Motion Detection and Other Imaging Operations for Debian-Based Mobile Devices

By Mehmet Tekman
MSc Computer Science
University College London

Supervisors: Robin Hirsch and Fabrizio Pece

30 August 2012

Abstract

Morphological image processing techniques were performed upon frames captured by the camera on the Nokia N900 smartphone to correctly detect motion for a wide variety of image sizes and motion ranges using the FCam API and CImg processing library in C++ within the Qt Framework.

A commandline interface was then developed to help facilitate time lapse operations in order to efficiently watch a target within a set period of time at specified intervals. The motion detection and time lapse components were merged into a single application dubbed 'WatchDog' and released successfully on public repositories.

IP streaming and webcam operations were also developed but they will be implemented into WatchDog at a further date.

Contents

[2] Acknowledgements

I would like to thank the following people:

At UCL:

Robin Hirsch For just being all-round decent guy: very flexible with my project idea, giving perceptive suggestions, and being incredibly helpful in finding me a secondary supervisor.

Fabrizio Pece For the speedy, concise, and friendly responses to all my queries, and for the truly helpful discussions and guidance into all aspects of the project.

At Home:

Family For being notably patient and forgiving with my odd schedules and moods.

'deed' For being the first user on the Maemo forum to test my application upon release, and the glowing review he gave it.

[2] Note: This write up contains colors that are best viewed from the PDF file on the CD

Part I

Introduction

This project is primarily about motion detection; what it is, how it works, the various approaches attempted, and the final product. There is also a significant amount of work done on time lapse operations, and streaming over an Internet Protocal (IP Streaming).

The project was developed over the course of several months with the initial focus being about IP streaming, but later shifted attention onto motion detection as my interest in the topic grew. The time lapse was intended as an optional feature, but I found myself dedicating a week or so to it since it slotted into the motion detector code so well.

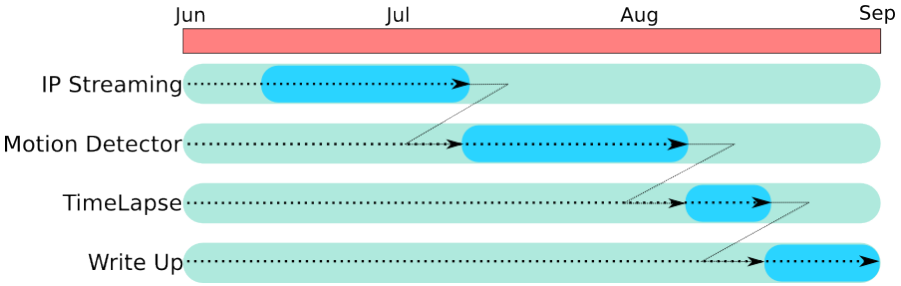


Figure 1: Schedule and division of work

What it is

The project is split into three components, each ordered in terms of importance:

1. *Motion Detector* is a device that is able to distinguish between different components of a scene (i.e. what is moving, and what is not) and react to it.

2. *Time Lapse* captures images at pre-specified intervals, so that when those images are viewed successively it appears that the transition of time has increased.

3. *IP Streaming* sends video or audio data from one machine to another in real time using internet protocols.

What it does

1. *MotionDetector* A simple example of motion detection is if there is only one moving object in a given scene, then often it is enough to simply subtract one image from the previous and watch the difference. The images are treated as bitmaps and their pixel values are mathematically subtracted so that a pixel at grid-coordinate (10,20) in image 1 is subtracted by the pixel at grid-coordinate (10,20) in image 2. The dark parts in this subtracted image would have cancelled each other out because they have not changed between frames. The parts that 'light up' are parts where there was a substantial difference between one frame or the next: movement.

Rarely is there ever only one moving object in a scene, and for a scene watching a man walk past we will not want to capture the rustling trees behind him. In this case a simple subtraction would no longer work and we must perform more advanced techniques (morpholigical operations) such as eroding and/or dilating the image which shrink out or embolden features of an image respectively. Combined, they are a powerful tool and by specifying the level of eroding and dilating one can control what represents significant motion or not.

2. *Time Lapse* On a mobile a time lapse application simply sets up scheduled tasks so that images can be taken at intervals. A video library is then used to compile images into a movie, which when played will make the target appear much livelier than if it was shot in real-time. An interface is used to set up parameters such as start/stop date, interval, image size, and frame rate, but after that the user can then close the application since all the tasks will be performed in the background and do not require user interaction. The user can open the application at any time to check on ongoing jobs.

3. *IP Streaming* Video from the front or back camera of the device is sent in a compressed/encoded format to another device that uncompresses/decodes it, allowing for real time streaming between machines without taking up too much bandwidth. The user can select an active remote device from a dropdown list and application will generate all the necessary code to initiate a stream, depending on what encoding is used, what protocols are used, and the type of target (e.g. a file, a webcam, X Window, or a browser window).

Approaches taken

A great deal of testing, planning, and trial-and-error was performed in order to get each seperate component of the application working correctly, and a great deal more was taken during the merging process of the motion detector and time lapse components to create 'WatchDog'. IP streaming is currently a standalone application at the moment dubbed 'MediaStream' but it will be later merged into WatchDog to create what I llike to call a 'super-app', i.e. an application capable of performing a wide range of video tasks.

Part II

Background

1 Hardware

1.1 The Device

The N900 was Nokia's third attempt at producing an internet tablet phone after the initial success of the N810. Like it's predecessor it sported a hardware keyboard, a TFT touchscreen, accelerometer, 600 MHz ARM Corex CPU, and the same PowerVR SGX Graphics chip came with the iPhone 3GS.

At the time of it's release in 2009 it boasted competitive specs with every other smart phone on the market; supported flash (unlike the iPhone) and could multitask multiple apps at the same time. However it sold poorly due to unfamiliarity of its user interface and lack of 3rd party developers - indeed the only crowd it attracted was its already loyal Nokia fanbase from previous phones.

Despite these setbacks, the N900 has been heralded as one of the most well designed phones of its generation due to its open framework, native interation of Skype, fast boot time via Ubuntu's upstart daemon. It was also one of the first smartphone's to successfully port Android (the Nitdroid project) and could emulate other linux distros.

Figure 2: Front

Figure 3: Back

1.2 Camera

The Nokia N900 came with a 5 MegaPixel resolution, with dual LED flash, and a Carl Zeiss lens with a 5.2mm focal length, a fixed aperture (regulator of light) and focus minimum of 10cm.

The camera sensor uses CMOS technology rather than CCD, which makes it better suited for video than for photo due to CMOS having a larger framerate because of its rolling shutter. CMOS also holds the advantage of being good for night vision, which is a useful feature for a motion detecting application to have.

It supports the following resolutions:

Image: (2576x1936),(2560x1440),(2048x1536),(1280x960)

Video: (848x480),(640x480),(320x240) at 25fps

2 Software

2.1 Maemo OS

The Maemo operating system is a Debian-based platform developed by Nokia for its internet tablets. For the N900 the Maemo OS was in it's 5th revision (Maemo 5) code-named 'Fremantle'. It is based mostly upon open-source code, though certain user-space elements such as telephone operations, taskbar applets, and system daemons were closed source much to the chagrin of the developers, but these were later 'opened' by keen developers after Nokia dropped it's over-the-air updates in 2011.

The desktop uses the Hildon GUI framework which is based on GTK. Software is installed via HAM Application Manager (UI) but also via the classic 'apt-get' command (from the terminal emultator Xterm) native to all Debian distributions supporting the aptitude package manager, since applications are compiled as .deb files as with all Debian flavours.

The media player uses the GStreamer multimedia framework to play videos....

For software development, applications can written in C, C++, Python, Ruby and Mono. Java applications can also be written but these require to a JME to be ran and gaining access to this from the phone is not facilitated easily.

For more detailed information regarding the Maemo OS and the Maemo community please consult page 86 in the Further Background section of the Appendix.

2.2 Development Tools

2.2.1 Scratchbox and the Rootstrap device

Scratchbox is a cross-compilation toolkit used for application development across Linux distributions aimed at mostly x86 and ARM architectures. For Maemo, this is the bare minimum required to develop applications since it comes shipped with the Maemo Application Development and Debugging Environment rootstrap device (or MADDE) required to build the .deb packages from source code.

The Rootstrap is effectively a directory containing all the same prebuilt ARM libraries and files that the phone has. Through a chrooted environment it is possible to install and run the same deb packages that Fremantle uses.
A chrooted environment is when a new directory is set as 'the root' of a filesystem in order to run a process in a modified environment so that the process does not have access to files outside of its current root. To have access to system devices and utilities, chrooted environments often have /dev (device), /sys (system), /proc (process) mounted and bound as well, giving the impression of a full-fledged system within a system.

2.2.2 Qt Framework

Qt is an open-source cross-platform application framework developed by Nokia to facilitate in the creation of developing and testing apps. It uses mainly C++ which even

comes with the GCC toolchain, but also uses QML (Qt Meta Language) which is a Javascript-based language for building applications where fluid animations for widgets is desired. Qt also has bindings for Python, Ruby, Perl, Php, Java, and numerous other languages.

Though Qt was mostly Maemo and Symbian based, the next version of Qt (version 5, currently v4.7) will be aimed at Android, iOS, and Windows 8 mobile devices in the near future, making it possibly the tool for all applplication development in the future. This was due to Digia buying Qt from Nokia in 2012[3]. This is a good thing, since it means that alot of applications will be extremely portable between platforms, and assuming that Project Tizen (see aforementioned Appendix page) has the same open framework as Maemo, may perhaps greatly extend the shelf life and portability of my own application.

Qt SDK

The QtSDK holds many similarities with the Android SDK in that they both have GUI for creating layouts (Qt Designer), both access unique assets via a resource management system (QResources) , and both autogenerate events for widgets.

Android vs Qt

In Android, to create a click event the SDK adds an OnClickEvent within the source file which acts as a listener for that particular widget. In Qt, to create a click event a Signal and Slot pair are created, where the widget gives the signal that it has been clicked, and the slot reacts to the signal by performing the code written by the user in the slot method (expanded in further chapters).

Qt also has external modules for SQL, comes with it's own WebKit (similar to Android SDK), and allows for CSS stylesheets to be used when styling widgets. The webkit and SQL modules are external to the base modules because they must be specified in .pro file of the project by appending them to QT+=sql webkit.

Differences from Android SDK are more due to the differences in their base languages (e.g. Java vs C++), where multiple inheritance is not allowed in Java, and chaining constructors are not allowed in C++.

User interfaces are split into three files: Ui source file, Ui header file, and Ui layout file. Like Android, the layout file is an XML-based heirarachy of tags which define layout order and parent/child widget. Unlike Android, the XML file itself cannot be editted from within the SDK (as of QtSDK version 2.1.2) and must be editted in the background by the Qt Designer interface.

Qt comes with it's own objects to help facilitate common tasks such as string handling via the QString class which lets the user manipulate strings without worrying about the memory management. Standard C++ libraries are included of course and if the user wishes to they can use the std::string module too (though they should remember to delete the string when it no longer needs to be used!)

Using Qt's Meta-Object Compiler (moc), a lot of files are autogenerated to handle

[3]Nokia Sells to Digia - http://www.techrepublic.com/blog/australia/nokia-sells-qt-to-digia/1268

Qt's C++ extensions. The moc tool simply reads the C++ header file and looks for Q_Object macros so that it can handle signals and slots which are native to Qt.

Qt is well integrated with the Scratchbox toolkit and is set up to have automatic access to the libraries defined in the /usr/lib/ directory of the chrooted rootstrap device. To use a new library in Qt, one can either append the directory to the LIBS += variable in the .pro file, or can natively install the ARM package in the rootstrap device using xdpkg, a dpkg clone, in the rootstrap directory via the command:
/../QtSDK/Maemo/4.6.2/bin/mad-admin xdpkg -i ¡package_name¿_armel.deb

Ubuntu vs Windows
The QtSDK was developed in unix and ported over to windows. Linux binaries comes in 32-bit and 64-bit architectures, the latter of which is desirably faster than the former in terms of hardware emulation, building, packaging, and syntax detection.

The Windows QtSDK is only 32 bit, mostly because of bugs in the Mingw64 project which aims to be a complete runtime environment for gcc which would support native Windows 64 and 32 bit binaries.

Naturally, the windows version of the QtSDK builds slower due to the smaller architecture limitation, however it is a great deal more stable as shown later in the report.

Hardware Device, Emulator
The QtSDK supports testing the application on both a Hardware Device and the QEMU Emulator often pre-packaged with Scratchbox (though in recent versions Qt it has been the other way around). Due to the 32-bit restriction of the QtSDK on Windows machines, the emulator can be very slow when run, however this is the opposite when run in 64-bit on Linux.

For the duration of this project the code was tested on my own Hardware Device. This can be setup via regular networking over WLAN, or through USB networking. In order to do this the 'Madde Developer' app must be installed on the phone to enable this functionality, and to also ease in the generation and sharing of SSH keys between the phone and the SDK.

2.2.3 Version Control

The QtSDK also comes shipped with integrated version control with good support for most of the popular versioning control systems such as Git, Mercurial, Subversion, Perforce, CVS, and Bazaar.

Changes to a piece of code can be committed and pushed either through simple Keyboard combinations or access through a dropdown menu.

This was my first time using a versioning tool since I generally backup my last data on a USB stick, and my current data on Dropbox. For this project I chose to use the newer Git distributed version control (DVCS) tool over the classic Subversion centralised version control (CVS) model, purely because this was a solo project and managing multiple commits to central server was not an issue, and also because Git is

generally faster, since there is no need to communicate with a central server. I also used Dropbox to sync my current work as an extra precaution.

The repository I chose to host my work was BitBucket which is very similar to the classic GitHub and Launchpad, but due to its lack of online community and its shameless copying of GitHub's webpage layout, recieves very little traffic[4]. By this logic it would mean that access to the site would be swift and that even if I made my repository public sometime in the future, there would be very few occurences of interfering users hovering over my code and pointing out its flaws. From the QtSDK, branches can be chosen through context menus and changes can be pushed to the server with an optional 'diff' screen to compare changes between versions.

It should be noted that the SDK does not come shipped with Git and it is down to the user to install and setup Git repositories independently of the SDK, which can only access existing branches.

Using the Git Handbook, I followed the 'master' 'develop' 'feature' 'hotfix' branching model where the 'master' branch holds the code for each stable release, 'develop' is branch for commiting new changes and features that are reasonably stable, but not yet ready for release. The 'feature' branch is branch that contains many branches for each new bleeding edge feature that is released such as 'feature/avaraging_method' or 'feature/subtract'. When these features have working prototypes they are then merged with the 'develop' branch. The 'hotfix' branch is a tiny branch that exists for quickly fixing small bugs that are found on a release.

2.2.4 LaTeX

Previous reports of mine have often been written in OpenOffice or Abiword, but this time I decided to try LaTex.

LaTex is an intuitive document creating language widely used in academia to produce proffesional papers and reports using the type-setting program Tex. It has native support for writing equations, producing tables, generating a table of contents, and simplifies bibliography management.

Every part of the document is declared in a single marking tag which can declare a section, or subsection, or subsubsection, or paragraph, or subparagraph.

For trees I used the QTree package, and for bibliographies I didn't want the reader to have to constantly flip back and forth to a bibliography at the back, so I used the *footbib*[5] package that placed references at the foot of the page whilst still maintaining the bibliography database. Footnotes were also frequently used whenever a citation was not likely to be quoted again.

[4] GitHub vs Bitbucket - http://www.pocoo.org/ blackbird/github-vs-bitbucket/bitbucket.html

[5] footbib - mirror.hmc.edu/ctan/macros/latex/contrib/footbib/footbib.pdf

3 Literature Review

Prior to development I was reading through a detailed review by Massimo Piccardi from the Computer Vision Research Group (CVRG) at the University of Technology, Sydney (UTS)[6] on background subtractions. The review highlighted many common problems and pitfalls that would occur in trying to detect motion and how to overcome them. Indeed on page 54 I describe how I fell into the trap of not updating the exposure of the camera and thus not detecting changes in illumination. Piccardi talks about how simple subtraction between one frame and the next is very sensitive to a threshold, since the background is simply the previous image. He goes on to mention a method of accumulating frames as a running averages to overcome this problem and you can see the implementation of both efforts on page 23.

Once I got to the stage where I needed to perform morphological operations I began to read about erosion and dilation techniques from H.E. Burdick's textbook Digital Imaging: Theory and Applications[7]. On chapter 5: Morphological Operations he discusses the uses of the different kernel masks neccesary for erosion, dilation, and what he deems "hit-and-miss operators". I expand more on the topic on page 32.

For timelapse operations there was not many prior works that I could read from, mostly because the topic is quite self-explanatory. I did read a great deal about cron and the alarmd framework as mentioned in detail on page 62.

IP streaming was based on the gstreamer framework and I discuss that at length (with examples) on page 14.

[6] Background Sutraction Techniques - http://www-staff.it.uts.edu.au/ massimo/BackgroundSubtractionReview-Piccardi.pdf

[7] Digital Imaging: Theory and Applications, H. E. Burdick, McGraw-Hill, 1997

Part III

Motion Detector

A Motion Detector is a camera device that uses hardware or software to detect a significant change between scenes and flag that movement has occurred. The first motion detector was invented by Samuel Bagno in the 1950s as a burglar alarm, making use of ultrasonic waves to measure the Doppler change in frequencies to measure movement. Modern day motion detectors make use of the same technology; using infrared to measure changes in heat, or microwave lasers to measure the distance of a scene.
Mobile phone's however do not come equipped with microwave lasers nor ultrasonic emitters, and though a significant amount of mobile phones have infra-red sensors and bluetooth radio sensors, the poor resolution of these sensors makes using them as reliable motion detectors somewhat tricky.
Modern smartphones of today do often come equipped with camera's whose images can be used to detect motion, but all the processing is done in the software which can be slow. Fortunately smartphones of today are fast enough to handle such image processing, and using lower-level languages such as C and C++ the strain on the phone can be greatly reduced.

3.1 Competition

Due to the small community of Maemo users, there are not many apps of this nature on the platform and thus not much competition at all. Even looking at previous Diablo apps or Harmattan apps found nothing that resembled a motion detector. I thought this strange, expecially considering how OpenCV had been ported to both, but perhaps there were unmet dependencies and other problems that occurred during development much like my own. See page 17 for more information.

4 Available Software Evaluation

To start a basic motion detector, the very least that I would need to do is take one frame and subtract it from another to see the difference. To do this I would need to store every pixel value in an array for each image and then perform a sweep over both arrays to compare their pixel values. There must be imaging libraries that I could use that would greatly facilitate such imaging techniques, but first I would need to choose a suitable camera framework in order to take an initial photo.

4.1 QtMobility vs Gstreamer vs FCam

QtMobility is a Qt library that enabed developers to have access to standard mobile functionality such as messaging, contacts, multimedia, and camera. The camera is contained within a QCamera widget and simple calls can be made to the imaging device.
Unfortunately the official stable version v1.1 had camera support for Symbian and

Meego(Harmattan), but not Fremantle. There was an unstable version 1.2 which included Fremantle support, but once I had installed libqtm-dev_1.2 on the rootstrap device, and tried to use the QCamera element from within a C++ class it did not work. This is because the QCamera element has to be accessed through QML, which would be doable but would not be optimized for speed. For this reason I dropped QtMobility.

GStreamer is an open source multimedia framework that is already native to Fremantle since it is the library that the default mediaplayer uses and so would reduce the dependencies of my application. GStreamer makes use of pipelining, byt connecting a number of elements in an ordered fashion so that the output (source) of one element is fed into the input(sink) of another element and so on until it reaches the final element which is usually a file or a network stream.

A previous app of mine was Gstreamer based and a typical command from it would be:

```
gst-launch v4l2src device=/dev/video0 ! \
videoscale ! video/x-raw-yuv, width=320, height=240 ! \
ffmpegcolorspace !  jpegenc 80 ! \
tcpserversink host=192.168.1.101 port=9000
```

This pipes the video-for-linux(version2) camera source into a video scaling element, into a raw video stream of defined height and width, into an element that convert the pixel color depths to type ffmpeg color sizes, into an element that encodes the raw stream into mjpeg format with a quality of 80%, which then pipes it into a tcp server element which broadcasts to the specified host and port.
On the recieving side, a gstreamer pipe would need to reverse the order of the elements (i.e. recieve from tcpserver, feed it into a jpeg decoder, and then specify an output window or file sink.
Gstreamer is very well developed, and has a Good, Bad, and Ugly heirarchy of stable to unstable plugins that can be used to enable a wide array of different multimedia functionalities and formats.

Despite the speed and efficient usage of system resources, GStreamer has an extremely involved API but is difficult to work with since it is prone to cascading errors due to the every element being dependant on the previous element to function. An example of the C++ implementation of the same code as above is shown in the appendix.

FrankenCamera (or **FCam**) is a very easy to use open-source C++ API for controlling digital cameras. The FCam architecture was developed as a joint reasearch project between Stanford Computer Graphics and Nokia's Research Center for controlling the N900's camera, but has since been used to run their own F2 camera built in their labs[8]. Normally camera architectures are not so easily accessible and it is very fortunate that FCam chose the N900 to develop on.

[8]Frankencamera: Experimental platform - http://doi.acm.org/10.1145/1833349.1778766

The FCam API is split into 4 components:

Device The camera device consists of a Lens, a Flash and other different devices. FCam has object classes for each of these with various functionlity enabled.

Shot The shot object holds typical parameter settings such as exposure time, frame time, white balance, and gain for capturing and processing a single image. It is applied to the sensor object.

Sensor The sensor object listens for calls for capture, and outputs a frame by performing all the image pipelining in it's own thread. Once pipeline exits (or the thread terminates) the sensor is ready to be configured for the next frame.

Frame Frames can be accessed from the sensor by calling the getFrame() function which performs blocking on any pending shots. The ease of API is shown here, as this is the *only blocking call in the entire API.*

[9] It has an extremely simple API, since all the user has to do is set the shot parameters, call a new frame, and grab the image from the frame. This can be called any times in succession using the same parameters with comparitavely little lines of code.
Despite being a relatively small project, the API was extremely detailed and had many examples that I could adopt. For these reasons, FCam was the acrhitecture used for the developement of this project.

To use it, I had to add an -lfcam dependency to the LIBS+= heading in the .pro file, and fcam-dev_1.2.deb had to be installed to the rootstrap device.

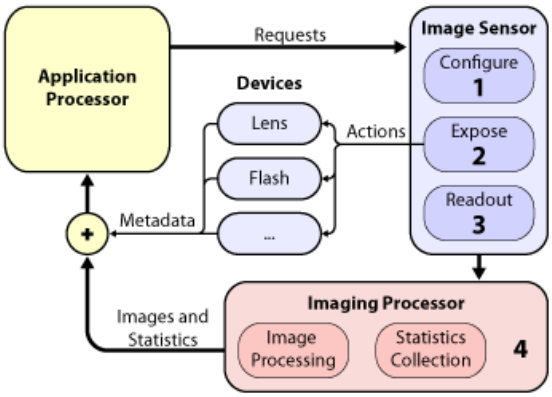


Figure 4: FCam Architecture

FCam has different image sizes depending on the colorspace of the frame that it is being extracted from. For example RAW formats only allow an aspect ratio of 4:3 (2592x1968, 1296x984, 648x492, ...160x120). However using RAW images may be better for precise image processing, but the memory it uses has a lot of overhead and for frequent repetitive use it is not very practical.

Instead the FCam::UYVY colorspace was adopted which captures the image using three components: Y, U(Cb), and V(Cr). Images in this format allow subsampling to reduce the color band resolution (U and V) with a constant brightness (Y) without affecting the image quality discerniable to the human eye. This is because the human eye percieves changes in colour much less significantly than changes in brightness. UYVY

[9]FCam Image - http://graphics.stanford.edu/papers/fcam/

format adopts the 4:2:2(Y:U:V) scaling system, where Y is sampled at every pixel and U and V are sampled at every second pixel horinzontally on each line. This greatly reduces the memory size of the image and speeds up processing with very little difference in quality. Supported UYVY image sizes are:
(2592x1968, 1280x960, 800x600, 640x480, 320x240, 160x120)

4.2 OpenCV vs CImg

Now that I had my camera driver configured, I needed to choose an appropriate image library to perform the processing. Technically I could have written my own library class to perform the simple image subtraction and addition functions that I wanted, but there may have been more features that I might want to use later in development and so it was useful to see what was available. OpenCV and CImg are both common imaging libraries for C, C++, and Python, for use with real-time image proccessing. Both are used widely in computer vision applications, and both contain approximately the same core capabilities, though OpenCV has more extraneous features.

4.2.1 OpenCV

The Open Source Computer Vision Library (or OpenCV) was developed by Intel for use with computer vision applications, but since was dropped and is now maintained by Willow Garage and Itseez.

OpenCV comes with many useful functions for motion detection: image subtraction, averaging, normalising, and even its own method for tracking movement. The older C-based version of OpenCV uses an image format called IPLImage which needs to contain all the image data in order to perform any processing on it, but since version 2.1 the IPLImage format was replaced by cv::Mat which was more C++ orientated and bears many of the extended structural similarities of a matrix struct. This enables it to hold as many different colour channels as it needs where previously the IPLImage format had only permitted four.

Motion can be detected through changes in colour and/or changes in intensity. A motion detector that harnesses changes in both colour and intensity will be more informative than one that measures changes in just intensity. But speed must be taken into account too, and an image with color has pixel values at least four times the size (32-bit) of that which measures just greyscale intensity (8-bit), meaning it will either require more reources or more time to determine whether motion has occured or not . I also reasoned that changes in an image which occur due to changes in colour is not significant enough to warrant motion, since brightness should decrease drastically too.

For these reasons, I surmised that the IPLImage format with the limited number of color channels was enough to perform image processing on, and so cv::Mat was ignored.

In order to perform imaging operations on the frames captured by FCam, I would need to find a way to convert from the FCam::Image to IPLImage. This proved difficult.

The FCam Image format is made up of:

FCam::Image::Image(

int width, int height,
Imageformat f,
unsigned char* data,
int srcByesPerRow
)

The IPLImage has BGR channel order (i.e the first channel is blue, second is green, third is red. This is different from standard RGB formats). I attempted to convert to a grayscale IPLImage by doing: .

Source Code: IPLconvert.cpp

```
IplImage * get_grayscale(FCam::Frame &frame){
    const FCam::Image & image = frame.image();
    IplImage * img = cvCreateImage(cvSize(image.width (), image.height ()), ↩
        IPL_DEPTH_8U, 1);
    for (unsigned int i = 0; i < image.height(); ++i)
    {
            for (unsigned int j = 0; j < image.width(); j += 2)
            {
                unsigned char * pixels = image (j, i);
                CV_IMAGE_ELEM (img, float, i, j) = pixels[1]/255.0;
                CV_IMAGE_ELEM (img, float, i, j + 1) = pixels[3]/255.0;
            }
    }
    return img;
}
```

as taken from some skeleton code for an FCam project at Cornel university[10]
This took a reference to the FCam's Frame, extracted the image from it, and attempted to convert it into an unsigned 8-bit gresycale IPL image with 1 color channel. The unsigned 8-bit image would be limited to 0-255 range.
Unfortunately several problems were encountered and almost three days were lost trying to get to the bottom of it:

A Maemo port of openCV does exist for Fremantle [11] but despite installing libcv, libcv-dev, and many of it's dependencies (libcvcore, libhighgui) the SDK would complain about many missing dependencies when building the application. The culprit turned out to be missing dependencies for libhighgui-dev, but this was false since running a quit dpkg -L ⟨dependency_name⟩ confirmed that the dependency was indeed installed. At one stage I copied my entire /usr/lib directory from the phone to the rootstrap but it would still fail when trying to build.
A few hours of trawling through the Maemo reading lists came up with a bug report [12] from the original maintainer claiming a bug within the build system.
For this reason OpenCV had to be abandoned and a new image processing library had to be used.

[10]Computer Vision course code- http://www.cs.cornell.edu/courses/cs4670/2010fa/projects/p1/codes/project1.tar.gz
[11]Maemo Packages LibCV - http://maemo.org/packages/view/libcv-dev/
[12]Missing dependencies bug - http://lists.maemo.org/pipermail/maemo-developers/2011-August/028583.html

4.2.2 CImg

CImg was suggested by my project supervisor after we discussed the bugs I had been experiencing. CImg is relateivly bug free since it does not exit in any packaged form. In fact all the functions and image processing capabilities are contained within a single header file containing 45,000 lines of code and coming up to 2.54MB in size.
Despite the substantial overhead of including this file in development process, Qt only had to parse the functions within it once and it was set to go.
Another bonus is that it removes a dependency from the project, and makes it all that much more portable. One thing I had to do first was to convert from FCam to CImg.
The CImg constructor is formed of:

CImg(
const unsigned int width, const unsigned int height,
const unsigned int size_z=1, //depth
const unsigned int size_c=1 //spectrum
)

To construct a CImg, all one has to do is call 'CImg⟨T⟩ Name'; where T is the type or size of each pixel value, and name is the name of the image. So for an image that that has 256 colorspace, T = unsigned char = 8bit.
In CIMG, the color channels are in RGB format. OpenCV's IPLImage only supports up to 4 color channels but CImg supports up to any number and this provided a source of initial confusion for me.[13]
Using the same format as the Cornell code I attempted to make: .

Source Code: IPLconvert.cpp

```
void convertToCImg(const FCam::Frame &frame){
    const FCam::Image &image = frame.image();
    CImg<unsigned int> img(image.width(), image.height(), 2, 1);
    for (unsigned int i = 0; i < image.height(); ++i)
    {
        for (unsigned int j = 0; j < image.width(); j += 2)
        {
            unsigned char * pixels = image (j, i);
            img(i,j) = pixels[1];
            img(i,j+1) = pixels[3];
        }
    }
    //return img;
}
```

This wasn't able to produce a consistent image (or any image at all) and I had to play around a wide variety of pixel values: .

[13] CIMG IPL conversion class header text - http://sph2grid.googlecode.com/svn-history/r14/trunk/CImg/plugins/cimg_ipl.h:

Source Code: imagetest.cpp

```
//MAGENTA
newimg1(i,j,0, 0) = pixels[0]; //red
newimg1(i,j,0 ,1) = pixels[1]; //green
newimg1(i,j,0, 2) = pixels[2]; //blue;
//
//BLACK IMAGE
newimg2(i,j,0, 0) = pixels[0]/255; //red
newimg2(i,j,0 ,1) = pixels[1]/255; //green
newimg2(i,j,0, 2) = pixels[2]/255; //blue;
//
newimg3(i,j,0) = pixels[0]; //red
newimg3(i,j,1) = pixels[1]; //green
newimg3(i,j,2) = pixels[2]; //blue;
//
newimg4(i,j,0) = pixels[0]/255; //red
newimg4(i,j,1) = pixels[1]/255; //green
newimg4(i,j,2) = pixels[2]/255; //blue;
//GREEN
newimg5(i,j,32, 0) = pixels[0]; //red
newimg5(i,j,32, 1) = pixels[1]; //green
newimg5(i,j,32, 2) = pixels[2]; //blue;
//BLACK IMAGE
newimg6(i,j,32, 0) = pixels[0]/255; //red
newimg6(i,j,32, 1) = pixels[1]/255; //green
newimg6(i,j,32, 2) = pixels[2]/255; //blue
```

The idea was that by specifying the (i,j) coordinates of the new image and grabbing the pixel values from the FCam image and sticking it into (i,j) coordinates of the new CImg. I attempted to divide by 255 in some of the tests because I wasn;t sure how large the FCam::Image was, and this was my idea of normalizing it. I later realised that the pixel values were already normalized to 8-bit values and that dividing by 255 limited the CImg to {0,1} values would ultimately produce a black image. By digging around through the source of the original fcam application for the N900, I was able to find a function that converted FCam by accessing the image data and reading from it: .

Source Code: CImgConvertFcam.cpp

```
CImg<unsigned char> newimg1(image.width(), image.height(),32,32){
    CImg<unsigned char> img; //constructor
    unsigned char * imgBuffer1 = img.data();
    for (int y=0;y<image.height();y++){
            for (int x=0;x<image.width();x++){
                    imgBuffer1[0] = (*(image(x,y)+1));//+*(image(x,y)+3))/2;
                    imgBuffer1++;
        }
    }
}
```

This resulted in images that looked like that shown in fig. 5

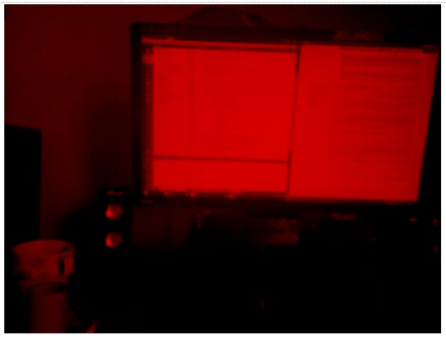


Figure 5: Very red image.

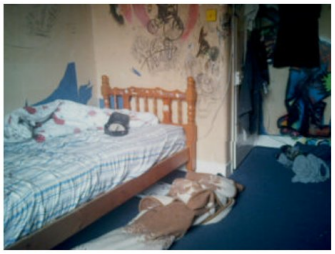


Figure 6: Image saved by FCam

Figure 7: Image saved by CImg

A similar image would appear using imgBuffer[1] instead of imgBuffer[0], and I didn't understand why until I looked up CImg's documentation and realised that out of the 32 channels I was initialising, the first channel is red and that was the only channel I was writing to. That is, for a greyscale image from 0-255 at pixel i,j - I was sticking it into CImg{Red,0,0,0,0....32 times.}, and thus ending up with a red image.
Once I limited CImg to 1 color channel (i.e. black and white) then I got my desired image as shown by the real and greyscale comparison shown in fig. 6 and fig. 7 respectively.

4.3 ImageMagick vs FFMpeg vs VLC vs Mencoder

Now that the image processing side of this was underway, I needed a program that could convert a series of images into a movie format so that when the user returns to their motion detector, they can watch all the highlights in the movie file over the course of a few seconds rather than having to flit through each individual image.

Imagemagick is an open-source image software library for processing images. It can perform much of the same functions and CImg and OpenCV, but it was not considered as a suitable image processing library because the development package had not been released for Maemo and compatibility was a concern. There is, however, a imagemagick package for Maemo that performs the functions from commandline and so accessing its functions would not be a problem if I was resigned to the fact that I would be calling a system process from Qt. CImg has good compatibility with Imagemagick since it's convertToJPEG command actually calls the 'convert' module of Imagemagick.

According to the documentation, to convert a series of jpegs to a movie format, all one has to do is call: 'convert -delay ⟨frametime⟩ input_jpegs_*.jpg output_movie.mpg' But this failed with Maemo specific errors: .

Source Code: Errors

```
Nokia-N900:/home/user/MyDocs/DCIM/MISC\# convert -delay 20 -monitor timelapse↩
    -0000*.jpg movie.mpg
load image[timelapse-000001.jpg]: 479 of 480, 100\% complete
load image[timelapse-000002.jpg]: 479 of 480, 100\% complete
...
load image[timelapse-000015.jpg]: 479 of 480, 100\% complete
mogrify image[timelapse-000015.jpg]: 9 of 10, 100\% complete
Composite/Image[movie.mpg]: 479 of 480, 100\% complete
..
Composite/Image[movie.mpg]: 479 of 480, 100\% complete
Save/Image//var/tmp[magick-XX9TmsAd7.jpg]: 479 of 480, 100\% complete
..
Save/Image//var/tmp[magick-XX9TmsAd63.jpg]: 479 of 480, 100\% complete
convert: Empty input file `timelapse-000001.jpg' @ error/jpeg.c/EmitMessage/233.
..
convert: Empty input file `timelapse-000010.jpg' @ error/jpeg.c/EmitMessage/233.
Nokia-N900:/home/user/MyDocs/DCIM/MISC\#
```

so I was not able to use imagemagick on the device. This is perhaps a good thing since the imagemagick suite is very large and would be a hefty dependency to add to my project

FFmpeg was my next and most immediate option - I have used it countless times before to convert movie files and I have found it's libavcodec library a powerful and useful tool in the past. FFMpeg is one of the most popular free multimedia frameworks around, supporting every single codec and format under the sun. It has a fast release time too and when a new format is released it is not usually a long wait before ffmpeg has a supporting library for it. FFmpeg is primarily used to stream, convert, decode, mux, demux and transcode video files into different formats, but is not used for computer vision.
For converting a series of images to a movie file the documentation states that one has to call:
'ffmpeg -i name_of_files_%05d.jpg -y movie.mpg -r fps'
It is key to note here that the y flag overwrites any output file that already exists, and that the frames-per-second flag '-r' must be placed at the end. FFMpeg requires that the filename for the images are named incrementally, and so the %05d just notifies FFmpeg that the format for the image files are padded by zeros by up to 5 digits.
Despite using different arrangements and different flags, this only partially worked, as the images would be compiled into a movie, but the framerate flag was temperamental and sometimes ignored.
Additionally ffmpeg does not come installed as default by the device, since gstreamer is main multimedia framework used by the mediaplayer and this would also be a needlessly large dependency to have for my project.

VLC is a comparitavely new open-source mediaplayer which can also perform transcoding, streaming, muxing and the same functions as FFMpeg. In fact alot of its codecs are actually taken from the ffmpeg project (e.g. libavcodec), but it also contains a lot of its own.

VLC has only been very recenly ported to Maemo and is accessed by adding user Qole's repository to the /etc/apt/sources.list. It is very incomplete, lacking a decent GUI, but it has a working backend. Unfortunately nowhere in the documentation does it mention being able to convert a series of jpegs to a movie file and I soon had to abondon VLC too.
This is for the best since VLC is not available in the main repositories and users would find it hard to access. This leaves...

Mencoder is yet another multimedia framework released under GPL, and comes as a module with mplayer, a very stable open-source media player that comes with all the codecs and formats that mencoder can use.
To convert a series of images to a movie file one simply has to call:
'mencoder "mf://path/to/images*.jpg" -mf fps=⟨fps⟩ -o movie.mpg -ovc lavc -lavcopts vcodec=mjpeg'
which specifies the that libavcodec is to be used, the framerate and also that the output videocodex should be mpeg format.
Mencoder is also not pre-installed with the device, but many Maemo users do tend to install it first chance they get due to the default Media Player's poor support for certain MPG formats. Mencoder is by far the most convenient of the four video processing libraries to use, not only because it doesnt require filenames for the images to be named incrementally, but also because the framerate is applied correctly and produces the correctly comiled movie. The freedom in the naming of the image filenames also has an additional benefit such that ifthe motiondetector does not record an image to file correctly and the filenames get mixed up, mencoder will still compile the images into a movie without throwing an error, whereas FFMpeg would perform a segmentaion fault. Mencoder was the clear winner, and was adopted into the application and added to the dependencies.

5 Image Processing Techniques

A motion detector works on the basic priniciple that there there exists some drastic difference between two images where motion has occred and motion has not occured. One way to see this represenation is to subtract one frame from another and look for the leftover pixels. Pixels where the image hasn't changed will have the same pixel value and will thus cancel each other out, whereas images where there is a difference between pixels at the same location will have values greater than zero. In the case of unsigned chars, any negative pixels will naturally wrap around and becomes high pixel values (white).

5.1 Subtraction

Subtraction is the technique that performs this, and CImg handles this operation using the basic numerica operator '-', so that for example: CImg⟨T⟩ diff = CImg⟨T⟩A - CImg⟨T⟩ B;

Figure 8 and fig. 9 were normalised to 2 discrete {0,1} values and then multiplied by 255 so that the 'hot pixels' could be viewed with the human eye. As you can see from fig. 8, the whiteness indicates that there was a significant difference between frames and so that one of the images had pixel values vastly greater than the other – indicating a change in the scene.

Figure 8: Image subtraction between two consecutive frames where the scene has changed inbetween

For fig. 9 it can be seen that there is very little difference exists between the frames, yet you can almost quite clearly makeout the edge of the objects between the frames (a computer screen and speaker). Neither the speaker, the monitor, nor the camera moved between those two images but nonetheless a small difference was detected.

Figure 9: Image subtraction between two consecutive static frames where no movement has occured

This served as my first model for image detection by performing a small test to see how many of these sparse white pixels are detected for static frames. This would then serve as the basis for a threshold, so that any subtracted frames that are significantly above this threshold are flagged as 'movement' frames and those below or up to the threshold are not.

5.1.1 Determining suitable thresholds for subtracted images

To find a suitable threshold I used the following pseudocode to determine a threshold for a static frame:

Source Code: subtract1.pseudo

```
1. previous_frame := takePhoto().getFrame();
2. count := 10
3. while count>0
4.  current_frame = takePhoto().getFrame()
5.
6.  subtracted_image = current_frame - previous_frame;
7.
8.  white_pixel_num = countPixelsOverZero(subtracted_image);
9.  print white_pixel_num
10.
11. previous_frame = current_frame;
12. count --;
13. finish
```

It essentially grabs an image and subtracts it subtracted one image from another and totalled up all the pixels with a value greater than zero. This gave the following table of results:

Non-Zero Counts from Subtracted Images					
Test	**Scene 1**	**Scene 2**	**Scene 3**	**Scene 4**	**Scene 5**
1	30747	47298	37881	18801	678
2	9067	9129	33326	7645	8488
3	798	35861	17947	6433	444
4	3841	30449	8273	23614	13385
5	34379	25246	6303	41523	23474
6	9144	256	14960	31458	25138
7	333474	312598	771731	33779	725415
8	34488	2676	14522	1256	12165
9	36351	12497	472	1036	18955
10	6133	23077	12343	31430	25410
Average	49842.2	49909	91776	19698	85355

Table 1

as shown in Table 1 the Average white pixel count is extremely high and varies greatly from scene to scene. Setting a threshold of less than 90000 would work for the five different scenes tested in my experiment, but there's no gauruntee that the fluctuation between frames would not be higher than 90000 for another scene.

So why is there so much fluctuation? For one, the average pixel values of one scene can be radically different to another scene. If the average pixel value for Scene1 is 45 and the average pixel value for Scene2 is 90, then assuming that the ambient light in a scene is decreased by 10%, then this would result in:
Scene1: 45-40.5 = 4.5, and Scene2: 90-81 = 9 — which are proportionately different

counts for the same reduction in ambient light.

My second attempt addresses the proportion problem, and inserts extra code into lines 5 and 7. .

Source Code: subtract2.pseudo

```
1. previous_frame := takePhoto().getFrame();
2. count := 10
3. while count>0
4.  current_frame = takePhoto().getFrame()
5.  current_frame = normalize(current_frame); //to 0,1
6.  subtracted_image = current_frame - previous_frame;
7.  subtracted_image = normalize(subtracted_image); //to 0,1
8.  white_pixel_num = countPixelsOverZero(subtracted_image);
9.  print white_pixel_num
10.
11. previous_frame = current_frame;
12. count --;
13. finish
```

Here it normalise the current frame into [0,1] binary colorspace after taking it. It then performs the subtraction, but this subtraction may result in pixel values greater than 1 due to overflow (i.e. 0-1 = -1 = 255 for 8-bit unsigned pixel), so a normalisation is performed on subtracted image as well.
This results in non-zero counts of subtracted images that are far more stabler as shown in Table 2

Non-Zero Counts from Subtracted Images					
Test	**Scene 1**	**Scene 2**	**Scene 3**	**Scene 4**	**Scene 5**
1	24272	18049	19899	19746	18833
2	23691	21275	19730	21381	20744
3	20607	23001	19261	19711	24503
4	19134	22938	23231	21930	21096
5	22282	20143	19616	24927	22492
6	23242	18886	19865	20627	21682
7	23769	24513	23851	23883	20993
8	21132	22961	23835	24969	23704
9	22978	23870	21845	21534	18847
10	22031	22578	18717	24348	20265
Average	22313.8	21821	20985	22306	21315.9

Table 2

Using the results from this table I decided to set a threshold of less than 26000 (i.e. a white-pixel count of less than 26000 will NOT trigger a movement flag. Anything above will).

However this threshold was purely for 320x240 images. At that stage I had not considered using different image sizes for motion detection, but since FCam supported

larger sizes I decided to test thresholds for these too, expecting a linear relationship between image size and non-zero count.

For this experiment I repeated the same experiment for the five different image sizes, but have only included the average counts per image size. See 6 in the Appendix for the complete table.

Average Non-Zero Counts from Normalized Subtracted Images						
Image Size	Scene 1	Scene 2	Scene 3	Scene 4	Scene 5	STDEV
320x240	22313.8	21821.4	20985	22305.6	21315.9	592.73
640x480	31312.1	31260.7	30740.9	29647.7	31904.8	847.98
800x600	42122.1	36453.9	38656.9	43281.6	42231.3	2877.6
1280x960	64307.7	75045.5	61701.7	65869	67133.8	5029.2
2592x1968	62439.9	70207.9	74170.6	65812.2	71354.6	4656.4

The chart below in fig. 10 is based upon the data in the short table. It takes the maximum averages for each image size and plots them against the image size, with error bars as standard deviation.

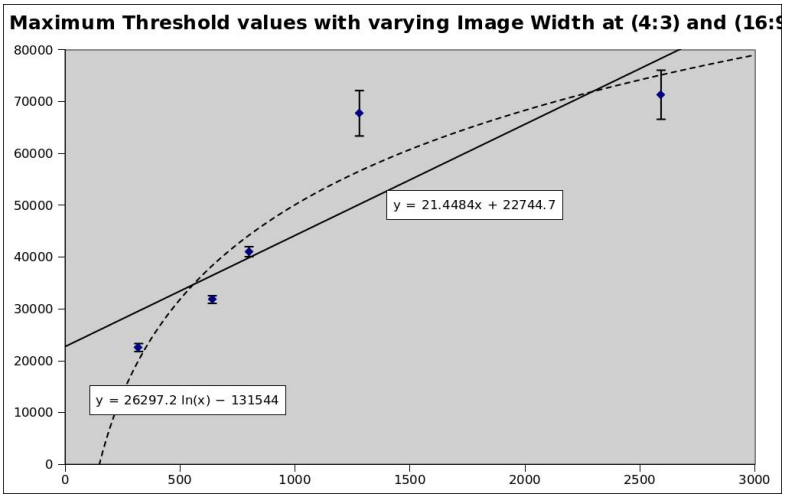


Figure 10

As can be seen in Chart fig. 10, two trendlines have been plotted - a linear one, as well as a more relevant logarithmic trendline because the threshold for 0x0 image sizes should tend towards 0, and not have a y coefficient.

It should be noted that the standard deviation in the average threshold grew as the image size grew, meaning that this was not the most accurate scale for determining a threshold. I did try setting a threshold with image size using the formula of:
Thresh = 26297*ln(image_width) - 131544(+10000 as a max limiter), giving appropriate theshold sizes of:
30145 (320), 48373(640), 54241 (800), 66600 (1280), 85155 (2592).
This method did work, but there were a fair few amount of false positives being flagged. The random scatter of pixels between static frames is caused by Noise, and I needed to find a way to get rid of it.

5.2 Noise

When a scene is captured in some medium, the data within that scene is split into two components:

Signal This is the useful part of the data and is what was either directly sent by the target object, or is the desired data from that target.

Noise This is the unwanted background static that contains no useful information for the specific task at hand, but penetrates through the Signal randomly reducing the quality of the object/transmission.

5.2.1 Thermal Noise/ Dark Current

My previous masters degree was in Astrophysics, a field that was once dominated by eager individuals peering curiously through telescopic lenses and scribbling away at their notes, but has since been replaced by less-eager individuals crouching in front of a screen trying to minimise the inevitable noise inherent in CCD's and CMOS chips.

Thermal Noise (or 'kTC' noise) is the random electical noise generated by the small fluctuations of electrons within a capacitor in a circuit. kTC refers to the formula $V_{rms} = \sqrt{k_B T C}$ where V_{rms} is the root-mean-square voltage of the noise, k_B is Boltzmann's constant, T is temperature, and C is the Capacitance. The formula demonstrates to the relationship between noise and temperature, where the hotter or the more energetic the circuit, the higher the rms noise voltage.

In optoelectronics, this is specifically referred to as *dark current* and it occurs within the wells of the capacitors of in the CMOS grid (the grid where incoming photons depart their energy to the electrons within the conducting metal via the photoelectric effect). Most noise in CCD or CMOS images comes from background noise of scattered pixels entering the photosensitive device from various different sources (other than the target scene), but dark current exists without incoming photons affecting the device - dark current is inherent in the device itself. The noise has a tendency to be spatially fixed during a session, so dark subtraction can be performed where a frame is taken first with the lens cap on, and then subtracted from an image with the lens cap off.

In astronomy this is all part of the standard procedure in obtaining an image with a good signal-to-noise ratio. For mobile imaging, we do not have such luxury to perform dark subtractions before every session (since it is the user who decides when the app is to be used) and so we must make do with other techniques to remove noise from an image.

5.3 Noise Removal

5.3.1 Running Average

In this method, a reference frame is created by capturing a quick succesion of frames. The pixel values from these frames are added to an accumulator and then divided by

the number of frames [14]

This 'average frame' now becomes the reference image which all incoming frames are now checked against it, as shown the pesudo code below

.

Source Code: UpdateReference.pseudo

```
01. REFERENCE_FRAME;
02.
03. function UpdateReferenceFrame(){
04.     count :=0
05.     accumulator;          //Holds pixel values
06.     camera.setFrameTime(0); //As fast as possible
07.     do{
08.         current_frame = takePhoto().getFrame();
09.    accumulator = accumulator + current_frame;
10.
11.         count = count + 1;
12.     }while(count<10)
14.     accumulator = accumulator / 10;
15.     REFERENCE_FRAME = normalise(accumulator);
14. }
```

The reason a method like this is used to reduce noise is because the noise over several frames is 'blurred' out and only the significant motion stands out. This blurring effect reduces the amplitude of the random noise, since it is unlikely that a noisy pixel will occur at the same position across all frames , thus for 100 frames:

if a noisy pixel occurs at position [10,5] with a value of 255 (white), then this will be reduced to a pixel value of 3 after the running average method is applied, since due to the arbitrary nature of noise, the noisy pixel is unlikely to occupy the same pixel position in successive frames. Pixels which retain their values over more than a few frames can then be said to be significant, since it is unlikely for random noise to strike the same pixel twice (unless the scene is generally very noisy)

It should be noted that this blurring effect occurs for all pixels and not just noisy ones, and so there is a slight - but generally acceptable - reduction in sharpness of the image.

The amount of noise reduction is proportional to the square-root of the number of frames being averaged, so in the 100 frame example the reduction of noise would be a factor of 10.

In the following pseudocode, the current frame is subtracted from the reference frame, and if the non-zero pixel count from the subtracted frame is over the (new) threshold amount, then the movement is flagged and the reference frame is updated by another

[14] CImg facilities this very easily by using standard operators to perform operations over all pixel values.
E.g. CImg⟨T2⟩ average = (CImg⟨T1⟩one + CImg⟨T1⟩ two + CImg⟨T1⟩ three) / 3;
Where: sizeof(T2) ⟩ sizeof(T1).
The size of the image types is significant. It should be noted once again that greyscale image sizes are 8-bit unsigned chars with a maximum and minimum values from 0 to 255, so appending a series of pixel values will result in values that overflow this range. For this reason, the accumulator needs to have pixel sizes that can contain accumulated pixel sum amounts. In this project the accumulator had long (64-bit) pixel values, and were then normalised in [0,1] colorspace when being assigned to the reference frame.

quick burst of frames.

Source Code: accumulator.pseudo

```
00. threshold := predefineThreshold();
01. REFERENCE_FRAME;
02.
03. function UpdateReferenceFrame(){
04.    count := 0;
05.    accumulator;         //Holds pixel values
06.    camera.setFrameTime(0); //As fast as possible
07.    do{
08.       current_frame = takePhoto().getFrame();
09.   accumulator = accumulator + current_frame;
10.
11.       count = count + 1;
12.    }while(count< predefineCount())
14.
15.    REFERENCE_FRAME = normalise(accumulator);
14. }
15.
16. //Main:
17. loop
18.     current_frame = takePhoto().getFrame()
19. current_frame = normalize(current_frame); //to 0,1
20.
21. subtracted_image = current_frame - REFERENCE_FRAME;
22. subtracted_image = normalize(subtracted_image); //to 0,1
23. white_pixel_num = countPixelsOverZero(subtracted_image);
24.
25.     if( white_pixel_num > threshold){
26.             print "Movement!"
27.             subtracted_image.savetoFile(DIR/FILE)
28.
29.             UpdateReferenceFrame()
30.     }
31.     // Continue looping
```

Note that the updateReference function has changed so that the reference frame is the accumulated normalisation over any number of frames.
All frames are normalised to [0,1] binary colorspace, as this flattens the image and produces less diference between one frame and the next, which helps distinguish where movement has occured.

As shown in fig. 11 a static scene shows very little difference from one scene to the next so that the image is coherent. For a dynamic scene, fig. 12 shows a wildly different image with traces of many different scenes superimposed.
This approach introduces the concept of having a background or reference, since by the subtraction method of motion detection we would have no knowledge of the background since we would only be comparing two temporally adjacent frames.

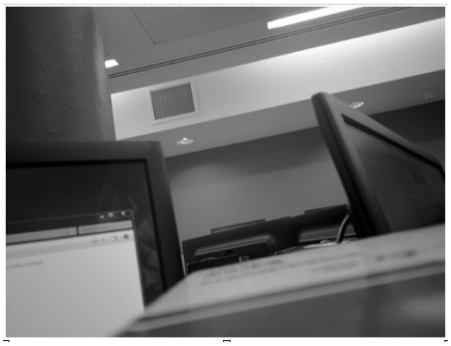


Figure 11: 10 frame accumulation for static scene

Figure 12: 10 frame accumulation for dynamic scene

Running repeated tests under this model confirms that the threshold value is much lower than it used to be due to the reduction in the noise, but there is still large growth of standard deviation as the image size increases. This is due to the scatter of noise that still persists the images, even averaging over 30 frames or more [15] leaves noise behind which are counted as non-zero white pixels in a normalised image.

Figure 13: Scatter of noise for a 1-frame running average

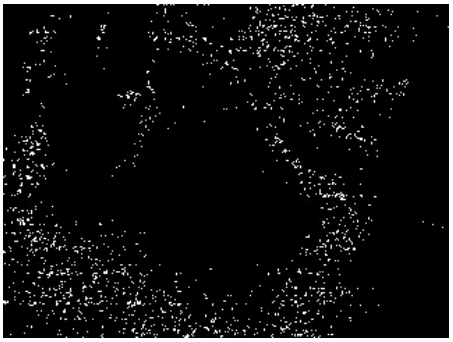


Figure 14: Scatter of noise for a 10-frame running average

The images above show how the running average method reduces the localised noise within certain regions, but increases the overall scatter of the noise as it accumulates around certain individual pixels as it is 'flattened' over several frames.

In fig. 13 the noise is clustered together and makes up a significant part of the static

[15] Averaging over higher even using the fastest frame times causes a significant delay in the application and can also cause segmentation faults.

image. But in fig. 14 the noise is much more sparsely spread and most of the noisy white-pixels are surrounded by empty neighbours.

Clearly another technique is needed to smooth the images so that sparse pixels surrounded by emptiness can be ignored and not contribute to the threshold count. Such techniques are called **morphological operations**

5.3.2 Image Morphology

Another way to reduce noise in an image is to 'smooth out' random white pixels, so that an image dotted sparsely with noise is converted into an image that contains only the most concentrated of clusters. To do this one needs to smooth the pixels. In binary colorspace, smoothing a pixel requires resetting the 0 or 1 value of the neighbouring pixels. If centre pixel A has 8 neighbours (adjacent/diagonal), then the 0 or 1 value of the neighbours is determined by the value of Pixel A.
To **Dilate** a neighbourhood pixel is set to 1 if the centre pixel is also 1.
To **Erode** a neighbourhood pixel is set to 0 if the centre pixel is also 0.
A simple example is shown below:

Original	to Dilate	to Erode
0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0
0 0 0 0 0 0 0	0 0 1 1 1 0 0	0 0 0 0 0 0 0
0 0 1 1 1 0 0	0 1 1 1 1 1 0	0 0 1 1 1 0 0
0 0 1 0 1 0 0	0 1 1 1 1 1 0	0 0 1 1 1 0 0
0 0 1 1 1 0 0	0 1 1 1 1 1 0	0 0 1 1 1 0 0
0 0 0 0 0 0 0	0 0 1 1 1 0 0	0 0 0 0 0 0 0
0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0

To specify the level of smoothing, one needs to consider a **kernel**

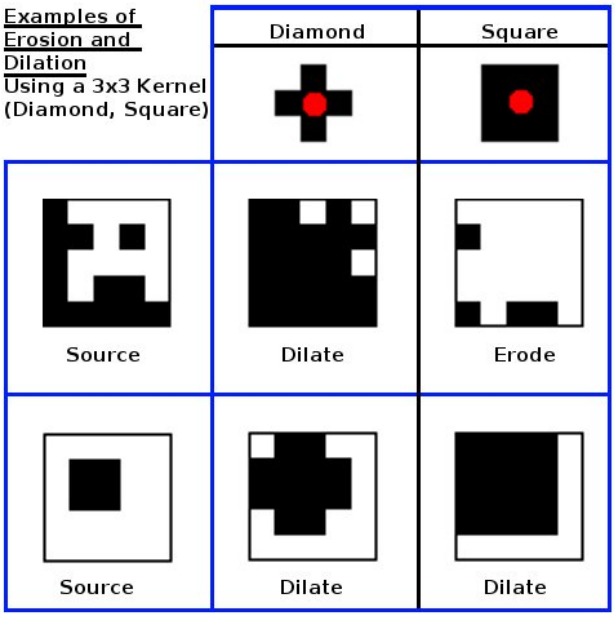


Figure 15

In algebra a kernel is a linear operator which performs an operation upon a set of data. In imaging, the data is a matrix of pixels, and the operator is a much smaller matrix of predefined size that acts as a structuring element and performs basic matrix superimposition upon the pixels and it's neighbours.

A kernel is therefore the size of the neighbourhood being considered when performing Erosion or Dilation operations. Generally a kernel is an nxn square matrix applied as a mask over each pixel, but it is not bound to this shape and can take any nxm best suited. A 2x2 kernel will consider the three surrounding neighbours of a pixel,

a 3x3 kernel will consider 8 surrounding neigbours of a pixel, a 4x4 kernel will consider 15 surrounding neighbours and so on.

For image processing, a kernel normally comes in two shapes: A *Diamond* structure or a *Square* structure. A diamond structure kernel is the least discriminating as it considers only the adjacent neighbours of a centre pixel and not the corners. A square structure is the most discriminating as it considers all the neighbours and will only affect the centre pixel if all the neighbours confirm to the rule of the operation.

Diamond kernels produce softer edges on images, and Square kernels produce blocky edges. Generally a square kernel is likely to remove more pixels than a diamond will, though a diamond is more likely to preserve the overall shape of the image.

A common technique in image smoothing is to perform 'opening/closing', which is simply an dilation followed by an erosion. This accentuates the important parts of the image and also filters out any sparse pixels. The following experiments demonstrate this.

5.3.3 Experiments

As of version 1.1.2, CImg comes with its own dilate and erode functions, and the kernel can be specified by a simple array. A series of tests were performed:

Starting image fig. 16 shows the difference between two completely different frames:

Figure 16: White pixel count of 87516.

Test 1: White pixel count variation with repeated Erosion/Dilation using a 2x2 mask (See the appendix for the code used.)

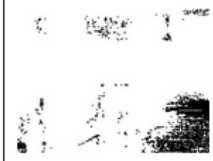
k	Erode	Dilate	EroDil	DilEro
1				
1	66634	112327	78087	103037
2				
2	56214	121853	78087	103037
3				
3	48617	128853	78087	103037
4				
4	45989	132925	78087	103037
5				
5	43721	135904	78087	103037

Figure 17: Erosion, Dilation, Erosion followed by Dilation, and Dilation followed by Erosion all repeated k times

As we can see from fig. 17 – repeated uses of Erosion/Dilation (or vice versa) does not affect the white pixel count for repeated usage as they do independently.
It is interesting to note that EroDil produces a more solid image (i.e. black parts dont have white spots within) than DilEro. This is most likely because the sparse pixels have been filtered out first, and then accentuated. If it were the other way then the sparse pixels would have been accentuated first by the dilation, and then persisted under the erosion. One thing to note is that not much has been clustered. It seemed like the 2x2 mask was too small and so I repeated the experiment with increasing mask sizes

Test 2: White pixel count variation with repeated Erosion/Dilation using an increasing mask
The same starting image is used, as loaded in from a previous save.

Acc. Mask	Erode	Dilate	EroDil	DilEro
2x2				
2x2	58327	73539	63390	72588
3x3				
3x3	49386	74485	60521	73665
4x4				
4x4	41903	75066	58105	73826
5x5				
5x5	34558	75708	56245	73876
6x6				
6x6	27143	76254	54549	73896

Figure 18: Erosion, Dilation, Erosion followed by Dilation, and Dilation followed by Erosion using kxk mask size

As expected the white pixel count increased in the EroDil and DilEro techniques with increasing square masks, however EroDil seemed to give the best quality bunching out of all the technique. Dilate by itself seemed pretty effective too, but this alone would not remove noise effectively. Dilero had to be dropped as useful image processing method since it seemed to be removing significant parts of the image and not retaining the overall blobby structure.

Test 3: Seeing how white pixel count varies with repeated seperate calls to Erosion and Dilation

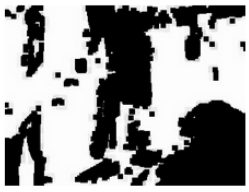
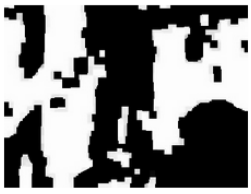
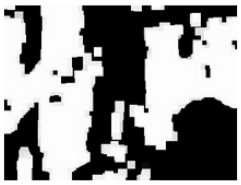
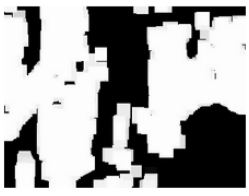
Mask:	67164		
2x2D		2x2E	
	58327		28436
3x3D		3x3E	
	49386		31079
4x4D		4x4E	
	41903		35101
5x5D		5x5E	
	34558		40587
6x6D		6x6E	
	27143		47209

Figure 19: Dilation and Erosion varying with a kxk mask

As expected, the white pixel count diminishes under the repeated Erosion calls, then grows under the repeated Dilation calls. Clearly using these techniques seperately serves no purpose other than accentuating white and black respectively. However combined together into an EroDil method, the structure of the image itself is accentuated and not just one specific color as shown in fig. 20

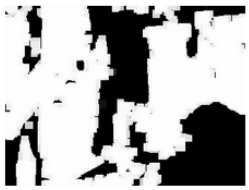
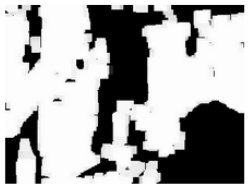
Erodil:	67164		
2x2		7x7	
	63390		53244
3x3		8x8	
	60521		52483
4x4		9x9	
	58105		51442
5x5		10x10	
	56245		51030
6x6		11x11	
	54549		50182

Figure 20: Dilation and Erosion varying with a kxk mask

The erodil images were taken at bi-intervals, but the intermediate mask sizes were performed. EroDil increases the white pixel count, but gradually – and this makes sense since the image is mostly white, so the increasing white pixel count is representative of this. In terms of tidiness, EroDil seems to produce a smoother image, and gives a white pixel count that is closer to the true white pixel count of 67164, than anything that Eroding or Dilating could do by themselves.

Test 4: Seeing how White-Pixel count varies with increasing Mask Size using EroDil method:

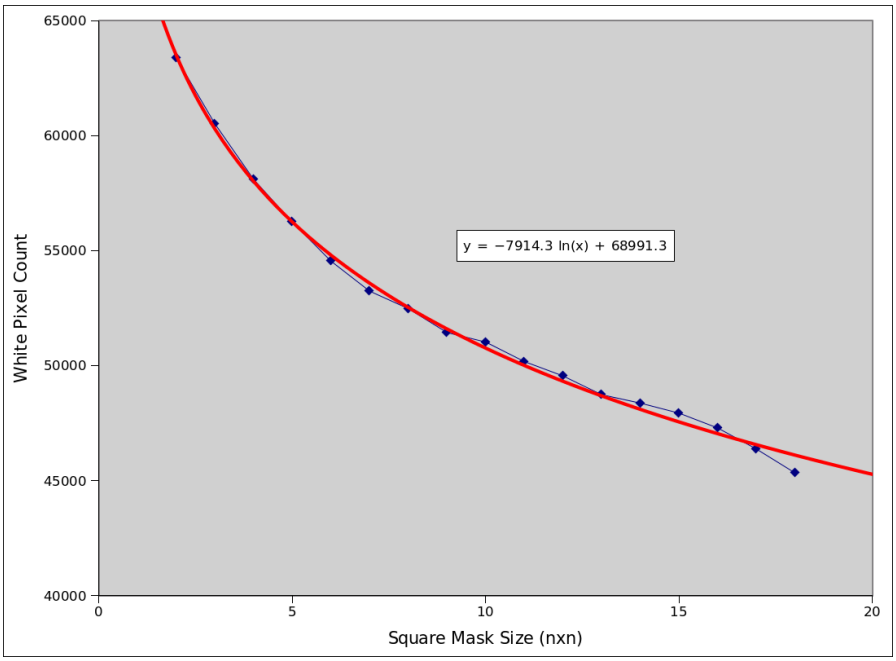


Figure 21

As seen in fig. 21 the white-pixel count follows a downward logarithmic trend that is very well approximated by the equation:
WhitePixelCount = 68991.3 - 7914.3 ln(MaskSize)

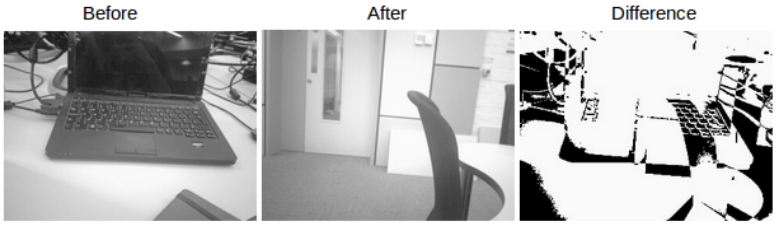
It should be well noted that applying a larger mask filter to an image that has been affected by a smaller mask filter, is **exactly the same** as simply just applying the larger filter. In my code in the appendix what I tended to do was repeatly loop over and update the existing image with each increasing mask, such that the input for mask B would be the output for mask A, where BxB ⟩ AxA. The reason why I did this was to prevent the application from constantly replacing the current image in the memory, rather than having to duplicate from the original for each mask.

Various other tests were performed: See page 95 in the Testing section of the Appendix, but these core 4 tests demonstrated all I really needed to know about opening/closing an image - i.e. As the mask size increases the white-pixel count converges downwards at an ever decreasing rate. Using this knowledge I then set on detecting different kinds of movement.

5.4 Movement Types

I broke movement down into 5 different types: Scene change (80% of the image), Medium movement (40%), Small movement (10%), and Tiny movement (less than 3%).

For scene changes, I moved the camera from one position to another between frames. As implied in Mask sizes have to get very large (to the same order as the image dimensions!) before a subtracted image for differing scenes can fully converge to a single colour:



Mask	Image	Mask	Image
12	50597	57	20817
17	48724	62	20639
22	46484	67	19716
27	43421	72	19167
32	39702	77	18643
37	37814	82	18101
42	34856	87	17258
47	25071	92	17258
52	23049	97	16694

Figure 22: Base image undergoing EroDil to large mask sizes. Even a 97x97 mask couldn't fully vanish the scene

Since the mask size required to completely ignore a scene change is of the same order of the dimensions of the image. To ignore a scene change, it seemed apt to simply set a lower limit on the White Pixel count (e.g ⟨25000 is good enough).
To detect medium movement I placed the camera to face the screen of a computer and dragged a window across the screen. fig. 23 shows the drop in white pixel count as the Mask size increased until vanished completely for a 16x16 mask size

Mask	Image 2964	Mask	Image
4	2315	11	767
5	2082	12	756
6	1842	13	756
7	1680	14	695
8	1568	15	499
9	1192	16	0
10	967	17	0

Figure 23: Medium movement, ignored at 16x16 mask

To detect small movement I repeated the same experiment, but this time made the contents within the window change (i.e. a movie) without moving the window itself. A small window in the computer screen will change between the images, but the overall scene wont change. This test will determine for which mask sizes movement is no longer detected. For filename reasons, the masks increase in multiples of (4*i)+2 (e.g 2,6,10,14,18,etc). Figure 24 shows that movement is ignored up to a mask size of 5x5:

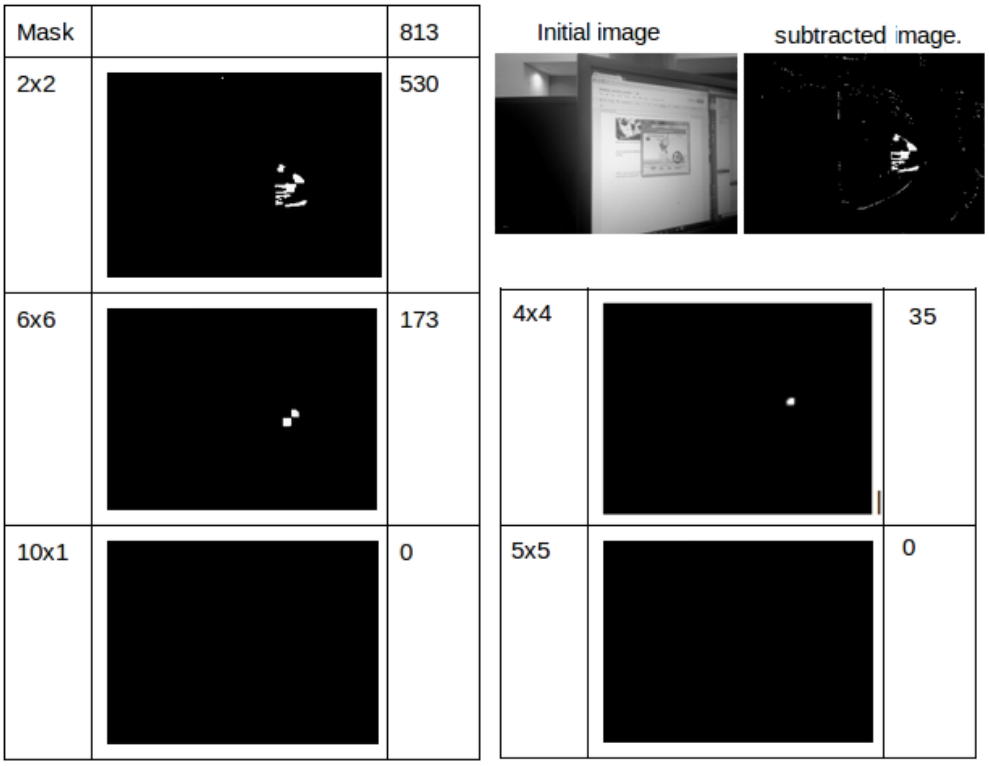


Figure 24: Small movement, ignored at 5x5 mask

To detect tiny movement I placed a single blinking cursor on the screen. The experiment was repeated twice.

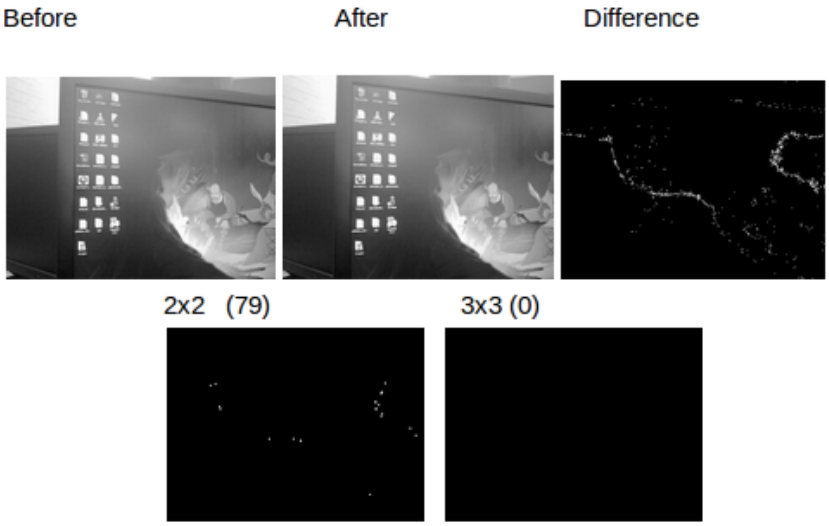


Figure 25: Blinking Cursor

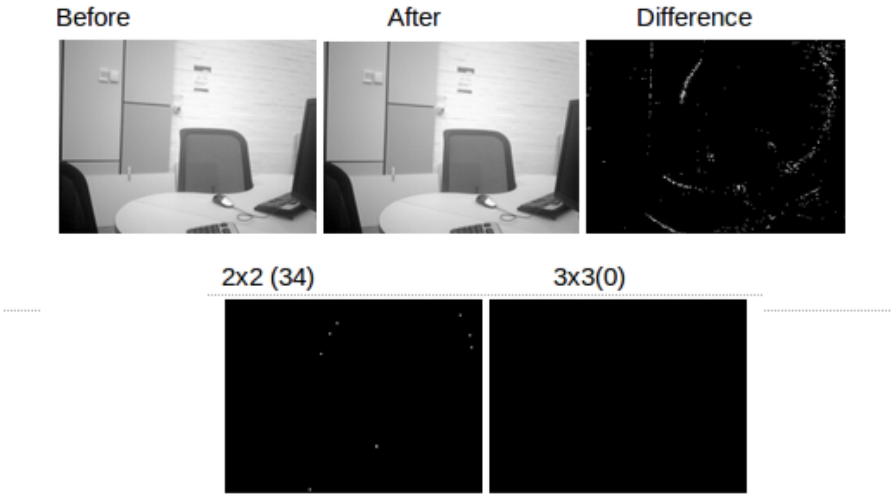


Figure 26: Static scene - Noise

As shown in fig. 25 and fig. 26, something appears to be picked up for a 2x2 mask, but both are ignored for a 3x3 mask. This means that tiny movement is of the same order as noise, and so a 3% difference between images cannot be accurately flagged as movement.

6 Threads

Threads are processes that appear to run at the same time to other processes. On multi-core devices, the illusion is true: two processes genuinely can run in different cores at the same time. On a single-core device such as this, the illusion is false: the OS scheduler switches between different processes to allocate time-slices to each process at regular intervals.

In mobile applications, threading is necessary as often a background task needs to be run which will halt the user interface until the task is complete. In the first draft of my application, this was the case - the buttons and widgets that controlled the camera execution would become unresponsive until the camera had finished its process, where control would then flow back to the user-interface.
This is fine in the situation that the user presses a button and walks away from the phone, but if more control is desired (i.e. the ability to terminate the application while it is running) then threading is necessary.

6.1 Camera Thread

Originally my Camera Class was simply an object class. I would create a Camera object, and tell it to perform functions, and then delete it when it was no longer needed. To place it into a thread, I had to change it from an QObject class to a QThread class. In Java this would be a problem, since only single-parent inheritance heirarchies are permitted, and so the class would lose it's object methods (clone(), delete()), but in C++ a class can have many parents, and so I was able to inherit both QObject and QThread classes. However, this caused many conflicts as errors sprung up detailing multipled definitions of the same function. By going through the Qt API I discovered that QThread actually already inherits QObject (as do all Q Classes) and so the Object methods that my Camera class originally had would not be lost if I simply inherited from QThread.

QThread has a protected method called start() which is the same as Java Thread's run(), which is a method that must be called to run the thread after the thread object is created. Originally this just performed the standard functions from before: initialise camera, check for movement, and terminate. But I wanted to be able to stop the thread safely at any point from the UI without crashing the application. In order to do this I had to use Qt's Signals/Slot's framework.

6.1.1 Signal/Slots

Signals and slots are a core mechanism in Qt that sets itself apart from all the others. It is essentially a notification framework, so that when Object A performs something then Object B can know about it. Traditionally, these are enabled through the use of callbacks, where a global pointer to a function is passed between objects to enable communication. However callback's are not very type-safe since the pointer can reference

anything and there is no gauruntee that the callback will be called with the correct parameters.

Signal and slots[16] provide an alternative to this. When a particular event occurs in Object A, then A emits a signal for that event. A slot is then called to handle the signal, where a slot is simply a function which has the extended ability to listen for signals. If Object B has a slot that is paired with a signal from Object A, then B will run it's slot when A emits its signal.

Pairing signals to slots is a relatively simple process: One simply calls: connect(ObjectA, SIGNAL(mySignal), ObjectB, SLOT(mySlot)). It should be noted that this is type-safe, such that the argument of mySignal must be of the same type as the argument of mySlot. Both are void functions. connect is a function of QObject, and so to enable signals/slots, one must inherit a QObject (or any one of it's subclasses). A Q_OBJECT macro will then have to be declared in the header file of that class to enable signal/slot functionality.

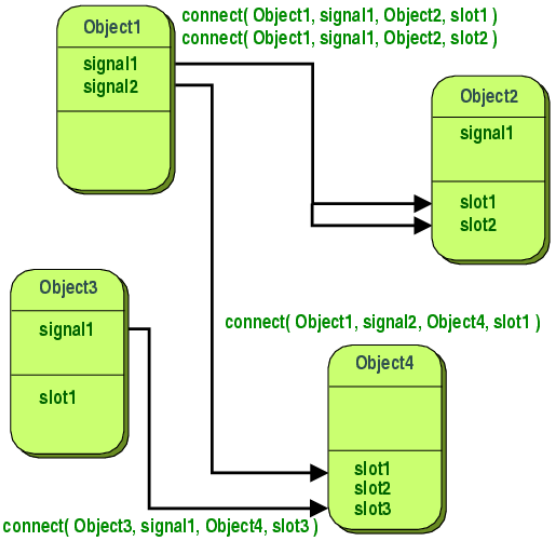


Figure 27: Representation

Pairing usually occurs automatically: When a button is clicked, Qt creates a slot for it in the source file and the user can then edit the slot to perform some task. The connect:: script is never seen in the source file, but is included in the autogenerated MOC files and so the implementation is seamless.

Custom slots are another story. Though this mechanism is completely native to Qt, it can very temperamental trying to create custom slots. Often you will need to clean, rebuild, make, clean, rebuild, make, and so forth until the custom signal or slot is accepted. This is undoubdetly a bug of Qt and has been reported by previous users.[17]

For the Camera thread, I paired a stop button signal from the UI thread with a stop() slot in the camera thread. The stop() slot in the camera thread simply changed a class boolean variable called 'stopNow' to True. In every function within the camera thread, whenever a loop occurred stopNow!=True would be a condition that would terminate the loop. This would exit the function early and cause the Camera Thread to perform its Deletion function and stop the camera thread safely without crashing the application.

As I got more used to the mechanism, I started to harness it more effectively to the point where I began passing whole images from the camera thread back to the UI. A useful feature of signals and slots is that it's not just a flagging system, but can also transport data between threads. To pass a whole image, one simply defines the argument

[16]Image (Slots/Signals) - http://doc.qt.nokia.com/4.7-snapshot/signalsandslots.html
[17]Bug report - https://bugreports.qt-project.org/browse/QTBUG-17072

of the Signal in the camera thread as having an argument of type FCam::Image, then when one wants to emit the signal, one simply calls' emit mySignal(image_arg)'. In order for Qt to recognize the type of object being passed between threads, the argument type needs to be registered via 'qRegisterMetaType⟨FCam::Image⟩("FCam::Image")'.

Passing whole images can be wasteful however, since it requires cloning the current image to memory and passing it to the UI. For small images this is not much of a problem, but for large image sizes this can cause significant overhead since cloning will take a long time and two copies of the same image will take up a significant amount of memory. Instead I passed a reference to the image '&image'. This is risky because that reference may refer to an address where the image is being use by another thread that changes it.

Technically a mutex variable would prevent the camerathread from changing the address of the image while the Ui thread is still reading it, and Qt comes with it's own QMutex object which can lock and unlock critical sections of code so that only one thread can be performing something critical to the same object at any one time.

I dedided against using this since I managed to get a stable application by specifying that the new frame was to be at a constant address and that the newImage signal was also constant 'const FCam::Image'. This made sure that I no longer needed to delete frames after I had processed them, since any new frames would simply be loaded into the same address overwriting the old frame. The only time I needed to delete an image was on thread exit and in the long intervals between frames. This greatly saved on system resources and reduced the size of the signal.

The newImage signal was only ever emitted when movement occurred, otherwise it lay dormant between regular unmoving frames. To save further overhead, Qt allows for signal/slot pairing to be unpaired with 'unconnect', so I created a tickbox in the Ui thread which when unticked would unpair the newImage signal with the slot, and when ticked would pair the two (see page 76 for the 'Show Image' tickbox in the WatchDog App).

6.1.2 QImage, QPixMap, and CImg

Once I was sure that the image was being recieved in the UI, I needed to display it in my main window. This proved harded than originally anticipated. To display an image in Qt, one needs to create a QLabel widget and apply a pixmap to it. A pixmap (or QPixmap) can be created from a resource file specified at build time, or can be specifed from a QImage. Since a QImage is merely a container and does not need to be an absolute resource, I decided to convert CImg to QImage.

My initial attempts brought about images that were misaligned, badly coloured, and often times blank. To get the root of the problem I attempted many different size configurations as shown by the following shell code: .

Source Code: shell1.cpp

```
for left in $(seq 0 50); do                    # Shell script calling
   for top in $(seq 0 50); do                  # -t flag for different
      for mod in $(seq 1 0.2 3); do            # image sizes
         for off in $(seq 0 10 100); do
            /opt/motiondetect/bin/motiondetect -t $mod $off $left $top
         done
      done
   done
done

###==========C++ code to handle -t flag==========##
if( (t_index=args.indexOf("-t"))!=-1)
{
       mod = args.at(t_index+1).toInt(); offset = args.at(t_index+2).toInt();
       left = args.at(t_index+3).toInt(); right = args.at(t_index+4).toInt();
}

QImage thumbQ(image(left,top),
    image.width()/mod + offset,    image.height()/mod + offset,
    QImage::Format_RGB32);
ui->imageLab->setPixmap(QPixmap::fromImage(thumbQ));
```

This attempted to make a thumbnail out of the image recieved from the signal and convert it into the appropriate format in a QImage. This would then be converted into a pixmap and assigned to Label for dispaying.

Arguments from commandline were passed in generated from a shell script to vary the image size slightly so that both the width and height of the original image were divided a 'mod' factor and 'offset' by a certain amount, as well as being offset from the top left of the image by pixel amounts of 'top' and 'left', varying from 0 to 50 in increments of 1. 'offset' was varied from 0 to 100 in steps of 10, and 'mod' was varied from 1 to 3 in increments of 0.3.

Needless to say the script took a long while to finish executing, but I wanted to be as thorough as possible in finding a suitable image size and alignment that displayed a usable image. Most of the images came out horribly deformed and squished, except those that had a mod value of 2 and offset 0. As shown in fig. 28 I also quickly learned that the top and left arguments simply state where the starting pixel to start drawing the image should start.

Figure 28: Two usable images with width and height divided by 2. The left has no offsets, but the right has a vertical offset of 17 pixels.

It seemed strange that the QPicture needs image boundaries at half the actual image size it is containing, but despite scouring through the QPicture API I could find no reasoning for it. Nonetheless, a usable image was found, I just needed to adjust the color channels for it. To do this I adjusted the shell script from section 6.1.2 to adjust just the Image format and size, just in case size was somehow dependent on the format (it shouldn't be since they are two completely different entities, but I wanted to cover all bases). .

Source Code: shell2.cpp

```
#Shell script to test Image Formats and Sizes
for ii in $(seq 4 16); do
    for jj in $(seq 1.4 0.2 3); do
        echo 'index='$ii 'divider='$jj
        /opt/motiondetect/bin/motiondetect -t $ii $jj
    done
done

void MainWindow::newImage(const FCam::Image &image){
    int ind = 0; float div = 2;
    ind = indy;  div = divvy;
    enum QImage::Format index;
    switch(ind){
    case 0:index = QImage::Format_ARGB32; break;
    case 1:index = QImage::Format_ARGB32_Premultiplied; break;
    case 2:index = QImage::Format_ARGB4444_Premultiplied; break;
    case 3:index = QImage::Format_ARGB6666_Premultiplied;break;
    case 4:index = QImage::Format_ARGB8555_Premultiplied;break;
    case 5:index = QImage::Format_ARGB8565_Premultiplied;break;
    case 6:index = QImage::Format_Indexed8;break;
    case 7:index = QImage::Format_Invalid;break;
    case 8:index = QImage::Format_Mono;break;
    case 9:index = QImage::Format_MonoLSB;break;
    case 10:index = QImage::Format_RGB16;break;
    case 11:index = QImage::Format_RGB32;break;
    case 12:index = QImage::Format_RGB444;break;
    case 13:index = QImage::Format_RGB555;break;
    case 14:index = QImage::Format_RGB666;break;
    case 15:index = QImage::Format_RGB888;break;
    }
    cout << "image/"<<div<<" format=" << ind << endl;
    QImage thumbQ(image(0,0),
                  (int)((float)(image.width())/(float)(div)),
                  (int)((float)(image.height())/(float)(div)), index);
    ui->imageLab->setPixmap(QPixmap::fromImage(thumbQ));
}
```

This resulted in yet another flurry of badly aligned images as shown in fig. 29 Out of all the images in fig. 29 , only three were actually usable - all ocurring where the image dimensions were halved. This put to rest my earlier unfounded suspicions that image size was affect by the image color format. The usable image formats were QImage::Format_ARGB32, QImage::Format_ARGB32_Premultiplied, and QImage::Format_RGB32.

Once again, the color was slightly off again, appearing more magenta than it actually was.

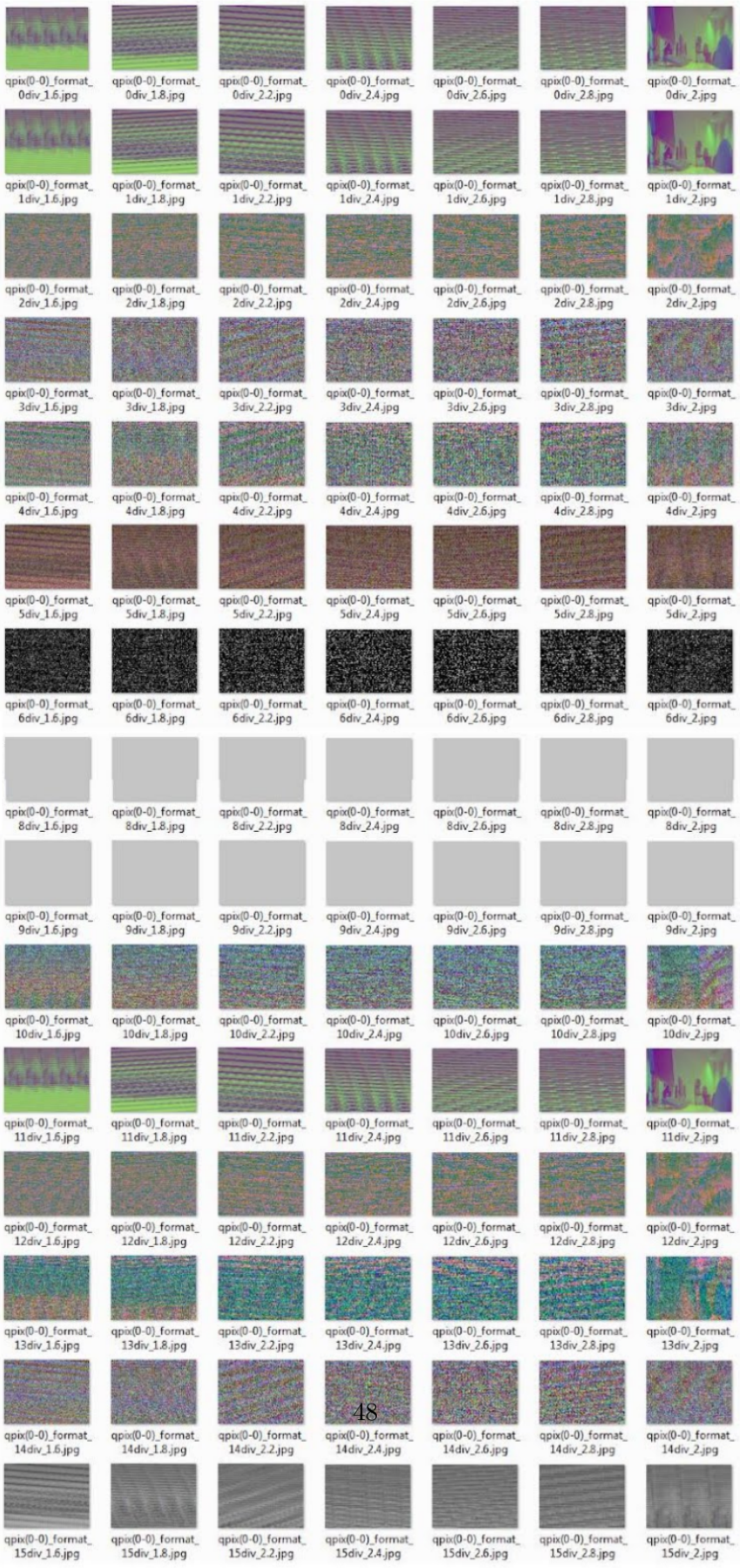


Figure 29: Usable images are where 'div' amount equals 2 and format is 0, 1, or 11

Figure 30

Wondering if the problem was to do with RGB channels being in different order for QImage's (the idea came to me after remembering that OpenCV's IPLImage format was by default BGR), I swapped the blue and red channels using QImage's swapRGB() function. To my surprise this produced very little difference in the outputted images as shown in fig. 30

The difference between RGB32 and ARGB32 is that the latter has an Alpha channel which is used for image types that support transparancy such as PNG. The Alpha channel itself holds the pixel value of the pixel behind the current image itself, enabling it to seem as though it is transparent. The premultiplied version is almost identical to the standard ARGB32 format, but the red green and blue channels are 'premultiplied' by the alpha channel so that the image can softly fade into transparency instead of the transparency suddenly cutting into the image. It produces very gradual and precise images, but with lots of processing overhead.

Considering how my application has no need for transparency I dropped these two formats to save image storage overhead, leaving me with RGB32.

This in itself was wasteful, since in CImg the format I was using was an 8-bit unsigned format, and so the image I should be passing should be the 8-bit CImg. Unfortunately Qt did not agree with me on this point since in order to pass it as a valid signal I would have to include the CImg header file in the header file of my camera class. Qt did not like this (by throwing errors) and only let me place the CImg header in the source file, and so I was not able to pass the CImg has the signal.

This meant I had to use FCam, and since I was only passing a reference to the FCam::Image, it made no difference how big the actual image was or what format it was. In fact it would have wasted more processing time, converting the FCam::Image into an 8-bit format to then be signalled. So instead I used the RGB32 format, a format that was 4 sizes larger than the 8-bit unsigned format I was using with CImg.

As of yet I have not found a way to display a movement image without the color problem, but this is not something that is essential to the backend functionality of the app, and so it is something that will be rectified as a hotfix in future releases.

6.2 Email Thread

A major selling point of the application is the feature to email the user in real time upon movement detection. This ensures that the user does not have to be within the same vicinity as the phone to be notified that movement has occured, and is therefore a useful tool in security and privacy concerns.

Before starting the camera the user can choose to set an email address, message, subject, and can also choose to attach an image to the email, so that they can catch the latest image with the movement occuring within it. As a burglar alarm, this would be the method of catching the culprit red-handed so to speak.

In order to do this a framework needed to be used to send the email. Qt provided an example in version 3.x of a class that could send emails to a specified SMTP server using QTCPSocket objects[18], but this was dropped in recent versions of Qt and despite trawlng the forums very few users had managed to use this class file effectively in Qt4.7. So I was unable to use this framework to send an email. This was not neccesarily a bad thing, since the SMTP server would have to be defined by me and I would have had to have some kind of address book of every SMTP address in order to send an email to gmail or hotmail or whatnot.

Instead I looked at the phone itself for a framework. QtMobility looked promising, but several problems were reported for Maemo[19] which was confirmed by several users, and so I had to look at another alternative.

A veteran user on the Maemo community site had written an app called 'mailcmd '[20] that was a simple commandline utility that sent emails using the default email account setup on the phone. It even allowed for attaching images and the sheer simplicity of its usage was exactly what I needed.

This added the 'mailcmd' dependency to my project, but it is a small package and I don't think users would mind too much. I created a class file that inherited a QThread which ran a QProcess to execute the mailcmd command in a new thread.

This means that any time during the application, there will be at maximum three threads running: UI, Camera, and Email. Email needed to be run on it's own thread, otherwise the camera thread would hang between frames as it waits for confirmation of the email to be sent. I wanted the application to be as responsive as possible and so I created a seperate thread for Email to avoid this problem.

6.3 Logic and Optimisations/Error Proofing

The underlying logic of the application lies in the Operations.cpp camera thread. The thread is based on the relatively simple FCam API examples[21], where the Camera is first initialised by:

[18] QTCPSocket example - http://lists.trolltech.com/qt-interest/2006-10/thread01158-0.html

[19] QtMobility bugs - http://www.developer.nokia.com/Community/Discussion/showthread.php?201846-Sending-Email-Using-Qt-Mobility

[20] Nicolai's Mailcmd Announce page - http://talk.maemo.org/showthread.php?t=82918

[21] FCam API Examples - http://fcam.garage.maemo.org/examples.html

Snippet initialise() function in Operations.cpp

```
     errorCheck();  //Call error check function

    // Set shot parameters of Camera to 0.08 seconds and keep the brightness at ↩
        an even gain of 1
    stream1.exposure = 80000;
    stream1.gain = 1.0f;

    /* Enable the histogram unit -- NECESSARY to autoexpose image!
         Histogram monitors the entire image for changes to intensity */
    stream1.histogram.enabled = true;
    stream1.histogram.region = FCam::Rect(0, 0, width, height);

    //  Request an image of specified size. UYVY Format is used for streaming/↩
        Video
    stream1.image = FCam::Image(width, height, FCam::UYVY);
```

The small error checking method referenced above is replicated from the FCam API online examples, and it simply performs a check that the FCam driver is running, and that other applications are not using it concurently (please see page **??** in the Appendix.

The rest of the code and logic is entirely mine and is derived from the following pseudocode: .

Source Code: MovementLogic1.pseudo

```
reference_image :=0
norm = 100;

initialise()
updateReferenceImage()
checkForMovement(interval, threshold, timeLimit)
if(convertToMovie) convertToMovie()

function updateReferenceImage(){
    #See accumulator.pseudo
}

function checkMovement(interval, threshold, timeLimit){
    current_image :=0
    do{
        image = getImage()
        current_image = convertToCImg(&image)/norm  #{0,25}

        sub = (current_image - reference_image).normalise() #{0,1}
        sub = sub.erode(mask).dilate(mask)

        total = countWhitePixels(sub)
        if(total > threshold){
            print "Movement!"

            recordToFile(10 frames, fast_as_possible)
            updateReferenceImage()

            if(email_user)  emailUser()
        }
        thread::sleep(interval)
    } while( currentTime() < timeLimit)
}
```

The camera is initialised, and an initial reference is taken using the accumulator method defined earlier. Then we check for movement by getting a fresh image from FCam, and converting it to a CImg (passing a reference to the image and not the entire image to save memory). The result is then 'normalised' by dividing the pixel values by 100 (norm = 100) which limits it to a colorspace of 25 (256/100). This is because I found that normalising it to binary colorspace ($\{0,1\}$) resulted in erratic behaviour when I subtracted the current frame to the reference frame because the extremely discriminatory nature of binary colorspace grouped small changes in an image the same as large changes in the image. Normalising to larger colorspace first and then subtracting fixed this problem, and then normalising was permitted in order for opening and closing of the image to occur.

Once the white pixels were counted in the image it was compared to the threshold amount set initially. If it was above the threshold then movement was flagged and the camera then begins recording 10 frames in succession as fast as possible (which is the time it takes to perform the exposure). Once recording is finished, the reference frame is updated and an email is sent (if specified) with the latest attached movement image.

The thread then forces the camera to sleep for the interval amount. Initially the interval was set by calling 'FCam::Shot::framTime(interval)', but this was temperamental and the camera did not always respond to changes in the interval, and so the thread was used instead to set the interval.

This loops until the current time is greater than the previously set time for the application to run (default 1 minute). Once finished, the recorded images and compiled into a movie using mencoder (if specified).

6.3.1 Interval Scaling

On the train one morning it occured to me that if the app were to run for days on end (which it most certainly can) then there may be long periods of time where no movement occurs at all. During this time, the application would have dutifully taken frame after frame after frame at constant intervals, whilst wasting system resources and learning nothing about its environment To rectify this I immediately wrote a python script (see page 101 in the Appendix) as an outline of how I wanted the interval to react to its current situation - it should alter the interval to reflect the past level of movement. To give an idea:

1. A maximum interval is set. A minimum interval is set. A counter is set to zero.
2. The camera starts with an interval halfway between the maximum and minimum interval.
3. For every frame without movement, the counter is incremented.
4. If movement *has not* occured for 10 frames, then the interval is *doubled* and the counter reset. The camera has learned that nothing is likely to happen in the near future and so there is no need to check for movement so often.

5. If movement *has not* occured for 10 frames with the new interval, then it is *doubled* again the counter reset. This continues every 10 frames of no movement (each time the framerate decreasing with every counter reset) until it reaches the maximum interval and cannot be doubled anymore.

6. If movement *does* occur within a single frame, then the interval is immediately *halved* and the counter reset. The camera has learned that the scene has become more active, and now checks more frequently. It does not wait for a counter to reach 10 to react, but reacts immediately.

7. If movement *does* occur yet again, then the interval is immediately *halved* again and the counter reset. This conintiues until the interval reaches the minimum interval and cannot be halved anymore.

The code below gives a representation of this logic in a newer version of the checkMovement function: .

Source Code: MovementLogic2.pseudo

```
function checkMovement(int_min, int_max, int_modifier, threshold, timeLimit){
    current_image :=0
    consecutive_no_movement :=0
    current_interval = int_max/2   //interval can rise and fall

    do{
        image = getImage()
        current_image = convertToCImg(&img)/norm  #{0,25}

        sub = (current_image - reference_image).normalise() #{0,1}
        sub = sub.erode(mask).dilate(mask)

        total = countWhitePixels(sub)
        if(total > threshold){
            print "Movement!"
            consecutive_no_movement = 0;  //reset

            recordToFile(10 frames, recordLogic())
            updateReferenceImage()

            if(email_user)  emailUser()
        }
        else{
            consecutive_no_movement ++
            if(consecutive_no_movement > 10){
                // Increase interval due to inactivity
                modified_int = current_interval/int_modifier
                current_interval = modified_int //If greater than int_min
                consecutive_no_movement = 0;
            }
        }

        thread::sleep(interval)
    } while( currentTime() < timeLimit)
}
```

It should be noted that the halving and the doubling have been replaced by an interval *modifier* which is a number between 0.01 and 1. The modifier was determined by trial and error to have a default value of 0.16. As the modifier tends towards 1, the camera reacts unenthusiastically to changes in the scene (e.g. for mod=0.9, it will jump from an interval of 10 seconds to 9 seconds). When the modifier equals 1 the minimum interval is ignored and interval scaling is disabled, taking frames at a constant interval equal to the maximum interval. When the modifier is low, it reacts very eagerly to changes in the scene and for a mod=0.16 it will jump from a 10 seconds interval to 1.6 second interval and then back to 10 seconds (should no more movement occur). A modifier value of 0 is completely prohibited and I made sure the GUI would not allow it.

Another thing to note is the recordLogic() function. I noticed at one point that for short intervals (1 second) the application would hang on the record function as it tried to record 10 frames as fast as possible. To prevent this from occuring I scaled the record function so that depending on the current interval value, the number of frames it would record and the interval at which it would record them would be adjusted accordingly to this following method:

Snippet from checkMovement function in Operations.cpp

```
//7. Record logic:
            float scale = ((float)(current\_interval)/(float)(max\_interval));

            int record\_interval = (int)(scale*400)+100;  //max=500, min = 100
            int record\_frames = (int)(10- (scale*8));       //longest stint=10,↩
                 shortest = 2;
```

So now the interval would be fast for large number of frames (500 milliseconds for 10 frames), or slow for short number of frames (100 milliseconds for 2 frames) with scaling in between,

6.3.2 AutoExpose

I noticed sometimes that when I left the motion detector to face the garden at night the application would crash before the next morning with the last saved images being extremely white. This was because when I initially started the app, the target scene was dark, and so the histogram initially increased the exposure and gain to accomodate for this. But the following morning, when the target scene was now brighter, the exposure and gain weren't adjusted accordingly and so when it updated the reference image, the reference image was a completely white image and so when it performed subtraction, the white pixel count was always above the threshold making the app think that movement was continuously occuring! Once I realised my mistake I simply made the motion detector call the autoExpose algorithm whenever it updated the reference image and the problem was solved.

6.3.3 Lens Check

One of the easiest errors to make when using the app is to forget to open the lens cap before starting it. By looking through the FCamera source code mentioned previously I found a reference to file in /sys/devices/ that echoed the current state of the lens position:

```
bool lensClosed() \{
    FILE * ff = fopen("/sys/devices/platform/gpio-switch/cam_shutter/state", "r"↩
        );
    char state = fgetc(ff);
    fclose(ff);
    return state == 'c'; //o = open, c = close
\}
```

6.3.4 Kernel Compatibility

Early on in the development of the app, I came across problems where the FCam driver would not initialize properly, and no warning messages were given. Upon reboot it sometimes worked, but it would tend to work the most when I used the default phone kernel PR-1.3. For this reason I enabled a kernel check at the start of the Camera thread which would issue out a warning whenever a kernel different from the PR-1.3 kernel was being used. This was a simple shell call to 'uname -r'

```
%//kernel-powerv51 doesnt work. v50 is supposed to compatible.
bool validKernel() \{
    QString command = "uname -r";
    command = terminalAction(command, true);
    return command.contains("omap1") \textpipe\textpipe !command.contains("51");
\}
```

However the root of the problem turned out to be that I was not initializing the camera properly when I started the thread, and so when I fixed this problem, the validKernel() function became redundant and I deleted it from the source.

Part IV

TimeLapse

Time-lapse photography is the process by which a camera is set up to watch a particular object for extended periods of time. Instead of the camera taking frames continuously for that period, intervals are specified which are much lower than the normal framerate (e.g. 1 frame per hour). Once the frames are accumulated and played at a normal frame rate (e.g 30 frame per second) then time appears to run fast and lapsing occurs. The technique has been used to capture crowds of people, traffic, and other everyday entities which at a normal speed seem unimpressionable, but at high speeds become a fast flurry of activity.

There were three approaches I could take to make a TimeLapse application:

1. Run the app continuously for the full duration of the time lapse period specified.
2. Run the app continuously in the background for the full duration of the time lapse period.
3. Schedule the app to run and exit at specified times until the full duration of the time lapse period

The first approach would have meant that the application would have been in the main foreground while all the frames were being taken. If I wanted to capture a frame every hour for two days, then the app would have to be open and active for that entire time. This would be a waste of RAM, since the application will occupy a chunk of memory for two days without actually doing any work for most of that time. This will also occupy window space since the application UI will be active in this time, which is a major problem because the user may unwittingly close the application preventing it from completing it's job.

The second approach is the same as the first; waste of RAM over an extended period of time, but this time the window of the application is hidden and so the user will not be able to accidentally cancel the task once it is set. Doing this is very easy, since in Qt all one has to do is get rid of the myClass.show() function to run the app in the background. But as before, this is not optimal and wastes RAM.

This leaves the third option, which is to set scheduled events for the app so that it starts and stops at specific times with minimal usage of system resources. The question now is how to enable the MotionDetector to do this.

7 Command Line Switches

Commandline switches enable the user to issue commands to a program through a line of text. It is essentially a text-based user interface and the user can specify the level of control through arguments, which are extra text appended (with spaces) to the program name. By convention 'flags' or 'switches' are often used which specify arguments using dash or double-dash notation such as '–flag'.

7.1 Argument parser

Qt (like most SDk's) places all the arguments to a program in a list of some kind. In Qt, the arguments are accessed via QApplication::arguments() which returns the arguments in a QStringList (very similar to an ArrayList in Java).

My first step was to create a CommandLine class. This class would take a reference to the QStringList and check the tokens within that for certain key flag values. The key flag values needed would have to be same options that can be specified by the user, and so I settled with the standard convention of providing longhand and shorthand names for flags with '- -name' and '-n' respectively: .

Source Code: help.sh

```
usage: motiondetect [OPTION...]
  -h, --help     * Print this help
  -v, --version    * Display version
  -m, --mask INT    * Size of mask in square pixels
  -q, --quiet    * Silent mode. Messages recorded to log file
  -i, --images DIRECTORY  * Where images are saved. Default:~/MyDocs/DCIM/MISC/
  -c, --convert    * Convert all images in image directory  to an MPG file
  -d, --delete    * Delete images on exit. Convert flag must be specified too
  -s, --size WIDTH HEIGHT    * Valid sizes: 320 240, 640 480, 800 600, 1280 960
  -w, --whitepix INT    * White pixel count threshold. Default: 100
  -r, --range MIN MAX    * Intervals (secs) to restrict camera. Default: 1 10
  -a, --adapt FLOAT    * Adaptiveness of interval. Default: 0.16 (v.responsive)
  -t, --time secs OR mm:ss OR hh:mm:ss OR dd:hh:mm:ss
  -l, --log    * Output to log file: ~/.config/motion_detect.log
  -e, --email ADDRESS MESSAGE SUBJECT [Y/N]
    * Email upon movement. Y attaches the last image to email.
```

To search for flags, I used QStringLists' ".contains(key)" method which performs a search over all the tokens in the argument list. This can be very slow, especially if other less important flags are searched first. For this reason, flags such as '–help' or '–version' were the first two flags to be searched, since these will print a quick message and then terminate without having to check for further flags.

The other flags followed a different logic: .

Source Code: checkflag.cpp

```
void CommandLine::checkForFlagName(){
    int f_index = 0;
    flag_value = 5; //default

    if( ((f_index=args.indexOf("-f"))!=-1) || ((f_index=args.indexOf("--flag"))↩
        !=-1) )
    {
        flag_value = args.at(f_index+1).toInt(); //return 0 if not Int
        //Check for Errors
        if(flag_value == 0 || flag_value < someAmount){
            cerr << "Error: Bad argument" << endl;
            exit(1);
        }
        args.removeAt(f_index+1);  //Remove flag value from array
        args.removeAt(f_index);    //Remove flag name from array
    }
}
```

First a default value would be set (in this case 5) for the flag_value, then a search would commence within the if statement. The '.indexOf(key)' method would find the index of the flag in the array, or else return -1 if not found. I followed the convention where I would first check for the shorthand name first, and then the longer one since I reasoned that it was less effort to type a shortflag than a longer one, and was thus more likely to occur. This is useful, since C++ supports short-circuit evaluation and so if the first condition is deemed true, it will not need to search the other end of the OR statement.

The flag_value is then assigned the integer (or other type) version of index after the flag (e.g. We want to grab the 3 from the argument '–flag 3'). In Qt, this is returned as 0 if it could not convert it properly and so some error checking needs to be performed.

7.1.1 Switch Detection

Once the flag has had all the useful arguments taken from it, it is then *removed* from the argument array in order of highest index for flag, to lowest index for flag (the flag name). This order is necesary because when you remove from an Array at a certain index, all the indexes above it are shifted down meaning you now have to subtract by 1 to access a certain item from the higher indexes which can lead to errors. It is therefore far easier to remove from high index to low index to ensure that the lower indexes are unaffected by changes in the higher indexes.

Why is removal even important here? It's an optimization. By removing the arguments from the array after using them, future searches to that same array for different flag values will be *faster* because there are less items to check. This also gives another benefit in that if the user misspells or enters an unknown flag, it will be left over in the argument array , and so the program can notify the user precisely of which flags were not recognised and which one's were: .

Source Code: unknowns.cpp

```
void CommandLine::unknowns(){
    if(args.length()!=0){

          cerr << "Could not parse: "; //Start listing
          for(int i=0; i< args.length(); i++){
              cerr << args.at(i).toUtf8().data() << " ";
          }
          cerr << endl;                 //Stop listing

          cerr << "Please try --help for more info " << endl;
          exit(1);
    }
}
```

So if the user types into a shell:

' motiondetect -m 7 –size 320 240 –roger the dodger -w 100'

The app will spit out the following error:

'Could not parse: –roger the dodger

Please try –help for more info '

since these will be the leftover arguments that were left in the argument list. Had all the flags been valid, the argument list would have been empty.

Some flags rely on other flags such as '–delete' and '–convert'. Delete specifies that the images should be deleted after all operations complete, and Convert specifies that the images should be converted to a movie after all operations complete. If the user wishes to convert the images after completion, then they are free to choose to delete or not to delete. But if they wish to simply delete without first converting, then an error pops up highlighting how pointless the operation will be - since it will be taking photos and then deleting them after.

Terminal Colours

To make the commandline look as interesting as possible I also implemented colours into the shell. Fremantle uses the busybox shell by default, which is a lightweight shel that comes prepackaged with many standard utils such as 'ls' 'mv' 'cp' and so forth which are symlinked to /usr/bin/ so as to appear independent of busybox. Busybox also shares the same as terminal colour codes as Bash, which are called via non-printing escape sequences which are enclosed in \[\033[and \] pairs. Using the following colour codes in table 3, different colours could be printed

Color codes	
char red[ ]	"\033[0;31m\033[1m"
char cyan[ ]	"\033[0;36m"
char yellow[ ]	"\033[0;33m"
char green[ ]	"\033[0;32m"
char stop[ ]	"\033[0m"

Table 3: Red is a longer sequence than the other colours because default red was not very visible, so I made it **bold** too by appending \033[1m" to it.

With these codes I could then control what colours I wanted to use by simply calling: cout yellow ⟨⟨ "Warning:" ⟨⟨ green ⟨⟨ "This shell supports" ⟨⟨ cyan ⟨⟨ "multiple"⟨⟨ red ⟨⟨ "colors!" ⟨⟨ stop ⟨⟨ endl;

The stop colour at the end is neccesary to reset the shell to its default color, otherwise the last color set will be maintained for the next message which may not be desirable.

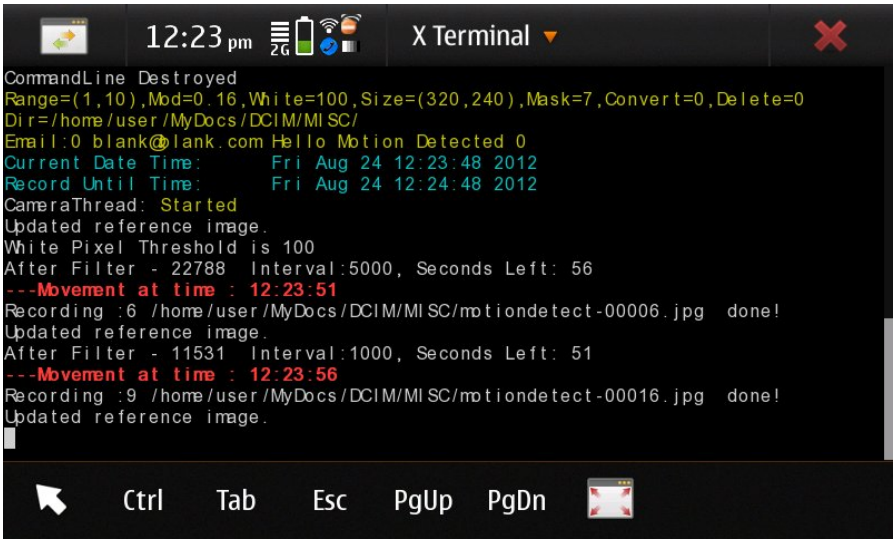


Figure 31: Screenshot fom phone

One of the main benefits of having a commandline implementation is that enables remote usage. A user can simply ssh or telnet into the phone from another machine and start and control the application from the shell on their own computer. The colours will also print correctly too.

It should be noted that at this stage std::cout replaced qDebug, the default Qt output stream, and message handlers are also attempted in order to enable verbose output for commandline. The implementation attempt for this is detailed on page 87 in the Other Implementations section of the Appendix.

7.1.2 Reusing existing variables

For standard motiondetection commandline calls such as mask, whitepixel, size, etc as shown in help.sh on page 57, individual variables were created to hold these values, which were mirror variables for the same ones in the CameraThread. When the Commandline finishes parsing arguments it switches flow control back to the motiondetector (main window UI) class which then reassigns these variables to those of the camerathread as shown below: .

Snippet from MotionDetector (mainwindow UI)

```
//op and commands are pointers to other class (the camera thread: Operations, ↩
    and the commandline class: CommandLine)
op = new Operations; //camera thread

if(commands == 0){            //Not assigned
        ..
        op->timelapse = false;
        op->echo_to_log = false;
        ..
}
else{   //Pointer not empty: perform commandLineOps

        //Assign values from commandline to CameraThread
```

```
        op->timelapse = commands->timelapse;
        op->limitVal = commands->white;
        op->interval_max = commands->max;
    ..
        //Terminate app when camera thread finishes
        connect(op,SIGNAL(finished()), this, SLOT(closeAndExit()));

        op->start();        //Initialise camera, and start!

        delete commands;  //destroys commandline and frees memory
}
```

The main MotionDetector class does not have variables in the header file that hold all the variables, like CommandLine and Operations (the camera thread) do. MotionDetector is simply a UI class, it acts as a waypoint or a go-between for connecting commandline arguments to the camerathread.

When commandline arguments are not given, the 'commands' pointer is initialised to zero and is thus empty. In this case the UI is active and so Operations gets its values directly from the UI. When the pointer is not empty, the UI is ignored and Operations gets its values directly from CommandLine. An exit slot is also connected in this case, so that when the camera thread finishes, the entire application safely closes (which overrides the default finish behaviour, where the UI simply becomes active and listens for input again).

The real trick is when we want to perform a timelapse operation. Timelapse operations need only five arguments:
Image width, frame rate, convert images to movie, delete images after conversion, and end date.
(height does not need to be specified since it can be inferred from width, reducing the number of neccesary arguments).
Image width, convert and delete are variables used in motiondetection operations, so these values can be assigned correctly. But what do we do with frame rate and end date? These variables do not exist for motion detection operations, and it would be wasteful to create new variables for them, because as discussed above; whatever variables we create in commandline we must also create in Operations (the camera thread). Creating a framerate and enddate variable will be making variables that will be unused most of the time.

Instead, we reuse the one's that exist already but are not being used. To pass framerate into operations, we simply pass it as a mask variable (they are both integers) and to pass the end date we pass it as a time variable that is normally used in the motiondetector as duration. This reduces the number of class variables and in turn reduces overhead.

This maybe frowned upon in certain circles due to the error prone nature of using a single variable for multiple different operations, but as long as it is done safely it is much more efficient. In this case, the camera thread needs to know whether the operation it is about to perform is a motion detection or a time lapsing one, and so a timelapse boolean had to be created. This may seem like a step backwards after what I just said about reusing variables, but it is neccesary because for a motion detection operation all

the other variables will be used up, plus this ensures that should I wish to create further options for the timelapse part of the app I can just slot them into the currently inactive motiondetector variables.

8 Scheduled Events

Scheduled events are events that have been schedules to run at specific times. They are often used to automate system daemons, but can also be acessed by the user to run scripts or programs at specific times.

8.1 Cron

In Linux 'cron' is the job scheduler, and it performs its operations by reading from a 'crontab' file which is a configuration file detailing the date and frequency, and job-specific detail of the job.

Cron jobs are prepended by an intuitive 5 character sequence which is used to define the date and frequency via:

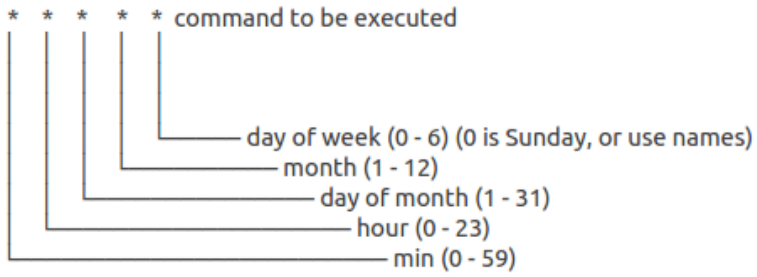


Figure 32: Cron example from Wikipedia

Quick Crash Course in Crontabs						
Run every minute for eternity:	*	*	*	*	*	
Run at 11am and 2pm every day:	0	11,14	*	*	*	
Run every mon,tue, wed,fri at 12 pm every 10 mins:	*/10	12	*	*	1-3,5	
Run at 8:14 and 8:15 every October 21st that is a Tuesday	14,15	8	21	*	2	

Table 4: Cron tab examples

So to run out motiondetection application every alternate winter morning at 8am, I would set a cronjob as:
' * 8 */2 12,1,2 * motiondetect –my-flag'

8.2 TimeLapse hidden command flag/switch

The importance of the commandline functionality mentioned in the previous section was to ensure that the app could be issued a command that would enable it to run and exit

by itself. For scheduled events this would mean using a specific flag to issue time lapse operations.

This flag is hidden from the user (i.e. it is not documented in the help text as shown on page 57. This is because it is the app itself that sets the scheduled event, and not the user. The user should not be able to set a scheduled event accidentally/maliciously from the commandline.

To ensure this I made the flag for running the timelapse operations an obscure mix of two words that are normally not likely to be next to each other in a given sentence: 'motiondetect –teenage-diplomacy ⟨DateTime⟩ ⟨Convert⟩ ⟨Delete⟩ ⟨Width⟩ ⟨fps⟩'. This is a reference to 'Blinded By the Light' song by Bruce Springsteen later covered by ELO. I figured no one would actually use that as a command call.

The code is as follows: .

Snippet from CommandLine.cpp

```
void CommandLine::secretDate(){  // --teenage-diplomacy <DateTime> <Convert> <Delete> <Width> <fps>
    int d_index=0; timelapse = false;
    if ( ( d_index=args.indexOf("--teenage-diplomacy"))!=-1){  //Two words unlikely to be typed next to each other
        //Date will be yyyy:mm:dd:HH:MM:SS
        QStringList date_time = args.at(d_index+1).split(":");

        //Booleans: Delete and Convert. 1= true, -1= false;
        int in_convert = args.at(d_index+2).toInt();
        int in_del = args.at(d_index+3).toInt();

        //Parse framerate
        mask = args.at(d_index+5).toInt();

        if( in_convert==0 || in_del==0 || date_time.length()!=6 || mask==0){
            cerr << red << "Nice try." << stop << endl;
            exit(1);
        }

        //Parse Bools:
        convert = (in_convert==1); del = (in_del==1);

        //Parse size:
        width = args.at(d_index+4).toInt();
        switch(width){
        case 320:  height=240; break;
        case 640:  height=480; break;
        case 800:  height=600; break;
        case 1280: height=960; break;
        default:
            cerr << red << "Yep. Something went wrong. Debug to find out." << stop << endl;
            exit(1);
        }

        //Parse Date:
        int year = date_time.at(0).toInt(), month = date_time.at(1).toInt(), day = date_time.at(2).toInt();
        int hour = date_time.at(3).toInt(), minute = date_time.at(4).toInt(), second = date_time.at(5).toInt();
```

```
        time = new QDateTime( QDate(year,month,day), QTime(hour,minute,second) )↩
            ;

        //Finish
        timelapse = true;
        args.removeAt(d_index+5); args.removeAt(d_index+4);
        args.removeAt(d_index+3); args.removeAt(d_index+2);
        args.removeAt(d_index+1); args.removeAt(d_index);
    }
}
```

As you can see the error/exit messages of the function call arent very informative either with an unhelpful "Nice Try" error message whenever the arguments aren't of the right length or format.

8.2.1 Logic of repeated calls

The snippet of code shown on page 64 shows how to call the timelapse operation. This function is called repeatedly for scheduled events, and it is called with the same arguments every time for each task set. This might seem somewhat counterintuitive to be using the same DateTime value for each job within a task but the logic is sound:

1. I set a DateTime value for 10th October 2012 3pm (the date I want the task to end.) I set an image size of 320x240.
2. I set that I want it to convert the images to a movie on completion, and that I want the the images deleted. I set a fps of 25.
3. I set an interval of 30 minutes for how often I want to set a job until it terminates at the DateTime value I set in the previous step.
4. The app sets jobs to the scheduler specifying that the system should run:
   ' motiondetector –teenage-diplomacy 10:10:2012:15:00:00 1 1 320 25'
   every half hour until 10th October 2012 3pm.
5. When the app executes using this flag, it compares the *current DateTime* with the end DateTime. If the current is less than the end, then it takes a photo using the size parameter and exits. Otherwise if the current date is *later* than the end date, it knows to convert (at 25 fps), delete, then exit.

All this, purely by using the same argument values.

To see the TimeLapse in operation, please visit the Youtube channel in the footer for a list of public videos demonstrating it in action[22]

9 Available Software Evaluation

In order to get repeated application calls in the first place, we need to first schedule the jobs. Fremantle uses cron (as do most systems), but it also has it's own native scheduler daemon 'Alarmd'.

[22]Timelapse videos - http://www.youtube.com/channel/UCbs6SUFR4UW6oREchDsRSpQ?feature=mhee

9.1 Cron vs Alarmd

9.1.1 Cron

Cron is native to all Linux-based OS's and to set a job, one has to edit /etc/cron.daily/(add item here). A job is added as a cron string with a command appended and it is the duty of the user to remove the string when that job is to terminate. To do this, the app must read all jobs from the file and filter out which ones belong to the application and which ones dont, then check to see if any of them require retiring, and if so removing them. This is doable, but very pinneckety.

Unfortunately this is seen as somewhat risky, and can lead to unpredictable behaviour, mostly because cron is only used on the device for system specific actions (such as /etc/cron.daily/locate which periodically updates the phone's tracker for media files), so any custom user-specific actions are somewhat temperamental. I therefore had to abandon cron and look for something else.

9.1.2 Alarmd

Alarmd is a Maemo-native system daemon that *manages* queued alarm events which "executes actions defined in events either automatically or after user input via system ui dialog service"[23]. At first glance it may seem like a front-end for cron, but by checking the dependencies it is seen that it talks directly to dbus. DBus is a low-level inter-system messaging/communication system for making system calls between applications, that it has been so well-engrained in Linux operating systems that it has only a single dependency.

Alarmd is commonly used by the system to manage making multiple wake up alarms, but it is not restricted to that function and technically can be used to execute any alarm at any given time. It has a much easier API then cron, and does a lot of useful scheduling background tasks by itself such as removing a job from the list once complete.

In alarmd, jobs are contained between three objects: Cookies, Events, and Actions.

Cookies ID for the job. Can be cast as a long, and cast back into a cookie_t object

Actions This describes what the alarm will do when it is called, and how it is called. It has two triggers (event listeners): Type triggers and When Triggers.

1. Type triggers determine what functionality the alarm has and common types are ALARM_ACTION_WHEN_TYPE_SNOOZE (snooze button that offputs the alarm for 10 more minutes) but for executing commands, the trigger I was using was ALARM_ACTION_TYPE_EXEC.
2. When triggers determine the context in which the alarm will be triggered. Common types are ALARM_ACTION_WHEN_RESPONDED but I was using ALARM_ACTION_WHEN_TRIGGERED.

[23] Alarmd Maemo Package - http://maemo.org/packages/view/alarmd/

Triggers have hex values, and an action is usually performed as a bitwise OR pair of Type and When triggers:
'act-⟩flags |= (ALARM_ACTION_WHEN_TRIGGERED |ALARM_ACTION_TYPE_EXEC)';
which performs bitwise OR upon these two triggers and sets the flag to "'true'" when one of these is satisfied.

Events These are containers for Actions, and are what Alarmd holds. To add an alarm to Alarmd, you add an 'Event'. Events also hold the time_t for the alarm, and has support for seconds, something that cron does not have.

To use Alarmd, one needs to use libalarmd, which is native to the phone and so the rootstrap device already had it installed. The class I created borrowed the template for a few of it's functions from the examples at the Maemo Wiki[24] . Alarmd jobs have APPID's, which uniquely match a job to an application making it far easier to retrieve and edit jobs without affecting other jobs or having to filter through them.

10 Optimisations/Error Proofing

The Commandline class is not called until arguments!=0, at which point it is then deleted after all input has been parsed.

Even with the decent examples at the Maemo wiki, the application was error prone: sometimes jobs would be added correctly and sometimes not, sometimes the jobs would execute correctly and sometimes not.

This was actually one of the most longest bugs to fix due to the temperamental nature of the errors. I eventually did find a fix by searching through the source code *Alarmed*[25], a python GUI for adding alarmed events to the scheduler using python bindings for libalarmd. The syntax was much the same and very soon I realised that the error turned out to be a problem with alarmd running commands from busybox, but should instead be running them from the Ash shell by prepending '/usr/bin/ash' to the executed strings. The app added and executed events flawlessly after that stage.

10.1 Kill Switch

Another problem that occured early on was accidentally setting the interval to every second. I did this to my mistake and was greeted by a "hello" prompt every second for at least 10 minutes while I simultaneously killed each process ID matching the alarm, and deleted each alarm individually from the app. That was a quick lesson I learned: Always have a kill switch.

First I created a button that deleted *all* alarmd jobs for the application, and then I created a function that killed all process ID's matching a timelapse operation from the shell. The call I made was:

[24] Alarmd Framework – http://wiki.maemo.org/Documentation/Maemo_5_Developer_Guide/Using_Generic_Platform_Compo
[25] Alarmed python package and source - http://maemo.org/packages/view/alarmed/

Kill Function

```
void killAllOccurencesOfApp(){
    QStringList pids = terminalAction("ps aux | grep teenage\\-dipl | awk '{↩
        print $1}'", true).split('\n');
    QString kill = "echo 'kill ";
    for (char i=0; i< pids.length(); i++)
    {
        kill.append(pids.at(i)+" ");
    }
    kill.append("' | root");
    std::cout << "Command: " << kill.toUtf8().data() << std::endl;

    terminalAction(kill);
```

The function prints a list of all active processes and greps the lines where 'teenage-diplomat' match, then pipes the output into awk which grabs the first column (containing the PID's) and splits them into an array. Then a kill command is issued by appending the contents of the array into a string and run as root.

The kill switch is a hidden function; it is not to be used lightly and so there is not one button that calls it. Instead it is activated by a sequence of button pressed in succession each other. This is to enable that it is not accidentally pressed by the user, but is only used in emergencies.

Setting the interval to less than 30 is not permitted either, but this is discussed later in the User Interfaces Section on page 76.

10.2 Easter Egg

Also included in the commandline is a small easter egg that launches a Asteroids/Snake game I wrote using the curses library in python. It was written in between journeys to UCL and home during the development and writeup, and it felt like a shame to leave it out out of the app. It was written on the phone itself, using the nano text editor.

To access it, the user must type 'motiondetect –easter'. The code can be found on page 102 in the Appendix.

Part V

IP Camera

An IP Camera is a video device that can send or recieve data through LAN or the internet. The camera is usually positioned in some remote location, and accessed by the user from another location usually for surveillance applications. Normally CCTV and other security monitoring software is employed for these types of tasks, but these technologies are restricted by the fact that they are merely hardware video devices performing the very specific task of streaming video from one place to another.

On the otherhand, smartphones are far more flexible, mainly because they offer multiple functionalities that CCTV doesn't. They are wireless devices with camera's and microphone's embedded, as well as speakers for producing warning noises if the user should so wish.

Smartphones are the ideal surveillance device, since they are portable, lightweight, and offer multiple uses not restricted to just video.

11 PhoneStreamer

Phonestreamer was the name of an app I made for Maemo in 2010[26]. It was essentially a front-end for a few gstreamer-scripts, with a wide variety of features namely:

1. Streams
   (a) Video and Audio
   (b) Front and Back camera, with supported image sizes of 320x240, and 640x480.
   (c) Uses protocols: HTTP, RTP, and UDP
2. Uses many diferent encoders and decoders (jpeg, smoke, h264, vp8) and has quality control to restrict bandwidth
3. Outputs to
   (a) Web browser,
   (b) VLC media player (creates a configuration file for recieving streams)
   (c) X Window (linux only)
   (d) Local file on the phone
   (e) File on a remote system (linux only)
   (f) Webcam (linux only). It mounts itself as the default video device on the remote system (/dev/video0) which can then be used as a portable webcam device in applications such as skype and google chat.

[26] PhoneStreamer - http://talk.maemo.org/showthread.php?t=70877

The app worked well, but it was my first attempt at writing in C++ and I had come from a scripting background of ActionScript 2, meaning a lot of cleaner object-oriented approaches were not known to me as well as not knowing when to delete an object. The entire app was written in one class, which though may be more efficient made it very hard to filter through 800 lines of code. It was unreadable.

It was a useful app however and so I believed it would be useful to bring it up to my current standard after learning how to properly program 2 years later.

11.1 Rewriting

An application that can perform all the functions mentioned above should not have been written in a single class, and yet it was. Worst of all it performed all of it's operations by directly calling shell commands as described by the example gstreamer script on page 15. On top of that, all the UI elements appeared within one class, since I had not known then how to switch between UIs.

A rewrite was neccesary and the first job was to seperate the application into seperate components. I split the UI into 8 different components:

ChooseOp - A mainwindow class that would enable the user to choose which operation the user would like to do (i.e. Ip stream, watchdog, timelapse). The idea was to incorporate the motion detector and timelapse into Phonestreamer, but they became apps of their own and so the idea was dropped.

About - A popup window which explained the application and its usage, as well as for donating

CameraView - A mainwindow class with would act as a viewfinder for the camera so that the user could see what they were streaming.

RemoteOps - A mainwindow class for quickly connecting to a presaved connection using presaved settings, or default ones.

ConnectionWind - A popup window which specifies the address and port to stream to.

ScanWind - A list popup window for listing active addresses on the network currently and updating ConnectionWind accordingly when an address is selected.

ConnectionSettings - A mainwindow class for specifying streaming options such as what format to stream to (VLC, HTTP, Webcam, etc), which camera to use, and what size.

WindowSCP - A popup window for generating and sending configuration files to the remote system in order to recieve an incoming stream

This design made the app all the more readable and much more maintainable for future development, as well as making it much easier for the user to quickly connect.

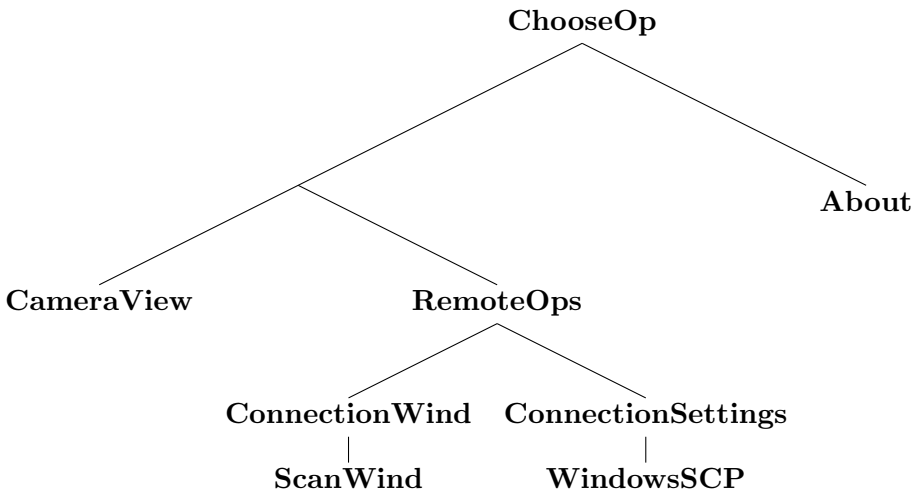


Figure 33: Tree diagram showing transitions from one window to the next

11.2 New features

Other than splitting the app into seperate classes, I also made a number of upgrades:

11.2.1 Automatic Connection Finder

Whenever I wanted to initiate a connection between my phone and my computer, I would always have to open a shell type 'ifconfig wlan0' to find the PC's ip address. This was too much work for a relatively simple connection and so I wanted to find the ip address of a machine on the same network as the phone, without the user having to manually go check it themselves.

Ping

Ping is a very simple and fundamental tool for checking the response times between two different machines. A typical ping syntax would be:
'ping [address]' which would print lines detailing how long it took to send a packet for each packet that it sent. One way to find all active connections on a network would then be:

Code for pinging all addresses on a network

```
#Grab all but end part of address
address=$(ifconfig usb0 | grep "inet addr" | awk '{split($2,a,":");split(a[2],b↩
    ,".");print b[1]"."b[2]"."b[3]}')
echo $address  # (e.g. 192.168.254) //last part split off
all=

for ii in $(seq 0 100); do
       add=$address.$ii                                  # e.g 192.168.254.0
       result=$(ping $add -i 0.2 -c 1)     #ping for 0.2 seconds, 1 package only
       result=$($result | grep Unreachable)     #Look for errors
       len=${#result}                                    #count length of result
```

```
        if [[ "$len" -ne 0]]; then
            all=$all\n$add                                  #if no errors, append to ↩
                list
        fi
done

echo $all               #print list
```

The shell code I wrote above works, and will find all the active connections on a network – eventually. It is very methodical, and will ping all connections from (for example) 192.168.254.1 to 192.168.254.100, assuming there are even 100 assigned addresses on the network. However the user will die of boredom before a list can be populated and so I had to search for another way.

Arp-Scan

Arp-Scan is a tool that "sends ARP packets to hosts on the local network and displays any responses that are received" [27]. It is scores faster than pinging methodically, and usually returns a list within 5 seconds of being called (depending on how large the network is. For WAN's this may take up to 10 minutes). A Maemo port of arp-scan exists [28] and so I could simply add it to my project as a dependency and harness it from the shell.

Due to reasonable delay between scanning the network and displaying the results, I created a small Thread class for it so that the UI would remain active while it scanned, and also so that a loading spinner would spin so as to ensure the user that something was happening in the background:

Snippet from MyThread.cpp for Scanning network and Retrieving data

```
void MyThread::run() {     data = getNetworkInfo(); }
QString MyThread::getNetworkInfo()
{
    QStringList args;
    args << "-c" << QString("echo %2arp-scan --interface=wlan0 --localnet | ↩
        egrep '[[:digit:]]{1,3}%1.[[:digit:]]{1,3}%1.[[:digit:]]{1,3}%1.'%2 | ↩
        root").arg(QChar(0x5c),QChar(0x22));
    QProcess *proc = new QProcess(this);
    proc->start("/bin/sh", args);
    proc->waitForFinished();
    return proc->readAll();
}
```

Snippet from ScanWind.cpp which calls MyThread

```
    //Spinning Loader Gif
    QMovie *wer = new QMovie(":/myImage/images/loader.gif");
    ui->label_scanwind_loader->setMovie(wer)->start();
    //Thread performs search and populates list on completion
    scanthread = new MyThread;  scanthread->start();
    QObject::connect(scanthread, SIGNAL(finished()), this, SLOT(populateList()))
```

After the list is populated, the user then selects the address from a dropdown menu and then the window closes, updating the address textbox appropriately, ensuring that

[27] Arp-Scan man page - http://linux.die.net/man/1/arp-scan

[28] Arp-Scan for Maemo, Colin Stephane - http://talk.maemo.org/showthread.php?t=73467

the user will never have to manually type in an address if they dont want to. Screenshots of this in action is shown on page 76

11.2.2 MySQL

Previously versions of PhoneStreamer would remember the last set settings and then simply reload them when the app started up upon next use (see page 79 for a detailed summary of how to persist data in Qt). This was useful when the user wanted to connect to the same machine twice, using the same settings without having to retype the address,port, video type and other parameters all over again. However it was limited to having a single-level history, so that if a user connected to machine A and then connected to machine B, only machine B would be remembered and machine A would be overwritten, meaning that the user would have to retype everything for machine A (which in turn would overwrite the details for machine B).

A user on the official announcement page of the application requested a feature where the application would store different profiles for work and home[29]. To overcome this I decided to implement a MySQL database, so that the user could switch between different configurations easily. I have used MySQL before: once for Android application, and many times for web development so I was well familiar with the declarative syntax and porting over between projects was relatively simple.

In order to enable SQL functionality, a 'QT +=sql' must be added to .pro file of the application in order for qmake to compile correctly.

The database was relatively small and simple and didn't require much reduction (to third normal form) to optimize it. Table 5 depicts the structure of the table reduced to it's third normal form so as to avoid duplicates and functional dependencies, but mostly to minimise the size of the database.

Table Name	Fields
Names:	nameID *, Name+
Connections:	conID*, nameID , details

Table 5: Database Structure for Profiles. * = key field, += unique field, and = foreign key. In Names, the Name field is unique purely to prevent the user from having two profiles called 'Work' and 'Work', because it would hard to know which 'Work' profile held the relevant settings. The 'nameID' field in Connections allows for duplicates, since different users can have multiple connection settings.

In the RemoteOps window, the user would click on the Profiles button which would launch a dropdown menu with two buttons at the bottom (similar to ScanWind popup). The menu would list all the current 'Names' (with a default name of "Home") by calling the MySQL command:

"SELECT 'Name' FROM 'Names'"

and then splitting the results into an array of Names that will be placed as Items in

[29] Angry Yoda Master (AMY) - http://talk.maemo.org/showthread.php?p=1054381&highlight=profiles#post1054381

the dropdown menu. Upon selecting a name the dropdown list would then update itself with all the connections for that particular name via the MySQL command:

"SELECT ‘details‘ FROM ‘Connections‘ , ‘Names‘ WHERE ‘nameID‘ = ‘Names‘.‘nameID‘ AND ‘Names‘.‘Name‘ = [the name picked by the user]"

The user would then select the relevant connection and these would update the ConnectionSettings UI and ConnectionWind UI, and then return the user back to the RemoteOps window where they could then quickly connect without having to manually enter in anything.

The two buttons at the bottom right of the dropdown menu would float above the list, and would be called 'New...' and 'Delete...' respectively. When the user clicked 'New...', they would be met with a window that would take in the name of their profile, and the details of the connection (a tickbox would exist that would ask the user whether to save the current settings. If ticked, the input textbox for details would become inactive). For 'Delete...' a list of current connections would be displayed and the user would select the one they wated to delete.

The Profiles.cpp code on page **??** in the appendix shows the easy C++ semantics of inserting and deleting connections sand users alike.

Unfortunately this was as far as I got with this MySQL implementation, since I didn't have time to build the controller that would feed data from the UI to the backend Profiles.cpp class. It is currently put on a TODO list until after this writeup finishes.

11.2.3 Webkit

Like the Android SDK, Qt also has it's own webkit for displaying a browser a within an application and enabling Javascript and displaying HTML functionality. This may not seem very useful for an application that just needs to stream video, but early on in development I wanted the user to easily select connections through an intuitive graphical interface. The idea I had was to use Google Maps Javascript API to plot the coordinates of each IP address (not LAN) on a map with a marker. The user would then browse the map to their work or to their home through the webbrowser within the app, and then select a connection by clicking on the marker.

Javascript would then read the details of the marker and display it HTML title, which in turn would have it's contents read by the QWebView object it is contained within by simply calling "QWebView::title()". MySQL would then perform a search of the database matching those details:

SELECT 'Name', 'details' FROM ‘Names‘,‘Connections‘
WHERE ‘Names‘.‘nameID‘ = ‘Connections‘.‘nameID‘
AND ‘Connections‘.‘details‘ = [Details selected from map]"

This may seem a little farfetched, but towards the beginning I already had a working prototype, the PHP code of which is shown on page 105 of the appendix.

The PHP code would retrieve from a database, and plot the stored coordinates onto the map, while displaying the details in a HTML table to the right of the map. At the

time, I had not got the MySQL component part of the application working, and so I was grabbing IP addresses from an SMF messaging board, by embedding a small SWF flash object in the signature of posts:

Snippet from counter.fla

```
var rex:RegExp = /[\s\r\n]*/gim; // Regex STRIP WHITESPACE

function onDataLoad(evt:Event)\{
    myLoader.removeEventListener(Event.COMPLETE, onDataLoad);
    //Get IP Address of connecting browser
    var ipAddress:String = evt.target.data.cant.split('\n')[0];
    ipAddress = ipAddress.replace(rex,'');
    //Create Url for PHP GET method
    var urlSend:String = "http://tetricity.net78.net/stats/insert.php?topical="+↩
        ipAddress+"\&submit=Submit";
    //Submit to DB
    var textReq:URLRequest = new URLRequest(urlSend);
    textLoader.load(textReq);
    textLoader.addEventListener(Event.COMPLETE, textLoadComplete);
\}
```

This object would grab the IP address of any user visiting any page that my signature appeared on and insert into a database along with the global coordinates of the IPaddress which was found by querying 'http://whatismyipaddress.com/ip/[insert_ip_address]:'

Snippet from insert.php

```
//////Part 2 - Insert new IPAddress and User
\$ip = \$\_GET[topical];
\$url = 'http://whatismyipaddress.com/ip/' . \$ip;
\$content = file\_get\_contents(\$url);

//Regex: Country, State, City
preg\_match('/\>Country.*img/', \$content, \$match);
\$country = split(':',\$match[0]);
..
..

\$latlng = \$latlng[0];
\$country = strip\_tags(trim(\$country[1]));
\$state = strip\_tags(\$state[1]);
\$city = strip\_tags(\$city[1]);

///Insert values into DB:
..
..
if(strlen(\$country)>2)\{
     \$squire="REPLACE INTO `flash_users` (`ID` ,`Address` ,`Country` ,`State` ↩
         ,`City`, `LatLng`) VALUES (NULL , '\$ip', '\$country', '\$state', ↩
         '\$city', '\$latlng')";
     //Inserts
     if(!\$res=mysql\_query(\$squire,\$con)) die('<h1>Details not inserted.</h1>↩
         ');
\}
```

The webpage displaying this prototype has since been taken down (much to my anger) and I was unable to take a screenshot, but the code is given in the appendix and

if tested on a server will work.

However, the accuracy of coordinates was not very good (often a 20 mile radius around an IP address was necessary) and people connecting from central London all seemed to connect from one central hub, which overlapped markers and made it hard to pick out different connections.

With much reluctance, I had to drop this web-side component too.

11.3 Afterthoughts

In the very near future I hope to merge Phonestreamer with the rest of the Watchdog application to make a fully-fledged image-processing application. With this in mind, my new version of Phonestreamer was called MediaStream and had buttons included in the main UI as empty slots for calling the two components of watchdog (the motion detector, and the time lapser).

Part VI

User Interfaces

12 GUI Map

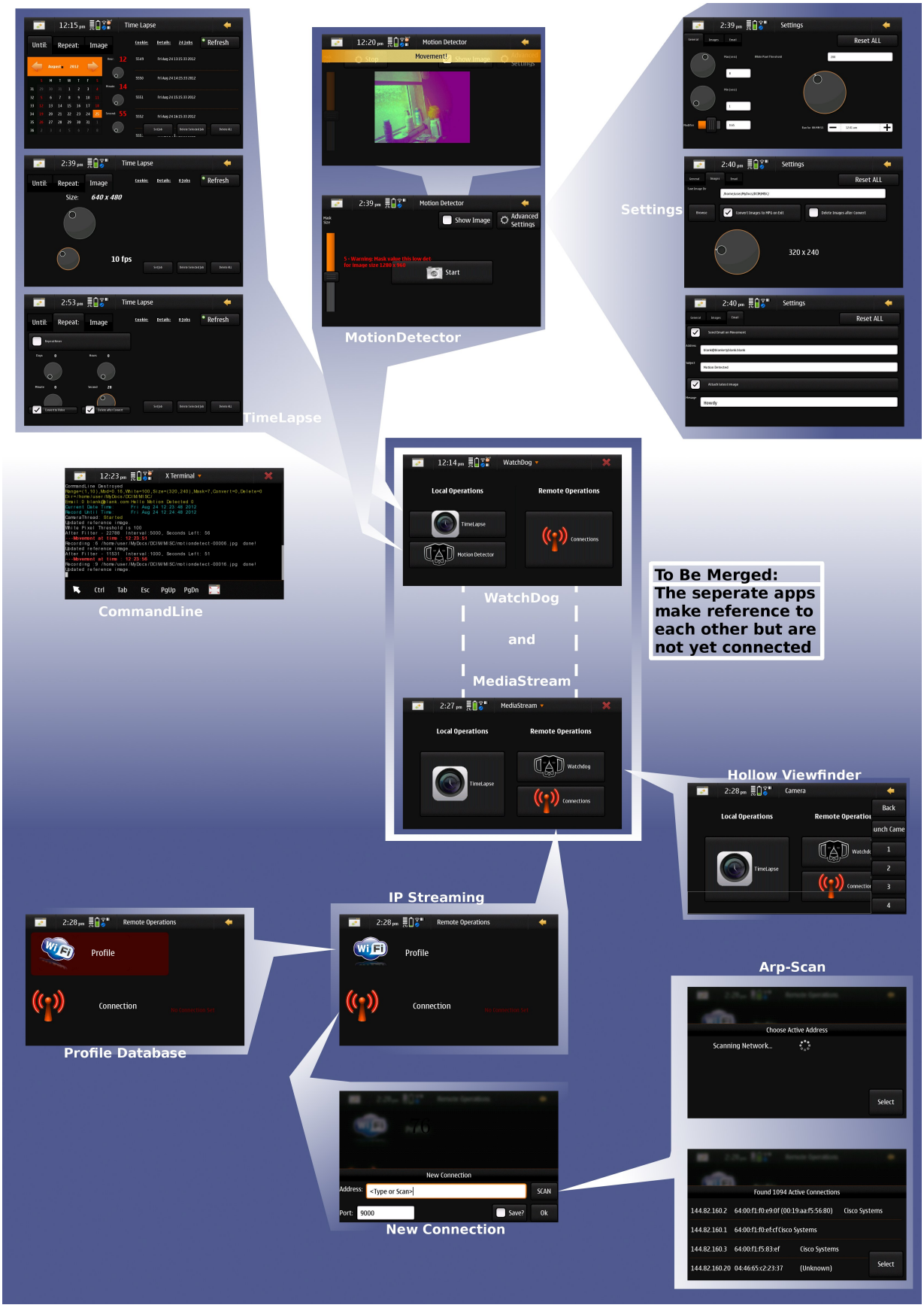


Figure 34: Map of every UI window for both WatchDog(motiondetector and timelapse) and Mediastream(IP streaming), and how they link to each other.

13 Widgets

Widgets in Qt can be very difficult to work with, and certain common operations such as applying styles, or inserting images, or inserting animated gifs, or even getting the UI to slide from window to the next without getting dangling pointers can be slightly difficult to implement.

To find out how I overcame these problems please consult page 88 in the Other Implementation section of the Appendix.

14 Window management and Maemo5 Stacked Windows

Unlike iPhone and Android user interfaces, Maemo does not automatically slide between different windows when initiated, but instead creates a new window and shifts view to it, whilst the previous window still exists.

This isn't much of a problem if the user is solely using the one app because the user will not notice it. However Maemo was one the first platforms to support multitasking windows for mobiles, and so if the user has multiple windows open at any one time, then switching between windows suddenly becomes tediousy difficult as there now many windows open at any one time.

One solution is to simply call QMainindow::hide() or QMainWindow::show() between different windows so that only one window is shown at any one time, but this is inelegant, and ultimately leads to confusion of how many windows are being hidden which can waste memory if those windows are not being used.

To overcome this I trawled through the forums and came across Maemo5StackedWindow. This is a window property that enables sliding transitions and works only on the Maemo5 device. To enable it one must place the following in the constructor in *every* window of the project:

```
#ifdef Q_WS_MAEMO_5
    this->setAttribute(Qt::WA_Maemo5StackedWindow);
    this->setWindowFlags(Qt::Window);
#endif
```

This solves the sliding window transition problem and saves window space when multitasking. However it does still waste memory, since the window(s) that were switched from are still active, but just hidden.

To overcome this problem I implemented pointer control

14.1 Pointer Control

In Qt, when creating a new Window (or any object), one can create it by simply calling 'new MyWindow(this)'. The this is an optional argument given to all windows which simply refers the child Window to the parent via a pointer. The child can perform very limited operations upon the parent window such as hide() or show() mentioned previously, but usually it is not used very much.

But how can the parent control the child? By pointers. Instead of creating a new window and throwing into the memory, the parent creates a *pointer* to the child window so that it knows exactly where it is and what to do with it. This is shown by calling: 'MyWindow *point = new MyWindow(this)'.

So now the parent can delete the child at any time it wants by simply calling the delete command, forcing the child window to be destroyed. This may seem slightly evil, but it is neccesary. Consider the followin situation:

1. User opens app, MainWindow pops up
2. User wants to access Motion Detector, so they select it and the MotionDetector-Window slides into view
3. User wants to change a Setting within MotionDetector, so they press the Settings button and the SettingWindow slides into view. We now have 3 windows open and active (MainWindow, MotionDetectorWindow, and SettingsWindow).
4. User finishes setting Settings, and goes back to MotionDetector. An eager developer may choose to delete the SettingsWindow at this point since it is not being of any use, however a cautios developer may reason that the user may want to change another setting in the nearby future and so it would be easier to keep SettingsWindow in memory to delete and recreate it.
5. User finishes with MotionDetector and goes back to MainWindow. Again, it would be unwise to delete MotionDetectorWindow just in case the user goes back into it, and so it should remain in memory (along with SettingsWindow) to enable ease of access.
6. User wants to use TimeLapse function, so they now click on Timelapse button instead of MotionDetector button, and TimeLapse button slides into view. At *this* point there are now 4 windows open (MainWindow, MotionDetectorWindow, SettingsWindow, TimeLapseWindow) which is a *waste of memory* because the user is unlikely to use the MotionDetector while they are using TimeLapse. It is at this stage that a developer should *delete MotionDetectorWindow and SettingsWindow.* This will leave just two windows open (MainWindow and TimeLapseWindow) and save memory.

The key underlying logic here is that when the user chooses TimeLapse functionality, all MotionDetector windows and subwindows are deleted, and vice versa (when the user chooses MotionDetector functionality, all TimeLapse windows and subwindows are deleted).

This may seem complicated, but it is rather trivial. The command 'delete settings-Pointer' is embedded in the deconstructor for MotionDetectorWindow, so that when the motion detector is deleted, so is the Settings subwindow.

Likewise, when the TimeLapseWindow is closed, any of its subwindows are deleted by referencing the child pointers. The implementation is shown below:

Snippet from ChooseOp.cpp (MainWindow)

```
//constructor, create Null pointers to motionwindow, and timelapsewindow
chooseop::chooseop(QWidget *parent=0){
    ...
    ui->setupUi();
    //Set pointers to both classes as 0 initially.
    mw = 0; tlw = 0;
}
//destructor removes dangling pointers
chooseop::~chooseop(){
    //Remove dangling pointers
    if(0!=mw) delete mw;
    if(0!=tlw) delete tlw;
    delete ui;
}
void chooseop::on_button_choose_watchdog_clicked(){
    if(0!=tlw)  {delete tlw; tlw = 0;} //If tlw is active, delete it and set ↩
        pointer to null;
    if(0==mw){  //if not initialised, create new one and launch
        mw = new MotionWindow(this);
        mw->showMaximized();
    }
    else if(0!=mw){ //already initialised; no need to create a new one
        mw->showMaximized();
    }
}
void chooseop::on_button_choose_timelapse_clicked(){
    if(0!=mw) {delete mw; mw = 0;} //if mw is active, delete it and set pointer ↩
        to null;
    if(0==tlw){ //if not initialised, create new one and launch
        tlw = new TimeLapseWind(this);
        tlw->showMaximized();
    }
    else if(0!=tlw){ //already initalised
        tlw->showMaximized();
    }
}
```

15 Persisting data

How does a developer share data between two different classes? One method is to make one class a child of another and then let the parent access data from the child by calling public its get() methods. This is a secure method of ensuring that data is accessible when requested (i.e. the pointer to the child is not null), but it makes it hard to share data between classes which share no heirarchy (or are of the same 'level').

On top of this one may want to the application to remember the settings the user set for the various widgets in the UI.

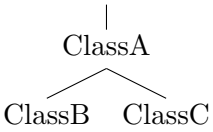
To get around this, we use QSettings.

15.1 QSettings

QSettings is Qt's implementation of providing persistent platform settings. QSettings writes a configuration file to /.config/appname/settingsname.ini which can preserve set-

tings between application usage, but also has the hidden benefit that it enables different classes sharing no parent-child relationship to share data between them.

This may seem a small overhead of continuously reading and writing from disk, but it is far easier implementation that the alternative, which is to create duplicate class variables so that for example; Class A with public variables vara,varb,varc, and Class B with public variables vara,varb,varc, and Class C with public variables vara,varb,varc. Say that ClassB has varb changed by the user, and Class C wants to access that setting. Now imagine the following implementation:



So that in order to get from Class B to Class C, one must go through Class A first. Then the only way to persist that setting so that ClassA has varb changed too is to assign:

```
ClassA *apoint = new ClassA;
ClassB *bpoint = new ClassB(apoint);
bpoint->varb = [changed by user];

//transfer setting from ClassB to C
apoint->varb = bpoint->varb;
delete bpoint;
ClassC *cpoint = new ClassC(apoint);
cpoint->varb = apoint->varb;
```

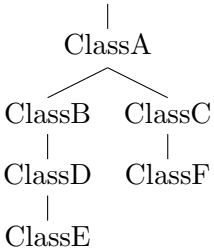
But this is clearly a waste of variables: ClassA is clearly a waypoint between ClassB and ClassC, and so does not utilize its own class variables, but merely uses them as a holder to pass variables between ClassB and ClassC.

One solution to this is to not delete bpoint until absolutely neccesary and directly assign ClassB to ClassC variables via:

```
.. //as before

//transfer setting from ClassB to C
ClassC *cpoint = new ClassC(apoint);
cpoint->varb = bpoint->varb;
delete bpoint;
```

But this is only a solution for classes that are connected via the tree diagram defined previously.
For Classes that have more than one relationship between them such as:



Then to persist data from ClassE to ClassF, one would have to go via E-D-B-A-C-

F, meaning that the user must duplicate class variables for all classes and then assign ClassD.varb = ClassE.varb, ClassB.varb = ClassD.varb, and so forth deleting pointers each time. But this is a waste of duplicate class variables as before, since only ClassE and ClassF should be the only classes with duplicate variables.

To try my previous solution of not deleting pointers to until absolutely neccesary, would require me assigning ClassF.varb = ClassE.varb directly, so long as I dont delete Classes E,D,B,A,C,F until the very end. This means that 6 windows would be taking up space in memory when only 2 are needed.

QSettings is the answer to this. The user simply commits settings to file in one class, and another class reads it at a later point without having duplicate variables or leave idle windows open. It's implementation is simple:

```
//Within ClassA
QSettings *setting = QSettings("Appname","Group");
setting.writeSetting("varb", this->varb);

//Within ClassB
QSettings *setting = QSettings("Appname","Group");
this>varb = setting.value("varb").toType();
```

It should be noted that QSettings wraps primitives with objects, and so the user must then convert them back to the relevant type when assigining them again.

Part VII
Release

16 Versioning

The project initially started with Mediastream taking over from Phonestreamer after a 2 year hiatus. Phonestreamer did not have version control and so I started a new repository for it on bitbucket. At that stage I was not familiar with branching since I was relateively new to the versioning process, and so I was making all my commits (stable/unstable) to the master branch.

Once development for the motion detector began I experimented more with Git, and created three new repositories called 'fcam_gstreamer', 'fcam_opencv' , and 'fcam_cimg' when I was first trying out the different API's. They were contained within different folders and I treated them as seperate projects. In hindsight it would have made more sense to create an 'fcam' repository, and then simply create three *branches* from the master and call them 'develop/fcam_gstreamer', 'develop/fcam_opencv', and 'develop_cimg'. The master would contain an initial setup of methods and which I would then work on independently and push any respective changes to. Once I had exhausted Gstreamer and OpenCV, I would then merge the 'develop_cimg' to the master and resume work from there.

As mentioned in the Background section, Qt is very temperamental depending on

which OS is used to develop on. For the longest time I found that I was unable to actually *build a deb package* for the application due to some meaningless error that refused to cite any useful information or even an exit code. I got around this by manually copying the build files to the device into the right locations.

One day I decided to create a new project called 'WatchDog' (the current name of the application) and imported my code from fcam_cimg to it. To my surprise (and relief) Qt built a deb package for it, and so I began developing in this new watchdog project, effectively resuming from fcam_cimg. Being relatively new to Git I did not know how to rename the fcam repository, keep all the previous changes, and then append the new changes I was making to watchdog ontop - so instead I created a new repository for watchdog and resumed development from there.

With watchdog I got braver with Git and began to try branching using the recommended master,develop,feature, hotfix standard model. I would push all my new changes to the develop branch, and when I wanted to try something completely new (i.e a new feature) I would create a new feature branch (e.g. feature/averaging, feature/subtraction) and then commit to that branch until I was reasonably sure that the method was stable, where I would then merge my changes with the develop branch. The hotfix branch was never used as I didn't release my application to the public Maemo repositories until much later so I had nothing to fix.

For a transcript of commits please see page 101 in the Appendix.

17 Publishing

To publish an app for the N900 there are two mediums: The Nokia Ovi store, and Maemo Extras.

17.1 Ovi

The Ovi store is similar to Android's Marketplace, where users can download apps,music, and movies from their browser. Apps that are published for Ovi do not require the source for publishing, and can be either free or paid. However in order to publish for Ovi, the deveoper must play a minimum fee of 1 euro, and Nokia gets to retain from 30% of the developers profits. This was something I wasn't too happy about, ethically nor financially. Besides, should someone want to use my code in the future to build something better then I would be more than proud to give them my source. As of recently the Ovi brand is to be discontinued and replaced with the Nokia brand.

17.2 Extras

The more common option amongst Maemo developers is to use the Extras repositories on the Maemo site. The extras repository is preconfigured on the N900 and the user simply has to enable it using the default package manager HAM (Hildon Application Manager) or other.

Like most repositories, it is split into two sections: free and non-free, with free applications being open-source and non-free being closed-source. However the user doesn't pay to use either.

To upload a project to Extras, one can either upload directly using a Maemo Garage account, or by using the Extras Assistant Wizard. The wizard is more commonly used, and it requires a .changes file and the compressed .tar.gz archive of the source.

An autobuilder then automatically builds the package (or fails with errors which the developer must address before trying to reupload), and then an autotester tests the app to see if it installs cleanly and without conflicts, and finally an autostager rates the app and stages it on the repositories to be downloaded by the end-user[30]. This process can take from 3 minutes to 3 days depending on how many new applications are being uploaded and how large/complex the application is.

Completley new applications are staged to the Extras-Devel repository, which is where all the new (and largely unstable) apps are first placed. Applications that are deemed stable by their developers can be pushed to Extras-testing repository by clicking a 'Promote Package' button on the information page for the application. Assuming all dependencies are met and a valid category for the app (multimedia/games/etc) is specified, then it should be promoted within the day[31].

Once in testing, it then undergoes a human rating system who thumb the application up or down depending on stable (or usually how good) it is. After a package recieves 10 positive votes and has had a minimum of 10 days within extras-testing, it is then promotes to the main Extras repository[32].

17.3 Announce Page

To promote the application, developers usually create an Announce thread on the Maemo Talk forums with a subject heading of '[Announce] Appname - Description'. Developers who are too meek to upload to Extras-Devel and don't quite want to reveal their source can attach their .deb packages to the first post where users can then download it and try it out without having to go through the repositories. All issues concerning the application and future versions and releases are updated/posted on this thread.

In the case of my old Phonestreamer app (now MediaStream) I got 10,000 downloads within a week, and 40,000 within a month[33], but that was two years ago when the N900 was still relatively new and cutting edge.

In the case of my current WatchDog app, the response has been less eager simply due to the lack N900 users still lingering around. The responses that I have recieved however have been overwhelmingly supportive and impressed and I have even dedicated this project write up to the first user who responded to my announce page (see dedication above the contents)[34]. The lack of users is not surprising, but I am not worried since the

[30]Repository process - http://wiki.maemo.org/Extras_repository_process_definition
[31]Promoting to Testing - http://wiki.maemo.org/Extras-devel#Promoting_packages_to_extras-testing
[32]Promoting to Extras - http://wiki.maemo.org/Extras-devel#Promoting_packages_to_extras-testing
[33]Phonestreamer Announce page - http://talk.maemo.org/showthread.php?t=70877
[34]'deed' appraisal http://talk.maemo.org/showpost.php?p=1255370&postcount=2

watchdog application is extremely portable (the underlying logic does not solely rely on FCam or CImg) and due to the cross-platform nature of the Qt framework, means that I can port this application to another phone in the very near future (though to my dismay it is most likely to be a Windows phone because of the Qt requirement). Watchdog is also a good 'end app' for the phone since many users simply discard or swap in their old phones when upgrading, but now watchdog gives it a use as security device.

Part VIII
Conclusions

I believe the app has been a success. Due to the dying state of the Maemo community and the lack of N900 users it perhaps hasn't received all the recognition that it deserves - but by itself it is a good decent application that performs a wide variety of different tasks, and does them *well*.

It satisfies all the requirements that a good motion detector needs such as detecting movement under a wide range of different scenarios and settings such as light/dark environments, or different types of movement by allowing the user to specify different mask size and thresholds for large/small movement.

It even goes beyond the initial requirements and extends its abilities, such that the user can opt to have their images converted into a movie after each session, or the user can choose to be emailed for each movement that occurs (with the option to have the image containing the movement emailed to them). This gives the application new function, so that it is no longer just a motion detector, but a full-blown security device that can notify and alert its owner of activities around the phone (such as emailing the mugshot of the guilty culprit attempting to steal something of the owner's).

The implementation of the commandline emphasizes this side of the application, since the user does not have to be within the vicinity of the phone to set up the watchdog, but can now ssh or telnet into the device remotely and perform operations without anyone knowing. This greatly extends its security applications by enabling the user to convert their phone into a spy device at will.

The time lapse operations give a sense of novelty back into the application so that the more artistic users can compile movies of objects moving over the course of days, and the native integration of the scheduling the photos using the phone's native scheduler makes it feel less of a burden, since it does not require installing any extra dependencies.

The IP streaming part of the application shifts back towards the practicality of application, much like the motion detector, but away from the security side of things and more for use as a video conferencing tool, so that the user can be moving about anywhere and streaming it live to a remote computer elsewhere.

On Maemo, there is nothing like this app out there on the repo's. However on other platforms, there are plenty of contendors who have more complete version of different *parts* of my app:

Windows mobile already has a motion detection app 'Vio'[35] which boasts features similar to mine (email user with image, convert to video), but can also send an SMS text. I didn't think to do that feature myself, mostly because I have a pay-as-you-go contract and so don't rely on network services (most of my communication is IP-based), but the implementation is easy to do especially since the QtMobility framework simplifies these tasks.

Vio lacks a commandline interface, and is closed-source. For me these are major contesting points, since a user should have access to their phone in every way possible, and they should also know that the security software they have just installed on their device is not itself a security risk (which is why I dont trust AVG or Avast AntiVirus software).

iPhone and Android have similar motion detectors (e.g 'Motion Detector' for iPhone[36], and 'Motion Detector Pro' for Android[37]), both of which have plenty of downloads but are not as fully developed as my app is in terms of features.

Once the IP streaming component is well integrated into the application, I believe it will become a must-have app due to sheer multitasked functionality that it offers. Future plans are to extend the commandline functionality to take in an input video file and detect the motion within it, but this may require a great deal of careful planning, since it would mean that I would have to completely seperate my motion detection logic from the FCam API of which it is (loosely) bound.

I've previously discussed porting the app to future phones, most likely Windows Mobile due to Qt being well-integrated into it, and this is the aim at the moment. Before I do that I need to secure a loyal fanbase who are familiar with my work, so I believe a required stepping stone for transitioning from Fremantle to Windows (without being called a traitor) is Harmattan . Due to Fremantle and Harmattan (Meego) using a similar framework, porting from one to the other will not be difficult and will help secure a loyal fanbase, since most of hardcore N900 Fremantle users switched to Harmattan N9 rather than choose the alternative (iPhone, Android, Windows).

In the meantime, the app serves as a good end product for a neglected (but otherwise fantastic) phone and may be a possible winner of the Maemo 2012 Coding Competition.

[35] Vio detection - http://vio-detection.com/

[36] Motion Detector - http://itunes.apple.com/gb/app/motion-detector/id331443079?mt=8

[37] Motion Detector Pro - http://motiondetector.homepagepublisher.com/

Part IX
Appendix

18 Further Background

18.0.1 Community

Maemo initially started off as being maintained by Nokia, with users recieving flashable firmware updates every 6 months or so, and over-the-air package updates bimonthly until it was stopped being maintained in 2011. At that point the community was responsible for updates and now the latest edition of Fremantle is known as Maemo CSSU (Community Supported Service Updates) which have transformed the device into a very fast and almost completely open-source device.

Further developements to the Kernel were made, and soon Kernel-Power (now version 51) is most widely used kernel on the device replacing Nokia's PR 1.3 due to it's overclocking support, USB hostmode capabilites, wifi packet-injection, and even support for 720p video (not a feature of the device, nor any device, at the time of its release in 2009).

Applications follow the same open-source framework as Debian applications, where any packages in the main repositories must be open source, and any third party closed-source applications must be installed through Nokia's Ovi store which has slow access through the phone's browser.

18.0.2 Future of Nokia

Nokia's previous OS was Symbian, which at the time was one of the most popular mobile operating system. Every one in two mobile phones sold in 2009 carried the Symbian OS[38] a stark contrast to the current leading smartphone OS's of today. Maemo was Nokia's next brain child, the open-source succesor to Symbian which it replaced in the release of it's then next-gen internet tablet phones, a bold and somewhat puzzling move by Nokia after the success of their initial Symbian OS. Maemo's open framework attracted many hardcore linux developers hungry to port linux applications onto the device, but sadly that was mostly all the phone was good for. The initial Diablo release for the Nokia N810 was superseded by the Fremantle release for the N900, which has now been surpassed by the Harmattan release for the N9 and N950.

Harmattan was supposed to be the stepping stone between Maemo and Meego, Nokia's next inventure, which was a merge between Moblin OS and Maemo. However though initial development images worked well with Nokia devices the Meego project was dropped by Nokia and the community had to once again pick up the slack, and are currently maintining Meego under the codename 'Mer' which is now being used as the template for the next completely open-source mobile OS 'Tizen' which will be launched on Samsung phones in the near future.

[38] Blookmber Business - http://www.businessweek.com/globalbiz/content/nov2009/gb2009114_367401.htm

One cannot blame Nokia for trying something new, but it is often said that the reason that sales for Nokia internet tablets did not do so well was because the CEO's were not comfortable fully backing the mostly unknown Maemo and Meego OSes. Had there been more advertizing and promotional support from Nokia's PR department, the mobile phone market may have been very different from the current state that is in now.

At the moment Nokia have backed behind the Windows Mobile 8 OS in a last bid attempt to re-establish themselves in the market to new customers with the Nokia Lumia smartphone series. This is essentially a complete reversal of the open-source ideals it had been pursuing not so many months before, leaving a lot of the original Nokia developers feeling somewhat betrayed.

19 Other Implementations

19.1 qDebug vs std::cout

qDebug is the default output stream in Qt. It has very basic usage almost similar to cout, but it automatically flushes and appends a linebreak to any output that is echoed to it.

Initially I was using qDebug to handle all my output messages, and every unlinked class would have the ⟨ QDebug⟩ module imported in order to use it. It was a slight overhead in terms of the extra module that had to be built with the app, but it was very easy to use and automatically converted all QObjects into their QString counterparts.

However when I developed the commandline functionality of the application, I realised that the messages were not being printed to the console which was a problem if a remote user who had ssh/telnetted into the device wanted to know what was happening.

In order to fix this I then switched from qDebug() to std::cout, which is the default output stream for C and C++ and prints directly to standard out. Using this option, the messages were printed correctly to the terminal, and terminal colours could also be used with them too.

Switching from qDebug() to std::cout was not much of a problem, since it mostly involved just replacing the former with the latter and removing ⟨ QDebug⟩ includes. FCam used methods that relied on std::strings and so to use std::cout I didn't need to import any extra modules since the standard ⟨stdlib.h⟩ ⟨stdio.h⟩ libraries were already being used. In most cases I dropped the std:: prefix by simply adopting the std namespace.

std::cout is slightly more harder to use than qDebug(), mostly because it couldn't handle QObjects very well and so I was constantly manually converting QObjects to their char * counterparts (e.g. QString::toUTF8().data()) which was a minor inconvenience at most. std::cout has the benefit that the developer can choose when to flush output to the console or when to end a line, which saved a lot of wasted lines.

Another useful feature of printing directly to the console is that I can use the \r operator to clear any current output on a given line and overwrite it. For place where I was outputting the current filename of the recorded file, this saved a lot of wasted lines from being printed as the filenames simply overwrote each other in the same line until

the recording process finished, and I then moved to a new line.

19.1.1 Message Handlers

Before I used cout, I attempted to use a qMessageHandler() that would automatically pipe qDebug() streams directly to std::cout. On the QtGlobal API[39], I found the following snippet of code:

```
#include <qapplication.h>
#include <stdio.h>
#include <stdlib.h>

void myMessageOutput(QtMsgType type, const char *msg)
{
    switch (type) {
    case QtDebugMsg:
        fprintf(stderr, "Debug: %s\n", msg);
        break;
    case QtWarningMsg:
        fprintf(stderr, "Warning: %s\n", msg);
        break;
    case QtCriticalMsg:
        fprintf(stderr, "Critical: %s\n", msg);
        break;
    case QtFatalMsg:
        fprintf(stderr, "Fatal: %s\n", msg);
        abort();
    }
}

int main(int argc, char **argv)
{
    qInstallMsgHandler(myMessageOutput);
    QApplication app(argc, argv);
    ...
    return app.exec();
}
```

The code identified the type of output message (Debug, Warning, Critical, Fatal) and piped it into the cout stream. Unforunately this was too much overhead, since there was a significant delay, and adding to the current overhead of using qDebug() in the first place was too much. I simply abondoned the approach and replaced qDebug() with std::cout.

20 UI related implementations

20.1 Labels and PixMaps

A QPixmap is essentially a container for a bitmap, optimized for displaying image. It is different from a QImage in that QImage can store and manipulate image data, but not display it.

[39] qInstallMessageHandler - http://doc.qt.nokia.com/4.7-snapshot/qtglobal.html#qInstallMsgHandler

To display a QPixmap one must assign it to a QLabel. In my MediaStream application I played around with QPixmaps alot creating my own custom QPushButton, so that when the user clicked on the button, the image icon rotated clockwise, and then rotated back when the user released the button.

To do this I needed to override the default event handlers for click, push, release actions. C++ is different from Java in overriding events in the way that in Java, and '@override' tag must be placed above the overriding function. In C++ one does not need to do such a thing.

20.1.1 Overriding events

There are two events I needed to override: push and release. I needed to intercept these events and grab the x,y coordinates of the 'mouse' pointer and check whether the coordinates match within the region of the button that the user is pressing:

```
//Overrides default mousePressEvent behaviour
void MainWindow::mousePressEvent(QMouseEvent * event)
\{
    if(ui->widget\_main\_Connection->geometry().contains(  event->x(),event->y()↩
        ))
    \{
        //shake img forward
        QPixmap qpm = spinPixMap(*ui->label\_main\_connection\_img->pixmap(), ↩
            20);
        ui->label\_main\_connection\_img->setPixmap(qpm);
    \}

    if(ui->widget\_main\_Profile->geometry().contains(  event->x(),event->y()))
    \{
        //shake img forward
        QPixmap qpm = spinPixMap(*ui->label\_main\_profile\_img->pixmap(), 20);
        ui->label\_main\_profile\_img->setPixmap(qpm);
    \}
\}

void MainWindow::mouseReleaseEvent(QMouseEvent * event)
\{
    if(ui->widget\_main\_Connection->geometry().contains(  event->x(),event->y()↩
        ))
    \{
        //shake img back
        QPixmap qpm = spinPixMap(*ui->label\_main\_connection\_img->pixmap(), ↩
            -20);
        ui->label\_main\_connection\_img->setPixmap(qpm);
        (new ConnectionWind)->exec();  //Launch Window
    \}

    if(ui->widget\_main\_Profile->geometry().contains(  event->x(),event->y()))
    \{
        //shake img back
        QPixmap qpm = spinPixMap(*ui->label\_main\_profile\_img->pixmap(), -20);
        ui->label\_main\_profile\_img->setPixmap(qpm);
    \}
\}
```

This performed two different operations when the user pressed and released a button (i.e. titled forward, then tilted back the QPixmap)

The actual tilting came by modifying and replacing the existing pixmap. This was done using QPainter which can draw graphics to a QPixmap object as well as perform operations upon it such as rotating and translating the object. The spinPixMap() function below displays this:

```
QPixmap MainWindow::spinPixMap(QPixmap original, int angle)
\{
    QPixmap rotatedpix(original.size());
    rotatedpix.fill(QColor::fromRgb(0,0,0,0));
    QPainter *ppp = new QPainter(\&rotatedpix);
    QSize size = original.size();
    ppp->translate(size.height()/2,size.height()/2);
    ppp->rotate(angle);
    ppp->translate(-size.height()/2,-size.height()/2);
    ppp->drawPixmap(0, 0, original);
    ppp->end();
    delete ppp;
    return rotatedpix;
\}
```

20.1.2 CSS stylesheets

Not only can the QPixmap change upon events, but different stylesheets can be set as well. Stylesheets are exactly the same in Qt as they are HTML, and it was through those that I created the red oval borders around the the Profile and Connection buttons (see page 76 bottom left for the Profile Database window to see both the rotating pixmap and the stylesheet in action.

20.2 Animated Gifs and QMovie

QPixmap by default can display a whole range of static images (jpg, png, bmp, gif, etc), but it cannot display animated gifs or other media that has more than one frame. To overcome this one must declare a QMovie object, initialise it with the animated gif (or wmv) file and then assign it to a QLabel. It can then be controlled via start() and stop() methods:

```
QMovie *movieLogo = new QMovie(":/myImage/images/logo.gif");
movieLogo->setScaledSize(QSize(250,250));
ui->myLabel->setMovie(movieLogo);
```

21 Manual

21.1 System Manual

21.1.1 Getting the source

The source code can be from the CD provided or can also be accesseed on the maemo packages database[40] and search for WatchDog. Then select any Extras-Devel repository for the latest code, and click on source. Untar the file with 'tar -zxvf watchdog_*.tar.gz'.

21.1.2 Building with Qt

First download the QtSDK for either Windows or Linux from the official Qt site[41]. Windows is recommended for stability reasons. A normal installation will take up 2.66GB of data, but a lot of this not needed so I recommend doing a custom installation. To install the neccesary components needed to run the app are:

1. Tick the core QtSDK (greater or equal to v1.2.1-1)
2. Untick experimental
3. Untick Documentation
4. Untick APIs (Qt Mobility, Qt Quick (QML), and Notifications are not needed)
5. In Development Tools tick:
   (a) Qt Designer
   (b) Qt Creator (untickable)
   (c) Qt Simulator (if you do not own Nokia N900)
   (d) Desktop Qt (if you do not own Nokia N900)
   (e) Maemo Toolchain
   (f) Device Files
6. Untick Miscellaneous (Qt Examples and Sources not needed)

This should limit the install to 925 MB and be much easier to install. Once complete, launch Qt and click 'Open Project...' and select the source directory either from the CD or the directory you had just previously untarred. Qt will then launch a prompt asking you for which target to build the app for: Tick Maemo5 if you have a Nokia N900 and wish to run the application on there, or else tick Qt Simulator for an emulator. Qt will then update the .pro file with the relevant target(s).

[40] Package Search - http://maemo.org/packages/
[41] Qt Downloads - http://qt.nokia.com/downloads

To build and run, simply press Build... then Run on the status bar, or Ctrl+B and Qt will run qmake to create the relevant moc files and then compile the app, finally packaging the application and sending it to the device (or emulator) to be launched.

If no target has been set up, then go to Tools... Options... select the Linux Devices tab and create a new target configuration. Click Add... and follow the wizard to set up a hardware device or an emulator. A development account will be set up on the device (with default username 'developer' which has root priveleges for installing/uninstalling) and SSH keys will need to be generated in order to let the device automatically send data between machines without prompting for a password every time.

22 API example usage

22.1 GStreamer Pipelining in C++

The following is just a small snippet from an example Camera class from the Maemo Documentation Wiki[42]

.

Source Code: gst_class.cpp

```
gst_init(argc, argv);  /* Initialize Gstreamer */
/* Create pipeline and attach a callback to it's
 * message bus */
pipeline = gst_pipeline_new("test-camera");

bus = gst_pipeline_get_bus(GST_PIPELINE(pipeline));
gst_bus_add_watch(bus, (GstBusFunc)bus_callback, appdata);
gst_object_unref(GST_OBJECT(bus));

/* Save pipeline to the AppData structure */
appdata->pipeline = pipeline;

/* Create elements */
/* Camera video stream comes from a Video4Linux driver */
camera_src = gst_element_factory_make(VIDEO_SRC, "camera_src");
/* Colorspace filter is needed to make sure that sinks understands
 * the stream coming from the camera */
csp_filter = gst_element_factory_make("ffmpegcolorspace", "csp_filter");
/* Tee that copies the stream to multiple outputs */
tee = gst_element_factory_make("tee", "tee");
/* Queue creates new thread for the stream */
screen_queue = gst_element_factory_make("queue", "screen_queue");
/* Sink that shows the image on screen. Xephyr doesn't support XVideo
 * extension, so it needs to use ximagesink, but the device uses
 * xvimagesink */
screen_sink = gst_element_factory_make(VIDEO_SINK, "screen_sink");
/* Creates separate thread for the stream from which the image
 * is captured */
image_queue = gst_element_factory_make("queue", "image_queue");
/* Filter to convert stream to use format that the gdkpixbuf library
 * can use */
image_filter = gst_element_factory_make("ffmpegcolorspace", "image_filter");
/* A dummy sink for the image stream. Goes to bitheaven */
image_sink = gst_element_factory_make("fakesink", "image_sink");
```

[42] Camera Example - http://wiki.maemo.org/Documentation/Maemo_5_Developer_Guide/Using_Multimedia_Components/C

```
/* Check that elements are correctly initialized */
gst_element_set_state(pipeline, GST_STATE_PLAYING);
```

23 Tables

Non-Zero Counts from Normalized Subtracted Images						
320x240						
Test	Scene 1	Scene 2	Scene 3	Scene 4	Scene 5	STDEV
1	24272	18049	19899	19746	18833	2416.9
2	23691	21275	19730	21381	20744	1455.8
3	20607	23001	19261	19711	24503	2249.9
4	19134	22938	23231	21930	21096	1648.7
5	22282	20143	19616	24927	22492	2119.1
6	23242	18886	19865	20627	21682	1680.1
7	23769	24513	23851	23883	20993	1378.9
8	21132	22961	23835	24969	23704	1418.5
9	22978	23870	21845	21534	18847	1901.6
10	22031	22578	18717	24348	20265	2167.2
Average	22313.8	21821.4	20985	22305.6	21315.9	592.73
640x480						
Test	Scene 1	Scene 2	Scene 3	Scene 4	Scene 5	STDEV
1	33866	31422	28292	29681	32563	2223.3
2	31297	30610	26705	28487	33717	2687.2
3	31533	28780	29280	31056	28660	1339.0
4	27640	33356	31471	27337	30149	2554.9
5	34987	30582	34525	27789	28794	3282.9
6	35157	30299	34499	29315	26250	3720.3
7	27937	29603	35060	30154	35903	3531.9
8	35198	27293	32847	30177	33796	3144.6
9	28123	34739	28183	30261	35580	3570.8
10	27383	35923	26547	32220	33636	4046.1
Average	31312.1	31260.7	30740.9	29647.7	31904.8	847.98
800x600						
Test	Scene 1	Scene 2	Scene 3	Scene 4	Scene 5	STDEV
1	40923	34485	32805	39334	36086	3362.4
2	49864	41539	35681	48360	32024	7756.8
3	42711	42388	44199	41864	34134	3967.3
4	42281	34134	45012	47533	49251	5931.5
5	45101	39482	49579	46799	47585	3835.0
6	47585	38467	40767	38339	47910	4786.5
7	33722	37084	34591	33932	34743	1339.6
8	36251	30004	39643	44400	49334	7425.6
9	48505	30668	30818	46060	42960	8500.3
10	34278	36288	33474	46195	48286	6994.8
Average	42122.1	36453.9	38656.9	43281.6	42231.3	2877.6

Non-Zero Counts from Normalized Subtracted Images						
1280x960						
Test	Scene 1	Scene 2	Scene 3	Scene 4	Scene 5	STDEV
1	53002	81944	40951	46610	65069	16336
2	83293	73420	86805	85784	86103	5560.7
3	75139	63225	75406	68526	65532	5539.1
4	65790	52952	41361	56894	75100	12804
5	41518	80293	44516	44907	71352	17952
6	81478	84669	81585	51753	74673	13409
7	46704	83295	54751	68210	64852	13915
8	48528	67619	84869	84948	64196	15370
9	73393	78231	57920	82735	45935	15319
10	74232	84807	48853	68323	58526	13895
Average	64307.7	75045.5	61701.7	65869	67133.8	5029.2
2592x1968						
Test	Scene 1	Scene 2	Scene 3	Scene 4	Scene 5	STDEV
1	49690	79785	78167	66792	66172	12065
2	72796	86125	51517	60103	66779	13092
3	60313	86716	83282	55395	52380	16164
4	41171	65568	86522	66336	75404	16743
5	41280	53591	74107	89517	66416	18566
6	83867	79419	53495	57972	78860	13910
7	71485	63420	69769	81518	88248	9896.3
8	58867	46546	76395	40210	51333	13927
9	73673	74847	89235	76566	89943	8045.0
10	71257	66062	79217	63713	78011	6930.0
Average	62439.9	70207.9	74170.6	65812.2	71354.6	4656.4

Table 6

24 Tests

Due to the 120 page filelimit, these had to be omitted from the report. The extensive testing however can be accessed via the following GoogleDocs link:

ShortUrl - `http://alturl.com/pbaxn`[43]

and by clicking on the first link. This will take you through twelve tests performed and analysed, but not typed up very well, but proving that extensive work was done for this part of the project.

[43]URL - `https://docs.google.com/document/d/1Q4bQBR2-eptudjRmMpVTXe3FIAUZg3uGiBaAiKVZEeE/edit`

25 Application Code

The most significant parts of the code have already been sampled and discussed at great length within the main text (more than 15 pages worth). However some of the code that has been unmentioned is the GUI-related *error proofing of user-inputs*. This is not the focus of the project, which is why I did not make it into a chapter in the main text, but a significant amount of work has been done to make the application(s) reasonably error-free and so the source is provided in the CD for those who are curious enough.

The code displayed below are the small classes that have been referenced in the main text, but not discussed much in great detail.

Please note that the autogenerated moc files from Qt have been omitted, as well as any UI XML files (the GUI Map in the User Interfaces section is suitable enough). Also note that the header files for the classes were also left out because the headers merely outlined the source file, which can speak for itself. Please consult the CD for full contents of all header and development files.

25.1 WatchDog

The code here is used by both parts of the application. The subsections denote the more specific classes. CImg.h was left out, not only because it is an external library, but because all 45,000 lines of it may extend the length of this report somewhat.

.

functions.cpp

```
#include <QProcess>
#include "CImg.h"
#include <FCam/FCam.h>
#include <iostream>
#include <string.h>


using namespace cimg_library;

char red_col[] = "\033[0;31m\033[1m";
char cyan_col[] = "\033[0;36m";
char yellow_col[] = "\033[0;33m";
char green_col[] = "\033[0;32m";
char stop_col[] = "\033[0m";

//Shared QProcess
QProcess *qp;

//convert address FCam to CImg
void convertImage(const FCam::Image &image,CImg<unsigned char> &img)
{
    unsigned char * imgBuffer = img.data();  //down cast 2bytes(uint) to 1byte (↩
        uchar)
    for (unsigned int y=0;y<image.height();y++){
        for (unsigned int x=0;x<image.width();x++){
            imgBuffer[0] = (*(image(x,y)+1));
            imgBuffer++;
        }
```

```
    }
}

QString terminalAction(QString command, bool getoutput=false)
{
    //TODO: Show output

    QString result = "";
    QStringList args;
    args << "-c" << command;
    qp = new QProcess;
    qp->start("/bin/sh", args);
    qp->waitForFinished();
    if(getoutput)
    {
        result = QString(qp->readAll());
    }
    qp->close();
    qp->terminate();
    return result;
}

//replace with phone function in main app
void alert(QString text, bool echo=true){
    if (echo) std::cout << red_col << text.toUtf8().data() << stop_col << std::↩
        endl;
    QString notif = QString("dbus-send --type=method_call --dest=org.freedesktop↩
        .Notifications /org/freedesktop/Notifications org.freedesktop.↩
        Notifications.SystemNoteInfoprint string:%1"+text+"%1").arg(QChar(0x22))↩
        ;
    terminalAction(notif);
}

void clearImages(QString &dir)
{
    QString clear = "rm "+dir+"*.jpg";
    terminalAction(clear);

    std::cout << "Images cleared from " << dir.toUtf8().data() << std::endl;
}


void convertToMP4(QString &dir, bool clear, QString videoname="movement", int ↩
    fps=25)
{
    alert("Converting to MP4 -- please wait");

    if(!dir.endsWith('/'))  dir.append('/');
    if(!videoname.endsWith(".mpg")) videoname.append(".mpg");
    bool timelapse = videoname.contains("timelapse");

    std::cout << "Converting " << dir.toUtf8().data() << "*.jpg to " << (dir+↩
        videoname).toUtf8().data() << std::endl;
    if(timelapse) std::cout << "Timelapse" << std::endl;

/*    //FFMPEG -- very temperamental with the framerate
    QString fileformat = (!timelapse)?"motiondetect-%05d.jpg":"timelapse-%06d.↩
        jpg";
    QString convert = "ffmpeg -i "+dir+fileformat+" -y "+dir+videoname+" -r "+↩
        QString::number(fps); //y forces overwrite, r must come at end to ↩
        specify OUTPUT framerate*/
```

```
    // Mencoder -- much more stable
    QString fileformat = (!timelapse)?"motiondetect-*.jpg":"timelapse-*.jpg";
    QString convert = "mencoder \"mf://"+dir+fileformat+"\" -mf fps="+QString::↩
        number(fps)+" -o "+dir+videoname+" -ovc lavc -lavcopts vcodec=msmpeg4v2:↩
        vbitrate=900";
    std::cout << convert.toUtf8().data() << std::endl;

    terminalAction(convert, true);

    std::cout << "Finished converting." << std::endl;

    if(clear) clearImages(dir);

    alert("Finished!", false);
}

//Taken from FCamera
bool lensClosed() {
    FILE * ff = fopen("/sys/devices/platform/gpio-switch/cam_shutter/state", "r"↩
        );
    char state = fgetc(ff);
    fclose(ff);
    return state == 'c'; //o = open, c = close
}

void echoLog(std::string text){
    FILE *ff = fopen("/home/user/.config/motion_detect.log", "a+");
    fprintf(ff, text.c_str());
    fclose(ff);
}

void killAllOccurencesOfApp(){
    QStringList pids = terminalAction("ps aux | grep teenage\\-dipl | awk '{↩
        print $1}'", true).split('\n');
    QString kill = "echo 'kill ";
    for (char i=0; i< pids.length(); i++)
    {
        kill.append(pids.at(i)+" ");
    }
    kill.append("' | root");
    std::cout << "Command: " << kill.toUtf8().data() << std::endl;

    terminalAction(kill);
}
```

25.1.1 Motion Detector

MotionWindow.cpp and Settings.cpp were omitted for page limiting reasons, they were simply too long to put in the Appendix. Most of their logic stems from Pointer control, QSettings implementations, and Error proofing user-inputs, all of which have been discussed in the main text. Operations.cpp has already been discussed in great detail too. Consult CD for full source code.

emailthread.cpp

```
#include "emailthread.h"
#include <QStringList>
#include <iostream>
```

```
EmailThread::EmailThread(const QString &address, const QString &subject, const ↩
    QString &message, bool attach, const QString &attach_name)
{
    QString email = "echo '"+message+"' | mailcmd "+
            ((attach)?("-a "+attach_name+" "):(" "))
            +"-s '"+subject+"' "+address;

    std::cout << "Emailed: " << email.toUtf8().data() << std::endl;
    QStringList args;
    args << "-c" << email;
    qp = new QProcess;
    qp->start("/bin/sh", args);
    qp->waitForFinished();
}

EmailThread::~EmailThread()
{
    qp->terminate();
    std::cout << "email sent.";
}

//TODO: Timestamp for image
```

25.1.2 Time Lapse

The code for TimeLapseWindow is also very long since the GUI has many elements (tabbed window), and so the code consist of mostly various error-proofing and QSetting related functions. Please consult CD for more information.

alarmd_ops.cpp

```
#include <alarmd/libalarm.h>
#include <iostream>
#include <QStringList>

#define APPID "timelapse"

QStringList cookie_info;
int num_jobs = 0;

// Very useful guide here:
//http://wiki.maemo.org/Documentation/Maemo_5_Developer_Guide/↩
    Using_Generic_Platform_Components/Alarm_Framework#↩
    Adding_Alarm_Event_to_Queue

//True - deleted, False - unable to.
bool deleteAlarmd(cookie_t cookie)
{
    return alarmd_event_del(cookie) != -1;
}

int fetchMultipleJobs(){
    cookie_t *list = 0;
    cookie_t cookie = 0;
    alarm_event_t *eve = 0;

    cookie_info.clear();
    int i=0;
```

```
    if( (list = alarmd_event_query(0,0, 0,0, APPID)) == 0 )
    {
        std::cout << "Could not fetch" << std::endl;
        goto cleanup;
    }
    for(i = 0; (cookie = list[i]) != 0; ++i )
    {
        alarm_event_delete(eve); //clear current details
        if( (eve = alarmd_event_get(cookie)) == 0 )
        {
            std::cout << "Details for cookie " << (long)cookie << " could not be↩
                acessed" << std::endl;
            continue;
        }
        //Add results to lists;
        cookie_info.append(QString::number( (long)(cookie) )+'\t'+  QString(↩
            ctime(&eve->trigger)));
    } num_jobs = i;

cleanup:
    free(list);
    alarm_event_delete(eve);
}

cookie_t addAlarmdJob(QString command, uint secs){
    cookie_t cookie = 0;
    alarm_event_t *eve = 0;
    alarm_action_t *act = 0;

    /* Create alarm event structure, set application identifier and dialog ↩
        message */
    eve = alarm_event_create();
    alarm_event_set_alarm_appid(eve, APPID);

    /* Use absolute time triggering, show dialog
       * ten seconds from now */
    eve->alarm_time = secs; //time(0) + secs;

    /* Add command*/
    act = alarm_event_add_actions(eve,1);
    act->flags |= (ALARM_ACTION_WHEN_TRIGGERED | ALARM_ACTION_TYPE_EXEC);

    QString action = "/bin/ash -c '"+command+"'";

//    std::cout << "ACTIONNY! " << action.toUtf8().data() << std::endl;
    alarm_action_set_exec_command(act, action.toUtf8().data());

    /* Send the alarm to alarmd */
    cookie = alarmd_event_add(eve);

    /* Free all dynamic memory associated with the*/
    //alarm_event_delete(eve); //<-- Invalid pointer exception
    return cookie;
}
```

25.2 MediaStream

25.3 IP Streamer

IP Streamer was omitted from the Appendix due to most of the developed code in the project being UI related, or otherwise developed as a continuation of PhoneStream. Please consult 'MediaStream' folder in CD for full source code.

25.4 Commit Log

Due to the page restrictions, this has also been omitted. Please see my public BitBucket account for a full list of commits: `https://bitbucket.org/tetris11/`

25.5 Other

Moved to the CD due to page limit restrictions

Interval Python Script

.

interval.py

```
#!/usr/bin/env python

import commands, curses

def drawer(timer, mod, queit_frames):
  f=0

  flip = True;  move = False
  consec_quiet = 0

  while f!=-1:
      move = False
      try:
         inp = screen.getch()
      except IOError:
         pass
      except TypeError:
         pass
      screen.nodelay(True);
      if inp == ord('a'):
         move = True;
      try:
      # -- Move logic, halves on movement.
         if move:
            if timer> 100:
               timer = int(timer*mod)
            consec_quiet = 0
         else:
            consec_quiet = consec_quiet+1

      # -- Timer doubled if quiet for (10) frames
         if consec_quiet > quiet_frames:
            timer = int(timer / mod);
```

```
      # -- Alternates on clearing and printing per interval
          if flip:
             screen.addstr(10,10,str(timer) )
             flip = False
          else:
            screen.addstr(10,10,"       " )
            flip = True
       except:
           pass
       curses.napms(timer)
       screen.refresh();

screen = curses.initscr();
curses.curs_set(0)
curses.noecho()
try:
  drawer(timer=1000, mod=0.7, quiet_frames=10)

except KeyboardInterrupt:
    curses.endwin()
    exit()
```

Easter Egg python game

.

easter.py

```
#!/usr/bin/env python

import commands, curses
from traceback import print_stack as trace
from math import radians, cos, sin

global rewtable;

def conswidth():
    rows,columns = commands.getoutput("stty size").split()
    global col; global row;global n;col = int(columns);row = int(rows)

def drawBombs():
   screen.border();
   screen.addstr(0,2,"a -left , s - down, d-right")

def drawer(timer, cycle):
  trails = []; rewtable = []; projectiles = []
  global currX;global currY;
  currX=currY=10;digg = tral ='.';x=y=f=g=dangle=vel=0;back=True;

  fire = False; turn = 25

  jj = ord('d')
  while f!=-1:
      fire = False
      vel = vel*0.985   # drag
      try:
         inp = screen.getch()
      except IOError:
         pass
      except TypeError:
         pass
```

```
        screen.nodelay(True);
        if inp == -1: #repeats last
           inp = jj
        elif inp == ord('d') or inp==ord('#'):
           dangle -=turn;
        elif inp == ord('w'):
           vel +=1; #turn +=2;
        elif inp == ord('a') or inp==ord('*'):
           dangle += turn;
        elif inp == ord('s'):
           vel -=1; #turn -=2;
        elif inp == ord('e'):
           if cycle < 30:
             cycle += 1
        elif inp == ord('q'):
           if cycle > 0:
             cycle -= 1
        elif inp == ord('c'):
           screen.clear(); screen.border()
           drawBombs()
        elif inp == ord('m'):
           fire = True

        curses.flushinp() # stops key queing
        jj = inp
        try:
           angle = radians(dangle)
           dy = float(-0.5*(vel*sin(angle)));
           dx = float(vel*cos(angle));
           currY = currY + dy; currX = currX + dx;
           digg, vect = dispChar(dy,dx)

        except:
            pass
        curses.napms(timer)
        screen.refresh();
        currY = currY % row; currX = currX % col
        try:
           screen.addstr(currY, currX, ord(digg))
        except:
           pass
#          curses.endwin(); exit(); trace;

#######   Add position and projectile #######
        place = (currY,currX,digg)
        rewtable.append(place)
        if fire == True:
           speed = 2
           life = 10
           bullet = [currY,currX,speed*dy,speed*dx, life]
           screen.addstr(0,5,str(len(projectiles)))
           projectiles.append(bullet)    # start pos and vector

#######   Draw Trails   ########
        if len(rewtable)> 30:
           rewtable = rewtable[-30:]
        trails = rewtable[-cycle:]
        if len(trails)>=3:
           gin = trails[-3];  #get semilast added
           screen.addch(gin[0],gin[1],ord(tral))
           gin = trails[-2]  #gets last position overrides with #
           screen.addch(gin[0],gin[1],ord(vect))
```

```
      if len(trails)>=cycle:
          jin = trails[0]
          screen.addch(jin[0],jin[1],ord(' '))
#########-- Draw Bullets --#########
          for b in xrange(0, len(projectiles)):
               try:
                   bull = projectiles[b]
                   if bull[4] < 0:
                      projectiles.pop(b)
                      screen.addch(bull[0]-1,bull[1],ord('#') )
                      screen.addch(bull[0]+1,bull[1],ord('#') )
                      screen.addch(bull[0],bull[1]-1,ord('#') )
                      screen.addch(bull[0],bull[1]+1,ord('#') )
                   else:
                      newy = (bull[0]+(bull[2]) ) % row
                      newx = (bull[1]+(bull[3]) ) % col
                      screen.addch(bull[0],bull[1],ord(' ') )
                      screen.addch(newy,newx,ord('o') )
                      projectiles[b][0]= newy;
                      projectiles[b][1]= newx
                      projectiles[b][4]= projectiles[b][4] - 1  # decrement life

               except IndexError:
                     pass
               except:
                     pass

def dispChar(yy,xx):
    b1,b2 = 0.5, 0.65
    arr = []

    if(yy >= b1):
       if abs(xx)< b2:
          return 'V','|'
       elif xx>=b2:
          return '\\','\\'
       elif xx<=-b2:
          return '/','/'
    elif(abs(yy) < b1):
       if abs(xx) < b1:
          return 'o','O'
       elif xx>=b1:
          return '}','='
       elif xx<=-b1:
          return '{','='
    elif(yy <= -b1):
       if abs(xx) < b2:
          return '^','|'
       elif xx>=b2:
          return '\\','/'
       elif xx<=-b2:
          return '/','/'
    return '+','+'


screen = curses.initscr();row = 0; col = 0;conswidth();
curses.curs_set(0)

curses.noecho()
try:
  drawBombs()
  drawer(timer=100, cycle = 12)
```

```
except KeyboardInterrupt:
    curses.endwin()
    exit()

#except:
#    curses.endwin()
#    exit()
#    trace
```

GMaps profile implementation
Please consult the 'Other' folder in the CD, and open gmaps_connect.html

References

[1] Eric Domenjoud, The footbib package for Latex, February 2007, http://mirror.hmc.edu/ctan/macros/latex/contrib/footbib/footbib.pdf Page 12

[2] Massimo Picardi, Background Sutraction Techniques, Computer Vision Research Group, University of Technology, Sydney http://www-staff.it.uts.edu.au/~massimo/BackgroundSubtractionReview-Piccardi.pdf Page 13

[3] Howard E Burdick, Digital Imaging:Theory and Applications, McGraw-Hill 1997, University of Virginia, ISBN:0079130593, 9780079130594 Page 13

[4] Chris Ducket, Nokia Sells Qt to Digia, techrepublic, August 9 2012, http://www.techrepublic.com/blog/australia/nokia-sells-qt-to-digia/1268 Page 10

[5] Scott Chacon (schacon), GitHub vs BitBucket, Pocoo http://www.pocoo.org/~blackbird/github-vs-bitbucket/bitbucket.html Page 12

[6] Adams, A., Talvala, E., Park, S. H., Jacobs, D. E., Ajdin, B., Gelfand, N., Dolson, J., Vaquero, D., Baek, J., Tico, M., Lensch, H. P., Matusik, W., Pulli, K., Horowitz, M., and Levoy, M - The Frankencamera: an experimental platform for computational photography, ACM SIGGRAPH 2010 http://doi.acm.org/10.1145/1833349.1778766 Page 15

[7] Noah Snavelly, Introduction to Computer Vision, Cornell University, New York http://www.cs.cornell.edu/courses/cs4670/2010fa/ Page 18

[8] Maemo Packages - LibCV http://maemo.org/packages/view/libcv-dev/ Page 18

[9] Oliver Hookins, Missing dependencies bug, Maemo mail lists http://lists.maemo.org/pipermail/maemo-developers/2011-August/028583.html Page 18

[10] Hon-Kwok Fung, CIMG IPL conversion class header text http://sph2grid.googlecode.com/svn-history/r14/trunk/CImg/plugins/cimg_ipl.h Page 19

[11] QML Signals and signal hander miscategorization, Qt Bug reports https://bugreports.qt-project.org/browse/QTBUG-17072 Page 44

[12] QtMobility bugs http://www.developer.nokia.com/Community/Discussion/showthread.php?201846-Sending-Email-Using-Qt-Mobility Page 50

[13] "Brian G", Creating and sending email with Qt, Qt Mail Lists http://lists.trolltech.com/qt-interest/2006-10/thread01158-0.html Page 50

[14] Nicolai's Mailcmd Announce page http://talk.maemo.org/showthread.php?t=82918 Page 50

[15] Eino-Ville Talvala, Andrew Adams et al, FCam API Documentatio and Examples http://fcam.garage.maemo.org/examples.html Page 50

[16] Simo Piiroinen, Alarm Daemon, Source: http://maemo.org/packages/view/alarmd/ Page 65

[17] Carol Alexandru, Alarmed Scheduling Application Package and Source - http://maemo.org/packages/view/alarmed/ Page 66

[18] Maemo Documentation Wiki, Alarmd Framework - http://wiki.maemo.org/Documentation/Maemo_5_Developer_Guide/Using_Generic_Platform_Components/Alarm_Framework Page 66

[19] Mickey.Mouse.Theme.Song - Youtube Account created to demonstrate TimeLapse videos, (more suitable unique names were hard to create), http://www.youtube.com/channel/UCbs6SUFR4UW6oREchDsRSpQ?feature=mhee Page 64

[20] Mehmet Tekman (tetris11_), Phonestreamer Announce Page, Maemo Forums http://talk.maemo.org/showthread.php?t=70877 Page 68, Page 83

[21] Roy Hills, Arp-Scan Author, http://linux.die.net/man/1/arp-scan Page 71

[22] Colin Stephane, Arp-Scan for Maemo Announce Page, Maemo Forums http://talk.maemo.org/showthread.php?t=73467 Page 71

[23] Angry Yoda Master(AMY), Feature request on Phonstreamer Announce thread, Maemo Forums, http://talk.maemo.org/showthread.php?p=1054381\&highlight=profiles\#post1054381 Page 72

[24] Exras Repository Process Definition, Maemo Wiki, http://wiki.maemo.org/Extras_repository_process_definition Page 83

[25] Extras-Devel and Promoting to Extras-Testing, Maemo Wiki, http://wiki.maemo.org/Extras-devel#Promoting_packages_to_extras-testing Page 83

[26] Extras-Testing and Promiting to Extras, Maemo Wiki, http://wiki.maemo.org/Extras-devel#Promoting_packages_to_extras-testing Page 83

[27] User 'deed', Forum Post, PhoneStremer Announce Page, Maemo Forums, http://talk.maemo.org/showpost.php?p=1255370&postcount=2 Page 83

[28] Natasha Lomas, Blookmber Business, Nov 2009, http://www.businessweek.com/globalbiz/content/nov2009/gb2009114_367401.htm Page 86

[29] QinstallMessageHandler, QtGlobal API http://doc.qt.nokia.com/4.7-snapshot/qtglobal.html#qInstallMsgHandler Page 88

[30] Maemo Packages Database, Maemo http://maemo.org/packages/ Page 91

[31] Qt Downloads, Nokia Labs http://qt.nokia.com/downloads Page 91

[32] GStreamer Example Class C++, Maemo Documentation Wiki http://wiki.maemo.org/Documentation/Maemo_5_Developer_Guide/Using_Multimedia_Components/Camera_API_Usage Page 92

[33] Vio Detection, Motion Detection Application, http://vio-detection.com/ Page 85

[34] Senstic , Motion Detector, Iphone http://itunes.apple.com/gb/app/motion-detector/id331443079?mt=8 Page 85

[35] Motion Detector Pro, Android, (Anonymous Developer) http://motiondetector.homepagepublisher.com/ Page 85