Project Abstract
The need for tracking systems with close to live feeds provides an excellent and convenient way
to obtain useful data. The use of a camera sensor network connected via server to a mobile
phone will allow a user from the phone to obtain views from a specific region (indoor, or
outdoor) and be able to obtain previously collected and current data based on an image process
analysis of acquired images from the camera sensor network. The purpose of this project is to
provide home invasion, specifically home burglaries monitoring and surveillance to a user of the
Google Android mobile phone. Our project plan will be to take the common camera surveillance
idea and link it to be used with the Google Android phone platform. Homeowners will be able to
receive notifications via picture on their phone if there is an intruder in the sight of the camera, in
which then they can immediately call one of their on-screen emergency contacts. Not only will
the user be able to get notifications on their phone, but also they will be able to view a feed of the
camera whenever they wish.

Document Revision History
Revision 1.0 2009-02-24 (Initial)
Revision 2.0 2009-03-11

System Architecture
The following content, component, and description tables also describe the system at a high
level:

<table>
<thead>
<tr>
<th>Content</th>
<th>Component</th>
<th>Comment Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter Pin</td>
<td>Masked Number Dialog</td>
<td>Numbered 0-9 per cell and can be scrolled to each number</td>
</tr>
<tr>
<td>Submit</td>
<td>Button</td>
<td>Will send the pin to server and react to validity of the number</td>
</tr>
<tr>
<td>Exit</td>
<td>Button</td>
<td>Will exit the program and return to the menu</td>
</tr>
</tbody>
</table>
### Main Menu

<table>
<thead>
<tr>
<th>Content</th>
<th>Component</th>
<th>Comment Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>On-Monitor</td>
<td>Button</td>
<td>Will enable monitoring mode so the cameras begin streaming information, but won’t detect motion</td>
</tr>
<tr>
<td>On-Arm</td>
<td>Button</td>
<td>Will enable the cameras streaming and detecting of motion; Will actually send response to user</td>
</tr>
<tr>
<td>Settings</td>
<td>Button</td>
<td>Will enter the settings menu so the user can configure the system</td>
</tr>
<tr>
<td>Emergency</td>
<td>Button</td>
<td>This will create an emergency override in case the user feels the need to contact the authorities or a person to check their home</td>
</tr>
<tr>
<td>Status</td>
<td>Dynamic Icon</td>
<td>Will display the current status of the system; a quick view at whether the cameras are on, off, or detecting motion</td>
</tr>
<tr>
<td>Back</td>
<td>Button</td>
<td>Will return to lock screen and close up the system</td>
</tr>
</tbody>
</table>

### Camera Stream

<table>
<thead>
<tr>
<th>Content</th>
<th>Component</th>
<th>Comment Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Zoom</td>
<td>Finger Double Tap</td>
<td>Will zoom into a single camera from a four camera view mode</td>
</tr>
<tr>
<td>Digital Zoom-Out</td>
<td>Finger Double Tap</td>
<td>Will zoom out if zoomed into a certain image</td>
</tr>
<tr>
<td>Change Cameras</td>
<td>Finger swipe</td>
<td>Will switch camera shots if in single camera mode</td>
</tr>
<tr>
<td>Menu Items</td>
<td>Tap camera screen</td>
<td>Will bring out menu items from background when in camera view</td>
</tr>
</tbody>
</table>

### Intrusion Notification

<table>
<thead>
<tr>
<th>Content</th>
<th>Component</th>
<th>Comment Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alert</td>
<td>Dialog Notification</td>
<td>Will draw over any other application and alert the user of intrusion</td>
</tr>
</tbody>
</table>
Design Details

**Presentation Module** - We will be using Widgets that the SDK provides for us and providing functionality for the widgets the android.widget library provided will give us tools necessary to develop the graphical user interface as well as button object provided that gives us functionality with the xml layout editor that we intend to use as our button developer. The displays from the phone will be provided by certain libraries such as android graphics library as well as view library. Even though we work with the XML layout editor it our code will still extend into java as we display images and data from the server.

**Service Module** - The service module provides communication between different modules on the User end it will take care of modules Presentation and network link module and return the work specified by the user. In our case if communication from phone to server will play an important role so we must make sure that the service module in between optimizes that to its fullest.

The logic for the Service module on the phone will be provided by the Phone while on the server end it will be hardcoded through java and any libraries that can handle the task for us. Some libraries include the android......http which will allow us to provide contact between network link module and the presentation module.

**Network Link Module** - An important module that will handle the communication between the Server and the phone for our subtle case we will be using MySQL along with apache tomcat for easier integration between the two systems because of the language which is JAVA. There is a connection library from MySQL for java and apache Tomcat is designed to work better with JAVA.

The phone also supports JAVA and we intend to develop support with the android.net.http libraries or java.net or the org.apache.net

Preliminary pseudo code of the main components in the applications interface are as follows:

<table>
<thead>
<tr>
<th>Check Password Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Check password in login, if true grant access</td>
</tr>
<tr>
<td>• Send query to MySQL lookup to test validity</td>
</tr>
</tbody>
</table>
```java
public boolean checkPassword(int[] list) {
    if mySql.list == true
        return true
        // access granted
        system.interface();
    else
        return false
checkPassword()
{
    count += count;
    if count >= 4
        system.interface.timeout(60);
        // lock out for 60 seconds
}
```

### onMode Function
- Test if user enters on mode
- If the check test is true, then send query for image and update about movement

```java
public boolean onMode(boolean check) {
    if check == true
        serverQuery.send()
        // if on send a query to the server for image
        interface.signal();
        // send updates to phone about movement
    else
        false;
}
```

### offMode Function
- Test if off and turn system off

```java
public boolean offMode(boolean check) {
    if check == true
        serverQuery.send() == false;
        // turn on system
    else
        false;
        // system is on
```
Monitor Mode Function

- If monitor mode is on, query for a server image, but don’t send updates about movement

```java
public boolean monitorMode(boolean check)
{
    if (check == true)
    {
        serverQuery.send();
        // query server for image
        interface.signal() == false;
        // don't send updates about movement
    }
    else
    {
        false;
    }
}
```

Many features that need to be implemented on to this design require many different components. To handle the linking of all these components, we will mainly be using the Google Android SDK. The Google Android SDK has diverse amount libraries that are very useful and this will help us extend the features of our applications to be more user-interactive.

The following class diagram represents the front-end classes used in the SDK.
The following diagram represents the flow of events in the User Interface (front-end):

The Server
On the server, software coded in JAVA will be used to communicate to the database and the camera. A pseudo code example of the connection with the database may look like:

<table>
<thead>
<tr>
<th>Connect and Insert into Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>function connect_insert()</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>mysql_connect(&quot;server&quot;, &quot;username&quot;, &quot;password&quot;)</td>
</tr>
<tr>
<td>mysql_database(&quot;images&quot;)</td>
</tr>
<tr>
<td>mysql_insert(&quot;image to upload&quot;)</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

Database Design
We will be using MySQL to act as our database on the project. Three main programs will be responsible for communicating with this database and they include: the RMSE image comparing program, the program to handle obtaining images from the camera, and the Android phone.

Image Retrieval and Storage
Requests will be made to the camera from a piece of software on the server. After receiving the images they will then be stored in the image table of our database. The image table will consist of a column for the actual image, whether it is safe and nothing has changed or irregularity was found.

Detection of Image Irregularities on the Server
Two images will be retrieved from the image table in the database. One image will be the base image which is the safe image and the other will be a freshly captured image from the camera. Using RMSE or some form of edge detection we will determine the status of the new image. If the image is deemed not safe, it will be uploaded to the server and marked with an irregularity. This image will be relevant in the logs and in determining status on the Android phone.

The following class diagrams represent the back end connection to the SDK. The image analysis will take place on the server. The image simulator will also be ran on the server and the connection will be made to the Google Andriod SDK.

Camera Algorithms
The main process around our idea is that the camera is going to be taking images of a certain location that it will be guarding. These images will be sent to the server, and only there will it make computations to manipulate the image for our implementations.
Processing images in the server gives us the advantage of giving less work to the camera that might be too computationally expensive. In addition, we have the benefit of developing efficient image processing algorithms to detect changes in the image, edge detection, and any other features we might want to do with an image.

**RMSE (Root Mean Square Error)**

To calculate if a current image has changed, we need to do several things, which are suitable for an algorithm to perform. To measure change in an image, we are going to use a well-known technique in image processing to compare images. RMSE is the measure of the error between two quantities. The RMSE algorithm in image processing essentially takes two images, and compares how alike they are. This will happen by comparing the pixel values in one image with its corresponding pixel values in the other image. By subtracting the values of the two images in their corresponding pixels, squaring them, and taking a sum, we are able to detect if the image changed or if it has had no change. To assess if it has changed, we look at the differences of the sums. If it has changed, the difference will be very large. If it has not, the difference will be close to zero.

The image above shows the RMSE being executed over two images that are the same. The calculations would begin like this:

\[
\text{RMSE Result} = \sqrt{\left(233-233\right)^2 + \left(176-176\right)^2 \ldots + \left(51-51\right)^2}
\]

In this example, the RMSE result would be zero, because all the differences give us zero. However, comparing different images would yield a non-zero result. A tentative function prototype would look something like this:

```python
RMSE Detection
```

- 10 -
public boolean detect_change( image current, image old )
{
    //perform RMSE between the 2 images
    For all pixel values
        pixel[i] = iterate_current - iterate_old;
        pixel[i] = value * value
    rmse_result = sum all squared pixel values
    //return true if change is large, false otherwise
    if rmse_result > threshold
        return true;
    else
        return false;
}

Where image current is the very current image that has been taken (the one we are assessing), image old is the image before the current one. After the RMSE has been performed and there is a change, the image will be sent to the user so he can assess what to do with the image. However, this algorithm is only helpful to detect large changes in the image. It does not help us know if that change is because there was someone in the image, or a random object intervening. The camera must be placed where there will be little to no change in normal times, and only large changes when there is a situation that the user wants to pay more attention to.

Advantages and Disadvantages of RMSE
There are a few advantages for using this algorithm over others. Since the customer’s needs are a priority to us, we decided that using RMSE was the best choice. This is because the algorithm is very fast at computing the difference between two images. A fast algorithm is needed in order minimize the time it will take for the system to do a transfer of the file from the server to the phone.

A disadvantage is that this algorithm will only detect changes in an image. We have to check what kind of change is large enough to be considered. Also, the camera has to be in a very static environment and not moving around. This is because any simple change in an image will be detected, and we don’t want to have false detections.

Other Processes of Images
An optional process to do in an image is to detect in a more efficient way whether two images are the same (Perhaps a more light weight method of computing RMSE). However, this would be done on the camera side. This would be useful to reduce the amount of traffic between the camera and the server. If two images are the same, then there should be no need to send the
image to the server. This would result in a more efficient system rather than doing useless computations on images that are always identical. Also, there should be a database of previous images where the user can easily access them at any time from his/her phone. Some kind of pruning should also take place so that very old images do not always stay in the server and are deleted over time. Overall, only those images with significant amount of change will be sent to the server and then saved. This is because in essence, the user only wants to access those images because those are the most important ones.

Example Pseudo Code

| • Enter Main Menu once user has inputted correct password |
| • From Password acceptance we enter the main menu mode |

```java
public interface menuSelection() {
    System.out.draw(mainMenu); //Draw the main interface
    query response from user; //accept touch response
    if response == On-Monitor
        if monitorMode(check) == true //check that monitor mode
            return System.Interface.Stream(); // is not enabled
        else
            System.Interface.Monitor(); //Move to next menu
    else if response == On-Armed //same as monitor mode
        if onMode(Check) == true
            return System.Interface.Stream();
        else
            System.Interface.Armed();
}
```

| • Entering the Log Interface |
| • We are going to maintain logs of significant information for the users convenience |

```java
public interface menuLogs() {
    System.out.draw(logsUI); //draw layout without information
    localLogsDB = serverQuery.logDB(); //reach server download log
    System.Interface.draw(localLogsDB); //apply log information to
    //interface
}
```
• Entering the settings menu allows the user to configure the system
• The settings menu has camera functions

```java
public interface settings()
{
    System.out.draw(Settings);
    localSettings = serverQuery.settings;
    system.Interface.draw(localSettings);
    query response from user;
    apply response to settings array;
    serverQuery.send(localSettings);
}
```

• Emergency Button requires security measures
• A pin has to be applied and then an emergency contact selected

```java
public interface emergencyOverride()
{
    System.out.draw(emergencyButton);
    System.out.draw(pinDialog);
    Query user to enter pin;
    if serverQuery.pin(pin)
        System.out.draw(contactList);
        Query input
        contactList.emergencyCall(input);
    else
        redo pin;
}
```

**Unit Testing**
As a group we will be able to test the product one piece at a time. Methods for handling the storage of images, connection from phone to server, connection of server to camera, and every other part of the software will be tested unit by unit. In order to test the method responsible for the retrieval of images we will use a stub to simulate the action camera.

**Integration Testing**
After unit testing each method in the code we will begin to integrate them to confirm that they work with each other. The integration will happen one module at a time to ensure that each module works flawlessly with the other. For example: If the code to retrieve the pictures from the camera works, and the code to store pictures into the database works, then testing must be done to make sure that the two pieces of code work with each other. When the main sub
catagorical parts are integrated we use use automated code to test each aspect of the code. Automation will be used to run through the menu interfaces on the phone, or quick communication between the phone and the server and so on. This automation will save us the time of manually running through each part of the software.

System Testing
When we integrate all parts of the software such as: communication between the camera, the server, and the phone, as well as the interface of the phone and storage of information we will test the system as a working whole. In the end, all parts should be able to work together with little to no bugs.

Acceptance Testing
A variety of users must test our software in order to ensure that no bugs are present. Alpha and Beta tests will be released so the program could be used in a real world environment which could help us find unexpected flaws. Our testers will make sure that the software does everything that we need of it such as properly store images, check for motion, and interact fluently with the Android phone.

Regression Testing
When a bug is found in any of the testing procedures, the method to find such bug will be recorded so that way we can use that same test for future versions of the program. By doing this we will guarantee that the same bug that was once eliminated does not come back. For example, if clicking a button on the phone results in all the colors turning pink we will note what caused the bug, fix it and run the same test every time we make a change to the program to ensure that the problem does not reoccur.

Plan
Assigned weekly will be a leadership role of one team member. The team member assigned will be held specifically responsible and accountable for the collaboration and accomplishment of those weeks’ tasks. Specifically, that team member will be that weeks “team captain”, making sure that everyone is focused on the weekly task. The team captain will:

1. Assign roles and tasks to each team member of that week.
2. Responsible for holding and calling meetings for group unification.
3. Deliver the accomplished task to teacher’s assistant in lab meeting.
4. Take lead on any presentation given on that weeks lecture meetings.

The following responsible party will not be the only person working on that specific assignment; he will just take the lead role for that particular task. The responsible party will perform an equal amount of work as all others in the group. In sum, all team members will be performing an equal
assessment of the task. For the team member that exceeds their tasks in that certain week, he will be except from the assignment of Responsible Party. The duration of the specific task is an approximation of how long (from the date of the submission of this document 2/24/2009) will take to be deployed. Duration times are subject to change along with the responsible party and tasks.

Below is a projected 6 week timeline from next week (Week 7) through Week 12. After week 12 will be the debugging and further modification for the application:

![6 week Timeline]

Below are in-depth descriptions of the tasks that are to be delivered:

**Interface raw delivery** - The interface raw delivery will be front-end development using the Google Android Software Development Kit integrated with Eclipse (java). This front-end development will consist of menus, application development specifically the graphical interface of the application. The specific usage of the word raw will mean no buttons or menus will actually implement their specific given function. This interface will be simply an interface that allows the group to roughly demo how the project will operate from a user’s perspective. This application development will allow the development team to begin to integrate more of the fundamental operations into the software (e.g sql-lite queries, etc.).

**Sql-Lite setup** - The sql-lite setup will be an implementation of a lookup table of data, specifically images that have been captured by the Citric Mote (camera sensor network if provided by this time). This lookup table will provide data to be viewable through the application MMSG1 (Mobile Monitoring System G1). This is a fundamental step in the delivery of our application because it allows an interface between the Google Android mobile phone with the Citric Mote. The steps that follow allow the image processing of these images to be performed.
**Ubuntu Server setup** - The ubuntu server setup will allow the integration between the Citric Mote and Google Android mobile phone, specifically the processing of data between the citric mote and the Google Android G1 phone. A specific integration of the ubuntu (Linux-based) platform will be to provide a response to the Google Android application MMSG1 on movement within the Citric Mote (within a certain threshold). Pixel intensity values based on the movement will be analyzed by edge detection on the image along with a threshold of alerting. Meaning, to be alerted on the Google Android mobile phone of movement within the Citric Mote a threshold (not yet been set) within the edges of an image will allow a concise report to the user of MMSG1.

**Integration I, II and Interface polish delivery** - These 3 steps will allow the complete integration of all of the proposed use cases and effect described in the Proposal document. The interface polish delivery will be the final presentation of our project MMSG1 with the graphical interface complete and the system ready to use for purposes describe in the Proposal document.

**Weekly Milestones**

**Week 6 (2/23)**
- Complete Design and Planning Document

**Week 7 (3/2)**
- Set Up Ubuntu Server
- Set Up SQL

**Week 8 (3/9)**
- Working Graphical Interface On Android Phone
- Prepare Touch Interface

**Week 9 (3/16): Midterm demo on 3/19**
- Ensure Interfaces Are Working Properly
- Prepare For Phone To Server Communication

**Week 10 (3/23): Spring Break**
- Have Phone To Server Communication Complete
- Prepare Server For Use With Camera

**Week 11 (3/30)**
- Complete Server And Camera Interaction
- Create Edge Detection Algorithm For Photos

**Week 12 (4/6)**
- Activate Camera Via The Android Phone
- Set Up Password Protection

**Week 13 (4/13)**
- Testing And Debugging

**Week 14 (4/20)**
- Testing And Debugging

**Week 15 (4/27)**
- Testing And Debugging
- Prepare For Demo Presentation
Week 16 (5/4): Final demo on 5/7

• Testing And Debugging
• Prepare For Demo Presentation