View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by DigitalCommons@USU

Utah State University

DigitalCommons@USU

Undergraduate Honors Capstone Projects

Honors Program

5-2015

Developing a Portable System for Measuring Human Motor Learning

Karen Elizabeth Tew
Utah State University

Follow this and additional works at: https://digitalcommons.usu.edu/honors

Part of the Biology Commons

Recommended Citation

Tew, Karen Elizabeth, "Developing a Portable System for Measuring Human Motor Learning" (2015). *Undergraduate Honors Capstone Projects*. 562.
https://digitalcommons.usu.edu/honors/562

This Thesis is brought to you for free and open access by the Honors Program at DigitalCommons@USU. It has been accepted for inclusion in Undergraduate Honors Capstone Projects by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.

UtahStateUniversity
MERRILL-CAZIER LIBRARY

DEVELOPING A PORTABLE SYSTEM FOR MEASURING HUMAN MOTOR LEARNING

by

Karen Elizabeth Tew

Thesis submitted in partial fulfillment
of the requirements for the degree

of

HONORS IN UNIVERSITY STUDIES
WITH DEPARTMENTAL HONORS

in

Biology
in the Department of Biology

Approved:

[illegible]
Thesis/Project Advisor
Dr. Sydney Schaefer

[illegible]
Departmental Honors Advisor
Dr. Kimberly Sullivan

[illegible]
Thesis Committee Member
Dr. Scott Bernhardt

[illegible]
Director of Honors Program
Dr. Kristine Miller

UTAH STATE UNIVERSITY
Logan, UT

Spring 2015

ABSTRACT

Developing a Portable System for Measuring Human Motor Learning

by

Karen Tew, Departmental Honors

Utah State University, 2015

Project Advisor: Dr. Sydney Schaefer, Department of Health, Physical Education, and Recreation

Departmental Honors Advisor: Dr. Kimberly Sullivan, Department of Biology

Point-to-point reaching is a commonly used paradigm in the field of human motor control. By studying how people move their arms from one location in space to another, researchers have gained insight into how the central nervous system controls and learns skilled movement. Many experimental methods that are designed to study reaching are not portable. This makes it difficult for researchers to access certain clinical populations with limited mobility or motor dysfunction. We have addressed this issue by developing a point-to-point reaching system that can capture key movement variables (e.g. speed and accuracy) yet is portable and inexpensive. We have developed this system with MATLAB software and MaKey MaKey hardware, a commercially-available, single-board microcontroller. Participants will reach with a metal stylus to and from targets on a tabletop made of aluminum foil (i.e. point-to-point reaching). Our current prototype system counts and time-stamps when the stylus touches each aluminum target, then exports these data to a continuously updating log in Microsoft Excel. In addition to the low cost and portability of this system, it allows the experimenter to adjust the reaches' difficulty without modifying the data acquisition code by simply changing the size, number, and/or distance between the targets.

Our next step is to pilot this system in a motor learning study in which participants repetitively reach from point to point as training.

This work was supported in part by the USU Office of Research and Graduate Studies (RC #A28037) and USU Department of Health, Physical Education, and Recreation.

ACKNOWLEDGEMENTS

This paper is dedicated to my amazing family, friends, and professors. I wouldn't have completed this degree without them. I'd especially like to thank Dr. Sydney Schaefer for all her help, both as a school-advisor and as a life-advisor; her advice has been invaluable. I'm grateful to Dr. Elizabeth Vargis, Dr. Scott Bernhardt, Mike Bartsch, Zach Bent and Buffy Evans for their encouragement and support.

TABLE OF CONTENTS

Abstract i

Acknowledgements iii

Table of Contents iv

List of Figures v

Chapter 1: Literature Review 1

1.1 Introduction 1

1.2 Stroke and Rehabilitation 2

1.3 Quantification 5

1.4 Current Systems and Limitations 6

Chapter 2: Methodology 8

2.1 Hardware 8

2.2 Physical Set-Up 9

2.3 Software 10

2.4 MATLAB Code and Commentary 11

Chapter 3: Results 37

2.1 Ease of Use 37

2.2 Portability 38

2.3 Total Cost 38

Chapter 4: Discussion 40

References 42

Appendix A: Author Biography 45

Appendix B: Reflective Writing 46

LIST OF FIGURES

#	*Description*	*Page*
1.	Robotic Therapy vs. Traditional Therapy Comparison	4
2.	MaKey MaKey Device	8
3.	Three Target Design	9
4.	Modified Alligator-Clip Stylus	10
5.	Prototype System Set-Up	37
6.	Current vs. Future Prototype Cost Analysis	41

CHAPTER 1

Literature Review

1.1 Introduction

Studying human motor movement is a valuable way to assess neurological function. Voluntary arm movement is controlled by complex neural pathways that communicate through hierarchical feedback loops. Neurological events, such as strokes, that damage these pathways can negatively affect upper-extremity mobility. By tracking how the arm moves from point to point in space, researchers can determine the extent of neurological damage after a stroke. By capturing how people reach from one point in space to another (e.g. three-dimensions, or in 3D), inferences can be made about how the brain is controlling (or attempting to control) these movements in both healthy and clinical populations. For example, one's movement (i.e. "motor performance") can be quantified according to smoothness, speed, or efficiency. Differences in these factors correlate to differing levels of brain damage and can aid researchers and clinicians in deciding what form of rehabilitation will be of most value to the patient.

Unfortunately, many 3D tracking machines are expensive, bulky and non-portable. This is a problem for researchers or clinicians who may have limited funds available to them. Additionally, many stroke survivors that would benefit from 3D reaching studies have limited mobility due to post-stroke cognitive impairment; they are unable to participate in studies that don't take place in their own homes. Therefore, an inexpensive and portable tracking system is needed to study people who have experienced stroke.

1.2 Stroke and Rehabilitation

Stroke is a leading cause of adult disability in the United States, with about 800,000 people experiencing stroke and 130,000 dying from stroke each year (Lloyd-Jones, 2010). On average, one American dies from stroke every four minutes, and the total cost for the United States is upwards of $34 billion (CDC, 2015). In 2009, 66% of people hospitalized for stroke were over the age of 65 (Hall, 2012). Many survivors experience post-stroke hemiparesis - weakness in either the entire left or entire right side of the body. Typically, only one arm is negatively affected by stroke, termed the paretic arm. 80% of stroke survivors have some arm disability immediately after stroke, including spasticity and weakness in the paretic arm (Mozaffarian, 2015, Broeks, 1999). Fewer than 20% of these individuals recover enough arm movement to functionally use in daily activities (Gowland, 1982).

Rehabilitation can help stroke patients recover motor skills, particularly if it is long-lasting, repetitive, and begins within 24 to 48 hours post-stroke (Volpe, 2003). Since upper limbs are frequently used in daily living to manipulate objects, the recovery of motor skills in the arms and hands are of key importance to successful rehabilitation (Hillman, 2001).

Intensive therapy is essential in regaining language skills after a stroke; however, there is some debate about intensive therapy's role in physiological or occupational therapy. Recent studies show little benefit from intensive therapy on arm function (Norouzi-Gheidari, 2012), though evidence from the majority of clinical analyses show better stroke recovery with more intense therapy in a short amount of time. Type of therapy is also important in stroke recovery. Repetitive training or resistance exercises produce few benefits, whereas task-specific therapy (e.g. buttoning buttons, reaching for a glass) shows good results. A recent study showed that increased use of the paretic hand formed new corticospinal networks during task-specific therapy following a stroke (Papathanassiou, 2003).

Several methods of rehabilitation are used to return stroke survivors to their highest possible level of independence. Repetitive physical activities are traditionally used in rehabilitation to strengthen muscles, improve coordination and improve range-of-motion. Using mobility aids, such as canes or leg braces, can help survivors regain the ability to walk. Cognitive and behavioral rehabilitation is used to help regain speech ability, test cognitive skills, and assess emotional adjustment. Medications can be prescribed to help combat stroke-related depression or to help increase mobility. Biological therapies, such as stem cell therapy, are being investigated but are not yet being used for functional rehabilitation outside of clinical trials. Technology-assisted rehabilitation is a promising new area of research that utilizes electricity, robotic devices, computer programs, new imaging techniques, and physiological monitoring to improve post-stroke mobility.

Fasoli et. al. (2004) showed that robotic therapy is an effective way to reduce impairment in stroke patients with moderate to severe hemiparesis. Therapists who utilize robotic devices for simple repetitive motions of the paretic limb can spend more time working on complex, task-oriented movements with their patients while the patient practices repetitive movements alone.

This reduces one-on-one therapist time while maintaining rehabilitation intensity and, consequently, saves patients money (Krebs, 1998). One study compared average per-patient cost of therapy for robot-assisted therapy and traditional therapy and found the robotic therapy to be 30% less expensive, after accounting for therapist hourly cost and machine cost (Lo, 2010).

Robotic devices have been shown to be more effective for patients with severe hemiparesis, whereas in moderately affected patients, it has no advantage over traditional physical training (Morone, 2012). However, this research is complicated by the lack of standardized rehabilitation schedules in published works. Each study shows varying doses, or frequency, of rehabilitation sessions. For example, if one researcher directed participants to do 500 trials over the course of two weeks using robotic therapy, while another studied 200 trials over the course of four weeks using traditional physical therapy, the results would not be directly comparable. Lo et. al. (2010) concluded that neither robotic or intensive therapist-driven therapy is inherently better than the other for rehabilitation, when the dose was equal (see Figure 1).

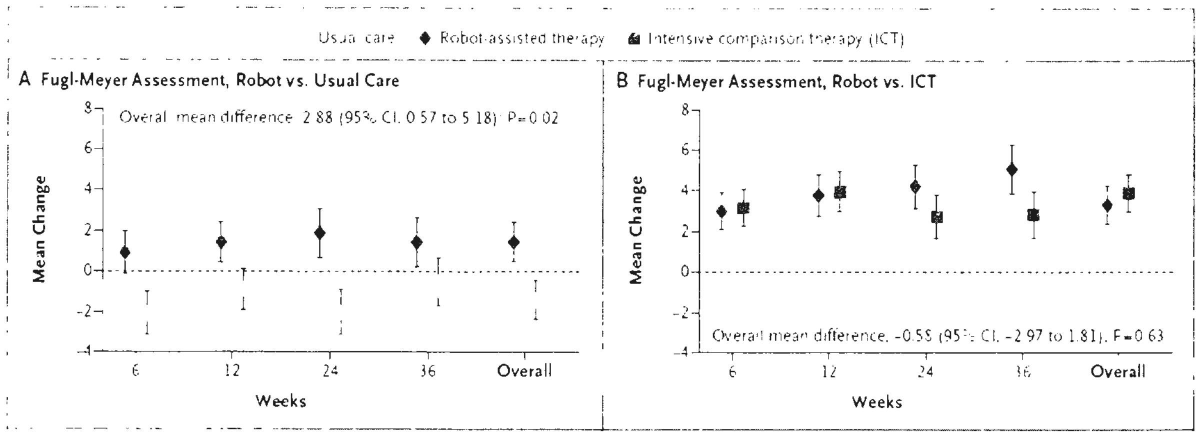

Figure 1. Subjects were subjected to one of three rehabilitation methods: usual care, robot-assisted therapy, or intensive comparison therapy (rigorous physical therapy). Each Each subject got the same frequency/dose of rehabilitation sessions during the 36 week study. Figures A and B show the least-squares means at each time point and overall and have been normalized to fit baseline scores for the Fugl-Meyer Assessment. Higher scores mean better functioning. Figures A and B show that while robot-assisted therapy was more effective than usual care for recovery, there was no significant difference between robotic therapy and intensive comparison therapy. This implies that the dose is more important in recovery than the method of rehabilitation (Lo, 2010)

1.3 Quantification

Researchers can only understand the objective benefit of rehabilitation on regaining arm movement by quantifying motor performance. Currently used assessments include: the Fugl-Meyer Assessment, the Action Research Arm Test, and the Box and Block Test. Each of these tests has been used for many years as clinical assessments of arm function; however, they do not provide data on how the arm is moving through space. This makes the results less generalizable between people and can under- or over-estimate the amount of therapy still needed (Rohrer, 2004).

One way to quantify movement involves point-to-point reaching evaluations where the participant reaches from target to target on a horizontal plane. Researchers record the spatial position of the arm in 3D as it moves from point to point. They can then determine overall trends in the movements that are associated with neurological function, such as velocity and acceleration between targets, accuracy in hitting targets and efficiency of movement (e.g. shakiness, end-point trajectory, or excessive vertical displacement) (Rohrer, 2004). Normal reaching movements are smooth in both position and velocity, whereas post-stroke reaching movements are more irregular and often show multi-peak velocity profiles (Flash, 1985). Since reaching is involved in many daily activities, this method is a good indicator of functional rehabilitation (McCrea, 2005).

1.4 Current Systems and Limitations

Several robotic systems exist that act both as rehabilitation aides and 3D movement trackers for post-rehabilitation assessment. These systems decrease the therapist time needed per patient, thus lowering the cost of treatment, and improving a clinic's productivity. Of the several commonly used robotic systems the average cost was $230,750 (Wagner, 2011). With added maintenance contract plans and financing, the total cost for a robotic system is around $422,532 (Wagner, 2011).

Although Wagner et. al. (2011) determined that per-session costs of robotic therapy were lower than per-session costs of intensive physical therapy, the up-front cost of a robotic system can be prohibitive to many researchers and clinics. Additionally, robotic systems are bulky and engineered to be immobile. Their size makes them impractical for use outside of clinics. Therefore, these systems are not adequate for researchers studying populations who have limited mobility, such as older adults or people who have had strokes. Our lab is investigating motor function in individuals who are both >65 years old and have had strokes. We have had difficulty recruiting this population to participate in clinical trials, partially because they don't have the ability to transport themselves to our laboratory to do testing on our in-house 3D monitoring system.

We needed to develop a simple motion-tracking system that was transportable, inexpensive and user-friendly. Since Lo et. al. (2010) determined that repetition was the most important variable, we needed to create a non-robotic device that would allow for a high dosage of therapy while still being able to accurately quantify therapeutic progress.

CHAPTER 2

Methodology

2.1 Hardware

With the primary goals of portability and affordability, we chose to use a MaKey MaKey microcontroller for the device base. MaKey MaKey is a handheld, printed circuit board with an ATMega32u4 microcontroller running Arduino Leonardo firmware. It uses high resistance switching to detect any closed switches on the board, effectively determining whether or not a circuit is being made. When a circuit is completed, MaKey MaKey translates the connection into an electronic signal that passes through a USB cable into an attached computer. This signal can be programmed to activate a keyboard key, a mouse click, or a mouse motion. As of March 2015, a basic MaKey MaKey kit costs 50 dollars, making it a very economical choice for our project.

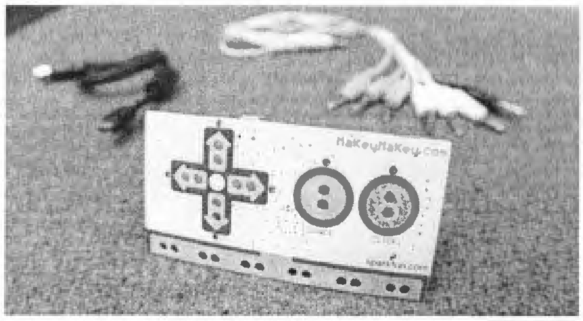
Figure 2. Front side of MaKey MaKey board with USB cable and alligator clips in background. (© MaKey MaKey, 2015)

In order to recreate a point-to-point reaching trial without expensive robotics equipment, we simplified the experimental set-up while maintaining a similar physical layout. Rather than tracking exact hand movements on the XYZ plane, we only collected overall trial time data. This should directly correlate with motor learning in that the more a subject practices a movement, the shorter their trial time will be. So while our system is not capable of collecting specific information about cumulative distance traveled, it is robust enough to capture general progress in arm rehabilitation.

2.2 Physical Set-Up

We created a three target design using one 8.5”x11” sheet of paper and one 8.5”x12” sheet of heavy-duty aluminum foil. Four equal sized holes with 45 mm diameters were cut distally in the paper at positions of 65°, 90°, and 115° relative to the home target. The cut paper was then taped to the front of the aluminum foil sheet, leaving a 1 inch strip of exposed foil at the top (see Figure 3).

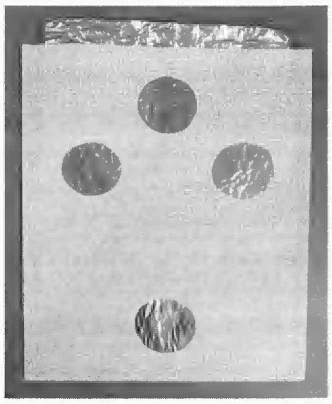
Figure 3. Prototype three target design using paper and aluminum foil.

We physically connected the MaKey MaKey to the computer using the USB cable included in the basic kit. No additional software download is needed; the MaKey MaKey is natively set up to be compatible with Mac, Windows, Chromebook and many Linux distributions. We connected one end of an alligator clip to the “spacebar” area of the MaKey MaKey and the other end to the exposed inch of tinfoil on our target design (see Figure 3). Another alligator clip was modified into a stylus (see Figure 4) and clipped to the bottom non-programmed row of holes on the MaKey MaKey. This set-up effectively forms an open circuit. Each time the stylus touches a target, the circuit is closed and the computer registers it as a particular action (e.g. spacebar press).

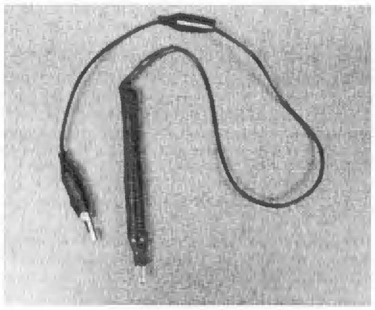
Figure 4. Stylus made from an alligator clip, pen and electrical tape.

Three young adult participants (19, 20 and 21 years old, all right hand dominant) and two older adult participants (70 and 73 years old, right hand dominant and left hand dominant respectively) were recruited to test our prototype MaKey MaKey device. In these preliminary tests, subjects were instructed to use their non-dominant hand to hold the stylus. During each trial, right-handed participants touched in order: home target, left-most distal target, home target, center distal target, home target, right-most distal target time. Left-handed participants mirrored the right-handed target reaching by touching in order: home target, right-most distal target, home target, center distal target, home target, left-most distal target. Both groups repeated the pattern five times and the trial ended when the final distal target was touched.

2.3 Software

We wrote a user-editable MATLAB program to record participant information, count target touches and collect trial times from the MaKey MaKey device, and append those data to a Microsoft Excel experimental log file. This program can be seen in its entirety in Appendix A, and will also be described briefly here. To use these files, copy them into .m files into your working folder within MATLAB and save with the bolded titles as file names.

When the MATLAB program is first opened (using command CounterGUI) a graphical user interface (i.e. GUI) window opens. In this window are several editable boxes where the researcher can input participant data such as: subject ID number, trial number, hand used by subject, and relative day of testing. Each entered value is saved as a new variable that can later be exported to Excel. This window also includes a start push-button, stop push-button, and a small non-editable target counter.

After filling in the relevant information, the researcher pushes the "Tab" key on the keyboard until the start button is active. The subject can then begin the trial whenever they want – the first circuit made will press the start button and begin a timer. Simultaneously, the target counter will begin. When the counter reaches the correct number (30 in our experiment) the stop button is automatically pressed and the timer is stopped. Additionally, all relevant information including trial time and user-inputted data is imported into the next open row of our continuously updating Excel log file. To start a new trial, the researcher simply needs to change the subject information in the GUI and begin a new trial. The variables will be updated and the timer will restart.

2.4 MATLAB Code and Commentary

Filenames are in bold print. Comments are marked with a %. Each file is separated with a solid line. In the xlsappend code, my adaptations are marked with italics. Italics are not recognized in MATLAB, it is only marked for clarification in this thesis.

CounterGUI

```
% GUI for basic info to export to Excel

% save as CounterGUI.m


i=[0];

uicontrol('Pos',[110 280 60 19],'Style','text','String','Trial #');

a = uicontrol('Pos',[175 280 246 19],'Style','edit');

uicontrol('Pos',[110 262 60 19],'Style','text','String','Subject ID');

b = uicontrol('Pos',[175 262 246 19],'Style','edit');

uicontrol('Pos',[110 243 60 19],'Style','text','String','Hand');

c = uicontrol('Pos',[175 243 246 19],'Style','edit');

uicontrol('Pos',[110 224 60 19],'Style','text','String','Day');

d = uicontrol('Pos',[175 224 246 19],'Style','edit');

e = uicontrol('Pos',[200 200 60

19],'Style','pushbutton','String','Start','Callback','getvar');

f = uicontrol('Pos',[300 200 60

19],'Style','pushbutton','String','Stop','Callback','stopfx');

g = uicontrol('Pos',[265 200 30 18],'Style','pushbutton','String',1, 'Callback',

'pushcallback');

uicontroll


%start pushbutton function – 1) starts tic-toc timer 2) Assigns user

%inputted strings to variables 3) switch active button to counter 4) Starts count of

space
```

```
%presses/circuits made for the touch test 5 )When touches reach a certain
%number, automatically 'hit' stop

%stop pushbutton function    1) Get and assign variable to toc 2) export entered
data into excel file in the right place (should move down 1 row for each trial
append)
```

---

getvar

```
% Set Variables for Excel Export

uicontrol(g)
i [0];
tic;
get(a,'string');
A    cellstr(ans);
get(b,'string');
B  = cellstr(ans);
get(c,'string');
C = cellstr(ans);
get (d,'string');
D = cellstr(ans);
```

```
H = cellstr(datestr(now));
```

---

pushcallback

```
% Callback for Start Button Press
% save as pushcallback.m


i = i+1;
set(g,'string',i);
I;
if (I ==  31);
   run stopfx;
elseif (i >31);
   char('Too many touches')
end


%Set I to whatever you want it to be to get 15 touches.  I have it set as they start
with stylus off %the aluminum foil, start by touching the bottom target, touch
each of the three top targets 5 %times each (15 touches total for top targets) then
it ENDS when they hit the bottom target one %last time after the 15 top ones.
% Example:
%      2
```

% 1 3

%

% b

%

% Subject would hit b1b2b3 b1b2b3 b1b2b3 b1b2b3 b1b2b3 b

% So a total of 31 target hits, so i=31

% If you want to end on 3 instead of b, change I to 30

---

stopfx

% After correct "i" number variable is hit, find toc and export user inputted data to Excel

% save as stopfx.m

toc;

I toc;

SUCCESS xlsappend('C:\Users\Lab User\Documents\MATLAB\MATLAB Output.xlsx',[H,A,B,C,D,I])

% Be sure to change path to wherever you are storing your excel file.

---

xlsappend

% save as xlsappend.m

% adapted from "xlsappend" written by Brett Shoelson, PhD, 8/30/2010.

Copyright 1984-2010 The MathWorks, Inc.

```
function [success,message] = xlsappend(file,data,sheet)

% XLSAPPEND Stores numeric array or cell array to the end of specified Excel
sheet.
%
%   REQUIRES ONLY ONE CALL TO THE EXCEL ACTXSERVER, so the
overhead is less
%   than for successive xlsread/xlswrite calls.
%
%   [SUCCESS,MESSAGE]=XLSAPPEND(FILE,ARRAY,SHEET) writes
ARRAY to the Excel
%   workbook, FILE, into the area beginning at COLUMN A and FIRST
UNUSED
%   ROW, in the worksheet specified in SHEET.
%   FILE and ARRAY must be specified. If either FILE or ARRAY is empty, an
%   error is thrown and XLSAPPEND terminates. The first worksheet of the
%   workbook is the default. If SHEET does not exist, a new sheet is added at
```

% the end of the worksheet collection. If SHEET is an index larger than the

% number of worksheets, new sheets are appended until the number of

worksheets

% in the workbook equals SHEET. The success of the operation is

% returned in SUCCESS and any accompanying message, in MESSAGE. On

error,

% MESSAGE shall be a struct, containing the error message and message ID.

% See NOTE 1.

%

% [SUCCESS,MESSAGE] XLSAPPEND(FILE,ARRAY) writes ARRAY to

the Excel

% workbook, FILE, in the first worksheet.

%

% [SUCCESS,MESSAGE]=XLSAPPEND(FILE,ARRAY) writes ARRAY to

the Excel

% workbook, FILE, starting at cell A[firstUnusedRow] of the first

% worksheet. The return values are as for the above example.

%

% XLSAPPEND ARRAY FILE, is the command line version of the above

example.

%

% INPUT PARAMETERS:

% file: string defining the workbook file to write to.

```
%         Default directory is pwd; default extension 'xls'.
%     array:  m x n numeric array or cell array.
%     sheet:  string defining worksheet name;
%             double, defining worksheet index.
%
%   RETURN PARAMETERS:
%     SUCCESS: logical scalar.
%     MESSAGE: struct containing message field and message_id field.
%
%   EXAMPLES:
%
%   SUCCESS = XLSAPPEND('c:\matlab\work\myworkbook.xls',A) will write
A to
%   the sheet 1 of workbook file, myworkbook.xls. On success, SUCCESS will
%   contain true, while on failure, SUCCESS will contain false.
%
%   NOTE 1: The above functionality depends upon Excel as a COM server. In
%   absence of Excel, ARRAY shall be written as a text file in CSV format. In
%   this mode, the SHEET argument shall be ignored.
%
% See also XLSREAD, XLSWRITE.
%
```

```
% Written by Brett Shoelson, PhD, 8 30 2010.
% Copyright 1984-2010 The MathWorks, Inc.

% Set default values.
Lastwarn('');
Sheet1    1;

if nargin    3
   sheet    Sheet1;
end

if nargout    0
   success    true;
   message    struct('message',{''},'identifier',{''});
end

% Handle input.
Try
   % handle requested Excel workbook filename.
   If -isempty(file)
      if   ischar(file)
         error('Exlswrite:InputClass','Filename must be a string.');
      end
```

```
    % check for wildcards in filename
    if any(findstr('*', file))
        error('E:xlswrite:FileName', 'Filename must not contain *.');
    end
    [Directory,file,ext]=fileparts(file);

    if isempty(ext) % add default Excel extension;
        ext = '.xls';
    end
    file = abspath(fullfile(Directory,[file ext]));
    [a1 a2] = fileattrib(file);
    if a1 && ~(a2.UserWrite == 1)
        error('E:xlswrite:FileReadOnly', 'File cannot be read-only.');
    end
else % get workbook filename.
    Error('E:xlswrite:EmptyFileName','Filename is empty.');
end

% Check for empty input data
if isempty(data)
    error('E:xlswrite:EmptyInput','Input array is empty.');
end
```

```
% Check for N-D array input data

if ndims(data) > 2

    error('I:xlswrite:InputDimension',...

        'Dimension of input array cannot be higher than two.');

end


% Check class of input data

if ~(iscell(data) || isnumeric(data) || ischar(data)) && ~islogical(data)

    error('I:xlswrite:InputClass',...

        'Input data must be a numeric, cell, or logical array.');

end



% convert input to cell array of data.

 If iscell(data)

    A = data;

 else

     A = num2cell(data);

 end


if nargin > 2

    % Verify class of sheet parameter.

    If ~(ischar(sheet) || (isnumeric(sheet) && sheet > 0))
```

```
            error('Exlswrite:InputClass',...
                'Sheet argument must be a string or a whole number greater than 0.');
        end
        if isempty(sheet)
            sheet = Sheet1;
        end
        % Parse sheet
        if ischar(sheet) && ~isempty(strfind(sheet,':'))
            sheet = Sheet1;% Use default sheet.
        End
    end

catch exception
    if ~isempty(nargchk(2,4,nargin))
        error('Exlswrite:InputArguments',nargchk(2,4,nargin));
    else
        success = false;
        message = exceptionHandler(nargout, exception);
    end
    return;
end
%---------------------------------------------------------------------------
% Attempt to start Excel as ActiveX server.
```

```
Try
    Excel = actxserver('Excel.Application');
    % open workbook
    Excel.DisplayAlerts = 0;

    %Workaround for G313142.  For certain files, unless a workbook is
    %opened prior to opening the file, various COM calls return an error:
    %0x800a9c64.  The line below works around this flaw.  Since we have
    %seen only one example of such a file, we have decided not to incur the
    %time penalty involved here.
    %     aTemp = Excel.workbooks.Add(); aTemp.Close();

    ExcelWorkbook = Excel.workbooks.Open(file);
    %ExcelWorkbook.ReadOnly
    format = ExcelWorkbook.FileFormat;
    if strcmpi(format, 'xlCurrentPlatformText') == 1
       error('Exlsread:FileFormat', 'File %s not in Microsoft Excel Format.', file);
    end

    %Sheets = Excel.ActiveWorkBook.Sheets;
    activate_sheet(Excel,sheet);
    readinfo = get(Excel.Activesheet,'UsedRange');
    %get(Excel.Activesheet,'Name')
    if numel(readinfo.value)==1 && isnan(readinfo.value)
        m2 = 0;
    else
        [m2,n2] = size(readinfo.value);
    end

catch exception %#ok<NASGU>
    warning('E:xlswrite:NoCOMServer',...
        ['Could not start Excel server for export.\n' ...
        'XLSWRITE will attempt to write file in CSV format.']);
    if nargout > 0
        [message.message,message.identifier] = lastwarn;
    end
    % write data as CSV file, that is, comma delimited.
    File = regexprep(file, '(\.xls[^.]*+)$', '.csv');
    try
        dlmwrite(file,data,','); % write data.
    Catch exception
        exceptionNew = Mexception('E:xlswrite:dlmwrite', 'An error occurred on
data export in CSV format.');
        exceptionNew = exceptionNew.addCause(exception);
        if nargout == 0
            % Throw error.
            Throw(exceptionNew);
        else
            success = false;
            message.message = exceptionNew.getReport;
            message.identifier = exceptionNew.identifier;
        end
    end
    return;
end
%---------------------------------------------------------------------------
try
    % Construct range string
    % Range was partly specified or not at all. Calculate range.
    % Select range of occupied cells in active sheet.
    % Activate indicated worksheet.
    Message = activate_sheet(Excel.sheet);
    [m,n] = size(A);
    range = calcrange('',m,n,m2);
catch exception
    success = false;
    message = exceptionHandler(nargout, exception);
    return;
end
```

```
%--------------------------------------------------------------------------

try
    bCreated    exist(file,'file');
    ExecuteWrite;
catch exception
    if (bCreated && exist(file, 'file') == 2)
        delete(file);
    end
    success   false;
    message   exceptionHandler(nargout, exception);
end
    function ExecuteWrite
            26lipped    onCleanup(@()cleaner(Excel, file));
            if bCreated
               % Create new workbook.
               %This is in place because in the presence of a Google Desktop
               %Search installation, calling Add, and then SaveAs after adding data,
               %to create a new Excel file, will leave an Excel process hanging.
               %This workaround prevents it from happening, by creating a blank file,
               %and saving it.  It can then be opened with Open.


               %ExcelWorkbook = Excel.workbooks.Add;
        switch ext
            case '.xls' %xlExcel8 or xlWorkbookNormal
                xlFormat = -4143;
            case '.xlsb' %xlExcel12
                xlFormat = 50;
            case '.xlsx' %xlOpenXMLWorkbook
                xlFormat = 51;
            case '.xlsm' %xlOpenXMLWorkbookMacroEnabled
                xlFormat = 52;
            otherwise
                xlFormat = -4143;
        end
        ExcelWorkbook.SaveAs(file, xlFormat);
        ExcelWorkbook.Close(false);
    end


    %Open file
    %ExcelWorkbook = Excel.workbooks.Open(file);
    if ExcelWorkbook.ReadOnly ~= 0
        %This means the file is probably open in another process.
        Error('Exlswrite:LockedFile', 'The file %s is not writable.  It may be
locked by another process.', file);
    end
```

```
        try % select region.
           % Select range in worksheet.
           Select(Range(Excel,28lippe('%s',range)));

        catch exceptionInner % Throw data range error.
           Throw(Mexception('Exlswrite:SelectDataRange', 28lippe('Excel
returned: %s.', exceptionInner.message)));
        end

        % Export data to selected region.
        Set(Excel.selection,'Value',A);
        ExcelWorkbook.Save
    end
end
function cleaner(Excel, filePath)
    try
      %Turn off dialog boxes as we close the file and quit Excel.
      Excel.DisplayAlerts = 0;
      %Explicitly close the file just in case. The Excel API expects
      %just the filename and not the path. This is safe because Excel
      %also does not allow opening two files with the same name in
      %different folders at the same time.
      [~, n, e] = fileparts(filePath);
```

```
        29lipped29 = [n e];
        Excel.Workbooks.Item(29lipped29).Close(false);
    catch exception %#ok<NASGU>
        %If something fails in closing, there is nothing to do but attempt
        %to quit.
    End
    Excel.Quit;
end
%-------------------------------------------------------------------------
function message = activate_sheet(Excel,Sheet)
% Activate specified worksheet in workbook.

% Initialise worksheet object
WorkSheets = Excel.sheets;
message = struct('message',{''},'identifier',{''});

% Get name of specified worksheet from workbook
try
    TargetSheet = get(WorkSheets,'item',Sheet);
catch exception %#ok<NASGU>
    % Worksheet does not exist. Add worksheet.
    TargetSheet = addsheet(WorkSheets,Sheet);
    warning('Exlswrite:AddSheet','Added specified worksheet.');
    if nargout > 0
        [message,message,message,identifier] = lastwarn;
    end
end

% activate worksheet
Activate(TargetSheet);
end
%--------------------------------------------------------------------------
function 30lipped3030 = addsheet(WorkSheets,Sheet)
% Add new worksheet, Sheet into 30lipped3030 collection, WorkSheets.

If isnumeric(Sheet)
    % iteratively add worksheet by index until number of sheets == Sheet.
    While WorkSheets.Count < Sheet
        % find last sheet in worksheet collection
        lastsheet = WorkSheets.Item(WorkSheets.Count);
        30lipped3030 = WorkSheets.Add([],lastsheet);
    end
else
    % add worksheet by name.
    % find last sheet in worksheet collection
    lastsheet = WorkSheets.Item(WorkSheets.Count);
    31lipped3131   WorkSheets.Add([],lastsheet);
end
% If Sheet is a string, rename new sheet to this string.
If ischar(Sheet)
    set(31lipped3131,'Name',Sheet);
end
end
%--------------------------------------------------------------------------
function [absolutepath]=abspath(partialpath)


% parse partial path into path parts
[pathname filename ext]   fileparts(partialpath);
% no path qualification is present in partial path; assume parent is pwd, except
% when path string starts with '~' or is identical to '~'.
If isempty(pathname) && isempty(strmatch('  ',partialpath))
    Directory   pwd;
elseif isempty(regexp(partialpath,'(.:      )','once')) && ...
        isempty(strmatch('  ',partialpath)) && ...
        isempty(strmatch('~',partialpath));
    % path did not start with any of drive name, UNC path or '~'.
    Directory   [pwd,filesep,pathname];
else
    % path content present in partial path; assume relative to current directory,
    % or absolute.
    Directory = pathname;
end

% construct 32lipped32 filename
absolutepath = fullfile(Directory,[filename,ext]);
end
%---------------------------------------------------------------------------
function range = calcrange(range,m,n,offset)
% Calculate full target range, in Excel A1 notation, to include array of size
% m x n

range = upper(range);
cols = isletter(range);
rows = ~cols;
% Construct first row.
If ~any(rows)
    firstrow = offset+1; % Default row.
Else
    firstrow = str2double(range(rows)); % from range input.
End
% Construct first column.
If ~any(cols)
```

```
    firstcol = 'A'; % Default column.
Else
    firstcol = range(cols); % from range input.
End
try
    lastrow = num2str(firstrow+m-1);  % Construct last row as a string.
    Firstrow = num2str(firstrow);     % Convert first row to string image.
    Lastcol = dec2base27(base27dec(firstcol)+n-1); % Construct last column.

    Range = [firstcol firstrow ':' lastcol lastrow]; % Final range string.
Catch exception
    error('Exlswrite:CalculateRange', 'Invalid data range: %s.', range);
end
end
%-----------------------------------------------------------------------
function string = index_to_string(index, first_in_range, digits)

letters = 'A':'Z';
working_index = index - first_in_range;
outputs = cell(1,digits);
[outputs{1:digits}] = ind2sub(repmat(26,1,digits), working_index);
string = fliplr(letters([outputs{:}]));
end
```

```
%------------------------------------------------------------------------

function [digits first in range]   calculate range(num to convert)


digits    1;

first in range    0;

current sum    26;

while num to convert    current sum

   digits    digits + 1;

   first in range    current sum;

   current sum    first in range    26.^digits;

end

end

%-----------------------------------------------------------------------------

function s    dec2base27(d)


%   DEC2BASE27(D) returns the representation of D as a string in base 27,

%   expressed as 'A'..'Z', 'AA','AB'...'AZ', and so on. Note, there is no zero

%   digit, so strictly we have hybrid base26, base27 number system.  D must be a

%   negative integer bigger than 0 and smaller than 2^52.

%

%   Examples

%       dec2base(1) returns 'A'

%       dec2base(26) returns 'Z'
%       dec2base(27) returns 'AA'
%-----------------------------------------------------------------------------


d = d(:);
if d ~= floor(d) | any(d(:) < 0) | any(d(:) > 1/eps)
    error('I:xlswrite:Dec2BaseInput',...
        'D must be an integer, 0 <= D <= 2^52.');
end
[num_digits begin] = calculate_range(d);
s = index_to_string(d, begin, num_digits);
end
%-----------------------------------------------------------------------------
function d = base27dec(s)
%   BASE27DEC(S) returns the decimal of string S which represents a number in
%   base 27, expressed as 'A'..'Z', 'AA','AB'...'AZ', and so on. Note, there is
%   no zero so strictly we have hybrid base26, base27 number system.
%
%   Examples
%       base27dec('A') returns 1
%       base27dec('Z') returns 26
%       base27dec('IV') returns 256
%-----------------------------------------------------------------------------
```

```
if length(s) == 1
  d = s(1) - 'A' + 1;
else
   cumulative = 0;
   for I = 1:numel(s)-1
      cumulative = cumulative + 26.^I;
   end
   indexes_fliped = 1 + s - 'A';
   indexes = fliplr(indexes_fliped);
   indexes_in_cells = mat2cell(indexes, 1, ones(1,numel(indexes)));
   d = cumulative + sub2ind(repmat(26, 1,numel(s)), indexes_in_cells{:});
end
end
%%--------------------------------------------------------------------------


function messageStruct = exceptionHandler(nArgs, exception)
   if nArgs == 0
      throwAsCaller(exception);
   else
      messageStruct.message = exception.message;
      messageStruct.identifier = exception.identifier;
   end
end
```

CHAPTER 3

Results

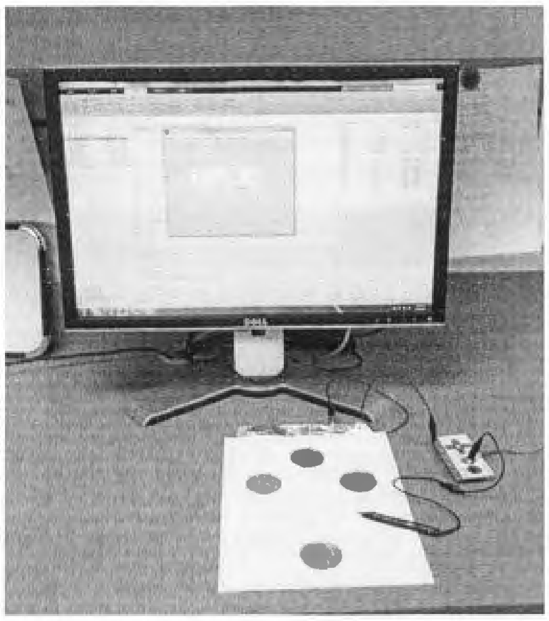
Figure 5. MaKey MaKey system running on a desktop computer

After five preliminary tests of the prototype system, we succeeded in creating a functional point-to point reaching system that is user-friendly, portable, and low-cost.

3.1 Ease of Use

The MATLAB code was written to be accessible and understandable to researchers that may not have extensive experience in coding. We included troubleshooting tips and simple explanations of the code in each .m file to assist researchers who may want to use this code for their own experiments.

Other point-to-point reaching systems have been designed that show virtual targets on a vertical screen placed in front of the patient. In these systems, a sensor is secured onto the subject's finger and moved on the horizontal plane while a virtual representation of the sensor position is displayed on the vertical screen. Older adults sometimes find this set-up difficult to use because their finger isn't moving in the same plane as the virtual representation shows. We simplified this design by using physical targets rather than virtual targets and a pen-like stylus instead of a sensor taped onto the participant's finger. Our set-up is more intuitive, especially for older adults.

In order to increase the relative difficulty of this protocol, the researcher only needs to decrease the diameter of targets, increase the number of targets, and/or increase the distance between targets.

3.2 Portability

If the software is downloaded onto a laptop rather than a desktop computer as shown in Figure 5, our system is small enough to fit into a briefcase or backpack. The MaKey MaKey, paper, heavy duty tinfoil and connector wires have an overall weight of ~5 oz. Coupled with a laptop, this system weighs less than 6 lbs. This level of portability is ideal for studying older populations who are often difficult to recruit to come to the Utah State University campus for participation in research studies. Many older adults have limited mobility, particularly among the group we are most interested in studying – older adults with cognitive impairment due to strokes. Having a portable system to measure motor function is a valuable way to collect data from this understudied population.

3.3 Total Cost

Total cost for the project was $200.07 – see Figure 6 for specific item price information. The bulk of the cost came from the price of the software we chose to use. At the time of writing, MATLAB costs $50 for a student copy and $150 for a home-use copy. Decreasing the total cost could be achieved by using a free-ware version of MATLAB such as Octave. Octave has a fairly large online support base and very similar verbiage to MATLAB, so it is ideal for both those who are new to programming and those who have extensive MATLAB experience.

CHAPTER 4

Discussion

We have tried several alterations to our original design, with varying degrees of success. After several trials, the aluminum foil in the targets started to tear because of the sharp stylus. If the participant happened to hit the stylus in the tear, the MaKey MaKey didn't always count the target touch as a closed circuit, and the trial time was falsely elevated. To solve this problem, we replaced the aluminum backing with a galvanized steel sheet of the same size.

Moreover, the paper tended to peel off the backing (aluminum foil or galvanized steel) and impeded participant reaching by catching on the stylus during trials. We replaced the plain paper with an equally sized thin magnetic sheet. After cutting target holes out of this sheet and arranging it on top of the galvanized steel, the accuracy of testing was improved. These changes did increase the weight and price of the device – 0.6 lbs more and $3.43 more – but the increased durability may outweigh these drawbacks.

We had some compatibility (Windows to Mac) issues with the Excel-append code ("xlswrite"). In future iterations, we will instead use MATLAB's native CSV-writing commands (e.g. csvwrite, dlmwrite). CSVs retain simple formatting and can be opened in Excel for later analysis, but are compatible across many platforms.

	Current Prototype	Future Device
Hardware	MaKey MaKey basic kit ($50)	MaKey MaKey basic kit ($50)
Target Design	Aluminum Foil/Paper ($0.07)	Galvanized Steel/Magnetic Sheet ($3.50)
Software	MATLAB ($150)	Octave (Free)
Total Cost	**$200.07**	**$53.50**

Figure 6. Item-by-item price comparison of the current device versus future prototypes with suggestions to decrease costs

REFERENCES

Broeks, J. G., Lankhorst, G. J., Rumping, K., & Prevo, A. J. H. (1999). The long-term outcome of arm function after stroke: results of a follow-up study. *Disability and Rehabilitation*, *21*(8), 357–364.

CDC, NCHS. Underlying Cause of Death 1999-2013 on CDC WONDER Online Database, released 2015. Data are from the Multiple Cause of Death Files, 1999-2013, as compiled from data provided by the 57 vital statistics jurisdictions through the Vital Statistics Cooperative Program. Accessed Feb. 3, 2015.

Flash, T., & Hogan, N. (1985). The coordination of arm movements: An experimentally confirmed mathematical model. The Journal of Neuroscience : The Official Journal of the Society for Neuroscience, 5(7), 1688-1703.

Gowland, C. (1982). Recovery of motor function following stroke: Profile and prediction. Physiotherapy Journal. 82(34), 77–84.

Hall MJ, Levant S, DeFrances CJ. Hospitalization for stroke in U.S. hospitals, 1989–2009. NCHS data brief, No. 95. Hyattsville, MD: National Center for Health Statistics; 2012.

Hillman, E. M., Hebden, J. C., Schweiger, M., Dehghani, H., Schmidt, F. E., Delpy, D. T., & Arridge, S. R. (2001). Time resolved optical tomography of the human forearm. *Physics in Medicine and Biology*, *46*(4), 1117–1130.

Krebs, H. I., Hogan, N., Aisen, M. L., & Volpe, B. T. (1998). Robot-aided neurorehabilitation. *IEEE Transactions on Rehabilitation Engineering: A Publication of the IEEE Engineering in Medicine and Biology Society*, *6*(1), 75–87.

Lloyd-Jones, D., Adams, R. J., Brown, T. M., Carnethon, M., Dai, S., De Simone, G., … Wylie-Rosett, J. (2010). Heart Disease and Stroke Statistics-2010 Update A Report From the American Heart Association. *Circulation, 121*(7), E46–E215.

Lo, A. C., Guarino, P. D., Richards, L. G., Haselkorn, J. K., Wittenberg, G. F., Federman, D. G., … Peduzzi, P. (2010). Robot-Assisted Therapy for Long-Term Upper-Limb Impairment after Stroke. *New England Journal of Medicine*, *362*(19), 1772–1783.

McCrea, P. H., Eng, J. J., & Hodgson, A. J. (2005). Saturated muscle activation contributes to compensatory reaching strategies after stroke. Journal of Neurophysiology, 94(5), 2999-3008.

Morone, G., Iosa, M., Bragoni, M., De Angelis, D., Venturiero, V., Coiro, P., … Paolucci, S. (2012). Who may have durable benefit from robotic gait training?: a 2-year follow-up randomized controlled trial in patients with subacute stroke. *Stroke; a Journal of Cerebral Circulation*, *43*(4), 1140–1142. http://doi.org/10.1161/STROKEAHA.111.638148

Mozaffarian D, Benjamin EJ, Go AS, et al. (2015). Heart disease and stroke statistics—2015 update: a report from the American Heart Association. Circulation. 131(4), 29-322.

Norouzi-Gheidari, N., Archambault, P. S., & Fung, J. (2012). Effects of robot-assisted therapy on stroke rehabilitation in upper limbs: systematic review and meta-analysis of the literature. *Journal of Rehabilitation Research and Development*, *49*(4), 479–496.

Papathanasiou, I., Filipović, S. R., Whurr, R., & Jahanshahi, M. (2003). Plasticity of motor cortex excitability induced by rehabilitation therapy for writing. *Neurology*, *61*(7), 977–980.

Rohrer, B., Fasoli, S., Krebs, H. I., Volpe, B., Frontera, W. R., Stein, J., et al. (2004). Submovements grow larger, fewer, and more blended during stroke recovery. Motor Control, 8(4), 472-483.

Volpe, B. T., Krebs, H. I., & Hogan, N. (2003). Robot-aided sensorimotor training in stroke rehabilitation. *Advances in Neurology*, *92*, 429–433.

Wagner, T. H., Lo, A. C., Peduzzi, P., Bravata, D. M., Huang, G. D., Krebs, H. I., … Guarino, P. D. (2011). An economic analysis of robot-assisted therapy for long-term upper-limb impairment after stroke. *Stroke; a Journal of Cerebral Circulation*, *42*(9), 2630–2632.

All pictures taken by the author unless otherwise specified.

APPENDIX A

Author Bio

Karen Tew graduated from Utah State University with a Bachelor of Science in Biology and minors in Chemistry and Public Health. She was heavily involved with undergraduate research, and worked in a variety of academic, commercial, clinical and government laboratories. She will be completing her graduate work in bioinformatics following a biological engineering internship at Sandia National Laboratories.

APPENDIX B

Reflective Writing

The bulk of my time on this project was spent learning how to use MATLAB. As a biology major, I had never taken any computer science classes, but I realized that coding was a useful skill in an increasingly tech-savvy world. I started by picking out a book to get the bulk of my MATLAB learning from – "Basics of MATLAB and Beyond" by Andrew Knight. I worked through the basic tutorials that were listed in that book to get a general idea of how MATLAB could work for me. Then, after getting specific instructions and constraints about the project from my advisor, I started working on the program myself. I found the resources online at the official MATLAB (mathworks.com) support website to be valuable in finding solutions to the problems that cropped up whilst coding. The "doc" and "help" commands in MATLAB were also helpful when trying to understand operators and contingencies (e.g. opening a document in read-only vs. opening a document to re-write) for your commands.

As with most coding projects, I found that I learned the best by trial and error. The program got done bit by bit as I troubleshot each successive error message. Most of the time, the errors were nominal and easily fixed, like missing parentheses or mislabeled variables. The difficult errors to solve involved Mac/PC compatibility issues and differentiating between "string", "cell array" and "matrix" values when exporting to Excel. Luckily, most of these problems have already been solved, and the creators have posted their solutions online at the MATLAB support website. With a little modification (and due credit), I incorporated their fixes into my own code and avoided wasting time on unnecessary work.

After learning MATLAB for this project, I had the skills to contribute to other related projects in my lab. We had collected some 3D reaching data sets, and I used parts of this code to create a program to analyze that data and export it to Excel in a similar way. While there are differences in the details, many motion-monitoring activities have the same principles (measuring trial time and efficiency). Thus, creating a useful, user-friendly code for this MaKey MaKey project has helped me complete successive projects much more easily. Taking pictures of my work and saving my code were important in helping me build a portfolio to take to job interviews.

For any future honors students who are interested in learning to code or completing a coding project for your thesis, allow me to offer you some advice.

First: I found that I was much more productive when I devoted a solid chunk of time (4-6 hours) to working on my project. Especially when I was first learning how to use MATLAB, I found that I couldn't get much done in only 1 hour, and what work I did would be mostly forgotten by my next session. I would waste 15-30 minutes trying to remind myself what I had worked on last, and it wasn't an efficient use of my time. I also found it helpful to start out my projects by graphically drawing what I wanted my project to look like. I'd spend 3-5 minutes at the beginning of each work session thinking about what elements I needed to add to my existing code and drawing/writing those elements out in plain English so that I could better translate them into MATLAB code later on.

Second: Do yourself and future users a favor - add a lot of descriptive comments to your code. Explain exactly what each part of your code does so that you can spend less time reminding yourself what you did, and so that others who don't have as much coding experience can easily understand what's happening. Including some potential troubleshooting information can be helpful, too. For instance, in my code I was sure to comment on exactly what parts the user needs to change before running each .m file. This saved my coworkers a lot of trouble. Formatting your code can also be a nice way to clarify your meaning. Spend an hour figuring out what customization options MATLAB has for .m files and use the ones you think are most useful. These can include things like: changing text color, indenting, or inserting line separators.

Third: Save often, and save multiple versions. It was useful to me to save my work as several different iterations rather than replacing the same file with every save. Sometimes my code stopped working and it was useful to go back to previous versions and see what I had changed. Plus, if your advisor decides to take the project in a different direction, you can do back to your earlier (presumably more general) codes and not have to start clear from scratch.

As for writing the thesis, I'd encourage future students to read A LOT of background research. The more papers you read, the better you will write and the less you will have to work at providing a review of your topic. If you have a general understanding of the background of your project, figuring out what order to put your paragraphs in becomes easy. Try to make your literature review section tell a story, work your way down from broad ideas to your small, specific project. Make it easy to read – the less the reader has to 'work' to understand your objectives and results, the better they will like your paper and the more likely they will actually read through it in its entirety. My most important piece of advice to future honors students is this: go get started. It isn't fun to have "write my thesis" on your to-do list for an entire semester, hanging over you and constantly being in the back of your mind. Trust me – if you start with a good format and outline, work on the main writing portion a little bit each week, continue reading related papers, and ask for other people to proofread it often, your thesis will be done quickly and with minimal stress.