

Sign Language Interpreter

Using Leap Motion Sensor

Sanket R Bhimani

June 22, 2016

Contents

1	Tutorial Name:	3
2	Prerequisites:	4
3	Requirement	5
4	Theory and Description	6
5	Implementation:	23
6	Exercise	25
7	References	26

1 Tutorial Name:

Sign language Interpreter Using Leap Motion Sensor

Objective:

Objective behind this project is to help deaf and dumb people to communicate with normal people. So this system will try to interpreter their sign language into natural voice so normal people can understand what that person want to say.

Short Intro:

Here we are using leap sensor for detecting gesture and then we will convert in audio with mp3 module

2 Prerequisites:

Before going ahead it is better to learn these basic things

- Basic knowledge of Python
- Basic knowledge of javascript and HTML
- Basic knowledge of Linux
- Basic knowledge of NLTK
- Basic Electronics knowledge

3 Requirement

- Hardware Requirement:

- Galileo Board
- Leap Motion Sensor
- microSD card of minimum size of 8 GB with card reader
- speaker
- MP3 Module
- LAN cable or wifi module(for connecting board with PC)

- Software Requirement:

- Python editor (vi is enough)
- Leap sdk
- LeapTrainer.js
- NLTK for Python
- Tornado Web-socket
- Any web browser
- Bitwise SSH Client
- Wireshark(optional)

4 Theory and Description

This much things you must learn before starting this project:

- Leap motion library for Python
- LeapTrainer.js
- Python programming
- Websocket
- getting started with Galileo board
- interfacing with MP3 module

Leap motion Library:

We will see about python library. All class's name are given as human body part so it is very easy to understand. Main and useful classes are,

- Frame
- Hand
- Finger
- Pointable
- Arm

For getting data from leap sensor first we need to register listener class with controller object and for that first we need to create listener class. And register that listener class object with controller object you can find the basic code for getting basic frame data from given code examples “connection_with_leap.py”

Frame:

A frame is an object which can be generated from controller object. Leap sensor sends data through this frame object in listener class. From frame object, we can find all other objects. Means all row data comes with the frame. As per leap FPS `on_frame(self,controller)` method of listener class is called with a new frame. Each frame has its unique id and time stamp.

In `on_frame` method from controller object, we can get the current frame. It also supports serialization.

Hand:

Hands are the main entity tracked by the Leap Motion controller. The Hand class represents a physical hand detected by the Leap. A Hand object provides access to lists of its pointables as well as attributes describing the hand position, orientation, and movement.

From current frame object, we can get the current set of detected hand as an array. We can get hand from hands object by these properties,

```
1 frame = controller.frame()
2 hands = frame.hands
3
4 leftmost = hands.leftmost
5 rightmost = hands.rightmost
6 frontmost = hands.frontmost
```

Listing 1: getting frame and hand object

We can get these characteristics from hand object,

- `isRight`, `isLeft` — Whether the hand is a left or a right hand.
- `palm_position` — The center of the palm measured in millimeters from the Leap Motion origin.
- `palm_velocity` — The speed and movement direction of the palm in millimeters per second.

- `palm_normal` — A vector perpendicular to the plane formed by the palm of the hand. The vector points downward out of the palm.
- `direction` — A vector pointing from the center of the palm toward the fingers.
- `grab_strength`, `pinch_strength` — Describe the posture of the hand.
- `stabilized_palm_position` — The stabilized palm position of this Hand.
- Motion factors — Provide relative scale, rotation, and translation factors for movement between two frames.

We can get a pitch, yaw, and roll of hand by these properties,

```
1 pitch = hand.direction.pitch
2 yaw = hand.direction.yaw
3 roll = hand.palm_normal.roll
```

Listing 2: getting pitch yaw and roll from hand object

Figures & Pointables:

Fingers are Pointable objects that the Leap Motion software has classified as a finger. Get valid Finger objects from a Frame or a Hand object.

This can be generated from frame object or hand object. As this way,

```
1 frame_pointables = frame.pointables
2 hand_pointables = hand.pointables
3 known_pointable = hand.pointable(known_id)
4 finger = Finger(pointable)
```

If pointable is detected as the particular figure then and then only it can be returned as finger object. But mostly this pointer is detected as a finger. We can get a bone object from finger object and get the position of

all joints.

```
1 bone = finger.bone(Bone.TYPE_PROXIMAL)
```


inputs:

- 0 = TYPE_METACARPAL – The metacarpal bone.
- 1 = TYPE_PROXIMAL – The proximal phalanx; the closest bone segment to the hand.
- 2 = TYPE_INTERMEDIATE – The intermediate, or middle, phalanx.
- 3 = TYPE_DISTAL – The distal phalanx; the bone segment furthest from the hand.

```
1
2 Joint_position = finger.joint_position(0)
3
```

inputs:

- 0 = JOINT_MCP – The metacarpophalangeal joint, or knuckle, of the finger.
- 1 = JOINT_PIP – The proximal interphalangeal joint of the finger. This joint is the middle joint of a finger.
- 2 = JOINT_DIP – The distal interphalangeal joint of the finger. This joint is closest to the tip.
- 3 = JOINT_TIP – The tip of the finger.

We can get this type of properties from pointable object,

- id - A unique ID assigned to this Pointable object.
- Frame - The Frame associated with this Pointable object.
- Hand - The Hand associated with this finger.
- tip_position - The tip position in millimeters from the Leap Motion origin.
- tip_velocity - The rate of change of the tip position in millimeters/second.
- direction - The direction in which this finger is pointing.

- width - The estimated width of the finger in millimeters.
- length - The estimated length of the finger in millimeters.
- is_finger - Whether or not the Pointable is classified as a finger.
- stabilized_tip_position - The stabilized tip position of this Pointable.
- time_visible - The duration of time this Pointable has been visible to the Leap Motion Controller.

Arm:

There is also an arm object which gives data related elbow_position, arm.wrist_position, width, direction, the basis of all three axes.

Arm object is generated from hand object.

```
1 hand = frame.hands.frontmost
2 arm = hand.arm
```

attributes:

- basis:

Basis vectors specify the orientation of an arm.

 - x_basis. Perpendicular to the longitudinal axis of the arm; exits laterally through the sides of the wrist.
 - y_basis or up vector. Perpendicular to the longitudinal axis of the arm; exits the top and bottom of the arm. More positive in the upward direction.
 - z_basis. Aligned with the longitudinal axis of the arm. More positive toward the elbow.

```
1 basis = arm.basis
2 x_basis = basis.x_basis
3 y_basis = basis.y_basis
4 z_basis = basis.z_basis
```

```

5 center=arm.elbow_position+(arm.wrist_position-arm.
  elbow_position*.05
6 arm_transform = Leap.Matrix(x_basis , y_basis , z_basis ,
  center)

```

- direction - The normalized direction of the arm from elbow to wrist.
- elbow_position - The position of the elbow.
- width - The average width of the arm.
- wrist_position - The position of the wrist.

Here we will use only some properties like,

- In hand,
 - direction
 - palm_position
 - stabilized_palm_position
- In pointable,
 - direction
 - tip_position
 - stabilized_tip_position

LeapTrainer.js:

This API is used gesture and pose learning and recognition framework for leap motion sensor. This is javascript based API, so using that framework we can detect gestures and poses in a web browser, and then using websocket it will go to python program. So here is the overview of this API, **Uses:**

```
1 <script src="http://js.leapmotion.com/0.2.0/leap.min.js"></script>
2 <script src="/path/to/leaptrainer.min.js"></script>
```

You need to include these two files for using this framework. The first file is for connection with leap sensor and second for this framework. Then we need to create leap sensor's object and trainer object

```
1 var leapController = new Leap.Controller();
2 var trainer = new LeapTrainer.Controller({controller:
  leapController});
3 leapController.connect();
```

Training the system: We can train the system by adding new gesture or pose to system, and we can add new gesture or pose by calling this method,

```
1 trainer.create('Halt');
```

here in place of 'Halt' give any name related to your gesture or pose. And we can do something when this gesture is recognized like,

```
1 trainer.on('Halt', function() { console.log('Stop right there!')
  ; });
```

Importing and Exporting from LeapTrainer:

This feature is most useful in real application. Because normally whatever data we create like learning gestures or poses, are temporary means once you reload the script all objects will be reinitialized so all saved gestures or poses data will be lost. So by using this feature we can save that data. This framework exports data in the form of JSON and also accepts imports in the form of JSON.

```
1 var savedGesture = trainer.toJSON( 'Halt' );  
2 //Exporting to JSON  
3  
4 trainer.fromJSON(savedData);  
5 //Importing from JSON
```

This can be used for saving and fetching gesture or pose data into or from file
Options: Here are some options for changing the setting of LeapTrainer,

- **controller:** An instance of Leap.Controller class from the Leap Motion Javascript API. This will be created with default settings if not passed as an option.
- **pauseOnWindowBlur:** If this variable is TRUE, then the LeapTrainer Controller will pause when the browser window loses focus, and resume when it regains focus (default: FALSE)
- **minRecordingVelocity:** The minimum velocity a frame needs to be measured at in order to trigger gesture recording. Frames with a velocity below this speed will cause gesture recording to stop. Frame velocity is measured as the fastest moving hand or fingertip in view (default: 300)
- **maxRecordingVelocity:** The maximum velocity a frame can measure at and still trigger pose recording, or above which to pose recording will be stopped (default: 30)
- **minGestureFrames:** The minimum number of frames that can contain a recognizable gesture (default: 5)
- **minPoseFrames:** The minimum number of frames that need to hit as recordable before pose recording is actually triggered. This higher this number, the longer a pose needs to be held in position before recognition will be attempted. (default: 75)

- **minPoseFrames:**The minimum number of frames that need to hit as recordable before pose recording is actually triggered. This higher this number, the longer a pose needs to be held in position before recognition will be attempted. (default: 75)
- **minPoseFrames:**The minimum number of frames that need to hit as recordable before pose recording is actually triggered. This higher this number, the longer a pose needs to be held in position before recognition will be attempted. (default: 75)
- **hitThreshold:**The return value of the recognition function above which a gesture is considered recognized. Raise this to make gesture recognition more strict (default: 0.7)
- **trainingCountdown:**The number of seconds after startTraining is called that training begins. This number of training-countdown events will be emitted. (default: 3)
- **trainingGestures:**The number of training gestures required to be performed in training mode (default: 1)
- **convolutionFactor:**The factor by which training samples will be convolved over a gaussian distribution in order to expand the input training data. Set this to zero to disable convolution (default: 0)
- **downtime:**The number of milliseconds after a gesture is identified before another gesture recording cycle can begin. This is useful, for example, to avoid a 'Swipe Left' gesture firing when a user moves his or her hand quickly back to center directly after performing a 'Swipe Right' (default: 1000)

Options can be passed to the LeapTrainer.Controller constructor like so:

```
1 new LeapTrainer.Controller({ minRecordingVelocity: 100, downtime: 100});
```

Events:

The LeapTrainer controller will emit events to listening components during training and recognition. The framework events are:

- **gesture-created:** Fired when a new gesture is added to a Leap-Trainer controller object - either by a call to the `create()` function or by importing a saved gesture via the `fromJSON()` function. Carries two parameters: `gestureName` and `trainingSkipped`. The latter will be true if this gesture was created by a call to the `create()` function in which the second parameter was true.
- **training-countdown:** Fired a configurable number of times, once per second, after the `startTraining` function is called, before the training started event fires and actual training begins
- **training-started:** Fired when training begins on a gesture - carries a single parameter, `gestureName`
- **training-complete:** Fired when training completes on a gesture - carries three parameters, `gestureName`, `trainingGestures`, and `isPose`. The second parameter is the array of encoded gestures recorded during training. The final parameter indicates whether a pose or a gesture has been learned.
- **training-gesture-saved::** Fired during training when a new training gesture is recorded - carries two parameters, `gestureName`, and `trainingGestures`. The latter is the array of encoded gestures recorded so far during training, the last entry of which will be the gesture just recorded.
- **started-recording:** Fired when the `recordableFrame` function returns TRUE and causes gesture recording. Fired when the `recordableFrame` function returns FALSE and recording was active, causing gesture recording to stop
- **gesture-detected:** Fired when any gesture is detected, regardless of whether it is recognized or not. This event fires before recognition are attempted and carries two parameters, `gesture` and `frameCount`. The first is the recorded encoded gesture, the second is the number of frames in the gesture.
- **gesture-recognized:** Fired when a known gesture is recognized - carries two parameters, `hit` and `gestureName`. The `fit` parameter is a value between 0.0 and 1.0 indicating how closely the recognized gesture was matched to the training data, with 1.0 being a 100% match.

- **gesture-unknown:** Fired when a gesture is detected that doesn't match any known gesture sufficiently to be considered a match. This event carries two parameters, `bestHit` and `closestGestureName`, identifying the closest known gesture.

These events can be registered as this way,

```
1 trainer.on('training-started', function(movementName) {  
2     console.log('Started training ' + movementName);  
3 });
```


Installation of Linux image in Galileo board:

In Galileo board, we can install Linux, windows IOT or os x. So the interface with board become easy. Here we will use Linux image. For that, we need to write Linux image on the sd card. From sd card, Galileo board will boot.

So How to prepare sd card???

Step 1:

Download Linux image from this location.

<https://downloadmirror.intel.com/26028/eng/iot-devkit-prof-dev-image-galileo-201605.zip>

You will get one zip file extract it. You will find one 'direct'. You need to write this into sd card

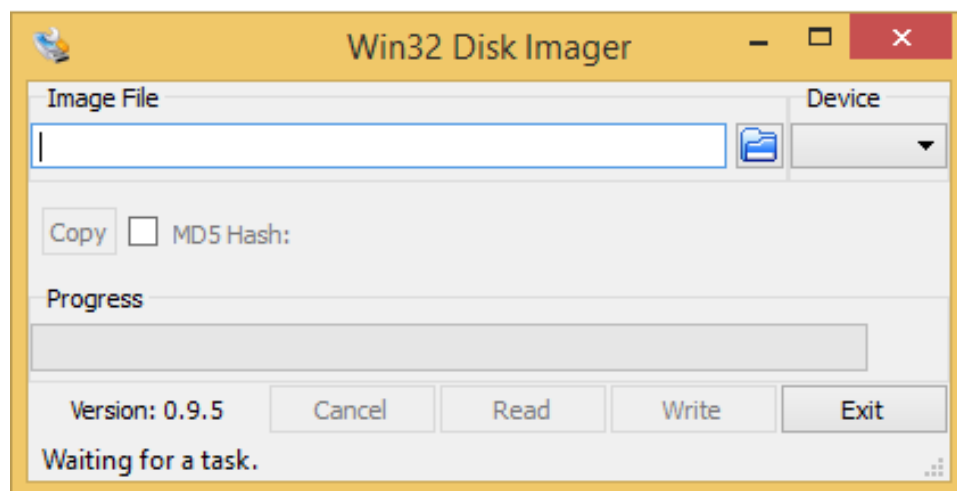
Step 2:

Download this software from this location,

http://download.softpedia.com/dl/d5a643aad00816ee735372c2d530a1a2/57584b56/100173006/software/cd_dvd_tools/Win32DiskImager-0.9.5-binary.zip

zip

this will be helpful for writing an image in sd card.



Select Image File which you have downloaded earlier in 'Image File' tab('direct' file)

Select sd card in 'Device tab'

Now click on 'Write' button, it will take some time. And wait until write process completes

Step 3:

Now insert sd card in galileo board and connect LAN cable with your PC and galileo board. Now connect the power supply with galileo board.

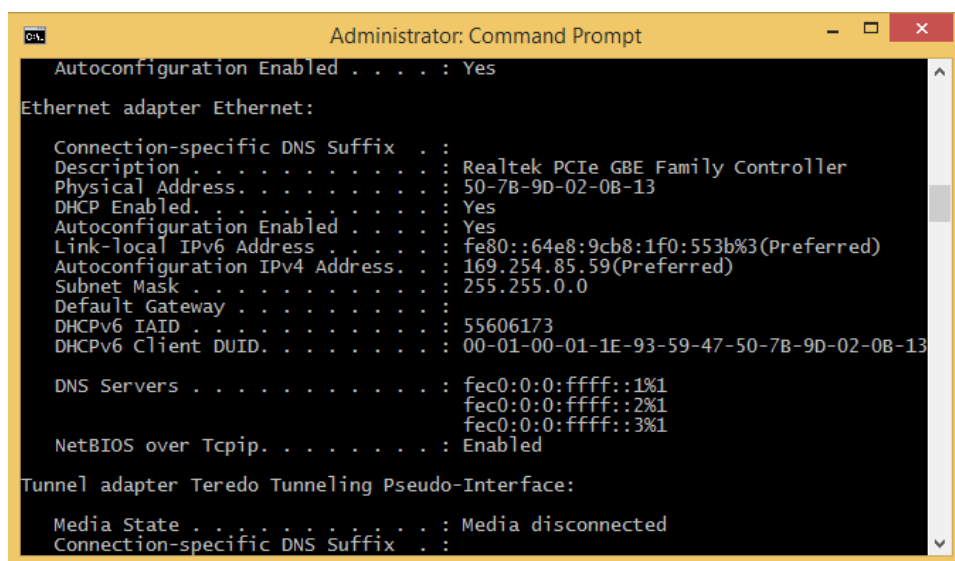
now the board will automatically boot. There will be blinking LED near sd card port. Wait until it stops lighting. Because it shows IO related sd card. So, when it become stop, IO related sd card is also stopped so the booting process is completed. Now you need to find the IP address of your board. For that, you can use any network packet tracer software like Wireshark, ettercap or any other.

I am using Wireshark.

Now open Wireshark select ethernet as a network adapter. It will start tracing all packet data on ethernet. But here on an ethernet network, there is only two devices are there your PC and board. So there is only two IP address data. (ignore IPs with 255, 251, 224 and more than 200 value in last two part like 192.168.0.255 or 255.255.255.0) so find your PC IP address using this command

```
1 ipconfig (for windows)}
2 ifconfig (for Linux)}
```

Now another IP showing on Wireshark is your board IP.



```
Administrator: Command Prompt
Autoconfiguration Enabled . . . . : Yes
Ethernet adapter Ethernet:

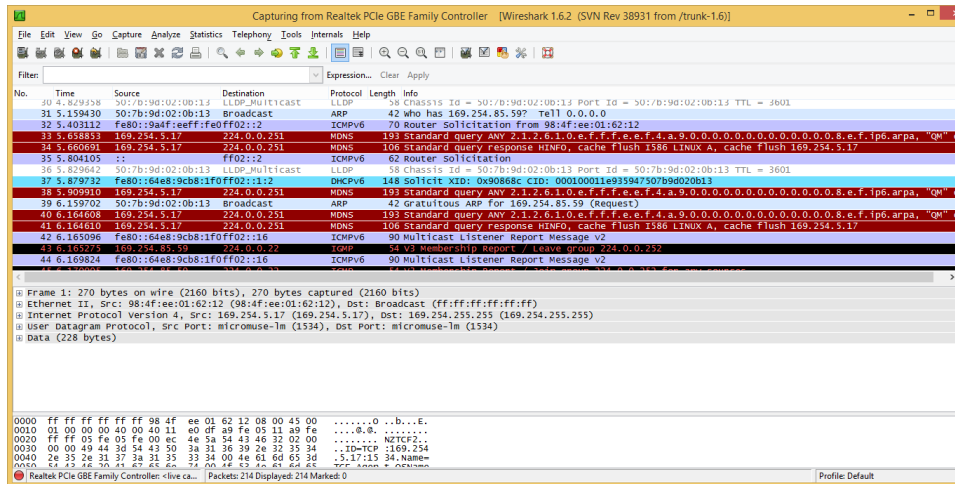
Connection-specific DNS Suffix  . : 
Description . . . . . : Realtek PCIe GBE Family Controller
Physical Address. . . . . : 50-7B-9D-02-0B-13
DHCP Enabled. . . . . : Yes
Autoconfiguration Enabled . . . . : Yes
Link-local IPv6 Address . . . . . : fe80::64e8:9cb8:1f0:553b%3(Preferred)
Autoconfiguration IPv4 Address. . : 169.254.85.59(Preferred)
Subnet Mask . . . . . : 255.255.0.0
Default Gateway . . . . . : 
DHCPv6 IAID . . . . . : 55606173
DHCPv6 Client DUID. . . . . : 00-01-00-01-1E-93-59-47-50-7B-9D-02-0B-13

DNS Servers . . . . . : fec0:0:0:ffff::1%1
                       fec0:0:0:ffff::2%1
                       fec0:0:0:ffff::3%1
NetBIOS over Tcpip. . . . . : Enabled

Tunnel adapter Teredo Tunneling Pseudo-Interface:

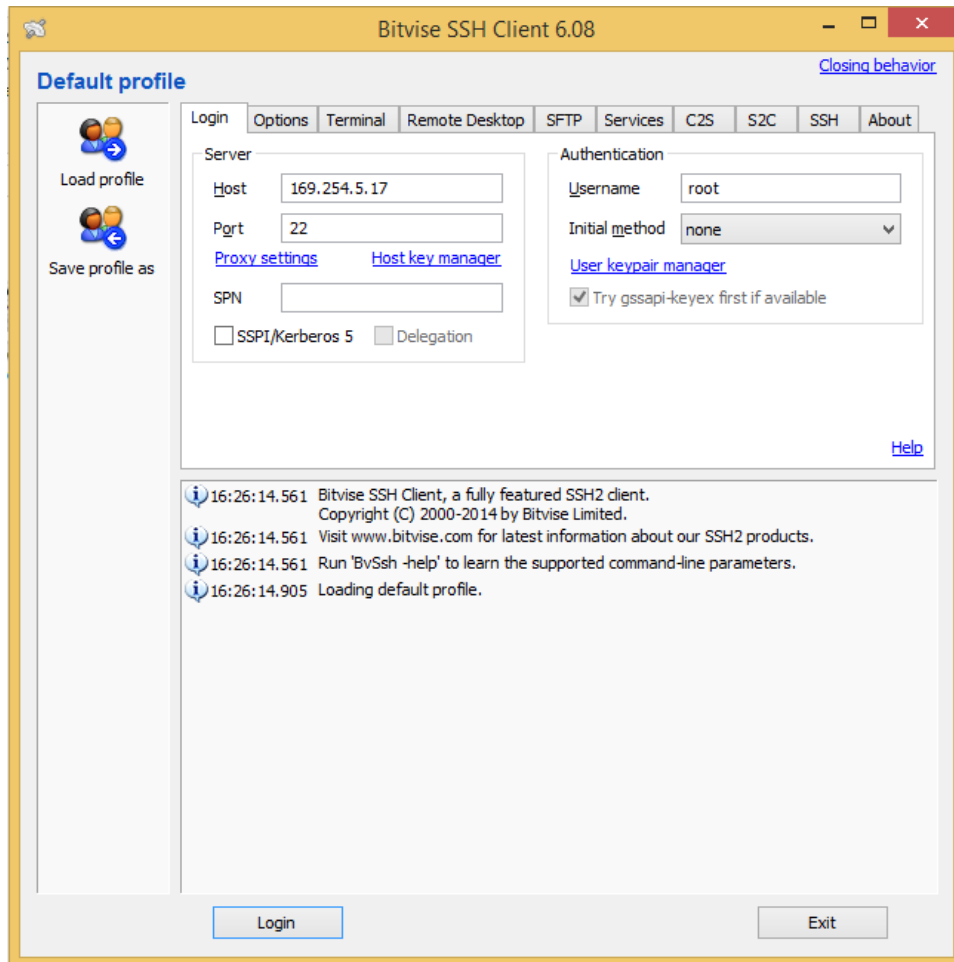
Media State . . . . . : Media disconnected
Connection-specific DNS Suffix  . :
```

Your Wireshark windows:

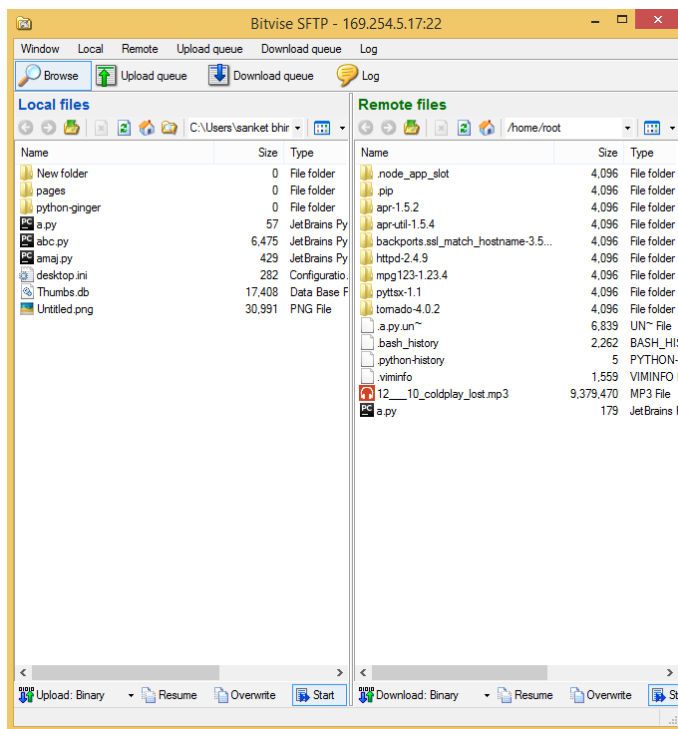


Step 3:

Now, from any SSH client make connection with board and use. :?)
(I'm using Bitvise SSH client)
Enter IP address and username: root there is no password.



Click on login, and wait. Now two windows will open one for terminal and another for file transfer. Your board is ready! :?)



Interfacing MP3 module:

MP3 module is used to play audio files stored in separate memory card attached with module. Control of this module is done the hex code sent from any controller via UART port. Here we will send code from Galileo board from Rx Tx pin to the board, there are some specified method to generating hex code which you can find on it's data sheet. We will se about that in next topic

5 Implementation:

- Alphabet training and recognizing with leap motion
- Basic structure of system
- Sending and saving data generated through LeapTrainer.js to Galileo board
- Changes in Interface of LeapTrainer
- Make natural sentence from words using NLTK
- Make simple player for MP3 module
- Link main project with MP3 module

Alphabet training and recognizing with leap motion:

Here we want to make a programme which records or say learns pose hand and then it will recognize. Means there will be two mode learning and recognizing. In Learning mode we need to perform pose so this pose will be saved for future reference and need to provide one alphabet for that pose. and In recognizing mode we need to perform specific pose so programme will recognize that pose and give related alphabet

Here we have used pointables x direction, pointable z diirection, pointable x stabilized tip position, pointable z stabilized tip position, stabilized palm position of x axis and z axis.

Here we want to implement a programme which is independent of hand's position means we can perform action at any position in range of sensor. so we have take data relative palm center so we have used stabilized palm position but here only x and z axis is useful because we don't need to care about height from sensor.

Here we will find radius distance from palm center to every tip of finger so we need to find mean position of tip's x and z co-ordinates related to palm center

We are also taking direction vectored so we can increase range of detection of radius distance

Then sum interface related programming. whole programme is in python language you can find at github https://github.com/sanketbhimani/eYSIP-2016-Sign-Language-Interpreter-Leap-Motion-Sensor-/blob/master/alphabet_using_leap.py

Silly mistake I made and also solve!

While copying object I was directly write like `a = b`. so programme was not working properly and only large performed pose goes to every object. I was not understanding what was happening. what was happening, when I write `a = b` it just transfer a pointer no new memory is allocated so when I change value for one object all values will change because they all are pointing to same memory location. then i used `a = copy.deepcopy(b)` then it will allocate new memory for that object. and it is working fine!!!

6 Exercise

< Give few sample experiment for practice. Include quiz or questions(as required).>

7 References

< Mention all references, books, articles or links required to understand the topic.>