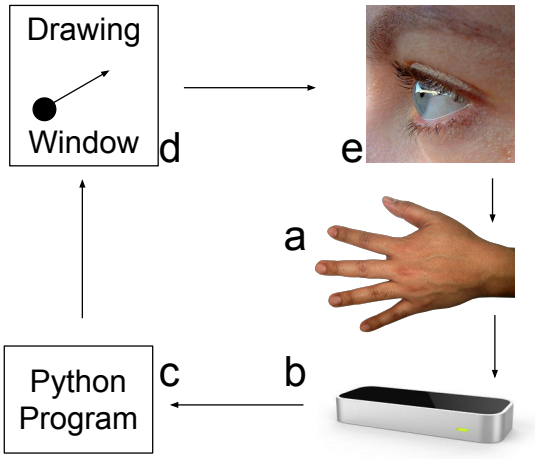

Figure 1: Establishing an interaction between the user, the Leap Motion controller and your Python program. The user moves her hand (**a**); the motion is captured by the controller (**b**); the gesture data is sent to your Python program (**c**); the data is used to update a visualization in the drawing window (**d**); the user observes the visualization (**e**) and reacts (**a**).

CS228: Human Computer Interaction

Deliverable 1

Description

In this first deliverable, you will establish a connection between a Python program and the Leap Motion controller. You will do this by...

1. Installing the Leap Motion software and then running the Visualizer demo that comes with it. When you do, you should see this.
2. You'll then learn how to draw graphics in real time using Python. When you're successful, you should see this.
3. Finally, you'll capture hand gesture data from the controller and use it to move the point as shown here.
4. This will create a continuously-running interaction between the user and the device as shown in Fig. 1. You will spend the rest of this course complicating this feedback loop in various ways.

Instructions

1. Download the Leap Motion Skeletal Tracking software from here (get V2, not Orion Beta).

2. Plug in your controller and run the Visualizer as shown here.

3. Play around with the Visualizer a bit to get a feel for how Leap Motion works. Note that it is not perfect: for certain movements of your hand, the controller fails to capture the resulting data correctly. This will be important later.

4. You now need to make a video record that you have completed the previous step. Using a smartphone, capture a few seconds of you interacting with the Visualizer as shown here.

5. Upload the video to YouTube, and include it in a new YouTube playlist called `2016_UVM_CS228_YourLastName_YourFirstName_Deliverable_1`. Make sure the playlist and the video is set to 'Public' so the grader can view and grade your submission.

6. Read the overview of the Leap Motion Application Programming Interface (API) here to familiarize yourself with what the Leap Motion device is.

---

7. Now let us switch to Python for a bit. (Python is the only language we will be using in this course.) If you already have Python installed on your computer, continue to the next step. If you don't, you have two choices.

   (a) You can install Python and then manually install the Python libraries that will be introduced below.

   (b) Or, you can download and install Enthought's free Canopy Express software. Canopy contains the Python interpreter as well as a number of libraries that we will be making use of later.

8. If you have never coded in Python before, or you're a bit rusty, you might consider working through Codecademy's online Python tutorial before proceeding.

9. To warm up, let us write a simple 'Hello World' Python program. In your favorite text editor or IDE (doesn't matter which one), create a file called `HelloWorld.py`.

10. In this file, write
    `print "Hello World!"`
    Save the file and run it. It should print out this string.

11. Let us now create a second Python program called `RealTimeDraw.py`, which will continuously draw a point that moves randomly:

    (a) `import matplotlib.pyplot as plt`

    (b) `import random`

    (c) `plt.ion()`

    (d) `xPt = 0`

    (e) `yPt = 0`

    (f) `pt, = plt.plot(xPt,yPt,'ko',markersize=20)`

(g) `for i in range(1,200):`

(h) `    xPt = pt.get_xdata()`

(i) `    xPt = xPt + (2.0 * random.random()-1.0)*0.001`

(j) `    pt.set_xdata(xPt)`

(k) `    plt.draw()`

This program uses the Python drawing library matplotlib, which is first imported into your program on line (a). Then, a library that allows you to create random numbers is imported (b). Line (c) turns on interactive graphics. Lines (d) and (e) put the point initially at the origin: position $(0, 0)$. Line (f) plots the point in the drawing window. (Details about this plotting function can be found here.) Line (g) creates a loop that will be iterated through 200 times. Line (h) gets the current x-coordinate of the point. Line (i) adds a small random number in the range $[-0.001, +0.001]$ to this value. Line (j) changes the x-coordinate of the point. Finally, line (k) re-draws the scene.

12. When you run this program, you should see a black dot moving randomly to the left and right.

13. Now insert three new lines between lines (j) and (k) that also modify the y-coordinate of the point. When you run your modified program, you should see the dot move as shown in this video. Capture a 5-second video with your phone of the dot moving; upload it to YouTube; and append it to the YouTube playlist you created in step #5.

---

14. We are now going to incorporate the Leap Motion device into this code such that you can use the tip of your index finger to dictate the position of the dot.

15. Install the Leap Motion Python SDK from here if you haven't already.

16. When installed, it will create a directory called `LeapDeveloperKit...` somewhere on your computer. Find it.

17. Copy all of the files from `LeapDeveloperKit.../LeapSDK/lib` into the directory that contains your `RealTimeDraw.py` program.

18. At the top of your Python program, include the Leap Motion python library:

    (a) `import Leap`

    Run your program. There should be no change in its behavior.

19. Put this line

    (a) `controller = Leap.Controller()`

between lines 11(f) and (g) in your code. This will create an instance of the Controller class. The controller allows you to access data captured by the Leap Motion device. The controller class is described in more detail here. Run your code; there should be no change in its behavior.

20. Now put this line

    (a) `frame = controller.frame()`

    between lines 11(g) and (h). The Controller object continuously grabs frames of data from the device while running. The new line will grab data from the current frame every time a new pass through the loop begins. More details about frames are available here.

21. We want to change the position of the dot only when you hover your hand over the device. To do so, place the following conditional just after line 20(a):

    ```
    (a)     if ( ...  > 0):
    (b)         print "hand detected."
    ```

    You will have to write some code to replace the ellipsis (the dots) on line (c). The code should cause the if clause to evaluate to `True` whenever there is at least one hand detected above the controller, and `False` otherwise. It is up to you to figure out how to extract the appropriate data from the `frame` data structure and use it in this clause to make this happen. If successful, in addition to the moving dot, you should see the message on line (b) continuously printed to a text window while there is at least one hand above the controller, and no messages printed when there are no hands above the controller. (**Hint:** You will want to use the frame.hands list, which returns a list of hand data structures. In other words, when no hands are present, frame.hands has a length of zero; when only one hand is present, it has a length of one; when two hands are present, it has a length of two.)

22. Let's now extract information about the first hand — frame.hands[0] — whenever a hand is above the device:

    ```
    (a)     if ( ...  > 0):
    (b)         hand = frame.hands[0]
    (c)         print hand
    ```

    (You can delete line 21(b) now.) Now, you should see that a hand ID message is printed out whenever you hover your hand over the device.

23. Now you need to extract information about the index finger from the `hand` data structure:

    ```
    (a)     if ( ...  > 0):
    (b)         hand = frame.hands[0]
    (c)         fingers = hand.fingers
    ```

(d) `indexFingerList = fingers.finger_type(...)`

(e) `indexFinger = indexFingerList[...]`

(f) `print indexFinger`

Refer here and here to figure out how to fill in the ellipses on lines 23(d) and (e). When it's working, you should see

```
Finger Id:1771
```

(or some other number) printed whenever your hand is over the device.

24. Now we need to grab information about the distal bone in your index finger. Before you do, read the Finger Bone Basics section on this page and study the hand anatomy figure there. We're aiming to grab information about the distal phalange: the bone in the tip of your index finger. Delete line 23(f) and replace it with

    (a) `distalPhalange = indexFinger.bone(...)`

    (b) `print distalPhalange`

    When it's working, you should see a Bone index message displayed whenever your hand is over the device.

25. Finally, the Bone data structure contains the prev_joint and next_joint data structures, which contain the positions of the base and the tip of the bone, respectively. We want the tip of the distal phalange:

    (a) `distalPhalange = indexFinger.bone(...)`

    (b) `tip = distalPhalange.next_joint`

    (c) `print tip`

    Now, instead of an ID message, you should see sets of 3D coordinates being printed: these are the 3D positions of the tip of your index finger.

26. Have a look at the ranges of the axes in the drawing window containing the moving dot: you'll notice that the x and y ranges are very narrow. We need to change them to contain the positions captured by the Leap Motion device, so add these lines

    (a) `fig = plt.figure()`

    (b) `ax = fig.add_subplot(111)`

    (c) `ax.set_xlim(-1000,1000)`

    (d) `ax.set_ylim(-1000,1000)`

    between lines 11(c) and 11(d). Line 26(c) widens the range of the horizontal axis and line 26(d) widens the range of the vertical axis. When you run your code now, you should see the dot moving imperceptibly near the center of the window.

27. Now for the fun part: let's take the x and y position of the tip of your index finger, and use it to set the position of the dot. Delete lines 11(h) and (i) in your code, and change line 11(j) to

    (a) `pt.set_xdata(tip[0])`

    This will set the horizontal position (xdata) of the point (pt) equal to the horizontal position of the tip of your index finger (tip[0]). Add an additional line after this that sets the vertical position of the dot to the vertical position of the tip of your index finger. Make sure that your two new lines are contained within the `if` clause, as that the `draw()` command is outside of it.

28. You'll notice that the dot only moves within a small part of the screen. Let's now set the range of the axes dynamically so that the dot moves across the whole screen. Our strategy for doing this is to 'push' the right hand edge of the window further to the right, if the dot goes beyond the right edge of the window. Same for the upper, left, and lower edge of the window. Here's how we do this. First, somewhere at the top of your code before you enter the if statement, create four variables that contain the edges of the window:

    (a) `xMin = 1000.0`
    (b) `xMax = -1000.0`
    (c) `yMin = 1000.0`
    (d) `yMax = -1000.0`

29. Now, add the following lines just after line 25(b):

    (a) `x = tip[0]`
    (b) `if ( x < xMin ):`
    (c) `        xMin = x`
    (d) `if ( x > xMax ):`
    (e) `        xMax = x`

    Add additional lines for `y`.

30. Now take lines 26(c) and (d) and **move** them to just before line 11(k). Change the moved lines to now use the variables, rather than the fixed values:

    (a) `ax.set_xlim(xMin,xMax)`
    (b) `ax.set_ylim(yMin,yMax)`

31. What happens if you store the third value from `tip` in y, rather than the second value? (No need to submit an answer; just try it and see what happens.) Switch y back to the second value in `tip`.

32. Capture a 5-second video now of *you* moving the dot; upload it to YouTube; and append it to the YouTube playlist you created in step #5.

33. Rename `HelloWorld.py` to `Del1.py` ('Deliverable 1').

34. Finally, submit your playlist to BlackBoard by the deadline. All you have to do is copy and paste the URL that points to your YouTube playlist into the BlackBoard submission. Make sure all the videos and the playlist are set to public or unlisted (not private).

Notes

Some students have reported problems if they run their program inside of an IDE. If you are having problems, try running your program directly from the directory in which it is stored.

If you are having problems and you are working with a Mac...

1. Open a terminal.
2. Navigate to the directory containing your Python program.
3. Type `python HelloWorld.py`

If you are having problems and you are working on a PC...

1. Navigate to the directory containing your Python program.
2. Double click on `HelloWorld.py`