David Lim & Bhomit Pahilwani

MAE 219

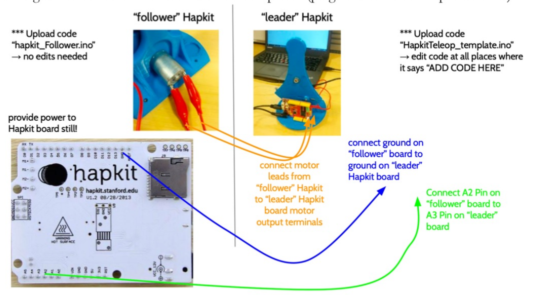
11/07/24

Assignment 5

2. Hapkit Teleoperation: Submit your Arduino code and narrative responses

A. Create a basic **proportional-derivative position-exchange bilateral teleoperator** using two Hapkits. Download template code from Canvas (two different files), and follow the instructions below:

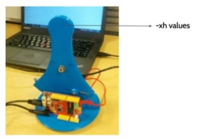
- Use one Hapkit Board to connect to both motors. Call the Hapkit with this board the "leader".
- Bring the signal from the MR sensor on the "follower" Hapkit Board to the "leader" Hapkit board as follows: → Connect A2 Pin on "follower" board to A3 Pin on "leader" board
- Connect the ground pin from "follower" board to ground pin on the "leader" board
- Don't forget that the "follower" MR sensor needs power (plug in both USB and power cable)



"follower" Hapkit
"leader" Hapkit
*** Upload code "hapkit_Follower.ino" → no edits needed
*** Upload code "HapkitTeleop_template.ino" → edit code at all places where it says "ADD CODE HERE"
provide power to Hapkit board still!
connect motor leads from "follower" Hapkit to "leader" Hapkit board motor output terminals
connect ground on "follower" board to ground on "leader" Hapkit board
Connect A2 Pin on "follower" board to A3 Pin on "leader" board

Other Notes:

- It is a good idea to check xh1 and xh2 values prior to other coding
- For template code to work as is: make sure that both hapkits output negative xh values when handle moves to the right and positive xh values when handle moves to the left



B. Play with gains and describe how the trade-off between stability and transparency manifests itself in your dual Hapkit system.

C. Implement a bilateral teleoperator that does position scaling. Include in your submission a description of the scaling effect (scale up or down), the amount of scaling, and write the equation(s) used to implement this teleoperator.

Part 2

A. The Arduino code (**A5P2.ino**) and videos (**A5P2A.mov** and **A5P2B.mov**) for this part will be submitted to Canvas along with this document.

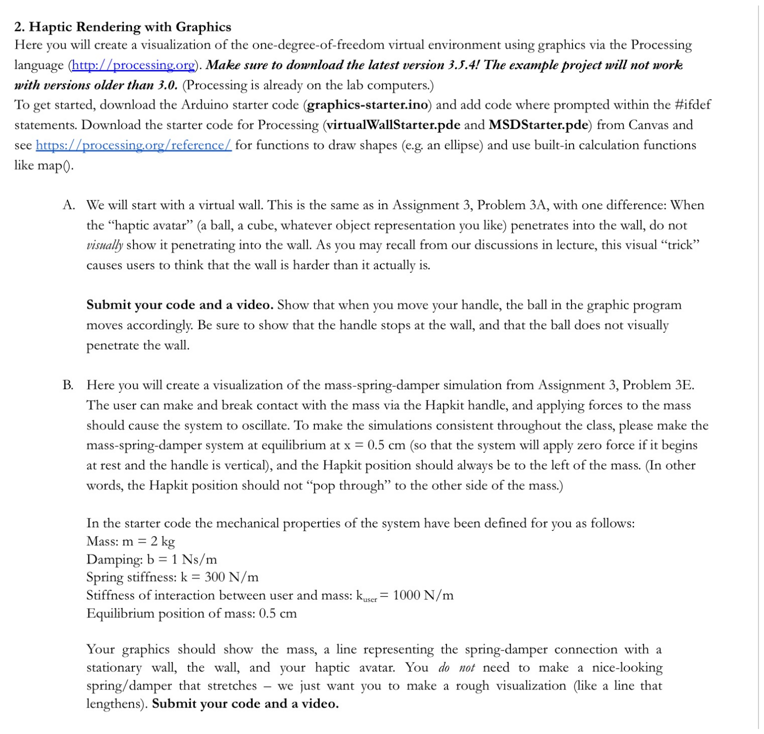
B. Increasing the Kp gain increases teleoperation transparency but reduces system stability. With Kp much less than 50, friction prevents the follower from reflecting small movements in the leader, resulting in apparent steady state error and the feeling of reduced control sensitivity. Increasing Kp reduces the maximum steady state error and the control sensitivity, but the system becomes less stable. The user feels noticeable "spikes" in force likely due to noise and quantization in the position measurements. Increasing the Kd gain from 0 initially improves the follower tracking of the leader by reducing overshoot and smoothing the motion. Kd much greater than 3 increases the noticeability of noise, since numerical differentiation of the position measurement amplifies the noise. Filtering of the velocity measurement helps reduce the effect of noise but also increases the delay.

C. We scaled the follower's motion to be twice the leader's motion (scaled up). Conversely, we also scaled the leader's motion to be half the follower's motion (scaled down) to make the system stable. Scaling was only applied to the proportional controller as to avoid amplifying noise with the derivative controller. We used the following equation to implement bilateral position exchange with scaling:

$$F_1 = k_{p,1}(x_{h,2} - s \cdot x_{h,1}) - k_{d,1}(\dot{x}_{h,2} - \dot{x}_{h,1})$$

$$F_2 = k_{p,2}(s \cdot x_{h,1} - x_{h,2}) - k_{d,2}(\dot{x}_{h,1} - \dot{x}_{h,2})$$

with $s = 2$.



2. Haptic Rendering with Graphics

Here you will create a visualization of the one-degree-of-freedom virtual environment using graphics via the Processing language (http://processing.org). ***Make sure to download the latest version 3.5.4! The example project will not work with versions older than 3.0.*** (Processing is already on the lab computers.)

To get started, download the Arduino starter code (**graphics-starter.ino**) and add code where prompted within the #ifdef statements. Download the starter code for Processing (**virtualWallStarter.pde** and **MSDStarter.pde**) from Canvas and see https://processing.org/reference/ for functions to draw shapes (e.g. an ellipse) and use built-in calculation functions like map().

A. We will start with a virtual wall. This is the same as in Assignment 3, Problem 3A, with one difference: When the "haptic avatar" (a ball, a cube, whatever object representation you like) penetrates into the wall, do not *visually* show it penetrating into the wall. As you may recall from our discussions in lecture, this visual "trick" causes users to think that the wall is harder than it actually is.

   Submit your code and a video. Show that when you move your handle, the ball in the graphic program moves accordingly. Be sure to show that the handle stops at the wall, and that the ball does not visually penetrate the wall.

B. Here you will create a visualization of the mass-spring-damper simulation from Assignment 3, Problem 3E. The user can make and break contact with the mass via the Hapkit handle, and applying forces to the mass should cause the system to oscillate. To make the simulations consistent throughout the class, please make the mass-spring-damper system at equilibrium at x = 0.5 cm (so that the system will apply zero force if it begins at rest and the handle is vertical), and the Hapkit position should always be to the left of the mass. (In other words, the Hapkit position should not "pop through" to the other side of the mass.)

   In the starter code the mechanical properties of the system have been defined for you as follows:
   Mass: $m = 2$ kg
   Damping: $b = 1$ Ns/m
   Spring stiffness: $k = 300$ N/m
   Stiffness of interaction between user and mass: $k_{user} = 1000$ N/m
   Equilibrium position of mass: 0.5 cm

   Your graphics should show the mass, a line representing the spring-damper connection with a stationary wall, the wall, and your haptic avatar. You *do not* need to make a nice-looking spring/damper that stretches – we just want you to make a rough visualization (like a line that lengthens). **Submit your code and a video.**

Part 3

A. The Arduino code (**A5P3.ino**), processing code (**A5P3A.pde**), and video (**A5P3A.mov**) for this part will be submitted to Canvas along with this document.

B. The Arduino code (same as for part A), processing code (**A5P3B.pde**), and video (**A5P3B.mov**) for this part will be submitted to Canvas along with this document.