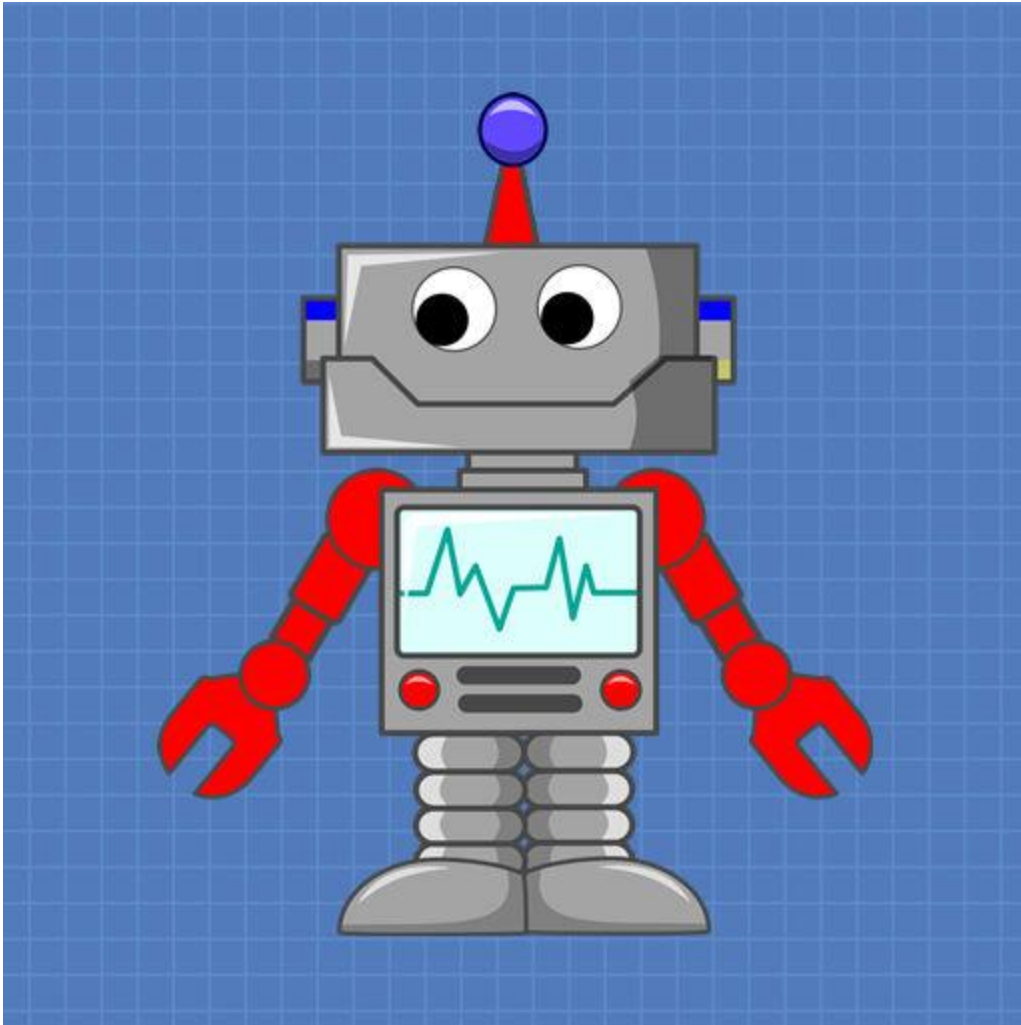


# LoRa – Getting Started with Arduino, ESP32 & Pico



DroneBot Workshop Tutorial

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

Today, we will start with a fantastic radio technology, one that is perfect for IoT projects. Meet LoRa, the inexpensive **Long-Range** radio system that lets you transmit over long distances without a license.

## Introduction

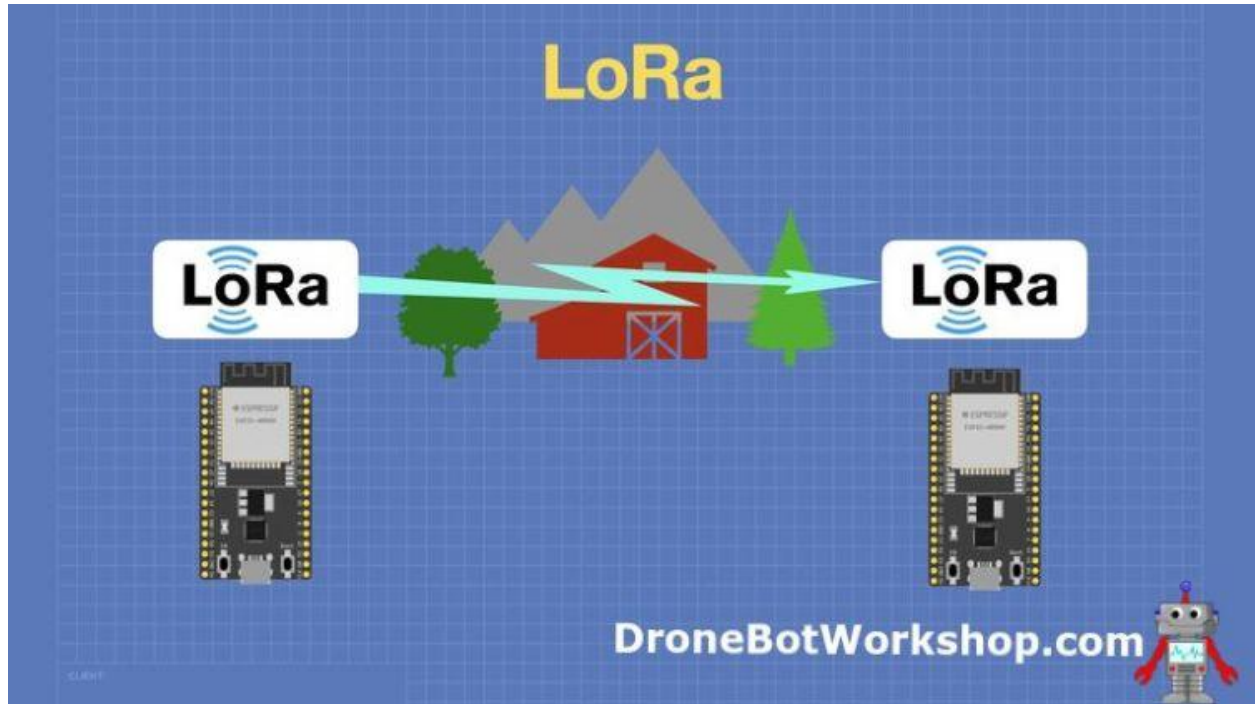
There are many different methods of connecting sensors and actuators in an IoT or remote control installation. WiFi and Bluetooth are pretty common choices and can work well to move large volumes of data, such as video and audio. However, these can be limited in distance, especially when used outdoors.

Cellular technology can be used to send data over wide areas, and for high-bandwidth data like video, it can be a good choice. Of course, a choice like this comes with a higher price tag than simple WiFi or Bluetooth.

But in many IoT and remote control installations, all you really need to send is small bits of data, such as temperature or soil moisture readings. For those applications, there is a much better choice – LoRa.

## LoRa

LoRa is a **Long Range** radio system. It uses an unlicensed radio band and a technology called “Chirp Spread Spectrum” (CSS) to broadcast low-bandwidth data over remarkably large areas. Because of the low power requirements, LoRa devices are inexpensive and consume very little power, so they are ideal for outdoor applications where they can even be solar-powered.



LoRa has a very low bandwidth, typically 300 to 50,000 bits per second. If you are old enough to remember the early dial-up modems, you're familiar with this data rate! It also is best suited for data that isn't constantly streaming, but is instead sent in chunks or bursts.

Because of these constrictions, you won't use LoRa for video or audio; WiFi and Bluetooth are better choices here.

But LoRa is perfect for sensor data, where you just need to send a few bytes of information and can send it periodically. This is a common requirement; things like humidity and soil moisture don't typically change too rapidly, so you can read those sensors every 10 seconds (or even every minute) and still obtain information that is current enough to be useful.

LoRa has an incredible range for something so low-powered. I tested one of the experiments we did today and achieved a range of over two blocks using simple wire antennas and a transmitter **locked in a metal box** in my basement (more on that "metal

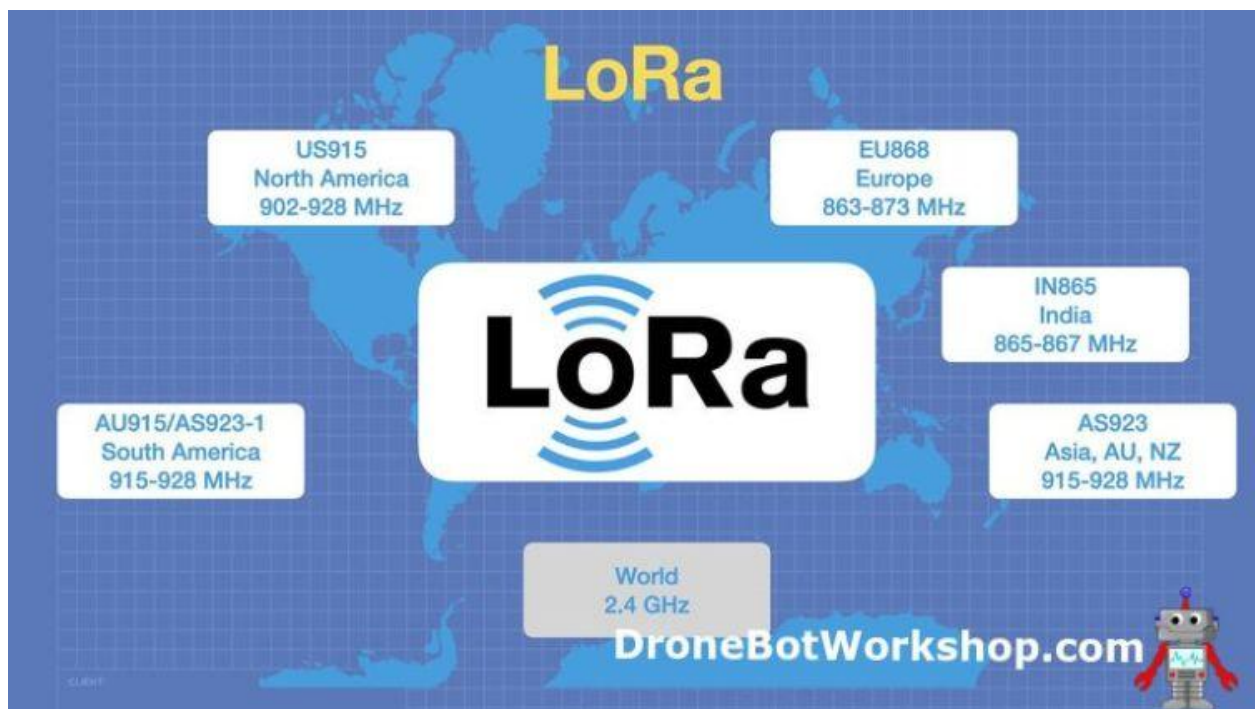
For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

box” later!). With a proper antenna setup, ranges of several miles or kilometers are possible.

The current record for LoRa range was set in the Netherlands in 2020 by Thomas Telkamp, CTO and co-founder of Lacuna Space. He used a balloon to achieve an [unbelievable range of 832 kilometers, or 512 miles!](#)

## Legalities of Using LoRa

LoRa uses the Industrial, Scientific, and Medical or *ISM* band, which doesn’t require a license for low-powered devices. You’re operating within the law, providing that you don’t modify your LoRa radio module to increase its output.



There are different frequencies used for the ISM band around the world, so you’ll need to ensure that you are using the frequency that is legal in your area. This [Frequency](#)

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

*Plans by Country* document by [The Things Network](#) will help you find the correct frequency for your area.

LoRa itself was originally developed by Cycleo, a French company founded in 2002. They focused on creating a low-cost technology that allowed low-power and long-range communications for short bursts of data.

In 2012, Cycleo was acquired by [Semtech Corporation](#). Semtech continues to develop LoRa and holds the patent for its proprietary modulation technique. To design a radio product with LoRa technology, you'd typically need to purchase chips or modules from Semtech or its partners. But you can use commercial LoRa components for your designs without infringing on patents.

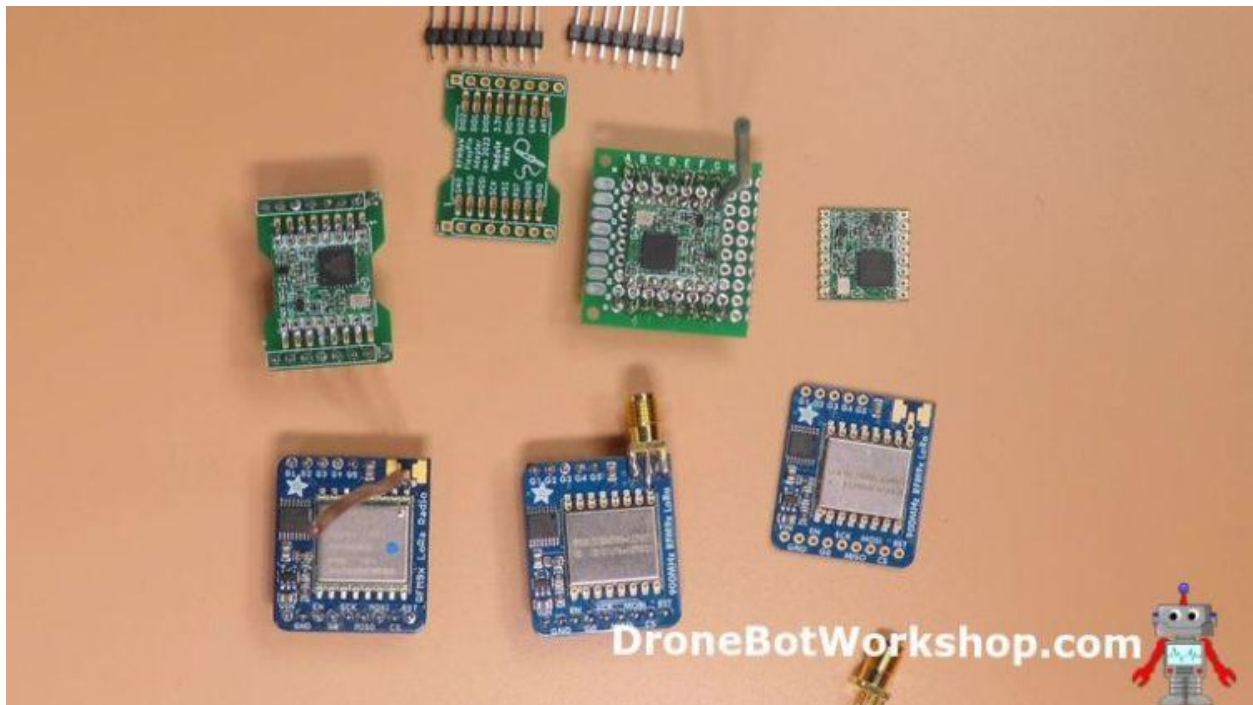
LoRa technology itself is not open-source. However, the protocol built upon LoRa, known as *LoRaWAN* (Long Range Wide Area Network), is open-source. We will discuss LoRaWAN later in this article.

In short, you don't need to worry about using LoRa in your designs; the manufacturer of the LoRa radio module you use has already paid Semtech its license fee. Just make sure to use the appropriate ISM band frequency for your location.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

## LoRa Modules

We will be using a couple of LoRa modules in our experiments today. Actually, they are two versions of the same module, the HopeRF RFM95W.




<https://dronebotworkshop.com>



## HopeRF RFM95W

The [HopeRF RFM95W](#), whose specs and pinout are illustrated here, is a tiny LoRa radio module with an SPI interface.



**HopeRF RFM95W**

- LoRa Modem module
- Power output to **100 mW**
- Sensitive down to **-148 dBm**
- **10 mA** receive current
- **SPI connection & GPIO**
- **3.3 VDC** power @ **120 mA**
- **3.3-Volt Logic only**
- **Not breadboard-friendly!**

**DroneBotWorkshop.com**

Pinout labels (left to right): GND, MISO, MOSI, SCK, NSS, RESET, DIO5, GND, DIO2, DIO1, DIO0, 3.3V, DIO4, DIO3, GND, ANT.

And by “tiny,” I really mean tiny! This module is so small that its pins are spaced closer together than the standard 0.1-inch header spacing common to most modules. As such, you’ll need to find a way of mounting it so that you can use it on a solderless breadboard.

One method is to do it all by hand, using a perfboard. I did this, and it worked well; I used 30 gauge wires to connect the HopeRF RFM95W pins to the pins of a couple of male headers.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

You can also buy adapters. I picked up a few that are made by Solder Party. They are intended for use with “FlexyPins” (spring-loaded contact pins) to provide a quick-connect method of using the module, but you can also solder the module on.

Note that this module runs on 3.3-volts and only uses 3.3-volt logic, so take care if interfacing with a 5-volt microcontroller.




## Adafruit RFM9x

The [Adafruit RFM9x](#) is available in two models:

- The RFM95 LoRa – Use with 868 and 915 MHz LoRa
- The RFM96 LoRa – Use with 433 MHz LoRa

The two modules are identical, aside from their frequency differences.




The image shows a blue PCB module with a central silver chip. The chip has 'FCC ID: 2AD66-1276C1' and 'IC ID: 21278-1276C1' printed on it. The module has various pins labeled: G1, G2, G3, G4, G5 at the top; VIN, EN, SCK, MOSI, RST at the bottom; and GND, G0, MISO, CS at the bottom. A small antenna is visible on the right side. The text '900MHz RFM9x LoRa' is printed vertically on the right edge of the PCB.

### Adafruit RFM9x LoRa

- FSK Packet Radio
- Power output to 100 mW
- 50 mA to 150 mA transmission
- 30 mA receive current
- SPI connection
- 3.3 - 6 VDC power @ 150 mA
- 3.3 - 5 Volt Logic Safe

DroneBotWorkshop.com



These modules use the same HopeRF RFM95W LoRa modules described previously, but Adafruit has added a few nice features.

First, there is an onboard voltage regulator. This allows the modules to be powered with any DC supply from 3 to 6 volts. Second, there is onboard logic-level conversion, so the modules are safe with both 3.3 and 5-volt logic devices.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

The pins on the module are also spaced the standard 0.1-inch, but it's a bit of a stretch to call these "solderless breadboard friendly" as the module is too wide for most breadboards. I'll show you a few ways to accommodate it later in the article.

You also need to figure out how to connect an antenna. If you are using a simple wire antenna, it can be soldered directly to the module. You can also solder a coax cable or an RF connector to the module.

Note that this module is packaged with header pins that are not soldered in. It does NOT come with an RF connector.

## Swapping Modules

As these two modules are really the same device in different packaging, they can be used interchangeably in our experiments. Just take note of the differences:

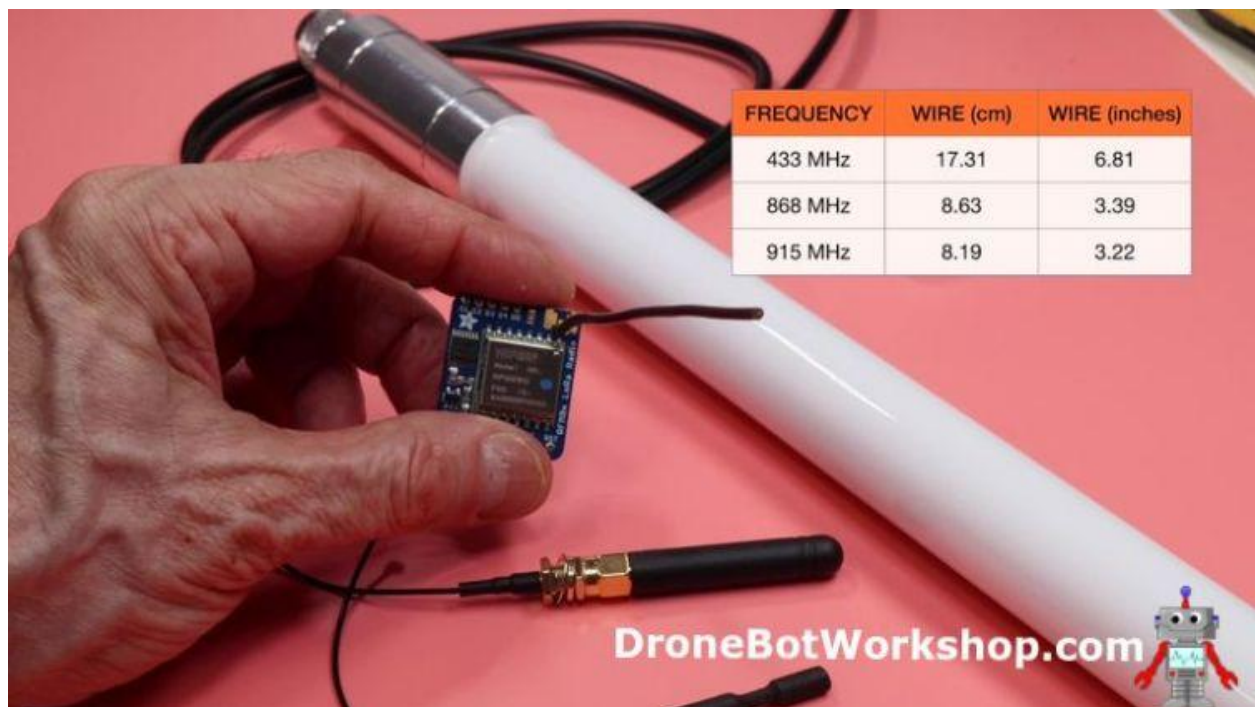
- The HopeRF RFM95W can only use a 3.3-volt power supply.
- The HopeRF RFM95W can only interface with 3.3-volt logic.
- Some of the pin labels are different between the modules.

On that last point, here is a diagram that will let you convert. Keep the other two bullet points in mind if you use the HopeRF RFM95W.

## LoRa Antennas

Regardless of which module you choose, you will need an antenna. **You should never run a LoRa module without an antenna, it can be damaged by doing this.**

An antenna can be as simple as a piece of wire, in fact, that is exactly the type of antenna I used for all of these experiments. Not just any wire will do, though; it has to be the correct length.



The length of the wire depends upon the frequency at which you operate the radio module. For a quarter-wave antenna, the lengths are as follows:

- **433 MHz** – 17.31 cm (6.81 inches)
- **868 MHz** – 8.63 cm (3.39 inches)
- **915 MHz** – 8.19 cm (3.22 inches)

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

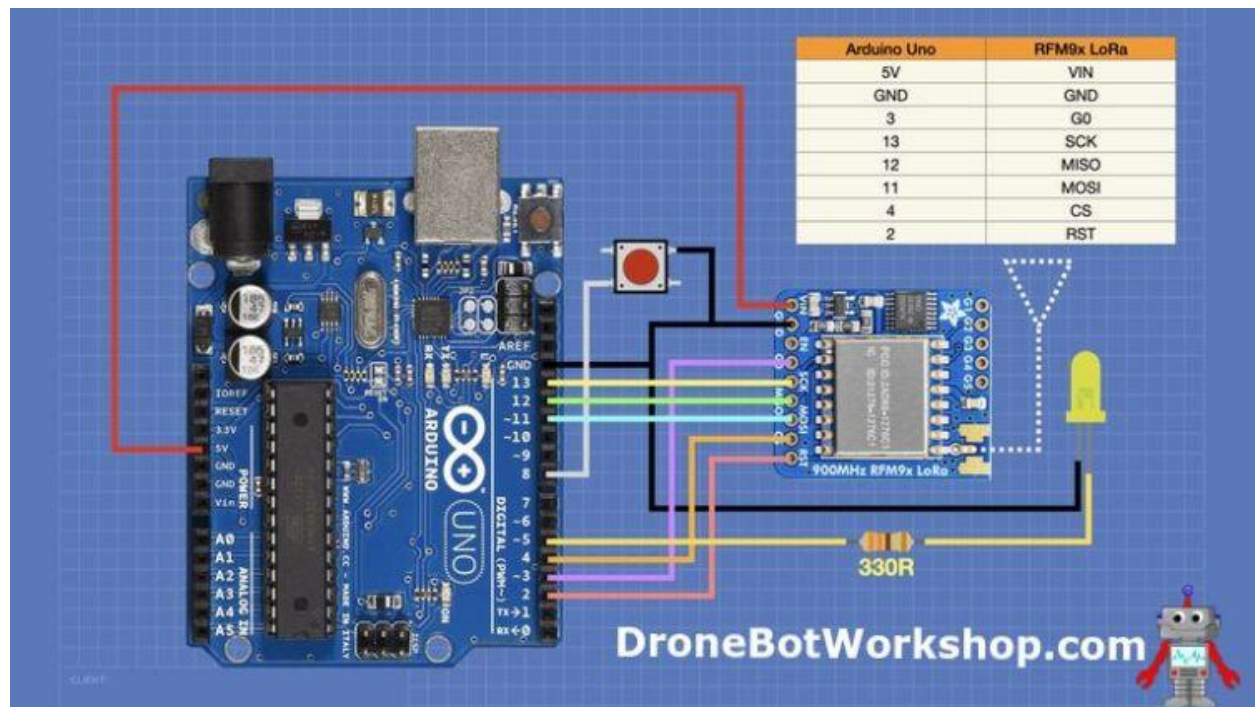
You can also use a commercial antenna, there are several to choose from. Just make sure to select an antenna that is made for the frequency band that you are running your LoRa radio module.



<https://dronebotworkshop.com>

## Arduino Hookup

We will do a few experiments using a couple of Arduino Uno boards. The Uno is a 5-volt logic device, so we will use the Adafruit RFM9x module. Remember to purchase the module that suits your local ISM band frequency.



You can use either a classic Uno R3 (or clone) board or one of the new Uno R4 boards for the experiments. The only consideration is the value of the LED dropping resistor; if you use an R4 board, keep this at 330 ohms or higher to avoid drawing too much current from the I/O port. With the classic board, it can be as low as 150 ohms.

You can, of course, use other microcontrollers instead of the Uno for the experiments. You must know the default SPI connections and rewire the circuit accordingly. Also, the G0 pin must be connected to an interrupt-capable pin on the microcontroller. Using a

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

3.3-volt microcontroller, you could use the HopeRF RFM95W instead of the Adafruit module.

In addition to the Arduino, you'll also need the Adafruit LoRa module (make sure to choose the correct one for your frequency), a pushbutton switch, an LED, and a dropping resistor for that LED. Wire them up as per the diagram; for the LED, the anode (longer lead) goes to the dropping resistor, and the cathode is grounded.

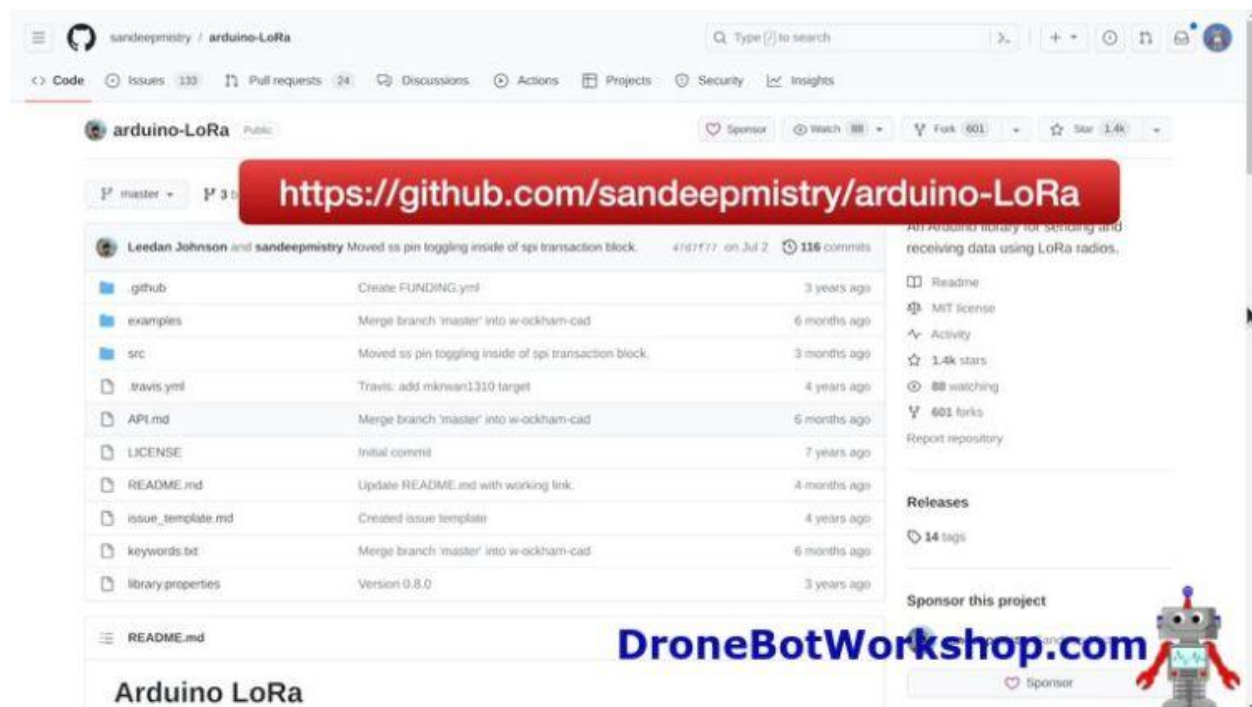
## The LoRa Library for Arduino

As with most Arduino circuits, when you want to work with an external device, there's a library for that! Actually, there are many LoRa libraries available.

The library we will use is written by Sandeep Mistry and is simply called the *LoRa Library*. It makes sending and receiving LoRa packets very simple; transmitting packets with this library is about as simple as printing to the serial monitor. It also implements a receive (and transmit) callback system, making coding easy. I'll illustrate some examples of this in a bit.



For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>



You can read the [implementation details for the LoRa library](#) on its GitHub page. The API.md document has all the information you'll need, and the main page shows the wiring information in case you wish to use the library with a different processor or LoRa module.

## Installing the LoRa Library

Although you can install the library directly from GitHub, using the Arduino IDE Library Manager is easier.

Open the Library Manager. With the newer IDE 2.x, this can be done by clicking the “library book” icon on the sidebar. You can also open it with the top menu on both versions of the IDE.

Search for “lora”. You'll get several results, including unrelated libraries with the letters “l o r a” in their name!

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

Scroll down a bit and look for LoRa by Sandeep Mistry. Click on the *Install* button to add this library to your Arduino IDE.

The library will also install several example sketches. You can look them over to get a better idea as to how to use it.

## Demo 1 – Simple Data Transfer

Our first demo is pretty basic. We will just send a packet of information from one Arduino to the other using LoRa. We will display the packet number on both ends using the serial monitor.

Despite its simplicity, the example will do a fine job illustrating how you use the LoRa library, which I will assume you have already installed in your Arduino IDE.

One of our Arduino boards will be designated the Receiver and the other the Sender. Make note of which is which!

Each board will require its own sketch.

### Receiver Sketch

The Arduino designated as the Receiver will use this sketch. Before you load it, please read the notes regarding setting the correct operating frequency:

```
1  /*
2   LoRa Demo 1 Receiver
3   lora-demo1-receive.ino
4   Receives and displays contents of test packet
5   Requires LoRa Library by Sandeep Mistry -
6   https://github.com/sandeepmistry/arduino-LoRa
7
8   DroneBot Workshop 2023
9   https://dronebotworkshop.com
10  */
11
12  // Include required libraries
13  #include <SPI.h>
14  #include <LoRa.h>
15
16  // Define the pins used by the LoRa module
17  const int csPin = 4;      // LoRa radio chip select
18  const int resetPin = 2;   // LoRa radio reset
19  const int irqPin = 3;     // Must be a hardware interrupt pin
20
21  void setup() {
22    Serial.begin(9600);
23    while (!Serial)
24      ;
25
26    // Setup LoRa module
27    LoRa.setPins(csPin, resetPin, irqPin);
28
29    Serial.println("LoRa Receiver Test");
```

```
29
30 // Start LoRa module at local frequency
31 // 433E6 for Asia
32 // 866E6 for Europe
33 // 915E6 for North America
34
35 if (!LoRa.begin(915E6)) {
36     Serial.println("Starting LoRa failed!");
37     while (1)
38         ;
39 }
40 }
41
42 void loop() {
43
44     // Try to parse packet
45     int packetSize = LoRa.parsePacket();
46     if (packetSize) {
47         // Received a packet
48         Serial.print("Received ");
49
50         // Read packet
51         while (LoRa.available()) {
52             Serial.print((char)LoRa.read());
53         }
54
55         // Print RSSI of packet
56         Serial.print("' with RSSI ");
```

```
57     Serial.println(LoRa.packetRssi());  
58 }  
59 }
```

We start by including the SPI library required for the LoRa module connections. Of course, we also need to include the LoRa library.

We then define constants for some of the pins undefined in the SPI library and for an interrupt-capable pin to allow us to use the modules' callback functions.

In Setup, we initialize the serial monitor and then set the pins on the LoRa module using the constants we just defined.

The next step is to actually start the module using the command "*LoRa.begin(xxxx)*". The "xxxx" refers to the frequency that your LoRa radio module operates, and it needs to be the proper one for your country.

The value is actually the frequency in scientific notation, with the last number representing the number of digits.

- 433E6 – 433 MHz
- 866E6 – 866 MHz
- 915E6 – 915 MHz

The example I'm showing is for 915 MHz, which is good in North and South America. If you require a different frequency, you must edit the sketch with the appropriate value.

After starting the LoRa module successfully, we exit Setup.

In the Loop, we see if we can parse a data packet. If we can, we print to the serial monitor that data is available. We then loop through the data character by character and print it to the serial monitor.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

We finish the Loop by printing the signal strength or RSSI of the packet.

Load the sketch to the receive Arduino, and ensure you have edited it for the correct ISM frequency, if necessary.

## Sender Sketch

Now, we move on to the sender sketch.



```
1  /*
2   LoRa Demo 1 Sender
3   lora-demo1-send.ino
4   Sends test packet with packet count
5   Requires LoRa Library by Sandeep Mistry -
6   https://github.com/sandeepmistry/arduino-LoRa
7
8   DroneBot Workshop 2023
9   https://dronebotworkshop.com
10  */
11
12  // Include required libraries
13  #include <SPI.h>
14  #include <LoRa.h>
15
16  // Define the pins used by the LoRa module
17  const int csPin = 4;          // LoRa radio chip select
18  const int resetPin = 2;      // LoRa radio reset
19  const int irqPin = 3;        // Must be a hardware interrupt pin
20
21  // Message counter
22  byte msgCount = 0;
23
24  void setup() {
25
26    Serial.begin(9600);
27    while (!Serial)
28      ;
```

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
29 // Setup LoRa module
30 LoRa.setPins(csPin, resetPin, irqPin);
31
32 Serial.println("LoRa Sender Test");
33
34 // Start LoRa module at local frequency
35 // 433E6 for Asia
36 // 866E6 for Europe
37 // 915E6 for North America
38
39 if (!LoRa.begin(915E6)) {
40     Serial.println("Starting LoRa failed!");
41     while (1)
42         ;
43 }
44 }
45
46 void loop() {
47
48     Serial.print("Sending packet: ");
49     Serial.println(msgCount);
50
51     // Send packet
52     LoRa.beginPacket();
53     LoRa.print("Packet ");
54     LoRa.print(msgCount);
55     LoRa.endPacket();
56
```

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
57 // Increment packet counter
58 msgCount++;
59
60 // 5-second delay
61 delay(5000);
62 }
```

The sender sketch starts the same as the receiver one, adding one variable, a message counter.

The Setup is identical to the last sketch. Be sure to edit it for the correct LoRa ISM frequency.

The Loop illustrates how easy it is to send data using the LoRa Library.

We start the Loop by printing both a message and the counter value to the Serial monitor. The counter value is what we will be sending as a message to the receiver.

We then do a *LoRa.beginPacket()* to start forming a LoRa data packet. After that, we do a series of *LoRa.print()* statements, each one adds data to the packet.

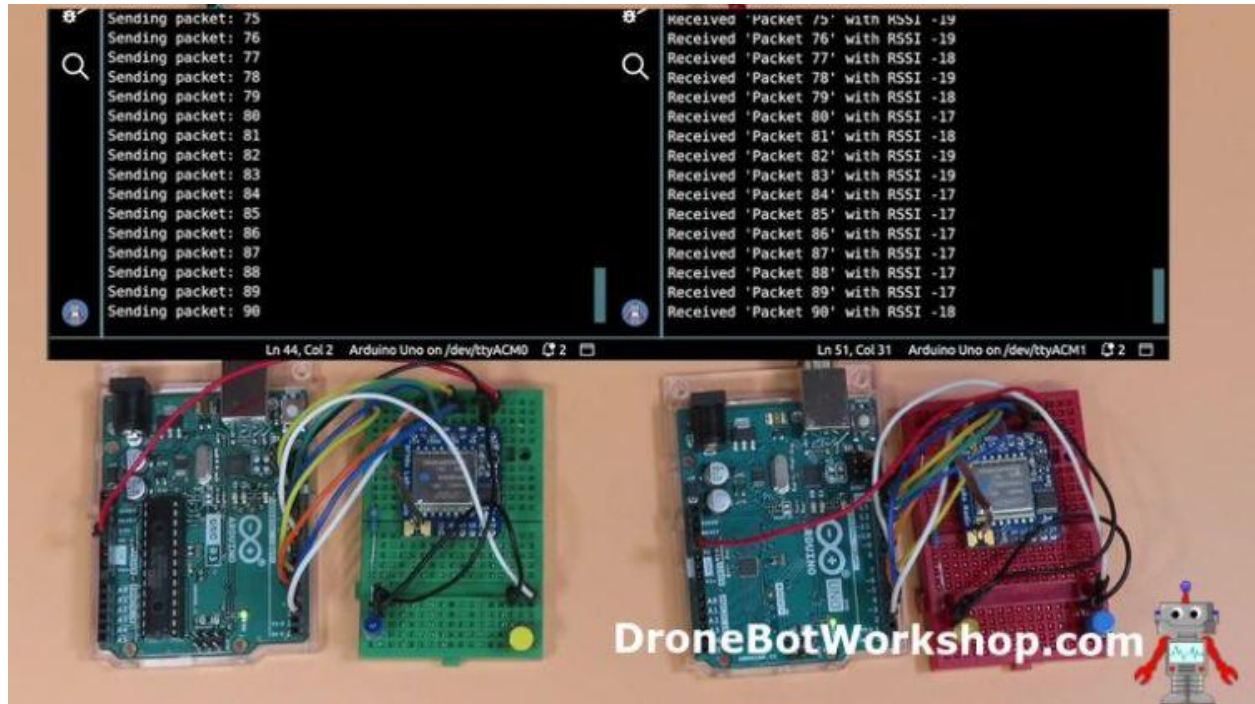
Finally, we call *LoRa.endPacket()*. This finishes forming the packet and also sends it.

We finish the Loop by incrementing the message counter and delaying for five seconds. Remember, with LoRa, you only send bursts of data. You can reduce this delay if you're impatient!

Load this sketch up to the sender Arduino, again noting that it is set for the correct frequency.

Now run both Arduino's. If you are using the newer IDE 2.x, you can open two instances of it and use two serial monitors, as I did here.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>



You should see the packet number on the sender incrementing and the same number being received on the other Arduino.

Experiment by changing the values used in the send data; try adding more `LoRa.print()` statements and observe the results.

This illustrates how simple it is to send and receive data using LoRa. Now let's do something with that data!

<https://dronebotworkshop.com>

## Demo 2 – One-way Remote Control

As we have wired pushbuttons and LEDs up to both Arduino's, you can probably guess what we will do next!

Yes, indeed, we will use a pushbutton to control a remote LED! In this first example, we will just do it in one direction, so one Arduino is the Receiver, and one is the Sender again. You can designate the same ones as last time or switch them around; they are wired up the same!

### Receiver Sketch

Here is the Receiver side sketch:

```
1  /*
2   LoRa Demo 2 Receiver
3   lora-demo2-receive.ino
4   Receive LoRa signal to control LED
5   Requires LoRa Library by Sandeep Mistry -
6   https://github.com/sandeepmistry/arduino-LoRa
7
8   DroneBot Workshop 2023
9   https://dronebotworkshop.com
10  */
11
12  // Include required libraries
13  #include <SPI.h>
14  #include <LoRa.h>
15
16  // Define the pins used by the LoRa module
17  const int csPin = 4;      // LoRa radio chip select
18  const int resetPin = 2;   // LoRa radio reset
19  const int irqPin = 3;     // Must be a hardware interrupt pin
20
21  // LED connection
22  const int ledPin = 5;
23
24  // Receive message variables
25  String contents = "";
26  String buttonPress = "button pressed";
27  bool rcvButtonState;
28
```

```
29 void setup() {  
30  
31   // Set LED as output  
32   pinMode(ledPin, OUTPUT);  
33  
34   Serial.begin(9600);  
35   while (!Serial)  
36     ;  
37  
38   // Setup LoRa module  
39   LoRa.setPins(csPin, resetPin, irqPin);  
40  
41   Serial.println("LoRa Receiver");  
42  
43   // Start LoRa module at local frequency  
44   // 433E6 for Asia  
45   // 866E6 for Europe  
46   // 915E6 for North America  
47  
48   if (!LoRa.begin(915E6)) {  
49     Serial.println("Starting LoRa failed!");  
50     while (1)  
51       ;  
52   }  
53 }  
54  
55 void loop() {  
56
```



```
57 // Try to parse packet
58 int packetSize = LoRa.parsePacket();
59
60 // Received a packet
61 if (packetSize) {
62
63     Serial.print("Received packet ");
64
65     // Read packet
66     while (LoRa.available()) {
67         contents += (char)LoRa.read();
68     }
69
70     // Print RSSI of packet
71     Serial.print(" with RSSI ");
72     Serial.println(LoRa.packetRssi());
73     Serial.println(contents);
74
75     // Toggle button state
76     if (contents.equals(buttonPress)) {
77         rcvButtonState = !rcvButtonState;
78     }
79
80     // Drive LED
81     if (rcvButtonState == true) {
82         digitalWrite(ledPin, HIGH);
83         Serial.println("led on");
84     } else {
```

```
85     digitalWrite(ledPin, LOW);  
86     Serial.println("led off");  
87 }  
  
88  
89 // Clear contents  
90 contents = "";  
91 }  
92 }
```

On the receiver side, we start like all our sketches have so far. We also define a few other variables:

- A constant for the LED pin
- A variable for the received message contents.
- A string with the contents "button pressed". This is the text string we are trying to match.
- A boolean to hold the state of the receive button toggle.

The only thing we do in the Setup that is different is to set the LED pin as an output. Once again, ensure you have configured your LoRa.begin statement for the correct frequency.

In the Loop, we read the incoming data packet as before. But this time, we compare the contents with the saved string. If they match, we toggle the button state boolean, which sets the LED status.

We finish by clearing the contents of the incoming message variable.

## Sender Sketch

And, of course, the sender side sketch:

```
1  /*
2   LoRa Demo 2 Sender
3   lora-demo2-send.ino
4   Use pushbutton to control LED on receiver
5   Requires LoRa Library by Sandeep Mistry -
6   https://github.com/sandeepmistry/arduino-LoRa
7
8   DroneBot Workshop 2023
9   https://dronebotworkshop.com
10  */
11
12  // Include required libraries
13  #include <SPI.h>
14  #include <LoRa.h>
15
16  // Define the pins used by the LoRa module
17  const int csPin = 4;      // LoRa radio chip select
18  const int resetPin = 2;   // LoRa radio reset
19  const int irqPin = 3;     // Must be a hardware interrupt pin
20
21  // Message counter
22  byte msgCount = 0;
23
24  // Pushbutton variables
25  int buttonPin = 8;
26  int sendButtonState;
27
28  void setup() {
```

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
29 // Set pushbutton as input
30 pinMode(buttonPin, INPUT_PULLUP);
31
32 Serial.begin(9600);
33 while (!Serial)
34     ;
35
36 // Setup LoRa module
37 LoRa.setPins(csPin, resetPin, irqPin);
38
39 Serial.println("LoRa Sender");
40
41 // Start LoRa module at local frequency
42 // 433E6 for Asia
43 // 866E6 for Europe
44 // 915E6 for North America
45
46 if (!LoRa.begin(915E6)) {
47     Serial.println("Starting LoRa failed!");
48     while (1)
49         ;
50 }
51 delay(1000);
52 }
53
54 void loop() {
55
56     // Get pushbutton state
```

<https://dronebotworkshop.com>

```
57  sendButtonState = digitalRead(buttonPin);
58
59  // Send packet if button pressed
60  if (sendButtonState == LOW) {
61      // Send packet
62      LoRa.beginPacket();
63      LoRa.print("button pressed");
64      LoRa.endPacket();
65      msgCount++;
66      Serial.print("Sending packet: ");
67      Serial.println(msgCount);
68      delay(500);
69  }
70 }
```

The sender sketch also builds upon the last sketch, adding two new variables.

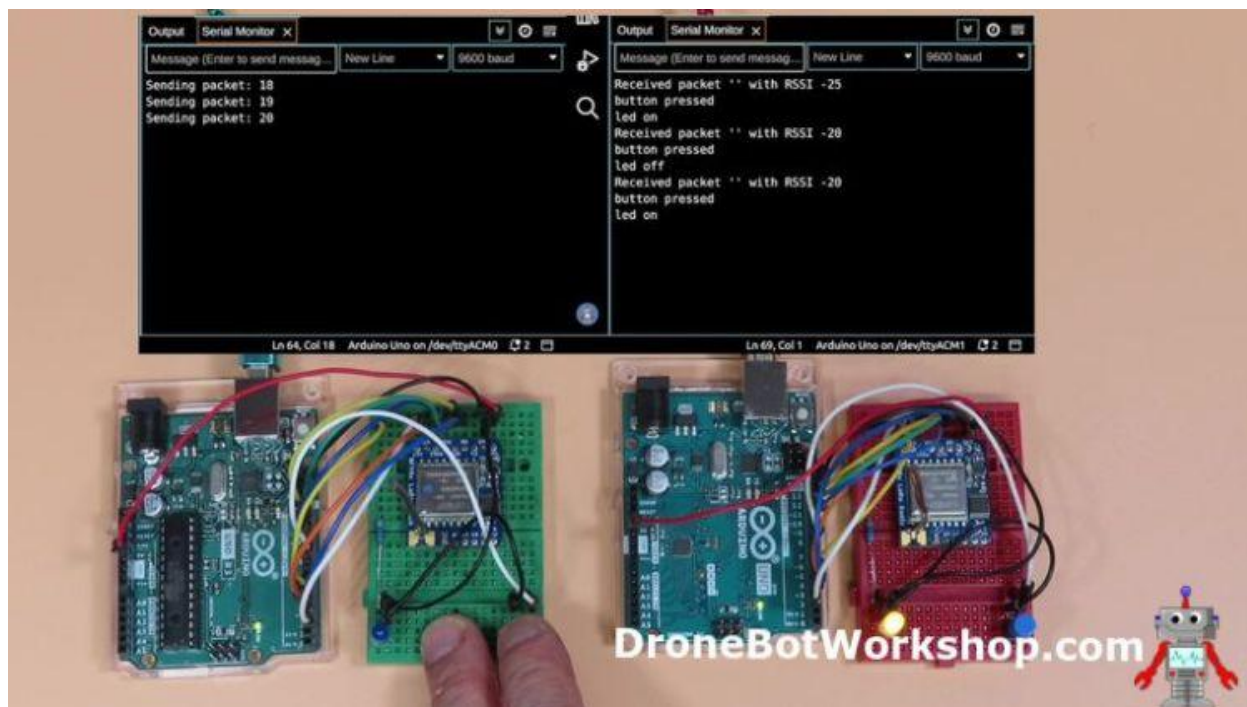
- An integer to represent the pin connected to the pushbutton.
- Another integer, this one represents the state of the pushbutton.

The only thing added to the Setup is to define the pushbutton pin as an input using the internal pull-up resistors.

In the Loop, we read the pushbutton and hold its state in the integer variable. If the state is LOW, we know the button has been pressed, so we compose a packet using the LoRa library and send it to the receiver. The packet contains the message “button pressed”, the same string we are on the lookout for on the receive side.

We add a half-second delay to debounce the pushbutton before exiting and repeating the Loop.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>



Load up the two sketches and observe them in action. You can also watch the serial monitors for each board. You should be able to control the receiver LED using the pushbutton on the sender.

<https://dronebotworkshop.com>

## Demo 3 – Two-way Remote Control

Of course, we have LEDs and pushbuttons on both Arduino boards, so we will want to control them both ways. To do that, we will introduce a few new concepts that will improve our coding and performance.

### Receive Callback

In the following sketch, we'll be using a "receive callback function," which we have used before when coding for other communications technologies.

A "callback" is a signal produced when an event happens; the signal is used to trigger a function, very much like an interrupt. A "receive callback" is initiated whenever data is received by the LoRa module. It's actually the reason we require an interrupt-capable pin to interface with the LoRa module's GPIO pin.

We treat a callback like an interrupt by writing a function to handle it. So, in this case, we will write a function that is called every time a packet of data is received. As with an interrupt handler, it's best not to put too much code into a callback function.

### Two-way Remote Sketch

In this demonstration, we'll use the same sketch on both ends. There is a modification you can make to one sketch if you wish; more on that in a bit.



```
1  /*
2   LoRa Demo 3
3   lora-demo3.ino
4   Bi-directional LED control (duplex communications)
5   Requires LoRa Library by Sandeep Mistry -
6   https://github.com/sandeepmistry/arduino-LoRa
7   sendMessage & onReceive functions based upon "LoRaDuplexCallback" code sample by
8   Tom Igoe
9
10  DroneBot Workshop 2023
11  https://dronebotworkshop.com
12  */
13  // Include required libraries
14  #include <SPI.h>
15  #include <LoRa.h>
16
17  // Define the pins used by the LoRa module
18  const int csPin = 4;      // LoRa radio chip select
19  const int resetPin = 2;   // LoRa radio reset
20  const int irqPin = 3;     // Must be a hardware interrupt pin
21
22  // LED connection
23  const int ledPin = 5;
24
25  // Outgoing message variable
26  String outMessage;
27
28  // Message counter
```

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
29 byte msgCount = 0;
30
31 // Receive message variables
32 String contents = "";
33 String buttonPress = "button pressed";
34 bool rcvButtonState;
35
36 // Source and destination addresses
37 byte localAddress = 0xBB; // address of this device
38 byte destination = 0xFF; // destination to send to
39
40 // Pushbutton variables
41 int buttonPin = 8;
42 int sendButtonState;
43
44 void setup() {
45
46     // Set pushbutton as input
47     pinMode(buttonPin, INPUT_PULLUP);
48
49     // Set LED as output
50     pinMode(ledPin, OUTPUT);
51
52     Serial.begin(9600);
53     while (!Serial)
54         ;
55
56     Serial.println("LoRa Duplex with callback");
```

<https://dronebotworkshop.com>

```
57
58 // Setup LoRa module
59 LoRa.setPins(csPin, resetPin, irqPin);
60
61 // Start LoRa module at local frequency
62 // 433E6 for Asia
63 // 866E6 for Europe
64 // 915E6 for North America
65
66 if (!LoRa.begin(915E6)) {
67     Serial.println("Starting LoRa failed!");
68     while (1)
69         ;
70 }
71
72 // Set Receive Call-back function
73 LoRa.onReceive(onReceive);
74
75 // Place LoRa in Receive Mode
76 LoRa.receive();
77
78 Serial.println("LoRa init succeeded.");
79 }
80
81 void loop() {
82
83 // Get pushbutton state
84 sendButtonState = digitalRead(buttonPin);
```

```
85
86 // Send packet if button pressed
87 if (sendButtonState == LOW) {
88
89 // Compose and send message
90 outMessage = buttonPress;
91 sendMessage(outMessage);
92 delay(500);
93
94 // Place LoRa back into Receive Mode
95 LoRa.receive();
96 }
97 }
98
99 // Send LoRa Packet
100 void sendMessage(String outgoing) {
101   LoRa.beginPacket(); // start packet
102   LoRa.write(destination); // add destination address
103   LoRa.write(localAddress); // add sender address
104   LoRa.write(msgCount); // add message ID
105   LoRa.write(outgoing.length()); // add payload length
106   LoRa.print(outgoing); // add payload
107   LoRa.endPacket(); // finish packet and send it
108   msgCount++; // increment message ID
109 }
110
111 // Receive Callback Function
112 void onReceive(int packetSize) {
```

```
113  if (packetSize == 0) return;  // if there's no packet, return
114
115  // Read packet header bytes:
116  int recipient = LoRa.read();    // recipient address
117  byte sender = LoRa.read();     // sender address
118  byte incomingMsgId = LoRa.read(); // incoming msg ID
119  byte incomingLength = LoRa.read(); // incoming msg length
120
121  String incoming = ""; // payload of packet
122
123  while (LoRa.available()) {      // can't use readString() in callback, so
124      incoming += (char)LoRa.read(); // add bytes one by one
125  }
126
127  if (incomingLength != incoming.length()) { // check length for error
128      Serial.println("error: message length does not match length");
129      return; // skip rest of function
130  }
131
132  // If the recipient isn't this device or broadcast,
133  if (recipient != localAddress && recipient != 0xFF) {
134      Serial.println("This message is not for me.");
135      return; // skip rest of function
136  }
137
138  // If message is for this device, or broadcast, print details:
139  Serial.println("Received from: 0x" + String(sender, HEX));
140  Serial.println("Sent to: 0x" + String(recipient, HEX));
```

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
141 Serial.println("Message ID: " + String(incomingMsgId));
142 Serial.println("Message length: " + String(incomingLength));
143 Serial.println("Message: " + incoming);
144 Serial.println("RSSI: " + String(LoRa.packetRssi()));
145 Serial.println("Snr: " + String(LoRa.packetSnr()));
146 Serial.println();
147
148
149 // Toggle button state
150 if (incoming.equals(buttonPress)) {
151     rcvButtonState = !rcvButtonState;
152 }
153
154 // Drive LED
155 if (rcvButtonState == true) {
156     digitalWrite(ledPin, HIGH);
157     Serial.println("led on");
158 } else {
159     digitalWrite(ledPin, LOW);
160     Serial.println("led off");
161 }
}
```

The sketch starts by declaring all of the variables we have used already. We then declare a local and destination address, and it's here that you can make modifications if you wish.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

In this sketch, we introduce the concept of assigning an address to each device. This is broadcast as part of the LoRa packet. It's not a "LoRa address," just something we include in our self-defined packet.

The address of "FF" is the broadcast address and will work on any device. So, the sketch does not necessarily need to be modified, as it sends out to "FF". But you can experiment with assigning different addresses to both units. The experiment gets even better when you have three or more units!

Before we look at the Loop, let's move to the bottom of the sketch to examine the two functions, which are key to understanding how this all works.

The first one, *sendMessage*, just sends a LoRa packet with the specified message. The packet is formed using LoRa library statements. It includes both the source and destination address, as well as the message and message length. The length is used as a simple error-checking method on the receive side.

The next function is the receive callback.

The function begins by checking to see if the packet is valid. If it is, it extracts the dates from it and checks the data length to see if it matches what it received.

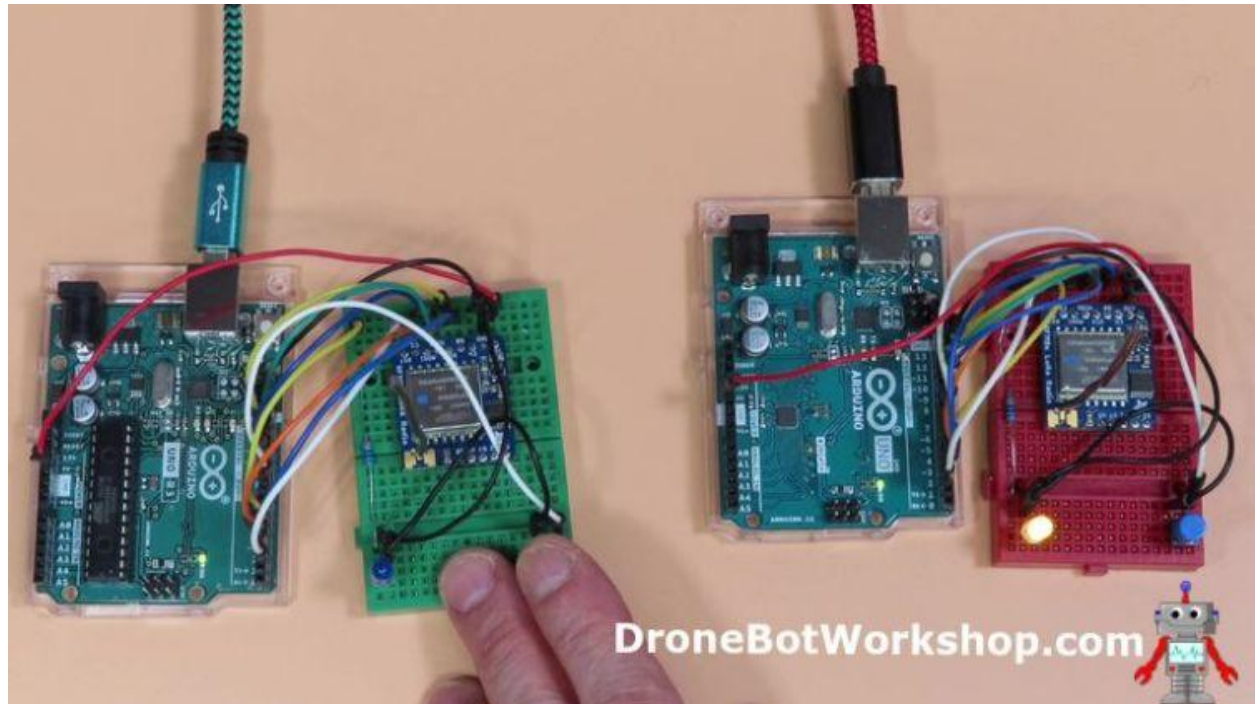
Assuming that all the data is good, the function checks the address to see if it is a broadcast packet or one with the local address. If it isn't, the function exits, as this message is not for us.

If the message is for us, we toggle the button state variable and use it to drive the LED, as we did in the last sketch.

Now, back to the Loop!

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

The Loop is almost identical to the one we used in the last sketch. We read the button, and if it has been pressed, we send a packet and then delay a bit to debounce the pushbutton. The differences are that we use a function to send data this time and intentionally put the LoRa module in receive mode when we finish sending.



Load the sketch to both modules; you can assign them different addresses to experiment with. You should be able to use either pushbutton to toggle the opposite Arduino board's LED.

<https://dronebotworkshop.com>

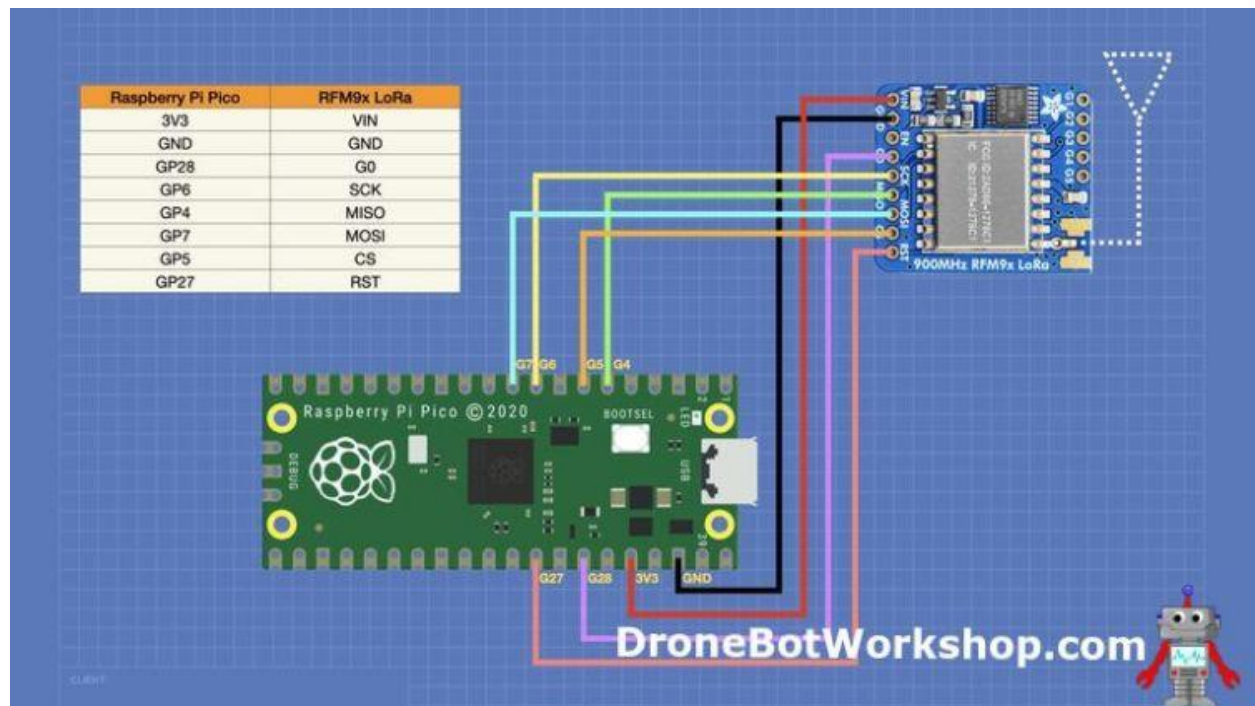


# LoRa with MicroPython

Up until now, all the experiments we have done have been using the Arduino IDE and C++. But we can also use other languages to work with LoRa.

So, we will use the Raspberry Pi Pico running MicroPython for the next experiment. You can substitute any other MicroPython-capable board if you wish.

Here is how we will hook up the Pico:

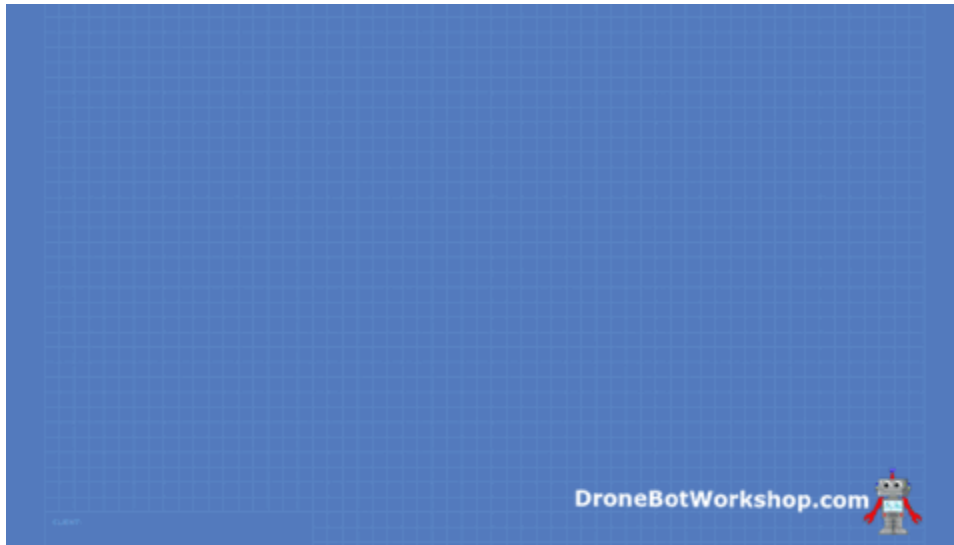


## Installing MicroPython on the Pico

Before we get started, you'll need to grab two Raspberry Pi Pico boards. You can use a Pico, a Pico W, or both. Our experiment won't make use of the WiFi on the Pico W.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

Once you have your boards, you must install the latest version of the MicroPython interpreter onto them. It's a pretty easy process, illustrated in the following animation:



You can get the latest version of the [MicroPython firmware for the Raspberry Pi Pico from the official MicroPython website](#).

Once you have MicroPython installed, you will need a way of working with it. There are several good editors; a popular one and one that is ideal for the Pico is the [Thonny IDE](#). You can get it for Windows, Mac, or Linux.

## ulora MicroPython Library & Examples

Once again, we will rely upon the services of a library to do all the “heavy lifting” and make writing scripts for LoRa very simple.

The [ulora library for MicroPython by Martyn Wheeler](#) is a port of the LoRa Library for Python used with the Raspberry Pi microcomputers. In addition to the Pico, ulora can be used on other MicroPython-capable microcontrollers, such as the ESP32.

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

You can clone the library from GitHub or just download the ZIP and extract it. Once you have the library files, you can examine the three relevant ones.

- `ulora.py` – The library itself.
- `client.py` – The MicroPython script for the client side.
- `server.py` – The MicroPython script for the server side.

These scripts are ready to run on our Pico boards. You need to distribute them as follows:

**Server** – `ulora.py` & `server.py`

**Client** – `ulora.py` & `client.py`

You can use the Thonny IDE to copy the files onto the Pico boards. Just open them from your computer and save them on the Pico.

## MicroPython Client Script

Both scripts are pretty simple. We will begin with the client script, which sends a test message to the server every ten seconds.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
1 from time import sleep
2 from ulora import LoRa, ModemConfig, SPICongig
3
4 # Lora Parameters
5 RFM95_RST = 27
6 RFM95_SPIBUS = SPICongig.rp2_0
7 RFM95_CS = 5
8 RFM95_INT = 28
9 RF95_FREQ = 915.0
10 RF95_POW = 20
11 CLIENT_ADDRESS = 1
12 SERVER_ADDRESS = 2
13
14 # initialise radio
15 lora = LoRa(RFM95_SPIBUS, RFM95_INT, CLIENT_ADDRESS, RFM95_CS, reset_pin=RFM95_RST,
16             freq=RF95_FREQ, tx_power=RF95_POW, acks=True)
17
18
19 # loop and send data
20 while True:
21     lora.send_to_wait("This is a test message", SERVER_ADDRESS)
22     print("sent")
23     sleep(10)
```

The script starts with imports from the time and ulora libraries. We then define the connections to the LoRa module, as well as the module parameters.

Note that on line 9, the LoRa ISM frequency is defined. Make sure that you edit this line for the correct frequency.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

After that, we initialize the LoRa radio module. Note the wealth of parameters you can pass to the module, including its transmit power.

Then, we simply create a loop with a 10-second delay, during which we send our LoRa packet.

Run this script on one Raspberry Pi Pico to send out LoRa data.

## MicroPython Server Script

Now for the script on the server side. On this side, we will receive the data from our client.

```
1 from time import sleep
2 from ulora import LoRa, ModemConfig, SPICongig
3
4 # This is our callback function that runs when a message is received
5 def on_recv(payload):
6     print("From:", payload.header_from)
7     print("Received:", payload.message)
8     print("RSSI: {}; SNR: {}".format(payload.rssi, payload.snr))
9
10 # Lora Parameters
11 RFM95_RST = 27
12 RFM95_SPIBUS = SPICongig.rp2_0
13 RFM95_CS = 5
14 RFM95_INT = 28
15 RF95_FREQ = 868.0
16 RF95_POW = 20
17 CLIENT_ADDRESS = 1
```

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

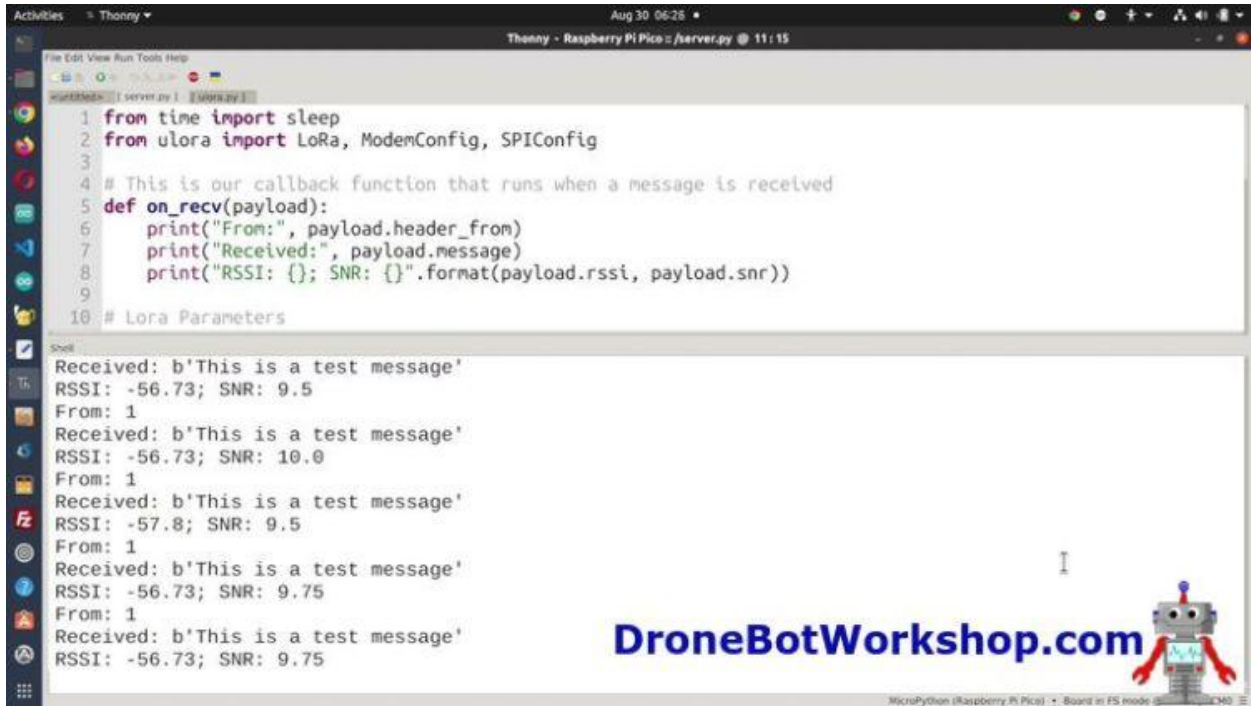
```
18 SERVER_ADDRESS = 2
19
20 # initialise radio
21 lora = LoRa(RFM95_SPIBUS, RFM95_INT, SERVER_ADDRESS, RFM95_CS, reset_pin=RFM95_RST,
22             freq=RF95_FREQ, tx_power=RF95_POW, acks=True)
23
24 # set callback
25 lora.on_recv = on_recv
26
27 # set to listen continuously
28 lora.set_mode_rx()
29
30 # loop and wait for data
31 while True:
32     sleep(0.1)
```

We are using the same libraries we used on the client script. The parameters are also the same; make sure you edit the RF95\_FREQ for your local ISM band frequency.

This script uses a callback similar to the one we used in the last Arduino sketch. In this case, the callback prints the message contents, signal strength, and signal-to-noise ratio.

The rest of the script initializes the LoRa radio module and places it into listen mode.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>



```
1 from time import sleep
2 from ulora import LoRa, ModemConfig, SPIConfig
3
4 # This is our callback function that runs when a message is received
5 def on_recv(payload):
6     print("From:", payload.header_from)
7     print("Received:", payload.message)
8     print("RSSI: {}; SNR: {}".format(payload.rssi, payload.snr))
9
10 # Lora Parameters
```

Shell

```
Received: b'This is a test message'
RSSI: -56.73; SNR: 9.5
From: 1
Received: b'This is a test message'
RSSI: -56.73; SNR: 10.0
From: 1
Received: b'This is a test message'
RSSI: -57.8; SNR: 9.5
From: 1
Received: b'This is a test message'
RSSI: -56.73; SNR: 9.75
From: 1
Received: b'This is a test message'
RSSI: -56.73; SNR: 9.75
```

**DroneBotWorkshop.com**

Load this script to the second Pico and use the shell in the Thonny IDE to observe the printout. You should see data, signal strength, and signal-to-noise ratio, updating every ten seconds.

<https://dronebotworkshop.com>

# Multiple Node LoRa – Remote Environment Sensors

The following project can be viewed in many ways:

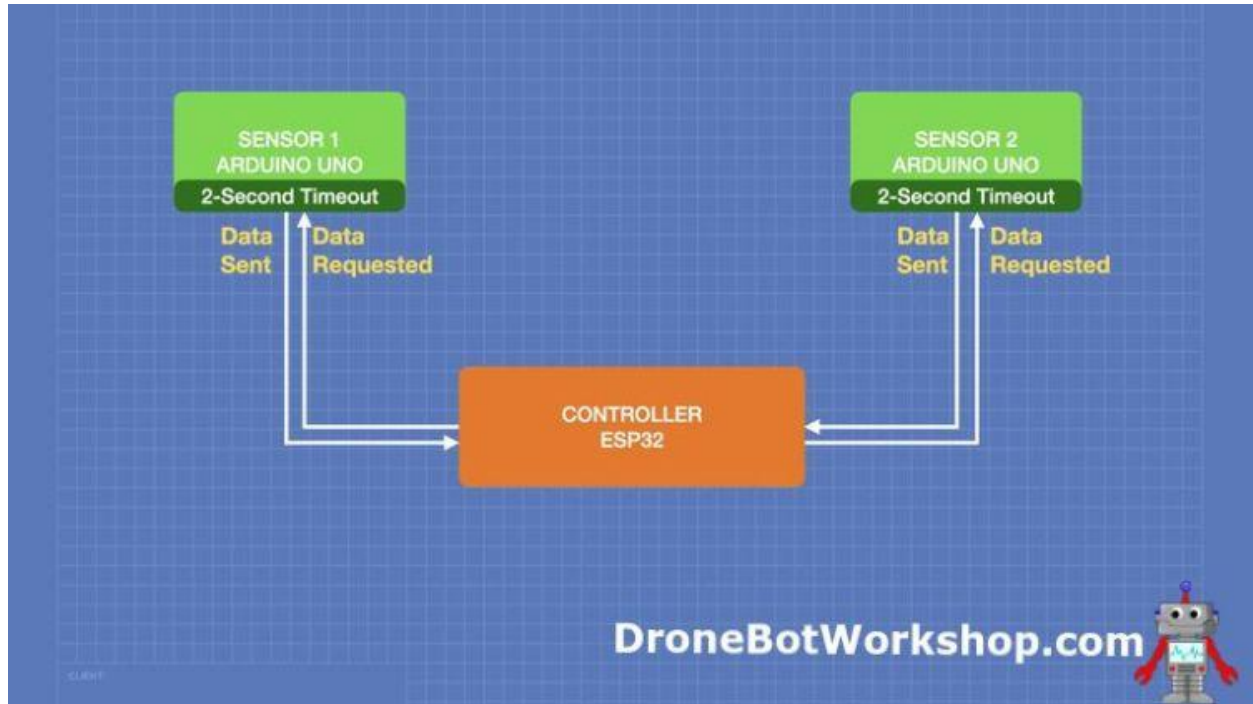
- As an experiment illustrating data-gathering techniques using LoRa.
- As a practical project.
- As the basis for an advanced data-gathering project.

The project itself is pretty basic. It's a controller with an OLED display displaying the temperature and humidity from two remote sensors. Of course, it uses LoRa for communications, so the remote sensors can be very remote indeed!

By itself, this is a useful application or learning tool. But it can also serve as the basis of a more advanced data gathering project, collecting more than just temperature and humidity data.

Let's review how it works. Once you understand its operation, you can modify it for any data-gathering application.





## Principle of Operation

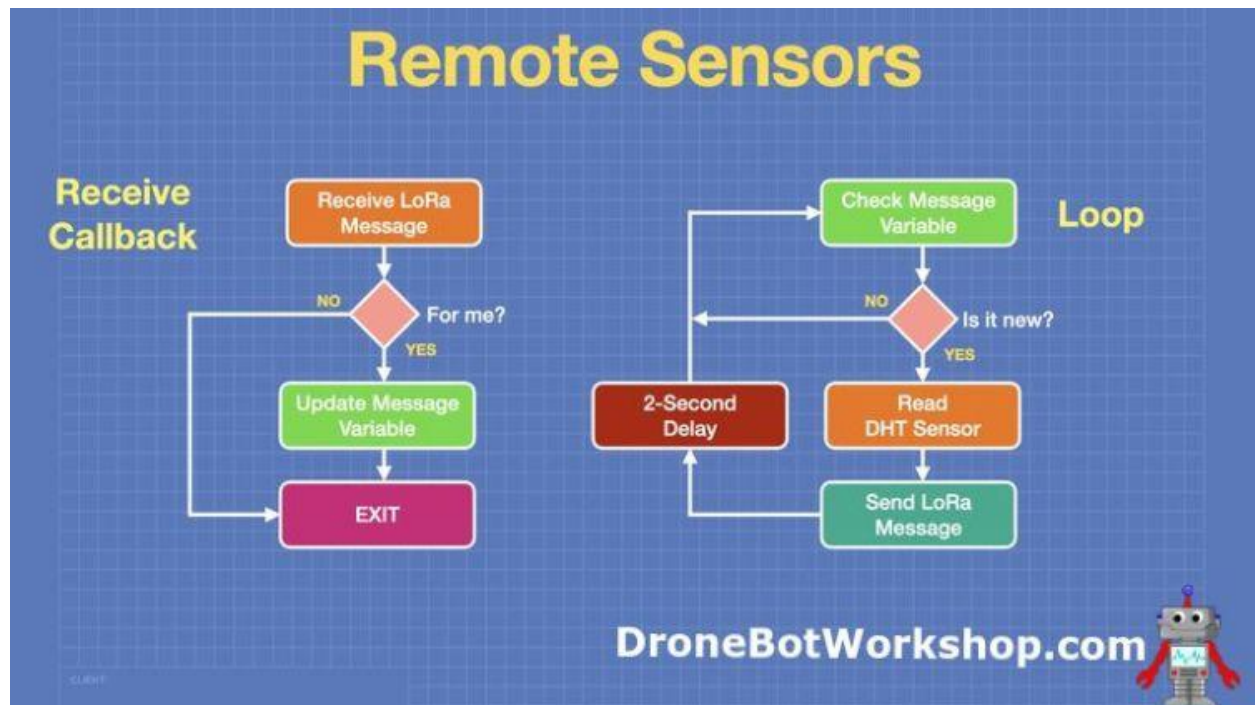
The operation of the system can be broken down into bullet points:

- There is a central controller based on an ESP32
- There are two remote sensors based on Arduino Uno boards.
- The remote devices have DHT22 temperature and humidity sensors.
- The controller polls each board sequentially, once every three seconds.
- The remote responds with a temperature and humidity value when polled.
- After each response, the remote initiates a 2-second timeout to give the DHT22 time to stabilize. During this time, it will ignore requests for new data.
- The central controller displays the temperature and humidity readings on the OLED display. It alternates the display once every three seconds, displaying the data in two different formats.
- If the central controller loses connection with a remote, it displays a series of question marks.

The code relies upon receive callbacks and code in the loop.

## Remote Sensor Operation

This flowchart illustrates the operation of the Receive Callback and Loop code for the remote sensors. This is the code that runs on the Arduino Uno boards.



The receive callback is initiated whenever we receive a message. We check to see if it was meant for us, if it isn't, then we exit the callback.

If the message is for us, we update the "message variable". This is just the packet number received from the controller. We use this variable to see if the request is new.

In the Loop, we check that message variable and compare it to the last one we examined. If it differs, then we know it is a new request. We read the DHT22 sensor and

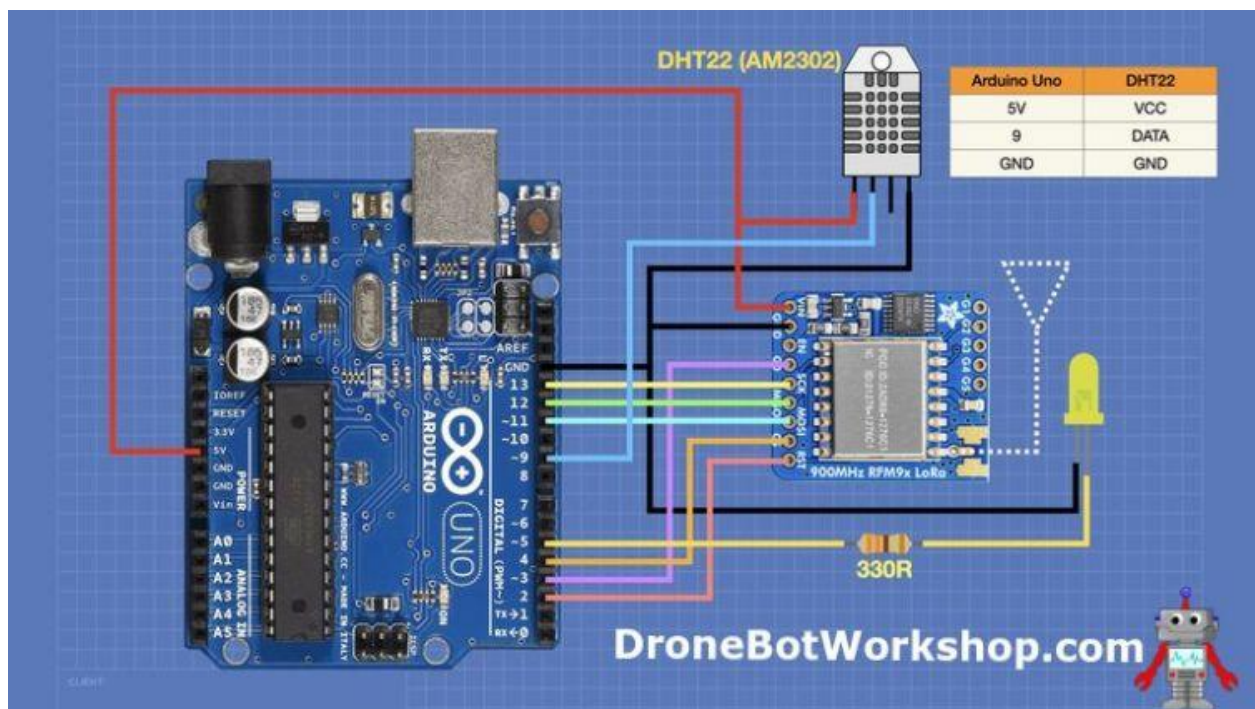
For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

send the temperature and humidity values to the controller in a LoRa packet. Then, we apply a 2-second delay.

The 2-second delay is for the DHT22 sensor, which needs it to stabilize. If we get a new request while the delay is still on, we will ignore it until we finish.

## Remote Wiring

Since we already have two Arduino Uno boards wired to Adafruit RFM9x LoRa radio modules, we can reuse them for this project. We need to add a DHT22 (AM2302) temperature and humidity sensor to pin 9, and that's it.



The LED will be used again in this project, but the pushbutton is unused. You can leave it there, which is what I did. It is removed from the wiring diagram only for clarity.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

## Remote Sensor Sketch (Arduino Uno & DHT22)

The LoRa remote temperature and humidity sensor sketch requires a library for the DHT22 that you may not have installed in your Arduino IDE. You will need to install the [DHTLib Library by Rob Tillaart](#), which you can find in your Arduino IDE Library Manager.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
1  /*
2   LoRa Multi-Sensor Temperature & Humidity Monitor - Sensor
3   lora-temp-humid-sensor.ino
4   Remote Node for LoRa Temperature and Humidity Monitor
5   Uses Arduino Uno, Adafruit RFM9x LoRa Module & DHT22 Temp/Humid sensor
6   Sends Temperature and Humidity data to central controller
7   Requires LoRa Library by Sandeep Mistry -
8   https://github.com/sandeepmistry/arduino-LoRa
9   Requires DHTlib Library by Rob Tillaart
10
11   DroneBot Workshop 2023
12   https://dronebotworkshop.com
13 */
14
15 // Include required libraries
16 #include <SPI.h>
17 #include <LoRa.h>
18 #include <dht.h>
19
20 // DHT22 Connection
21 #define DHTPIN 9
22
23 // LED connection (if used)
24 const int ledPin = 5;
25
26 // DHT Temp/Humid sensor object
27 dht DHT;
28
29 // Define the pins used by the LoRa module
```

<https://dronebotworkshop.com>

```
29 const int csPin = 4;          // LoRa radio chip select
30 const int resetPin = 2;      // LoRa radio reset
31 const int irqPin = 3;        // Must be a hardware interrupt pin
32
33 // Outgoing message variable
34 String outMessage;
35
36 // Controller data variable
37 String inMessage;
38
39 // Previous value Controller data variable
40 String inMessageOld;
41
42 // Outgoing Message counter
43 byte msgCount = 0;
44
45 // Source and destination addresses
46 byte localAddress = 0xAA;    // address of this device (must be unique, 0xAA or
                                0xBB)
47
48 byte destination = 0x01;     // destination to send to (controller = 0x01)
49
50 // Receive Callback Function
51 void onReceive(int packetSize) {
52     if (packetSize == 0) return; // if there's no packet, return
53
54     // Read packet header bytes:
55     int recipient = LoRa.read(); // recipient address
56     byte sender = LoRa.read();   // sender address
57     byte incomingMsgId = LoRa.read(); // incoming msg ID
```

```
57  byte incomingLength = LoRa.read();  // incoming msg length
58
59  String incoming = "";  // payload of packet
60
61  while (LoRa.available()) {          // can't use readString() in callback, so
62      incoming += (char)LoRa.read();  // add bytes one by one
63  }
64
65  if (incomingLength != incoming.length()) {  // check length for error
66      Serial.println("error: message length does not match length");
67      return;  // skip rest of function
68  }
69
70  // If the recipient isn't this device or broadcast,
71  if (recipient != localAddress && recipient != 0xFF) {
72      Serial.println("This message is not for me.");
73      return;  // skip rest of function
74  }
75
76  // If we are this far then this message is for us
77  // Update the controller data variable
78  inMessage = incoming;
79 }
80
81 // Send LoRa Packet
82 void sendMessage(String outgoing) {
83     LoRa.beginPacket();          // start packet
84     LoRa.write(destination);     // add destination address
```

```
85  LoRa.write(localAddress);      // add sender address
86  LoRa.write(msgCount);         // add message ID
87  LoRa.write(outgoing.length()); // add payload length
88  LoRa.print(outgoing);         // add payload
89  LoRa.endPacket();             // finish packet and send it
90  msgCount++;                   // increment message ID
91 }
92
93 void setup() {
94
95   Serial.begin(9600);
96   while (!Serial)
97     ;
98
99   // Set LED as output (if used)
100  pinMode(ledPin, OUTPUT);
101
102  // Setup LoRa module
103  LoRa.setPins(csPin, resetPin, irqPin);
104
105  // Start LoRa module at local frequency
106  // 433E6 for Asia
107  // 866E6 for Europe
108  // 915E6 for North America
109  if (!LoRa.begin(915E6)) {
110    Serial.println("Starting LoRa failed!");
111    while (1)
112      ;
```



```
113 }  
114  
115 // Set Receive Call-back function  
116 LoRa.onReceive(onReceive);  
117  
118 // Place LoRa in Receive Mode  
119 LoRa.receive();  
120  
121 Serial.println("LoRa init succeeded.");  
122 }  
123  
124 void loop() {  
125  
126 // Run only if requested  
127 if (inMessage != inMessageOld) {  
128 // New message variable, take reading and send to controller  
129  
130 int readData = DHT.read22(DHTPIN); // Reads the data from the sensor  
131 float t = DHT.temperature; // Gets the values of the temperature  
132 float h = DHT.humidity; // Gets the values of the humidity  
133  
134 // Printing the results on the serial monitor  
135 Serial.print("Temperature = ");  
136 Serial.print(t);  
137 Serial.print(" *C ");  
138 Serial.print(" Humidity = ");  
139 Serial.print(h);  
140 Serial.println(" % ");
```

```
141
142     // Format the outgoing message string
143     String outMsg = "";
144     outMsg = outMsg + t + ":" + h;
145
146     // Send data as LoRa packet
147     sendMessage(outMsg);
148
149     // Print controller variables
150     Serial.print("Old Controller Data = ");
151     Serial.println(inMessageOld);
152     Serial.print("New Controller Data = ");
153     Serial.println(inMessage);
154
155     // Update the "old" data variable
156     inMessageOld = inMessage;
157
158     // Place LoRa in Receive Mode
159     LoRa.receive();
160
161     // Optional 2-second LED pulse (remark out if LED not used)
162     digitalWrite(ledPin, HIGH);
163     Serial.println("led on");
164
165     // 2-second delay for DHT sensor
166     delay(2000);
167
168     // Optional 2-second LED pulse (remark out if LED not used)
```

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
169     digitalWrite(ledPin, LOW);  
170     Serial.println("led off");  
171 }  
172 }
```

The sketch begins by including the required libraries. We also define the pins for the DHT22 sensor and LoRa module connections.

We define a number of strings to hold message variables. One for the outgoing message and two for the incoming message. The incoming message is the packet number sent by the controller, and we will save the most recent one in a variable so we can use it to determine if the controller request is fresh.

**Note line 46 with the local address. You will need to change this on one remote sensor. Keep one sensors code at 0xAA, and change the second one to 0xBB.**

This is the sensor's identity code and is the ONLY difference between the two remote sensors.

The Receive Callback function *onReceive* checks the validity of the incoming data packet. If it is a valid packet intended for us, then we update the local message variable with the packet contents.

The *sendMessage* function is identical to what we saw in the two-way LED control example; it just uses a number of LoRa library functions to build and send a LoRa packet.

In the Setup, we initialize the LoRa Module, set the LED as an output, and put LoRa into receive mode.

In the Loop, we constantly scan for the value of the message variable and compare it to the one we have. If it matches, then there is no new request from the controller. But if

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

there is a mismatch, we take the temperature and humidity readings and print them to the serial monitor.

We then format a string to send to the controller, essentially a colon-delimited string with temperature and humidity. We send that string using the *sendMessage* function.

After that, we implement a 2-second delay, during which we will turn on the LED. This results in the LED flashing on for two seconds periodically, which makes a good indication that the sensor is connected to the controller.

As a recap, before you load the sketch to an Arduino, check the following:

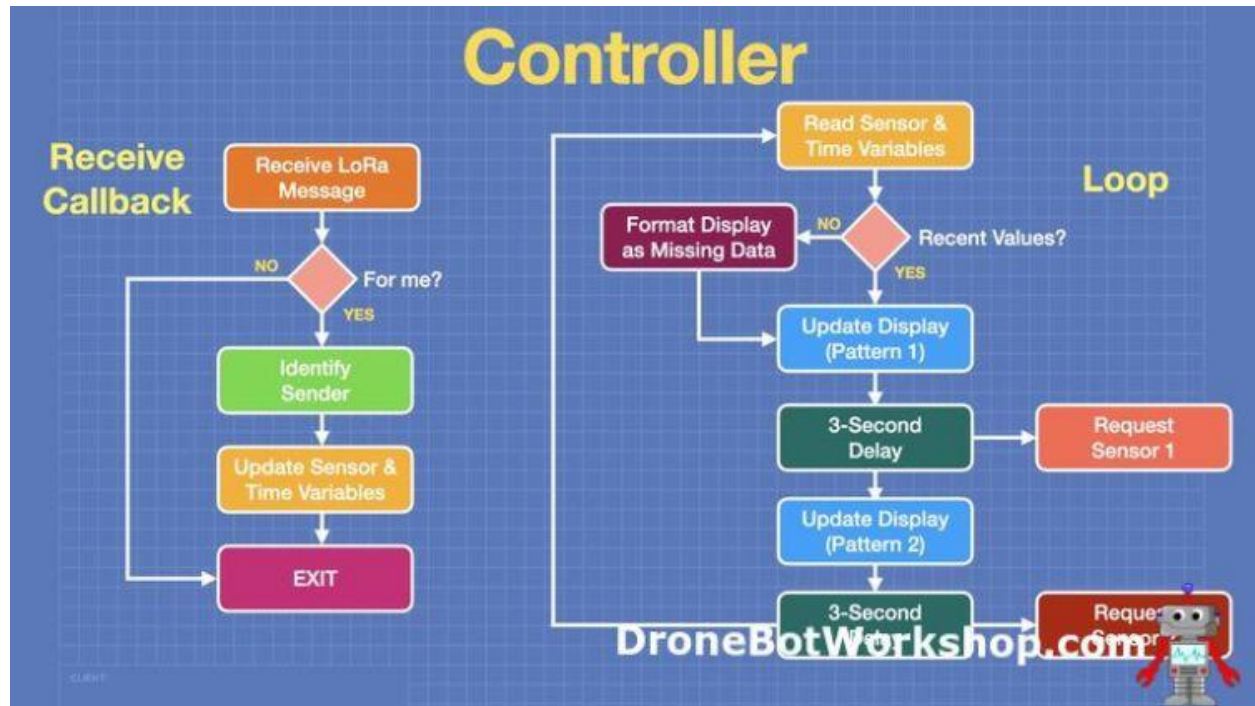
- The LoRa frequency (line 109)
- The Sensor Address (line 46)

After verifying that, load the sketch to the first Arduino. Unplug it, plug in the second one, change the address on line 46, and upload the edited sketch to the second Arduino.

## Controller Operation

Now, it's time to focus on the controller, which is based on an ESP32. You can use any ESP32 module, and we are not using WiFi or Bluetooth, so you could also use another microcontroller. If you do change processors, you'll need to note the default SPI and I2C connections, as both are used in this design.

The following diagram shows the information flow in both the Receive Callback and Loop of the controller code:



When the Receive Callback is initiated, we check to see if the message is for us. If it isn't, we exit the callback.

If the message is for us, we identify the sender to know which remote unit is sending us data. We then update the local temperature and humidity variables for that sensor.

We also update a timestamp variable for that sensor; this tells us how fresh the data is. Then, we exit the callback.

In the Loop, we check the sensor and timestamp variables. If the data is fresh, we update the OLED display with it; if it is old, we change the display to indicate that a remote sensor is offline.

We wait for 3 seconds and then change the display format. We also send a request to the first sensor for new data.

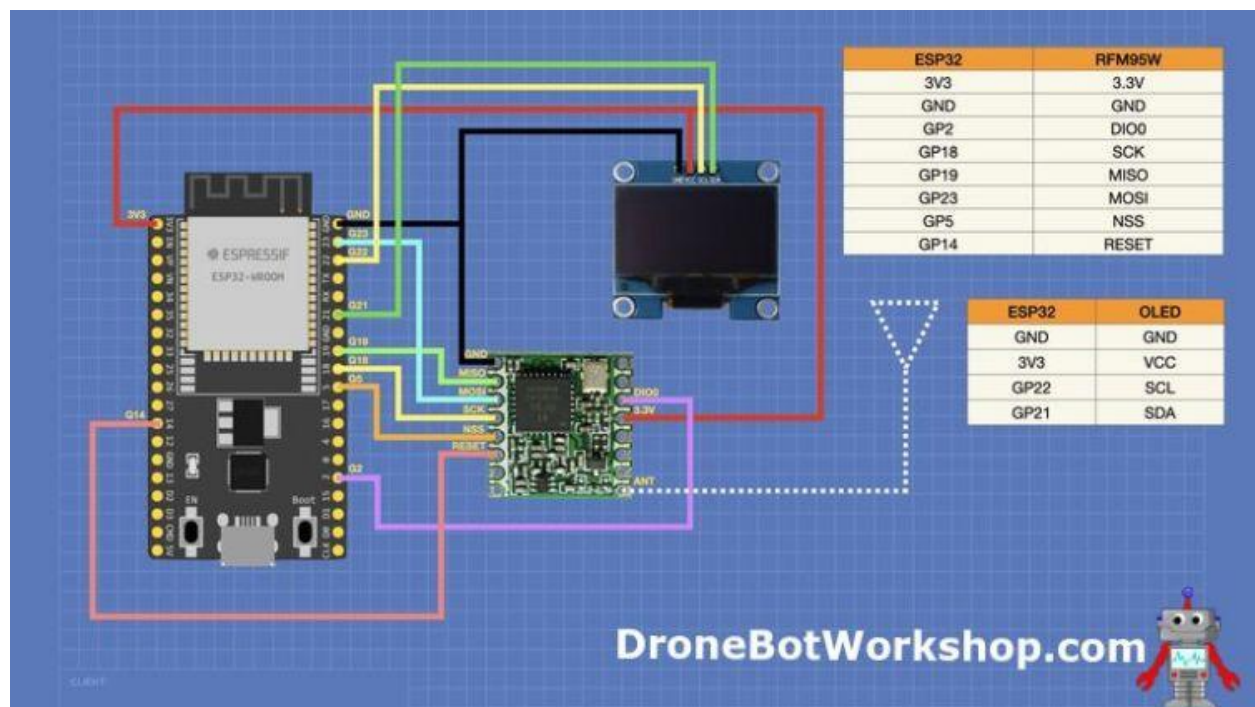
For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

We then wait three seconds and send a second request. Then the Loop ends, and we do it all over again.

This method lets us stagger the data requests to the sensors, so the chances of data returning from the sensors colliding are nil. It also allows the sensors plenty of time to execute that 2-second delay the DHT22 needs.

## Controller Wiring

The controller uses an ESP32, an OLED display, and the HopeRF RFM95W module. Its hookup is shown here:



You can substitute the Adafruit LoRa module for the HopeRF one; they are operationally equivalent.

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

## Controller Sketch (ESP32 & OLED)

The controller sketch requires the Adafruit SSD1306 and GFX libraries. You can install these from the Library Manager if you need to. They are pretty common libraries, so you may already have one or both already installed.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
1  /*
2   LoRa Multi-Sensor Temperature and Humidity Monitor - Controller
3   lora-temp-humid-control.ino
4   Central Controller for LoRa Temperature and Humidity Monitor
5   Uses ESP32, RFM95W LoRa & SSD1306 I2C OLED Display
6   Displays Temperature and Humidity readings from remote sensors
7   Requires LoRa Library by Sandeep Mistry -
8   https://github.com/sandeepmistry/arduino-LoRa
9   Requires Adafruit GFX and SSD1306 libraries
10
11   DroneBot Workshop 2023
12   https://dronebotworkshop.com
13 */
14
15 // Include required libraries
16 #include <SPI.h>
17 #include <LoRa.h>
18 #include <Wire.h>
19 #include <Adafruit_GFX.h>
20 #include <Adafruit_SSD1306.h>
21
22 // Define the pins used by the LoRa module
23 const int csPin = 5;      // LoRa radio chip select
24 const int resetPin = 14;  // LoRa radio reset
25 const int irqPin = 2;     // Must be a hardware interrupt pin
26
27 // Source and sensorAddress1 addresses
28 byte localAddress = 0x01; // Address of this device (Controller = 0x01)
29 byte sensorAddress1 = 0xAA; // Address of Sensor 1
```

<https://dronebotworkshop.com>



For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
29 byte sensorAddress2 = 0xBB; // Address of Sensor 2
30
31 // OLED parameters
32 #define SCREEN_WIDTH 128 // OLED display width, in pixels
33 #define SCREEN_HEIGHT 64 // OLED display height, in pixels
34 #define OLED_RESET -1 // Reset pin # (or -1 if sharing Arduino reset pin)
35 #define SCREEN_ADDRESS 0x3C // Change if required
36
37 // Define display object
38 Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
39
40 // Remote temperature and humidity variables
41 // Data variable
42 String remoteData1 = "TT.tt:HH.hh";
43 String remoteData2 = "TT.tt:HH.hh";
44
45 // Sensor 1
46 String remoteTemp1;
47 String remoteHumid1;
48 // Sensor 2
49 String remoteTemp2;
50 String remoteHumid2;
51
52 // Remote sensor time variables
53 unsigned long currentActive1 = millis();
54 unsigned long currentActive2 = millis();
55 const long checkInterval = 12500; // 12.5 second sensor check interval
56
```

<https://dronebotworkshop.com>

```
57 // Outgoing Message counter
58 byte msgCount = 0;
59
60 // FUNCTION newDisplay() - Refresh the display with new data
61 void newDisplay(String temp1, String humid1, String temp2, String humid2, int
62   displayOrder) {
63
64   // Print display header
65   display.clearDisplay();
66   display.setTextColor(WHITE);
67   display.setTextSize(1);
68   display.setCursor(0, 0);
69   display.print("REMOTE TEMP & HUMID");
70
71   // If displayOrder = 1 then reverse display order
72
73   if (displayOrder == 1) {
74
75     // Remote Sensor 2 is first
76
77     display.setTextSize(2);
78     display.setCursor(0, 16);
79     display.print("T2: ");
80     display.setCursor(38, 16);
81     display.print(temp2);
82     display.print("C");
83
84     display.setCursor(0, 34);
85     display.print("H2: ");
```

```
85     display.setCursor(38, 34);
86     display.print(humid2);
87     display.print("%");
88
89     // Node 1 in smaller font
90
91     display.setTextSize(1);
92
93     display.setCursor(0, 55);
94     display.print("T1: ");
95     display.setCursor(18, 55);
96     display.print(temp1);
97     display.print("C");
98
99     display.setCursor(60, 55);
100    display.print("H1: ");
101    display.setCursor(78, 55);
102    display.print(humid1);
103    display.print("%");
104
105 } else {
106
107     // Remote Sensor 1 1 is first
108
109     display.setTextSize(2);
110     display.setCursor(0, 16);
111     display.print("T1: ");
112     display.setCursor(38, 16);
```

```
113     display.print(temp1);
114     display.print("C");
115
116     display.setCursor(0, 34);
117     display.print("H1: ");
118     display.setCursor(38, 34);
119     display.print(humid1);
120     display.print("%");
121
122     // Node 2 in smaller font
123
124     display.setTextSize(1);
125
126     display.setCursor(0, 55);
127     display.print("T2: ");
128     display.setCursor(18, 55);
129     display.print(temp2);
130     display.print("C");
131
132     display.setCursor(60, 55);
133     display.print("H2: ");
134     display.setCursor(78, 55);
135     display.print(humid2);
136     display.print("%");
137 }
138
139 display.display();
140 }
```

```
141
142 // FUNCTION getValue() - Extract value from delimited string
143 String getValue(String data, char separator, int index) {
144     int found = 0;
145     int strIndex[] = { 0, -1 };
146     int maxIndex = data.length() - 1;
147
148     for (int i = 0; i <= maxIndex && found <= index; i++) {
149         if (data.charAt(i) == separator || i == maxIndex) {
150             found++;
151             strIndex[0] = strIndex[1] + 1;
152             strIndex[1] = (i == maxIndex) ? i + 1 : i;
153         }
154     }
155     return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
156 }
157
158 // FUNCTION onReceive() - Receive call-back function
159 void onReceive(int packetSize) {
160     if (packetSize == 0) return; // if there's no packet, return
161
162     // read packet header bytes:
163     int recipient = LoRa.read(); // recipient address
164     byte sender = LoRa.read(); // sender address
165     byte incomingMsgId = LoRa.read(); // incoming msg ID
166     byte incomingLength = LoRa.read(); // incoming msg length
167
168     String incoming = ""; // payload of packet
```

```
169
170 while (LoRa.available()) {           // can't use readString() in callback, so
171     incoming += (char)LoRa.read();    // add bytes one by one
172 }
173
174 if (incomingLength != incoming.length()) { // check length for error
175     Serial.println("error: message length does not match length");
176     return; // skip rest of function
177 }
178
179 // if the recipient isn't this device
180 if (recipient != localAddress) {
181     Serial.println("This message is not for me.");
182     return; // skip rest of function
183 }
184
185 // Determine sender, then update data variables and time stamps
186 if (sender == sensorAddress1) {
187     //Remote Sensor 1
188     remoteData1 = incoming;
189     currentActive1 = millis();
190 } else if (sender == sensorAddress2) {
191     //Remote Sensor 2
192     remoteData2 = incoming;
193     currentActive2 = millis();
194 }
195 }
196
```

```
197 // FUNCTION sendMessage() - Send LoRa Packet
198 void sendMessage(String outgoing, byte target) {
199     LoRa.beginPacket();           // start packet
200     LoRa.write(target);           // add sensorAddress1 address
201     LoRa.write(localAddress);     // add sender address
202     LoRa.write(msgCount);         // add message ID
203     LoRa.write(outgoing.length()); // add payload length
204     LoRa.print(outgoing);         // add payload
205     LoRa.endPacket();             // finish packet and send it
206     msgCount++;                  // increment message ID
207 }
208
209 // FUNCTION getValues() - get the temperature and humidity values from the data
210 // variables
211 void getValues() {
212     // Check to see if sensors have reported in recently
213     // Get current timestamp value
214     unsigned long currentMillis = millis();
215
216     // See if we have exceeded the check interval time limit
217     // Sensor 1
218     if (currentMillis - currentActive1 <= checkInterval) {
219         // Data is good, extract temp and humid
220         remoteTemp1 = getValue(remoteData1, ':', 0); // Remote 1 Temperature
221         remoteHumid1 = getValue(remoteData1, ':', 1); // Remote 1 Humidity
222     } else {
223         remoteTemp1 = "??.??";
224         remoteHumid1 = "??.??";
225     }
226 }
```

```
225 // Sensor 2
226 if (currentMillis - currentActive2 <= checkInterval) {
227     // Data is good, extract temp and humid
228     remoteTemp2 = getValue(remoteData2, ':', 0); // Remote 1 Temperature
229     remoteHumid2 = getValue(remoteData2, ':', 1); // Remote 1 Humidity
230 } else {
231     remoteTemp2 = "??.??";
232     remoteHumid2 = "??.??";
233 }
234 }
235
236 void setup() {
237     Serial.begin(9600);
238     while (!Serial)
239         ;
240
241     // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
242     if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
243         Serial.println(F("SSD1306 allocation failed"));
244         for (;;)
245             ; // Don't proceed, loop forever
246     }
247
248     // Clear the display buffer
249     display.clearDisplay();
250     display.display();
251
252     // Refresh OLED
```



```
253  newDisplay("XX.XX", "XX.XX", "XX.XX", "XX.XX", 0);
254
255  // Setup LoRa module
256  LoRa.setPins(csPin, resetPin, irqPin);
257
258  Serial.println("LoRa Receiver Test");
259
260  // Start LoRa module at local frequency
261  // 433E6 for Asia
262  // 866E6 for Europe
263  // 915E6 for North America
264
265  if (!LoRa.begin(915E6)) {
266    Serial.println("Starting LoRa failed!");
267    while (1)
268      ;
269  }
270
271  LoRa.onReceive(onReceive);
272  LoRa.receive();
273  Serial.println("LoRa init succeeded.");
274 }
275
276 void loop() {
277
278  // Get latest data values
279  getValues();
280
```

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
281 Serial.print("Temp 1: ");
282 Serial.print(remoteTemp1);
283 Serial.print(" - Humid 1: ");
284 Serial.println(remoteHumid1);
285
286 Serial.print("Temp 2: ");
287 Serial.print(remoteTemp2);
288 Serial.print(" - Humid 2: ");
289 Serial.println(remoteHumid2);
290
291 // Update OLED
292 newDisplay(remoteTemp1, remoteHumid1, remoteTemp2, remoteHumid2, 0);
293
294 // Delay 3 seconds to hold display
295 delay(3000);
296
297 // Send message to remote 1
298 String outMsg1 = "";
299 outMsg1 = outMsg1 + msgCount;
300 sendMessage(outMsg1, sensorAddress1);
301
302 // Place LoRa back into Receive Mode
303 LoRa.receive();
304
305 // Refresh the data values
306 getValues();
307
308 // Update OLED - reverse display
```

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
309  newDisplay(remoteTemp1, remoteHumid1, remoteTemp2, remoteHumid2, 1);
310
311  // Delay 3 seconds to hold display
312  delay(3000);
313
314  // Send message to remote 2
315  String outMsg2 = "";
316  outMsg2 = outMsg2 + msgCount;
317  sendMessage(outMsg2, sensorAddress2);
318
319  // Place LoRa back into Receive Mode
320  LoRa.receive();
321 }
```

Aside from the Adafruit libraries, which are for the OLED display, we require the SPI, I2C, and LoRa libraries.

After defining the LoRa module connections, we define the local and remote addresses as follows:

- Local Address – 0x01
- Sensor 1 Address – 0xAA
- Sensor 2 Address – 0xBB

Don't change these values. They need to match the values on the remote sensors.

We define some parameters for the OLED display and then create an object to represent it.

We create two data variable strings, one for each sensor. Likewise, we also create variables to store the temperature and humidity values for both remote sensors.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

Now for a few functions! The first one uses the temperature and humidity data to format and display data on the OLED. It can output in two modes:

- **Mode 1** – Sensor 1 Large Font, Sensor 2 Small Font
- **Mode 2** – Sensor 2 Large Font, Sensor 1 Small Font

The next function, *getValue*, extracts the characters from the colon-delimited string. It extracts them individually, and you specify if you want to extract the characters before or after the colon.

The receive Callback does the usual check to the packet. If it is a valid one for us, then it checks the sender's address to see who sent it. It then updates the data variable and timestamp for that sensor.

The *sendMessage* function is the same one we have used before.

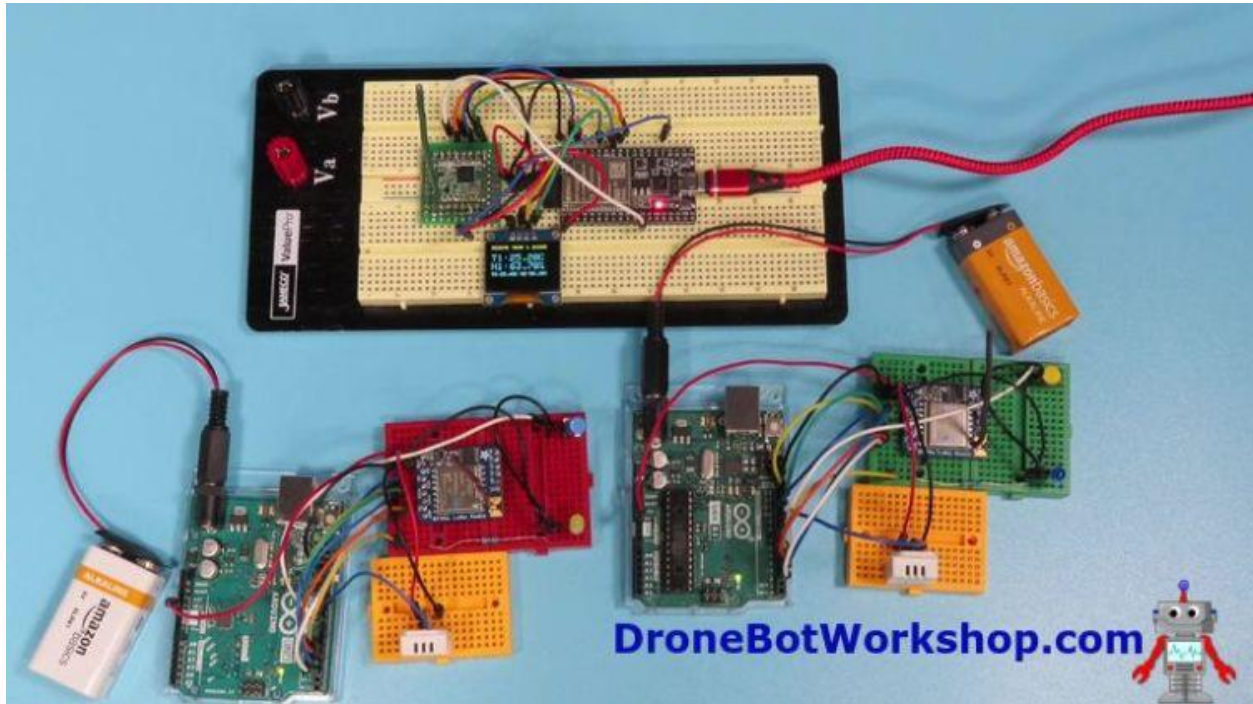
The *getValues* function uses the data variables to get the actual temperature and humidity values. It also tests the data age; if it is too old, it changes the display to all question marks.

A lot of the Setup is to initialize the OLED display with all "X", which will be what we see when we start the controller. We also do the usual LoRa setup; again, remember to set the frequency correctly for your local ISM band.

In the Loop we get the values for temperature and humidity and print them on the display in mode 1. We then wait three seconds, send out a LoRa message for sensor 1 data, and change display to mode 2. After another three seconds, we send out a request for sensor 2 data and end the Loop.

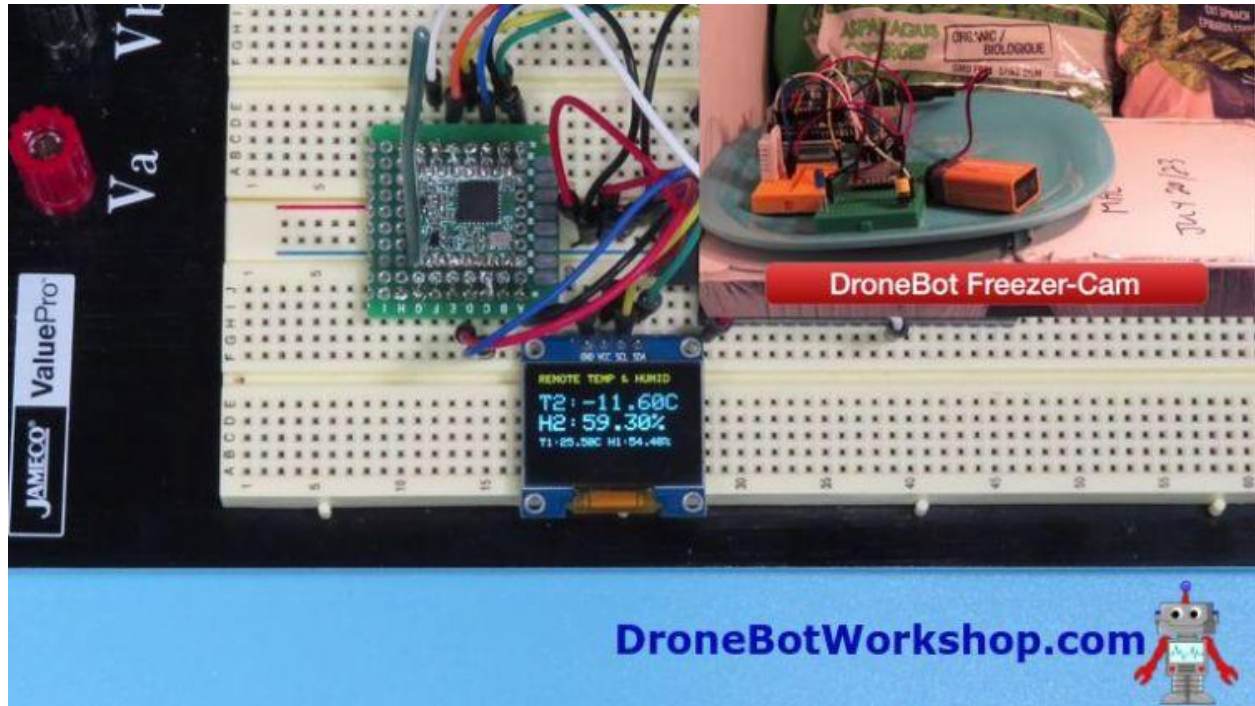
For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

Once you have all the code loaded, give it a try. The controller should boot up with all “X” displays, followed by temperature and humidity from both sensors. The sensors should have a flashing LED.



Try and see how far you can get for range. I was able to get two blocks with one of my remote sensors in the freezer in my basement! All this with a 3-inch wire antenna!

<https://dronebotworkshop.com>



## Conclusion

LoRa is an amazing technology that will really make gathering data from remote sensors simple.

The LoRa radio modules I used today are by no means the only ones available to experimenters, and new modules and microcontrollers with integrated LoRa are being introduced frequently. LoRa is here to stay!

Keep your eyes glued to this website and the YouTube channel, as the next time we visit LoRa we will be working with LoRaWAN, a protocol that allows for sophisticated and long-range LoRa networking.

Until then, enjoy working with the experiments. See what kind of range you can get with your LoRa setup, and let us all know in the comments!