EEL 7170 : Introduction to IoT

Lab Report

Name:	**Anushkaa Ambuj**
Roll Number:	**B21ES006**
Program:	**B.Tech in ES**

Lab 4: LoRa Module with Raspberry Pi

1 InLab

1.1 Objective

The objective of this lab is to explore the usage of the LoRa module, configure it on a Raspberry Pi, and demonstrate communication between two teams using the LoRa module.

1.2 Components Used

- Raspberry Pi
- LoRa SX126X Module
- LiDAR Sensor (VL53L0X)
- OLED Display
- Jumper Wires (female to female)

1.3 Procedure

Part 1: Setting Up Raspberry Pi for LoRa Module

- **Step 1: Configuring Raspberry Pi**
  - Run the command `sudo raspi-config` to open the configuration menu.
  - Navigate to **Interface Options** using the arrow keys.
  - Select **Serial** from the options.
  - In the pop-up window, select **No** for the first option and **Yes** for the second option to enable the serial interface.
- **Step 2: Downloading the LoRa Module Code**
  - Run the following commands to download and unzip the LoRa module code:

```
cd Documents
wget https://files.waveshare.com/upload/1/18/SX126X_LoRa_HAT_CODE.zip
unzip SX126X_LoRa_HAT_CODE.zip
```

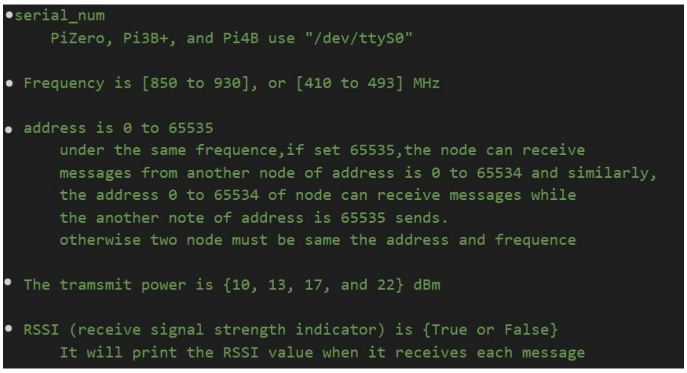
- **Step 3: Running the Sample Code**
  - Navigate to the code directory and run the Python code for the LoRa module:

```
cd ~/Documents/SX126X_LoRa_HAT_Code/raspberrypi/python/
sudo python3 main.py
```

- **Step 4: Configuring the LoRa Node**
  - Set the LoRa Node parameters as follows:

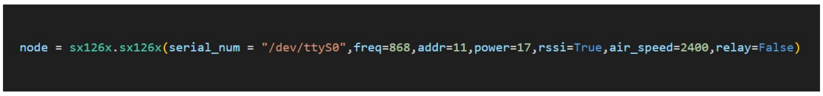
    ```
    node = sx126x.sx126x(serial_num="/dev/ttyS0", freq=868, addr=0,
    power=22, rssi=True, air_speed=2400, relay=False)
    ```

  - **freq**: Transmission frequency (set to 868 MHz) → Range:[850 to 930], or [410 to 493] MHz
    addr: Node address → Range: 0 to 65535
    power: Set transmission power → Range: 10, 13, 17, and 22 dBm
    rssi: Display RSSI value → Range: True or False



```
•serial_num
    PiZero, Pi3B+, and Pi4B use "/dev/ttyS0"

• Frequency is [850 to 930], or [410 to 493] MHz

• address is 0 to 65535
    under the same frequence,if set 65535,the node can receive
    messages from another node of address is 0 to 65534 and similarly,
    the address 0 to 65534 of node can receive messages while
    the another note of address is 65535 sends.
    otherwise two node must be same the address and frequence

• The tramsmit power is {10, 13, 17, and 22} dBm

• RSSI (receive signal strength indicator) is {True or False}
    It will print the RSSI value when it receives each message
```

  - Set the address and power of the LORA module



```
node = sx126x.sx126x(serial_num = "/dev/ttyS0",freq=868,addr=11,power=17,rssi=True,air_speed=2400,relay=False)
```

Part 2: Experiment

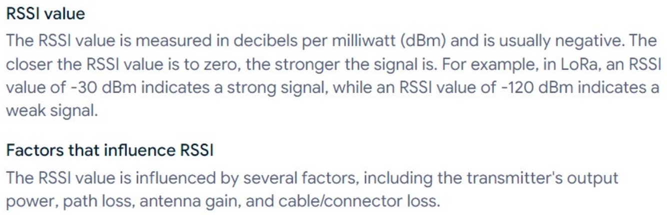
- **Task 1: Changing Transmission Power and Checking RSSI**
  - Modify the transmission power in the LoRa configuration.
  - Observe and note the changes in RSSI values at the receiver end.
- **Task 2: Transmitting Receiving Data**
  - To transmit a message, press `i` and input the following:

    ```
    {RECIEVER_ADDRESS},868,{Text_To_Send}
    ```

  - Type the correct address of Team B's Lora module to whom you want to send the data.

1.4 Observations

- Significance of RSSI Value (Received Signal Strength Index): The stronger the strength of the received signal, the closer the value of RSSI to zero.



RSSI value

The RSSI value is measured in decibels per milliwatt (dBm) and is usually negative. The closer the RSSI value is to zero, the stronger the signal is. For example, in LoRa, an RSSI value of -30 dBm indicates a strong signal, while an RSSI value of -120 dBm indicates a weak signal.

Factors that influence RSSI

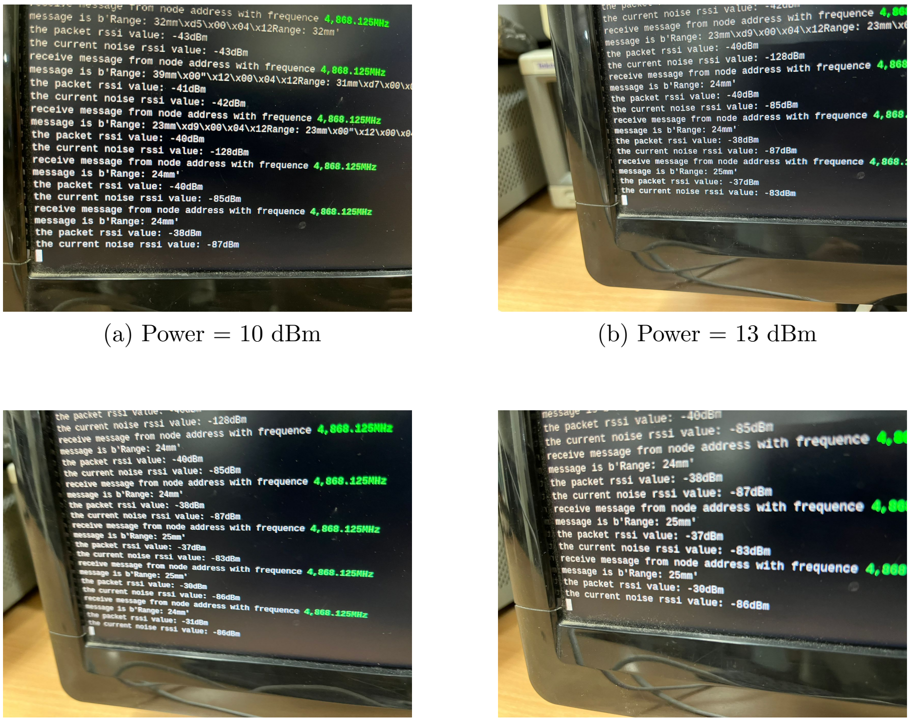
The RSSI value is influenced by several factors, including the transmitter's output power, path loss, antenna gain, and cable/connector loss.

- The RSSI value changed according to the transmission power adjustments, allowing us to observe the effect of power on signal strength.

Transmission Power	RSSI Value
10 dBm	-38 dBm
13 dBm	-37 dBm
17 dBm	-31 dBm
22 dBm	-30 dBm

Table 1: Measured RSSI values for various Transmission Power

- We generally find that the transmission power is directly proportional to the received signal strength.



```
receive message from node address with frequence 4,868.125MHz
message is b'Range: 32mm\xd5\x00\x04\x12Range: 32mm'
the packet rssi value: -43dBm
the current noise rssi value: -43dBm
receive message from node address with frequence 4,868.125MHz
message is b'Range: 39mm\x00"\x12\x00\x04\x12Range: 31mm\xd7\x00\x
the packet rssi value: -41dBm
the current noise rssi value: -42dBm
receive message from node address with frequence 4,868.125MHz
message is b'Range: 23mm\xd9\x00\x04\x12Range: 23mm\x00"\x12\x00\x0
the packet rssi value: -40dBm
the current noise rssi value: -128dBm
receive message from node address with frequence 4,868.125MHz
message is b'Range: 24mm'
the packet rssi value: -40dBm
the current noise rssi value: -85dBm
receive message from node address with frequence 4,868.125MHz
message is b'Range: 24mm'
the packet rssi value: -38dBm
the current noise rssi value: -87dBm
```

(a) Power = 10 dBm

```
the current noise rssi value: -42dBm
receive message from node address with frequence 4,86
message is b'Range: 23mm\xd9\x00\x04\x12Range: 23mm\x
the packet rssi value: -40dBm
the current noise rssi value: -128dBm
receive message from node address with frequence 4,868
message is b'Range: 24mm'
the packet rssi value: -40dBm
the current noise rssi value: -85dBm
receive message from node address with frequence 4,868.
message is b'Range: 24mm'
the packet rssi value: -38dBm
the current noise rssi value: -87dBm
receive message from node address with frequence 4,868.
message is b'Range: 25mm'
the packet rssi value: -37dBm
the current noise rssi value: -83dBm
```

(b) Power = 13 dBm

```
the current noise rssi value: -128dBm
receive message from node address with frequence 4,868.125MHz
message is b'Range: 24mm'
the packet rssi value: -40dBm
the current noise rssi value: -85dBm
receive message from node address with frequence 4,868.125MHz
message is b'Range: 24mm'
the packet rssi value: -38dBm
the current noise rssi value: -87dBm
receive message from node address with frequence 4,868.125MHz
message is b'Range: 25mm'
the packet rssi value: -37dBm
the current noise rssi value: -83dBm
receive message from node address with frequence 4,868.125MHz
message is b'Range: 25mm'
the packet rssi value: -30dBm
the current noise rssi value: -86dBm
receive message from node address with frequence 4,868.125MHz
message is b'Range: 24mm'
the packet rssi value: -31dBm
the current noise rssi value: -86dBm
```

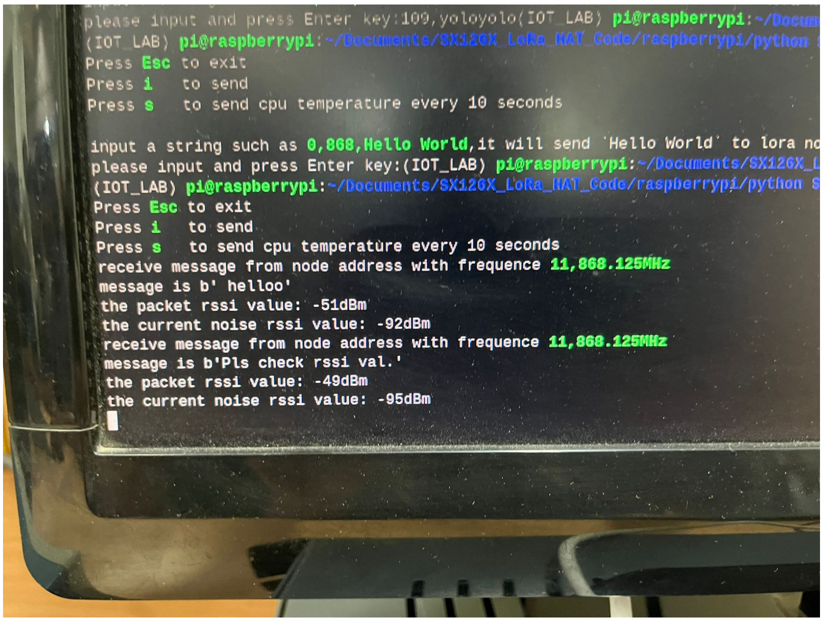
(c) Power = 17 dBm

```
the packet rssi value: -40dBm
the current noise rssi value: -85dBm
receive message from node address with frequence 4,8
message is b'Range: 24mm'
the packet rssi value: -38dBm
the current noise rssi value: -87dBm
receive message from node address with frequence 4,86
message is b'Range: 25mm'
the packet rssi value: -37dBm
the current noise rssi value: -83dBm
receive message from node address with frequence 4,868
message is b'Range: 25mm'
the packet rssi value: -30dBm
the current noise rssi value: -86dBm
```

(d) Power = 22 dBm

Figure 1: Power output at different levels

- The LiDAR data was successfully transmitted from Team A to Team B, and the OLED module displayed the received data.



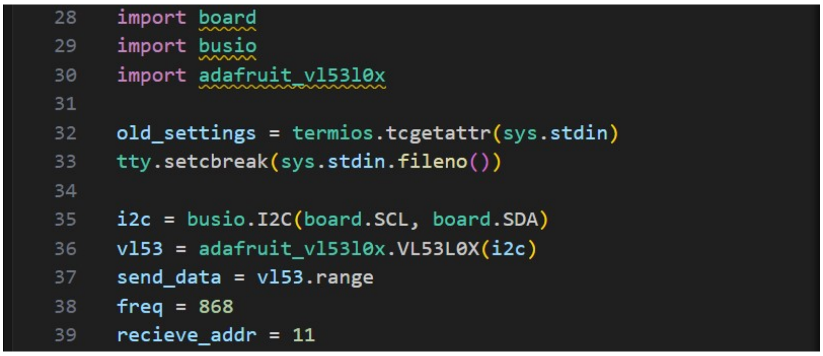
```
please input and press Enter key:109,yoloyolo(IOT_LAB) pi@raspberrypi:~/Docum
(IOT_LAB) pi@raspberrypi:~/Documents/SX126X_LoRa_HAT_Code/raspberrypi/python
Press Esc to exit
Press i   to send
Press s   to send cpu temperature every 10 seconds

input a string such as 0,868,Hello World,it will send `Hello World` to lora no
please input and press Enter key:(IOT_LAB) pi@raspberrypi:~/Documents/SX126X_L
(IOT_LAB) pi@raspberrypi:~/Documents/SX126X_LoRa_HAT_Code/raspberrypi/python
Press Esc to exit
Press i   to send
Press s   to send cpu temperature every 10 seconds
receive message from node address with frequence 11,868.125MHz
message is b' helloo'
the packet rssi value: -51dBm
the current noise rssi value: -92dBm
receive message from node address with frequence 11,868.125MHz
message is b'Pls check rssi val.'
the packet rssi value: -49dBm
the current noise rssi value: -95dBm
```

2 Assignment

2.1 Procedure

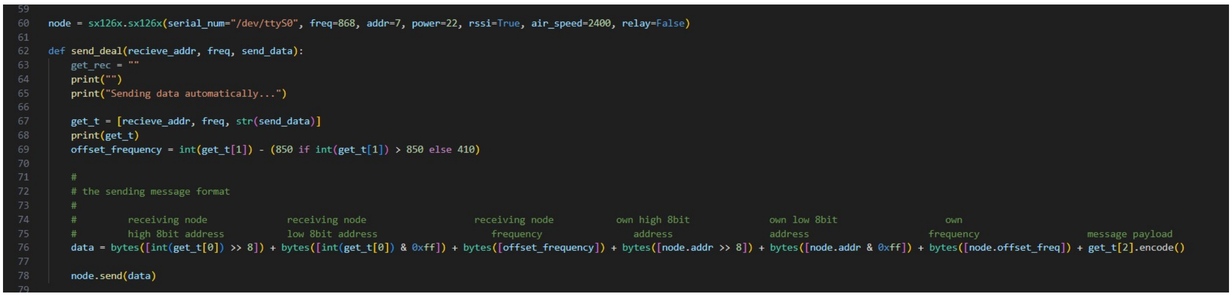
2.1.1 Part-1: LiDAR Data Transmission (From Team A to Team B)

- Import necessary libraries for LIDAR sensor. Set the address and frequency of LORA module.



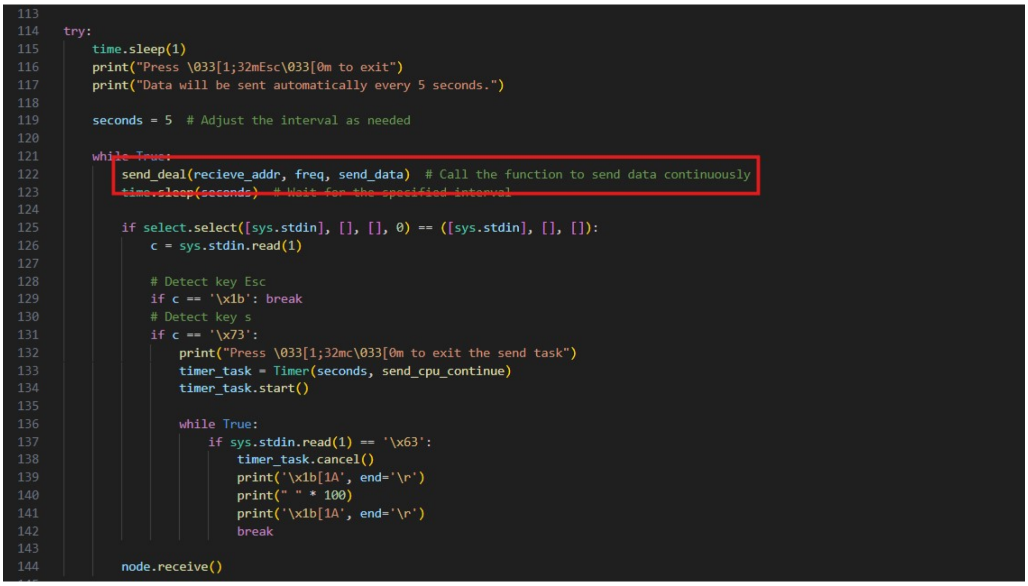
```python
import board
import busio
import adafruit_vl53l0x

old_settings = termios.tcgetattr(sys.stdin)
tty.setcbreak(sys.stdin.fileno())

i2c = busio.I2C(board.SCL, board.SDA)
vl53 = adafruit_vl53l0x.VL53L0X(i2c)
send_data = vl53.range
freq = 868
recieve_addr = 11
```

- Change `get_t` object in the code in `send_deal()` function.



```python

node = sx126x.sx126x(serial_num="/dev/ttyS0", freq=868, addr=7, power=22, rssi=True, air_speed=2400, relay=False)

def send_deal(recieve_addr, freq, send_data):
    get_rec = ""
    print("")
    print("Sending data automatically...")

    get_t = [recieve_addr, freq, str(send_data)]
    print(get_t)
    offset_frequency = int(get_t[1]) - (850 if int(get_t[1]) > 850 else 410)

    #
    # the sending message format
    #
    #         receiving node              receiving node                   receiving node           own high 8bit           own low 8bit                 own
    #         high 8bit address           low 8bit address                    frequency                address              address                   frequency                  message payload
    data = bytes([int(get_t[0]) >> 8]) + bytes([int(get_t[0]) & 0xff]) + bytes([offset_frequency]) + bytes([node.addr >> 8]) + bytes([node.addr & 0xff]) + bytes([node.offset_freq]) + get_t[2].encode()

    node.send(data)

```

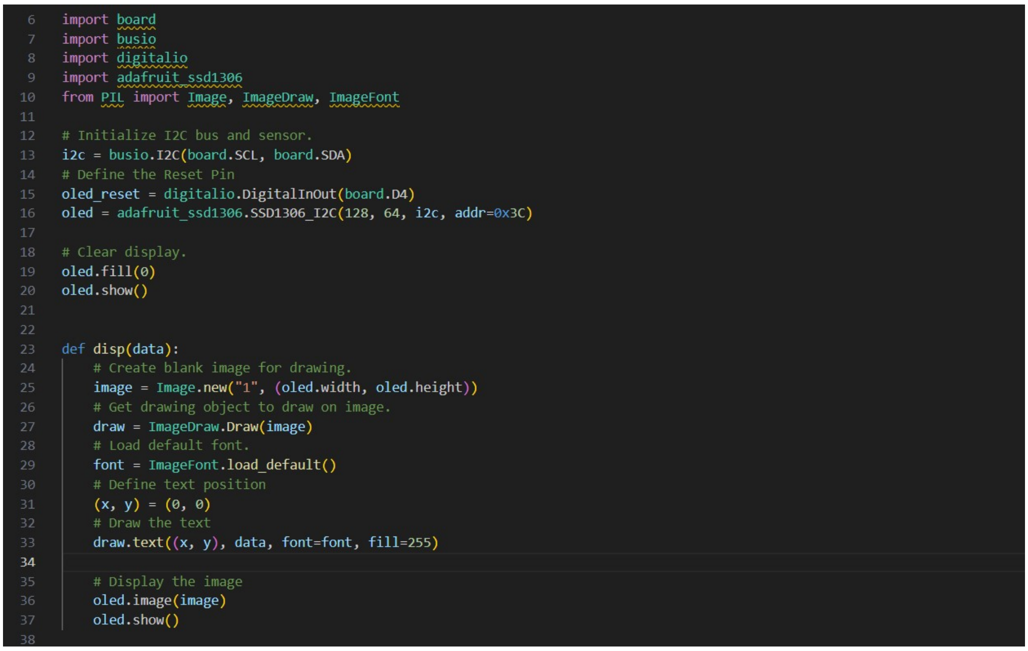
- Call the `send_deal()` function.



```
try:
    time.sleep(1)
    print("Press \033[1;32mEsc\033[0m to exit")
    print("Data will be sent automatically every 5 seconds.")

    seconds = 5  # Adjust the interval as needed

    ~~while True:~~
        send_deal(recieve_addr, freq, send_data)  # Call the function to send data continuously
        ~~time.sleep(seconds)  # Wait for the specified interval~~

        if select.select([sys.stdin], [], [], 0) == ([sys.stdin], [], []):
            c = sys.stdin.read(1)

            # Detect key Esc
            if c == '\x1b': break
            # Detect key s
            if c == '\x73':
                print("Press \033[1;32mc\033[0m to exit the send task")
                timer_task = Timer(seconds, send_cpu_continue)
                timer_task.start()

                while True:
                    if sys.stdin.read(1) == '\x63':
                        timer_task.cancel()
                        print('\x1b[1A', end='\r')
                        print(" " * 100)
                        print('\x1b[1A', end='\r')
                        break

        node.receive()
```

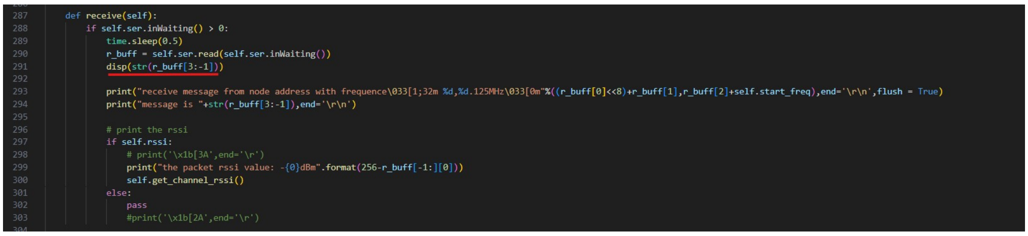
2.1.2 Part-2: Team B displays the received data on the OLED screen

- Import libraries for OLED Display and the define `disp()` function.



```
import board
import busio
import digitalio
import adafruit_ssd1306
from PIL import Image, ImageDraw, ImageFont

# Initialize I2C bus and sensor.
i2c = busio.I2C(board.SCL, board.SDA)
# Define the Reset Pin
oled_reset = digitalio.DigitalInOut(board.D4)
oled = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c, addr=0x3C)

# Clear display.
oled.fill(0)
oled.show()


def disp(data):
    # Create blank image for drawing.
    image = Image.new("1", (oled.width, oled.height))
    # Get drawing object to draw on image.
    draw = ImageDraw.Draw(image)
    # Load default font.
    font = ImageFont.load_default()
    # Define text position
    (x, y) = (0, 0)
    # Draw the text
    draw.text((x, y), data, font=font, fill=255)

    # Display the image
    oled.image(image)
    oled.show()
```

- Modify the `recieve()` function, where we use buffer to store the data received from Team A.



```
def receive(self):
    if self.ser.inWaiting() > 0:
        time.sleep(0.5)
        r_buff = self.ser.read(self.ser.inWaiting())
        disp(str(r_buff[3:-1]))

        print("receive message from node address with frequence\033[1;32m %d,%d.125MHz\033[0m"%((r_buff[0]<<8)+r_buff[1],r_buff[2]+self.start_freq),end='\r\n',flush = True)
        print("message is "+str(r_buff[3:-1]),end='\r\n')

        # print the rssi
        if self.rssi:
            # print('\x1b[3A',end='\r')
            print("the packet rssi value: -{0}dBm".format(256-r_buff[-1:][0]))
            self.get_channel_rssi()
        else:
            pass
            #print('\x1b[2A',end='\r')
```

2.2 Code

Code Modification as Transmission End (in `main.py`)

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

#
#    this is an UART-LoRa device and there is a firmware on the Module
#    users can transfer or receive the data directly by UART and don't
#    need to set parameters like coderate, spread factor, etc.
#    |============================================ |
#    |   It does not support LoRaWAN protocol !!!   |
#    | ============================================|
#
#    This script is mainly for Raspberry Pi 3B+, 4B, and Zero series
#    Since PC/Laptop does not have GPIO to control HAT, it should be configured by
#    GUI and while setting the jumpers,
#    Please refer to another script pc_main.py
#

import sys
import sx126x
import threading
import time
import select
import termios
import tty
from threading import Timer
import time

import board
import busio
import adafruit_vl53l0x

old_settings = termios.tcgetattr(sys.stdin)
tty.setcbreak(sys.stdin.fileno())

i2c = busio.I2C(board.SCL, board.SDA)
vl53 = adafruit_vl53l0x.VL53L0X(i2c)
send_data = vl53.range
freq = 868
recieve_addr = 11


#   serial_num
#       PiZero, Pi3B+, and Pi4B use "/dev/ttyS0"
#
#    Frequency is [850 to 930], or [410 to 493] MHz
#
#    address is 0 to 65535
#        under the same frequency, if set 65535, the node can receive
#        messages from another node of address 0 to 65534 and similarly,
#        the address 0 to 65534 of node can receive messages while
#        the another node of address 65535 sends.
#        otherwise two nodes must be the same address and frequency
#
#    The transmit power is {10, 13, 17, and 22} dBm
#
#    RSSI (receive signal strength indicator) is {True or False}
#        It will print the RSSI value when it receives each message
#

node = sx126x.sx126x(serial_num="/dev/ttyS0", freq=868, addr=7, power=22,
   ↪ rssi=True, air_speed=2400, relay=False)

def send_deal(recieve_addr, freq, send_data):
    get_rec = ""
    print("")
    print("Sending␣data␣automatically...")

    get_t = [recieve_addr, freq, str(send_data)]
    print(get_t)
    offset_frequency = int(get_t[1]) - (850 if int(get_t[1]) > 850 else
       ↪ 410)

    #
    # the sending message format
    #
    #         receiving node                receiving node
       ↪                  receiving node             own high 8bit
       ↪             own low 8bit                    own
    #         high 8bit address             low 8bit address
       ↪                   frequency                 address
       ↪                address                      frequency
       ↪                  message payload
    data = bytes([int(get_t[0]) >> 8]) + bytes([int(get_t[0]) & 0xff]) +
       ↪ bytes([offset_frequency]) + bytes([node.addr >> 8]) + bytes([
       ↪ node.addr & 0xff]) + bytes([node.offset_freq]) + get_t[2].encode
       ↪ ()

    node.send(data)

#    Need to disable the serial login shell and have to enable serial
   ↪ interface
#    command `sudo raspi-config`
#    More details: see https://github.com/MithunHub/LoRa/blob/main/Basic
   ↪ %20Instruction.md
#
#    When the LoRaHAT is attached to RPi, the M0 and M1 jumpers of HAT
   ↪ should be removed.
#

#    The following is to obtain the temperature of the RPi CPU
def get_cpu_temp():
    tempFile = open("/sys/class/thermal/thermal_zone0/temp")
    cpu_temp = tempFile.read()
    tempFile.close()
    return float(cpu_temp) / 1000

def send_cpu_continue(continue_or_not=True):
    if continue_or_not:
        global timer_task
        global seconds
        #
        # broadcast the CPU temperature at 868.125MHz
        #
        data = bytes([255]) + bytes([255]) + bytes([18]) + bytes([255]) + bytes([255]) + bytes([12]) + "CPU Temperature:".encode() + str(get_cpu_temp()).encode() + " C".encode()
        node.send(data)
        time.sleep(0.2)
        timer_task = Timer(seconds, send_cpu_continue)
        timer_task.start()
    else:
        data = bytes([255]) + bytes([255]) + bytes([18]) + bytes([255]) + bytes([255]) + bytes([12]) + "CPU Temperature:".encode() + str(get_cpu_temp()).encode() + " C".encode()
        node.send(data)
        time.sleep(0.2)
        timer_task.cancel()
        pass


try:
    time.sleep(1)
    print("Press \033[1;32mEsc\033[0m to exit")
    print("Data will be sent automatically every 5 seconds.")

    seconds = 5  # Adjust the interval as needed

    while True:
        send_deal(recieve_addr, freq, send_data)  # Call the function to send data continuously
        time.sleep(seconds)  # Wait for the specified interval

        if select.select([sys.stdin], [], [], 0) == ([sys.stdin], [], []):
            c = sys.stdin.read(1)

            # Detect key Esc
            if c == '\x1b': break
            # Detect key s
            if c == '\x73':
                print("Press \033[1;32mc\033[0m to exit the send task")
                timer_task = Timer(seconds, send_cpu_continue)
                timer_task.start()

                while True:
                    if sys.stdin.read(1) == '\x63':
                        timer_task.cancel()
                        print('\x1b[1A', end='\r')
                        print(" " * 100)
                        print('\x1b[1A', end='\r')
                        break

        node.receive()

except:
    termios.tcsetattr(sys.stdin, termios.TCSADRAIN, old_settings)

termios.tcsetattr(sys.stdin, termios.TCSADRAIN, old_settings)
```

[language:python]

Code Modification as Receiver End (in `sx126x.py`)

```
# This file is used for LoRa and Raspberry pi4B related issues
import RPi.GPIO as GPIO
import serial
import time

import board
import busio
import digitalio
import adafruit_ssd1306
from PIL import Image, ImageDraw, ImageFont

# Initialize I2C bus and sensor.
i2c = busio.I2C(board.SCL, board.SDA)
# Define the Reset Pin
oled_reset = digitalio.DigitalInOut(board.D4)
oled = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c, addr=0x3C)

# Clear display.
oled.fill(0)
oled.show()


def disp(data):
    # Create blank image for drawing.
    image = Image.new("1", (oled.width, oled.height))
    # Get drawing object to draw on image.
    draw = ImageDraw.Draw(image)
    # Load default font.
    font = ImageFont.load_default()
    # Define text position
    (x, y) = (0, 0)
    # Draw the text
    draw.text((x, y), data, font=font, fill=255)

    # Display the image
    oled.image(image)
    oled.show()



class sx126x:
    M0 = 22
    M1 = 27
    # if the header is 0xC0, then the LoRa register settings dont lost when it poweroff, and 0xC2 will be lost.
    # cfg_reg = [0xC0,0x00,0x09,0x00,0x00,0x00,0x62,0x00,0x17,0x43,0x00,0x00]
    cfg_reg = [0xC2,0x00,0x09,0x00,0x00,0x00,0x62,0x00,0x12,0x43,0x00,0x00]
    get_reg = bytes(12)
    rssi = False
    addr = 65535
    serial_n = ""
    addr_temp = 0

    #
    # start frequence of two lora module
    #
    # E22-400T22S           E22-900T22S
    # 410~493MHz      or    850~930MHz
    start_freq = 850

    #
    # offset between start and end frequence of two lora module
    #
    # E22-400T22S           E22-900T22S
    # 410~493MHz      or    850~930MHz
    offset_freq = 18

    # power = 22
    # air_speed =2400

    SX126X_UART_BAUDRATE_1200 = 0x00
    SX126X_UART_BAUDRATE_2400 = 0x20
    SX126X_UART_BAUDRATE_4800 = 0x40
    SX126X_UART_BAUDRATE_9600 = 0x60
    SX126X_UART_BAUDRATE_19200 = 0x80
    SX126X_UART_BAUDRATE_38400 = 0xA0
    SX126X_UART_BAUDRATE_57600 = 0xC0
    SX126X_UART_BAUDRATE_115200 = 0xE0

    SX126X_PACKAGE_SIZE_240_BYTE = 0x00
    SX126X_PACKAGE_SIZE_128_BYTE = 0x40
    SX126X_PACKAGE_SIZE_64_BYTE = 0x80
    SX126X_PACKAGE_SIZE_32_BYTE = 0xC0

    SX126X_Power_22dBm = 0x00
    SX126X_Power_17dBm = 0x01
    SX126X_Power_13dBm = 0x02
    SX126X_Power_10dBm = 0x03

    lora_air_speed_dic = {
        1200:0x01,
        2400:0x02,
        4800:0x03,
        9600:0x04,
        19200:0x05,
        38400:0x06,
        62500:0x07
    }

    lora_power_dic = {
        22:0x00,
        17:0x01,
        13:0x02,
        10:0x03
    }

    lora_buffer_size_dic = {
        240:SX126X_PACKAGE_SIZE_240_BYTE,
        128:SX126X_PACKAGE_SIZE_128_BYTE,
        64:SX126X_PACKAGE_SIZE_64_BYTE,
        32:SX126X_PACKAGE_SIZE_32_BYTE
    }

    def __init__(self,serial_num,freq,addr,power,rssi,air_speed=2400,\
                 net_id=0,buffer_size = 240,crypt=0,\
                 relay=False,lbt=False,wor=False):
        self.rssi = rssi
        self.addr = addr
        self.freq = freq
        self.serial_n = serial_num
        self.power = power
        # Initial the GPIO for M0 and M1 Pin
        GPIO.setmode(GPIO.BCM)
        GPIO.setwarnings(False)
        GPIO.setup(self.M0,GPIO.OUT)
        GPIO.setup(self.M1,GPIO.OUT)
        GPIO.output(self.M0,GPIO.LOW)
        GPIO.output(self.M1,GPIO.HIGH)

        # The hardware UART of Pi3B+,Pi4B is /dev/ttyS0
        self.ser = serial.Serial(serial_num,9600)
        self.ser.flushInput()
        self.set(freq,addr,power,rssi,air_speed,net_id,buffer_size,crypt,relay,lbt,wor)

    def set(self,freq,addr,power,rssi,air_speed=2400,\
            net_id=0,buffer_size = 240,crypt=0,\
            relay=False,lbt=False,wor=False):
        self.send_to = addr
        self.addr = addr
        # We should pull up the M1 pin when sets the module
        GPIO.output(self.M0,GPIO.LOW)
        GPIO.output(self.M1,GPIO.HIGH)
        time.sleep(0.1)

        low_addr = addr & 0xff
        high_addr = addr >> 8 & 0xff
        net_id_temp = net_id & 0xff
        if freq > 850:
            freq_temp = freq - 850
            self.start_freq = 850
            self.offset_freq = freq_temp
        elif freq >410:
            freq_temp = freq - 410
            self.start_freq  = 410
            self.offset_freq = freq_temp

        air_speed_temp = self.lora_air_speed_dic.get(air_speed,None)
        # if air_speed_temp != None

        buffer_size_temp = self.lora_buffer_size_dic.get(buffer_size,None)
        # if air_speed_temp != None:

        power_temp = self.lora_power_dic.get(power,None)
        #if power_temp != None:

        if rssi:
            # enable print rssi value
            rssi_temp = 0x80
        else:
            # disable print rssi value
            rssi_temp = 0x00

        # get crypt
        l_crypt = crypt & 0xff
        h_crypt = crypt >> 8 & 0xff

        if relay==False:
            self.cfg_reg[3] = high_addr
            self.cfg_reg[4] = low_addr
            self.cfg_reg[5] = net_id_temp
            self.cfg_reg[6] = self.SX126X_UART_BAUDRATE_9600 +
               ↪ air_speed_temp
            #
            # it will enable to read noise rssi value when add 0x20 as
               ↪ follow
            #
            self.cfg_reg[7] = buffer_size_temp + power_temp + 0x20
            self.cfg_reg[8] = freq_temp
            #
            # it will output a packet rssi value following received
               ↪ message
            # when enable eighth bit with 06H register(rssi_temp = 0x80)
            #
            self.cfg_reg[9] = 0x43 + rssi_temp
            self.cfg_reg[10] = h_crypt
            self.cfg_reg[11] = l_crypt
        else:
            self.cfg_reg[3] = 0x01
            self.cfg_reg[4] = 0x02
            self.cfg_reg[5] = 0x03
            self.cfg_reg[6] = self.SX126X_UART_BAUDRATE_9600 + air_speed_temp
            #
            # it will enable to read noise rssi value when add 0x20 as follow
            #
            self.cfg_reg[7] = buffer_size_temp + power_temp + 0x20
            self.cfg_reg[8] = freq_temp
            #
            # it will output a packet rssi value following received message
            # when enable eighth bit with 06H register(rssi_temp = 0x80)
            #
            self.cfg_reg[9] = 0x03 + rssi_temp
            self.cfg_reg[10] = h_crypt
            self.cfg_reg[11] = l_crypt
        self.ser.flushInput()

        for i in range(2):
            self.ser.write(bytes(self.cfg_reg))
            r_buff = 0
            time.sleep(0.2)
            if self.ser.inWaiting() > 0:
                time.sleep(0.1)
                r_buff = self.ser.read(self.ser.inWaiting())
                if r_buff[0] == 0xC1:
                    pass
                    # print("parameters setting is :",end='')
                    # for i in self.cfg_reg:
                        # print(hex(i),end=' ')

                    # print('\r\n')
                    # print("parameters return is  :",end='')
                    # for i in r_buff:
                        # print(hex(i),end=' ')
                    # print('\r\n')
                else:
                    pass
                    #print("parameters setting fail :",r_buff)
                break
            else:
                print("setting␣fail,setting␣again")
                self.ser.flushInput()
                time.sleep(0.2)
                print('\x1b[1A',end='\r')
                if i == 1:
                    print("setting␣fail,Press␣Esc␣to␣Exit␣and␣run␣again")
                    # time.sleep(2)
                    # print('\x1b[1A',end='\r')

        GPIO.output(self.M0,GPIO.LOW)
        GPIO.output(self.M1,GPIO.LOW)
        time.sleep(0.1)

```

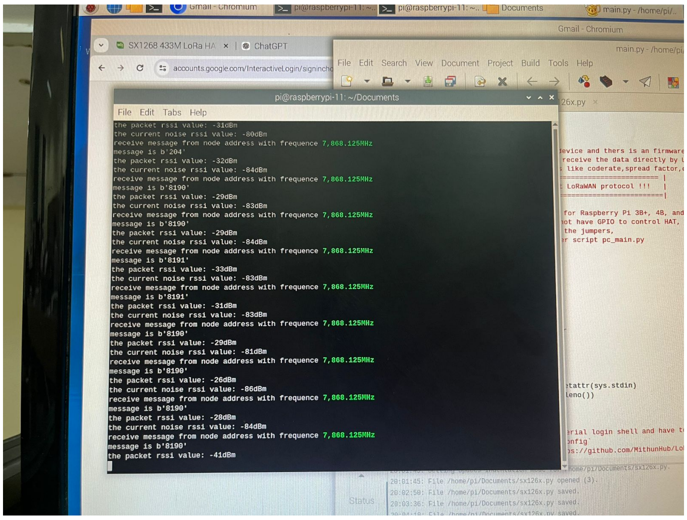
```
    def get_settings(self):
        # the pin M1 of lora HAT must be high when enter setting mode and get parameters
        GPIO.output(M1,GPIO.HIGH)
        time.sleep(0.1)

        # send command to get setting parameters
        self.ser.write(bytes([0xC1,0x00,0x09]))
        if self.ser.inWaiting() > 0:
            time.sleep(0.1)
            self.get_reg = self.ser.read(self.ser.inWaiting())

        # check the return characters from hat and print the setting parameters
        if self.get_reg[0] == 0xC1 and self.get_reg[2] == 0x09:
            fre_temp = self.get_reg[8]
            addr_temp = self.get_reg[3] + self.get_reg[4]
            air_speed_temp = self.get_reg[6] & 0x03
            power_temp = self.get_reg[7] & 0x03

            print("Frequence␣is␣{0}.125MHz.",fre_temp)
            print("Node␣address␣is␣{0}.",addr_temp)
            print("Air␣speed␣is␣{0}␣bps"+ lora_air_speed_dic.get(None,air_speed_temp))
            print("Power␣is␣{0}␣dBm" + lora_power_dic.get(None,power_temp))
            GPIO.output(M1,GPIO.LOW)

#
# the data format like as following
# "node address,frequence,payload"
# "20,868,Hello World"
    def send(self,data):
        GPIO.output(self.M1,GPIO.LOW)
        GPIO.output(self.M0,GPIO.LOW)
        time.sleep(0.1)

        self.ser.write(data)
        # if self.rssi == True:
            # self.get_channel_rssi()
        time.sleep(0.1)


    def receive(self):
        if self.ser.inWaiting() > 0:
            time.sleep(0.5)
            r_buff = self.ser.read(self.ser.inWaiting())
            disp(str(r_buff[3:-1]))

            print("receive␣message␣from␣node␣address␣with␣frequence\033[1;32m␣%d,%d.125MHz\033[0m"%((r_buff[0]<<8)+r_buff[1],r_buff[2]+self.start_freq),end='\r\n',flush = True)
            print("message␣is␣"+str(r_buff[3:-1]),end='\r\n')

```

```python
            # print the rssi
            if self.rssi:
                # print('\x1b[3A',end='\r')
                print("the packet rssi value: -{0}dBm".format(256-r_buff[-1:][0]))
                self.get_channel_rssi()
            else:
                pass
                #print('\x1b[2A',end='\r')

    def get_channel_rssi(self):
        GPIO.output(self.M1,GPIO.LOW)
        GPIO.output(self.M0,GPIO.LOW)
        time.sleep(0.1)
        self.ser.flushInput()
        self.ser.write(bytes([0xC0,0xC1,0xC2,0xC3,0x00,0x02]))
        time.sleep(0.5)
        re_temp = bytes(5)
        if self.ser.inWaiting() > 0:
            time.sleep(0.1)
            re_temp = self.ser.read(self.ser.inWaiting())
        if re_temp[0] == 0xC1 and re_temp[1] == 0x00 and re_temp[2] == 0x02:
            print("the current noise rssi value: -{0}dBm".format(256-re_temp[3]))
            # print("the last receive packet rssi value: -{0}dBm".format(256-re_temp[4]))
        else:
            # pass
            print("receive rssi value fail")
            # print("receive rssi value fail: ",re_temp)
```

[language:python]

2.3 Observations

- LiDAR data was transmitted and received successfully between teams.



```
the packet rssi value: -31dBm
the current noise rssi value: -80dBm
receive message from node address with frequence 7,868.125MHz
message is b'204'
the packet rssi value: -32dBm
the current noise rssi value: -84dBm
receive message from node address with frequence 7,868.125MHz
message is b'8190'
the packet rssi value: -29dBm
the current noise rssi value: -83dBm
receive message from node address with frequence 7,868.125MHz
message is b'8190'
the packet rssi value: -29dBm
the current noise rssi value: -84dBm
receive message from node address with frequence 7,868.125MHz
message is b'8191'
the packet rssi value: -33dBm
the current noise rssi value: -83dBm
receive message from node address with frequence 7,868.125MHz
message is b'8191'
the packet rssi value: -31dBm
the current noise rssi value: -83dBm
receive message from node address with frequence 7,868.125MHz
message is b'8190'
the packet rssi value: -29dBm
the current noise rssi value: -81dBm
receive message from node address with frequence 7,868.125MHz
message is b'8190'
the packet rssi value: -26dBm
the current noise rssi value: -86dBm
receive message from node address with frequence 7,868.125MHz
message is b'8190'
the packet rssi value: -28dBm
the current noise rssi value: -84dBm
receive message from node address with frequence 7,868.125MHz
message is b'8190'
the packet rssi value: -41dBm
```

```
20:01:45: File /home/pi/Documents/sx126x.py opened (3).
20:02:50: File /home/pi/Documents/sx126x.py saved.
20:03:36: File /home/pi/Documents/sx126x.py saved.
```

pi@raspberrypi-11: ~/Documents
File Edit Tabs Help
SX1268 433M LoRa HA
ChatGPT
accounts.google.com/InteractiveLogin/signinch
File Edit Search View Document Project Build Tools Help
Gmail - Chromium
Documents
Status

- The OLED display correctly showed the LiDAR data.