RTK-GPS PROJECT REPORT

Transmitting ZED-F9P Real-Time Kinematic RTCM3 Correction Data using LoRa Protocol

By

YONGCHANG HE

TABLE OF CONTENTS

TABLE OF CONTENTS 1

CHAPTER 1 INTRODUCTION 3

CHAPTER 2 PROJECT DESIGN 4

2.1 Architecture 4

2.2 Modules and Tools 5

2.2.1 ZED-F9P Module 5

2.2.2 LoRa Module 5

2.2.3 CT008 Module 6

2.2.4 U-Center 6

CHAPTER 3 IMPLEMENTATION 7

3.1 ZED-F9P Implementations 7

3.1.1 Module Setup 7

3.1.2 Base & Rover Configuration 7

3.1.2.1 Using Configuration File with u-center 7

3.2 Feather M0 LoRa Module Implementation 8

3.2.1 Module Pinout 8

3.2.2 Module Setup 8

3.2.3 Module Configuration with Base 9

3.2.4 Module Configuration with Rover 15

3.3 CT008 Module Implementation 21

CHAPTER 4 EXPERIMENTS 24

4.1 RTCM3 LoRa Transmission - office test 24

4.1.1 Test Overview 24

4.1.2 Objectives 24
4.1.3 Test Result 25
4.2 RTK RTCM3 LoRa Transmission - Field Test 25
4.2.1 Test Overview 25
4.2.2 Objectives 25
4.2.3 Test Process 26
4.2.4 Screenshots 27
4.2.5 Longest LoRa Communication Range test result 29
CHAPTER 5 CONCLUSIONS AND FUTURE WORK 31
5.1 Conclusion 31
5.2 Future Work 31
CHAPTER 6 LIST OF RESEAECH RESOURCES 32

CHAPTER 1

INTRODUCTION

This project, spanning eight months (part-time) from September 1, 2022, to April 30, 2023, involved an extensive period of research, development, and testing. we have developed a reliable framework for Real-Time Kinematic (RTK) positioning by integrating the u-blox C099 ZED-F9P module with the Adafruit Feather M0 LoRa module. The ZED-F9P module is a high-precision GNSS module that provides centimeter-level accuracy using multi-band RTK technology. The Adafruit Feather M0 LoRa module, on the other hand, is a lightweight and low-power microcontroller that uses the Long Range (LoRa) wireless communication protocol, enabling the transmission of RTCM3 correction data over long distances.

Our solution combines the advantages of the ZED-F9P module's RTK capabilities with the Adafruit Feather M0 LoRa module's long-range data transmission, along with CT008 module, resulting in an efficient system for successfully transmitting RCTM3 correction data. In this document, we will provide an overview of the project, discuss the key components, and detail the implementation and testing process.

CHAPTER 2

PROJECT DESIGN

2.1 Architecture

Our object is to design a reliable framework proof-of-concept (PoC) for real-time kinematic (RTK) positioning. The project architecture is shown in the figure below. The framework contains two main components: BASE component and ROVER component. BASE component is designed to receive and process multi-band GNSS signals from various satellite constellations, and to wirelessly transmit RTCM3.x correction data to the ROVER component; ROVER component is designed to wirelessly receive the RTCM3.x correction data and generate its real-time coordination data with centimeter-level accuracy.

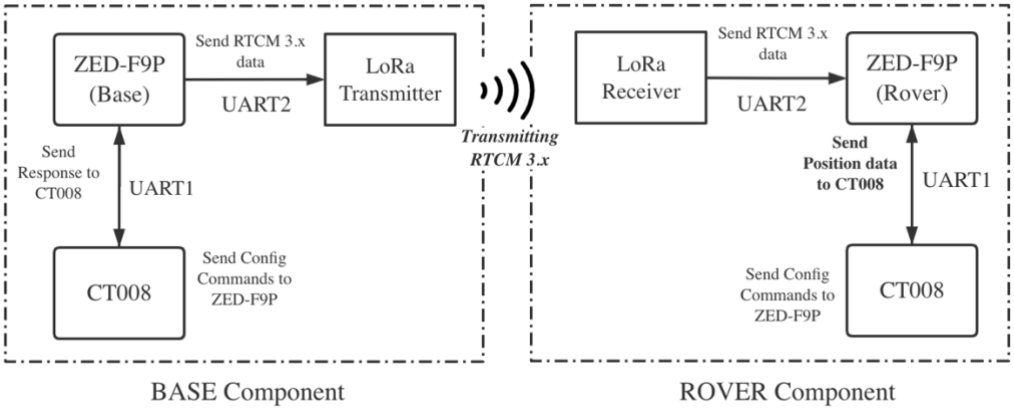

Figure 2.1 – PoC framework architecture

Each components contains three modules: ZED-F9P, CT008, and LoRa. For the BASE component, ZED-F9P (Base) module acts as a base station, receiving GNSS signals through its own antenna and producing RTCM3.x correction data through UART channel; LoRa module receives RTCM3.x correction data through UART channel from ZED-F9P (Base) and transmit it wireless to LoRa receiver; CT008 module sends configuration commands to ZED-F9P (Base) module through UART channel.

For the ROVER component, ZED-F9P (Rover) module acts as a rover station, receiving GNSS signals through its own antenna and producing UBX-NAV-PVT centimeter-level positioning data through UART channel; LoRa module receives RTCM3.x correction data wirelessly from LoRa transmitter, and sends the data to ZED-F9P module; CT008 module sends configuration commands to ZED-F9P (Rover) module through UART channel and can get centimeter-level positioning data from ZED-F9P (Rover) module.

2.2 Modules and Tools

2.2.1 ZED-F9P Module

The C099 ZED-F9P Evaluation Board, as shown below, is a GNSS (Global Navigation Satellite System) evaluation platform developed by u-blox. This evaluation board features the ZED-F9P multi-band GNSS module, which can provide centimeter-level positioning accuracy by employing Real-Time Kinematic (RTK) technology. The board is designed to facilitate the development, testing, and integration of high-precision GNSS applications in various industries.

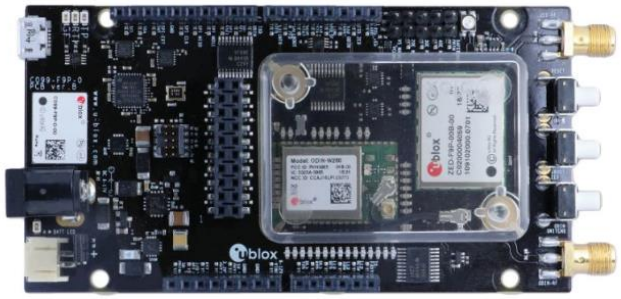

Figure 2.2 - C099 ZED-F9P Evaluation Board with ZED-F9P module

The ZED-F9P module on the evaluation board processes signals from multiple GNSS constellations, such as GPS, GLONASS, Galileo, and BeiDou. By using RTK positioning and receiving RTCM3 correction data from a base station, the ZED-F9P effectively mitigates errors introduced by ionospheric, tropospheric, and other atmospheric disturbances.

The C099 Evaluation Board comes with a comprehensive set of features and interfaces, enabling integration with various host systems and peripherals. Additionally, the board provides access to u-blox's proprietary UBX protocol and standard NMEA messages, offering extensive configuration options and detailed diagnostic data.

Official website: https://www.u-blox.com/en/product/zed-f9p-module

2.2.2 LoRa Module

The Adafruit Feather M0 LoRa module, as shown below, is a compact microcontroller board that combines the capabilities of the ATSAMD21G18 ARM Cortex M0+ processor with the long-range LoRa (Long Range) wireless communication protocol. The module is a choice for various IoT (Internet of Things) applications, remote sensing, telemetry, and other projects that require low-power, long-range wireless data transmission.

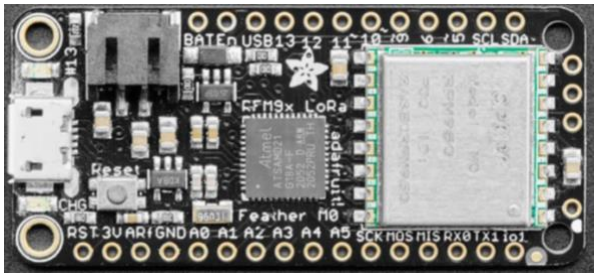

Figure 2.3 – Adafruit Feather M0 LoRa module

The key feature of the Feather M0 LoRa module is its integration with the RFM95W LoRa transceiver, which enables long-range wireless communication over distances of up to kilometers, depending on the environment and settings. Furthermore, the board is compatible with the Arduino development environment and Adafruit's extensive library of software.

Official website: https://www.adafruit.com/product/3178

2.2.3 CT008 Module

The module is designed by Environmental Instruments Canada Inc (EIC). The module is a microcontroller designed specifically to run Espruino, an open-source JavaScript interpreter.

2.2.4 U-Center

u-center, as shown below, is a GNSS evaluation and configuration software developed by u-blox. Designed to work seamlessly with u-blox GNSS receivers, u-center offers a user interface for monitoring, analyzing, and configuring ZED-F9P modules. u-center provides real-time visualization of GNSS data, including position, velocity, time, and satellite information. It also allows to easily configure and control u-blox GNSS receivers, including settings related to power modes, update rates, and supported satellite constellations.

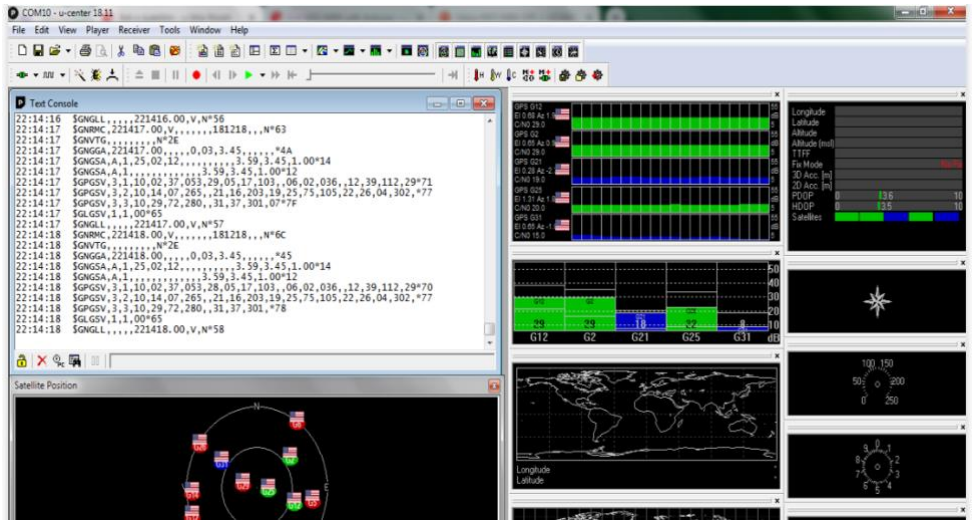

Figure 2.4 -u-center user interface for ZED-F9P module monitoring and configuration (running on Windows PC)

CHAPTER 3

IMPLEMENTATION

3.1 ZED-F9P Implementations

3.1.1 Module Setup

The following steps are to set up the ZED-F9P module.

1. Download the u-center evaluation software and documentation from www.u-blox.com/u-center.
2. Connect the provided ANN-MB GNSS antenna to the ZED-RF connector. Ensure good signal reception.
3. Connect the board to a Windows PC via the USB port to power the board. The board driver will install automatically from Windows Update when connected for the first time.
4. Start u-center and connect to the COM port identified as "ZED-F9P" using the Device Manager. Set the baud rate to 115200 baud.
5. u-center should display the received signals and data. The TP LED will blink in a blue color.

3.1.2 Base & Rover Configuration

3.1.2.1 Using Configuration File with u-center

The following steps are to configure the ZED-F9P as Base station.

1. Download base and rover configuration files from u-blox official GitHub: https://github.com/u-blox/ublox-C099_F9P-uCS/tree/master/zed-f9p.
2. Open u-center > View > Generation F9 configuration view > Advanced configuration.
3. Click 'load' and select the configuration file named *F9P Base config C99.txt*. Click 'Send'.
4. Click 'load' and select the file named *F9P Base Survey in start.txt*. Click 'Send'.
5. We can check that the module started Survey-in with message UBX-NAV-SVIN. The status should be 'In progress', as the figure shown below.
6. The alternate method to steps 4 and 5 is to use configuration view and message UBX-CFG-TIME3, select survey-in mode and input the position accuracy.

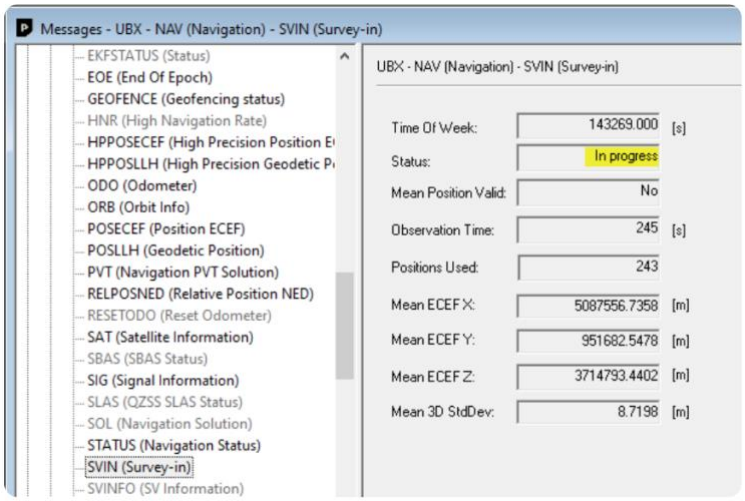

Figure 3.1 – Base station survey-in status: in progress

3.2 Feather M0 LoRa Module Implementation

3.2.1 Module Pinout

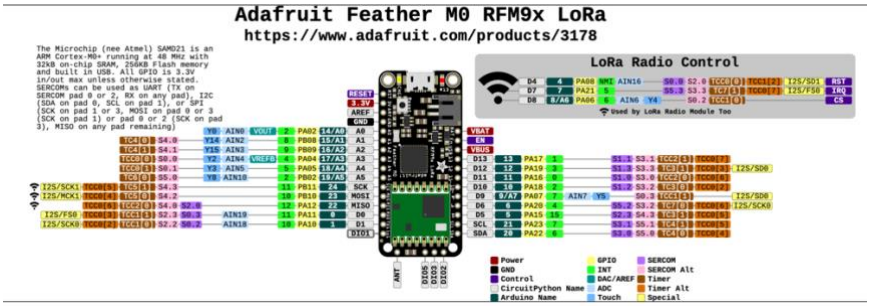

Figure 3.2 – Feather M0 LoRa Module pin out

More description can be found in: https://www.adafruit.com/products/3178

3.2.2 Module Setup

The first thing needed is to install the latest release of the Arduino IDE. The Board Manager URL should be entered at Arduino IDE File > Preferences, as is shown in Figure 3.3.

The URL is : https://adafruit.github.io/arduino-board-index/package_adafruit_index.json.

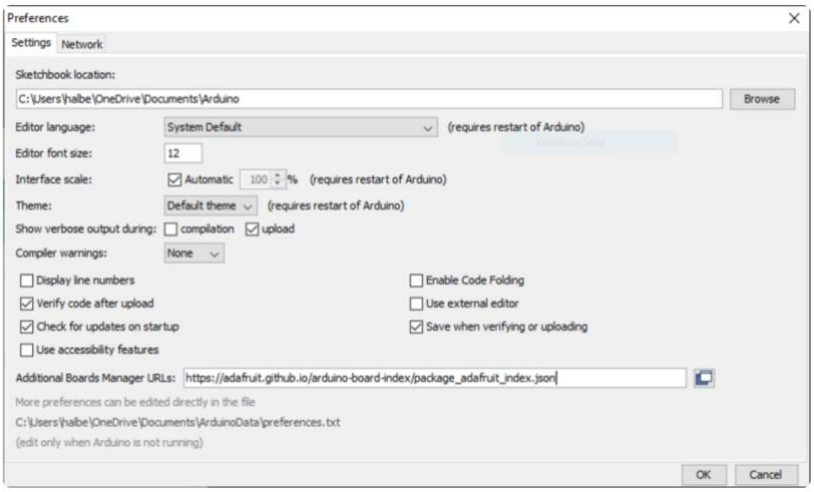

Figure 3.3 – Screenshot for adding the additional Board Manager URL

Next, we open the Boards Manager by navigating to the Tools > Board menu to install the following libraries that makes the board work properly with Arduino:

- Adafruit AVR Boards
- Adafruit SAMD Boards
- Arduino Leonardo & Micro

We need also install the Radio library to enable the module to talk with radio:

- RadioHead.zip (link to download: http://www.airspayce.com/mikem/arduino/RadioHead/)

3.2.3 Module Configuration with Base

The following code can be flashed into LoRa Module configured as transmitter to work with Base ZED-F9P module.

```
-----------------------------------------------------------------------------------------------------------------------------
#include <Arduino.h>       // required before wiring_private.h
#include "wiring_private.h" // pinPeripheral() function
#include <SPI.h>
#include <RH_RF95.h>
#include <Adafruit_SleepyDog.h>

#define RTCM_START 0xd3
#define BUFLEN 2000    //max size of data burst from GPS we can handle
#define SER_TIMEOUT 50 //Timeout in millisecs for reads into buffer from serial
                       // - needs to be longer than byte time at our baud rate
```

```
                // and any other delay between packets from GPS
// for feather m0
#define RFM95_CS 8
#define RFM95_RST 4
#define RFM95_INT 3

// Change to 434.0 or other frequency, must match RX's freq!
#define RF95_FREQ 915.0

// Singleton instance of the radio driver
RH_RF95 rf95(RFM95_CS, RFM95_INT);

// We will use Serial2 - Rx on pin 11, Tx on pin 10
Uart Serial2 (&sercom1, 11, 10, SERCOM_RX_PAD_0, UART_TX_PAD_2);
void SERCOM1_Handler()
{
  Serial2.IrqHandler();
}

// The LED is turned on when 1st byte is received from the serial port. It is
//  turned off after the last byte is transmitted over LoRa.
#define LED 13

int sendSize; //Will hold result of a call to RF95.maxMessageLength(). This
          // value will be the length of packet we send.

static bool SendHeartBeat;  //Set to true if it is time to send a heartbeat packet
unsigned long lastHeartBeatTime; //holds last millis() value when heartbeat was sent

void setup()
{
  pinMode(LED, OUTPUT);
```

```
 pinMode(RFM95_RST, OUTPUT);
 digitalWrite(RFM95_RST, HIGH);

 Serial.begin(115200);
while (!Serial) { //Waits for the Serial Monitor
  delay(1);
}

 Serial2.begin(115200);

 // Assign pins 10 & 11 SERCOM functionality
 pinPeripheral(10, PIO_SERCOM);
 pinPeripheral(11, PIO_SERCOM);
 delay(100);

 Serial.println("Feather LoRa TX");

 // manual reset
 digitalWrite(RFM95_RST, LOW);
 delay(10);
 digitalWrite(RFM95_RST, HIGH);
 delay(10);

 while (!rf95.init())
 {
   Serial.println("LoRa radio init failed");
   while (1);
 }
 Serial.println("LoRa radio init OK!");

 // Defaults after init are 434.0MHz, modulation GFSK_Rb250Fd250, +13dbM
 if (!rf95.setFrequency(RF95_FREQ))
```

```
  {
    Serial.println("setFrequency failed");
    while (1);
  }
  Serial.print("Set Freq to: "); Serial.println(RF95_FREQ);

  // Defaults after init are 434.0MHz, 13dBm, Bw = 125 kHz, Cr = 4/5,
  //  Sf = 128chips/symbol, CRC on

  // The default transmitter power is 13dBm, using PA_BOOST.
  // If you are using RFM95/96/97/98 modules which uses the PA_BOOST
  //  transmitter pin, then you can set transmitter powers from 5 to 23 dBm:
  rf95.setTxPower(23, false);
//  rf95.setSignalBandwidth(62500L);
//  rf95.setCodingRate4(6);

  Serial2.setTimeout(SER_TIMEOUT);
  sendSize = rf95.maxMessageLength();
  lastHeartBeatTime = millis(); //initialize time for activity check
  int countdownMS = Watchdog.enable(3000); //Set watchdog timer to 3.0 seconds.
}

void loop()
{
  uint8_t buf[BUFLEN];
  uint8_t* bufptr;
  int bytesRead;
  int bytesLeft;
  Watchdog.reset();


#ifdef USE_HEARTBEAT
```

```
  //Timer to determine if time to send a heartbeat
  if(millis()-lastHeartBeatTime > 1500)
  {
    SendHeartBeat = true;
    lastHeartBeatTime = millis(); //reset time
  }
#endif

#ifndef USE_HEARTBEAT
  SendHeartBeat = false;
#endif

  if (Serial2.available() || (SendHeartBeat==true))
  {
    digitalWrite(LED, HIGH);

    // If timeout is set properly, this should read an entire burst from the GPS.
    bytesRead = Serial2.readBytes((char *) buf, BUFLEN);

    Serial.println(bytesRead);  //For debugging purposes

    // Process the entire receive buffer (buf) in individual sendSize byte
    //  packets. bufptr points to the start of the bytes to be transmitted.
    //  It is moved through buf until all bytes are transmitted.
    bufptr = buf;

#ifdef USE_HEARTBEAT
    // If nothing received, we here only to send a heartbeat
    if(bytesRead == 0)
    {
      buf[0] = 'H'; //Put our heartbeat character in the buffer
      bytesRead = 1; //Indicate there is a character in the buffer
```

```
    }
#endif

    bytesLeft = bytesRead;
    while(bytesLeft > 0)
    {
      if( bytesLeft < sendSize)
      {
        rf95.waitPacketSent();
        rf95.send(bufptr, bytesLeft);
        bytesLeft = 0;
      }
      else
      {
        rf95.waitPacketSent();
        rf95.send(bufptr, sendSize);
        bufptr += sendSize;
        bytesLeft -= sendSize;
      }
    }

#ifdef USE_HEARTBEAT
    //Reset the heartbeat flag and timer
    SendHeartBeat = false;
    lastHeartBeatTime = millis(); //reset time
#endif

    digitalWrite(LED, LOW);
  }
}
```

3.2.4 Module Configuration with Rover

The following code can be flashed into LoRa Module to work with Rover ZED-F9P module.

```
#include <Arduino.h>  // required before wiring_private.h
#include "wiring_private.h" // pinPeripheral() function
#include <SPI.h>
#include <RH_RF95.h>

#define ACTIVITY_PIN 12 //held HIGH if data being received, otherwise LOW
                        // This could just be a heartbeat from the
                        // transmitter, not necessarily data
#define DATA_OK_PIN 6 //held HIGH if large data packets are being received

// Max size of data burst we can handle (5 full RF buffers)-arbitrarily large
#define BUFLEN (5*RH_RF95_MAX_MESSAGE_LEN)

// Maximum milliseconds to wait for next LoRa packet
//  Up to 300, program sends each packet to GPS. 350 causes it to get multiple
//  packets. At 350, sometimes it is getting two bursts together, i.e. about
//  1200 bytes, when a burst is 600 bytes.
#define RFWAITTIME 1 //1 will cause packet to be sent to GPS immediately

// For feather m0
#define RFM95_CS 8
#define RFM95_RST 4
#define RFM95_INT 3

// Change to 434.0 or other frequency, must match RX's freq!
#define RF95_FREQ 915.0

// Singleton instance of the radio driver
```

```
RH_RF95 rf95(RFM95_CS, RFM95_INT);

// We will use Serial2 - Rx on pin 11, Tx on pin 10
Uart Serial2 (&sercom1, 11, 10, SERCOM_RX_PAD_0, UART_TX_PAD_2);
void SERCOM1_Handler()
{
  Serial2.IrqHandler();
}

// LED is turned on at 1st LoRa reception and off when nothing else received.
//  It gives an indication of how long the incoming data stream is.
#define LED 13

unsigned long lastActCheckTime; //holds last millis() value when activity was checked

void setup()
{
  pinMode(LED, OUTPUT);
  pinMode(RFM95_RST, OUTPUT);
  pinMode(ACTIVITY_PIN, OUTPUT);
  pinMode(DATA_OK_PIN, OUTPUT);

  digitalWrite(RFM95_RST, HIGH);
  digitalWrite(ACTIVITY_PIN, LOW); //Initialize to no activity
  digitalWrite(DATA_OK_PIN, LOW);  //Initialize to no packet activity

  Serial.begin(115200);
  while (!Serial) { //Waits for the Serial Monitor
    delay(1);
  }

  Serial2.begin(115200);
```

```
// Assign pins 10 & 11 SERCOM functionality
pinPeripheral(10, PIO_SERCOM);
pinPeripheral(11, PIO_SERCOM);
delay(100);

Serial.println("Feather LoRa RX");

// manual reset
digitalWrite(RFM95_RST, LOW);
delay(10);
digitalWrite(RFM95_RST, HIGH);
delay(10);

while (!rf95.init()) {
  Serial.println("LoRa radio init failed");
  while (1);
}
Serial.println("LoRa radio init OK!");

for(int ii=0; ii<100; ii++) { Serial.println("TEST"); }

// Defaults after init are 434.0MHz, modulation GFSK_Rb250Fd250, +13dbM
if (!rf95.setFrequency(RF95_FREQ)) {
  Serial.println("setFrequency failed");
  while (1);
}
Serial.print("Set Freq to: "); Serial.println(RF95_FREQ);

// Defaults after init are 434.0MHz, 13dBm, Bw = 125 kHz, Cr = 4/5,
//  Sf = 128chips/symbol, CRC on
```

```
  // The default transmitter power is 13dBm, using PA_BOOST.
  //  If you are using RFM95/96/97/98 modules which uses the PA_BOOST
  //  transmitter pin, then you can set transmitter powers from 5 to 23 dBm:
  rf95.setTxPower(23, false);
  //rf95.setSignalBandwidth(62500L);

  lastActCheckTime = millis(); //initialize time for activity check
}

void loop()
{
  uint8_t buf[BUFLEN];
  unsigned buflen;

  uint8_t rfbuflen;
  uint8_t *bufptr;
  unsigned long lastTime, curTime;
  static bool RX_Activity;  //Set to true if we are receiving anything
  static bool Data_OK;  //Set to true if we are receiving large packets

  bufptr = buf;
  buflen = 0;

  if (rf95.available())
  {
    RX_Activity = true; //Indicate that we have activity
    digitalWrite(LED, HIGH);
    rfbuflen = RH_RF95_MAX_MESSAGE_LEN;
    if(rf95.recv(bufptr, &rfbuflen))
    {
      bufptr += rfbuflen;
      lastTime = millis();
```

```
    while(((millis()-lastTime) < RFWAITTIME) && ((bufptr - buf) < (BUFLEN -
RH_RF95_MAX_MESSAGE_LEN))) //Time out or buffer can't hold anymore
    {
     if (rf95.available())
     {
      rfbuflen = RH_RF95_MAX_MESSAGE_LEN;
      if(rf95.recv(bufptr, &rfbuflen))
      {
       bufptr += rfbuflen;
       lastTime = millis();
      }
      else
      {
       Serial.println("Receive failed");
      }
     }
    }
  }
  else
  {
   Serial.println("Receive failed");
  }

  buflen = (bufptr - buf);  // Total bytes received in all packets

  if (buflen > 100) Data_OK = true; //Indicate we are likely getting real data

  Serial.print(rf95.lastRssi(), DEC);
  Serial.print(" ");
  Serial.println(buflen);   // For debugging
  Serial2.write(buf, buflen); //Send data to the GPS
```

```
  digitalWrite(LED, LOW);
 }

 //Activity indicator output
 if(millis()-lastActCheckTime > 1500)
 {
  if(RX_Activity)
  {
   digitalWrite(ACTIVITY_PIN, HIGH);  //Indicate that we received data
  }
  else
  {
   digitalWrite(ACTIVITY_PIN, LOW); //Indicate no data received
  }
  RX_Activity = false;  //set flag to false - receive activity (if any) will set it to true

  if(Data_OK)
  {

   digitalWrite(DATA_OK_PIN, HIGH);  //Indicate that we are receiving large packets
  }
  else
  {
   digitalWrite(DATA_OK_PIN, LOW); //Indicate not receiving large packets
  }
  Data_OK = false;  //set flag to false - will be set elsewhere if receiving large packets

  lastActCheckTime = millis(); //reset time
 }
}
```

3.3 CT008 Module Implementation

The code below can be run on CT008 Espruino IDE to configure ZED-F9P module.

```
// Initialize communication with Base/Rover via CT008 Board
// Baud rate is set to 115200, RX and TX is set to pins 23 and 19
Serial1.setup(115200,{rx:D23, tx:D19});

// Call this function to start sending RTCM3 data through URAT channel
function set_uart1_usb_cfg_msgout_rtcm3_on() {
  data = ['\xb5b\x06\x8a@\x00\x00\x05\x00\x00\xbe\x02\x91 \x01_\x03\x91 \x01d\x03\x91
\x01i\x03\x91 \x01n\x03\x91 \x01\x04\x03\x91 \x05\xc0\x02\x91 \x01a\x03\x91 \x01f\x03\x91
\x01k\x03\x91 \x01p\x03\x91 \x01\x06\x03\x91 \x05\x1bI'];
  Serial1.write(data);
  setTimeout(set_ubx_nav_pvt_svin_usb_on,1000);
}

// Call this function to stop sending RTCM3 data through URAT channel
function set_uart1_usb_cfg_msgout_rtcm3_off() {
  data = ['\xb5b\x06\x8a@\x00\x00\x05\x00\x00\xbe\x02\x91 \x00_\x03\x91 \x00d\x03\x91
\x00i\x03\x91 \x00n\x03\x91 \x00\x04\x03\x91 \x00\xc0\x02\x91 \x00a\x03\x91 \x00f\x03\x91
\x00k\x03\x91 \x00p\x03\x91 \x00\x06\x03\x91 \x00\x07s'];
  Serial1.write(data);
}

// Call this function to send the message UBX-NAV-PVT through USB channel.
// This message contains real-time positioning information.
function set_ubx_nav_pvt_svin_usb_on() {
  data = ['\xb5b\x06\x8a\x0e\x00\x00\x05\x00\x00\t\x00\x91 \x01\x8b\x00\x91 \x01\x9b*'];
  Serial1.write(data);
  setTimeout(set_usboutprot_rtcm3_on,1000);
}
```

```
// Stop sending message UBX-NAV-PVT through USB channel
function set_ubx_nav_pvt_svin_usb_off() {
  data = ['\xb5b\x06\x8a\x0e\x00\x00\x05\x00\x00\t\x00\x91 \x00\x8b\x00\x91 \x00\x99#'];
  Serial1.write(data);
}

// Call this function to start sending RTCM3 data through USB channel
function set_usboutprot_rtcm3_on() {
  data = ['\xb5b\x06\x8a\t\x00\x00\x05\x00\x00\x04\x00x\x10\x01+\xee'];
  Serial1.write(data);
  setTimeout(set_base_rtcm3_out_off,1000);
}

// Call this function to stop sending RTCM3 data through USB channel
function set_usboutprot_rtcm3_off() {
  data = ['\xb5b\x06\x8a\t\x00\x00\x05\x00\x00\x04\x00x\x10\x00*\xed'];
  Serial1.write(data);
}

function set_base_rtcm3_out_on() {
  data =
['\xb5b\x06\x8a"\x00\x00\x05\x00\x00\x01\x00s\x10\x01\x02\x00s\x10\x00\x04\x00s\x10\x00\x01\x0
0t\x10\x01\x02\x00t\x10\x00\x04\x00t\x10\x01\xdd\x8f'];
  Serial1.write(data);
}

function set_base_rtcm3_out_off() {
  data =
['\xb5b\x06\x8a"\x00\x00\x05\x00\x00\x01\x00s\x10\x01\x02\x00s\x10\x00\x04\x00s\x10\x00\x01\x0
0t\x10\x01\x02\x00t\x10\x00\x04\x00t\x10\x00\xdc\x8e'];
  Serial1.write(data);
```

```
}

function set_rover_uart1_inprot_outprot_101_110() {
  data =
['\xb5b\x06\x8a"\x00\x00\x05\x00\x00\x01\x00s\x10\x01\x02\x00s\x10\x00\x04\x00s\x10\x01\x01\x0
0t\x10\x01\x02\x00t\x10\x01\x04\x00t\x10\x00\xde\xa4'];
  Serial1.write(data);
}

// First call this function to configure the module as Base
function config_as_base() {
  set_uart1_usb_cfg_msgout_rtcm3_on();
}

// Call this function to start sending RTCM3 message Through UART channel
function base_start() {
  set_base_rtcm3_out_on();
}

// Call this function to stop sending RTCM3 message Through UART channel
function base_pause() {
  set_base_rtcm3_out_off();
}

// Callback function
// Will print data if getting data from UART channel
Serial1.on('data', function (data){
  console.log("got:", JSON.stringify(data));
  }
);
```

CHAPTER 4

EXPERIMENTS

4.1 RTCM3 LoRa Transmission - office test

4.1.1 Test Overview

In this experiment, both BASE component and ROVER component are placed in an office room or an outdoor area in which the distance between the two components is within 10 meters. The office room should not have metal roof so that the ZED-F9P module can get excellent signals from satellites. This experiment is to ensure all the modules in the Base component and Rover component are configured properly and do their job as expected. This experiment is the basis for field test described in section 4.2. Experiment 4.2 can be conducted only when experiment 4.1 fulfills all the functionalities.

4.1.2 Objectives

The objective of this test is to verify the correctness of our project's functionalities, as described below:

1. ZED-F9P modules can be powered up and recognized by PC properly;
2. Data input and output Via ZED-F9P can be monitored through software: u-center (Windows version);
3. Signals from multiple GNSS constellations, such as GPS, GLONASS, Galileo, and BeiDou can be captured by the ZED-F9P Base module;
4. ZED-F9P module can be configured as Base and Rover by CT008 boards through UART;
5. The RTCM3 correction data can be sent out accurately through UART channel according to configuration;
6. RTCM3 correction data can be received by LoRa transmitter module, and sent out through LoRa radio;
7. RTCM3 correction data can be received by LoRa receiver module, and relayed to UART channel of ZED-F9P Rover module;
8. ZED-F9P Rover module can successfully recognize all RTCM3 correction data packets and process positioning calculation;
9. ZED-F9P Rover module can output its accurate position data UBX-NAV-PVT to u-center.

4.1.3 Test Result

We conducted test 4.1 on April 13, 2023. Modules are powered up by the Windows PC. Data sending and receiving and monitored by u-center. The Test result reveals that all the functionalities listed in section 4.1.2 can be fulfilled by our project framework correctly. In the office or small open-air area (within 10 meters), the RTK RTCM3 correction data packets can be sent by BASE component and be received by the ROVER component without any error. The picture for the test environment is shown below.

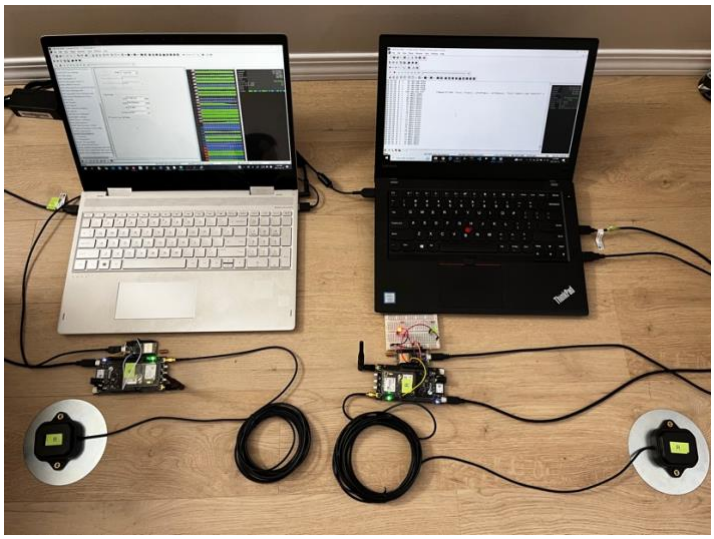

Figure 4.1 - Office test

4.2 RTK RTCM3 LoRa Transmission - Field Test

4.2.1 Test Overview

This experiment is to explore the longest working distance of LoRa radio to transmit RTCM3 correction data packets from BASE component to ROVER component. In this experiment, both BASE component and ROVER component are placed in an outdoor area where there are no obvious obstacles in between, so that the ZED-F9P modules can get excellent signals from satellites, and the LoRa radio can work in the best condition.

4.2.2 Objectives

The objective of this test is to explore the longest working distance of our project according to the baseline functionalities described in the section 4.1.2.

4.2.3 Test Process

The test was conducted at 8 street east near the intersection with Mcormond Road on April 13, 2023, at around 15:30PM. The road is straight and has very little traffic. The sky was clear. ZED-F9P modules was pre-configured by the CT008 module as base station and rover station. ZED-F9P was also configured to start to send out RTK RTCM3 correction data through URAT channel and USB port. LoRa modules were setup as data transmitter and receiver.

BASE component was placed on the road as a fixed position. ROVER component was placed on the car. Antenna for receiving satellites' data was placed on the car roof. Data was monitored with u-center. Several check points were pre-set to check out the stability of the LoRa Data connection. The car was gradually moving away from BASE component, until the LoRa data connection lost.

Figure 4.2 - Field Test

4.2.4 Screenshots

Figure 4.3 is a screenshot of Arduino IDE. It reveals that Rover station is receiving RTK RTCM3 data packets successfully. ’15:39:37.441 -> -99 251’ shows at this moment the LoRa module received 251 bytes data packet with Received Signal Strength Indicator (RSSI) -99.

Figure 4.4 shows 5 types of RTK RTCM3 correction data were being received by the Rover without issue.

- 1005: Stationary Antenna Reference Point (ARP) coordinates – This message provides the precise coordinates of the reference station's antenna.
- 1074: GPS MSM4 (Multiple Signal Message) – This message includes high-precision code and phase pseudorange, Doppler, and signal-to-noise ratio (SNR) data for GPS signals.
- 1084: GLONASS MSM4 – Similar to the GPS MSM4 message, but for GLONASS signals.
- 1094: Galileo MSM4 – Similar to the GPS MSM4 message, but for Galileo signals.
- 1124: BeiDou MSM4 – Similar to the GPS MSM4 message, but for BeiDou signals.
- 1230: GLONASS Code-Phase Biases – This message provides GLONASS code-phase bias information, which can be used to improve the accuracy of GLONASS positioning.

Figures 4.4 and 4.5 show Rover station was in the Fixed mode. In this mode Rover station had locked onto enough satellite signals and had computed a highly accurate and stable position. The 3D accuracy is 0.02m (2cm), and 2D accuracy is 0.01m (1cm), as shown in Figure 4.4 and 4.5.

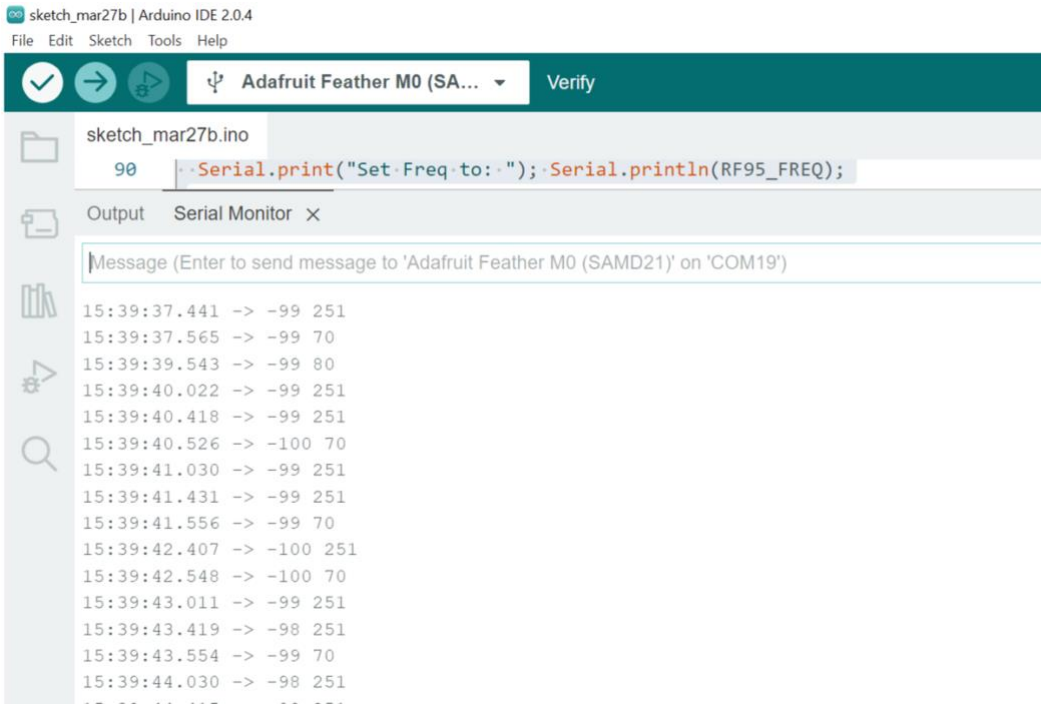

Figure 4.3 RTK RTCM3 data packets receiving status

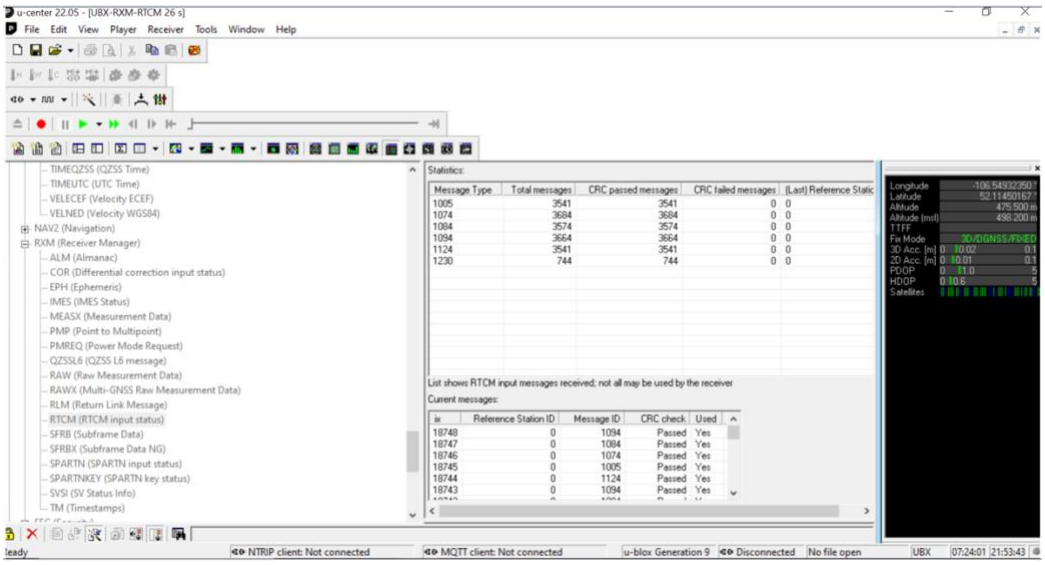

Figure 4.4 RTK RTCM3 data packets is being received by the rover station without error

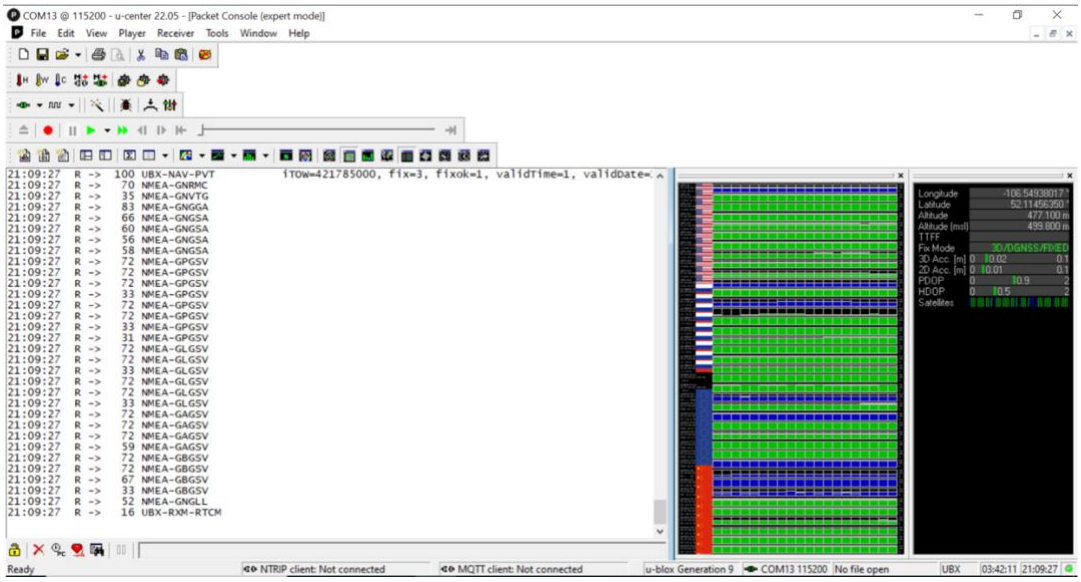

Figure 4.5 RTK RTCM3 data packets is being received by the rover station without error(2)

4.2.5 Longest LoRa Communication Range test result

The longest LoRa communication rang is 1.57km for this time, as shown in Figure 4.6 measured distance in google maps. However, this result was affected by the uneven terrain. The result could be better if placing both BASE and ROVER component in a higher position, and on a more even terrain.

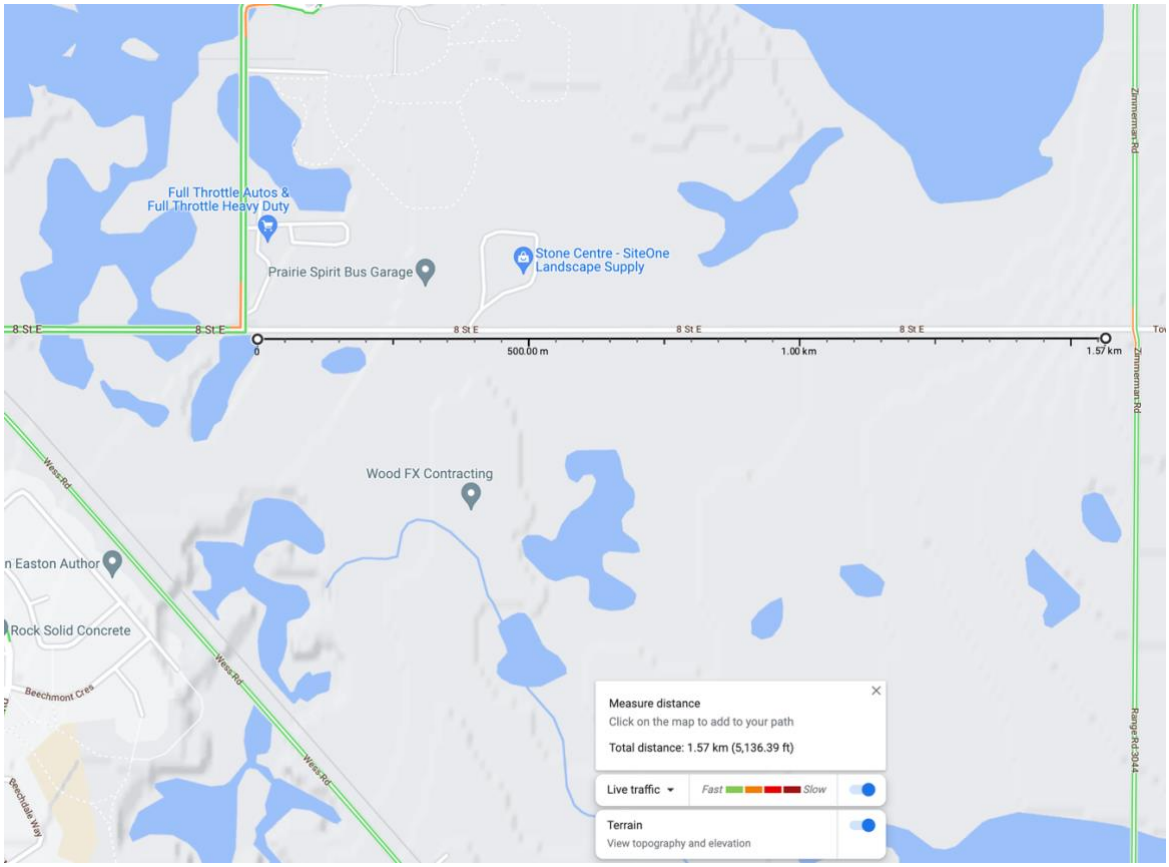

Figure 4.6 - Longest LoRa communication range test

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusion

This project has demonstrated the successful integration of the u-blox C099 ZED-F9P module with the Adafruit Feather M0 LoRa module and CT008 module, yielding a reliable framework for Real-Time Kinematic (RTK) positioning. Over the course of eight months, we dedicated significant effort to research, development, and testing to ensure the optimal performance of the combined solution. By harnessing the high-precision GNSS capabilities of the ZED-F9P module and the long-range data transmission offered by the Adafruit Feather M0 LoRa module, we have created a PoC capable of accurately transmitting RTCM3 correction data through LoRa protocol. This comprehensive document serves as a guide to understanding the project's objectives, key components, implementation, and testing procedures.

5.2 Future Work

1. The CT008 module has already developed an updated version containing two serial channels, empowering it capable of communicating with two serial devices. The updated CT008 module can future act as a configuration center and data-relay center, communicating with ZED-F9P and a LoRa chip (e.g., SX1276) (rather than an Arduino-based microcontroller). Research can be conducted future on exploring a novel way to re-design such a Base-Rover communication framework.
2. ZED-F9P Rover module should send back its live positioning data UBX-NAV-PVT to ZED-F9P Base module, letting the Base station know the accurate location of Rover station. Research related with this direction can be conducted. The difficult part of this research is how to enable this functionality without affect the efficiency of transmitting RTCM3 correction data.

CHAPTER 6

LIST OF RESEAECH RESOURCES

This chapter lists related links and documents of research materials.

ZED-F9P module datasheet:
chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://content.u-blox.com/sites/default/files/ZED-F9P-04B_DataSheet_UBX-21044850.pdf

C099-F9P application board user guide:
chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://content.u-blox.com/sites/default/files/C099-F9P-AppBoard-ODIN-W2-uCX_UserGuide_%28UBX-18055649%29.pdf

ZED-F9P module interface description:
chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://cdn.sparkfun.com/assets/f/7/4/3/5/PM-15136.pdf

ZED-F9P module integration manual:
chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://content.u-blox.com/sites/default/files/ZED-F9P_IntegrationManual_UBX-18010802.pdf

ZED-F9P module base & rover configuration file:
https://github.com/u-blox/ublox-C099_F9P-uCS/tree/master/zed-f9p

u-center user guide:
chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://content.u-blox.com/sites/default/files/u-center_Userguide_UBX-13005250.pdf

Feather M0 LoRa module protocol:
chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://cdn-learn.adafruit.com/downloads/pdf/adafruit-feather-m0-basic-proto.pdf

LoRa radio library resource:

http://www.airspayce.com/mikem/arduino/RadioHead/

Espruino IDE and Serial configuration:

https://www.espruino.com/USART