Progetto di Ingegneria Informatica
GPSDataLogger and Parser (Presentazione)

Autore: Luca Padalino
Matricola: 935033
Codice Persona: 10695959

a.a. 2021/2022

Tutor: prof. Mirko Reguzzoni

Indice

1 Specifiche

1.1 Obiettivo

Lo scopo del progetto è *scrivere uno strumento in grado di registrare (in tempo reale) un flusso di dati binario fornito da un ricevitore GPS, nonché analizzarlo convertendolo in formati di dati standard. Lo strumento deve gestire uno o più ricevitori u-blox EVK-M8T (disponibili per il test), ciascuno dei quali è collegato a una porta COM. Dopo aver registrato ciascuno stream, lo strumento deve decriptarlo per produrre due file ASCII nei formati RINEX e NMEA contenenti le informazioni di posizionamento. È inoltre necessario scrivere un file di sincronizzazione che colleghi i time-stamp del GPS e del computer.*

È possibile consultare e scaricare il codice sorgente dell'intero software sviluppato visitando la **repository GitHub** dedicata. Le istruzioni per l'installazione e l'utilizzo sono presenti nel file *readme* presente all'interno della repository.

Il software sarà installato e quindi utilizzato in un computer situato a bordo di un autogiro dell'Università degli Studi di Ferrara. L'autogiro è impiegato in attività di rilievo geodetico e fotogrammetrico a supporto di misure di radioattività ambientale.

1.2 Specifica

Il software consente di registrare stream di byte provenienti da uno o più ricevitori, contemporaneamente e in tempo reale.

Dagli stream acquisiti vengono campionate le **stringhe (sentences) NMEA**[1] e i **messaggi UBX-RXM-RAWX**[2] da cui si estraggono a loro volta i **time-stamp** necessari alla sincronizzazione dei tempi tra il GNSS e computer locale.

Al termine delle operazioni, è possibile generare i **files RINEX** (Receiver INdependent EXchange format) relativi alle osservazioni e alle navigazioni a partire dagli stream `ubx` salvati in precedenza, mediante l'utilizzo della libreria *CONVBIN* del toolkit RTKLib.

[1] NMEA è uno standard di comunicazione utilizzato nella trasmissione dati satellitare (GPS) e nautica. Le *sentences* che vengono inviate al ricevitore sono in formato ASCII stampabile e possono includere informazioni quali posizione, velocità, profondità, assegnazione di frequenza, ecc...

[2] UBX-RXM-RAWX è una tipologia di messaggi del protocollo `ubx`, proprio dei ricevitori, contenente informazioni utili alla generazione del file RINEX relativo alle osservazioni e dettagli temporali per sincronizzare il tempo del sistema satellitare globale di navigazione (GNSS) con quello del PC.

1.3 Analisi della Specifica

Ciascuna delle caratteristiche previste dalla specifica è realizzata tramite opportuni metodi e funzioni inserite all'interno di apposite classi di codice, descritte nella sezione successiva. La specifica è stata completata con funzionalità e operazioni a supporto delle istanze del committente.

In definitiva, le principali funzionalità offerte dal software sono:

1. **scelta della directory** in cui salvare le acquisizioni e i files generati;
2. **registrazione dello stream** di (uno o più) ricevitori connessi via porta COM **oppure caricamento di file(s)** `.ubx` pre-registrati con software di terze parti (es. u-center);
3. **avvio dell'acquisizione** in parallelo e in tempo reale degli stream;
4. **interruzione dell'acquisizione** degli stream;
5. **generazione dei files RINEX** (`.obs` e `.nav`);

2 Implementazione

2.1 Dettagli Implementativi

Il software **"GPSDataLogger and Parser"** è stato realizzato in *Python 3.9*. La scelta di questo linguaggio scaturisce dalla volontà di realizzare un applicativo cross-platform, versatile e scalabile.

Python si è rivelato il linguaggio più adatto per via della sua facile comprensione ed implementazione. Inoltre, può contare su un supporto molto esteso a librerie proprie e di terze parti (*NumPy* per l'elaborazione numerica; *pySerial* per l'integrazione con le porte COM; *os* per l'integrazione con il sistema operativo in termini di esecuzione di applicativi o costruzione di percorsi per salvare/leggere files; *threading* per la parallelizzazione delle operazioni mediante thread) ed integra strutture dati come dizionari, liste e array, più semplici da gestire rispetto ai puntatori del C; supporta la programmazione orientata agli oggetti e la programmazione concorrente, entrambe molto vantaggiose nell'ottica del riuso e della parallelizzazione delle operazioni.

Tutte le funzionalità sono accessibili grazie ad una **GUI** sviluppata con il framework *PyQt5*, che viene lanciata all'avvio del programma e consente l'esecuzione di tutte le operazioni in modo semplice e veloce. La GUI è stata progettata in modo da guidare l'utente, bloccando elementi grafici, se le operazioni da eseguire non sono state completate nel corretto ordine. La scelta

di sviluppare una GUI deriva inoltre dal possibile utilizzo del programma in spazi ristretti in cui non è possibile utilizzare mouse e tastiera ma soltanto un display touch-screen.

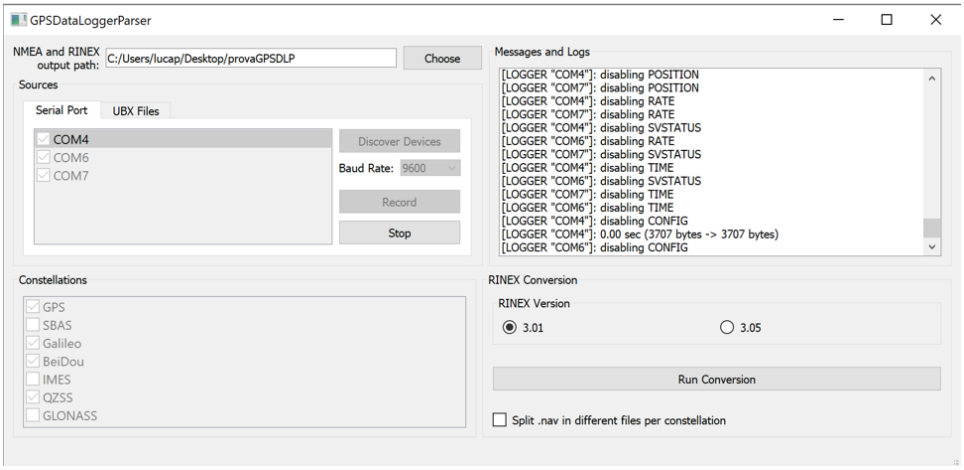
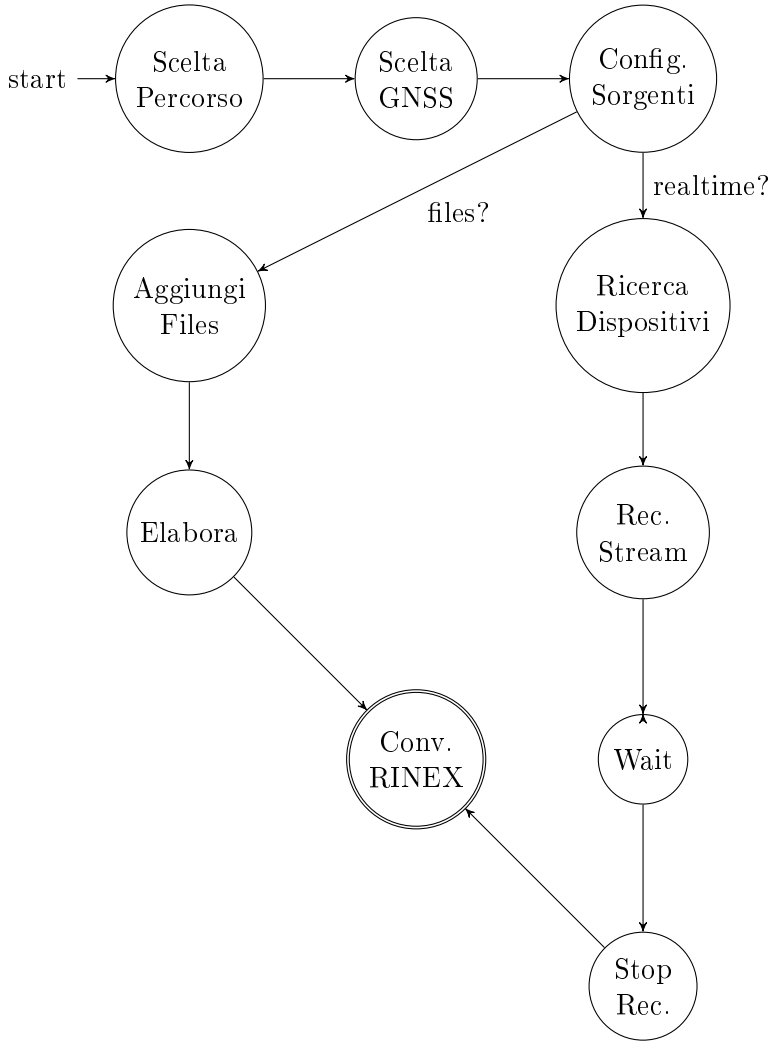


Figura 1: GUI di *GPSData Logger and Parser*

La GUI prevede una finestra di log contenente tutte le informazioni sulle operazioni, sullo stato delle periferiche ed eventuali errori.

2.2 Funzionamento

Segue la descrizione della sequenza delle operazioni che il software svolge mediante l'utilizzo di una macchina a stati finiti (FSM).



2.2.1 Scelta del Percorso

La **scelta del percorso** è la prima operazione che il software richiede all'utente. Consiste nella scelta di una directory in cui vengono salvati tutti i files che il programma genera, ossia le acquisizioni degli stream, i file contenenti le stringhe NMEA, i files relativi alle sincronizzazioni dei tempi e i files RINEX.

2.2.2 Scelta dei GNSS

Successivamente, viene richiesto di **scegliere i GNSS**[3] da cui ricevere i dati, escludendo le costellazioni non di interesse. Nel momento in cui verrà avviata l'acquisizione, il programma invia ai ricevitori un comando per esprimere da quali GNSS voler ricevere i dati (precisamente un messaggio di tipo UBX-CFG-GNSS).

2.2.3 Configurazione delle Sorgenti

A questo punto è necessario **configurare le sorgenti** da cui acquisire i dati da elaborare. Come riportato nella sezione precedente, è possibile registrare i flussi in tempo reale e contemporaneamente da più ricevitori connessi alla porta seriale COM e/o caricare files `ubx` di acquisizioni già svolte.

2.2.4 Ricerca dei Dispositivi u-blox

Nel caso si desidera acquisire i dati in tempo reale, occorre prima **ricercare i dispositivi**. Questa operazione avviene con la finalità di mostrare all'utente (in una checklist) i dispositivi connessi alla porta seriale che effettivamente sono ricevitori u-blox, escludendo periferiche che non lo sono. Alla pressione del pulsante `Discover Devices` viene inviato un messaggio a tutti i dispositivi e, se ciascun dispositivo risponde con un ACK, questo può essere considerato un ricevitore e viene aggiunto in lista, altrimenti no. Dopo aver selezionato i ricevitori a cui connettersi per acquisire, è possibile procedere con la **registrazione dello stream**.

2.2.5 Avvio della Registrazione dello Stream

La **registrazione dello stream** è l'operazione che consente di acquisire le informazioni provenienti dal ricevitore ed avviene in parallelo ad altre operazioni: quando l'utente clicca sul pulsante `Record`, il programma crea un thread per ciascun ricevitore, parallelizzando l'operazione di campionamento. I thread termineranno la propria esecuzione quando l'utente interromperà la raccolta dei dati cliccando sul pulsante `Stop`.

[3] Global Navigation Satellite System

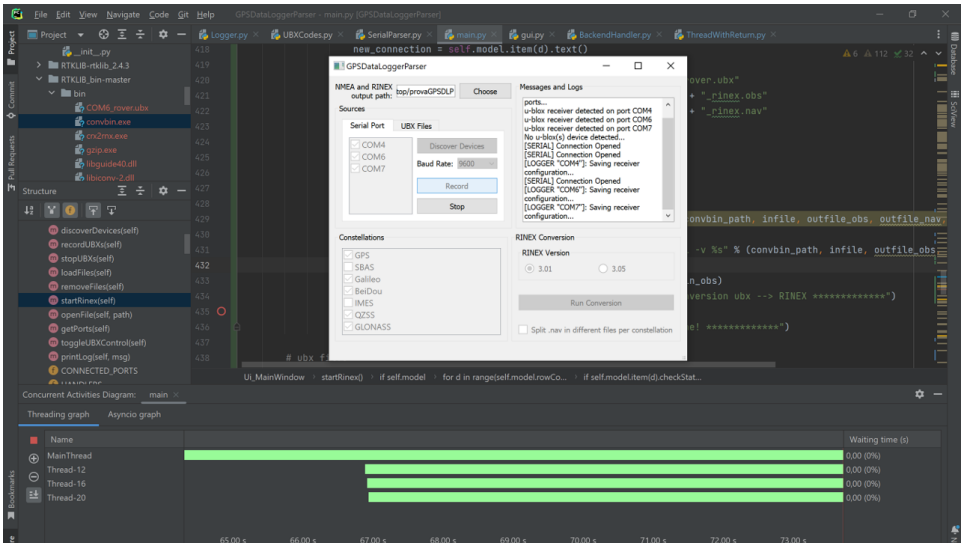


Figura 2: debug del software con monitoraggio dei thread in esecuzione

Ciascun thread registra il flusso di ciascuna periferica soltanto dopo averle inviato messaggi di configurazione, che comprendono:

- salvataggio della configurazione attuale del ricevitore;
- configurazione del rate con cui devono viaggiare i messaggi (1Hz);
- richiesta di inviare al PC messaggi inerenti esclusivamente le costellazioni (GNSS) selezionate;
- richiesta di inviare al PC messaggi di tipo UBX-RXM-RAWX e UBX-RXM-SFRBX, indispensabili per la generazione dei files RINEX;
- richiesta di inviare al PC messaggi NMEA di tipo GGA, disabilitando tutti gli altri;

Successivamente a questi comandi, vengono aperti i vari files all'interno dei quali saranno memorizzate le informazioni, passando così allo stato **"wait"**.

2.2.6 Registrazione dello Stream

Nello stato **wait** avviene la raccolta dei dati presenti nel buffer di entrata della periferica e il campionamento delle stringhe NMEA e dei messaggi UBX, note le strutture di seguito riportate che rendono possibile l'isolamento dei vari messaggi.

Infatti, una stringa NMEA inizia sempre con un carattere "$" e termina con il carattere speciale di ritorno a capo ("CRLF").

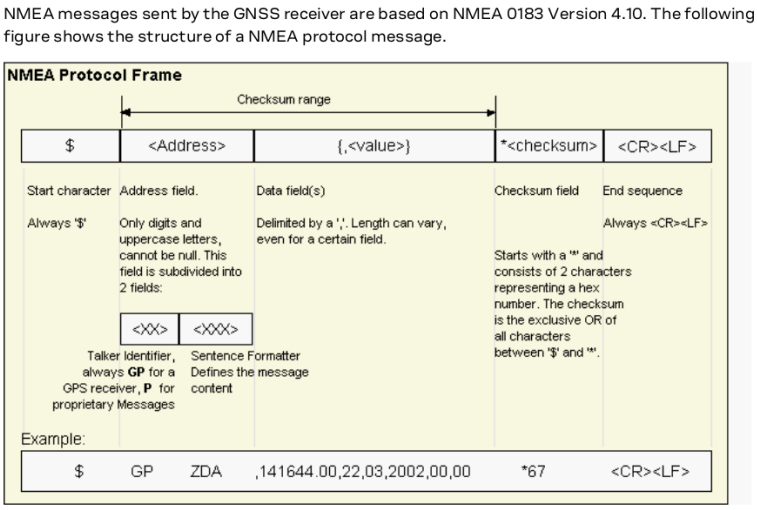


Figura 3: Struttura di una *sentence* NMEA

Un messaggio UBX possiede invece una struttura più complessa: inizia sempre con due bytes "B5 62" e, a seconda della classe e dell'identificativo (UBX-classe-id, es. UBX-RXM-RAWX), varia la struttura del `PAYLOAD`. I messaggi UBX terminano con due bytes di verifica (checksum) dell'integrità del messaggio, calcolati utilizzando l'algoritmo di Fletcher a 8-Bit, impiegato anche nello standard TCP (RFC1145).

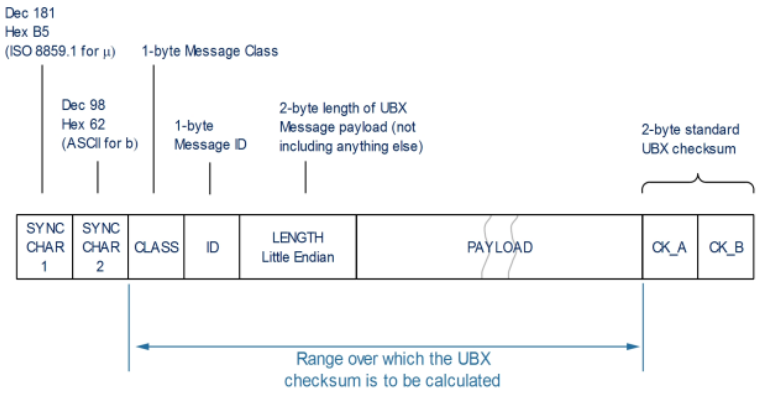


Figura 4: Struttura di un frammento UBX

Message	UBX-RXM-RAWX					
Description	**Multi-GNSS raw measurements**					
Firmware	Supported on: • u-blox 8 / u-blox M8 protocol versions 18, 19, 19.1, 19.2, 20, 20.01, 20.1, 20.2, 20.3, 22, 22.01, 23 and 23.01 (**only with ADR or High Precision GNSS or Time Sync products**)					
Type	Periodic/Polled					
Comment	This message contains the information needed to be able to generate a RINEX 3 multi-GNSS observation file (see ftp://ftp.igs.org/pub/data/format/). This message contains pseudorange, Doppler, carrier phase, phase lock and signal quality information for GNSS satellites once signals have been synchronized. This message supports all active GNSS. The only difference between this version of the message and the previous version (**UBX-RXM-RAWX-DATA0**) is the addition of the version field.					
Message Structure	Header	Class	ID	Length (Bytes)	Payload	Checksum
	0xB5 0x62	0x02	0x15	16 + 32*numMeas	see below	CK_A CK_B

Payload Contents:

Byte Offset	Number Format	Scaling	Name	Unit	Description
0	R8	-	rcvTow	s	Measurement time of week in receiver local time approximately aligned to the GPS time system. The receiver local time of week, week number and leap second information can be used to translate the time to other time systems. More information about the difference in time systems can be found in the RINEX 3 format documentation. For a receiver operating in GLONASS only mode, UTC time can be determined by subtracting the leapS field from GPS time regardless of whether the GPS leap seconds are valid.
8	U2	-	week	weeks	GPS week number in receiver local time.
10	I1	-	leapS	s	GPS leap seconds (GPS-UTC). This field represents the receiver's best knowledge of the leap seconds offset. A flag is given in the recStat bitfield to indicate if the leap seconds are known.
11	U1	-	numMeas	-	Number of measurements to follow

Figura 5: Contenuto del Payload di un messaggio UBX-RXM-RAWX

Le stringhe NMEA vengono aggiunte al file relativo alle stringhe (*"..._NMEA.txt"*). Da ciascuna stringa si estrae il tempo UTC (es. "$GNGGA, **071338.00**, 4233.83741,..."), quindi si aggiunge una nuova riga al file (*"..._NMEA_times.txt"*) contenente le corrispondenze tra tempo "locale" del PC e tempo UTC di ciascun messaggio NMEA-GGA.

Per quanto concerne i frammenti UBX, si considerano soltanto i messaggi UBX-RXM-RAWX, in quanto contengono il dato relativo al tempo della settimana `rcvTow`, da inserire in un file simile al precedente, anch'esso contenente la corrispondenza tra il tempo "locale" del PC e il valore riportato nel messaggio (*"..._times.txt"*).

2.2.7 Interruzione della Registrazione

Cliccando sul pulsante `Stop`, l'utente decide di **interrompere l'acquisizione** e passare quindi allo stato successivo. In questa fase, vengono chiusi i files aperti e progressivamente riempiti nello stato precedente e diventa poi pos-

sibile generare i files RINEX, cliccando sul pulsante `Run Conversion`, dopo aver selezionato la versione del file.

2.2.8 Generazione Files RINEX

In questo stato avviene la **generazione dei file RINEX** mediante la libreria CONVBIN. Il programma lancia un comando riferendosi all'eseguibile della libreria, indicando il file .ubx di ciascun ricevitore. Al **termine della generazione**, il software chiede all'utente se desidera aprire la directory all'interno della quale troverà tutti i risultati delle elaborazioni.

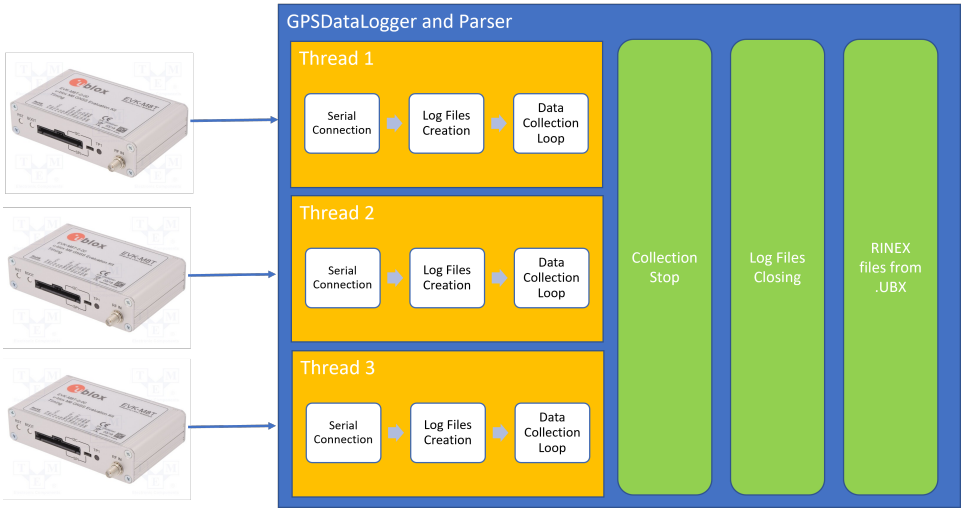


Figura 6: Sintesi del funzionamento del programma

2.3 Architettura BackEnd del Software

Il software è stato strutturato in moduli, ciascuno dei quali con un ruolo ben preciso. Ogni modulo corrisponde ad una classe di codice Python.

Figura 7: riepilogo delle classi di codice implementate

Il file principale del software è `main.py` che si occupa di istanziare la GUI e associare a ciascun elemento grafico le operazioni da eseguire. `main.py` contiene le routine principali che consentono la scelta della cartella, la scelta dei GNSS, la ricerca dei dispositivi, il caricamento di files da elaborare e il codice per l'esecuzione da codice di CONVBIN per la generazione dei RINEX.

Alla pressione del pulsante per la registrazione del flusso UBX, per ogni ricevitore, viene creata un'istanza dell'oggetto `BackendHandler`. Questa classe è il cuore del programma, infatti:

- gestisce la connessione alla periferica creando un oggetto `SerialParser`, che consente di monitorare lo stato della periferica, dopo aver aperto la connessione con un dato baudrate, e di gestire eventuali errori;
- crea un'istanza dell'oggetto `Logger` che contiene tutte le funzioni per la codifica/decodifica dei messaggi da e verso il ricevitore, grazie a `SerialParser`. In particolare, contiene la funzione *logData* per l'ascolto dei dati dal ricevitore, e quindi la creazione dei files che conterranno le stringhe NMEA, i time-stamp e il flusso UBX;
- crea un'istanza di `ThreadWithReturn`, che estende la classe `Thread`, specificando come funzione target la procedura *logData* contenuta nell'oggetto `Logger` precedentemente creato. L'esecuzione di questa istanza comporta la creazione e l'avvio di un thread che morirà soltanto alla pressione del pulsante che interrompe l'acquisizione del flusso per poi finalizzare le operazioni.

3 Test Eseguiti

Al fine di verificare il corretto funzionamento del programma in ogni sua componente, sono stati condotti numerosi test in vari momenti del processo di sviluppo. In particolare:

1. il primo ciclo di test è stato condotto al termine dello sviluppo dell'oggetto `SerialParser`. Nel dettaglio, è stato connesso un ricevitore e sono state testate tutte le funzioni per l'apertura e chiusura della connessione, la gestione degli errori dovuti ad eventi imprevisti (es. disconnessione improvvisa del device) e l'acquisizione dei dati inviati dalla periferica al pc mediante lettura del buffer in ingresso e svuotamento dello stesso;

2. il secondo test è stato incentrato sull'oggetto `Logger`, in particolare sulle funzioni che - dato lo stream di byte - elaborano il contenuto estrando le stringhe NMEA e i messaggi UBX, creando poi i vari files;

3. i focus del terzo test sono stati gli oggetti `ThreadWithReturn` e `BackendHandler`. È stata testata l'operatività del programma con più ricevitori (in parallelo, grazie alla libreria per il multithreading);

4. il quarto set di test è stato incentrato sulla GUI. In particolare, è stata testata l'integrazione con gli oggetti sviluppati e provati singolarmente e sono stati raccolti elementi utili a migliorare l'interfaccia e l'esperienza utente;

5. il quinto ed ultimo gruppo di test è stato dedicato al funzionamento generale del programma, inizialmente con un ricevitore, per poi passare a più ricevitori e terminare con più dispositivi e più files, testando contemporaneamente entrambi gli scenari di utilizzo del software.

4 Conclusioni e Commenti

Il progetto è stata un'occasione importante per misurare quanto appreso in questi tre anni di università in un'attività di sperimentazione legata al mondo delle architetture dei sistemi di elaborazione.

Aver sviluppato un software per l'università degli Studi di Ferrara, in collaborazione con i docenti del Dipartimento di Ingegneria Civile e Ambientale, mi ha permesso di approfondire tutte le fasi che precedono lo sviluppo di un software, come l'ascolto delle necessità a cui rispondere con una soluzione in grado di soddisfare i requisiti richiesti. Successivamente, dopo aver chiarito tutti gli aspetti implementativi è iniziata la fase di sviluppo che ha visto la nascita di un prodotto open-source innovativo, veloce, scalabile, modulare e cross-platform: non è presente nelle principali repository di software

open-source un programma sviluppato in Python con le finalità di *GPSData-Logger and Parser*. L'accesso libero al codice e l'approccio modulare seguito nelle fasi di sviluppo consentono l'utilizzo di questo software all'interno di progetti più grandi e il suo ampliamento con ulteriori funzionalità, come ad esempio il plot dei risultati acquisiti in tempo reale o la comunicazione dei flussi acquisiti su server in rete.

4.1 Bibliografia e Sitografia

- manuale del protocollo UBX: `https://content.u-blox.com/sites/default/files/products/documents/u-blox8-M8_ReceiverDescrProtSpec_UBX-13003221.pdf`
- manuale operativo del software u-center: `https://content.u-blox.com/sites/default/files/u-center_Userguide_UBX-13005250.pdf`
- manuale operativo del tool RTKLib: `https://www.rtklib.com/prog/manual_2.4.2.pdf`
- dettaglio dei ricevitori utilizzati: `https://content.u-blox.com/sites/default/files/products/documents/EVK-M8T_UserGuide_%28UBX-14041540%29.pdf`
- descrizione dello standard NMEA e RINEX;
- documentazione delle librerie utilizzate:
  - pySerial: `https://pyserial.readthedocs.io/en/latest/`
  - os: `https://docs.python.org/3/library/os.html`
  - NumPy: `https://numpy.org/doc/`
  - threading: `https://docs.python.org/3/library/threading.html`