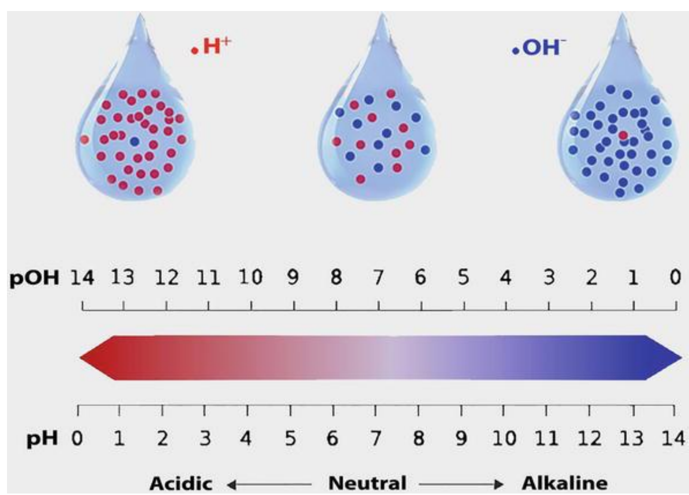
Monitoring Device

Introduction:

Since the industrial revolution, the world has discovered new sources of pollution nearly every day. So, air and water can potentially become polluted everywhere. Little is known about changes in pollution rates. The increase in water -related diseases/issues provide a real assessment of the degree of pollution in the environment. According to its quality, water can be classified into four types. Those four water quality types are discussed through an extensive review of their important common attributes including physical, chemical and biological parameters. These water quality parameters are reviewed in termed of definition, sources, impacts, effects, and measuring methods. Based upon its source, water can be divided into ground water and surface water. Both types of water can be exposed to contamination risks from agricultural, industrial, and domestic activities, which may include many types of pollutants such as heavy metals, pesticides, fertilizers, hazardous chemicals, and oils. Water quality can be classified into four types -potable water, palatable water, contaminated (polluted) water, and infected water.

Chemical Properties of water:

Ph is one of the most important parameters of water quality. It is defined as the negative logarithm of the hydrogen ion concentration. It is a dimensionless number indicating the strength of an acidic or a basic solution. PH of water is a measure of how acidic/basic water is. Acidic water contains extra hydrogen ions (H^+) and basic water contains extra hydroxyl (OH^-) ions. The amount of oxygen in water increases as pH rises. Low-pH water will corrode or dissolve metals and other substances



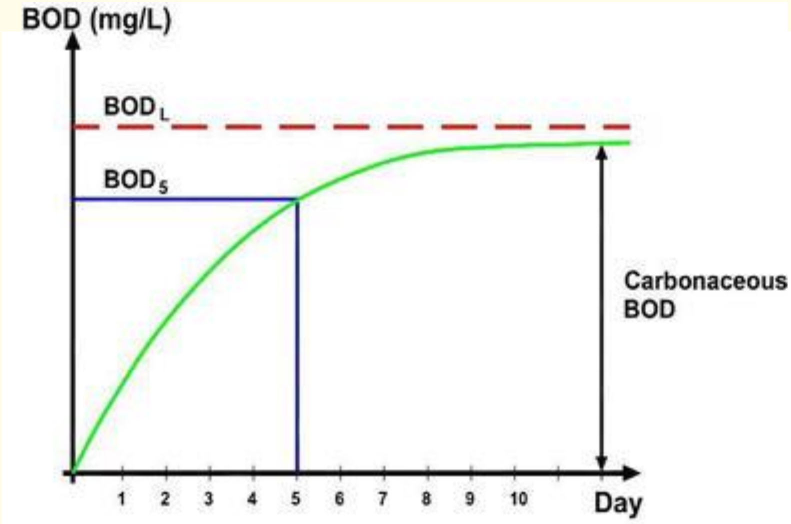
Dissolved oxygen (DO) is one of the most important parameters of water quality in streams, rivers, and lakes. It is a key test of water pollution. The higher the concentration of dissolved oxygen, the better the water quality.

Oxygen is slightly soluble in water and very sensitive to temperature. For example, the saturation concentration at 20°C is about 9 mg/L and at 0°C is 14.6 mg/L.

The actual amount of dissolved oxygen varies depending on pressure, temperature, and salinity of the water. Dissolved oxygen has no direct effect on public health but drinking water with very little or no oxygen tastes unpalatable to some people.

There are three main methods used for measuring dissolved oxygen concentrations: the colorimetric method—quick and inexpensive, the Winkler titration method—traditional method, and the electrometric method

Biological oxygen Demand



(BOD) Bacteria and other microorganisms use organic substances for food. As they metabolize organic material, they consume oxygen. The organics are broken down into simpler compounds, such as CO_2 and H_2O, and the microbes use the energy released for growth and reproduction.

When this process occurs in water, the oxygen consumed is the DO in the water. If oxygen is not continuously replaced by natural or artificial means in the water, the DO concentration will reduce as the microbes decompose the organic materials. This need for oxygen is called the biochemical oxygen demand (BOD). The more organic material there is in the water, the higher the BOD used by the microbes will be. BOD is used as a measure of the power of sewage; strong sewage has a high BOD and weak sewage has low BOD. The complete decomposition of organic material by microorganisms takes time, usually 20 d or more under ordinary circumstances. The quantity of oxygen used in a specified volume of water to fully decompose or stabilize all biodegradable organic substances is called the ultimate BOD or BOD_L

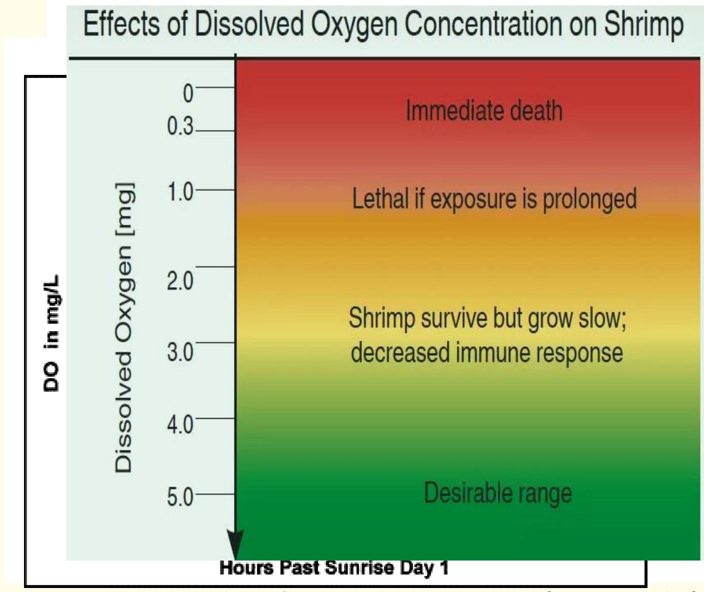
BODt=BODL×(1−10−kt)   E10

where BOD_t = BOD at any time t, mg/L; BOD_L = ultimate BOD, mg/L; k = a constant representing the rate of the BOD reaction; t = time, d.

Chemical oxygen demand (COD)

The chemical oxygen demand (COD) is a parameter that measures all organics: the biodegradable and the non-biodegradable substances . It is a chemical test using strong oxidizing chemicals (potassium dichromate), sulfuric acid, and heat, and the result can be available in just 2 h. COD values are always higher than BOD values for the same sample

In this we will typically discuss about the measures how to correct the DO and Ph parameters of water. For that we are introducing a device that will detect all the parameters in water.



Motive behind to Develop a different approach: In shrimp farming there is a monitored DO value is required in pond, If **DO** value of water drops, the shrimp coming into starving situation and as per study we found that during nighttime there is drastic fall in DO of water body and natural DO can't meet the aerobic and anaerobic requirements of pond. Environmental temperature has a profound effect on the metabolic rate of all poikilothermic organisms (cold-blooded organisms, have variable body temperature like temperature of its environment), and shrimp is no exception. Normally, a reduction in feed consumption results of around 10 percent (dry weight) for each degree Celsius lower water temperature. Because rains can lower pond water temperature by 3 to 5 degrees C, a minimum, temporary reduction of 30 percent in feed consumption can be expected.

As mentioned before, due to the relatively low density of rainwater a cold layer of fresh water will tend form under the pond's densest and warmest water. The effect of this pond water stratification or halocline with deeper colder water layers will slow down water heating by the sun. It is important to remove this cold, fresher layer of water or at least homogenize the pond water through some mechanical intervention to minimize the magnitude and speed of temperature change.

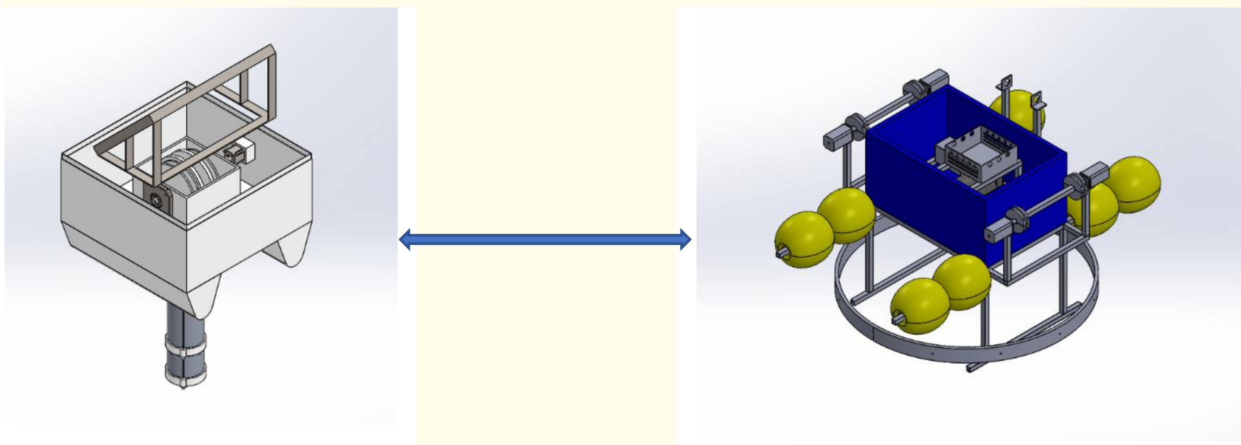
In addition to the reduction of appetite, these thermal stratification conditions will make the shrimp migrate towards pond areas with higher temperature and salinity and possibly away from the sound of rain on the pond surface. One consequence is a significant increase in the shrimp density in some deeper pond areas, where dissolved oxygen levels are the lowest and H_2S concentrations are the highest in the entire pond. If normal feed rations continue to be applied, bacterial decomposition of feed leftovers will aggravate the situation even more due to depression of pH and increase of BOD by aerobic respiration by heterotrophic bacterial populations.

Solution to this Problem: Bariflo labs is designing a Sensor Based Monitoring Device to monitor and rejuvenate the water parameters of water. This device will sense the DO, PH, ORP of water using Nanotech/Asmik/Discrete Optical Based sensors. The sensors will continuously read the values and send it to Aeration Device for pumping oxygen into water

Schematic Diagram:

Monitoring Device

Aeration Device



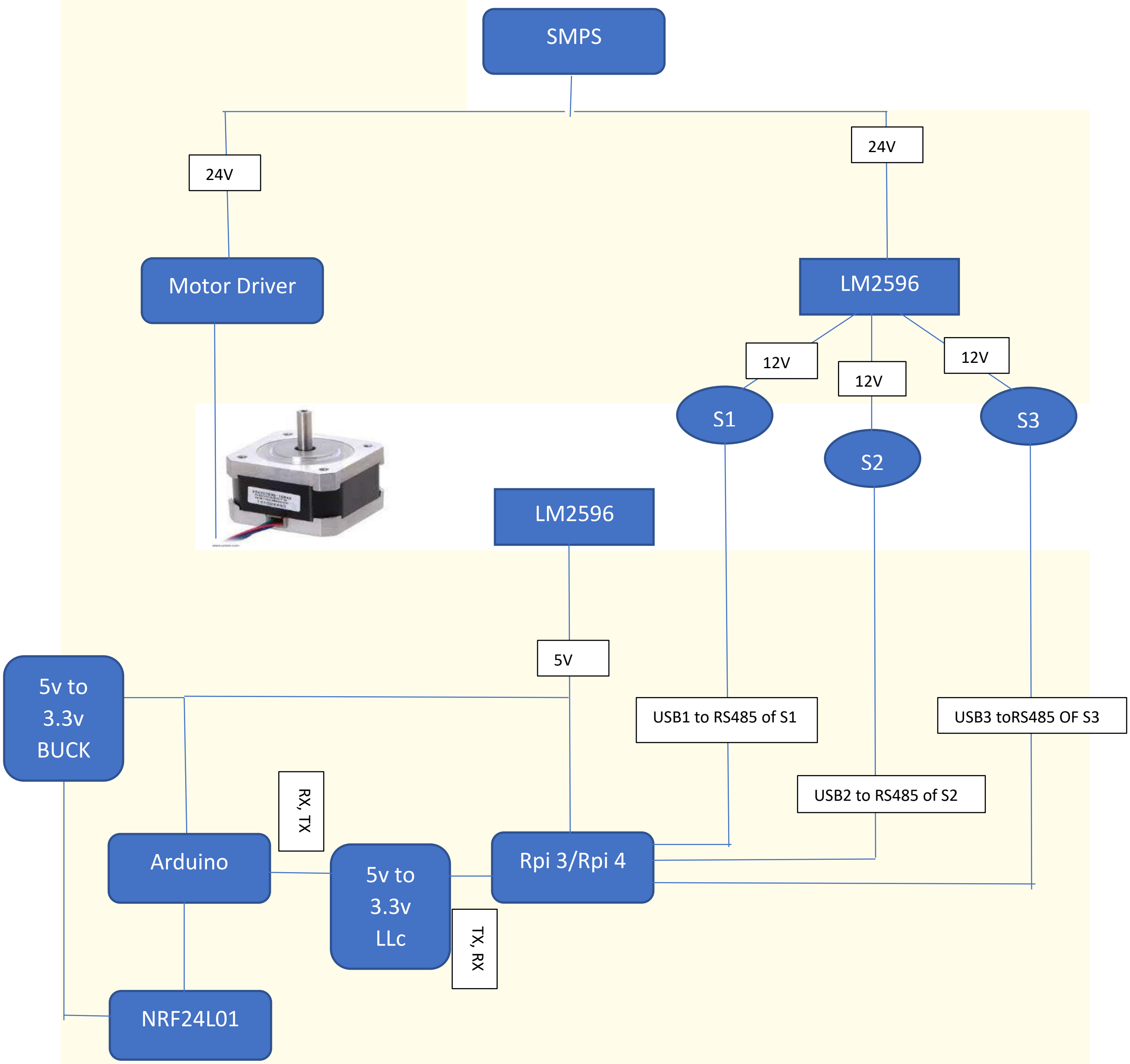
Following Points:

1.The monitoring Device will send continuously **DO** values to Aeration Device **via radio frequency communication**

2.After receiving the DO value the Aerator will start and it will run until the monitoring device sense the stable DO value again.

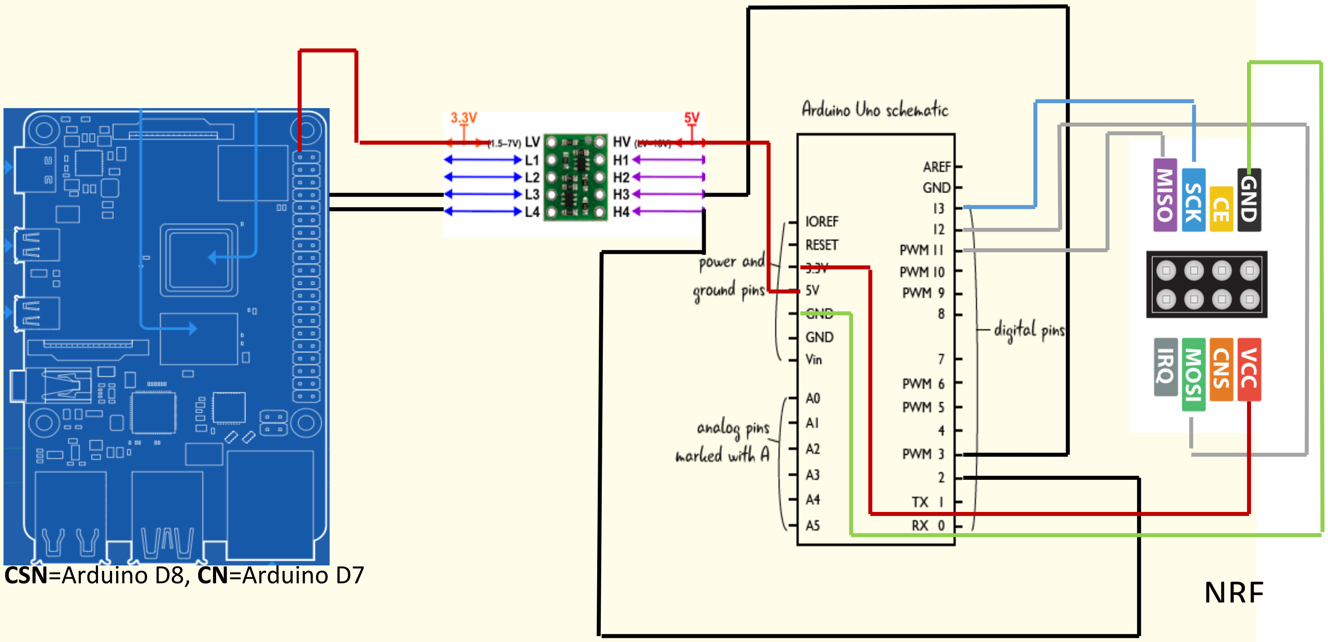
3. In return the aerator will send the Compressor current value to Monitoring Device for checking that aerator is working or not.

Algorithm of Monitoring Device:



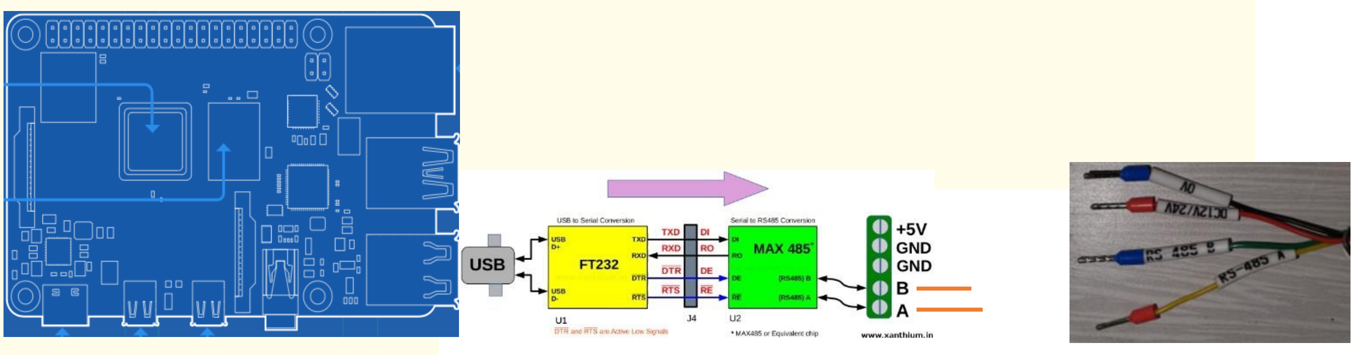
Connections:

Connection between Arduino and rpi AND nrf24l01:



CSN=Arduino D8, **CN**=Arduino D7

Connection between Rpi and Sensors:



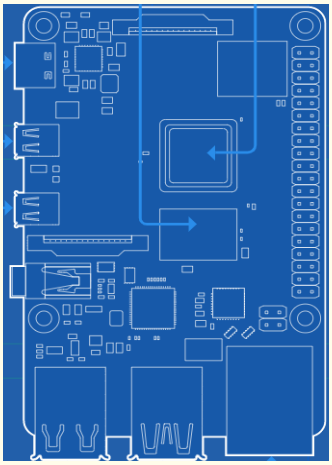
RS48(A)-Sesnor(A)

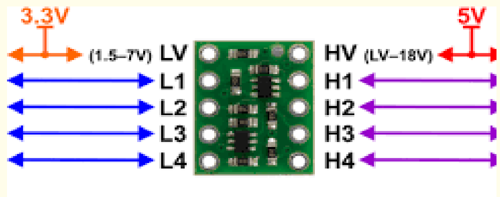
RS48(B)-Sesnor(B)

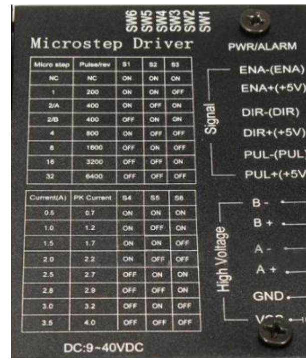
12 v supply-Sensor(12V)

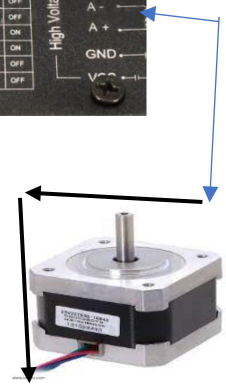
GND-GND

Connection between Rpi and motor:









GPIO22-LV2(for Direction)

GPIO17-LV3(for step)

3.3volt-LV1

GND-GND

HV2=DIR+

HV3-Pul+

DIR-,PUL- =GND

A+,A- = Motor (A=,A-) coil

B+,B-= Mptor(B+,B-) Coil

GND=GND

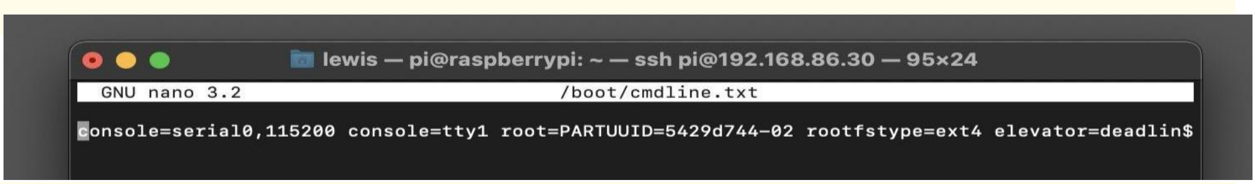
VCC=24 Volt(SMPS SUPPLY)

Software settings for Rpi:

For serial communication:

1. **python3 -m pip install pyserial'**
2. **sudo nano /boot/config.txt**
3. Scroll down to the end of the document and at the very end under '[all]' add a new line which reads:
   dtoverlay=disable-bt
4. We also need to disable the serial console. For this you need to edit the /boot/cmdline.txt file. Do this by using the command:
5. **sudo nano /boot/cmdline.txt**
6. Find the following text and remove it: 'console=serial0,115200'

Before:



After:

```
lewis — pi@raspberrypi: ~ — ssh pi@192.168.86.30 — 95×24
 GNU nano 3.2                    /boot/cmdline.txt                    Modified

console=tty1 root=PARTUUID=5429d744-02 rootfstype=ext4 elevator=deadline fsck.repair=yes rootw$
```

Reference code for Nobo-tech Sensor:

```
import time
import RPi.GPIO as GPIO
from time import sleep
import math
import serial
import binascii
import sys
from struct import unpack
from datetime import datetime
import random
import os
# This code is all about Ph,ORP,DO.
# IMPORTANT--> 1. Connect sensors in this order 1 by 1 -> PH, ORP, DO.  2.ORP sensor does not have temperature sensor.

GPIO.setwarnings(False)                    # Ignore warning for now
GPIO.setmode(GPIO.BCM)                     # Use physical pin numbering
pulpin = 17
dirpin = 27
GPIO.setup(pulpin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(dirpin, GPIO.OUT, initial=GPIO.HIGH)
#Calculating All sensor values
def calc(c,b,a) :
        data1 = c
        data2 = b
        sum1 = data1 + data2
        an_integer1 = int(data1, 16)
        an_integer2 = int(data2, 16)
        an_integer3 = int(sum1, 16)
        hex_value1 = hex(an_integer1)
        hex_value2 = hex(an_integer2)
        hex_value3 = hex(an_integer3)
        if a == "sensor" or a == "temp":
```

```
    res = int(hex_value3, 16)
    numstr = str(res)
   if a == "orp":
    res = int(hex_value2, 16)
    fdata = str(res)
   if a == "sensor":
    act1 = numstr[0]
    act2 = numstr[1]+numstr[2]+numstr[3]
    e1 = float(act1)
    e2 = float(act2)
    e22 = float(e2/1000)
    fdata = e1 + e22
   if a == "temp":
    act1 = numstr[0]+numstr[1]
    act2 = numstr[2]+numstr[3]
    e1 = float(act1)
    e2 = float(act2)
    e22 = float(e2/100)
    fdata = e1 +e22
   return fdata

#Calculating all sensor values
def read_data(x):
   if x == "do" or x == "temp_do":
    serialport2= "/dev/ttyUSB2"
   if x == "orp":
    serialport2= "/dev/ttyUSB0"
   if x == "ph" or x == "temp_ph":
    serialport2= "/dev/ttyUSB1"
   port2
=serial.Serial(serialport2,baudrate=9600,parity=serial.PARITY_NONE,stopbits=serial.STOPBITS_
ONE,bytesize=serial.EIGHTBITS,timeout=None)
   data=[0x01,0x03,0x00,0x00,0x00,0x02,0xC4,0x0B]
   port2.write(serial.to_bytes(data))
   time.sleep(.1)
   if port2.inWaiting()>0:
    data5=[]
    while port2.inWaiting()>0:
     data5.append(binascii.hexlify((port2.read())))
    if x == "ph" or x == "do":
```

```
          fdata = calc(data5[3],data5[4],"sensor")
         if x == "orp":
          fdata = calc(data5[3],data5[4],"orp")
         if x == "temp_ph" or x == "temp_do":
          fdata = calc(data5[5],data5[6],"temp")
         if x == "ph":
          print("PH Value of the water is " + str(fdata))
         if x == "do":
          print("DO Value of the water is " + str(fdata))
         if x == "temp_ph":
          print("PH Sensor temperature value is " + str(fdata))
         if x == "temp_do":
          print("DO Sensor temperature value is " + str(fdata))
         if x == "orp":
          print("ORP Value of the water is  " + str(fdata))
         return fdata
        else:
         if x == "ph":
          print("No Response from PH sensor")
         if x == "do":
          print("No Response from DO sensor")
         if x == "orp":
          print("No Response from ORP sensor")
         if x == "temp_ph":
          print("No Temperature from PH sensor")
         if x == "temp_do":
          print("No Temperature from DO sensor")
         sleep(1)
def sensors():
#Sensor reading taking loop

  motor(pul,0)
#Motor running function calling backward direction, change the direction motor will reverse the rotation
  #sleep(8)
  q=0
  global phavg
  global orpavg
  global doavg
  phavg = 0.0
```

```
orpavg = 0.0
doavg = 0.0
phc=0 #PH count
orpc=0#ORP count
doc=0#DO count
print("Sensor started taking reading...")
while q<3:
 s = read_data("ph")
 k = read_data("temp_ph")
 k2 = read_data("temp_do")
 p = read_data("orp")
 g = read_data("do")
 if(s!=None):
  phavg = float(s) + phavg
  phc +=1
  if(s>14):
   s = 14
 if(p!=None):
  orpavg = float(p) + orpavg
  orpc+=1
  if(int(p)>500):
   p = 330
 if(g!=None):
  doavg = float(g) + doavg
  doc +=1
  if(g>10):
   g = 9
 file = open("sensor_2datas.csv", "a")
 if os.stat("sensor_2datas.csv").st_size == 0:
    file.write("PH Data,PH Temperature,ORP Data,DO Data,DO Temperature\n")
 file.write(str(s)+","+str(k)+","+str(p)+","+str(g)+","+str(k2)+","+"\n")
 file.flush()
 q+=1
 if (q<3):
  sleep(20)
 else:
  if(phc!=0):
   phavg = phavg/phc
  if(orpc!=0):
   orpavg = orpavg/orpc
```

```
    if(doc!=0):
     doavg = doavg/doc
  print("Sensor Reading Taken.")
  print("data stored")
  print(str(phavg) + " Average PH Value")
  print(str(orpavg) + " Average ORP Value")
  print(str(doavg) + " Average DO Value")
  motor(pul,1)
  sleep(2)
#Motor Running Function
def motor(pul,dire):
  j=0
  if(dire==0):
   print("Motor Running Backward...")
   GPIO.output(dirpin, GPIO.LOW)
  if(dire==1):
   print("Motor Running Forward...")
   GPIO.output(dirpin, GPIO.HIGH)

  while True:
   GPIO.output(pulpin, GPIO.HIGH)
   sleep(0.0001)
   GPIO.output(pulpin, GPIO.LOW)
   sleep(0.0001)

   j+=1

   if(j==pul):
    GPIO.output(dirpin, GPIO.LOW)
    GPIO.output(pulpin, GPIO.LOW)
    print("Motor Stoped")
    break

while True:
 while True:
  try:
     rotation = 6                                        #how many rotations you need
     steps =1600                                         #no of steps per rotation.
     pul = rotation * steps
     #pul = 10000
```

```
    sensors()
    sleep(0.90)
  except Exception as e:
   sleep(1)
   error = 1
   print("")
   print(e)
   print("")
```

Reference code for Asmik Sensor:

```
# -*- coding: utf-8 -*-
import math
import time
import serial
import binascii
import time
import sys
import os
from time import sleep
import Adafruit_ADS1x15
import RPi.GPIO as GPIO
from datetime import datetime
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
#Motor pins Direction=22,Pulse=17
GPIO.setup(22,GPIO.OUT)                    #direction
GPIO.setup(17,GPIO.OUT)                    #pulse
step = 6000
speed = 0.000001
#First insert PH Sensor USB and then insert DO sensor USB to Raspberry pi
#PH sensor port will become USB0
#DO sensor port will be USB1
def calc(c,b,a) :
        data1 = c
        data2 = b
        sum1 = data1 + data2
        an_integer1 = int(data1, 16)
        an_integer2 = int(data2, 16)
        an_integer3 = int(sum1, 16)
        hex_value1 = hex(an_integer1)
```

```
        hex_value2 = hex(an_integer2)
        hex_value3 = hex(an_integer3)
        res = int(hex_value3, 16)
        #print("The decimal number associated with hexadecimal string is : " + str(res))
        if a == "temp":
                fdata = res*0.1
        if a == "sensor":
                fdata = res*0.01
        #print(phdata)
        return fdata
def read_data(x) :
        if x == "ph" or x == "temp_ph":
                serialport2= "/dev/ttyUSB0"
        if x == "do" or x == "temp_do":
        serialport2= "/dev/ttyUSB1"
        #port =
serial.Serial(serialPort,baudrate=9600,parity=serial.PARITY_NONE,stopbits=serial.STOPBITS_ON
E,bytesize=serial.EIGHTBITS,timeout=None)
        port2
=serial.Serial(serialport2,baudrate=9600,parity=serial.PARITY_NONE,stopbits=serial.STOPBITS_
ONE,bytesize=serial.EIGHTBITS,timeout=None)
        data=[0x01,0x03,0x00,0x00,0x00,0x03,0x05,0xCB]
        port2.write(serial.to_bytes(data))
        time.sleep(.1)
        if port2.inWaiting()>0:
                data5=[]
                while port2.inWaiting()>0:
                        data5.append(binascii.hexlify((port2.read())))
                #print("DO:")
                #print(data5)
                if x == "ph" or x == "do":
                        fdata = calc(data5[5],data5[6],"sensor")
                if x == "temp_ph" or x == "temp_do":
                        fdata = calc(data5[3],data5[4],"temp")


                if x == "ph":
                        print("PH Value of the water is " + str(fdata))
                if x == "do":
                        print("DO Value of the water is " + str(fdata))
```

```
        if x == "temp_ph":
            print("PH Sensor temperature value is " + str(fdata))
        if x == "temp_do":
            print("DO Sensor temperature value is " + str(fdata))
        return(fdata)
    else:
        if x == "ph":
            print("No Response from PH sensor")
        if x == "do":
            print("No Response from DO sensor")
        if x == "temp_ph":
            print("No Temperature from PH sensor")
        if x == "temp_do":
            print("No Temperature from DO sensor")
        sleep(2)
def Asmic_sensors():
    sleep(8)
    q=0
    global phavg
    global doavg
    phavg = 0.0
    doavg = 0.0
    phc=0 #PH count
    doc=0#DO count
    print("Sensor started taking reading...")
    s=read_data("ph")
    k=read_data("temp_ph")
    g=read_data("do")
    k2=read_data("temp_do")
    file = open("sensor_datas.csv", "a")
    if os.stat("sensor_datas.csv").st_size == 0:
    file.write("PH Data,PH Temperature,DO Data,DO Temperature\n")
    file.write(str(s)+","+str(k)+","+str(g)+","+str(k2)+","+"\n")
    file.flush()
while True :
    try:
        Asmic_sensors()
        sleep(0.90)
    except Exception as e:
        sleep(1)
```

```
            error = 1
            print("")
            print(e)
            print("")
```

Reference code for Stepper Motor:

```
def motor(pul,dire):
 j=0
 if(dire==0):
  print("Motor Running Backward...")
  GPIO.output(dirpin, GPIO.LOW)
 if(dire==1):
  print("Motor Running Forward...")
  GPIO.output(dirpin, GPIO.HIGH)
 while True:
  GPIO.output(pulpin, GPIO.HIGH)
  sleep(0.0001)
  GPIO.output(pulpin, GPIO.LOW)
  sleep(0.0001)
  j+=1
  if(j==pul):
   GPIO.output(dirpin, GPIO.LOW)
   GPIO.output(pulpin, GPIO.LOW)
   print("Motor Stoped")
   break
while True:
 while True:
  try:
    rotation =1                                    #how many rotations you need
    steps =1600     #no of steps per rotation. check stepper motor's no of steps and update here
    pul = rotation * steps
#Motor Forward
    motor(pul,1)
#Motor Reverse Running
    motor(pul,0)
    sleep(0.90)
  except Exception as e:
   sleep(1)
   error = 1
   print("")
```

```
  print(e)
  print("")
```

Reference code to send data from Rpi to Arduino and collecting data from Arduino to Rpi:

```
if __name__=='__main__':
  ser=serial.Serial('/dev/ttyAMA0',9600,timeout=1)
  ser.flush()
  while True:
    do=(random.randint(0,9))
    ph=(random.randint(0,20))
    orp=(random.randint(0,30))
    l1=[str(do)]
    l2=[str(ph)]
    l3=[str(orp)]
    l2=[str(do),str(ph)]
    l=[str(do),str(ph),str(orp)]
    send1=''.join(l1)
    send+=”\n”
    ser.write(send1.encode('utf-8'))
    time.sleep(1)
    line=ser.readline().decode('utf-8').rstrip()        // Reading data from Arduino
    print(line)                                         //Printing data from Arduino
    time.sleep(1)
```

Reference code to send data from Arduino NRF-1to Arduino NRF-2(Bidirectional):

```
#include <SPI.h>
#include "RF24.h"
#include <SoftwareSerial.h>
SoftwareSerial mySerial (2, 3);               //SoftwareSerial(rxPin, txPin, inverse_logic)
RF24 radio(8, 7);                             // CE=7,CSN=8
String data="";
byte node_A_address[6] = "NodeA";
byte node_B_address[6] = "NodeB";
void setup() {
 Serial.begin(9600);
 mySerial.begin(9600);
```

```
  radio.begin();
  // Set the PA Level low to prevent power supply related issues since this is a
  // getting started sketch, and the likelihood of proximity of the devices. RF24_PA_MAX is
default.
  radio.setPALevel(RF24_PA_LOW);
  // Open a writing and reading pipe on each radio, with opposite addresses
  radio.openWritingPipe(node_B_address);
  radio.openReadingPipe(1, node_A_address);
  // Start the radio listening for data
  radio.startListening();
}
void loop()
{
  radio.stopListening();                          // First, stop listening so we can talk.
  if (mySerial.available() > 0)
{
    data = mySerial.readStringUntil('\n');
 }
   float  myInt = data.toFloat();
   Serial.print("Incomming data :");
   Serial.println(myInt);
  float msg_from_B[20];
  unsigned long start_time = micros();
  if (!radio.write( &myInt, sizeof(myInt) ))
{
   Serial.println(F("failed"));
  }
  radio.startListening();                                 // Now, continue listening
  unsigned long started_waiting_at = micros();    // Set up a timeout period, get the current µSec
  boolean timeout = false;         // Set up a variable to indicate if a response was received or not
  while ( ! radio.available() )                          // While nothing is received
 {
   if (micros() - started_waiting_at > 200000 )
   {
     timeout = true;
     break;
   }
  }

  if ( timeout )
```

```
 {
   Serial.println(F("Failed, response timed out."));
  } else {
   float msg_from_B;                  // Grab the response, compare, and send to debugging spew
   radio.read( &msg_from_B, sizeof(msg_from_B) );
   unsigned long end_time = micros();
   // Spew it
   Serial.println(F("Sent '"));
  // Serial.print(msg_to_B);
   Serial.print(myInt);
   Serial.print(F("', Got response '"));
   Serial.print(msg_from_B);
   mySerial.print(msg_from_B);
   //mySerial.print(F("', Round-trip delay "));
  // mySerial.print(end_time - start_time);
  // mySerial.println(F(" microseconds"));
  }
 // Try again 1s later
 delay(3000);
}
```

Reference code to send data from Arduino NRF-2to Arduino NRF-1(Bidirectional)

```
#include <SPI.h>
#include "RF24.h"
#include <SoftwareSerial.h>
RF24 radio(7, 8);                          // 7(CE) & 8(CSN)
SoftwareSerial mySerial(2,3);               //SoftwareSerial(rxPin, txPin, inverse_logic)
byte node_A_address[6] = "NodeA";
byte node_B_address[6] = "NodeB";
String data1="";
void setup()
{
 Serial.begin(9600);
 mySerial.begin(9600);
 Serial.println("Namma");
 radio.begin();
 radio.setPALevel(RF24_PA_LOW);
 // Open a writing and reading pipe on each radio, with opposite addresses
```

```
  radio.openWritingPipe(node_A_address);
  radio.openReadingPipe(1, node_B_address);
  // Start the radio listening for data
  radio.startListening();
}
void loop()
{
  if (mySerial.available() >0)
  {
    data1=mySerial.readStringUntil('\n');
  }
  float myInt1=data1.toFloat();
  Serial.print("Current Sensor:");
  Serial.println(myInt1);
  float msg_from_A;
  if ( radio.available())
 {
    while (radio.available())                                    // While there is data ready
{

      radio.read( &msg_from_A, sizeof(msg_from_A) );            // Get the payload
    }
    radio.stopListening();                                      // First, stop listening so we can talk
    //char msg_to_A[20] = "Hello from node_B";
                                                               // Send the final one back.
    radio.startListening();                  // Now, resume listening so we catch the next packets.
    Serial.print(F("Got message '"));
    Serial.print(msg_from_A);
    mySerial.print(msg_from_A);
    Serial.print(F("', Sent response :'"));
    Serial.print(myInt1);
    Serial.println(F("'"));
  }
  delay(3000);
}
```