A Novel Application of Real-Time Video Streaming and Recording to Wheelchair Skills Training
ENPH 479 - Group 1252

Daniel L. Lu
Anson Liang
Alec Douglas

January 14, 2013
Executive Summary

This project implements a novel application of wireless real-time video streaming technology for use in the remote training of new wheelchair users. Professional training for new wheelchair users can significantly improve their quality of life and reduce the risk of injury during wheelchair operation. However, such training is often expensive, time-consuming, and difficult to access in many areas. An alternative to one-on-one training with a therapist is to provide wheelchair users with a portable device which displays instructional videos about proper wheelchair operation technique. To this end, Dr. Ian Mitchell and Andy Kim of UBC Computer Science and Dr. William Miller and Ed Giesbrecht of UBC Occupational Sciences & Occupational Therapy have developed an application called EPICWheelS (Enhancing Participation In the Community by improving Wheelchair Skills) for tablet devices running the Android operating system. In addition to displaying instructional videos, EPICWheelS provides users with the ability to communicate with a remote wheelchair therapist through voicemail.

This project explores the possibility of adding important functionality for the EPICWheelS app to allow a remote therapist to more effectively evaluate the performance of the wheelchair trainee. Our solution allows the app to wirelessly stream video from a camera mounted on a secondary Android device to a tablet running the application. This enables the user to record themselves demonstrating new skills. The video will be transmitted in real time from the camera device to the tablet over a WiFi network, and be simultaneously recorded by the tablet. The recorded video can then be uploaded to a wheelchair therapist for evaluation.

Our solution achieves a satisfactory performance of 15 frames per second for videos with the standard VGA resolution (640 × 480 pixels) with JPEG compression per frame. The delay associated with transmitting the video from the camera device to the tablet is less than 300 ms and is acceptable for our purposes.
Contents

Executive Summary .......................... 1

List of Figures ............................ 4

List of Tables .............................. 5

1 Introduction .................................. 6
   1.1 Background ................................ 6
   1.2 Statement of Problem ...................... 6
   1.3 Project Objective ......................... 7
   1.4 Organisation of this Report .............. 8

2 Technical Discussion ....................... 8
   2.1 Comparison of Different Approaches to Video Streaming ............ 8
       2.1.1 USB Video Camera ...................... 8
       2.1.2 Bluetooth ............................ 9
       2.1.3 Network Streaming ..................... 10
   2.2 Brief Overview of Streaming and Recording ..................... 10
       2.2.1 Transmission of video data over a network .................. 10
       2.2.2 Recording of video data .................. 11
   2.3 Software Used ............................ 11
       2.3.1 IP Camera ............................ 11
       2.3.2 MJPEG Libraries ....................... 12
       2.3.3 FFMPEG Library ....................... 13

3 Experimental Equipment ..................... 13
   3.1 Structure and Operation of Demo Application ................... 14
       3.1.1 Connecting ............................ 14
       3.1.2 Displaying Video ....................... 16
       3.1.3 Recording Video ....................... 16
       3.1.4 Encoding Video ......................... 18
   3.2 Experimental Procedure ................... 20
       3.2.1 Latency Tests .......................... 20
       3.2.2 Framerate Tests ......................... 21
       3.2.3 Practical Usage Tests .................. 22

4 Results and Discussion ..................... 22
   4.1 Latency .................................. 22
   4.2 Framerate ................................ 24
   4.3 Practical Usage Tests ..................... 25

5 Conclusions ................................ 26
6 Project Deliverables
   6.1 List of Deliverables .......................................................... 26
   6.2 Financial Summary ............................................................. 27
   6.3 Ongoing Commitments by Team Members ............................... 27

7 Recommendations 28

8 Appendix: FFmpeg build instructions 28

References 29
List of Figures

1 View an instruction video on wheelchair usage in the EPICWheelS application. ......................................................... 7
2 Photo of an example of a wireless USB camera [8]. ................................................................. 9
3 Photo of an example of a USB to micro-USB adapter. ................................................................. 9
4 Screenshot of IP Webcam application upon startup. ................................................................. 12
5 General overview flowchart illustrating the basic operation of the demo application developed for this project. ......................................................................................................................... 14
6 Screenshot of the demo application on startup (top), and after the “Connect” button has been pressed (bottom). ................................................................. 15
7 Screenshot of the demo application during video streaming. At this stage the user may either begin recording by pressing “Record” or disconnect from the camera by pressing “Disconnect”. ................................................................. 17
8 Screenshot of the demo application immediately after pressing the “Record” button. The application prompts the user for the name of the video to be saved. 17
9 Screenshot of the demo application while it is recording the video. The amount of elapsed time is shown in the center top and the user may stop the recording at any time by pressing “Stop Recording”. ................................................................. 18
10 Screenshot of the demo application while it is encoding the video after recording. 19
11 A flowchart containing the display, recording, and encoding processes where items in green are handled by the MjpegView class and items in blue are handled by the MjpegInputStream class. ......................................................................................................................... 19
12 Diagram illustrating the experimental set up for measuring latency, and the latencies from the two sources of latency. ......................................................................................................................... 20
13 One of the photos from the latency test. In the background is a stopwatch; on the right is a device running the IP Webcam application, and on the left is a device running the demo application. ......................................................................................................................... 21
14 Graph of latency times for three different sets of settings. For more details, please refer to Tables 1, 2, 3. ......................................................................................................................... 24
15 Frames extracted from the sample videos corner.mov, midrange.mov, and straight.mov respectively, in clockwise order. ......................................................................................................................... 25
List of Tables

1. Test results for latency in milliseconds from stopwatch to camera for different settings of video resolution and frame compression quality. . . . . . . . . . . 23
2. Test results for latency in milliseconds from camera to tablet for different settings of video resolution and frame compression quality. . . . . . . . . . . 23
3. Test results for total system latency in milliseconds for different settings of video resolution and frame compression quality. . . . . . . . . . . 23
4. Test results for average number of frames per second for different settings of video resolution and frame compression quality. Due to fluctuations in wireless network bandwidth, the framerate can vary with an uncertainty of ±2 fps. For results labeled N/A, the application became unresponsive. . . . 25
5. Financial summary of components used in this project. . . . . . . . . . . . . 27
1 Introduction

1.1 Background

New users of wheelchairs benefit greatly from professional instruction in efficient and safe operation of a wheelchair. A low-intensity systematic course on wheelchair use over seven weeks yields a noticeable increase in mechanical efficiency of propulsion techniques of wheelchair users [1]. Wheelchair skill may be enhanced even further after a brief period of extra training such as the Wheelchair Skills Training Program in addition to the standard amount of initial rehabilitation [2]. As such it is desirable for new wheelchair users to receive as much training as possible. However, training programs are very time-consuming and expensive for hospitals and rehabilitation centers. Furthermore, wheelchair users who live in remote areas can find it extremely difficult to access the nearest training facility.

A natural solution to these problems is to provide the trainee with a portable device which displays training material. A tablet computer is the ideal candidate since it is affordable, lightweight, and easy to use. Using apps on such mobile devices has been shown to have many benefits such as reducing the cost of healthcare provision [3]. The proliferation of tablet computers running the Android operating system or iOS in recent years has rendered them sufficiently affordable to be provided to users for training purposes. For example, the vitfiz system [4] uses the accelerometer in Android devices to provide feedback to patients as they are performing physical exercises. The StrokeLink application for the iPad plays videos to stroke patients to teach them rehabilitation exercises [5].

An Android application called EPICWheelS (Enhancing Participation In the Community by improving Wheelchair Skills) has been developed by Dr. Ian Mitchell and Andy Kim of UBC Computer Science and Dr. William Miller and Ed Giesbrecht of UBC Occupational Sciences & Occupational Therapy. The EPICWheelS application allows a wheelchair user to learn new wheelchair skills and proper operation technique by watching a series of short, pre-recorded videos (Figure 1). The application also contains voicemail functionality to allow the trainee to communicate with their therapist and tracks usage time.

The EPICWheelS application has several advantages over traditional one-on-one training. Firstly, it is capable of providing wheelchair skills training to persons residing in areas without easy access to rehabilitation centres and therapists. Secondly, the email and voicemail functionality in EPICWheelS allows a therapist to review and evaluate the trainee’s progress without necessarily being present at the time at which the trainee is performing the training exercises.

1.2 Statement of Problem

A key shortcoming of having only email and voicemail communication is that the remote therapist cannot directly observe the trainee performing the training routines. Consequently the therapist would be unable to evaluate the performance and accuracy of the wheelchair user applying the newly learnt techniques. As such, the therapist cannot provide useful
feedback with the same level of helpfulness as that which can be provided by a one-on-one therapist.

This problem is resolved by implementing a system to allow the wheelchair user to record videos of themself performing the training exercises, which can then be sent to the remote therapist. Since wheelchair users are limited in their mobility, special care must be taken to design the system such that a wheelchair user can easily take videos of themselves.

After setting an external camera in a stationary position, the user must be able to see what the camera is recording, so as to ensure that the frame of view is sufficient to capture the entire training routine (which may involve a large area). The camera must also be wireless, because the training routine may involve a large amount of motion that will entangle and damage any wires connecting the tablet to the camera.

In our solution, the process of recording oneself performing wheelchair training routines consists of the following: First, the tablet running EPICWheelS is situated on the wheelchair with the user (e.g. on the user’s lap). After the wheelchair user places the camera device on a stable surface (e.g. a shelf or desk), they can begin streaming live video from the camera to the tablet. They may then start or stop the recording simply by touching the respective controls on the tablet.

After researching different options for wireless video cameras, we have decided to use a second Android device with a built-in camera, such as a smartphone, as the external camera. This will be used in conjunction with a free and open-source application that transforms the device into an IP camera, allowing it to stream video over a WiFi network.

1.3 Project Objective

The objective of our project is to develop software that allows an Android application to receive an incoming stream from an IP camera whilst simultaneously displaying and recording it. The application should have a simple user interface that allows the user to easily connect to the IP camera and start or stop the recording at any time. At the end of the recording,
the software should output a compressed video file in a suitable format, which may then be sent to a remote therapist for evaluation as needed.

1.4 Organisation of this Report

This paper will detail our project by first providing an overview of theory of the technical implementation of video streaming. Then, our choice of specific technologies and a comparison of alternative methods will be presented. The organisation and functionality of the demonstration Android application we develop for this project will be discussed. Following will be the details of the experimental procedure performed to validate our solution, the results of performance testing of our solution, and analysis of said results. The paper will conclude with a list of project deliverables, financial summary, ongoing commitments by team members, and recommendations for future work.

2 Technical Discussion

2.1 Comparison of Different Approaches to Video Streaming

Several approaches exist to solve the problem of streaming video from an external camera to any digital device, wirelessly or not. Here we will present a brief discussion of the advantages and disadvantages of three of the most common methods: USB video cameras, Bluetooth, and streaming over the network.

2.1.1 USB Video Camera

A class of webcams and other portable cameras known as USB video cameras connect to the host using a Universal Serial Bus (USB) interface [7]. Although most such devices require a USB cable, rendering them unsuitable for our desired wireless functionality, there are some USB video cameras that are wireless. These often rely on transmitting video data using a proprietary format from the wireless camera to a USB receiver that is plugged into the host device. One example of such a camera is depicted in Figure 2, which features a wireless antenna that transmits the video stream to a USB receiver through Gaussian Frequency-Shift Keying modulation in a radio signal. Most commercially available wireless USB cameras have a range of 10 m to 100 m in open space and are often used as surveillance cameras.

Since most Android tablets do not have a USB port but instead possess a micro-USB port, it is necessary to obtain an adapter for using an Android tablet with a USB device (Figure 3).

Although the hardware is readily available, a significant challenge lies in using the Android operating system to recognise and capture the video stream from a USB camera. Because most Android devices have a built-in camera, there is little or no support for an external camera. We were unable to find any existing API or driver that would allow the USB camera to be detected by the Android operating system. Implementing a driver and API
for a USB camera is difficult and time consuming since this type of implementation needs to be done at a very low level and may require modifications to the kernel of the Android operating system itself. This would in turn require rebuilding and installing a custom build of the operating system on the device, which is not guaranteed to be compatible with all Android devices, and is a significant amount of work. As such, the USB camera option was not chosen for this project.

2.1.2 Bluetooth

Bluetooth is a wireless technology that allows data to be transmitted between compatible devices at a short range. There are many Bluetooth devices such as headsets and peripherals that transmit data within a short range of no more than 10 m. In theory, it should be possible to stream video wirelessly using Bluetooth as well. However, despite recent advances
in Bluetooth speed [9], Bluetooth is designed for low bandwidth data transmission and the transmission speed can decrease dramatically when distance is increased by even a few metres.

There are, however, existing Bluetooth cameras that are commercially available such as the Looxcie Bluetooth camera [10]. These solutions typically rely on proprietary communication protocols and video codecs and it is impossible to integrate the functionality into our application. We were unable to find any open-source solutions that can be utilised for our purposes, and in any case the limited bandwidth of Bluetooth severely restricts the video quality, so this option was not chosen.

### 2.1.3 Network Streaming

An obvious method to transmit data is to send it over the Internet. Data of any kind can be transmitted over a local network such as a wireless WiFi network. To stream live video from a camera, the camera would have to be configured similarly to a web server that continually uploads new video data. Such cameras may have an Internet Protocol address (IP address) and be referred to as IP cameras. A convenient way to create an IP camera is to simply use a secondary Android device and run an application which allows it to stream video from its built-in camera. This approach is more feasible than the previous two approaches because of the availability of software libraries and application interfaces (API). See Section 2.2 for more information about streaming video over a network and Section 2.3 for more details about the specific software packages used.

The only downside to network streaming is that the user must have a local WiFi network. However, most homes nowadays have such networks, and if they do not, WiFi routers are very cheap and a technician may be sent to help the wheelchair user set up the network. Note that the WiFi network need not be connected to the internet for local video streaming to work.

### 2.2 Brief Overview of Streaming and Recording

#### 2.2.1 Transmission of video data over a network

A video stream may be transmitted over the network in real time as a sequence of image frames. However, the naive approach to this, which is to send each frame in the bitmap format, requires a large amount of bandwidth and is not feasible. Thus, each frame must be compressed. A common approach is the Motion JPEG format (MJPEG), in which each video frame of the digital video sequence is separately compressed as a JPEG image. The frame is then sent over the local network using the Hypertext Transfer Protocol (HTTP).

Since any internet network can contain many other agents that are transmitting data, the available bandwidth for the video streaming can fluctuate. The phenomenon of packet loss can also lead to loss of data. As such, not all outgoing frames from the camera device will be received by the tablet. However, because each frame is compressed independently
and transmitted separately, the loss of a few single frames does not lead to failure and is in fact not noticeable.

### 2.2.2 Recording of video data

During recording, each of the received frames is saved to a temporary folder. The final output video must be one single file for ease of handling. There exist a variety of open source libraries that can concatenate and encode separate frames into a single video file, such as FFmpeg (see Section 2.3.3). After the frames are processed into a single video file, the temporary frames are deleted.

### 2.3 Software Used

#### 2.3.1 IP Camera

Internet protocol cameras (IP cameras) are a class of video cameras or webcams that are connected to an internet network. For example, the IP camera may be attached to a wireless WiFi network in the vicinity of the Android device. An IP camera has a unique IP address on the network and functions as a webserver which feeds a stream of video frames to any client that is connected to it through a specific port.

A free and open-source Android application called IP Webcam [6] transforms any Android device with a camera into an IP camera (Figure 4). This application sends an outgoing stream in MJPEG format. It allows the user to choose several parameters for its outgoing video stream, such as the following:

- **Video resolution** (labelled “Resolution”). This determines the pixel dimensions of each video frame. The higher the resolution, the finer the detail that one can resolve but the slower the performance, since more computation would be necessary to process each frame. The user is allowed to select from several common presets, such as VGA (640 × 480), QVGA (320 × 240), WVGA (800 × 480), and WXGA (1280 × 720).

- **Compression quality for JPEG compression of outgoing frames** (labelled “Quality”). The user is allowed to choose a positive integer not exceeding 100, where 100 is the best quality and 1 is the worst quality. A higher quality compression preserves finer detail and has fewer JPEG compression artifacts, at the expense of greater file size and possibly slower performance.

- **Username and password** (labelled “Login/password”). Optionally, the video stream may be protected by a username and password which must be entered upon connection. This allows the user to have improved privacy.

- **Maximum framerate** (labelled “FPS Limit”). The higher the framerate, the smoother motion appears and the better one is able to discern fast movements. However, a higher framerate causes more video frames to be generated, which can increase storage
space requirements for the video. In any case, if a maximum framerate is not set, the video would run at the greatest possible framerate afforded by hardware computation capabilities and network bandwidth.

and some other advanced features which are not necessary to change.

![IP Webcam application screenshot](image)

Figure 4: Screenshot of IP Webcam application upon startup.

Upon establishing a connection, the IP Webcam application displays the video from the camera on the camera device’s own screen in real time as well as its IP address, which is required for connecting to it (see Section 3.1.1).

2.3.2 MJPEG Libraries

Two classes are used in this project for receiving MJPEG video streams. The `MjpegInputStream` and `MjpegView` classes are used to parse and display the input stream asynchronously. They are modified versions of open source classes with the same names that can be found in multiple locations online but have no clear original author (they are therefore assumed to be in the public domain).
2.3.3 FFMPEG Library

Fast forward MPEG (FFMPEG) is an open source project that aims to provide a cross-platform multimedia handling solution. It contains libraries for decoding, encoding, transcoding, muxing, demuxing, streaming, filtering, and playing a large variety of different multimedia files. A branch of the FFMPEG build from the official Git (a version control system) repository developed specifically for the ARM architecture is used in our project [11]. Most Android devices run on ARM processors. The detailed build instruction for the FFMPEG library is listed in Section 8. For our project we will include the FFMPEG executables in our Android application.

The command that we use from FFMPEG is called ffmpeg, a command line tool that is used for converting multimedia files between formats and concatenating and encoding multiple still frames into a video. We therefore only need to build ffmpeg from the source code, and not include other unrelated tools such as ffplay (a media player) and ffprobe (a multimedia stream analyser). The ffmpeg calls various libraries included in the FFMPEG source code, so it is necessary to include all required libraries and ffmpeg into the Android application that we are building. The list of dependent libraries that are required is detailed in the ffmpeg build instructions in Section 8.

Our application invokes the executable:

```
ffmpeg -vcodec mjpeg -i frames_%05d.jpeg -r 15 video.mov
```

Here, the -vcodec option specifies the video output codec, which is specified to be MJPEG as discussed earlier. The -i option specifies the input file, which is composed of all files with filenames of the form frames_%05d.jpeg where %05d indicates integers with five digits. That is, all captured frames from frames_00000.jpeg to frames_99999.jpeg (or whatever the highest number is) will be used. The -r option specifies the frame rate, which is determined by our application automatically based on the number of frames received over the time period. The last parameter is the file name of the output video. Here the output file name is shown to be video.mov as an example, but it could be any .mov file. The .mov extension is a typical extension for MJPEG videos and can be played with a variety of media players, such as Apple QuickTime.

The command line can be invoked from within a Java Android application by using the built-in class java.lang.Process.

3 Experimental Equipment

To test the feasibility of our solution, an Android application that fulfills the desired functionality has been created for demonstration purposes. This will hereafter be referred to as the demo application.
3.1 Structure and Operation of Demo Application

The demo application can be broken up into four main processes: connecting to the video stream, displaying the video, recording video, and encoding video. These processes constitute the majority of the code produced during this project, with the rest of the code handling UI elements and transitioning between the four main processes. The basic functionality of the application is summarised in Figure 5.

3.1.1 Connecting

The IP address of the camera is required to establish a connection between the tablet and the camera. An IP address consists of four integers from 0 to 255, inclusive, separated by dots. In order to connect, the user must first press the button labelled “Connect” and then, when prompted, enter the fourth value of the IP address provided by the IP Webcam application. The reason only the fourth IP address value is required is that the first three values can be automatically detected by looking at the local IP address of the device running the display.
app (they will be the same on both devices) and the port number is always the same (8080). If the app has already been run, the IP address value that had previously been entered is automatically entered in the prompt for faster reconnection. Refer to screenshots in Figure 6.

![Screenshot of the demo application on startup](image1)

![Screenshot of the demo application after the “Connect” button has been pressed](image2)

Figure 6: Screenshot of the demo application on startup (top), and after the “Connect” button has been pressed (bottom).

After the connection prompt has been passed, the application attempts to connect to the specified address and access the video feed. If the HTTP status code returned from the connection attempt is 401, it indicates an authorisation error. In this case, the application alerts the user that the feed requires a username and password and allows the user to enter them before trying again. If no status code is returned before the 5 second timeout, then the user is told the connection attempt has failed and the app returns to the first screen so they may try again. If the connection is successful, the application now has an input stream that can be displayed.

In order to maintain the responsiveness of the user interface during the attempt to connect, the connection is completed asynchronously and the “Connect” button is temporarily disabled during the process to prevent the user from attempting a second connection before the first one is complete.
3.1.2 Displaying Video

After the application has successfully connected to the input stream, the stream is passed to the **MjpegView** class to display it, which in turn uses the **MjpegInputStream** class. Once the **MjpegView** class has the input stream, it creates a new thread that is used to read in frame data and draw it to the screen.

The way the IP Webcam app streams MJPEG video is by sending the data for individual JPEG image frames with a short header and boundary string in between each frame. The short header contains the content type (“image/jpeg”), while some other MJPEG sources send both the content type and the content length (number of bytes in the current JPEG image) in the header. The boundary string is a random combination of letters and numbers prefixed by two dashes and stays the same over the duration of the connection. An example illustrating the format of the data from the IP Webcam app for the first three frames is as follows:

```
--dkj2DNdzZUX9d (boundary string)
Content-type: image/jpeg

--dkj2DNdzZUX9d (boundary string)
Content-type: image/jpeg

--dkj2DNdzZUX9d (boundary string)
Content-type: image/jpeg
```

Once the header is parsed for useful information (i.e. content length), the **MjpegInputStream** class reads in bytes from the stream until the JPEG Start Of Image (SOI) marker is found. Once the SOI marker is encountered, bytes from the stream are read and stored until the JPEG End Of Image/File (EOI or EOF) marker is read. At this point, the frame data between the SOI and EOI markers is decoded into a bitmap which is then returned to the **MjpegView** class and subsequently drawn to the screen. This process is repeated very quickly (typically 10-30 times per second, depending on the resolution of the source video and the device running the demo app) to produce the video seen on the device, until the user either disconnects from the stream or closes the application. Refer to screenshot in Figure 7.

At this point, there is a successful connection and video is being streamed and drawn to the display, so that the user can begin recording the video.

3.1.3 Recording Video

Once the video stream is displayed on the screen, the user can press the “Record” button to begin the process of recording video. After pressing the button, a prompt is shown to allow the user to enter a file name to save the video as. The prompt has a default save name containing a unique string of numbers generated from the current date and time, which the user may choose to use if a more descriptive file name is not required. Refer to screenshot in Figure 8.
Figure 7: Screenshot of the demo application during video streaming. At this stage the user may either begin recording by pressing “Record” or disconnect from the camera by pressing “Disconnect”.

Figure 8: Screenshot of the demo application immediately after pressing the “Record” button. The application prompts the user for the name of the video to be saved.

Once the video save name has been entered and the prompt is closed by pressing “OK”, a flag is set that triggers every frame to be saved as a JPEG image to the device’s storage. These files are given a unique numeric ID so they can be identified later when they are combined and encoded into a video. The saving of each frame is done asynchronously from the display process so that the display framerate is not reduced while waiting for the JPEG files to be saved during recording.

The asynchronous function that writes the frame data to a JPEG file uses built-in Java IO and Android Graphics classes to do so. The main classes used are the Bitmap, FileOutputStream, BufferedOutputStream classes. The Bitmap class is used to compress the bitmap image of the frame into JPEG format given a quality value ranging from 0 to 100, where 0 will produce the smallest file size and lowest quality while 100 will produce the largest file size and highest quality. Currently, we are using a value of 30 for the quality. The two OutputStream classes are used to save the compressed JPEG to the device’s storage.
The file naming convention for these JPEG files is as follows:

```
[recording ID]_frame_[frame number].jpg
```

where the **frame number** is simply the number of the current frame, starting from zero and incrementing every time a frame is saved to JPEG. These JPEG files are saved in a temporary directory that is cleared every time the application is started.

During recording, the “Disconnect” button is disabled so that recording cannot be interrupted by simply disconnecting. This is done in order to avoid having the user accidentally disconnect part way through a recording, which can lead to unexpected behaviour (since the JPEG frames would not be encoded to a proper video file). The user must stop the recording first before disconnecting, or close the application if the need to stop is urgent. Also displayed on screen near the top is the current duration of the recording, in \texttt{mm:ss} format where \texttt{mm} is the number of minutes elapsed and \texttt{ss} is the number of seconds. Refer to screenshot in Figure 9.

![Figure 9: Screenshot of the demo application while it is recording the video. The amount of elapsed time is shown in the center top and the user may stop the recording at any time by pressing “Stop Recording”.

---

### 3.1.4 Encoding Video

When the user wants to stop recording they must press the “Stop Recording” button. Once this button is pressed, any subsequent incoming frames are no longer saved as JPEG files and the process that encodes the saved JPEG files into a playable video begins. The actual encoding is done by the \texttt{ffmpeg} library discussed in section 2.3.3. On the UI side of the process, both “Disconnect” and “Record” buttons are disabled and a progress bar is overlaid on the screen to inform the user of the current progress of the encoding process. Refer to screenshot in Figure 10.

The progress percentage displayed on the progress bar is calculated by parsing the standard output of the \texttt{FFMPEG} library during encoding for the current frame number and comparing that with the total number of frames that need to be encoded.
Figure 10: Screenshot of the demo application while it is encoding the video after recording.

Once encoding is complete, the temporary JPEG files are permanently deleted. This deletion process also has a progress bar because it can take multiple seconds to delete the hundreds of temporary files that are created during recording. However, typically the deletion process takes only a tenth of the time the encoding process requires.

The video file is saved as whatever name was specified at the beginning of recording, and it is saved in the QuickTime (.mov) file format. After the encoding is complete, the application returns to the video display mode to display the live stream, allowing the user to start a second recording or disconnect. The display, recording, and encoding process, from the time a successful connection is made to when the stream is disconnected is summarised in Figure 11.

Figure 11: A flowchart containing the display, recording, and encoding processes where items in green are handled by the MjpegView class and items in blue are handled by the MjpegInputStream class.
3.2 Experimental Procedure

3.2.1 Latency Tests

The time it takes for the video to be taken by the camera and then displayed on the screen of the demo application is the latency we are interested in measuring for our system. If it is too high, there can be a noticeable difference between what the user is doing and what appears on their screen, resulting in a difficult to use system with complications involving when the user should start and stop recording.

There are two main sources of latency in the system. The first is on the camera device and is the time it takes for the IP Webcam app to capture the video from its camera. While this delay is not directly seen by the user, it does add to the overall latency. The second source of latency is the time it takes for the frames to be sent across the network from the IP Webcam app to the demo application and then displayed on the user’s device. While the first source of latency is outside of our control, we can attempt to minimize the second source of latency. In order to measure each of these latencies, we used a stopwatch and a second camera to get a snapshot of the latencies from each source. By pointing the IP Webcam app at the stopwatch, holding the device running the demo app next to it, and taking a picture with the second camera that shows the stopwatch, the first camera, and the demo app all together, it is possible to get a timestamp at each stage and measure the latencies from that. A pictorial example of this test is shown in Figure 12, and Figure 13 is a photo from the test.

![Diagram illustrating the experimental set up for measuring latency, and the latencies from the two sources of latency.](image)

Latency data was collected for three different sets of settings. These settings were changed on the IP Webcam app, and included resolution and image quality. The settings tested were:

- Resolution: 320 × 240; Compression Quality: 50
Figure 13: One of the photos from the latency test. In the background is a stopwatch; on the right is a device running the IP Webcam application, and on the left is a device running the demo application.

- Resolution: 800 × 480; Compression quality: 50
- Resolution: 800 × 480; Compression Quality: 50

The tests were performed using a Sony Ericsson Xperia as the camera running IP Webcam and a Samsung Galaxy S III running the demo application. Note that the screen refresh rate and response time for both the camera device’s screen and the receiver device’s screen are on the order of 10 ms, so any measured result should have an uncertainty of about 10 ms.

3.2.2 Framerate Tests

The framerate is the rate at which video frames are recorded. It is primarily limited by the network bandwidth, the speed at which the camera device can capture new frames, and the rate at which the tablet running the demo application can save the frames.

The framerate of the video stream is continuously monitored by the demo application. While the framerate may fluctuate slightly during the recording in part due to random network effects such as packet loss, it is not anticipated to cause any problems for our purposes. At the end of the recording, the application automatically determines the average framerate $f$, which is defined as such:
where \( N \) is the number of frames recorded for the video and \( t \) is the time duration of the video. The output video file is encoded with framerate \( f \). The framerate was measured for resolutions 320 \( \times \) 240, 640 \( \times \) 480, 800 \( \times \) 480, and 1280 \( \times \) 720 at compression qualities of 1, 25, 50, 75, 100. The tests were performed using a Sony Ericsson Xperia as the camera running IP Webcam and a Samsung Galaxy S III running the demo application.

3.2.3 Practical Usage Tests

Since the ultimate goal of the project is to allow a human therapist to remotely view a wheelchair trainee performing new skills, it is important to show that, in addition to the benchmark figures, the video is also of sufficient subjective quality to provide a pleasant viewing experience which allows one to clearly see what is going on. To test the system under real-world conditions, three videos were recorded:

- \texttt{corner.mov} (1.76 MiB, 14 seconds): a person on a wheelchair navigates around a perpendicular bend in a corridor.
- \texttt{midrange.mov} (3.89 MiB, 31 seconds): a person on a wheelchair navigates around two stools placed in a corridor in a figure-eight pattern.
- \texttt{straight.mov} (2.67 MiB, 21 seconds): a person on a wheelchair travels along a corridor towards the camera.

From the tests of framerate and latency, the IP Webcam setting of a resolution of 640 \( \times \) 480 and quality 50 was chosen to ensure acceptable performance. These tests were performed using a Samsung Galaxy S II as the camera running IP Webcam and a Nexus 7 as the tablet running the demo application.

4 Results and Discussion

4.1 Latency

For each set of three settings, 14 to 16 data points were collected and the full table of data can be found in Tables 1, 2, 3. The results are graphed in Figure 14.

From Figure 14, it can be seen that a change in image quality from 25 to 50 had a negligible effect on latency. However, a change in resolution had a much greater effect, presumably because a higher resolution requires more data to be sent in each frame, causing a corresponding increase in the amount of processing time per frame. It is worth noting again that the latency from the stopwatch to the camera is out of our control, and it makes
Table 1: Test results for latency in milliseconds from stopwatch to camera for different settings of video resolution and frame compression quality.

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Qual.</th>
<th>Mean ±σ (min, max)</th>
<th>Trials</th>
</tr>
</thead>
<tbody>
<tr>
<td>320 × 240</td>
<td>50</td>
<td>115.50 ± 22.00 (88, 132)</td>
<td>16</td>
</tr>
<tr>
<td>800 × 480</td>
<td>25</td>
<td>167.14 ± 24.72 (132, 220)</td>
<td>14</td>
</tr>
<tr>
<td>800 × 480</td>
<td>50</td>
<td>163.60 ± 27.11 (122, 220)</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 2: Test results for latency in milliseconds from camera to tablet for different settings of video resolution and frame compression quality.

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Qual.</th>
<th>Mean ±σ (min, max)</th>
<th>Trials</th>
</tr>
</thead>
<tbody>
<tr>
<td>320 × 240</td>
<td>50</td>
<td>40.12 ± 51.09 (0, 176)</td>
<td>16</td>
</tr>
<tr>
<td>800 × 480</td>
<td>25</td>
<td>93.71 ± 97.75 (0, 396)</td>
<td>14</td>
</tr>
<tr>
<td>800 × 480</td>
<td>50</td>
<td>88.00 ± 81.77 (0, 274)</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 3: Test results for total system latency in milliseconds for different settings of video resolution and frame compression quality.

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Qual.</th>
<th>Mean ±σ (min, max)</th>
<th>Trials</th>
</tr>
</thead>
<tbody>
<tr>
<td>320 × 240</td>
<td>50</td>
<td>155.63 ± 56.71 (88, 308)</td>
<td>16</td>
</tr>
<tr>
<td>800 × 480</td>
<td>25</td>
<td>260.86 ± 101.32 (176, 572)</td>
<td>14</td>
</tr>
<tr>
<td>800 × 480</td>
<td>50</td>
<td>251.60 ± 61.43 (214, 396)</td>
<td>15</td>
</tr>
</tbody>
</table>
Figure 14: Graph of latency times for three different sets of settings. For more details, please refer to Tables 1, 2, 3.

up approximately two thirds of the overall latency in the system. The total latency rarely increases above one third of a second, which is an acceptable amount of delay for the purposes of our demo application. While the latency may be noticeable by the user, it should not significantly affect the ability for the wheelchair user to record videos of themselves. In the worst case, the user may prematurely stop the recording a third of a second before the entire training routine is recorded, but under normal circumstances the last third of a second of any training routine is of little or no importance.

### 4.2 Framerate

The results from the framerate test are shown in Table 4. We see that for a resolution of 640 × 480, the framerate is around 15 fps. The higher the framerate, the smoother the appearance of moving entities in the video. If the framerate were insufficient, then fine nuances in the actions and movements of the wheelchair trainee may not be possible to perceive. Conversely, a very high framerate requires a large amount of bandwidth and has diminishing returns on visual clarity due to the ability of the human eye to perceive motion at a high framerate. The flicker fusion limit, which is generally accepted to be the minimum flicker rate beyond which the human eye is less likely to discern additional motion detail, is generally accepted to be approximately 15 Hz [12]. That is, a flickering light that turns on and off faster than 15 Hz will generally be perceived to be a steady light. This limit is intrinsically related to the acceptable framerate. Our experimental results show that our video should approximately meet this limit.

In our practical usage test, it is our subjective opinion that the resulting video is smooth enough to easily capture wheelchair operation.
Table 4: Test results for average number of frames per second for different settings of video resolution and frame compression quality. Due to fluctuations in wireless network bandwidth, the framerate can vary with an uncertainty of ±2 fps. For results labeled N/A, the application became unresponsive.

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Q=1</th>
<th>Q=25</th>
<th>Q=50</th>
<th>Q=75</th>
<th>Q=100</th>
</tr>
</thead>
<tbody>
<tr>
<td>320 × 240</td>
<td>30</td>
<td>30</td>
<td>29</td>
<td>29</td>
<td>25</td>
</tr>
<tr>
<td>640 × 480</td>
<td>16</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>N/A</td>
</tr>
<tr>
<td>800 × 480</td>
<td>16</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>N/A</td>
</tr>
<tr>
<td>1280 × 720</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### 4.3 Practical Usage Tests

From visual inspection, the videos were acceptably smooth and detailed. The average framerate was 15 fps, which is enough to capture motion at the speeds expected for a wheelchair. It was immediately clear to the viewer what the wheelchair user is doing, and nuances such as arm and wrist positioning could be easily discernible if the wheelchair is not more than 8 m away. Even in the narrow corridor in which this test was performed, the field of view was sufficient to capture all the manoeuvres that a wheelchair trainee is expected to perform.

![Image](corner.mov), ![Image](midrange.mov), and ![Image](straight.mov) respectively, in clockwise order.

In addition to being able to effectively perceive the details from the training routine, the video files were a reasonably small size (1.76 MiB, 3.89 MiB, and 2.67 MiB). This allows
them to be uploaded quickly. For example, an internet connection upload speed of 1 MB/s would require about 10-30 seconds to upload such video clips.

5 Conclusions

Wheelchair training can be costly and time-consuming, and in many places, difficult to access. EPICWheelS is an Android application that aims to provide a low-cost and easily accessible method of training wheelchair users in techniques and skills for safe and efficient wheelchair usage. Although EPICWheelS contains much of the functionality to teach skills to new wheelchair users, it is missing the ability for a user to easily record a video of themselves demonstrating new skills for evaluation by a remote wheelchair therapist. Our demo application has been designed to fill in that missing functionality.

By streaming video from one Android device’s camera over a local network to another Android device, our demo application allows the user to see and record themselves in near real time (less than 300 ms delay) as they perform new wheelchair skills. Running on a Samsung Galaxy S III phone, the demo application can display and record video with a resolution of $640 \times 480$ pixels at a frame rate of approximately 15 frames per second. Although it was designed for use with a specific Android IP Webcam application, the source code requires minimal modification to be used with any other MJPEG video stream. The simple interface of the demo application makes this functionality easy to use, even for users with little experience handling touch-interface devices.

If the functionality from our demo application is incorporated into the EPICWheelS application, EPICWheelS will be one step closer to achieving its goal of providing the benefits of working with a physical therapist to wheelchair users that don’t have direct access to rehabilitation training.

6 Project Deliverables

6.1 List of Deliverables

The following is a list of deliverables originally planned for this project:

- An addition to the application’s source code that allows video to be streamed in real time from an external camera and displayed on the tablet’s screen.
  
  - This has been changed to writing a standalone application that contains the video streaming and recording feature as a proof of concept. This is because the current version of EPICWheelS is slated to undergo trials and is therefore in feature freeze.

  - Our sponsor will receive this deliverable by checking out the source code located in the Mercurial repository on Bitbucket, and he will receive a document containing instructions for compiling and building the application from the source code.
• A commercially available camera capable of streaming video to the tablet.

  – Our solution uses a secondary Android device as the camera. There exist many such commercially available devices that can be used for this purpose. An IP camera application is required for the Android device to output a MJPEG stream. The Android online App Store is replete with free applications for this purpose; the one that has been tested for our project is called IP Webcam [6]. This application is also open source, so we can customise it for our specific use.

The next two deliverables were originally planned as extra objectives to be completed only if we have additional time. However, we have decided to focus on the first two deliverables instead.

• Modifications to the EPICWheelS application source code that implements an SQLite database to store user and file information locally, replacing the manifest files currently being used.

• A redesigned graphical user interface that improves usability, is more visually appealing, and delivers a better user experience.

### 6.2 Financial Summary

Refer to Table 5 for a list of components used in this project.

<table>
<thead>
<tr>
<th>Component</th>
<th>Quantity</th>
<th>Vendor</th>
<th>Total cost ($)</th>
<th>Purchased by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xoom Tablet</td>
<td>1</td>
<td>Motorola</td>
<td>approx. 350</td>
<td>Sponsor</td>
</tr>
<tr>
<td>Nexus 7 Tablet</td>
<td>1</td>
<td>Google</td>
<td>250</td>
<td>Sponsor</td>
</tr>
<tr>
<td>Nexus 7 Tablet</td>
<td>1</td>
<td>Google</td>
<td>250</td>
<td>Self</td>
</tr>
<tr>
<td>Galaxy S III</td>
<td>1</td>
<td>Samsung</td>
<td>N/A</td>
<td>Self</td>
</tr>
<tr>
<td>Galaxy S II</td>
<td>1</td>
<td>Samsung</td>
<td>N/A</td>
<td>Self</td>
</tr>
<tr>
<td>Xperia</td>
<td>1</td>
<td>Sony Ericsson</td>
<td>N/A</td>
<td>Self</td>
</tr>
<tr>
<td>Wheelchair</td>
<td>1</td>
<td>Unknown</td>
<td>N/A</td>
<td>Project Lab</td>
</tr>
<tr>
<td>WiFi Router</td>
<td>1</td>
<td>Various</td>
<td>N/A</td>
<td>Self</td>
</tr>
</tbody>
</table>

### 6.3 Ongoing Commitments by Team Members

Here is a list of current ongoing commitments by team members:

• Return all hardware to sponsor.

• Documentation of the demo application source code.

• Technical support for sponsor.
7 Recommendations

Future work may include merging the demo application source code into the full EPICWheelS application so as to integrate video streaming/recording capability into the final product. The voice mail sending capabilities can be augmented to include the ability to send video clips. A fork of the IP Webcam application can be created and customised for an easy and consistent user experience, for example, by automatically determining the default settings and displaying the IP address in a prominent fashion for the user to enter (see Section 3.1.1). Furthermore, the graphical user interface theme may be improved for aesthetic appeal and ease of use.

8 Appendix: FFmpeg build instructions

The following instructions have been tested on Linux and Mac OS X and should work on most UNIX-like operating systems.

- Download FFmpeg for Android from [11]:
  
  http://bambuser.com/r/opensource/ffmpeg-4f7d2fe-android-2011-03-07.tar.gz

  and decompress the downloaded file.

- In terminal, navigate to the decompressed file directory ffmpeg_android and run ./extract.sh.

- Download android-ndk-r5c (possibly r8 will also work, although it is untested) and decompress the downloaded file.

- In terminal, execute the following after replacing <PATH_TO_NDK> and <PATH_TO_ffmpeg_android> with their respective correct paths:

  $<PATH_TO_NDK>/build/tools/make-standalone-toolchain.sh --platform=android-3 --install-dir=<PATH_TO_ffmpeg_android>/toolchain
  $export PATH=/<PATH_TO_ffmpeg_android>/toolchain/bin:$PATH
  $export CC=arm-linux-androideabi-gcc

- In terminal, navigate to the ffmpeg folder in ffmpeg_android.

- In terminal, execute the following after ensuring that the file path for sysroot has been set correctly:

  --optimization-flags=-O2 --enable-encoder=mpeg2video --enable-encoder=nellymoser --enable-protocol=file --prefix=../build/ffmpeg/aromeabi --disable-doc --extra-cflags= --extra-ldflags=
$make
$make install

NOTE: com.example.demoapp corresponds to the specific package name specified in AndroidManifest.xml for the intended Android application. For example: in this case it is:

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android
    package="com.example.demoapp" android:versionCode="1" android:
    versionName="1.0" />
```

- Look for the following required files in `../build/ffmpeg/armeabi`:
  - ffmpeg
  - libavcodec.so
  - libavcodec.so.52
  - libavcodec.so.52.99.1
  - libavcore.so
  - libavcore.so.0
  - libavcore.so.0.16.0
  - libavdevice.so
  - libavdevice.so.52
  - libavdevice.so.52.2.2
  - libavfilter.so
  - libavfilter.so.1
  - libavfilter.so.1.69.0
  - libavformat.so
  - libavformat.so.52
  - libavformat.so.52.88.0
  - libavutil.so
  - libavutil.so.50
  - libavutil.so.50.34.0
  - libswscale.so
  - libswscale.so.0
  - libswscale.so.0.12.0

and place these files into the assets directory in your Android source code directory. Refer to Section 2.3.3 for usage details.
References


http://strokelink.ca
Accessed 22 September 2012

*IP Webcam.*
Accessed 22 September 2012

http://www.usb.org/developers/devclass_docs/
Accessed 22 September 2012

[8] *EZ-Robot shop.* EZ-Robot Inc.
http://www.ez-robot.com/shop/
Accessed 22 September 2012

http://www.newswireless.net/index.cfm/article/629
Accessed 22 September 2012

Accessed 22 September 2012
http://bambuser.com/opensource
Accessed 7 January 2013