ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA E SCIENZE INFORMATICHE

Implementazione in Python della modulazione digitale Olivia

Elaborato in

METODI NUMERICI

Relatore

Damiana Lazzaro

Presentata da

Federico Santandrea

Sessione unica

Anno Accademico 2019/20

Indice dei contenuti

Introduzione

Questo lavoro di tesi si propone di ricostruire ed illustrare in modo completo la struttura del sistema di trasmissione Olivia a partire dai documenti disponibili, arrivando allo sviluppo di un'implementazione pratica e chiara in linguaggio Python della quale verranno misurate le prestazioni in scenari simulati e nel mondo reale.

Olivia è una modulazione digitale per la comunicazione di dati testuali in condizioni di elevato rumore, nata per utilizzo in ambito radioamatoriale ove sia necessaria un'alta resistenza ai disturbi e non si richiedano velocità di trasmissione elevate. È un sistema di trasmissione non banale che unisce varie tecniche di elaborazione numerica e correzione degli errori.

Allo stato attuale la specifica formale di Olivia consiste sostanzialmente nell'implementazione di riferimento in C++ e in un documento riassuntivo non completo. Esiste al momento una limitata scelta di programmi per computer gratuiti o commerciali che la supportino. Non è ancora disponibile una implementazione snella e facilmente comprensibile in un linguaggio di programmazione moderno.

Dai tempi delle sue prime incarnazioni, Python ha vissuto e sta vivendo un'intensa evoluzione da semplice linguaggio di scripting ad ecosistema completo. Questo lo rende una piattaforma sempre più appetibile per la ricerca ed il calcolo scientifico e numerico. L'ampia disponibilità di librerie open source di ogni tipo consente la verificabilità delle implementazioni degli algoritmi da parte della comunità scientifica. L'efficacia della sintassi e l'elevato grado di astrazione dell'hardware e delle interfacce sottostanti danno allo sviluppatore la libertà di concentrarsi sulla risoluzione del problema in esame senza spendere troppe energie per la gestione di basso livello, facendone un valido sistema di prototipazione rapida. Al contempo la solidità e l'efficienza raggiunte dopo anni di miglioramenti lo rendono adatto per la generazione finale di codice di produzione stabile e ottimizzato. Queste caratteristiche ne fanno una scelta naturale per perseguire gli obiettivi sopra esposti.

La tesi è organizzata come segue:

- nel Capitolo 1, “Background teorico”, si illustreranno i concetti alla base delle modulazioni digitali e dell’elaborazione digitale dei segnali, necessari per la comprensione della materia.
- nel Capitolo 2, “La modulazione Olivia”, si mostrerà la struttura del sistema di trasmissione Olivia, identificando e descrivendo i blocchi funzionali che compongono il modulatore e il demodulatore;
- nel Capitolo 3, “Dettagli implementativi”, saranno evidenziate le soluzioni tecniche scelte per l’implementazione realizzata, riportando e commentando i frammenti di codice più significativi;
- nel Capitolo 4, “Esperimenti e misure”, si riporteranno i risultati di prove pratiche di trasmissione e ricezione effettuate utilizzando i programmi implementati, raccogliendo dati quantitativi sulle prestazioni in vari scenari;
- nel Capitolo 5, “Conclusioni”, verranno infine riassunte le conclusioni tratte durante lo svolgimento delle fasi precedenti.

Capitolo 1

Background teorico

In questo capitolo si illustrano i concetti alla base delle modulazioni digitali e dell'elaborazione digitale dei segnali, necessari per la comprensione della materia.

Una **modulazione**, nel senso classico del termine, è una tecnica di trasmissione utilizzata per imprimere l'informazione contenuta in un segnale continuo nel tempo e nei valori (detto *modulante*) su una forma d'onda (detta *portante*, tipicamente sinusoidale) avente frequenza ordini di grandezza superiore a quella massima del segnale modulante e che quindi meglio si adatta alle caratteristiche del canale di trasmissione. Questa impressione si ottiene mediante la modifica di qualche proprietà del segnale portante in funzione dell'andamento del segnale modulante analogico. [LIT-1]

Una **modulazione digitale** è una tecnica che si utilizza per trasmettere dati (quindi sequenze di simboli) anziché segnali analogici. Concettualmente si potrebbe considerare qualunque sequenza di simboli digitali come un segnale modulante caratterizzato dal susseguirsi di diversi valori di ampiezza o frequenza, o una qualunque combinazione di essi e quindi applicare una modulazione analogica a questo ipotetico segnale. Nella pratica però il fatto che il segnale modulante sia discreto e possa assumere solo un numero finito di valori consente una notevole semplificazione della trattazione teorica e anche dell'implementazione pratica.

Infatti in questo caso non è solitamente necessario ricorrere a complicati circuiti analogici non lineari o algoritmi digitali complessi per ottenere il segnale modulato, ma è possibile generarlo direttamente, anche mediante tecniche di sintesi software che richiedono poca potenza di calcolo. Ad esempio, si possono utilizzare due toni di diversa frequenza ed eguale durata per rappresentare i bit 0 e 1; questa è la più semplice modulazione **FSK** (*Frequency Shift Keying*).

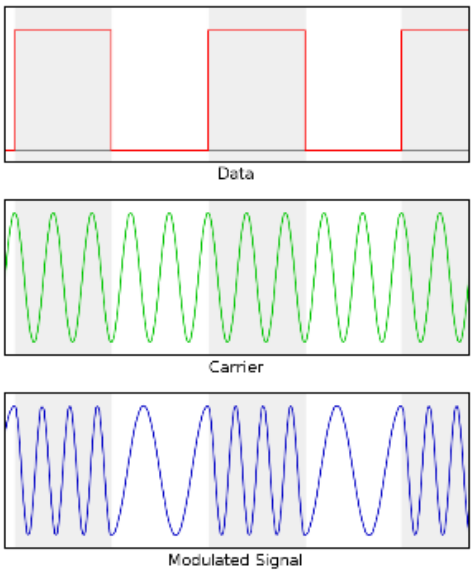


Figura 1: Una semplice modulazione FSK. [WIKI-1]

Naturalmente è possibile utilizzare più di due toni: in questo caso la rilevazione di uno specifico tono possibilità conterrà più di un bit di informazione. Con *n* toni, l'informazione ottenuta dalla rilevazione di un tono sarà pari a $\log_2(n)$ bit.

Esiste un'ampia varietà di modulazioni digitali che a partire da un flusso di dati generano una forma d'onda in banda base con spettro di larghezza molto limitata, tipicamente compresa all'interno di un range che spazia da 500 a 2500 Hz. Queste modulazioni vengono anche dette **sound card modes**, perché si prestano ad essere generate e interpretate da computer, utilizzando le normali schede audio per l'emissione e l'immissione del segnale. Le tipiche schede audio di fascia consumer integrate in ogni PC moderno lavorano alla frequenza di campionamento di 44100 Hz, ben superiore quindi alla frequenza di Nyquist di un segnale in banda base con frequenza massima di 2500 Hz. Utilizzando queste *sound card modes* è possibile scambiare dati digitali tramite segnali audio, prelevando il segnale generato dall'uscita per altoparlanti della scheda e viceversa immettendo il segnale ricevuto nell'ingresso per microfono.

È possibile realizzare anche la comunicazione in radiofrequenza, interponendo fra il computer trasmittente e quello ricevente un

ricetrasmettitore radio dotato di **modulazione SSB** (*single-sideband*). La modulazione SSB, tecnicamente una modulazione d'ampiezza privata della portante e di una delle due bande laterali, consiste in sostanza nella traslazione verso destra dello spettro del segnale modulante, di una quantità pari alla frequenza del segnale portante.

Il segnale audio risultato dalla modulazione digitale, prodotto dal computer, può quindi essere fornito in ingresso ad un apparato trasmittente SSB, che si occuperà di traslare verso l'alto lo spettro del segnale ad una frequenza appropriata alla propagazione desiderata. In ricezione, dopo aver effettuato la traslazione inversa mediante un ricevitore SSB, si fornirà il segnale audio ottenuto in ingresso al computer, il quale si occuperà della demodulazione digitale. Con questa tecnica si liberano computer e cablaggio dalle complicazioni dell'elettronica in radiofrequenza, e si opera semplicemente in elaborazione digitale su un segnale a bassa frequenza che può essere acquisito ed interpretato in software senza particolari problemi.

Generare il segnale modulato in caso di modulante digitale è quindi semplice: è sufficiente generare ed emettere i singoli campioni della forma d'onda d'uscita prevista teoricamente per la modulazione in uso. Anche la demodulazione risulta teoricamente più semplice e nei casi più elementari si può effettuare con metodi diretti. Ad esempio, si può determinare la frequenza di un segnale sinusoidale puro semplicemente contando quante volte la forma d'onda cambia di segno in un'unità di tempo (*zero crossing*); nel caso della FSK, è possibile in questo modo distinguere in ricezione toni sinusoidali di varie frequenze associati a simboli differenti.

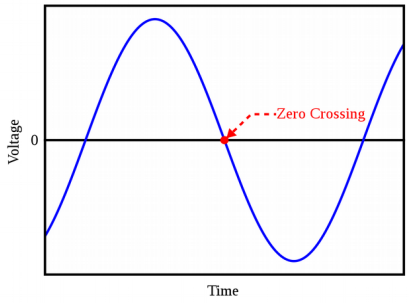


Figura 2: Rilevazione della frequenza mediante conteggio degli zeri. [WIKI-2]

1.1 Analisi di Fourier discreta

Lo strumento principe per l'analisi di forme d'onda rimane comunque la **trasformata di Fourier discreta** (DFT). Sebbene sia un metodo relativamente oneroso a livello computazionale, mette a disposizione del ricevitore tutte le informazioni sulla fase e l'ampiezza dei toni sinusoidali che compongono il segnale. Si aprono quindi molte possibilità per imprimere il contenuto informativo dei simboli nella forma d'onda generata in trasmissione.

La trasformata converte una sequenza finita di campioni equispaziati di un segnale analogico, in una sequenza finita di campioni equispaziati del suo spettro di frequenza. Rappresenta quindi una discretizzazione della trasformata di Fourier. Questa discretizzazione consente di affrontare il calcolo in modo numerico, senza dover trovare il modo di esprimere il segnale in ingresso in forma simbolica e derivarne poi lo spettro. Si applica per questo motivo molto bene all'implementazione con calcolatori elettronici.

Supponendo che la forma d'onda di un segnale da trasformare sia periodica, lo spettro corrispondente sarà discreto e corrisponderà alla sequenza dei coefficienti della serie di Fourier. Dovendo trasformare una qualunque porzione limitata nel tempo di una forma d'onda in generale non periodica, si otterrebbe invece uno spettro continuo; tuttavia è possibile considerare come forma d'onda la ripetizione periodica della porzione considerata, ottenendo così anche in questo caso una sequenza discreta di coefficienti.

Se inoltre la forma d'onda in ingresso viene campionata, ossia espressa come una sequenza di coefficienti che ne indicano i valori assunti a intervalli discreti di tempo, si avrà come conseguenza delle proprietà della trasformata che la sequenza dei coefficienti della serie di Fourier sarà periodica e, dunque, ne potrà essere considerato un singolo periodo senza che si verifichi perdita di informazione.

La combinazione di limitazione nel tempo ad una porzione del segnale in ingresso e della sua discretizzazione consente quindi di esprimerne il

contenuto armonico sotto forma di una sequenza discreta e finita di coefficienti.

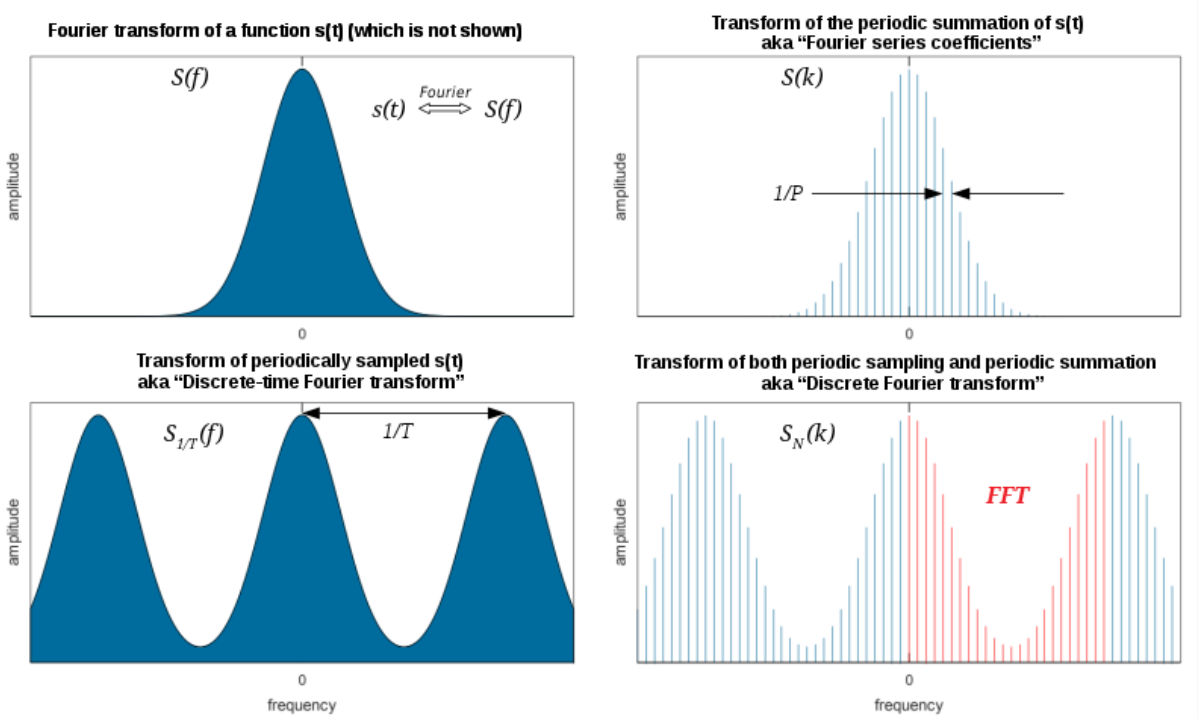


Figura 3: Visualizzazione della discretizzazione della trasformata di Fourier. [WIKI-3]

La successione dei campioni della forma d'onda è messa in relazione a quella dei campioni dello spettro dalla seguente formula:

$$X_k = \sum_{n=0}^{N-1} x_n \mathrm{e}^{-ik\frac{2\pi}{N}n} \qquad k = 0,\ldots,N-1$$

dove X_k è la successione dei campioni dello spettro, x_n la successione dei campioni della forma d'onda e N la lunghezza delle successioni. Si può notare che ogni campione dello spettro dipende da ogni campione della forma d'onda. I valori complessi della sequenza ottenuta rappresentano ognuno fase e ampiezza di una componente sinusoidale della forma d'onda del segnale d'origine. [LIT-2]

La quantità di informazione messa a disposizione dalla trasformata consente di implementare modulazioni in ampiezza, in frequenza e anche in quadratura. Unendo la potenza dell'elaborazione digitale, si possono implementare tecniche di multiplazione a divisione di tempo, di frequenza, *spread spectrum* e molto altro ancora. Questa flessibilità ne

fa un passaggio quasi obbligato dopo l'acquisizione del segnale analogico di origine in fase di ricezione, fatta eccezione per i casi di modulazione più semplici.

A livello teorico la trasformata di Fourier, e così anche la sua discretizzazione, opererebbero su valori complessi e non reali. Tuttavia, dal campionamento di un segnale analogico reale risultano campioni totalmente reali. Il risultato in uscita da una trasformata discreta di Fourier fatta su campioni reali sarà una sequenza di valori complessi (campioni dello spettro) simmetrica rispetto al centro. I valori sono complessi poiché ognuno di essi ha una ampiezza assoluta e una fase e questo consente di mantenere l'informazione sulla relazione di fase che esiste fra le varie componenti di frequenza.

In molti casi, come in quello in esame in questo documento, l'informazione sulla relazione di fase può essere ignorata, ad esempio quando si vuole identificare la presenza o meno di una certa componente sarà sufficiente controllare che l'ampiezza superi una certa soglia. In queste situazioni si possono considerare i valori in uscita nel loro valore assoluto. Sotto queste condizioni semplificative si può quindi considerare un algoritmo di calcolo della trasformata discreta di Fourier che ha come ingresso una sequenza di valori reali e come uscita un'altra sequenza di valori anch'essi reali.

1.2 Algoritmi di trasformazione veloce

Data l'utilità della trasformata di Fourier discreta, riconosciuta già prima dell'avvento dei calcolatori elettronici, molti sforzi di ricerca sono stati investiti nel tentativo di renderne più agevole il calcolo. Eseguire manualmente tutti i passaggi richiesti era infatti un lavoro pesante. In letteratura è possibile trovare vari algoritmi veloci sviluppati per calcolare la DFT.

Uno in particolare, quello di **Cooley-Tukey**, è diventato estremamente popolare per le implementazioni software in quanto può facilmente lavorare con un numero di campioni che sia una potenza di 2. È un algoritmo ricorsivo che si basa sull'espressione della DFT in due DFT di dimensione dimezzata.

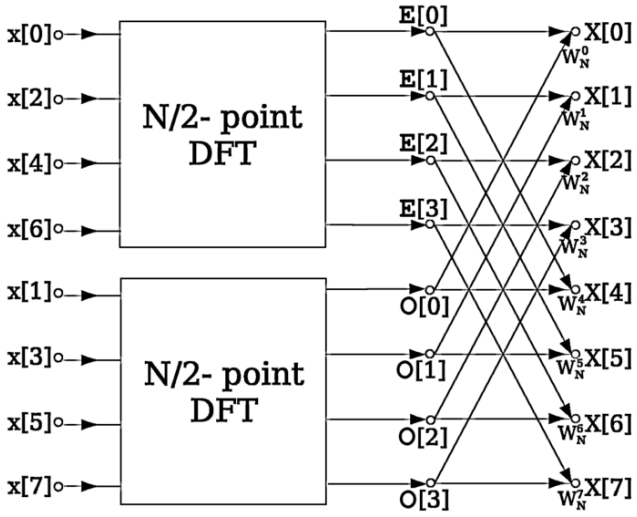


Figura 4: Schema concettuale dell'algoritmo di Cooley-Tukey. [WIKI-4]

Dal momento che il numero di campioni è pari, è possibile calcolare separatamente le trasformate discrete di Fourier della successione dei termini pari e di quella dei termini dispari ed esprimere la trasformata iniziale come combinazione di esse:

$$X_k = \sum_{m=0}^{N/2-1} x_{2m} \mathrm{e}^{\frac{-2\pi i}{N}(2m)k} + \sum_{m=0}^{N/2-1} x_{2m+1} \mathrm{e}^{\frac{-2\pi i}{N}(2m+1)k}$$

Inoltre, dato che N è una potenza di 2, anche la lunghezza delle trasformate risultanti sarà pari e quindi è possibile applicare ricorsivamente la scomposizione fino ad arrivare ai calcoli elementari.

L'applicazione di questo algoritmo consente di ridurre notevolmente la complessità di calcolo della DFT, da $O(n^2)$ a $O(n \log n)$. Alle potenze di calcolo attualmente disponibili diventa possibile l'applicazione a segnali acquisiti in tempo reale.

1.3 Trasformata di Walsh-Hadamard

La **trasformata di Walsh-Hadamard**, appartenente ad una classe generalizzata di trasformate di Fourier, è un'operazione lineare che agisce su un insieme di numeri reali la cui cardinalità sia una potenza di 2 (2^m). Può essere considerata la combinazione di più DFT di dimensione 2, ed è effettivamente equivalente ad una DFT multidimensionale con dimensione 2x2x2...x2, con tante dimensioni

quante i numeri considerati. Si utilizza per scomporre il vettore in ingresso in una sovrapposizione di *funzioni di Walsh* (allo stesso modo in cui la DFT scompone il vettore di ingresso in una sovrapposizione di funzioni sinusoidali). Queste funzioni di Walsh sono tutte ortogonali e quindi si può dire che ogni funzione discreta nel tempo abbia uno spettro di Walsh. Allo stesso modo la trasformata può essere invertita, ricostruendo dallo spettro la funzione originale.

Nella pratica, la trasformata si calcola mediante una matrice (matrice di Hadamard) definita ricorsivamente come segue:

$$H_m = \frac{1}{\sqrt{2}} \begin{pmatrix} H_{m-1} & H_{m-1} \\ H_{m-1} & -H_{m-1} \end{pmatrix}$$

dove si suppone essere H_0 la matrice identità. La costante moltiplicativa spesso, come anche nel caso dell'implementazione in analisi, può essere omesso. [LIT-3]

Ognuna delle funzioni che formano la base per la trasformata ha un numero diverso di passaggi per lo zero. Queste funzioni possono essere ordinate in modi differenti, ai quali corrisponde un diverso ordinamento delle righe della matrice di Hadamard: è evidente che per invertire con successo la funzione è necessario utilizzare lo stesso ordinamento della base in entrambi i sensi.

Anche questa trasformata ha una sua implementazione numerica veloce: la **FWHT** (*Fast Walsh-Hadamard Transform*). Concettualmente simile all'algoritmo DFT, si basa sul calcolo ricorsivo di trasformate su un numero dimezzato di elementi, fino all'ottenimento del risultato desiderato, e discende naturalmente dalla definizione ricorsiva della WHT sopra riportata.

L'algoritmo della FWHT, espresso formalmente nel codice dell'implementazione illustrato nel capitolo 3, fa uso solamente di addizioni e sottrazioni fra numeri reali e quindi è molto meno oneroso da eseguire rispetto a quello della DFT, anche utilizzando il metodo Cooley-Tukey. Inoltre dal momento che lo spettro risultante è composto da soli numeri reali anche i requisiti di memoria per rappresentarlo nel calcolatore risultano inferiori.

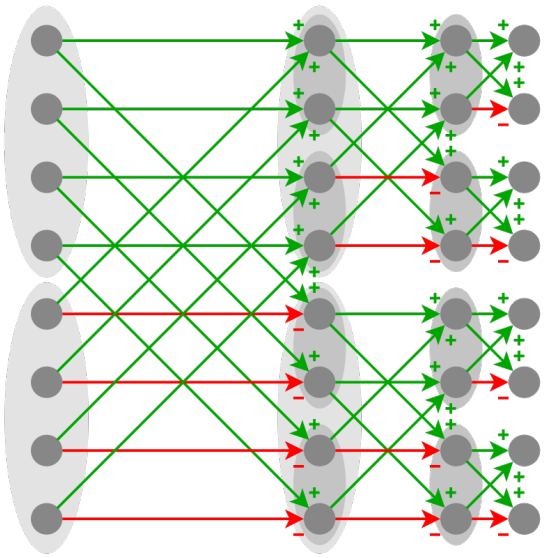


Figura 5: Schema che mostra la combinazione ricorsiva dei valori in WHT più piccole. [WIKI-5]

La proprietà di interesse di questa trasformata è che può essere utilizzata per implementare schemi di correzione degli errori. Infatti, se si altera (introduzione di errore) una quantità non troppo elevata di elementi del vettore di uscita, applicando la trasformata inversa al vettore alterato si otterrà una funzione simile a quella originale. Selezionando accuratamente un sottoinsieme di possibili combinazioni che possono essere applicate in ingresso, è possibile determinare quella con la quale il segnale ricostruito ha maggior somiglianza, e concludere così che con elevata probabilità doveva trattarsi dell'ingresso originale.

Un modo semplice di sfruttare questa proprietà è quella di limitare i possibili vettori di ingresso a quelli con un singolo elemento diverso da zero. Alterando i valori in uscita dalla trasformata, e poi applicando l'inversa al vettore alterato, si otterrà tipicamente un vettore con più di un elemento diverso da zero. Tuttavia, uno di questi elementi sarà più grande degli altri e si tratterà di quello originariamente valorizzato; i valori inferiori degli altri elementi sono interpretabili come rumore/errore.

Codificando l'informazione nell'indice del singolo elemento valorizzato si ottiene così un efficace modo di rilevare e correggere l'errore applicato ai valori in uscita.

Capitolo 2

La modulazione Olivia

In questo capitolo si mostrerà la struttura del sistema di trasmissione Olivia, identificando e descrivendo i blocchi funzionali che compongono il modulatore e il demodulatore.

2.1 Panoramica

La modulazione Olivia è una *sound card mode* sviluppata nel 2003 da Pawel Jalocha per utilizzo radioamatoriale. L'obiettivo fondamentale era quello di creare un canale di comunicazione testuale molto resistente al rumore, particolarmente su canali soggetti ad attenuazioni di intensità variabile o intermittenti (*fading*). Per ottenere questo è necessario introdurre tecniche di correzione dell'errore e quindi scendere a compromessi sulla velocità di trasmissione e sull'efficienza spettrale; di conseguenza, nelle configurazioni standard, il segnale Olivia ha una larghezza di banda relativamente ampia per una sound card mode ed una velocità di trasmissione molto più bassa rispetto a modulazioni con requisiti di banda similari.

Non esiste una specifica ufficiale definitiva, tuttavia ne è disponibile una bozza preliminare pubblicata sul sito dell'ARRL all'indirizzo web http://www.arrl.org/olivia [ARRL-1]. L'implementazione di riferimento dell'autore, pubblicata sul web dall'autore e successivamente archiviata su GitHub, disponibile nel repository all'indirizzo https://github.com/mrbostjo/Olivia_MFSK [GITH-1], si può considerare la specifica formale.

Il principio fondamentale è quello di utilizzare sequenze a bassa velocità di toni, scelti fra un numero sovrabbondante di possibilità e distribuiti in una banda ampia. In aggiunta a questo sono presenti stadi di scrambling, interleaving e codifica che aiutano statisticamente a ridurre l'incidenza degli errori.

Olivia può funzionare in diverse configurazioni, che si differenziano per il numero di toni impiegati e per la larghezza di banda occupata. Alcune di queste configurazioni consentono velocità di trasferimento leggermente

più elevate, mentre altre aumentano il livello di robustezza al rumore. Nel corso di questo documento si identificherà una configurazione Olivia con la sintassi *n/b@f* , dove *n* è il numero di toni equispaziati da utilizzare, *b* la larghezza della banda all'interno della quale essi sono equamente distribuiti, e *f* la frequenza centrale in banda base dello spettro del segnale.

La configurazione che meglio dimostra un valido bilanciamento fra prestazioni e resistenza è quella con 32 toni distribuiti su una banda di 1 KHz. È consuetudine centrare lo spettro del segnale modulato nella banda base disponibile; essendo che la tale banda, nelle sound card modes, è tipicamente di 3 KHz, si ottiene una frequenza centrale per lo spettro modulato di 1500 Hz. La configurazione risultante è quindi 32/1000@1500, ed è questa che si utilizzerà per le prove sperimentali incluse in questo documento. L'implementazione presentata è comunque in grado di operare anche su configurazioni differenti.

La velocità effettiva di trasmissione (symbol rate) dipende dalla configurazione secondo la formula:

$$f_s = \frac{b}{n}$$

Si ha quindi che il symbol rate coincide con la spaziatura in frequenza fra un tono utilizzato e il successivo. Allargando la banda a parità di numero di toni utilizzati si ha un aumento di symbol rate; parimenti diminuendo il numero di toni utilizzati si ha un aumento di symbol rate a parità di banda, conservando così un certo grado di resistenza al rumore.

I vari stadi della modulazione, di seguito dettagliati, formano un codice di correzione dell'errore teoricamente bidimensionale. Volontà dell'autore era di realizzare un algoritmo iterativo per sfruttare questa bidimensionalità al fine di ottenere una demodulazione più accurata ma questo risultato non è ancora stato raggiunto.

Olivia è concepita per la trasmissione di testo ASCII, ovvero caratteri di 7 bit. Se la si vuole utilizzare per trasferire flussi di byte, come files o stringhe di testo UTF-8, occorre eseguire una preliminare trasformazione del flusso di byte in un flusso di caratteri ASCII. Un modo standard per

ottenere questo risultato è la codifica Base64, RFC 4648 (https://tools.ietf.org/html/rfc4648) [RFC-4648].

Principalmente utilizzata per il trasferimento di file via e-mail, Base64 produce in uscita un flusso composto esclusivamente di numeri e lettere maiuscole e minuscole più due caratteri speciali, ovvero un sottoinsieme dei caratteri ASCII a 7 bit. La codifica causa un'espansione del contenuto, infatti 24 bit di input vengono mappati su 4 caratteri a 7 bit, per un totale di 28 bit. Si introduce quindi un overhead di circa il 17%, a fronte della possibilità di modulare bytes arbitrari.

Dopo la demodulazione, si applicherà la decodifica Base64 al flusso di caratteri ASCII ottenuto, ricostruendo così il flusso di bytes originale.

2.2 Stadi di modulazione

Lo schema a blocchi mostra gli stadi concettuali attraverso i quali passa l'informazione durante la modulazione, dalla sorgente digitale fino a divenire un segnale audio analogico.

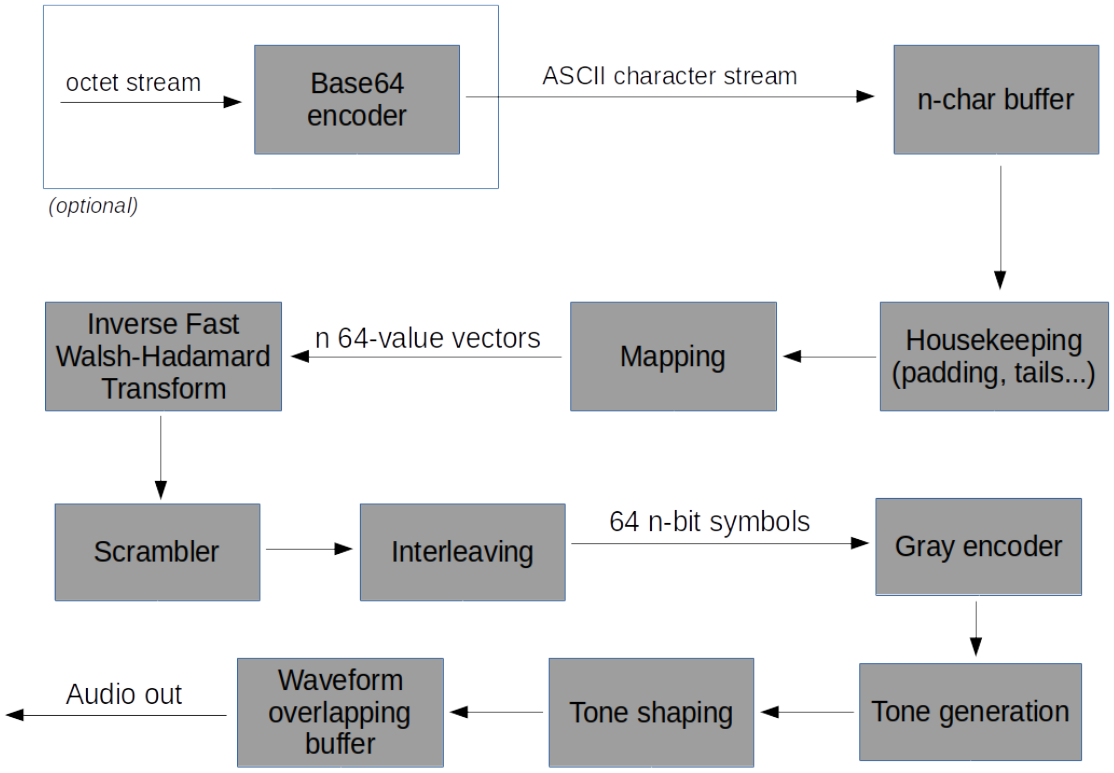


Figura 6: Schema a blocchi del modulatore Olivia.

Sono di seguito illustrati e approfonditi i singoli stadi della modulazione.

2.2.1 Buffering e housekeeping

I caratteri vengono trasmessi in blocchi di $\log_2(n)$, dove n è il numero di toni disponibili nella configurazione scelta. Ad esempio, usando una configurazione a 32 toni, i caratteri verranno trasmessi in gruppi di 5. Trattandosi di caratteri a 7 bit, si avrà un contenuto informativo pari a 7n bit per blocco; nell'esempio con n=32, si avranno 35 bit di informazione utile trasmessa per blocco.

È necessario quindi prevedere uno stadio per accumulare il flusso di caratteri in ingresso e consegnare allo stadio successivo solo blocchi

completi di $\log_2(n)$ caratteri, eventualmente completati con padding di caratteri ASCII 0 (NUL) nel caso che la trasmissione termini con un numero di caratteri non multiplo della lunghezza del blocco.

Per semplificare la sintonizzazione manuale dei ricevitori in radiofrequenza, si trasmettono anche due “code”; una prima dell’inizio della trasmissione dei blocchi, detta preambolo; e una finale, successivamente all’ultimo blocco. Le code sono composte da due ripetizioni del tono più basso e di quello più alto nella configurazione scelta alternati, e non contengono dati.

2.2.2 Mappatura da caratteri a vettori

La funzione di trasformazione utilizzata nello stadio successivo, per ragioni successivamente illustrate, richiede in ingresso vettori di 64 valori, tutti nulli tranne uno. Si adotta una semplice codifica per convertire i caratteri ASCII di 7 bit in vettori di questo tipo:

$$w_i = \begin{cases} 1 & q < 64, i = q \\ -1 & q \geq 64, i = q - 64 \quad \forall\ i \in [0;63] \\ 0 & \textit{altrimenti} \end{cases}$$

dove *w* indica il vettore che si sta costruendo, *q* il carattere al quale corrisponde. Notare che essendo *q* un carattere ASCII a 7 bit, il suo valore sarà sempre compreso fra 0 a 127; per ognuno di questi valori di *q*, uno e un solo elemento di *w* sarà diverso da zero.

In uscita da questo stadio si otterranno quindi *n* vettori di 64 valori, ciascuno dei quali corrisponde a uno degli *n* caratteri del blocco in uscita dal buffer precedente.

2.2.3 Inverse Fast Walsh-Hadamard Transform

Si applica ai singoli vettori ottenuti la trasformata inversa di Walsh-Hadamard. Questo introduce una prima dimensione di controllo dell’errore.

La ragione per la quale si utilizza la trasformata *inversa* in fase di trasmissione, è che il vettore generato allo stato precedente, avendo un singolo elemento valorizzato, può essere considerato come il più semplice spettro di Walsh di una funzione: una singola riga dotata di una

certa ampiezza unitaria. Invertendo questo spettro si otterrà quindi una funzione di base di Walsh.

Ad ogni vettore in ingresso corrisponde quindi uno specifico vettore trasformato in uscita. Come si vedrà successivamente in fase di demodulazione, questo passaggio introduce un livello di ridondanza per il quale sarà possibile risalire al vettore in ingresso originale anche in presenza di alterazioni su un numero sufficientemente basso di valori del vettore generato.

2.2.4 Scrambler

Una possibile tipologia di rumore sul canale è quello a banda stretta, ad esempio un tono sinusoidale a frequenza fissa, sempre presente sul canale. La presenza di questo tipo di disturbo può rendere consistentemente difficile la discriminazione fra toni adiacenti nelle sue immediate vicinanze, introducendo una maggior probabilità di errori ogniqualvolta questi toni vengono utilizzati. Per contrastare questa eventualità è opportuno fare sì che statisticamente i toni vengano utilizzati con probabilità uniforme.

Si introduce quindi uno stadio che combina i valori in uscita dalla trasformata WH con un vettore pseudocasuale. Questo vettore è stato scelto e standardizzato una volta per tutte, ed è:

$$k = (E\,257\mathrm{E}6\,D\,0291574\,EC)_{16}$$

Questa chiave viene applicata direttamente in XOR bit per bit al primo vettore di un blocco; per i vettori successivi nello stesso blocco, viene ruotata ogni volta di 13 bit verso destra prima dell’applicazione, in modo da generare sequenze differenti anche in presenza di caratteri uguali nello stesso blocco.

I valori in uscita da questo stadio avranno una distribuzione quasi uniforme. Essendo però utilizzato un valore chiave pseudocasuale noto e stabilito a priori, sarà possibile in fase di ricezione invertire deterministicamente questo passaggio.

2.2.5 Interleaving

Un’altra tipologia di rumore che si può incontrare è quello ad ampio spettro ma limitato nel tempo. Una tipica sorgente di questo tipo di rumore è la scarica elettrostatica, frequentemente spontanea in atmosfera o generata da attività umane. Questa situazione può introdurre un errore di decisione sulla decodifica di uno o pochi simboli intorno ad un determinato istante di tempo. Si vuole evitare che l’errore di decisione su un simbolo causi la decodifica errata di caratteri.

Un simbolo contiene $\log_2(n)$ bit di informazione, numero per costruzione coincidente con la lunghezza del blocco in caratteri. Si introduce uno schema di rimaneggiamento dei vettori per fare sì che ogni bit di un simbolo trasporti informazione relativa ad un diverso vettore, così che ogni carattere del blocco abbia un contributo nella determinazione di ogni simbolo da emettere. Di conseguenza in fase di ricezione l’eventuale errore su un simbolo sarà distribuito su tutti i vettori che compongono un blocco, prestandosi così molto bene ad essere corretto dalla trasformata di Walsh-Hadamard.

Lo schema di interleaving arbitrariamente deciso e standardizzato per Olivia è il seguente:

- per ogni vettore W costituente il blocco e definito *w* il suo indice,
  - per ogni simbolo S costituente un blocco e definito *s* il suo indice,
    - sia $q = (w - s) \ mod \ \log_2(n)$,
    - il bit *w*-esimo del simbolo *S* sia eguale al al bit *s*-esimo del *q*-esimo vettore.

2.2.6 Codifica Gray

Le frequenze dei toni audio che verranno generati dallo stadio successivo, corrispondenti ai vari simboli possibili, hanno distanze sullo spettro molto ridotte. Di conseguenza, l’errore più probabile che si possa commettere in fase di ricezione è quello di confondere un tono con quello precedente o successivo. Per limitare l’errore commesso in questo scenario, si utilizza sui simboli da inviare la codifica Gray.

La codifica Gray, inventata nel 1953 da Frank Gray, fu introdotta per rappresentare i valori digitali in uscita da dispositivi reali (come serie di interruttori o encoder rotativi). In questi dispositivi, interruttori o microswitches vengono azionati meccanicamente da una struttura, di solito rotante. Utilizzando la normale codifica binaria dei numeri interi si incontra un problema nella transizione fra due stati adiacenti le cui rappresentazioni binarie differiscono per più di un bit: infatti fisicamente non è possibile realizzare interruttori che cambino stato nel medesimo esatto istante. Si avranno quindi fugaci transizioni intermedia indesiderate ad uno o più stati non rilevanti (*alea*).

La codifica Gray associa ai numeri interi stringhe binarie costruite in modo da minimizzare il numero di bit che variano durante una transizione fra due stati adiacenti. Una codifica Gray a *n*-bit si può costruire con un algoritmo ricorsivo:

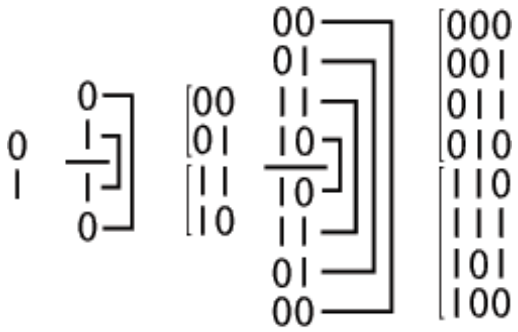


Figura 7: Costruzione di una codifica Gray. [WIKI-6]

1. Scrivere i numeri 0 e 1 incolonnati in verticale.
2. Riflettere le stringhe verso il basso lungo una linea immaginaria sotto all'ultima.
3. Aggiungere uno 0 in testa alle stringhe sopra la linea, e un 1 in testa a quelle sotto.
4. Ripetere dal punto 2 fino ad ottenere una codifica della lunghezza desiderata.

Come si può notare, la distanza di modifica fra una stringa e una adiacente ad essa è costante e sempre uguale ad 1 bit.

In fase di demodulazione, l'errore che si commette scambiando un simbolo per uno ad esso adiacente è variabile da 1 a *n* bit a seconda dei casi. Supponiamo uno schema a 32 toni, ovvero con simboli a 5 bit. L'errore commesso scambiando il tono 16 (10000) con il tono 17 (10001) sarebbe di 1 bit, mentre quello commesso scambiando il tono 15 (01111) con il tono 16 (10000) sarebbe di ben 5 bit.

Applicando la codifica Gray agli indici dei simboli ottenuti all'uscita del precedente stadio, si minimizza l'errore commesso scambiando un simbolo con uno adiacente. Infatti, il numero di bit che differiscono fra la codifica Gray di un simbolo e quella del precedente (o successivo) è, come conseguenza di quanto visto sopra, costante e uguale al minimo possibile di 1.

Un algoritmo semplice per implementare la codifica Gray su un calcolatore, utilizzato anche in questa implementazione, è effettuare lo XOR bit-per-bit del valore con il valore shiftato verso destra di una posizione.

2.2.7 Generazione dei toni

I toni rappresentati dai simboli ottenuti vengono sintetizzati direttamente, calcolando i campioni da emettere per ogni istante di tempo. Noti i parametri della modulazione (numero di toni, frequenza centrale e larghezza di banda) è possibile calcolare i valori dei campioni come segue:

$$f_t = (f_c - \frac{B}{2}) + \frac{f_{sep}}{2} + f_{sep} N$$
$$f(k) = A \sin(2 \pi f_t k + \phi_0)$$

dove:

- k è l'istante di tempo corrispondente al campione;
- f_c la frequenza centrale della modulazione;
- B la larghezza di banda della modulazione;
- f_{sep} la separazione fra un tono e il successivo (ovvero B/n);
- N il numero del tono da produrre;

- A un fattore moltiplicativo utilizzato per regolare la potenza del segnale in uscita;
- ϕ_0 lo scarto di fase.

Lo scarto di fase è scelto casualmente e consiste in un ritardo oppure anticipo di $\pi/2$. È introdotto per evitare la generazioni di sinusoidi continue nel caso (improbabile) di produzione continua dello stesso tono, limitando così i problemi di sincronizzazione del ricevitore.

I campioni del tono prodotto vengono modellati tramite una funzione di filtro prestabilita e condivisa. La funzione filtro è la seguente:

$$1+1.1913785723\cos(f(k))-0.0793018558\cos(2f(k))$$
$$-0.2171442026\cos(3f(k))-0.0014526076\cos(4f(k))$$

I coefficienti di questa funzione sono stati determinati sperimentalmente per limitare il più possibile la modulazione intersimbolo e fanno parte della specifica.

2.2.8 Trasmissione in frequenza audio

I toni così modellati sono formati da una prima fase nella quale la loro ampiezza armonicamente aumenta, ed una seconda nella quale diminuisce. Si predispone un buffer audio di uscita per combinare le forme d'onda in modo che la seconda metà di tono uno si sovrapponga alla prima metà del successivo. Così facendo il segnale in uscita è privo di brusche variazioni e quindi ha un contenuto armonico basso che non sporca il segnale in uscita dal trasmettitore e non interferisce con eventuali altri segnali adiacenti.

In ogni finestra di campioni è comunque sempre possibile in fase di ricezione determinare la prevalenza di un tono sull'altro, misurandone l'ampiezza e considerando quello più forte. Infatti la fase di ampiezza discendente di un tono si sovrappone a quella di ampiezza ascendente del tono successivo, ottenendo una transizione graduale dalla predominanza di un tono a quella dell'altro.

2.3 Stadi di demodulazione

Lo schema a blocchi mostra gli stadi concettuali attraverso i quali passa l'informazione durante la demodulazione, dal segnale audio analogico in ingresso fino al flusso digitale ricostruito in uscita.

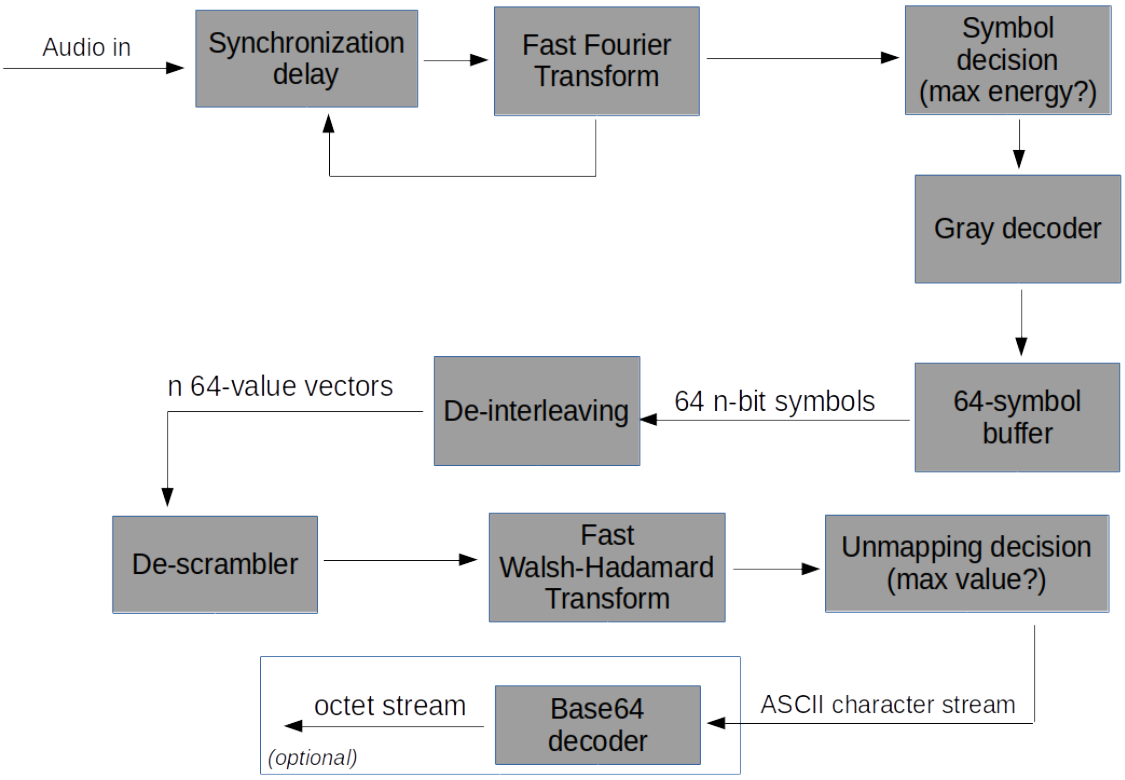


Figura 8: Schema a blocchi del demodulatore Olivia.

Sono di seguito illustrati e approfonditi i singoli stadi della demodulazione.

2.3.1 Fast Fourier Transform

La modulazione Olivia è progettata per sfruttare al meglio questa trasformata veloce. La frequenza di campionamento tipicamente usata per Olivia (8000 Hz) è scelta affinché le frequenze dei vari toni utilizzati si trovino, in configurazioni standard, esattamente a distanza 1 sullo spettro discreto. Questo può semplificare l'implementazione dei demodulatori.

Vengono misurate le ampiezze di tutte le frequenze corrispondenti ai possibili toni e si determina quale con maggior probabilità sia il tono

corretto, scartando le possibilità che più verosimilmente sono determinate dal rumore.

Per sincronizzare la finestra della trasformata con il segnale prodotto dal ricevitore, ovvero far sì che la modulazione di un simbolo sia il più possibile contenuta in un buffer che viene trasformato, si può ad esempio adottare un approccio a misura di energia: si lascia scorrere la finestra, scartando i campioni in ingresso fino a che l'energia del segnale nella finestra considerata non supera una certa soglia. In questo modo si evita di andare a trasformare una finestra contenente silenzio nella prima parte e solo una piccola quantità di segnale nella seconda, scenario che darebbe sicuramente un errore maggiore essendo che il rapporto fra l'energia del segnale effettivo e quella del rumore sarebbe più basso.

2.3.2 Decodifica Gray

La codifica Gray applicata ai simboli in uscita dal modulatore ha il ruolo di proteggere l'informazione da un eccessivo errore durante la fase di rilevamento della frequenza. Giunti a questo punto è possibile invertirla per ottenere i valori corrispondenti ai simboli trasmessi, a meno degli errori precedentemente introdotti in fase di decisione.

Un algoritmo semplice per implementare la decodifica Gray, utilizzato anche in questa implementazione, è quello di considerare tutti i valori derivanti dai possibili shift verso destra del valore in ingresso e applicare lo XOR bit-a-bit fra tutti.

2.3.3 Buffering

Prima di poter procedere alla decodifica anche di un solo carattere, è necessario ricevere una quantità di simboli corrispondente ad un intero blocco trasmesso. Una quantità di simboli inferiore non contiene sufficienti informazioni per la decodifica di alcun carattere. Il blocco trasmesso, qualunque sia *n,* è formato da 64 simboli, ciascuno dei quali porterà una quantità di informazione variabile in funzione di *n*. È quindi necessario predisporre un buffer temporaneo che accumuli 64 simboli prima di passarli allo stadio successivo.

Nell'implementazione considerata questo blocco non è un semplice accumulatore di valori, ma è implementato mediante un array rotante e

questo è particolarmente importante per ottenere una correzione dell'errore affidabile con una bassa complessità del codice, come sarà meglio illustrato di seguito nella sezione dedicata.

2.3.4 De-interleaving

Si esegue l'operazione inversa dell'interleaving effettuato in modulazione, in modo da ottenere *n* vettori da 64-bit.

A causa dell'incertezza sulla decisione dei simboli ricevuti, questi vettori non saranno tipicamente identici a quelli che erano in ingresso allo stadio di interleaving del modulatore, soprattutto in condizioni di rumore.

Lo schema del de-interleaving, deciso arbitrariamente e standardizzato per Olivia e complementare a quello di interleaving precedentemente illustrato, è il seguente:

- per ogni vettore W costituente il blocco e definito *w* il suo indice,
  - per ogni simbolo S costituente un blocco e definito *s* il suo indice,
    - sia $q = (w + s)\ mod\ \log_2(n)$,
    - il bit *s*-esimo del vettore *W* sia eguale al *q*-esimo bit del simbolo S.

A questo punto ci si è ricondotti alla situazione nella quale ogni vettore rappresenta un singolo carattere del blocco ricevuto.

2.3.5 Descrambler

Questo stadio è identico allo scrambler presente nel modulatore. Infatti applicando nuovamente lo XOR bit per bit della stessa chiave condivisa ai vettori, ruotata ogni volta di 13 bit per ogni vettore successivo nel blocco, si annulla esattamente l'operazione. Questo stadio non introduce errore in quanto i valori in uscita sono esattamente determinati da quelli in ingresso.

2.3.6 Fast Walsh-Hadamard Transform

Si applica al vettore ottenuto la trasformata di Walsh-Hadamard. A questo punto il vettore è una versione alterata di una funzione base di Walsh. Idealmente, in assenza di errore, per come è stato costruito il

segnale modulato il vettore dovrebbe essere esattamente una funzione base e di conseguenza la sua trasformata dovrebbe generare uno spettro contenente una singola riga di ampiezza unitaria (a meno di costante moltiplicativa di normalizzazione che dipende dalla lunghezza del vettore).

Il vettore ottenuto dall'applicazione della trasformata consisterà in uno spettro con una riga di ampiezza predominante rispetto alle altre. La differenza fra l'ampiezza della riga predominante e la media delle ampiezze delle altre righe dipenderà dalla quantità di errore introdotto dal canale e dagli stadi precedenti del demodulatore. La correzione dell'errore si concretizza quindi nel semplice procedimento di porre l'elemento del vettore contenente il valore maggiore identicamente a 1 (se positivo) o -1 (se negativo); e porre tutti gli altri valori identicamente a zero.

2.3.7 Mappatura da vettori a caratteri

A questo punto si dispone di $\log_2(n)$ vettori lunghi 64 elementi, di cui uno solo valorizzato con uno di due possibili valori. Si inverte l'operazione di mappatura eseguita in fase di modulazione, per determinare a quale dei 128 possibili caratteri ASCII corrispondano i vettori. I caratteri così ricostruiti formano il testo ricevuto e la demodulazione è completata.

Capitolo 3

Dettagli implementativi

In questo capitolo sono evidenziate le soluzioni tecniche scelte per l'implementazione realizzata, riportando e commentando i frammenti di codice più significativi.

La scelta del linguaggio da utilizzare per l'implementazione è ricaduta su Python. Questo linguaggio ha una sintassi semplice e di facile apprendimento, ed è sempre più ampiamente utilizzato per applicazioni scientifiche a causa della vastissima scelta di librerie di terze parti disponibili, oltre alla compatibilità su un amplissimo ventaglio di diverse piattaforme.

Il punto di forza di Python è l'ecosistema: in poco tempo si sono reperite librerie già pronte e di grande diffusione che hanno consentito di concentrare lo sforzo di sviluppo sulle problematiche del tema in esame, interfacciandosi con l'hardware in modo pratico ed evitando di dover reimplementare metodi e algoritmi già conosciuti.

1. `numpy`. Si tratta di una libreria che rende Python adatto al calcolo numerico, applicazione per la quale non era originariamente stato ideato. Comprende un gran numero di funzioni scientifiche e di calcolo, molte con un'interfaccia ispirata a MATLAB, del quale raggiunge una funzionalità di base equivalente anche se con sintassi molto diversa; ma soprattutto introduce come tipo di dati l'array multidimensionale utilizzabile in modo generico nelle funzioni, aprendo possibilità di calcolo vettoriale e rappresentazione degli algoritmi in modo molto più compatto. L'introduzione di questa libreria ha consentito ai ricercatori di avvicinarsi a Python come ad un ecosistema plausibile per le implementazioni e la modellazione rapida dei problemi, anche grazie all'introduzioni di ambienti integrati come Spyder, portando il linguaggio fuori dalla sua tradizionale applicazione di scripting di sistema e realizzazione di programmi applicativi.

2. `scipy`. Questa libreria comprende, fra il resto, implementazioni di molti algoritmi utili per il calcolo numerico, per esempio funzioni per il calcolo delle trasformate di Fourier e di Walsh-Hadamard dirette ed inverse e moltissimi altri strumenti per l'elaborazione digitale dei segnali, l'algebra lineare, interpolazione e machine learning.
3. `sounddevice`. Questa libreria consente di utilizzare i dispositivi audio di ingresso e uscita del calcolatore tramite un'interfaccia a buffer di campioni, astraendo completamente dalle complessità dei driver sottostanti, complicati da usare e dipendenti dalla piattaforma. Una volta creato un oggetto di ingresso o uscita la libreria si occupa in background di emettere tramite il dispositivi di output selezionato i buffer di campioni che vengono consegnati tramite l'API, così come di fornire al programma applicativo un flusso costante di buffer di ingresso ottenuti dal dispositivo di input selezionato. L'emissione e l'acquisizione di suoni si riduce quindi ad una semplice preparazione o analisi di array di valori che rappresentano i campioni nel tempo del segnale analogico corrispondente, già quantizzati e convertiti in un tipo di dati facilmente manipolabile.

3.1 Interfaccia a linea di comando

Per la realizzazione pratica del software che implementa la modulazione si è scelta un'interfaccia a linea di comando per i seguenti motivi:

1. praticità. L'interfaccia testuale è più flessibile, efficace e di rapida realizzazione.
2. compatibilità. L'assenza di elementi grafici consente di ridurre al minimo la quantità di librerie di terze parti sulle quali si fa affidamento, inoltre è più universalmente compatibile fra vari sistemi.
3. *scriptability*. Si presta ad essere utilizzata in combinazione, o integrata, con altri programmi o in maniera automatica. I programmi ottenuti possono essere visti come filtri Unix e inseriti in script con poco sforzo.

Il programma viene invocato con un parametro, opzionale, che indica la configurazione desiderata per la modulazione:

n/b@f

Il valore di default in caso di assenza del parametro sarà: *32/1000@1500,* modalità ampiamente utilizzata e che rappresenta un compromesso fra velocità di trasmissione e robustezza al rumore.

Se *standard input* è un terminale, viene visualizzato un messaggio di benvenuto (o di aiuto, se i parametri inseriti non sono corretti) e per la trasmissione, viene visualizzato il prompt `olivia>` .

Nel caso che non si stia invocando il programma da un terminale ma da uno script o mediante pipes/redirezione di file, non viene stampato alcun messaggio o prompt, per rendere più semplice l'integrazione.

3.2 Trasmissione asincrona mediante coda di invio

Si è scelto di implementare per il trasmettitore una logica asincrona: il modulatore funziona continuamente in background e non viene bloccato in attesa di input da tastiera. Quando non ci sono dati da trasmettere, il modulatore trasmette continuamente sequenze vuote di padding. Un thread separato in parallelo legge l'input da tastiera e, dividendolo in blocchi, lo inserisce in una coda di trasmissione. Il thread di trasmissione preleva i dati da questa coda e si occupa di modularli. Questo realizza la logica di buffering descritta nel paragrafo 2.2.1.

La funzione `initSound` del trasmettitore prepara la libreria affinché utilizzi una callback su un thread dedicato per ottenere i campioni quando ne ha necessità:

```
def initSound():
    '''
    Prepares global OutputStream for sample playback.
    '''
    global sout

    sout = sd.OutputStream(samplerate=SAMPLE_RATE,
blocksize=64*wlen,
        channels=1, callback=callback, dtype=np.float32)
```

```
    sout.start()
```

La callback si occupa di verificare la presenza di eventuali blocchi da trasmettere sulla coda. Nel caso che la coda di trasmissione sia vuota, predispone l'invio di un blocco nullo, per evitare che la generazione del suono si interrompa a causa della mancanza di un flusso costante di campioni; altrimenti viene selezionato il successivo blocco nella coda.

```
def callback(outdata, frames, time, status):
    '''
    Handles sample generation and playback asynchronously.
    '''

    # If first call, generate the preamble.
    if callback.firstCall and ENABLE_PREAMBLE:
        callback.firstCall = False
        outdata[:,0] = generatePreamble()/ATTENUATION
        return None

    # If last call, just empty the samples buffer.
    if callback.lastCall:
        outdata.fill(0)
        return None

    # If transmission queue is empty, transmit null blocks.
    try:
        piece = q.get_nowait()
    except:
        piece = "\0" * spb

    outdata[:,0] = generateBlock(piece)/ATTENUATION

    # If transmission is over, transmit ending tail.
    if piece == None:
        callback.lastCall = True

    return None
```

Il vantaggio di questa tecnica è che l'intero ciclo di vita del programma modulatore dà luogo ad una sola trasmissione continua, sulla quale

possono passare o meno dati effettivi a seconda della disponibilità, minimizzando l'informazione persa in fase di sincronizzazione del ricevitore. Infatti una volta che il ricevitore si è sincronizzato sull'inizio del segnale, può continuare a seguirlo per tutto il tempo che il trasmettitore rimane in funzione senza doversi riallineare salvo che in caso di eccessivo errore. Questo aumenta l'affidabilità del canale e riduce la latenza di trasmissione dei blocchi di caratteri, dal momento che il modulatore è già in funzione quando i dati diventano disponibili per la trasmissione.

Per ottenere i campioni partendo da un blocco di caratteri, la callback fa uso della funzione `generateBlock`, che provvede alla parziale sovrapposizione delle forme d'onda ottenute dal codice di generazione dei toni (si veda il paragrafo 3.3) ed eventualmente all'inserimento di preamboli e code:

```
def generateBlock(piece):
    '''
    Transmits samples corresponding to a full block.
    '''
    global trail

    wf = np.zeros(64*wlen+wlen)

    # Overlaps trail of last symbol, if any
    wf[0:wlen] += trail

    # If transmission is being stopped, add trailing tail
    if piece == None:
        if not ENABLE_PREAMBLE:
            return wf[0:64*wlen]
        trail = np.zeros(wlen)
        tail = generateTail()
        if len(tail) < 64*wlen:
            wf[wlen:wlen+len(tail)] = tail
        return wf[0:64*wlen]

    syms = prepareSymbols(piece)

    for i in range(0, 64):
```

```
        # Tone number is symbol number after Gray encoding
        # This minimized error made by mistaking one tone for
        # another one right next to it (1 wrong bit only).
        tone = oliviaTone(gray(syms[i]))
        wf[wlen*i:wlen*i+len(tone)] += tone

    trail = wf[64*wlen:64*wlen+wlen]
    return wf[0:64*wlen]
```

Il cuore dell'implementazione del modulatore è la funzione `prepareSymbols`. È qui che vengono implementati gli stadi che trasformano il flusso di dati in ingresso in un flusso di simboli.

```
def prepareSymbols(chars):
    '''
    Transform a block of characters into a block of symbols
    ready for transmission.
    '''
    w = np.zeros((spb, 64))
```

La prima parte della funzione prepara la chiave che sarà utilizzata per lo *scrambling*, esprimendo la chiave esadecimale standardizzata come array Numpy di bit, allo scopo di semplificare le successive manipolazioni:

```
      # Key is a 64-bit fixed value and can be found in
specification.
    #   key = 0xE257E6D0291574EC
    # It is a pseudorandom value and its role is to make the
    # output stream appear random.
    # Here it is decomposed in a bit array for easier use.
    key = np.flip(np.array(
          [1, 1, 1, 0, 0, 0, 1, 0,
           0, 1, 0, 1, 0, 1, 1, 1,
           1, 1, 1, 0, 0, 1, 1, 0,
           1, 1, 0, 1, 0, 0, 0, 0,
           0, 0, 1, 0, 1, 0, 0, 1,
           0, 0, 0, 1, 0, 1, 0, 1,
           0, 1, 1, 1, 0, 1, 0, 0,
           1, 1, 1, 0, 1, 1, 0, 0]))
```

Successivamente è realizzato il mapping da caratteri a vettori illustrato nel paragrafo 2.2.2.

```
        # Character to 64-value vector mapping to provide
redundancy.
    # Characters are 7-bit (value from 0 to 127)
    # Values from 0 to 63 are mapped by setting nth value to 1
    # Values from 64 to 127 are mapped by setting nth value to
-1
    # Every other value in vector is 0.
    for i in range(0, spb):
        q = ord(chars[i])
        if q > 127:
            q = 0

        if q < 64:
            w[i, q] = 1
        else:
            w[i, q-64] = -1
```

Si applica la trasformata inversa di Walsh-Hadamard per introdurre la ridondanza (paragrafo 2.2.3):

```
        # Inverse Walsh-Hadamard Transform to encode redundancy
        w[i,:] = ifwht(w[i,:])
```

A questo punto si esegue lo scrambling (paragrafo 2.2.4) dei vettori ottenuti, usando la chiave preparata in precedenza, ruotata di 13 bit per ogni vettore successivo.

```
        # XOR with key to ensure randomness
        # (XOR is made by multiplying with -1 or 1)
        w[i,:] = w[i,:] * (-2*np.roll(key, -13*i)+1)
```

Si esegue l’interleaving (paragrafo 2.2.5) dei bit risultanti per ottenere i bit che rappresentano i simboli da emettere:

```
    syms = np.zeros((64, spb))

    # Bit interleaving to spread errors over symbols
```

```
    for bis in range(0, spb):
        for sym in range(0, 64):
            q = 100*spb + bis - sym
            if w[q % spb,sym] < 0:
                syms[sym,bis] = 1

    # Convert to integer to find symbol numbers
    symn = np.zeros(64)
    for i in range(0,64):
        symn[i] = bits2int(np.flip(syms[i]))

    return symn
```

Sarà la funzione `generateBlock` sopra riportata, prima della trasmissione, ad applicare la codifica Gray (paragrafo 2.2.6) mediante la funzione di utilità:

```
def gray(n):
    '''
     Utility function to calculate Gray encoding of an integer
value
    '''
    n = int(n)
    return n ^ (n >> 1)
```

prima di passare i simboli al codice di generazione.

3.3 Generazione dei toni

Per generare le finestre di campioni corrispondenti alla trasmissione dei toni si ricorre alla sintesi diretta:

```
def oliviaTone(toneNumber):
    '''
    Tone generator. Creates output waveform for specified tone
number.
    '''
    toneFreq = (CENTER_FREQUENCY - BANDWIDTH/2) + fsep/2 + fsep
* toneNumber
    t = np.arange(0, 2/fsep, 1/SAMPLE_RATE)
    ph = np.random.choice([-pi/2, pi/2]);
    ret = np.sin(2*pi*toneFreq*t + ph)
    return toneShaper(ret)
```

La funzione `oliviaTone` produce i campioni dell'onda sinusoidale che rappresenta il tono richiesto nella configurazione corrente, introducendo lo scarto aleatorio di fase previsto. Il tono in uscita viene modellato dalla funzione `toneShaper` per ridurre gli effetti della modulazione intersimbolo, come visto nel paragrafo 2.2.7:

```
def toneShaper(toneData):
    '''
    Tone shaping to avoid intersymbol modulation.
       Cosine coefficients are fixed and can be found in
specification.
    '''
    x = np.linspace(-pi, pi, len(toneData));
    shape = (1. + 1.1913785723*np.cos(x)
             - 0.0793018558*np.cos(2*x)
             - 0.2171442026*np.cos(3*x)
             - 0.0014526076*np.cos(4*x))
    return toneData * shape
```

Una trasmissione Olivia tipicamente include un preambolo e una coda, costituiti dall'alternanza del tono più alto e di quello più basso nella configurazione corrente, alternati, per una durata totale di un secondo. Queste sequenze di toni sono inserite per aiutare un operatore umano a sintonizzare più esattamente il ricevitore, nonché a rendersi conto del fatto che è in uso proprio la modulazione Olivia ed eventualmente stabilirne la larghezza di banda per scegliere il demodulatore appropriato.

Trattandosi di un aiuto all'operatore, queste sequenze non sono essenziali alla trasmissione dei dati e infatti non contengono alcuna informazione. Nello script del modulatore implementato vengono dunque generate nel caso sia possibile farlo all'interno di una singolo buffer di riproduzione del dispositivo audio, ovvero se la durata totale in campioni del preambolo o della coda è inferiore al sample rate. In caso contrario il

preambolo o coda viene omesso senza inficiare la funzionalità trasmissiva, per evitare di complicare eccessivamente il codice.

La funzione `generateTail` costruisce i toni di preamboli e code preparando un buffer di campioni lungo un'intero secondo, che contiene le onde sinusoidali pure, alternativamente alle frequenze massime e minime.

```
def generateTail():
    '''
    A tail is made in this way:
        first tone, last tone, first tone, last tone
    each one lasting 1/4 seconds.
    '''
    pl = int(SAMPLE_RATE/4)
    t = np.arange(0, 1/4, 1/SAMPLE_RATE)
    wf = np.zeros(SAMPLE_RATE)
    wf[0:pl] = toneShaper(np.sin(2*pi*(CENTER_FREQUENCY-
BANDWIDTH/2+fsep/2)*t)/2)
    wf[pl:2*pl] =
toneShaper(np.sin(2*pi*(CENTER_FREQUENCY+BANDWIDTH/2-fsep/2)*t)
/2)
    wf[2*pl:3*pl] = wf[0:pl]
    wf[3*pl:4*pl] = wf[pl:2*pl]
    return wf
```

Per quanto riguarda il ricevitore non è stato necessario implementare logiche asincrone, in quanto il flusso di campioni audio in ingresso dal driver è costante e non ci sono richieste di dati da tastiera che possano essere bloccanti per un tempo indefinito, per cui si è utilizzata la libreria sounddevice in modalità sincrona.

3.4 Rilevamento degli errori

In sé la modulazione Olivia fornisce due possibili dimensioni di controllo dell'errore, la prima sul riconoscimento dei simboli modulati tramite la trasformata di Fourier e la seconda sulla determinazione dell'elemento del vettore da valorizzare dopo la trasformata di Walsh-Hadamard. Per l'implementazione in analisi si è scelto di evitare la complessità di codice e computazionale dovuta alla sincronizzazione del buffer per la

trasformata di Fourier e si è verificato che questo lascia comunque spazio ad una correzione dell'errore efficace.

La trasformata di Fourier (paragrafo 2.3.1) viene eseguita dalla funzione `detectSymbol`, direttamente sulle finestre di campioni acquisite da sounddevice:

```
def detectSymbol():
    '''
    Applies Fourier transform to audio buffer to detect
    symbol corresponding to sampled tone.

    Returns
    -------
    int
        Most likely symbol number.
    '''
    spectrum = np.abs(fft(buf))
    ix = CENTER_FREQUENCY - BANDWIDTH/2 + fsep/2
    measures = np.zeros(SYMBOLS)

    for i in range(0, SYMBOLS):
        ix += fsep
        measures[i] = spectrum[int(ix * wlen / SAMPLE_RATE)]
    mix = np.argmax(measures)

    return degray(mix)
```

Prima di ritornare il simbolo, la funzione rimuove anche la codifica Gray (paragrafo 2.3.2) mediante la funzione di utilità:

```
def degray(n):
    mask = n
    while mask != 0:
        mask >>= 1
        n ^= mask
    return n
```

Tralasciando la sincronizzazione della finestra di campioni sulla quale si esegue la trasformata di Fourier, ci si può trovare nella situazione nella quale solo una piccola parte dei campioni trasformati contiene una vera

emissione audio relativa alla trasmissione di un simbolo, mentre la prima parte può essere costituita da silenzio. Questa possibilità è sostanzialmente aleatoria e dipende dall'istante di tempo nel quale il trasmettitore ha iniziato a trasmettere e da quello nel quale il ricevitore ha iniziato a ricevere.

Dal momento che i toni modulati in uscita sono sovrapposti, si avrà sempre in ogni finestra considerata che l'ampiezza di un certo tono sarà predominante, eliminando così ambiguità sul quale debba essere considerato (considerando chiaramente il fatto che il rumore potrebbe avere alterato il segnale in ingresso). Tuttavia è possibile che sia introdotto artificalmente un tono aggiuntivo all'inizio della trasmissione, dovuto alla cattura di una piccola coda di segnale al termine della prima finestra di ricezione. Dalla successiva finestra in poi il problema non sussiste poiché saranno presenti fin da subito uno o due segnali modulati, con preponderanza di uno in ogni caso.

Si può quindi verificare con una certa probabilità che il blocco di simboli rilevato dalla trasformata di Fourier e consegnati allo stadio successivo del modulatore contenga un primo simbolo spurio. Oltre a questo si deve considerare che se non viene impostata una soglia minima di energia (*squelch*) per considerare valido il risultato della trasformata, in assenza di modulazione essa estrarrà comunque il simbolo più probabile dal rumore di fondo.

Entrambi questi problemi sono stati risolti implementando prima dello stadio di deinterleaving un buffer a scorrimento (paragrafo 2.3.3): una volta ricevuto dalla trasformata un blocco completo di simboli, esso viene demodulato e si misura l'energia del rumore sullo spettro di Walsh a valle della trasformata di Walsh-Hadamard. Se tale errore supera una certa soglia, non viene scartato il blocco completo, ma semplicemente il primo simbolo. Alla ricezione del successivo simbolo, si tenterà nuovamente di decodificare il blocco costituito dai simboli rimanenti più il nuovo simbolo ricevuto.

```
    while True:
        # Fetch new samples
        updateBuffer()
```

```
        sym = detectSymbol()
        syms.append(sym)

        if len(syms) == 64:
            # Enough symbols to decode a block
            if decodeAndPrintBlock(syms):
                # Block decoded successfully, waiting for a new
one
                syms = []
            else:
                # Probably not a complete block, try rolling
                syms = syms[1:]
```

Quando il blocco considerato sarà composto unicamente da simboli realmente trasmessi dal modulatore – e quindi non da simboli estratti dal silenzio – il tasso di errori sarà più basso della soglia e il blocco risulterà valido. A questo punto il demodulatore è sincronizzato e si possono acquisire i successivi interi blocchi di simboli senza ulteriori problemi.

Questa è a tutti gli effetti l'implementazione di una finestra a scorrimento nel dominio di Walsh anziché nel dominio del tempo. Utilizzando questa logica di scorrimento, i problemi sopra evidenziati vengono affrontati dallo stesso codice utilizzato per la correzione dell'errore dovuto al rumore senza necessità di implementare onerosi metodi aggiuntivi di sincronizzazione del segnale audio ricevuto.

Viene eseguito il de-interleaving (paragrafo 2.3.4) dei simboli facenti parte del blocco:

```
    for i in range(0, spb):
        for j in range(0, 64):
            bit = (syms[j] >> ((i+j) % spb)) & 1
            if bit == 1:
                w[i,j] = -1
            else:
                w[i,j] = 1
```

La chiave standardizzata viene applicata in XOR bit-a-bit ai vettori ottenuti (paragrafo 2.3.5), così da annullare lo scrambling che era stato effettuato in fase di trasmissione:

```
w[i,:] = w[i,:] * (-2*np.roll(key, -13*i)+1)
```

Si procede ad applicare la trasformata di Walsh-Hadamard (paragrafo 2.3.6) per effettuare la correzione dell'errore. Viene misurata la riga massima nello spettro di Walsh per determinare quale sia con maggior probabilità il vettore originante. Se l'incertezza è maggiore di una soglia preimpostata si considera dubbio il vettore ricostruito.

```
w[i,:] = fwht(w[i,:])

c = np.argmax(np.abs(w[i,:]))

if abs(w[i,c]) < BLOCK_THRESHOLD:
    doubt += 1
```

Infine si ottengono i caratteri invertendo la mappatura carattere-vettore (paragrafo 2.3.7):

```
if w[i,c] < 0:
    c = c + 64
if c != 0:
    output += chr(c)
```

Il blocco di caratteri ricevuto viene però riportato in uscita solo se non si hanno dubbi sulla sua decodifica. La presenza di almeno un dubbio in un blocco lo rende non valido e causa lo scorrimento di un simbolo del buffer illustrato sopra, oltre a causare la mancata stampa a schermo del blocco:

```
if doubt == 0:
    print(output, end="", flush=True)
    return True
else:
    return False
```

3.5 Ordinamento della trasformata di Walsh-Hadamard

La diffusissima libreria SymPy contiene già codice di alta qualità per il calcolo della trasformata di Walsh-Hadamard e del suo inverso, mediante le funzioni `fwht` e `ifwht`. In prima istanza si è tentato, con successo, di utilizzare queste funzioni già pronte; il sistema era funzionante ma risultava non essere compatibile con le implementazioni di Olivia già esistenti. Ad una analisi approfondita è emerso che l'implementazione fornita da SymPy utilizza un ordinamento della matrice differente da quello utilizzato dall'implementazione di riferimento di Olivia. Ciò nulla toglie alla qualità dell'implementazione SymPy; entrambe le modalità infatti sono corrette e differiscono semplicemente nella presentazione dell'informazione in ingresso e in uscita.

Piuttosto che prevedere una riordinamento dei vettori per sfruttare l'implementazione SymPy ai fini di Olivia, si è preferito reimplementare da zero il codice della trasformata e del suo inverso, prendendo ad esempio il codice sorgente dell'implementazione di riferimento in C++ e traducendolo esattamente in Python. Si è potuto riscontrare che la conversione da C++ a Python è stata molto semplice in quanto non è stato necessario lato Python prendersi cura a basso livello della gestione della memoria dedicata ai vettori.

```
def fwht(data):
    '''
    Fast Walsh-Hadamard transform.
    '''
    step = 1
    while step < len(data):
        for ptr in range(0, len(data), 2*step):
            for ptr2 in range(ptr, step+ptr):
                bit1 = data[ptr2]
                bit2 = data[ptr2+step]

                newbit1 = bit2
                newbit1 = newbit1 + bit1

                newbit2 = bit2
                newbit2 = newbit2 - bit1
```

```
                data[ptr2] = newbit1
                data[ptr2+step] = newbit2

        step *= 2
    return data


def ifwht(data):
    '''
    Inverse Fast Walsh-Hadamard transform.
    There is a similar ready-made transform in sympy, but its
      output ordering (Hadamard order) is different from the
Olivia
    specified one, and it's more efficient to reimplement it
    directly rather than converting the output.

    This is a 1:1 translation from the Olivia C++ reference
    implementation.
    '''
    step = int(len(data)/2)
    while step >= 1:
        for ptr in range(0, 64, 2*step):
            for ptr2 in range(ptr, step+ptr):
                bit1 = data[ptr2]
                bit2 = data[ptr2+step]

                newbit1 = bit1
                newbit1 = newbit1 - bit2
                newbit2 = bit1
                newbit2 = newbit2 + bit2

                data[ptr2] = newbit1
                data[ptr2+step] = newbit2
        step = int(step/2)
    return data
```

Le porzioni di codice relative ad inizializzazione, interpretazione dei parametri ed interfaccia utente non sono state riportate. Il codice sorgente completo e direttamente eseguibile del modulatore e del demodulatore è stato pubblicato su GitHub e si può trovare all'indirizzo: https://github.com/sntfrc/olivia-python .

Capitolo 4

Esperimenti e misure

In questo capitolo si riportano i risultati di prove pratiche di trasmissione e ricezione effettuate utilizzando i programmi implementati, raccogliendo dati quantitativi sulle prestazioni in vari scenari.

4.1 Interoperabilità con implementazioni di riferimento

Una volta completata l'implementazione del codice si è reso necessario collaudarlo. A questo scopo sono stati condotti dei test di trasmissione e ricezione utilizzando, all'altro capo della comunicazione, il software Fldigi, sviluppato da Dave Freese a partire dal 2007 e ampiamente utilizzato dai radioamatori per gestire una grande quantità di sound card modes, compresa Olivia. Questo pacchetto software può essere considerato uno standard del settore.

Si è verificato che lo script `olivia-tx.py` produce un segnale audio decodificabile perfettamente da Fldigi; viceversa, Fldigi produce un segnale audio perfettamente decodificabile dallo script `olivia-rx.py`. Questo significa che è possibile utilizzare gli script implementati per comunicare con altre implementazioni Olivia senza particolari accorgimenti.

Si riscontrano alcune differenze di prestazioni in fase di ricezione: lo script `olivia-rx.py` presenta una latenza di ricezione inferiore a Fldigi, ma quest'ultimo riesce a sopportare quantità di rumore più elevate prima di presentare errori nel testo ricevuto. Il motivo di questa differenza risiede nella correzione degli errori in fase di ricezione: l'algoritmo di correzione qui implementato è più semplice di quello utilizzato in Fldigi, poiché si basa semplicemente sulla ridondanza che il segnale presenta in un singolo blocco. Fldigi è in grado di considerare i valori di più blocchi consecutivi per determinare con più confidenza se i caratteri ricevuti siano corretti o meno. Come mostrato più in dettaglio nelle seguenti sezioni, anche l'implementazione più semplice della correzione degli errori è in grado di fornire ottime prestazioni fino a livelli di rumore molto alti.

4.2 Prestazioni in assenza di rumore

Si è eseguito innanzitutto un test di prestazioni in assenza di rumore che servirà da metro di confronto per i test successivi. La metodologia di test, che sarà eguale per le prove successive è descritta di seguito.

1) Trasmissione del seguente breve testo:

`Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla vehicula purus purus, iaculis pharetra sapien sagittis in. Nulla quis dui nisi.`

tramite lo script `olivia-tx.py`, dopo che lo stesso si è stabilizzato con la trasmissione di diversi blocchi nulli.

2) Ricezione della modulazione con lo script `olivia-rx.py`, utilizzando come sorgente di input il driver “Stereo Mix” di Windows, che riporta con precisione tutti i suoni generati in uscita dal PC senza quindi l’introduzione di eventuali rumori ambientali che vi sarebbero nel caso di emissione e riproduzione mediante dispositivi esterni.

3) Contestuale ricezione dello stesso segnale tramite “Stereo Mix” da parte di Fldigi, allo scopo di visualizzare lo spettro, ottenere una misura del rapporto segnale/rumore e fare un confronto sulla qualità della demodulazione garantita dai due applicativi.

Il messaggio in questione richiede un tempo di circa 60 secondi per la trasmissione in Olivia con configurazione 32/1000.

La prova in assenza di rumore presenta uno spettro di questo tipo:

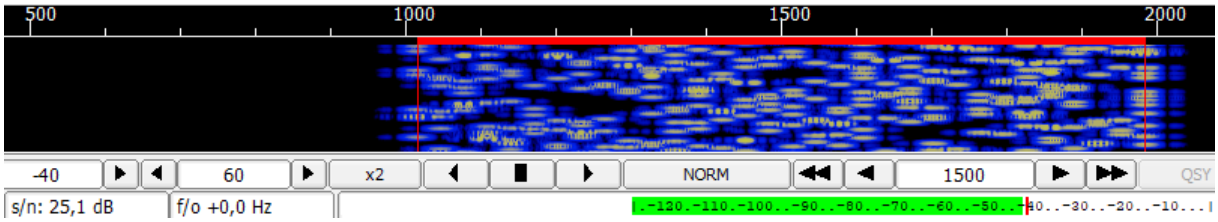


Figura 9: Spettro del segnale modulato in assenza di rumore.

Il rapporto segnale/rumore misurato con il volume di emissione utilizzato è di circa 25 dB, mentre il livello assoluto del segnale è -40 dBFS. Si manterrà invariato il volume di emissione nei test successivi per rendere significativo il confronto.

Come si può vedere in figura i toni emessi hanno una forma morbida sulla cascata dello spettro e hanno bande laterali che sfumano dolcemente nel tono successivo e precedente. Questo è una conseguenza del tone shaping e della sovrapposizione fatte per diminuire l'incidenza della modulazione intersimbolica.

L'assenza totale di rumore in questo test si può verificare osservando come la porzione di spettro che non comprende il segnale modulato, sotto i 1000 Hz e sopra i 2000, appaia totalmente nera (assenza di segnale).

Il testo è stato ricevuto perfettamente, con esattamente zero errori di trasmissione, sia da `olivia-rx.py` che da Fldigi.

4.3 Prestazioni in presenza di rumore bianco

Si esegue una prova in presenza di rumore bianco. Viene avviato un generatore di rumore bianco sulla stessa macchina che esegue gli script; in questo modo l'uscita audio del modulatore sarà combinata con il rumore mediante "Stereo Mix" prima di essere presentata all'ingresso del demodulatore.

Il generatore di rumore bianco viene regolato in modo da generare lo stesso livello audio misurato nella modulazione pura del test precedente (-40 dBFS). Lo spettro relativo al solo segnale di rumore è il seguente:

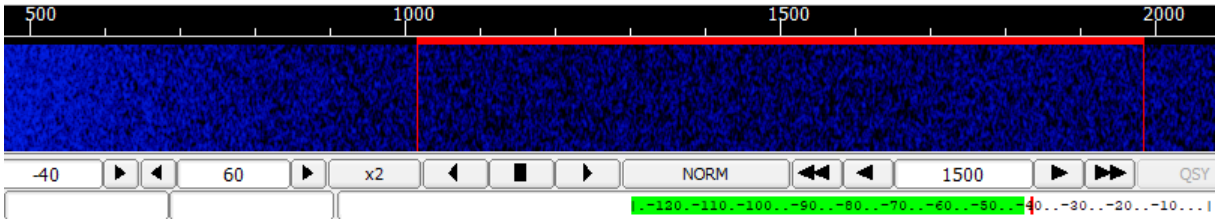


Figura 10: Spettro del solo rumore di fondo a -40 dBFS.

Come ci si aspettava, lo spettro è uniforme. Avviando a questo punto il demodulatore, in assenza di segnale modulato, esso estrae dal rumore l'informazione più coerente che sia possibile rilevare. I simboli generati per i quali c'è sufficiente "certezza" vengono stampati a video. Il risultato, nella prova in questione, è stato il seguente:

```
S$←?HP↑↕rZ:Gc]0}diMa,9♣¶8I4_♂S¶Q6{$!!&_DZ♥          94uA♦:↓w♀↑(+hVq☺&
```

Queste sequenze di caratteri prive di significato sono costruite a partire dalle oscillazioni casuali del rumore puro che siano compatibili con l'emissione di un modulatore. È possibile eliminarle richiedendo che il segnale in ingresso al demodulatore abbia una ampiezza superiore ad una certa soglia prestabilita (*squelch*) prima di essere considerato significativo. Questo non va comunque ad inficiare in alcun modo i risultati ottenuti in presenza di segnale modulato.

Si è proceduto ad applicare il segnale modulato sopra al segnale di rumore, ottenendo il seguente spettro:

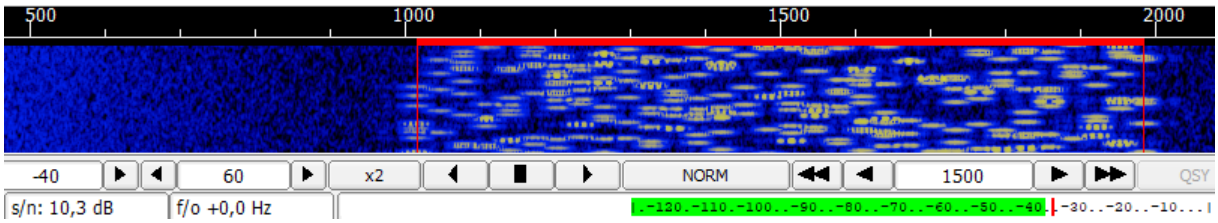


Figura 11: Spettro del segnale modulato sul rumore di fondo a -40 dBFS.

Si può notare come le bande laterali dei vari toni vadano a confondersi con il rumore sottostante. Il rapporto segnale/rumore è passato a circa 10 dB, una variazione di -15 dB rispetto al caso senza rumore. Nonostante ciò, le immagini dei toni sullo spettro sono ancora perfettamente visibili e discernibili. In effetti, il testo ricevuto sia da `olivia-rx.py` che da Fldigi è ancora totalmente privo di errori.

Il test successivo è stato eseguito con rumore di fondo molto più elevato, di ampiezza pari a -30 dBFS. Lo spettro del rumore considerato è:

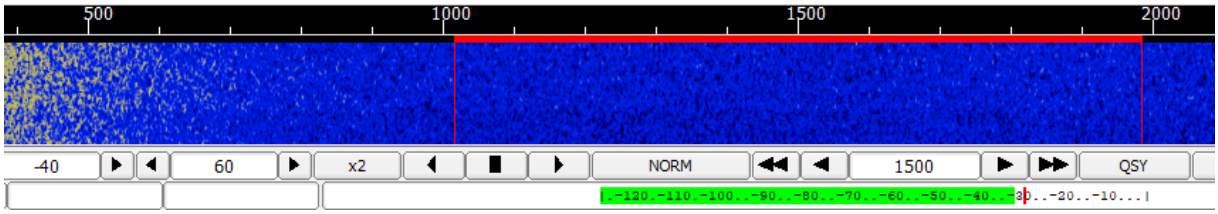


Figura 12: Spettro del solo rumore di fondo a -30 dBFS.

Il picco di ampiezza del rumore che si può notare a frequenze prossime allo zero è dovuto ad un limite dell'hardware audio. Il fenomeno (che si può notare in forma più lieve anche nel test precedente) non crea comunque problemi al test dal momento che è totalmente contenuto in

frequenze inferiori ai 1000 Hz e dunque non interagisce mai con il segnale modulato.

Il rumore generato è quindi un'ordine di grandezza maggiore in ampiezza rispetto a quello del test precedente. Si procede ad emettere il segnale modulato sopra a questo nuovo rumore.

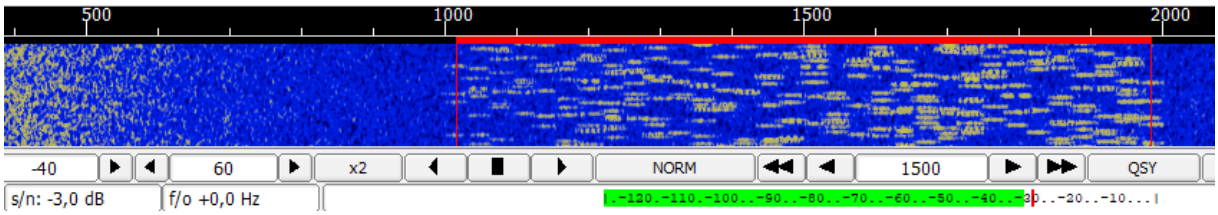


Figura 13: Spettro del segnale modulato sul rumore di fondo a -30 dBFS.

Si può notare come aumenti ulteriormente la confusione delle bande laterali dei toni corrispondenti ai simboli, che a questo punto sono praticamente scomparse nel rumore. Il rapporto segnale rumore è diventato negativo: -3 dB. Questo significa che al momento la potenza del rumore è il doppio di quella del segnale utile!

Nonostante il segnale abbia una potenza totale addirittura inferiore a quella del rumore, la demodulazione è ancora una volta perfetta, e il testo ricevuto sia da `olivia-rx.py` che da Fldigi è totalmente privo di errori. Questo denota una grande robustezza al rumore e fondamentalmente che l'obiettivo principale della modulazione Olivia si può considerare ben raggiunto.

A questo punto si sta utilizzando la potenza massima del generatore di rumore. Per simulare condizioni di rumore ancora più elevate si procede ad attenuare progressivamente il segnale modulato mantenendo costante l'ampiezza del rumore sottostante. Questo test è eseguito con il segnale modulato attenuato di 3 dB rispetto a quello dei test precedenti, e livello del rumore sottostante mantenuto ai precedenti -30 dBFS:

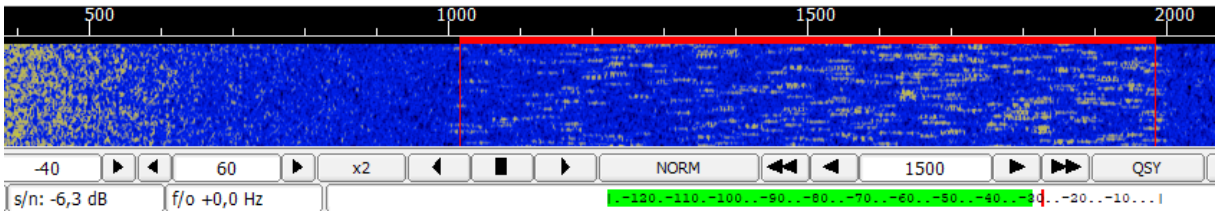


Figura 14: Spettro del segnale attenuato di 3 dB, sul rumore di fondo di -30 dBFS.

Visualmente si può notare come la forma dei toni sulla cascata dello spettro inizi ad essere seriamente intaccata dal rumore. Il rapporto segnale/rumore, come ci si aspettava è di circa -6 dB, ovvero 3 dB inferiore a prima a causa dell'attenuazione introdotta. A questo punto la potenza del segnale modulato è un quarto di quella del rumore.

Nonostante ciò, la demodulazione è ancora una volta perfetta, e il testo ricevuto sia da `olivia-rx.py` che da Fldigi è totalmente privo di errori.

Si procede ad aumentare a 6 dB l'attenuazione del segnale modulato, ottenendo il seguente spettro:

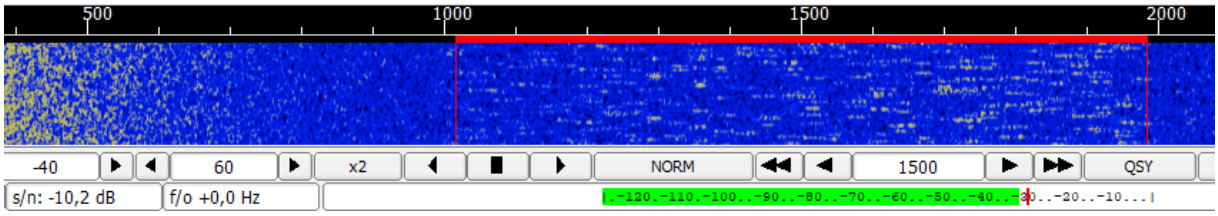


Figura 15: Spettro del segnale attenuato di 6 dB, sul rumore di fondo di -30 dBFS.

Il rapporto segnale/rumore misurato è di circa -10 dB, vicino ai -9 dB teorici che ci si aspettava. A questo punto la potenza del segnale modulato è un'ordine di grandezza inferiore a quella del rumore.

Nonostante ciò, la demodulazione è ancora una volta perfetta, e il testo ricevuto sia da `olivia-rx.py` che da Fldigi è totalmente privo di errori.

Si procede ad aumentare l'attenuazione del segnale modulato, portandola a 10 dB. Lo spettro ottenuto è il seguente:

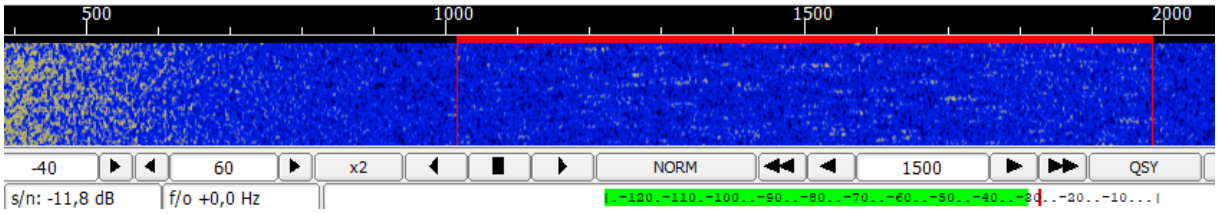


Figura 16: Spettro del segnale attenuato di 10 dB, sul rumore di fondo di -30 dBFS.

Visualmente si può notare come il segnale modulato sia quasi scomparso nel rumore e sia diventato molto difficile distinguere le immagini nello spettro dei singoli toni. Il rapporto segnale/rumore è di circa -12 dB, corrispondente a ciò che ci si aspettava.

A questo punto vediamo che si iniziano ad introdurre errori nel testo ricevuto dal demodulatore. Nella prova in considerazione, il testo ricevuto dal demodulatore `olivia-rx.py` è:

```
Lor!m ipsuor sit amensectetur adipiscing elit. Nulla vea purus purus,sis pharetra sapien sla qu
```

Sono stati introdotti i seguenti errori:

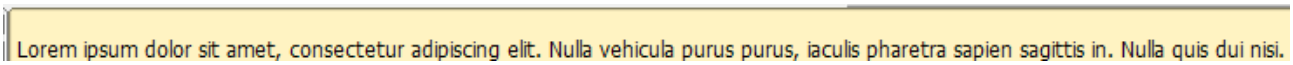
- “Lorem” è diventato “Lor!m”. Il blocco di 5 caratteri ricevuto contiene quindi errore su un singolo carattere. Il dubbio elevato su un carattere è stato considerato accettabile (sotto soglia) dal codice del ricevitore, quindi il blocco è stato stampato.
- “ipsum dolor” è diventato “ipsuor”: in questo caso un intero blocco di 5 caratteri (“m dol”) è stato scartato dal codice di ricezione poiché non vi era un sufficiente grado di certezza sul suo contenuto, ovvero vi era dubbio considerevole su più di un carattere del blocco.
- “vehicula” è diventato “vea”: il blocco di 5 caratteri “hicul” è stato scartato per lo stesso motivo del punto precedente.
- “, iaculis” è diventato “,sis”: il blocco “ iacu” è stato scartato per dubbio, e “lis” è diventato “sis” per introduzione di un singolo errore.
- I blocchi “agitt”, “is in”, “. Nul”, “is du”, “i nis”, “i.” sono stati scartati per dubbio.

Per vedere la comparsa dei primi errori è stato quindi necessario ridurre la potenza del segnale modulato fino a renderla più di dieci volte inferiore a quella del rumore. Fino a una potenza pari a un decimo di quella del rumore l’informazione è stata copiata perfettamente. Rimane comunque possibile discernere parzialmente il senso del testo.

Essendo il rumore di fondo per sua natura aleatorio, ripetendo la prova con gli stessi livelli probabilmente si avranno errori di tipo diverso e collocati in punti diversi del testo. Quindi eventualmente, in caso di comunicazione bidirezionale, è possibile chiedere di ripetere la

trasmissione per copiare le parti mancanti ed è quindi possibile avere ancora un canale di comunicazione funzionante.

Occorre però osservare che Fldigi ha copiato perfettamente anche questo segnale:



Questo si spiega con il fatto che Fldigi utilizza algoritmi di integrazione più complessi, ottimizzati per massimizzare il numero di caratteri copiati perfettamente, combinando i valori ottenuti dalla decodifica di più blocchi di seguito, a fronte di una maggior latenza in ricezione prima della comparsa del testo demodulato. La modulazione Olivia in sé quindi continua ad essere efficace anche a questi (e probabilmente anche maggiori) livelli di rumore; aumentando la sofisticazione del codice ricevente è possibile ancora estrarre informazione.

4.4 Prestazioni in presenza di rumore strutturato

Le prove con rumore bianco uniforme sono utili per misurare le caratteristiche della modulazione. Tuttavia nelle applicazioni reali può capitare che il segnale venga trasmesso sopra rumori di fondo più strutturati. Pensiamo ad esempio al caso in cui la trasmissione avvenga veramente tramite onde di pressione (ad esempio in uscita tramite il diffusore di uno smartphone e in ingresso tramite il microfono di un laptop). Tipicamente si avranno rumori ambientali non bianchi con potenze sensibilmente superiori a quelle del rumore di fondo.

4.4.1 Musica

Si riproduce un brano musicale sulla stessa macchina che esegue gli script; in questo modo l'uscita audio del modulatore sarà combinata con la musica mediante "Stereo Mix" prima di essere presentata all'ingresso del demodulatore.

Il volume della musica viene calibrato affinché l'energia del segnale musicale nel suo complesso sia pari a -30 dBFS e si rimuove ogni attenuazione applicata al segnale modulante, portandosi così alla situazione del secondo test con rumore bianco. La differenza è che in questo caso l'energia del segnale di rumore non è uniformemente

distribuita nello spettro, ma si condensa in alcuni punti creando sullo spettro delle immagini che assomigliano molto a quelle dei toni prodotti dal modulatore.

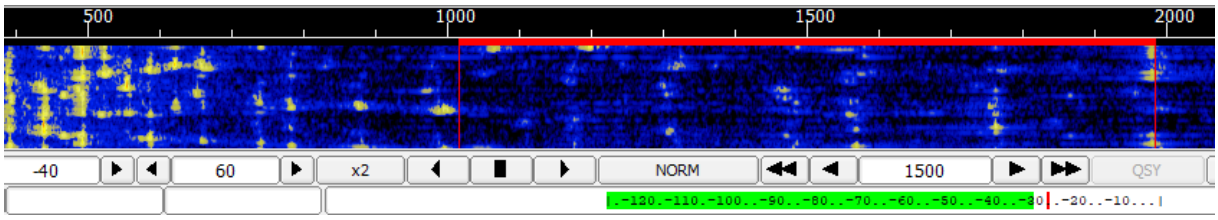


Figura 17: Spettro del solo rumore musicale a -30 dBFS.

Lo spettro risultante dalla modulazione del segnale sopra al rumore musicale è il seguente:

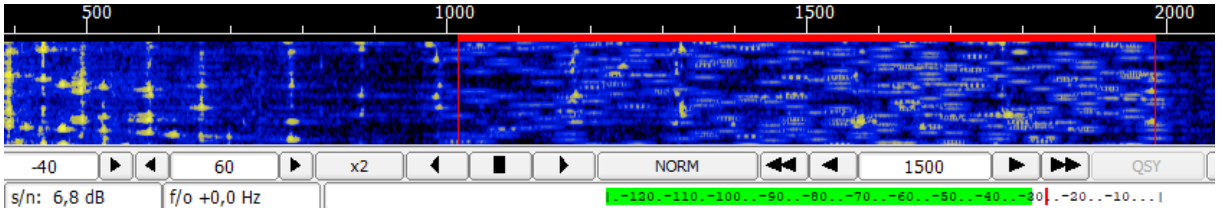


Figura 18: Spettro del segnale modulato sopra al rumore musicale a -30 dBFS.

Si può notare come effettivamente i toni generati dal modulatore vadano a confondersi con le note acute del brano. Ci si aspetterebbe una grande quantità di errori in ricezione, ma il fenomeno viene contenuto dalla proprietà di correzione dell'errore del demodulatore. L'errore finale nel testo demodulato da `olivia-rx.py` è basso:

```
Lorem ipsum dolor sit amet, consectetur adipiscing
elithicula purus purus, iaculis pharetra sapien s☺
gittis in. Nulla quis dui nisi.
```

Si tratta di un singolo carattere errato e dell'omissione di due blocchi.

Ancora una volta, Fldigi ha copiato invece perfettamente il testo trasmesso, grazie alle sue proprietà di integrazione. Vediamo quindi che il comportamento dei demodulatori è simile (sebbene migliore) a quello del test effettuato con rumore bianco in presenza di una attenuazione del segnale modulato pari a 10 dB.

Ne deduciamo che a parità di energia, un rumore più concentrato in frequenza è più deleterio per la modulazione rispetto ad un rumore di

densità uniforme. Questo è compatibile con il fatto che Olivia è stato ideato per utilizzo radioamatoriale, dove tipicamente il rumore di canale è uniforme. Un utilizzo diretto su onde di pressione non sarà quindi l'applicazione più brillante per questo tipo di modulazione.

4.4.2 Parlato

L'ultima prova è stata effettuata usando come segnale di rumore un TED Talk tenuto da una voce femminile (la scelta di una voce acuta è stata necessaria per far rientrare la maggior parte del rumore nella banda passante del segnale modulato).

Si è riprodotto il video con il parlato sulla stessa macchina di esecuzione del codice per cui l'uscita audio del modulatore è stata combinata con il parlato mediante "Stereo Mix" prima di essere presentata all'ingresso del demodulatore.

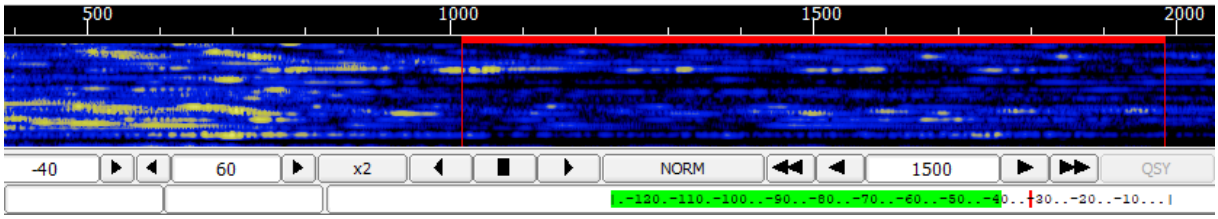


Figura 19: Spettro del solo segnale di rumore vocale.

Si può notare come in questo caso il rumore sia di tipologia intermedia fra quello bianco e quello musicale: l'energia rimane infatti concentrata su specifici intervalli di frequenza, ma più ampi rispetto a quelli delle note musicali. Osservando la cascata si vede come le immagini sullo spettro prodotte dal parlato siano tipicamente larghe più di un simbolo.

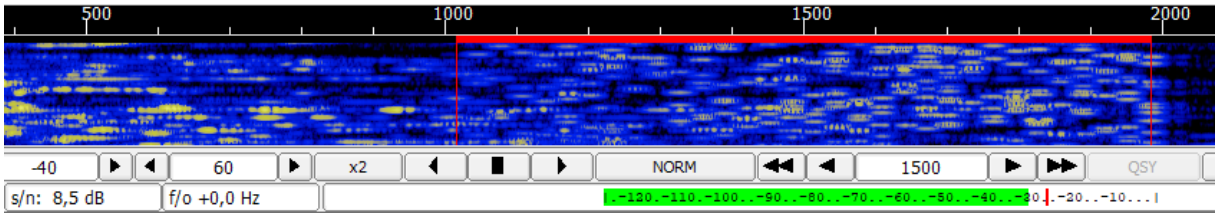


Figura 20: Spettro del segnale modulato sopra al rumore vocale.

Il testo ricevuto dal demodulatore è il seguente:

`Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla vehicula purus purus, iaculis pharetra sapien Qis in. Nulla quis dui nisi.`

Si nota quindi un solo blocco omesso e un solo carattere errato. Il rumore a parità di intensità è stato effettivamente più dannoso di quello bianco, e meno di quello musicale. Questo conferma l'interpretazione che la concentrazione dell'energia di rumore su intervalli di frequenza ristretti sia più penalizzante a parità di altre condizioni.

Rileviamo infine che Fldigi, come in tutti gli altri test, ha copiato perfettamente il segnale trasmesso senza alcun errore. Ne deduciamo che l'implementazione del modulatore fatta in `olivia-rx.py` è da considerarsi rapida e computazionalmente poco intensa, ma a discapito delle prestazioni.

Le misure fatte su Fldigi convincono del fatto che è possibile portare la modulazione Olivia ancora più al limite. Occorre però ricordare che in questi test si stanno considerando nel complesso livelli di rumore molto più elevati di quelli che possono verificarsi in applicazioni reali. La differenza fra le due implementazione probabilmente non sarà rilevante nell'utilizzo pratico.

4.5 Esperimento di trasmissione in radiofrequenza

Si è infine effettuata una prova pratica di comunicazione via radio, utilizzando apparati PMR446 off-the-shelf (reperibili in un comune negozio di elettronica). Questi apparati consentono di trasmettere con potenza di 500 mW, senza licenza, in modulazione analogica FM in un range di frequenze intorno ai 446 MHz (UHF). È stato utilizzato il canale PMR446 numero 7, corrispondente ad una frequenza di 446.08125 MHz, utilizzando una deviazione di frequenza massima di 2,5 kHz.

La stazione ricevente fissa (identificativo 1HFAF) è stata dotata di un apparato PMR446 modello Radioddity PR-T1, collegato tramite un cavo alla scheda audio USB del computer che esegue lo script di ricezione. Contemporaneamente sul computer è tenuto in esecuzione Fldigi allo scopo di graficare lo spettro e misurare il rapporto segnale/rumore.

Figura 21:
Stazione fissa 1HFAF.

Figura 22:
Stazione mobile 1HFAG.

La stazione trasmittente mobile (identificativo 1HFAG), dotata di apparato PMR446 modello Midland G9 Pro, è stata collocata ad una distanza di 200 metri in linea d'aria dalla stazione fissa, interponendo vari muri di cemento armato allo scopo di abbattere la qualità del segnale e rendere più rumoroso il canale per il test. È stato utilizzato un registratore audio ad alta fedeltà con il quale è stato in precedenza acquisito il segnale modulato dallo script di trasmissione corrispondente a questo messaggio:

`1HFAF DE 1HFAG Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla vehicula purus purus, iaculis pharetra sapien sagittis in. Nulla quis dui nisi. K`

L'altoparlante del registratore è stato accoppiato acusticamente al microfono dell'apparato radio senza l'utilizzo di cavi, potenzialmente introducendo ulteriore rumore ambientale.

Si evidenzia che l'utilizzo della modalità di modulazione FM anziché la più tipica SSB risponde a requisiti di semplicità, contenimento dei costi e di più ampia diffusione delle radio senza licenza, ma tipicamente peggiora il rapporto segnale/rumore delle modulazioni digitali a parità di potenza utilizzata per l'emissione. I risultati ottenuti sono da considerarsi pessimistici in confronto alle reali prestazioni ottenibili in una situazione senza ostacoli interposti fra le due stazioni e possibilmente utilizzando come substrato modulazioni analogiche più adatte.

Lo spettro risultante è il seguente:

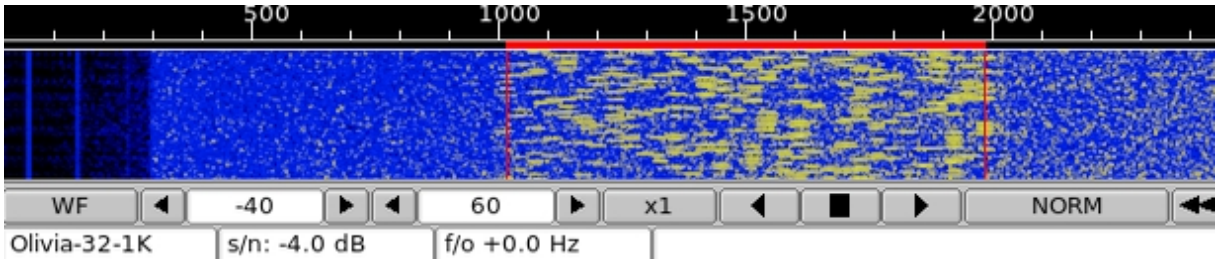


Figura 23: Spettro del segnale modulato ricevuto tramite canale FM in radiofrequenza UHF.

Il testo è stato copiato perfettamente nonostante il rapporto segnale/rumore negativo, di -4 dB, che indica una potenza totale del rumore di fondo (del tipo bianco) di più del doppio rispetto a quella del segnale. La situazione risulta simile a quella del precedente test con rumore bianco di fondo a -30 dBFS.

Raccogliendo i risultati in una tabella (*Tavola 1*) si evidenzia come l'implementazione realizzata fornisca le prestazioni migliori su rumore di tipo bianco; tuttavia rimane utilizzabile con ottimi risultati anche su rumori di tipo strutturato a patto che la potenza del disturbo non superi all'incirca il doppio di quella del segnale emesso.

Tipologia di rumore	Potenza di emissione	Rapporto segnale/ rumore	# blocchi trasmessi	# blocchi con errori	% blocchi con errori
nessuno	0	25,1 dB	29	0	0%
-40 dBFS (bianco)	0	10,3 dB	29	0	0%
-30 dBFS (bianco)	0	-3 dB	29	0	0%
-30 dBFS (bianco)	-3 dB	-6,3 dB	29	0	0%
-30 dBFS (bianco)	-6 dB	-10,2 dB	29	0	0%
-30 dBFS (bianco)	-10 dB	-11,8 dB	29	11	37,9%
-30 dBFS (musicale)	0	6,8 dB	29	3	10,3%
-30 dBFS (parlato)	0	8,5 dB	29	2	6,9%
rumore di fondo UHF	500 mW all'antenna	-4 dB	32	0	0%

Tavola 1: Tabella riassuntiva delle prove sperimentali effettuate.

Capitolo 5

Conclusioni

Riguardo alla modulazione Olivia si sono potute apprezzare quantitativamente le sue notevoli qualità di resistenza a rumore di vario tipo, confermandola come una delle modalità di comunicazione più robuste per gli scopi radioamatoriali, per i quali è stata creata. L'uso piuttosto inefficiente della banda passante del segnale, nonché la velocità di modulazione decisamente bassa ne limitano l'utilità pratica nelle comunicazioni di tutti i giorni. Tuttavia la relativa complessità ed eterogeneità degli stadi che la compongono la rende un buon caso di studio per stabilire la flessibilità e l'adeguatezza di un sistema di implementazione di calcolo numerico.

L'ecosistema Python, costituito dal linguaggio in sé e dalla completissima raccolta di librerie liberamente disponibili si è nel tempo evoluto da semplice sistema di scripting a vera e propria piattaforma utilizzabile per gli scopi più disparati. Adatto sia per la modellazione rapida ed intuitiva che per l'implementazione ottimizzata e definitiva, si presta molto bene al calcolo numerico, e risulta superiore ad altri sistemi comparabili per la disponibilità del codice sorgente, realizzato in maniera collaborativa da contribuenti di tutto il mondo e quindi verificabile nel suo funzionamento dalla comunità scientifica in generale, e per l'assenza di costi di licenza proibitivi per molti. Questi punti di forza gli stanno rapidamente conferendo una dignità generalmente riconosciuta in ambito accademico e lo rendono, oltre che un linguaggio informatico, un linguaggio di comunicazione efficace per la condivisione di algoritmi e la loro specifica formale.

La modulazione Olivia è fra le più complesse fra quelle ormai classiche utilizzate nel mondo radioamatoriale poiché è composta da molti stadi e fa uso di vari concetti di elaborazione dei segnali non banali. È degna di nota l'agilità e la chiarezza con la quale è stato possibile realizzarne un'implementazione in Python, ottenendo un codice sorgente chiaro e effettivamente quasi autodocumentante, che potrebbe tranquillamente essere utilizzato come specifica formale.

Un confronto con l’implementazione di riferimento in C++ rende chiaro come il livello di astrazione garantito da Python consenta la comprensione ad un’occhiata rapida di quello che il codice intende fare, riducendo al minimo le parti di codice *boilerplate* di supporto e gestione delle questioni più squisitamente informatiche collegate all’esecuzione del programma.

Grazie all’ampio ventaglio di piattaforme che supportano Python, il codice preparato può essere facilmente utilizzato e integrato in altri software del tipo più disparato, ad esempio app per smartphone o sistemi embedded come modem M2M o telemetria senza grandi sforzi di riconversione.

I benefici visti in questo caso di studio sono applicabili anche ad implementazioni di modulazioni o algoritmi di calcolo di maggior utilità commerciale e pratica e confermano come la transizione da sistemi di sviluppo più tradizionali a Python possa essere una scelta vincente sia per il mondo accademico che per quello industriale.

Bibliografia e sitografia

[LIT-1] F. Xiong - *Digital Modulation Techniques (2nd edition),* Artech House Telecommunications Library, 2006

[LIT-2] E. O. Brigham - *The Fast Fourier Transform and its applications*, Prentice-Hall, 1988

[LIT-3] S. Agaian, H. Sarukhanyan, K. Egiazarian, J. Astola - *Hadamard Transforms*, SPIE Press, 2011

[ARRL-1]: *The Draft Specification For The Olivia HF Transmission System.* http://www.arrl.org/olivia .

[GITH-1]: Repository contenente il codice dell'implementazione di riferimento, archiviato dal sito originale. https://github.com/mrbostjo/Olivia_MFSK

[RFC-4648]: The Base16, Base32, and Base64 Data Encodings. https://tools.ietf.org/html/rfc4648

[WIKI-1]: Immagine da https://upload.wikimedia.org/wikipedia/commons/thumb/3/39/Fsk.svg/800px-Fsk.svg.png . CC BY-SA 3.0.

[WIKI-2]: Immagine da https://en.wikipedia.org/wiki/Zero_crossing#/media/File:Zero_crossing.svg . Pubblico dominio.

[WIKI-3]: Immagine da https://en.wikipedia.org/wiki/Discrete_Fourier_transform#/media/File:Fourier_transform,_Fourier_series,_DTFT,_DFT.svg . CC0.

[WIKI-4]: Immagine da https://en.wikipedia.org/wiki/File:DIT-FFT-butterfly.png . CC BY 3.0.

[WIKI-5]: Immagine da https://en.wikipedia.org/wiki/Fast_Walsh%E2%80%93Hadamard_transform#/media/File:Fast_walsh_hadamard_transform_8.svg . CC BY-SA 3.0.

[WIKI-6]: Immagine da https://upload.wikimedia.org/wikipedia/commons/a/af/Gray_code_reflect.png . Pubblico dominio.