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.

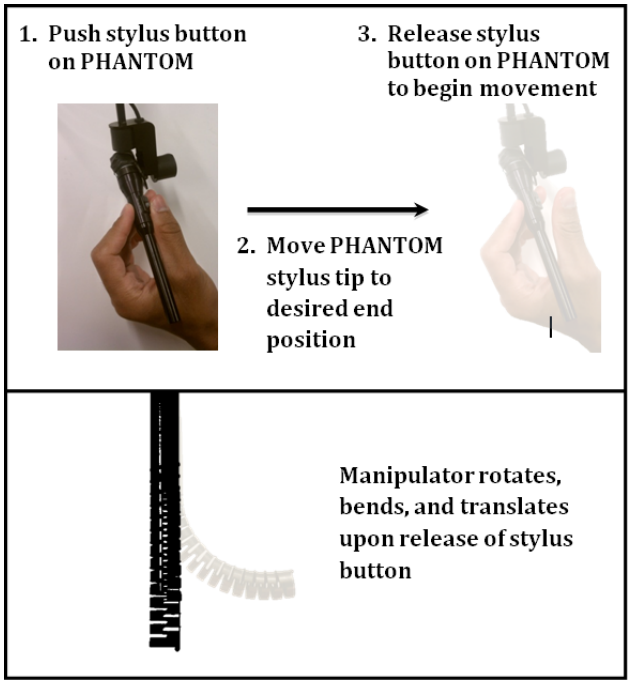

Figure 1: Point/click interface

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.

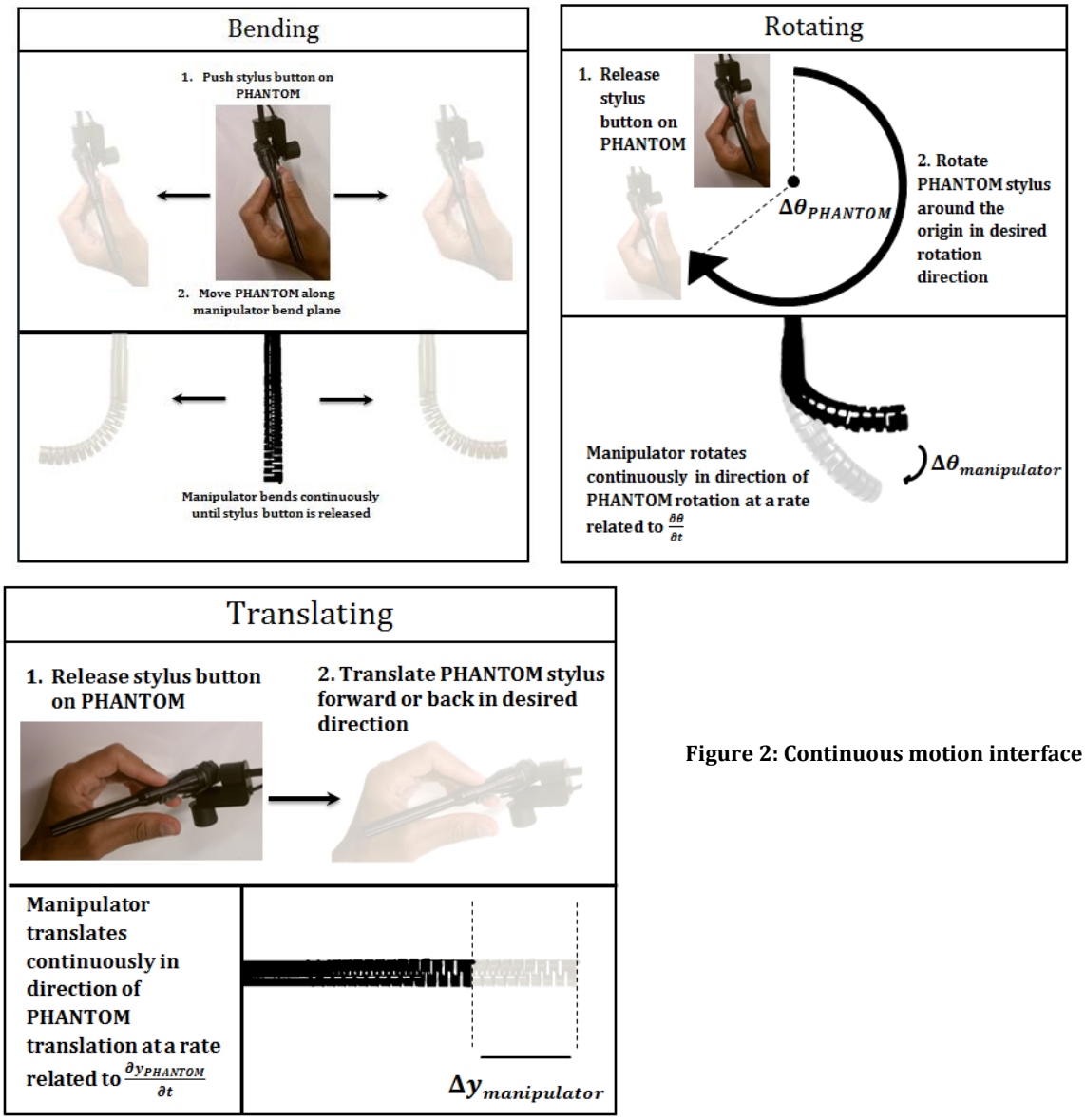

Figure 2: Continuous motion interface

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**		