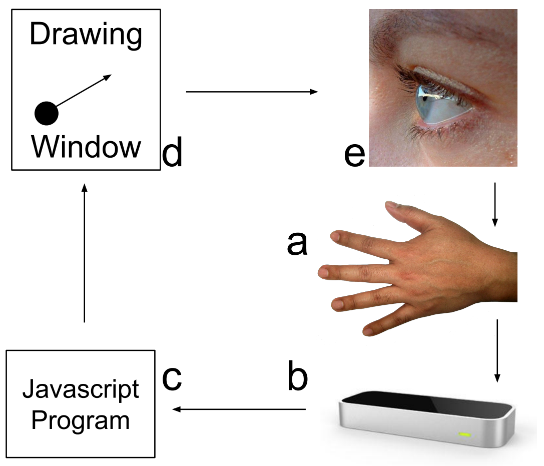

Figure 1: Establishing an interaction between the user, the Leap Motion controller and your Javascript program. The user moves her hand (**a**); the motion is captured by the controller (**b**); the gesture data is sent to your Javascript program (**c**); the data is used to update a visualization in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

# CS228: Human Computer Interaction

## Deliverable 1

In this first deliverable, you will establish a connection between a Javascript 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 Javascript. When you're successful, you should see this randomly-moving dot.
3. Finally, you'll capture hand gesture data from the controller and use it to move the dot 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.

### Receiving Leap Motion

5. At this point, you should have received a Leap Motion device from the instructor. If you have, please type 'yes' in column G next to your name here.
6. If you have not received it yet, please contact the instructor immediately at `jbongard@uvm.edu`.

7. Please remember to keep both the Leap box and the USPS box. You will be returning the device by mail to the instructor at the end of the course (or earlier if you drop this course). Instructions for how to return it will be provided at the end of the course. If you drop this course before then, please contact the instructor for further instructions.

## Installing Leap Motion

8. ~~Now, you must download, install and run the Leap Motion SDK. Depending on your computer platform, some of you may experience some issues with installation. To help the T.A. help you with installation, please enter the type of computer platform you have in column J here.~~

9. ~~Download V2 of the SDK from here under the 'V2' section. Follow the instructions for how to install the software on your platform.~~

   ~~Note: If you are a Windows 10 user, you may experience installation issues. If you do, try this hotfix first. If that fails, please contact the T.A. If you have a platform other than Windows 10 and also experience installation issues, please contact the T.A.~~

10. ~~Plug in your controller and run the Visualizer as shown here (ignore everything after 0m42s). It may take up to a minute for your computer to recognize the device after it is plugged in, so give it some time. You may see this kind of Visualization instead. Either is fine; as long as you can see something being drawn to the screen that corresponds to the motion of your hand.~~

11. ~~When you see your own hand on the screen, enter 'yes' for your name in column H in this spreadsheet.~~

12. ~~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.~~

13. ~~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.~~

14. ~~Upload the video to YouTube, and include it in a new YouTube playlist called `2020_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.~~

## Getting started with Javascript

15. Now let us switch to Javascript for a bit. (Javascript is the only language we will be using in this course.)

16. If you have never coded in Javascript before, or you're a bit rusty, you might consider working through Codecademy's online Javascript tutorial before proceeding. For this course, we will expect you're familiar with the concepts from lessons 1 through 6 inclusive. However,

~~you do not need to know them all now. You can simply learn them, via the online tutorial, when you come across these concepts in these programming assignments.~~

## 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.~~

## Creating a Hello World program.

~~22. In your git repository on your machine, using your favorite text editor or IDE (doesn't matter which one), create a file called `Del01.html`.~~

~~23. Inside this file, we're going to create an empty web page. Insert this text into your html file:~~

```
~~<html>~~
    ~~<head>~~
    ~~</head>~~
    ~~<body>~~
        ~~<main>~~
        ~~</main>~~
    ~~</body>~~
~~</html>~~
```

~~This text creates an empty HTML head and an empty HTML body.~~

~~24. Open this file in your favorite browser. You should see a blank page.~~

~~25. Now, let's commit this change to your git repository. If you're a Windows user, open the command prompt. If you're a Mac user, open the Terminal.~~

~~26. Navigate to the directory where you cloned your git repository.~~

27. ~~Add the changes you just made to your html file by typing~~

    ~~`git add *.`~~

28. ~~Now add a comment to remind yourself, later on, what changes were made in this particular commit. It's a good idea to include the assignment number and step number in your commit message. This way, if you ever get stuck, you can always roll back your code to the last step at which your code was still working. So, for instance, your commit message might be something like~~

    ~~`git commit -m "Just got deliverable 1, step 28 working"`~~

29. ~~In this file, write~~

    ~~`Hello World!`~~

    ~~between `<main>` and `</main>`. When you open this file in a browser now (or, if you the page is still open in your browser and you refresh the web page), you should see that message printed in your web page.~~

30. ~~Add this change to your git repo and commit it as described in steps 27 and 28 above. From now on, you will not be prompted to add and commit changes to your repository: instead, it will be assumed that you will get into the habit of doing so after each step in each programming assignment. You can see the list of your commits by navigating to~~

    ~~`github.com/YourUsername/CS228/commits`~~

    ~~You can click on any commit message to see the changes you made to your code during the implementation of that step.~~

## Including the Leap Motion library.

31. ~~We are now going to include the Leap Motion Javascript library by putting this line~~

    ~~`<script src="http://js.leapmotion.com/leap-0.6.3.min.js"></script>`~~

    ~~between `<head>` and `</head>`. When you open this page in your browser, you should see no change, because we're not using this library yet.~~

32. ~~Now, create a new, empty file in your repository called `leapInfiniteLoop.js`. Make sure this file is added to your repository.~~

33. ~~This file will include Javascript code that, as the name implies, runs an infinite loop. During each pass through this loop, one frame of data will be collected from the device, and your code may manipulate and/or display that data however you see fit.~~

34. ~~Include this line~~

    ~~`var controllerOptions = {};`~~

    ~~at the top of your new file and these lines~~

    ~~`Leap.loop(controllerOptions, function(frame)`~~

    ~~`{`~~

~~}~~

~~);~~

~~after it. Clearly, the loop at the moment is empty. If you open `Del01.html` in a browser, nothing should have changed from the previous step: this is because your web page hasn't included this Javascript code yet.~~

35. ~~Let's do that now. Include this line~~

    ~~`<script src="leapInfiniteLoop.js"></script>`~~

    ~~in your html file, immediately after the point where you include the Leap Motion Javascript library. If you open your html file in a browser, you still should not see any change: that's because the infinite loop doesn't do anything.~~

36. ~~Let's make sure the infinite loop is working by defining a variable called $i$ and setting it to zero just before the loop begins in `leapInfiniteLoop.js`.~~

37. ~~Now, include this line~~

    ~~`console.log(i)`~~

    ~~between the curly braces of the infinite loop to continuously print the value of the variable to the console. As noted here, this data will not be seen by a user, but instead logged for debugging purposes: specifically, here, to test whether our infinite loop is working.~~

38. ~~Let's see this in action now. Open your html file in a browser, and open the console in the browser. (There are different ways to do this, depending on your browser. If you don't know how, google 'open console' and the name of your browser.) Once the console is open, refresh the page. You should see something continuously updating in the console window, like this in Chrome.~~

39. ~~Let's practice debugging our Javascript code in the browser console: if you're not used to this, this will be extremely useful going forward. In your javascript code, increment the value of your variable by one immediately after it's printed to the console. (Make sure the increment is within the infinite loop.) Save the file, open the html file in the browser, open the console, and refresh the page. You should see numbers rapidly increasing in value. (Also: are you remembering to add and commit to your repository after each step?)~~

40. ~~What happens if you refresh the page as the numbers are increasing? You should see something that convinces you that your Javascript code is starting over again.~~

41. ~~Remember that we're using the Leap library for the infinite loop. This means that, if you unplug your leap device and try running your code again, the loop won't work. Just remember then to keep the device plugged into your computer while you work on these assignments.~~

## Drawing with P5.

42. ~~For the moment, the data collected from the Leap controller during each pass through the infinite loop is thrown away. Shortly, we are going to start using that data to draw stuff to the screen. But first, let's get used to drawing. To do so, we'll be using the P5 library.~~

~~43. Include the P5 library by placing this line~~

~~`<script src="https://cdn.jsdelivr.net/npm/p5@1.1.9/lib/p5.js"></script>`~~

~~within your HTML file, where you included the other libraries so far.~~

~~44. Now create a new Javascript file called `prepareToDraw.js`. Remember to add it to your repository as explained in step #32.~~

~~45. Add these lines to this new file to ensure that a drawing canvas is 'spread' over the entire browser window:~~

~~`function setup() {`~~

~~`createCanvas(window.innerWidth,window.innerHeight);`~~

~~`}`~~

~~Note that this is the first time you've created a Javascript function for these assignments.~~

~~46. Now let's include this script by adding it to the `head` block in your HTML file in the same way you included `leapInfiniteLoop.js`. Again, when you open the file in a browser you still shouldn't see any change, because we're not drawing anything to the canvas yet.~~

~~47. Let's do so now. Just before entering the loop in `leapInfiniteLoop.js`, define two variables, $x$ and $y$. Set $x$ to half of the browser window's width, and $y$ to half of the window's height. (There is a strong hint about how to do so a few steps back.)~~

~~48. Call P5's circle function inside the infinite loop, and supply $x$ and $y$ as the circle's position. Give the circle a radius of 50. When you open your file in the browser now, you should see a circle in the center of the browser window.~~

~~49. Let's move the circle around by modifying its position, slightly and randomly, during every pass through the loop. To start, create a random integer between -1 and +1, inclusive, inside the loop. Then, add this variable to $x$ inside the loop. This should result in the circle jigging and jagging randomly to the left and right.~~

~~50. You should see the circle leaving a black trail behind it, because the previously drawn circles are not erased. Let's erase them by adding~~

~~`clear();`~~

~~right at the entry to (but still inside) the infinite loop. You should see the circle moving now and not leaving a trail. (You can also remove the definition and updating of the $i$ variable now, as well as the printing of it to the console, because the movement of the circle is proof that the infinite loop is working.)~~

51~~. Create a new random integer in $[-1, 1]$ inside the loop and add it to $y$. You should now see the circle moving horizontally and vertically.~~

~~52. Capture a short video of the randomly-moving dot, upload it to YouTube, and append it to the YouTube playlist you created in step 14.~~

## Grabbing Leap Data.

~~53. Now let's collect some data from the Leap controller and use it to set the position of the circle. As mentioned above, each pass through the infinite loop grabs a frame of data from the device: the frame contains all information about the hands (if any) that are currently over the device. Let's investigate a frame by commenting out every line inside the infinite loop, and adding this line~~

~~`console.log(frame);`~~

~~to print out each frame as it is captured. Save your code, open the file in the browser, and open the console. Refresh the page, and then wave your hand in and out of the device's field of view. You should see a new line of information printed continuously. In some browsers, you might see an ellipsis (three dots) at the end of a line, indicating that, if you click on it, the rest of the information will be shown.~~

~~54. One datum stored in a frame is how many hands are over the device. This information is stored in frame.hands. Modify the log statement in your loop to print this instead. When you view the console now, you'll see that this is an array than changes in length, depending on how many hands are over the device.~~

~~55. Now let's print this array only if one hand is over the device. To do so you'll have to include an if statement in the loop, and test the length of the array. Run your code now: you should *not* see the array printed when zero or both of your hands are over the device.~~

~~56. Modify the log statement to print just the first element in the hands array now, not the entire array. What do you see when you run your code now?~~

~~57. Store this element in variable called `hand`, and just log this variable now instead.~~

58. ~~`Hand` in turn contains another array, which stores information about the five fingers in that seen hand.~~

~~59. Extract that array from `hand` and store it in a variable called `fingers`. Print this variable to the console. Now what do you see in the console? Why is the length of this array not surprising?~~

~~60. As you're probably realized, the frame data structure contains a nested set of variables that loosely correspond to the anatomy of the human hand: most humans have two hands, each hand has five fingers, and each finger has a series of bones. So, let's include a for loop now that iterates over every element in `fingers`, and logs each of those elements to the console inside the for loop. (It doesn't matter which of several methods you use to iterate over the array.) What do you see in the console now?~~

~~61. Each line is now printing out information about `finger`, five of which are stored in the array `fingers`. We're going to control the position of the circle using just your index finger, so we need to extract just that element from `fingers`. Have a look at the attributes listed for `finger` again. One of them can be used to determine whether the current finger is the index finger. So, inside the for loop, include an if statement that will only print information about the current finger if it is the index finger. Write information to the console to convince yourself that you're indeed only printing information about the index finger.~~

62. ~~Let's pause at this point, because our code is getting messy: we have an if statement inside of a for loop, which in turn is inside of an if statement, which in turn is inside of an infinite loop. Take all code inside the infinite loop and move it to a function called `HandleFrame`. Supply `frame` as the only argument to this function. Back inside the loop, include just one line: a call to this function. Run your code, and observe the console to make sure your code's function hasn't been changed: we've just made it more readable. By the end of this course, your code base will likely grow to several hundreds or thousands of lines. So, it will be responsibility to perform this~~ refactoring ~~at regular intervals to keep your code well organized. If you don't, it will become nearly impossible to make changes to your code toward the end of the course, because you won't know what effect a change will have.~~

63. ~~Let's continue refactoring. Leave the outer if statement and the creation of the `hand` variable in `HandleFrame`, but move all the code inside that statement into a new function called `HandleHand`, and pass in `hand` as the only argument. Check your code's function hasn't been altered.~~

64. ~~Let's do this a final time. Create a function `HandleFinger`, cut and paste the relevant code into it, pass it a single finger as an argument, and ensure your code's behavior hasn't been affected by this change.~~

65. ~~Now, let's grab `tipPosition` from the index finger: we're going to use the data stored inside to set the position of the circle. Log the information inside `tipPosition` to the console. What do expect to see? What did you see?~~

## Drawing based on Leap data.

66. ~~Uncomment the `clear` statement inside the infinite loop, and make sure it's called before you call `HandleFrame`.~~

67. ~~Store the three elements of `tipPosition` in variables $x$, $y$, and $z$ respectively.~~

68. ~~Uncomment and move the `circle` call to just after these variables are created, and pass two of these three variables in as the position of the circle. Run your code now. Do you see the circle moving in response to the movement of the tip of your index finger? What happens if you hold your index finger still but wiggle your other fingers? What happens if you bring both hands over the device? What other tests can you think to perform with your new code base? A good coder is a curious coder.~~

69. ~~Throughout this course, we are going to design visualizations that help the user learn our system. The better our visualizations, the less explanation we need to provide to them. Let's tackle our first visualization improvement. Obviously, the user can move their hand through 3D space, but we're projecting that down onto a 2D surface (the screen). Which of the two variables, of the three available, should you use here? You will have to apply an algebraic operation to one of these variables, so that when the user moves their hand left then right, the circle moves left then right, and when the user moves their finger up then down, the circle moves up then down. Add that operation now. The circle's movement should now match that of the user's finger better now.~~

~~HINT: You'll want the circle to rise up from the bottom of the canvas when the user raises their finger upward, and the position of the bottom of the screen is stored in `window.innerHeight`.~~

70. ~~Your circle is also probably not centered well, or may move off the canvas sometimes. To keep the circle always within the canvas, we will need to convert the raw finger position coordinates to canvas-based coordinates. As a first step, we will need to know what the range of values the raw finger positions can take. To do so, create four global variables in your javascript code — `rawXMin`, `rawXMax`, `rawYMin`, and `rawYMax` — and set the two minimum variables to very positive values, and set the two maximum variables to very low values. Run your code: since we're not using these variables yet, they should not have affected your code.~~

71. ~~Now, whenever finger tip data is available, add four if statements. The first one checks whether the current horizontal position of the finger tip is less than the value stored in `rawXMin`; if it is, that value is stored in `rawXMin`. The logic is similar for the other three if statements. Create them now, and log all four variables to the console whenever this part of the code is reached. How do you expect these variables to change as your code runs? Does what you see conform to that expectation? If not, go back and fix your code. Go to the next step when it's doing what you expect it to do.~~

72. ~~Now we're ready to translate the finger positions into canvas positions. At any point, we know the $x$ position of the finger lies between `rawXMin` and `rawXMax`, but we want to scale it to lie between 0 and `window.innerWidth`. Google how to scale a value from one range to another in code, and add that to your code when modify $x$. Make sure you do so just before you draw the circle. Run your code: you should see the circle remaining within the horizontal boundaries of the canvas.~~

73. ~~Perform the same scaling for $y$. Run your code, and check now that your circle never leaves the boundary of the canvas.~~

74. ~~Capture a short video now of *you* moving the dot, upload it to YouTube, and append it to the YouTube playlist you created in step 14. Make sure we can see the device, your screen, and your hand in the video.~~

75. ~~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).~~