Assignment 7:
Haptic Rendering on a 1-DoF Kinesthetic Haptic Device

PDF file due on Canvas by 11:59 pm PDT on Thursday, May 28, 2020
(please include code in the PDF submission)

Optional introduction (for students who are new to the use of Arduino-type boards)

A. The Hapkit board uses the same ATmega328 chip as the Arduino UNO, so we will introduce it just as one would introduce the Arduino.

   If you are new to Arduino, begin by reading these webpages:
   What is Arduino?: http://arduino.cc/en/Guide/Introduction
   Arduino Development Environment: http://arduino.cc/en/Guide/Environment

   Also, watch this 6-minute video which is an introduction to the Hapkit Board: https://youtu.be/r_90LseJauM (The video also mentions an FSR – force sensitive resistor – and SD card, neither of which we are using this quarter.)

B. If the Arduino Desktop IDE is not already installed on your computer, go to the website http://arduino.cc/en/Guide/HomePage and click on the installation instructions link for your operating system. Then follow the instructions outlined for your operating system.

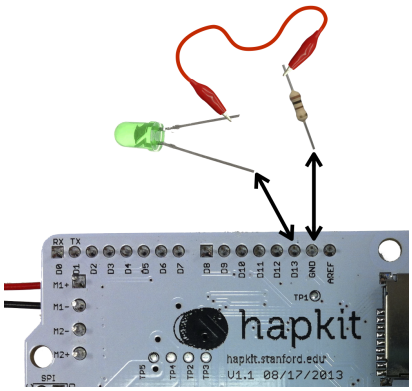
   We encourage you to use Windows or Mac OS X, since we cannot provide any support for Linux. If you can use either Windows or Mac OS X, here are some thoughts to help you decide which one to use: In our experience, maintaining serial communication between the board and the computer has been easier with Mac OS X and so this may be better for beginners. However, we have found many interesting add-on programs/libraries that are made for Windows (and fewer for Mac OS X), so if you want to use the Hapkit Board for activities other than this class, you might prefer Windows.

   Then follow the instructions specific for Arduino Uno here:
   https://www.arduino.cc/en/Guide/ArduinoUno. As you go through the installation steps on the website, here are some important points (and slight differences between the Arduino and Hapkit Board) that you should to pay attention to:

   - You will use your Hapkit Board and the included USB cable.
   - When connecting the board, be careful with the microUSB end of the cable, because the connection on the board is somewhat delicate.
   - When you select your board in the Arduino IDE, use "Arduino Uno". (There is no option for the Hapkit Board!)
   - When you select your serial port, you may find that your port is not listed. The first thing to check is that the USB cable is actually plugged in (both ends). If that's done and your board still doesn't show in the Tools > Serial Port menu, try rebooting your computer without the Hapkit Board connected, then connect it after you open the Arduino IDE. The USB connection is also something you can check – if you are using a dongle to, say, go from USB-C on the computer to the USB-A connector we provide (which then connects to the Hapkit Board using a micro-USB connector on the other end), make sure that the dongle is functional by testing communication with another USB-A device connected. If you have the latest version of Mac OS X, Catalina, you may need to update your Arduino.

- The testing sequence include a first test making an LED blink. To open the Blink sketch, in the Arduino IDE go to: File > Examples > 01.Basics > Blink, and upload/run the sketch by clicking on the Upload button (right arrow). When you upload the program, the Hapkit board does not have an LED built in to pin 13, so you will not see an LED blink unless you have added your own LED as described in the next step. (If LEDs are new to you, check out this 8-minute introductory video: https://youtu.be/kb-5Ju71JyY.)

C. To test the Blink program, connect an LED and resistor to the GND and D13 pins as shown in the image at right. (Your resistor may have a different value from the one shown.) The black arrows indicate that you will place the wires on the LED and resistor into the pin holes on the other side of the board. An LED and resistor are included in your Hapkit parts. If you don't have an intact alligator clip to connect the LED and resistor, just hold the wires together manually.



The digital pin 13 outputs 5 V when high. Our LED's specifications are found here: http://www.mouser.com/ds/2/216/WP7113GD-56080.pdf. You can use this spec sheet to find the forward voltage and recommended current. Note that the ATmega pins can source (provide positive current) or sink (provide negative current) up to 40 mA (milliamps) of current to other devices/circuits. Then, if you have not done so already, upload the Blink" program as described the installation instructions for the Arduino Uno. The LED should now be blinking! You should achieve this level of functionality before moving on to the next problem.

If the LED is not blinking, make sure:

- You have the LED anode closer to the D13 pin and the cathode closer to GND. (It doesn't matter if the resistor is before or after the LED, and it doesn't matter if the resistor is flipped.)
- That your Hapkit board is powered. The power LED (labeled PWR) should be on. The power from the USB should be sufficient, but you can plug in the power adaptor as well.
- Your program was actually uploaded to the Hapkit board. The Arduino IDE will notify you (at the bottom of the window) whether or not it was successfully uploaded.

To get a little practice manipulating an Arduino program (sketch), see if you can change the blink rate of the LED through a couple very simple changes in the program.

Having trouble with the circuit or need a refresher? These videos from Allison's online haptics course may help.
Circuit basics: https://youtu.be/K-vEusIOmDg
Blinking LED circuit on Hapkit Board: https://youtu.be/kb-5Ju71JyY
Arduino programming and blinking LED: https://youtu.be/AVS5uz1MynQ

If you have previously programmed in C, Java, or many other programming languages, you will find that no additional tutorial is needed in order to program in the Arduino environment. I recommend that you simply start with our sample code (provided with the assignment) and use the Arduino Language Reference (http://arduino.cc/en/Reference/HomePage) as needed.

However, if you are very new to programming or want a refresher, here is a 45-minute video that gives an introduction to programming with the Arduino: https://youtu.be/yGzzRCoWxBE. Beyond this, you can find a number of web tutorials that will guide you through the design and

implementation of Arduino programs (sketches). But don't be afraid to dive in and play with your Hapkit code directly -- we highly recommend learning by doing!

1. Calibrating and testing your Hapkit

In this problem, you will start with a template Arduino sketch and modify it to:
• calibrate the position sensor, and print the calibrated value to the screen
• cause the motor to output a constant torque

A. Download and examine the Hapkit Template 1 sketch from Canvas (Files > Hapkit > Hapkit_Template1.zip

This is a zip file that consists of a folder called **Hapkit_Template1**, with a single file inside, called **Hapkit_Template1.ino**. This file-within-a-folder-of-the-same-name structure is required by the Arduino IDE. Place the **Hapkit_Template1** folder in a convenient location on your computer.

Open **Hapkit_Template1.ino** in the Arduino IDE and examine its contents. The only function that you will need to edit is the **loop()** function. This main loop has five main sections. The beginning of each of these sections is labeled with a loud comment with lots of ******'s. All of the variables that you will need to use have already been defined at the top of the template code and are written in **bold** in the subsequent text.

The sections of the main loop are:

- **Section 1: Compute position in counts (do not change):** This is a somewhat complicated piece of code that computes the "raw" motor pulley position/orientation in counts. This is required because the magnet rotates many times throughout the movement of the Hapkit handle within the workspace of the device. Each time the magnet rotates 180 degrees, the MR sensor output "flips" to a new set of readings. The software needs to keep track of these flips and sum the total rotation of the magnet. The code is careful to check for these flips, which is why it stores historical data (variables with **last** in the name). At this point, you don't need to understand or edit this code in order to use it -- we wrote if for you. You just need to know that the output of these calculations is the variable **updatedPos** (in units of counts), which describes the total rotation of the motor.

- **Section 2: Compute position in meters:** You will add code to this task in order to compute the Hapkit handle position variable, **x**, from **updatedPos**. This will be done in Part B below.

- **Section 3: Assign a motor output force in Newtons:** You will need to add code to this task in order to assign a force to be output to the motor. This will be done in Part C below. In this problem, you will simply assign a constant force (variable name **force**) such as 0 or 1 Newton. In the next problem, this is where you will write the code to render virtual environments. In this problem, you will also need to add code to compute the motor pulley torque (**Tp**) required to generate the assigned force.

- **Section 4: Force output:** This code takes the desired motor pulley torque (**Tp**), computes the corresponding duty cycle, and sets the direction of the motor output (positive or negative). You will not need to change this code.

B. Calibrate the position sensor: In this part, you will see how the orientation of the sector pulley changes with the computed variable **updatedPos**. You will record data and determine a calibration equation that your sketch can use to calculate **xh** (x position of the handle) from **updatedPos (total**

rotation of the motor in units of counts).

Section 2 of the template code includes comments that show which steps go where. For example, the second substep below is called Step B.1, and in the template code it says:

```
// Step B.1: print updatedPos via serial monitor
```

Here are the substeps to follow for this part:

1. In Section 2 of **loop()**, add code to print to the value of the variable **updatedPos** to screen. *Hint:* Use the function **Serial.println()** with the appropriate argument(s).
2. After you have uploaded your program to the Hapkit Board (and keeping the USB plugged in), use the Arduino IDE's built-in **serial monitor** to view the result as you move the handle around. *Hint:* click on Tools > Serial Monitor in the Arduino IDE. You should see the values change as you move the handle around. Try checking or unchecking the box that says "Autoscroll" as you move the handles, and see what happens.
3. You might notice that when you move quickly, the MR sensor tracking code loses count, and the position measurement shifts. You will also get large position errors if the drive slips. If this happens (now or in any future lab), press the Hapkit Board's physical "reset" button (top left corner of the board) while the handle is centered (pointing straight up) to re-zero the measurement.
4. Now you are ready to calibrate the MR sensor output. Manually record a series of about 8 sector angles (in degrees) and the corresponding values of **updatedPos**. (Making 10-degree tick marks on the sector pulley will help with this.) This is your calibration data. *Note:* When the sector pulley is facing you, movements such that the handle moves to the right are positive. Movements such that the handle moves to the left are negative.
5. Now you need to fit a line to this data, where the angle (in degrees) is on the vertical axis, and the value of **updatedPos** are on the horizontal axis. There are many ways to do this. You can fit a linear trendline using Microsoft Excel, fit a first-order polynomial using Matlab, or do it by hand. The result of your fit should be an equation that looks like this: y = **m***x + **b**, where y is the angle in degrees, x is **updatedPos**, and **m** (slope) and **b** (y-intercept) are the fit parameters.
6. Program this calibration equation into Section 2 of the loop function. That is, compute the angle of the sector pulley (I suggest you call the calculated variable **ts**, for theta_sector, instead of y) as a function of **updatedPos.**
7. The variable **ts** is the angle of the sector pulley in degrees, and you want to compute the handle position (we've defined for you a variable called **xh**, for x_handle) in meters. Thus, you must (1) convert **ts** to *radians* and then (2) use kinematic equations to find **xh**. Your kinematic equations will require physical measurement of the handle length (**rh**); you can use a ruler to measure from the center of rotation of the sector pulley to a point on the handle where you expect the user to make contact.
8. Finally, print **xh** via the serial monitor. You can comment out the printing of **updatedPos** so it doesn't get in the way. *Hint:* Since **xh** is in meters, the number will be very small. As shown on this webpage, you can give an optional second parameter to the Serial.println() function to specify the number of decimal places to use; 5 should suffice.
9. With the sector pulley facing you, move the Hapkit handle as far as possible to the right (while maintaining contact with between the motor pulley and the sector pulley), and write down the value of **xh** (in meters) shown on the serial monitor. Call this the maximum position. Then move the handle as far as possible to the left, and write down the value of **xh** -- this is the minimum position. When you have completed this step, comment out the printing of **xh** to the serial monitor.

WHAT TO SUBMIT: Your collected data and trendline noting your values of **m** and **b** from the linear fit (Step B.5 above), as well as the **maximum** and **minimum xh** positions recorded in Step B.8. Thus, you have a plot and 4 numbers to give for this part.

C. Output a force: In this part of the lab, you will get your Hapkit to output a force. You will need to calculate the motor pulley torque required to generate this force. (Make sure that your power supply is plugged in, or the motor will not have enough current to generate a significant torque/force.)

Section 3 of the template includes comments that show which steps go where. For example, the first substep below is called Step C.1, and in the template it says:

```
// Step C.1: force = ?;
```

Here are the substeps to follow for this part of the lab:

1. In Section 3 of **loop()**, assign the variable **force** to equal 0.5 N.
2. In the next line of code, write an equation that computes the motor pulley torque (**Tp**) needed to generate this force. When the sector pulley is facing you, a positive force should move the handle to the right, and a negative force should move the handle to the left. You will need to use a version of the force-torque relationships given in lecture in order to perform this computation. Use the same measurements of the handle length (**rh**) taken earlier, as well as a measurement of the radius of the sector pulley (**rs**) and the radius of the motor pulley (**rp**). You can use a ruler to measure from the center of rotation of the sector pulley to the point of contact with the motor pulley for **rs**. *Note: The computation of the necessary duty cycle to achieve a certain **Tp** is already done for you in Section 4 of the code. We did this calibration for you.*
3. Now you can upload your sketch to the Hapkit Board. Firmly hold the Hapkit handle in place, so that it does not move when the force is applied. Otherwise, the handle will immediately be pushed all the way to one side, since a constant force is being applied.
4. You should be able to feel 0.5 Newtons easily, and if you start to release your hand, the Hapkit handle should move in response to the force. If the forces required to move the handle are much larger than 0.5 Newtons, this means that there could be too much friction between the shaft collar and sector pulley. Check that when the hapkit power is off, the handle can move freely.
5. Try applying different forces ranging from 0.1 to 2.5 Newtons, and write down (1) the minimum assigned force you can feel (you should feel a slight jump in force when the Hapkit starts running the uploaded program -- you can hit the reset button to look for this) and (2) the maximum force for which the cable does note slip. *Hint:* The values of force that you are writing down are not measured values, but rather the value(s) of **force** that you assign in the sketch. When you have completed this step, comment out the line of code assigning the variable **force** to a value.

WHAT TO SUBMIT: Give your minimum and maximum values of **force** recorded in Step C.5. Thus, you have 2 numbers to submit for this part.

2. Render a Virtual Spring

Using the template from the previous problem, render a virtual linear spring using the equation:
force = -k*xh
(The negative sign may need to be there or not depending on how you have hooked up your motor.)

This is not a virtual wall – the spring should be bidirectional. Start with the softest virtual spring (minimum stiffness value) that will return the Hapkit to center if you displace the handle and let go, then increase the stiffness to the largest (maximum) value that behaves "well". This is, the spring should not oscillate, buzz, or go unstable. Note that the very stiff virtual spring saturates quickly due to limited motor power!

WHAT TO SUBMIT: Give your minimum and maximum values of **stiffness** of the virtual spring. Thus, you have 2 numbers to submit for this part.

3. Render Basic Haptic Virtual Environments

Download and examine the Hapkit Template 2 sketch from Canvas (Files > Hapkit > Hapkit_Template2.zip

Create the following haptic effects in a single program, added to the template that you just downloaded for problem 3 and that builds upon the template that you used in the previous assignment (you will need to copy and paste in the lines of code you added from problems 1 and 2 into the corresponding locations in Template 2). Use `#define` commands to select which haptic effect is active, so that your program can easily be changed to switch between effects for testing.

For many of these effects, you will need an estimate of the Hapkit handle velocity. We have provided you with a discrete differentiation of the position signal and included an appropriate low-pass filter in order to get a sufficiently smooth (but not too laggy!) velocity estimate. This velocity estimate uses an approximation of the Arduino loop speed. Adding print statements will cause the velocity estimate to be less accurate (and could affect other parts of your virtual renderings).

A. **Virtual wall.** Program a virtual wall at x_{wall} = 0.5 cm and the maximum stiffness k (N/m) that you can implement without encountering significant noise or stability issues.

   You should compute your virtual wall interaction in x coordinates, which means that you will need to use your position sensor calibration and kinematics computed in the previous problem to measure the x position of your handle along the arc that is traveled. Note that $x = 0$ is when the handle points straight up. You can print the position and force to the screen for debugging, as you did in Part A.

   Show any calculations you performed. Give the maximum value of k you could display stably and relatively noise-free in units of N/m.

B. **Linear damping.** The user should feel a significant, constant, linear damping as the Hapkit handle is moved back and forth. What is the maximum damping coefficient you can use that feels good (i.e. stable, noise-free) in units of N-s/m?

C. **Bump and valley.** Create a virtual environment in which there is a bump on the left side of the Hapkit workspace (where $x<0$) and a valley on the right side of the Hapkit workspace (where $x>0$). The bump and valley should be sufficiently separated in space so that they are recognized as two separate haptic effects. The effects should feel the same whether the user is moving right to left or left to right.

Remember that you can create such effects with a one-degree-of-freedom haptic device by using lateral forces.

WHAT TO SUBMIT:

- Answers to questions in parts A, B, C
- Clean up your code and submit it along with the rest of the assignment to Canvas. Please comment well -- you should clearly explain the purpose of each line of code you added. Copy and paste the code from Arduino into a document and then convert that into a PDF.

Optional renderings

Here are some additional, option things you can render if desired!

Nonlinear friction. The user should feel significant friction as the Hapkit handle is moved back and forth. (This is supposed to feel like Coulomb friction with additional damping during dynamic friction – this is called the Karnopp model.)

Texture. Create a texture of your own design that generates a compelling sensation of moving across a rough/textured surface (with high spatial frequency) as you move the Hapkit handle back and forth.

A hard surface. Create a surface that feels harder than the maximum stiffness that can be stably displayed by the Hapkit using stiffness alone. Do this by generating a decaying sinusoid upon impact with a virtual wall. Select values for the amplitude-scaling factor (so that the amplitude depends on the impact velocity) and decay rate that generate a compelling sensation of contact, without it feeling unnaturally active.