CS228: Human Computer Interaction

Deliverable 2

Due: Monday, September 15, 11:59pm

Description

In this deliverable, you will simplify the way in which you extract data from the Leap Motion device. Then, you will expand from drawing in two dimensions to three dimensions.

1. In the previous deliverable, we relied on Leap Motion's `Listener` class: every time it captured data from the device, we used it to alter our drawing. In this deliverable we are first going to simplify the way in which our Python program interacts with the Leap Motion device. In short, we are going to create an infinite loop, and during each pass through the loop we will grab the latest data from the device rather than waiting for `Listener` to tell us there is data available.

2. To begin, create a new, empty python program. Call it `Del2.py`.

3. Create an infinite loop:

   (a) `while ( True ):`

   (b) `    print 1`

   when it runs, '1' should be printed to the screen until you stop your program.

4. Now let us grab some Leap Motion data during every pass through this program. First, include all of the libraries that we will need for this deliverable by placing this line

   (a) `import Leap, sys, thread, time, random`

   at the top of `Del2.py`. Run your program; there should be no change in its behavior.

5. Now create an instance of the `Controller` class by placing this line

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

   just after line 4(a) (and thus just before the infinite loop is entered). Run your program; there should be no change in its behavior.

6. Now place this line

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

just below line 3.(a). This will grab the most recent frame of data from the Leap Motion device during each pass through the infinite loop. Run your program; there should be no change in its behavior.

7. Replace `print 1` with `print frame` and re-run your program. You should now see the ID number of each captured frame output to the screen.

8. Now just below line 6.(a), insert a conditional that is only entered if a hand is detected above the device, just like you did in the previous deliverable. Make sure that the frame ID is now only printed if this condition is met:

   (a) `if (...):`

   (b) `    print frame`

9. In this deliverable we are going to draw three-dimensional rather than two-dimensional graphics. To do so, we will need to import some new libraries. At the top of your program, add the following lines:

   (a) `import matplotlib.pyplot as plt`

   (b) `from mpl_toolkits.mplot3d import Axes3D`

   (c) `import matplotlib`

   Run your program; there should be no change in its behavior.

10. Now, initiatialize a 3D drawing window by placing this lines

    (a) `matplotlib.interactive(True)`

    (b) `fig = plt.figure( figsize=(8,6) )`

    (c) `ax = fig.add_subplot( 111, projection='3d' )`

    (d) `plt.draw()`

    just before you enter the infinite loop. When you run your program now you should see three orthogonal axes, but nothing drawn within them.

11. As we did before, let us extract the position of the tip of the index finger. Place the following lines

    (a) `hand = frame ...`

    (b) `indexFinger = hand ...`

    (c) `distalPhalangeOfIndexFinger = indexFinger ...`

    (d) `positionOfBoneTip = distalPhalangeOfIndexFinger ...`

    (e) `print positionOfBoneTip`

just after line 8(a). Fill in the ellipses with the appropriate code that captures this data. Also, delete line 8(b). Now when you run your program you should see triplets of numbers printed whenever a hand is above the device. These are the x, y and z coordinates of the tip of your index finger.

12. Let us also extract the position of the base of this bone. Add these lines

    (a) `positionOfBoneBase = distalPhalangeOfIndexFinger ...`

    (b) `print positionOfBoneBase`

    after line 11(e) and replace the ellipsis with the appropriate code. Now when you run your code you should see two triplets, corresponding to the current position of the base and tip of this bone.

13. Now let us store the x, y and z coordinates of the bone's tip and base. The following lines

    (a) `xBase = positionOfBoneBase[0]`

    (b) `yBase = ...`

    (c) `zBase = ...`

    (d) `xTip = ...`

    (e) `yTip = ...`

    (f) `zTip = ...`

    should be added after line 12(b) and the appropriate code should replace these ellipses.

14. Now let us draw a line in 3D that represents this bone. After line 6(a), place this line

    (a) `lines = []`

    which will create a new data structure that will store all of the lines drawn to the screen.

15. Now, add these lines

    (a) `lines.append(ax.plot([xBase,xTip],[yBase,yTip],[zBase,zTip],'r'))`

    (b) `plt.draw()`

    just after line 12(b). This ensures that during each pass through the loop, a new red line is drawn to the screen. When you run your code now you should see red lines accumulating in the drawing window.

16. You can remove lines 11(e) and 12(b) now, as we have a visual (rather than text) representation of the data we have captured from the device.

17. Now we will erase a line just after we draw it. Add the following lines

    (a) `while ( len(lines) > 0 ):`

```
(b)        ln = lines.pop()
(c)        ln.pop(0).remove()
(d)        del ln
(e)        ln = []
```

just after line 14(b). When you run your program now you should see a line that moves whenever your hand is over the device. Note also that the axes change as well.

18. We are now going to draw all of the bones detected by the device. To do this, you must create two nested loops: one that iterates over each of the five fingers, and another that iterates over each of the four major bones in each finger. These two loops should be placed immediately after line 11(a), and should replace lines 11(b)-(d) so that you now have

```
(a) hand = frame ...
(b) for i in range(0,5):
(c)        finger = hand ...
(d)        for j in range(0,3):
(e)             bone = finger ...
(f)             boneBase = bone ...
(g)             boneTip = bone ...
(h)             [extract coordinates from base and tip as in step 13]
(i)             lines.append(...)
(j) plt.draw()
(k) [erase all of the lines drawn above as in step 17]
```

Note that the lines are drawn (line 18(j)) only after all of the lines have been added to the plot (line 18(i)). This is indicated by the fact that line 18(j) is aligned with line 18(b). When you run your code now, you should see something like this.

19. Capture a short video of your code in action at this stage. Upload it to YouTube and add it to a new playlist called `2014_UVM_CS228_YourLastName_YourFirstName_Deliverable_2`. Make sure the playlist and the video is set to Public so the grader can view and grade your submission.

20. Let's now stabilize the axes. Add the following lines

```
(a) ax.set_xlim(-1000,1000)
(b) ax.set_ylim(-1000,1000)
(c) ax.set_zlim(0,1000)
```

between lines 10(c) and 10(d). Run your code again. You should now see a small hand moving around in a much larger volume.

21. You will notice that the drawn hand is probably moving differently than your actual hand. To correct this, we will first need to swap the y and z axes. Alter line 15(a) to

    (a) `lines.append(ax.plot([xBase,xTip],[zBase,zTip],[yBase,yTip],'r'))`

    You will now see that the hand moves as your hand does, but from the wrong viewpoint.

22. Let us correct the view by rotating the virtual camera to point at the virtual hand from another position. We can do this by changing the azimuth of the camera by 90 degrees. Add this line

    (a) `ax.view_init(azim=90)`

    just after line 20(c). When you run your code now you should see that the virtual camera is looking at the virtual hand from the same direction that your eyes are looking at your real hand. In other words, your real wrist should be closer to your eyes than your real hand, and it should seem that the wrist of the virtual hand is coming 'out' of the screen and the virtual hand is pointing 'into' the screen.

23. Finally, you should notice that the hand appears 'flipped': when you move your real hand left, the virtual hand moves right, and vice versa. You can correct this by negating the values of the x coordinates: change line 21(a) to

    (a) `lines.append(ax.plot([-xBase,-xTip],[zBase,zTip],[yBase,yTip],'r'))`

    to accomplish this.

24. Now, using the strategy from the previous deliverable, determine the minimum and maximum values obtained for the x, y, and z coordinates attained by any of the finger bones. Copy these six numbers into lines 20(a)-(c). The hand should now be larger, and move within a smaller volume like this.

25. Capture a short video of your system in action, upload it to YouTube, append it to your `Deliverable_2` playlist, and submit the playlist link to BlackBoard by the deadline.