

Dr Meenal Dahiya

Assistant Professor

Department of Computer Science

Maharaja Surajmal Institute

Submitted By:

Divyanshu Sengar

# 

# ABSTRACT

*Motion Detection System, is a project created with the use of Raspberry Pi, which is a credit card sized computer, based on a 4 core ARM processor. The motive of this project is to utilize the Raspberry Pi as a surveillance/automation hub, which, in conjuncture with the camera module, would provide wireless monitoring capabilities over the internet in real-time. All programming for the facilitation of proposed features would be done on python, as it provides tight integration with the Raspbian OS, the native operating system for Raspberry Pi. The python IDE in the OS is very flexible, and works at a low level, thus allowing us to expand the system as and when required to accommodate more cameras, other sensors, and creating monitoring hubs for a multi-computer setup.*

Table of Contents

[DECLARATION 3](#__RefHeading___Toc5667_1809370720)

[ACKNOWLEDGEMENT 4](#__RefHeading___Toc5881_1809370720)

[ABSTRACT 5](#__RefHeading___Toc5883_1809370720)

[1. INTRODUCTION 8](#__RefHeading___Toc5477_1809370720)

[1.1 Background Information 9](#__RefHeading___Toc5479_1809370720)

[1.2 Reason to Choose Python and OpenCV 10](#__RefHeading___Toc5485_1809370720)

[1.3 Applications and Software Attributes 10](#__RefHeading___Toc5487_1809370720)

[1.4 Objectives of the Project 11](#__RefHeading___Toc5489_1809370720)

[2. METHODS AND MATERIALS USED 13](#__RefHeading___Toc5493_1809370720)

[2.1 Background Subtraction 14](#__RefHeading___Toc5495_1809370720)

[2.1.1 Software Requirements 14](#__RefHeading___Toc5497_1809370720)

[2.1.1.1 Python 14](#__RefHeading___Toc5499_1809370720)

[2.1.1.2 OpenCV 15](#__RefHeading___Toc5501_1809370720)

[2.1.1.3 Raspbian 16](#__RefHeading___Toc5503_1809370720)

2.1.1.4 MAKE 16

[2.1.2 Hardware Requirements 20](#__RefHeading___Toc5505_1809370720)

[2.1.3 External Libraries used for Development 20](#__RefHeading___Toc5507_1809370720)

2.1.3.1 Dropbox API [20](#__RefHeading___Toc5511_1809370720)

[2.1.3.2 NumPy 20](#__RefHeading___Toc5513_1809370720)

[2.2 Principles of Background Subtraction 21](#__RefHeading___Toc5515_1809370720)

2.2.1 Conventional Approaches 21

2.2.2 Using frame differencing 22

2.2.3 Mean Filter 22

2.2.4 Running Gaussian Average 23

[3. FUNCTIONING OF APPLICATION 27](#__RefHeading___Toc5517_1809370720)

[3.1 Installing OpenCV on Raspberry Pi 28](#__RefHeading___Toc5519_1809370720)

[3.2 Setting up Pi Camera 37](#__RefHeading___Toc5521_1809370720)

3.2.1 The JSON configuration File 38

3.2.2 Integration with Dropbox 39

[3.3 Main Code 41](#__RefHeading___Toc5523_1809370720)

[3.4 Screenshots 46](#__RefHeading___Toc5671_1809370720)

4. FUTURE SCOPE 47

5. CONCLUSION 49

6. REFRENCES 51

**1. INTRODUCTION**

## Background Information

Home security is a major concern. Annually, over 2,00,000 home thefts are reported in India. This equates to one home break-in every 18 seconds. Out of these 2,00,000 burglaries, less than 30,000 burglars are caught, due to the lack of evidence. However, even with the alarming number of thefts taking place, there are not enough precautions being taken by homeowners, which makes this one of the easiest crimes to pull off, with a very low chance of getting caught. The reason for this can be linked to the unwillingness of home owners to take steps to protect their homes. Most home surveillance systems start at ₹10,000+, and people are just not willing to invest this amount of money, which is often topped with annual subscription charges.

Over 70% of break-ins happen through front doors, during day hours, when homeowners may not be present, and is almost always perpetrated, with the help of, or, by domestic help hired by the resident. Yet, this cannot be ascertained without visual proof.

## 1.2 Reason to Choose Python and OpenCV

* Definitely decreases your development time.
* It's easier to debug.
* An Image is basically an nd-array for which Python's NumPy and SciPy.org modules can be used.
* The array manipulations in NumPy module are highly optimized for speed. The Multi-dimensional image processing (submodule in SciPy is concentrated for image processing.)
* You don’t have to worry about memory management directly.
* You can combine the conciseness of Python with high performance powers of C using Python: C-Extensions for Python. It can provide a speed boost of up to 100x or even higher compared to native Python code depending on your application.
* *Sometimes in large scale projects, not everyone is comfortable with Python and codes in C++, so one has to conform to the unified coding style.*

## 1.3 Applications and Software Attributes

While providing basic motion detection and recording abilities, the software can be extended to provide full network support, including Dropbox integration, so that the system can be monitored from anywhere across the world. Also, as the software utilizes OpenCV, powerful image processing can be done, and the software can be used to perform facial recognition, which can be then cross referenced to public records to know exactly who is in the camera. This greatly increases the capabilities of the software, without any additional expense on hardware costs.

The entire software is written in Python, and utilizes OpenCV and Dropbox APIs, along with the picamera libraries to use the Raspberry Pi camera. These libraries are open source, and can be freely implemented across any platform.  
Some of the applications are written as follows:

* **Energy Savings:** An actively monitored system can detect when a room or an area is not in use, and turn off additional appliances consuming electricity. This system can be implemented throughout the area of deployment, and can greatly reduce the carbon footprint of an area.
* **Access Control:** The Raspberry Pi can be given additional biometric scanning abilities, which can allow it to detect when an authorized person is about to enter a room. This ability can be used to prevent unauthorized access, without causing hindrance to authorized personnel, unlike conventional authentication protocols.
* **Pre-emptive actions:** The system can be programmed to perform a certain set of actions when a certain event occurs. This may include dialling a set of numbers, unlocking the garage door, etc.
* **Occupancy Counter:** This system can be effectively used to count the number of people inside a given area. This functionality is useful when data is required regarding how many people attended a certain event is required, additional statistics, etc.

As well as being flexible and expandable, this project is highly marketable, as equally priced offerings in the market don’t have as many features, while those having equivalent features are much more expensive.

## 1.4 Objectives of the Project

The objective of our project is to provide a low cost, extensible surveillance platform, which can be easily implemented by anyone, with zero to minimal operational costs, and a one-time hardware investment. The requirement of such a system is high, and with a cost considerably less than other systems available in market, it’s easily sellable. As there are no recurring costs, more people will be willing to buy it.

The primary reason for building it as a DIY project is the cost advantage. All the necessary hardware can be purchased for under ₹5000. Given the open nature of the Raspberry Pi platform, the build can be further improved, and additional features can be implemented as and when required.

1. Open platform: Raspberry Pi runs on Raspbian, which is an Open Source Linux distribution. Any and all parts of the operating system can be modified to allow for extensibility and features.
2. Network compatibility: As the Raspberry Pi is Wi-Fi enabled, the whole setup can be put anywhere with minimal wiring requirements.

# 

# 2. METHODS AND MATERIALS USED

**2.** In order to fulfil the objectives that have been spelled out in the previous chapter, an exploratory research was carried out. To successfully execute Motion Detection, we need to use the principle of background subtraction implemented using Python and OpenCV on the Raspberry Pi and a Raspberry Camera

## 2.1 To utilise the software to implement background subtraction

### 2.1.1 Software Requirements

The following is the software requirements for the project:

* Python
* OpenCV
* Raspbian OS
* MAKE

##### 2.1.1.1 Python

Python is a high-level, interpreted and general-purpose dynamic programming language that focuses on code readability. The syntax in Python helps the programmers to do coding in fewer steps as compared to Java or C++.

The Python language has diversified application in the software development companies such as in gaming, web frameworks and applications, language development, prototyping, graphic design applications, etc. This provides the language a higher plethora over other programming languages used in the industry. Some of its advantages are:

* ***Extensive Support Libraries***

It provides large standard libraries that include the areas like string operations, Internet, web service tools, operating system interfaces and protocols. Most of the highly used programming tasks are already scripted into it that limits the length of the codes to be written in Python.

* **Integration Feature**

Python integrates the Enterprise Application Integration that makes it easy to develop Web services by invoking COM or COBRA components. It has powerful control capabilities as it calls directly through C, C++ or Java via Python. Python also processes XML and other mark-up languages as it can run on all modern operating systems through same byte code.

* ***Improved Programmer’s Productivity***

The language has extensive support libraries and clean object-oriented designs that increase two to tenfold of programmer’s productivity while using the languages like Java, VB, Perl, C, C++ and C#.

* ***Productivity***

With its strong process integration features, unit testing framework and enhanced control capabilities contribute towards the increased speed for most applications and productivity of applications. It is a great option for building scalable multi-protocol network applications.

##### 2.1.1.2 OpenCV

OpenCV (Open Source Computer Vision Library) is released under a BSD license and hence it’s free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform.

Adopted all around the world, OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. Usage ranges from interactive art, to mines inspection, stitching maps on the web or robotics.

**Advantages:**

* First and foremost, OpenCV is available **free** of cost
* Since OpenCV library is written in C/C++ it is quite **fast**
* **Low RAM usage** (approx. 60–70 MB)
* It is **portable** as OpenCV can run on any device that can run C

##### 2.1.1.3 Raspbian OS

Raspbian is a free operating system based on Debian optimized for the Raspberry Pi hardware. An operating system is the set of basic programs and utilities that make your Raspberry Pi run. However, Raspbian provides more than a pure OS: it comes with over 35,000 packages, pre-compiled software bundled in a nice format for easy installation on your Raspberry Pi.

The initial build of over 35,000 Raspbian packages, optimized for best performance on the Raspberry Pi, was completed in June of 2012. However, Raspbian is still under active development with an emphasis on improving the stability and performance of as many Debian packages as possible.

Raspbian uses PIXEL, Pi Improved Xwindows Environment, Lightweight as its main desktop environment as of the latest update. It is composed of a modified LXDE desktop environment and the Openbox stacking window manager with a new theme and few other changes. The distribution is shipped with a copy of computer algebra program Mathematica and a version of Minecraft called Minecraft PI as well as a lightweight version of Chromium as of the latest version. The initial build was completed in June 2012. The operating system is still under active development. Raspbian is highly optimized for the Raspberry Pi line's low-performance ARM CPUs.

**2.1.1.4 MAKE**

In software development, Make is a build automation tool that automatically builds executable programs and libraries from source code by reading files called Makefiles which specify how to derive the target program. Though integrated development environments and language-specific compiler features can also be used to manage a build process, Make remains widely used, especially in UNIX and Unix-like operating systems.

Besides building programs, Make can be used to manage any project where some files must be updated automatically from others whenever the others change.  
  
***Origin***

There are now a number of dependency-tracking build utilities, but Make is one of the most widespread, primarily due to its inclusion in Unix, starting with the PWB/UNIX 1.0, which featured a variety of tools targeting software development tasks. It was originally created by Stuart Feldman in April 1976 at Bell Labs. Feldman received the 2003 ACM Software System Award for the authoring of this widespread tool.

Feldman was inspired to write Make by the experience of a co-worker in futilely debugging a program of his where the executable was accidentally not being updated with changes:

Make originated with a visit from Steve Johnson (author of yacc, etc.), storming into my office, cursing the Fates that had caused him to waste a morning debugging a correct program (bug had been fixed, file hadn't been compiled, cc \*.o was therefore unaffected).

“As I had spent a part of the previous evening coping with the same disaster on a project I was working on, the idea of a tool to solve it came up. It began with an elaborate idea of a dependency analyzer, boiled down to something much simpler, and turned into Make that weekend. Use of tools that were still wet was part of the culture. Makefiles were text files, not magically encoded binaries, because that was the Unix ethos: printable, debuggable, understandable stuff."

— Stuart Feldman, The Art of Unix Programming, Eric S. Raymond 2003

Before Make's introduction, the Unix build system most commonly consisted of operating system dependent "make" and "install" shell scripts accompanying their program's source. Being able to combine the commands for the different targets into a single file and being able to abstract out dependency tracking and archive handling was an important step in the direction of modern build environments.  
  
  
***Derivatives***

Make has gone through a number of rewrites, including a number of from-scratch variants which used the same file format and basic algorithmic principles and also provided a number of their own non-standard enhancements. Some of them are:

Sun DevPro make was a rewrite of the UNIX make program that appeared in 1986 with SunOS-3.2. With SunOS-3.2, DevPro make was delivered as optional program; with SunOS-4.0, SunPro make was made the default make program. In December 2006, Sun DevPro make was made OpenSource as part of the efforts to open-source Solaris, in order to regain lost market share.

Dmake or Distributed Make that came with Sun Solaris Studio as its default make, but not the default one on the Solaris Operating System (SunOS). It was originally required to build OpenOffice, but in 2009 the build system was rewritten to use GNU make. While Apache OpenOffice still contains a mixture both build systems, the much more actively developed LibreOffice only uses the modernized "gbuild" now.

BSD Make (pmake, bmake or fmake), which is derived from Adam de Boor's work on a version of Make capable of building targets in parallel, and survives with varying degrees of modification in FreeBSD, NetBSD and OpenBSD. Distinctively, it has conditionals and iterative loops which are applied at the parsing stage and may be used to conditionally and programmatically construct the makefile, including generation of targets at runtime.

GNU Make (short gmake) is the standard implementation of make for Linux and OS X. It provides several extensions over the original make, such as conditionals. It also provides many built-in functions which can be used to eliminate the need for shell-scripting in the makefile rules as well as to manipulate the variables set and used in the makefile. For example, the foreach function sets a variable to the list of all files in a given directory. GNU Make has been required for building gcc since version 3.4. It is required for building the Linux kernel, Apache OpenOffice and LibreOffice, Mozilla Firefox and many others.

Rocky Bernstein's remake is a fork of GNU make and provides several extensions over GNU make, such as better location and error-location reporting, execution tracing, execution profiling, and it contains a debugger.

Glenn Fowler's nmake is unrelated to the Microsoft program of the same name. Its input is similar to make, but not compatible. This program provides shortcuts and built-in features, which according to its developers reduces the size of makefiles by a factor of 10.

Microsoft nmake, a command-line tool which normally is part of Visual Studio. It supports pre-processor directives such as includes and conditional expressions which use variables set on the command-line or within the makefiles. Inference rules differ from make; for example they can include search paths. The make tool supplied with Embarcadero products has a command-line option that "Causes MAKE to mimic Microsoft's NMAKE."

Mk replaced Make in Research Unix, starting from version 9. A redesign of the original tool by Bell Labs programmer Andrew G. Hume, it features a different syntax. Mk became the standard build tool in Plan 9, Bell Labs' intended successor to Unix.

POSIX includes standardization of the basic features and operation of the Make utility, and is implemented with varying degrees of completeness in Unix-based versions of Make. In general, simple makefiles may be used between various versions of Make with reasonable success. GNU Make, BSD Make and Makepp can be configured to look first for files named "GNUmakefile", "BSDmakefile" and "Makeppfile" respectively, which allows one to put makefiles which use implementation-defined behaviour in separate locations.

### 2.1.2 Hardware requirements

The following is the hardware requirements for the project:

* Raspberry Pi 3 with inbuilt Wi-Fi
* Raspberry Pi Camera

### 2.1.3 External Libraries Used in Development

#### **2.1.3.1** **Dropbox API**

#### Dropbox is a storage service that syncs files online and across your computers via installed software. The API allows read/write access to the same files, piggybacking on Dropbox's file synchronization and storage.

Some of the features of Version 1 are as follows:

* **Increased security:** You’ll never have to enter your Dropbox account info into a third-party app on version 1. Instead, you’ll approve access using the official Dropbox app on mobile or via the web at dropbox.com.
* **App folders:** Version 1 adds support for apps that can only read or write to a single folder in your Dropbox. You can rename or move this App Folder wherever you want in your Dropbox, and the app will keep working normally.
* **Search, revisions, and more:** We’ve added a lot of new calls to the API to make apps even more powerful. Developers can now build search and sharing right into their apps. Version 1 also exposes the full power of the Dropbox revision system. This includes undeleting files, reverting files to past versions, and not overwriting changes when two people update a file at the same time.
* **More supported platforms:** Version 0 of the API was designed for mobile apps. While they’re still getting plenty of love, we’ve now added better support for web apps! This includes new tutorials, documentation, and SDKs for Python, Ruby, and Java developers.

# 2.1.3.2 NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

* a powerful N-dimensional array object
* sophisticated (broadcasting) functions
* tools for integrating C/C++ and Fortran code
* useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

NumPy is licensed under the BSD license, enabling reuse with few restrictions.

**2.2 Principle of Background Subtraction to implement Motion Detection**

**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 foreground. After the stage of image pre-processing (which may include image de-noising, post processing like morphology etc.) object localisation is required which may make use of this technique.

Background subtraction is a widely used approach for detecting moving objects in videos from static cameras. The rationale in the approach is that of detecting the moving objects from the difference between the current frame and a reference frame, often called "background image", or "background model". Background subtraction is mostly done if the image in question is a part of a video stream. Background subtraction provides important cues for numerous applications in computer vision, for example surveillance tracking or human poses estimation.

Background subtraction is generally based on a static background hypothesis which is often not applicable in real environments. With indoor scenes, reflections or animated images on screens lead to background changes. Similarly, due to wind, rain or illumination changes brought by weather, static backgrounds methods have difficulties with outdoor scenes.

## 2.2.1 Conventional Approaches

A robust background subtraction algorithm should be able to handle lighting changes, repetitive motions from clutter and long-term scene changes. The following analyses make use of the function of *V*(*x*,*y*,*t*) as a video sequence where *t* is the time dimension, *x* and *y* are the pixel location variables. E.g. *V*(1,2,3) is the pixel intensity at (1,2) pixel location of the image at *t* = 3 in the video sequence.

### 2.2.2 Using frame differencing

A motion detection algorithm begins with the segmentation part where foreground or moving objects are segmented from the background. The simplest way to implement this is to take an image as background and take the frames obtained at the time t, denoted by I(t) to compare with the background image denoted by B. Here using simple arithmetic calculations, we can segment out the objects simply by using image subtraction technique of computer vision meaning for each pixels in I(t), take the pixel value denoted by P[I(t)] and subtract it with the corresponding pixels at the same position on the background image denoted as P[B].

In mathematical equation, it is written as:

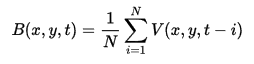
P [ F ( t ) ] = P [ I ( t ) ] − P [ B ] {\displaystyle P[F(t)]=P[I(t)]-P[B]} e1

C:\Users\SirVM\AppData\Local\Microsoft\Windows\INetCache\Content.Word\e2.pngThe background is assumed to be the frame at time *t*. This difference image would only show some intensity for the pixel locations which have changed in the two frames. Though we have seemingly removed the background, this approach will only work for cases where all foreground pixels are moving and all background pixels are static. A threshold "Threshold" is put on this difference image to improve the subtraction.

| P [ F ( t ) ] − P [ F ( t + 1 ) ] | > T h r e s h o l d {\displaystyle |P[F(t)]-P[F(t+1)]|>\mathrm {Threshold} }

This means that the difference image's pixels' intensities are 'thresholded' or filtered on the basis of value of Threshold. The accuracy of this approach is dependent on speed of movement in the scene. Faster movements may require higher thresholds.

### 2.2.3 Mean filter

For calculating the image containing only the background, a series of preceding images are averaged. For calculating the background image at the instant *t*,

B ( x , y , t ) = 1 N ∑ i = 1 N V ( x , y , t − i ) {\displaystyle B(x,y,t)={1 \over N}\sum \_{i=1}^{N}V(x,y,t-i)}

where *N* is the number of preceding images taken for averaging. This averaging refers to averaging corresponding pixels in the given images. *N* would depend on the video speed (number of images per second in the video) and the amount of movement in the video. After calculating the background *B(x,y,t)* we can then subtract it from the image *V(x,y,t)* at time *t*=t and threshold it. Thus the foreground is

| V ( x , y , t ) − B ( x , y , t ) | > T h {\displaystyle |V(x,y,t)-B(x,y,t)|>\mathrm {Th} } e4

where *Th* is threshold. Similarly we can also use median instead of mean in the above calculation of *B*(*x*,*y*,*t*).

Usage of global and time-independent Thresholds (same Th value for all pixels in the image) may limit the accuracy of the above two approaches.

### 2.2.4 Running Gaussian average

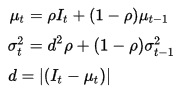
For this method, Wren et al. propose fitting a Gaussian probabilistic density function (PDF) on the most recent n {\displaystyle n} frames. In order to avoid fitting the pdf from scratch at each new frame time t {\displaystyle t} t, a running (or on-line cumulative) average is computed.

The pdf of every pixel is characterized by meane5μ t {\displaystyle \mu \_{t}} and varianceC:\Users\SirVM\AppData\Local\Microsoft\Windows\INetCache\Content.Word\e6.pngσ t 2 {\displaystyle \sigma \_{t}^{2}} . The following is a possible initial condition (assuming that initially every pixel is background):

e7

where I t {\displaystyle I\_{t}} Captureis the value of the pixel's intensity at time t {\displaystyle t} t. In order to initialize variance, we can, for example, use the variance in x and y from a small window around each pixel.

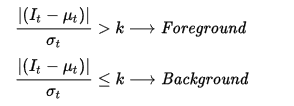
Note that background may change over time (e.g. due to illumination changes or non-static background objects). To accommodate for that change, at every frame t {\displaystyle t} t, every pixel's mean and variance must be updated, as follows:

μ t = ρ I t + ( 1 − ρ ) μ t − 1 {\displaystyle \mu \_{t}=\rho I\_{t}+(1-\rho )\mu \_{t-1}} σ t 2 = d 2 ρ + ( 1 − ρ ) σ t − 1 2 {\displaystyle \sigma \_{t}^{2}=d^{2}\rho +(1-\rho )\sigma \_{t-1}^{2}} 

d = | ( I t − μ t ) | {\displaystyle d=|(I\_{t}-\mu \_{t})|} Where e10ρ {\displaystyle \rho } e9determines the size of the temporal window that is used to fit the PDF (usually ρ = 0.01 {\displaystyle \rho =0.01} e10) and d {\displaystyle d} is the Euclidean distance between the mean and the value of the pixel.

*d*

We can now classify a pixel as background if its current intensity lies within some confidence interval of its distribution's mean:

| ( I t − μ t ) | σ t > k ⟶ F o r e g r o u n d {\displaystyle {\frac {|(I\_{t}-\mu \_{t})|}{\sigma \_{t}}}>k\longrightarrow {\mathit {Foreground}}} | ( I t − μ t ) | σ t ≤ k ⟶ B a c k g r o u n d {\displaystyle {\frac {|(I\_{t}-\mu \_{t})|}{\sigma \_{t}}}\leq k\longrightarrow {\mathit {Background}}} 

where the parameter k {\displaystyle k} k is a free threshold (usually k = 2.5 {\displaystyle k=2.5} k=2.5). A larger value for k {\displaystyle k} k allows for more dynamic background, while a smaller k {\displaystyle k} k increases the probability of a transition from background to foreground due to more subtle changes.

In a variant of the method, a pixel's distribution is only updated if it is classified as background. This is to prevent newly introduced foreground objects from fading into the background. The update formula for the mean is changed accordingly:

μ t = M μ t − 1 + ( 1 − M ) ( I t ρ + ( 1 − ρ ) μ t − 1 ) {\displaystyle \mu \_{t}=M\mu \_{t-1}+(1-M)(I\_{t}\rho +(1-\rho )\mu \_{t-1})} e12

where M = 1 {\displaystyle M=1} M=1 whenI t {\displaystyle I\_{t}} Captureis considered foreground and M = 0 {\displaystyle M=0} M=0 otherwise. So when M = 1 {\displaystyle M=1} M=1, that is, when the pixel is detected as foreground, the mean will stay the same. As a result, a pixel, once it has become foreground, can only become background again when the intensity value gets close to what it was before turning foreground. This method, however, has several issues: It only works if all pixels are initially background pixels (or foreground pixels are annotated as such). Also, it cannot cope with gradual background changes: If a pixel is categorized as foreground for a too long period of time, the background intensity in that location might have changed (because illumination has changed etc.). As a result, once the foreground object is gone, the new background intensity might not be recognized as such anymore.

# 

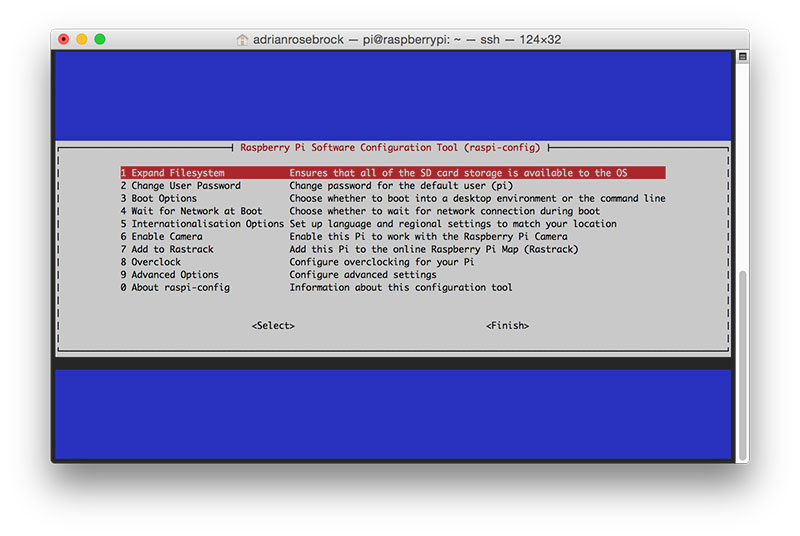
# 3. FUNCTIONING OF THE APPLICATION

## 3.1 Installing OpenCV and Python on the Raspberry Pi

### *Step #1: Expand filesystem*

Expand the filesystem to include the entire space of the SD card if using the latest version of Raspbian Jesse.

**$sudo raspi-config**



**Figure 1:** Expanding the filesystem on your Raspberry Pi 3.

Once prompted, select the first option, ***“1. Expand File System”***, ***hit Enter*** on your keyboard, arrow down to the ***“<Finish>”*** button, and then reboot the Pi:

|  |  |
| --- | --- |
|  | **$sudo reboot** |

After rebooting, the file system should have been expanded to include all available space on your micro-SD card. We can verify that the disk has been expanded by executing df-h  and examining the output:

**$df-h**

**Filesystem      Size  Used Avail Use%Mounted on**

**/dev/root       7.2G  3.3G  3.6G  48%/**

**devtmpfs        459M     0  459M   0%/dev**

**tmpfs           463M     0  463M   0%/dev/shm**

**tmpfs           463M  6.4M  457M   2%/run**

**tmpfs           5.0M  4.0K  5.0M   1%/run/lock**

**tmpfs           463M     0  463M   0%/sys/fs/cgroup**

**/dev/mmcblk0p1   60M   20M   41M  34%/boot**

**tmpfs 93M 0 93M 0%/run/user/1000**

### *Step #2: Install dependencies*

The first step is to update and upgrade any existing packages:

**$sudo apt-getupdate**

**$sudo apt-getupgrade**

We then need to install some developer tools, including [CMake](https://cmake.org/), which helps us configure the OpenCV build process:

**$sudo apt-getinstall build-essential cmake pkg-config**

***Timing: 40s***

We will then be installing some image I/O packages that allow us to load various image file formats from disk. Examples of such file formats include JPEG, PNG, TIFF, etc.:

**$sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev**

***Timing: 32s***

Just as we need image I/O packages, we also need video I/O packages. These libraries allow us to read various video file formats from disk as well as work directly with video streams:

**$sudo apt-getinstall libavcodec-dev libavformat-dev libswscale-dev libv4l-dev**

**$sudo apt-getinstall libxvidcore-dev libx264-dev**

***Timing: 34s***

The OpenCV library comes with a sub-module named highgui  which is used to display images to our screen and build basic GUIs. In order to compile the highgui  module, we need to install the GTK development library:

**$sudo apt-getinstall libgtk2.0-dev**

***Timing: 3m 6s***

Many operations inside of OpenCV (namely matrix operations) can be optimized further by installing a few extra dependencies:

**$sudo apt-getinstall libatlas-base-dev gfortran**

***Timing: 46s***

These optimization libraries are especially important for resource constrained devices such as the Raspberry Pi.

Now, to install Python 2.7 and Python 3 so we can compile OpenCV with Python bindings:

**$sudo apt-getinstall python2.7-dev python3-dev**

***Timing: 45s***

### *Step #3: Download the OpenCV source code*

After installing the necessary dependencies,we have to install the latest archive of OpenCV (3.1.0,in this case) from the official OpenCV repository.

**$cd~**

**$wget-Oopencv.ziphttps://github.com/Itseez/opencv/archive/3.1.0.zip**

**$unzip opencv.zip**

***Timing: 1m 26s***

To have access to features such as SIFT and SURFin OpenCV 3, we also need to grab the [opencv\_contrib](https://github.com/itseez/opencv_contrib) repository as well:

**$wget Oopencv\_contrib.ziphttps://github.com/Itseez/opencv\_contrib/archive /3.1.0.zip**

**$unzip opencv\_contrib.zip**

***Timing: 43s***

You might need to expand the command above using the “<=>” button during your copy and paste. The .zip  in the 3.1.0.zip  may appear to be cutoff in some browsers. The full URL of the OpenCV 3.1.0 archive is:

https://github.com/Itseez/opencv\_contrib/archive/3.1.0.zip

### *Step #4: Python 2.7 or Python 3?*

Before we can start compiling OpenCV on our Raspberry Pi 3, we first need to install pip, a Python package manager:

**$wget https://bootstrap.pypa.io/get-pip.py**

***Timing: 20s***

Now we need to install both virtualenv and virtualwrapper :

**$sudo pip install virtualenv virtualenvwrapper**

**$sudo rm-rf~/.cache/pip**

***Timing: 9s***

Now that both virtualenv and virtualenvwrapper have been installed, we need to update our ~/.profile file to include the following lines at the bottom of the file:

**# virtualenv and virtualenvwrapper**

**export WORKON\_HOME=$HOME/.virtualenvs**

**source/usr/local/bin/virtualenvwrapper.sh**

After updating our~/.profile, we need to reload it to make sure the changes take effect. You can force a reload of your ~/.profile file by:

1. Logging out and then logging back in.
2. Closing a terminal instance and opening up a new one
3. Or our personal favourite, ***just use the source command:***

$source~/.profile

**Creating the Python Virtual Environment**

Next, let’s create the Python virtual environment that we’ll use for computer vision development:

**$mkvirtualenv cv-ppython2**

This command will create a new Python virtual environment named cv using ***Python 2.7***.

If you instead want to use ***Python 3***, you’ll want to use this command instead:

**$mkvirtualenv cv-ppython3**

|  |  |
| --- | --- |
|  |  |

### Installing NumPy on your Raspberry Pi

Our only Python dependency is [NumPy](http://www.numpy.org/), a Python package used for numerical processing:

**$pip install numpy**

***Timing: 9m 39s***

### *Step #5: Compile and Install OpenCV*

OpenCV can finally be compiled together. Double-check that you are in the cv\_virtual environment by examining your prompt (you should see the (cv) text preceding it), and if not, simply execute workon:

**$workon cv**

Once you have ensured you are in the cv virtual environment, we can setup our build using CMake:

**$cd~/opencv-3.1.0/**

**$mkdirbuild**

**$cdbuild**

**$cmake-DCMAKE\_BUILD\_TYPE=RELEASE\**

**-DCMAKE\_INSTALL\_PREFIX=/usr/local\**

**-DINSTALL\_PYTHON\_EXAMPLES=ON\**

**-DOPENCV\_EXTRA\_MODULES\_PATH=~/opencv\_contrib-3.1.0/modules\**

**-DBUILD\_EXAMPLES=ON..**

***Timing: 1m 57s***

OpenCV compilation can finally be done :

**$make-j4**

***Timing: 1h 12m***

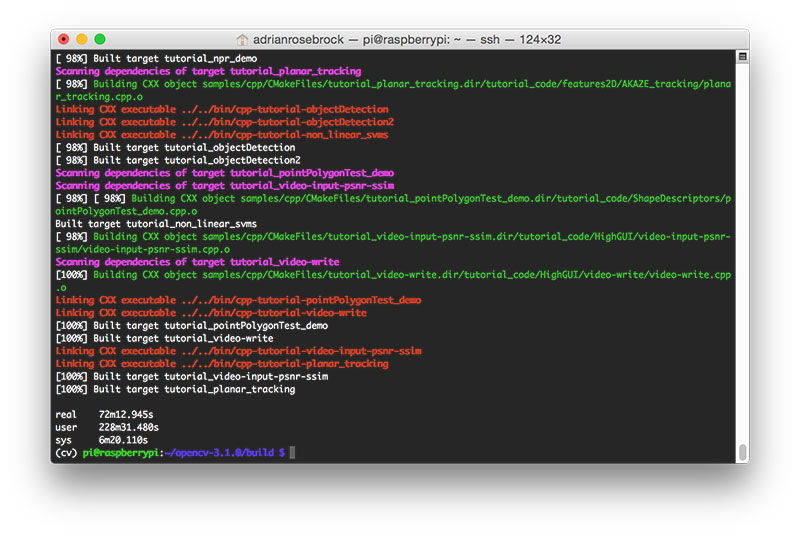
The -j4  command controls the number of cores to leverage when compiling OpenCV 3. The Raspberry Pi 3 has four cores, thus we supply a value of 4 to allow OpenCV to compile faster.

However, due to race conditions, there are times when make errors out when using multiple cores. If this happens to you, I suggest starting the compilation over again and using only one core.

**$makeclean**

**$make**

Once OpenCV 3 has finished compiling, the output should look:



**Figure 5:** Our OpenCV 3 compile on Raspbian Jessie has completed successfully.

Now all that’s left is to install OpenCV 3 on your Raspberry Pi 3:

**$sudo makeinstall**

**$sudo ldconfig**

***Timing: 52s***

### *Step #6: Finish installing OpenCV on your Pi*

We’re almost done — just a few more steps to go and you’ll be ready to use your Raspberry Pi 3 with OpenCV 3.

#### For Python 2.7:

Provided your **Step #5** finished without error, OpenCV should now be installed in /usr/local/lib/python2.7/site-pacakges . You can verify this using the ls  command:

**$ls-l/usr/local/lib/python2.7/site-packages/**

**total1852**

**-rw-r--r--1root staff1895772Mar2020:00cv2.so**

Our final step is to sym-link the OpenCV bindings into our cv  virtual environment for Python 2.7:

**$cd~/.virtualenvs/cv/lib/python2.7/site-packages/**

**$ln-s/usr/local/lib/python2.7/site-packages/cv2.socv2.so**

|  |  |
| --- | --- |
|  |  |

#### For Python 3:

After running make install , the OpenCV + Python bindings should be installed in /usr/local/lib/python3.4/site-packages . Again, you can verify this with the ls  command:

**$ls-l/usr/local/lib/python3.4/site-packages/**

**total1852**

**-rw-r--r--1root staff1895932Mar2021:51cv2.cpython-34m.so**

we can sym-link our OpenCV bindings into the cv  virtual environment for Python 3.4:

**$cd~/.virtualenvs/cv/lib/python3.4/site-packages/**

**$ln-s/usr/local/lib/python3.4/site-packages/cv2.socv2.so**

### *Step #7: Testing your OpenCV 3 install*

Open up a new terminal, execute the source  and workon  commands, and then finally attempt to import the Python + OpenCV bindings:

**$source~/.profile**

**$workon cv**

**$python**

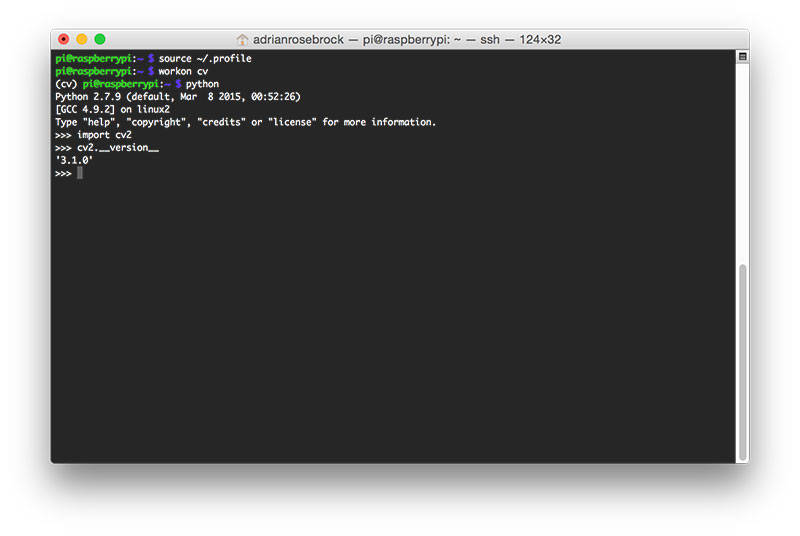
**>>>import cv2**

**>>>cv2.\_\_version\_\_**

**'3.1.0'**

**>>>**

From the below shot, we can see that, **OpenCV 3 has been successfully installed on our Raspberry Pi 3 + Python 2.7 environment:**



**Figure 5:** Confirming OpenCV 3 has been successfully installed on my Raspberry Pi 3 running Raspbian Jessie.

Once OpenCV has been installed, you can remove both the opencv-3.1.0  and opencv\_contrib-3.1.0  directories to free up a bunch of space on your disk:

**$rm-rf opencv-3.1.0opencv\_contrib-3.1.0**

## 3.2 Setting up the Pi Camera using Python for Surveillance

We’ll start by taking a look at the directory structure of our project:

Home surveillance and motion detection with the Raspberry Pi, Python, and OpenCV

**|---pi\_surveillance.py**

**|---conf.json**

**|---pyimagesearch**

**|    |---\_\_init\_\_.py**

**|    |---tempimage.py**

Our main home surveillance code and logic will be stored in pi\_surveillance.py. And instead of using command line arguments or hardcoding values inside the pi\_surveillance.py file, we’ll instead use a JSON configuration file named conf.json .

Finally, we’ll define a pyimagesearch  package for organization purposes, which will house a single class, TempImage , which we’ll use to temporarily write images to disk before they are shipped off to Dropbox.

So with the directory structure of our project in mind, open up a new file, name it pi\_surveillance.py, and start by importing the following packages:

Home surveillance and motion detection with the Raspberry Pi, Python, and OpenCV.

**# import the necessary packages**

**frompyimagesearch.tempimage importTempImage**

**frompicamera.arrayimportPiRGBArray**

**frompicamera importPiCamera**

**importargparse**

**importwarnings**

**importdatetime**

**importdropbox**

**importimutils**

**importjson**

**importtime**

**importcv2**

**# construct the argument parser and parse the arguments**

**ap=argparse.ArgumentParser()**

**ap.add\_argument("-c","--conf",required=True,**

**help="path to the JSON configuration file")**

**args=vars(ap.parse\_args())**

**# filter warnings, load the configuration and initialize the Dropbox**

**# client**

**warnings.filterwarnings("ignore")**

**conf=json.load(open(args["conf"]))**

**client=None**

The first import statement simply imports our TempImage class from the PyImageSearch package. **Lines 3-4** import classes from picamera that will allow us to access the raw video stream of the Raspberry Pi camera. And then **Line 8** grabs the Dropbox API. The remaining import statements round off the other packages we’ll need.

**Lines 15-18** handle parsing our command line arguments. All we need is a single switch, --conf, which is the path to where our JSON configuration file lives on disk.

**Line 22** filters warning notifications from Python, specifically ones generated from urllib3 and the dropbox packages. And lastly, we’ll load our JSON configuration dictionary from disk on **Line 23** and initialize our Dropbox client on **Line 24**.

## 3.2.1 The JSON configuration file

**{**

**"show\_video":true,**

**"use\_dropbox":true,**

**"dropbox\_access\_token":"YOUR\_DROPBOX\_KEY",**

**"dropbox\_base\_path":"YOUR\_DROPBOX\_PATH",**

**"min\_upload\_seconds":3.0,**

**"min\_motion\_frames":8,**

**"camera\_warmup\_time":2.5,**

**"delta\_thresh":5,**

**"resolution":[640,480],**

**"fps":16,**

**"min\_area":5000**

**}**

This JSON configuration file stores a bunch of important variables, which are :

* **show\_video :** A boolean indicating whether or not the video stream from the Raspberry Pi should be displayed to our screen.
* **use\_dropbox :** Boolean indicating whether or not the Dropbox API integration should be used.
* **dropbox\_access\_token :** Your public Dropbox API key.
* **dropbox\_base\_path :** The name of your Dropbox App directory that will store uploaded images.
* **min\_upload\_seconds :** The number of seconds to wait in between uploads. For example, if an image was uploaded to Dropbox 5m 33s after starting our script, a second image would not be uploaded until 5m 36s. This parameter simply controls the frequency of image uploads.
* **min\_motion\_frames :** The minimum number of **consecutive** **frames** containing motion before an image can be uploaded to Dropbox.
* **camera\_warmup\_time :** The number of seconds to allow the Raspberry Pi camera module to “warmup” and calibrate.
* **delta\_thresh :** The minimum absolute value difference between our current frame and averaged frame for a given pixel to be “triggered” as motion. Smaller values will lead to more motion being detected, larger values to less motion detected.
* **resolution :** The width and height of the video frame from our Raspberry Pi camera.
* **fps :** The desired Frames Per Second from our Raspberry Pi camera.
* **min\_area :** The minimum area size of an image (in pixels) for a region to be considered motion or not. Smaller values will lead to more areas marked as motion, whereas higher values of min\_area  will only mark larger regions as motion.

## 3.2.2 Integrating with Dropbox

If we want to integrate with the Dropbox API, we first need to setup our client:

**# check to see if the Dropbox should be used**

**ifconf["use\_dropbox"]:**

**# connect to dropbox and start the session authorization process**

**client=dropbox.Dropbox(conf["dropbox\_access\_token"])**

**print("[SUCCESS] dropbox account linked")**

On **Line 27** we make a check to our JSON configuration to see if Dropbox should be used or not. If it should, **Line 29** authorizes our app with the API key.

**At this point it is important that you have edited the configuration file with your API key and Path**. To find your API key, you can create an app on the app creation page on the Dropbox site. Once you have an app created, the API key may be generated under the OAuth section of the app’s page on the App Console (simply click the “Generate” button and copy/paste the key into the configuration file).

## 3.3 Home surveillance and motion detection with the Raspberry Pi: Main Code:

## 

|  |  |
| --- | --- |
| **32**  **33**  **34**  **35**  **36**  **37**  **38**  **39**  **40**  **41**  **42**  **43**  **44** | **# initialize the camera and grab a reference to the raw camera capture**  **camera=PiCamera()**  **camera.resolution=tuple(conf["resolution"])**  **camera.framerate=conf["fps"]**  **rawCapture=PiRGBArray(camera,size=tuple(conf["resolution"]))**    **# allow the camera to warmup, then initialize the average frame, last**  **# uploaded timestamp, and frame motion counter**  **print("[INFO] warming up...")**  **time.sleep(conf["camera\_warmup\_time"])**  **avg=None**  **lastUploaded=datetime.datetime.now()**  **motionCounter=0** |

We setup our raw capture to the Raspberry Pi camera on **Lines 33-36**.

We’ll also allow the Raspberry Pi camera module to warm up for a few seconds, ensuring that the sensors are given enough time to calibrate. Finally, we’ll initialize the average background frame, along with some bookkeeping variables on **Lines 42-44**.

Now we start looping over frames directly from our Raspberry Pi video stream:

**# capture frames from the camera**

**forfincamera.capture\_continuous(rawCapture,format="bgr",use\_video\_port=True):**

**# grab the raw NumPy array representing the image and initialize**

**# the timestamp and occupied/unoccupied text**

**frame=f.array**

**timestamp=datetime.datetime.now()**

**text="Unoccupied"**

**# resize the frame, convert it to grayscale, and blur it**

**frame=imutils.resize(frame,width=500)**

**gray=cv2.cvtColor(frame,cv2.COLOR\_BGR2GRAY)**

**gray=cv2.GaussianBlur(gray,(21,21),0)**

**# if the average frame is None, initialize it**

**ifavg isNone:**

**print("[INFO] starting background model...")**

**avg=gray.copy().astype("float")**

**rawCapture.truncate(0)**

**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))**

We pre-process our frame a bit by resizing it to have a width of 500 pixels, followed by converting it to grayscale, and applying a Gaussian blur to remove high frequency noise and allowing us to focus on the “structural” objects of the image.

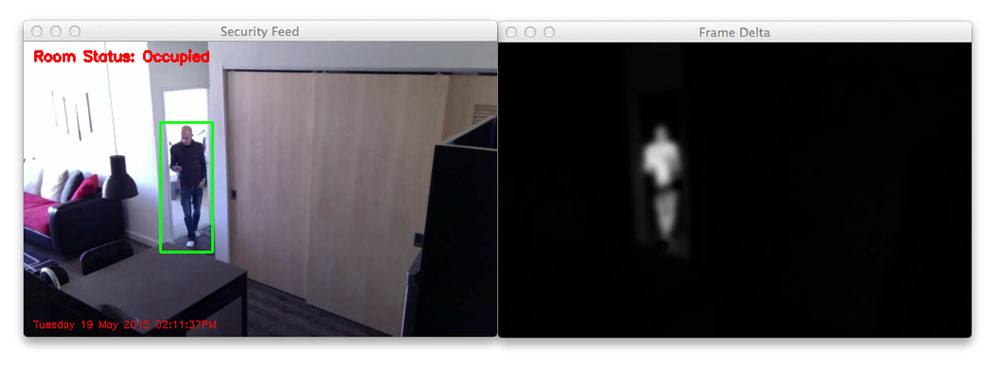
On **Line 60** we make a check to see if the avg frame has been initialized or not. If not, we initialize it as the current frame.

**Lines 69 and 70** are really important as the time of day changes (and lighting conditions change), and as new objects are introduced into our field of view, our system may falsely detect motion where there is none.

To combat this, we instead take the weighted mean of previous frames along with the current frame. This means that our script can dynamically adjust to the background, even as the time of day changes along with the lighting conditions. This is still quite basic and not a “perfect” method to model the background versus foreground, but it’s much better than the previous method.

Based on the weighted average of frames, we then subtract the weighted average from the current frame, leaving us with what we call a frame delta:

delta = |background\_model – current\_frame|



**Figure 3:** An example of the frame delta, the difference between the averaged frames and the current frame.

We can then threshold this delta to find regions of our image that contain substantial difference from the background model — these regions thus correspond to “motion” in our video stream:

**# threshold the delta image, dilate the thresholded image to fill**

**# in holes, then find contours on thresholded image**

**thresh=cv2.threshold(frameDelta,conf["delta\_thresh"],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=cnts[0]ifimutils.is\_cv2()elsecnts[1]**

**# loop over the contours**

**forcincnts:**

**# if the contour is too small, ignore it**

**ifcv2.contourArea(c)<conf["min\_area"]:**

**continue**

**# compute the bounding box for the contour, draw it on the frame,**

**# and update the text**

**(x,y,w,h)=cv2.boundingRect(c)**

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

**text="Occupied"**

**# draw the text and timestamp on the frame**

**ts=timestamp.strftime("%A %d %B %Y %I:%M:%S%p")**

**cv2.putText(frame,"Room Status: {}".format(text),(10,20),**

**cv2.FONT\_HERSHEY\_SIMPLEX,0.5,(0,0,255),2)**

**cv2.putText(frame,ts,(10,frame.shape[0]-10)**

**cv2.FONT\_HERSHEY\_SIMPLEX,0.35,(0,0,255),1)**

To find regions in the image that pass the thresholding test, we simply apply contour detection. We then loop over each of these contours individually (**Line 82**) and see if the pass the min\_area test (**Lines 84 and 85**). If the regions are sufficiently larger enough, then we can indicate that we have indeed found motion in our current frame.

**Lines 89-91** then compute the bounding box of the contour, draw the box around the motion, and update our text variable.

Finally, **Lines 94-98** take our current timestamp and status text and draw them both on our frame.

**Code to upload to Dropbox**:

**# check to see if the room is occupied**

**iftext=="Occupied":**

**# check to see if enough time has passed between uploads**

**if(timestamp-lastUploaded).seconds>=conf["min\_upload\_seconds"]:**

**# increment the motion counter**

**motionCounter+=1**

**# check to see if the number of frames with consistent motion is**

**# high enough**

**ifmotionCounter>=conf["min\_motion\_frames"]:**

**# check to see if dropbox sohuld be used**

**ifconf["use\_dropbox"]:**

**# write the image to temporary file**

**t=TempImage()**

**cv2.imwrite(t.path,frame)**

**# upload the image to Dropbox and cleanup the tempory image**

**print("[UPLOAD] {}".format(ts))**

**path="/{base\_path}/{timestamp}.jpg".format(**

**base\_path=conf["dropbox\_base\_path"],timestamp=ts)**

**client.files\_upload(open(t.path,"rb").read(),path)**

**t.cleanup()**

**# update the last uploaded timestamp and reset the motion**

**# counter**

**lastUploaded=timestamp**

**motionCounter=0**

**# otherwise, the room is not occupied**

**else:**

**motionCounter=0**

We make a check on **Line 101** to see if we have indeed found motion in our frame. If so, we make another check on **Line 103** to ensure that enough time has passed between now and the previous upload to Dropbox — if enough time has indeed passed, we’ll increment our motion counter.

If our motion counter reaches a sufficient number of consecutive frames (**Line 109**), we’ll then write our image to disk using the TempImage class, upload it via the Dropbox API, and then reset our motion counter and last uploaded timestamp.

If motion is not found in the room (**Lines 129 and 130**), we simply reset our motion counter to 0.

Now to finish handling if we want to display the security stream to our screen or not:

**# check to see if the frames should be displayed to screen**

**ifconf["show\_video"]:**

**# display the security feed**

**cv2.imshow("Security Feed",frame)**

**key=cv2.waitKey(1)&0xFF**

**# if the `q` key is pressed, break from the lop**

**ifkey==ord("q"):**

**break**

**# clear the stream in preparation for the next frame**

**rawCapture.truncate(0)**

We use this code to make a check to see if we are supposed to display the video stream to our screen (based on our JSON configuration), and if we are, we display the frame and check for a key-press used to terminate the script.

As a matter of completeness, let’s also define the TempImage class in our pyimagesearch/tempimage.py file:

**# import the necessary packages**

**importuuid**

**importos**

**classTempImage:**

**def\_\_init\_\_(self,basePath="./",ext=".jpg"):**

**# construct the file path**

**self.path="{base\_path}/{rand}{ext}".format(base\_path=basePath,**

**rand=str(uuid.uuid4()),ext=ext)**

**defcleanup(self):**

**# remove the file**

**os.remove(self.path)**

This class simply constructs a random filename on **Lines 8 and 9**, followed by providing a cleanup  method to remove the file from disk once we are finished with it.

# Raspberry Pi Home Surveillance: Execution

Simply navigate to the source code directory for this post and execute the following command:

**$python pi\_surveillance.py--conf conf.json**

**3.4 Screenshots:**

****

1. **FUTURE SCOPE**

The USP of the product is that it is based on open source, freely available code, which can be modified in any way possible. While the current functions of the product are limited to "track and report" functions only, by means of OpenCV and Dropbox, OpenCV can be implemented at a deeper level to provide features like facial recognition, and can cross reference it with official records to perform highly advanced attendance monitoring, and audience mapping functions.   
  
These can be further integrated with embedded functions of the Raspberry Pi, which can run protected web servers, which further reduces its dependence on third party applications like Dropbox, and can effectively serve as a standalone solution for surveillance, with no subscription fees, top-of-the line features, all in the same, credit-card sized form factor.

1. **CONCLUSION**

This project achieves the goal of creating a low cost home surveillance system, which is open source, extendible, and easy to implement. The product, if brought to market, will cost less than the cheapest offerings by established companies, while offering features that can be compared to the mid-high range of products in the market. Because all the code of the project is open source or developed in-house, it can easily be modified to handle advanced features, customisations, personalised user requirements, etcetera.

As the product is self-contained, it is easy to implement, and can be easily brought to everyday households, corporate buildings, and in public spaces where surveillance may be required. If implement in a major scale, the product can be instrumental in the fight against crime, and will become an integral tool for law enforcement organisations worldwide.

**5. REFERENCES**

* Andrew B. Godbehere, Akihiro Matsukawa, Ken Goldberg, *Visual Tracking of Human Visitors under Variable-Lighting Conditions for a Responsive Audio Art Installation*
* P. KaewTraKulPong and R. Bowden, *An Improved Adaptive Background Mixture Model for Real-time Tracking with Shadow Detection*
* Zoran Zivkovic, *Improved Adaptive Gaussian Mixture Model for Background Subtraction*
* Ahmed Elgammal, *Background Subtraction: Theory and Practice*
* Derek Molloy, *Exploring Raspberry Pi: Interfacing to the Real World with Embedded Linux*
* Warren Gay, *Mastering the Raspberry Pi*
* Prateek Joshi, *OpenCV with Python By Example*