NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

UNMANNED SYSTEMS: A LAB-BASED ROBOTIC ARM FOR GRASPING PHASE II

by

Pedro R. Hayden

December 2016

Thesis Advisor:	Richard M. Harkins
Second Reader:	Gamani Karunasiri

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704–0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2016	**3. REPORT TYPE AND DATES COVERED** Master's thesis	
4. TITLE AND SUBTITLE UNMANNED SYSTEMS: A LAB-BASED ROBOTIC ARM FOR GRASPING PHASE II			**5. FUNDING NUMBERS**
6. AUTHOR(S) Pedro R. Hayden			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number: 2017.0023-DD-N.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

A 6 Degrees of Freedom (DOF) Leap Motion Controller (LMC) was characterized in position and accuracy for robotic arm control. Tests were conducted with linear and planar trajectories for input into the Kinova Jaco interface. The objective was to produce an intuitive and adaptive system that mimicked natural hand motion. Algorithms in C++ were produced to translate the LMC Cartesian position information to the Jaco Arm reference frame. Data showed that the LMC detector was quite sensitive to human hand jitter. Post-processing low-pass Fast Fourier Transform (FFT) filter techniques were employed to mitigate this problem. The LMC hand motion volume parameters were empirically scaled, in the Cartesian frame, to match Jaco motion operational requirements. It was determined that the LMC can be successfully used as an input device for the Jaco robotic arm control. Robotic arm trajectory latency issues were negligible when the Jaco Arm parameters for displacement trajectory rates were not violated and this was successfully managed in program code and user visual input.

14. SUBJECT TERMS robotics, 6 DOF Jaco Arm, Leap Motion Controller, inverse kinematics, DH parameters.			**15. NUMBER OF PAGES** 89
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540–01-280-5500 Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

UNMANNED SYSTEMS: A LAB-BASED ROBOTIC ARM FOR GRASPING PHASE II

Pedro R. Hayden
Lieutenant Junior Grade, Peruvian Navy
B.S., Escuela Naval del Peru, 2011

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED PHYSICS

from the

NAVAL POSTGRADUATE SCHOOL
December 2016

Approved by: Richard M. Harkins
Thesis Advisor

Gamani Karunasiri
Second Reader

Kevin Smith
Chair, Department of Physics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

A 6 Degrees of Freedom (DOF) Leap Motion Controller (LMC) was characterized in position and accuracy for robotic arm control. Tests were conducted with linear and planar trajectories for input into the Kinova Jaco interface. The objective was to produce an intuitive and adaptive system that mimicked natural hand motion. Algorithms in C++ were produced to translate the LMC Cartesian position information to the Jaco Arm reference frame. Data showed that the LMC detector was quite sensitive to human hand jitter. Post-processing low-pass Fast Fourier Transform (FFT) filter techniques were employed to mitigate this problem. The LMC hand motion volume parameters were empirically scaled, in the Cartesian frame, to match Jaco motion operational requirements. It was determined that the LMC can be successfully used as an input device for the Jaco robotic arm control. Robotic arm trajectory latency issues were negligible when the Jaco Arm parameters for displacement trajectory rates were not violated and this was successfully managed in program code and user visual input.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

API	Application Program Interface
DOF	Degrees of Freedom
DH	Denavit Hartenberg
FFT	Fast Fourier Transform
IDE	Integrated Development Environment
LED	Light Emitting Diode
LMC	Leap Motion Controller
RML	Robotic Manipulation Laboratory
SDK	Software Development Kit
USB	Universal Serial Bus

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my thesis advisor and second reader for their support and encouragement.

I would like to acknowledge my mother as being my inspiration and role model.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION AND OBJECTIVES

Sophisticated robots have become a part of everyday life. They are used to perform tasks that are difficult for humans to accomplish. This includes but is not limited to tedious precision tasks or operations in hazardous environments. Robotic arms, for example, in the tele-op mode, have been around for decades in factory assembly lines. Today their use spans a spectrum of applications from machine loading, pick-and-place operations, assembly, manufacturing, medical applications and assisting disabled individuals as shown by Niku [1].

The 6 DOF Jaco Arm is a versatile robotic arm designed by Kinova. This arm is made of carbon fiber and uses harmonic gear motors to move silently and smoothly. According to [2] “The robot has been used for empowering people with disabilities to push beyond their current boundaries and limitations.” For example, after recognizing the challenges that elderly and impaired people face and the potential of robotics, Bassily [3] proposed the implementation of an ambient assisted living (AAL) using the Jaco Arm controlled by a new gesture and position tracking system with sub-millimeter accuracy.

The Naval Postgraduate School Physics Department has implemented a robotic manipulation laboratory (RML) to explore learning opportunities for student lab experiments [4]. To date an intuitive user interface has not been fully realized. Jacinto’s [5] work focused on understanding and applying the kinematics and dynamics of a 6 DOF Jaco Arm in a controlled lab environment for different predetermined tasks under a glove sensor controller. In his research, the main goal was to integrate the Jaco Arm with a glove sensor controller in an effective, real-time, and human-like system. He successfully set up the Jaco software and showed standalone operation using the Jaco software and the Kinova joystick. In addition, he effectively completed tests on the glove sensor controller that simulated hand movements. However, he was not able to interface the robotic arm with the glove controller.

Palacios [6] continued Jacinto's work by implementing a virtual joystick and testing it on Jaco's Arm capabilities and performance to lift objects with different features. In his research, Palacios proved the value of employing the virtual joystick instead of the Kinova joystick for cyclic tasks using preplanned trajectories.

The purpose of this research is to follow up Palacios and Jacinto's work and demonstrate real time control between the 6 DOF Jaco Arm and the Leap Motion Controller in an intuitive and adaptive system in which the trajectory motion emulates human hand movements. To achieve this, we explain the kinematics of the Jaco Arm in a controlled lab environment for various predetermined tasks under the Leap Motion Controller.

The secondary goal is to control the Jaco Arm remotely using a wireless network. This will empower a future platform that hosts the Jaco Arm with new features, such as manipulating objects with precision.

II. THEORY

In this chapter, we will look at the fundamental physics that govern our robotic actuator. Inverse kinematics and Denavit-Hartenberg (DH) parameters will be briefly explained.

A. POSITION ANALYSIS

According to [3] and [7], there are two ways to model the behavior of the robotic arm. In the first method, we determine where the robot's end effector (hand) is if all joint actuator variables are known. The second method enables us to locate the hand at a particular point and a particular orientation. This is the "inverse kinematic" method and allows us to calculate the actuator's position in order to move the robot's end effector to a specific point in space.

B. DENAVIT-HARTENBERG PARAMETERS

Robots are made of a succession of joints and links in different dispositions [3]. In the specific case of the Jaco Arm, the joints are revolute (rotational), move in different planes, have offsets, and the links have fixed lengths. The DH matrix representation is a technique used to analyze and model robot links and joints for various robot configurations [3]. We used DH parameters to represent transformations in Cartesian coordinates. However, it is also possible to use it in cylindrical, spherical or Euler coordinates. DH representation of a general purpose joint-link combination is shown in Figure 1. The general procedure defined by [3] and [7] to represent a transformation matrix using the DH matrix follows.

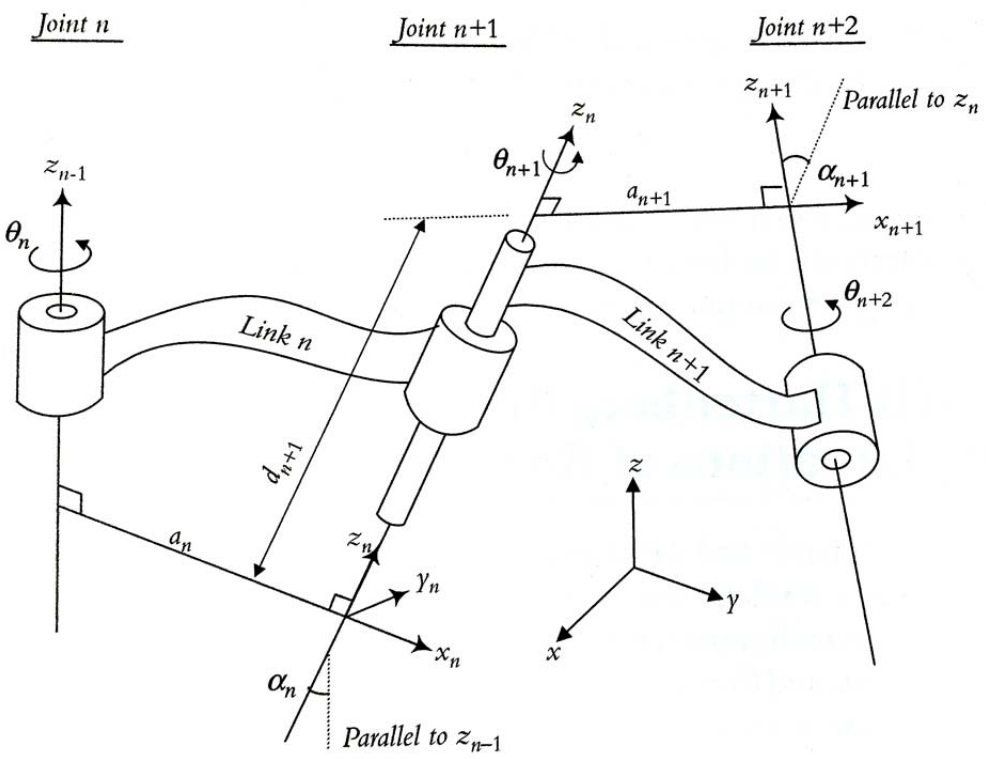


Figure 1. DH Representation of a General Purpose Joint-Link Combination. Source: [7].

1. Assign a reference frame to each joint.
   - Assign a local reference frame for each and every point.
   - Assign z-axis and x-axis to each joint.
   - Index number for the z-axis of the joint n is n-1.
   - There is always one link perpendicular to any two skew links.
   - Assign the x-axis of the local reference frame in the direction of the common normal.
   - a_n is the common normal between z_n and z_{n-1}.
   - x_n is along a_n.
2. Define a general procedure to transform from one joint to another.
   - Rotate about the z_n-axis an angle θ_{n+1}.
   - Translate along the z_n-axis a distance d_{n+1}.

- Translate along the x_n-axis an angle α_{n+1}.

The matrix ${}^{n}T_{n+1}$ (see Equation 1) between two successive frames represents the steps mentioned previously.

$$
\begin{aligned}
{}^{n}T_{n+1} &= A_{n+1} = Rot(z,\theta_{n+1}) \times Trans(0,0,d_{n+1}) \times Trans(a_{n+1},0,0) \times Rot(x,\alpha_{n+1}) \\
&= \begin{pmatrix} C\theta_{n+1} & -S\theta_{n+1} & 0 & 0 \\ S\theta_{n+1} & C\theta_{n+1} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_{n+1} \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & a_{n+1} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & C\theta_{n+1} & -S\alpha_{n+1} & 0 \\ 0 & S\theta_{n+1} & C\alpha_{n+1} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (1)
\end{aligned}
$$

$$
A_{n+1} = \begin{pmatrix} C\theta_{n+1} & -S\theta_{n+1}C\alpha_{n+1} & S\theta_{n+1}C\alpha_{n+1} & a_{n+1}C\theta_{n+1} \\ S\theta_{n+1} & C\theta_{n+1}C\alpha_{n+1} & -C\theta_{n+1}S\alpha_{n+1} & a_{n+1}S\theta_{n+1} \\ 0 & S\alpha_{n+1} & C\alpha_{n+1} & d_{n+1} \\ 0 & 0 & 0 & 1 \end{pmatrix}
$$

THIS PAGE INTENTIONALLY LEFT BLANK

III. EXPERIMENTAL SETUP

The experimental set up is shown in Figure 2. The 6 DOF Jaco Arm, the Leap Motion Controller and the computer communicate via serial connections. The software development kit (SDK) and the Cartesian control method is the basis for the algorithm used to control the robotic arm.

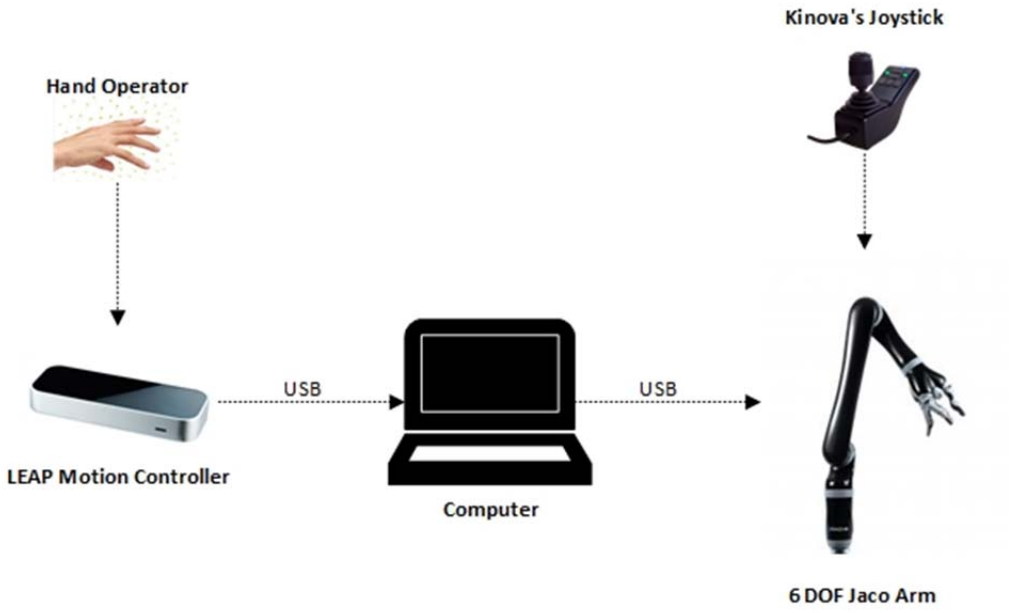


Figure 2. Overall System Containing the 6 DOF Jaco Arm, Leap Motion Controller and Communication Flow. Adapted from [8] and [9].

A. IMPLEMENTATION

The Jaco Arm and Leap Motion Controller interface, shown in Figure 3, was implemented using the C++ programming language through the integrated development environment Visual Studio 2013. The algorithm receives hand motion input in Cartesian coordinates. The Leap sensor detects if fingers are extended or retracted during each available frame using the Leap Motion Controller. Hand position in the Cartesian coordinates x, y and z, and fingers extended or retracted are sent to the Jaco application that scales these values to

shorter ranges than 0.9 m, +/- 0.5 m and +/- 0.5 m in the x, y and z axes respectively. Each internal function of the Jaco application takes previous frame information and sends the robotic arm a new position.

To achieve real-time motion control, as a first approach, a preplanned trajectory demo was implemented. A text file with dummy hand position data was created to simulate actuator positions. This was then interpreted by the Jaco application to move the arm (see Appendix A). The second approach used the Leap Motion Controller to detect human hand and arm positions while data was written to a space delimited text file. As before, this file was interpreted by the Jaco application (see Appendix B) to move the robotic arm. These two techniques were post event processed and not real-time. To realize near real-time motion, the read/write functions were integrated into the Jaco motion control algorithm as illustrated in Appendix C.

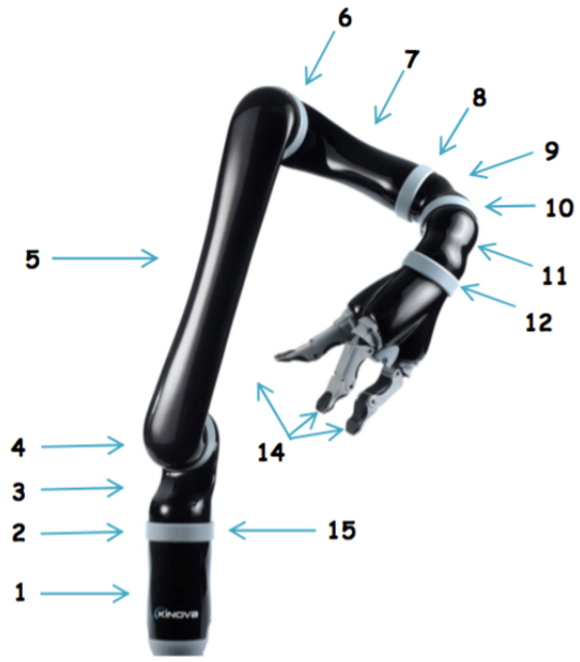
Figure 3. Jaco Arm and Leap Motion Controller Interface.

B. 6 DOF JACO ARM

The Jaco Arm specifications (See Table 1) are detailed in the *JACO Research Edition User Guide*:

> The arm has a maximum reach of 0.9 m, +/- 0.5 m and +/- 0.5 m in the x, y and z direction respectively. The three wrist joints are capable of continuous rotation. All the joints have a range limit. For example, actuator 1, 4, 5 and 6 have a range of -10,000 to +10,000 degrees, actuator 2 has a range of +42 to +318 degrees and actuator 3 has a range of +17 to +343 degrees. If we send a command past these limits, we will cause the arm to move to its hard-wired limit and consequently the Jaco Arm will stop. The range of input for all three fingers on the Jaco is approximately 0 (fully open) to 60 (fully closed). [8]

The different actuators are identified in Figure 4.



PART ID	NAME
1	Controller
2	Actuator #1
3	Shoulder
4	Actuator #2
5	Arm
6	Actuator #3
7	Forearm
8	Actuator #4
9	Wrist #1
10	Actuator #5
11	Wrist #2
12	Actuator #6
13	Gripper
14	Fingers
15	Plastic Ring

Figure 4. 6 DOF Jaco Arm Identification Parts. Source: [8].

The Jaco Arm has a multilateral action gripper as shown in Figure 5. This device has a binary action (opened or closed) and the specifications are shown in Table 2.

Figure 5. Jaco Arm Gripper. Source: [8].

Table 1. Jaco Arm Specifications. Adapted from [8].

SPECIFICATIONS
ENVIRONMENT GENERAL Ambient temperature from 0 to 30 C degrees. Can be used under light rainfall for limited period. Can be used under normal atmospheric pressure conditions. STORAGE Ambient temperature from 0 to 50 C degrees. Relative Humidity: 55% max.
ELECTRICAL INPUT POWER Voltage: 18V to 29V d.c. Current: 2A in normal use, 10A max. Powered by the external power supply. OUTPUT POWER (controller port) Voltage: 24V +/- 20% d.c. Current: 1.5A continuous, 3A max. POWER SUPPLY Input power: 100V to 240V a.c., 50Hz to 60 Hz, 2.0A Output power: 24 d.c., 5.0A
MECHANICAL GENERAL Total weight: 5.6 Kg +/- 5% Maximum Load: 1.5 Kg at mid-range (45 cm) Maximum Load: 1.0 Kg at end-range (90 cm) Reach: 90 cm Maximum linear arm speed: 15 cm/s HAND AND FINGERS 3 or 2 fingers simultaneous utilization. Independent control available for each finger. Finger force limited to 7N (1.54 lb_f). Flexible fingers for durability.
FIRMWARE Each axis controlled independently. Redundant security on each axis/fingers. Redundant error check in joints and in control system. Position and error calculation every 0.01 second. Automatic recovery on system fault.

Table 2. Jaco Arm Gripper Specifications. Adapted from [8].

SPECIFICATIONS	
FINGERS	3
ACTUATION SYSTEM	Under actuated
ACTUATORS	One per finger
OPENING (fingertip)	175 mm
MIN CYLINDRICAL GRIP	45 mm
MAX CYLINDRICAL GRIP	100 mm
TOTAL WEIGHT	727 g
GRIPPING FORCE	40 N
OPENING OR CLOSING TRAVEL TIME	1.2 s
OPERATING TEMPERATURE	-10 to 40 C degrees

C. KINOVA SDK

The Kinova software development kit is “a complete set of functions, documentation, software tools and examples that allow the operator to interact with the Jaco Arm” [10]. This section describes the use of the Kinova SDK for the experiment.

1. The Development Center

Kinova Robotics provides users a friendly interface as shown in Figure 6, which allows modifying general and advanced settings, monitoring and controlling the robotic arm by a virtual joystick or uploading trajectory points.

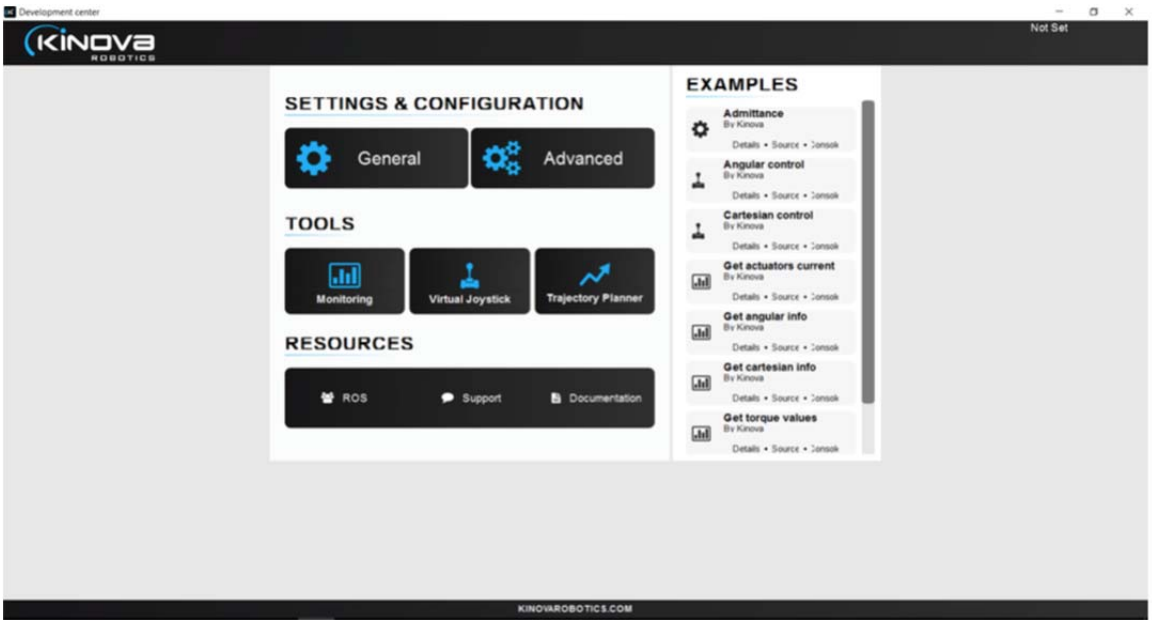


Figure 6. Kinova Development Center Main Window.

2. Major SDK Components

The Kinova SDK is comprised of functions and data structures that allow the user to store position and orientation data of the end effector with respect to the origin (base cylinder) and assign new tasks to the Jaco Arm.

a. CartesianPosition

This data structure holds the position of the end effector in Cartesian coordinates (X, Y and Z), a number from 0 to 60 for each of the three fingers and the orientation as an angle in the range of +/-6.28 rad (pitch, roll and yaw).

b. TrajectoryPoint

This data structure represents a point of a trajectory. It contains the new position, orientation and limitation that we can apply.

c. InitStruct

This method will initialize all the variables to 0.

d. Position.Type

This function represents the position type used in our program. For example, if the position type is CARTESIAN_POSITION, then the robot's end effector will move to that position using inverse kinematics. But if the position type is CARTESIAN_VELOCITY then the program requires velocity command inputs. For this experiment, we used CARTESIAN_POSITION type to command the robotic arm.

e. CARTESIAN_POSITION

This function defines a Cartesian_Position described by a translation in X, Y, Z and an orientation of the end effector with angles ThetaX, ThetaY, ThetaZ.

f. MyGetCartesianCommand

This function gets the actual X, Y and Z position of the end effector. In other words, this function allows the user to get each value of the structure CartesianPosition.

g. MySendBasicTrajectory

This function sends a trajectory point that will be added to the robotic arm.

D. LEAP MOTION CONTROLLER

According to Spiegelmock, “The Leap device uses a pair of cameras and an infrared pattern projected by LEDs to generate an image of your hand with depth information” [11]. As described in [12], The sensors are directed along the y-axis (vertical axis) when the controller is in its standard operating position as shown in Figure 7. They have a field of view of about 150 degrees and the effective range extends from approximately 25 to 600 millimeters above the device.

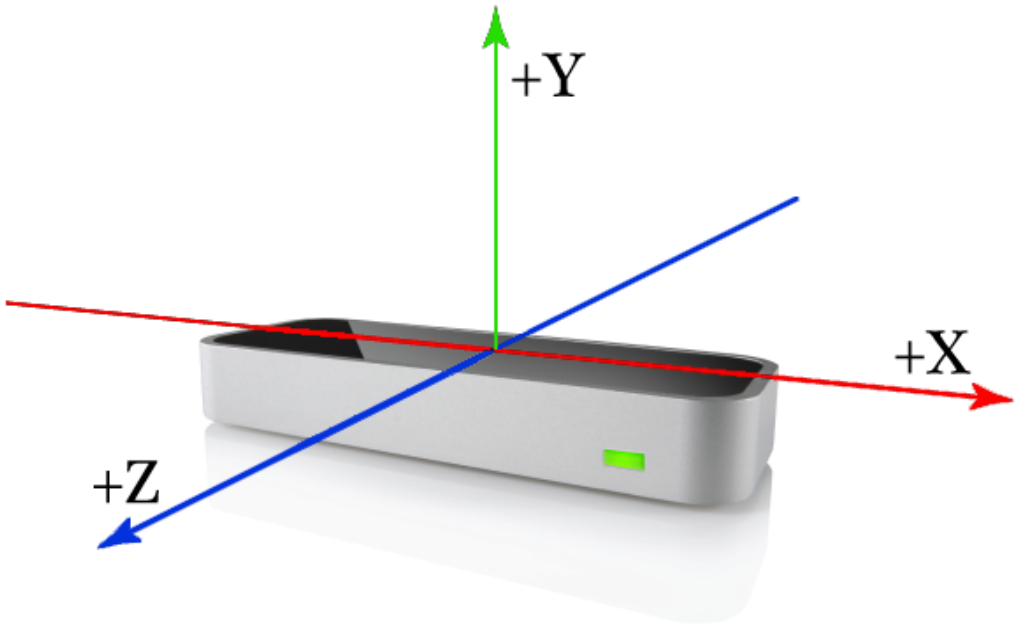


Figure 7. Leap Motion Controller in Standard Operating Position.
Source: [11].

E. LEAP SDK

This section will describe the steps to set up the files required to run the Leap SDK in Visual Studio 2012/2013. In addition, we explain the features and software interface exposed by the Leap Motion SD, which will allow us to obtain the hand motion input.

1. Set up Procedure in Visual Studio 2012/2013

The procedure presented is a shortcut adapted from Leap SDK:

- Create a new project in Visual Studio.
- Make a LEAP_SDK system environment variable to point to your Leap Motion SDK folder. In Windows Desktop, open Start Menu/ Computer/ System/ Advance system settings. Enter LEAP_SDK for variable name and Leap Motion SDK for Variable value.
- In project property page set up the debug configuration.
- Under Configuration Properties select C/C++ General.
- In properties panel, add the SDK include directory. Enter $ (LEAP_ SDK) \include.
- Under Configuration Properties, select Linker General.
- In the properties panel, add the SDK lib\x86/. Enter $(LEAP_SDK) \lib\x86.
- Select Linker Input.
- Add Leap.lib to the Additional Dependencies field.
- Under Configuration Properties, select Build Events then Post-Build Event and add xcopy /yr "$(LEAP_SDK) \lib\x86\Leap.dll" '$ (TargetDir). [13]

After these steps, the user is able to use and modify the Leap SDK examples or create his program.

2. Major SDK Components (Class)

a. Controller

The Controller class is a connection between the Leap motion device and our application that allows the user to pass tracking data in the form of frame objects [14]. According to Spiegelmock, "We can interact with the device configuration, detected displays, current and past frames, and set up event handling with our listener subclass" [11].

b. Frame

All possible data that our device can get is contained in a frame. A frame object represents a specific point in time in which the Leap controller is capable of detecting your hand as show in Figure 8. For instance, each time we call a frame object we request a series of consecutive snapshots of the Leap Motion Controller sight of view.

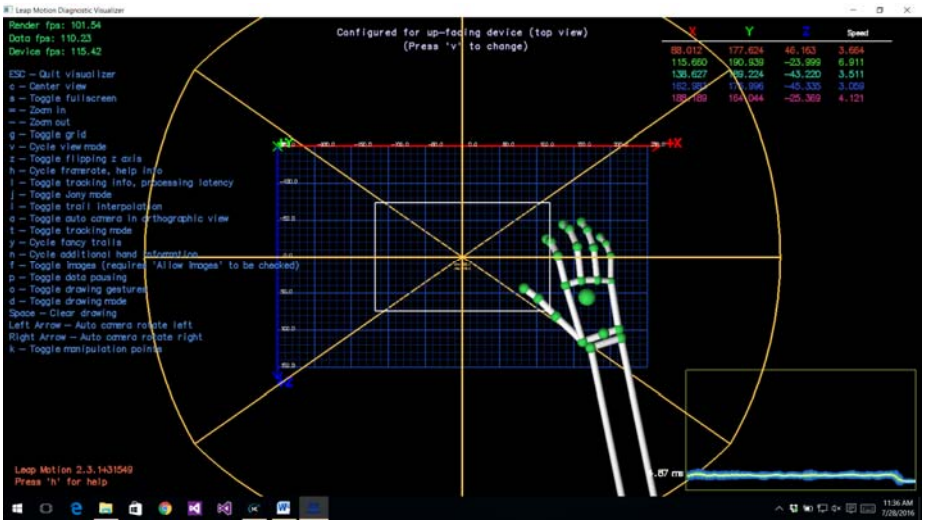


Figure 8. Leap Motion Controller Tracking Mode.

c. Hand

The only part of the human body that the Leap Motion Controller can detect are the hands. The controller has an inner model of a hand and compares it with the information received by the sensors in each frame. Furthermore, a hand object provides valuable information about the user's hand such as position, orientation and movement.

F. CARTESIAN CONTROL

The Jaco Arm and the LMC are controlled using Cartesian coordinates. In the case of the LMC, the y-axis is taken as the vertical axis (upwards or downwards) as shown in Figure 9. For this reason, an appropriate relation

between the robotic arm and the motion controller has to be set. On the other hand, the z-axis is taken as the vertical axis in the Jaco Arm and uses inverse kinematics to move the end effector. In other words, you specify a coordinate in the x, y and z direction with its orientation and the end effector goes there if it can [15].

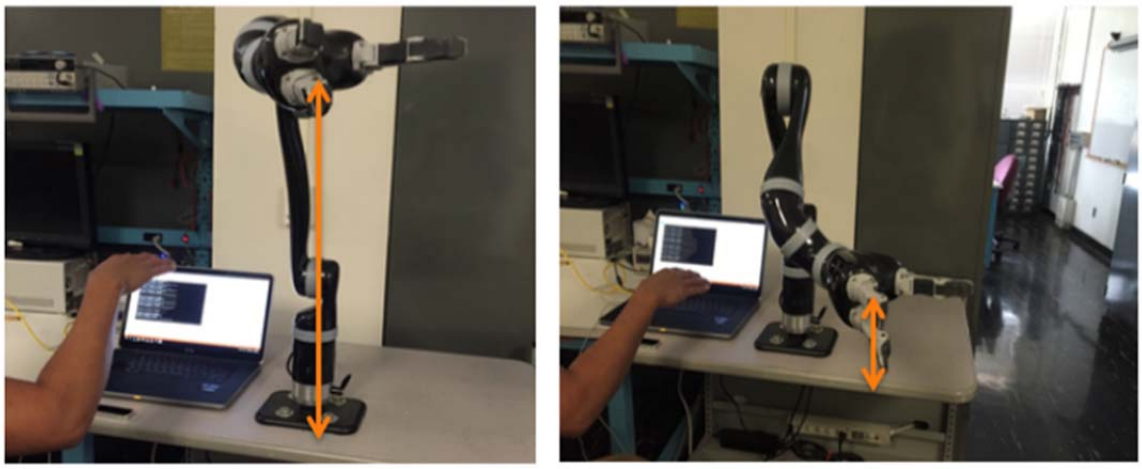


Figure 9. Cartesian Control in the Z-axis.

1. Grasp and Release Control

Grasp and release control is obtained in a similar way. The motion controller detects the condition of the hand operator (opened or closed) if only the thumb, index and pinky fingers are extended or retracted during each frame as shown in Figure 10.

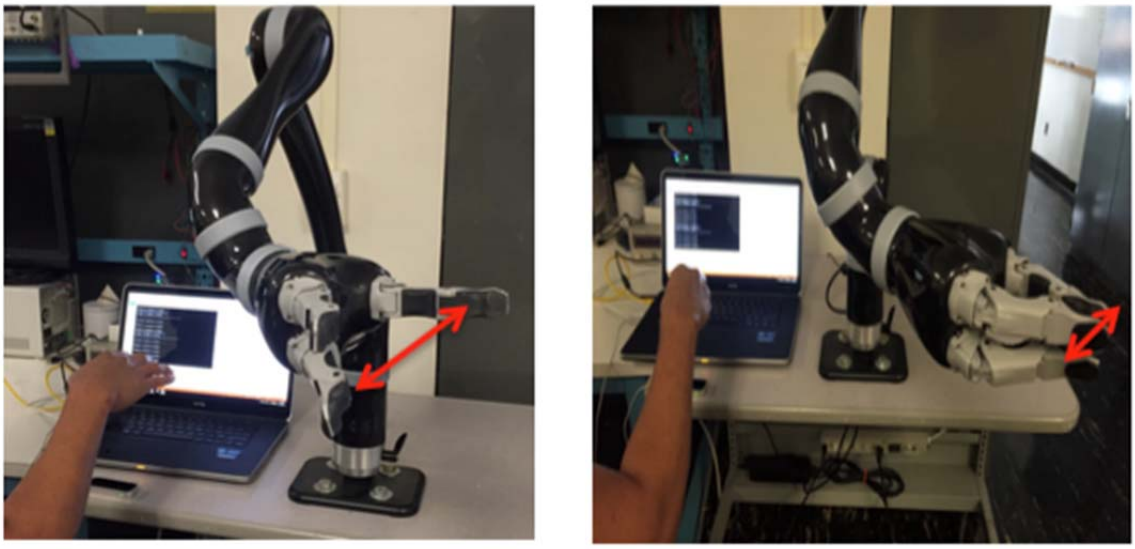


Figure 10. Grasp and Release Control.

IV. EXPERIMENTAL RESULTS

In this chapter, we discuss the results of the interface between the Jaco Arm and the Leap Motion Controller. First, we analyze the standalone operation and precision of the motion controller by getting output data in a text file and plot them using MATLAB. Second, we use a text file containing different positions, send them to the Jaco Arm application, track each new position, write them in a text file and plot them. Finally, we plot the output both from the motion controller and the robotic arm while they are interfaced operating at the same time.

A. LEAP MOTION CONTROLLER STANDALONE RESULTS

To characterize the performance of the Leap Controller, code was written to conduct various pre-determined hand motions as shown in Appendix A. Initial experiments collected using LMC data with no time delay between read and write operations. A discussion of these tests and results follows.

1. Vertical Motion Test

For this experiment, the operator moved his hand, up and down, in the Y direction. A lab jack was used as a reference frame to keep the motion as straight as possible. LMC trajectory data was collected and new position data was written to a text file. The data are plotted using MATLAB, for different views, to characterize the accuracy of the LMC.

In general, the LMC successfully tracked the vertical hand motion from a distance of 0.11 m to 0.45 m as shown in Figure 11. For hand positions less than 0.11 m, data was dispersed and not reliable. Figures 12 and 13 show offsets of 0.005 m and 0.011 m in the X and Z directions, respectively, proving that the device detects hand tremor and involuntary motion even with attempts to keep the motion vertical.

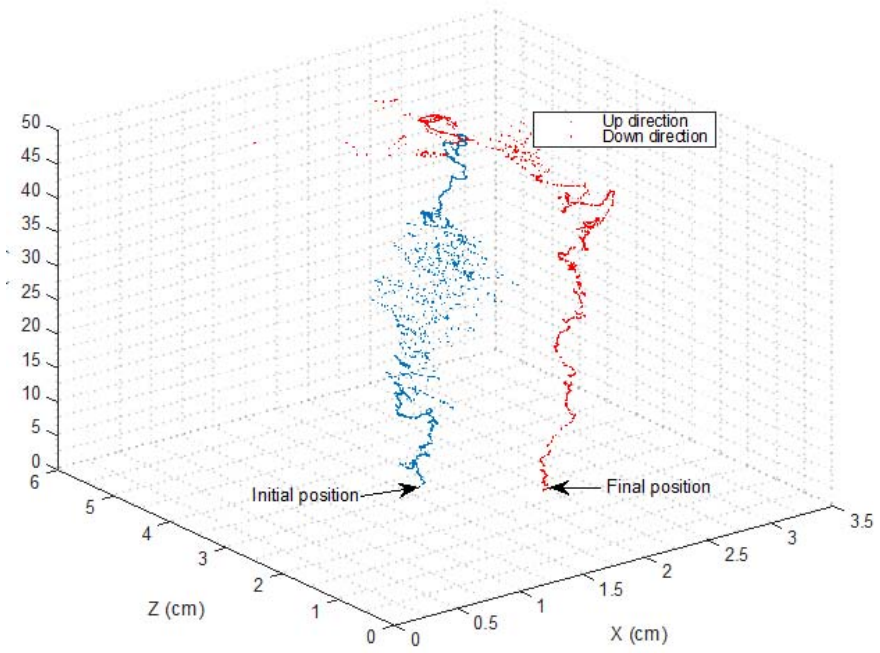


Figure 11. 3D Plot of the Leap Motion Controller Tracking Points during Vertical Motion of a Hand Operator.

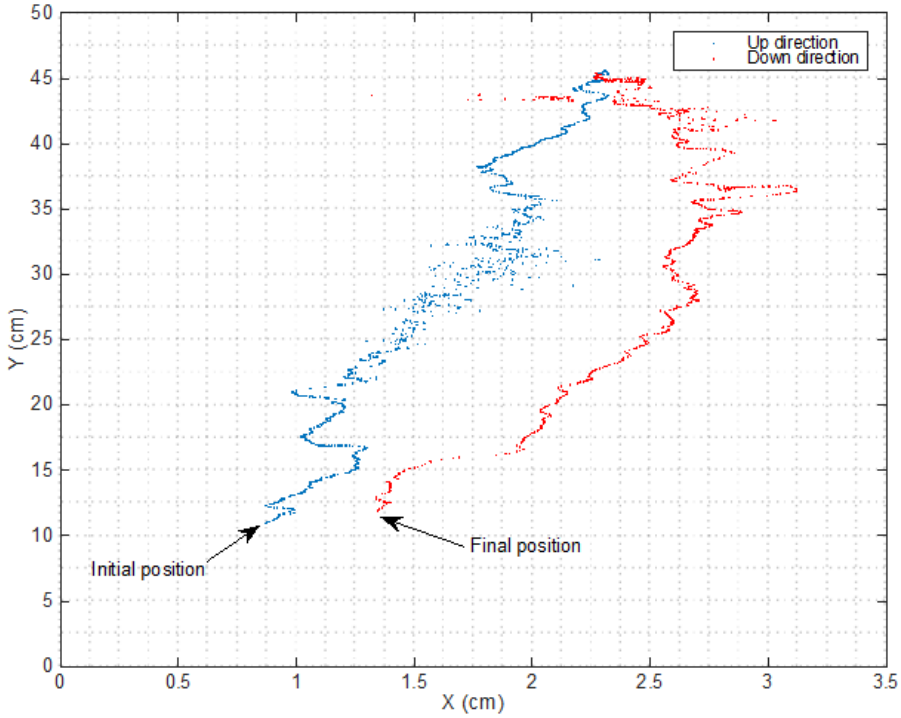


Figure 12. Front View of the Leap Motion Controller Tracking Points During Vertical Motion of a Hand Operator.

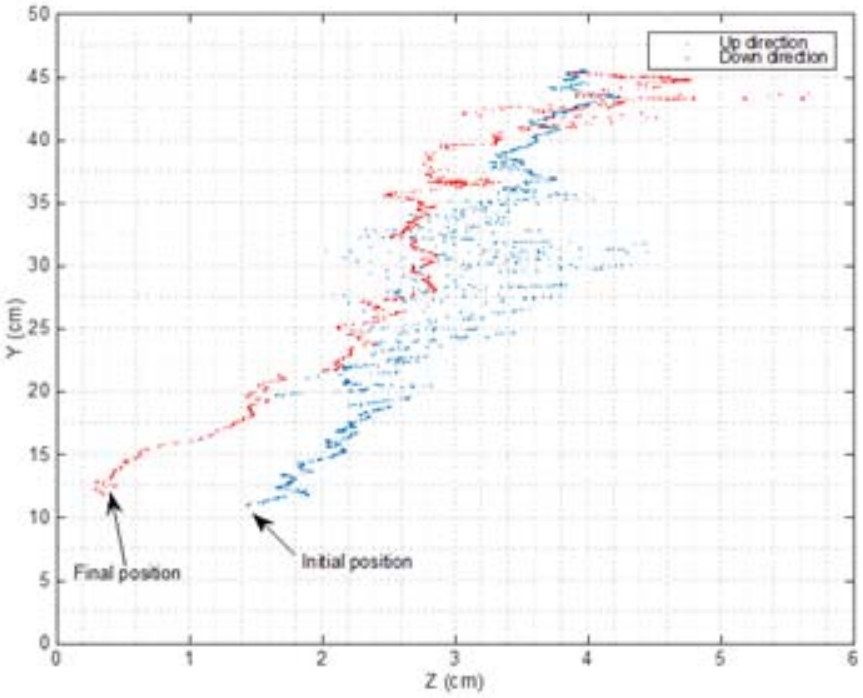


Figure 13. Rear View of the Leap Motion Controller Tracking Points During Vertical Motion of a Hand Operator.

Then, the vertical up and down trajectory was plotted and shifted a distance of 0.005 m and 0.011 m in the X and Z directions, respectively, to correct for horizontal offset, see Figure 14. The LMC detected part of the jack lab and represented it as noisy data during the up trajectory, see Figure 15. The final results are displayed in Figure 16.

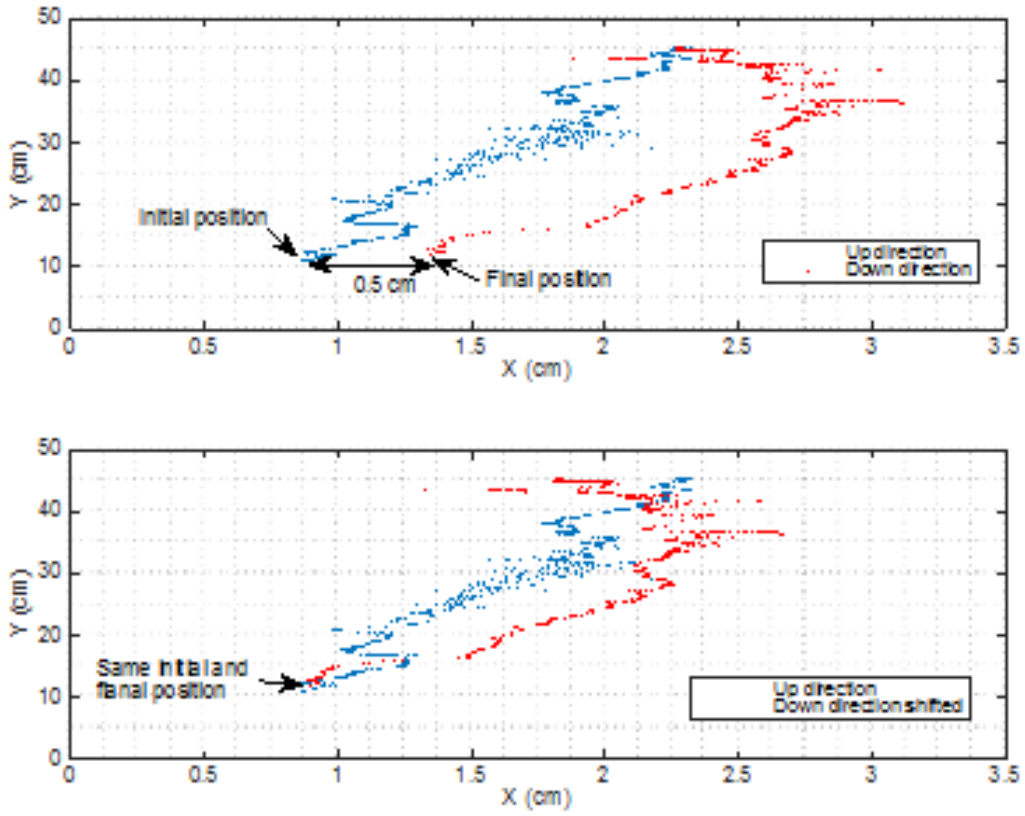


Figure 14. Shift Done in the X Direction over the Down Hand Vertical Motion.

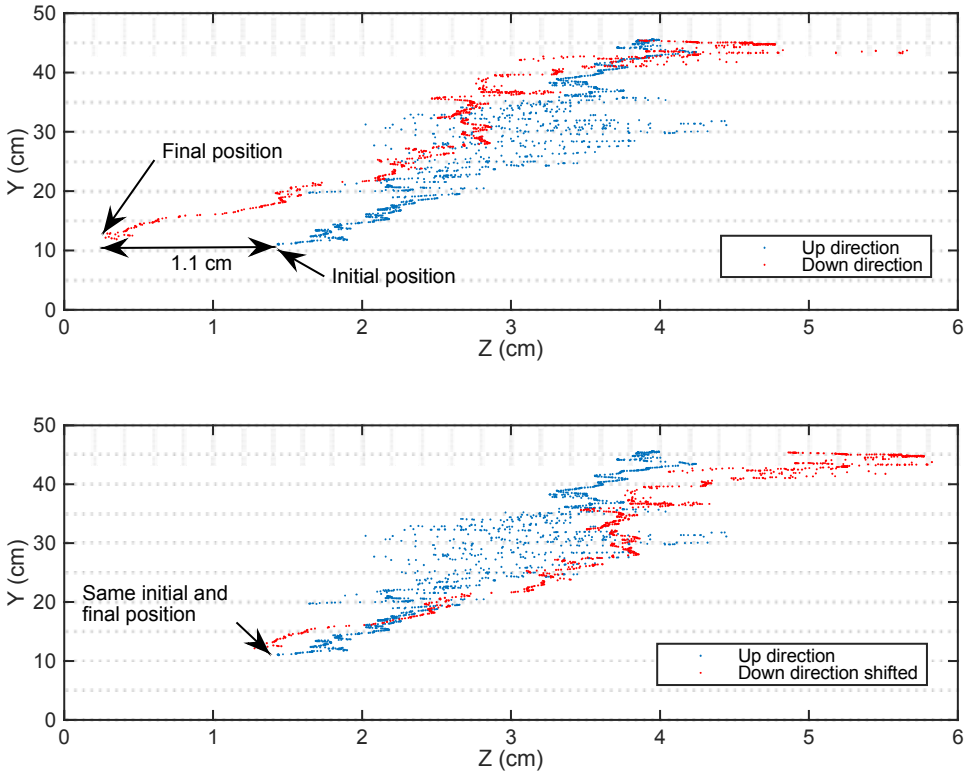


Figure 15. Shift Done in the Z Direction over the Down Hand Vertical Motion.

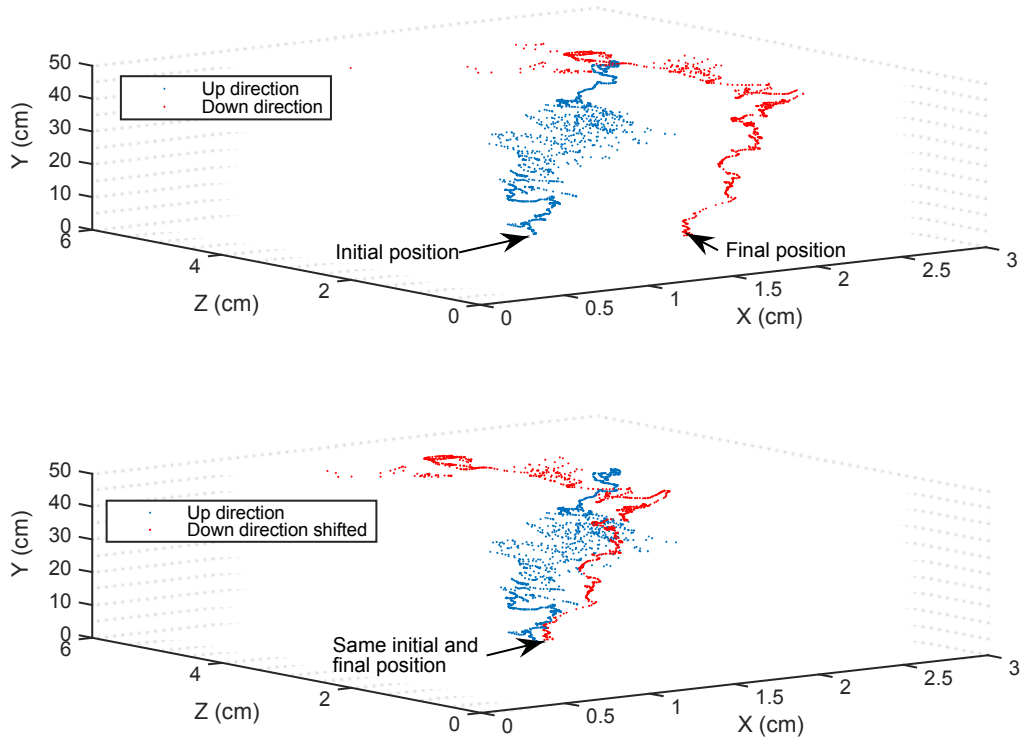


Figure 16. Final Hand Trajectory Motion in the Vertical Direction after Shifts Were Implemented.

The equations of the lines for the shifted trajectories were obtained using the curve-fitting tool in MATLAB, as shown in Figure 17, Figure 18, Figure 19 and Figure 20.

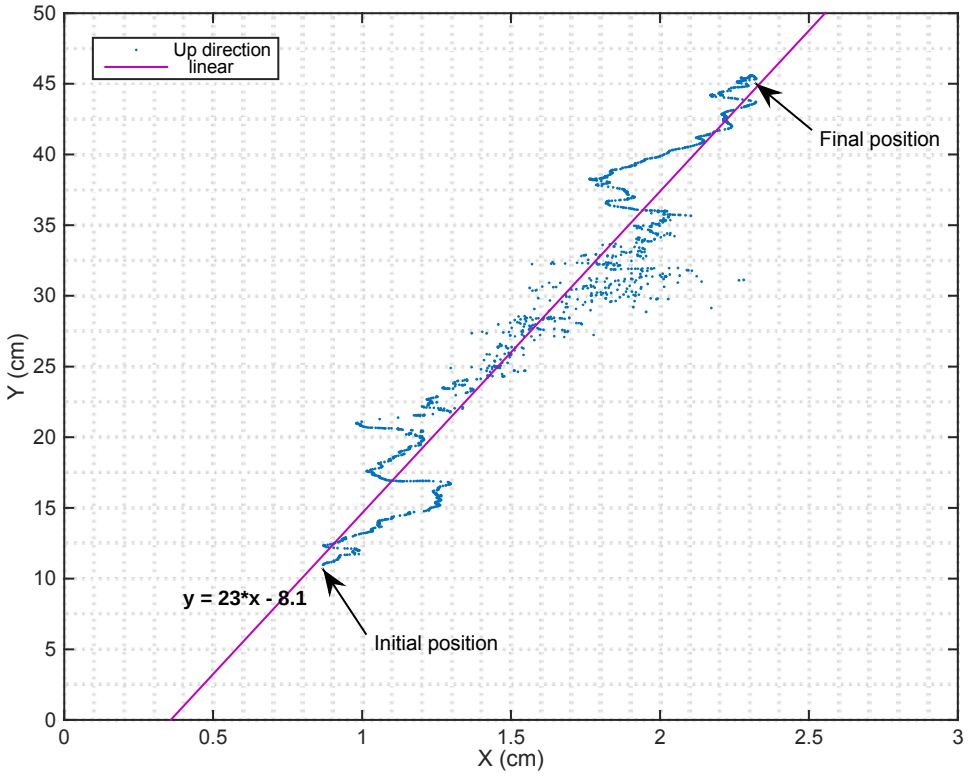


Figure 17. Equation of Line of the Up Trajectory in the X Direction.

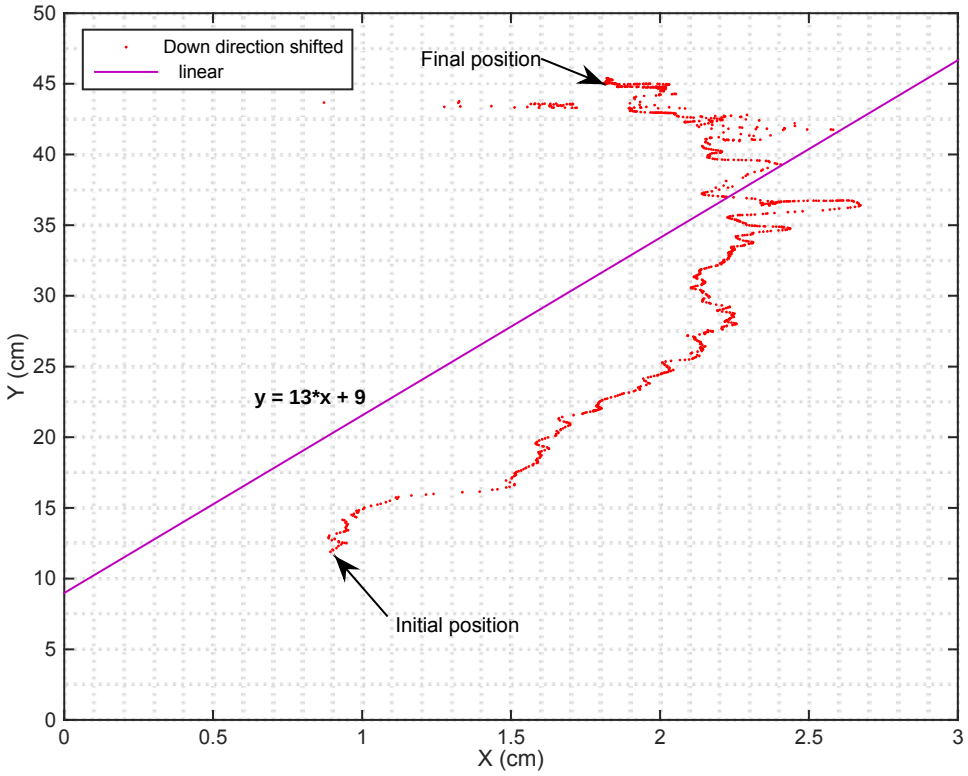


Figure 18. Equation of Line of the Down Trajectory Shifted in the X Direction.

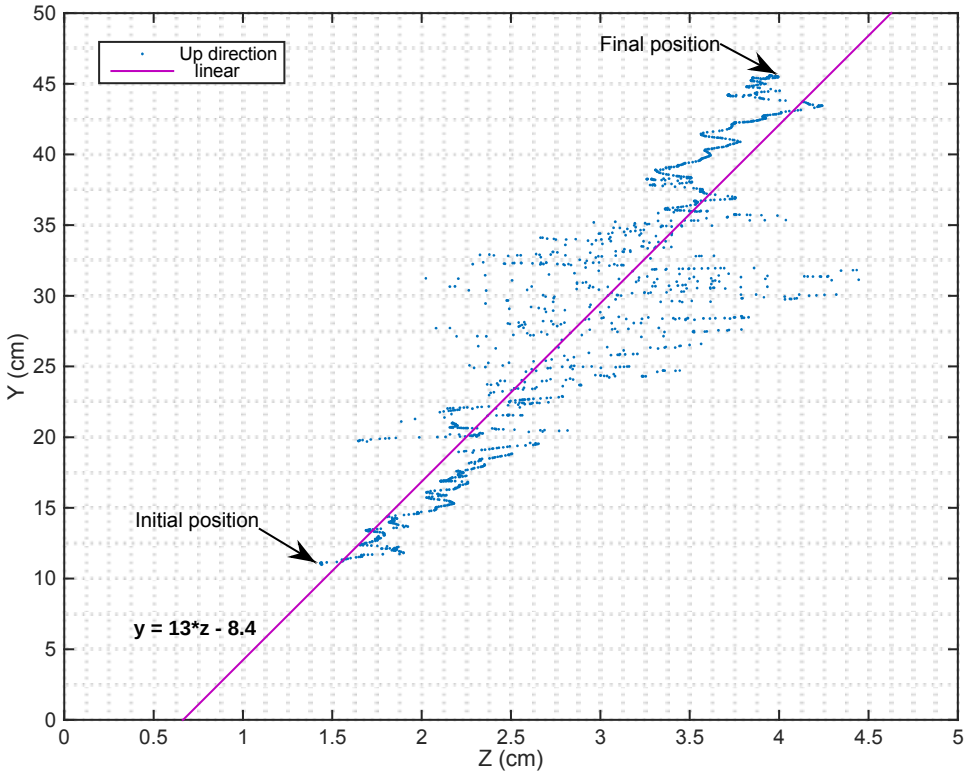


Figure 19. Equation of Line of the Up Trajectory Shifted in the Z Direction.

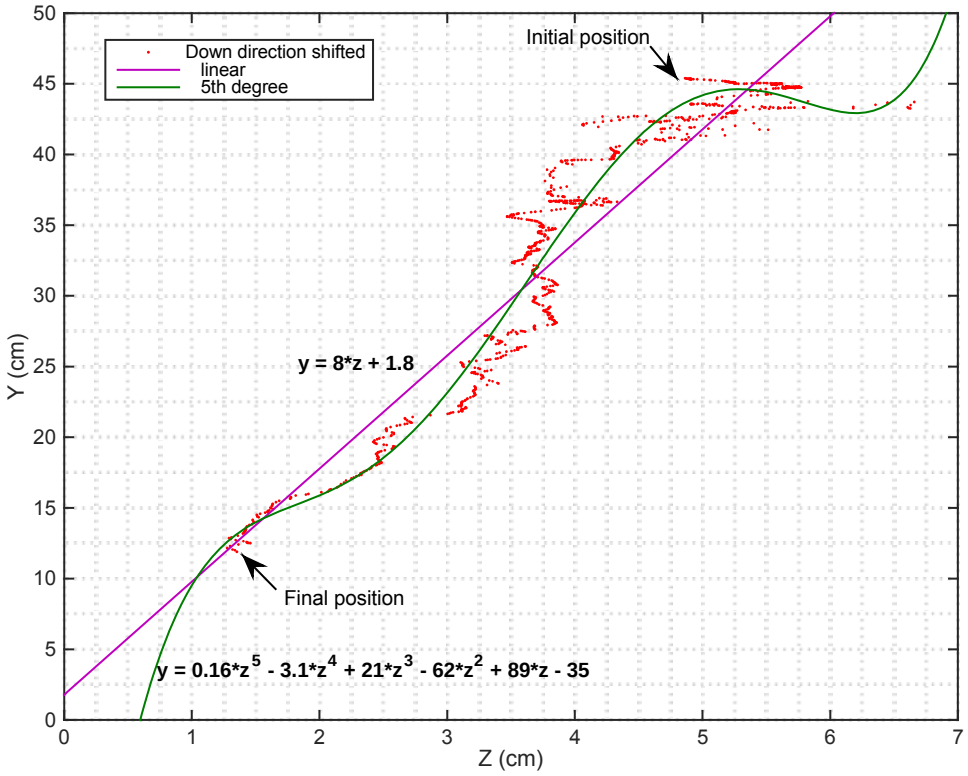


Figure 20. Equation of Line and Curve Fifth Degree Fitting Function of the Down Trajectory Shifted in the Z Direction.

In addition, position versus time graphs and their respective curve fits were obtained in order to characterize the motion in the x direction as shown in Figure 21.

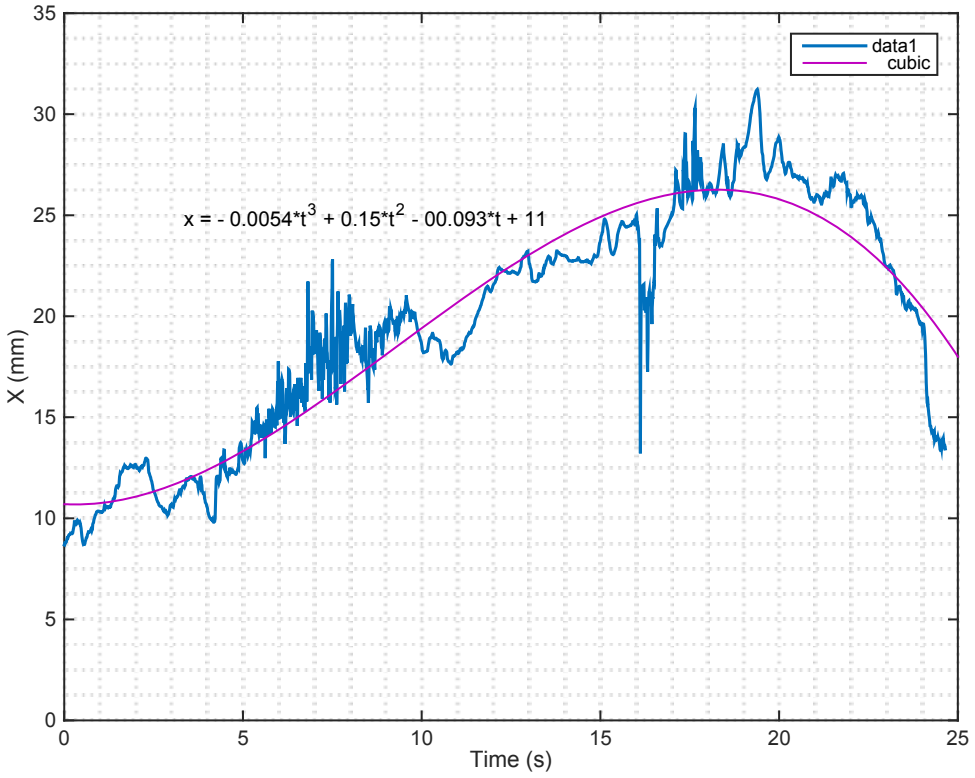


Figure 21. Position in the X Direction versus Time.

To filter for jitter, we used the Fourier transform to plot the amplitude versus frequency graph (also called spectral plot) of the trajectory in the x direction and implemented a 0.5 Hz bandpass filter to ignore higher frequencies, as shown in Figure 22. Then, the inverse Fourier transform was used to reconstruct the trajectory in the x direction, filtered, and eliminated the noise as shown in Figure 23. The same process was implemented for the y and z direction. The results are shown in Figure 24, Figure 25, Figure 26, Figure 27, Figure 28 and Figure 29.

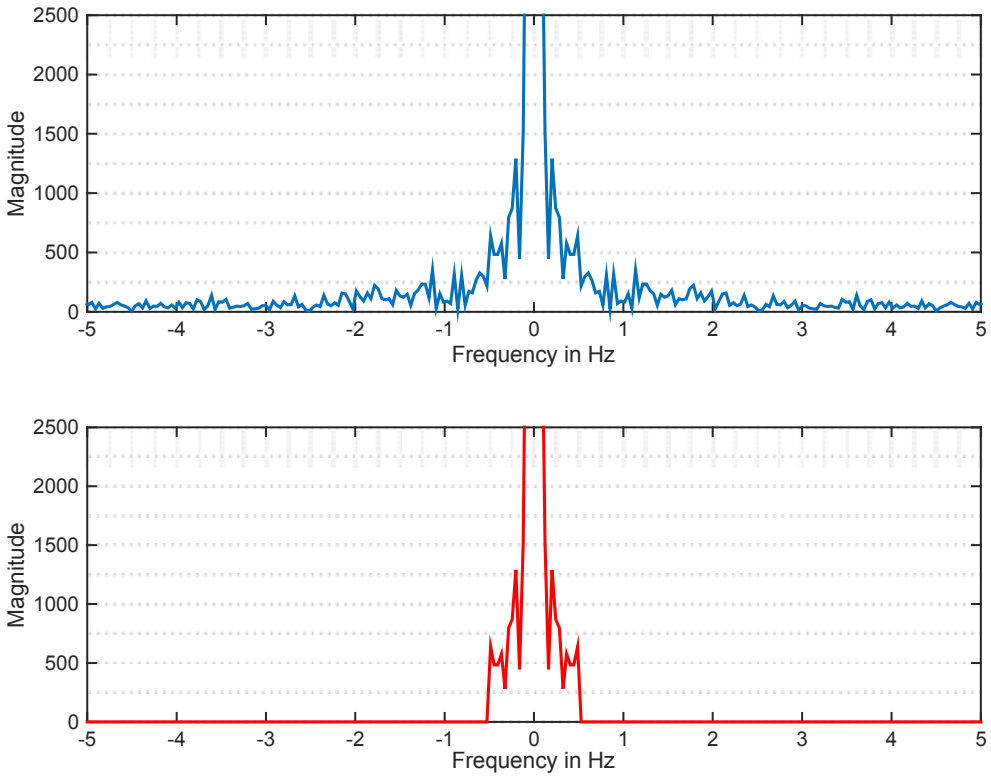


Figure 22. X Direction Spectral Plot after a Bandpass Filter for 0.5 Hz Is Applied.

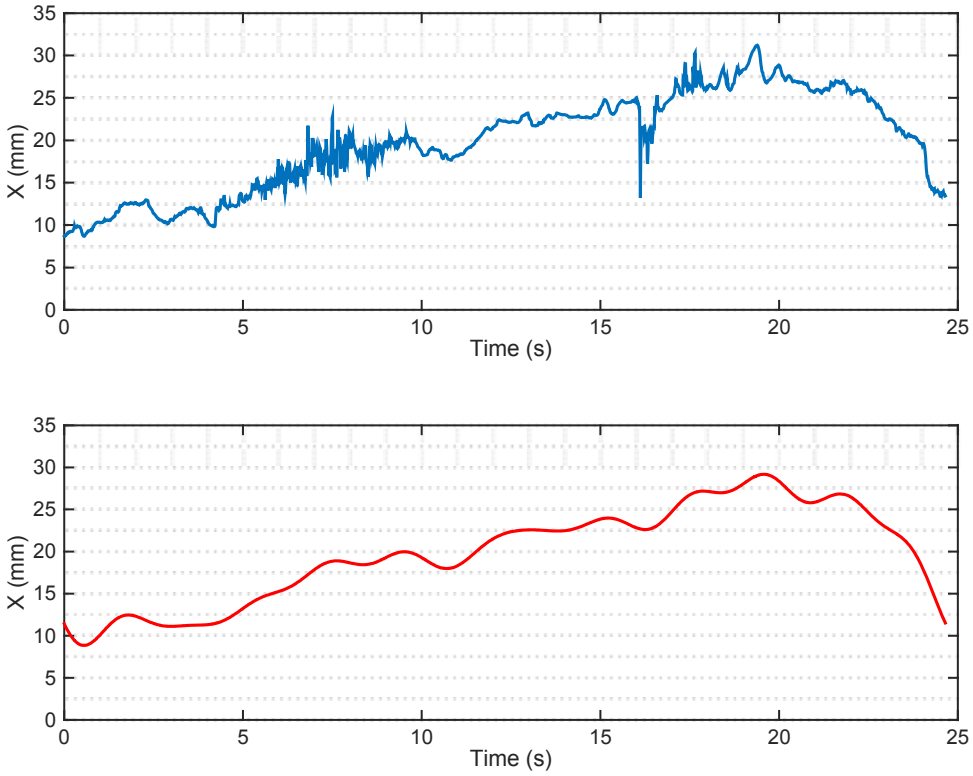


Figure 23. X Direction Trajectory in the Time Domain after a 0.5 Hz Bandpass Filter Is Applied.

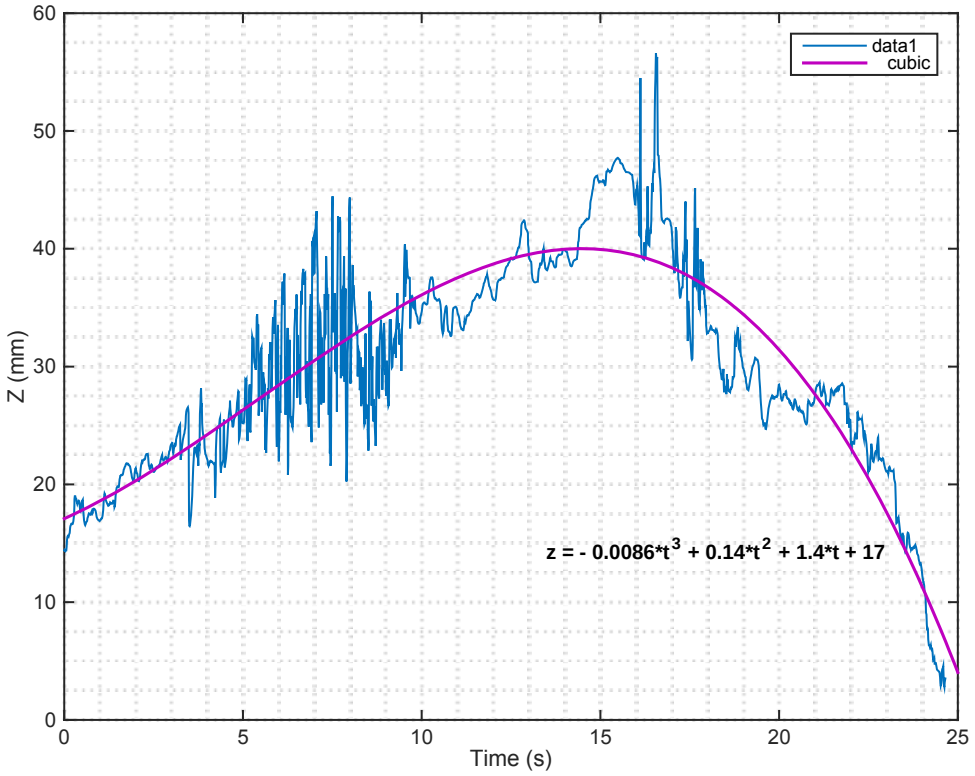


Figure 24. Position in the Z Direction versus Time.

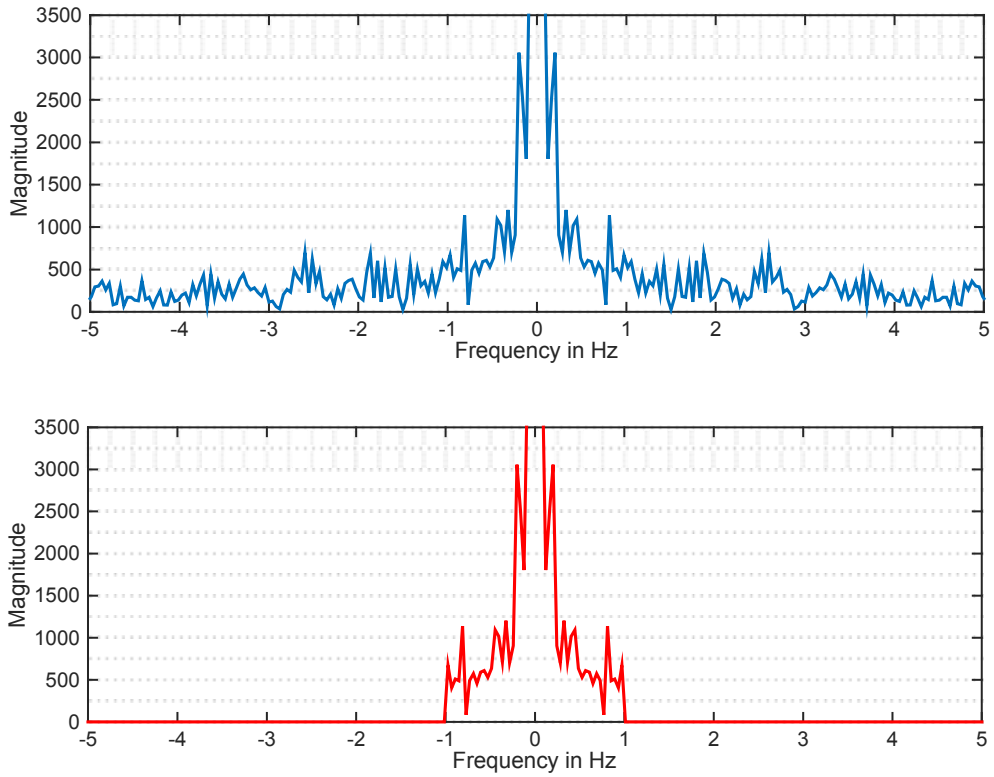


Figure 25. Z Direction Spectral Plot after a Bandpass Filter for 1.0 Hz is Applied.

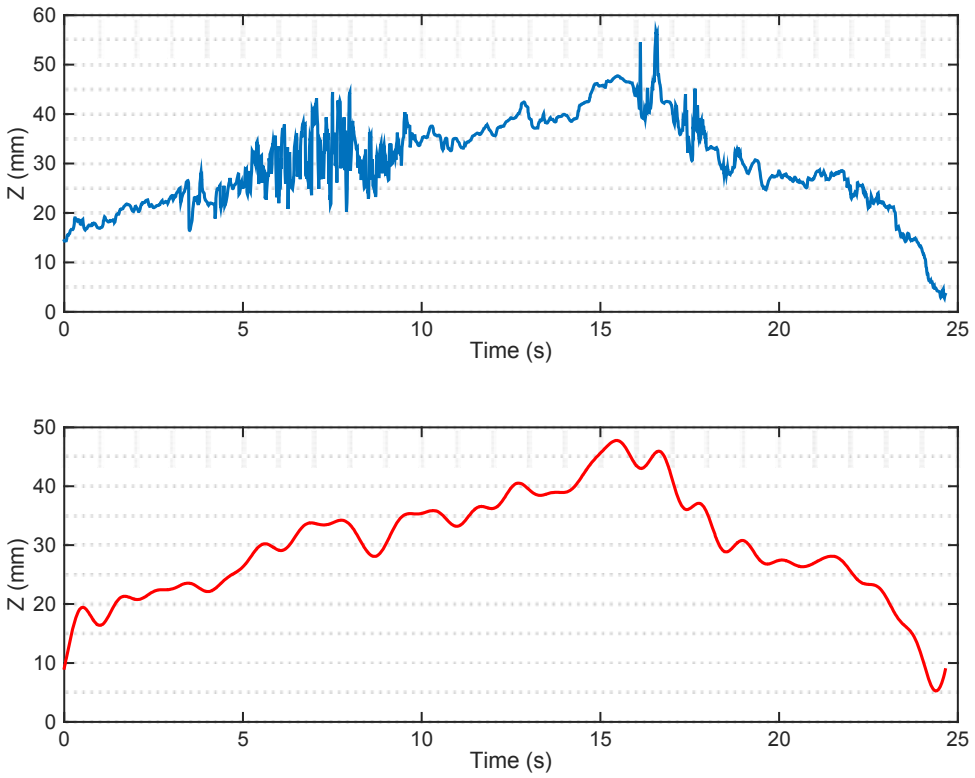


Figure 26. Z Direction Trajectory in the Time Domain after a 0.5 Hz Bandpass Filter Is Applied.

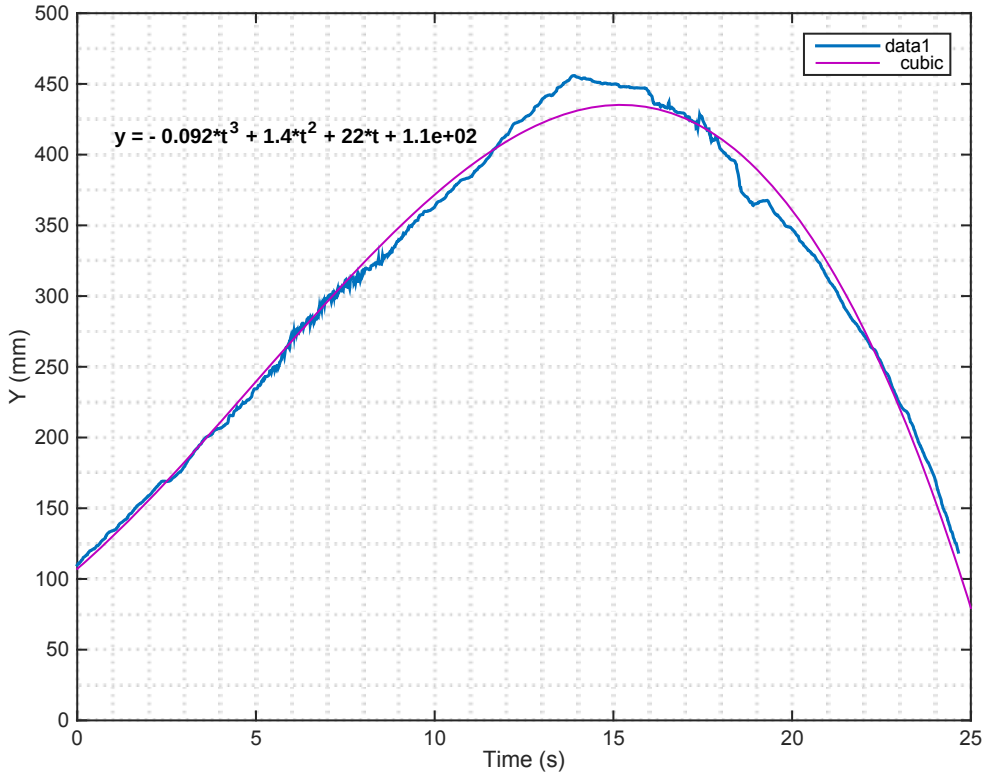


Figure 27. Position in the Y Direction versus Time.

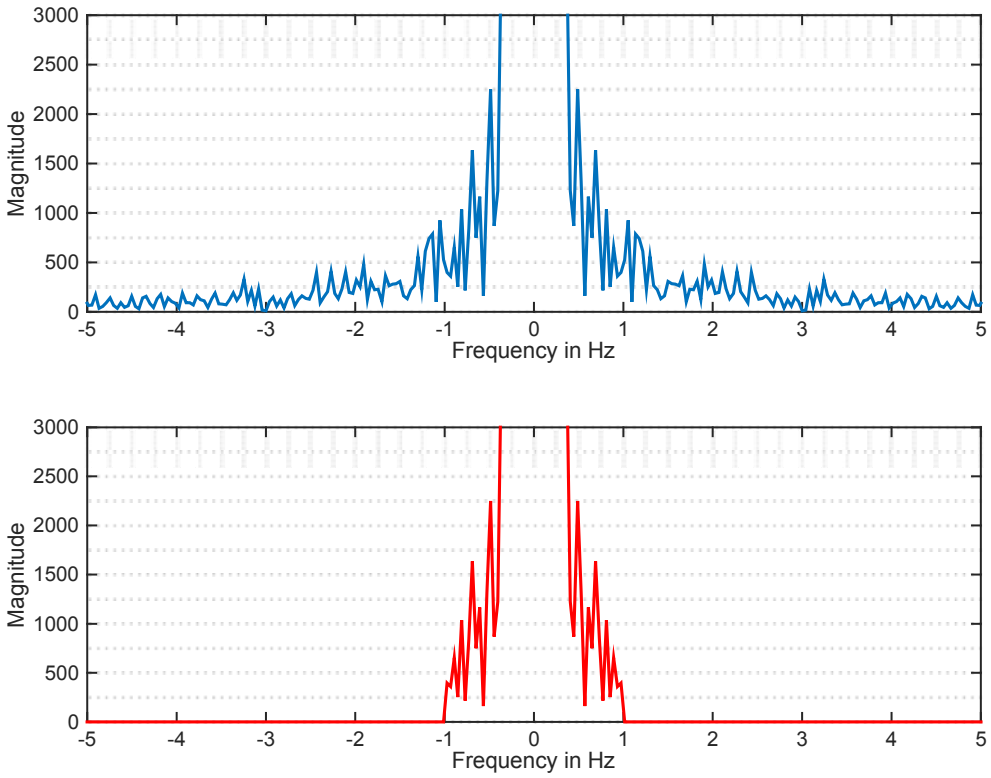


Figure 28. Y Direction Spectral Plot after a Bandpass Filter for 1.0 Hz Is Applied.

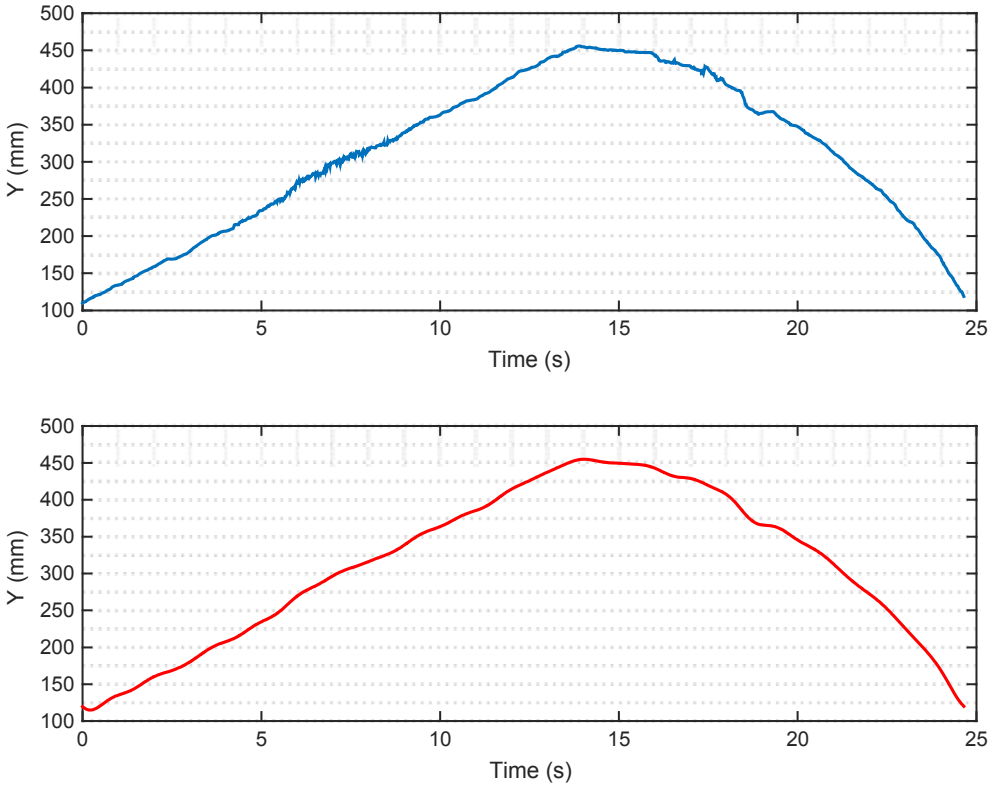


Figure 29. Y Direction Trajectory in the Time Domain after a 1.0 Hz Bandpass Filter Is Applied.

Analysis of these initial experiments gave us confidence that the trajectory data provided by the LMC was sufficient for 3D trajectory input to the Jaco Arm controller. Although the data was noisy (hand jitter), techniques could be employed via curve fitting and FFT methods to suppress unwanted motion.

2. Circular Motion Test

A more complicated motion was analyzed next. The operator moved his hand in a circular pattern and each new position was tracked and written in a text file with a 5 second delay. The data was plotted using MATLAB for different views to characterize the accuracy of the device. In general, the device tracks the circular pattern motion successfully at a vertical distance of 0.2 m, as shown in Figure 30 and Figure 31. There is an offset of 0.054 m in the vertical direction

between the initial and end point and a few unwanted positions caused by the hand tremor.

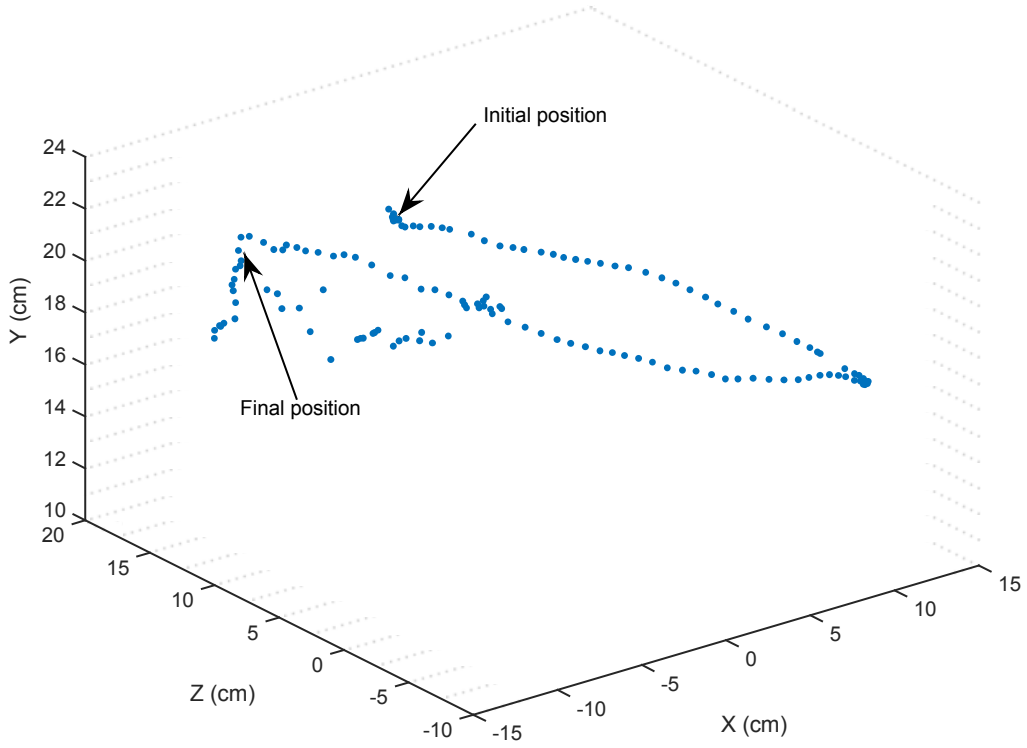


Figure 30. 3D Plot of the Leap Motion Controller Tracking Points During Circular Motion of a Hand Operator.

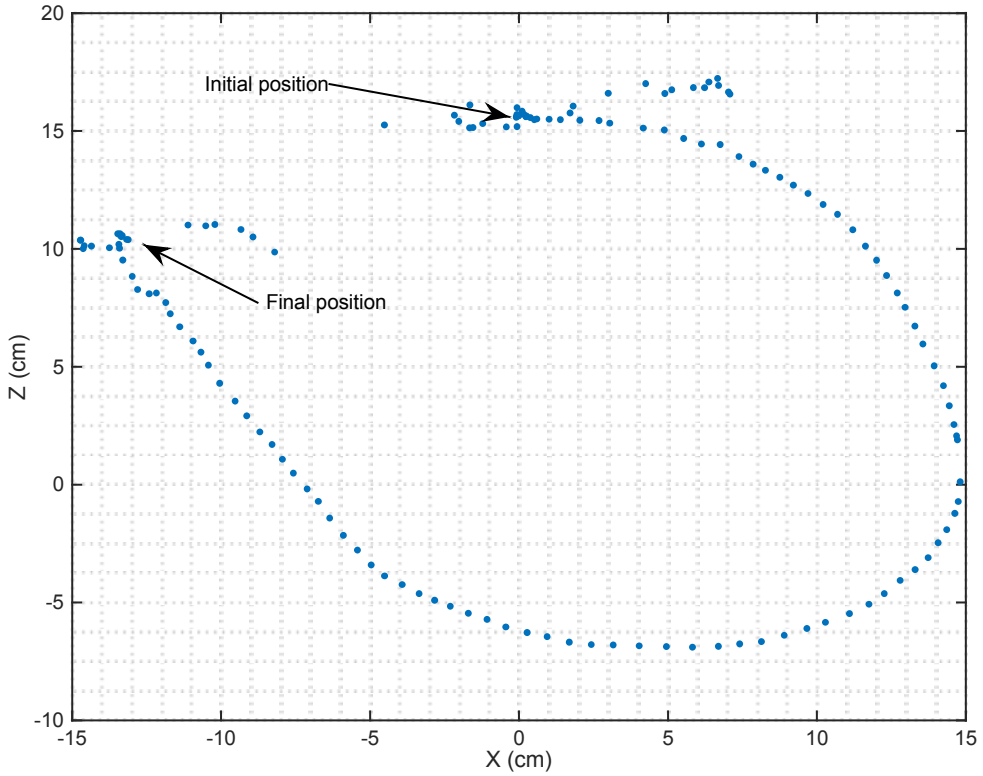


Figure 31. Top View of the Leap Motion Controller Tracking Points During Circular Motion of a Hand Operator.

3. Helicoid Motion Test

During this test, the operator moved his hand following a helicoid pattern using the LMC. As before, each new position is tracked and written to a text file. The data were then plotted using MATLAB for different views. The LCM tracked the helicoid pattern motion successfully from a vertical distance of 0.1 to 0.347 m as shown in Figure 32, Figure 33 and Figure 34.

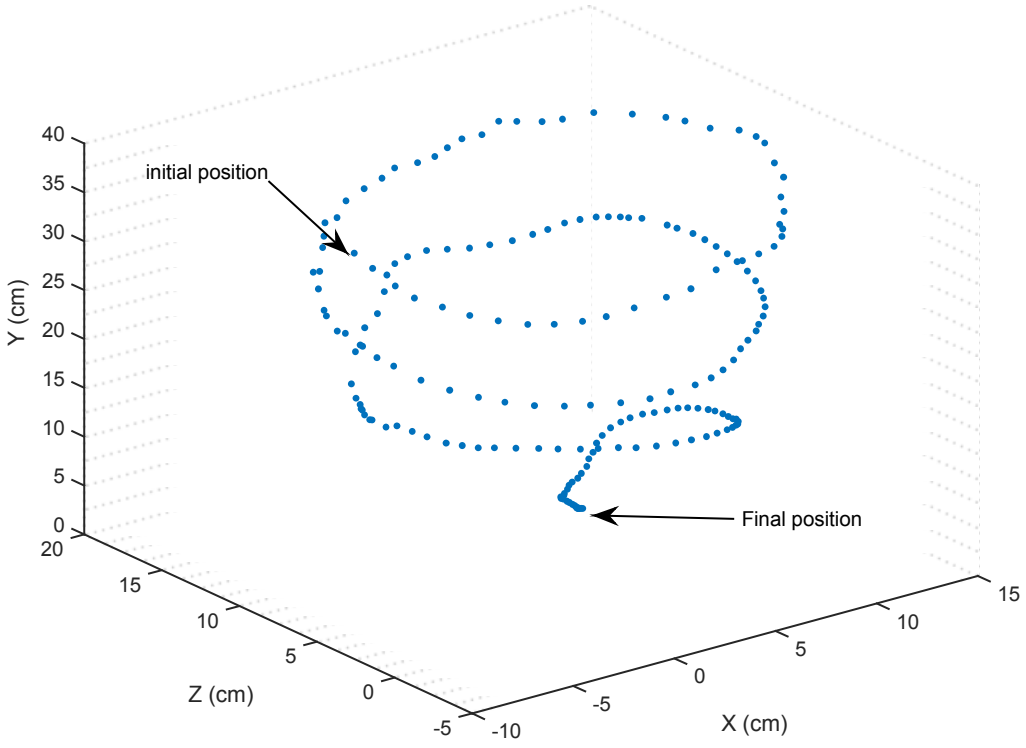


Figure 32. 3D Plot of the Leap Motion Controller Tracking Points During Helicoid Motion of a Hand Operator.

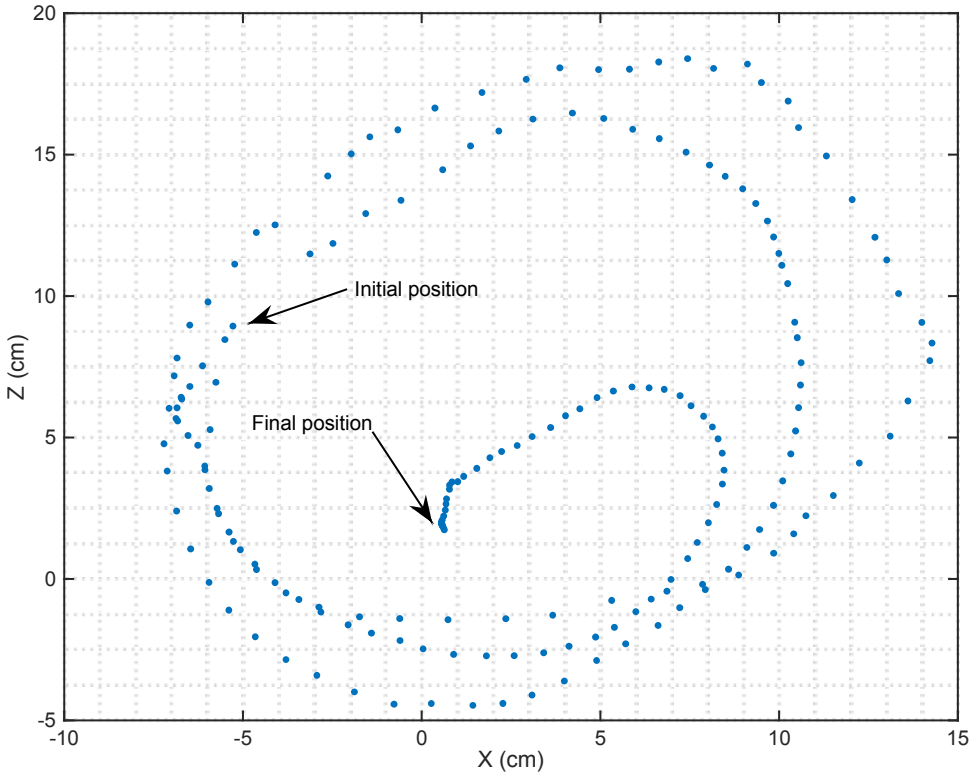


Figure 33. Top View of the Leap Motion Controller Tracking Points During Helicoid Motion of a Hand Operator.

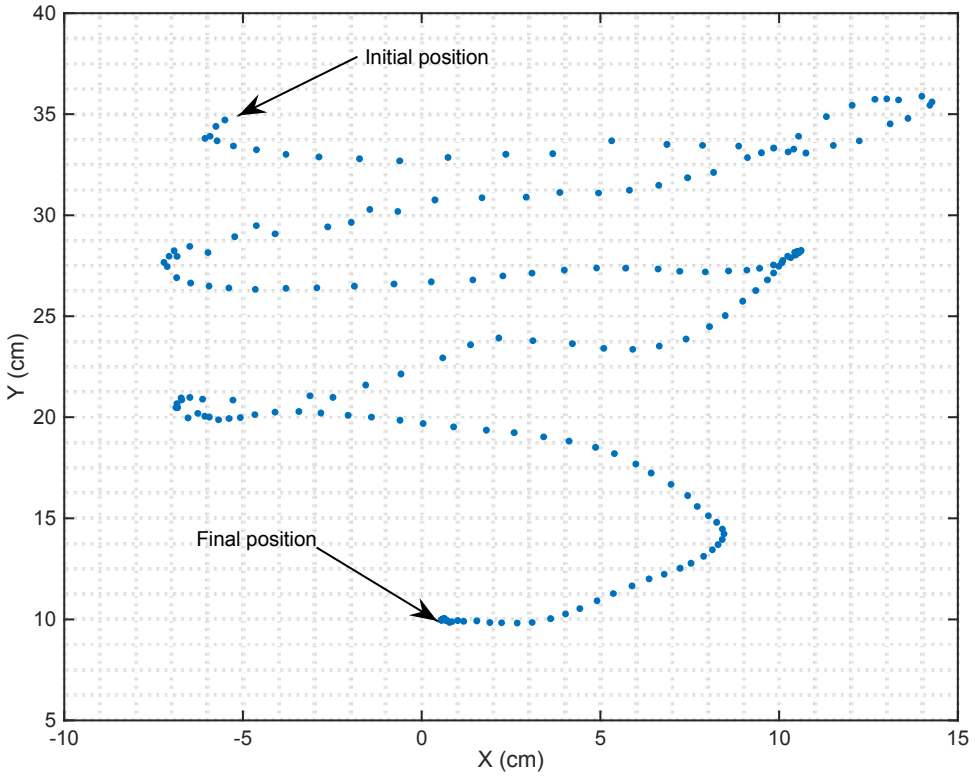


Figure 34. Front View of the Leap Motion Controller Tracking points During Helicoid Motion of a Hand Operator.

B. MOVING THE JACO ARM BY READING A TEXT FILE

Our next objective was to move the Jaco Arm with previously recorded data. The experiment was designed to read a text file (see Appendix B), which contained predetermined positions in meters, and move the Jaco Arm accordingly. As shown in Figure 35, the Jaco Arm successfully followed the trajectory. Figures 36 and 37 show the scaled results for that motion. The motion was scaled 3.5 times in the x direction, 0,1 in the y direction and twice in z direction. The scale factors were obtained at empirically to match the motion of the hand in the presence of the LMC to the actual operating volume of the end effector in the Jaco Arm reference frame.

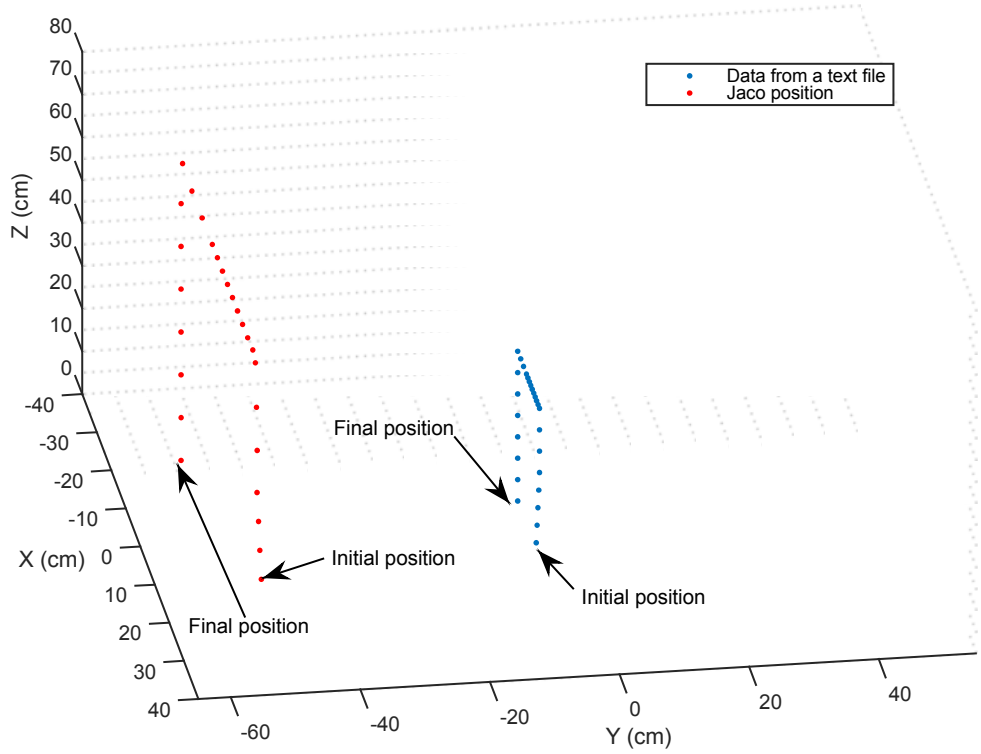


Figure 35. 3D plot of the Jaco Arm Movement by Reading New Positions Contained in a Text File.

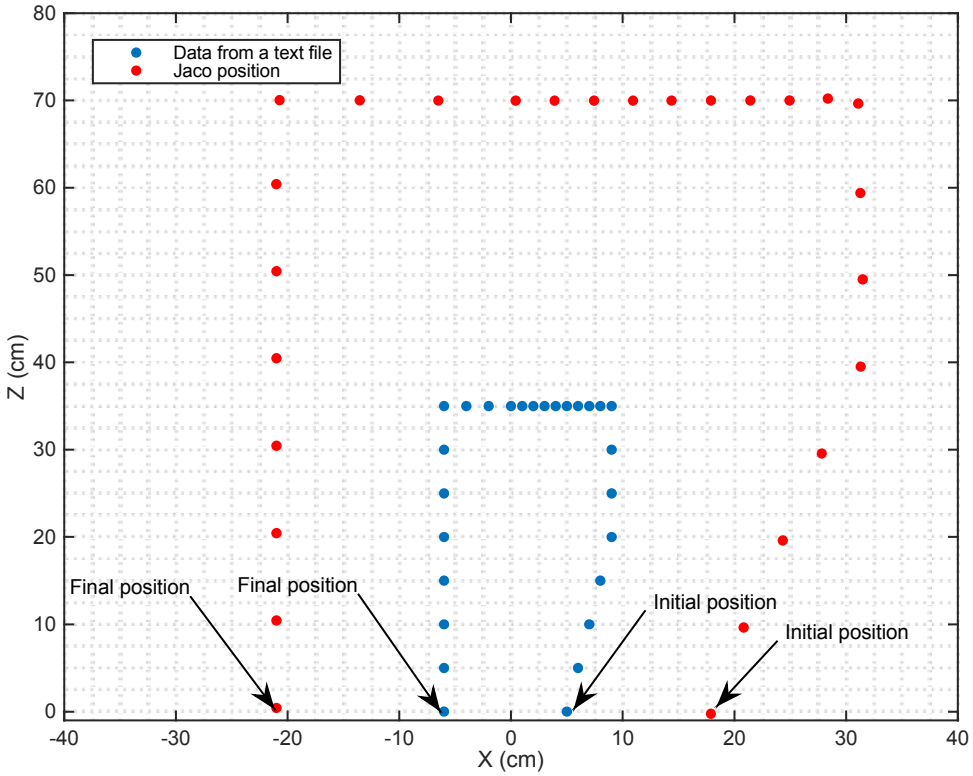


Figure 36. Front View of the Jaco Arm Movement by Reading New Positions Contained in a Text File.

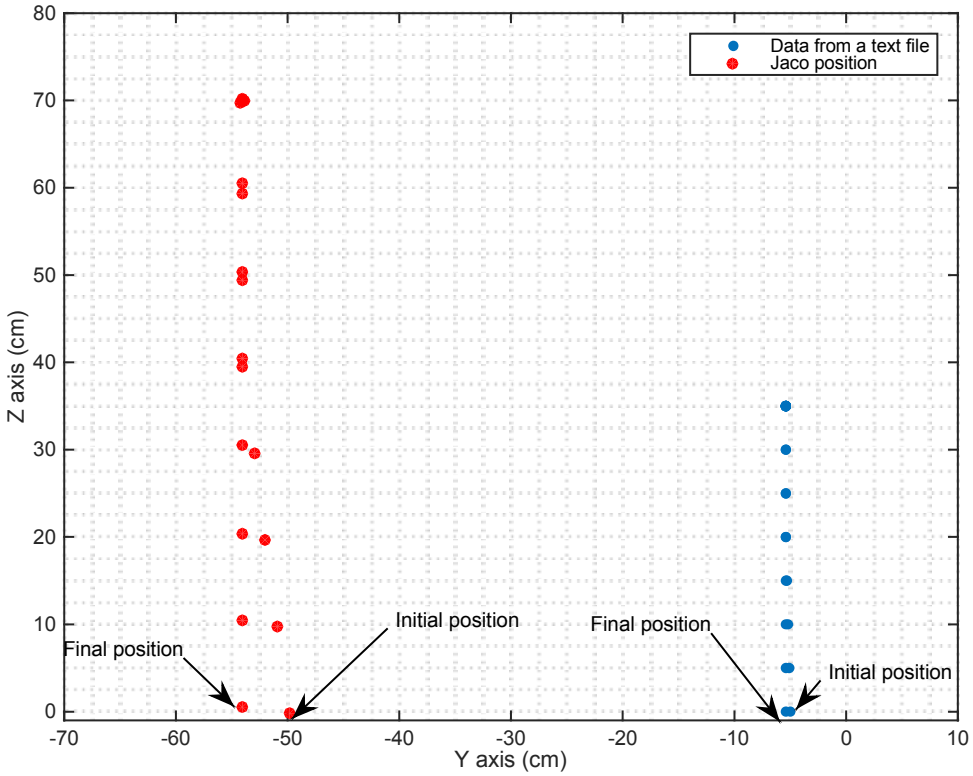


Figure 37. Rear View of the Jaco Arm Movement by Reading New Positions Contained in a Text File.

C. REAL TIME INTERFACE BETWEEN THE JACO ARM AND THE LEAP MOTION CONTROLLER

1. Motion in the Vertical Direction

Our final experiment was designed to remove the middleman read/write operation and move the arm in real time as the operator moved his hand in the LMC field of view. During this test, we kept x and y movement in the Jaco Arm fixed. Only new positions in the vertical direction are sent to the Jaco Arm as shown in Figure 38. The results indicate that there are shifts of 3 cm along the x direction, as shown in Figure 39, and just a few millimeters in the y direction, as shown in Figure 40.

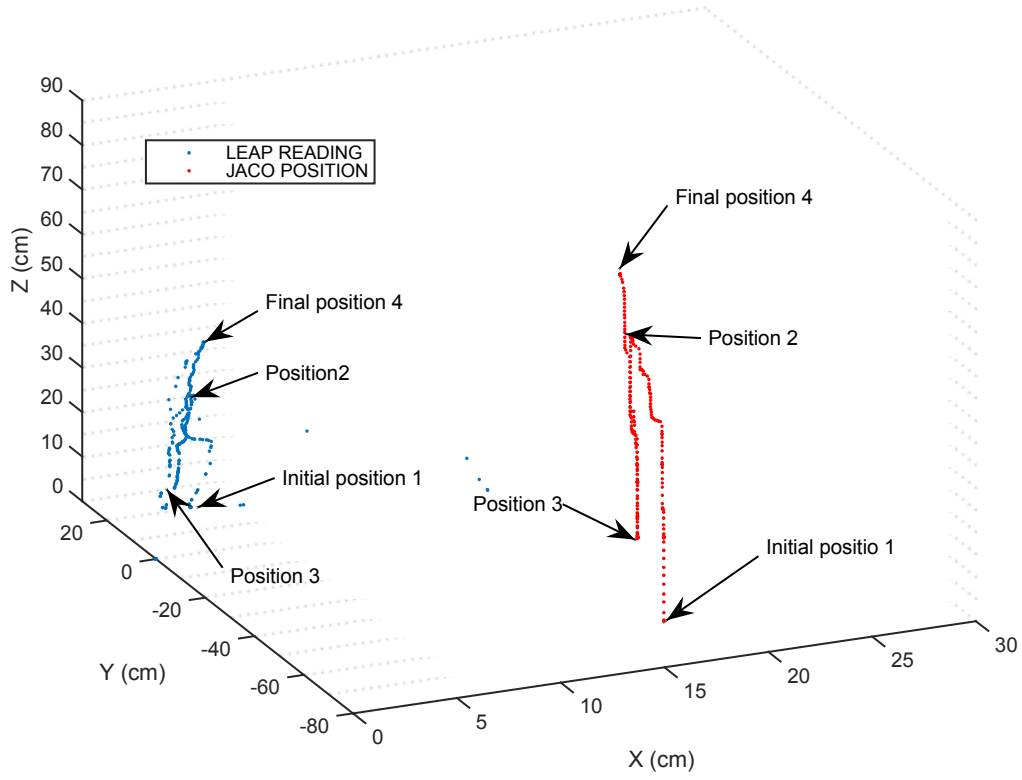


Figure 38. 3D Plot of the Jaco Arm Trajectory in the Vertical Direction Keeping X and Y Motion Fixed.

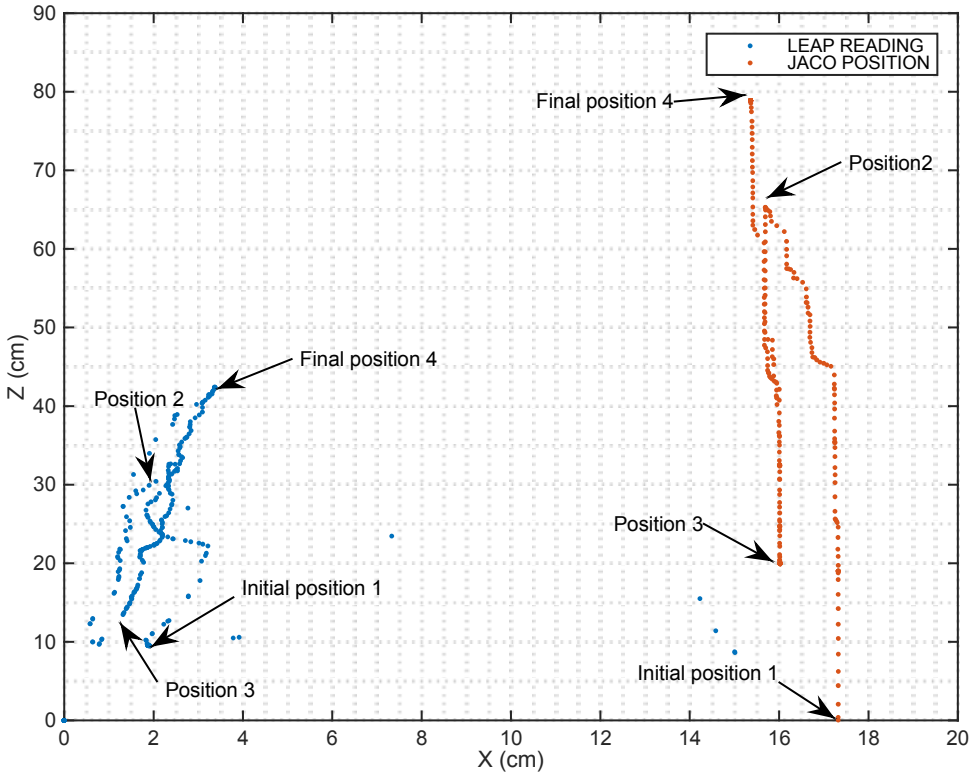


Figure 39. Front View Plot of the Jaco Arm Trajectory in the Vertical Direction Keeping X and Y Motion Fixed.

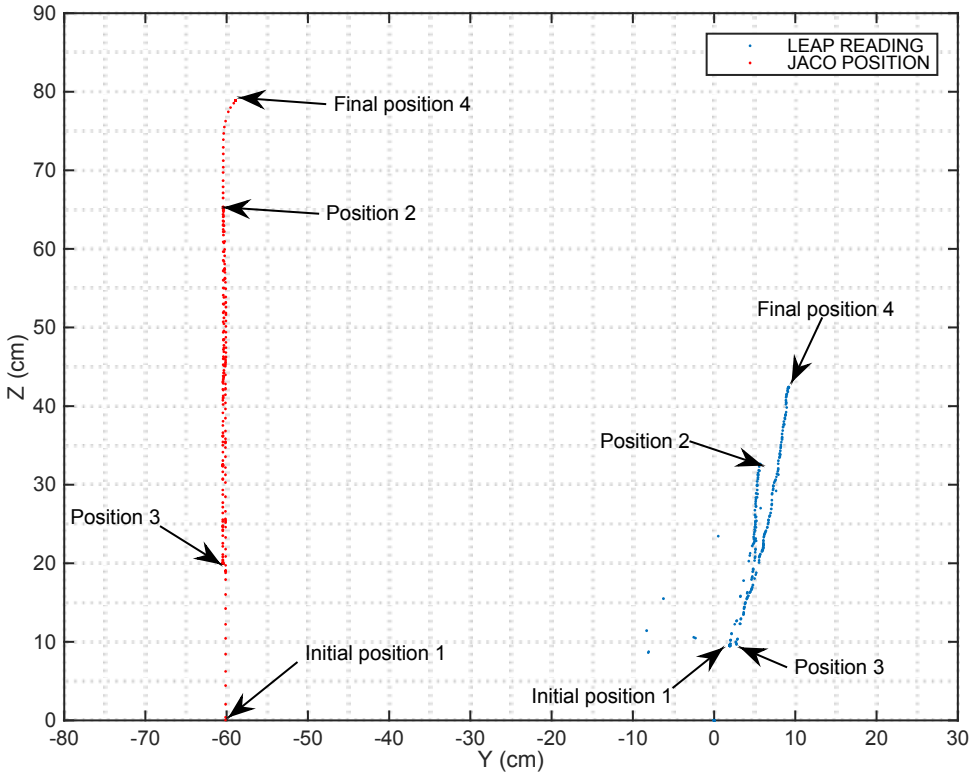


Figure 40. Rear View of the Jaco Arm Trajectory in the Vertical Direction Keeping X and Y Motion Fixed.

2. Real Time Connection in the X Direction

In this case, only positions in the x direction will be sent to the Jaco Arm. We constrained the end effector motion in the y and z axes as shown in Figure 41, using the function MyGetCartesianCommand. The results indicate that there are shifts of 1 cm along the z direction, as shown in Figure 42 and just a few millimeters in the y direction, as shown in Figure 43. In addition, we obtained limitation values in the x-axis for the end effector at different vertical distances. These values are shown in Table 3.

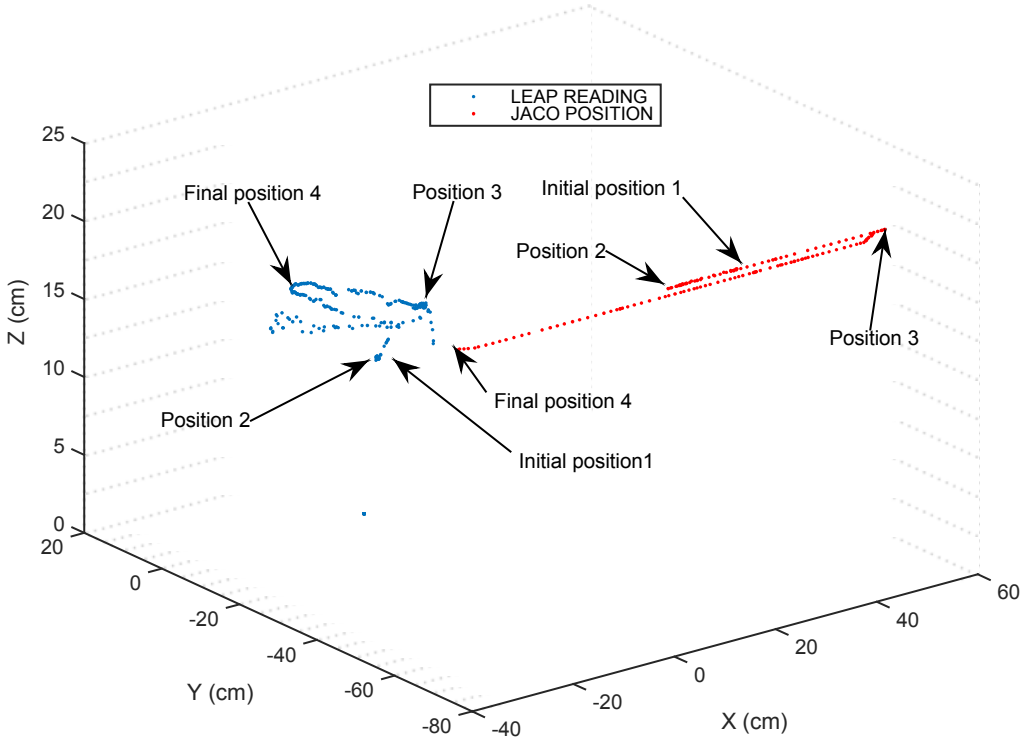


Figure 41. 3D Plot of the Jaco Arm Trajectory in the X Direction Keeping Y and Z motion Fixed.

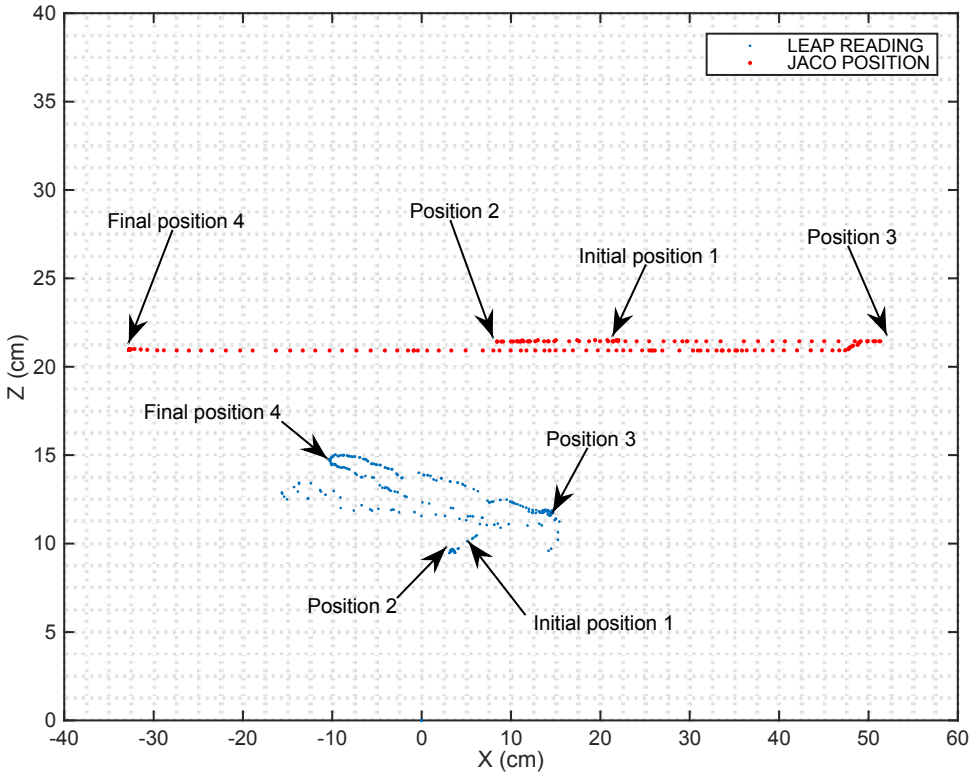


Figure 42. Front Plot of the Jaco Arm Trajectory in the X Direction Keeping Y and Z Motion Fixed.

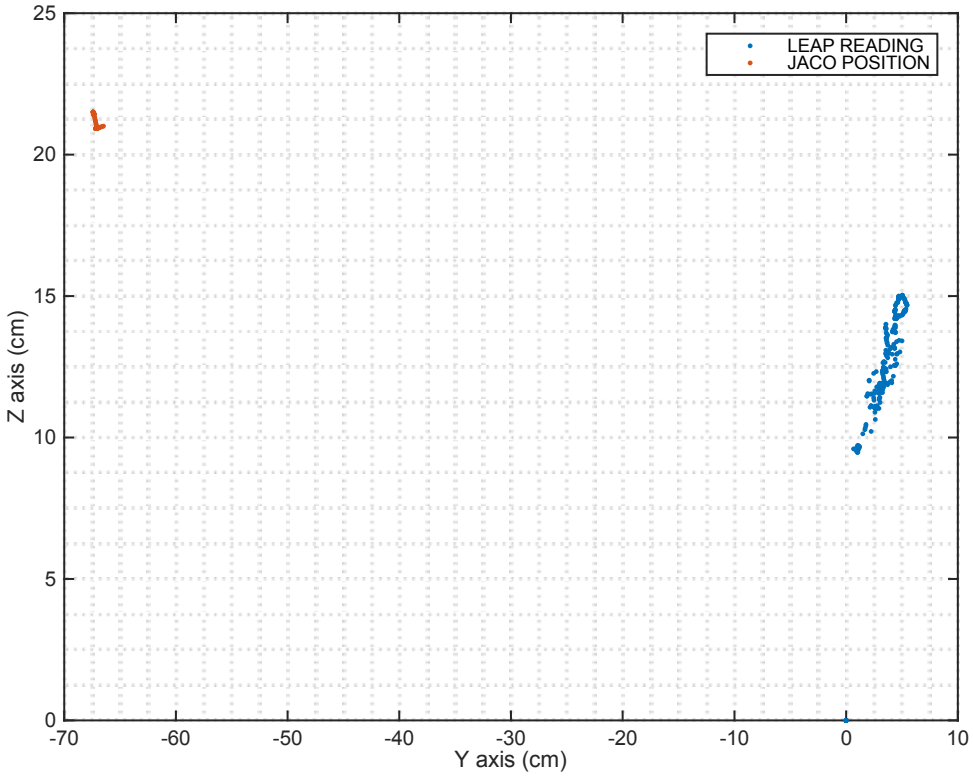


Figure 43. Rear Plot of the Jaco Arm Trajectory in the X Direction Keeping Y and Z Motion Fixed.

Table 3. Maximum and Minimum Values of the End Effector in the X Direction at Different Vertical Distances.

X		Y	Z
-0.42	0.42	-0.540	-0.158
-0.55	0.60	-0.527	0
-0.58	0.64	-0.562	0.2084
-0.59	0.59	-0.482	0.552
-0.40	0.50	-0.461	0.762

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The 6 DOF LMC controller was successfully interfaced to the Jaco Arm as a method to mimic natural hand motion that is considered intuitive and adaptive. Trajectory data provided by the LMC was sufficient for 3D Cartesian input to the Jaco Arm controller. Noisy data (hand jitter) was handled in post-processing with low pass filter techniques. This included FFT and curve fitting methods.

Two-dimensional motion proved to be quite accurate, while full 3D motion posed problems with trajectory lag under fast motion. This was attributed to read/ write frame rates that were not optimized.

The LMC and Kinova SDK user interfaces were successfully exploited, via a C++ program algorithm, to realize Jaco Arm motion in the lab environment.

B. RECOMMENDATIONS

Further work is recommended with regard to algorithm optimization for full 3D motion, particularly in relation to the read/write methods employed in the near real-time operational environment. Normal user visual feedback, under normal operation conditions, is sufficient to compensate for trajectory latencies, but it would be better to fix the program code to manage this without an ad hoc fix.

More data sets for motion in regard to roll, pitch and yaw will need to be completed. These data need to be analyzed and characterized to safely claim the LMC sensor is integrated with the Jaco Arm in a full, real-time operational environment.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. CODE TO OBTAIN HAND POSITION IN CARTESIAN COORDINATES USING THE LEAP MOTION CONTROLLER

Using the code presented below, we got X, Y and Z position of the center of the hand's operator using the Leap Motion Controller and wrote them in a text file.

```
#include <iostream>
#include <cstring>
#include <fstream> // create and wittre info to a file
#include "Leap.h"
#include <Windows.h>

using namespace Leap;
using namespace std;
int CFLAG = 0;

class SampleListener : public Listener {
  public:
    virtual void onConnect(const Controller&);
    virtual void onFrame(const Controller&);
  private:
};

void SampleListener::onConnect(const Controller& controller) {
}

void SampleListener::onFrame(const Controller& controller) {
       ofstream outfile;
       outfile.open("example.txt," ios::out | ios::app);
       // Get the most recent frame and report information about the frame
       const Frame frame = controller.frame();
       // Get info about right hand
       Leap::Hand hand = frame.hands().rightmost();
       // handcenter gets position vector (x,y,z) of hand.
       // Vector class gets the components as separate parameters
       Leap::Vector handcenter = hand.palmPosition();
       float x = handcenter[0];
       float y = handcenter[1];
       float z = handcenter[2];
       //print in the screen the palm psoition x component in mm
       std::cout << x << " ";
       //print in the screen the palm position z component in mm
       std::cout << z << " ";
       //print in the screen palm position y component in mm. It is the veritical
       component
       std::cout << y << std::endl;
```

```
        //print x, y and z in “mm” in the *.txt y is the vertical coordinate
        outfile << x << “ “ << z << “ “ << y << std::endl;

        //wait 0.5sec to get other reading
        Sleep(50);
        if (CFLAG){
                outfile.close();
        }
}


int main(int argc, char** argv) {

  // Create a sample listener and controller
        SampleListener listener;
        Controller controller; // it just connected
  // Have the sample listener receive events from the controller
  controller.addListener(listener);
  // Keep this process running until Enter is pressed
  std::cout << “Press Enter to quit...” << std::endl;
  std::cin.get();
  CFLAG = 1;
  // Remove the sample listener when done
  controller.removeListener(listener);

  return 0;
}
```

APPENDIX B. CODE FOR MOVING THE END EFFECTOR BY READING A TEXT FILE

This code allows the user to read a text file containing X, Y and Z position of the center of the hand's operator using the Leap Motion Controller and move the end effector to the new positions.

```
#include "Lib_Examples\CommunicationLayerWindows.h"
#include "Lib_Examples\CommandLayer.h"
#include <conio.h>
#include "Lib_Examples\KinovaTypes.h"
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <Windows.h>
#include <fstream>
#include <sstream>

using namespace std;

//A handle to the API.
HINSTANCE commandLayer_handle;

//Function pointers to the functions we need
int(*MyInitAPI)();
int(*MyCloseAPI)();
int(*MySendBasicTrajectory)(TrajectoryPoint command);
int(*MyGetDevices)(KinovaDevice devices[MAX_KINOVA_DEVICE], int &result);
int(*MySetActiveDevice)(KinovaDevice device);
int(*MyMoveHome)();
int(*MyInitFingers)();
int(*MyGetCartesianCommand)(CartesianPosition &);

int main(int argc, char* argv[])
{
	//We load the API.
	commandLayer_handle = LoadLibrary(L"CommandLayerWindows.dll");
	CartesianPosition currentCommand;

	int programResult = 0;

	//We load the functions from the library (Under Windows, use
	GetProcAddress)
	MyInitAPI = (int(*)()) GetProcAddress(commandLayer_handle, "InitAPI");
	MyCloseAPI = (int(*)()) GetProcAddress(commandLayer_handle, "CloseAPI");
	MyMoveHome = (int(*)()) GetProcAddress(commandLayer_handle, "MoveHome");
	MyInitFingers = (int(*)()) GetProcAddress(commandLayer_handle,
	"InitFingers");
	MyGetDevices = (int(*)(KinovaDevice devices[MAX_KINOVA_DEVICE], int
	&result)) GetProcAddress(commandLayer_handle, "GetDevices");
```

```
MySetActiveDevice          =          (int(*)(KinovaDevice          devices))
GetProcAddress(commandLayer_handle, "SetActiveDevice");
MySendBasicTrajectory              =              (int(*)(TrajectoryPoint))
GetProcAddress(commandLayer_handle, "SendBasicTrajectory");
MyGetCartesianCommand         =         (int(*)(CartesianPosition         &))
GetProcAddress(commandLayer_handle, "GetCartesianCommand");

//Verify that all functions have been loaded correctly
if ((MyInitAPI == NULL) || (MyCloseAPI == NULL) || (MySendBasicTrajectory
== NULL) ||
       (MyGetDevices   ==   NULL)   ||   (MySetActiveDevice   ==   NULL)   ||
       (MyGetCartesianCommand == NULL) ||
       (MyMoveHome == NULL) || (MyInitFingers == NULL))

{
       cout << "* * *  E R R O R  D U R I N G  I N I T I A L I Z A T I O
       N  * * *" << endl;
       programResult = 0;
}
else
{
       cout << "I N I T I A L I Z A T I O N  C O M P L E T E D" << endl <<
       endl;

       int result = (*MyInitAPI)();

       KinovaDevice list[MAX_KINOVA_DEVICE];

       int devicesCount = MyGetDevices(list, result);
       int i = 0;
       int J = 1;
       int start;
       float Xleap;
       float Yleap;
       float Zleap;
       float Xjaco;
       float Yjaco;
       float Zjaco;

       // defining scale factors for z axis (m). The arm will move up from
       0 to the value Jzmax in the z axis.
       float Lzmax = 0.4f;
       float Jzmax = 0.8f;

       // defining scale factors for x axis (m). the arm will move from -
       Jxmax to Jxmax.
       float Lxmax = 0.1f;
       float Jxmax = 0.35f;

       // defining scale factors for y axis (m). the arm will move from -
       Jymax to Jymax.
       float Lymax = 0.07f;
       float Jymax = 0.7f;

       // readed values from the file
       float x, y, z;
```

```
// reading the file line by file
std::string line;
ifstream myfile;
myfile.open("example.txt");

// writing file for the position of the hand
ofstream outfile;
outfile.open("example_output.txt," ios::out | ios::app);

while (J == 1)
{

// check if the file is open
if (myfile.is_open())
{
	while (getline(myfile, line))	//read stream line by line,
	{
		std::istringstream in(line);	//make a stream for
		the line
		in >> x >> y >> z;

			//Setting the current device as the active
			device.
			MySetActiveDevice(list[i]);

			TrajectoryPoint pointToSend;
			pointToSend.InitStruct();

			//We specify that this point will be an
			angular(joint by joint) position.
			pointToSend.Position.Type = CARTESIAN_POSITION;

			//Converting the the data of Leap to meters.
			Xleap = x;
			Yleap = y;
			Zleap = z;

			//We get the actual angular command of the
			robot.
			MyGetCartesianCommand(currentCommand);

			// check reading position of the file
			std::cout << "READING THE FILE..." << endl;
			std::cout << Xleap << " " << Yleap << " " <<
			Zleap<< '\n';

			// Mapping the coordinates to the Jaco app

			Xjaco = (Jxmax/Lxmax)*Xleap;
			std::cout << "Xjaco is: " << endl;
			std::cout << Xjaco << endl;

			Yjaco = (Jymax/Lymax)*Yleap;
			std::cout << "Yjaco is: " << endl;
			std::cout << Yjaco << endl;

			Zjaco = (Jzmax/Lzmax)*Zleap;
```

```
                                    std::cout << "Zjaco is: " << endl;
                                    std::cout << Zjaco << endl;

                                    pointToSend.Position.CartesianPosition.X       =
                                    Xjaco;
                                    pointToSend.Position.CartesianPosition.Y       =
                                    Yjaco;
                                    pointToSend.Position.CartesianPosition.Z       =
                                    Zjaco;
                                    pointToSend.Position.CartesianPosition.ThetaX  =
                                    currentCommand.Coordinates.ThetaX;
                                    pointToSend.Position.CartesianPosition.ThetaY  =
                                    currentCommand.Coordinates.ThetaY;
                                    pointToSend.Position.CartesianPosition.ThetaZ  =
                                    currentCommand.Coordinates.ThetaZ;
                                    MySendBasicTrajectory(pointToSend);

                                    Sleep(100);
                                    //We get the actual angular command of the
                                    robot.
                                    MyGetCartesianCommand(currentCommand);

                                    // check current position of the hand
                                    std::cout << "THE NEW POSITION OF THE HAND IS: "
                                    << endl;
                                    std::cout << currentCommand.Coordinates.X << " "
                                    <<  currentCommand.Coordinates.Y  <<  "  "  <<
                                    currentCommand.Coordinates.Z << endl;
                                    std::cout                                     <<
                                    "*********************************************"
                                    << endl;
                                    //print x, y and z in "mm" in the *.txt y is the
                                    vertical coordinate
                                    outfile << currentCommand.Coordinates.X << " "
                                    <<  currentCommand.Coordinates.Y  <<  "  "  <<
                                    currentCommand.Coordinates.Z << std::endl;
                            }
                                    cout << "press 1 to continue: ";
                                    cin >> J;
                                    start = J;
                    }
                    myfile.close();
                    outfile.close();
            }



            cout << endl << "C L O S I N G   A P I" << endl;
            result = (*MyCloseAPI)();
            programResult = 1;
    }

    FreeLibrary(commandLayer_handle);

    return programResult;

}
```

APPENDIX C. CODE TO ALLOW INTERFACE BETWEEN JACO ARM AND LEAP MOTION CONTROLLER IN REAL TIME

The following code, based on the Cartesian position example provided by [5], allows to interface the Jaco Arm and the Leap Motion Controller in real time. It was written in C++ using the IDE Visual Studio 2013 and includes libraries from both Kinova SDK and Leap SDK. I used Cartesian control method to command the arm. The Jaco Arm has a maximum reach of about 0.9m but I limited to 0.3m in the x axis and 0.5 m in the z axis.

```
#include “Lib_Examples\CommunicationLayerWindows.h”
#include “Lib_Examples\CommandLayer.h”
#include <conio.h>
#include “Lib_Examples\KinovaTypes.h”
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <Windows.h>
#include <sstream>
#include <cstring>
#include “Leap.h”

float x, y, z;
float a, b, c;
int extended_fingers;

//Leap Function
using namespace Leap;
using namespace std;

class SampleListener : public Listener {
public:
	virtual void onConnect(const Leap::Controller&);
	virtual void onFrame(const Leap::Controller&);

private:
};

void SampleListener::onConnect(const Leap::Controller& controller) {
}



void SampleListener::onFrame(const Leap::Controller& controller) {
	// Get the most recent frame and report information about the frame
	const Frame frame = controller.frame();
```

```
        // Check if the fingers are extended
        extended_fingers = frame.fingers().extended().count();
        std::cout<< extended_fingers <<std::endl;

        // Get info about right hand
        Leap::Hand hand = frame.hands().rightmost();
        // handcenter gets position vector (x,y,z) of hand.
        // Vector class gets the components as separate parameters and converts in
Meters

        Leap::Vector handcenter = hand.palmPosition();
        x = handcenter[0]*0.001f;
        y = handcenter[1]*0.001f;

        z = handcenter[2]*0.001f;

        //print in the screen the palm position x component in mm
        std::cout << x << " ";
        //print in the screen the palm position z component in mm
        std::cout << z << " ";
        //print in the screen palm position y component in mm. It is the vertical
        component
        std::cout << y << std::endl;
}

int myleap(int argc, char** argv) {
        // Create a sample listener and controller
        SampleListener listener;
        Leap::Controller controller; // it just connectes
        // Have the sample listener receive events from the controller
        controller.addListener(listener);
        Sleep(30);
        // Remove the sample listener after read one frame
        controller.removeListener(listener);
        return 0;
}

//Jaco function
using namespace std;

//A handle to the API.
HINSTANCE commandLayer_handle;

//Function pointers to the functions we need
int(*MyInitAPI)();
int(*MyCloseAPI)();
int(*MySendBasicTrajectory)(TrajectoryPoint command);
int(*MyGetDevices)(KinovaDevice devices[MAX_KINOVA_DEVICE], int &result);
int(*MySetActiveDevice)(KinovaDevice device);
int(*MyMoveHome)();
int(*MyInitFingers)();
int(*MyGetCartesianCommand)(CartesianPosition &);


int main(int argc, char* argv[])
{
        //We load the API.
```

```
commandLayer_handle = LoadLibrary(L"CommandLayerWindows.dll");
CartesianPosition currentCommand;

int programResult = 0;


//We load the functions from the library (Under Windows, use
GetProcAddress)

MyInitAPI = (int(*)()) GetProcAddress(commandLayer_handle, "InitAPI");
MyCloseAPI = (int(*)()) GetProcAddress(commandLayer_handle, "CloseAPI");
MyMoveHome = (int(*)()) GetProcAddress(commandLayer_handle, "MoveHome");

MyInitFingers = (int(*)()) GetProcAddress(commandLayer_handle,
"InitFingers");
MyGetDevices = (int(*)(KinovaDevice devices[MAX_KINOVA_DEVICE], int
&result)) GetProcAddress(commandLayer_handle, "GetDevices");
MySetActiveDevice = (int(*)(KinovaDevice devices))
GetProcAddress(commandLayer_handle, "SetActiveDevice");
MySendBasicTrajectory = (int(*)(TrajectoryPoint))
GetProcAddress(commandLayer_handle, "SendBasicTrajectory");
MyGetCartesianCommand = (int(*)(CartesianPosition &))
GetProcAddress(commandLayer_handle, "GetCartesianCommand");

//Verify that all functions has been loaded correctly
if ((MyInitAPI == NULL) || (MyCloseAPI == NULL) || (MySendBasicTrajectory
== NULL) ||(MyGetDevices == NULL) || (MySetActiveDevice == NULL) ||
(MyGetCartesianCommand == NULL) ||(MyMoveHome == NULL) || (MyInitFingers ==
NULL))
{
	cout << "* * *  E R R O R   D U R I N G   I N I T I A L I Z A T I O
	N  * * *" << endl;
	programResult = 0;
}
else
{

	cout << "I N I T I A L I Z A T I O N   C O M P L E T E D" << endl <<
	endl;

	int result = (*MyInitAPI)();
	KinovaDevice list[MAX_KINOVA_DEVICE];
	int devicesCount = MyGetDevices(list, result);
	int i = 0;
	int J = 1;

	float Xleap;
	float Yleap;
	float Zleap;
	float Xjaco;
	float Yjaco;
	float Zjaco;

	// defining scale factors for z axis (m). The arm will move up from
	0 to the value Jzmax in the z axis.

	float Lzmax = 0.23f;
```

```
float Jzmax = 0.5f;

// defining scale factors for x axis (m). the arm will move from -
Jxmax to Jxmax.

float Lxmax = 0.1f;
float Jxmax = 0.3f;

// defining scale factors for y axis (m). the arm will move from -
Jymax to Jymax.

float Lymax = 0.1f;

float Jymax = 0.43f;


	//Setting the current device as the active device.
	MySetActiveDevice(list[i]);
	MyInitFingers();
	TrajectoryPoint pointToSend;
	pointToSend.InitStruct();

	//We specify that this point will be an angular(joint by
	joint) position.

	pointToSend.Position.Type = CARTESIAN_POSITION;

	while (J == 1)

{

	//Call Leap function
	myleap(argc,argv);
	Xleap = x;
	Yleap = z;
	Zleap = y;
	if (extended_fingers == 0)
	{
		a=57.0f;
		b=57.0f;
		c=57.0f;
	}
	else {
		a=0.25f;
		b=0.25f;
		c=0.25f;
	};

	//We get the actual angular command (Position) of the robot.
	MyGetCartesianCommand(currentCommand);

	// Mapping the coordinates to the Jaco app

	Xjaco = (Jxmax/Lxmax)*Xleap;
	std::cout << "Xjaco is: " << endl;
	std::cout << Xjaco << endl;
```

```
                Yjaco = (Jymax/Lymax)*Yleap;
                std::cout << “Yjaco is: “ << endl;
                std::cout << Yjaco << endl;

                Zjaco = (Jzmax/Lzmax)*Zleap;
                std::cout << “Zjaco is: “ << endl;
                std::cout << Zjaco << endl;

                pointToSend.Position.CartesianPosition.X = Xjaco;
                pointToSend.Position.CartesianPosition.Y =
                currentCommand.Coordinates.Y;
                pointToSend.Position.CartesianPosition.Z = Zjaco;

                pointToSend.Position.CartesianPosition.ThetaX =
                currentCommand.Coordinates.ThetaX;
                pointToSend.Position.CartesianPosition.ThetaY =
                currentCommand.Coordinates.ThetaY;
                pointToSend.Position.CartesianPosition.ThetaZ =
                currentCommand.Coordinates.ThetaZ;

                pointToSend.Position.Fingers.Finger1 = a;
                pointToSend.Position.Fingers.Finger2 = b;
                pointToSend.Position.Fingers.Finger3 = c;

                MySendBasicTrajectory(pointToSend);

                //We get the actual angular command of the robot.
                MyGetCartesianCommand(currentCommand);

                // check current position of the hand
                std::cout << “THE NEW POSITION OF THE HAND IS: “ << endl;
                std::cout << currentCommand.Coordinates.X << “ “ <<
                currentCommand.Coordinates.Y << “ “ <<
                currentCommand.Coordinates.Z << endl;

                std::cout << “*******************************************”
                << endl;
        }

        cout << endl << “C L O S I N G   A P I” << endl;
        result = (*MyCloseAPI)();
        programResult = 1;
}
FreeLibrary(commandLayer_handle);
return programResult;
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] S. Niku, *Introduction to Robotics, Analysis, Control, Applications.* 2nd ed. Hoboken, NJ: John Wiley & Sons, 2011, pp. 19–95.

[2] Assistive Robotics. (n.d.). Kinova Robotics. [Online]. Available: http://www.kinovarobotics.com/assistive-robotics/?section=assistive. Accessed Oct. 15, 2016.

[3] D .Bassily, C. Georgoulas, J. Güttler, T. Linner, T. bock, "Intuitive and adaptive robotic arm manipulation using Leap Motion Controller," in *41st International Symposium in Robotics,* München, Germany, 2014, pp. 78–83.

[4] R. Harkins, private communication, Oct. 2015.

[5] A. Jacinto, "Unmanned systems: A lab-based robotic arm for grasping,"M.S. thesis, Dept. Physics, Naval Postgraduate School, Monterey, CA, 2015.

[6] R. Palacios, "Robotic arm manipulation laboratory with a six degree of freedom Jaco Arm," M.S. thesis, Dept. Physics, Naval Postgraduate School, Monterey, CA, 2015.

[7] R. D. Klafter, T. A. Chmielewski and M. Negin, *Robotic Engineering: An Integrated Approach*, 1st ed. Englewood Cliffs, NJ: Prentice Hall, 1989, pp. 561–633.

[8] *JACO Research Edition User Guide*, 1st ed. Kinova, Boisbriand, Canada, 2011, pp. 7–36.

[9] System Arquitecture. (n.d.). Leap Motion Developers. [Online]. Available: https://developer.leapmotion.com/documentation/cpp/devguide/Leap_Architecture.html. Accessed Nov. 23, 2016.

[10] *Kinova SDK User Guide*, 1st ed., Kinova, Boisbriand, Canada, 2012, pp. 1–33.

[11] M. Spiegelmock, *Leap Motion Development Essentials*, 1st ed. Birmingham, United Kingdom: Packt Publishing Ltd, 2013, pp. 5–13.

[12] Developers. (n.d.). Leap Motion Developers. [Online]. Available: https://developer.leapmotion.com/documentation/cpp/devguide/Leap_Overview.html. Accessed Oct. 18, 2016.

[13] Setting up a project. (n.d.). Leap Motion Developers. [Online]. Available: https://developer.leapmotion.com/documentation/cpp/devguide/Project_Setup.html. Accessed Oct. 15, 2016.

[14] Connecting to the Controller. (n.d.). Leap Motion Developers. [Online]. Available: https://developer.leapmotion.com/documentation/v2/csharp/devguide/Lea.p_Controllers.html.

[15] Kinova API programming guide, 1st ed., Kinova, Boisbrand, Canada, 2012, pp. 19–20.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California