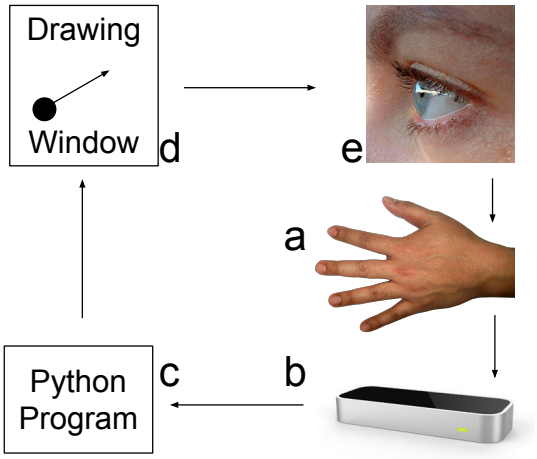

Figure 1: Establishing an interaction between the user, the Leap Motion controller and your Python program. The user moves her hand (**a**); the motion is captured by the controller (**b**); the gesture data is sent to your Python program (**c**); the data is used to update a visualization in the drawing 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. First, you must download, install and run the Leap Motion SDK.

2. A word of caution before proceeding: some of you will encounter installation headaches. This is due to the fact that the version of the `Leap` package we will be using requires Python version 2.7. However, the other package we will be using, `pygame`, tends to work better with Python 3. Details about how to overcome these hurdles are explained below.

Installing Leap Motion SDK

3. The major challenge in this assignment is to get all of the various software components you'll need running on your particular computer platform. To help the instructor and T.A. help with any installation issues that arise, please type in your particular computer platform (e.g. MacOS 10.12.6 or Windows 10) in your row, in column D, in this spreadsheet. Note that this spreadsheet is editable by all of the students in this class, so please enter data here carefully.

4. If you're using Windows or a Mac, download v2.3.1 of the SDK from here.

5. If you're using Linux, you're on your own: we suggest you either find or borrow a Mac or Windows laptop, or install a dual Windows / Linux partition. The instructor and the T.A. can only help those with a Windows or Mac machine.

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

7. If it's working, please enter 'yes' in your row, in column E, in this spreadsheet. Note that this spreadsheet is editable by all of the students in this class, so please enter data here carefully.

8. 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.

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

10. Upload the video to YouTube, and include it in a new YouTube playlist called `2019_UVM_CS228_YourLastName_YourFirstName_Deliverable_1`. Make sure the playlist and the video are set to 'Public' so the grader can view and grade your submission.

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

Installing Python 2.7

12. Now let us switch to Python for a bit. (Python is the only language we will be using in this course.) The Leap Motion SDK is only compatible with the 32-bit version of Python 2.7. So, if you do not have this version of Python on your computer, please google how to install it on your machine and do so.

13. Make sure that you know how to run a Python program using this particular version of Python, even if you have other versions of Python on your machine. On a Mac, you can do so in the Terminal app as shown here.

14. Once you successfully have Python 2.7 up and running, please enter 'yes' in your row, in column F, in this spreadsheet. Note that this spreadsheet is editable by all of the students in this class, so please enter data here carefully.

15. 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.

16. To warm up, let us write a simple 'Hello World' Python program.

Creating a git repository.

17. Before we write our first program, we are going to ensure that all of the code you develop for this course is managed by git, a version control system. This will ensure that if you accidentally delete or otherwise break your code, you can always roll your code back to a working version and proceed forward again. **In this course, the T.A. will not accept the excuse that your computer crashed and you lost all your code.** It is your responsibility to back up your code in git.

18. If you are unfamiliar with git, watching and following along with this tutorial is a good place to get started. Only the skills described in videos 1.1 through 1.6 are required for this course.

19. If you do not already have one, create an account on github.

20. Create an empty repository called `CS228` in github.

21. Clone it to your machine.

22. In your git repository on your machine, using your favorite text editor or IDE (doesn't matter which one), create a file called `HelloWorld.py`.

23. In this file, write
    ```
    print('Hello World!')
    ```
    Save the file and run it. It should print out this string. Add this file to your git repository, commit it, and push it to your master branch. If you are unsure what this means, please consult the git tutorial mentioned above, or another one of your choosing.

24. Let us now create a second Python program called `RealTimeDraw.py`, which will continuously draw a point that moves randomly. (Remember to add, commit, and push it to your github repository.) To do so, we will make use of the pygame Python library.

Installing pygame.

25. Please download and install pygame for Python 2.7. The best way to do so is to use pip in Terminal on a Mac, or in the command prompt on Windows:

    ```
    pip install pygame
    ```

    Make sure you're using the pip associated with Python 2.7 and not pips associated with other versions of Python you have running on your computer. You can usually explicitly request installing the package for Python 2 using

    ```
    pip2 install pygame
    ```

    if you have multiple versions of Python on your machine.

26. If you're on a Mac and you get a 'permission denied' message, you will need to install this as the superuser:

    ```
    sudo pip install pygame
    ```

27. If you're still having problems, trying googling a solution using something like `'pygame python 2.7 pip'`.

28. To ensure that you have successfully downloaded and installed pygame, add one line to `RealTimeDraw.py`:

    (a) `import pygame`

    and run it (remember to run it with Python 2.7).

29. If you get no error messages, it is installed correctly. Please indicate this in your row, in column G, in this spreadsheet.

30. If you get error messages, copy and paste them into google to find a solution.

Real time drawing.

31. You are now ready to do some real time drawing. To do so, we will create an infinite loop inside of your program. Each iteration through the loop will draw something different to the screen. So, add this code below the import statement:

    (a) `while True:`

    (b) `    print('Draw something.')`

32. This program should run forever, printing out line after line until you halt it.

33. We will now create a Pygame window, within which we will do our drawing. To do so, we will create a Python class called `pygameWindow`.

    At this point we will begin using **Object-Oriented Programming**, or **OOP** for short. If you are not familiar with OOP in Python, please consult the first three tutorials starting here. We will only make use of classes, objects, class variables and methods in this course. So, only the first three tutorials are necessary for this course.

34. To create the pygame window class, create a new file called pygameWindow.py. (Remember to add, commit, and push it to your github repository.) Insert an empty constructor into it as follows:

    (a) `class PYGAME_WINDOW:`

    (b) `    def __init__(self):`

    (c) `        pass`

35. Now, back in `RealTimeDraw.py`, import this class by adding this line at the top

    (a) `from pygameWindow import PYGAME_WINDOW`

36. Now, create an instance of this class called `pygameWindow` by calling the class's constructor near the top of your program.

37. Comment out the infinite loop for a moment, and add a line that prints `pygameWindow`. You should see something like this:

    `<pygameWindow.PYGAME_WINDOW instance at 0x292c3a0>`

38. Now let's actually create the window itself. Remove the line that imports pygame into `RealTimeDraw.py` and add it to the start of `pygameWindow.py`. (We are going to hide all pygame-specific calls inside of `pygameWindow.py`.)

39. Remove the pass statement from the constructor and add these instead:

    (a) `pygame.init()`

    (b) `self.screen = pygame.display.set_mode((300,100))`

    Consult the pygame documentation to see what the numbers 300 and 100 represent.

40. Now create two empty class methods called `Prepare` and `Reveal`. (An empty method just has one line inside of it: `pass`.)

41. Back in `RealTimeDraw.py`, uncomment the infinite loop, and replace the print statement inside of it with these three lines:

    (a) `pygameWindow.Prepare()`

    (b) `pass`

    (c) `pygameWindow.Reveal()`

42. This establishes the main foundation for the code base you will be developing over the coming semester: at the beginning of each iteration through the loop, the pygame window will be wiped clean and prepared for drawing; then, something will be drawn to the window (the pass statement is currently a placeholder for this); finally, the drawn material will be revealed to the user. This will continue until the user closes the window.

43. Run your code now, and you should see a black-filled window appear. You may also see some warning messages, and only the top-left quadrant of the window may be filled with black, as shown here. This is fine. You also may not be able to click on the exit button on the window itself, but halt your program to close the window. This is also fine.

44. If you see a gray rather than black screen, you may need to add these line

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        self.done = True
```

to the `Prepare` method to make it appear correctly.

45. Now, change the background of the window from black to white by replacing the pass statement in the `Prepare` method with `self.screen.fill(...)`. This should prepare the screen for drawing by filling it with a white background. Consult the pygame documentation to see what the ellipsis should be replaced with to make this happen.

46. When you run your code now, you should still see a black window. This is because although a white background was drawn to the window, it was not revealed to the user. To do so, replace the pass statement in the `Reveal` class method with

    (a) `pygame.display.update()`

    You should now see a white window when you run your code.

47. You are currently drawing a 300-pixel wide and 100-pixel deep window. You may want to alter the size of your window often during the development of your code. So, create a new file called `constants.py` (remember to add, commit, and push it to your github repository), create two variables inside called `pygameWindowWidth` and `pygameWindowDepth`, assign the two numbers to them, and replace these two numbers in `pygameWindow.py` with calls to these two variables. Try changing these values in `constants.py` to see how the window changes.

48. Now you will draw a black circle to the screen every time through the infinite loop. Add a new method to `pygameWindow` called `Draw_Black_Circle(x,y)`. The two coordinates indicate where in the window to draw the circle. Consult the pygame documentation to determine how to draw a circle, and include the single line you find into your new method. You will note that drawing a circle requires a 'surface' as one of the arguments in the call: `self.screen` is a surface.

49. Now replace the `pass` statement in the infinite loop in `RealTimeDraw.py` with a call to this method. Pass in coordinate values that draw the circle at the center of the window. When you run your code now, you should see a black circle placed at the center of the window. If you do not, you may need to add the code mentioned in step 44.

50. Change the size of the circle, and increase the size and width of your window until it looks more or less like this, with the exception that your circle doesn't move. Don't worry about getting to look exactly like the video: just that the window is square-shaped, your circle is clearly a circle, and there is plenty of window space for the circle to move around in, once we get it moving as described below.

51. Now let's get the circle moving. Create two variables x and y at the top of `RealTimeDraw.py`. These will encode the current position of the circle. Set their initial values to place the circle at the center of the window. Replace the numbers with these variables in your method call that draws the circle inside of the infinite loop. When you run your code now, you should see no change, because we are not yet changing the position of the circle.

52. Add a function to `RealTimeDraw.py` called `Perturb_Circle_Position`. Inside this function, we are going to simulate the rolling of a four-sided die. If the number 1 comes up, we are going to decrease the circle's horizontal position by one; if 2, then increase its horizontal position by one; if 3, decrease its height by one; if 4, increase its height by one.

53. Thus, we will need to tell the function that there are two global variables we wish to change. We can do so by adding this line to the function:

    ```
    global x, y
    ```

54. Now, import the `random` Python library into this program, call its `randint` function to 'roll' a four-sided die, and store the result in a variable called `fourSidedDieRoll`.

55. Add an `if-elif-else` set of statements that modifies x or y according to the value of the roll.

56. When you run your code now, you should see your circle moving like this. Run your code several times and watch the movement of the circle. If the circle is indeed moving randomly, it should never end up in the same quadrant every time, or always move downward, or always move upward, and so on. If it does, there is likely to be an error in your `Perturb_Circle_Position` function.

57. If your circle is moving too slowly for your taste, create a new variable in your constants file called `circleVelocity`, set it to some integer value, and increase or decrease x or y by that amount instead.

Moving the circle with your finger.

58. 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.

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

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

61. Move the directory that contains your CS228 git repository into the `LeapDeveloperKit.../LeapSDK/lib` directory. Your program will need to know where the Leap Motion libraries are. You are going to tell your program that the libraries are in the parent directory.

62. But first, create a copy of RealTimeDraw.py called Del01.py. We are now going to modify Del01.py to enable finger-directed movement of the dot. Make sure to add this new file to your repository, commit the change, and push it to github.

63. Comment out every line in Del01.py and add these lines at the top:

    (a) `import sys`

    (b) `sys.path.insert(0, '..')`

    (c) `import Leap`

    Line (b) tells your program that, if it can't find the libraries it needs along its usual paths, it should also look in the parent directory. Now, when it looks for the Leap library requested on line (c), it will find them in the parent directory (which is `LeapDeveloperKit.../LeapSDK/lib`).

64. If you run your code now, it should do nothing. If you get error messages at this point, it is likely because you are using an incompatible version of Python, or your program cannot find the Leap libraries.

    If you're on a Mac...

    (a) you will need the 32-bit version of Python2.7. You can download it here, and then you should be able to run

    `python Del01.py`

    from the Terminal.

    (b) This might also require you to install pygame for this new installation of Python. You can do so with

    `pip install pygame`

    (c) If this gives you a 'permission denied' message, you might need to install pygame as the super user:

    `sudo pip install pygame`

    If you have a Windows machine...

    (a) Copy all files in lib/x86 into CS228 directory.

    (b) Open the command prompt.

    (c) Navigate into the lib/x86 directory.

    (d) Type `python Del01.py`

65. Once your program runs without errors, put this line

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

    just before the commented-out infinite loop. 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.

66. Now, uncomment the infinite loop, but leave all the calls within the loop commented. Put just this line inside the loop:

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

    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.

67. Now print the frame variable immediately after it is captured. If you see `Invalid Frame` printed continuously, your Leap device is either not plugged in, or your computer cannot access data from the Leap device. Make sure your Leap device is working by running the Visualizer application again. Also, it sometimes takes up to a minute or two after plugging in the device for your computer to recognize it. When your computer is able to grab frames from the device, you should see something like `Frame Id:2167298` printed over and over again.

68. Delete the print statement.

69. 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 66a:

    (a) `    if ( ...  > 0):`

    (b) `        print "hand detected."`

    You will have to write some code to replace the ellipsis (the dots) on line (a). 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.)

70. Now, replace line 69b with a call to a new function, which takes as its only argument the frame of data captured from the device:

(a) `Handle_Frame(frame)`

Create this function and have it print the frame of data whenever it is called. Run your code to make sure that this print statement is only called when there is a hand above the device.

71. Let's now extract information about the first hand — frame.hands[0] — whenever a hand is above the device by replacing the print statement in the function with these lines:

    (a) `hand = frame.hands[0]`

    (b) `print(hand)`

    Now, you should see that a hand ID message is printed out whenever you hover your hand over the device. Here is information about this data structure.

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

    (a) `fingers = hand.fingers`

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

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

    (d) `print(indexFinger)`

    Refer here and here to figure out how to fill in the ellipses on lines 72b and 72c. When it's working, you should see

    `Finger Id:1771`

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

73. 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 phalanx: the bone in the tip of your index finger. To do so, delete line 72d and replace it with

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

    (b) `print(distalPhalanx)`

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

74. 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) `distalPhalanx = indexFinger.bone(...)`

    (b) `tip = distalPhalanx.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.

75. 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. Uncomment all of the lines related to pygame in your code, except the one that calls `Draw_Black_Dot`. Make sure the frame grab and conditional statement in the infinite loop is sandwiched between the uncommented `Prepare` and `Reveal` calls. When you run your code now, you should the pygame window, but no dot, since we're not drawing it yet. You should however see the positions of the tip of your index finger still being printed.

76. Uncomment the creation of the global x and y variables at the top of your code. Inside of `Handle_Frame`, store the first element from the tip list in x, and the second element in the list in y. Make sure that this function knows that x and y are global variables and not local variables to be created when these two assignments occur.

77. Just after the conditional statement in the infinite loop, call the pygameWindow method that draws your circle, and supply it with x and y. Your program will crash, complaining that it requires positions to be integers. So, when storing values into x and y in `Handle_Frame`, convert them to integers first.

78. When you run your code now, you should see the circle moving in response to your finger moving over the device. However, you'll note that the motion often takes the circle off the screen, and the circle does not always move in the same direction as your finger.

79. We will handle the first of these two issues first.

80. 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 loop, 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`

81. Now, add these lines at the end of `Handle_Frame`:

    (a) `if ( x < xMin ):`

    (b) `        xMin = x`

    (c) `if ( x > xMax ):`

    (d) `        xMax = x`

Make sure that this function knows xMin and xMax are global variables.

82. Add similar lines for `y`.

83. Print these four new variables at the end of `Handle_Frame`, run your code, and wave your hand over the device. By paying attention to the position of your finger and how the values change, you should be able to gain an intuition for this new part of your code.

84. Before drawing the dot, we have to convert the values of x and y. This is because they currently lie in a window whose size is defined by the device, but we need them to lie in the pygame window, which has a different size. This image summarizes what we need to do.

85. So, create a function that takes five arguments: A value (argument 1) that lies within a range defined by arguments 2 and 3 should be scaled such that it now lies within the new range defined by arguments 4 and 5. This function should return the scaled value.

86. Now, call this function twice, just after `Handle_Frame` is called, like this:

    (a) `pygameX = Scale...(x,...)`

    (b) `pygameY = Scale...(y,...)`

87. Print these two new variables immediately after they are created, run your code, and watch how these numbers change. Do they lie within the range of values allowed in your pygame window? (If other variables are also being printed out, delete them.) You will notice that there is an issue at the beginning, when `xMin == xMax` and `yMin == yMax`. You will have to add a conditional statement in `Scale` to handle these cases.

88. Now, just after these new variables are created, call `Draw_Black_Dot`, but supply it with these two new values instead. If you've scaled the values correctly, the dot should never leave the window. (You may need to declare these new variables as global variables for this to work.)

89. Depending on how you converted the coordinates, you may notice that when you move your finger upward, the dot moves downward, and vice versa. This is because, in a pygame window, that increasing y values descend from the top of the window toward the bottom (consult this image again). This is also why we defined a variable called `pygameWindowDepth` rather than `pygameWindowHeight`.

90. There are many ways to invert the y value so that the the circle correctly tracks the height of your finger. Please formulate one and implement it.

91. 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`.

92. 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 10.

93. 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 using Terminal (on Macs) or Command Prompt (on Windows).

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 RealTimeDraw.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 `RealTimeDraw.py`