10 MAY 2012

HAPTIC INTERFACE FOR SURGICAL MANIPULATOR

GROUP 5

THE FINAL REPORT

Students: Manish Mehta, Piyush Poddar, Jessie Young
Mentors: Dr. Mehran Armand, Michael Kutzer, Ryan Murphy

ABSTRACT

A surgical manipulator, the JHU/APL snake manipulator, intended to make hip osteolysis surgery more effective and faster with fewer complications has been developed. It has translational, rotational, and bend modes of motion, and its current controller utilizes a keyboard and mouse. In this paper, we describe the successful development and implementation of two distinct interfaces for the manipulator using a PHANTOM Premium haptic controller, which aim to increase the intuitiveness of control. The first interface allows the user to select a target position and have the manipulator move to it. The second allows the user to continuously control the position of the manipulator. We also discuss our various additions and enhancements to the base interfaces, including the use of audio, visual, and force feedback.

1. BACKGROUND

1.1. PROBLEM

The number of total hip arthroplasties (THAs) is expected to increase by over 174% in the next twenty years. Component wear and osteolysis, the active resorption of bone around components, are the primary phenomena responsible for the shortened life spans of total hip replacements. Wear of the polyethylene liner of the implant causes macrophage activation and subsequent bone resorption. This osteolysis comprises the integrity of the bone and thus introduces the need for frequent revision surgery to remove the lesion and maintain the previously-used fixed acetabular component. Osteolytic lesions of the bone around the implant, if not removed, may lead to complications such as bone fracture or component loosening or disconnection [3].

Minimally-invasive approaches aim at replacing the polyethylene liner while preserving the acetabular and femoral components of the THA. In this manner, surgeons minimize the risk of introducing bone fracture. Minimal-invasiveness is achieved by accessing lesions through screw holes in the bone drilled from the original implant. A major challenge, however, is fully accessing the entire volume inside the lesion to clean the cavity; studies have shown that on average less than half of the lesion is cleaned during manual procedures using curettes and other tools. Eventually, this lack of coverage forces the need for majorly invasive surgery in which the hip replacement is removed, the lesion is cleaned out, and another hip replacement is introduced [3]. Of course, an additional issue is that since the surgery site is being accessed through a screw hole during the minimally-invasive procedure, the surgeon also cannot visualize the lesion. He must use x-ray fluoroscopy,

though this use must be limited to prevent overexposure of the patient and staff to radiation. Therefore, a highly dexterous manipulator that can cover the majority of the volume inside the lesion during the minimally invasive surgery as well as a way to track it is essential.

The Johns Hopkins University Whiting School of Engineering and Johns Hopkins Applied Physics Laboratory have developed such a cable-driven surgical manipulator system (see 'Devices' below). However, accurate and smooth operation of the manipulator is difficult, and the system lacks force feedback that allows the surgeon to "feel" his way around the cavity as he does during manual surgery. Since the system does not have a navigation system that allows the surgeon to orient himself within the lesion, he can only estimate where in the cavity the manipulator is currently located. Prior research in the surgical value of haptic feedback has shown that robotic systems incorporating haptic feedback can significantly improve surgical performance. Shortened operative times; increased consistency, precision, and performance; and decreased frequency of surgical errors are all commonly demonstrated benefits drawn from comparing robotic systems with and without haptic feedback [6][8]. With no consensus as to how best implement haptic feedback, a variety of haptic-deliverance paradigms have been implemented and evaluated. With the integration of haptics into the robotic control interface, we aim to bring the aforementioned benefits to the cable-driven surgical manipulator system during osteolysis revision surgery.

1.2 DEVICES

1.2.1 JHU/APL SNAKE MANIPULATOR

The Johns Hopkins University Whiting School of Engineering and Johns Hopkins Applied Physics Laboratory have developed a cable-driven surgical manipulator system. Simulations of the system have suggested 85–95% coverage rates of surgically relevant osteolytic cavities [3], a significant improvement from traditional manual surgical methods. It has a snake-like cannula that can access osteolytic lesions through the lumen of a larger, rigid guide cannula [3].The actuation unit consists of a Y-θ stage with cable drive motors. The stage is composed of a translational (Y—forward and back) actuator and a rotational (θ) actuator, both of which are controlled by DMX-UMD-23-3 integrated stepper motors (ArcusTechnology, Inc.) [3]. The manipulator can also be bent in plane in either direction by adjusting cable lengths with PMX-2ED-SA 2-axis stepper motor controllers (Arcus Technology, Inc). These directions are denoted as the X* and Y* direction [**not** the same X and Y axes of manipulator space]; likewise, the cables on either side of the manipulator are correspondingly denoted as the X* and Y* cables. To bend the manipulator in the X* direction, the X* cable is shortened until the desired bend is achieved. The Y* cable is left slack. A similar process is used to bend the cable in the Y* direction.

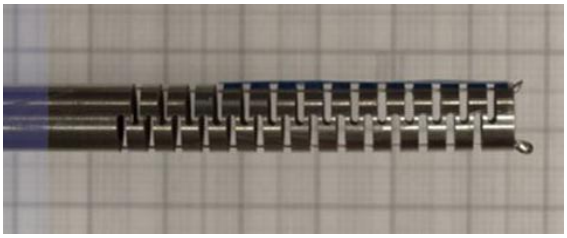

Figure 1: The manipulator in a straight configuration

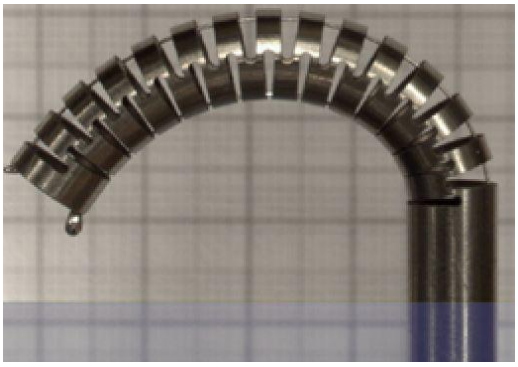

Figure 2: The manipulator in a bent configuration

The manipulator also has two 10-bit analog inputs at its base on either side[3], which report the loads on the cables for both the X* and Y* directions.

Currently, the actions of all motors and so the movement of the manipulator may be fully controlled using keyboard commands via the MATLAB keystroke controller as the motors are interfaced with the PC via USB. An application may control the motors by invoking an ASCII command set on the PMX and DMX C++ dlls. In this way, the keystroke controller also allows for custom motor commands to be sent to each motor and for force readings, positions readings, etc. to be polled from each motor as specified in the PMX and DMX user's guides. Finally, video is available via a camera mounted on the unit. However, the camera translates with the manipulator depriving it of translational information compromising some of its utility.

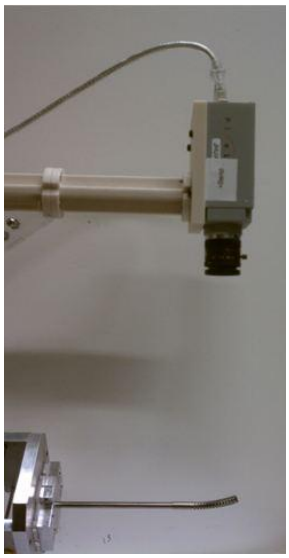

Figure 3: The manipulator with camera

1.2.2 PHANTOM PREMIUM

The PHANTOM Premium is a haptic controller that allows for movements and force feedback in 6 degrees of freedom (DOF). Studies using this system have shown the benefit of using force feedback is two-fold: the applied forces can both act as a physical constraint on hand motion and as a guide to for planning intended motion [6][9]. The PHANTOM also reports positional and velocity measurements for the 6 DOF. Finally, the PHANTOM has a button on the stylus that can be toggled at the preference of the user.

Figure 4: The PHANTOM Premium haptic controller

Figure 5: The PHANTOM Premium stylus, complete with button

Interacting with the PHANTOM is done through a C++ interface. Within this interface, multiple options are available. For example, force feedback can be applied to simulate a variety of different natural forces, including friction and viscosity. In addition, the PHANTOM can be made to act as though the stylus tip is attached to a given point in space via a spring of given spring constant. Another mode of force feedback is used to constrain the workspace of the PHANTOM. A CAD model can be loaded and the PHANTOM will generate force feedback to simulate the dimensions and shape of the object. A visualization of the workspace is also generated.

2. APPROACH

2.1 INTERFACES

In designing the interface between the PHANTOM and the manipulator, two different methods of control were considered. The first is based on positional data and allows the user to move the manipulator to a specific target location.

The second interface allows for continuous control of the manipulator via the PHANTOM. It relies on velocity data gathered from the PHANTOM for translation and rotation and positional data for bend.

2.1.1 POINT/CLICK

The point/click interface is based on positional control between the PHANTOM and the manipulator. Essentially, the position of the PHANTOM, relative to a pre-defined origin, is passed into the control program, scaled, and then the manipulator is made to move to that target position.

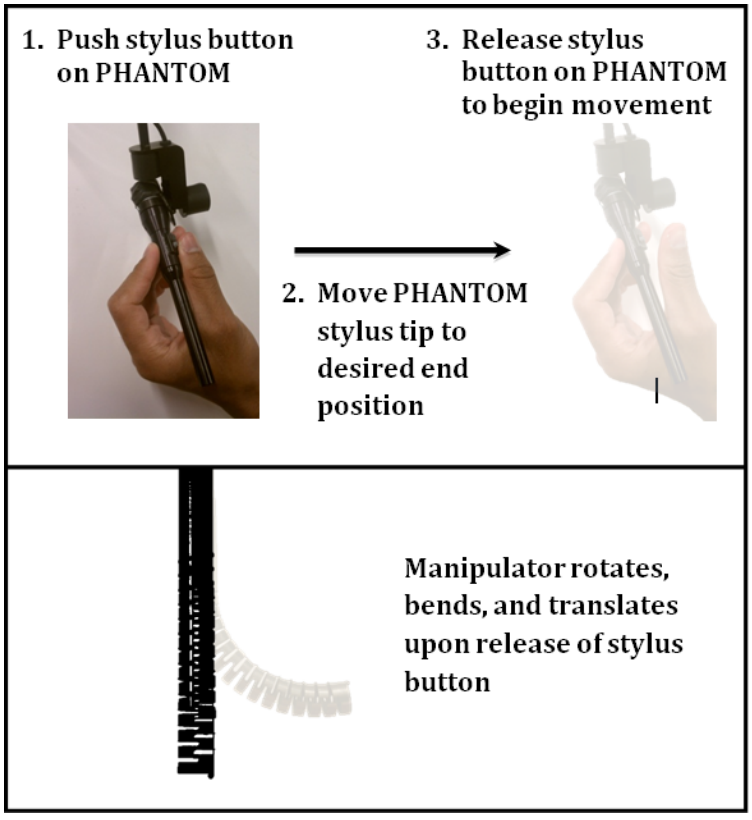

Figure 6: Point/click interface

Movement in this mode is achieved through the inverse kinematic model. Such a model allows the position of the PHANTOM Premium to be read as the desired position of the manipulator with some scaling. This desired position will then be the target position for the manipulator. Because of its motion, a natural way to model the workspace of the manipulator is as a cylinder. Thus, its movement can intuitively be written in cylindrical coordinates. In this scheme, translation is in the y-plane of the manipulator, rotation is in the x-z plane of the manipulator and about the y-axis, and the bend of the manipulator is$r = \sqrt{x^2 + z^2}$.

Rotation and translation were mapped by finding the number of motor pulses needed to translate/rotate a certain distance. These conversion factors were then tested repeatedly and updated with each trial. The final values predicted final position to sub-millimeter/sub-degree accuracy. These conversion factors are in the code and allow a desired movement to be translated into the necessary number of motor pulses to complete that movement.

Table 1: Conversion factors for translation and rotation

Translation conversion factor	Rotation translation factor
52.79 mm/pulse	22.29 deg/pulse

Mapping bend in this way proved to be trickier because of the cable-driven actuation of the unit. During calibration, the configuration of the manipulator (i.e. x and y coordinates of the tip) was correlated with the number of motor pulses to achieve that configuration. Load data was also taken. Then, while the manipulator was operated, the number of motor

pulses needed to achieve a desired configuration was known. Calibration was done experimentally by bending the manipulator a certain number of motor pulses with constant step size and tracking the position at each point using MATLAB's *ginput* function which allows users to record mouse click positions. Using these points as a foundation, linear interpolation is used to find the number of motor pulses required to bend the manipulator to a certain configuration and thus achieve the target position. This also has the benefit of acting as a software stop on the bend of the manipulator. During calibration, the manipulator is not allowed to bend past 90 degrees, well within the safe bend range of the manipulator. Later, if a desired position requires bend outside the range measured during calibration, the number of motor pulses allowed is simply the maximum measured number of pulses.

A major issue with this control scheme is hysteresis—that is, the dependence of the system's current state on its past states. Because of cable slip that occurs during normal manipulator operation and other issues, the relationship between motor pulses and manipulator configuration is not constant. This leads to inaccuracy in the manipulator's motion to a target position. Another issue occurs in that cables must be replaced when they break. The variance in material properties and the length and tightness of the cable in the unit leads to inconsistencies between predictions based on calibration data and actual movement.

To correct this, various methods have been considered. These include also correlating configuration with the load read by the load cells at the manipulator tip. When cables slip, are tightened, or have to be replaced, their slack length changes–that is, the amount of

cable that must be cycled through before it becomes taut and is able to bend the manipulator changes. However, load data can be used, rather than motor pulses, to predict end-effector position. As the relation between load and configuration seems to be constant over many trials and movements, this may serve as a reliable method of handling changing slack lengths.

Another method involves calibrating the manipulator prior to each session. This also leverages the fact that the relation between load and configuration is constant. Rather than performing the entire calibration process each session, the manipulator bends until the load cell reaches a certain threshold. The number of motor pulses required to reach this load is compared to the number of motor pulses required to reach the same load in the old calibration. A new calibration table is made that is offset by the difference in required motor pulses. Thus, prior to each subsequent session, the manipulator can automatically calibrate itself. Automating this prevents serious inconvenience to the user. The method of off-setting the motor pulse calibration rather than positioning based on load was chosen as calibrating on load would fail when the manipulator was contacted externally.

To actually operate the manipulator in this mode, users move the PHANTOM stylus tip to a target position. They then click and release the button on the stylus to make the manipulator move to the target position (see Fig 6).

2.1.2 Continuous Motion

The continuous motion mode allows the user to continuously control the motion of the robot. Moving the PHANTOM stylus tip forward and back without the button pushed translates the manipulator forward and back. Rotating the PHANTOM stylus about a center

point without the button pushed rotates the manipulator. The center point is defined as the point where the button was released. Finally, bending the manipulator is done by holding down the button and moving the PHANTOM stylus in the bend plane.

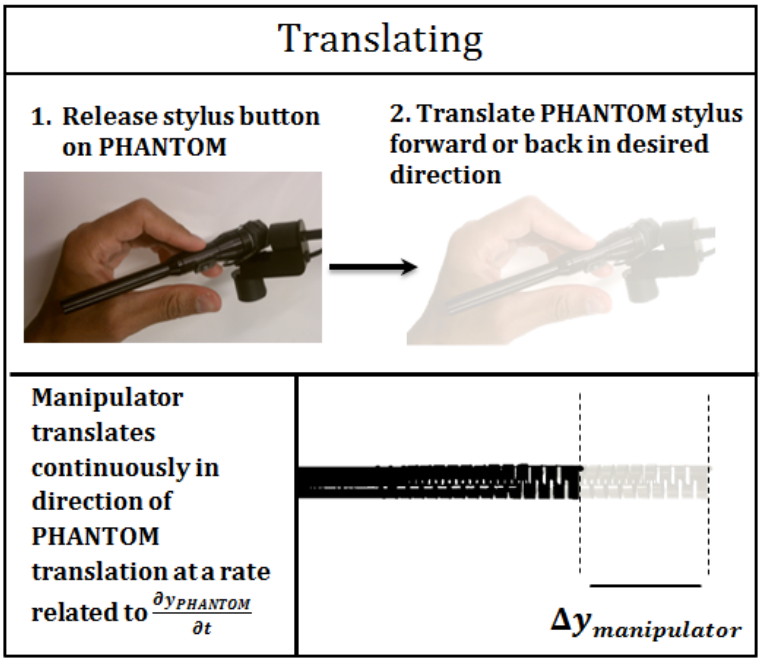

Figure 8: T the manipulator with the continuous control interface

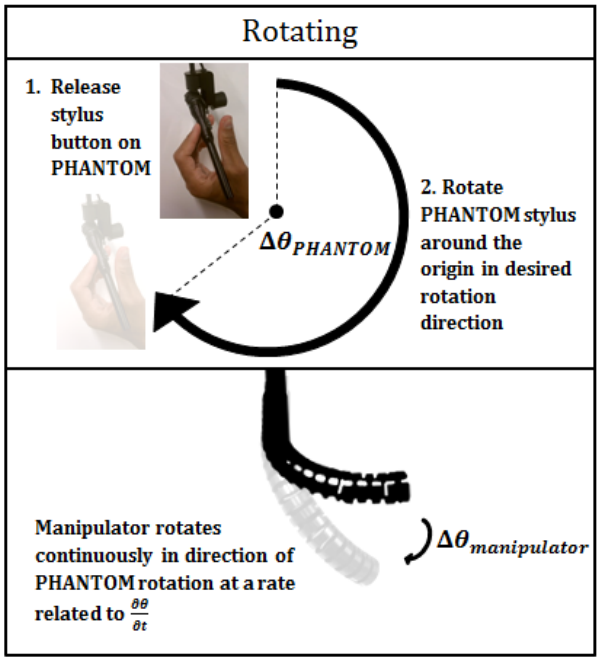

Figure 7: Rotating the manipulator with the continuous control interface

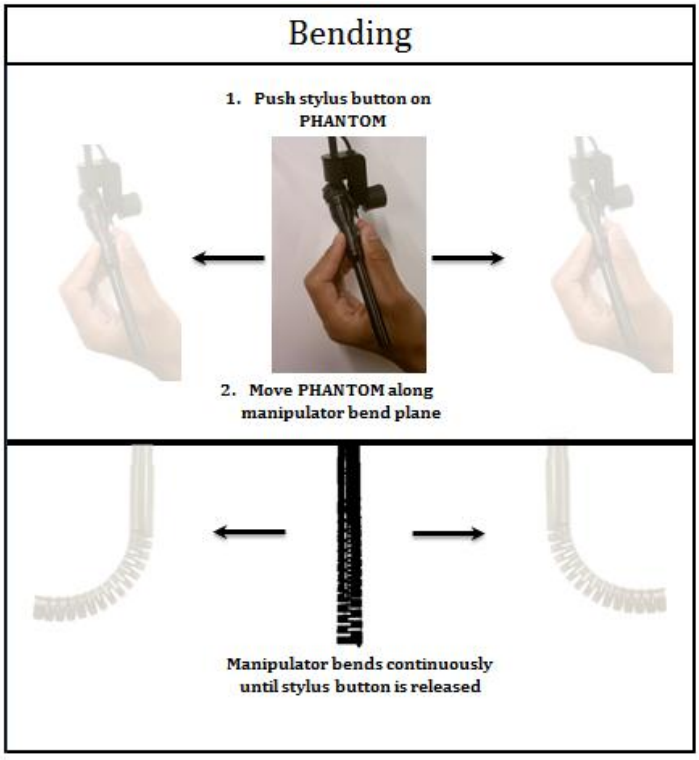

Figure 9: Bending the manipulator with the continuous control interface

Initially, position control was used for continuous motion control. However, because there is no on-the-fly position control in the software of our DMX motors for translation and rotation, this resulted in commands stacking up and being dumped or resulted in a jittery, stop-go movement. As a result, the manipulator was not as responsive or smooth as was desired. To remedy this problem, the PHANTOM was polled at varying rates instead of continuously. However, no optimal trade-off between responsiveness and smooth motion could be found.

As a result, velocity control was pursued rather than positional control. The PHANTOM can be polled for its velocity in the x, y, and z directions. Moreover, the DMX motors have functions that allow on-the-fly speed change, which precludes the problems with continuous positional control describe above. For translation, the velocity of the PHANTOM in its z direction was taken, scaled, and then fed to the motor controller, which then moves at the desired speed. The velocity was scaled through the function below:

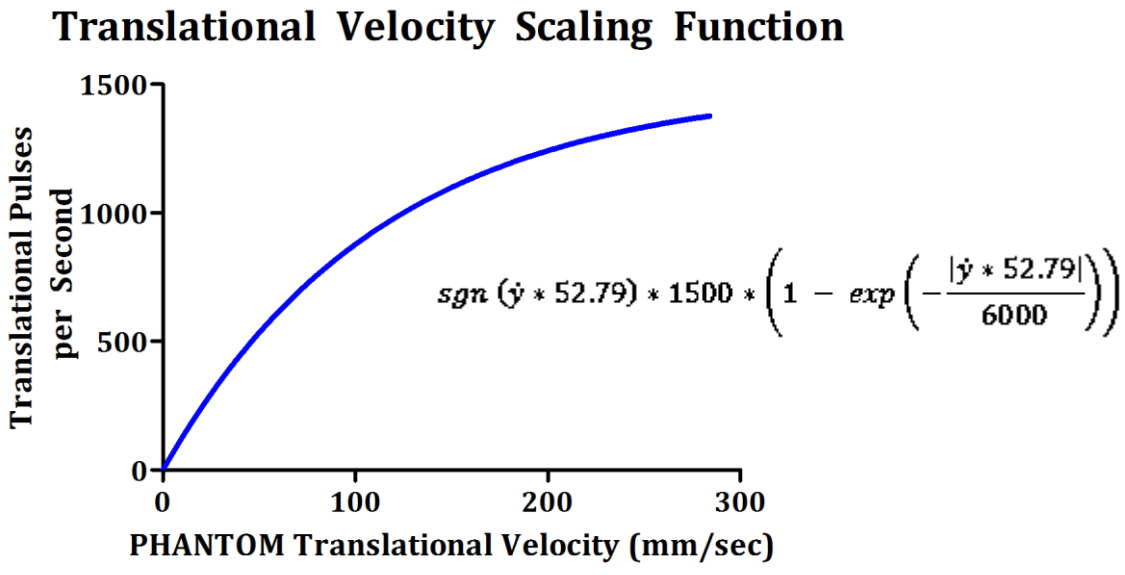

Figure 10: Translation velocity scaling function

As the function asymptotes at a pre-determined value, this function serves as a software limit on the velocity of the manipulator. Past a certain velocity, regardless of the actual PHANTOM velocity, the manipulator moves at some pre-determined safe max velocity.

Rotation proved to be more complicated than translation because the PHANTOM only reports linear velocities and angular velocity was needed to properly control the DMX rotation motor. An expression for angular velocity was found as the time derivative of θ. Again, θ is defined to be the angle of rotation from the positive x axis in the x-z plane, namely $\arctan\left(\frac{z}{x}\right)$. The expression for angular velocity is $\frac{d\theta}{dt} = \frac{x*\dot{z}-z*\dot{x}}{x^2+z^2}$. Clearly, angular velocity depends on x and z position and linear velocity. These values can be polled from the PHANTOM and so angular velocity can be found. Similar to translation, the below function was used to allow fine rotations and act as a software limit on max rotational velocity.

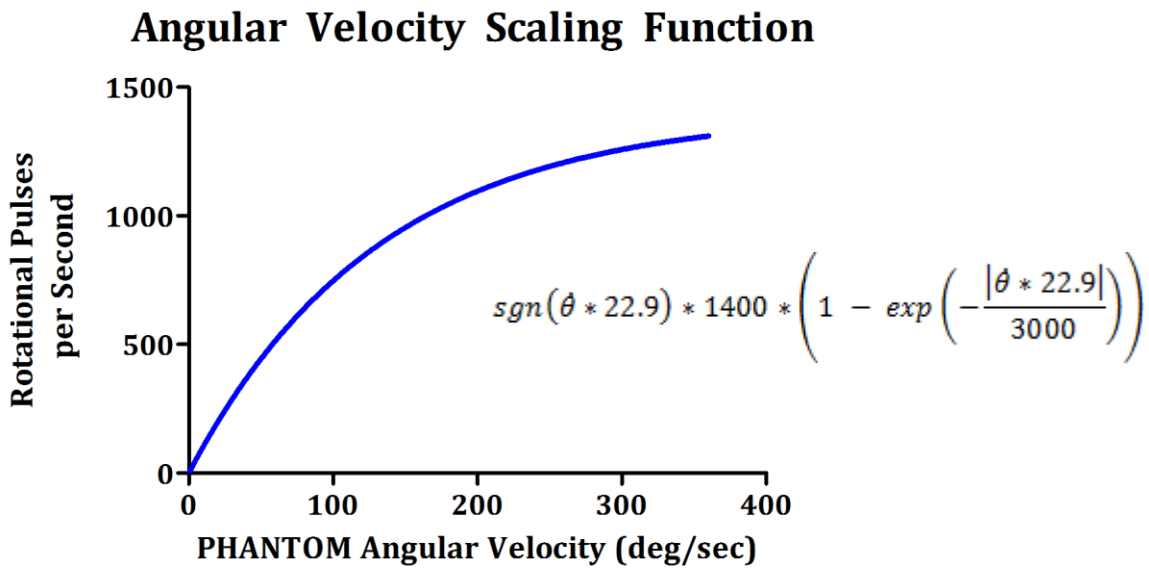

Figure 11: Angular velocity scaling function

Because of the way the manipulator is rotated (rotating the PHANTOM about a central point), some intuitive way is needed to allow the user to keep track of that central point. Otherwise, the manipulator may act in unexpected ways. To help the user keep track of this information, the PHANTOM's spring force was used. The virtual spring is attached to the PHANTOM stylus at one end and the center point at the other. The spring constant was experimentally determined to allow the user to 'feel' the location of the central point without restricting the user's motion.

For both translation and rotation, a high-pass filter was built into the code to prevent unintended motion of the manipulator due to tremor or small movements of the hand.

Bend is controlled by PMX motors with the ability to change target position on-the-fly. This precludes the problems experienced with continuous positional control of translation and rotation. As a result, positional control was used to control bend in this interface. The PHANTOM workspace was divided into two parts via a line perpendicular to the bend plane so that if the tip of the stylus was to one side of the dividing line, the manipulator would bend to that side. Additionally, the farther the stylus was from the dividing line, the more the manipulator would bend. The value of the scaling was experimentally determined to allow for fine control of the manipulator bend.

When the manipulator is unrotated, this dividing line is simply a vertical line that passes through the origin of the PHANTOM workspace. Moving the stylus to left of the line bends to manipulator to the left and moving it to the right bends the manipulator to the right. To retain this sense of intuitiveness (i.e. the manipulator bends in the same direction that the PHANTOM is moved), the dividing line must rotate with the manipulator. Thus, if the

manipulator has been rotated by 90°, the dividing line should now be horizontal so that moving the PHANTOM upwards bends the manipulator up and moving it downwards bends the manipulator down. To test which side of the threshold the PHANTOM tip is on, the following check is used. If the following expressions are satisfied, the X* cable is tightened:

$$\begin{cases} z > \tan(\theta + 90.01) * x & if\ \theta < 180° \\ z < \tan(\theta + 90.01) * x & if\ \theta > 180° \end{cases}$$

The greater than sign is flipped for a rotation of greater than 180 degrees because this is the point at which the X* cable transitions from 'above' the threshold line to 'below' the threshold line.

Initially, continuous control was in effect for bend, translation, and rotation all at the same time. While testing this interface, though, it was found that controlling all of these modes of motion at the same time was too complicated. It was especially difficult to differentiate between bend and rotation. Thus, bend was decoupled from the other modes of motion to give the user more control over the manipulator's movement. This is done by only sending commands to the DMX motors for translation and rotation while the stylus button is not pressed. While the stylus button is pressed, commands are instead sent to the PMX motors to control bend. An important factor to consider was what to do with the bend of the manipulator when the button was released. It was deemed most intuitive to have the manipulator retain the bend while being rotated and translated. To ensure that the bend can then be modified from that position once rotation/translation is complete, the position of the manipulator tip is stored as a vector whose magnitude depends on the bend of the

manipulator. This vector is translated and rotated along with the motion of the manipulator to store the position accurately as the manipulator moves.

Additionally, in response to feedback from users, translation and rotation were decoupled to allow easier control of the PHANTOM. To achieve this, the absolute value of the scaled translational and rotational velocities are compared. The minimum of the two values is set to zero, and this information is sent to the motor controller. Since translational and rotational velocities are not directly comparable, the values must be rescaled. Experimentally, it was found that doubling the rotational velocity results in proper comparisons leading to the desired motion of the manipulator. The idea behind this is that the user will move more quickly in the way of the desired motion. Thus, the unintended motion should have a lower velocity than the desired one. This essentially allows the user to 'select' which mode of motion he wants simply by moving the PHANTOM in the appropriate way.

Because bend is controlled by two cables, each actuated by a different motor, changing the direction of the bend takes a relatively large amount of time. For example, if the manipulator is bent to the left, and we want to bend it to the right, the left cable must first be slacked. Then the right cable can be tightened to achieve the desired amount of bend. The left cable must be slacked before the right cable is tightened to prevent both cables from being tight at the same time. At best, this will result in a great deal of cable slip, and at worst, it will cause a cable to snap. Both are undesirable in operation of the manipulator.

To compensate for this delay and to increase the responsiveness of the manipulator in bending, the velocity of the motors was adjusted depending on whether they are tightening

the cable or slacking it. If a motor is slacking the cable, it moves very quickly (1200 pulses/sec) while if it is tightening the cable, it moves more slowly (700 pulses/sec). This difference in velocities is instituted to prevent one cable from being tightened before the other cable is slacked. Again, this could result in major cable slip or a cable snapping. Experimentally, this was found to significantly increase the responsiveness of the manipulator.

2.2 SOFTWARE ARCHITECTURE

The OpenHaptics Toolkit is an architecture that allows users to develop applications for haptic devices such as the PHANTOM Premium. The API is implemented in the C++ programming language and allows users to interact with the device by creating and displaying environment geometries, setting force and stiffness parameters, and setting up callback responses to interactions.

The motor controllers also provide libraries that an application can use to interface with the motors rather than sending ASCII commands directly to the PMX and DMX motors to set up a serial port connection. The existing MATLAB keystroke controller creates an object that can then call the motor controller dll to control the manipulator through a GUI. This dll can also be utilized to send commands to the motors based on input from the PHANTOM.

We propose two different approaches to creating an interface integrating the PHANTOM and the motor controllers.

2.2.1 Option 1: Porting MATLAB Controller to C++

Our first option was to port, or convert, the existing MATLAB controller to C++ and then execute PHANTOM calls from within the same C++ environment. To accomplish this, we planned to build upon an existing C++ GUI developed by APL for the PMX control using the Qt GUI development framework. After this was complete, we planned to port our inverse kinematics model code, along with other relevant code, from MATLAB to C++ as well. Former students have developed a simplified forward kinematics model of the manipulator using RobWorks, a framework for simulation and control of robotics, also written in C++, which we hoped to incorporate into our program in this option as part of the visualization.

2.2.2Option 2: Calling MATLAB from C++

Our second option was to write the PHANTOM controller in C++ and create a pointer to a MATLAB engine session within the program to allow us to pass variables and commands from C++ to MATLAB. MATLAB provides a library that allows C or C++ programs to start a non-graphical (GUI-less) MATLAB process for using MATLAB as a computational engine. This library includes functions to either directly execute command strings in MATLAB or to pass buffers back and forth between C/C++ and MATLAB. This method would build upon the existing keystroke controller and would allow us to use the inverse kinematics code that we would develop in MATLAB.

2.2.3 Final Choice

We experimented with both options and began by porting the keystroke controller to C++. We successfully built and integrated the Johns Hopkins CISST libraries into the Qt program and were able to initialize and control all axes of the robot via the controller.

Concurrently, we developed and prototyped the inverse kinematics model using MATLAB for the data collection and calculations. We built upon the MATLAB keystroke controller, which contained a rich set of functionalities developed by APL. These include functions for rotation calibration, camera calibration and display, and bump tests for finding the hard limits of the motors. To allow for full utilization of the keystroke controller, we decided to use option 2 (calling MATLAB from C++). This also allowed us to save time by avoiding having to port the inverse kinematics and bend calibration functions we developed in MATLAB to C++. This proved to be a sound choice as it granted us faster turnover for prototyping and testing our algorithms.

In its current state, the program accepts as input the control mode of the manipulator (1 for point/click, 2 for continuous control). It initializes the connection to the PHANTOM to first set up a virtual scheme in which the tip of the PHANTOM is represented as the tip of cone. It then creates a MATLAB process, creates a MATLAB keystroke controller object, displays the keystroke control GUI, and initializes the robot by going through a bend calibration routine and bump tests for bend and translation motors to find the hard limits. Then, depending on the control mode, the program executes the appropriate callback functions for the press and the release of the button on the PHANTOM joystick. It logs position data from the PHANTOM and manipulator throughout the course of the program execution. When the program exists, the manipulator resets to a predetermined, constant home position. For more details, please refer to the reference manual.

2.3 AUDIO FEEDBACK

In addition to the haptic feedback built into each interface, alternative forms of sensory information for communicating force feedback from the manipulator tip to the userwere explored. In teleoperated procedures for dexterous surgical procedures, such as suturing, the use of sound has proven to increase both accuracy and precision and presents a useful alternative to fully realistic haptic feedback [5].

Our primary challenge was to communicate the interaction forces felt at the manipulator tip to the user. In order to accurately display end effector forces for truly realistic teleoperation, at least six DOF for position and orientation and additional task-specific DOFs for tasks such as gripping are required. Currently, the manipulator has no force sensors at its tip. It only has force sensors at its base that measure the load on each of the cables. Therefore, any force information may only be inferred based on the geometry of the cables.

A scheme was developed to infer the in-bend contact forces by comparing the measured load cell forces with the estimated forces from pre-recorded trials. Note that this feedback is not true force feedback. That is, the forces read by the load cells are not the forces at the tip of the manipulator. As such, this method is able to detect forces that result in changes in cable tension, such as pushing the manipulator towards or against its direction of bend if a cable is not slack. A threshold was experimentally determined and coded so that for forces below that threshold, the system will not emit a sound. This accounts for jitter in the load measurement. Above the threshold, the program will emit beeps at a frequency directly proportional to the difference between measured and expected forces.

2.4 VISUALIZATION

The use of visual cues was explored for the purpose of giving the user a better idea of his position within the lesion. Two methods were developed: a live image overlay and a software simulation.

2.4.1 LIVE IMAGE OVERLAY

The first method involves overlaying a marker on the live video feed from the camera placed to track the calculated position of the manipulator tip. To provide a sense of depth (z position) in this bird's-eye-view perspective, the size of the marker was scaled according to the manipulator tip's distance from the camera lens. A bent manipulator rotated 90 degrees so that the tip faced the camera, for example, would generate a larger marker than a perfectly straight manipulator.

2.4.2 SOFTWARE SIMULATION

While useful for validating our kinematics model, a live video overlay would not be feasible in a true surgical setting since visualizing the lesion is already a problem. There is no good way to obtain a video feed of the surgical site. As such, a software simulation was also implemented for visualization. This visualization method tracks the tip position from an oblique perspective and displays the marker's position relative to external markers. In the program's current state, there are three fixed markers that represent the positions of the posts used for user trials. Rather than using a dot of varying size, the height (z position) of the tip is conveyed along with the x and y position as the simulation is fully 3D.This implementation is faster than the video overlay, which experienced a small but significant lag, and more flexible since it is not line-of-sight dependent. Its flexibility also extends to

the fact that, as the workspace changes, a 3D CAD model of the new environment can be swapped with the current model in the simulation.

2.5 User Trials

To compare the efficacy and intuitiveness of the test mappings, trials (IRB approval obtained on 4 April 2012) were conducted on subjects inexperienced with the PHANTOM Premium. They used the PHANTOM to control the manipulator. Trials involved 6 subjects with no experience in any kind of surgery. The idea is to gauge how easily each subject can learn to use the interface effectively, allowing us to measure how intuitive each interface is. An apparatus that has three posts of different heights was designed using Autodesk Inventor and then 3D printed. On each post is a colored marker in a different orientation and at a different height.

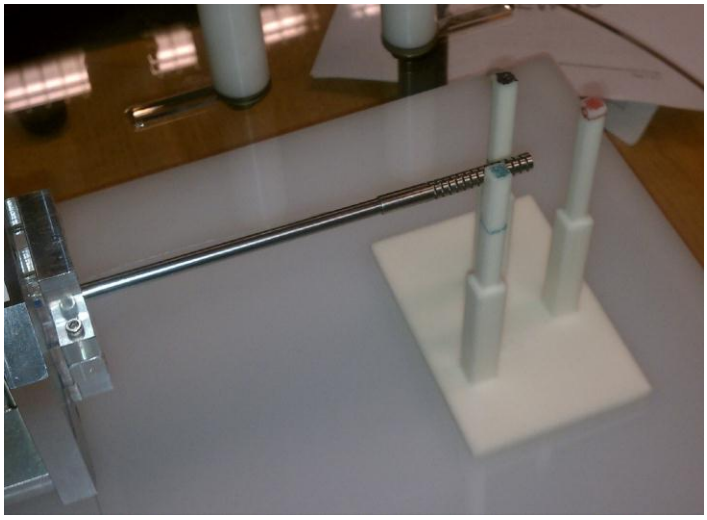

Figure 12: Trial phantom and manipulator. Subjects were to touch the green post first, followed by red and then black

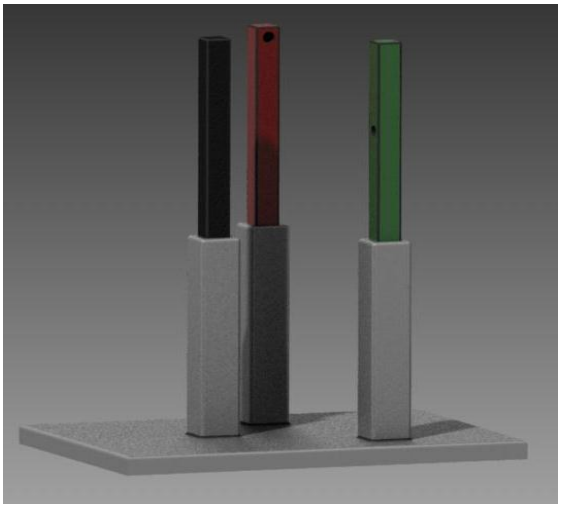

Figure 13: CAD model of the trial phantom. Note the different heights and orientations of the target holes in the posts

Subjects were asked to move the PHANTOM to cause the manipulator to touch the colored markers in a given order. The time it took for the subject to perform this task was recorded.

Subjects were then asked to repeat the task twice for each interface: keyboard control, point/click, and continuous control. The times were compared across interfaces.

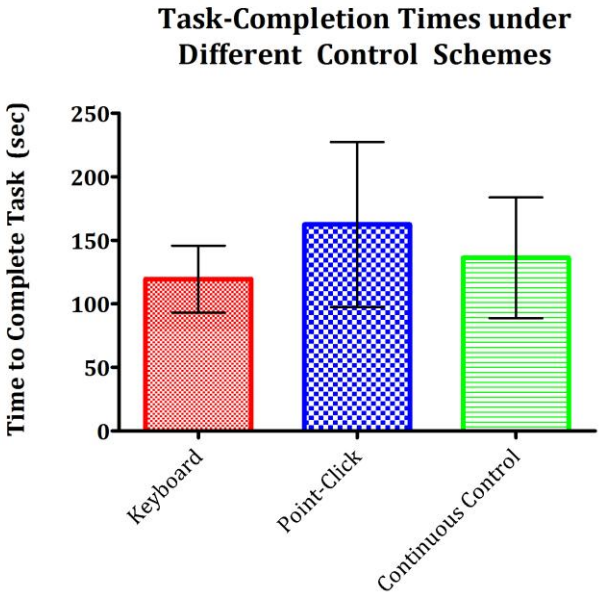

Figure 14: Task-completion times with different control schemes for inexperienced user trials

Fig 14 shows that the keyboard actually seems to be the most efficient method of controlling the PHANTOM, followed by continuous control and then point-click. However, this ordering is well within the rather large standard deviations of each set of data, indicated by the error bars. Further trials are necessary to better understand the efficaciousness of the different interfaces. Further, during testing, many of the trials were interrupted by program bugs that caused motors to stop moving or to move in unexpected ways. Other possible explanations for this discrepancy are poor instructions on how to operate in a given interface and also that users are likely much more experienced using a keyboard than with a PHANTOM haptic controller.

Qualitative feedback obtained from users after the trials has allowed for improvements to be made to each interface. These include decoupling translation from rotation for the

continuous control interface. In addition, for the point/click interface, the desired position of the manipulator was originally based on the final position (judged when the button was released) of the PHANTOM relative to its starting position (judged when the button was pressed down). Now, the target position of the manipulator is based on the absolute location of the PHANTOM when the button is pressed. The absolute coordinate system has its origin at the position where the PHANTOM tip was when the stylus button was first pressed after the session was initialized. Users also suggested a function for the continuous motion interface that allows the position tracking and manipulator movement to be paused. These, and other refinements, call for further trials to be done.

In addition to times, load data, positional data, and motor pulse data were collected from the manipulator, and positional and velocity data were collected from the PHANTOM throughout the trials. Using these, trajectories and planned paths of the users were determined, analyzed, and compared between interfaces. Two corresponding trajectories are shown below.

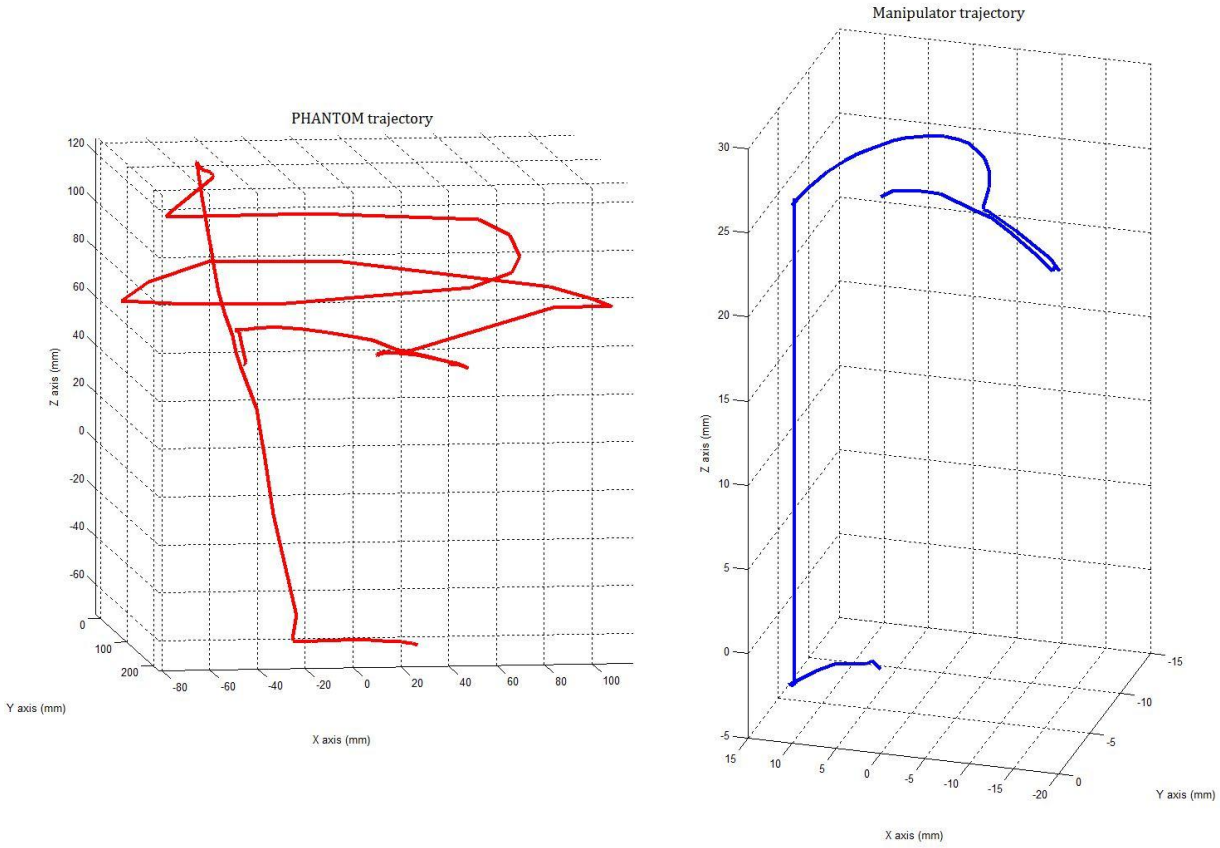

Figure 15: PHANTOM stylus tip trajectory (L) and corresponding manipulator trajectory (R)

Note that the general motions indicated by these trajectories is the same. The manipulator is first bent, then translated and then rotated before being bent further and then unbent. The scaling between the PHANTOM and manipulator workspaces is clear, especially for rotation.

In addition to inexperienced user trials, we had also hoped to have a surgeon, Dr. Mears, use our system and provide us with qualitative feedback. However, due to availability issues, it was not possible to do so prior to the poster presentation date.

3. CONCLUSION

We have made significant progress in making the control of the surgical manipulator more intuitive and easy to learn. At the beginning of this semester, the only existing method of controlling the manipulator was a keyboard/mouse controller. To rotate the manipulator, one needed to look at the computer instead of the manipulator itself (an issue that many users pointed during trials).By the end of the semester, however, we have developed two distinct control schemes: a point/click interface and a continuous motion one. Both of these allow the user to focus on the manipulator while moving the PHANTOM Premium to control its motion.

We have also explored several options for enhancing and refining these interfaces. They include haptic feedback to allow the user a sense of location within the workspace, audio feedback to enhance the user's sense of collision detection, and a 3D visualization that will allow the user to plan movements more accurately and track them as well. This last addition will be especially useful for target selection within the point/click interface.

As we continue with the project, we will work with the interfaces we have developed and continue experimenting with different combinations of implementations of the above and testing them to find the simplest and most effective interface. Though it is difficult to define a finish line as there are many variations and combinations and additions that can be made, we have and will continue to make controlling the manipulator as intuitive as possible.

4. MANAGEMENT SUMMARY

The scope of this project was rather expansive. It required working with two pieces of hardware, each with its own interface. The PHANTOM's interface was written in C++ while the four motors actuating the manipulator were controlled through MATLAB. An inverse kinematic model was needed to allow positional control of the manipulator. Additionally, the integration of haptic feedback, the creation of a visualization of manipulator position, and the use audio feedback all presented their own challenges during development.

In the course of working on the project, Piyush was at least partially responsible for every aspect of the project. He worked on designing and implementing the interfaces (including the inverse kinematic model), integrating haptic feedback into the control, developing audio feedback, and helped with user trials.

Manish also worked on designing and implementing interfaces, including the inverse kinematic model. He designed user trials, including obtaining IRB approval and analyzing data. He also maintained the website and was responsible for writing the content of this report.

Jessie worked on software development in C++. She worked on porting the MATLAB keystroke controller to C++ and on interfacing MATLAB and C++ via the MATLAB engine. She also worked on the bend calibration with the kinematics model, developed the visualization and audio feedback, oversaw user trials, and worked on the poster.

Table 2: Compares initial milestones and final milestones. Also compares planned dates and actual achieved dates

Initial Milestones	Planned Date	Accomplished	Actual Date	Final Milestones
Get PHANTOM® Premium 1.5 interfaced and running using provided Sensable software interface	20 Feb 2012	Yes	20 Feb 2012	**Get PHANTOM® Premium 1.5 interfaced and running using provided Sensable software interface**
Control software for PHANTOM®	22 Feb 2012	Yes	1 March 2012	**Control software for PHANTOM®**
Identify/create and implement test mappings from PHANTOM® to a graphical interface of the manipulator	28 March 2012	Yes	6 March 2012	**Identify/create and implement test mappings from PHANTOM® to a graphical interface of the manipulator**
Be able to control manipulator using keystrokes in MATLAB	28 Feb 2012	Yes	1 March 2012	**Be able to control manipulator using keystrokes in MATLAB**
Draft and submit IRB proposal for testing	7 March 2012	Yes	7 March 2012	**Draft and submit IRB proposal for testing**
Develop inverse kinematics model	9 March 2012	Yes	9 March 2012	**Develop inverse kinematics model**
Develop initial phantom-manipulator mapping schemes incorporating haptic feedback on paper	29 Feb 2012	Yes	9 March 2012	**Develop initial PHANTOM®-manipulator mapping schemes incorporating haptic feedback on paper**
		Yes	12 April 2012	**Convert MATLAB interface to C++**
		Yes	5 April 2012	**Calibrate DMX and PMX Motors**
Develop dynamic 3D visualization of the manipulator (eventually to become part of PHANTOM® GUI controller)	28 March 2012	Yes	8 May 2012	**Develop dynamic 3D visualization of the manipulator (eventually to become part of PHANTOM® GUI controller)**
		Yes	8 April 2012	**Verify inverse kinematics model**
Control manipulator using PHANTOM® by implementing mapping schemes and be able to gather	28 March 2012	Yes	19 April 2012	**Control manipulator using PHANTOM® by implementing mapping schemes and be able to gather**

positioning/movement data from manipulator and import into MATLAB				**positioning/movement data from manipulator and import into MATLAB**
Incorporate force feedback into mapping schemes	3 April 2012	Yes	15 April 2012	**Incorporate force feedback into mapping schemes**
Complete preliminary testing and refine mapping scheme as necessary	17 April 2012	Yes	21 April 2012	**Complete preliminary testing and refine mapping scheme as necessary**
Have surgeon provide qualitative feedback	20 April 2012	No		**Have surgeon provide qualitative feedback**
Testing and trials with inexperienced users	27 April 2012	Yes	9 May 2012	**Testing and trials with inexperienced users**
Poster presentation	10 May 2012	Yes	10 May 2012	**Poster presentation**

Table 2 shows the comparison between our initial stated milestones and our final milestones. In the course of the project, several new milestones were added, including porting the MATLAB controller to C++, calibrating the DMX and PMX motors, and verifying the inverse kinematic model. Additionally, while some milestones were met by their planned dates, others took far longer. These tasks, such as developing the 3D visualization, took longer than expected for a variety of reasons. In some cases, the development of a component of the interface needed to be delayed until later in the process that was originally anticipated. In other cases, the fulfillment of a milestone was far more complicated than originally anticipated.

Table 3: Minimum, expected, and maximum deliverables

Minimum Deliverables	
Develop well-defined and reliable interface coupling phantom and manipulator, incorporating force feedback. Well-defined here means all manipulations of the phantom along each degree of freedom has a specified mapping to the possible motion by the manipulator has a specified mapping to the phantom. By reliable, we mean that the manipulator should always respond in the same predictable fashion to movement in the phantom	Complete!
Develop a low-level inverse kinematic model utilizing simplifying assumptions (namely, that all joint angles are the same).	Complete!
Expected Deliverables	
Develop and incorporate a 3D visualization of the manipulator for testing and training purposes.	Complete!
Increase interaction force estimation to enhance/increase haptic feedback to the user.	Complete!
Schedule and document intermittent system trials with at least one mentor on a bi-weekly (i.e. every other week) basis to offer feedback on progress.	Complete!
Define and run quantifiable trials having inexperienced subjects learn to operate manipulator using the PHANTOM® interface, and perform a simple set of tasks. Compare multiple sets of scaling parameters and gestures to find best one for the specified task.	Complete!
Maximum Deliverables	
Schedule and document a final system trial with a collaborating surgeon to provide qualitative feedback for future system enhancements	In Progress
Draft a preliminary conference paper documenting the use of this haptic interface to control the manipulator.	Complete!
Draft a preliminary conference paper describing outcome of user trials.	Complete!

With respect to our deliverables, shown in Table 3, all our minimum and expected deliverables were achieved. We have also completed two of our three maximum deliverables, namely drafting conference papers describing our interfaces and the outcome of user trials. However, due to availability issues, we have not been able to garner qualitative feedback from Dr. Mears.

We plan to continue work on the project through the summer and potentially beyond. Further steps that will be taken include scheduling a meeting with Dr. Mears to get his feedback concerning the PHANTOM-manipulator interfaces. Additionally, haptic feedback will be refined to better relay collision detection to the user. We also plan to experiment with combined PHANTOM and keyboard control, voice control, and other additions to our interface to make it even more intuitive. We also plan to make our 3D visualization more sophisticated. This could involve showing a planned trajectory of the manipulator's movement for a certain target position. It may also involve having an actual simulation of the entire manipulator body.

In the course of the project, we learned the design process is full of unexpected pitfalls. Tasks that seem simple initially only become increasingly difficult as one makes progress through the details of them. We learned that IRB approval is an intensive process with much paperwork. We confirmed our suspicion that MATLAB is a much simpler language to code in than C++ and learned that simple solutions are sometimes the best solutions. For example, while developing our inverse kinematic model, we spent many weeks working with a complex forward kinematic model trying to estimate joint angles based on desired position. However, we found that using experimental results and a lookup table via interpolation worked well in controlling the manipulator. It was also far simpler and more efficient to implement. We learned how to interface C++ and MATLAB via the MATLAB engine. This proved to be very helpful as we could access the MATLAB code to move the manipulator from within the PHANTOM control interface. We were also able to personally experience the benefit of force feedback in a control interface. No amount of paper reading could substitute for feeling the difference between trying to rotate the manipulator with

and without the spring force in our continuous motion interface. Finally, we learned that while interfaces are simple to develop conceptually, they are much more difficult to actually implement because of the host of issues that must be considered and problems that can arise.

6. REFERENCES

[1] W.P. Liu, B.C. Lucas, K. Guerin, and E. Plaku, Sensor and Sampling-based Motion Planning for Minimally Invasive Robotic Exploration of Osteolytic Lesions, in Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, Sept. 2011.

[2] S.M. Segreti, M.D.M. Kutzer, R.J. Murphy, and M. Armand.Cable Length Prediction for a Compliant Surgical Manipulator, in Proceedings of the 2012 IEEE International Conference on Robotics and Automation, May 2012, in Press.

[3] M.D.M. Kutzer, S.M. Segreti, C.Y. Brown, R.H. Taylor, S.C. Mears, and M. Armand, Design of a new cable driven manipulator with a large open lumen: Preliminary applications in the minimally-invasive removal of osteolysis, in Proceedings of the 2011 IEEE International Conference on Robotics and Automation (ICRA2011), May 2011, pp. 2913-2920.

[4] B. T. Bethea, A.M. Okamura, M. Kitagawa, T.P. Fitton, S.M. Cattaneo, V.L. Gott, W.A. Baumgartner, and D.D. Yuh. Application of Haptic Feedback to Robotic Surgery.in Journal of Laparoendoscopic Advanced Surgical Techniques 14.3, 2004, pp. 191-95.

[5] A.M. Okamura, Methods for Haptic Feedback in Teleoperated Robot-assisted Surgery. in Industrial Robot: An International Journal 31.6, 2004, pp. 499-508.

[6] C.R. Wagner, N. Stylopoulus, and R.D. Howe, The Role of Force Feedback in Surgery: Analysis of Blunt Dissection, Proc. HapticsSymp., Mar. 2002.

[7] G. Tholey , G.P. Desai, A.E. Castellanos, Force feedback plays a significant role in minimally invasive surgery - results and analysis, Ann of Surg, 2005;241(1): pp. 102–109.

[8] O.A.J. Meijden, M. P. Schijven, The Value of Haptic Feedback in Conventional and Robot-assisted Minimal Invasive Surgery and Virtual Reality Training: A Current Review, Surgical Endoscopy, 2009, pp. 1180-190.

[9] C.R. Wagner, R. D. Howe. Mechanisms of Performance Enhancement with Force Feedback, WHC '05 Proceedings of the First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2005, pp. 21-29.

Group 5

Haptic Interface for Surgical Manipulator User Manual, v 1.00

Manish Mehta, Piyush Poddar, and Jessie Young

May 10, 2012

Chapter 1

User API Main Page

1.1 Introduction

The JHU/APL snake manipulator can currently be controlled using three different modes of operation: separate control of each axis (rotation, translation, and bend cables) using the MATLAB keystroke controller, a point-and-click method that moves the manipulator to the PHANTOM® Premium haptic device's position, and a continuous mode where the manipulator tracks the PHANTOM position.

Researchers at the Johns Hopkins Applied Physics Lab, in collaboration with the Johns Hopkins University have developed the MATLAB keystroke controller. The point-and-click and continuous interfaces, developed as part of the Haptic Interface for Surgical Manipulator System's Spring 2012 Computer-Integrated Surgery II project, is a Win32 program developed in Visual Studio 2008 written in C++. It interfaces with the PHANTOM's API, the Open Haptics API, and the MATLAB engine library to call functions in MATLAB, such as one that initializes the keystroke controller.

1.2 Functional Overview

The manipulator can bend left and right in a single plane. It is actuated by two wire cables threaded through the hollow cannula of the manipulator, which pull to bend the end effector. The two cables are actuated by 2 PMX stepper motors from Arcus Technologies, Inc., which interface with the PC via USB. The manipulator is mounted on a Y-θ stage, which is actuated by a DMX integrated stepper motor controller. This unit allows for rotation and translation.

1.3 Instructions

Modify the state in the main.cpp file in the Visual Studio solution to change the control mode. Set state = 1 to enable point-and-click and state = 2 to enable continuous motion control. Build the Visual Studio solution in either Debug or Release Academic Edition, which is necessary for using the Open Haptics API.

To run the program, make sure first that a ManualControlGUI (the MATLAB manipulator keystroke controller) instance does not currently exist in the MATLAB workspace. It is recommended that the user close all sessions of MATLAB to delete any current ManualControlGUI instances.

Turn on the power for both the PMX and DMX motor controllers and for the PHANTOM. The user can verify that the PHANTOM is responsive and oriented correctly if he or she suspects that it is not, by running the PhantomTest program packaged with the PHANTOM driver software to view each of its encoder inputs.

Run the main.cpp program file. Wait for a new MATLAB process window to pop up. The manipulator will then do a bump test of the PMX motors, a translational bump test, and automatic bend calibration of each cable. Wait until the motors have stopped moving. A GUI window with the MATLAB keystroke controller should pop up; if it does not, it may mean that the program was not able to connect to the serial port properly. Close MATLAB, stop the C++ program, and toggle the power for the manipulator to reset the connection. You may also need to disconnect or reconnect the FireWire camera USB cable, but this is usually not necessary. If the expected events happen in the correct order, the PHANTOM is now ready to control the manipulator tip position using one of the abovementioned modes.

Note that if at any point you notice that either one of the PMX motors has moved all the way back on its axis (towards each motor's respective J+ direction) but is unable to achieve full bend, this may either mean that the cable has slipped and needs to be re-tensioned. The MATLAB script AutomaticBendCalibration is called every time the C++ program

is run, so the user does not need to worry about manually re-calibrating bend after adjusting cable tensions or replacing cables.

To end the program, close MATLAB. The manipulator will automatically return to its home position by rotating to its start position so that the cables will not be twisted the next time the program runs. It will also slacken all cables and translate to the home position. Close the C++ program and shut off power to the manipulator and to the PHANTOM. Failure to do so may cause the hardware to overheat.

1.3.1 Point-and-Click Mode

This mode was implemented using position control. The user specifies an (x, y, z) position in Cartesian coordinates by pressing the button on the PHANTOM stylus and releasing at when the stylus tip is at the desired target position. Note that every point in the PHANTOM workspace directly maps, after being scaled by a configurable scaling factor, to a point in the manipulator workspace.

Please make sure that the previous command has finished executing (after all the motors have stopped moving) before executing the next command.

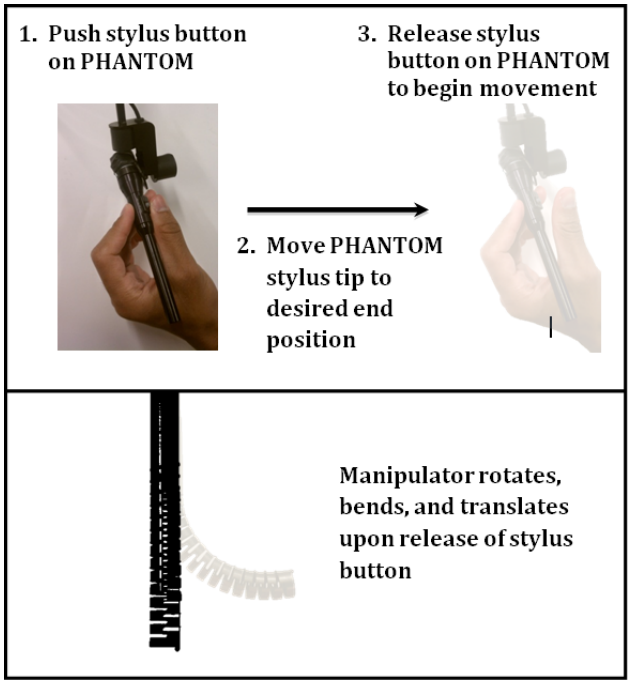

Figure 1: Point/click interface

1.3.2 Continuous Mode

In continuous mode, the manipulator tracks the position of the PHANTOM's rotation and translation when the stylus button is not held down. To begin, first press the button to activate the continuous mode. To bend the cable, hold down the stylus button and move the PHANTOM along the PMX cable axes (as currently indicated by black arrows on blue tape on the manipulator body). To rotate the manipulator, trace out steady circles in either clockwise or counterclockwise directions without pressing the button. To translate, move the stylus backwards and forwards without pressing the button.

Based on suggestions from subject trials, we de-coupled the rotational and translational degrees of freedom. Therefore, when the user is trying to only rotate, the manipulator should not experience unwanted translation and vice versa.

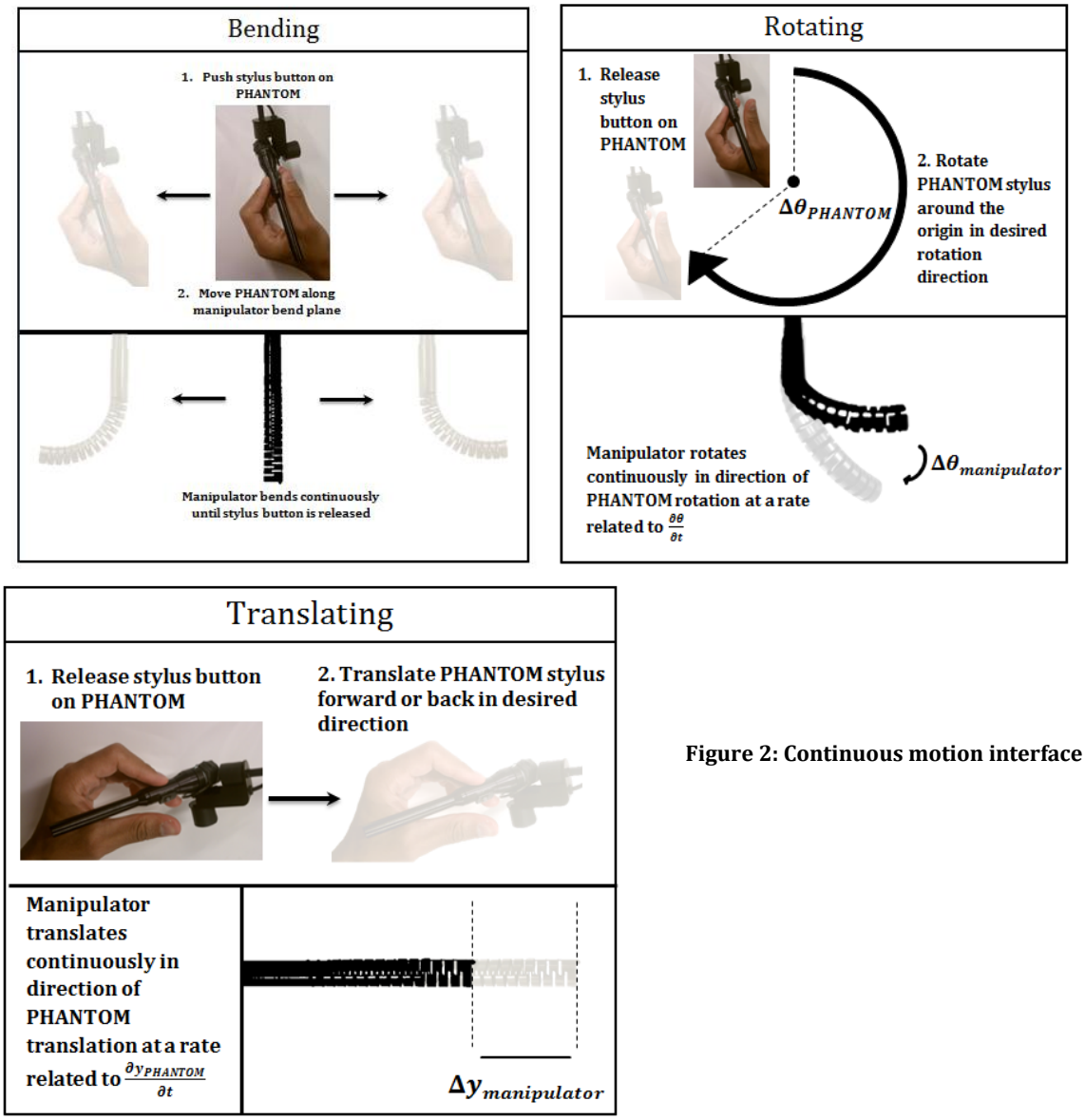

Figure 2: Continuous motion interface

Chapter 2

User API Function Listing

This section contains a selected list of functions and accompanying brief descriptions.

File Name	Input	Output
AutomaticBendCalibration◊		
AutomaticBendCalibration automatically loads the bend calibration table and updates motor positions so they are correct for the current cable lengths based on load-cell values		
BendCalPointCapture◊		
BendCalPointCapture allows for the experimental determination of the range of bend of the manipulator based on user input. Additionally, PX and PY of the relevant motor are stored as are the loads on both load cells at that position		
Cart2Joint	4: a, b, c, obj	4: pulseBendX, pulseBendY, theta, y
Given a Cartesian point, *Cart2Joint* outputs the motor pulses in X and Y direction needed to achieve position as well as rotation in degrees and translation from home position		
Cart2Motion	5: xM, yM, zM, obj, varargin	None
Given Cartesian coordinates in manipulator space, *Cart2Motion* calls the appropriate functions to move the manipulator. Also logs the previous manipulator position		
Cart2Pulse	5: a, b, c, obj, varargin	4: pulseBendX, pulseBendY, pulseRotation, pulseTranslation
Given the desired Cartesian coordinates, *Cart2Pulse* calls the appropriate functions to calculate commands that should be sent to the motors		
Cart2Speed	6: a, b, c, velA, velB, velC	2: scaledPulseSpeedTheta, scaledPulseSpeedY
Given the current location and velocity of the PHANTOM, *Cart2Speed* outputs commands that should be sent to the motor controllers in terms of motor speeds		

getForces	1: obj	4: x_volt_e, y_volt_e, x_volt_a, y_volt_a
Given the ManualControlGUI object, *getForces* finds the expected and actual forces of the load cells in the manipulator in millivolts		
initializeMotors	1: obj	None
Given the ManualControlGUI object, *initializeMotors* sets PX for all motors to 0 and is run to set the home position at the beginning of a session		
Joint2Pulse	2: theta, y	2: pulseRotation, pulseTranslation
Given the output of Cart2Joint, *Joint2Pulse* outputs commands that should be sent to the motor controllers in terms of motor pulses		
load2freq	4: x_volt_e, y_volt_e, x_volt_a, y_volt_a	None
Given the expected and actual x and y loads, *load2freq* calculates the frequency that the audio feedback should be played at based on the differences between actual and expected		
main*		
Main is the main program and initializes the PHANTOM and MATLAB keystroke controller. It also interfaces the PHANTOM with the controller via the MATLAB engine		
Pulse2Motion	5: pulseBendX, pulseBendY, pulseRotation, pulseTrans, obj	None
Given the desired number of motor pulses to send to each motor, *Pulse2Motion* sends that number of motor pulses to each motor.		
resetMotors	1: obj	None
Given the ManualControlGUI object, *resetMotors* resets both the PMX and DMX motors to their home positions		
TransBumpTest◊		
TransBumpTest moves the manipulator as far back as it can go and then moves it forward by a set amount to send it to the home position		
visualization_setup◊		
visualization_setup plots the 3D position of the CAD trial phantom posts for use in the visualization		
writeData	2: filename, data	None
Given a filename and data, *writeData* writes the data stored in 'data' along with the clock time to a text file for later analysis		
***main.cpp* is a C++program—all others are located in m-files of same name ◊These programs are all MATLAB scripts**		