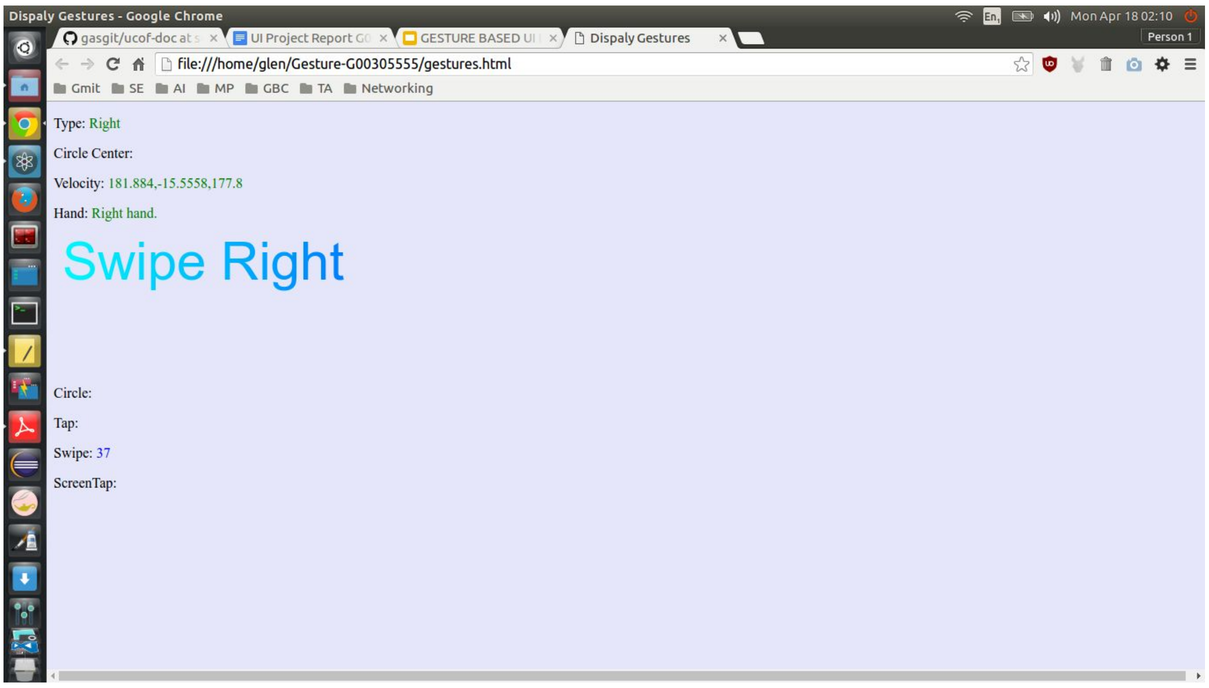
Gesture Based User Interface

Purpose

To experiment with the Leap Motion device. To try out the different gestures available and how it they could be used. Using the Leap Motion hardware, Javascript API and SDK to create HTML pages to demo different gestures. The Leap hardware uses optical and infrared sensors to map hands, fingers and gestures.

Gestures

The first page (gestures.html) in my demo project use the Leap Loop and frames to identify the gestures.I have created some tags to print out sample information as well as a canvas element to print the gesture itself. Other gestures can include mapping of the finger tips as pointables and recognizing there is a tool present. Creating a leap controller and leap loop the device is collecting data on each frame. Enabling gestures it possible to capture the gestures by name e.g 'swipe', 'circle' as well as capture finger positions. The controller can also identify which hand is in the frame.



Type: Right

Circle Center:

Velocity: 181.884,-15.5558,177.8

Hand: Right hand.

Swipe Right

Circle:

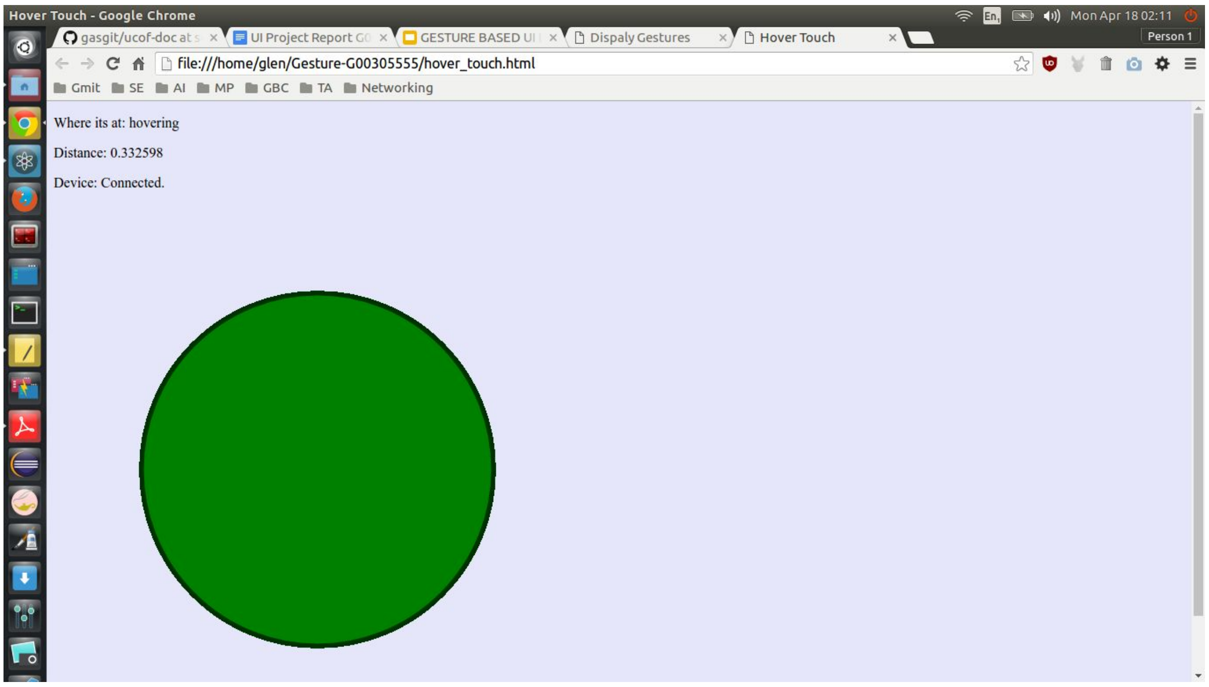
Tap:

Swipe: 37

ScreenTap:

Touch Zone

Touch zone simulates a touching gesture. The range is between 1 and -1 where a reading of 1 is not on the zone, between 1 and 0 hovering as ready to touch and -1 is touching. I created tags to print where it's at, the point between the zones and a simple 2D canvas changing color from blue, green to red to demonstrate each zone. The gesture to use here is pointing toward the screen slowly to enter each zone.


Hover Touch - Google Chrome
file:///home/glen/Gesture-G00305555/hover_touch.html

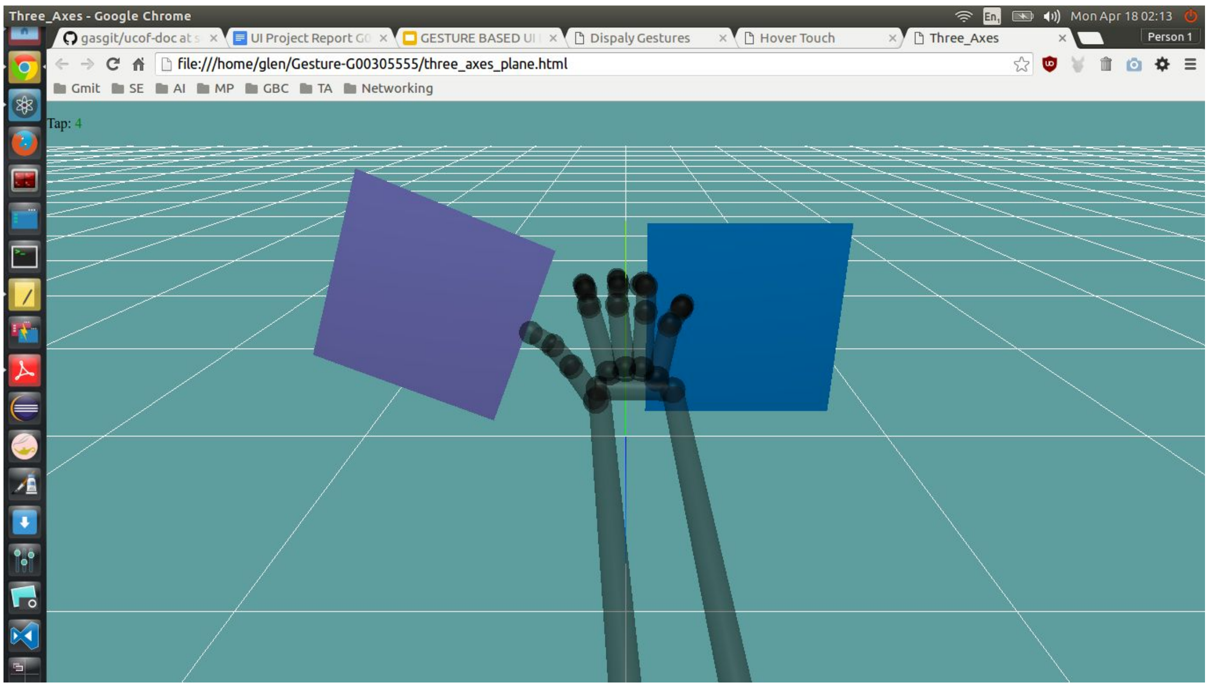
Where its at: hovering

Distance: 0.332598

Device: Connected.

Three Scene

Scene set up using Three js. Added to the view are, scene, camera, renderer, axis helper, grid helper, bonehand widget. Using Three mesh, material and plane with orbitcontrols, leap plugins and widgets the scene open with 2 planes. One plane simulates a button with two modes lock and release. Color changes to show state and when released a new plane is created to interact with. Gestures are then used to flip, spin and delete using colors again to demonstrate. Tapping count reaching 20 will delete all plane objects created from the scene.


Three_Axes - Google Chrome
file:///home/glen/Gesture-G00305555/three_axes_plane.html

Tap: 4

Gestures Identified and Used

The leap loop, frames and controller has capability to map fingertips, fingers, bones, and hands. Below is a list of gestures used in this project to demo, others are possible including pinch and grab.

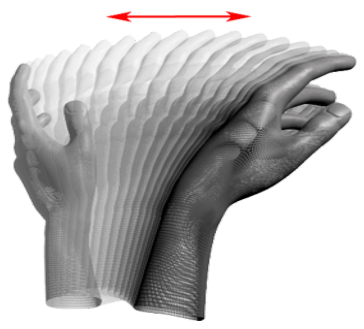
Key Tap - downward motion simulates key pressing. The motion should be short so not to mix up with the down swipe. All fingers are captured by key tap.

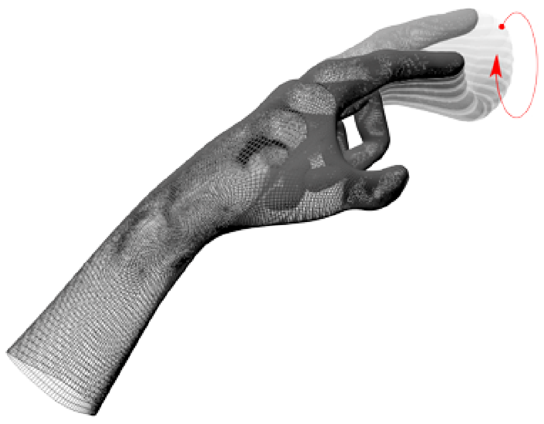
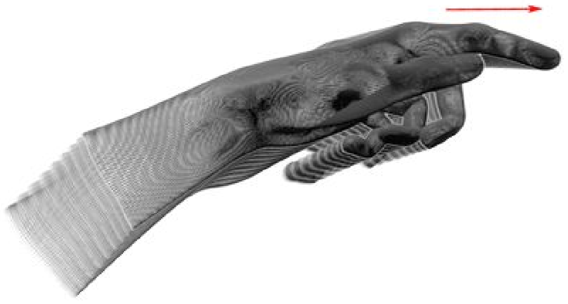
Screen Tap - point directly toward virtual screen direction and back. The level should stay consistent and short for best results. This gesture is definitely more pronounced that the touch zone point.

Swipe - left, right, up and down directions. Swipes are the easiest gestures to get right out of all. The device captures in a continuous motion and recognises start and end of gesture. The swipe can be full length of interaction area or as short as crossing the length of the device itself. It needs to be above 5 cm or so.

Circle - clockwise, anticlockwise. The full circle gestures are captured easily enough. Circle will work as long as it is within range of the device zone. The center of the circle can be captured a well as simulating the circumference.

Incomplete circle - clockwise and anticlockwise are difficult to simulate when trying with smaller circumference.

	
Swipe	Key Tap

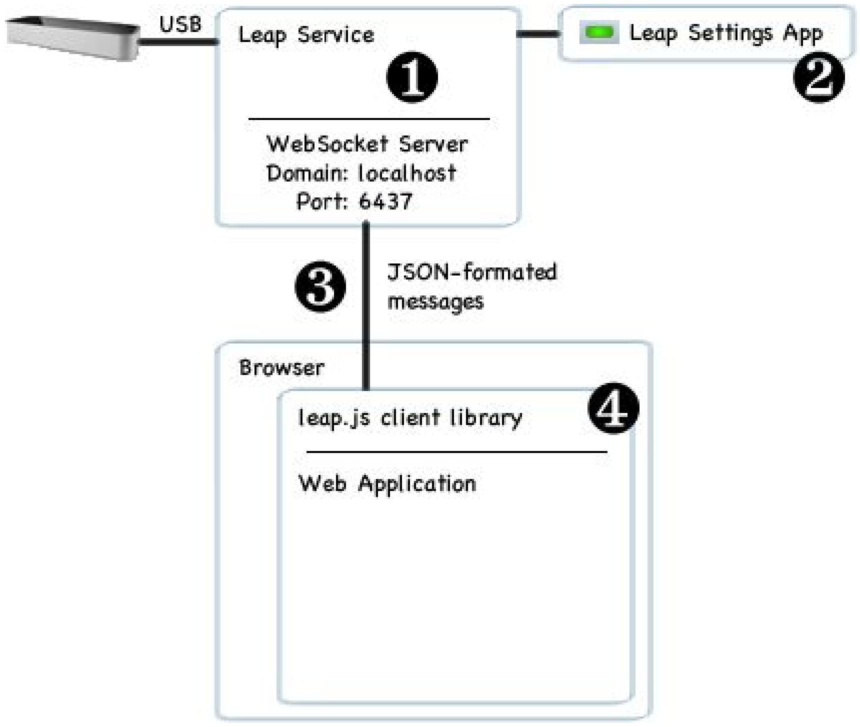
	
Circle	Screen Tap

Architecture

Once the leap SDK is installed there is a service running in the background enabling plug and play style connection. Depending on which API or environment the developer chooses the service and settings panel are accessible for calibration, testing and more.

For this assignment I choose to use the JavaScript libraries and the websocket API.

Websocket API.



This api allows the developers to create web applications within the browser using HTML, JavaScript and the Leap SDK. This also enables the hosting of the application online and once a device and service are present it easy to test the demo. To capture gestures on a 2d web page the sdk only requires the leap.x.x.x.js files included.
This can be included or obtained from https://js.leapmotion.com/leap-0.6.4.js as long as you online to use.

Using THREE.js to create a scene requires a few more javascript libraries included. Leap, leap-widgets, leap-plugins, three-orbitcontrols and three.js. The three.js and OrbitControls are local and the leap js files are sourced online.

```
<script src="js/three.js"></script>
<script src="https://js.leapmotion.com/leap-0.6.4.js"></script>
<script src="https://js.leapmotion.com/leap-plugins-0.1.11.js"></script>
<script src="https://js.leapmotion.com/leap-widgets-0.1.0.js"></script>
<script src="js/OrbitControls.js"></script>
```

Conclusions

The leap device range is pretty restrictive and outside of slideshows and scrolling web page types of scenario on desktop I can see that it is used for cool enough menu type interactions. Even applications where using boneHands on the desktop type environment don't spring to mind easily. Perhaps in retail outlets or other automated experiences when we don't want to touch the current devices.

The likes of the 3D scene I have used here the accuracy of using a mouse or touchpad are not under any threat yet. On the other hand using it in VR environment the types of gesture like pinching and grabbing would have more effect and utilized better.

Thing is we have all experienced and observed another person trying to get to grips with a trackpad for the first time. Mouse to trackpad, trackpad to touch screen and logically we are heading in the direction of gesture. When context and gesture are combined the experience will be fantastic.