CS228: Human Computer Interaction

Deliverable 7

Description

In this and the remaining deliverables, you will gradually build a system that teaches a user the first 10 ASL numbers. We will assume that the user does not know ASL and has never used a Leap Motion device before.

In this deliverable, you will devise and implement two functions: the first will help the user start interacting with your system for the first time. The second should help the user move and then keep their hand near the center of the device's field of view.

Instructions

1. Imagine that you are a user who has never used a Leap Motion device before. How might this user fail to interact with the device — and your educational software — correctly? Some obvious ones are that they hover their hand outside its field of view. What might you draw to the screen to help the user discover where the device's limits of view are? What might you draw on the screen to motivate the user to hover their hand over the device in the first place?

2. Make a copy of `Del6.py` and call it `Del7.py`.

3. In `Del7.py`, comment out all of the lines that have to do with the classifier. This should lead to code that just draws the user's hand to the screen.

4. Now, add a visualization to `Del7.py` that the user sees when they start up the program. This visualization should motivate the user to hover their hand over the device. If you were to test this visualization with several users (you're not going to yet), you could time how long it takes for a user to bring their hand into the device's field of view, once the program starts running. You might average these times across several users, and then try to modify your visualization in such a way that you bring this time interval down across successive rounds of programming and user testing.

   Note: For this affordance — and the rest of the affordances you build into your system — you are not allowed to use explicit text messages such as 'move your hand over the device.' Only graphics, animations or short video clips are allowed. To help you develop affordances, imagine that your system is being used by someone who does not speak English.

   Hint #1: Python's `matplotlib` allows you to create and draw to multiple panels within a single figure. (An example can be found here.) So, you can pursue one of two alternatives now: you can draw everything within a single panel, or draw the user's virtual hand in one panel and additional information and guidance in another panel.

   Hint #2: `matplotlib` also allows you to embed images within a panel. (Example here.) Perhaps you might show an image to the user to help them figure out what to do.

Hint #3: Last year, some students migrated their code from matplotlib to pygame. Some students found it easier to embed multiple graphics widgets in pygame than in matplotlib. If you find that you're having a hard time making progress in this deliverable using matplotlib, you are welcome to switch over to pygame or some other graphics-based python package at this point.

5. Create a global variable in your code called `programState`. This variable can take on the value 0 or 1, which indicates which of the two states your system is currently in:

   (a) 0 = the program is waiting to see the user's hand.

   (b) 1 = the user's hand is present.

6. By adding this global variable, you are turning your program into a stateful program: the program does different things depending on which state it is in. At this point, restructure your code as follows:

```
(a) while ( True ):
(b)       if ( programState == 0 ):
(c)           HandleState0()
(d)       elif ( programState == 1 ):
(e)           HandleState1()
```

7. The visualization you just created should only be displayed if `programState==0`. To do this, fill in the `HandleState0` function as follows:

```
(a) def HandleState0():
(b)       DrawImageToHelpUserPutTheirHandOverTheDevice()
(c)       if HandOverDevice():
(d)           programState=1
```

8. Next, create an interactive visualization that starts up whenever a hand is detected above the device. This visualization should be drawn every pass through `HandleState1`. This visualization should motivate the user to move their hand to some point near the center of the device's field of view. This will require you to detect, at each time step: (1) where the user's hand currently is, (2) where the user's hand should be, and (3) a visual cue to the user about which direction they should move in.

9. You should now add in an `if` clause to `HandleState1` that will set `programState` back to 0 if the hand is no longer detected above the Leap Motion device.

10. If the user is able to keep their hand at the origin for a sufficiently long period of time (you decide what that time span should be), provide a visual cue to the user that they have 'succeeded'.

11. Add another possible value to `programState` such that now

    (a) 0 = the program is waiting to see the user's hand.

    (b) 1 = the user's hand is present but not centered.

    (c) 2 = the user's hand is present and centered.

12. If the user's hand becomes uncentered, the visualization you created in step #8 should be displayed. If the user's hand leaves the device's field of view, the visualization you created in step #4 should be displayed. This can be accomplished by adding an `if/else` clause at the end of `HandleState2`.

13. Now, if the user's hand is centered, pick one of the 10 ASL numbers at random, and show that digit to the user. Create some visualizations that help the user to successfully make the corresponding gesture. You may attempt to show an image of the gesture corresponding to this digit in a separate panel (see Hint #2 above). However, many students have found that this causes matplotlib to re-draw the image during each pass through their main loop, thus slowing down their code too much. You may address this challenge by (1) creating a low-resolution version of the image you wish to show; (2) show a virtual hand in the separate panel to demonstrate what the user should do; or (3) come up with another alternative.

14. Finally, detect whether the user has made the correct sign or not: this will require you to uncomment the code related to your kNN classifier. If the user has signed the number correctly, that digit should be output by the classifier for at least 10 frames in a row.

15. Add a new value to `programState`:

    (a) 0 = the program is waiting to see the user's hand.

    (b) 1 = the user's hand is present but not centered.

    (c) 2 = the user's hand is present and centered.

    (d) 3 = the user has correctly signed the current number.

16. When the user correctly signs the current number, a 'success' visualization should be presented for a short time period, a new number should be chosen at random, and `programState` should be set back to 0, 1 or 2 (depending on the user's hand's current position).

17. Shoot a short video demonstrating each of the above visualizations in action. Upload the video to YouTube and submit to BlackBoard.