**An\_Najah National University  
Computer Engineering Department**

**AMYNTAS**  
Keep an Eye Wherever You Are

**Senior Project 1**

**Siba Fadda Munia Touqan**

**Supervised By:**

**Dr.Luai Malhis**



Table of Contents:

**Abstract** ……...……...……...……...……...……...……...……...4

**Introduction**  ……...……...……...……...……...……...……...……...5

**Main System**

*Image Analysis* ……...……...……...……...……...……...……...……...6

Face Detection ……...……...……...……...……...……...……...……...6

Face Recognition ……...……...……...……...……...……...……...……...9

*Video Stream* ……...……...……...……...……...……...……...……..13

Camera ……...……...……...……...……...……...……...……..13

Server ……...……...……...……...……...……...……...……..14

Pc-Client …………......……...……...……...……...……...……..16

*Android Application* ……...……...……...……...……...……...……...……..19

*System Communication* ……...……...……...……...……...……...……...……..22

Local Network ……...……...……...……...……...……...……...……..22

External ……...……...……...……...……...……...……...……..22

*Work Flow* ……...……...……...……...……...……...……...……..23

**Some Problems We Faced and How We Overcame Them** ..…………………………..…25

GUI Freeze Problem ……...……...……...……...……...……...……...……..25

Camera Access ……...……...……...……...……...……...……...……...27

Android Threads .…...……...……...……...……...……...……...………28

**Possible Improvements to the Project** ……...……...……...……...……...……...…….29

**Conclusion** ……...……...……...……...……...……...……...……..30

**Appendix** ……...……...……...……...……...……...……...……..31

Table of Figures:

**Fig. 1** Face detection example ……...……...……...……...……...……...……...……...8

**Fig. 2** Example1 of face recognition ……...……...……...……...……...……...……...……..11

**Fig. 3** Example2 of face recognition ……...……...……...……...……...……...……...……..11

**Fig. 4** Example3 of face recognition ……...……...……...……...……...……...……...……..12

**Fig. 5** Server operation ……...……...……...……...……...……...……...……..15

**Fig. 6** Pc Client ……...……...……...……...……...……...……...……..16

**Fig. 7** Pc-client interface ……...……...……...……...……...……...……...……..17

**Fig. 8**  Image notification in Pc-client .... ...…….....…...…….......……...……...……...……..17

**Fig. 9 Video streaming in** Pc-client. .. …..…….....…...…….......……...……...……...……..18

**Fig. 10 Android application** ……...……...……...……...……...……...……...…….19

**Fig. 11a** Our current android interface……...……...……...……...……...……...…….......…20

**Fig. 11b** Our end goal interface .……...……...……...……...……...……...…….......…20

**Fig. 12** Image notification in android ……...……...……...……...……...……...……...……..21

**Fig. 13** Work flow ……...……...……...……...……...……...……...……..23

Abstract

It is normal that every individual can only get peace of mind if his house, workplace or important locations are secured. Our project is a simple addition to any environment that provides a feeling of security by being able to get information about any event occurring at any given time.

To add such item of security we used face detection, face recognition, video streaming and android technology to discover any intrusion inside the house, workplace…etc and to get live video stream about any activity inside them if needed.

Our project AMYNTAS is able to perform all previous tasks and can be developed to achieve more security items if needed.

Introduction:

This documentation will describe in details, all aspects of our Senior Project I: Software.

The method we will use to present this data will be through tackling each major part of this project, and explaining its implementation and logic behind it separately, then, at the end, we will demonstrate how all these parts come together in a highly structured way to give us the final outcome, security.

The name of the project “Amyntas” is a Greek name derived from the word “Amyntro”, meaning “defender”.

Amyntas is a security system that can be installed in any environment with very few pre-requirements. For the sake of this documentation, from now on, we will assume that this system has been installed in a house environment. The cameras installed at different rooms capture video feeds and sends 10-second long packets back to the server. The cameras are also equipped with a face detection code implemented in C++ for portability that enables the camera to perform instantaneous processing for the live feed the camera is recording, and when a face is detected, it sends out the captured image back to the server to conduct further analysis. The server performs face recognition algorithm on the face snapshot, and determines whether the face detected is a “trusted” person or not. In the case of a possible-intruder detection, a notification is sent to the OWNER on his phone (at the moment, only an android application is available). When the OWNER receives the notification, he has the option to turn on video streaming and is able to watch live feedback from the room of their choice. Please note that the OWNER is capable of turning on video streaming functionality at any time, provided the system is running at their environment.

We have also developed a client application that runs on PC for a better viewing experience.The reason we thought of this as a senior project, is because we were thinking of an idea that would make people feel a little bit safer. Just because they left their home, factory, office or any other environment that is vital for them, it doesn’t mean they can no longer know what’s going on there, and they can still keep an eye on it, make sure everything is ok, everyone is safe, and their environment is stable.

System Architecture:

* Image Analysis

We implemented this part using opencv. Opencv is a free, open-source computer vision library for C/C++. It has so many capabilities that simplifies working with image analysis in general for all body parts, and for our particular case, we used it to assist us with face detection and recognition.

* Face Detection

Opencv uses Viola-Jones method for face detection. This method depends on four concepts:

* Haar features.
* Integral image.
* AdaBoost machine-learning method.
* A cascade classifier.

Haar features:

Before we explain this term, we will start with haar wavelets which are single wavelength square waves (one high interval and one low interval). In two dimensions, a square wave is a pair of adjacent rectangles one light and one dark. In visual object detection, the rectangle combinations are not true haar wavlets. Instead they contain rectangle combinations better suited to visual recognition tasks. Because of that difference, these features are called Haar features rather than Haar wavelets.

To determine if the haar feature is present, we subtract average dark-region pixel value from the average light-region pixel value. If the difference is above a threshold which is set during learning, the feature is present.

Integral image:

It is the technique that is used to determine the presence or absence of hundreds of haar features at every image location and at several scales. This is done by adding the integral pixels values. The integral value for each pixel is the sum of all the pixels above it and to its left. Starting at the top left and traversing to the right and down, the entire image can be integrated with a few integer operations per pixel. After integration, the value at each pixel location, (x,y), contains the sum of all pixel values within a rectangular region that has one corner at the top left of the image and the other at location (x,y). To find the average pixel value in this rectangle, you'd only need to divide the value at (x,y) by the rectangle's area.

AdaBoost machine-learning:

To select the specific Haar features to use, and to set threshold levels, Viola and Jones use a machine-learning method called AdaBoost. AdaBoost combines many "weak" classifiers to create one "strong" classifier.

Cascade classifier:

Viola and Jones combined a series of AdaBoost classifiers as a filter chain, that's especially efficient for classifying image regions. Each filter is a separate AdaBoost classifier with a fairly small number of weak classifiers. The acceptance threshold at each level is set low enough to pass all face examples in the training set. The filters at each level are trained to classify training images that passed all previous stages. (The training set is a large database of faces, maybe a thousand or so.) During use, if any one of these filters fails to pass an image

region, that region is immediately classified as "Not Face." When a filter passesan image region, it goes to the next filter in the chain. Image regions that pass through all filters in the chain are classified as "Face."

Fig. 1 shows the output of our face detection implementation.

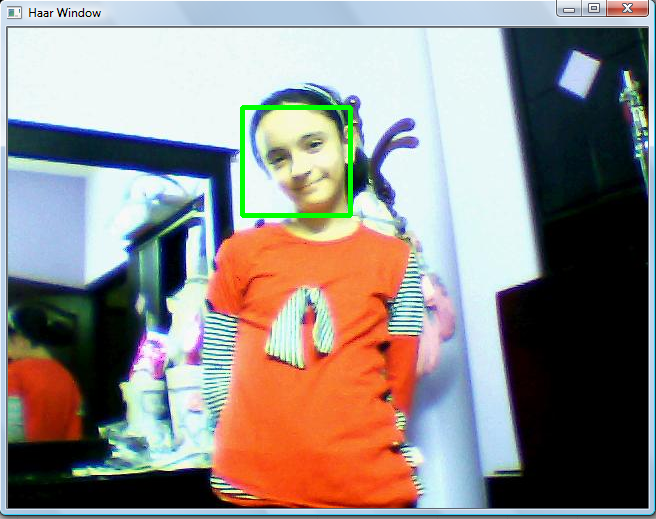


Fig. 1 Face detection example

* Face Recognition:

We used face recognition in order to decide if the person who enters any room belongs to our pre specified trusted database. In our implementation every person in our database has a separate folder in which we save images that is used for recognition. We also used two input files; train.txt, test.txt. Both files use the same format; person number, whitespace, path to image file. Train.txt includes information about our database (the number of each person and the path for his images). Test.txt contains the path of the image we want to recognize.

As we mentioned before we used opencv in our implementation. Opencv uses eigenface algorithm.

We can divide the implementation into two parts: learn phase and recognition phase.

* Learning phase contains mainly three steps:

1. Load the training data

loads face images and person ID numbers (The face images assumed here to be all the same size)

|  |
| --- |
| 1. Do PCA on it to find a subspace   Very often, the information of interest has a much lower dimensionality than the number of measurements. In the case of an image, each pixel's value is a measurement. Most likely, we can (somehow) represent the information that would allow us to distinguish between faces from different individuals with a much smaller number of parameters than 2,500 (assume 2500 pixels for each image). Maybe that number's 100; maybe it's 12. We don't claim to know in advance what it is, only that it's probably much smaller than the number of pixels. This method is called dimensionality reduction.  There are many methods for dimensionality reduction. The one that eigenface uses is called Principal Components Analysis, PCA for short, which can be defined as the maximum separation of a dataset. Since the dimensionality for images is high ,we can have more principal components in a dataset made up of images. In eigenface, each 50x50 face image is treated as one data point (in a 2,500 dimensional "space"). So the number of principal components we can find will never be more than the number of face images minus one.  3. Project the training faces onto the PCA subspace  After we have found a subspace using PCA, we can convert the training images to points in this subspace; this step is called "projecting" the training image.   * Recognition phase:  1. Project the test image onto the PCA subspace. 2. Compute a "distance" between the new image and each of the data set faces   Distance, in the original eigenface, is measured as the point-to-point distance. This is also called Euclidean distance. In two dimensions (2D), the Euclidean distance between points P1 and P2 is   d12 = sqrt(dx2 + dy2),  where dx = x2-x1, and dy = y2-y1.   1. Select the image that's closest to the new one as the most likely known person 2. If the distance to that face image is above a threshold, "recognize" the image as that person, otherwise, classify the face as an "unknown" person.   Fig. 1, Fig. 2 and Fig. 3 show three different examples for the output of our face recognition  implementation.s1,s2,s3 and s4 represent our data set, each folder associated with one person.  S5 contains the test image that we want to recognize (as we explained before all these information is  specified in test.txt and train.txt files)        Fig. 2 Example 1 of face recognition      Fig. 3 Example 2 of face recognition    Fig. 4 Example 3 of face recognition |

* Video Stream

The video stream part is divided into three major subsections as follows. Together, they handle video streaming all over the system. It starts with the “Camera” that saves packets of video then send them to the server immediately, which eliminates the need for a large

storage space. When the server receives the videos, it places them in appropriate

locations depending on which room it is from. Lastly, the client (here we will only

discuss the PC-Client. The Android user will be detailed in a following section) will start streaming when they choose to, and packets stored in the server will then be sent depending on the newest and displayed at the clients end.

* Camera

What the camera does is that it captures video packets, and immediately after, sends them to the server. Java is used for capturing video packets, particularly the media package. C language is used to perform camera-server communication and send the video packets. System communication will be explained in details in an upcoming section.

Step by step guide:

* First, we had to download the media library to be used. To do so, we had to download JMF (Java Media Framework). In addition to providing the media jar files needed, it contains useful UI systems that proved beneficial when we wanted to find out what is the “location” of our camera on our capturing devices. It also helped us perform initial testing for playing videos, compatibility, recording, ,,,etc
* Now that our Java environment is ready and media library-aware, we’ll describe how the code works.
* Since we are dealing with video, first we need to locate our capturing device, which is in this case either a webcam or an external cam connected

to our laptops. After that we set the format we want for the raw data we get from the camera. We then save packets of video as needed, in our case, we chose to have a 10 second long paclet. These packets are later sent to the server.

* What the server does and how it deals with these packets are described below.
* Server

Our server uses C++ to communicate with the rooms, and in its implementation of face recognition. And uses Java to perform video conversion.

The server continuously receives new packets of videos from all the rooms our security system covers. It then converts them into an appropriate format that the android device could play (.3gp). The server only keeps the 4 most recent packets and deletes the ones before. We did that for the sake of memory storage, it is however, possible to keep the packets for later viewing and for archiving purposes, but in that case, we would advise having external storage so not to overload the server. The server supplies the requested video packets to the client (have it be an android client or a PC-Client). In addition to continuously receiving video packets from all the rooms, it also receives any detected picture of a person

and then performs image recognition as explained earlier, and in case of an intruder, the user is notified.

Fig. 5 shows a flow chart that explains server operation.



Fig. 5 Server operation

* PC-Client

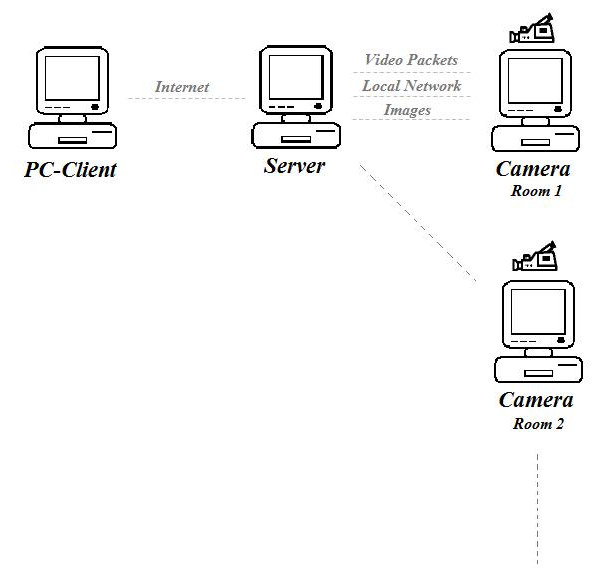
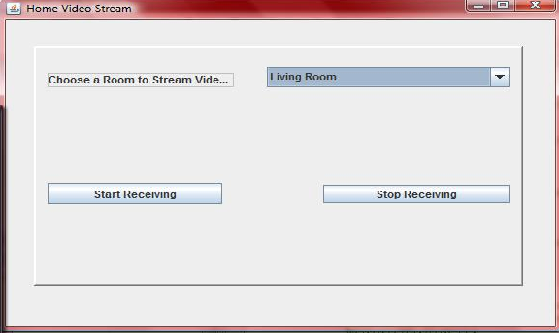


Fig. 6 Pc client

In addition to the android application (which is explained below), we have a Java-based user application. When started, it gives the user a notification every time an unauthorized person enters any of the rooms in their distant environment (home for instance). The user has the choice to start video stream from any of the rooms at any time s/he sees fit. The application then starts streaming from the most recent video packet present at the server. S/He can freely switch between different rooms. The reason we had a PC application in addition to our android application, is because laptops have higher processing speeds, better cache, and superior performance in general, so we thought it will give the user a more enhanced experience if s/he is in an environment where a laptop is available, work would be a good example of such an environment.

 Fig. 7 Pc-client interface

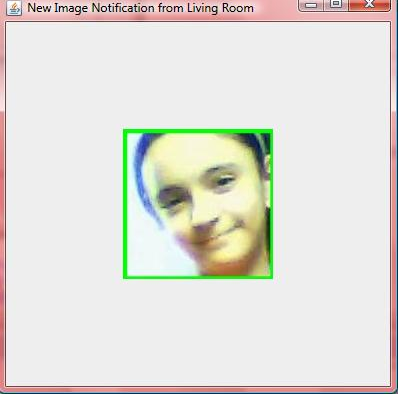


Fig. 8 Image notification in Pc-client.

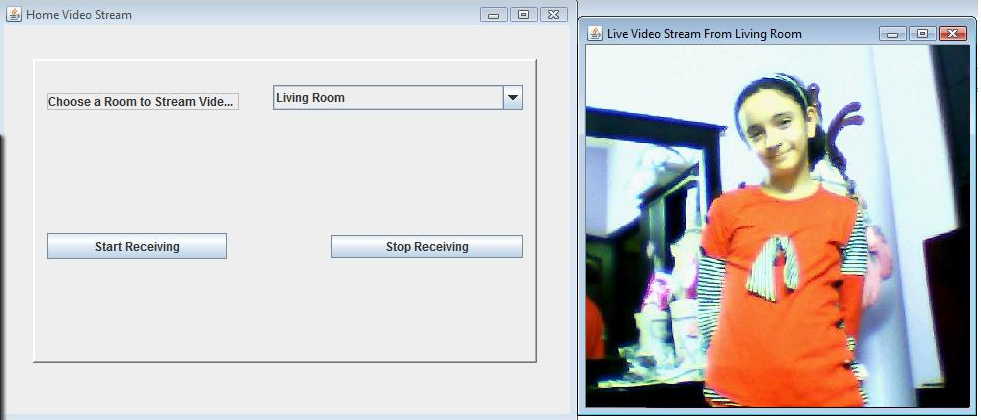


Fig. 9 Video streaming in pc-client

* Android Application

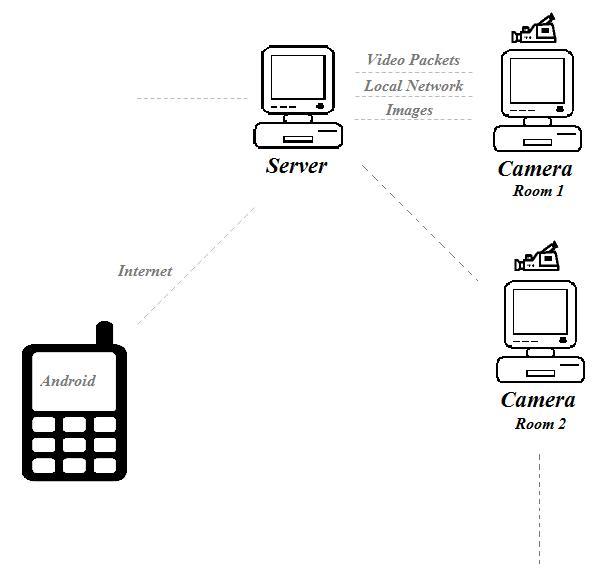


Fig. 10 Android application

Since this is a security system, and its efficiency depends on notifying the user, portability and accessibility was a major key concern of ours. And there is no arguing that the most portable, accessible, and available device in this time and age is the mobile phone. We chose to implement this application to be compatible with Android devices due to its rising popularity and widespread reach. IPhone was not an option at this stage because it requires an apple-approved developer’s account in addition to other implementation complexities.

* Video Streaming

This enables you to see live stream from any room from your house at any time.

This app enables you to choose the room from which you want to start streaming. The server contains a separate folder for each room where it keeps room’s video packets (the maximum number of packets at any time is 4). The server also keeps the index of the latest video packet for each room. After you have chosen the room, the application will start to communicate with the server. First it will find

the index of the latest video on the server to start streaming from. Then two threads will start running, the first thread will constantly download videos packets from the server, and the second thread will play these packets.



Fig. 11a Our current android interface Fig. 11b Our end goal interface

* Image Notification Part

When any person enters a room in your house, if the person isn’t included in your pre specified trusted database. Then the application will notify you and it will enable you to see the person’s image and give you the option to start streaming from that room.  
When you start this part, it will start executing in the background. The application will continuously contact the server to search for any new unrecognized person in any room. If it finds a new image of an unrecognized person, a notification containing which room has been entered will be sent to the user and it will tell him to go to the application for more details. When the user open the application he will have two choices, to display the image or to start streaming. When the

user choose to display the image, a new intent will start. The intent contains image view in which we display the person face image.

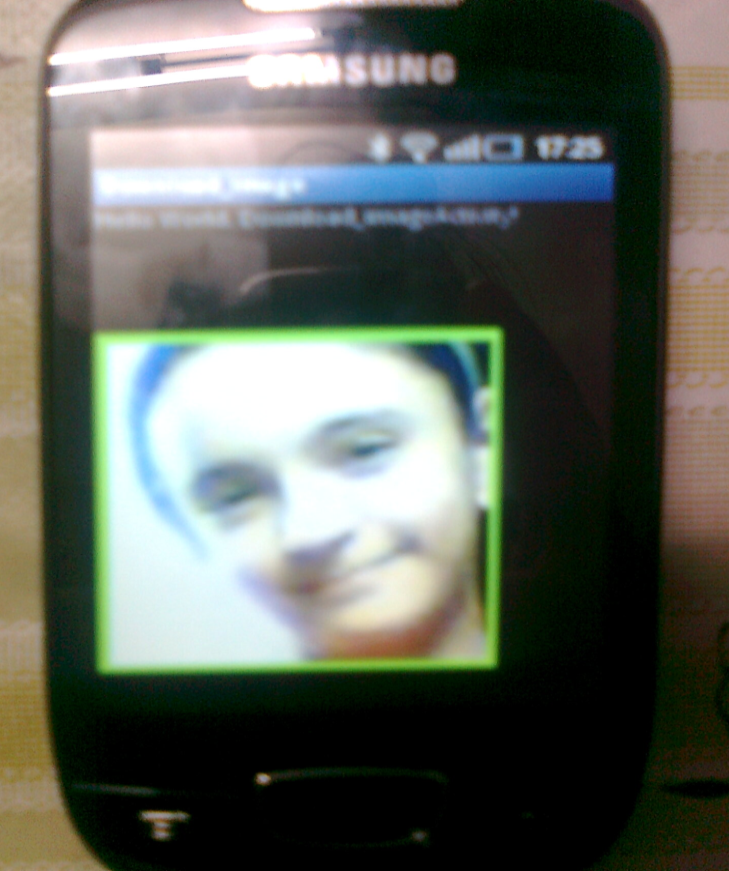


Fig. 12 Image notification in android

* System Communication
* Local Network:

The cameras and the server are connected using local network which we implemented using C++. Each camera is communicating with the server via two ports, the first is for video packets and the second is for detected faces. The server is continuously listening to all ports, and all the sockets are established using separate thread for each socket.

* External network:

The pc and android applications are communicating with the system through the server using the internet. The server that we have used is Apache server.

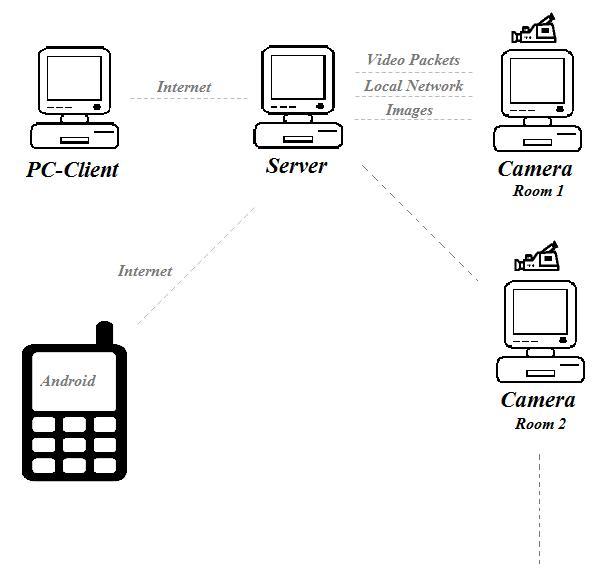
Work Flow

Fig. 13 Work flow

* The system is started, the server is initialized, the cameras are turned on and now the system is ready.
* The cameras work constantly, they save packets of videos with a predefined length (we use 10-second packets) and then sends them to the server. The cameras also perform face detection on its visual feed, in the case that it detected a face, it sends only the face back to the server for further analysis.
* When the server receives the video packets, it converts them to an Android-friendly format (.3gp) so they can played on the device when requested by the user and puts them in their appropriate folder. When the server receives face snapshots from the cameras, it performs face recognition depending on the database present within our system. If the person was recognized, no action will be done, but if the person was not recognized, then the user application-whether the Android app or PC app- will display a notification when it connects to the server. Remember that the server receives new video packets and new images from the cameras using ports assigned to each room that are used to establish a communication.
* When the user is using an Android application, it starts streaming from the most recent video present at the server and then while that one is displaying, the application meanwhile is downloading the video being received at the server side and displays it next, and so on. It also displays a “new image” notification to alert the user of an unauthorized person trespassing their property.
* Our PC-application does the same functionality as the android application. It notifies the user when an unauthorized person is present. As well as providing streaming option.

Problems we faced and how we overcame them

* GUI Freeze Problem

One of the problems we faced was our client-PC’s GUI would freeze when streaming would start and would become unresponsive. After some research, we found out the reason of this problem was misusing the Swing event dispatch thread (EDT).

In any Swing application, there can be a maximum of three different kinds of threads: initial thread, UI event dispatch thread (EDT) and worker threads. The initial thread is the

one that is in charge of running the main method of any application. Its main purpose is to start the GUI, while in some applications it may also be used to read arguments and initiate a few objects. However, regardless of the tasks it is commanded to run, there can only be a single initial thread. UI event dispatch thread (EDT) is also a single thread (under no circumstances can any Swing application have more than one EDT). This thread’s functions include drawing the GUI components, updating them when necessary and calling the application’s event handlers with response to the user interaction. Therefore, we can say that EDT is the only thread that interacts with our UI components and runs event handlers. Since this is the case, all tasks running on the EDT should finish fast enough, allowing our GUI to be responsive to future user input. The worker threads are, as can be understood from their names, the threads that should do all long-running tasks and calculations. Reading or writing large files, communications between databases, transmitting or receiving huge or long data packets can be given as examples of a long-running task. In short, anything that may cause delay or interfere with UI event handling should be given to the worker threads.

This is where our problem begins. In any Swing application, every GUI interaction and event passes through EDT and causes it to become unresponsive to further commands for a certain amount of time. Ideally, this amount of time should not be more than 30 to 100 milliseconds. This concept can be understood better with the following graphical illustration:



The figure above shows the typical response time graph of an EDT performing a task. The time interval between A and B is when the EDT is busy with the task and is not responsive to any user input. Running tasks that will take a long time on EDT will prevent any further instructions to be given during this process and possibly cause the GUI to freeze or crash. Since any programmer will want the users to be able to communicate and give commands to the application at all times, long-running tasks should not be run on EDT. Instead, using worker threads for such tasks will solve the

application’s problem of becoming unresponsive since the response time graph of an EDT that uses worker threads to run time consuming tasks will be as shown below:



As seen from the figure above, EDT is responsive for a longer period of time because the long-running task is now performed on the worker thread. This will allow the user to continue interacting with the GUI and therefore EDT without causing any freezes or crashes.

* How we solved it

We solved this through using `SwingWorker` class that can be seen at the appendix. We invoked the codes to receive live transmission through an instance of the swing worker, and thus using a different worker thread than that of the EDT which helped us pull through from the freezing problem

* Camera Access Problem

When we were working on the projects parts separately, both codes for capturing video packets from our camera and performing face detection on the visual feed coming from the camera required constant access to the camera, so naturally, when we combined both codes in one to have both functionality for our camera as we need them both at the same time, a conflict arose. Our camera cannot be accessed by two applications at the same time, even if both of them are just reading the visual feed it provides.

* How we solved it

The first technique we used to approach this issue was that we decided to capture a picture (frame) after each video packet (which is 10 seconds long). This approach proved highly inefficient, first of all, we thought that capturing only one frame, one snapshot of the environment every 10 seconds will be counterproductive, and it will not provide the level of security we desire. In addition to that, the process of getting the camera ready to capture a snapshot then save it takes around 5 seconds, which means a relatively big delay between our recorded video packets which will make the video streaming later on, whether streaming on the Android or PC, it will make the stream look very broken and incoherent.

So, obviously, we had to seek an alternative, one that will provide us with an appropriate level of security when it comes to how frequent we analyze frames from the cameras visual feed, plus, to be able to do so while also minimizing the amount of delay between video packets captured.

The second technique we adopted was to take 10-second video packets and then analyze each packet before being sent to the server to see if we find any irregularities. After we would capture the video packet, we would then perform our implemented face detection algorithm every dynamically determined amount of time on the appropriate frame. We thought it would be best if we performed this analysis every 4 seconds. Employing this technique enabled us to continuously and competently analyze the environment without risking missing out on a major event occurring. Moreover, we avoided having a big delay between video packets which allowed us to enjoy a smoother stream.

* Android Threads Problem

We faced several problems in working with android threads to download and play the video packets. When we tried to display a number of video packets from the server ,the application was waiting until downloading all the packets is done then playing the last one and if you try to add while loop all the application will stop working

* How we solved it

First we tried to solve this by implementing “Async task” which contains a background function in which we tried to implement a code to download the packets but it didn’t work properly. Finally we solved the problem using handler messages (the thread sends the index of the video to a handler by a message which downloads that video).

Possible Improvements to the Project

* For the purpose of this senior project, we are using a Dual Core laptop, with Windows Vista OS playing the role of the server. A proper server would greatly improve the performance of our system. Especially the video streaming part where delay is most evident.
* Having an entry log that saves location, person, and time location of movements within our environment.
* Instead of deleting old videos at the server, we could add external storage to the system, and save these videos as an archive for later viewing. We could have a weeklong archive and add a “History” functionality for the user to view a certain date-time combination upon request.
* Having an IPhone application that supports our functionality to widen the base of users who can use our security system since these two are the most widespread amongst others.

Conclusion

This system was implemented using a combination of Java and C++ programming languages.

Each language alone can’t perform required tasks easily and efficiently. So we tried to combine the strengths of both languages. Despite the fact that we faced a lot of problems while integrating, but still, it was much more efficient utilizing the strengths of two languages to achieve the end goal, rather than trying to work with just one language, and one environment and trying to overcome all of its weaknesses. We advise people interested in working in this field to intelligently combine working with two languages and know how to properly integrate them and call each from the other as it would make your life so much easier.

Android was fairly easy to deal with, because of its similarities to Java. On the other hand, working with videos was a bit of a challenge, as android’s SDK is still continuously under development, and the media package is still not very strong and is somewhat the underdog compared to other areas of Android.

Appendix

Full project source code will be submitted on a separate CD later on.