Allison Okamura

Lab Assignment 7:
2-D Haptic Rendering

In this week's lab assignment, you will render two-dimensional haptic virtual environments using your Haplink. This laboratory has four parts:

- Step 1: Test Haplink position sensing and force output
- Step 2: Render the inside of a box
- Step 3: Render the outside of circle
- Step 4: Render something else

Demonstrate each of your virtual environments to Allison.

Step 1: Test Haplink

In this part of the lab, you will test that your Haplink is correctly sensing position and outputting force.

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

1. Check the device kinematics. In last week's lecture, we gave the kinematics of the Haplink, which you can compare against the function `calculatePositionHandleAndJacobian()` in the file **haplink_position.cpp** (note that this will be in the section under `#ifdef ENCODERS`). You'll see that the code uses the following constants, which are defined in the file **haplink_position.h**.

```
#define CX              0.0         // center point x coordinate
#define CY              0.0         // center point y coordinate
#define L_A             87.0        // length of a linkage in mm
#define L_B             100.0       // length of b linkage in mm
#define R_A             75.0        // radius of Sector a in mm
#define R_B             75.0        // radius of Sector b in mm
#define R_MA            5.0         // radius of Motor a in mm
#define R_MB            5.0         // radius of Motor b in mm
#define DELTATHETA_A    0.8727      // theta a offset in rad (50 deg)
#define DELTATHETA_B    -1.4        // theta b offset in rad (-80 deg)
```

   Use a ruler/calipers to check these measurements, using the diagram given in lecture to match these constants with physical locations on the Haplink.

   The, fix a small (but important!) bug in the code. In the file **haplink_position.cpp**, in the function `calculatePositionHandleAndJacobian` (make sure it is in the `#ifdef ENCODERS` section), add a negative sign in from of the first term in the calculation for `theta_b`, as shown here:

```
theta_b = -theta_mb*R_MA/R_A + THETA_B_OFFSET_RAD;
```

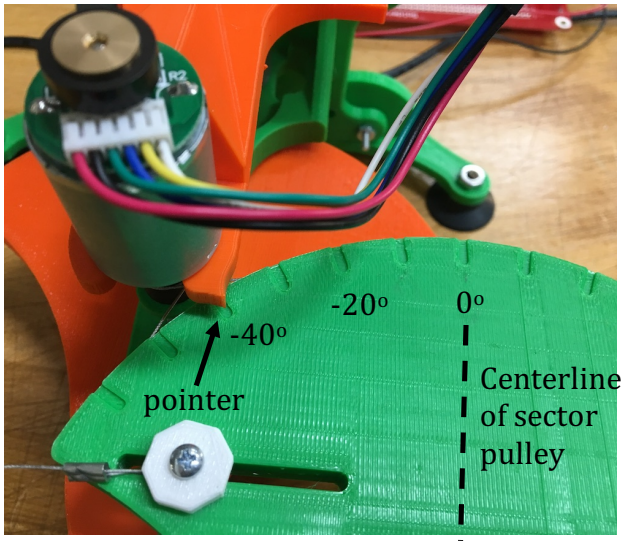
2. Hook up the electronics for both motors (see http://web.stanford.edu/class/me20n/labmaterials/haplink2-electronics.pdf, very last page), and connect the control board to your computer via USB. After you have finished the wire connections, plug in the 5 V power supply, but *wait* to plug in the 12 V power supply until Step 1.4.

3. As described in lecture, add code to **haplink_virtual_environments.cpp** (and the corresponding **.h** file) with a new function to do 2-D force output. This function could be called `haplinkForceOutput`. Also modify **main.h** to use the Haplink instead of the Hapkit, and in **main.cpp**, call your new function.

   In addition, enable printing and use a line like the following to check that you are reading from your sensors and outputting forces correctly:

```
serial.printf("%lf, %lf, %lf, %lf, %lf, %lf, %lf, %lf\r\n", getThetaA(),
getThetaB(), getRx(), getRy(), TorqueMotor1, TorqueMotor2, ForceX, ForceY);
```

4. Place your Haplink in the "zero" position. This is where Sector A is centered and Sector B is at an angle of -40 degrees. (This is the default value of `THETA_B_OFFSET` in the **main.h** file, designed so that you will start inside the virtual box that you will render in the next step!) Note that the pointy corner on the base (see image below) indicates the angle you are at, and the centerline of Sector B is zero degrees. Each tick mark on Sector B corresponds to 10 degrees. Rotating Sector B clockwise gives more negative values of `theta_b`.



Compile and download this program to your board when the Haplink is in this correct starting configuration, and then check the Serial Monitor in the Arduino environment to be sure that the measurements and outputs make sense.

5. Now that you know that to expect, place your Hapkit handle in the vertical position and plug in the motor power supply (12 V). Hold onto the handle. You should be able to feel a weak force at a 45-degree angle, since you requested 0.5 N in the x-direction and 0.5 N in the y-direction. You can increase the force values and change the relative x and y values as needed to make sure you understand the force directions. Make a note about which directions are positive/negative x, and which are positive/negative y based on the forces you feel, and make sure they correspond to the measured positions you view in the Serial Monitor.

WHAT TO RECORD BEFORE DEMONSTRATING TO ALLISON:

- Which directions are +x, -x, +y, and –y.

Step 2: Render the Inside of a Virtual Box

As described in lecture, create a virtual box centered at (26 mm, 90 mm) with a width of 10 mm and a height of 10 mm. You will render a scenario in which the user is stuck *inside* the box. The suggested value for `K_WALL_X` and `K_WALL_Y` is 500 N/m.

WHAT TO RECORD BEFORE DEMONSTRATING TO ALLISON:

- Nothing, just make sure that you (and Allison) can feel the box.

Step 3: Render the Outside of a Virtual Circle

As described in lecture, create a virtual circle (really a disk) centered at (-9 mm, 90 mm) with a radius of 30 mm. You will render a scenario in which the user is stuck *outside* the circle. We suggest a value of 500 N/m for the stiffness of the circle. With the location and size of the circle given, you can use the same `THETA_B_OFFSET` as in the parts above.

WHAT TO RECORD BEFORE DEMONSTRATING TO ALLISON:

- Nothing, just make sure that you (and Allison) can feel the circle.

Step 4: Render Something Else

Try something new! Make sure that you have a starting position that will not generate a large force as soon as the program starts running.

WHAT TO RECORD BEFORE DEMONSTRATING TO ALLISON:

- Prepare to explain to Allison what you aimed to render, before she tries it.