# LoRa – Internal project

Task documentation

Introduction - LoRa:

LoRa is a wireless modulation technique derived from Chirp Spread Spectrum (CSS) technology. It encodes information on radio waves using chirp pulses - similar to the way dolphins and bats communicate. LoRa modulated transmission is robust against disturbances and can be received across great distances.

LoRa module: SX1262 (Transceiver)

Datasheet:
https://drive.google.com/drive/folders/12BDdpUGZjJlHjT2dPzhDq6xYcmyNpv5C

Task 1: To achieve basic communication through two LoRa modules

Task 2: To integrate a sensor in one LoRa module and receive its output in another and to check the Transceiver functionality by sending back an acknowledgement of output reception.

## **Task 1: To achieve basic communication through two LoRa modules**

The basic communication is achieved by configuring one LoRa module as transmitter and another as Receiver.

Controller for LoRa module integration: ESP32 Lilygo(Dev board)

Communication protocol: SPI

Power supply to the module: 5V (through ESP32)

Transmitter session pin connections are as follows.

| **SX1262** | **ESP32 Lilygo** |
|---|---|
| CS | 5 |
| DI01 | 34 |
| NRST | 32 |
| BUSY | 33 |
| MISO | 19 |

|  |  |
|---|---|
| MOSI | 23 |
| CLK | 18 |
| GND | GND |
| 5V | 5V-OUT |

The same applicable for receiver session as well.

Basic communication test is carried out by sending a string from transmitter and checking for its reception in receiver, the firmware code covering the configuration and application are as follows:

## Firmware

IDE of testing: Arduino IDE

Library used: RadioLib.h // https://www.arduino.cc/reference/en/libraries/radiolib/

## Transmitter:

```
// include the library
#include <RadioLib.h>

    // SX1262 has the following connections:
    // NSS pin:   5
    // DIO1 pin:  34
    // NRST pin:  32
    // BUSY pin:  33
SX1262 radio = new Module(5, 34, 32, 35);

void setup() {

//intiliazing serial communication
  Serial.begin(9600);

  // initialize SX1262 with default settings
  Serial.print(F("[SX1262] Initializing ... "));

// select appropriate frequency as per the country
  int state = radio.begin(868.0, 250.0, 7, 5, 0x34, 20, 10, 0, false);
  if (state == RADIOLIB_ERR_NONE) {
    Serial.println(F("success!"));
  } else {
    Serial.print(F("failed, code "));
    Serial.println(state);
    while (true);
  }
}

void loop() {
  Serial.print(F("[SX1262] Transmitting packet ... "));


// Transmitting a string to the receiver in same band of frequency
int state = radio.transmit("Hello World!");

 if (state == RADIOLIB_ERR_NONE) {
    // the packet was successfully transmitted
    Serial.println(F("success!"));

// print measured data rate
    Serial.print(F("[SX1262] Datarate:\t"));
    Serial.print(radio.getDataRate());
    Serial.println(F(" bps"));

} else if (state == RADIOLIB_ERR_PACKET_TOO_LONG) {
    // the supplied packet was longer than 256 bytes
    Serial.println(F("too long!"));

  } else if (state == RADIOLIB_ERR_TX_TIMEOUT) {
    // timeout occured while transmitting packet
    Serial.println(F("timeout!"));

  } else {
    // some other error occurred
    Serial.print(F("failed, code "));
    Serial.println(state);

  }

  // wait for a second before transmitting again
  delay(1000);
}
```

## Receiver:

```
// include the library
#include <RadioLib.h>

/ SX1262 has the following connections:
// NSS pin:   5
// DIO1 pin:  34
// NRST pin:  32
// BUSY pin:  33
SX1262 radio = new Module(5, 34, 32, 35);



void setup() {
  Serial.begin(9600);


  Serial.print(F("[SX1262] Initializing ... "));

// select appropriate frequency as per the country
  int state = radio.begin(868.0, 250.0, 7, 5, 0x34, 20, 10, 0, false);

  if (state == RADIOLIB_ERR_NONE) {
    Serial.println(F("success!"));
  } else {
    Serial.print(F("failed, code "));
    Serial.println(state);
    while (true);
  }
}

void loop() {
  Serial.print(F("[SX1262] Waiting for incoming transmission ... "));


  String str;
  int state = radio.receive(str);


  if (state == RADIOLIB_ERR_NONE) {
    // packet was successfully received
    Serial.println(F("success!"));
    // print the data of the packet
    Serial.print(F("[SX1262] Data:\t\t"));
    Serial.println(str);

    Serial.print(F("[SX1262] RSSI:\t\t"));
    Serial.print(radio.getRSSI());
    Serial.println(F(" dBm"));


    Serial.print(F("[SX1262] SNR:\t\t"));
    Serial.print(radio.getSNR());
    Serial.println(F(" dB"));

  } else if (state == RADIOLIB_ERR_RX_TIMEOUT) {
    // timeout occurred while waiting for a packet
    Serial.println(F("timeout!"));

  } else if (state == RADIOLIB_ERR_CRC_MISMATCH) {
    // packet was received, but is malformed
    Serial.println(F("CRC error!"));

  } else {
    // some other error occurred
    Serial.print(F("failed, code "));
    Serial.println(state);

  }
}
```

**Serial monitor outputs:**

Transmitter:

Receiver:

**Task 2:**

**To integrate a sensor in one LoRa module and receive its output in another.**

**To test the transceiver functionality by sending back an acknowledgement of output reception.**

Controller for LoRa module integration: ESP32

Communication protocol: SPI, I2C

Power supply to the module: 5V (through ESP32)

Sensor + ESP32 + LoRa module pin connections are as follows, (Transmitter)

| **SX1262** | **ESP32 Lilygo** |
|---|---|
| CS | 5 |
| DI01 | 34 |
| NRST | 32 |
| BUSY | 33 |
| MISO | 19 |
| MOSI | 23 |
| CLK | 18 |
| GND | GND |
| 5V | 5V-OUT |
| **LiDAR Sensor** | **ESP32 Lilygo** |
| Vin | 3.3V |
| GND | GND |
| SCL | 22 |
| SDA | 21 |

ESP32+LoRa module(Receiver) pin connections as follows,

| **SX1262** | **ESP32 Lilygo** |
|---|---|
| CS | 5 |
| DI01 | 34 |
| NRST | 32 |
| BUSY | 33 |

|  |  |
|---|---|
| MISO | 19 |
| MOSI | 23 |
| CLK | 18 |
| GND | GND |
| 5V | 5V-OUT |

## Firmware

IDE of testing: Arduino IDE

Library used: RadioLib.h // https://www.arduino.cc/reference/en/libraries/radiolib/

**Sensor+Esp32+LoRa(transmitter ) firmware code :**

TMF882X liDAR sensor is integrated with ESP 32 using I2C protocol and the acquired value is sent as a string through LoRa module .Then command is initiated to wait for reception of acknowledgement from the receiver. Once the acknowledgement is received. Next set of values are transmitted to the receiver.

```
// Peer to Peer: Monitor Sensor Data using LoRa - Transmitter

// Include necessary libraries
#include <RadioLib.h>                  // Transmit & receive -
https://github.com/jgromes/RadioLib/archive/refs/heads/master.zip
#include"SparkFun_TMF882X_Library.h"  //http://librarymanager/All#SparkFun_Qwiic_
TMPF882X



    // SX1262 has the following connections:
    // NSS pin:   5
    // DIO1 pin:  34
    // NRST pin:  32
    // BUSY pin:  33
SX1262 radio = new Module(5, 34, 32, 35);

SparkFun_TMF882X myTMF882X;


// Structure to hold the measurement results - this is defined by the TMF882X
SDK.
```

```
static struct tmf882x_msg_meas_results myResults;

void setup() {
  delay(1000);
  Serial.begin(115200);
  Serial.println("");

  Serial.println("In setup");
  Serial.println("==============================");

  // Initialize the TMF882X device
  if (!myTMF882X.begin()) {
    Serial.println("Error - The TMF882X failed to initialize - is the board
connected?");
    while (1)
      ;
  } else
    Serial.println("TMF882X started.");

  // The device is now ready for operations

  // Initialize SX1262 with default settings
  Serial.print(F("[SX1262] Initializing ... "));

  int state = radio.begin(868.0, 250.0, 7, 5, 0x34, 20, 10, 0, false);


  if (state == RADIOLIB_ERR_NONE) {
    Serial.println(F("success!"));
  } else {
    Serial.print(F("failed, code "));
    Serial.println(state);
    while (true) ;
  }
}

void loop() {


  Serial.println(F("[SX1262] Transmitting packet ... "));

  if (myTMF882X.startMeasuring(myResults)) {
    // print out results
    Serial.println("Measurement:");
    Serial.print("     Result Number: ");
    Serial.print(myResults.result_num);
    Serial.print("  Number of Results: ");
    Serial.println(myResults.num_results);
    Serial.print("      conf: ");
    Serial.print(myResults.results[4].confidence);
    Serial.print(" distance mm: ");
    Serial.print(myResults.results[4].distance_mm);
    Serial.print(" channel: ");
    Serial.print(myResults.results[4].channel);
    Serial.print(" sub_capture: ");
    Serial.println(myResults.results[4].sub_capture);
    Serial.print("     photon: ");
    Serial.print(myResults.photon_count);
    Serial.print(" ref photon: ");
    Serial.print(myResults.ref_photon_count);
    Serial.print(" ALS: ");
    Serial.println(myResults.ambient_light);
    Serial.println();
  }


  // Create a String to send (data must be sent as a string, so we'll tell our
computer friend that we want our integers to be set as strings)
  String myData =  String(myResults.results[4].distance_mm)+ "mm " + "Channel :"
+ String(myResults.results[4].channel) ;  // I added some units for our data here
as well

  // Transmit data and units
  int state = radio.transmit(myData);

  if (state == RADIOLIB_ERR_NONE) {
    // The packet was successfully transmitted
    Serial.println(F("Success!"));

    // Print measured data rate
    Serial.print(F("[SX1262] Datarate:\t"));
    Serial.print(radio.getDataRate());
    Serial.println(F(" bps"));

    //wait for receiver ack
     String str;                     // Declare a data type to receive (this
must be a string)
  int ackstate = radio.receive(str);
  while(ackstate != RADIOLIB_ERR_NONE){
    ackstate = radio.receive(str);
```

```
      }
  // Print the data of the packet to the serial monitor
    Serial.print(F("ack from receiver:\t\t"));
    Serial.println(str);

  } else if (state == RADIOLIB_ERR_PACKET_TOO_LONG) {
    // The supplied packet was longer than 256 bytes
    Serial.println(F("Packet size too long!"));

  } else if (state == RADIOLIB_ERR_TX_TIMEOUT) {
    // Timeout occurred while transmitting packet
    Serial.println(F("Timeout!"));

  } else {
    // Some other error occurred
    Serial.print(F("Error"));
    Serial.println(state);
  }

  // Wait for a second before transmitting again
  delay(6000);}
```

**Esp32+LoRa(receiver) firmware code :**

Receiver LoRa module is configured to receive the LiDAR sensor values as a string and send back an acknowledgement to transmitter.

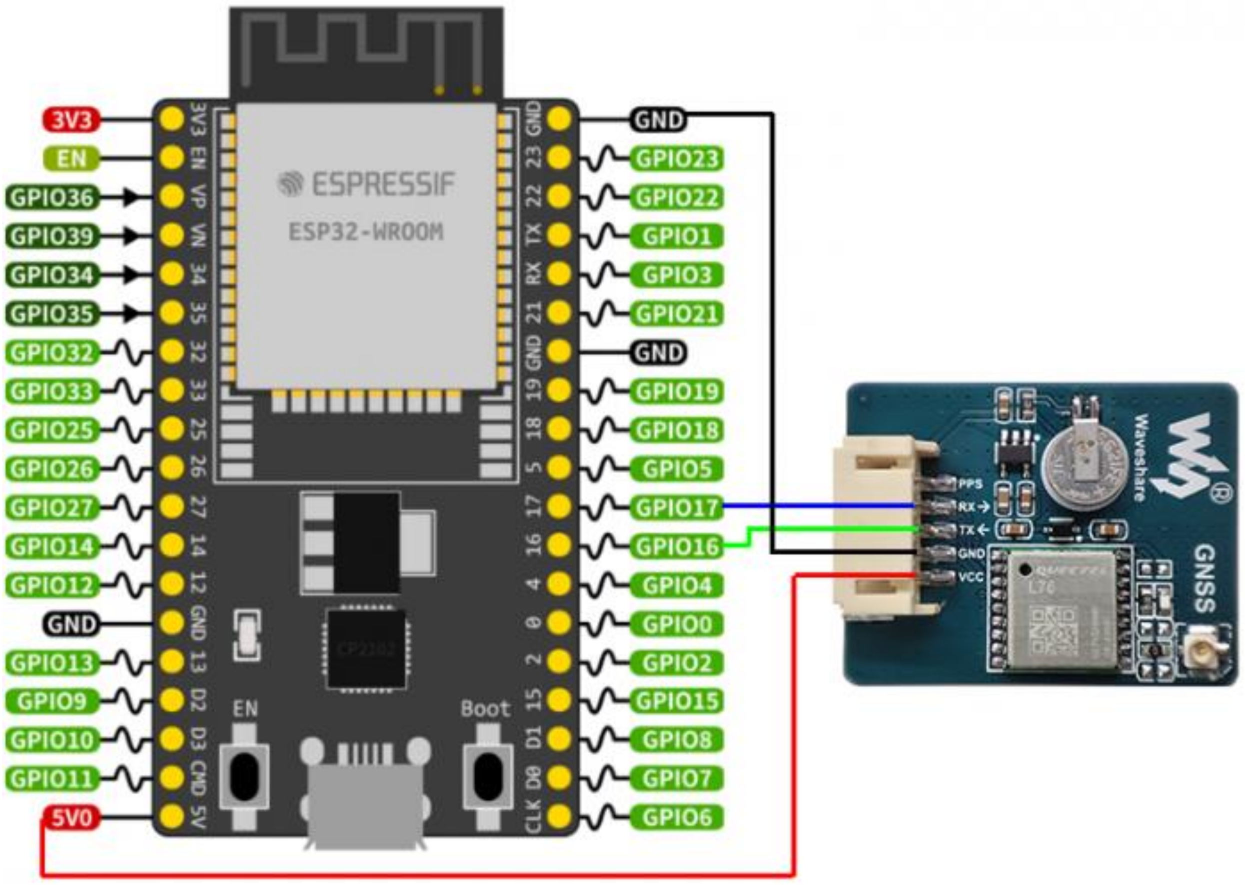
```
// Peer to Peer: Monitor Sensor Data using LoRa - Receiver
// SparkFun Electronics, Mariah Kelly, November 2022
// Original receive file can be found here:
https://cdn.sparkfun.com/assets/learn_tutorials/1/4/9/4/Receive-v3.ino

// Include necessary libraries
#include <RadioLib.h>

// SX1262 has the following connections:
// NSS pin:   5
// DIO1 pin:  34
// NRST pin:  32
// BUSY pin:  33
SX1262 radio = new Module(5, 34, 32, 35);


void setup() {
  Serial.begin(115200);

  // initialize SX1262 with default settings
  Serial.print(F("[SX1262] Initializing ... "));
  //int state = radio.begin();
  int state = radio.begin(868.0, 250.0, 7, 5, 0x34, 20, 10, 0, false);

  if (state == RADIOLIB_ERR_NONE) {
    Serial.println(F("success!"));
  } else {
    Serial.print(F("failed, code "));
    Serial.println(state);
    while (true)
      ;
  }
}

void loop() {

  Serial.print(F("[SX1262] Waiting for incoming transmission ... "));

  String str;                       // Declare a data type to receive (this must
be a string)
  int state = radio.receive(str);   // Receive data from transmitter
while(state != RADIOLIB_ERR_NONE)
{
  state = radio.receive(str);
}

  if (state == RADIOLIB_ERR_NONE) {
    // The packet was successfully received
    Serial.println(F("Success!"));


    // Print the data of the packet to the serial monitor
    Serial.print(F("[SX1262] Data:\t\t"));
    Serial.println(str);

    // Print the RSSI (Received Signal Strength Indicator) of the last received
packet
    Serial.print(F("[SX1262] RSSI:\t\t"));
    Serial.print(radio.getRSSI());
    Serial.println(F(" dBm"));
    // Print the SNR (Signal-to-Noise Ratio) of the last received packet
    Serial.print(F("[SX1262] SNR:\t\t"));
    Serial.print(radio.getSNR());
    Serial.println(F(" dB"));

    Serial.println("sending ack");
    int ackstate =radio.transmit("data received");
    while(ackstate != RADIOLIB_ERR_NONE){
    ackstate = radio.transmit("data received");
    }
    Serial.print("state :");
    Serial.print(ackstate);
  }
  else if (state == RADIOLIB_ERR_RX_TIMEOUT) {
    // Timeout occurred while waiting for a packet
    Serial.println(F("Timeout!"));  // We won't print this to the screen since
the Success/data print will freeze on its own if a timeout occurs

  } else if (state == RADIOLIB_ERR_CRC_MISMATCH) {
    // The packet was received, but is malformed
    Serial.println(F("CRC error!"));

  } else {
    // Some other error occurred
    Serial.print(F("Error"));
    Serial.println(state);
  }
}
```

Note : Serial monitor values are in the test record.

## GPS module in development board:

The LoRa development board has L76K GPS module and Sx1262 LoRa module. GPS module communicates with the controller using UART protocol,

For validating GPS module, connect ESP32 module and GPS modules RX, TX ,5v,gnd connections as shown below



| Development module | ESP32 |
|---|---|
| VCC | 5V |
| GND | GND |
| TX | P16 |
| RX | P17 |
| PPS | NC |

**Source code using Arduino:**

```
/*
 * Rui Santos
 * Complete Project Details http://randomnerdtutorials.com
 */

#include <TinyGPS++.h>
#include <SoftwareSerial.h>

// The serial connection to the GPS module
SoftwareSerial ss(16, 17);

// The TinyGPS++ object
TinyGPSPlus gps;

void setup(){
  Serial.begin(9600);
  ss.begin(9600);
}

void loop(){
    while (ss.available() > 0){
    // get the byte data from the GPS
    byte gpsData = ss.read();
    Serial.write(gpsData);
  }
}
```

Output:

Ln 29, Col 1
ESP32-WROOM-DA Module
on COM9 [not connected]

```
18:10:16.647 -> $GLGSV,3,3,09,68,05,107,16,0*4A
18:10:16.710 ->
$GNRMC,124016.000,A,1255.08917,N,08013.63896,E,0.00,150.53,260724,,,A,V*01
18:10:16.775 -> $GNVTG,150.53,T,,M,0.00,N,0.00,K,A*21
18:10:16.809 -> $GNZDA,124016.000,26,07,2024,00,00*4F
18:10:16.871 -> $GPTXT,01,01,01,ANTENNA OK*35
18:10:16.986 ->
$GNGGA,124017.000,1255.08918,N,08013.63896,E,1,17,0.6,36.0,M,-93.0,M,,*52
18:10:17.082 -> $GNGLL,1255.08918,N,08013.63896,E,124017.000,A,A*45
18:10:17.146 ->
$GNGSA,A,3,02,03,04,10,16,21,26,27,28,31,32,194,1.2,0.6,1.0,1*01
18:10:17.211 -> $GNGSA,A,3,70,73,80,79,85,,,,,,,,1.2,0.6,1.0,2*3E
```

```
18:10:17.243 ->
$GPGSV,4,1,15,02,29,273,42,03,16,328,43,04,06,295,35,08,,,32,0*54
18:10:17.337 ->
$GPGSV,4,2,15,10,33,131,29,16,37,173,29,21,35,255,38,23,,,17,0*58
18:10:17.404 ->
$GPGSV,4,3,15,26,61,120,26,27,08,183,34,28,31,024,45,31,58,352,40,0*62
18:10:17.466 -> $GPGSV,4,4,15,32,21,062,35,194,22,075,30,195,,,21,0*62
18:10:17.530 ->
$GLGSV,3,1,09,74,08,203,25,70,25,351,30,73,51,197,30,80,66,044,33,0*7F
18:10:17.594 ->
$GLGSV,3,2,09,79,13,028,36,69,30,064,23,84,15,211,,85,17,266,32,0*7F
18:10:17.690 -> $GLGSV,3,3,09,68,05,107,22,0*4D
18:10:17.728 ->
$GNRMC,124017.000,A,1255.08918,N,08013.63896,E,0.00,150.53,260724,,,A,V*0F
18:10:17.788 -> $GNVTG,150.53,T,,M,0.00,N,0.00,K,A*21
18:10:17.827 -> $GNZDA,124017.000,26,07,2024,00,00*4E
18:10:17.884 -> $GPTXT,01,01,01,ANTENNA OK*35
18:10:17.998 ->
$GNGGA,124018.000,1255.08917,N,08013.63896,E,1,17,0.6,36.0,M,-93.0,M,,*52
```

OUTPUT interpretation:

## $GNVTG

```
mathematica
Copy code
18:10:28.787 -> $GNVTG,150.53,T,,M,0.00,N,0.00,K,A*21
```

- **150.53,T**: Track made good (in degrees True)
- **,,M**: Track made good (in degrees Magnetic) - not available
- **0.00,N**: Ground speed in knots
- **0.00,K**: Ground speed in kilometers per hour
- **A**: Mode indicator (A = Autonomous)

## $GNZDA

```
bash
Copy code
18:10:28.823 -> $GNZDA,124028.000,26,07,2024,00,00*42
```

- **124028.000**: UTC time (12:40:28.000)
- **26,07,2024**: Day, Month, Year
- **00,00**: Local time zone offset from UTC (hours, minutes)

## $GPTXT

```
bash
Copy code
18:10:28.855 -> $GPTXT,01,01,01,ANTENNA OK*35
```

- **ANTENNA OK**: Status message indicating the antenna is functioning properly

## $GNGGA

```
mathematica
Copy code
18:10:28.979 ->
$GNGGA,124029.000,1255.08920,N,08013.63898,E,1,17,0.6,35.9,M,-93.0,M,,*50
```

- **124029.000**: UTC time (12:40:29.000)
- **1255.08920,N**: Latitude (12° 55.08920' N)
- **08013.63898,E**: Longitude (80° 13.63898' E)
- **1**: Fix quality (1 = GPS fix)
- **17**: Number of satellites being tracked
- **0.6**: Horizontal dilution of precision (HDOP)
- **35.9,M**: Altitude (35.9 meters above mean sea level)
- **-93.0,M**: Height of geoid (mean sea level) above WGS84 ellipsoid

## $GNGLL

```
mathematica
Copy code
18:10:29.075 -> $GNGLL,1255.08920,N,08013.63898,E,124029.000,A,A*4D
```

- **1255.08920,N**: Latitude (12° 55.08920' N)
- **08013.63898,E**: Longitude (80° 13.63898' E)
- **124029.000**: UTC time (12:40:29.000)
- **A**: Data status (A = valid)
- **A**: Mode indicator (A = Autonomous)

## $GNGSA

```
bash
Copy code
18:10:29.139 ->
$GNGSA,A,3,02,03,04,10,16,21,26,27,28,31,32,194,1.2,0.6,1.0,1*01
18:10:29.203 -> $GNGSA,A,3,70,73,80,79,85,,,,,,,,1.2,0.6,1.0,2*3E
```

- **A**: Auto 2D/3D fix
- **3**: 3D fix
- **02,03,04,...**: PRNs of satellites used in the fix
- **1.2**: PDOP (Position Dilution of Precision)
- **0.6**: HDOP (Horizontal Dilution of Precision)
- **1.0**: VDOP (Vertical Dilution of Precision)

## $GPGSV / $GLGSV

```
bash
Copy code
18:10:29.240 ->
$GPGSV,4,1,15,02,29,273,43,03,16,328,42,04,06,295,34,08,,,31,0*56
```

```
18:10:29.331 ->
$GPGSV,4,2,15,10,33,131,30,16,37,173,30,21,35,254,37,23,,,20,0*52
18:10:29.395 ->
$GPGSV,4,3,15,26,61,120,28,27,08,183,34,28,31,024,45,31,58,352,39,0*62
18:10:29.459 -> $GPGSV,4,4,15,32,21,062,34,194,22,075,31,195,,,17,0*67
18:10:29.522 ->
$GLGSV,3,1,09,74,08,203,25,70,25,351,29,73,51,197,28,80,66,043,33,0*79
18:10:29.587 ->
$GLGSV,3,2,09,79,13,028,36,69,30,064,23,84,15,211,,85,17,265,34,0*7A
18:10:29.683 -> $GLGSV,3,3,09,68,05,107,19,0*45
```

- **GPGSV**: GPS Satellites in view
- **GLGSV**: GLONASS Satellites in view
- **4,3,15**: Number of sentences for full data, sentence number, number of satellites in view
- **02,29,273,43**: Satellite PRN, Elevation, Azimuth, SNR (Signal to Noise Ratio)

## $GNRMC

```
mathematica
Copy code
18:10:29.715 ->
$GNRMC,124029.000,A,1255.08920,N,08013.63898,E,0.00,150.53,260724,,,A,V*07
```

- **124029.000**: UTC time (12:40:29.000)
- **A**: Status (A = Active, V = Void)
- **1255.08920,N**: Latitude (12° 55.08920' N)
- **08013.63898,E**: Longitude (80° 13.63898' E)
- **0.00**: Speed over ground (knots)
- **150.53**: Track angle (degrees True)
- **260724**: Date (26 July 2024)
- **A**: Mode indicator (A = Autonomous)
- **V**: Magnetic variation (not available)

## Summary:

- **Latitude and Longitude**: 12° 55.08920' N, 80° 13.63898' E
- **Date and Time**: 26 July 2024, 12:40:29 UTC
- **Altitude**: 35.9 meters above mean sea level
- **Speed**: 0 knots (stationary)
- **Track angle**: 150.53° True
- **Number of satellites tracked**: 17 (GPS), 9 (GLONASS)

## Way forward in LoRa Project:

**1.Test for more than 1 km**
2. To create a LoRaWan (LoRa Hub).
3. To achieve communication from all loRa devices to one Common LoRahub
4. To build Network server using thingsnetwork and Aws LoRawan and monitor data at the server.
5. Cross check server data from another device.
6. File transfer/image transfer using LoRa and OTA.

Supporting teams:
1. Electrical hardware team – LoRa hardware development
2. Software team – Server network in AwsLorawan or Thingsnetwork
3. Firmware team – built sustainable Firmware for the LoRa hardware