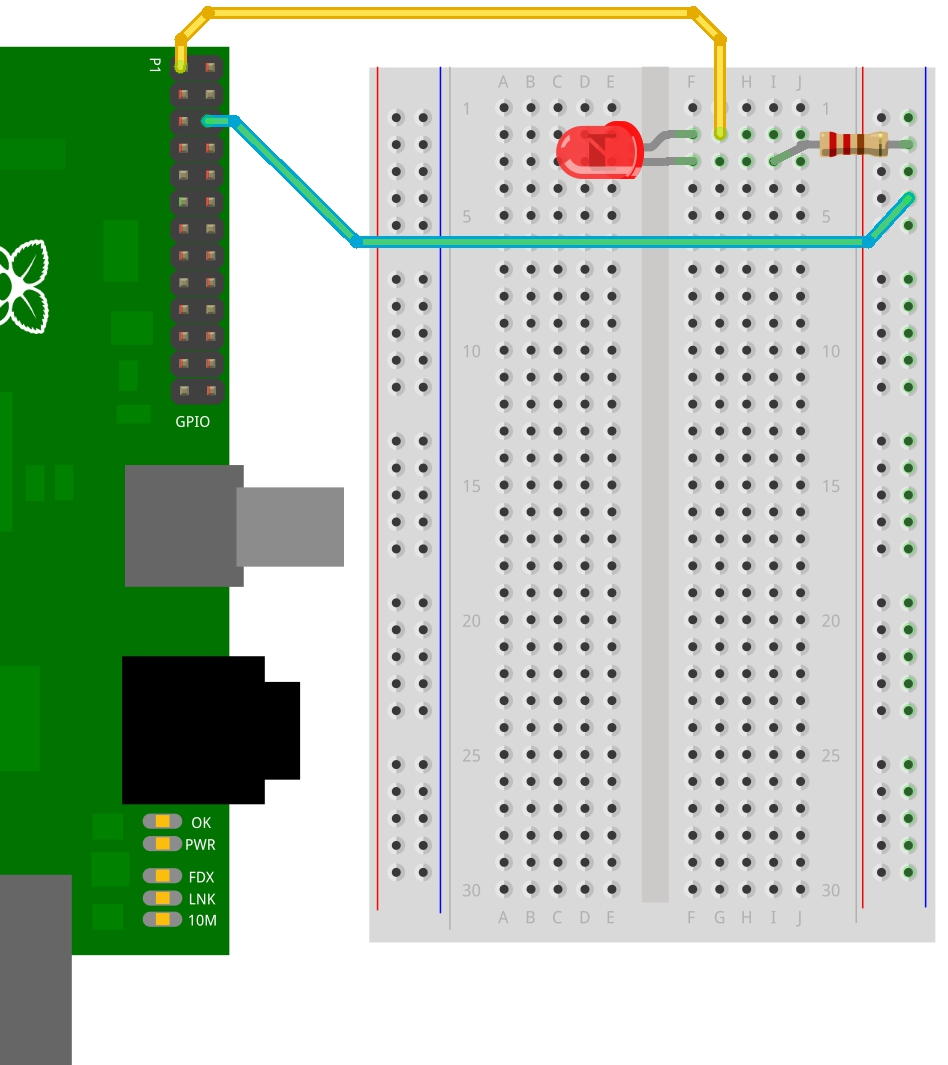
**DAY 1 SYLLABUS (led ,ldr , switch)**

**LED :**

****

import RPi.GPIO as GPIO

import time

# blinking function

def blink(pin):

GPIO.output(pin,GPIO.HIGH)

time.sleep(1)

GPIO.output(pin,GPIO.LOW)

time.sleep(1)

return

# to use Raspberry Pi board pin numbers

GPIO.setmode(GPIO.BOARD)

# set up GPIO output channel

GPIO.setup(11, GPIO.OUT)

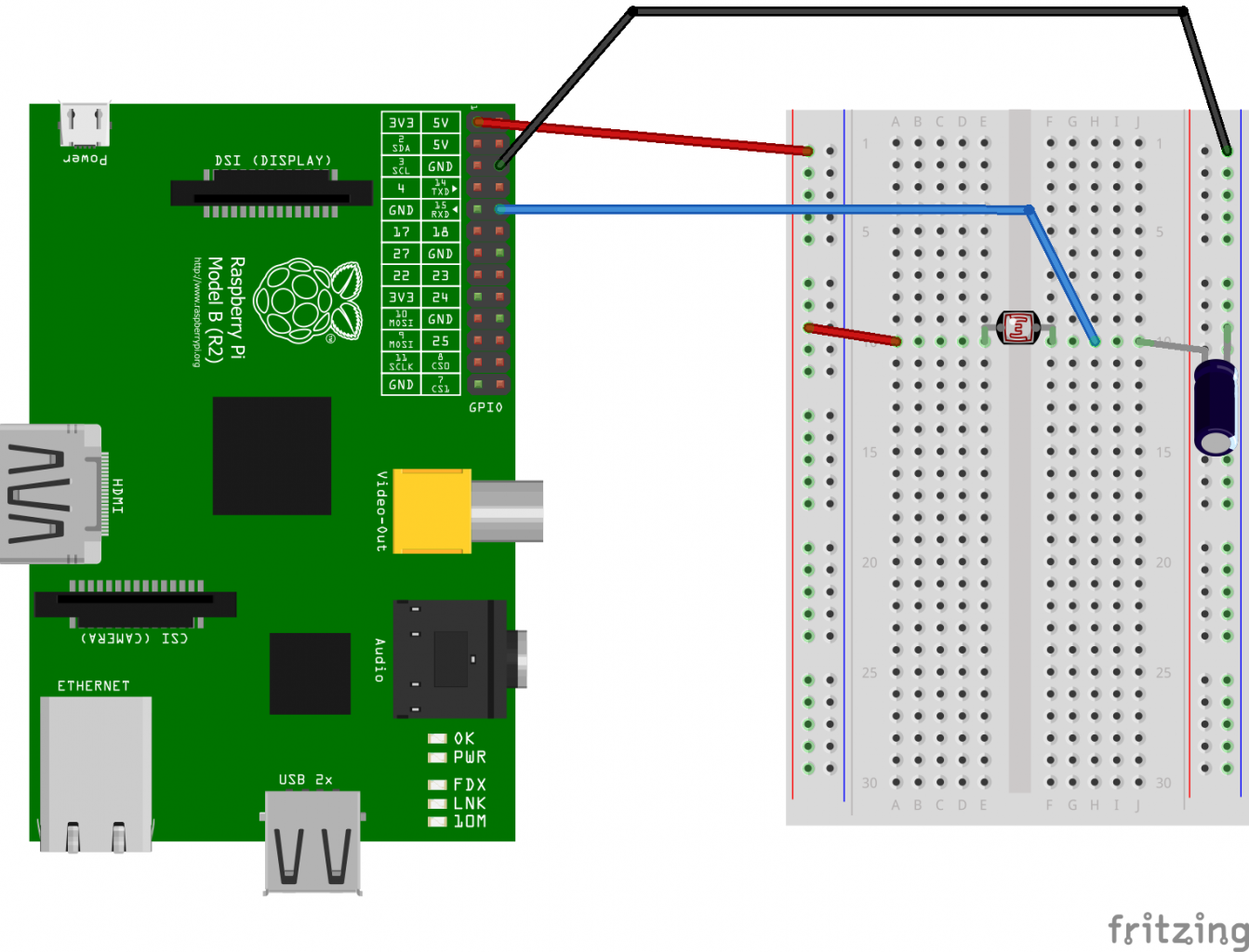
# blink GPIO17 50 times

for i in range(0,50):

blink(11)

GPIO.cleanup()

**LDR :**

import RPi.GPIO as GPIO, time # Get all the libraries we need

GPIO.setmode(GPIO.BOARD) # Set the GPIO library to use pin numbers from the Pi Board

def timer (pin): # Create a new function

reading = 0 # Create our counter and set it to zero

GPIO.setup(pin, GPIO.OUT) # Set the pin to output

GPIO.output(pin, GPIO.LOW) # Set the pin to low to discharge the capacitor

time.sleep(0.1) # wait for 100ms whilst the capacitor discharges

GPIO.setup(pin, GPIO.IN) # Set the pin to input

while (GPIO.input(pin) == GPIO.LOW): # keep looping until the capacitor is charged and the input hits high

reading += 1 # add one to our counter each loop

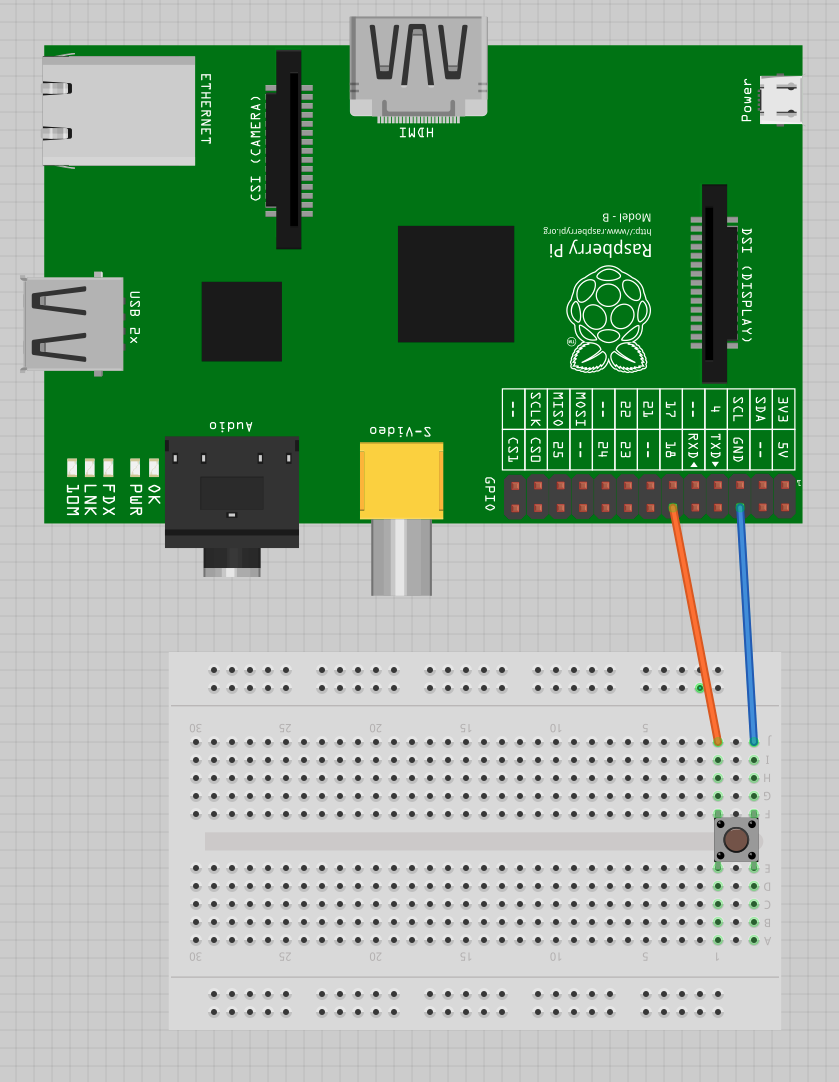
return reading # when the loop finishes, return the reading

while True:

result = timer(10)

print result

**BUTTON :**

****

import RPi.GPIO as GPIO

import time

GPIO.setmode(GPIO.BCM)

GPIO.setup(18, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)

while True:

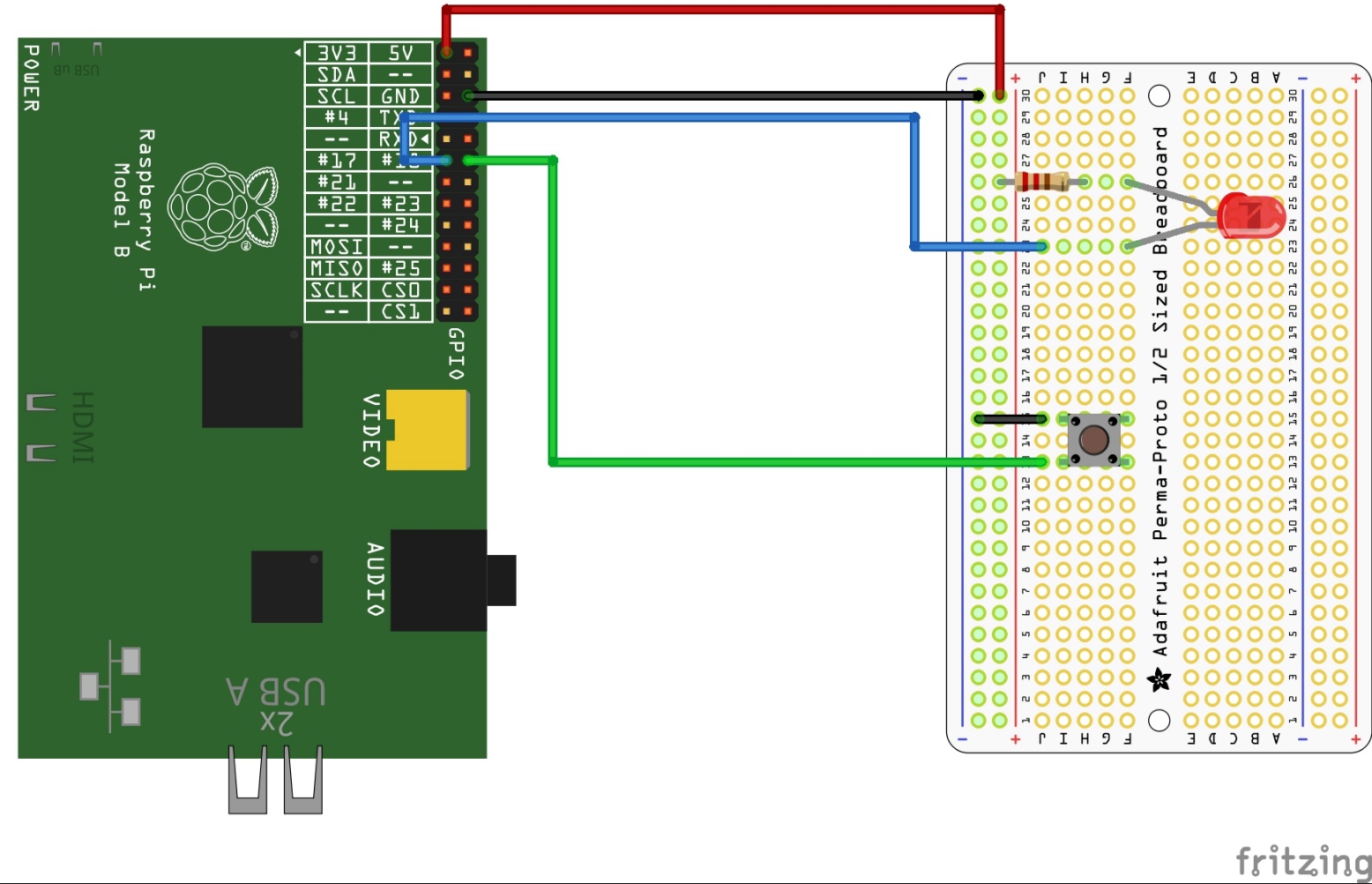
input\_state = GPIO.input(18)

if input\_state == False:

print('Button Pressed')

time.sleep(0.2)

**LED and BUTTON :**



import RPi.GPIO as GPIO

import time

led\_pin = 11 # Pin 11 will be managing the led

button\_pin = 12 # Pin 12 will be managing the inputs from the button

button\_status = 'off'

button\_pushed = False

GPIO.setmode(GPIO.BOARD)

GPIO.setup(led\_pin, GPIO.OUT)

GPIO.setup(button\_pin, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP) # Set button\_pin mode is input, and pull up to high level(3.3V)

GPIO.output(led\_pin, GPIO.HIGH) # Set LedPin high(+3.3V) to off led

try:

while True:

if GPIO.input(button\_pin) == GPIO.HIGH: # Check whether the button is pressed or not.

button\_pushed = True

else:

if button\_pushed:

button\_pushed = False

if button\_status == 'on':

button\_status = 'off'

GPIO.output(led\_pin, GPIO.HIGH)

else:

button\_status = 'on'

GPIO.output(led\_pin, GPIO.LOW)

print button\_status

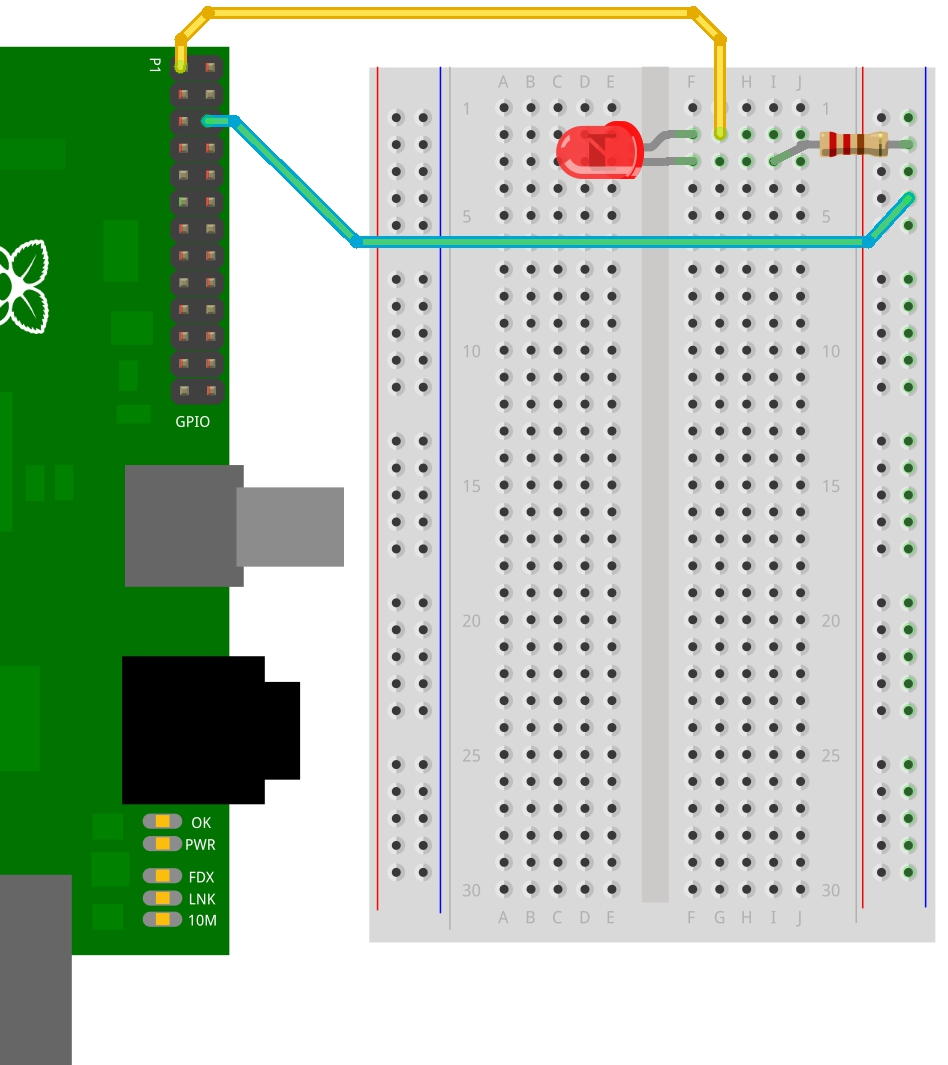
time.sleep(0.10)

except KeyboardInterrupt:

print "Ctrl-C - quit"

GPIO.cleanup()

**GUI CONTROL :**

****

from Tkinter import \*

import tkFont

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)

GPIO.setup(40, GPIO.OUT)

GPIO.output(40, GPIO.LOW)

win = Tk()

myFont = tkFont.Font(family = 'Helvetica', size = 36, weight = 'bold')

def ledON():

print("LED button pressed")

if GPIO.input(40) :

GPIO.output(40,GPIO.LOW)

ledButton["text"] = "LED ON"

else:

GPIO.output(40,GPIO.HIGH)

ledButton["text"] = "LED OFF"

def exitProgram():

print("Exit Button pressed")

GPIO.cleanup()

win.quit()

win.title("First GUI")

win.geometry('800x480')

exitButton = Button(win, text = "Exit", font = myFont, command = exitProgram, height = 2 , width = 6)

exitButton.pack(side = BOTTOM)

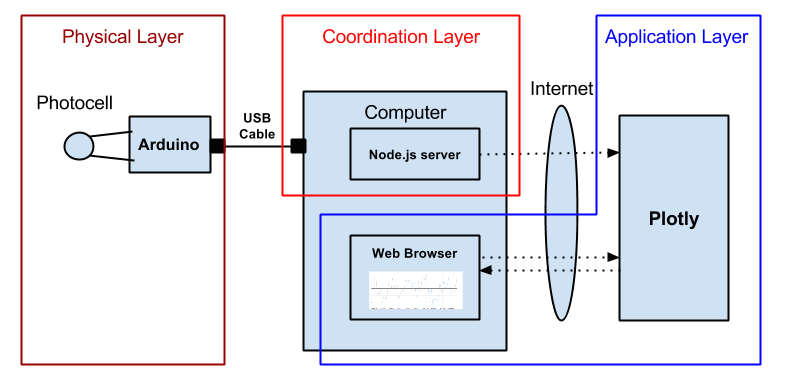
ledButton = Button(win, text = "LED ON", font = myFont, command = ledON, height = 2, width = 8 )

ledButton.pack()

mainloop()

**DAY 2 SYLLABUS (plotly ,Arduino , I2C display , SERVO motor , ULTRASONIC sensor )**

**PLOTLY :**



ARDUINO CODE : (analogRead serial)

**Server1.js program:**

var serialport = require('serialport');

var portName = '/dev/tty.usbmodem1411';

var sp = new serialport.SerialPort(portName, {

baudRate: 9600,

dataBits: 8,

parity: 'none',

stopBits: 1,

flowControl: false,

parser: serialport.parsers.readline("\r\n")

});

sp.on('data', function(input) {

console.log(input);

});

**Server2.js program:**

var serialport = require('serialport'),

plotly = require('plotly') ('Plotly\_UserName','Plotly\_API')

token = 'Plotly\_Token';

var portName = '/dev/tty.usbmodem1411';

var sp = new serialport.SerialPort(portName,{

baudRate: 9600,

dataBits: 8,

parity: 'none',

stopBits: 1,

flowControl: false,

parser: serialport.parsers.readline("\r\n")

});

// helper function to get a nicely formatted date string

function getDateString() {

var time = new Date().getTime();

// 32400000 is (GMT+9 Japan)

// for your timezone just multiply +/-GMT by 36000000

var datestr = new Date(time +32400000).toISOString().replace(/T/, ' ').replace(/Z/, '');

return datestr;

}

var initdata = [{x:[], y:[], stream:{token:token, maxpoints: 500}}];

var initlayout = {fileopt : "extend", filename : "sensor-test"};

plotly.plot(initdata, initlayout, function (err, msg) {

if (err) return console.log(err)

console.log(msg);

var stream = plotly.stream(token, function (err, res) {

console.log(err, res);

});

sp.on('data', function(input) {

if(isNaN(input) || input > 1023) return;

var streamObject = JSON.stringify({ x : getDateString(), y : input });

console.log(streamObject);

stream.write(streamObject+'\n');

});

});

**I2C LCD Display:**

import smbus

import time

# Define some device parameters

I2C\_ADDR = 0x27 # I2C device address

LCD\_WIDTH = 16 # Maximum characters per line

# Define some device constants

LCD\_CHR = 1 # Mode - Sending data

LCD\_CMD = 0 # Mode - Sending command

LCD\_LINE\_1 = 0x80 # LCD RAM address for the 1st line

LCD\_LINE\_2 = 0xC0 # LCD RAM address for the 2nd line

LCD\_LINE\_3 = 0x94 # LCD RAM address for the 3rd line

LCD\_LINE\_4 = 0xD4 # LCD RAM address for the 4th line

LCD\_BACKLIGHT = 0x08 # On

#LCD\_BACKLIGHT = 0x00 # Off

ENABLE = 0b00000100 # Enable bit

# Timing constants

E\_PULSE = 0.0005

E\_DELAY = 0.0005

#Open I2C interface

#bus = smbus.SMBus(0) # Rev 1 Pi uses 0

bus = smbus.SMBus(1) # Rev 2 Pi uses 1

def lcd\_init():

# Initialise display

lcd\_byte(0x33,LCD\_CMD) # 110011 Initialise

lcd\_byte(0x32,LCD\_CMD) # 110010 Initialise

lcd\_byte(0x06,LCD\_CMD) # 000110 Cursor move direction

lcd\_byte(0x0C,LCD\_CMD) # 001100 Display On,Cursor Off, Blink Off

lcd\_byte(0x28,LCD\_CMD) # 101000 Data length, number of lines, font size

lcd\_byte(0x01,LCD\_CMD) # 000001 Clear display

time.sleep(E\_DELAY)

def lcd\_byte(bits, mode):

# Send byte to data pins

# bits = the data

# mode = 1 for data

# 0 for command

bits\_high = mode | (bits & 0xF0) | LCD\_BACKLIGHT

bits\_low = mode | ((bits<<4) & 0xF0) | LCD\_BACKLIGHT

# High bits

bus.write\_byte(I2C\_ADDR, bits\_high)

lcd\_toggle\_enable(bits\_high)

# Low bits

bus.write\_byte(I2C\_ADDR, bits\_low)

lcd\_toggle\_enable(bits\_low)

def lcd\_toggle\_enable(bits):

# Toggle enable

time.sleep(E\_DELAY)

bus.write\_byte(I2C\_ADDR, (bits | ENABLE))

time.sleep(E\_PULSE)

bus.write\_byte(I2C\_ADDR,(bits & ~ENABLE))

time.sleep(E\_DELAY)

def lcd\_string(message,line):

# Send string to display

message = message.ljust(LCD\_WIDTH," ")

lcd\_byte(line, LCD\_CMD)

for i in range(LCD\_WIDTH):

lcd\_byte(ord(message[i]),LCD\_CHR)

def main():

# Main program block

# Initialise display

lcd\_init()

while True:

# Send some test

lcd\_string("CrazySchool <",LCD\_LINE\_1)

lcd\_string("I2C LCD <",LCD\_LINE\_2)

time.sleep(3)

# Send some more text

lcd\_string("> CrazySchool",LCD\_LINE\_1)

lcd\_string("> I2C LCD",LCD\_LINE\_2)

time.sleep(3)

main()

**ULTRASONIC SENSOR :**

import RPi.GPIO as GPIO

import time

GPIO.setmode(GPIO.BCM)

TRIG = 23

ECHO = 24

print "Distance Measurement In Progress"

GPIO.setup(TRIG,GPIO.OUT)

GPIO.setup(ECHO,GPIO.IN)

GPIO.output(TRIG, False)

print "Waiting For Sensor To Settle"

time.sleep(2)

GPIO.output(TRIG, True)

time.sleep(0.00001)

GPIO.output(TRIG, False)

while GPIO.input(ECHO)==0:

pulse\_start = time.time()

while GPIO.input(ECHO)==1:

pulse\_end = time.time()

pulse\_duration = pulse\_end - pulse\_start

distance = pulse\_duration \* 17150

distance = round(distance, 2)

print "Distance:",distance,"cm"

GPIO.cleanup()

**SERVO MOTOR GUI CONTROL :**

[**http://razzpisampler.oreilly.com/ch05.html**](http://razzpisampler.oreilly.com/ch05.html)

**LCD & ultrasonic**

import RPi.GPIO as GPIO

import time

import smbus

GPIO.setmode(GPIO.BCM)

TRIG = 23

ECHO = 24

print "Distance Measurement In Progress"

GPIO.setup(TRIG,GPIO.OUT)

GPIO.setup(ECHO,GPIO.IN)

GPIO.output(TRIG, False)

print "Waiting For Sensor To Settle"

time.sleep(2)

GPIO.output(TRIG, True)

time.sleep(0.00001)

GPIO.output(TRIG, False)

while GPIO.input(ECHO)==0:

pulse\_start = time.time()

while GPIO.input(ECHO)==1:

pulse\_end = time.time()

pulse\_duration = pulse\_end - pulse\_start

distance = pulse\_duration \* 17150

distance = round(distance, 2)

print "Distance:",distance,"cm"

lcd\_dist= str(distance)

# Define some device parameters

I2C\_ADDR = 0x27 # I2C device address

LCD\_WIDTH = 16 # Maximum characters per line

# Define some device constants

LCD\_CHR = 1 # Mode - Sending data

LCD\_CMD = 0 # Mode - Sending command

LCD\_LINE\_1 = 0x80 # LCD RAM address for the 1st line

LCD\_LINE\_2 = 0xC0 # LCD RAM address for the 2nd line

LCD\_LINE\_3 = 0x94 # LCD RAM address for the 3rd line

LCD\_LINE\_4 = 0xD4 # LCD RAM address for the 4th line

LCD\_BACKLIGHT = 0x08 # On

#LCD\_BACKLIGHT = 0x00 # Off

ENABLE = 0b00000100 # Enable bit

# Timing constants

E\_PULSE = 0.0005

E\_DELAY = 0.0005

#Open I2C interface

#bus = smbus.SMBus(0) # Rev 1 Pi uses 0

bus = smbus.SMBus(1) # Rev 2 Pi uses 1

def lcd\_init():

# Initialise display

lcd\_byte(0x33,LCD\_CMD) # 110011 Initialise

lcd\_byte(0x32,LCD\_CMD) # 110010 Initialise

lcd\_byte(0x06,LCD\_CMD) # 000110 Cursor move direction

lcd\_byte(0x0C,LCD\_CMD) # 001100 Display On,Cursor Off, Blink Off

lcd\_byte(0x28,LCD\_CMD) # 101000 Data length, number of lines, font size

lcd\_byte(0x01,LCD\_CMD) # 000001 Clear display

time.sleep(E\_DELAY)

def lcd\_byte(bits, mode):

# Send byte to data pins

# bits = the data

# mode = 1 for data

# 0 for command

bits\_high = mode | (bits & 0xF0) | LCD\_BACKLIGHT

bits\_low = mode | ((bits<<4) & 0xF0) | LCD\_BACKLIGHT

# High bits

bus.write\_byte(I2C\_ADDR, bits\_high)

lcd\_toggle\_enable(bits\_high)

# Low bits

bus.write\_byte(I2C\_ADDR, bits\_low)

lcd\_toggle\_enable(bits\_low)

def lcd\_toggle\_enable(bits):

# Toggle enable

time.sleep(E\_DELAY)

bus.write\_byte(I2C\_ADDR, (bits | ENABLE))

time.sleep(E\_PULSE)

bus.write\_byte(I2C\_ADDR,(bits & ~ENABLE))

time.sleep(E\_DELAY)

def lcd\_string(message,line):

# Send string to display

message = message.ljust(LCD\_WIDTH," ")

lcd\_byte(line, LCD\_CMD)

for i in range(LCD\_WIDTH):

lcd\_byte(ord(message[i]),LCD\_CHR)

def main():

# Main program block

# Initialise display

lcd\_init()

while True:

# Send some test

lcd\_string(lcd\_dist,LCD\_LINE\_1)

lcd\_string("cm",LCD\_LINE\_2)

time.sleep(3)

main()

**LCD and ultrasonic**

import RPi.GPIO as GPIO

import time

import lcd as LCD

GPIO.setmode(GPIO.BCM)

TRIG = 23

ECHO = 24

print "Distance Measurement In Progress"

GPIO.setup(TRIG,GPIO.OUT)

GPIO.setup(ECHO,GPIO.IN)

GPIO.output(TRIG, False)

print "Waiting For Sensor To Settle"

time.sleep(2)

GPIO.output(TRIG, True)

time.sleep(0.00001)

GPIO.output(TRIG, False)

while GPIO.input(ECHO)==0:

pulse\_start = time.time()

while GPIO.input(ECHO)==1:

pulse\_end = time.time()

pulse\_duration = pulse\_end - pulse\_start

distance = pulse\_duration \* 17150

distance = round(distance, 2)

print "Distance:",distance,"cm"

LCD.lcd\_init()

LCD.lcd\_string("Distance:"+str(distance)+"cm",LCD.LCD\_LINE\_1)

GPIO.cleanup()

This lcd (small lcd is the file name )as lcd is a filename of lcd.py program

MINI PROJECT : (ultrasonic sensor ,led and lcd )

import RPi.GPIO as GPIO

import time

import lcd\_lib as LCD

GPIO.setmode(GPIO.BOARD)

TRIG = 16

ECHO = 18

LED = 12

print "Distance Measurement In Progress"

LCD.lcd\_init()

GPIO.setup(TRIG,GPIO.OUT)

GPIO.setup(ECHO,GPIO.IN)

GPIO.setup(LED,GPIO.OUT)

GPIO.output(TRIG, False)

print "Waiting For Sensor To Settle"

time.sleep(2)

while True:

GPIO.output(TRIG, True)

time.sleep(0.00001)

GPIO.output(TRIG, False)

while GPIO.input(ECHO)==0:

pulse\_start = time.time()

while GPIO.input(ECHO)==1:

pulse\_end = time.time()

pulse\_duration = pulse\_end - pulse\_start

distance = pulse\_duration \* 17150

distance = round(distance, 2)

print "Distance:",distance,"cm"

LCD.lcd\_string("Distance: " + str(distance) + "cm",LCD.LCD\_LINE\_1)

if distance > 10:

GPIO.output(LED,GPIO.HIGH)

LCD.lcd\_string("LED ON",LCD.LCD\_LINE\_2)

else:

GPIO.output(LED,GPIO.LOW)

LCD.lcd\_string("LED OFF",LCD.LCD\_LINE\_2)

time.sleep(1)

GPIO.cleanup()