Video conferencing system for Telerehabilitation

*(UROP Project)*

**

Yaadhav Raaj

A0081202

CEG Year 2

Jan 14, 2013

Table Of Contents

Overview 3

Introduction 3

At a Glance 3

What was done 3

User Interface 4

Introduction 4

iOS 4

Development 5

Use Case Scenario 5

Software Contributions 6

Introduction 6

Log-in Interface 6

CameraView 6

QueueFTP 7

VideoLoader 8

Others 8

Hardware Contributions 9

Introduction 9

WiFly 9

Xbee 9

Wi-Fi Car Project 9

# Overview

## Introduction

My name is Yaadhav Raaj and as of completion of my UROP (Undergraduate Research Oriented Programme), am a Year 2 Computer Engineering Student. Since I was interested in specializing in Embedded Systems (ES), I found this particular project to be quite relevant. Having had experience in basic microcontrollers such as PIC and Arduino, and having had experience in server-side development and iOS development through my experience in the iCreate 2012 Competition, I was hoping to hone, develop and apply my skills in this UROP.

## At a Glance

The main portion of this project was actually a major FYP headed by Prof. Yen Yi Sheng and Prof. Arthur Tay, and was called “Telerehabilitation.” It was on the development of a commercially usable electronic rehabilitation system; that was to aid physiotherapists in measuring body movements accurately in various physiotherapeutic exercises. It also involved the development of a software/user-interface that could be used by both patient and therapist. This software involved both the development of an app and a server backend.

An example would be where a therapist has to measure how far a patient was able to bend is forearm. With sensors attached to the patient’s forearm, the system would be able to track how far the arm has been bent, and would remotely send this information back to a server backend where the therapist is able to view. More details can be read in the use-case scenario later.

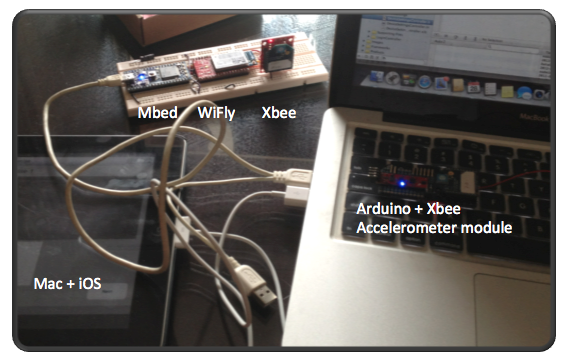
## What was done

When I joined the team in early August 2012, the hardware portion of the system had already been implemented and tested. Figure 2 is an image of a prototype set-up of the hardware interface.

An mbed processor with an onboard ARM chip was used as the main logic unit, and could be programmed via an online compiler. It also consisted of a WiFly chip that was capable of communicating with external devices via a TCP/UDP protocol and was used for communication with a PC. Lastly, it consisted of an Xbee wireless radio that used the high-speed Xbee protocol to communicate with the sensors. The sensors consisted of Arduino Fio modules with an Xbee wireless radio attached to them. The image shows a Mac and an iPad (iOS Device) but these were included later in the project.

The exact exercises to be conducted had been finalized by the therapist from NUH involved in the project as well. Figure 1 shows two exercises. The first one is “Shoulder Flexion” with a “Red Theraband” that provides resistance. In this example, it can be seen that two sensors are required; one in the arm and the other in the neck to ensure trunk rotation did not occur and affect the result.

There are several other exercises involving limbs, hips, shoulders and other functional activities and all of these protocols had already been decided as of joining.



# User Interface

## Introduction

Before joining, there already existed several user interfaces. There was one written for an Android phone and another java-based application running on Windows. However, most of these were crude and displayed raw data, such as angles sent from the sensors. There was no proper UI implementation/solution that could easily be used by Therapists and Patients, abstracting the unnecessary data.

## iOS

Before joining this project, the team was still undecided on which platform to choose for their flagship app. I managed to convince the team to choose iOS as their first platform of choice. These were several reasons why iOS was a better choice for this.

* Developing in iOS guranteed that the application would work seamlessly across all Apple devices. Devloping in Android would be cheaper, but would have resulted in fragmentation if the app was to be deployed across multiple Android devices. HTML5 would have ensure cross-platform compatability, but did not provided hardware access to the Camera as of iOS5 and could not open web sockets as easily as on a native platform
* iOS is still seen as a professional grade device, especially in the medical field. It may have been faster to start developing on Android since several members were familiar with Java, but developing on iOS is still considered niche, and forcing the team in the iOS development direction would ensure an iOS option for this system. An Android system could always easily be developed later since iOS developers are difficult to find.
* App deployment and management is done easily with iOS. Administrators could control which iOS device could have access to the app with various certificates and provisioning profiles. Furthermore, the App could easily be deployed on the Appstore if it becomes commercially viable
* HTML5 code can easily be accessed via a browser, and Anrdoid APK Packages can easily be decompiled to view the source code. This is because both platforms essentially run on a JIT (Just in time) compilation method, meaning the source code can be accessed. iOS apps are executed immediately and this ensure security.
* iOS has Facetime, which could be used to easily establish a video conference between therapist and patient.
* iOS gives direct access to both UDP and TCP Wireless protocols through Berkely C Sockets, ensuring access to low level code and full control over communication with the system.

These are some of the major reasons I stated as to why the team should proceed with iOS. After convincing the professors and the team, I ensured that the team became adept in iOS development and Objective-C, by providing assistance in coding/syntax issues, and certificate/profile issues. In this way, the transition to the iOS platform was rather seamless.

## Development

Multiple apps were to be developed, one for the Patient, and one for the Therapist. Also, a server backend was to be set up as well. The iOS apps were programmed in Objective-C/C++, the server backend in HTML/PHP and the hardware drivers in UNIX C. I was tasked to be involved in the development of the Patient app. Detailed below is the user case scenario in deciding what components were required for the app.

## Use Case Scenario

In order to understand the flow of the my part better, a detailed use case scenario is presented here to see what components are needed in the application:

|  |  |
| --- | --- |
| Patient needs to begin therapy, and opens the application. | Patient needs to be provided with the iOS device. Internet connection required. |
| Patient/Helper logs in with credentials | Server back end is required where patient credentials and information are stored. |
| Patient begins a Facetime conversation with Therapist | App automatically calls the respective Therapist. |
| Patient hits start to begin exercise and list of exercises are displayed | Exercises of the day need to be loaded. |
| Video preview of exercise is displayed | Videos of exercises being streamed required |
| Patient hits begin exercises and performs exercise | App communicates with hardware. Hardware drivers required. |
| iOS device takes a video of exercise being performed. | Video capture controller required |
| App shows that exercise is complete | Algorithms must make sense of data and tell user exercises are done. Algorithms required. |
| Video and data are uploaded | Upload controller required. |
| A graph is shown to the Patient at the end of all exercises | Graph controller is required. |

As seen from the UCS above, there were several requirements that needed to be fulfilled for my portion of the development of the Patient app.

# Software Contributions

## Introduction

Based on the above requirements, I begun to work on the various components.

## Log-in Interface

I wrote the initial login interface using traditional iOS4 methods. Settings were saved in .plist XML files and the storyboarding interface was not used. Pages were managed via pushing of view controllers. Page communication was done via pointers. However, these were later updated to iOS5 methods such as standardUserDefaults, NSNotification and seauge.

## CameraView

The app required a camera component to handle the recording of a video stream, while data was flowing in from the sensors. It also required the patient to be able to view himself on the camera. This was not possible using the traditional camera/recording API’s provided by Apple Documentation. Hence a custom solution needed to be implemented.

I implemented this solution using reference code provided by Apple, and packaged this into an easy to use class that can easily be dragged and dropped into the main storyboard. With a few function calls, video recording could begin asynchronously and could be stopped and saved. Quality control, image size and camera selection could also be controlled. Most importantly, the camera viewfinder could run as a separate window within the app.

The implementation of this can be found inside ExerciseController/camera. The API reference can be read below:

Setting Up:

* Add the following line of code to your main .h file which is linked to your storyboard
  + #import "CameraViewController.h"
* Drag and drop a UIView object onto your storyboard and adjust the dimensions (320\*240)
* Link this object with a UIView getter/setter
* @property(nonatomic,strong)IBOutlet UIView\* cameraRecording;
* @synthesize cameraRecording;
* Create a Camera object and link it with the UIView
* cam = [[CameraView alloc] init];
* [cameraRecording addSubview:cam.view];

Functions:

|  |  |
| --- | --- |
| (IBAction)toggleRecording | Start/Stop recording depending on state |
| (void)startRecording | Start Recording |
| (void)stopRecording | Stop Recording |
| (IBAction)captureStillImage | Capture an Image |

## QueueFTP

The app also required upload of large video files from the app to the server, where the therapist could then access and view the patient’s exercise. Uploading of raw data could be done via simple HTTP Post commands with the included NSURLConnection API. However, in order to upload the video file without corruption, an FTP connection was required to upload the file.

Unfortunately, the example solution provided by Apple Documentation only seemed to execute on the main thread of the app. This meant that while uploading, nothing else would be able to run on the app. Due to this issue, multiple 3rd party solutions were available online, such as SimpleFTPExample or other libraries. Unfortunately, most of these were only able to upload one file at a time, and used the CFNetwork framework to directly open a connection from within the app, resulting in several crashes and memory leaks.

To counter this, I devised a method where the data would be uploaded in packets through a FTP PHP script. This is similar to how files are uploaded in the browser. I then used the ASIHTTP framework to create a class, that would allow the input of FTP Parameters such as username/password, and have this object be added to a network queue. This way, the user could easily add large files to the queue and these would be asynchronously uploaded to the server. Also, errors could be handled with ease, depending on the response from the PHP script. Upload progress could also be tracked.

Usage

* Start my adding the following dependancy frameworks (CFNetwork) (SystemConfiguration) (MobileCoreServices) (Security) (libz.dylib) to your project
* Add ASIHTTPRequest framework by hitting other and selecting the framework attached above
* Drag QueueFTP.h and QueueFTP.m files into your project
* Add the PHP script to your server
* Start using! That easy!

Example Usage

QueueFTP**\*** ftp;

QueueFTPSettings**\*** ftpSettings;

ftpSettings**=**[[QueueFTPSettings alloc]init:@"phplink" **:**@"ftplink" **:**@"username" **:**@"password"];

ftp**=**[[QueueFTP alloc]init:ftpSettings];

[ftp setDelegate:self];

[ftp addFile:@"file1" **:**@"ftppath"];

[ftp addFile:@"file2" **:**@"ftppath"];

[ftp beginExecution];

Github Link

<https://github.com/soulslicer/QueueFTP>

More information regarding this class can be found here. However, it has been modified slightly in the existing version of the app.

## VideoLoader

The app required a preview video of the exercises to be playing in the background as well. It would start by first checking if the video was stored locally, if not, it would start streaming and downloading the video onto the iOS device. It would then load locally if that particular exercise is selected again. This file can be found in ExerciseController/video.

Setting Up:

* Add the following line of code to your main .h file which is linked to your storyboard
  + #import "VideoLoader.h"
* Drag and drop a UIView object onto your storyboard and adjust the dimensions (320\*240)
* Link this object with a UIView getter/setter
* @property(nonatomic,strong)IBOutlet UIView\* videoDisplay;
* @synthesize videoDisplay;
* Create a Camera object and link it with the UIView
* videoLoader=[[VideoLoader alloc]initWithID:CURRENT\_EX\_ID];
* [videoDisplay addSubview:videoLoader.view];

Functions:

|  |  |
| --- | --- |
| (id)initWithID:(EXERCISEID)exid; | Loads the respective Video |

## Others

Apart from those software elements, I also helped to set up the iOS development environment for my project team-mates, tested the Facetime video conferencing between the iOS devices and Macs, and helped to set-up the server scripts for uploading of files

# Hardware Contributions

## Introduction

The hardware portion of this project had more or less already been built, nevertheless, in order to get an in-depth understanding of the project, I decided to delve into how the hardware and software communicated during the December vacation period. Through this, I now have knowledge of how the driver API’s are able to communicate with the hardware.

## WiFly

I now have a clear understanding of how Berekely C Sockets are used to communicate with the WiFly device on the system using the UDP Protocol. I also learnt the differences between TCP and UDP protocols, as the driver had implementations of both. TCP was slower than UDP as it required the opening of a connection between the two platforms, and also did error checking through the checksum value. This however does ensure data integrity. UDP on the other hand is used for more time critical applications but data integrity cannot be ensured. Since the data from the sensors had to sent and read instantly, the UDP protocol was used.

## Xbee

I also read up on how Xbee communication occurred through the IEEE 802.15.4 networking protocol, and how it was used as master and slave nodes in the project. I also understood how the mbed processor had to swap between quering the multiple sensors.

## Wi-Fi Car Project

In order to ensure that I could apply what I learnt, I also spent the holidays working on my own project, particularly a WiFi Car that could be controlled via an iPhone. This project was a success, and I learnt how a similar implementation was done on the driver used in the app.

The github repository of the project can be found here:

<https://github.com/soulslicer/Wi-Fi-Car-Project>