INDOOR SURVEILLANCE ON ANDROID DEVICE OVER WiFi

by,

SUSHANT ARORA

Submitted in partial fulfilment of the requirements

For the degree of Master of Science

Thesis Advisor: Dr. Christos Papachristou

Department of Electrical Engineering and Computer Science

CASE WESTERN RESERVE UNIVERSITY

August 2013

CASE WESTERN RESERVE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

We hereby approve the thesis/dissertation of

Sushant Arora

candidate for the Master of Science degree *.

Committee Chair

Christos Papachristou

Committee Member

Daniel Saab

Committee Member

Francis Merat

Date of Defense

June 26th 2013

*We also certify that written approval has been obtained for any proprietary material contained therein

Table of Contents

List of Tables

Title **Page no**

List of Figures

Title **Page no**

List of Abbreviation

TCP: Transmission Control Protocol

UDP: User Datagram Protocol

T: Threshold

t_n: Thread 'n'

F_xC_y: Floor 'x', camera 'y'

UI Thread: User Interface Thread

RTSP: Real Time Streaming Protocol

RTP: Real Time Protocol

HTTP: Hypertext Transfer Protocol

HTTPS: Hypertext Transfer Protocol over Secure Socket Layer

FTP: File Transfer Protocol

SMTP: Send Mail Transfer Protocol

FTP: File Transfer Protocol

OPEN GL: Open Graphics Library

SGL: Scalable Graphics Library

GPS: Global Positioning System

RI: Reference Image

CI: Current Image

DI: Difference Image

ISP: Internet Service Provider

Indoor Surveillance on Android Device over WiFi

Abstract

By

SUSHANT ARORA

Some buildings require high level of surveillance and do not expect visitors after working hours. This thesis study deals in making a smarter application for Android device, which makes surveillance system more intelligent and handy. This application can be used in two different ways:

1- **It can be used to replace TV setup connecting all network cameras, as security officer can connect this device with any camera to stream live video.**
2- **It can detect motion in camera and trigger notification to security officer with live video from that camera, providing him with runtime activities and position of an intruder while chasing him.**

Chapter 1

Introduction

1.1- Surveillance in High security buildings

Surveillance is the monitoring of activities or behavior of a certain environment, it involves monitoring of mostly people. Cameras are the most common device used for surveillance. Consider a high security building with several floors, sometimes the number of floors can be in multiple of tens and each floor has several cameras. All cameras of building are connected to multiple screens in security room.

Security office might have big and expensive screens with only limited number of security officers to keep the track on the screen. Considering this building has lot of cameras and sometimes the number of camera might even exceed by hundred, now it might be little difficult for security officer to keep the track of all the screens connecting tens of hundreds of cameras. Bigger the building more cameras it will have and accordingly more number of expensive screens are required to view all videos.

Consider a scenario if somehow intruder has already entered inside the building, and he is been spotted by security officer. Now as soon as security officer steps out of security room, he won't have an updated location of intruder. So the thesis study deals with this problem as well.

Following are the problems that are been focused on:

1- Replacing expensive bug TV setups with Android device.
2- Updating Android device with the current position of intruder by displaying only active videos.

1.2- Motivation

The surveillance system available in market these days are becoming better, smarter and more intelligent. The camera do not only display video but it can even be scheduled for periodic recording, it has motion detection algorithm so recording can be triggered as soon as motion is detected, it can store video or even email videos or snapshot images to the user.

Problem with current setup is that it requires multiple screens inside security room for observation and if somehow intruder has already entered inside the building and he is walking, security officer has no way of spotting his current live position as soon as he steps out of the security room, he is totally unaware of current position as he is walking inside the building.

In this thesis study, I have introduced some further additional advanced features in surveillance system which makes it more intelligent and handy. It gives

flexibility to replace big expensive TV setups with a small handy smartphone or tablet which is much more compact. Not only that but a smarter algorithm has been implemented which makes surveillance more intelligent. It can detect the motion and displays only active video which has some motion in front of it and display those videos on android device with current location of an intruder.

1.3- Thesis Outline

Chapter 2 gives the information with the background work and basic overview of several technologies used in this thesis topic, it mostly gives basic introduction to network cameras and its types, introduction with Android operating system with its architecture, communication protocols used in IP cameras, motion detection algorithm

Chapter 3 provides with the detailed solutions and algorithm. The Solution is divided into three parts: Multithreading, Motion detection and video updating.

Chapter 4 explains one time configuration, software algorithm, future work, and conclusion.

Chapter 2

Background and Basics

2.1- Background Work

Several applications are available in Android market which deals with IP camera. Some applications are required to enter the IP address of camera and they can display live video on screen, or some of the applications require one time configuration with limited number of cameras and it splits the screen into part to display all the live videos. Some applications even have motion detection algorithm.

2.2 - Network Camera:

Network Camera also called as Internet protocol camera (IP Camera) is a digital video camera that is mostly used for surveillance purpose. This camera can receive or transmit data over the network. It can capture and transmit live videos or Images over the internet that can be viewed by user remotely on Internet. Each IP camera has its unique IP address and it can be connected to a

monitoring PC as long as IP camera has an access to network. This camera has some built-in functionality such as FTP Server, e-mail client, video recorder, video recorder with motion detection etc.

Following figure shows the basic connection setup:

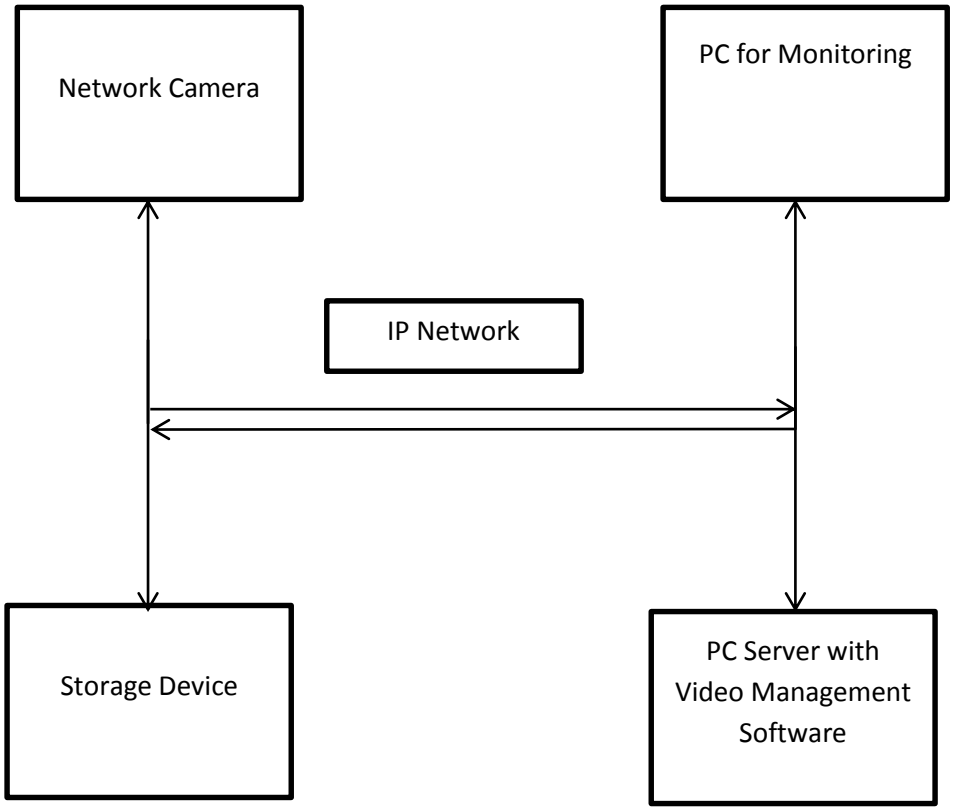


Figure 1: IP camera connection setup

Network Cameras

These are the digital IP cameras which have some additional features like Motion detection, alarm triggering, video recording on motion detection, continuous video recording etc. IP Cameras available in market these days are becoming smarter day by day and they have internal CPU, memory, Night Vision lenses, sometimes they even run on an embedded software and they can have both wired or wireless interface.

Storage Device

Most of the IP cameras available these days have an option to connect to an external storage device to store the recorded video. Camera can be connected with an external flash drive, memory card or hard disk drive. However, they are even store recorded data into network drive. IP cameras are classified in two different category based on their memory storage:

1- **Centralized IP cameras**: These types of camera require a different setup called NVR (Network Video Recorder). The NVR can be a separate device which runs on Embedded Operating System and it is connected with Camera to record video or trigger alarm.

2- **Decentralized IP cameras:** These types of camera do not require any separate NVR setup for video recording. These cameras have their own recording built-in unit which can record videos into connected internal

memory storage device such as Flash drive, hard drive, or any network storage device.

PC For Monitoring

A PC can be used for monitoring the video from IP camera. Even Internet explorer browser can also be used for viewing video or JPEG with a simple command such as:

http://xxxx.xxxx.xxxx.xxxx:yyyy/video.mpg

http://xxxx.xxxx.xxxx.xxxx:yyyy/test.jpeg

xxxx.xxxx.xxxx.xxxx: IP address of camera

yyyy: Port number

video.mpeg: To view video

test.jpeg: to view JPEG snapshot

PC with monitoring software

Monitoring software is mostly the software provided by the camera vendor to be installed in PC. All the cameras are required to be configured only once using this software. Monitoring software provides various additional features like video recording, snapshot capturing, splitting screen to view video from multiple cameras at a time, setting up FTP or SMTP server, etc.

Types of Network Camera

Network camera can also be classified whether they are designed for indoor or outdoor environment. Outdoor cameras require an external housing for their protection in the harsh environment like rain, snow, wind, humidity etc. Perhaps protection housing is not the only difference. The sunlight for outdoor camera is also another issue and has to be addressed. It is equipped with a separate Auto Iris Lens which regulates the amount of light that is exposed on image sensors inside the camera. Network camera can be further cauterized into following:

Fixed Network Camera

These are the IP cameras which provide a static view once mounted. They provide a view of only a specific area.

Figure 2: Fixed network camera

These cameras are available in Dome model as well, where in it is mounted inside a dome with extra housing protection. Dome model provides an added advantage that they can be pointed anywhere, and it is hard to view the direction from outside of where it is pointing to.

Figure 3: Fixed network camera dome type

Wireless Network Camera

This type of camera provides greater range of flexibility as the name says all. It does not require any wire for data transmission, so they provide greater freedom to be placed in the places like parking lot, college campus or any place where setting up network cables through ground is undesirable.

Figure 4: Wireless network camera

PTZ IP Camera(Pan, Tilt, Zoom)

PTZ camera provides with a great advantage of Pan, Tilt and zoom, these camera provides the greater viewing angle since camera can be turned and pointed to any direction. The PTZ operation can be manual or automatic. Manual PTZ camera can be operated with a joystick, operator can turn the camera and point it to any desired direction but within a certain range of angle. It also provides with the zooming capability that migh range from 10X to 26X, which varies with the camera.

PZT camera is also available in non-mechanical model, which uses wide angle lens and a megapixel camera to view the larger angle. Non-mechanical PTZ camera has couple of advantages over mechanical PTZ camera, since it does not have any moving part so it does not have any wear and tear, also the zooming is lot more faster in this type of camera.

Figure 5: Manual PTZ network camera

Following figure shows Non-Mechanical PTZ network camera

Figure 6: Non-mechanical PTZ network camera

2.3 - Motion Detection Basics:

The basic approach of motion detection is to subtract the current snapshot with the previous snapshot, if a difference is found, that means motion is detected. However there are few factors that have to be considered.

Idea is to take two snapshots and subtract it from the previous snapshot. Since an image consist of hundreds or thousands of pixels and each pixel represents three bytes of data for RGB image, storing value of Red, Green and Blue, one byte each. Simple technique for motion detection is to compare the value of each pixel one by one. If the result is zero, indicates no motion has been detected, however if result is non-zero means motion has been detected.

Following techniques are used for optimizing the above algorithm:

1- **Processing Speed:** Basic idea is to take each pixel from current frame and subtract it from the corresponding pixel of previous image one by one. For example we have an image with 800x800, so the total numbers of pixels are 800x800 = 640,000, so if a software function is written to subtract each pixel one by one, that function has to be executed 640,000 times. However if the image is been shrank to 16x16, total number of times function is executed will be 16x16 = 256 times, which will be lot faster than executing it 640,000 times. Another approach will be to take

the image and convert it to Gray scale, since memory size of gray scale image is much less than colored RGB Image and each pixel in gray scale require one byte of data.

2- **Moving object become Stationary:** Another problem with the above mentioned approach is that that since we are subtracting current image from the previous image, so consider a case where motion is detected, now the moving object in the background stopped and is stationary. Since we are subtracting current frame from previous frame and as there is no difference in between current frame and previous frame, so it will not detect any motion. This problem can be solved by taking a frame at the beginning, without an intruder and always subtracting first frame from current frame. So even if the intruder does not move, he will still be detected.

3- Reference frame has a mobile object: For example the first frame, also called as reference frame already has a mobile object like a car. As the car is been driven, motion will be detected, which is correct. However once the car is driven away from in front of the camera, subtracting the current snapshot frame from reference frame will still result in motion, perhaps there is no motion. This problem can be solved by replacing the first frame, i.e. reference frame with the current frame and making that as a reference frame.

4- Noise in the Image: There is possibly a good chance that one or more pixel might have been changed because of noise, which will result in

triggering of motion. This problem can be solved by keeping a threshold set point, so motion will be triggered only if numbers of pixel changed are greater than the threshold value.

2.4 –Multithreading:

Thread execution is a smallest sequence of programmed instruction that can be executed. Process is made up of thread, it can have one thread or it can even have multiple threads in it. For single core, one thread is executed by processor at a time, and multithreading generally occurs at Time Division Multiplexing, i.e. processor executes one thread for certain time period, then it switches to another thread and execute it, then to third one and so on. However threads are executed parallel for more than one core processor. Number of cores will be equal to the number of threads that can run parallel.

Multithreading improve the performance if used correctly, however it even increase the overhead on processor with every context switching, so care has to be taken while introducing multiple threads in an application.

Execution in multi-core

Multi core processor has more than one core, so the process execution will always be faster than single core processor, since each core in multicore can execute one thread at a time, which provide with an improved performance. The execution of threads happen based on time division multiplexing, however there are also some other scheduling policies like FIFO, priority based etc. Following are the steps of execution for single core processor:

1- OS sets up the timer which interrupts the system after certain interval of time.
2- Each interval i.e. time slot, is called as a time slice, and time slice is a time period given to each thread for execution.
3- An interrupt occurs after every time slice.
4- OS runs the scheduling routine and picks up the next thread to be executed as soon as it receives an interrupt.

Explained above was thread execution for single core. Perhaps the steps of execution are same for multi-core, however each core executes one thread at a time, parallel to each other.

Multithreading Memory

Each process is assigned with some memory this memory is divided into different slots mainly Stack, Heap, Data and Text. Memory sharing is very easy and fast in

multithreading. Following figure shows the memory organization for three threads.

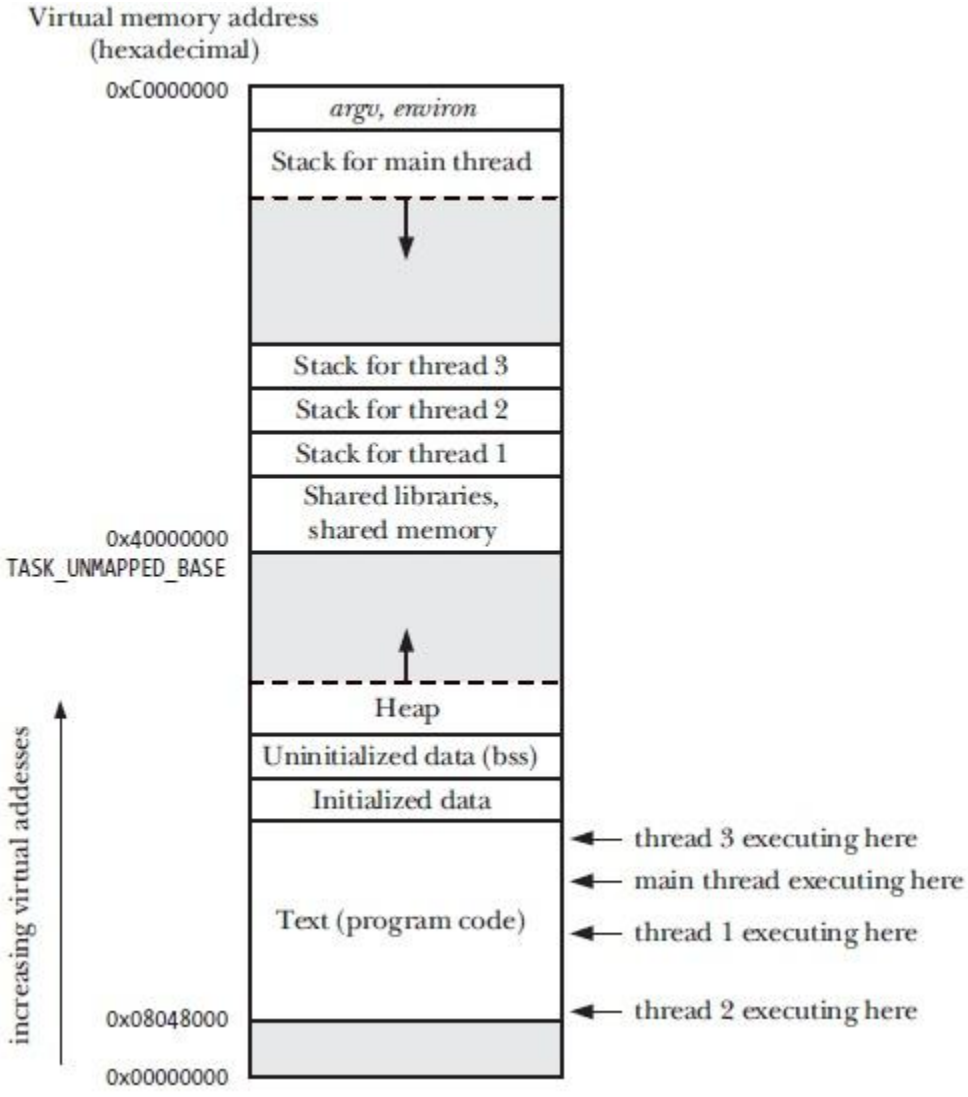


Figure 7:Multithreading memory organization

2.5 – Android Operating System:

Android, owned by Google is an open source Linux based operating system primarily designed for Smartphones and tablets. Android applications are primiraly written in Java, however it supports few other languages for development using NDK (Native Development Kit). Android has its own SDK (Software Development Kit), which is used for writing application code. The code is compiled with any data or resource file and archived all together into a package with .apk extension, which is a final package to be installed into the android device.

Following table gives the android versions so far:

Android Version	**API Level**
Android 1.0	API 1
Android 1.1	API 2
Android 1.5 Cupcake	API 3
Android 1.6 Donut	API 4
Android 2.0 Eclair	API 5
Android 2.0.1 Eclair	API 6
Android 2.1 Eclair	API 7
Android 2.2 – 2.2.3 Fyoro	API 8
Android 2.3 – 2.3.2 Gingerbread	API 9

Android 2.3.3 – 2.3.7 Gingerbread	API 10
Android 3.0 Honeycomb	API 11
Android 3.1 Honeycomb	API 12
Android 3.2 Honeycomb	API 13
Android 4.0 – 4.0.2 Ice Cream Sandwich	API 14
Android 4.0.3 – 4.0.4 Ice Cream Sandwich	API 15
Android 4.1 Jellybean	API 16
Android 4.2 Jellybean	API 17

Table 1: Android versions

Features of Android operating system:

- It's a multi-user Linux based operating system, which treats each application as a different user.
- Some of its famous networking supported features are GSM/EDGE, IDEN, CDMA, Bluetooth, WiFi, LTE, WiMAX.
- It supports several multimedia formats for video, audio and image such as H.264, MPEG-4, AAC, HE-AAC, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, BMP, GIF etc.
- Accelerometer Sensor, Camera, Proximity Sensor, GPS, Digital Compass are some of the supported sensors.
- It supports multi-tasking in an application.

- Each application runs on its own process and is assigned with a unique Linux user id.
- Each process is provided with its own Virtual Machine, so the application code can run in isolation from other process.

Android Operating System Architecture:

Following figure shows the architecture of Android Operating System:

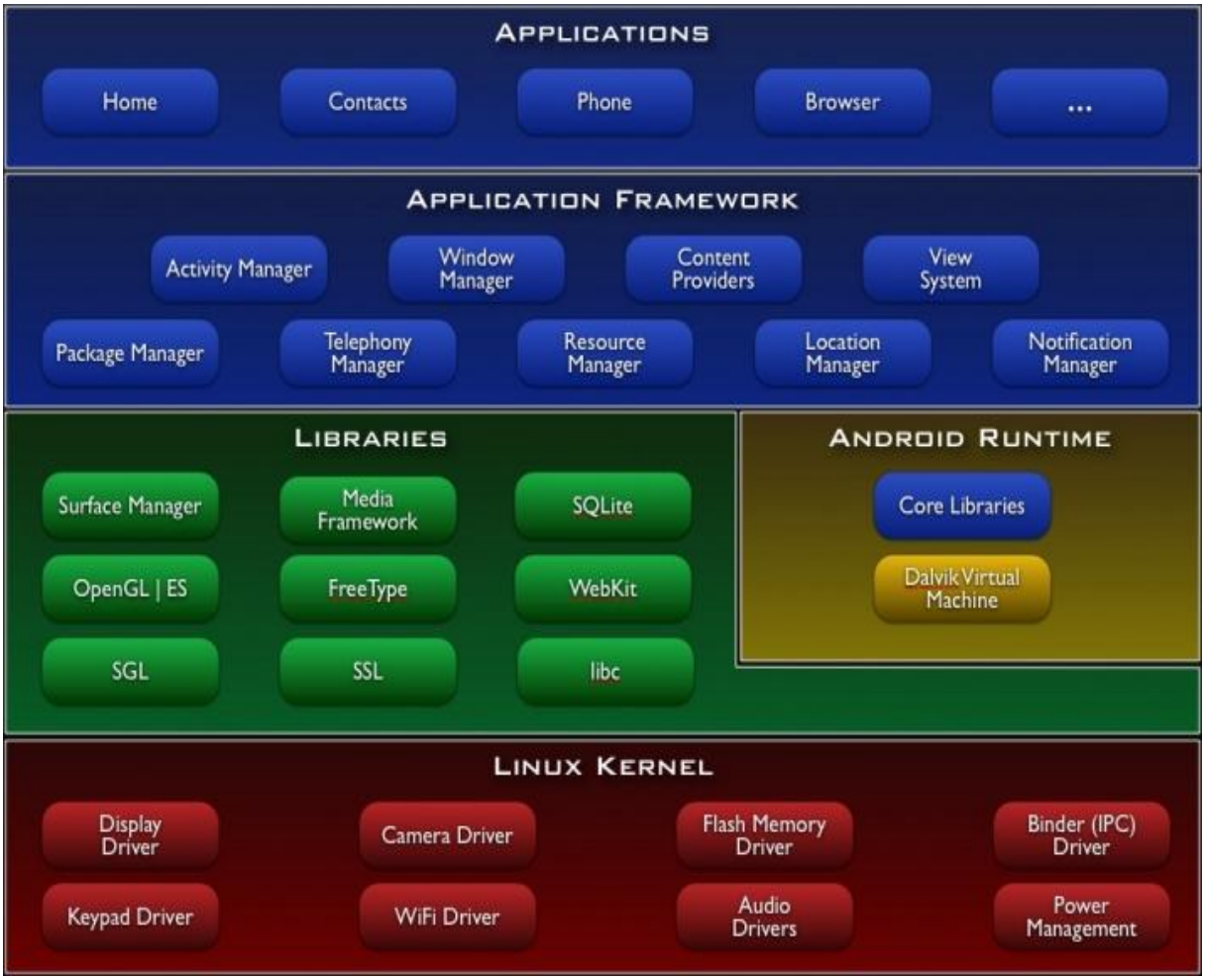


Figure 8: Android operating system architecture

Application

This is the topmost layer of Android operating system architecture and it contains mostly the applications that interact with user directly.

Application Framework

This layer sits right under the application layer and it provide interface for Application layer. This layer manages some of the basic functionalities of phone for e.g. resource management, voice management, notification management etc. Following are the functionalities of some of the basic blocks:

- **Activity Manager:** Manages the life cycle of an application.
- **Location Manager:** Manages the location, using GPS or cell phone tower.
- **Resource Manager:** It manages various resources used by an application.
- **Content Provider:** Content provider is responsible for managing the data sharing among different applications.

Libraries

This layer provides android developer with some of the libraries that enable developer to write applications in Java language. This layer provides some of the basic libraries such as:

- **SGL:** It provides 2D graphics.

- **Media Framework:** This supports playback and recording for some of the various formats of audio, video and images.
- **WebKit:** It provides the browsing engine.
- **Open GL|ES:** It provides the 3D library.

Located on the same level is Android Runtime layer which has two different functionalities:

Core Libraries: Core libraries provide Android developer with some libraries to write an application in Java.

Delvik Virtual Machine: Delvik Virtual Machine is open source software that runs an application on android device. It enables every android application to run on its own process with its own Delvik virtual machine. Following are some of the important features of Delvik virtual machine:

- It is a virtual machine designed for battery powered devices hence it has been optimized for low power consumption.
- It is even optimized for low memory consumption.
- It relies on underlying OS support for memory management, process management and threading.

Once an application is installed, it lives on its own security sandbox.

Linux Kernel:

This is the lowest layer in Android operating system architecture. It is one of the most important layer of android operating system. It does not interact directly with user, however it provides the interaction of an application with the hardware. Following are some of its important features of Linux kernel layer:

- Memory Management.
- Contains all low level device drivers for interaction with hardware such as display driver, touch screen driver, Camera driver, Audio driver, etc.
- Inter process communication.
- Power management.

Application Components:

Application components are some of the basic building blocks for an android application. There are four application components and each of them provides different point, through which system can enter into the application. Following are the four application components:

- **Activity:** Activity is a single screen that user interacts with. For example consider text message application. The screen that displays all the users who sent message is an activity. If user presses “write new message”, new screen opens up that’s another activity, if user opens any message, different screen opens up and that’s again another activity. Although all

activities work together to provide user with the cohesive experience, but they are all independent from each other.

- **Services:** Service does not provide a user interface, however it is an application component that runs in background. For example, music player. User can open any different application and music player can play music in the background. Any other component, like an activity can start service and let it run in the background.

- **Broadcast Receiver:** It is a component that responds to any broadcasting announcement. It may create a status bar to notify the user that a message has been broadcasted. An example of broadcasting component could be a low battery notification alert, or the screen has turned off, etc. An application can also initiate broadcast to let any other application know that the task has been completed and data is ready to be used.

- **Content Provider:** Content provider is a recommended method of data sharing across the different packages. Content provider behaves much like a data base that user can query it from, delete it, add its contents, edit its contents etc. For example, Android system provides a content provider that manages the contact information.

Lifecycle of an Activity:

Following figure shows the lifecycle of an activity and its various stages:

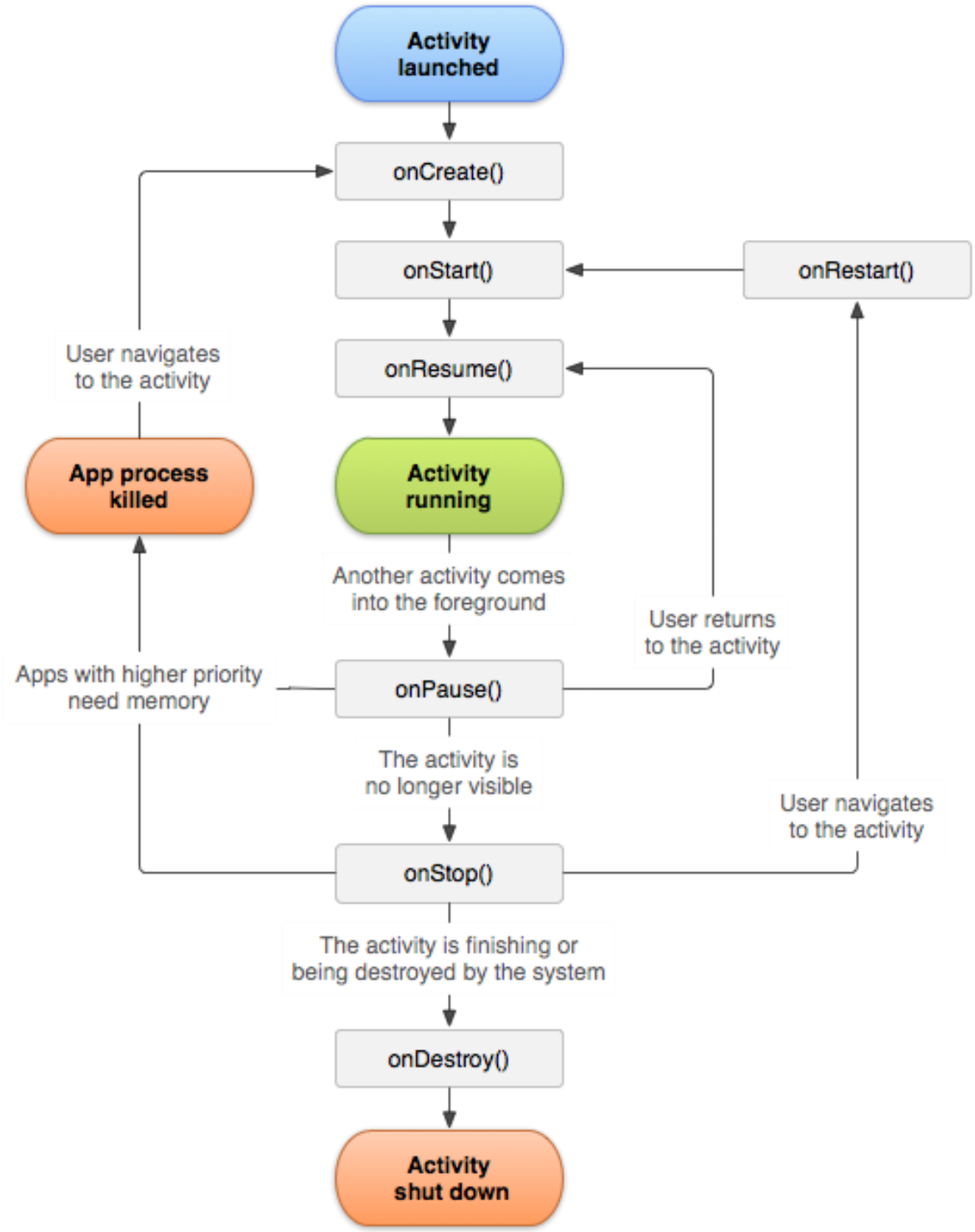


Figure 9: Activity life cycle

An Activity can exist essentially in three stages:

Resumed: Activity is in foreground at this stage and might be interacting with user

Paused: At this stage, some another activity is on the foreground, and this one is in the background. Another activity is present on top of the one that is in paused state and this one might be translucently visible. Activity at this stage is still alive, and all its data is still in memory, however it can be killed in case of low memory.

Stopped: Activity is still alive at this stage, but it is in the background. However, this activity is not attached with window manager at this time. Memory still holds the data for this activity at this time, perhaps this activity can be killed at any time in case of low memory.

Process and Threads in Android Operating System

Processes in Android:

Every application is provided with its own individual process by the kernel with a unique process id and by default all the components of same application shall run in its own single process. Android has a hierarchy tree called “importance hierarchy”, where it keeps each process based on its importance. Android can kill a process at any time in case it is running on low memory and the decision of killing a process is based on its importance in importance hierarchy.

Following figure shows the android process hierarchy:

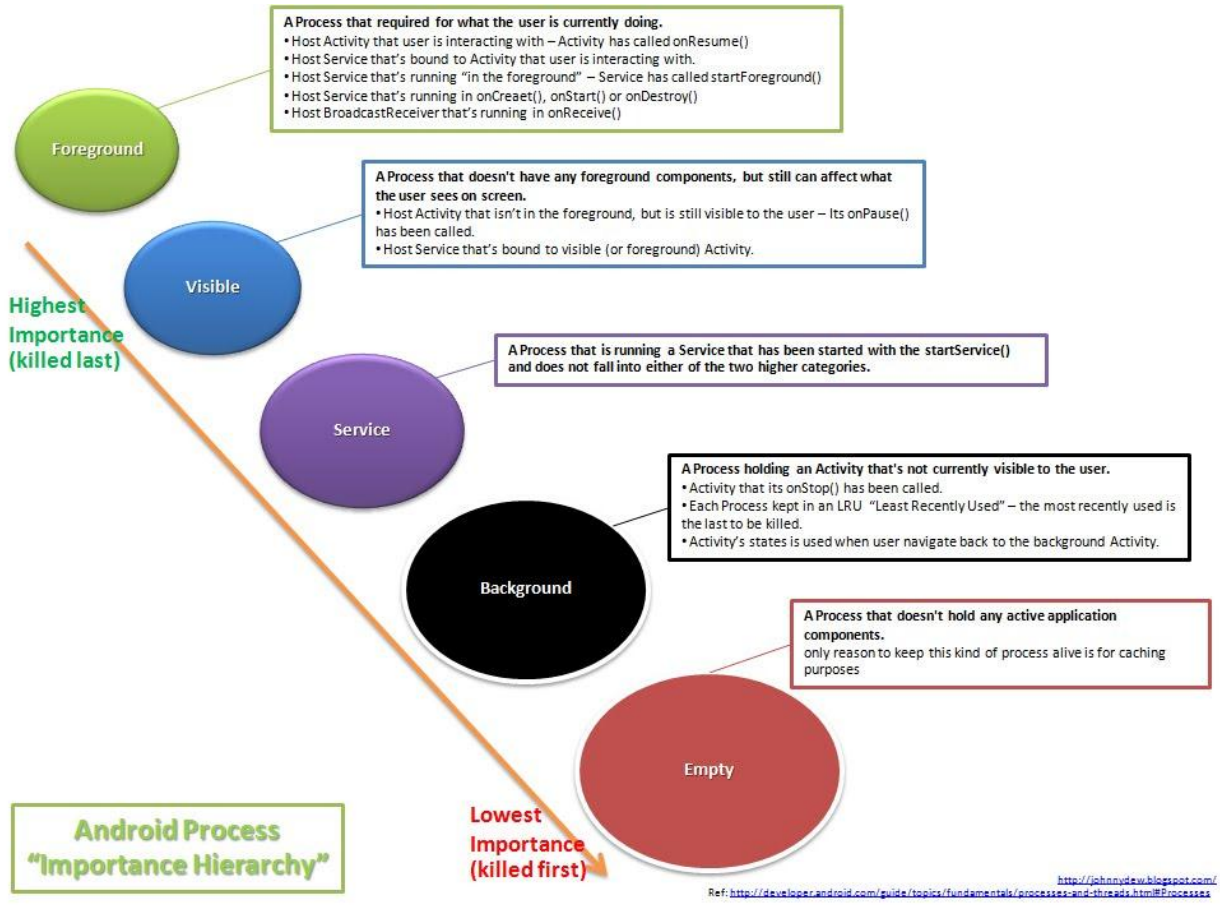


Figure 10: Process importance hierarchy

Process hierarchy has five different levels Foreground, Visible, Service, Background and Empty. Process with highest importance will be killed at the last and process with lowest importance will be killed first. For example Process in foreground state will have the highest priority and it will be the last one to be killed however process in Empty state will have the lowest priority and it will be the first one to be killed.

Any application process running in the background will always be given lower priority as compared to the process running in the foreground, however in some cases it may have a higher priority if some other processes are using it. For e.g. If process X is serving a content provider in process B, they are both treated with equal priority.

Threading in Android

Every time an application is created, it will have its own process and every process is provided with an important thread called as ‘main’ thread. This thread is also sometimes called as UI thread. It is responsible for dispatching an event to the appropriate user interface widget. When user touches any button on screen, the UI thread dispatches an event called “touch event”, this event further sets its pressed state and post the request to event queue.

If an application is required to perform any network operation or any kind of task which require long time, it is always suggested to create a new thread in the background and execute network operation in it. For example, if UI thread is engaged in any other task which require time in seconds, and if user presses a button on the screen in meantime, nothing will happen. It looks like the application is been hanged. Two important rules must be followed while writing an application:

1- UI thread should not be blocked

2- UI toolkit should not be accessed from anywhere outside the UI thread.

2.6- Transmission Protocols:

IP Camera uses several transmission protocols for video or Image transmission over the internet. Some protocol transmits live video to a PC, or sends the video/image to a web server or even sends an e-mail directly. Certain protocols are used for data transmission, however they use either TCP (Transmission Control Protocol) or UDP (User Datagram Protocol) as a carrier for their transmission.

TCP is a connection-based transmission protocol and it is used when a reliable transmission is required, it guarantees the delivery of packets from source to destination. Packets can be retransmitted if they are lost on their way to the destination however the drawback of this protocol is it is slower than UDP.

On the other hand UDP is a connectionless transmission protocol and it delivers the packets faster than TCP, but it does not guarantees if all the packets will be delivered to receiver as it does not provide retransmission on the lost packets.

Following table list some of the most common transmission protocols:

Protocol	**Transport Protocol**	**Common Usage**	**Network Video Usage**
FTP: File Transfer Protocol	TCP	It transfers file over the internet	Transfer of images or video from a network camera/video encoder to an FTP server or to an application
SMTP (Send Mail Transfer Protocol)	TCP	This protocol sends e-mail	A network camera/video encoder can send images or alarm notifications using its built-in e-mail client.
HTTP: Hyper	TCP	Used to browse web	The most

Text Transfer Protocol			common way to transfer video from a network camera/video encoder where the network video device essentially works as a web server making the video available for the requesting user or application server.
HTTPS: Hyper Text Transfer Protocol Over	TCP	Used to access web pages securely using encryption	Secure transmission of video from

Secure Socket Layer			network cameras/video encoders.
RTP: Real Time Protocol	TCP/UDP	RTP standardized packet format for delivering audio and video over the Internet— often used in streaming media systems or video conferencing	A common way of transmitting H.264/MPEG-based network video, and for synchronizing video and audio since RTP provides sequential numbering and time stamping of data packets, which enable the data packets to be reassembled

			in the correct sequence. Transmission can be either unicast or multicast.
RTSP: Real Time Streaming Protocol	TCP	Used to set up and control multimedia sessions over RTP	

Table 2: IP camera, supported transmission protocols

Chapter 3

Solution

Aim of this study focus on replacing existing expensive TV setups in security office with a small handy Android device, Or using the Android Device to catch an intruder if he has entered inside the building in non-working hours. Any camera in the building can be connected to Android device to receive live video, however it has an additional feature of detecting motion and android device will display only active videos, I.e. as long as an motion is detected in front of camera, it will continue to display video on phone.

This project involves receiving active real time videos over wireless network on Android device. An Android device such as Phone/Tablet can be used to receive live videos from ip camera over wireless network. Camera will transmit videos directly to Android device, this video is then processed to determine motion detection. Once the device detects motion in the live stream, it will be displayed on screen.

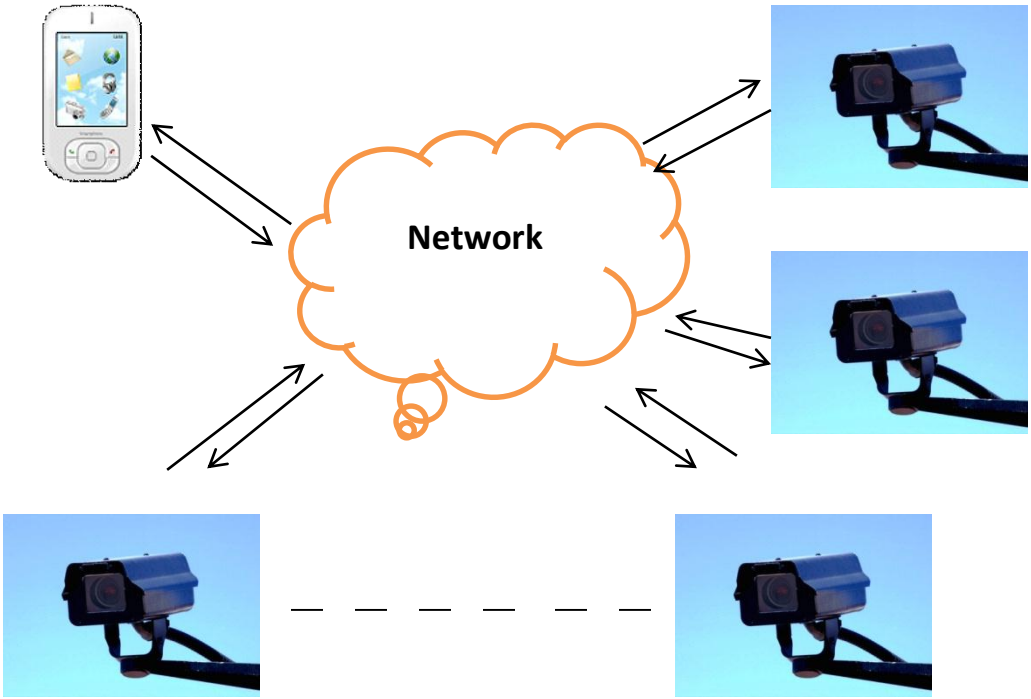


Figure 11: Network camera and android device, basic setup

The solution for this project can be divided into three parts:

1- Multi-Threading

2- Motion Detection

3- Updating Video

3.1- Multi-Threading:

Multiple threads are generated in an application. Each thread will be assigned the responsibility for one floor. For example, if building has ‘n’ floors, ‘n’ different threads are generated in application, one for each floor. However the total number of threads will be ‘n+1’, one extra thread is a UI thread (User Interface Thread). UI Thread is a default thread created by each application.

Responsibility of each thread is to receive the snapshot image from each camera one by one, process it and detect for motion.

For e.g. consider a seven floor building with five cameras in each floor. Since it has seven floors so total number of threads will be 8 (1: UI Thread + 7: Application generated, one for each floor), however number of threads will change in the run time if motion is detected.

Following explains the responsibility of each thread step by step:

1- Total number of threads when application started will be 8 (This number will change once motion is detected)

2- Thread1 opens the connection for C1F1.

3- It receives snapshot image from C1F1 (called as RI1) and close the connection.

4- Received RI1 is then processed.

5- Thread1 then goes to C2F1, opens the connection, receives its snapshot reference image (This image being called as: RI2), and process it.

6- Thread1 then goes to each camera one by one and captures RI (Reference Image) from each of them, until it reaches the last camera i.e. C5F1.

7- Once it captures RI from all five cameras, it goes to C1F1 again and detect for motion (Motion detection Algorithm is discussed later).

8- If motion not detected in C1F1, it goes to C2F1, then C3F1 and continues the process until motion is detected or application is closed by the user.

9- Perhaps at any time, if this thread detects motion, it sends the signal to UI thread, which connects the corresponding active camera to the screen and display video. At this time a new thread is created which has the responsibility to signal the UI thread to display video as long as this active camera has some motion in front of it. However, previous thread continues to detect for motion in all other cameras (Except the one in which motion has already been detected).

10-Each thread in all the floors will continue to go through same procedure asynchronously, independent of each other.

11-While UI thread is displaying video, in the meantime if software detects motion by any other thread, it can notify the user and user has an option to

switch between videos from different cameras in the building (Details are discussed later).

Naming Convention: CmFn: m^{th} camera on n^{th} floor.

For e.g. C2F1: 2^{nd} camera on 1^{st} floor.

RI: Reference Image

3.2- Motion Detection:

Details about multithreading has already been discussed in previous section, each thread will undergo following algorithm to detect for motion. Each network camera has a unique ip address and it can be connected to Android Device with a dedicated port number.

Multiple threads are created in application process, number of threads varies with number of floors in building. Each thread is responsible for one floor and will sequentially receive packets from each camera in its corresponding floor, it will then generate a scaled down image from it and subtract this image from a reference image to detect motion. However, single intruder can be seen from multiple cameras, so each camera will independently detect motion and Android device user can select the camera as per his own convenience.

Motion Detection Algorithm:

As soon as the application is started, it will grab a snapshot from each camera and process it (processing includes downscaling it to 64x64 and converting to Gray scale, this is done to increase program's computation speed and to take

less space in main memory). This image is called as a reference image and will be stored in main memory. After every few hundreds of milliseconds (approx. 300ms), corresponding thread will fetch an updated snapshot from each camera (called CI: Current Image), process it and subtract it from its respective reference image. If any difference is found in between these two images that means motion has been detected. However there is a possibility that one or more pixel value has been changed due to noise, this problem can be solved by checking if the difference Image value is above the calibrated Threshold value (T). If the difference of two images is greater than the calibrated value 'T', it concludes that motion has been detected, however if the difference is less than threshold value T, it can be ignored.

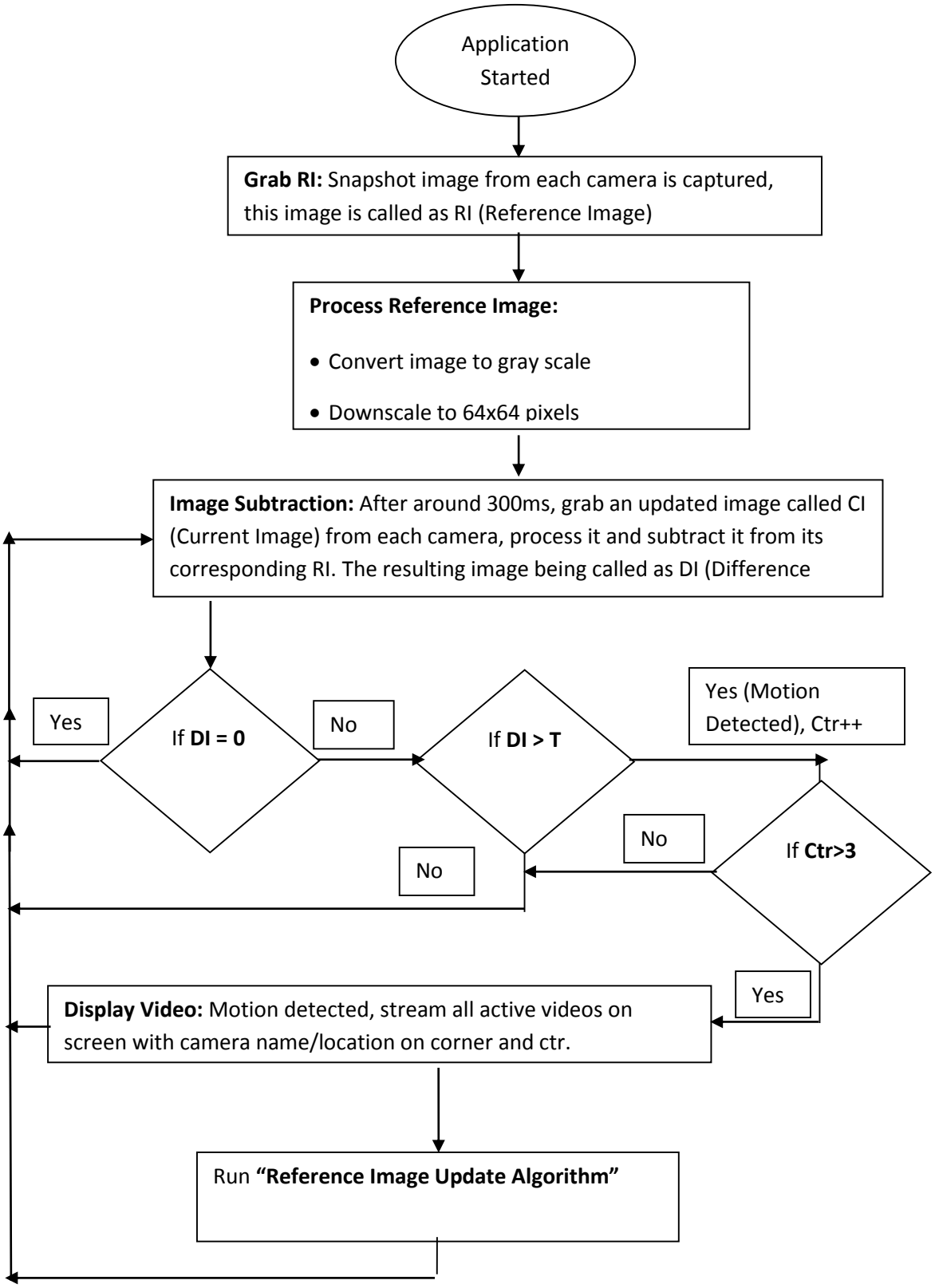


Figure 12: Motion Detection Flowchart

Explanation of Algorithm:

1- Grab RI:

As soon as application is started, it will go through following steps:

a) Create and spawn ‘n’ number of threads in run time (one thread for each floor)

b) Establish connection with all the cameras one by one.

c) Take first snapshot image from all cameras, this image is called as RI (Reference Image).

d) Save this image in main memory.

2- Process Reference Image:

Processing of reference image is done to improve Image processing speed and to reduce the main memory usage. Processing includes following steps.

a) Image is downgraded to 64x64 pixels.

b) Image is converted to gray scale.

Since gray scale image require one byte of data for each pixel, so the size of processed image is comparatively less than the color image.

3- Image Subtraction:

After around every 300 milliseconds, updated image from each camera is taken. This image is called as CI (Current Image):

a) CI is first processed (Downgraded to 64x64 pixels, converted to gray scale).

b) After processing, CI will then be subtracted from RI to get DI (Difference Image), i.e. DI = RI – CI.

c) If the result of subtraction for CI from RI is zero, this indicates RI and CI are same, hence no motion detected, however if difference is non-zero value, check is made to detect if the received data has any noise pixels.

4- Difference equal to zero:

If RI – CI = 0, i.e. DI = 0, indicates RI is same as CI hence no motion has been detected.

5- Difference greater than Threshold:

If DI = 0, indicates no motion has been detected, however if DI ≠ 0 indicates motion might be present. Perhaps, there is a possibility that value of one or more pixels might have been changed due to noise which will result in non-zero value of DI indicating motion. This problem can be solved by comparing DI with a calibrated threshold value (T). i.e.

If DI < T: Jump to previous step.

If DI > T: Motion might be present, jump to next step.

6- Counter value greater than three:

A check is made to see if counter value is greater than 3, video is displayed only if motion is detected in three consecutive snapshots.

7- Display Video:

The value of DI is greater than calibrated Threshold value (T) for three consecutive snapshots. Now the control is given to main UI thread which will connect this camera to screen and display live video. If motion is detected in more than one camera, each camera will be connected to phone to display live video one by one. Security officer can prioritize the video from each camera .Each camera has its unique ip address, and will be connected to phone on different socket.

8- Reference Image update algorithm:

This part of the program is executed every time a motion is detected. Consider following ambiguity:

a) Application is switched on.

b) RI is taken as soon as application is switched on.

c) Intruder is already standing in front of camera so he will be in reference pic.

d) Later when he moves, motion will be detected and camera will be connected to Android device.

e) Once he moves off the camera view, main memory still holds the RI with an intruder.

f) Later when CI (without intruder) is subtracted from RI, DI will still be a non-zero value and software will consider this scenario as motion detected.

This problem can be solved by taking several DI's (Difference Images) on the interval of five seconds. (DI1, DI2, DI3, DI4, DI5). Considering intruder has already gone off from in front of the camera, so all the DI's should be same.

i.e. DI1 == DI2 == DI3 == DI4 == DI5

If the above condition is true, Reference Image will be replaced with CI.

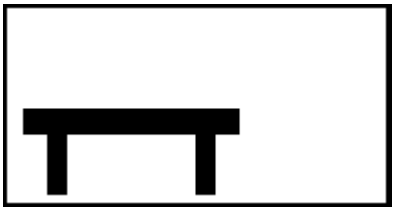
Possible Scenarios in Motion Detection

Following section describes some possible cases:

Case1: No motion detected

There are two images, RI (Reference Image) and CI (Current Image)

RI CI



Subtracting these two images will result in difference Image, i.e. DI. Since the two images are exactly same, so DI will be a blank image.

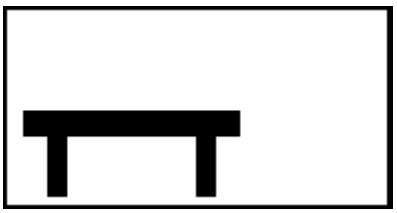
Subtracting these images will result in a blank Image

DI = RI - CI

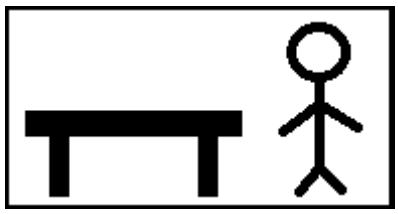
DI = 

Case 2: Intruder Detected

Assuming there is no intruder when application was first switched on,

Therefore RI = 

However intruder later appears in front of camera, so the current image

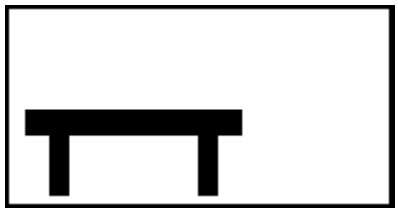
CI = 

Subtracting these two images results in the difference

DI = 

Case 3: Intruder detected but he is motionless (Not moving, idle state)

This case remains the same as previous case, as he can be detected by the software as long as there is a difference in between RI (Reference image) and CI (Current Image).

RI = 

Intruder is inactive (i.e. not moving) in current image.

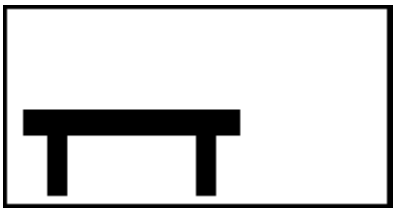
CI = 

Subtracting these two images results in the difference

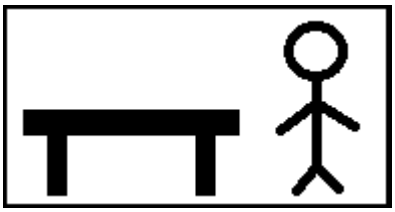
DI = 

Case 4: Intruder detected but he is moving only his body part

Following is the reference image:

RI = 

First current image with an intruder:

CI1 = 

DI1 = CI1 – RI

DI1 = 

Second Current Image, where intruder moves his body part.

CI2 = 

Difference Image

DI2 = 

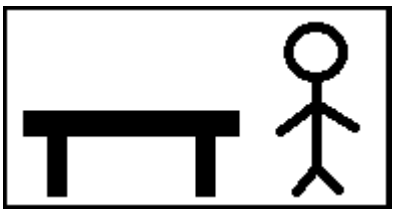
Software sees no difference in this case from Case 2, as the difference between current and reference image detects the motion.

Case 5: Intruder already in reference pic

This is a slightly complicated case since the reference image already has intruder present in it. It has a little fuzzy logic to solve this problem.

Since intruder is already present in the reference pic (as he was already present in front of camera when application was first started), he will not be detected as long as he as in a static state. As soon as he move he will be detected right away and the video from that camera is connected to Android device.

In this case, when the application was first started, all cameras took reference image and the intruder is already present in one of the reference image.

RI = 

CI = 

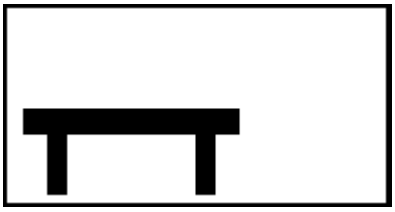
Subtracting Reference Image from Current Image will give the Difference Image:

DI = 

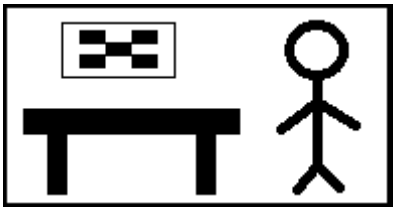
Note: Every time motion is detected, software runs “Reference pic update algorithm” to update reference pic and overcome this kind of ambiguity (Algorithm explained in section 4.2.1).

Case 6: Stationary object placed in front of camera:

Consider a scenario where a reference pic is taken and later a worker comes to hang a painting on a wall.

Initial RI1 = 

Later worker comes to hang a painting on a wall so current image is:

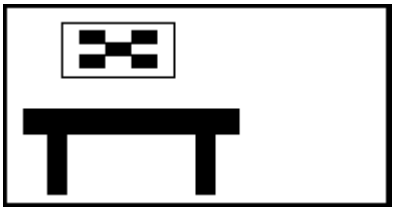
CI1 = 

As soon as the worker comes in front of camera, software detects motion and connects that video camera with Android device to display live video, now the decision will be taken by security officer holding the Android device. He can either continue to watch the live video or he can keep this camera on ignore for some time (He can select the time duration) by clicking on to it. For e.g. he keeps the camera on ignore for 5 minutes, timer will be started for 5 minutes and this camera is ignored.

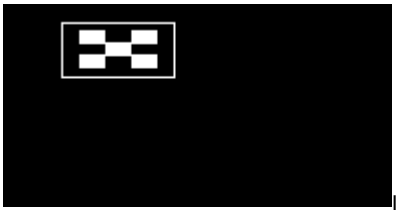
Considering a case where worker is still working at the same area even after five minutes, the camera again displays video on android device after five minutes (calibrated time period), security officer can again keep video from this camera on ignore for calibrated time period.

Now consider a case where the worker leaves the area before time runs out (5 minutes time).

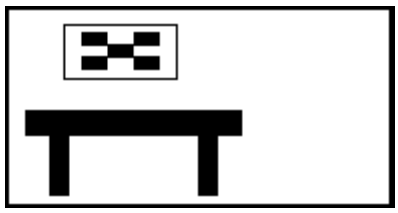
In this case CI2 will be:

The CI2 = 

Subtracting RI1 from CI2 will result in DI2

DI2 = 

Since DI2 ≠ 0 and might even be greater than Threshold value. In this case “Reference Image update algorithm” will be run to update Reference Image. Every time a reference image is subtracted from current image and its difference is above the threshold value, that indicates motion, however there is no motion in this case. This problem is solved by running “Reference Image update algorithm” Hence the new reference image after running Reference Image update algorithm will be

RI = 

Case 7: Intruder detected, but hiding behind an object

Considering intruder is hiding behind an object, and this object is in front of a camera. No motion will be detected as long as an intruder is hiding behind the

camera, however motion was already detected while he was walking towards an object and camera will be connected to Android device. Now there are two possible options:

a) Since motion was already detected and video is currently being displayed on Android device, so security officer can continue to keep this camera connected and watch live video (with intruder hiding behind an object).

b) If Security officer decides not to view the live video, then the video will be discontinued after calibrated time period of 5 seconds. However, Android device will be displaying the video from camera again as soon as intruder comes off from an object and gets in front of camera.

Case 8: Intruder runs away from the next available door (In front of camera).

Since intruder was already detected by software when he first appeared in front of camera so his activities can be seen in Android device by security officer, if he runs away from the next door and no camera is placed in this new location, software cannot detect him as there is no camera, perhaps security officer holding Android device is updated with his recent activity and position while he was in front of camera. However if he runs away to a different location where camera is placed he can be detected again by the software.

Following table describes the scenarios discussed above with brief description of solution:

Case	Scenario	Solution
1	No Motion detected	If DI = 0, so no motion detected
2	Intruder Detected	If DI > T, motion detected
3	Idle state of intruder	Although intruder is in idle state, but all the DI will still be greater than 'T'
4	Intruder moving only his body part	This case is similar to case 2, since RI – CI > T
5	Intruder already in Reference Pic	Since he is already in RI, so he will be detected as soon as he moves and later, Reference Image algorithm is ran
6	Stationary object placed in front of camera	Run "Reference Image Update Algorithm"
7	Intruder hiding behind an object	He was already detected as soon as he approached towards the object
8	Intruder runs away from next door	Intruder was already detected while he came in front of camera, he will be detected again as soon

		as he gets in front of any other camera in the building.

Table 3: Possible Scenarios for Motion Detection

3.3- Updating Video:

Main aim of this step is to display the video on screen as long as intruder stands in front of camera. As he moves away from the camera, the adjacent camera should be ready to display video as soon as intruder gets in front of other camera. Once motion is detected by a camera, thread will give a signal to UI Thread to display video. Now a new thread is created (i.e. Child Thread), the responsibility of this thread is to continuously fetch snapshots from connected camera after every 300ms and detect for motion. It will continue to signal UI thread to display video as long as motion is present in front of camera.

The parent thread, responsible for this floor will continue to receive snapshots from all other cameras in this floor (except the one in which intruder is detected, as this camera is already been taken care by child thread) so as soon as intruder gets in front of a different camera on same floor, he will be detected by first thread and this thread will signal the UI thread to display the video.

For example floor 2 has five cameras F2C1, F2C2, F2C3, F2C4 and F2C5.

Let t2 be the thread for 2nd floor. And UI: User Interface thread.

Following are the execution steps for this thread:

1- t2 detect motion in front of camera F2C2.

2- It signals UI thread to display video.

3- t2 creates a new thread called t2x (As a child thread).

4- t2x will continue to receive snapshots from camera F2C2 after the interval of 300ms and detect for motion. It will signal UI to display video as long as it detects motion in this camera.

5- In the meantime, t2 will poll all other cameras in that floor except F2C2.

6- Now if intruder gets away from camera F2C2 and walks toward adjacent camera i.e. F2C3 or F2C1, since t2 is already taking care of all other cameras and so he will be detected right away and T2x discontinues to signal UI and as soon as intruder gets in front of any other camera.

Also the active videos can be saved into an external memory card as video file.

Note: Thread receives snapshot image from each camera after every 300ms, this time will be reduced to 50 ms for the floor in which motion has been detected.

Following are the possible scenarios:

Case 1: More than one intruder present on same floor in front of same camera

Since motion has already detected as soon as they got in front of camera, so video will be displayed.

Case 2: More than one intruder present on same floor in front of different cameras

The thread will sequentially detect for motion for each camera and as soon as motion is detected by any camera, it will be connected to screen. However this thread will still continue to poll other cameras, so if motion is detected in front of any other camera, it will be signaled to UI thread and UI thread will display both videos one by one, Android device user has an option to select either of these videos.

Case 3: More than one intruder is present on different floors in front of different cameras

This case is similar to the previous case, Thread on each floor runs asynchronously and detects for motion from each camera. If thread t1 detects motion on floor one and at the same time, thread t2 detects motion on floor 2 as well, video from both these cameras will be displayed on phone and security officer can select either of the video.

Case 4: Intruder seen from multiple cameras

If one intruder is seen from multiple cameras, all the videos will be connected to the screen and security officer can select any one camera as per his choice.

Chapter 4

Configuration and Software Implementation

4.1- Configuration

Every device connected to network requires an IP address and since IP camera is a network device, so it needs an IP address too. And the IP address assigned to device can be either static (The one that is fixed and does not change) or dynamic (that changes over the time period). However static IP address requires user to pay some monthly fee to ISP (Internet Service Provider).

Problem in using dynamic IP address is that since every time the device I switched on, it communicates to the router and router assigns the available IP address to network device, which might be different than what was been assigned last time. So user is required to configure IP address every time camera is restarted.

Two possible solutions can be provided with the above mentioned problem. However the setup requires at least one time configuration.

1- Most of the routers available these days have an option for DHCP address reservation. This option enables user to provide with pre-configured network address to all network devices. So one time

configuration is required by user and he can assign a fixed address to all IP cameras and enter these addresses into an application with location of cameras.

2- Another possible option again requires one time configuration, user can enter MAC or physical address of each camera one by one along with the location number, and these addresses can be stored in a text file in SD card. Every time application is switched on, it will open this text file and retrieve current IP address of each camera one by one using Physical address. Since Android OS is based on Linux kernel, and kernel stores ARP Table information in /proc/net/arp.

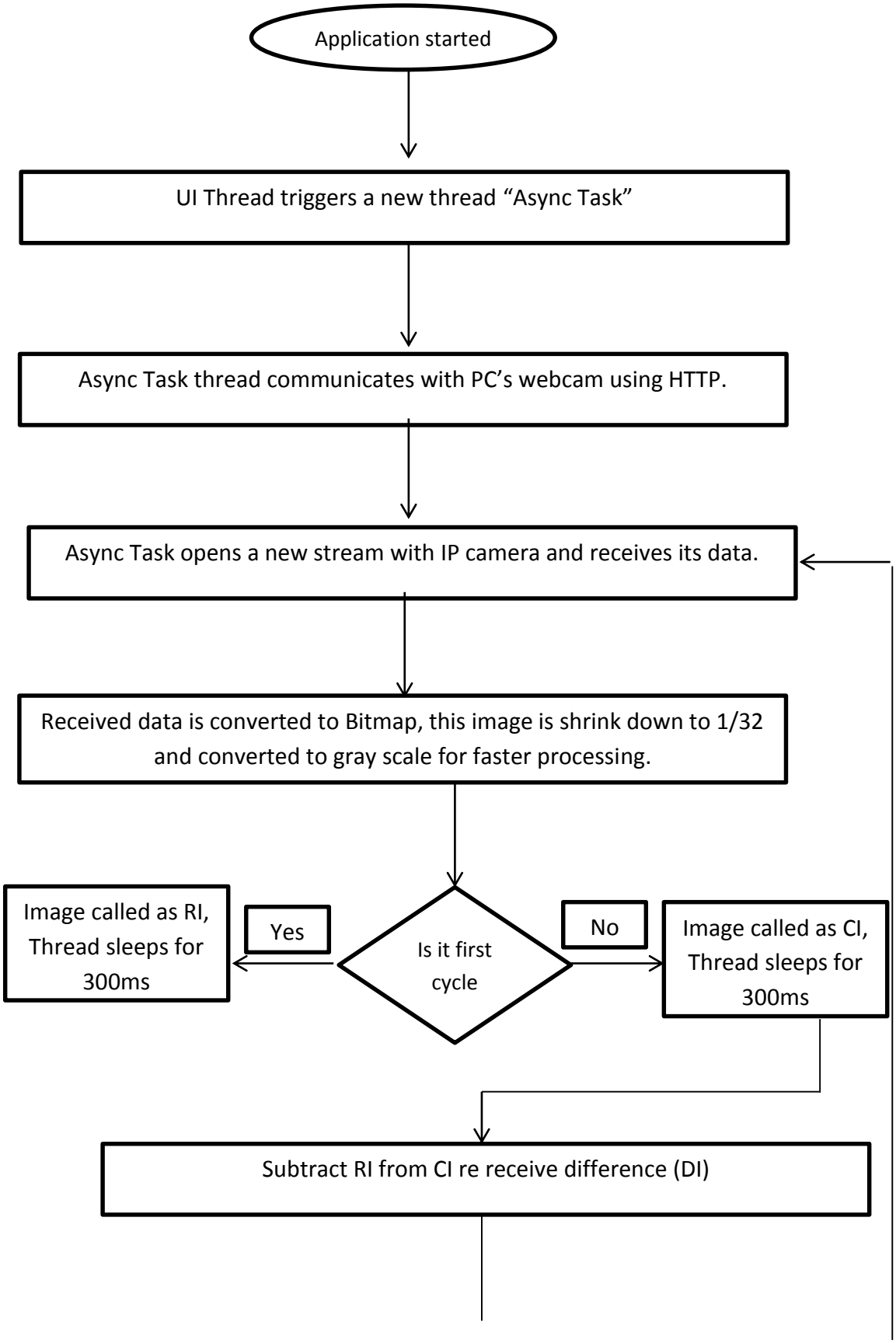
4.2- Software Implementation

Windows 8 XPS laptop's webcam is used as network camera. The laptop is installed with "Yawcam" software. This software takes webcam data and embeds it into network packets that will be transmitted. Android smartphone (Samsung Galaxy 2 Skyrocket with Android Version 4.0.4) is used on the receiving side.

Software is written in Java language using Android SDK. This software communicates with network camera over HTTP. It transmits a command to IP camera over a fixed Port number and then opens a stream to receive data from that port number. Received data is converted a Bitmap image and this image is later processed (scaled down to 1/32, then later converted to gray scale). First

snapshot received and processed is called RI, and the thread executing this operation sleeps for another 300 ms, and repeats the same process again.

It then subtract every received gray scale image (called CI) with RI, and check if the result is greater than threshold value, If not, whole process continues however if the result is greater than Threshold value (for three continuous snapshots), this thread communicates with UI thread to display video. Following figure shows the algorithm for software.


Application started
UI Thread triggers a new thread “Async Task”
Async Task thread communicates with PC’s webcam using HTTP.
Async Task opens a new stream with IP camera and receives its data.
Received data is converted to Bitmap, this image is shrink down to 1/32 and converted to gray scale for faster processing.
Is it first cycle
Yes
Image called as RI, Thread sleeps for 300ms
No
Image called as CI, Thread sleeps for 300ms
Subtract RI from CI re receive difference (DI)

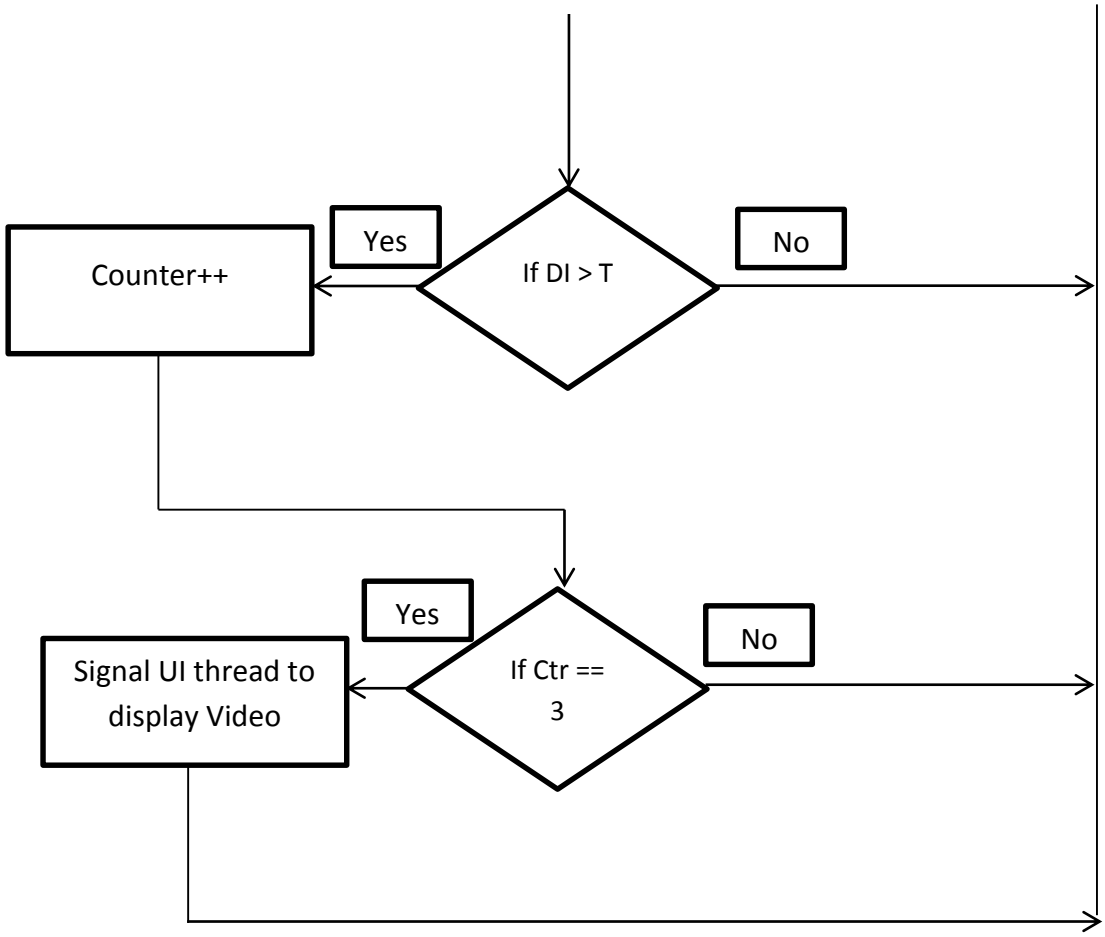


Figure 13: Software Implementation Algorithm

Chapter 5

Future Work, Conclusion and Result

5.1- Future work

The application is designed for indoor buildings, however a smarter motion detection algorithm can be introduced to make it work for outdoor environments. Since some extra factors are to be considered for outdoor environment such as animals, moving traffic, gushes of wind can cause movement in tree branches etc.

Currently this application is been designed for Android Operating system with Java language. The same algorithm can also be written in different language for different mobile phone operating system like Objective C for iOS.

5.2- Conclusion

The thesis study focus on designing a better, smarter, intelligent and a handy solution for surveillance system. Current surveillance is taken as base and some enhancement are been added on top of it to improve its intelligence and to reduce the size.

A compact surveillance system is being designed with following features:

- Big and expensive TV setups are replaced with a small handy Android smartphone or Android tablet.
- Surveillance system is made more intelligent by adding motion detection algorithm and streaming only active videos.
- Makes much easier for security officer to catch an intruder if he has already entered inside the building.

5.3- Result

1- Reference pic and incoming current images are same, so no motion has been detected. Hence video will not be displayed on screen.

Figure 14: Snapshot of incoming image (CI) in Laptop

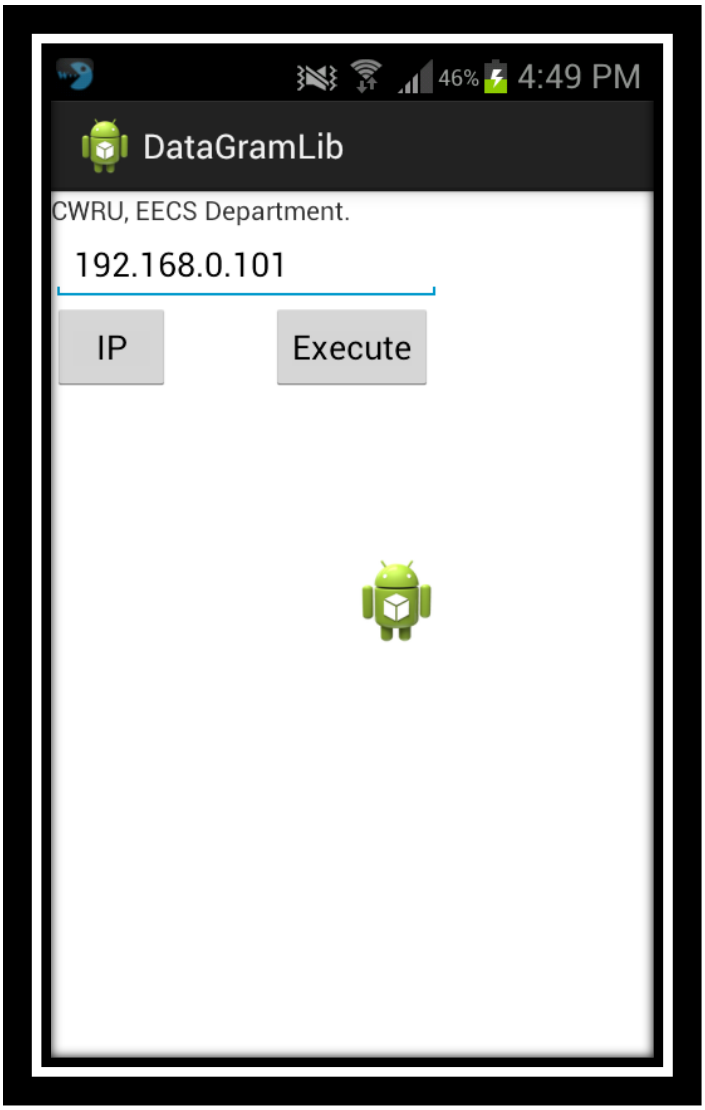


Figure 15: Snapshot of Android device

Since no motion has been detected, so video will not be displayed.

2-Reference Pic and current pic are different, however the difference is less than threshold so motion has been detected, hence video will not be displayed on screen.

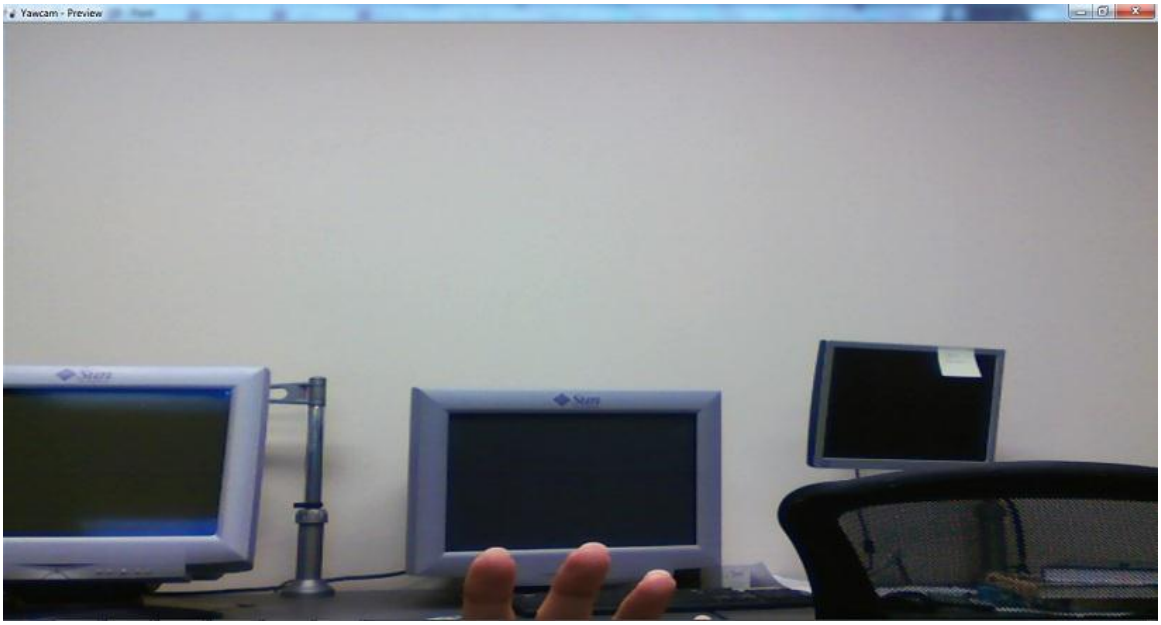


Figure 16: Snapshot of incoming image (CI) in Laptop.

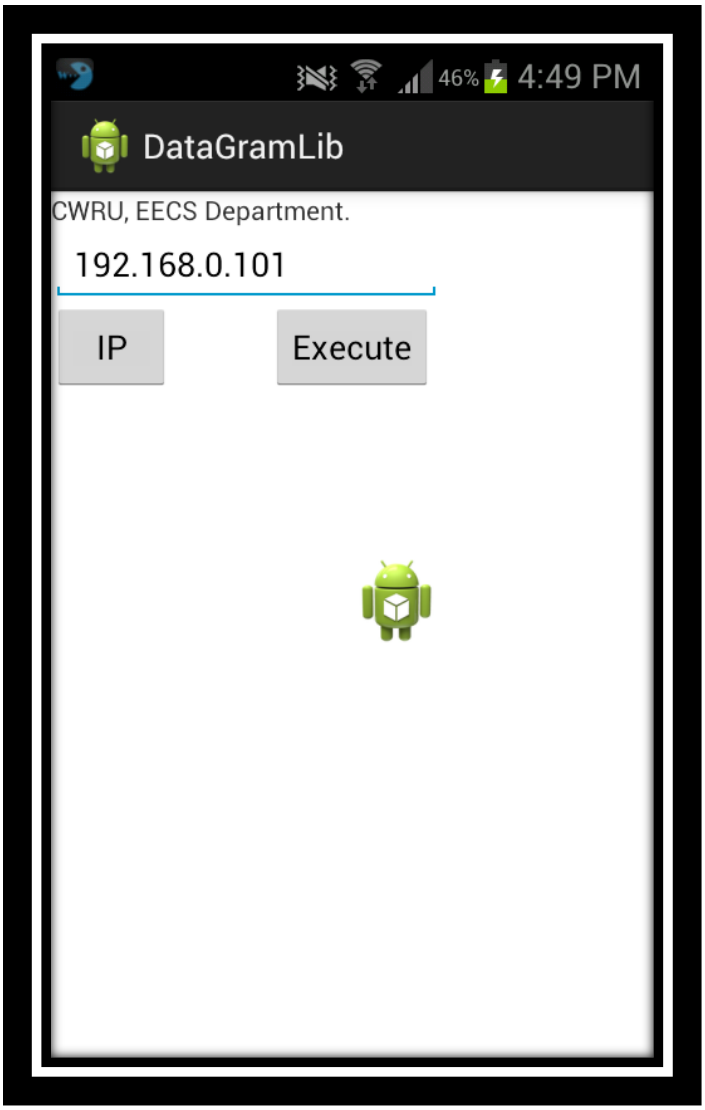


Figure 17: Snapshot of Android device

Since detected motion is below Threshold value, so video will not be displayed.

3-Reference Pic and incoming current images are different hence motion is detected and video will be displayed on screen.

Figure 18: Snapshot of incoming image (CI) in Laptop

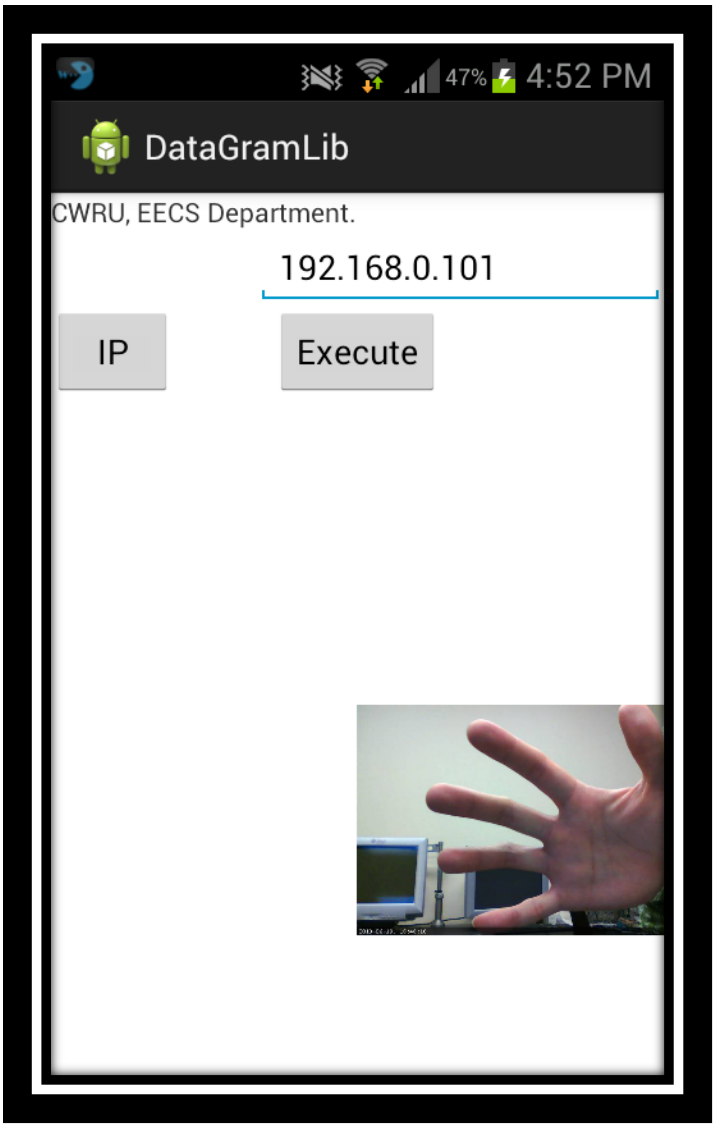


Figure 19: Snapshot of Android device

Snapshot of incoming image (CI) in Laptop

Since motion is detected, so video is displayed.

Appendix A

Code

MainActivity.Java

```
package com.example.datagramlib;

import android.os.Bundle;

import android.app.Activity;

import android.util.Log;

import android.view.Menu;

import android.view.View;

import android.widget.Button;

import android.widget.EditText;

import android.widget.ImageView;


public class MainActivity extends Activity {

	private static final String TAG = "DGLMainActivity";

	public static ImageView IV;
```

```
Button button, button2;

EditText text;

String ipUrl, url;

String preUrl = "http://";

String postUrl = ":8888/out.jpg";

ReceiveImage rcvImage;

Boolean execute = false;



@Override

protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);




        rcvImage = new ReceiveImage();

        IV = (ImageView)findViewById(R.id.imageView1);

        button = (Button)findViewById(R.id.button1);
```

```
button2 = (Button)findViewById(R.id.button2);

text = (EditText)findViewById(R.id.editText1);


button.setOnClickListener(

        new View.OnClickListener() {


            @Override

            public void onClick(View v) {

                // TODO Auto-generated method stub

                ipUrl = text.getText().toString();

                url = preUrl+ipUrl+postUrl;

                Log.d(TAG, "urlEntered: "+url);

            }

        }

        );


button2.setOnClickListener(
```

```
                new View.OnClickListener() {


                    @Override

                    public void onClick(View v) {

                        // TODO Auto-generated method stub

                        if(execute == false){

                            rcvImage.execute(url);

                            execute = true;

                            Log.d(TAG, "Inside button2,
thread executed");

                        }

                        else{}

                    }

                }

                );

            Log.d(TAG, "Before starting thread");

        }
```

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }
}
```

ProcessingImage.Java

```
//Processing of Image involves shrinking it down to 64x64 and converting it to GrayScale

package com.example.datagramlib;

import java.io.ByteArrayOutputStream;

import android.graphics.Bitmap;

import android.graphics.Bitmap.Config;

import android.graphics.BitmapFactory;

import android.graphics.Canvas;

import android.graphics.ColorMatrix;

import android.graphics.ColorMatrixColorFilter;

import android.graphics.Paint;

import android.util.Log;


public class ProcessImage {

	public static String TAG = "ProcessImage";
```

```
public Bitmap processImage(Bitmap image){
        Bitmap imageCopy = null;
        Log.d(TAG, "Inside ProcessImage");
        //imageCopy = image.copy(image.getConfig(), true);
        imageCopy = ShrinkImage(image);        //Function returns shrinked
and Grayscaled image

        //image.recycle();
        return imageCopy;
}

public static Bitmap ShrinkImage(Bitmap Image){
        Bitmap im;
        //int pixel;
        im = Image.copy(Image.getConfig(), true);
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        if(im.compress(Bitmap.CompressFormat.JPEG, 100, out))
                Log.d(TAG, "Image im Compressed");
```

```
byte[] b = out.toByteArray();

BitmapFactory.Options BFOpt=new BitmapFactory.Options();

BFOpt.inSampleSize = 32;

BFOpt.inPreferredConfig = Config.ARGB_8888;

Log.d(TAG, "Before decoding Image");

im =BitmapFactory.decodeByteArray(b, 0, b.length, BFOpt);

Log.d(TAG, "After decoding Image");

//pixel = im.getPixel(25,25);



//Log.d(TAG, "Color im 25,25 val: "+pixel);

Log.d(TAG, "Color im 25,25 val: "+im.getConfig());



//Log.d(TAG, "Processed Image size: "+im.getByteCount());

//Log.d(TAG, "Processed Image size: "+im.getHeight());

//Log.d(TAG, "Processed Image size: "+im.getWidth());



//Bitmap bmpGrayscale = Bitmap.createBitmap(width, height,
Bitmap.Config.RGB_565);
```

```
                Bitmap grIm = ConvertToGrayScale(im);
                //pixel = grIm.getPixel(25,25);
                //Log.d(TAG, "Grayscale Im 25,25 val: "+pixel);
                //Log.d(TAG, "Processed Image size: "+grIm.getByteCount());
                Log.d(TAG, "Processed Image size: "+grIm.getHeight());
                Log.d(TAG, "Processed Image size: "+grIm.getWidth());

                return grIm;
        }
        public static Bitmap ConvertToGrayScale(Bitmap bmpOriginal){
                int width, height;
           height = bmpOriginal.getHeight();
           width = bmpOriginal.getWidth();
           Bitmap bmpGrayscale = Bitmap.createBitmap(width, height,
Bitmap.Config.ARGB_8888);
           Canvas c = new Canvas(bmpGrayscale);
           Paint paint = new Paint();
           ColorMatrix cm = new ColorMatrix();
```

```
            cm.setSaturation(0);

            ColorMatrixColorFilter f = new ColorMatrixColorFilter(cm);

            paint.setColorFilter(f);

            c.drawBitmap(bmpOriginal, 0, 0, paint);

            return bmpGrayscale;

        }


}
```

ReceiveImage.Java

```
package com.example.datagramlib;

import java.io.IOException;

import java.io.InputStream;

import java.net.HttpURLConnection;

import java.net.URL;

import java.net.URLConnection;

import android.graphics.Bitmap;

import android.graphics.BitmapFactory;

import android.os.AsyncTask;

import android.util.Log;


public class ReceiveImage extends AsyncTask<String, Void, Bitmap> {
```

```
//GLOBAL VARIABLES

public static String TAG = "ReceiveImage";

ProcessImage procImage = new ProcessImage();

Bitmap processedImage = null;

Bitmap imageToDisplay = null;

DetectMotion motionDetect = new DetectMotion();




@Override

protected void onPreExecute(){

        Log.d(TAG, "In rcvImage onPreExecute");

}



@Override

protected Bitmap doInBackground(String... urlStr) {

        try {

                //Grab reference Image
```

```
                //String urlString = "http://onestep4ward.com/wp-
content/uploads/2011/12/Chilean-Lake-District.jpg";


                //String urlString = "http://192.168.0.101:8888/out.jpg";
                String urlString;
                urlString = urlStr[0];
                int[] RefImArray;
                int[] CurrImArray;
                boolean alwaysOne = true, motionDetected = false;
                int imageDifference = 0, arrayLength = 300, threshold = 100,
thresholdCtr = 0;
                //int RI_Array[][] = new int[40][30];


                RefImArray = arrayFromURL(urlString);
                if(RefImArray == null)
                        Log.d(TAG, "RefImArr is null");
                else{
                        Log.d(TAG, "RefImArr not null");
```

```
        //DisplayPixels(RefImArray,processedImage.getWidth(),
processedImage.getHeight());
                        }


                        //sleep for 300 ms, and then grab CURRENT IMAGE
                        while(alwaysOne){
                                Log.d(TAG, "Before Sleep");


                                Thread.sleep(50);
                                Log.d(TAG, "After Sleep");
                                CurrImArray = arrayFromURL(urlString);
                                //CurrImArray =
                                if(CurrImArray == null)
                                        Log.d(TAG, "CurrImArr is null");
                                else{
                                        Log.d(TAG, "CurrImArr not null");
```

```
        //DisplayPixels(RefImArray,processedImage.getWidth(),
processedImage.getHeight());
                                }
                                imageDifference =
motionDetect.findDifference(RefImArray, CurrImArray);
                                Log.d(TAG, "Pixel Difference in Images:
"+imageDifference);
                                if(imageDifference > threshold)
                                        thresholdCtr++;

                                if(thresholdCtr > 4)
                                        motionDetected = true;

                                else if (thresholdCtr < 4)
                                        motionDetected = false;

                                if(imageDifference < 100){
```

```
                    thresholdCtr = 0;
                    motionDetected = false;
                }

                if(motionDetected == true){
                    Log.d(TAG, "motion Detected:
"+imageDifference);
                    if(imageToDisplay == null)
                        Log.d(TAG, "In motion Detected,
imageToDisplay is NULL ");
                    else
                        publishProgress();
                }
            }

        } catch (IOException e) {
```

```
            e.printStackTrace();
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return processedImage;
}

int[] arrayFromURL(String url) throws IOException{
        int[] arr = null;
        Bitmap processedImage;
        //Function retruns InputStream
        InputStream in = getInputStream(url);
        if(in == null){
            Log.d(TAG, "InputStream is null");
            return arr;
        }
```

```
else{
        Log.d(TAG, "InputStream not null");


        //returns processed Image from a stream
        processedImage = getImageFromStream(in);
        if(processedImage == null){
                Log.d(TAG, "ProcessedImage is null");
                return arr;
        }
        else{
                Log.d(TAG, "ProcessedImage not null");


                //Function returns array from Image
                arr = PixelDataCopy(processedImage);
        }
}
in.close();
```

```
        return arr;
}

Bitmap getImageFromStream(InputStream in){
        Bitmap image = null;
        //Bitmap processedImage = null;
        image = BitmapFactory.decodeStream(in);
        imageToDisplay = Bitmap.createBitmap(image);
        if(image == null)
                Log.d(TAG, "returned image is null");
        else{
                Log.d(TAG, "returned image is not null, processing Image");
                //getProcessedImage(image);
                processedImage = procImage.processImage(image);

        }
        return processedImage;
```

```
        }

        /*Bitmap getProcessedImage(Bitmap image){
                Bitmap processImage;
                processImage = procImage.processImage(image);
                return processImage;
        }*/

        InputStream getInputStream(String url) throws IOException{
                InputStream in = openHttpConnection(url);
                return in;
        }

protected void onProgressUpdate(Void... v) {
        super.onProgressUpdate(v);
    Log.d(TAG, "In in onProgressUpdate");
```

```
    MainActivity.IV.setImageBitmap(imageToDisplay);
}

    protected void onPostExecute(Bitmap result)    {
    Log.d(TAG, "In onPostExecute");
    if(result == null)
            Log.d(TAG, "Image Null");
    else{
            Log.d(TAG, "Bitmap not null");
    }
    //MainActivity.IV.setImageBitmap(result);
}

    int[] PixelDataCopy(Bitmap im){
            int width = im.getWidth();
            int height = im.getHeight();
            int[] arr;
```

```
            arr = new int[width*height];
            Log.d(TAG, "Before Get Pixels");
            im.getPixels(arr,0,width,0,0,width,height);
            Log.d(TAG, "After Get Pixels");
            return arr;
        }

        void DisplayPixels(int[] arr, int width, int height){
            int i;
            int pixels;
            pixels = width*height;
            for (i = 0; i < pixels; i++)
                Log.d(TAG, "Pixel["+i+"]: " +arr[i]);
        }
/*      int[][] PixelData(Bitmap im){
            int col = im.getWidth();
            int row = im.getHeight();
```

```
        Log.d(TAG, "Inside Pixel Data");

        Log.d(TAG, "Bitmap Row: " +row);

        Log.d(TAG, "Bitmap Col: " +col);

        int[][] arr = new int[40][30];

        Log.d(TAG, "Before Image getpixel for loop");

        int i = 0, j = 0;

        for(i = 0; i < col; i++){

                for(j = 0; j < row; j++){

                        arr[i][j] = im.getPixel(i, j);

                        Log.d(TAG, "Inside for loop i:" +i+ ",j:" +j);


                }

        }

        Log.d(TAG, "After Image getPixel for loop");

        return arr;

} */

//opens http connection and returns InputStream
```

```
        public InputStream openHttpConnection(String urlString) throws
IOException{
                InputStream in = null;
                int response = -1;
                URL url = new URL(urlString);
                URLConnection conn = url.openConnection();
                if(!(conn instanceof HttpURLConnection))
                        throw new IOException("Not an HTTP Connection");
                try{
                        HttpURLConnection httpConn = (HttpURLConnection) conn;
                        httpConn.setAllowUserInteraction(false);
                        httpConn.setInstanceFollowRedirects(true);
                        httpConn.setRequestMethod("GET");
                        httpConn.connect();
                        response = httpConn.getResponseCode();
                        if(response == HttpURLConnection.HTTP_OK)
                                in = httpConn.getInputStream();
```

```
                } catch (Exception ex){
                        throw new IOException("Error Connecting");
                }

                return in;
        }
}
```

DetectMotion.Java

```
package com.example.datagramlib;

import android.util.Log;


public class DetectMotion {
        public static String TAG = "DetectMotion";
        public int findDifference(int[] RImage, int[] CImage){
                Log.d(TAG, "Inside DetectMotion");
                int diffInImage = 0, arrLength = 0, thresholdCtr = 0, i;
                arrLength = RImage.length;
```

```
int calValue1 = 65793;

int calValue2 = -65793;

int calValue3 = 131586;

int calValue4 = -131586;

int calValue5 = 197379;

int calValue6 = -197379;

int calValue7 = 263172;

int calValue8 = -263172;

int calValue9 = 328965;

int calValue10 = -328965;

int minus = -1;

Log.d(TAG, "Image Index Length"+arrLength);



for (i = 0; i < arrLength; i++){

        if(RImage[i] == CImage[i]){

                //Do Nothing, go back

        }
```

```
                else if(RImage[i] == (CImage[i] * minus)){
                        //Do Nothing, go back
                }
                else{
                        diffInImage = RImage[i] - CImage[i];

                        if(diffInImage == calValue1 || diffInImage ==
calValue2 || diffInImage == calValue3 || diffInImage == calValue4 || diffInImage
== calValue5 || diffInImage == calValue6 || diffInImage == calValue7 ||
diffInImage == calValue8 || diffInImage == calValue9 || diffInImage ==
calValue10){
                                //Do nothing
                        }
                        else{
                                thresholdCtr++;
                                Log.d(TAG, "diffInImage["+i+"]"+diffInImage);
                        }
                }
```

```
                }

                Log.d(TAG, "ThresholdCtr:"+thresholdCtr);

                return thresholdCtr;

        }


}
```

Bibliography

http://en.wikipedia.org/wiki/IP_camera

http://www.voipsupply.com/

http://www.cctvsystems.com/

http://www.ipcameraland.com/

http://www.axis.com/products/video/about_networkvideo/internet.htm

http://www.codeproject.com/Articles/10248/Motion-Detection-Algorithms

http://resource.dlink.com/connect/mastering-static-ip-addresses/ (MAC from IP address)

http://stackoverflow.com/questions/10827520/how-to-get-the-ip-address-of-a-system-using-android-phone

http://compnetworking.about.com/od/networkprotocolsip/l/aa062202a.htm

http://stackoverflow.com

http://scalibq.wordpress.com/2012/06/01/multi-core-and-multi-threading/

http://elinux.org/Android_Architecture

http://developer.android.com

http://www.edureka.in/

http://www.slideshare.net/VladimirKulyukin/mobicom-on-android-process-managementdatasharingssf01

http://johnnydew.blogspot.com/2011/12/infographic-android-process-importance.html

http://www.flattermann.net/2011/02/android-howto-find-the-hardware-mac-address-of-a-remote-host/ (Get mac from an IP.)

http://crackerpie.com/2013/04/01/android-4-1-2-jelly-bean-update-for-samsung-galaxy-s2-rolled-out/

http://www.securitysales.com/channel/vertical-markets/articles/2012/04/mall-adores-style-of-hybrid-surveillance/page/2.aspx

http://www.ecvv.com/product/2987793.html

http://learn.hackerearth.com/tutorial/operating-system/34/program-vs-process-vs-thread/

http://news.mydrivers.com/1/193/193302.htm

http://www.amazon.com/GoPro-CHDHX-301-HERO3-Black-Edition/dp/B009TCD8V8

http://thealarmspecialists.com/residential-alarm-systemshow-it-works/

http://blog.gsmarena.com/gopro-app-goes-live-for-android/

Beginning Android Application Development by "Wei-Meng Lee"

Intelligent Network Video: Understanding Modern Video Surveillance Systems by "Fredrik Nilsson"

The Linux Programming Interface: A Linux and UNIX System Programming Handbook by "Michael Kerrisk"