CS228: Human Computer Interaction

Deliverable 2

Description

1. In this deliverable, instead of drawing a point at the tip of the index finger, we are going to draw the whole hand by representing the bones as line segments.

2. To begin, make a copy of Del01.py and call it Del02.py. Make sure to add it to your git repository, commit it, and push to your master branch.

3. Now comment out all of the material between the call to Prepare and Reveal in the infinite loop. When you run your code, you should get an empty pygame window that does not respond to anything from Leap.

4. Uncomment the lines that grab a frame of Leap data, check to see whether a hand is hovering over the device and, if so, calls `Handle_Frame()`. Comment out everything in that function, add a pass statement so you won't get an error message, and run your code again. You should see no change, because you're not doing anything with the grabbed data yet.

5. Now we are going to create a nested loop in which we iterate over each of the five fingers of the observed hand. For each finger, we will iterate over each of its four bones and draw each one. (Recall the bone structure captured by the device as shown in the figure here.) Here's how.

6. Uncomment the lines up to and including the one that stores information about the fingers in a variable called `fingers`. Print this variable right after it is created, and run your code. You should see something printed out when your hand is over the device.

7. Delete the print statement. Now, we are going to iterate over this list, and store each element in the list in a variable called `finger`. Print `finger` immediately after it is assigned. When the loop finishes, place an `exit()` statement. When you run your code now and wave your hand over the device, you should see five print statements and then your program should terminate. This is because it terminates after handling each finger in this first captured frame of data.

8. Replace the print statement with a for loop that iterates over the numbers zero through three inclusive. Use this integer inside of the loop to extract the *b*th bone from `finger`, and store the result in a new variable, `bone`, like this:

   (a) `for finger in ...`

   (b) `    for b in ...`

   (c) `        bone = finger ...  b ...`

Consult the documentation regarding the finger data structure to see how to extract the bth bone from it.

9. Print `bone` immediately after it has been assigned. When you run your code now, you should see $5 \times 4 = 20$ print statements, and then your code should terminate.

10. Let's modularize our code a bit by creating a new function called `Handle_Finger(finger)`. Define it, cut and paste lines 8b and 8c into it, replace the cut lines with a call to the function, pass the function the current `finger` as an argument, and also cut and paste the print statement into it. When you run your code now you should see no change.

11. Let's modularize your code some more. Inside of `Handle_Finger`, replace line 8c with a call to a new function called

    `Handle_Bone(bone)`,

    where the `bone` argument represents the current bone and is created by combining `finger` and `b` as you did on line 8c. The function should contain a single print statement, which prints the current bone. When you run your code now you should see no change. We are now set up to use `Handle_Bone` to extract the coordinates of the current bone and draw it to the pygame window.

12. Now, in `Handle_Bone`, create two variables called `base` and `tip`. These two variables should store the data returned by `prev_joint` and `next_joint` respectively, as described in the documentation. Print these two variables now instead of `bone`. What did you expect to see? What do you actually see? If this looks correct to you, continue to the next instruction.

13. Now, extract the x and y coordinates from both of these data structures in `Handle_Bone`, as you did in Deliverable 01. For now, do not scale them as we did in Deliverable 01. Also, do not update `xMin, yMin, xMax` or `yMax`. This will give you four variables in bone to work with. Print these variables instead of printing the two Leap Vectors. What did you expect to see? What do you actually see?

14. Since you need to do the extraction twice, create a function called `Handle_Vector_From_Leap(v)` that returns the x and y coordinates extracted from the leap motion Vector v. Call this function once on `base` and then again on `tip`, replacing the lines that extract the coordinates which you just added in the previous step. When you run your code, you should see no change compared to the previous step.

15. We will now draw a line that connects points `(xBase,yBase)` and `(xTip,yTip)` together. Each line will represent one bone. To do so, create a new function in pygameWindow called `Draw_Black_Line` which takes these four arguments. Delete the print statement in `Handle_Vector_From_Leap(v)` and call `Draw_Black_Line` from within `Handle_Bone`. with a call to this function. Consult the pygame documentation to see how to draw a line using these coordinates within this function. Finally, remove the `exit()` statement from your code. When you run your code now, you will probably not see anything drawn to the screen. Can you think of the reasons why?

16. We need to scale these values to lie within the pygame window as we did in Deliverable 01. To do, scale the two coordinates inside of `Handle_Vector_From_Leap(v)` as you did previously. To do so, you will need to uncomment `xMin, yMin, xMax and yMax`, as these are required by your scaling function. For now, we will not alter the values of these four values while the code runs, but rather set `xMin` and `yMin` to sufficiently large negative values and `xMax` and `yMax` to sufficiently large positive values so the line segments can be seen are cannot move beyond the window's frame.

17. When this is working, you should see a small hand drawn near the center of the screen, like this. But, you will probably notice that the hand is upside down, and/or moving to the left when you move your actual hand to the right, and so on. Please capture a few seconds of your screen and upload to a new Deliverable 02 YouTube playlist. It doesn't matter 'how' your hand is drawn incorrectly, just that it is clear that it is drawn incorrectly in some way.

18. Please alter the x and y coordinates immediately after you capture them until the virtual hand is drawn 'right side up', and moves in the same direction as your actual hand.

    To get this to work correctly, you might also need to extract the third coordinate out of the Leap Vectors and store it in y, rather than the second coordinate.

    When it's working, the virtual hand should move left (or right) when you move your hand left (or right). Also, instead of the virtual hand moving upward on the screen when you raise your hand, the virtual hand should now move upward when you move your hand away further away from you. Similarly, the virtual hand should move downward on the screen when you pull your hand toward you, rather than when you lower it.

19. We can now alter the values of `xMin, yMin, xMax and yMax` in `Handle_Vector_From_Leap(v)` as we did in Deliverable 01 so that the drawing space is dynamically enlarged when the virtual hand hits the boundary of the window. You should obtain something like this. Please capture a few seconds of your screen, upload to YouTube, and append to your Deliverable 02 playlist.

20. Finally, we are going to add some aesthetic touches to make the virtual hand a little more legible. You'll notice that when line segments cross one another in the drawing window, it can be difficult to see if the virtual hand is matching the pose of your actual hand. To help with this, we will include a visual metaphor: bones closer to the wrist will be drawn with thicker lines than bones that are toward the tips of the fingers.

    Hint: You can do this by using the b variable to set the line width of the line drawn by pygame.

    When you get this working, you should see something like this. Capture a few seconds of your screen, upload to YouTube, and append to your playlist. It is OK if your line widths are not exactly the same as in the sample video, but we should be able to see line widths narrowing toward the tips of the virtual hand's fingers.

21. You have now completed this deliverable. You will notice that there are some functions left over in Del02.py that are no longer used. Delete them. Be sure to add, commit and push to your CS228 repository.

22. Go to BlackBoard and submit your playlist to the Deliverable 02 assignment there.