Orlati Andrea [1], Monti Lorenzo [1], Mariotti Sergio [1]

Software per la caratterizzazione e l'integrazione nel sistema di osservazione del SRT del ricevitore banda Q

Relazione Tecnica

11 settembre 2023

[1] INAF, Istituto di Radioastronomia di Bologna, Via Fiorentina, 3513, Medicina (BO).

INDICE

CAPITOLO 1
INTRODUZIONE

Il presente documento ha l'obiettivo di descrivere la struttura interna e le funzionalità del software denominato *calibrate-multifeed-receiver*, realizzato per agevolare la caratterizzazione e l'integrazione di un ricevitore. Si andrà dapprima a descrivere il setup hardware del sistema, l'utilizzo di un analizzatore di spettro, nello specifico i test sono stati effettuati con un dispositivo Anritsu MS2830A, verranno poi introdotti e descritti (i) il package implementato in Python 3, (ii) l'interfaccia grafica scritta in Tk per facilitare l'operatore e (iii) la classe Python che si occupa di gestire l'hardware remoto.

1.1 Setup Hardware

Come riportato in Tabella 1.1, saranno elencati gli strumenti che compongono il sistema di acquisizione insieme a una breve descrizione delle loro caratteristiche di interesse per il progetto di riferimento. In Figura 1.1 è visibile lo schema del setup utilizzato, con i livelli di astrazione indicati a sinistra.

1.1.1 Test Layer

Il test layer, nasce dall'esigenza di testare il software (vedi la sezione 1.1.3), dando in input un segnale che emulasse il suo corrispettivo reale all'analizzatore di spettro, prima che l'hardware del ricevitore fosse disponibile. In tal senso, sono stati predisposti due elementi hardware programmabili da remoto, come da Figura 1.2, per ottenere il risultato aspettato:

- **Alimentatore programmabile Siglent SPD 3303X Linear DC:** Il Siglent SPD3000X, visibile in Figura 1.2 (rettangolo blu), dispone di tre uscite isolate; Due canali regolabili e un

Tipologia	Modello	Descrizione	Link
Analizzatore di spettro	Anritsu MS2830	strumento che permette l'analisi dello spettro dei segnali posti al suo ingresso.	https://shorturl.at/gxyJP
Attenuatore	DAT64F	attenuatore programmabile di passo RF compatto, autonomo e controllato tramite USB con passi di 0,25 dB e un intervallo da 0 a 63 dB.	https://shorturl.at/dlDX9
Alimentatore	Siglent SDP 3003X	alimentatore programmabile, supporta la programmazione remota e dispone di un display in tempo reale TFT-LCD da 4.3 pollici.	https://shorturl.at/jstHI
Pc per l'operatore	Pc generico con OS Unix-like o Windows	//	//

Tabella 1.1: Elenco dei dispositivi hardware utilizzati

canale selezionabile da 2,5 V, 3,3 V e 5 V. Include al suo interno inoltre, la protezione da cortocircuito e sovraccarico in uscita e può essere utilizzato in produzione e sviluppo.

E' corredato di un pannello frontale, utile a poter visualizzare cinque gruppi di impostazioni di temporizzazione e controllo dell'uscita, fornendo agli utenti una semplice funzionalità di programmazione della potenza. Inoltre, collegandosi tramite interfaccia ethernet (oppure USB tramite software per PC EasyPower di Siglent) è possibile utilizzare una gamma completa di funzionalità di comunicazione e controllo. In tal senso, la comunicazione avviene tramite un socket di tipo TCP su una porta definita dal produttore del device. Nel dispositivo utilizzato è in uso la porta 5025. Una volta connessi al dispositivo è possibile spedire il messaggio.

Lo scopo principale del suo utilizzo è stato quello di simulare l'accensione e lo spegnimento della marca nella fase di test del software.

- **Attuatore DAT64F - 6GHz Programmable finestep attenuator:** L'attenuatore digitale programmabile DAT64F, visibile in Figura 1.2 (rettangolo giallo), di *DS Instruments* è un attenuatore di passo RF compatto, autonomo e controllabile tramite USB con passi di 0,25

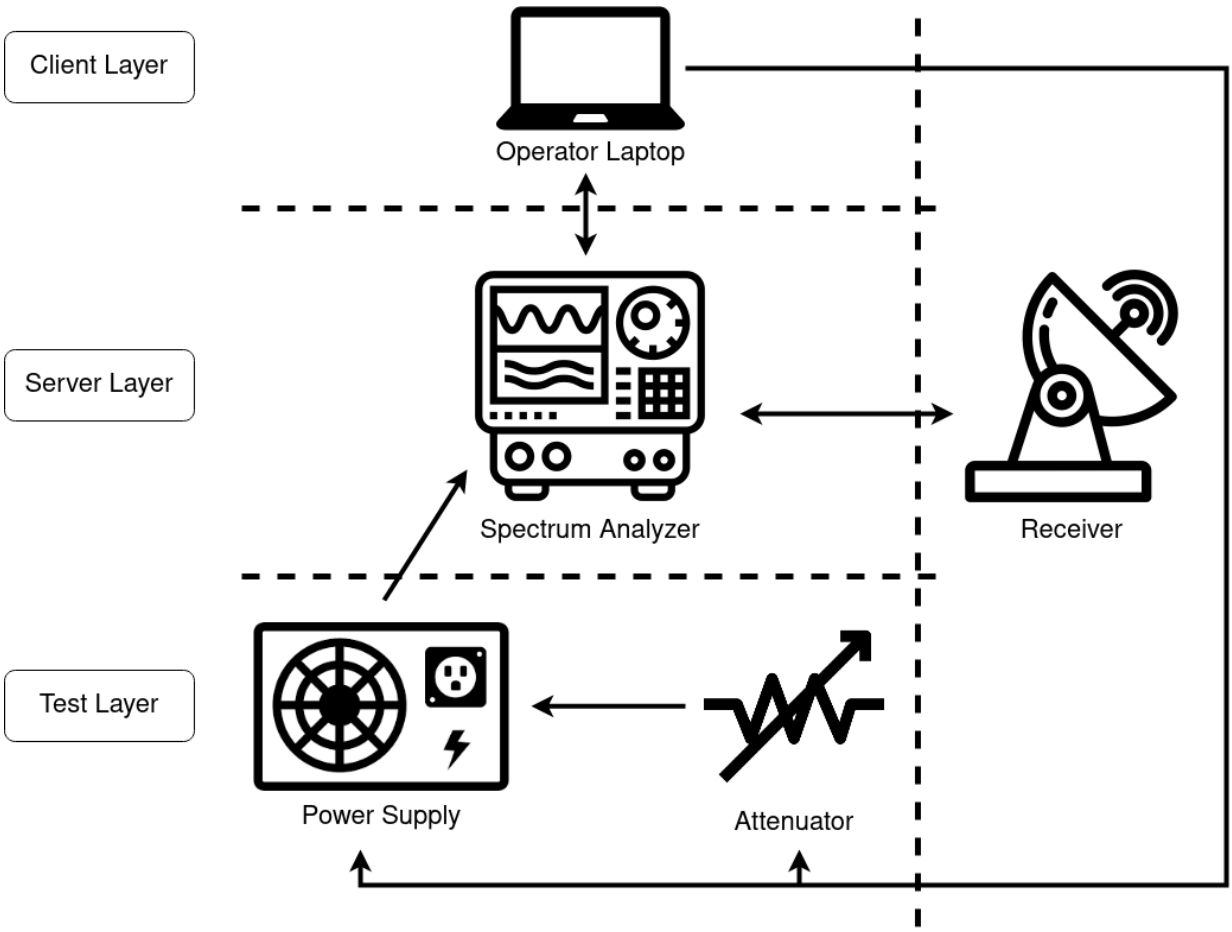


Figura 1.1: Schema di setup hardware utilizzato.

dB e un intervallo da 0 a 63 dB. La versione utilizzata è dotata di un display OLED luminoso e pulsanti di controllo utente frontali. La frequenza di ingresso varia da 100 MHz a 6 GHz.

Per utilizzare questo dispositivo non è necessario alcun PC host. Il controllo automatizzato tramite script è estremamente semplice e non richiede driver aggiuntivi, solo una semplice connessione alla porta COM virtuale e comandi di testo basati su SCPI. Nella fattispecie è stato scritto uno script in Python 3, eseguibile a riga di comando capace di comunicare con l'hardware sfruttando il protocollo di rete basato su *Telnet*. La porta, definita dal produttore del dispositivo, non è quella di default del protocollo utilizzato, ma bensì la 10001.

Lo scopo dell'utilizzo di questo dispositivo è la modifica dei dB di attenuazione che vengono impostati tramite i seguenti comandi SCPI:

– Per impostare il valore di attenuazione espresso in dB:

```
att <valore_floating_point_in_dB>\n
```

– Per leggere il valore impostato:

```
att?\n
```

Per maggior chiarezza, sono presentati tutti gli step di test con la marca (5) in Tabella 1.2, emulando un caso reale e sfruttando i dispositivi hardware sopra descritti:

	Step	**Alimentatore Siglent SDP 3003X**	**Attenuatore DAT64F**
1	Pc	OFF	63.0
2	Pc+m	ON	63.0
3	Ph+m	ON	21.5
4	Ph	OFF	21.5
5	Pc	OFF	63.0

Tabella 1.2: Step di test con la marca.

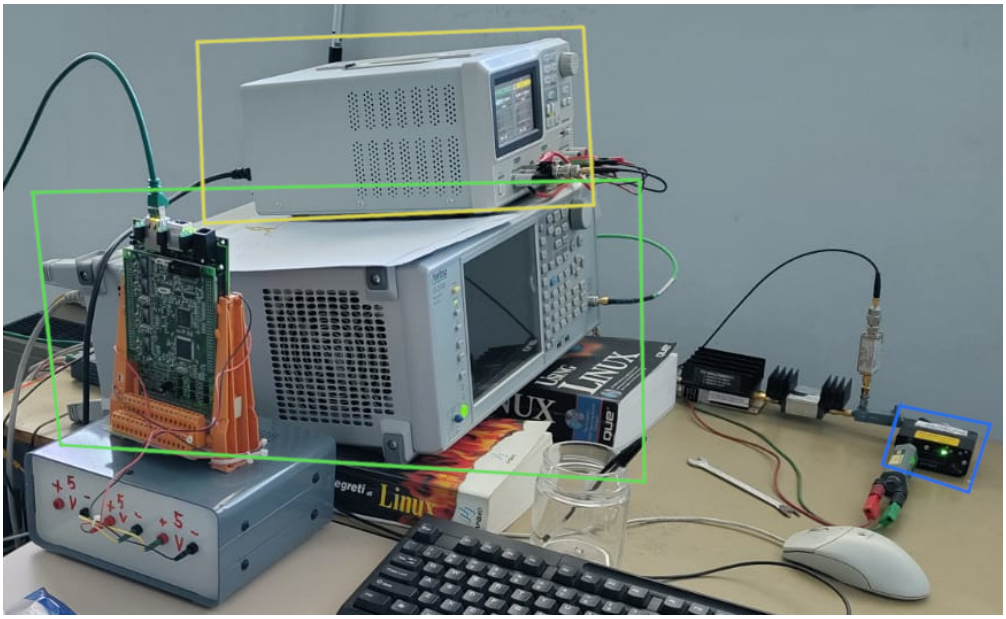


Figura 1.2: Setup hardware per i test in laboratorio.

1.1.2 Server Layer

In questo layer troviamo il dispositivo hardware centrale per la calibrazione e caratterizzazione di un ricevitore multifeed, l'*analizzatore di spettro*. E' visibile nel riquadro color verde presente in Figura 1.2.

Il modello utilizzato per gli esperimenti è l'MS2830A di casa Anritsu che supporta le tecnologie digitali e analogiche più comuni utilizzate nelle comunicazioni wireless cellulari, radio mobili terrestri/radio di pubblica sicurezza e IoT/M2M. Il dispositivo MS2830A consente all'utente di acquisire segnali dal mondo reale e analizzarli in laboratorio.

Il dispositivo in uso è configurabile sia attraverso interfaccia *General Purpose Interface Bus (GPIB)* che tramite collegamento Ethernet. Il software nominato *calibrate-multifeed-receiver* è stato implementato affinché sia idoneo alla connessione e successiva configurazione del dispositivo in entrambe le modalità sopracitate. In particolare, per quanto riguarda la configurazione GPIB, quella di default è la seguente:

GPIB::18::INSTR

Nel caso di connessione tramite interfaccia ethernet invece:

- *TCPIP0::*$< ip\_rete\_interna >$*::49153::SOCKET* per quanto riguarda l'utilizzo nella rete locale.
- *TCPIP0::localhost::49153::SOCKET* per l'utilizzo tramite rete remota.

1.1.3 Client Layer

Questo è il layer dove risiede effettivamente il software per la calibrazione e caratterizzazione di un ricevitore multifeed. Il software è installabile sul pc che dovrà interagire con il Server Layer 1.1.2. Il software è depositato in un repository Github, così da poterlo scaricare ed installare in locale, sul proprio sistema operativo. E' stata predisposta l'installazione sia per sistemi Unix-like che per sistemi Windows. Per maggiori dettagli leggere il *Manuale Utente - Software per la*

caratterizzazione e l'integrazione nel sistema di osservazione del SRT del ricevitore banda Q. I dettagli software verranno descritti con maggior perizia nel prossimo capitolo.

Durante la fase di progettazione del software, effettuata in collaborazione con il collega Sergio Mariotti, sono stati creati alcuni mockup presenti nel repository, all'interno della cartella *mockup/*. Questo è utile per scomporre in componenti unitarie i requisiti funzionali e testare le funzionalità del software finale. Per la creazione dei mockup stato scelto Glade, un tool open-source. Come visibile in Figura 1.3 è stato deciso di suddividere l'interfaccia grafica in tre sezioni (tabs) separate: *(i) Connection* per gestire la connessione all'hardware remoto, *(ii) Configuration* per applicare i comandi necessari all'hardware di riferimento e *(iii) Measure* per eseguire le misurazioni necessarie. Le sezioni appena descritte verranno approfondite estensivamente nel prossimo capitolo.

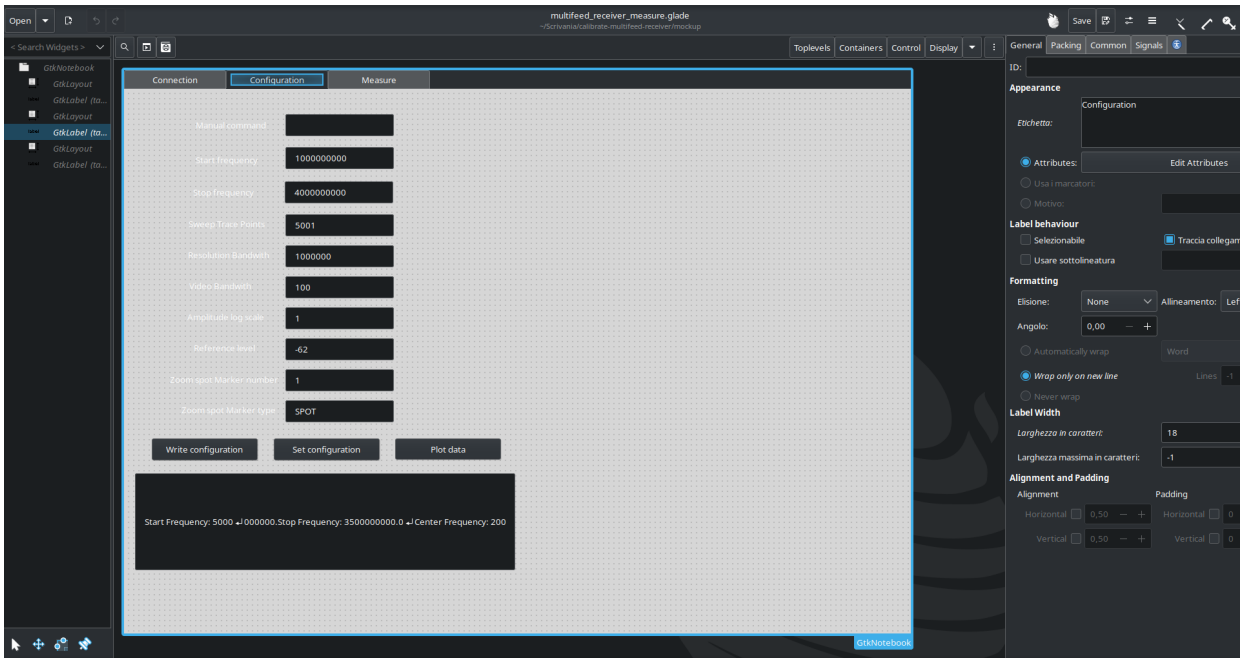


Figura 1.3: Interfaccia software per la creazione dei mockup.

CAPITOLO 2
CALIBRATE MULTIFEED RECEIVER - IL SOFTWARE

Come descritto brevemente nel capitolo precedente, il setup hardware si compone di tre layer separati, l'ultimo dei quali il *Client Layer* 1.1.3, il quale accoglie il software *calibrate-multifeed-receiver*. Tale software di calibrazione e caratterizzazione per un ricevitore multifeed è stato implementato interamente in Python 3. E' stata inoltre implementata un'interfaccia grafica cross-platform sfruttando la libreria *Tkinter* come interfaccia per il toolkit *Tk*. Verrà dapprima descritta la struttura del package e le scelte progettuali, di seguito l'interfaccia grafica e infine la classe di gestione del hardware remoto Anritsu_MS2830A.

2.1 Struttura del Package Python 3

2.1.1 config

In questa cartella sono archiviati i file di configurazione in formato *json*, attraverso i quali il programma legge/scrive i valori di default. In particolare: (i) *config_interface.json* contiene i valori presenti nella GUI, alla sezione *Connection* e (ii) *config_MS2830A.json* contiene i valori presenti nella GUI, alla sezione *Configuration* (di cui si parlerà più approfonditamente nel prossimo paragrafo).

2.1.2 data

Una volta effettuate tutte le misurazioni (5) e acquisiti conseguentemente tutti i dati, è possibile avviare il salvataggio tramite l'apposito pulsante *Save measurements* presente nell'area *Actions* della sezione *Measure*. Il file risultante, in formato csv, presenta alcune colonne derivanti dai dati acquisiti e altre da elaborazioni. La cartella *data* contiene questa tipologia di file.

2.1.3 manuals

In questa cartella sono presenti alcuni dei manuali relativi ai dispositivi hardware utilizzati. Questo può essere utile per avere una rapida lettura in caso di necessità.

2.1.4 mockup

Durante la fase di progettazione del software, effettuata in collaborazione con il collega Sergio Mariotti, sono stati creati alcuni mockup presenti nel repository, questa è la cartella che li contiene. Questo è utile per scomporre in componenti unitarie i requisiti funzionali e testare le funzionalità del software finale. Per la creazione dei mockup stato scelto *Glade*, un tool open-source, come già descritto in precedenza.

2.1.5 src

Il contenuto della cartella *src/* comprende tutta la *logica di business* per rendere operativo l'applicativo, e in maniera parziale la *logica di presentazione*. Come visibile in Figura 2.1, il diagramma del package mostra le relazioni tra i diversi file. Di seguito verranno presentati i file di scripting presenti nella cartella:

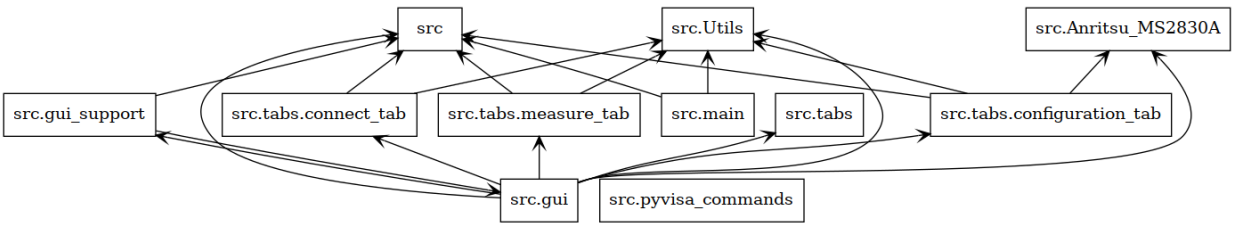


Figura 2.1: Struttura della directory *src*, riferita al package calibrate-multifeed-receiver.

- **src.gui:** entry point dell'applicativo, necessario per creare l'interfaccia utente, i dettagli verranno spiegati in maniera più precisa nel prossimo paragrafo.
- **src.gui_support:** file con funzioni di supporto alla classe *UserInterface* presente nel file *gui.py*.
- **src.Anritsu_MS2830A:** lo script si compone di una singola classe predisposta per la gestione della comunicazione con l'hardware remoto. I dettagli verranno spiegati con maggior perizia nei prossimi paragrafi.
- **src.main:** script di testing utilizzato per verificare diverse funzioni senza la necessità ogni volta di dover installare il package.
- **src.tabs:** directory che include le tre sezioni dell'interfaccia utente, come segue:
  - **src.tabs.connect_tab:** file modulo che si occupa di popolare visivamente la sezione chiamata *Connect*.
  - **src.tabs.configuration_tab:** file modulo che popola visivamente la sezione chiamata *Configuration*.
  - **src.tabs.measure_tab:** file modulo che istanzia tutti i widget nella sezione chiamata *Measure*.
- **src.pyvisa_commands:** questo file è utilizzato principalmente per testare comandi nuovi.
- **src.Utils:** file modulo dove sono raccolte alcune funzioni di utility necessarie al package, come ad esempio file management dei file di configurazione, plotting dei dati, User Interface e calcolo di formule.

2.1.6 theme

Questa cartella contiene tutti i file relativi alla *User Interface (UI)*. E' stato utilizzato un tema Tcl, in licenza MIT, dal nome Azure-ttk-theme, presente su Github.

2.1.7 File di gestione e configurazione del package

Il package Python in questione ha diversi file utili per la gestione, la configurazione, l'installazione delle dipendenze e del package. Andremo ad analizzarli di seguito:

- **.gitignore:** tipicamente aggiunto nella cartella root di un progetto, questo file di testo indica a Git quali file e cartelle non caricare in un progetto.
- **LICENSE** file di testo nel quale viene definita la licenza con cui un progetto open-source viene rilasciato. Nel caso specifico il progetto è stato rilascio con licenza MIT.
- **MANIFEST.in:** file di configurazione utile per la creazione di una *Source distribution(sdist)*. Esso contiene il file *setup.py* (contenente informazioni su modulo/metadati), i file sorgente dei moduli, file di dati, eccetera. Il risultato è un archivio che può quindi essere utilizzato per ricompilare tutto su qualsiasi piattaforma come Linux, Windows e Mac.
- **README.md:** file testuale che contiene informazioni importanti relative al progetto.
- **requirements.txt:** file che memorizza informazioni su tutte le librerie, i moduli e i pacchetti specifici oltre che alla risoluzione delle dipendenze. Il comando da lanciare una volta scaricato il repository, assicurandosi di essere nella directory giusta, è il seguente:

```
$ python -m pip install -r requirements.txt
```

- **setup.py:** in Python, setup.py è un modulo utilizzato per creare e distribuire pacchetti Python. Contiene informazioni sul pacchetto, ad esempio nome, versione e dipendenze, nonché istruzioni per la creazione e l'installazione del pacchetto. Queste informazioni vengono utilizzate dallo strumento *pip*, gestore di pacchetti per Python, che consente agli utenti di installare e gestire pacchetti Python dalla riga di comando. Eseguendo il file *setup.py* con lo strumento pip, puoi creare e distribuire il tuo pacchetto Python in modo che altri possano usarlo, come segue:

```
$ python setup.py install
```

Nel caso specifico, questo file è sfruttato per installare nel sistema il package. Questo avviene solamente nei sistemi Unix-like. Nei sistemi Windows vi è un eseguibile auto-contenuto.

- **generate_standalone_package.py:** Script che utilizza la libreria *PyInstaller* per creare un eseguibile auto-contenuto per sistemi Windows.

2.2 Interfaccia Grafica

L'interfaccia utente consente l'interazione uomo-macchina in modo visuale utilizzando rappresentazioni grafiche piuttosto che comandi a riga di comando. Nel caso d'uso specifico è utile per agevolare l'operatore durante la fase di misurazione. Questo è il motivo progettuale per il quale le tre sezioni, visibili in Figura 2.2 sono ben distinte. Dal punto di vista implementativo l'entry point dell'applicativo punta alla funzione *vp_start_gui()* all'interno del file Python *gui.py* come mostrato nel file *setup.py* (file che consente l'installazione nel sistema operativo), di seguito un estratto:

```
...
    setup(
        name='calibrate-receiver',
        version='0.8.0',
        description='A graphic tool used to calibrate a
                     multifeed receiver',
        long_description=README,
        long_description_content_type="text/markdown",
        url="https://github.com/LorenzoMonti/calibrate-multifeed-receiver",
        author='Lorenzo Monti',
        author_email='lorenzo.monti@inaf.it',
        packages=['config', 'src', 'src/tabs', 'theme'],
        include_package_data=True,
```

```
        entry_points={
        'console_scripts': [
            'calibrate_receiver=src.gui:vp_start_gui',
        ]
    },
...
```

La suddetta funzione è atta a:

1. Creare una root windows istanziando il costruttore TK di Tkinter (libreria grafica scelta), come segue: $Root = tk.TK()$.

2. Richiamare tutti i file tcl (*azure.tcl*) contenenti i widget necessari ad ottenere un tema coerente.

3. Istanziare la classe *UserInterface*, la quale si occupa di configurare e popolare la finestra di primo livello. Inoltre, vengono letti i file configurazione per la corretta configurazione dell'hardware in uso. Infine, vengono create le sezioni, di cui parleremo di seguito.

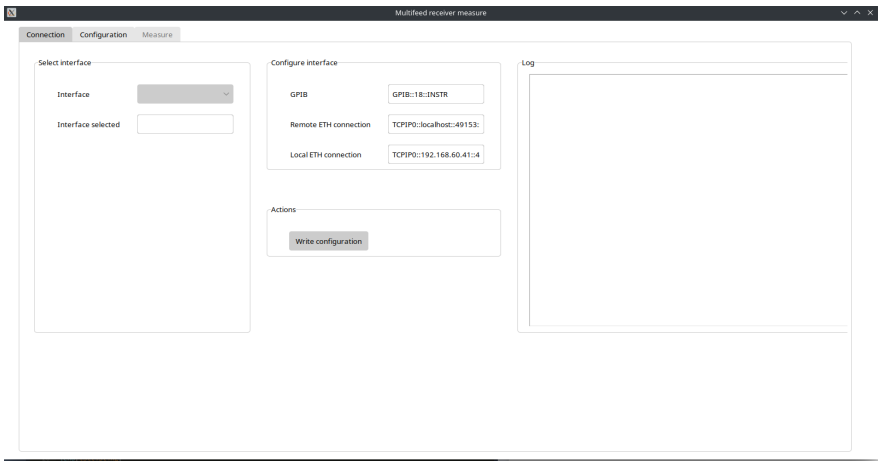
4. Inizializzare il metodo *mainloop()*, ciclo infinito che riceve automaticamente gli eventi dal sistema a finestre e li consegna ai widget dell'applicazione. Questo viene chiuso quando si fa clic sul pulsante di chiusura della barra del titolo. Pertanto, qualsiasi codice successivo a questo metodo non verrà eseguito.

L'implementazione delle tre sezioni si trova al percorso *src/tabs/*. La sezione *Connection* Figura 2.2 (a) deputata alla gestione e alla configurazione della connessione locale e remota. Il software permette la connessione utilizzando l'interfaccia General Purpose Interface Bus (GPIB) oppure tramite connessione TCP (in una rete locale e remota). Il file *connect_tab.py* popola di widget la sezione attraverso la funzione *connect()*. Il popolamento delle sezioni viene effettuato in maniera puntuale per ognuna di essere in ottica di riutilizzo del codice, nel caso si volesse espandere il numero di sezioni e il software in generale.

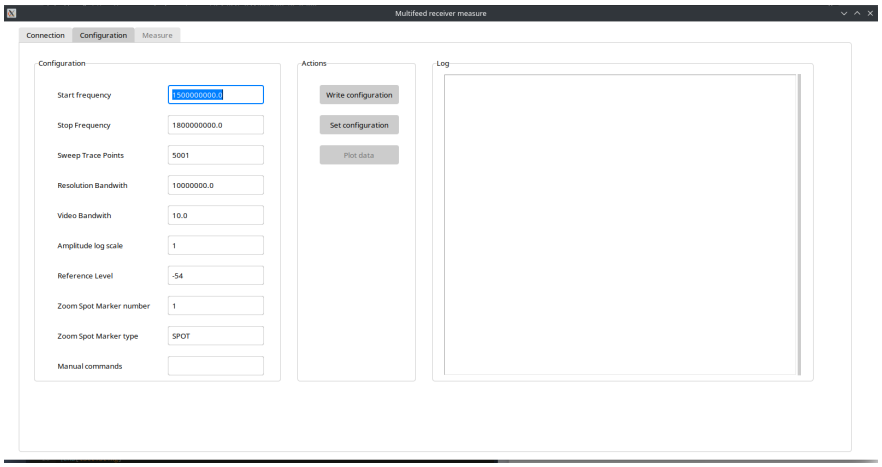
La sezione *Configuration* Figura 2.2 (b) è deputata all'apertura della connessione con l'hardware e alla configurazione dell'analizzatore di spettro. Nell'area Configuration sono stati impostati i valori utilizzati con maggior frequenza per lo scopo di questa tipologia di software. E' comunque possibile immettere comandi manuali purché aderiscano allo standard IEEE488.2 Common Device Messages. Per stabilire la connessione è sufficiente cliccare sul pulsante *Set configuration*, purché nella sezione precedente si sia impostato correttamente la tipologia di connessione voluta. Implementativamente è richiamata la funzione *set_conf()* che si occupa di richiamare i dati necessari per (i) connettersi al dispositivo remoto, (ii) secondariamente istanziare la classe dove risiedo tutti i metodi necessari alla comunicazione con quest'ultimo (Anritsu_MS2830A) e (iii) includere i log nell'apposita area. Una volta che la connessione è stabilita è possibile plottare i dati ricevuti dall'hardware di riferimento, così da poter modificare, se vi fosse la necessità, i valori necessari per visualizzare la configurazione desiderata. La funzione che se ne occupa è quella definita come *plot_data()*.

Infine, la sezione *Measure* Figura 2.2 (c) è preposta all'acquisizione e analisi dei dati per la successiva calibrazione e caratterizzazione di un ricevitore multifeed. L'area *Measures* è predisposta per rendere evidente ogni step dell'esperimento aiutando in questo modo l'operatore; Infatti, questo riquadro sottolinea i cambiamenti di stato sia in termini visivi (cambio colore di background sulla scritta) che sonori (un suono per indicare l'avvenuta acquisizione dei dati) ad ogni step dell'esperimento. Da un punto di vista implementativo ad ogni step viene richiamata la funzione *take_trace()*, tramite l'apposito pulsante *Take measurements* oppure premendo su tastiera backspace (come richiesto in fase progettuale). In entrambi i casi questo viene gestito come *evento* tramite un handler. In particolare, per ognuno degli step viene creato un thread (funzione get_thread_trace()) atto a richiedere la traccia su un thread separato. Questo per rendere non bloccante il processo e non bloccare interattivamente l'intero applicativo (l'elaborazione impiega diversi secondi). Una volta conclusa l'acquisizione dell'ultima traccia vengono (i) calcolate le frequenze, (ii) i valori D e D_R, (iii) elaborati i valori raw di partenza ed i (iv) valori Y, T_{rx}, T_m, T_m^h illustrati in dettaglio nel manuale utente. Infine, viene generato un grafico, utile per una veloce valutazione sulle misurazioni

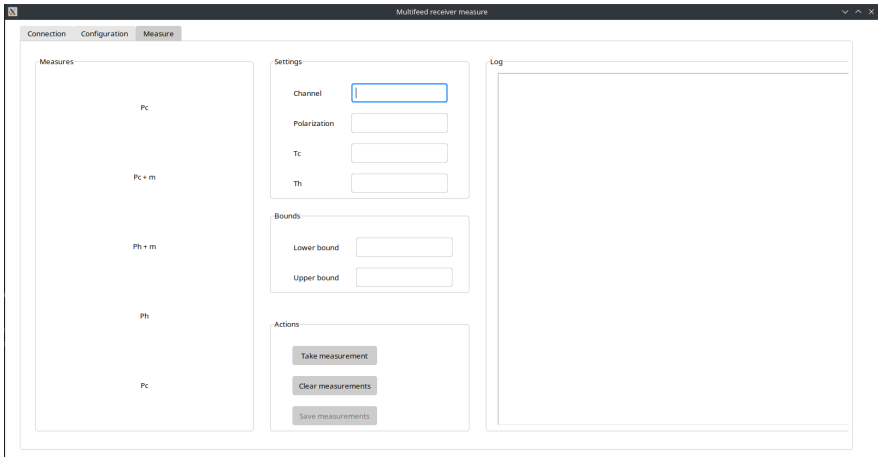
e intercettare possibili problemi avvenuti durante l'acquisizione.



(a)



(b)



(c)

Figura 2.2: Interfaccia grafica (GUI) composta da tre sezioni separate (tabs).

2.3 Classe Anritsu_MS2830A

La classe *Anritsu_MS2830A* è deputata alla corretta gestione dei messaggi verso l'analizzatore di spettro Anritsu MS2830A. La classe viene istanziata quando viene cliccato il pulsante *Set configuration*, quindi, implementativamente quando viene chiamata la funzione *set_conf()* che si trova nel percoso *src/tabs/configuration_tab.py*. Il costruttore si occupa di configurare il backend NI-VISA tramite l'oggetto *ResourceManager()*. Esso è messo a disposizione da *PyVISA*, una libreria Python che consente di controllare tutti i tipi di dispositivi di misurazione indipendentemente dall'interfaccia (ad esempio GPIB, RS232, USB, Ethernet). Inoltre, vengono esposti i metodi necessari per il controllo remoto della bandwith, il time sweep, la frequenza, power unit, il marker, l'amplitude e della traccia; La lista completa è visibile in Figura 2.3.

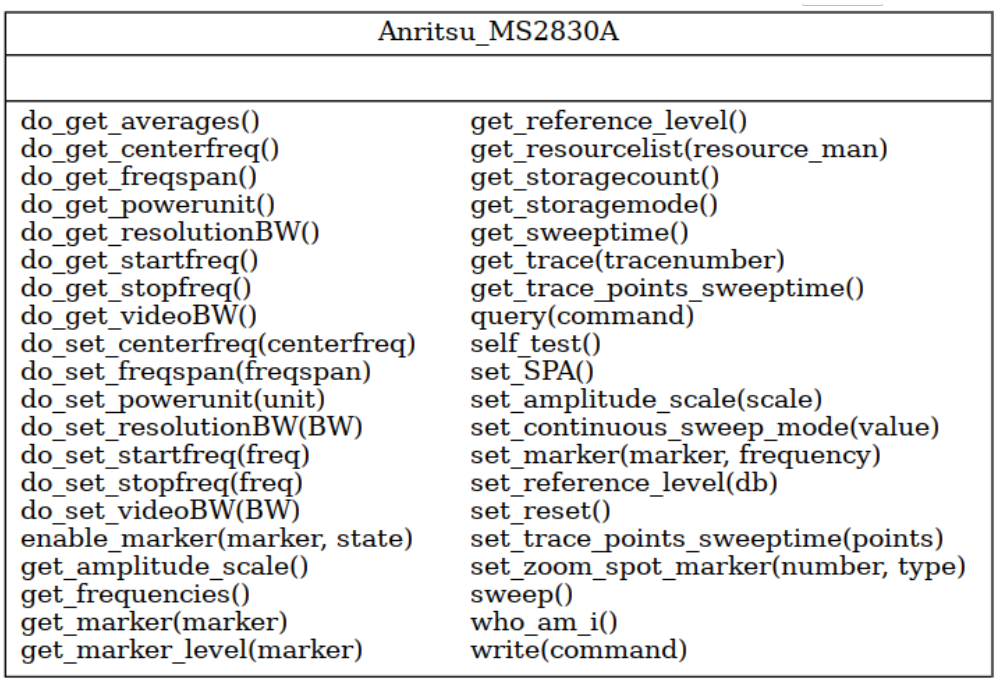


Figura 2.3: Metodi della classe Anritsu_MS2830A.

2.3.1 Implementare una classe per un nuovo dispositivo hardware

Il software *calibrate-multifeed-receiver* è stato progettato per poter implementare ed integrare facilmente un nuovo dispositivo hardware che utilizzi una delle seguenti interfacce: GPIB, RS232, USB oppure Ethernet. Nella fattispecie, è sufficiente istanziare l'oggetto *ResourceManager()* similmente a ciò che è stato fatto per la classe *Anritsu_MS2830A*:

```
...
# initialize instrument
_resource_man = pyvisa.ResourceManager()
self._list_res = _resource_man.list_resources()
self._visainstrument = _resource_man.open_resource(self._address,
                        write_termination = '\n',read_termination='\n')
self._visainstrument.encoding = "latin-1" # encoding
...
```

E successivamente implementare i metodi per l'invio del messaggio corretto, discernendo tramite metodi di *set* e di *get*. Nel primo caso verrà utilizzato il metodo *write()* mentre nel secondo il metodo *query()* messi a disposizione dalla libreria. Di seguito due esempi per la definizione dei metodi:

```
# metodo di set
def set_reset(self):
        return self._visainstrument.write('*RST')

# metodo di get
def who_am_i(self):
        return str(self._visainstrument.query('*IDN?'))
```