Amazing Race 2022

Problems Document

Index

Problem 1: BME 280 Temperature/Humidity/Pressure

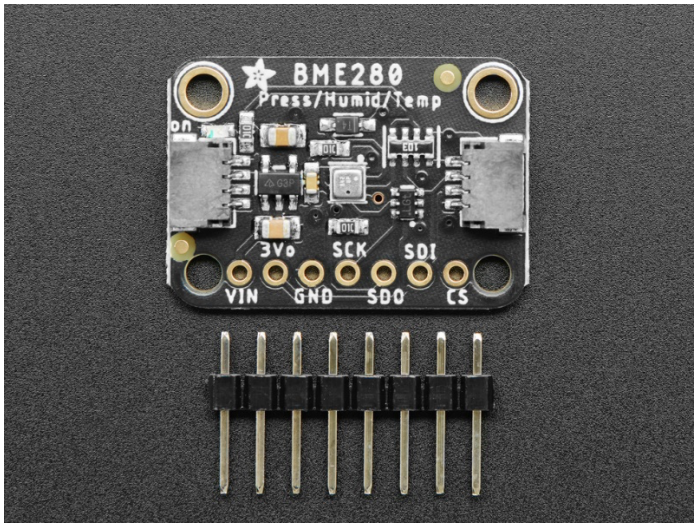

Figure 1: BME280 (adafruit.com)

Introduction:

The BME280 sensor is a low-cost environmental sensor measuring temperature, barometric pressure, and humidity. It supports both an I2C and SPI interfacing option but in this tutorial, we will focus on I2C usage and connection specifics.

Please see FAQ for common troubleshooting assistance.

Hardware Installation:

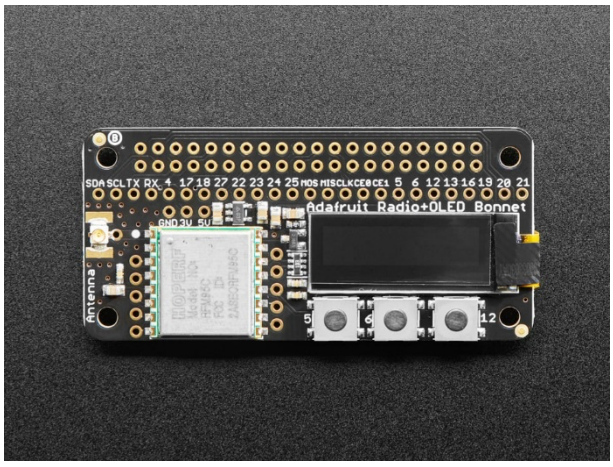

Figure 2.0: LoRa Radio Hat Top (adafruit.com)

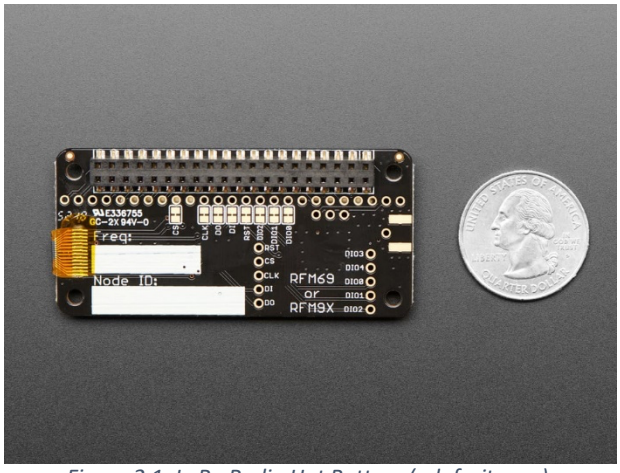

Figure 2.1: LoRa Radio Hat Bottom (adafruit.com)

In this setup it is recommended that you work with a breadboard and prototyping wire to provide the most solid connection without requiring a soldered header.

LoRa Radio Hat Connection:

The installation of the LoRa Radio hat is very straight-forward as it comes with pre-soldered female headers that can be directly installed on the Raspberry PI GPIO.

Verify that the OLED screen is facing upwards and install the hat by evenly sliding the female headers onto the GPIO pins with a modest touch. **It is important not to force the hat onto the GPIO as you may damage the male pins.** If it seems that too much force is required remove the hat and look at the FAQ for common troubleshooting help.

Once the hat is properly seated, look for the circular UFL connector on the far middle left of the hat. It should be easy to spot as it's the only connector of its kind and it shaped like a circle with a single pin in the middle, there are also two flat conductive strips on either side of it. This is where you will connect the antenna adapter cable. Take the cable connector (this should also be a small circle but with a small conductive pad) and click it into place over the UFL header. This pin and cable connection is very small and thus fragile. **It is important to make sure that the cable does not damage the header.** You can verify that the connection is solid as the cable show move around in a circular motion parallel with the board but not feel loose or fall off. You can slightly pull on the connection to verify its affixed properly

Sensor Connection:

This guide will focus on connecting the sensor via I2C. A general understanding of I2C is expected but not required.

Since we are working with a breadboard, a header has already been soldered onto the pins of the sensor and you may slide it into an open breadboard. **Verify that the pins are on independent terminal lines.**

See below for specific hardware connection steps:

Connection Pins:

- Sensor VIN to Shim 3V
- Sensor GND to Shim GND
- Sensor SCK to Shim SCL
- Sensor SDI to Shim SDA

You must locate the four pins of the LoRa hat as we will be connecting our sensor to the open female pin's slots in the top face of the hat.

Locate the four pins:

- 3V is below the pin slot labeled 17 and the GPIO
    - Verify this is 3V and NOT 5V
- GND is directly beside the 3V
- SCL is directly above the antenna connector
- SDA is directly above the antenna connector

You connect the prototyping wires by sliding one side into the pins slots on the hat and the other into four **different, corresponding** terminal lines of the breadboard. **Verify that the terminal lines are not directly connected and match the same connected terminal lines as the intended sensor pin.**

You should notice that the sensor has a status LED that is lit (often time this LED is Green)

Software Installation and Setup:

Installing External Packages:

We will need a few external packages to support our shim driving code and installation efforts.

Execute the following commands (it is recommended to do so as privileged user “sudo”)

```
sudo apt install git python3-pip -y

sudo apt-get install i2c-tools -y
```

Raspberry Pi Configuration:

The first step in setting up the software of the Raspberry PI is to enable the specific interfaces the hat and sensor expect to communicate over. We can do this by entering the privileged settings with the following command:

```
sudo raspi-config
```

Once in the menu navigate to the “Interfaces” option and then the “I2C” and “SPI” options must be selected and enabled. We are enabling SPI as the LoRa Radio hat communicates via SPI, even if the sensor is being connected via I2C for the hat to properly work we will need the SPI interface enabled.

Shim Main Driving Code Installation:

Prior to the next steps, verify that you login into the Raspberry Pi and navigate to the file location where you want the driving code installed. Make note of the location so that you can traverse the file system to edit file within the codebase.

To install the main driving code, we must utilize the git package to grab the public codebase and download it to our local machine (Raspberry PI in this case). Execute the follow command to do so:

```
git clone https://github.com/ECU-Sensing/Raspi_Zero_Node.git
```

Next, install the python dependencies required to run the application by utilizing the python3 package manager pip3. Execute the following command

```
sudo pip3 install -r Raspi_Zero_Node/requirements.txt
```

The last step of this procedure is to set the correct keys for the specific node. You will find a set of keys assigned to your device that you will need to enter into the keys.py file located in the "Raspi_Zero_Node" directory. It is recommended to use a default text editor such as nano to do so.

Sensor Specific Code Installation:

We installed the "esdn-sensing" library to handle the sensor specific implementations of the sensor reading and encoding of data for LoRa transmission. However, we still need to configure the data.py file of the "Raspi_Zero_Node" codebase to reflect our specific sensor needs.

Documentation for the esdn-sensing library is available at: https://ecu-sensing.github.io/esdn_sensing/

Open the data.py file with your favorite/default text editor. The following are items that will need to be changed for each sensor implementation:

- esdn-sensing import line
- get_data(): function
    - sensor_data variable needs to store the correct function call from the correct class

Once the file is configured to the correct sensor class we can move forward to testing. We may return to this section if testing presents errors relative to the application code.

Testing:

It is recommended that you follow the testing procedures in order.

Test Sensor Connection:

We will utilize an additional function from the esdn-sensing library to test the functionality of the sensor. Where originally we wanted to gather the encoding byte array from the sensor class this time we will utilize a simple pre-defined test function that will run a short sample with the sensor and print the output to the standard output. To use the test function we will need to create a new python script, it can be stored anywhere on the local machine. Open your favorite/default text editor and give the script a short name (such as 'test.py'). Follow these steps to write the script:

1. Import esdn-sensing specific sensor class
2. Include simple try except structure to check for SensorError
3. Call the test() function from the specific sensor class (Hint: not the get_data()

```
from esdn_sensing import Hydros, SensorError
import logging
import sys

#//=======================================
def get_data():
    try:
        sensor_data = Hydros().get_data()
    except (FunctionTimedOut, SensorError) as err:
        exception = 3
        print(err)

    return sensor_data
#//=======================================
```

Figure 3: Python Sensor Script Example

You should exit and save the python script and prepare to run the script. You will need to run the test with privledge since the dependencies were installed for the "sudo" user. To run your python script, see the following command as an example:

sudo python3 test.py

Once you execute this command you should watch the console window to monitor the output for sensor verification. The try except trap you set earlier should spring and alert you if the sensor is disconnected or unable to be read. See FAQ for more information on common errors.

Test Main Transmission Application:

Now it is important to test the main transmission application to verify that it runs into no Runtime errors. To execute the main application run the following command:

sudo python3 [insert Raspi_Zero_Node location]/txrx_ttn.py

This python script will handle the entire process of reading the keys from keys.py, receiving the data from data.py and completing the transmission. You should see the LoRa specification fields outputted to the console follow by a 'TX done' line to let you know the transmission is completed. **The script by nature will continue to run in the console until killed as it waits for a downlink**. We will not be using the downlink feature so you can **kill this script after transmission with a Ctrl+C.**

Check TTN for transmission:

To eliminate the possibility that your packets are being lost due to environment factors it is recommended that you run the script at least twice with a few seconds between execution to send multiple packets. You can monitor the TTN console provided to see when your transmission is received.

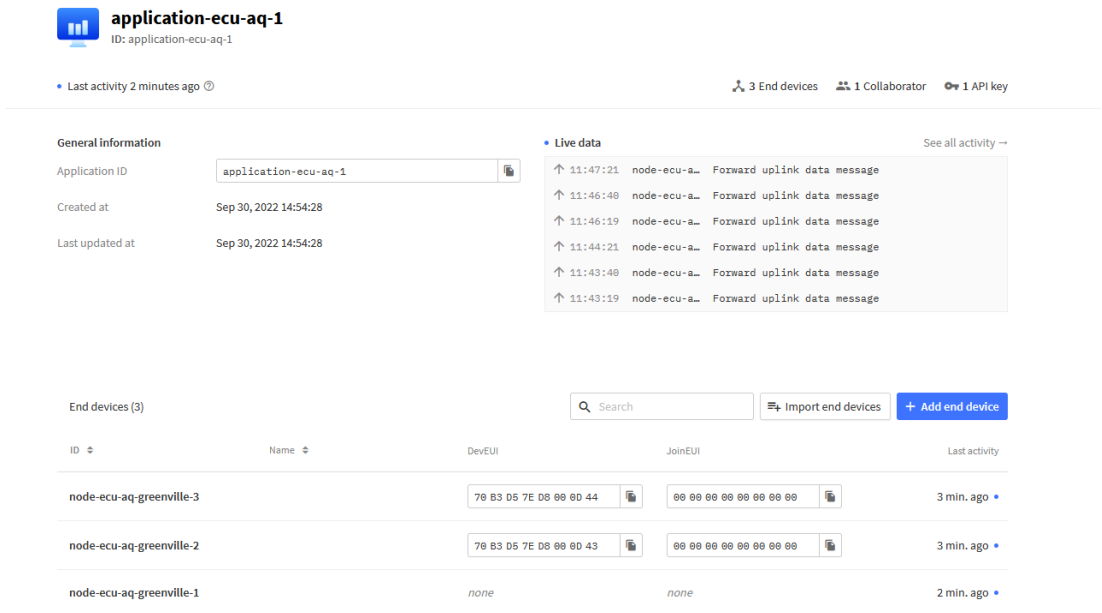

Figure 4: TTN Console with recent received uplinks

You have now successfully read a sensor and transmitted its output via LoRa!!

Problem 2: AHT20 Temperature/Humidity

Figure 5: AHT20 (adafruit.com)

Introduction:

The AHT20 sensor is a low-cost environmental sensor measuring temperature and barometric pressure. It supports both an I2C and SPI interfacing option but in this tutorial, we will focus on I2C usage and connection specifics.

Please see FAQ for common troubleshooting assistance.

Hardware Installation:

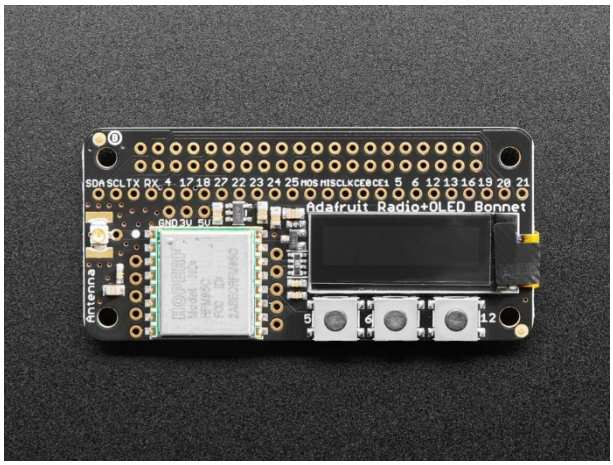

Figure 2.0: LoRa Radio Hat Top (adafruit.com)

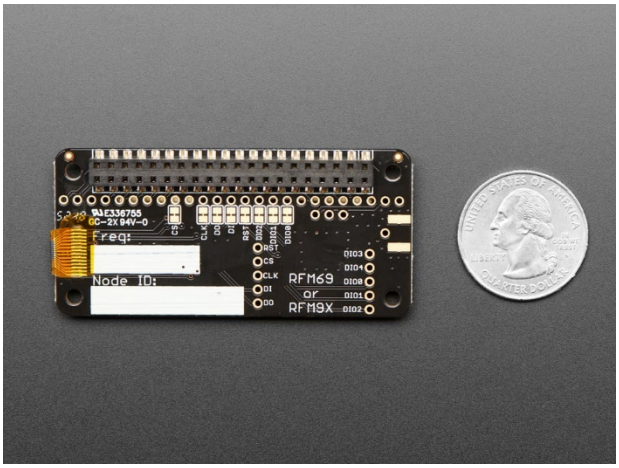

Figure 6.1: LoRa Radio Hat Bottom (adafruit.com)

In this setup it is recommended that you work with a breadboard and prototyping wire to provide the most solid connection without requiring a soldered header.

LoRa Radio Hat Connection:

The installation of the LoRa Radio hat is very straight-forward as it comes with pre-soldered female headers that can be directly installed on the Raspberry PI GPIO.

Verify that the OLED screen is facing upwards and install the hat by evenly sliding the female headers onto the GPIO pins with a modest touch. **It is important not to force the hat onto the GPIO as you may damage the male pins.** If it seems that too much force is required remove the hat and look at the FAQ for common troubleshooting help.

Once the hat is properly seated, look for the circular UFL connector on the far middle left of the hat. It should be easy to spot as it's the only connector of its kind and it shaped like a circle with a single pin in the middle, there are also two flat conductive strips on either side of it. This is where you will connect the antenna adapter cable. Take the cable connector (this should also be a small circle but with a small conductive pad) and click it into place over the UFL header. This pin and cable connection is very small and thus fragile. **It is important to make sure that the cable does not damage the header.** You can verify that the connection is solid as the cable show move around in a circular motion parallel with the board but not feel loose or fall off. You can slightly pull on the connection to verify its affixed properly

Sensor Connection:

This guide will focus on connecting the sensor via I2C. A general understanding of I2C is expected but not required.

Since this sensor is intended to connect via a Stemma QT connector for prototyping; a cable has been provided with one Stemma QT and open prototyping wires. You will still need to utilize the breadboard to make the connections between the Stemma QT cable and the Raspberry Pi. **Verify that the pins are on independent terminal lines.**

See below for specific hardware connection steps:

Connection Pins:

- Stemma RED to Shim 3V
- Stemma BLACK to Shim GND
- Stemma YELLOW SCK to Shim SCL
- Stemma WHITE or BLUE to Shim SDA

You must locate the four pins of the LoRa hat as we will be connecting our sensor to the open female pin's slots in the top face of the hat.

Locate the four pins:

- 3V is below the pin slot labeled 17 and the GPIO
    - Verify this is 3V and NOT 5V
- GND is directly beside the 3V
- SCL is directly above the antenna connector
- SDA is directly above the antenna connector

You connect the prototyping wires by sliding one side into the pins slots on the hat and the other into four **different, corresponding** terminal lines of the breadboard. **Verify that the terminal lines are not directly connected and match the same connected terminal lines as the intended sensor pin.**

You should notice that the sensor has a status LED that is lit (often time this LED is Green)

Software Installation and Setup:

Installing External Packages:

We will need a few external packages to support our shim driving code and installation efforts.

Execute the following commands (it is recommended to do so as privileged user “sudo”)

```
sudo apt install git python3-pip -y

sudo apt-get install i2c-tools -y
```

Raspberry Pi Configuration:

The first step in setting up the software of the Raspberry PI is to enable the specific interfaces the hat and sensor expect to communicate over. We can do this by entering the privileged settings with the following command:

```
sudo raspi-config
```

Once in the menu navigate to the “Interfaces” option and then the “I2C” and “SPI” options must be selected and enabled. We are enabling SPI as the LoRa Radio hat communicates via SPI, even if the sensor is being connected via I2C for the hat to properly work we will need the SPI interface enabled.

Shim Main Driving Code Installation:

Prior to the next steps, verify that you login into the Raspberry Pi and navigate to the file location where you want the driving code installed. Make note of the location so that you can traverse the file system to edit file within the codebase.

To install the main driving code, we must utilize the git package to grab the public codebase and download it to our local machine (Raspberry PI in this case). Execute the follow command to do so:

```
git clone https://github.com/ECU-Sensing/Raspi_Zero_Node.git
```

Next, install the python dependencies required to run the application by utilizing the python3 package manager pip3. Execute the following command

```
sudo pip3 install -r Raspi_Zero_Node/requirements.txt
```

The last step of this procedure is to set the correct keys for the specific node. You will find a set of keys assigned to your device that you will need to enter into the keys.py file located in the "Raspi_Zero_Node" directory. It is recommended to use a default text editor such as nano to do so.

Sensor Specific Code Installation:

We installed the "esdn-sensing" library to handle the sensor specific implementations of the sensor reading and encoding of data for LoRa transmission. However, we still need to configure the data.py file of the "Raspi_Zero_Node" codebase to reflect our specific sensor needs.

**Documentation for the esdn-sensing library is available at: https://ecu-sensing.github.io/esdn_sensing/*

Open the data.py file with your favorite/default text editor. The following are items that will need to be changed for each sensor implementation:

- esdn-sensing import line
- get_data(): function
    - sensor_data variable needs to store the correct function call from the correct class

Once the file is configured to the correct sensor class we can move forward to testing. We may return to this section if testing presents errors relative to the application code.

Testing:

It is recommended that you follow the testing procedures in order.

Test Sensor Connection:

We will utilize an additional function from the esdn-sensing library to test the functionality of the sensor. Where originally we wanted to gather the encoding byte array from the sensor class, this time we will utilize a simple pre-defined test function that will run a short sample with the sensor and print the output to the standard output. To use the test function we will need to create a new python script, it can be stored anywhere on the local machine. Open your favorite/default text editor and give the script a short name (such as 'test.py'). Follow these steps to write the script:

4. Import esdn-sensing specific sensor class
5. Include simple try except structure to check for SensorError
6. Call the test() function from the specific sensor class (Hint: not the get_data()

```
from esdn_sensing import Hydros, SensorError
import logging
import sys

#//=======================================
def get_data():
    try:
        sensor_data = Hydros().get_data()
    except (FunctionTimedOut, SensorError) as err:
        exception = 3
        print(err)

    return sensor_data
#//=======================================
```

Figure 7: Python Sensor Script Example

You should exit and save the python script and prepare to run the script. You will need to run the test with privledge since the dependencies were installed for the "sudo" user. To run your python script, see the following command as an example:

sudo python3 test.py

Once you execute this command you should watch the console window to monitor the output for sensor verification. The try except trap you set earlier should spring and alert you if the sensor is disconnected or unable to be read. See FAQ for more information on common errors.

Test Main Transmission Application:

Now it is important to test the main transmission application to verify that it runs into no Runtime errors. To execute the main application run the following command:

sudo python3 [insert Raspi_Zero_Node location]/txrx_ttn.py

This python script will handle the entire process of reading the keys from keys.py, receiving the data from data.py and completing the transmission. You should see the LoRa specification fields outputted to the console follow by a 'TX done' line to let you know the transmission is completed. **The script by nature will continue to run in the console until killed as it waits for a downlink**. We will not be using the downlink feature so you can **kill this script after transmission with a Ctrl+C.**

Check TTN for transmission:

To eliminate the possibility that your packets are being lost due to environment factors it is recommended that you run the script at least twice with a few seconds between execution to send multiple packets. You can monitor the TTN console provided to see when your transmission is received.

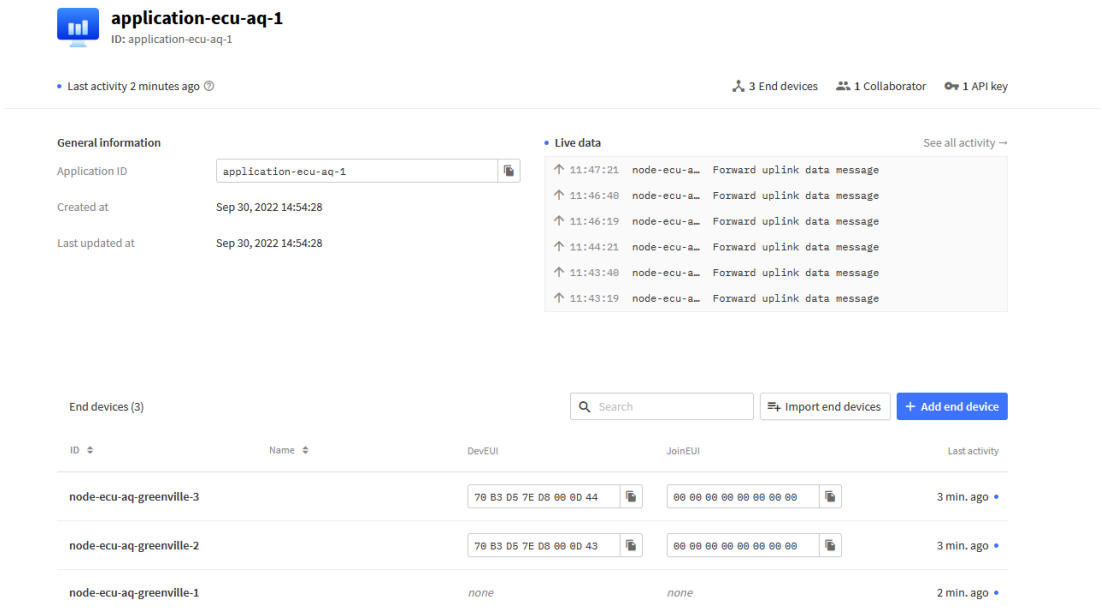

Figure 8: TTN Console with recent received uplinks

You have now successfully read a sensor and transmitted its output via LoRa!!

Problem 3: Soil Moisture Sensor

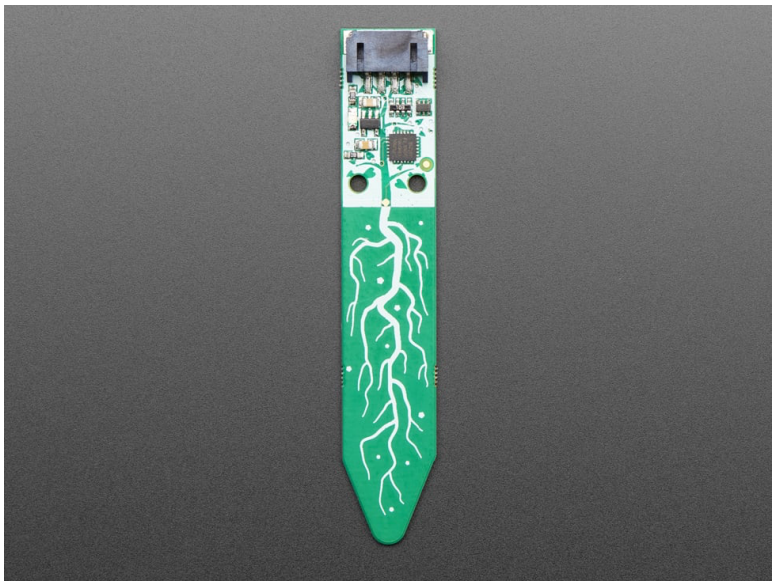

Figure 9: Stemma Soil Sensor (adafruit.com)

Introduction:

The Stemma Soil Sensor is a soil moisture sensor that measures temperature and capacitive soil moisture. It supports both an I2C and SPI interfacing option but in this tutorial, we will focus on I2C usage and connection specifics.

Please see FAQ for common troubleshooting assistance.

Hardware Installation:

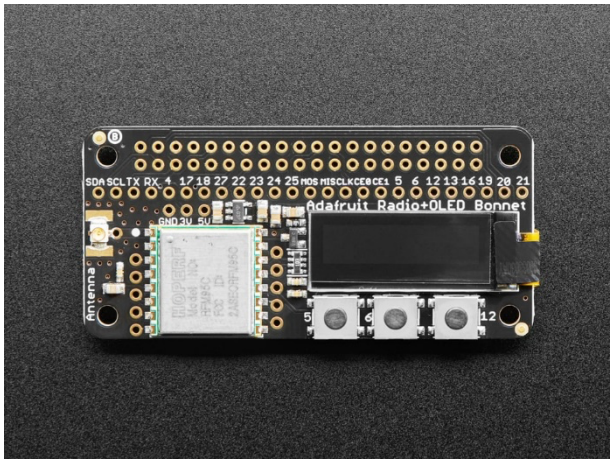

Figure 2.0: LoRa Radio Hat Top (adafruit.com)

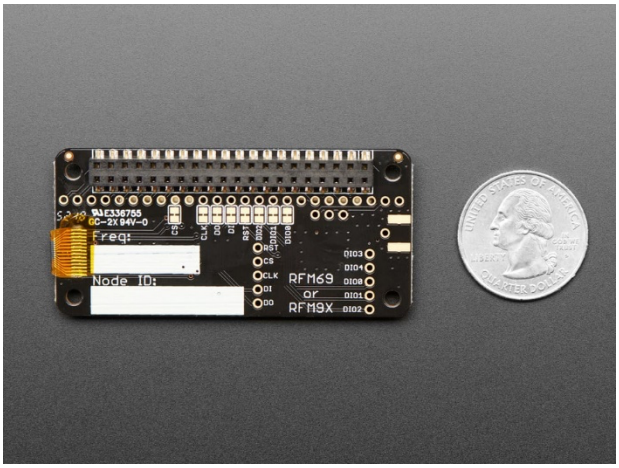

Figure 10.1: LoRa Radio Hat Bottom (adafruit.com)

In this setup it is recommended that you work with a breadboard and prototyping wire to provide the most solid connection without requiring a soldered header.

LoRa Radio Hat Connection:

The installation of the LoRa Radio hat is very straight-forward as it comes with pre-soldered female headers that can be directly installed on the Raspberry PI GPIO.

Verify that the OLED screen is facing upwards and install the hat by evenly sliding the female headers onto the GPIO pins with a modest touch. **It is important not to force the hat onto the GPIO as you may damage the male pins.** If it seems that too much force is required remove the hat and look at the FAQ for common troubleshooting help.

Once the hat is properly seated, look for the circular UFL connector on the far middle left of the hat. It should be easy to spot as it's the only connector of its kind and it shaped like a circle with a single pin in the middle, there are also two flat conductive strips on either side of it. This is where you will connect the antenna adapter cable. Take the cable connector (this should also be a small circle but with a small conductive pad) and click it into place over the UFL header. This pin and cable connection is very small and thus fragile. **It is important to make sure that the cable does not damage the header.** You can verify that the connection is solid as the cable show move around in a circular motion parallel with the board but not feel loose or fall off. You can slightly pull on the connection to verify its affixed properly

Sensor Connection:

This guide will focus on connecting the sensor via I2C. A general understanding of I2C is expected but not required.

Since this sensor does not come with open pins and is intended to connect via a Stemma QT connector; a cable has been provided with one Stemma QT and open prototyping wires. You will still need to utilize the breadboard to make the connections between the Stemma QT cable and the Raspberry Pi. **Verify that the pins are on independent terminal lines.**

See below for specific hardware connection steps:

Connection Pins:

- Stemma RED to Shim 3V
- Stemma BLACK to Shim GND
- Stemma YELLOW SCK to Shim SCL
- Stemma WHITE or BLUE to Shim SDA

You must locate the four pins of the LoRa hat as we will be connecting our sensor to the open female pin's slots in the top face of the hat.

Locate the four pins:

- 3V is below the pin slot labeled 17 and the GPIO
    - Verify this is 3V and NOT 5V
- GND is directly beside the 3V
- SCL is directly above the antenna connector
- SDA is directly above the antenna connector

You connect the prototyping wires by sliding one side into the pins slots on the hat and the other into four **different, corresponding** terminal lines of the breadboard. **Verify that the terminal lines are not directly connected and match the same connected terminal lines as the intended sensor pin.**

You should notice that the sensor has a status LED that is lit (often time this LED is Green)

Software Installation and Setup:

Installing External Packages:

We will need a few external packages to support our shim driving code and installation efforts.

Execute the following commands (it is recommended to do so as privileged user “sudo”)

sudo apt install git python3-pip -y

sudo apt-get install i2c-tools -y

Raspberry Pi Configuration:

The first step in setting up the software of the Raspberry PI is to enable the specific interfaces the hat and sensor expect to communicate over. We can do this by entering the privileged settings with the following command:

sudo raspi-config

Once in the menu navigate to the “Interfaces” option and then the “I2C” and “SPI” options must be selected and enabled. We are enabling SPI as the LoRa Radio hat communicates via SPI, even if the sensor is being connected via I2C for the hat to properly work we will need the SPI interface enabled.

Shim Main Driving Code Installation:

Prior to the next steps, verify that you login into the Raspberry Pi and navigate to the file location where you want the driving code installed. Make note of the location so that you can traverse the file system to edit file within the codebase.

To install the main driving code, we must utilize the git package to grab the public codebase and download it to our local machine (Raspberry PI in this case). Execute the follow command to do so:

```
git clone https://github.com/ECU-Sensing/Raspi_Zero_Node.git
```

Next, install the python dependencies required to run the application by utilizing the python3 package manager pip3. Execute the following command

```
sudo pip3 install -r Raspi_Zero_Node/requirements.txt
```

The last step of this procedure is to set the correct keys for the specific node. You will find a set of keys assigned to your device that you will need to enter into the keys.py file located in the "Raspi_Zero_Node" directory. It is recommended to use a default text editor such as nano to do so.

Sensor Specific Code Installation:

We installed the "esdn-sensing" library to handle the sensor specific implementations of the sensor reading and encoding of data for LoRa transmission. However, we still need to configure the data.py file of the "Raspi_Zero_Node" codebase to reflect our specific sensor needs.

***Documentation for the esdn-sensing library is available at: https://ecu-sensing.github.io/esdn_sensing/**

Open the data.py file with your favorite/default text editor. The following are items that will need to be changed for each sensor implementation:

- esdn-sensing import line
- get_data(): function
    - sensor_data variable needs to store the correct function call from the correct class

Once the file is configured to the correct sensor class we can move forward to testing. We may return to this section if testing presents errors relative to the application code.

Testing:

It is recommended that you follow the testing procedures in order.

Test Sensor Connection:

We will utilize an additional function from the esdn-sensing library to test the functionality of the sensor. Where originally we wanted to gather the encoding byte array from the sensor class this time we will utilize a simple pre-defined test function that will run a short sample with the sensor and print the output to the standard output. To use the test function we will need to create a new python script, it can be stored anywhere on the local machine. Open your favorite/default text editor and give the script a short name (such as 'test.py'). Follow these steps to write the script:

7. Import esdn-sensing specific sensor class
8. Include simple try except structure to check for SensorError
9. Call the test() function from the specific sensor class (Hint: not the get_data()

```
from esdn_sensing import Hydros, SensorError
import logging
import sys

#//=======================================
def get_data():
    try:
        sensor_data = Hydros().get_data()
    except (FunctionTimedOut, SensorError) as err:
        exception = 3
        print(err)

    return sensor_data
#//=======================================
```

Figure 11: Python Sensor Script Example

You should exit and save the python script and prepare to run the script. You will need to run the test with privledge since the dependencies were installed for the "sudo" user. To run your python script, see the following command as an example:

sudo python3 test.py

Once you execute this command you should watch the console window to monitor the output for sensor verification. The try except trap you set earlier should spring and alert you if the sensor is disconnected or unable to be read. See FAQ for more information on common errors.

Test Main Transmission Application:

Now it is important to test the main transmission application to verify that it runs into no Runtime errors. To execute the main application run the following command:

sudo python3 [insert Raspi_Zero_Node location]/txrx_ttn.py

This python script will handle the entire process of reading the keys from keys.py, receiving the data from data.py and completing the transmission. You should see the LoRa specification fields outputted to the console follow by a 'TX done' line to let you know the transmission is completed. **The script by nature will continue to run in the console until killed as it waits for a downlink**. We will not be using the downlink feature so you can **kill this script after transmission with a Ctrl+C.**

Check TTN for transmission:

To eliminate the possibility that your packets are being lost due to environment factors it is recommended that you run the script at least twice with a few seconds between execution to send multiple packets. You can monitor the TTN console provided to see when your transmission is received.

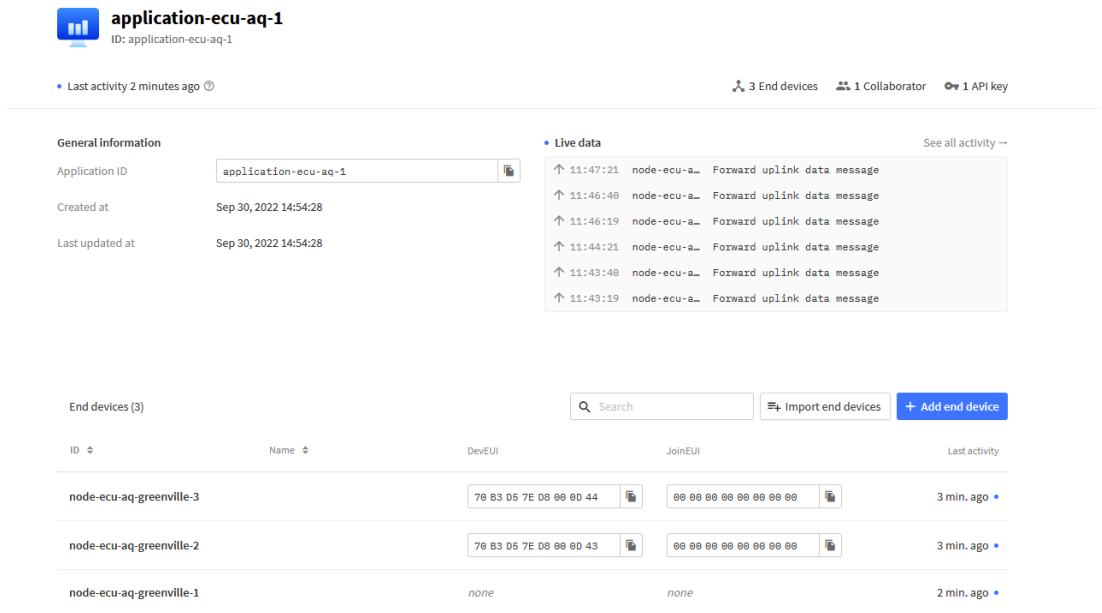

Figure 12: TTN Console with recent received uplinks

You have now successfully read a sensor and transmitted its output via LoRa!!

Problem 4: LT390 Light Sensor

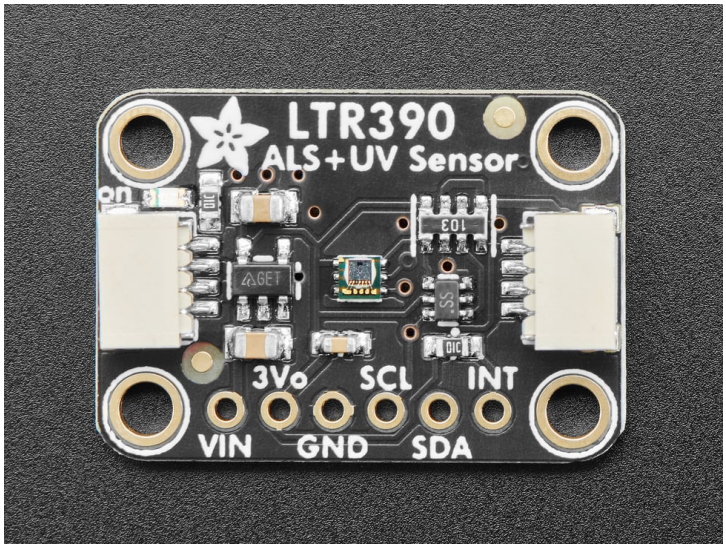

Figure 13: LTR390 (adafruit.com)

Introduction:

The LTR390 sensor is a low-cost environmental sensor measuring ultraviolet and ambient light. It supports both an I2C and SPI interfacing option but in this tutorial, we will focus on I2C usage and connection specifics.

Please see FAQ for common troubleshooting assistance.

Hardware Installation:

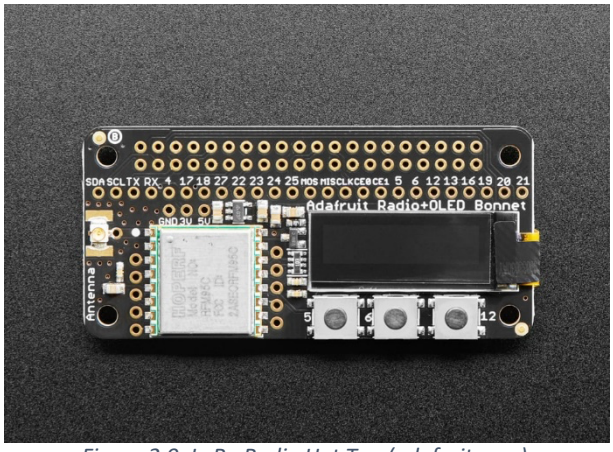

Figure 2.0: LoRa Radio Hat Top (adafruit.com)

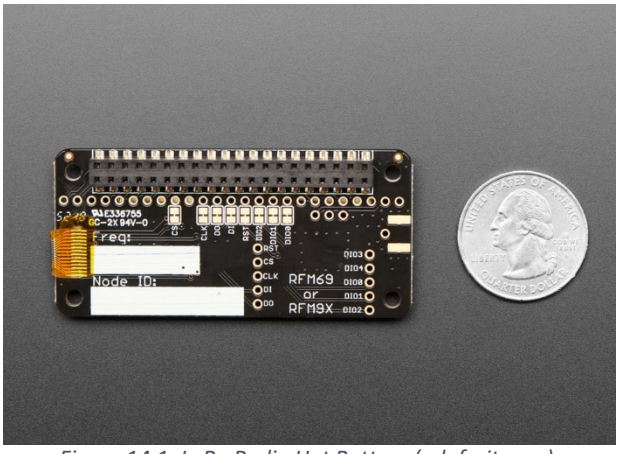

Figure 14.1: LoRa Radio Hat Bottom (adafruit.com)

In this setup it is recommended that you work with a breadboard and prototyping wire to provide the most solid connection without requiring a soldered header.

LoRa Radio Hat Connection:

The installation of the LoRa Radio hat is very straight-forward as it comes with pre-soldered female headers that can be directly installed on the Raspberry PI GPIO.

Verify that the OLED screen is facing upwards and install the hat by evenly sliding the female headers onto the GPIO pins with a modest touch. **It is important not to force the hat onto the GPIO as you may damage the male pins.** If it seems that too much force is required remove the hat and look at the FAQ for common troubleshooting help.

Once the hat is properly seated, look for the circular UFL connector on the far middle left of the hat. It should be easy to spot as it's the only connector of its kind and it shaped like a circle with a single pin in the middle, there are also two flat conductive strips on either side of it. This is where you will connect the antenna adapter cable. Take the cable connector (this should also be a small circle but with a small conductive pad) and click it into place over the UFL header. This pin and cable connection is very small and thus fragile. **It is important to make sure that the cable does not damage the header.** You can verify that the connection is solid as the cable show move around in a circular motion parallel with the board but not feel loose or fall off. You can slightly pull on the connection to verify its affixed properly

Sensor Connection:

This guide will focus on connecting the sensor via I2C. A general understanding of I2C is expected but not required.

Since this sensor does not come with open pins and is intended to connect via a Stemma QT connector; a cable has been provided with one Stemma QT and open prototyping wires. You will still need to utilize the breadboard to make the connections between the Stemma QT cable and the Raspberry Pi. **Verify that the pins are on independent terminal lines.**

See below for specific hardware connection steps:

Connection Pins:

- Stemma RED to Shim 3V
- Stemma BLACK to Shim GND
- Stemma YELLOW SCK to Shim SCL
- Stemma WHITE or BLUE to Shim SDA

You must locate the four pins of the LoRa hat as we will be connecting our sensor to the open female pin's slots in the top face of the hat.

Locate the four pins:

- 3V is below the pin slot labeled 17 and the GPIO
    - Verify this is 3V and NOT 5V
- GND is directly beside the 3V
- SCL is directly above the antenna connector
- SDA is directly above the antenna connector

You connect the prototyping wires by sliding one side into the pins slots on the hat and the other into four **different, corresponding** terminal lines of the breadboard. **Verify that the terminal lines are not directly connected and match the same connected terminal lines as the intended sensor pin.**

You should notice that the sensor has a status LED that is lit (often time this LED is Green)

Software Installation and Setup:

Installing External Packages:

We will need a few external packages to support our shim driving code and installation efforts.

Execute the following commands (it is recommended to do so as privileged user “sudo”)

```
sudo apt install git python3-pip -y

sudo apt-get install i2c-tools -y
```

Raspberry Pi Configuration:

The first step in setting up the software of the Raspberry PI is to enable the specific interfaces the hat and sensor expect to communicate over. We can do this by entering the privileged settings with the following command:

```
sudo raspi-config
```

Once in the menu navigate to the “Interfaces” option and then the “I2C” and “SPI” options must be selected and enabled. We are enabling SPI as the LoRa Radio hat communicates via SPI, even if the sensor is being connected via I2C for the hat to properly work we will need the SPI interface enabled.

Shim Main Driving Code Installation:

Prior to the next steps, verify that you login into the Raspberry Pi and navigate to the file location where you want the driving code installed. Make note of the location so that you can traverse the file system to edit file within the codebase.

To install the main driving code, we must utilize the git package to grab the public codebase and download it to our local machine (Raspberry PI in this case). Execute the follow command to do so:

```
git clone https://github.com/ECU-Sensing/Raspi_Zero_Node.git
```

Next, install the python dependencies required to run the application by utilizing the python3 package manager pip3. Execute the following command

```
sudo pip3 install -r Raspi_Zero_Node/requirements.txt
```

The last step of this procedure is to set the correct keys for the specific node. You will find a set of keys assigned to your device that you will need to enter into the keys.py file located in the "Raspi_Zero_Node" directory. It is recommended to use a default text editor such as nano to do so.

Sensor Specific Code Installation:

We installed the "esdn-sensing" library to handle the sensor specific implementations of the sensor reading and encoding of data for LoRa transmission. However, we still need to configure the data.py file of the "Raspi_Zero_Node" codebase to reflect our specific sensor needs.

Documentation for the esdn-sensing library is available at: https://ecu-sensing.github.io/esdn_sensing/

Open the data.py file with your favorite/default text editor. The following are items that will need to be changed for each sensor implementation:

- esdn-sensing import line
- get_data(): function
    - sensor_data variable needs to store the correct function call from the correct class

Once the file is configured to the correct sensor class we can move forward to testing. We may return to this section if testing presents errors relative to the application code.

Testing:

It is recommended that you follow the testing procedures in order.

Test Sensor Connection:

We will utilize an additional function from the esdn-sensing library to test the functionality of the sensor. Where originally we wanted to gather the encoding byte array from the sensor class this time we will utilize a simple pre-defined test function that will run a short sample with the sensor and print the output to the standard output. To use the test function we will need to create a new python script, it can be stored anywhere on the local machine. Open your favorite/default text editor and give the script a short name (such as 'test.py'). Follow these steps to write the script:

10. Import esdn-sensing specific sensor class
11. Include simple try except structure to check for SensorError
12. Call the test() function from the specific sensor class (Hint: not the get_data()

```
from esdn_sensing import Hydros, SensorError
import logging
import sys

#//=======================================
def get_data():
    try:
        sensor_data = Hydros().get_data()
    except (FunctionTimedOut, SensorError) as err:
        exception = 3
        print(err)

    return sensor_data
#//=======================================
```

Figure 15: Python Sensor Script Example

You should exit and save the python script and prepare to run the script. You will need to run the test with privledge since the dependencies were installed for the "sudo" user. To run your python script, see the following command as an example:

sudo python3 test.py

Once you execute this command you should watch the console window to monitor the output for sensor verification. The try except trap you set earlier should spring and alert you if the sensor is disconnected or unable to be read. See FAQ for more information on common errors.

Test Main Transmission Application:

Now it is important to test the main transmission application to verify that it runs into no Runtime errors. To execute the main application run the following command:

sudo python3 [insert Raspi_Zero_Node location]/txrx_ttn.py

This python script will handle the entire process of reading the keys from keys.py, receiving the data from data.py and completing the transmission. You should see the LoRa specification fields outputted to the console follow by a 'TX done' line to let you know the transmission is completed. **The script by nature will continue to run in the console until killed as it waits for a downlink**. We will not be using the downlink feature so you can **kill this script after transmission with a Ctrl+C.**

Check TTN for transmission:

To eliminate the possibility that your packets are being lost due to environment factors it is recommended that you run the script at least twice with a few seconds between execution to send multiple packets. You can monitor the TTN console provided to see when your transmission is received.

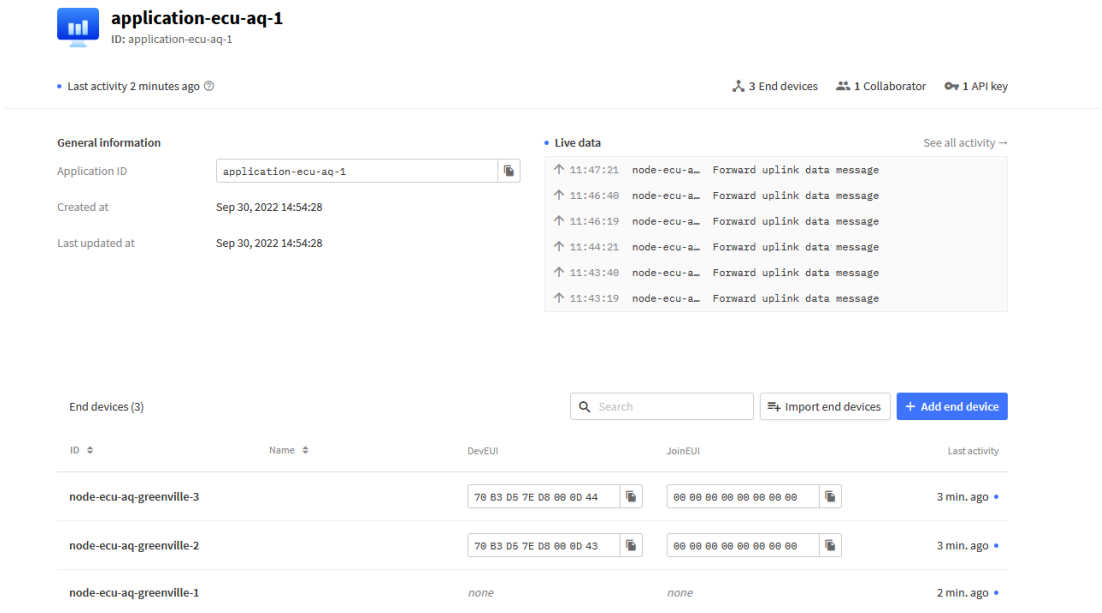

Figure 16: TTN Console with recent received uplinks

You have now successfully read a sensor and transmitted its output via LoRa!!

Problem 5: LTR 303 Visible Light

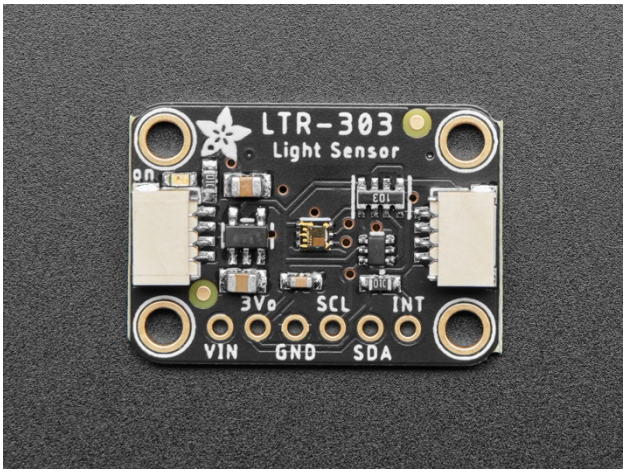

Figure 17: LTR 303 (adafruit.com)

Introduction:

The LTR 303 sensor is a low-cost environmental sensor measuring visible and infrared light. It supports both an I2C and SPI interfacing option but in this tutorial, we will focus on I2C usage and connection specifics.

Please see FAQ for common troubleshooting assistance.

Hardware Installation:

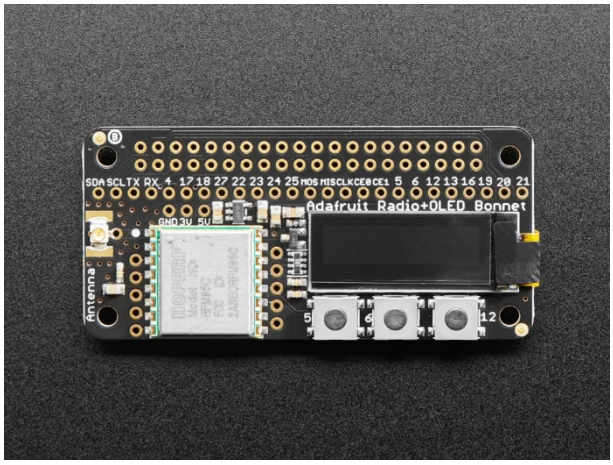

Figure 2.0: LoRa Radio Hat Top (adafruit.com)

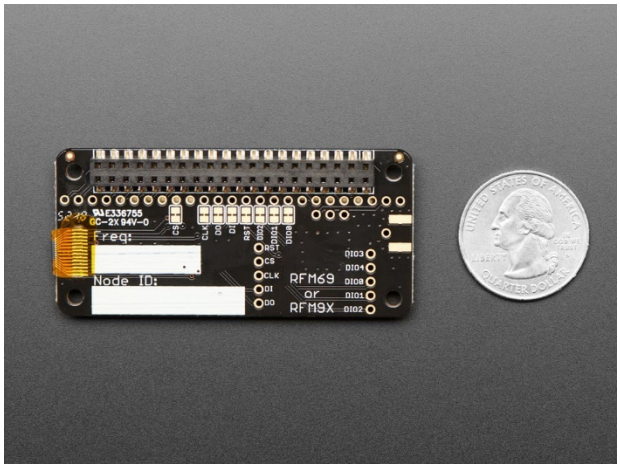

Figure 18.1: LoRa Radio Hat Bottom (adafruit.com)

In this setup it is recommended that you work with a breadboard and prototyping wire to provide the most solid connection without requiring a soldered header.

LoRa Radio Hat Connection:

The installation of the LoRa Radio hat is very straight-forward as it comes with pre-soldered female headers that can be directly installed on the Raspberry PI GPIO.

Verify that the OLED screen is facing upwards and install the hat by evenly sliding the female headers onto the GPIO pins with a modest touch. **It is important not to force the hat onto the GPIO as you may damage the male pins.** If it seems that too much force is required remove the hat and look at the FAQ for common troubleshooting help.

Once the hat is properly seated, look for the circular UFL connector on the far middle left of the hat. It should be easy to spot as it's the only connector of its kind and it shaped like a circle with a single pin in the middle, there are also two flat conductive strips on either side of it. This is where you will connect the antenna adapter cable. Take the cable connector (this should also be a small circle but with a small conductive pad) and click it into place over the UFL header. This pin and cable connection is very small and thus fragile. **It is important to make sure that the cable does not damage the header.** You can verify that the connection is solid as the cable show move around in a circular motion parallel with the board but not feel loose or fall off. You can slightly pull on the connection to verify its affixed properly

Sensor Connection:

This guide will focus on connecting the sensor via I2C. A general understanding of I2C is expected but not required.

Since this sensor does not come with open pins and is intended to connect via a Stemma QT connector; a cable has been provided with one Stemma QT and open prototyping wires. You will still need to utilize the breadboard to make the connections between the Stemma QT cable and the Raspberry Pi. **Verify that the pins are on independent terminal lines.**

See below for specific hardware connection steps:

Connection Pins:

- Stemma RED to Shim 3V
- Stemma BLACK to Shim GND
- Stemma YELLOW SCK to Shim SCL
- Stemma WHITE or BLUE to Shim SDA

You must locate the four pins of the LoRa hat as we will be connecting our sensor to the open female pin's slots in the top face of the hat.

Locate the four pins:

- 3V is below the pin slot labeled 17 and the GPIO
    - Verify this is 3V and NOT 5V
- GND is directly beside the 3V
- SCL is directly above the antenna connector
- SDA is directly above the antenna connector

You connect the prototyping wires by sliding one side into the pins slots on the hat and the other into four **different, corresponding** terminal lines of the breadboard. **Verify that the terminal lines are not directly connected and match the same connected terminal lines as the intended sensor pin.**

You should notice that the sensor has a status LED that is lit (often time this LED is Green)

Software Installation and Setup:

Installing External Packages:

We will need a few external packages to support our shim driving code and installation efforts.

Execute the following commands (it is recommended to do so as privileged user “sudo”)

sudo apt install git python3-pip -y

sudo apt-get install i2c-tools -y

Raspberry Pi Configuration:

The first step in setting up the software of the Raspberry PI is to enable the specific interfaces the hat and sensor expect to communicate over. We can do this by entering the privileged settings with the following command:

sudo raspi-config

Once in the menu navigate to the “Interfaces” option and then the “I2C” and “SPI” options must be selected and enabled. We are enabling SPI as the LoRa Radio hat communicates via SPI, even if the sensor is being connected via I2C for the hat to properly work we will need the SPI interface enabled.

Shim Main Driving Code Installation:

Prior to the next steps, verify that you login into the Raspberry Pi and navigate to the file location where you want the driving code installed. Make note of the location so that you can traverse the file system to edit file within the codebase.

To install the main driving code, we must utilize the git package to grab the public codebase and download it to our local machine (Raspberry PI in this case). Execute the follow command to do so:

```
git clone https://github.com/ECU-Sensing/Raspi_Zero_Node.git
```

Next, install the python dependencies required to run the application by utilizing the python3 package manager pip3. Execute the following command

```
sudo pip3 install -r Raspi_Zero_Node/requirements.txt
```

The last step of this procedure is to set the correct keys for the specific node. You will find a set of keys assigned to your device that you will need to enter into the keys.py file located in the "Raspi_Zero_Node" directory. It is recommended to use a default text editor such as nano to do so.

Sensor Specific Code Installation:

We installed the "esdn-sensing" library to handle the sensor specific implementations of the sensor reading and encoding of data for LoRa transmission. However, we still need to configure the data.py file of the "Raspi_Zero_Node" codebase to reflect our specific sensor needs.

Documentation for the esdn-sensing library is available at: https://ecu-sensing.github.io/esdn_sensing/

Open the data.py file with your favorite/default text editor. The following are items that will need to be changed for each sensor implementation:

- esdn-sensing import line
- get_data(): function
    - sensor_data variable needs to store the correct function call from the correct class

Once the file is configured to the correct sensor class we can move forward to testing. We may return to this section if testing presents errors relative to the application code.

Testing:

It is recommended that you follow the testing procedures in order.

Test Sensor Connection:

We will utilize an additional function from the esdn-sensing library to test the functionality of the sensor. Where originally we wanted to gather the encoding byte array from the sensor class this time we will utilize a simple pre-defined test function that will run a short sample with the sensor and print the output to the standard output. To use the test function we will need to create a new python script, it can be stored anywhere on the local machine. Open your favorite/default text editor and give the script a short name (such as 'test.py'). Follow these steps to write the script:

13. Import esdn-sensing specific sensor class
14. Include simple try except structure to check for SensorError
15. Call the test() function from the specific sensor class (Hint: not the get_data()

```
from esdn_sensing import Hydros, SensorError
import logging
import sys

#//=======================================
def get_data():
    try:
        sensor_data = Hydros().get_data()
    except (FunctionTimedOut, SensorError) as err:
        exception = 3
        print(err)

    return sensor_data
#//=======================================
```

Figure 19: Python Sensor Script Example

You should exit and save the python script and prepare to run the script. You will need to run the test with privledge since the dependencies were installed for the "sudo" user. To run your python script, see the following command as an example:

sudo python3 test.py

Once you execute this command you should watch the console window to monitor the output for sensor verification. The try except trap you set earlier should spring and alert you if the sensor is disconnected or unable to be read. See FAQ for more information on common errors.

Test Main Transmission Application:

Now it is important to test the main transmission application to verify that it runs into no Runtime errors. To execute the main application run the following command:

Sudo python3 [insert Raspi_Zero_Node location]/txrx_ttn.py

This python script will handle the entire process of reading the keys from keys.py, receiving the data from data.py and completing the transmission. You should see the LoRa specification fields outputted to the console follow by a 'TX done' line to let you know the transmission is completed. **The script by nature will continue to run in the console until killed as it waits for a downlink**. We will not be using the downlink feature so you can **kill this script after transmission with a Ctrl+C.**

Check TTN for transmission:

To eliminate the possibility that your packets are being lost due to environment factors it is recommended that you run the script at least twice with a few seconds between execution to send multiple packets. You can monitor the TTN console provided to see when your transmission is received.

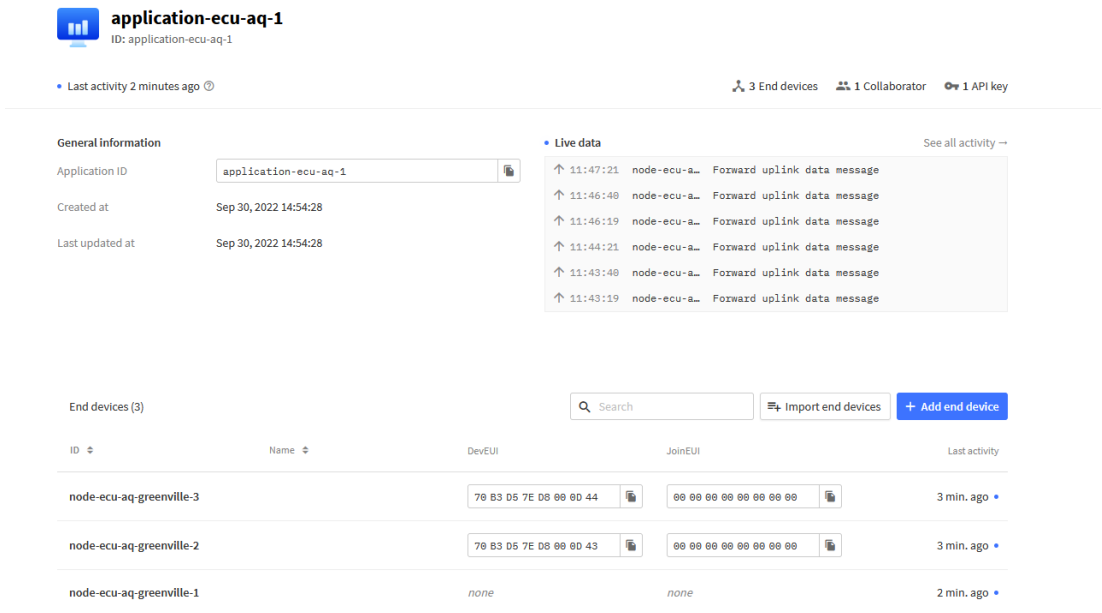

Figure 20: TTN Console with recent received uplinks

You have now successfully read a sensor and transmitted its output via LoRa!!

Problem 6: MiniGPS

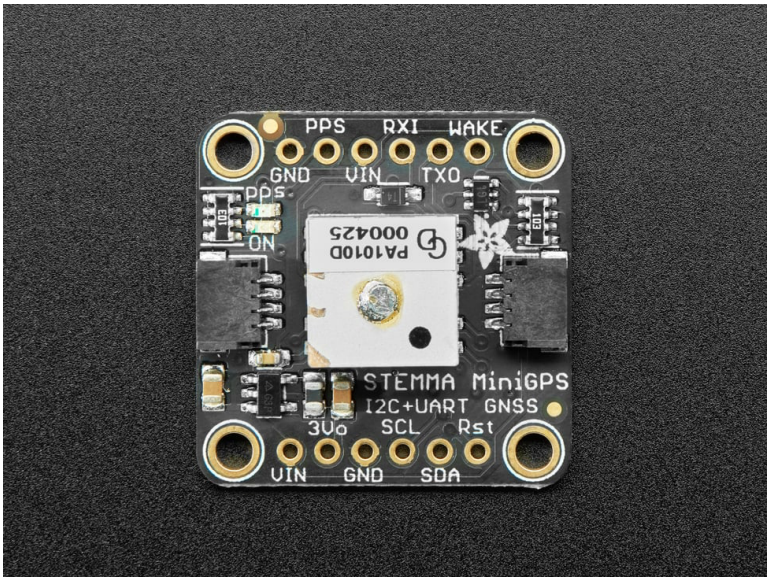

Figure 21: MiniGPS (adafruit.com)

Introduction:

The MiniGPS sensor is a low-cost GPS sensor that reports current longitude, latitude and satellite information. It supports both an I2C and SPI interfacing option but in this tutorial, we will focus on I2C usage and connection specifics.

Please see FAQ for common troubleshooting assistance.

Hardware Installation:

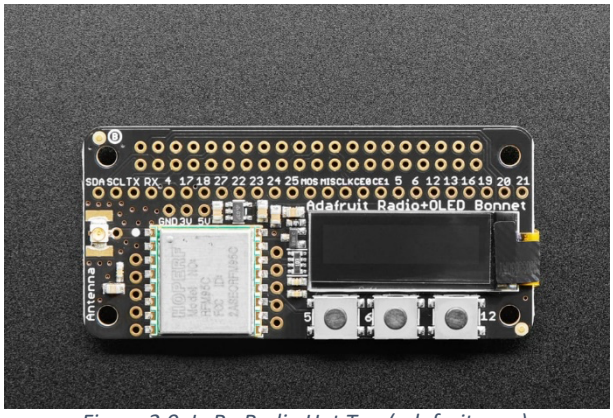

Figure 2.0: LoRa Radio Hat Top (adafruit.com)

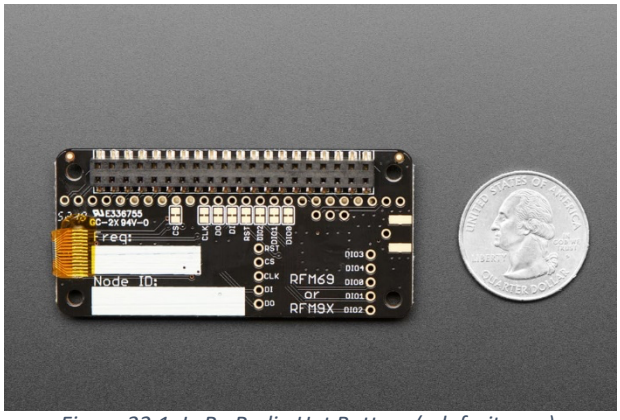

Figure 22.1: LoRa Radio Hat Bottom (adafruit.com)

In this setup it is recommended that you work with a breadboard and prototyping wire to provide the most solid connection without requiring a soldered header.

LoRa Radio Hat Connection:

The installation of the LoRa Radio hat is very straight-forward as it comes with pre-soldered female headers that can be directly installed on the Raspberry PI GPIO.

Verify that the OLED screen is facing upwards and install the hat by evenly sliding the female headers onto the GPIO pins with a modest touch. **It is important not to force the hat onto the GPIO as you may damage the male pins.** If it seems that too much force is required remove the hat and look at the FAQ for common troubleshooting help.

Once the hat is properly seated, look for the circular UFL connector on the far middle left of the hat. It should be easy to spot as it's the only connector of its kind and it shaped like a circle with a single pin in the middle, there are also two flat conductive strips on either side of it. This is where you will connect the antenna adapter cable. Take the cable connector (this should also be a small circle but with a small conductive pad) and click it into place over the UFL header. This pin and cable connection is very small and thus fragile. **It is important to make sure that the cable does not damage the header.** You can verify that the connection is solid as the cable show move around in a circular motion parallel with the board but not feel loose or fall off. You can slightly pull on the connection to verify its affixed properly

Sensor Connection:

This guide will focus on connecting the sensor via I2C. A general understanding of I2C is expected but not required.

Since this sensor does not come with open pins and is intended to connect via a Stemma QT connector; a cable has been provided with one Stemma QT and open prototyping wires. You will still need to utilize the breadboard to make the connections between the Stemma QT cable and the Raspberry Pi. **Verify that the pins are on independent terminal lines.**

See below for specific hardware connection steps:

Connection Pins:

- Stemma RED to Shim 3V
- Stemma BLACK to Shim GND
- Stemma YELLOW SCK to Shim SCL
- Stemma WHITE or BLUE to Shim SDA

You must locate the four pins of the LoRa hat as we will be connecting our sensor to the open female pin's slots in the top face of the hat.

Locate the four pins:

- 3V is below the pin slot labeled 17 and the GPIO
    - Verify this is 3V and NOT 5V
- GND is directly beside the 3V
- SCL is directly above the antenna connector
- SDA is directly above the antenna connector

You connect the prototyping wires by sliding one side into the pins slots on the hat and the other into four **different, corresponding** terminal lines of the breadboard. **Verify that the terminal lines are not directly connected and match the same connected terminal lines as the intended sensor pin.**

You should notice that the sensor has a status LED that is lit (often time this LED is Green)

Software Installation and Setup:

Installing External Packages:

We will need a few external packages to support our shim driving code and installation efforts.

Execute the following commands (it is recommended to do so as privileged user “sudo”)

```
sudo apt install git python3-pip -y

sudo apt-get install i2c-tools -y
```

Raspberry Pi Configuration:

The first step in setting up the software of the Raspberry PI is to enable the specific interfaces the hat and sensor expect to communicate over. We can do this by entering the privileged settings with the following command:

```
sudo raspi-config
```

Once in the menu navigate to the “Interfaces” option and then the “I2C” and “SPI” options must be selected and enabled. We are enabling SPI as the LoRa Radio hat communicates via SPI, even if the sensor is being connected via I2C for the hat to properly work we will need the SPI interface enabled.

Shim Main Driving Code Installation:

Prior to the next steps, verify that you login into the Raspberry Pi and navigate to the file location where you want the driving code installed. Make note of the location so that you can traverse the file system to edit file within the codebase.

To install the main driving code, we must utilize the git package to grab the public codebase and download it to our local machine (Raspberry PI in this case). Execute the follow command to do so:

```
git clone https://github.com/ECU-Sensing/Raspi_Zero_Node.git
```

Next, install the python dependencies required to run the application by utilizing the python3 package manager pip3. Execute the following command

```
sudo pip3 install -r Raspi_Zero_Node/requirements.txt
```

The last step of this procedure is to set the correct keys for the specific node. You will find a set of keys assigned to your device that you will need to enter into the keys.py file located in the "Raspi_Zero_Node" directory. It is recommended to use a default text editor such as nano to do so.

Sensor Specific Code Installation:

We installed the "esdn-sensing" library to handle the sensor specific implementations of the sensor reading and encoding of data for LoRa transmission. However, we still need to configure the data.py file of the "Raspi_Zero_Node" codebase to reflect our specific sensor needs.

Documentation for the esdn-sensing library is available at: https://ecu-sensing.github.io/esdn_sensing/

Open the data.py file with your favorite/default text editor. The following are items that will need to be changed for each sensor implementation:

- esdn-sensing import line
- get_data(): function
    - sensor_data variable needs to store the correct function call from the correct class

Once the file is configured to the correct sensor class we can move forward to testing. We may return to this section if testing presents errors relative to the application code.

Testing:

It is recommended that you follow the testing procedures in order.

Test Sensor Connection:

We will utilize an additional function from the esdn-sensing library to test the functionality of the sensor. Where originally we wanted to gather the encoding byte array from the sensor class this time we will utilize a simple pre-defined test function that will run a short sample with the sensor and print the output to the standard output. To use the test function we will need to create a new python script, it can be stored anywhere on the local machine. Open your favorite/default text editor and give the script a short name (such as 'test.py'). Follow these steps to write the script:

1. Import esdn-sensing specific sensor class
2. Include simple try except structure to check for SensorError
3. Call the test() function from the specific sensor class (Hint: not the get_data()

```
from esdn_sensing import Hydros, SensorError
import logging
import sys

#//=======================================
def get_data():
    try:
        sensor_data = Hydros().get_data()
    except (FunctionTimedOut, SensorError) as err:
        exception = 3
        print(err)

    return sensor_data
#//=======================================
```

Figure 23: Python Sensor Script Example

You should exit and save the python script and prepare to run the script. You will need to run the test with privledge since the dependencies were installed for the "sudo" user. To run your python script, see the following command as an example:

sudo python3 test.py

Once you execute this command you should watch the console window to monitor the output for sensor verification. The try except trap you set earlier should spring and alert you if the sensor is disconnected or unable to be read. See FAQ for more information on common errors.

Test Main Transmission Application:

Now it is important to test the main transmission application to verify that it runs into no Runtime errors. To execute the main application run the following command:

sudo python3 [insert Raspi_Zero_Node location]/txrx_ttn.py

This python script will handle the entire process of reading the keys from keys.py, receiving the data from data.py and completing the transmission. You should see the LoRa specification fields outputted to the console follow by a 'TX done' line to let you know the transmission is completed. **The script by nature will continue to run in the console until killed as it waits for a downlink**. We will not be using the downlink feature so you can **kill this script after transmission with a Ctrl+C.**

Check TTN for transmission:

To eliminate the possibility that your packets are being lost due to environment factors it is recommended that you run the script at least twice with a few seconds between execution to send multiple packets. You can monitor the TTN console provided to see when your transmission is received.

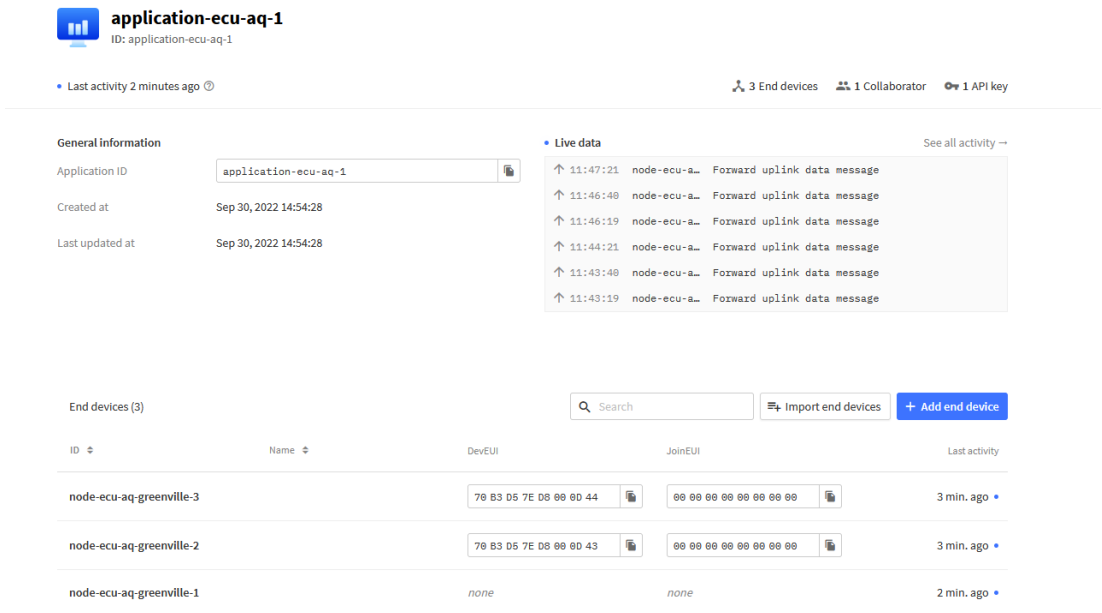

Figure 24: TTN Console with recent received uplinks

You have now successfully read a sensor and transmitted its output via LoRa!!

Problem 8: US 100 Distance Detection Sensor

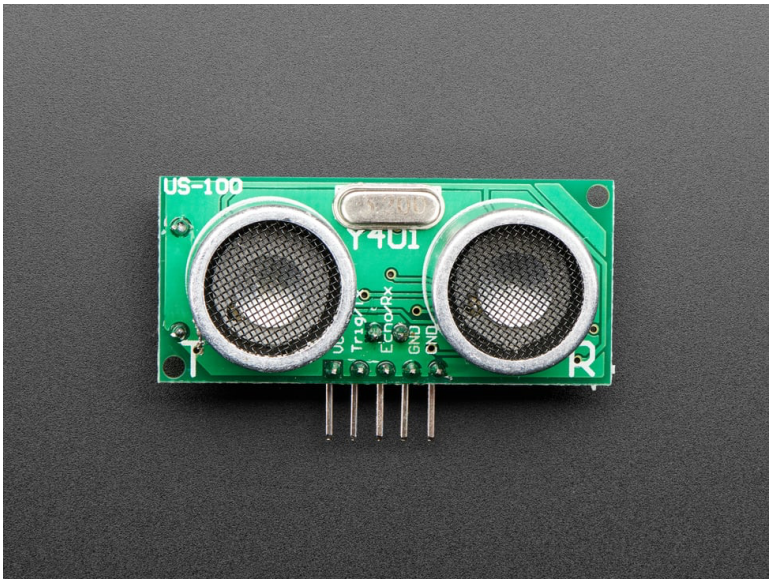

Figure 25: US 100 Sensor (adafruit.com)

Introduction:

The US 100 distance sensor is a low-cost sensor that measures distance from itself in millimeters. It supports both an UART and HC-SR04 interfacing option but in this tutorial, we will focus on UART usage and connection specifics.

Please see FAQ for common troubleshooting assistance.

Hardware Installation:

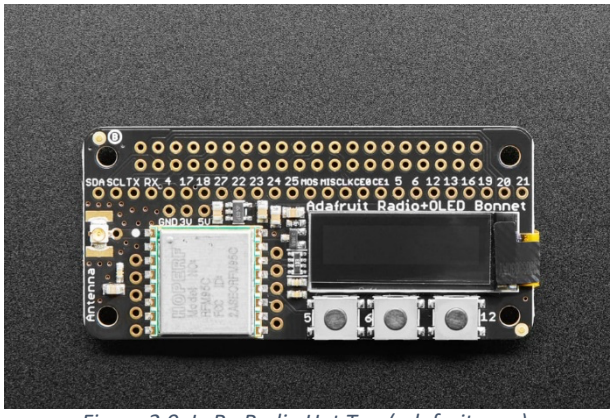

Figure 2.0: LoRa Radio Hat Top (adafruit.com)

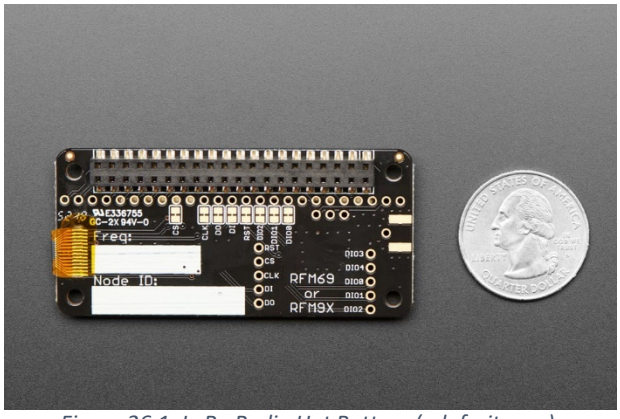

Figure 26.1: LoRa Radio Hat Bottom (adafruit.com)

In this setup it is recommended that you work with a breadboard and prototyping wire to provide the most solid connection without requiring a soldered header.

LoRa Radio Hat Connection:

The installation of the LoRa Radio hat is very straight-forward as it comes with pre-soldered female headers that can be directly installed on the Raspberry PI GPIO.

Verify that the OLED screen is facing upwards and install the hat by evenly sliding the female headers onto the GPIO pins with a modest touch. **It is important not to force the hat onto the GPIO as you may damage the male pins.** If it seems that too much force is required remove the hat and look at the FAQ for common troubleshooting help.

Once the hat is properly seated, look for the circular UFL connector on the far middle left of the hat. It should be easy to spot as it's the only connector of its kind and it shaped like a circle with a single pin in the middle, there are also two flat conductive strips on either side of it. This is where you will connect the antenna adapter cable. Take the cable connector (this should also be a small circle but with a small conductive pad) and click it into place over the UFL header. This pin and cable connection is very small and thus fragile. **It is important to make sure that the cable does not damage the header.** You can verify that the connection is solid as the cable show move around in a circular motion parallel with the board but not feel loose or fall off. You can slightly pull on the connection to verify its affixed properly

Sensor Connection:

This guide will focus on connecting the sensor via UART. A general understanding of UART is expected but not required.

Since we are working with a breadboard and a header comes manufactured with the pins of the sensor and you may slide it into an open breadboard. **Verify that the pins are on independent terminal lines.**

See below for specific hardware connection steps:

Connection Pins:

- Sensor 3V to Shim 3V
- Sensor GND to Shim GND
- Sensor TX to Shim TX
- Sensor RX to Shim RX

You must locate the four pins of the LoRa hat as we will be connecting our sensor to the open female pin's slots in the top face of the hat.

Locate the four pins:

- 3V is below the pin slot labeled 17 and the GPIO
    - Verify this is 3V and NOT 5V
- GND is directly beside the 3V
- TX is above the antenna connector and to the right
- RX is above the antenna connector and to the right

You connect the prototyping wires by sliding one side into the pins slots on the hat and the other into four **different, corresponding** terminal lines of the breadboard. **Verify that the terminal lines are not directly connected and match the same connected terminal lines as the intended sensor pin.**

You should notice that the sensor has a status LED that is lit (often time this LED is Green)

Software Installation and Setup:

Installing External Packages:

We will need a few external packages to support our shim driving code and installation efforts.

Execute the following commands (it is recommended to do so as privileged user “sudo”)

sudo apt install git python3-pip -y

sudo apt-get install i2c-tools -y

Raspberry Pi Configuration:

The first step in setting up the software of the Raspberry PI is to enable the specific interfaces the hat and sensor expect to communicate over. We can do this by entering the privileged settings with the following command:

sudo raspi-config

Once in the menu navigate to the “Interfaces” option and then the “I2C” and “SPI” options must be selected and enabled. We are enabling SPI as the LoRa Radio hat communicates via SPI, even if the sensor is being connected via I2C for the hat to properly work we will need the SPI interface enabled.

Shim Main Driving Code Installation:

Prior to the next steps, verify that you login into the Raspberry Pi and navigate to the file location where you want the driving code installed. Make note of the location so that you can traverse the file system to edit file within the codebase.

To install the main driving code, we must utilize the git package to grab the public codebase and download it to our local machine (Raspberry PI in this case). Execute the follow command to do so:

```
git clone https://github.com/ECU-Sensing/Raspi_Zero_Node.git
```

Next, install the python dependencies required to run the application by utilizing the python3 package manager pip3. Execute the following command

```
sudo pip3 install -r Raspi_Zero_Node/requirements.txt
```

The last step of this procedure is to set the correct keys for the specific node. You will find a set of keys assigned to your device that you will need to enter into the keys.py file located in the "Raspi_Zero_Node" directory. It is recommended to use a default text editor such as nano to do so.

Sensor Specific Code Installation:

We installed the "esdn-sensing" library to handle the sensor specific implementations of the sensor reading and encoding of data for LoRa transmission. However, we still need to configure the data.py file of the "Raspi_Zero_Node" codebase to reflect our specific sensor needs.

Documentation for the esdn-sensing library is available at: https://ecu-sensing.github.io/esdn_sensing/

Open the data.py file with your favorite/default text editor. The following are items that will need to be changed for each sensor implementation:

- esdn-sensing import line
- get_data(): function
    - sensor_data variable needs to store the correct function call from the correct class

Once the file is configured to the correct sensor class we can move forward to testing. We may return to this section if testing presents errors relative to the application code.

Testing:

It is recommended that you follow the testing procedures in order.

Test Sensor Connection:

We will utilize an additional function from the esdn-sensing library to test the functionality of the sensor. Where originally we wanted to gather the encoding byte array from the sensor class this time, we will utilize a simple pre-defined test function that will run a short sample with the sensor and print the output to the standard output. To use the test function, we will need to create a new python script, it can be stored anywhere on the local machine. Open your favorite/default text editor and give the script a short name (such as 'test.py'). Follow these steps to write the script:

1. Import esdn-sensing specific sensor class
2. Include simple try except structure to check for SensorError
3. Call the test() function from the specific sensor class (Hint: not the get_data()

```
from esdn_sensing import Hydros, SensorError
import logging
import sys

#//=======================================
def get_data():
    try:
        sensor_data = Hydros().get_data()
    except (FunctionTimedOut, SensorError) as err:
        exception = 3
        print(err)

    return sensor_data
#//=======================================
```

Figure 27: Python Sensor Script Example

You should exit and save the python script and prepare to run the script. You will need to run the test with privilege since the dependencies were installed for the "sudo" user. To run your python script, see the following command as an example:

sudo python3 test.py

Once you execute this command you should watch the console window to monitor the output for sensor verification. The try except trap you set earlier should spring and alert you if the sensor is disconnected or unable to be read. See FAQ for more information on common errors.

Test Main Transmission Application:

Now it is important to test the main transmission application to verify that it runs into no Runtime errors. To execute the main application run the following command:

sudo python3 [insert Raspi_Zero_Node location]/txrx_ttn.py

This python script will handle the entire process of reading the keys from keys.py, receiving the data from data.py and completing the transmission. You should see the LoRa specification fields outputted to the console follow by a 'TX done' line to let you know the transmission is completed. **The script by nature will continue to run in the console until killed as it waits for a downlink**. We will not be using the downlink feature so you can **kill this script after transmission with a Ctrl+C.**

Check TTN for transmission:

To eliminate the possibility that your packets are being lost due to environment factors it is recommended that you run the script at least twice with a few seconds between execution to send multiple packets. You can monitor the TTN console provided to see when your transmission is received.

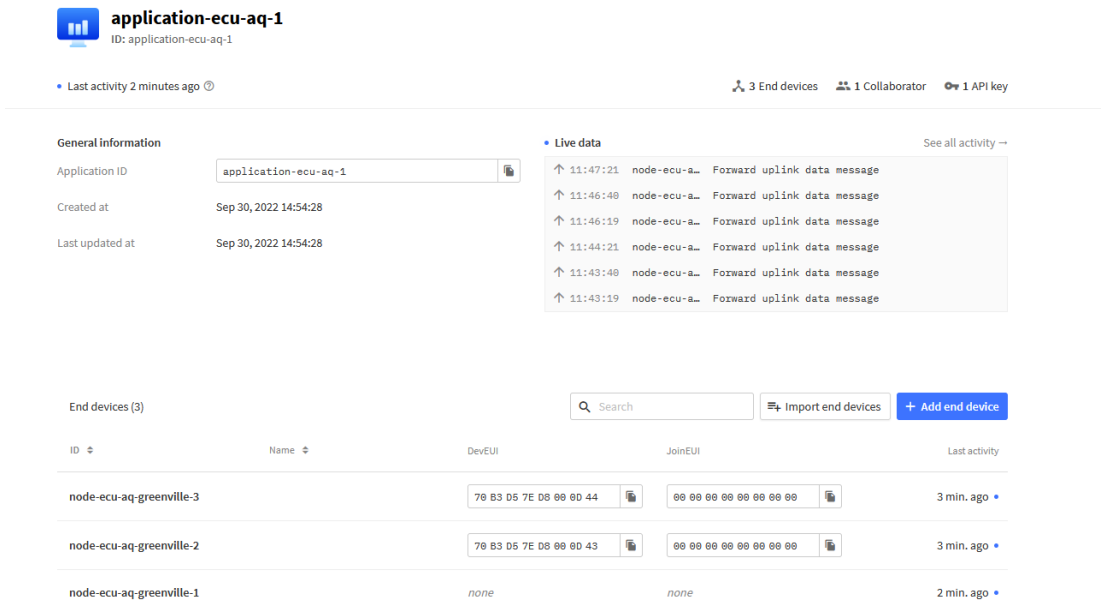

Figure 28: TTN Console with recent received uplinks

You have now successfully read a sensor and transmitted its output via LoRa!!

Problem 8: VL6180x Time of Flight Sensor

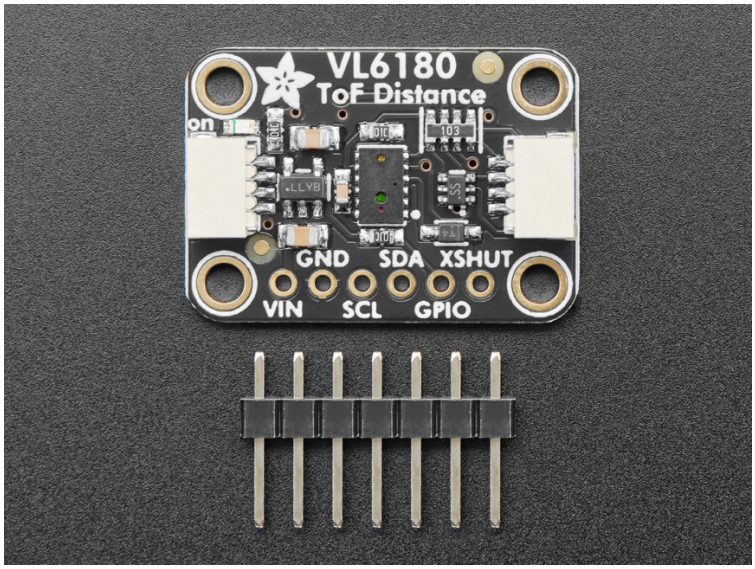

Figure 29: VL6180 (adafruit.com)

Introduction:

The VL6180x sensor is a low-cost laser ranging sensor that measures distance based on the timing of the laser in flight. It supports both an I2C and SPI interfacing option but in this tutorial, we will focus on I2C usage and connection specifics.

Please see FAQ for common troubleshooting assistance.

Hardware Installation:

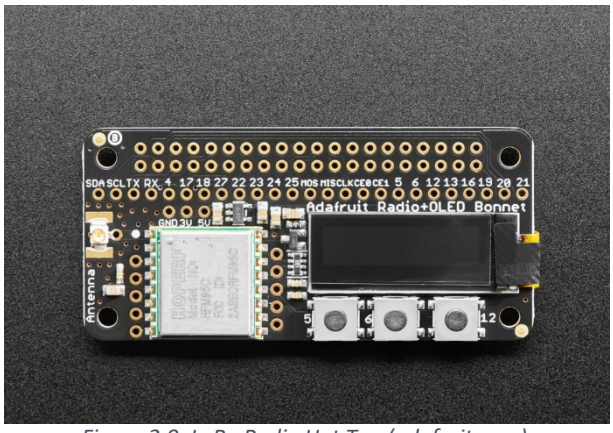

Figure 2.0: LoRa Radio Hat Top (adafruit.com)

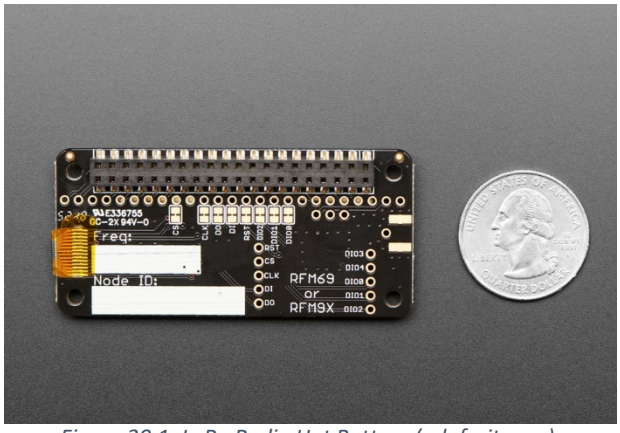

Figure 30.1: LoRa Radio Hat Bottom (adafruit.com)

In this setup it is recommended that you work with a breadboard and prototyping wire to provide the most solid connection without requiring a soldered header.

LoRa Radio Hat Connection:

The installation of the LoRa Radio hat is very straight-forward as it comes with pre-soldered female headers that can be directly installed on the Raspberry PI GPIO.

Verify that the OLED screen is facing upwards and install the hat by evenly sliding the female headers onto the GPIO pins with a modest touch. **It is important not to force the hat onto the GPIO as you may damage the male pins.** If it seems that too much force is required remove the hat and look at the FAQ for common troubleshooting help.

Once the hat is properly seated, look for the circular UFL connector on the far middle left of the hat. It should be easy to spot as it's the only connector of its kind and it shaped like a circle with a single pin in the middle, there are also two flat conductive strips on either side of it. This is where you will connect the antenna adapter cable. Take the cable connector (this should also be a small circle but with a small conductive pad) and click it into place over the UFL header. This pin and cable connection is very small and thus fragile. **It is important to make sure that the cable does not damage the header.** You can verify that the connection is solid as the cable show move around in a circular motion parallel with the board but not feel loose or fall off. You can slightly pull on the connection to verify its affixed properly

Sensor Connection:

This guide will focus on connecting the sensor via I2C. A general understanding of I2C is expected but not required.

Since this sensor does not come with open pins and is intended to connect via a Stemma QT connector; a cable has been provided with one Stemma QT and open prototyping wires. You will still need to utilize the breadboard to make the connections between the Stemma QT cable and the Raspberry Pi. **Verify that the pins are on independent terminal lines.**

See below for specific hardware connection steps:

Connection Pins:

- Stemma RED to Shim 3V
- Stemma BLACK to Shim GND
- Stemma YELLOW SCK to Shim SCL
- Stemma WHITE or BLUE to Shim SDA

You must locate the four pins of the LoRa hat as we will be connecting our sensor to the open female pin's slots in the top face of the hat.

Locate the four pins:

- 3V is below the pin slot labeled 17 and the GPIO
    - Verify this is 3V and NOT 5V
- GND is directly beside the 3V
- SCL is directly above the antenna connector
- SDA is directly above the antenna connector

You connect the prototyping wires by sliding one side into the pins slots on the hat and the other into four **different, corresponding** terminal lines of the breadboard. **Verify that the terminal lines are not directly connected and match the same connected terminal lines as the intended sensor pin.**

You should notice that the sensor has a status LED that is lit (often time this LED is Green)

Software Installation and Setup:

Installing External Packages:

We will need a few external packages to support our shim driving code and installation efforts.

Execute the following commands (it is recommended to do so as privileged user “sudo”)

sudo apt install git python3-pip -y

sudo apt-get install i2c-tools -y

Raspberry Pi Configuration:

The first step in setting up the software of the Raspberry PI is to enable the specific interfaces the hat and sensor expect to communicate over. We can do this by entering the privileged settings with the following command:

sudo raspi-config

Once in the menu navigate to the “Interfaces” option and then the “I2C” and “SPI” options must be selected and enabled. We are enabling SPI as the LoRa Radio hat communicates via SPI, even if the sensor is being connected via I2C for the hat to properly work we will need the SPI interface enabled.

Shim Main Driving Code Installation:

Prior to the next steps, verify that you login into the Raspberry Pi and navigate to the file location where you want the driving code installed. Make note of the location so that you can traverse the file system to edit file within the codebase.

To install the main driving code, we must utilize the git package to grab the public codebase and download it to our local machine (Raspberry PI in this case). Execute the follow command to do so:

```
git clone https://github.com/ECU-Sensing/Raspi_Zero_Node.git
```

Next, install the python dependencies required to run the application by utilizing the python3 package manager pip3. Execute the following command

```
sudo pip3 install -r Raspi_Zero_Node/requirements.txt
```

The last step of this procedure is to set the correct keys for the specific node. You will find a set of keys assigned to your device that you will need to enter into the keys.py file located in the "Raspi_Zero_Node" directory. It is recommended to use a default text editor such as nano to do so.

Sensor Specific Code Installation:

We installed the "esdn-sensing" library to handle the sensor specific implementations of the sensor reading and encoding of data for LoRa transmission. However, we still need to configure the data.py file of the "Raspi_Zero_Node" codebase to reflect our specific sensor needs.

Documentation for the esdn-sensing library is available at: https://ecu-sensing.github.io/esdn_sensing/

Open the data.py file with your favorite/default text editor. The following are items that will need to be changed for each sensor implementation:

- esdn-sensing import line
- get_data(): function
    - sensor_data variable needs to store the correct function call from the correct class

Once the file is configured to the correct sensor class we can move forward to testing. We may return to this section if testing presents errors relative to the application code.

Testing:

It is recommended that you follow the testing procedures in order.

Test Sensor Connection:

We will utilize an additional function from the esdn-sensing library to test the functionality of the sensor. Where originally we wanted to gather the encoding byte array from the sensor class this time, we will utilize a simple pre-defined test function that will run a short sample with the sensor and print the output to the standard output. To use the test function, we will need to create a new python script, it can be stored anywhere on the local machine. Open your favorite/default text editor and give the script a short name (such as 'test.py'). Follow these steps to write the script:

4. Import esdn-sensing specific sensor class
5. Include simple try except structure to check for SensorError
6. Call the test() function from the specific sensor class (Hint: not the get_data()

```
from esdn_sensing import Hydros, SensorError
import logging
import sys

#//=======================================
def get_data():
    try:
        sensor_data = Hydros().get_data()
    except (FunctionTimedOut, SensorError) as err:
        exception = 3
        print(err)

    return sensor_data
#//=======================================
```

Figure 31: Python Sensor Script Example

You should exit and save the python script and prepare to run the script. You will need to run the test with privledge since the dependencies were installed for the "sudo" user. To run your python script, see the following command as an example:

sudo python3 test.py

Once you execute this command you should watch the console window to monitor the output for sensor verification. The try except trap you set earlier should spring and alert you if the sensor is disconnected or unable to be read. See FAQ for more information on common errors.

Test Main Transmission Application:

Now it is important to test the main transmission application to verify that it runs into no Runtime errors. To execute the main application run the following command:

sudo python3 [insert Raspi_Zero_Node location]/txrx_ttn.py

This python script will handle the entire process of reading the keys from keys.py, receiving the data from data.py and completing the transmission. You should see the LoRa specification fields outputted to the console follow by a 'TX done' line to let you know the transmission is completed. **The script by nature will continue to run in the console until killed as it waits for a downlink**. We will not be using the downlink feature so you can **kill this script after transmission with a Ctrl+C.**

Check TTN for transmission:

To eliminate the possibility that your packets are being lost due to environment factors it is recommended that you run the script at least twice with a few seconds between execution to send multiple packets. You can monitor the TTN console provided to see when your transmission is received.

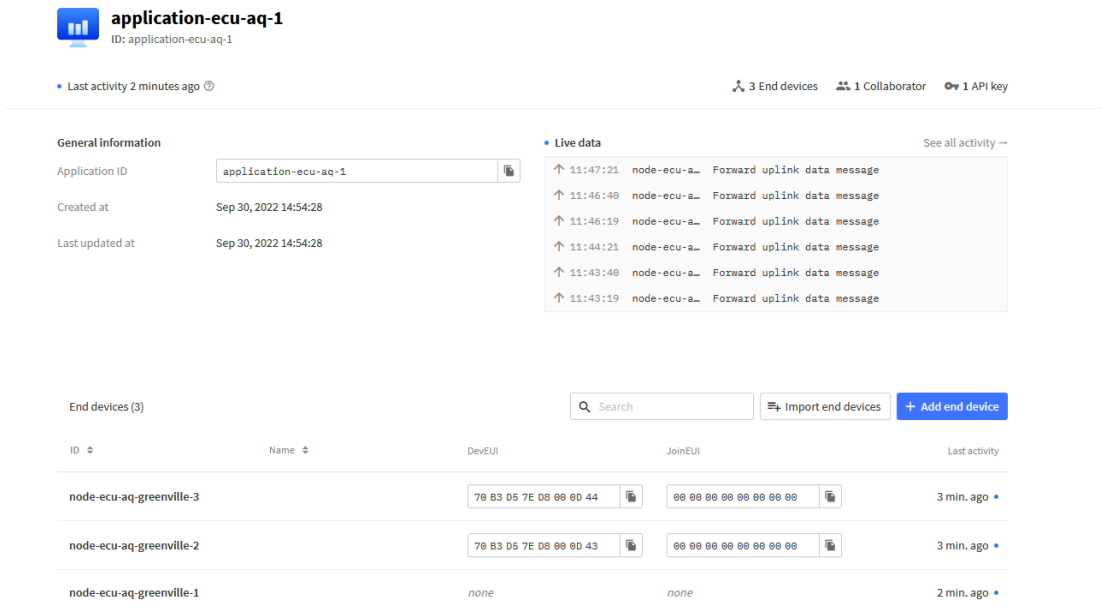

Figure 32: TTN Console with recent received uplinks

You have now successfully read a sensor and transmitted its output via LoRa!!

Problem 9: Beak Beam Sensor

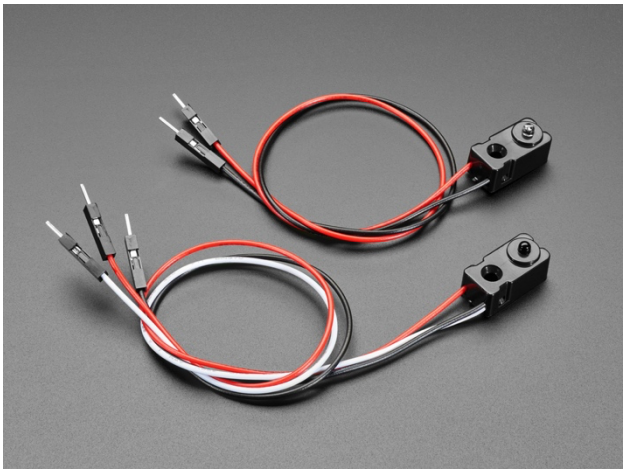

Figure 33: Break Beam Sensor (adafruit.com)

Introduction:

The Break Beam sensor is a low-cost sensor that reports when the beam is broken between the two ends. It support a digital interface option that this tutorial we will discuss how to connect and use.

Please see FAQ for common troubleshooting assistance.

Hardware Installation:

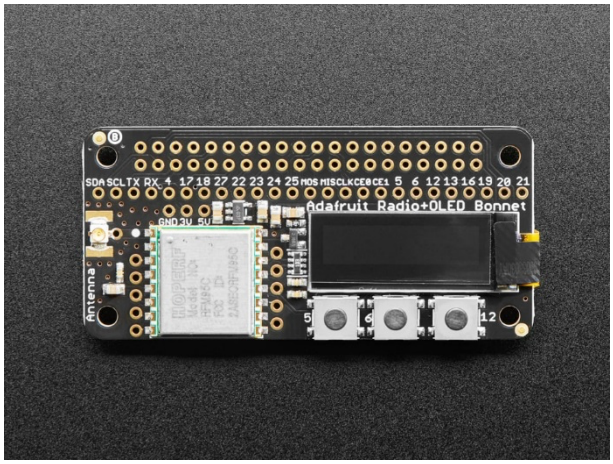

Figure 2.0: LoRa Radio Hat Top (adafruit.com)

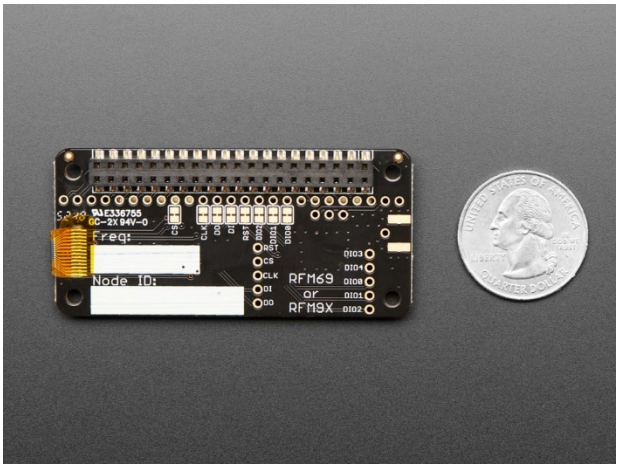

Figure 34.1: LoRa Radio Hat Bottom (adafruit.com)

In this setup it is recommended that you work with a breadboard and prototyping wire to provide the most solid connection without requiring a soldered header.

LoRa Radio Hat Connection:

The installation of the LoRa Radio hat is very straight-forward as it comes with pre-soldered female headers that can be directly installed on the Raspberry PI GPIO.

Verify that the OLED screen is facing upwards and install the hat by evenly sliding the female headers onto the GPIO pins with a modest touch. **It is important not to force the hat onto the GPIO as you may damage the male pins.** If it seems that too much force is required remove the hat and look at the FAQ for common troubleshooting help.

Once the hat is properly seated, look for the circular UFL connector on the far middle left of the hat. It should be easy to spot as it's the only connector of its kind and it shaped like a circle with a single pin in the middle, there are also two flat conductive strips on either side of it. This is where you will connect the antenna adapter cable. Take the cable connector (this should also be a small circle but with a small conductive pad) and click it into place over the UFL header. This pin and cable connection is very small and thus fragile. **It is important to make sure that the cable does not damage the header.** You can verify that the connection is solid as the cable show move around in a circular motion parallel with the board but not feel loose or fall off. You can slightly pull on the connection to verify its affixed properly

Sensor Connection:

This guide will focus on connecting the sensor via three digital input/output pins. A general understanding of serial communication is expected but not required.

Since we are working with a breadboard and a header has already been soldered onto the pins of the sensor you can directly plug them into the respective terminal lines. **Verify that the pins are on independent terminal lines unless you intend them not to be.**

See below for specific hardware connection steps:

Connection Pins:

- Sensor 1 RED AND Sensor 2 RED to Shim 3V
- Sensor 1 BLACK AND Sensor 2 BLACK to Shim GND
- Sensor YELLOW to Shim D0

You must locate the four pins of the LoRa hat as we will be connecting our sensor to the open female pin's slots in the top face of the hat.

Locate the four pins:

- 3V is below the pin slot labeled 17 and the GPIO
    - Verify this is 3V and NOT 5V
- GND is directly beside the 3V
- D0 is directly between the leftmost button and the LoRa transceiver chip on the bottom

You connect the prototyping wires by sliding one side into the pins slots on the hat and the other into four **different, corresponding** terminal lines of the breadboard. **Verify that the terminal lines are not directly connected and match the same connected terminal lines as the intended sensor pin.**

You should notice that the sensor has a status LED that is lit (often time this LED is Green)

Software Installation and Setup:

Installing External Packages:

We will need a few external packages to support our shim driving code and installation efforts.

Execute the following commands (it is recommended to do so as privileged user “sudo”)

```
sudo apt install git python3-pip -y

sudo apt-get install i2c-tools -y
```

Raspberry Pi Configuration:

The first step in setting up the software of the Raspberry PI is to enable the specific interfaces the hat and sensor expect to communicate over. We can do this by entering the privileged settings with the following command:

```
sudo raspi-config
```

Once in the menu navigate to the “Interfaces” option and then the “I2C” and “SPI” options must be selected and enabled. We are enabling SPI as the LoRa Radio hat communicates via SPI, even if the sensor is being connected via I2C for the hat to properly work we will need the SPI interface enabled.

Shim Main Driving Code Installation:

Prior to the next steps, verify that you login into the Raspberry Pi and navigate to the file location where you want the driving code installed. Make note of the location so that you can traverse the file system to edit file within the codebase.

To install the main driving code, we must utilize the git package to grab the public codebase and download it to our local machine (Raspberry PI in this case). Execute the follow command to do so:

```
git clone https://github.com/ECU-Sensing/Raspi_Zero_Node.git
```

Next, install the python dependencies required to run the application by utilizing the python3 package manager pip3. Execute the following command

```
sudo pip3 install -r Raspi_Zero_Node/requirements.txt
```

The last step of this procedure is to set the correct keys for the specific node. You will find a set of keys assigned to your device that you will need to enter into the keys.py file located in the "Raspi_Zero_Node" directory. It is recommended to use a default text editor such as nano to do so.

Sensor Specific Code Installation:

We installed the "esdn-sensing" library to handle the sensor specific implementations of the sensor reading and encoding of data for LoRa transmission. However, we still need to configure the data.py file of the "Raspi_Zero_Node" codebase to reflect our specific sensor needs.

Documentation for the esdn-sensing library is available at: https://ecu-sensing.github.io/esdn_sensing/

Open the data.py file with your favorite/default text editor. The following are items that will need to be changed for each sensor implementation:

- esdn-sensing import line
- get_data(): function
    - sensor_data variable needs to store the correct function call from the correct class.

Once the file is configured to the correct sensor class we can move forward to testing. We may return to this section if testing presents errors relative to the application code.

Testing:

It is recommended that you follow the testing procedures in order.

Test Sensor Connection:

We will utilize an additional function from the esdn-sensing library to test the functionality of the sensor. Where originally we wanted to gather the encoding byte array from the sensor class this time we will utilize a simple pre-defined test function that will run a short sample with the sensor and print the output to the standard output. To use the test function we will need to create a new python script, it can be stored anywhere on the local machine. Open your favorite/default text editor and give the script a short name (such as 'test.py'). Follow these steps to write the script:

1. Import esdn-sensing specific sensor class
2. Include simple try except structure to check for SensorError
3. Call the test() function from the specific sensor class (Hint: not the get_data()
    a. This sensor runs over a period before transmitting. It is important to verify it is accurately detecting the break in the beam during the window.

```
from esdn_sensing import Hydros, SensorError
import logging
import sys

#//=======================================
def get_data():
    try:
        sensor_data = Hydros().get_data()
    except (FunctionTimedOut, SensorError) as err:
        exception = 3
        print(err)

    return sensor_data
#//=======================================
```

Figure 35: Python Sensor Script Example

You should exit and save the python script and prepare to run the script. You will need to run the test with privledge since the dependencies were installed for the "sudo" user. To run your python script, see the following command as an example:

sudo python3 test.py

Once you execute this command you should watch the console window to monitor the output for sensor verification. The try except trap you set earlier should spring and alert you if the sensor is disconnected or unable to be read. See FAQ for more information on common errors.

Test Main Transmission Application:

Now it is important to test the main transmission application to verify that it runs into no Runtime errors. To execute the main application run the following command:

sudo python3 [insert Raspi_Zero_Node location]/txrx_ttn.py

This python script will handle the entire process of reading the keys from keys.py, receiving the data from data.py and completing the transmission. You should see the LoRa specification fields outputted to the console follow by a 'TX done' line to let you know the transmission is completed. **The script by nature will continue to run in the console until killed as it waits for a downlink**. We will not be using the downlink feature so you can **kill this script after transmission with a Ctrl+C.**

Check TTN for transmission:

To eliminate the possibility that your packets are being lost due to environment factors it is recommended that you run the script at least twice with a few seconds between execution to send multiple packets. You can monitor the TTN console provided to see when your transmission is received.

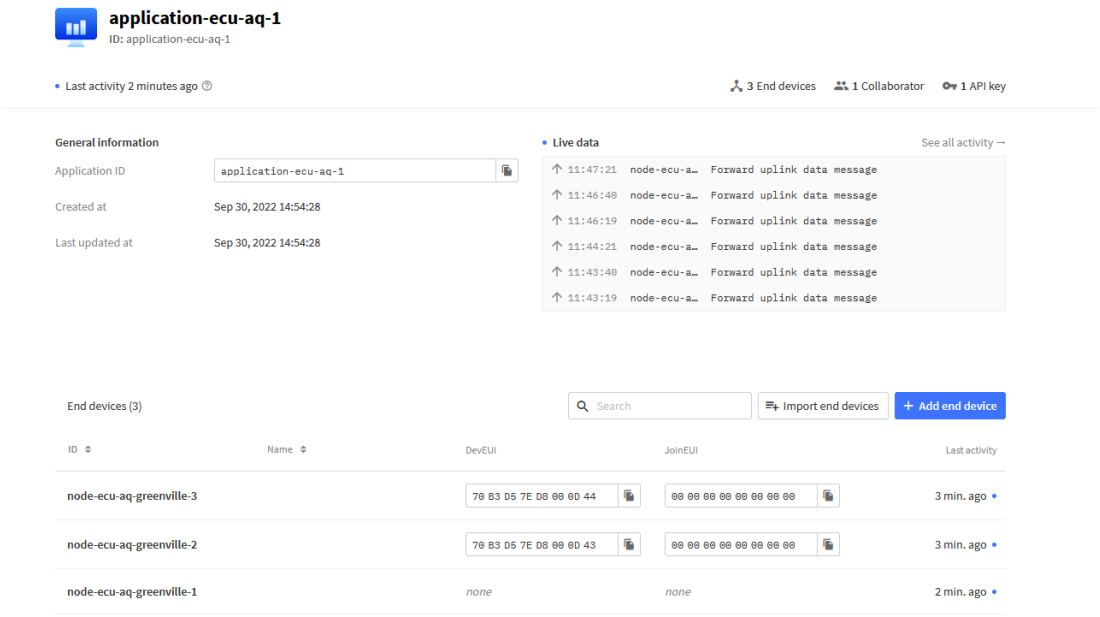

Figure 36: TTN Console with recent received uplinks

You have now successfully read a sensor and transmitted its output via LoRa!!

FAQ:

Q: LoRa Hat requires too much force to attach to GPIO

A: It is possible that the female header slots are obstructed, the best way to double-check this is to hold the hat up to the light and verify that every slot is clear and square shaped. If there is any obstruction it could prevent the pins from sliding into the slot.

Q: Sensor LED is lighting up but unable to read sensor data

A: This is likely a data line issue. We assume the sensor is getting the proper power from the 3V and GND connections but if you are unable to communicate with the sensor it could be that your SDA (Data) or SCL (Clock) lines are disconnected or crossed.

Q: Sensor LED shows no light

A: This is likely an issue with the 3V or GND connection. Check that both pins are connected properly and not on the same terminal line.

Q: Sensor is reading properly in test but not during transmission

A: This is likely that the sensor connection is loose, double check your connections. Otherwise it's likely you are referencing the get_data() function in data.py improperly.

Q:

A: