Course: Concurrent and Real-Time Programming

Written by:
Jan Svensson
Rickard Hallerbäck
Erik Wilstermann

Video-Surveillance-System

1.0 Introduction

This design document describes the system design of a camera surveillance system. The system consists of up to two cameras and one client. Each camera has one server which can be accessed through the TCP/IP protocol and the user connects to the cameras by entering the URL of the camera servers. Once a connection between the client and the server is established, the server regularly sends JPEG images taken by the camera. The client receives these images and displays them on a graphical user interface to the user.

Using the GUI of the application developed in this project the client is able to see images displayed from the camera where these images appear in a manner that is compensate for different network delays. If this compensation is active the received pictures from the cameras are shown a fixed time from when they were taken. However if the feature is not active any incoming picture is displayed as fast as possible. In addition to this, there is also a mode in which the client automatically determines if this compensations is be activated or not.

When it comes to the frame rate of the cameras, there is one fast mode (‘Movie’) and one slow mode (‘Idle’). The user is be able to set the cameras in either of these modes, but both cameras shall be set to the same mode. An elegant feature of this program is that i detects motion and if the cameras are in the ‘Idle’ mode and motion is detected, both cameras switch to the ‘Movie’ state and thus send images over the client-server connection as fast as they can. A sign labeled ’Motion Detected’ will turn green when this happens.

2.0 Server design

The server system implemented in this project was written in C using the POSIX superset of the standard C library intended to run on a embedded Linux AXIS camera device. The program was specifically compiled to run on an AXIS camera of the model M3006. The server opens two sockets for this program occupying the following ports:

Port 9090: used for the client application, outputting jpeg images and receives commands governing mode of operation.

Port 9091: a web interface displaying an image and a link where one can download images from.

2.1 Source code

In the source code of the project there is a specific directory called 'Server' where all files regarding the server is stored. There is a makefile that builds the server executable which can only be run on a machine that has the particular compiler for the model M3006 axis camera chip installed. In this directory there is also particular source code files that take part in this program and here is a brief description of what they do.

2.1.1 Socket Writer

This unit is used for sending jpeg data over socket opened from port 9090 to the client. The function 'stream_callback' within this translation unit is run as an independent POSIX thread entity that listens for incoming TCP/IP connections and starts a client-server streaming session. The communication between the client and the server is described in further detail in the Protocol section of this document.

2.1.2 Socket Listener

This is a translation unit that supports reading on port 9090 for client requests on mode of operation. It supports only one client at a time so make sure you are only using this application one at a time. The function 'stream_listener_callback' will run as an POSIX thread that listens on incoming messages on port 9090 over a TCP connection initiated in the thread running 'stream_callback' after a new connection is established. The communication between the client and the server is described in further detail in the Protocol section of this document.

2.1.3 Mode Handler

The server supports two modes of operation, these are described in further detail in the introduction of this document. This translation unit is responsible for setting the mode of operation in a thread-safe manner across all the independent thread running in the application.
This translation unit contains the reentrant function 'set_mode' that is used by all other threads.

2.1.4 Motion Listener

An elegant feature of the application is the possibility to detect motion and this translation unit has a function called 'motion_callback' that is run as an independent POSIX thread that notifies the rest of the system if it has detected motion and a switch of mode of operation might occur.

2.1.5 Socket

This translation unit is used as an interface for setting up and closing socket connections. It is used to establish the image streaming application as well as the independent web interface.

2.1.6 Web Listener

This translation unit is responisble for listens on the port 9091 that will work as a restful API. It will await a http get requests and outputs HTTP-headers describing that a (image/jpeg) file is transferred followed by a newline when it receives the restful API request "GET /image.jpg ...". Otherwise it will output a very basic HTML document containing a <img> DOM-element with a source attributed to it that corresponds to the HTTP request that gives the image.

2.1.7 Http Server

This translation unit contains the 'main' method that runs and initializes the streaming, motion detecting and web listening POSIX threads described above.

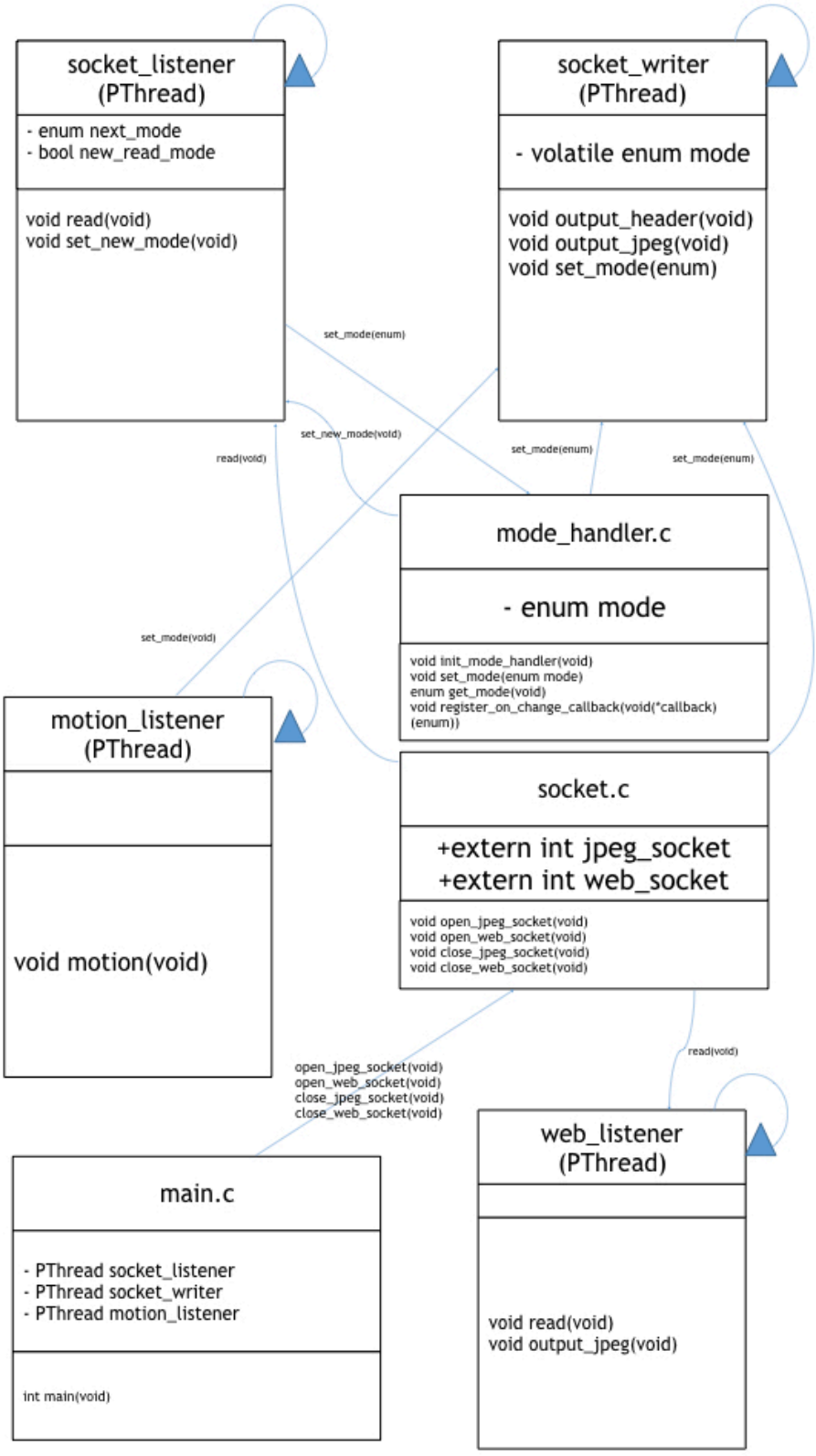

Figure 2.1.8 Server design UML

3.0 Client design

The client design is made with a poor type of MVC design containing three packages (Control, GUI and Model). The system uses three threads that are controlled inside a monitor that decides when they are able to run.

3.1.1 ServerReader

This is a thread that creates a socket from the path and port provided in the constructor method and reads from this socket.
It is responsible for parsing the incoming frames and headers from the server. There will be an instance of this class for each server. Upon receiving a frame and parse it for its time-stamp, it will create a ServerFrame instance and call the 'Monitor' to store it in its FrameBuffer with the 'addFrame' function. If the mode of operation from a camera has changed since the last frame, a call to 'changeMode' on the 'Monitor' will be called to tell the monitor to change the other servers mode if it is not the same as this ones. There will be at least two of these 'ServerReader' threads since the system should support 2 servers. The communication between server and client is described in the 'Protocol' section of this document.

3.1.2 FrameBuffer

The FrameBuffer stores ServerFrame objects in time descending order. This passive class stores all incoming frames and is able to peek and also return the oldest (first sent) frame from each server. The buffer works as a LIFO-queue in regards to the timestamps of every ServerFrame entry. The 'add' function iterates over the list and finds the proper place for the new entry to be stored.

3.1.3 ServerFrame

A container class with useful information related to received frames.

3.1.4 Updater

This thread is responsible for updating the contents of the JPanels in the GUI of the application. All calls to swing objects will be realized by using the 'SwingUtilities.invokeLater' command.
The 'Updater' thread will call the function 'update' on the 'Monitor'. As long as the 'FrameBuffer' has no 'ServerFrames', the method will put the thread to wait. When the 'FrameBuffer' has been updated the method will act according to the by the client set mode of operation.
There will be 3 supported modes of operation; Synchronous, Asynchronous and Auto. These will be described in greater detail below:

* *Synchronous:*

After a frame has been received and placed in the 'FrameBuffer' off the Monitor class by the 'ServerReader' the 'Updater' thread will be notified and calculate the time difference from the current time, t, to the time the server sent the image, ts, plus a delay, d, of initially 200 ms. The thread will then wait for (ts + d - t) milliseconds if this is a positive value. Otherwise, if negative the the Updater thread will increase its delay time, d, by another 200 ms and make the same attempt to wait. This process will continue until the desired delay time has reached a threshold value, say 2 seconds, after which it will simply display the image.

* *Asynchronous:*

In asynchronous mode the frames are retrieved from the ‘FrameBuffer’ and instantly notify the ‘Updater’ thread that will place the image in the JPanel of the GUI as soon as they are placed in the ‘FrameBuffer’.

* *Auto:*

In auto mode the client switches between synchronous and asynchronous mode automatically. If the time difference between when the images was taken and when they were received exceeds a certain threshold, the asynchronous mode will be chosen. However if this difference is less than the time threshold, synchronous mode is chosen. All time differences are calculated using a low pass filter making it more robust to frames with outlier delay times.
In order to avoid rapid switching between the two modes, in addition to the low pass filter, the time threshold is slightly higher when in asynchronous mode, and slightly lower when in synchronous mode. This creates a hysteresis band in regards to the time threshold and thus avoids rapid switching between the synchronous and asynchronous mode.

3.1.5 ServerWriter

This thread is responsible for writing to a specific server. There will be one ServerWriter for each open server connection. The purpose of the writings will be to set the mode of operation from Idle to Movie and vice versa. The ServerWriter will send a request to the server to switch mode of operation after a button has been pressed by the user or if one camera has entered ‘Movie’ mode upon motion detection while the other has not. It calls the function ‘nextMode’ to await requests for a new mode of operation.

3.1.6 Monitor

This monitor class is responsible for synchronizing the threads and making the appropriate logic operations. Synchronization is established by using synchronized methods and the wait/notify functionality. The ‘update’ method is called by the ‘Updater’ class and it is described under the ‘Updater’ class. The ‘nextMode’ method blocks until a request for a new mode of operation has been received. The request can be sent from the client or a server that entered movie mode, requiring the other servers to do the same. The changeMode function is called from a ServerReader or from a button when the client asks for a mode change. All GUI related actions are handled using the swing ‘invokeLater’ method.

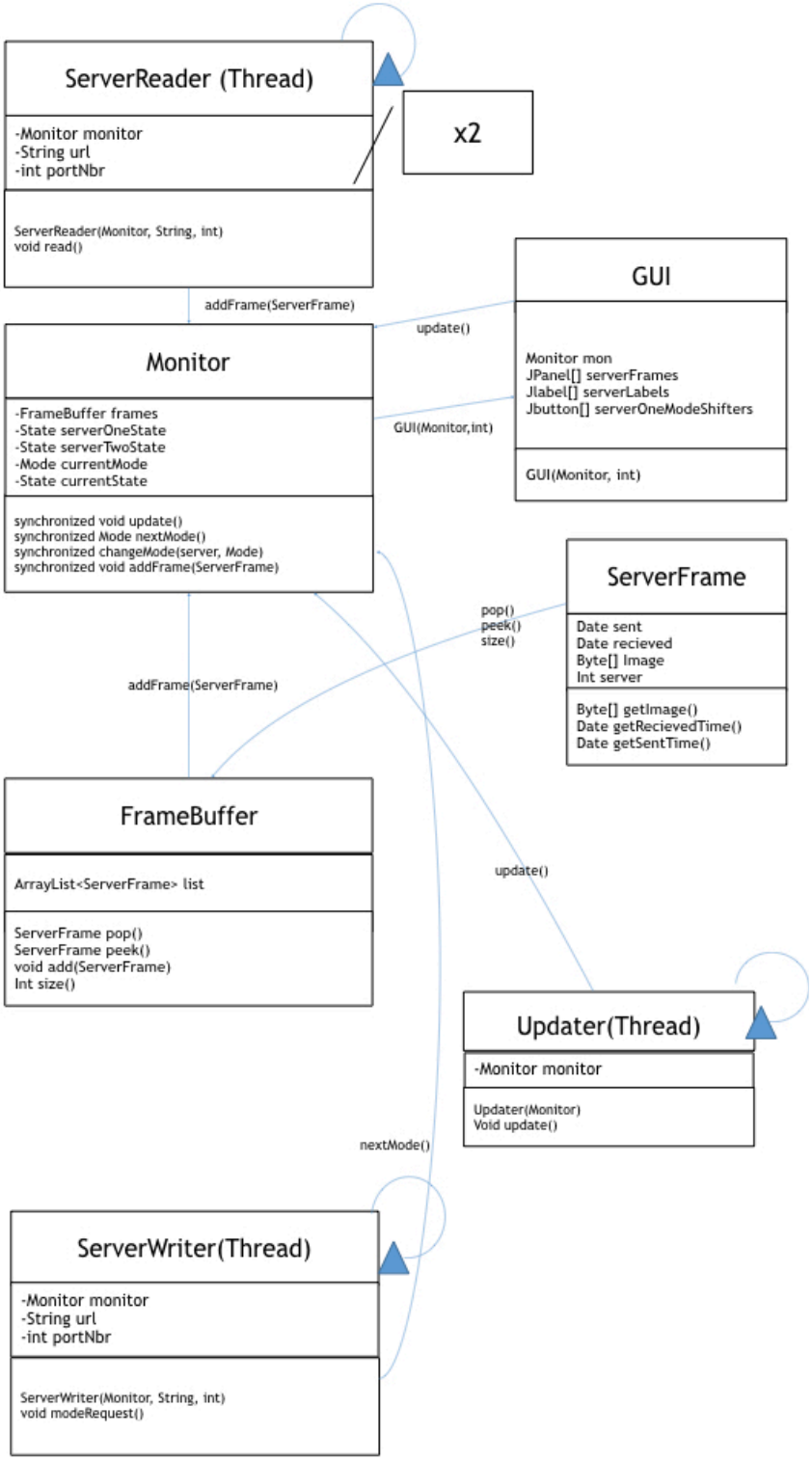

Figure 3.1.0: Client design UML

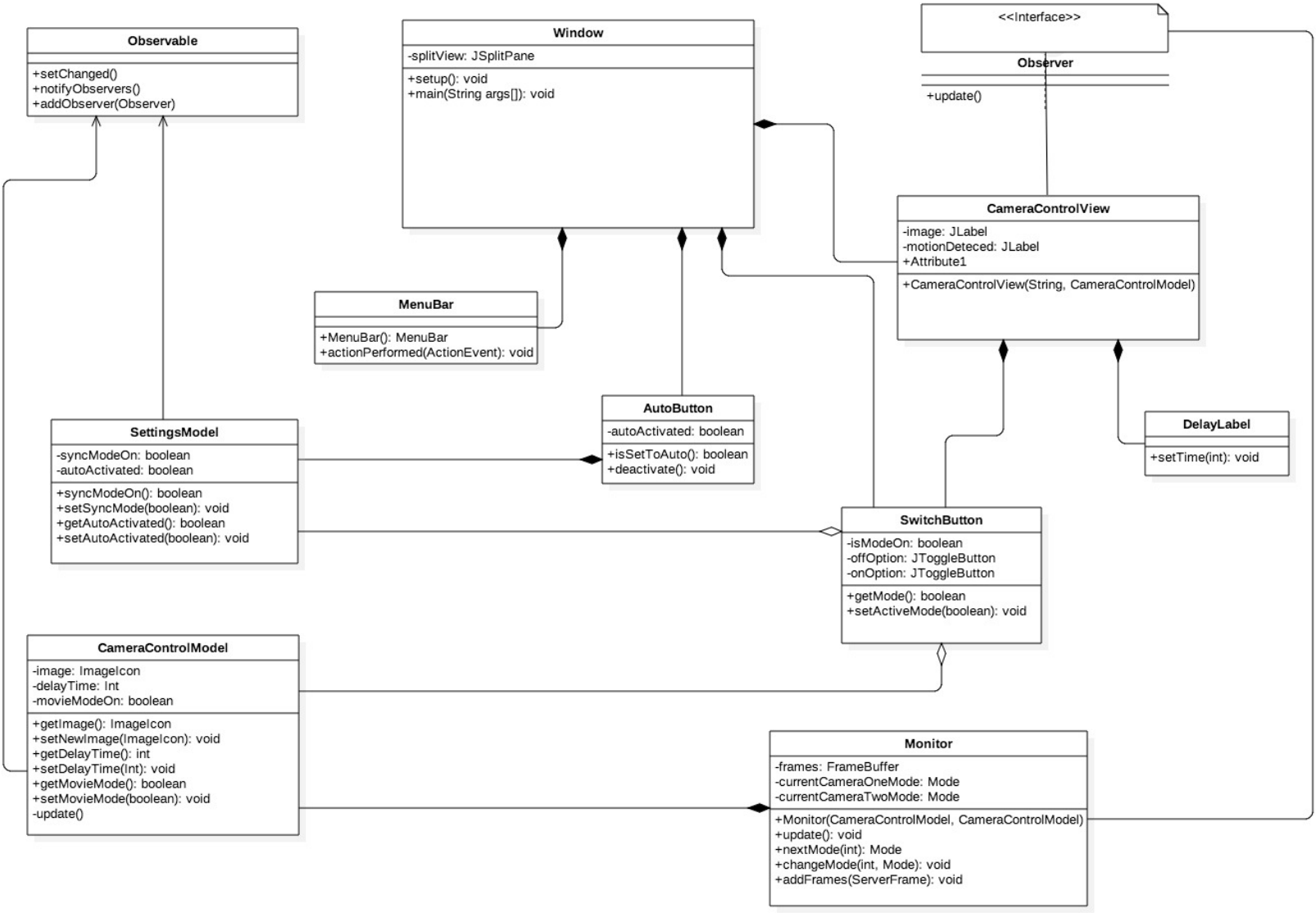

Figure 3.1

A UML-diagram of the GUI design, showing the Model integration with the Views and the Monitor.

3.2.0 GUI

This GUI is build with mainly Java.swing classes. The layout is a simple Java BoxLayout. Classes built on JButton and JToggleButton makes up the outer controls. Then JSplitPanel splits the view into two camera views that holds separate model data that can be changed with ActionListeners. JLabel's are then set inside the camera views to show both delay time and motion detection.

Furthermore, the models in the view are built around the Observer/Observable design pattern to easily update the view or monitor whenever the value in them are changed. A complete UML diagram is presented in figure 3.1 above.

4.0 Protocol

The client and the server communicates over a TCP/TP connection on port 9090 on the server. The server sends data packets adhering to the following header specification:

4.1 Header specification

ASCII [10 bytes]	Timestamp [8 bytes]	Image size [4 bytes]	Mode [1 byte]	JPEG [Image size bytes]
'STARTFRAME'	0xUNIXTIME	0xIMAGESIZE	Bit 1: [1|0]*, Bit 2: [1|0]*	JPEG Data

* The bit 1 in the mode byte tells the client what mode the server is running in, 1 means 'Movie' and 0 mean 'Idle'.

* The bit 2 in the mode byte tells the client wether or not a transition from 'Idle' to 'Movie' after motion has been detected has been acknowledged by the client. 1 means 'motion was detected' and 0 means 'movie was set by the client'. This helps the client program to determine wether or not it should send a request to set the other server to 'Movie' mode. If it recieves a 1 it should check the status of the other server then set it to 'Movie' if it is in 'Idle'.

4.2 Client to server messages

The connection is established over TCP which will make sure all packages are received in order. We thus communicate very sparsely from the client to the server by sending a request for a new mode to be set. This is done sending a byte containing a single ASCII character where '0' means 'set mode of operation to movie' and '1' means 'set mode of operation to Idle'.

5.0 Users Guide

The program has a fairly simple design with a smooth interface. The cameras are connected as soon as the program starts with the pictures displayed right away.

For controlling the Synchronous / Asynchronous mode the user can chose the desired mode at the bottom of the window, on a flip switch. If the user wishes to let the program select mode automatically, for the best mode with the network connection, there is a Auto button that can be enabled.

The two cameras are shown on beside each other with a title at the top, saying which camera is which. For each camera the user can choose either Movie - or Idle mode.

- Idle mode means that the camera displays pictures with a large interval
- Movie mode means that the camera plays a continuous video stream.

The cameras are also able to detect motion. When they do, the 'Motion detected' label will switch from **red** to **green**.

Lastly, the delay between each frame will be shown just below the label.

For closing the program, the user can choose File -> Exit.

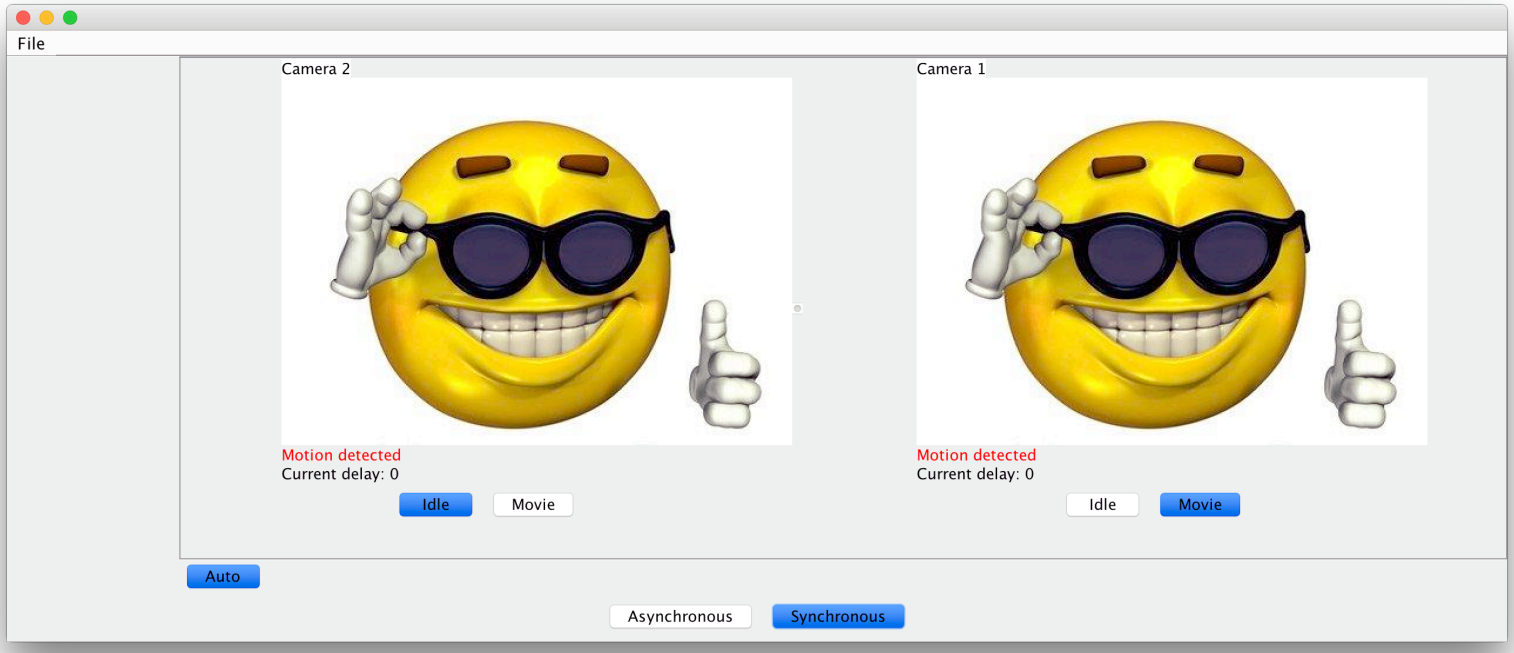

Figure 5.1

An image of the user interface. It has support for two cameras on either side with a controls for each camera and for the whole system.

Instructions for setting up the system (from the zip-file)

5.1 Setting up the server

It is important to note that the program comes shipped with an executable that needs to be downloaded to the camera. If the program instead is set up from source code it requires that the specific compiler for the AXIS camera is installed. Please investigate the prerequisites there for building be looking at the 'makefile' in the Server directory of the source code file archive. If an executable is already at hand the best way to get the server program to run is by following these instructions:

1. Make sure you have a SSH-client installed, most machines running MacOSX or Linux have these features installed and can easily be accessed from the terminal. Also make sure you have the program SCP installed

From the terminal window navigate to the server directory of this project files hierarchy and write the following commands:

2. scp http_server <USER>@<CAMERA-IP>:~

3. ssh <USER>@<CAMERA-IP>

You might be required to enter a password for the particular USER at this point

4. chmod ugo+x http_server

5. ./http_server

Now the camera should run this application and the ports 9090 and 9091 are open. If any socket opening or binding fails the program will be aborted.
Set the USER to any registered users on the camera commonly there is only 'root' as an option and more users can be configured later using the axis camera web interface provided in the firmware. Please consider reading the manual for the specific AXIS camera to retrieve this information. To view the web interface open a browser and access it using the following URL: http://<CAMERA-IP>:9091 where the camera IP depends on the network the camera is configured in.

5.2 Setting up the client

Its very important to have a server program running in order for this client program to make any sense, therefore please read the chapter 5.1 on how to set up the server program before running this client program.

The system comes with an archive containing source code. If the client program is set up from source code then we recommend using the 'Eclipse' IDE selecting the 'Client' directory of the

source code archive as a workspace and creating a new java project with the name 'Client' to make the files appear. The client program can after that be started simply after hitting the 'run' button in the IDE.

The client program is also shipped as a runnable JAR file one can start from the terminal. In windows one can start this JAR file by entering the following in the terminal:

<PATH-TO-JAR> <SERVER_1_URL> <SERVER_2_URL>

Where 'SERVER_1_URL' and 'SERVER_2_URL' are paths to where the server program is running. If the URL is invalid the window will appear displaying no content. So make sure the server is configured correctly.

The runnable JAR file thus takes to arguments that are URL:s to servers running the server program.