The University of British Columbia

Department of Electrical and Computer Engineering

EECE 496 Final Report

Interface Engineering of DIVAs

Ying Yin

#75821041

Technical Supervisor: Dr. Sidney Fels

04/11/2008

ABSTRACT

This project investigates the interface engineering of DIVAs which stands for DIgital Ventriloquized Actors. The DIVAs project investigates the use of hand gestures to synthesize audiovisual speech and songs. The efficiency of data streaming from the CyberGlove and the Polhemus tracker is critical for the whole system to work well. Hence, both the hardware and the software interfaces need to be re-engineered in order to streamline the dataflow, and to improve performance. Two external MAX/MSP objects written in C are created for getting streamed data from the CyberGlove and the Polhemus tracker. Both non-threaded and threaded versions are created, but the threaded versions are preferred. The objects are tested separately using MAX/MSP patches, and are verified for the correct functionality. The next steps are to further improve the robustness of the objects, and to integrate them with the rest of the system.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

FIGURES

TABLES

1.0 INTRODUCTION

This project investigates the interface engineering of DIVAs which stands for DIgital Ventriloquized Actors. The DIVAs project is an ongoing research project under Media and Graphics Interdisciplinary Centre (MAGIC) at the University of British Columbia. It investigates the use of hand gestures to synthesize audiovisual speech and songs. The DIVAs project has many aspects, including face synthesis, speech synthesis, adaptive control, and DIVAs interface design. My part of the project focuses on the interface design and engineering of the DIVAs prototype.

The objective of the interface engineering of the DIVAs prototype is to make it highly modulized, easy to use, and reliable. The existing interface of the prototype is based on MAX/MSP which is a graphical programming environment for music and audio. While MAX/MSP provides good interfaces for audio synthesis, it is not very suitable for efficient data streaming. However, the efficiency of data streaming from the CyberGlove and the Polhemus tracker is critical for the whole system to work well. Hence, both the hardware and the software interfaces need to be re-engineered in order to streamline the dataflow, and to improve performance.

For this project, two MAX/MSP external objects are created to handle data streaming from the CyberGlove and the Polhemus tracker. The existing MAX/MSP patch includes both the external objects and the MAX/MSP's own serial objects to get data from the hardware devices. The graphical code is rather cluttered and hard to maintain. The new objects are completely written in C for maximum efficiency, and encapsulate serial communication within them. They also use threads for data streaming. These features make the data streaming more efficient, the interface cleaner, and the code easier to maintain.

Correctness, reliability, and efficiency of data capturing are the important parameters for making the MAX/MSP external hardware interfacing objects. This report discusses how the objects are designed and implemented to meet these goals. The report divides into the following primary sections: serial communication, the CyberGlove object, the Polhemus tracker object, testing and results, and conclusion.

2.0 SERIAL COMMUNICATION

In order to encapsulate the data streaming task into one external object, the serial communication needs to be handled internally in the object instead of using the MAX/MSP serial object. In this way, the MAX/MSP code will be simpler and the data streaming will be more efficient. As serial communication is needed in both the CyberGlove object the Polhemus tracker object, a serial port communication library is created so that it can be included in the code of both external objects.

The current DIVAs system is developed for running on Mac computers which has Linux as the underlying operating system. As a result, POSIX API is used for serial port communication. There are three major parts of the serial port library created: serial port initialization and closing, sending command, and receiving data.

2.1 Serial Port Initialization and Closing

The initSerialPort(char*) call takes an argument of the full path name of the serial port. During the initialization, a serial port is opened for read and write (O_RDWR), with no controlling terminal (O_NOCTTY) and non-blocking I/O (O_NONBLOCK). As open() follows POSIX semantics: multiple open() calls to the same file will succeed, the TIOCEXCL ioctl is issued to prevent additional opens except by root-owned processes.

The attribute of the serial port (struct termios *newAtt*) is then set to the raw input (non-canonical) mode by calling cfmakeraw(&*newAtt*). For both the CyberGlove and the Polhemus tracker, the serial port settings are: 8 bits, no parity, and 38400 baud rate. Eight bits is the default setting for the raw input mode. The rest is achieved by the following code:

```
newAtt.c_cflag &= ~(CSTOPB);
cfsetispeed(&newAtt,B38400);
cfsetospeed(&newAtt,B38400).
```

During the data streaming, the I/O should be blocking to ensure efficiency. The function fcntl(*rdPort*, F_SETFL, 0) can be called to clear the O_NONBLOCKING flag, and the subsequent I/O will be blocking.

In the closeSerialPort( ) function, the original port attributes are restored, and then the port is closed.

2.2 Sending Command

Both hardware devices respond to ASCII commands. The sendCommand(cosnt Char*) function sends the command through serial port. The following is the code segment:

```
void sendCommand(const char* command)
{
        int numBytes;
        numBytes = write(wrPort, command, strlen(command));
}
```

2.3 Receiving Data

Two types of data receiving are provided: one is for non-blocking read and one is for blocking read. The non-blocking read, readNonBlock(char* *buf*, int *numBytes*) returns whatever number of bytes available in the serial buffer. The maximum number of bytes returned is *numBytes*. The blocking read, readBlock(char* *buf*, int *numBytes*) returns the exact number of bytes (*numBytes*) requested by the user. The following is the code segment for the readBlock() function:

```
int readBlock(char* buf, int numBytes)
{
        int numBytesRead=0,totalBytesRead = 0;
        while( totalBytesRead < numBytes )
        {
                if ( (numBytesRead = read( rdPort, (buf+totalBytesRead),
                        numBytes - totalBytesRead ) ) < 0 )
                        return -1;
```

```
            totalBytesRead += numBytesRead;
        }
        return totalBytesRead;
}
```

3.0 CYBERGLOVE OBJECT

The wireless CyberGlove is a motion capture data glove with18 high-accuracy joint-angle measurements. It uses proprietary resistive bend-sensing technology to accurately transform hand and finger motions into real-time digital joint-angle data [1]. The external MAX/MSP object for the CyberGlove is for streaming the joint-angle data of the fingers. This object can replace three existing objects in the MAX/MSP code. The CyberGlove communicates with the computer through Bluetooth which has a corresponding virtual serial port for data transmission. The object has one inlet which takes four messages: bang, “init”, “stop”, and “port”. It also has two outlets. The right outlet outputs the list of 18 values from the glove, and the left outlet outputs a bang. The bang message is intended to output the status of the glove; however, it is not used currently.

In the C code, the glove object has the following type definition:

```
typedef struct _glove
{
        t_object m_ob;
        long m_valueout[18]; // holding 18 values for output
        void *m_out1; //right outlet
        void *m_out2;//left outlet
        void *m_systhread; //reference to the thread
} t_glove;
```

3.1 “init” Message

The “init” message is used to initialize the serial port and the device. The serial port library function initSerialPort() is called passing the full path of the virtual serial port for the Bluetooth device. Other global variables are also re-initialized.

3.2 Bang Message and Data Streaming

Each bang message causes the list of 18 values from the glove to be outputted from the outlet (see Appendix A for the sensor data byte ordering of the 18 values). Each value is a byte from 0 to 255. The method glove_bang() is called when the object receives a bang message. If data streaming has not started, the command "S" is sent using the sendCommand() call to start streaming. The capitalized "S" command causes the data returned from the glove to be in the binary format. A thread is created to handle the receiving of the data from the serial port. The thread routine is void* glove_threadproc(t_glove *glove).

It is important to use a thread because the read is blocking. To ensure thread safety, MAX/MSP's thread API is used. The function systhread_create() is used to create and start a thread.

The thread contains an infinite loop that runs as long as the global variable g_started is true. Data is read from the serial port one frame (20 bytes) a time, and the 18 byte values are extracted. The is much more efficient than the original code which reads data one byte a time from the MAX/MSP serial object, and uses a state machine to parse the data. The frame format from the glove is [1] :

Binary Response

'S', s1, s2,…, s18, ['es'], null.

Under normal condition, twenty bytes are returned in a frame. If the actual sampling cannot meet the set sample period, an 'es' error code is returned before the trailing null [1].

The thread has a pointer to the glove object called **glove*. After obtaining one set of 18 values, the data is stored in *glove->m_valueout*. To ensure thread safety, it is important that the values are not directly outputted through the outlet in the thread. The outlet calls is only made in the main thread when there is a bang message to the object. Both the main thread and the thread reading the streaming data access the variable *glove-*

>*m_valueout*. Hence, the variable is a critical section, and should be protected by a lock. The MAX/MSP API calls for this are critical_enter(int) and critical_exit(int) [6].

3.3 "stop" Message

The "stop" message is used to stop the data streaming, and close the serial port. The command "^c" is used to stop the streaming of data. In addition, systhread_join() is called for the main thread to wait for the child thread to finish.

3.4 "port" Message

The "port" message is used to specify the serial port name of the device. As different computers may assign different port names, the name should be dynamically selected by the user. Only the name, not the full path of the port, is needed (for example, "WCGII-3017-WCGIIserialp-1" is the port name for the CyberGlove on the Mac mini).

Figure 1shows the screenshot of the MAX/MSP patch created for testing the CyberGlove object. A metro object is used to send a bang message every 10ms to output the data from the glove. When compared to the original patch for the same task (Figure 3 in Appendix B), it is clear that the new external object provides a better abstraction of the CyberGlove. The original code requires three main objects along with other supporting objects. The new MAX/MSP patch is much cleaner, and easier to understand and maintain.

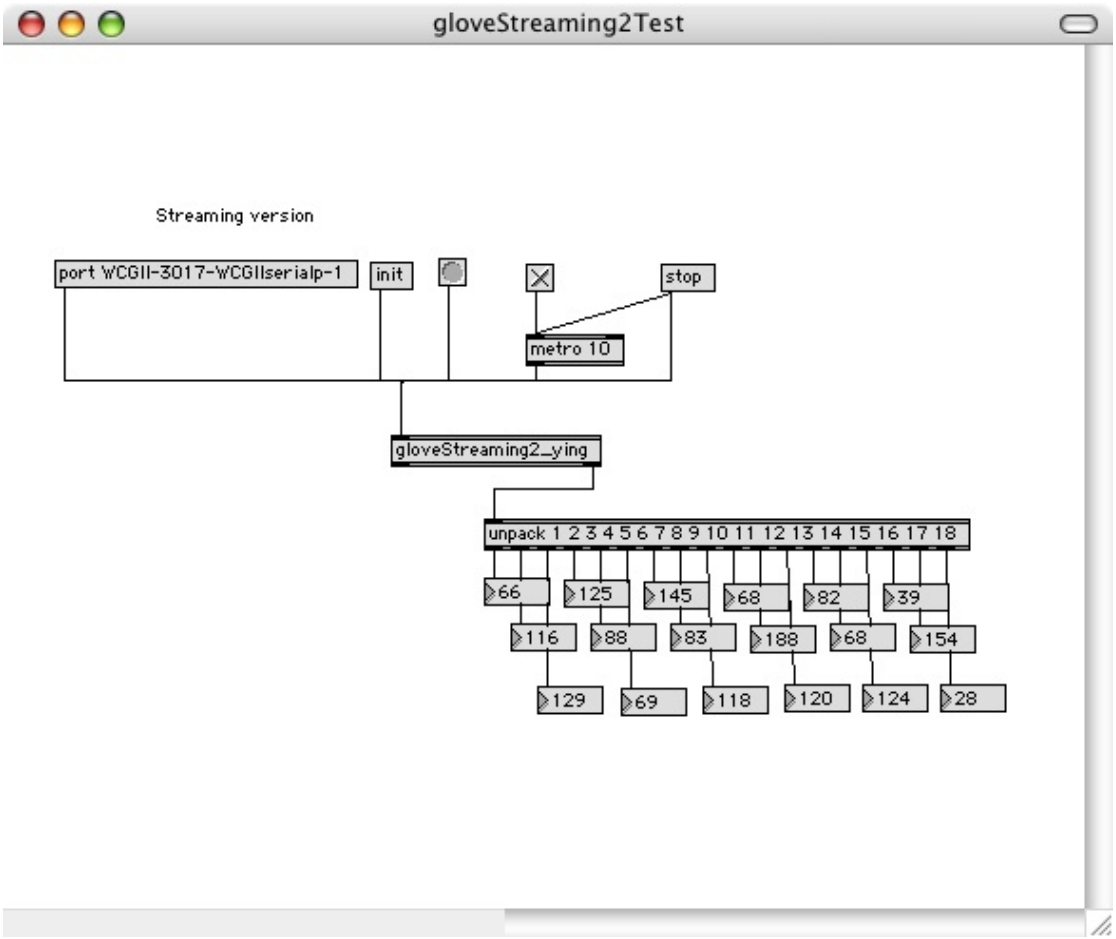

Figure 1 Screenshot of the CyberGlove object testing patch

4.0 POLHEMUS TRACKER OBJECT

Polhemus Patriot is a 6 degree-of-freedom tracking device. It is an expandable system that can accommodate up to two sensors [2]. The Polhemus tracker object is an external MAX/MSP object written in C for getting data from Polhemus Patriot. The object displays the current X, Y and Z coordinates of the hand position, as well as the pitch, roll, and yaw data of the hand, (a list of 6 values) as floating point values. The interface is much cleaner now because only one MAX/MSP object is needed instead of three in the original code.

The object has one inlet and two outlets. The inlet takes 5 messages: bang, “init”, “stop”, “H”, and “port”. As the Polhemus Patriot tracker has two stations, the outlets output the lists of 6 values for the two stations correspondingly. The left outlet outputs the data for Station 1, and the right one outputs the data for Station 2. In the C code, the tracker object has the following type definition:

```
typedef struct _tracker
{
 t_object m_ob;
 float m_dataPacket[4][6];  //holding 6 values for potentially 4 sensors
  void *m_out1; //right outlet
 void *m_out2; //left outlet
 void *m_systhread; //reference to the thread
} t_tracker;
```

4.1 “init” Message

The “init” message is used to initialize the serial port and the tracker object. The Polhemus Patriot tracker is connected to the computer using a USB cable. In the “/dev” directory, a corresponding USB-serial port is created for the USB port, and standard serial communication can be used.

The tracker object is more complicated than the glove object, and the hardware requires more initialization configuration. The following steps are done for initializing and setting the correct mode for the tracker hardware [5].

1) Re-initialize the tracker. Command: "^Y\r" (ctrl-Y and carriage return).
2) Set binary output mode. Command: "f1\r".
3) Set operational hemisphere. Command: "H*station*, *p1*,*p2*,*p3*\r".
4) Set baud rate to 38400, none parity. Command: "^O384,0\r".

For the 'H' command, *station* specifies the relevant source/sensor pair; *p1*, *p2*, and *p3* represent the initial X, Y, and Z components of a vector pointing in the direction of the operational hemisphere's zenith [5].

4.2 Bang Message and Data Streaming

The function of the bang message is similar to that of the CyberGlove object. However, there are still some differences. The data streaming command for the tracker is "c\r". The data receiving and parsing process in the thread is also different.

In binary mode, every data frame from the tracker contains an 8-byte header [5]. Table 1 shows the header format of the response data frame. When reading the data, the header is read first to find the number of bytes in the body. Then the body is read. To ensure that there is no error, the frame tag is checked, and the error indicator is verified to be 0x00 or 0x20 (These two error codes mean no error).

	Byte Index	**Type**	**Description**
HEADER	0,1	unsigned short	Frame Tag, always 'PA' for PATRIOT
	2	unsigned char	Station Number
	3	unsigned char	Initiating Command
	4	unsigned char	Error Indicator
	5	unsigned char	Reserved
	6,7	signed short	Response Size; number of bytes in the response body
	8-n		Binary Response Body

Table 1 Binary response format

For data streaming, the response body has 26 bytes: 4 bytes each for X, Y, Z coordinates and pitch, roll, and yaw values; the body is terminated by a carriage return (0x09) and a line feed (0x0A). The positional and rotational values are 32-bit single-precision floating-point numbers in IEEE format [5].

It is important to note that Mac is “big endian” while the data from the tracker is “little endian”. In the code, the macro:

```
#if def __BIG_ENDIAN__
```

is used to check the byte ordering of the operating system so that each four bytes can be converted to a float number correctly.

Although there is an error code in the header for checking data validity, testing shows that checking the error code is not sufficient because the header may be corrupted too. To increase fault tolerance of the system, additional checking is made to ensure reliability and robustness. When the tracker is just initialized, there may be junk data left in the hardware’s output buffer. As a result, at the beginning, the data are read one byte by one byte until the two consecutive bytes ‘PA’ are read. In this case, it is assumed that a header is found, and the initiating command byte and the error code are checked. If the header is valid, the rest of body is read, and a flag is set to indicate that the following data can be read in frames. If an error is encountered, the flag is unset, and data are read one byte a time again.

Storing and outputting the data are similar to that of the glove object.

4.3 “stop” Message

The function of the “stop” message is also similar to that of the CyberGlove object. The only thing different is that the stop streaming command for the tracker object is “p”.

4.4 “H” and “port” Messages

The “H” message is for users to dynamically set the hemisphere of operation through the MAX/MSP interface. It has four arguments, and the format of the message is similar to the actual command for the Polhemus tracker: “H *station p1 p2 p3*” (see Figure 2). The “port” message is very similar to that of the glove object.

A testing patch for the tracker object is also created. The interface is very similar to that of the glove object. The metro object has the same function of sending bang messages to the object for outputting data. Comparing with the original MAX/MSP patch (see Figure 4) for achieving the same task, the new one is better modulized. In addition, the original code parses the data bytes one by one using a state machine which is not very efficient. The new code parses the data by frames.

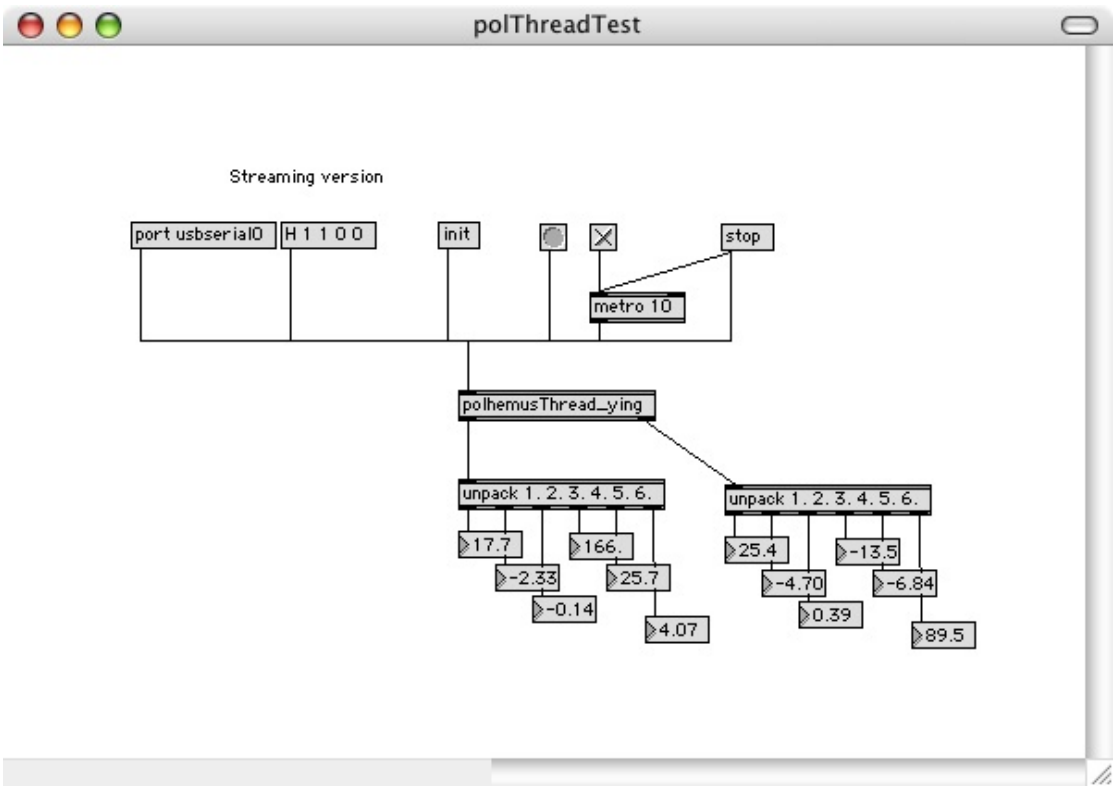

Figure 2 Screenshot of the Polhemus tracker object testing patch

5.0 TESTING AND RESULTS

As mentioned earlier, separate testing patches are created to test the correct functionality of the two objects. For both objects, a non-threaded version was also created at the beginning. The non-threaded version uses non-blocking I/O. As a result, reading from the serial port is only best effort, and the actual number of bytes to be read cannot be specified exactly. Due to this reason, the process of parsing the data is less efficient. However, they can still be useful for comparing the results and performance with the threaded versions.

5.1 Testing of Correctness and Performance

The two objects are tested to ensure both correctness and good performance. To test whether the output data is correct or not for the Polhemus tracker object, the sensor is first placed near the source, and then moved along the X direction. The displayed value is checked against the measured value allowing some small systematic errors due to the imperfection of the hardware and the presence of other magnetic fields and iron objects. The procedure is repeated to check the Y and Z values. The pitch, roll and yaw values are not checked because they are not used by the rest of the system.

The correctness of the glove object is tested in a similar fashion. The tester wears the glove, and bends one finger holding other fingers stationary. The corresponding output (see Appendix A) values are then checked.

To test the performance of the tracker object, the sensor is moved quickly in an arbitrary direction, and then stopped abruptly. The delay for the displayed output to reach the final values is monitored. The performance of the glove object is tested in the similar way.

It is found that the response of the glove is very good which means that once the fingers stop moving, the displayed output reaches the final values almost immediately too. However, for the tracker object, there is an observed delay for the displayed output to reach the final values of the sensor position when the movement is fast and over a relatively long distance. This may due to the fact that when the floating point numbers are displayed through MAX/MSP at high frequency, there is extra latency introduced.

5.2 Results and Known Bugs

The output values of both objects are tested to be correct. However, for the tracker object, the system does not respond when the tracker hardware is first powered up. After clicking “stop” and “init” again, it works fine. Subsequent re-initializations also work fine. The irresponsiveness of the system is caused by an “Invalid Command” error. A possible reason is that the software re-initialization command “^Y” is sent to the hardware every time when the “init” message is clicked. This may be an invalid command when the hardware is just powered up, and is already in an initialized state. This needs to be verified by more testing, and the problem needs to be rectified to ensure robustness of the system.

6.0 CONCLUSION

This project investigated the interface engineering of DIVAs. The two MAX/MSP external objects, the CyberGlove object and the Polhemus tracker object, are created and tested. They function relatively well for the data streaming tasks. These new objects are more modulized, and hence, they are easier to use and maintain. They are also more efficient for data streaming because the use of threads and better parsing algorithms. This also proves that it is possible to write more sophisticated external MAX/MSP objects for customized functionality and better performance.

Further work required include: (1) improving the robustness of the tracker object; (2) integrating both objects with the rest of the system for testing; (3) extending the Polhemus Patriot tracker object to work with Polhemus FASTRAK (another version of the tracker which has four output stations); (4) making other MAX/MSP external objects for scaling and voice mapping.

REFERENCES

[1] "Wireless Data Glove: The CyberGlove® II System." Internet: http://www.immersion.com/3d/products/cyber_glove.php, [Apr. 10, 2008].

[2] "PATRIOT." Internet: http://www.polhemus.com/?page=Motion_Patriot, [Apr. 10, 2008].

[3] Virtual Technologies, Inc. "CyberGlove Reference Manual." 1998.

[4] MAXMSP. "Event Priority in Max (Scheduler vs. Queue)," http://www.cycling74.com/story/2005/5/2/133649/9742, Sep. 09, 2004 [Apr. 08, 2008].

[5] Polhemus. (2004, Nov.). "Patriot User Manual." [On-line]. Available: http://www.polhemus.com/polhemus_editor/assets/PATRIOT%20Manual%20Rev%20E.pdf [Mar 10, 2008].

[6] Cycling '74. "Writing Externals for Max and Msp." 2005.

APPENDIX A GLOVE SENSOR DATA ORDERING

Byte Index	Sensor Name (Description)
0.	thumb rotation (angle of thumb rotating across palm)
1.	thumb MPJ (joint where the thumb meets the palm)
2.	thumb IJ (outer thumb joint)
3.	thumb abduction (angle between thumb and index finger)
4.	index MPJ (joint where the index meets the palm)
5.	index PIJ (joint second from finger tip)
6.	middle MPJ
7.	middle PIJ
8.	middle-index abduction (angle between middle and index fingers)
9.	ring MPJ
10.	ring PIJ
11.	ring middle abduction (angle between ring and middle fingers)
12.	pinkie MPJ
13.	pinkie PIJ
14.	pinkie ring abduction (angle between pinkie and ring fingers)
15.	palm arch (causes pinkie to rotate across palm)
16.	wrist pitch (flexion/extension)
17.	wrist yaw (abduction/adduction)

MPJ = Metacarpophalangeal Joint
PIJ = Proximal Interphalangeal Joint
IJ = Interphalangeal Joint

APPENDIX B SCREENSHOTS

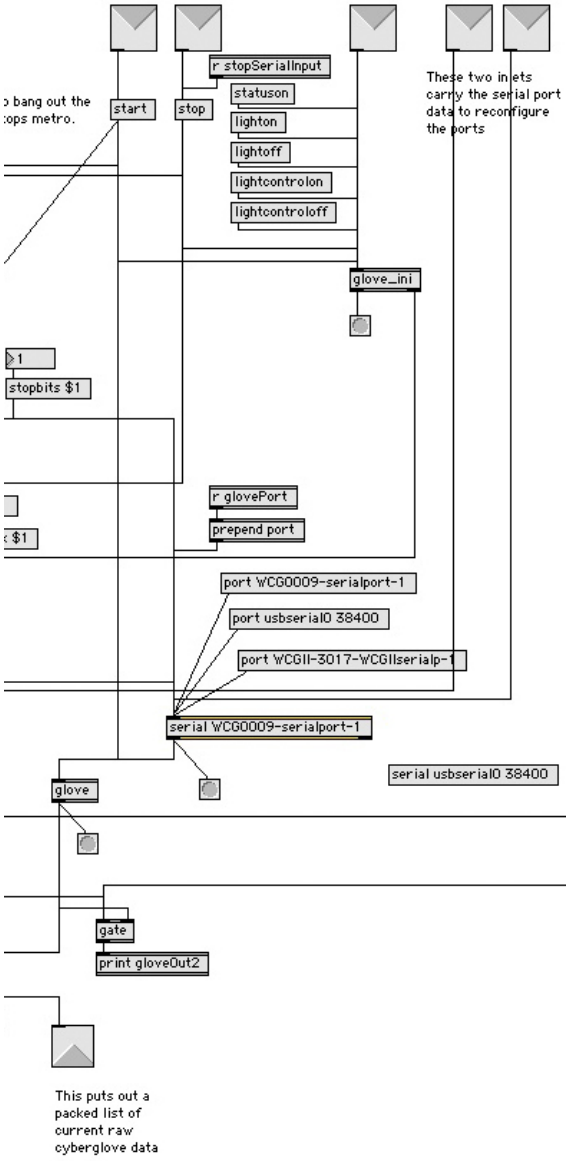

Figure 3 Original patch for getting data from the glove

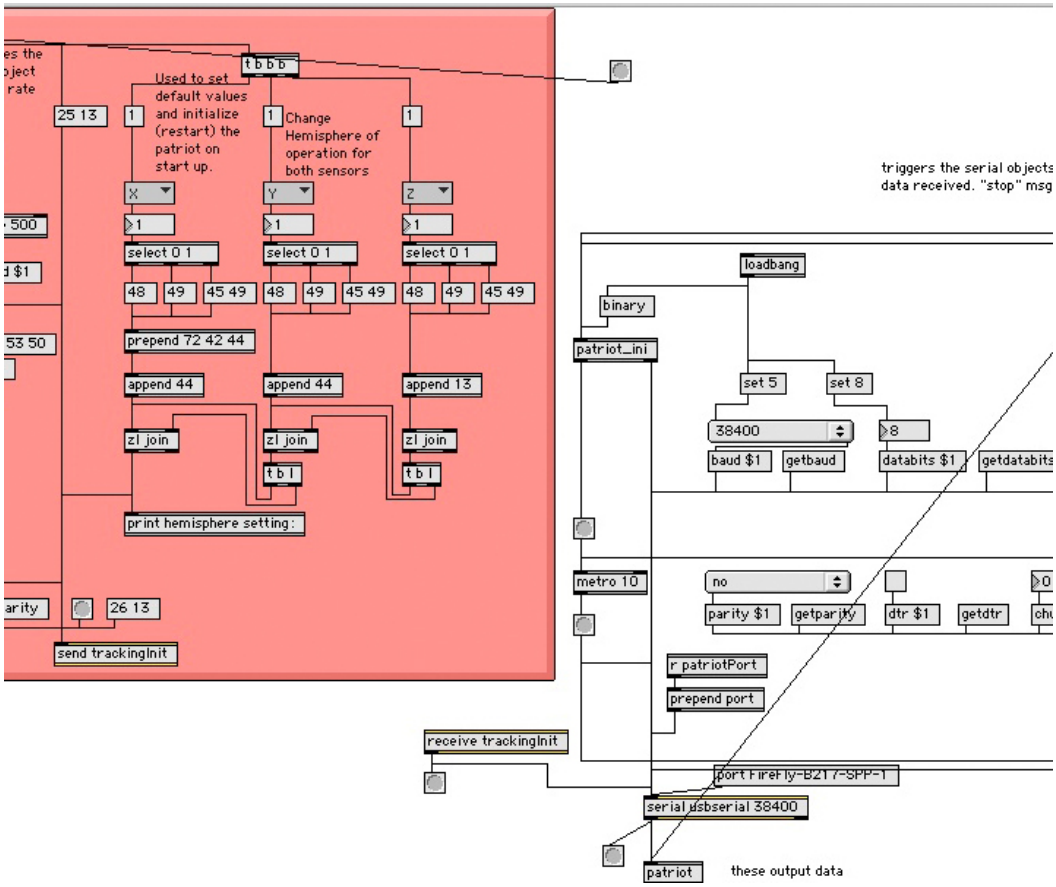

Figure 4 Original patch for getting data from the tracker