Project M1: Real-time neural control of cadaveric and robotic limbs

Jason J. Kutch

November 11, 2011

Contents

Chapter 1

Acknowldegements

This project would not be possible without the generous help of Manish U. Kurse.

Chapter 2

Introduction

This project is intended to provide a very general platform for real-time control of robotic and cadaveric limbs, to further our understanding of how neural circuitry controls many muscles simultaneously to perform useful tasks with the body, and how this control breaks down in disease. The M1 project is intended to give the user a "playground" to implement extremely sophisticated controllers, while reliving them of the burden of writing low-level hardware drivers, and worrying about sensors and real-time execution. The following documentation is a guide to this process, and you should contact me (Jason Kutch, kutch@usc.edu) with questions, concerns, bugs, etc.

Chapter 3

Hardware

Figure 3.3 shows the M1 system controlling a fresh cadaver hand to produce movements of the index finger. At present, this is considered the ultimate use of the M1 system, to use sophisticated algorithms to actuate the tendons of actual human tissue. This chapter describes all of the components of the M1 system that contribute to making this control possible.

3.1 BBDLSynthetic

3.1.1 General

The computer on which code is written and commands are deployed is called BBDLSynthetic. It is named this because, in this lab, synthetic systems consisting of either cadaveric or robotic hands are constructed. It is a Dell Precision T1600 Desktop (Figure 3.2). The username is "bbdlMember" and the password is "bbdl". The "Z:" drive of this computer is mapped to the BBDL central data server, called "interosseous" (interosseous.usc.edu), to facilitate data transfer and storage.

3.1.2 Communication

BBDLSynthetic is currently connected to the outside world through a Cisco Linksys E2000 router. This router provides communication to the internet, to the Brachioplex real-time system, to the Vicon Motion Capture system, and to the Adept robot (all described below). This router can assign static IP addresses to its clients using a control panel accessed from a browser:

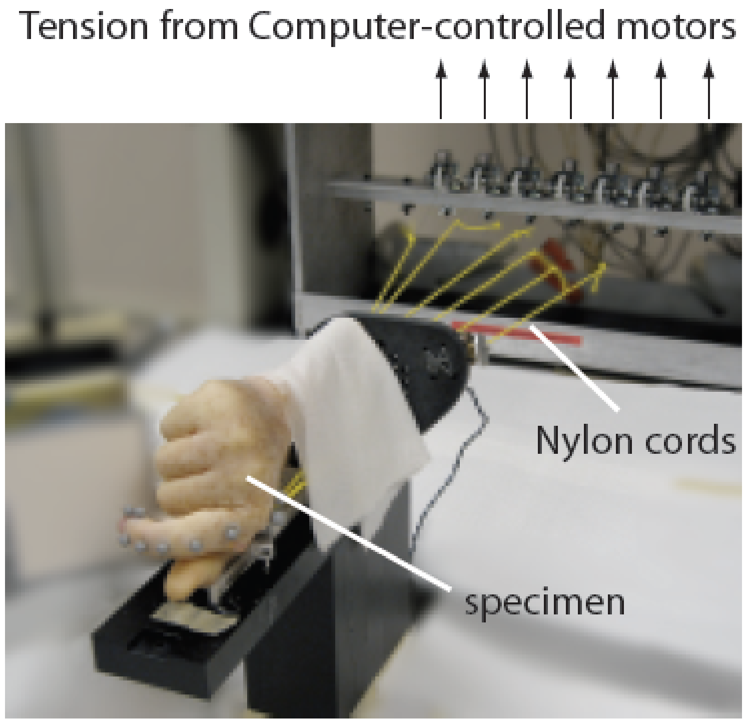


Figure 3.1: The M1 system controlling the ultimate piece of hardware, the human hand.

Figure 3.2: BBDLSynthetic. The desktop on which code is written and deployed.

1. Open a browser, and type 192.168.10.254 into the address bar.
2. The username and password will likely be autosaved, but they are both "bbdlMember"
3. Under DHCP server setting, click "DHCP reservation".
4. Here you can associate particular IP addresses with device MAC addresses.

3.1.3 Relevant Software on BBDLSynthetic

BBDL synthetic has several pieces of software that work together to make the control work. All of these can be found in the Windows task bar.

Command prompt

Useful for making sure that communication has been established with various pieces of hardware. For example, to make sure that BBDLSynthetic is talking to the Brachioplex system, type

```
C:\Users\Whatever:> ping 192.168.10.100
```

You should ensure that all packets go through without any loss. This indicates a good connection with Brachioplex.

Measurement and Automation Explorer (MAX)

MAX is a general piece of National Instruments software for managing both local (PCI card) and remote (PXI) systems. You can use it to set up data acquisition tasks, check that channels are operational, and get pin out sheets specific for the hardware in the lab.

Labview

As described below, Labview is used to control and monitor the experiments. However, it is simply used as a front end, with very little LabView programming required of the user. The bulk of the user programming is in C, within the CVI environment.

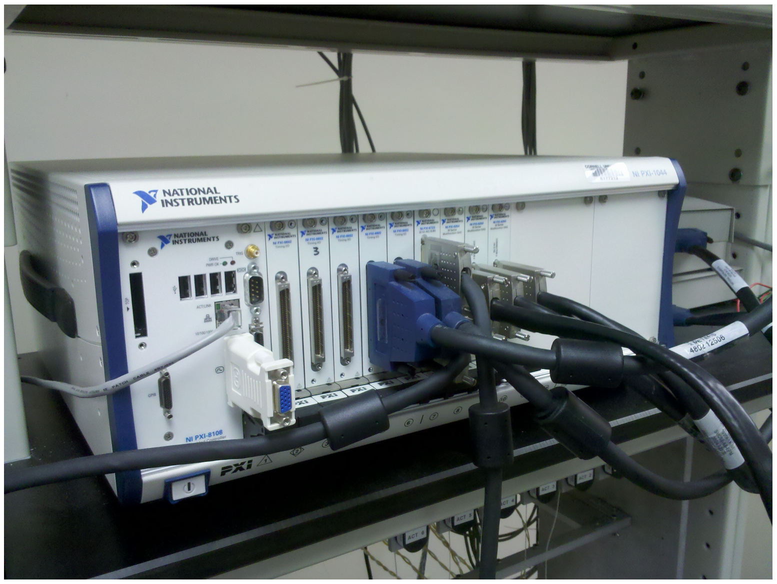


Figure 3.3: The Brachioplex Real-Time PXI system from National Instruments.

CVI

CVI is an integrated development environment provided by National Instruments, intended as a means of writing entire labview programs in C. However, it has an even more useful purpose that we are exploiting: it can be used to generate Dynamic Link Libraries (DLL) that can be executed on the real-time Brachioplex system. This DLL is referred to as the "Doer" below.

MATLAB

MATLAB is the interface to the Databaser, which is used to log and analyze data as soon it is available.

3.2 Brachioplex - the real-time PXI system

Brachioplex is our pet-name for the PXI system that performs the real-time data acquisition and control. It consists of chassis, a microcontroller (PXI-8108), and a series of slots with data acquisition cards installed. These cards communicate with the motors, tendon load cells, tendon encoders, as well as endpoint load cells such as the JR3.

When MAX is opened on BBDLSynthetic, there are two types of systems in the configuration panel on the left. “My System” refers to data acquisition cards installed in BBDLSynthetic, of which there are currently none. “Remote Systems” can be expanded to reveal “Brachioplex”. Expanding “Data Neighborhood” reveals “NI-DAQmx Tasks”. Expanding this list of tasks reveals all of the data acquisition tasks stored on Brachioplex. Some of these tasks are used in the M1 project, including “Write_servo_command”, but these tasks do not have to be manipulated by the day-to-day user of the system. Contact kutch@usc.edu for more information about these tasks.

Expanding “Devices and Interfaces” will reveal all of the information about the physical hardware in the Brachiplex chassis. This can be used to navigate to the pin-out sheets, which are useful when wiring new devices into the existing hardware.

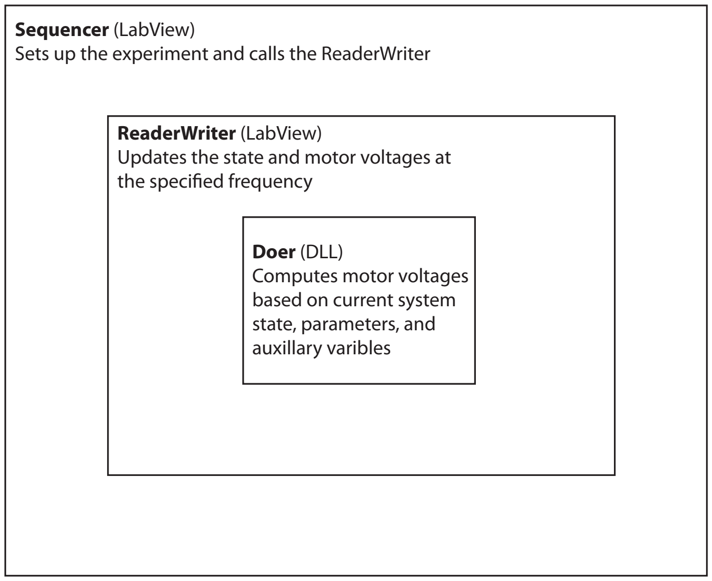
3.3 Vicon cameras and Nexus

The Vicon Ultranet box (or equivalent) needs to be connected to a network card directly using an ethernet cable. The IP address of that network card should be set to a static value of : 192.168.10.1, submet mask : 255.255.255.0 (Using Control Panel -¿ Network Connections-¿Properties). [** write]

3.4 Typical Start Sequence

The following define typical steps used to start up the system, assuming that everything is initially off.

1. Power on BBDLSynthetic, and log in.
2. Power on Brachioplex, wait 1 minute.
3. Start command prompt on BBDLSynthetic, and ping Brachioplex to ensure connection.
4. Start MAX.
5. Start Labview. Open M1.lvproj.


Sequencer (LabView)
Sets up the experiment and calls the ReaderWriter
ReaderWriter (LabView)
Updates the state and motor voltages at the specified frequency
Doer (DLL)
Computes motor voltages based on current system state, parameters, and auxillary varibles

6. Start CVI. Open M1.cws.
7. Start MATLAB. Navigate to Z:
   Documents
   DatabaserProject. Type "Databaser" at the prompt.
8. Compile desired DLL.
9. Launch and run Sequencer.vi in M1.lvproj.

3.5 Overview of Sequencer/ReaderWriter/Doer

The goal of writing new software to control the cadaveric specimens, and robotic systems, was to take advantage of the ease of programming in LabView, while programming user-specific parts (that tend to get ugly in LabView) in C. The sensing, actuation, and sequencing is performed by three components: a sequencer (LabView), a ReaderWriter (LabView - hidden from end user), and a Doer (DLL compiled from C code). The hierarchy of these components is shown in the figure below:

3.6 The Doer

The Doer is a piece of C code, compiled into a Dynamic Link Library (DLL), that converts information about system state and experiment progress into voltages that should be applied to the motors. The C code prototype for a Doer is:

```
int Doer (double *stateMatrix, int bufferInd, int numDataColumns,
double sampFreq, double *motorVoltages, double *param, double *auxVar,
double *user, double *exportVars)
```

Each of the inputs is described below.

3.6.1 The State Matrix, bufferInd, numDataColumns, sampFreq

The state matrix describes everything there is to know about the state of the cadaver actuation system, from the current time point backwards for the duration of the buffer. It is passed to the Doer as an array. The state matrix is an array made up of blocks of the state vector. The state

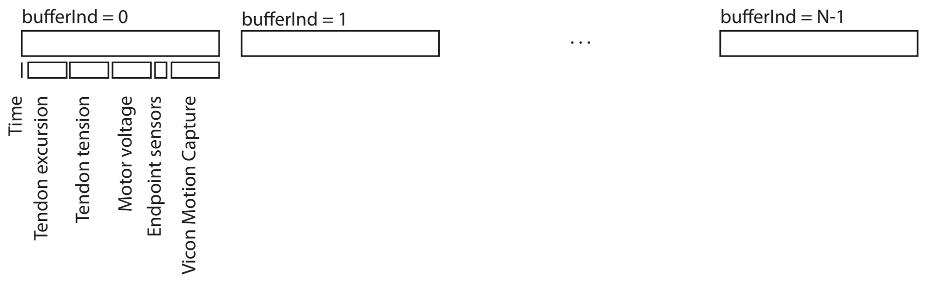


Figure 3.4: default

vector contains the time of each state vector, the tendon excursion, the tendon tension, the motor voltage, the value of any endpoint sensor (accelerometer, load cell, etc.) and the position of each Vicon motion capture marker. The total length of the state vector is called numDataColumns, and is passed to the Doer. Therefore, the state matrix is an array of length numDataColumns*sampFreq*bufferDuration. bufferInd ranges from 0 to N

= sampFreq*bufferDuration. The data is continuously overwritten, so for example, if bufferInd = 1, then the previous 2 timesteps have bufferInd = 0 and bufferInd = N-1.

3.6.2 motorVoltages

Upon completion of the Doer, voltages are immediately applied to the motors. These voltages are set by setting the various elements of the array motorVoltages, which has as many elements as there are motors.

3.6.3 param and auxVar

param and auxVar are arrays that are passed through successive calls to the Doer, and can be used by the user to retain information. The only difference between param and auxVar is that param is static and auxVar can change.

For example, suppose that you want to simulate an integrate-and-fire neuron, and use the spikes of that neuron to drive the motors. param can store static properties of the set of neurons, such as resistance and conductance, whereas auxVar can store the membrane potential and the time of the previous spike.

3.7 The Reader/Writer

The Reader/Writer performs the tedious operations of realtime computing, so that these operations will be hidden from the user in development of their specific applications. Advanced users may need to modify the Reader/Writer in the future, but this is not envisioned at this time.

The ReaderWriter is a LabView VI designed to run on a realtime PXI module. It takes 5 variables as input: the file to store data to, the rate at which signals are sampled and the Doer is called, the length of the buffer - which determines how many previous time points are available in the stateMatrix and the frequency at which data is written from RAM to disk, the initial value of the auxVar array, and the param array.

The ReaderWriter is running 2 timed loops at different priorities. A high priority loop is updating the state matrix and calling the Doer. A low priority loop is running at 10 Hz and is updating plots on the Reader/Writer front panel (so that the user could monitor low-level signals during debugging), and is writing buffered data to disk.

Buffering is accomplished by the following. There are 3 copies of the stateMatrix in RAM. One copy, called stateMatrix, is continuously updated and passed to the Doer. Another 2 copies, stateMatrix0 and stateMatrix1, are alternately filled and appended to the datafile.

3.7.1 Updating the state matrix

The state vector and state matrix are updated by a C function named GetStateMatrixFromSignals that is called by the Reader/Writer. The parameters passed to GetStateMatrixFromSignals are

bufferInd : Row number in the state matrix to be updated with the new state vector.

numDataCols : Number of columns in the stateMatrix (this will change depending on the number of motion capture markers used and the number of export varibles.).

frameTime : This is the current time of execution and is recorded as the first element of the state vector.

sampFreq : The sampling frequency used by the main execution loop of the Labview vi.

motorVoltages : Voltages being sent to the motors (vector of 20).

encoderAnglesCurrent : Current angles measured by the encoders (vector of 20).

lcVoltages : Voltages from the 1DOF load cells measuring tendon tension (vector of 20).

jr3Voltage : Voltage from the 6 channels of the JR3. (vector of 6).

numMarkers : Number of Vicon motion capture markers.

viconString : String containing names of Vicon motion capture markers and their X,Y,Z positions generated by UNAM.exe that is copied from the clipboard.

encoderAnglesPrevious : vector of previous encoder angles. (vector of 20)

jr3Bias : Bias voltage of the JR3 when there is no load. (vector of 6)

stateVector : Vector with frame time, shaft displacements, load cell forces, motor voltages, jr3 outputs and mocap marker positions (vector of of 67+numMarkers*3 = numDataCols).

stateMatrix : Matrix consisting of state vectors in every time step. It has a fixed number of rows and it cycles through the rows.

The function assumes that a fixed number (20) of motors, encoders and load cells is being used. The shaft radii of the motors, calibration matrix and bias for the JR3 load cell and the calibration slopes and biases for the 1DOF load cells are also assumed fixed and the values are hard-wired in the function.

GetStateMatrixFromSignals calls three subfunctions GetForcesFromLoadCellVoltages, GetShaftDisplacements, GetJR3Voltage and GetMarkerDataFromString

GetForcesFromLoadCellVoltages : The load cell voltages are linearly related to the forces being measured. Load cell force = slope x voltage - bias

GetShaftDisplacements : We determine shaft displacements i.e. tendon excursions over a single time step by calculating the change in encoder angle over the time step and multiplying it with the shaft radius. But since the rotary encoder measures angles from 0 degrees to 360 degrees (or -360 degrees) only, we need to determine the true change in angle over the time step which depends on where in the encoder cycle the current time step occurs (see Fig. 3.5).

GetJR3FromVoltage We obtain the 6 DOF forces and moments measured by the JR3 load cell using the calibration matrix provided by JR3 and the voltage recorded when there is no loading (bias). JR3 readings = [Calibration matrix] x Voltage - Bias.

GetMarkerDataFromString We parse the string containing the positions and labels of the Vicon motion capture markers originally generated by UNAM.exe to obtain, in double format, the locations of the markers.

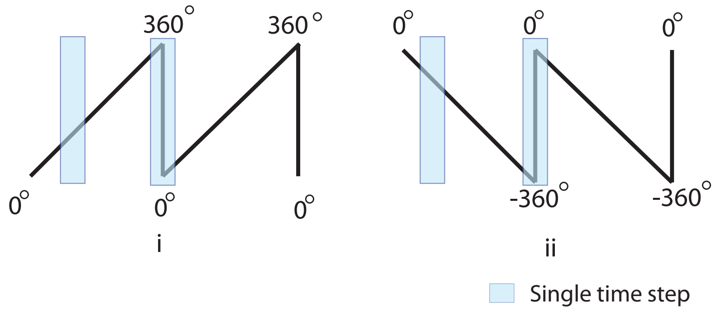


Figure 3.5: Encoder angles cycle between 0 and 360 degrees (or -360 degrees). i and ii are for clockwise and counterclockwise rotations of the encoder.

> The order of the markers remains the same as that in the original string generated by UNAM.exe.

The state vector and the row number given by bufferInd of the state matrix are then updated with these new values. The GetStateMatrixFromSignals function also checks three conditions to ensure that too much current is not sent to the motors which would result in their damage. It ensures that the net shaft displacement is not greater than a max displacement value (1 meter), the forces measured by each 1DOF load cell is below 20 N and the motor voltage is below 6V for each motor. If any of these conditions are not met, the function returns the number of the motor for which they were violated. This would in turn result in the amplifier being turned off and the program termination.