UNIVERSITÀ DEGLI STUDI DI UDINE

Dipartimento Politecnico di Ingegneria ed Architettura
Corso di Laurea triennale in Ingegneria Elettronica
(Cl. L-8)

Tesi sperimentale

SCRIPT LUA PER IL CONTROLLO REMOTO DI DRONI

Relatore:

Pier Luca Montessoro

Laureando:

Marco Sgobino

Anno Accademico 2020-2021

Indice

1 PREMESSA

Il seguente elaborato di tesi sperimentale si pone l'obiettivo di realizzare degli script in Lua per *OpenTX*, un firmware open source per radiocomandi, con particolare attenzione per i modelli di radiocomando *RadioMaster TX16S* e *Taranis QX7S*. È stato dunque rilevante lo studio della documentazione ufficiale relativa al firmware e al supporto Lua offerto da *OpenTX*. In definitiva, è stato possibile produrre degli script in Lua in grado di estendere le funzionalità in principio messe a disposizione dal firmware dei radiocomandi oggetto di sperimentazione, con uno sviluppo e una sperimentazione interamente effettuati sull'apposito software di simulazione del radiocomando, anch'esso parte integrante di OpenTX.

2 INTRODUZIONE

OpenTX [1] è un firmware per radiocomandi, scritto in C++, con supporto per oltre 20 modelli di trasmittenti radio [2].

Il software è rilasciato con licenza *GNU General Public License version 2* ed è dunque integralmente open source [3]. Lo sviluppo è attualmente in corso ed è gestito da una comunità di sviluppatori e piloti di aeromodelli e droni. Oltre a poter essere scaricato, compilato ed installato nel proprio radiocomando, il firmware è disponibile preinstallato su radiocomandi di vari produttori, fra cui *FrSky* e *RadioMaster*, aziende che hanno realizzato i due modelli adottati nella sperimentazione, il *Taranis FrSky QX7S* ed il *RadioMaster TX16S*.

Il modello di sviluppo open source del software presenta una moltitudine di vantaggi rispetto al modello del software proprietario. Alla base dei progetti, dei prodotti e delle iniziative OSS (Open Source Software) e OSH (Open Source Hardware) vi è lo scambio di informazioni, la cooperazione, la trasparenza, lo sviluppo rapido, la meritocrazia, e miglioramenti orientati alla comunità [4] (Burdziakowski, P., Razmjooy, N., Estrela, V., & Hemanath, J., 2020). Il software OSS è un software assieme al quale è rilasciato il codice sorgente che chiunque può sottoporre a scrutinio, modificare, e migliorare (Burdziakowski, P., Razmjooy, N., Estrela, V., & Hemanath, J., 2020).

Poiché i programmatori possono aggiungere funzionalità al codice sorgente di un software open source o modificare parti che non funzionano sempre correttamente, il programma può essere migliorato da chiunque ne sia in grado. Alcuni progetti nell'ambito dei *MAV* (Micro Aerial Vehicles) e della *fotogrammetria* che fanno uso del modello di sviluppo open source sono *Mission Planner* [5], *Ardupilot* [6], *OpenDroneMap* [7], *Web ODM* [8] e il firmware *OpenTX* (Burdziakowski, P., Razmjooy, N., Estrela, V., & Hemanath, J., 2020).

Vi è infatti una serie di ragioni per preferire un modello di sviluppo OSS al posto di un modello di sviluppo proprietario [4], fra le quali vi sono:

- il *controllo,* poiché il codice può essere ispezionato e testato con la consapevolezza dovuta alla comprensione del suo funzionamento interno;

- l'*addestramento,* dal momento che studenti e programmatori possono liberamente esaminare il codice e condividerlo con altri sviluppatori durante le fasi dello sviluppo, apprendendone da esso il funzionamento ed eventuali errori commessi con una profondità altrimenti impossibile;

- la *sicurezza,* poiché il codice liberamente ispezionabile può essere analizzato da esperti di sicurezza e sviluppatori, i quali sono in grado di individuare errori che gli autori originali non hanno identificato. Lo sviluppo più veloce ha come ulteriore e benefica conseguenza la correzione più rapida degli errori rispetto al modello a sorgente proprietario;

- la *stabilità,* dal momento in cui il modello OSS tende a sopravvivere nel tempo e ad incorporare standard aperti.

L'integrazione degli UAV (Unmanned Aerial Vehicles) nel *remote sensing* in ambito civile sta avendo un impatto sempre più significativo nel settore della fotogrammetria, fotografia, cinematografia, marketing, operazioni di soccorso, ma anche per generiche spedizioni e consegna di materiali in ambito medico e farmaceutico [9] (Cummings, Anthony R.; McKee, Arlo; Kulkarni, Keyur; Markandey, Nakul, 2017).

L'azienda di ricerche di mercato DOXA Marketing Advice ha compiuto uno studio in Italia relativo all'industria dei droni; nel 2016, questo mercato ha raggiunto livelli record come avvenuto nel 2013, anno nel quale l'ENAC ha introdotto le linee guida per i velivoli a pilotaggio remoto. Nel medesimo studio, viene messo alla luce che le industrie intervistate ritengono che le aree di maggior potenziale per i velivoli a pilotaggio remoto sono l'agricoltura, la fotogrammetria e i rilievi topografici [10] (Andrea Cardamone, 2017).

Dunque, in prospettiva alla crescita del mercato dei velivoli a pilotaggio remoto diviene di estrema importanza l'utilizzo e il contributo a progetti Open Source Software ed Open Source Hardware, al fine di accrescere l'offerta a disposizione sia per il consumatore che per il cliente professionista, nonché di produrre un ambiente di sviluppo collaborativo ove sia possibile includere o progettare degli standard aperti in un ecosistema software gestito da delle comunità di sviluppatori e dalle aziende interessate.

3 MATERIALI E METODI

3.1 Utilizzo e scopo di OpenTX

Il firmware OpenTX per radiocomandi è composto da due distinte applicazioni[11],

- il *System Firmware*, il quale si occupa della gestione e del mantenimento delle impostazioni dell'utente, della lettura degli input dei comandi e delle elaborazioni necessarie al funzionamento del radiocomando;
- il *Transmitter Firmware*, responsabile della generazione del segnale radio alla frequenza opportuna e codificato secondo il protocollo supportato dalla radio.

La memorizzazione delle informazioni nel radiocomando avviene tramite una memoria di tipo non volatile (ad esempio una memoria a stato solido *SD*), la quale è in grado di immagazzinare i dati di impostazioni, di calibrazione, le informazioni di tutti i modelli, gli script Lua, i suoni, le immagini. Il System Firmware può quindi leggere i dati in memoria e utilizzarli per pilotare il Transmitter Firmware, il quale genererà l'opportuno segnale radio tramite la circuiteria.

Il firmware *OpenTX* dunque si occupa sia di gestire l'input proveniente dagli slider (cursori), stick (leve), switch (interruttori) e knob (manopole) che compongono il telecomando producendo degli output nei 32 canali di trasmissione radio, sia di generare un'interfaccia grafica su uno schermo LCD, con le necessarie schermate di telemetria ed opzioni. Esso consente la configurazione completa della radio, dei modelli salvati in memoria su un supporto SD, o la visualizzazione di telemetria e dati ulteriori, come la batteria residua del radiocomando, la posizione mediante GPS, o l'indicatore *RSSI* della potenza del segnale in ricezione. In aggiunta a ciò, è possibile riprodurre file audio dall'altoparlante della ricetrasmittente per creare annunci personalizzati ed aiutare il pilota con feedback vocali durante le sessioni di volo.

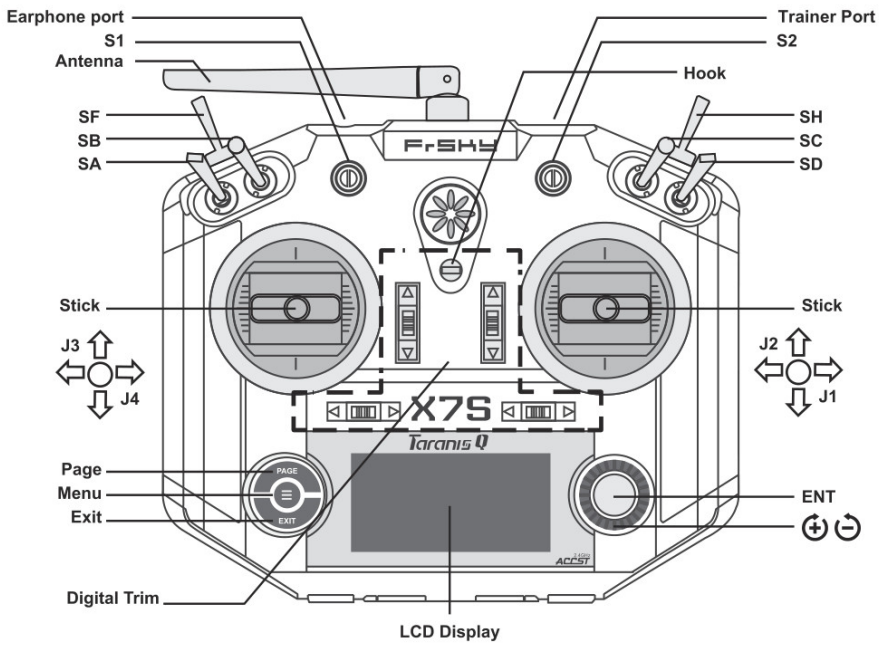

Figura 3.1: Leve e cursori del *Taranis QX7S*.

OpenTX rende inoltre disponibile un software ad interfaccia grafica, *OpenTX Companion* [1], disponibile per Windows, macOS e Linux. Tale software velocizza e facilita la configurazione del radiocomando con l'ausilio di un computer e consente di svolgere varie operazioni, fra cui:

- scaricare ed installare il firmware OpenTX per il proprio radiocomando;
- modificare le configurazioni del radiocomando e dei modelli;
- eseguire backup delle configurazioni e ripristinarle all'occorrenza;
- modificare la schermata d'avvio del radiocomando;
- simulare i radiocomandi ed i modelli tramite il software *OpenTX Simulator*.

3.1.1 Gestione degli input ed elaborazione degli output mediante i mixer

OpenTX gestisce gli input e l'output e la loro elaborazione tramite *cicli di calcolo* eseguiti varie volte al secondo; con ogni ciclo vengono lette le posizioni dei controlli, elaborati i *mixer* e trasmessi gli output nei corretti canali di trasmissione radio. Tutto ciò avviene con sufficiente rapidità da garantire una risposta più rapida possibile alle variazioni degli input e da simulare una continuità nel tempo di tale risposta.

Le sorgenti

Le *sorgenti* sono l'insieme delle leve, manopole e cursori che compongono i controlli del radiocomando. L'input viene letto assegnando alla posizione di ciascuno di essi un valore numerico:

- un valore continuo da -100 a $+100$ per gli stick, gli slider ed i knob. Gli stick sono indicati con *Ele* (Elevator stick), *Ail* (Aileron stick), *Thr* (Throttle stick) e *Rud* (Rudder stick). Quando uno stick è collocato nella sua posizione centrale, il suo valore di input è pari a 0. I knob invece sono indicati con *S1* ed *S2* (in alcuni radiocomandi con *F1* ed *F2*) e si tratta di manopole la cui funzione può essere liberamente impostata dall'utente. Gli slider di sinistra e destra (LS, RS) sono invece i potenziometri per gli stick, rispettivamente, di sinistra e di destra;
- -100, 0 oppure 100 per gli switch. Sono indicati con *SA, SB, SC* e così via. Sono azionabili su tre posizioni: interruttore al centro (valore 0), interruttore giù (valore 100) ed interruttore su (valore -100);
- 100 oppure 0 per gli slider con solo due posizioni (presenti soltanto in alcuni modelli).

I Mixer

L'input proveniente dalle sorgenti viene elaborato ed inviato ai canali di output attraverso i *mixer*. Ciascun mixer rappresenta l'*interazione fra uno o più input ed uno o più output*.

Un mixer assegna un input (sorgente) ad un output (canale), elaborandolo mediante varie modalità. OpenTX rende possibile applicare tre diversi tipi di configurazione:

1. *da una sorgente ad un canale*: una singola sorgente viene assegnata ad un singolo canale di uscita;
2. *da una sorgente a canali multipli*: una singola sorgente viene assegnata a più canali di uscita contemporaneamente. Questa configurazione è comune se si intende ad esempio pilotare più alettoni simultaneamente con un unica leva;
3. *da più sorgenti ad un singolo canale*: gli effetti di più sorgenti vengono sommati in un unico canale di uscita, e ciascuna sorgente fornisce un contributo indipendente al canale di uscita.

Tipologie di mixer

I mixer sono in grado di attribuire un *peso* ai valori d'ingresso applicando al valore del segnale un coefficiente espresso con una percentuale, e di assegnare al segnale risultante almeno un canale di

output. È possibile creare configurazioni combinando l'azione di più mixer per l'elaborazione dei vari input.

Un esempio è quello di impostare gli input dei quattro stick verso 4 canali di output, facendo in modo che il movimento completo dello stick produca un movimento completo del servo nel modello. Una possibile configurazione per tale scenario fa uso di 4 mixer, ed è la seguente:

```
Configurazione dei canali di output
Canale      Input       Mixer
CH1         I1:Rud      Weight (+100%)
CH2         I2:Ele      Weight (+100%)
CH3         I3:Thr      Weight (+100%)
CH4         I4:Ail      Weight (+100%)
```

Per l'input `Rud` è stato assegnato l'output nel canale `CH1` con peso `+100%`, per l'input `Ele` è stato assegnato l'output nel canale `CH2` anch'esso con peso `+100%`, e così via per gli altri due input `Thr` ed `Ail`.

Opzioni alternative possono essere quelle di ridurre l'effetto dei singoli stick, dando ad essi un peso minore (ad esempio la metà, 50%), oppure invertire il segno del peso di modo da produrre un movimento del servo nella direzione opposta.

È inoltre possibile impostare un *offset* per il valore in uscita, mediante la formula

$$\text{output} \;=\; \text{sorgente} \cdot \text{peso} \;+\; \text{offset}\,.$$

In aggiunta ai *pesi* e agli *offset*, OpenTX consente di valutare le sorgenti attraverso varie funzioni più complesse:

- *Diff*: riduce l'effetto dello stick in una direzione del movimento, applicando un peso, specificato in percentuale;
- *Expo*: applica una curva esponenziale al segnale di input;
- *Function*: applica una funzione matematica, o una disuguaglianza al segnale. Le funzioni sono in grado, ad esempio, di fornire in uscita il valore assoluto del segnale d'ingresso, oppure di prelevare l'input soltanto nel caso esso sia positivo $(x > 0)$;
- *Curve*: applica una curva impostata dall'utente. Esse sono configurabili assegnando manualmente il valore della curva in alcuni punti, il cui numero varia da 2 a 17.

3.2 Supporto OpenTX per gli script Lua

Con la versione 2.0 [12] OpenTX ha introdotto il supporto al *Lua*, un linguaggio di scripting general-purpose [13]. Gli script Lua sono immagazzinati nella memoria SD, tipicamente al percorso `/SCRIPTS/` con un nome massimo di 6 caratteri, esclusa l'estensione. Gli script Lua sono eseguiti dal firmware automaticamente o a discrezione dell'utente a seconda della tipologia dello script.

Gli script in Lua sono suddivisi in due categorie maggiori: gli *one-time script* o *script monouso*, e gli *script persistenti*.

Script persistenti

Si tratta degli script caricati dal firmware all'avvio, che rimangono in esecuzione fino a quando il radiocomando è in funzione oppure fino a quando viene selezionato un modello differente. Il firmware rende limitato l'utilizzo della memoria RAM da parte degli script Lua, pertanto è possibile eseguire contemporaneamente *fino ad un massimo di 7 script persistenti*. Il firmware inoltre si occupa della gestione della memoria e delle disponibilità di calcolo, interrompendo l'esecuzione di qualsiasi script che impieghi una quantità eccessiva di risorse.

Vi sono più tipi di script persistenti:

- gli *script di modello* o *model script*: sono eseguiti dal momento in cui viene selezionato un modello fino a quando esso viene deselezionato. Gli script di modello vengono anche detti *mixes script* poiché svolgono una funzione analoga, sebbene più avanzata, dei mixer in precedenza descritti nella sezione 3.1.1. Essi permettono dunque di elaborare un input o una serie di input restituendo un singolo valore o una table in output;

- gli *script di funzione* o *function script*: sono analoghi ai precedenti script di modello, con la differenza che essi vengono eseguiti quando una condizione su un interruttore o una leva si verifica (ad esempio, un interruttore viene azionato). Lo script viene ugualmente precaricato all'avvio del modello, e resta in attesa del verificarsi della condizione per cui esso è invocabile. Essi dunque svolgono un ruolo analogo a quello dei mixer, ma successivamente al verificarsi di una condizione. Possono anche gestire annunci vocali ed allarmi personalizzati, ma non sono in grado di stampare stringhe o valori numerici su schermo;

- gli *script di telemetria* o *telemetry script*: lo scopo di questi script è quello di presentare al pilota dati di volo e informazioni sullo stato del modello sulle apposite schermate di telemetria.

Sono in grado di interagire e modificare con la schermata del radiocomando. Possono inoltre riprodurre annunci vocali;

- i *widget*: sostituiscono gli script di telemetria, di cui sono l'evoluzione, nei modelli di radiocomandi più avanzati (ad esempio il *RadioMaster TX16S*). Essi vengono caricati ed eseguiti quando sono selezionati dalla schermata dei widget. Svolgono le stesse funzioni degli script di telemetria presentando maggiori funzionalità: oltre alla possibilità di interagire con la schermata e di stampare su schermo stringhe testo e figure, sono in grado di assumere una dimensione e collocazione sullo schermo variabile, con la facoltà di collocarne più di uno per schermata. Essi forniscono inoltre all'utente la capacità di modificarne parametri e variabili mediante un menù delle opzioni direttamente accessibile dall'interfaccia del radiocomando.

One-time script

Sono script invocati da una specifica funzione della radio oppure dall'utente tramite un menù contestuale. L'esecuzione di uno one-time script disabilita temporaneamente l'esecuzione di tutti gli altri script persistenti liberando la memoria RAM per lo script monouso. Gli script persistenti vengono poi ripristinati al termine dell'esecuzione dello one-time script. Gli one-time script possono essere adoperati per creare dei programmi di installazione o configurazione di modelli con interfaccia grafica direttamente dalla radio.

3.2.1 Struttura di uno script di modello

I *model script* sono eseguiti dal firmware successivamente al caricamento del modello nelle impostazioni del radiocomando. L'esecuzione non cessa mai fino alla selezione di un nuovo modello. Essi sono eseguiti con priorità *inferiore* rispetto ai mixer integrati visti nella sezione 3.1.1, e presentano un periodo di esecuzione dell'ordine dei $30ms$. La loro esecuzione è secondaria rispetto ai mixer, e a seconda delle disponibilità di risorse di calcolo e di memoria può non essere garantita.

La struttura di uno script di modello è la seguente:

- funzione `init()` (opzionale): eseguita una singola volta all'avvio dello script. Essa può essere adoperata per eseguire operazioni iniziali all'avvio dello script;

- funzione `run()`: eseguita periodicamente, essa può opzionalmente ricevere degli argomenti e restituire dei valori;

- tabella `input` (opzionale): table contenente uno o più elementi corrispondenti agli input dello script;
- tabella `output` (opzionale): table contenente uno o più elementi corrispondenti agli output dello script;
- statement `return`: collocato alla fine del codice, restituisce una tabella a cui associamo agli elementi `run`, `init`, `input` ed `output` le stringhe dei nomi delle corrispondenti funzioni o tabelle. Con esso si compie l'associazione fra le funzioni o variabili riconosciute dal firmware (`run()`, `init()`, `input`, `output`) con i nomi delle corrispondenti funzioni nello script, arbitrariamente scelti dall'utente.

La funzione `run()` non richiede parametri di input ed il return all'interno di essa è opzionale. La struttura di un model script con la sola funzione `run()`, a cui è stato assegnato il nome di `esegui_calcoli()`, sarà dunque la seguente:

```
local function esegui_calcoli()
      -- Esegui calcoli periodicamente
end

return { run=esegui_calcoli } -- esegui_calcoli e' ora
      -- associata alla funzione run() dello script
```

Le tabelle opzionali dei valori di input e di output contengono la stringa con il nome dei corrispondenti valori di ingresso ed uscita e vengono mostrate direttamente dall'interfaccia grafica del radiocomando, con a fianco il corrispondente valore. Per essi, OpenTX supporta la visualizzazione su schermo di nomi con lunghezza fino ad 8 caratteri. Gli ingressi di un model script possono essere più di uno, fino ad un totale di 6, e sono di due tipi:

- `VALUE`: valore numerico. Esso fornisce un valore costante in ingresso, definibile dall'utente mediante l'interfaccia grafica. La sua sintassi è una tabella in Lua del tipo

  ```
  { "Nome", VALUE, <minimo>, <massimo>, <predefinito> }
  ```

- `SOURCE`: sorgente di input da cui prelevare il segnale. Le sorgenti possono essere canali, interruttori, valori di telemetria, stick, e così via. La sintassi per le sorgenti è la seguente:

  ```
  { "Nome", SOURCE }
  ```

Un esempio più completo di struttura di un model script a 2 ingressi e 3 uscite è il sottostante:

```
local ingressi = {
      { "Sorgente", SOURCE}, -- Sorgente di input
      { "Numero", VALUE, 0, 100, 50} -- Valore numerico
                        -- compreso fra 0 e 100, e con
                        -- valore predefinito 50
}
-- contiene le stringhe dei nomi dei valori d'uscita
local uscite = {
      "Valore1:",
      "Valore2:",
      "Valore3:"
}
local function inizializza()
      -- Esegui operazioni all'avvio
      -- dello script
end
local function esegui_calcoli(sorgente_ingresso, numero)
      -- Gli ingressi della funzione esegui_calcoli
      -- sono rispettivamente i valori
      -- della table "ingressi"

      -- Esegui calcoli periodicamente

      return valore_uno, valore_due, valore_tre
      -- Questi valori sono restituiti
      -- in questo ordine
      -- nella table "uscite"
end
return { run=esegui_calcoli, init=inizializza, input=ingressi,
   output=uscite }
```

La funzione `esegui_calcoli(sorgente_ingresso, numero)` ha due variabili come suoi argomenti. Esse sono automaticamente prelevate, in ordine di dichiarazione, dalla table `ingressi`, perciò a `sorgente_ingresso` verrà associato il valore dell'input "Sorgente", il pri-

mo presente nella table `ingressi`, mentre a `numero` verrà associato il valore dell'input "Numero", dichiarato successivamente. Similmente, nell'istruzione di `return` sono restituiti tre valori: il primo, `valore_uno`, sarà associato all'uscita "Valore1", il secondo, `valore_due`, all'uscita "Valore2", ed infine `valore_tre` sarà associato all'uscita "Valore3".

Nel frammento di codice sopra si evidenzia come tutte le variabili, comprese le definizioni di funzioni, siano dichiarate come *variabili locali*. Eventuali variabili dichiarate non localmente sarebbero infatti visibili agli altri script in esecuzione nell'ambiente Lua di OpenTX, con possibilità di generare errori o interazioni fra script impreviste dovute all'utilizzo improprio della stessa variabile da parte di più script. I file `.lua` degli script di modello vanno collocati al percorso `/SD/SCRIPTS/MIXES`. I casi d'uso tipici di un model script sono:

- sostituzione di mixer complessi e non critici per il funzionamento del modello;
- elaborazione complessa di input e sorgenti e riproduzione di annunci vocali in relazione al loro comportamento;
- filtraggio di valori di telemetria.

3.2.2 Struttura di uno script di telemetria

Gli elementi di uno script di telemetria sono i seguenti:

- funzione `init()` (opzionale): viene eseguita all'avvio dello script di telemetria;

```
local function inizializza()
      -- Esegui operazioni all'avvio
end
```

- funzione `run()`: viene eseguita periodicamente, solamente quando la schermata di telemetria è visibile;

```
local function funzione_run()
      -- Esegui operazioni quando
      -- la schermata e' visibile
end
```

- funzione `background()` (opzionale): viene eseguita periodicamente, sia quando la schermata di telemetria è visibile che quando è nascosta;

```
local function funzione_background()
      -- Esegui operazioni sia quando
      -- la schermata e' visibile che
      -- quando la schermata e' nascosta
end
```

- statement `return`: restituisce la tabella a cui associamo agli elementi `run`, `background` e `init` le stringhe dei nomi delle corrispondenti funzioni o tabelle.

```
return { run=funzione_run, background=funzione_background,
   init=inizializza }
```

Dunque, le funzioni `run()` e `background()` svolgono un ruolo analogo a quanto osservato per la funzione `run()` degli script di modello, con la differenza che la loro chiamata dipende dalla schermata che viene visualizzata sul display del radiocomando. In aggiunta, esse hanno la facoltà di modificare l'interfaccia della radio nelle apposite schermate di telemetria e di stampare su schermo stringhe e valori.

Gli script di telemetria vengono caricati appena il modello è selezionato dai menù della radio. Viene inizialmente eseguita, se presente, la funzione `init()`, e al termine di essa sono periodicamente chiamate le funzioni `run()` e `background()` a seconda che sia mostrata la schermata di telemetria o una schermata differente[1].

A seconda della complessità dello script, potrebbe risultare conveniente consentire all'utente di configurare lo script di telemetria secondo le proprie esigenze. La configurazione degli script di telemetria non avviene mediante interfaccia grafica. La soluzione adottata è quella di dichiarare opportune variabili di configurazione nella parte iniziale dello script, il cui valore è da assegnare direttamente nel codice mediante un editor di testo:

```
-- Eventuali variabili di configurazione
-- il cui valore e' assegnato direttamente nel codice
local VARIABILE_DI_CONFIGURAZIONE = 0
local ALTRA_VARIABILE = 100
```

La collocazione degli script di telemetria nella scheda SD è al percorso `/SCRIPTS/TELEMETRY/`. I tipici casi d'uso degli script di telemetria possono essere:

[1] Nel caso in cui sia mostrata la schermata di telemetria, vengono eseguite in alternanza sia la funzione `run()` che la funzione `background()`.

- stampa su schermo di stringhe e dati di telemetria;
- riproduzione di annunci vocali personalizati;
- rappresentazione su schermo mediante figure e numeri di dati di telemetria.

3.2.3 Struttura di uno script Widget

I widget sono una variante più complessa degli script di telemetria. La loro struttura è la seguente:

- stringa `name`: il nome del widget (massimo 10 caratteri). Essa è dichiarata come variabile locale o passata direttamente tramite il `return` alla fine del codice;
- funzione `create()`: viene chiamata una singola volta alla selezione del widget dal menù del radiocomando. Crea il widget, e si occupa di passare alle altre funzioni presenti nello script le *dimensioni* (dette `zone`) del widget e la table `options` delle opzioni;

```
local function create (zone, options)
      -- Assegno al widget la zona e le opzioni
      -- ricevute come argomento dal firmware
      local widget = { zone=zone, options=options }
      return widget
end
```

- funzione `update()`: viene chiamata ogni volta che un'opzione presente nel menù contestuale del radiocomando viene modificata. Essa aggiorna la table delle opzioni con la nuova tabella passata dal firmware;

```
local function update(widget, options)
      -- Verifico che il widget
      -- sia stato modificato
      -- correttamente
      if (widget ~= nil) then
            widget.options = options
      end
end
```

- funzione `refresh()`: eseguita periodicamente, solo quando la schermata di telemetria è visibile;

```
local function refresh(widget)
      -- Esegui operazioni periodicamente
end
```

- funzione `background()`: eseguita periodicamente quando il widget non è visibile nella schermata. Si osservi che il comportamento di questa funzione differisce dal comportamento presentato dalla funzione `background()` degli script di telemetria, in quanto essa viene invocata solo quando il widget non è presente sulla schermata, e non più anche quando esso è visibile;

```
local function background(widget)
      -- Esegui operazioni periodicamente quando non e' visibile
end
```

- table `options`: tabella delle opzioni configurabili dal pilota tramite l'interfaccia grafica. Esse presentano una sintassi del tutto simile alla tabella `input` degli script di modello;

```
local options = {
      { "Color", COLOR, BLUE },
      { "Number", VALUE, 10, 1, 100 },
      { "Flag", BOOL, 0 }
}
```

- statement `return`: restituisce la tabella a cui associamo agli elementi `name`, `create`, `update`, `refresh`, `background` ed `options` le stringhe dei nomi delle corrispondenti funzioni o tabelle analogamente a quanto avveniva per gli script di telemetria o gli script di modello.

```
return { name="NomeWidget",
      options=options,
      create=create,
      update=update,
      background=background,
      refresh=refresh }
```

I possibili tipi di dato utilizzabili nelle opzioni sono `VALUE`, `SOURCE`, `BOOL` e `COLOR`. In particolare:

- `VALUE`: valore numerico. Esso fornisce un valore costante, definibile dall'utente mediante l'interfaccia grafica. La sua sintassi è una tabella in Lua del tipo:

```
{ "Nome", VALUE, <predefinito>, <minimo>, <massimo> }
```

- `SOURCE`: sorgente di input da cui prelevare il valore. Le sorgenti possono essere canali, interruttori, valori di telemetria, stick, e così via. La sintassi per le sorgenti è la seguente:

```
{ "Nome", SOURCE }
```

- `BOOL`: variabile booleana avente possibili valori `0` (vero) ed `1` (falso). La sintassi è la table:

```
{ "Nome", BOOL, <vero/falso> }
```

- `COLOR`: variabile che contiene un colore, principalmente utilizzata per impostare il colore in cui verranno stampati testo, elementi dell'interfaccia o sfondo del widget. È utilizzata all'interno di una table avente formato:

```
{ "Nome", COLOR, <rgb> }
```

  I colori sono espressi in formato `RGB565`, oppure è possibile assegnare i colori predefiniti dal firmware `WHITE`, `GREY`, `LIGHTGREY`, `DARKGREY`, `BLACK`, `YELLOW`, `BLUE`, `RED` e `DARKRED`.

I widget vanno collocati nel percorso `/WIDGETS/<NomeWidget>/main.lua`, ovverosia come un file di nome `main.lua` e dentro una cartella avente lo stesso nome di quello assegnato al widget e indicato all'interno dello script.

3.2.4 Interazione con il display del radiocomando

Gli script di telemetria e i widget offrono la possibilità di interagire con il display LCD del radiocomando. L'accesso è limitato, ed è esclusivo delle funzioni `run()` degli one-time script e telemetry script, e della funzione `refresh()` dei widget. Le caratteristiche del display LCD variano profondamente a seconda del modello di radiocomando utilizzato. Gli script in Lua che si interfacciano con l'utente tramite il display devono essere progettati tenendo conto delle caratteristiche dello schermo

LCD dei vari modelli. In questo elaborato esporremo le caratteristiche dello schermo dei due modelli adoperati, il *Taranis QX7S* ed il *RadioMaster TX16S*.

Display LCD del Taranis QX7S

Il display LCD del *Taranis* ha una risoluzione di 128×64 pixel e supporta 1 bit di colore. La schermata si presenta come in Figura 3.2.

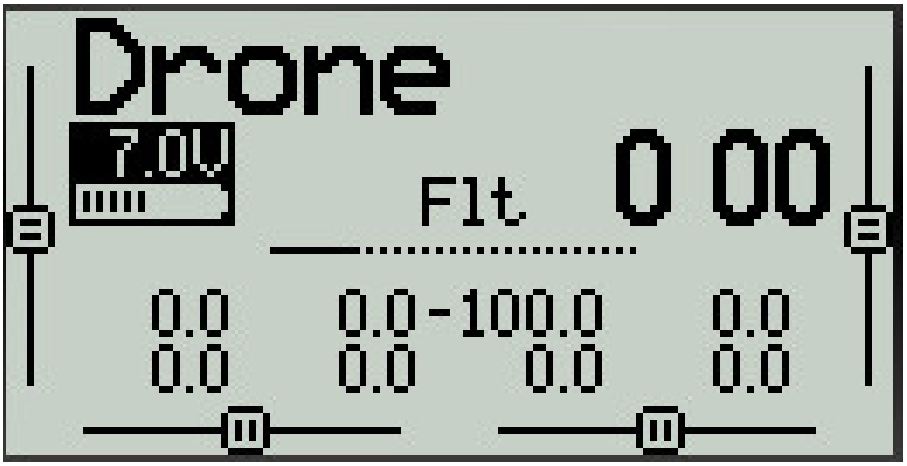

Figura 3.2: Schermata principale del *Taranis QX7S*.

La schermata a due colori (chiaro e scuro) rende possibile la lettura di scritte e semplici figure.

Display LCD del RadioMaster TX16S

Il display LCD del *RadioMaster* ha una risoluzione di 480×272 pixel e supporta la modalità di colore `RGB565`. La schermata principale si presenta come in Figura 3.3. Si può osservare che lo schermo, diversamente dal *Taranis*, supporta una modalità a colori avanzata. La risoluzione è infatti sufficiente per visualizzare figure complesse e scritte di varie dimensioni e colore.

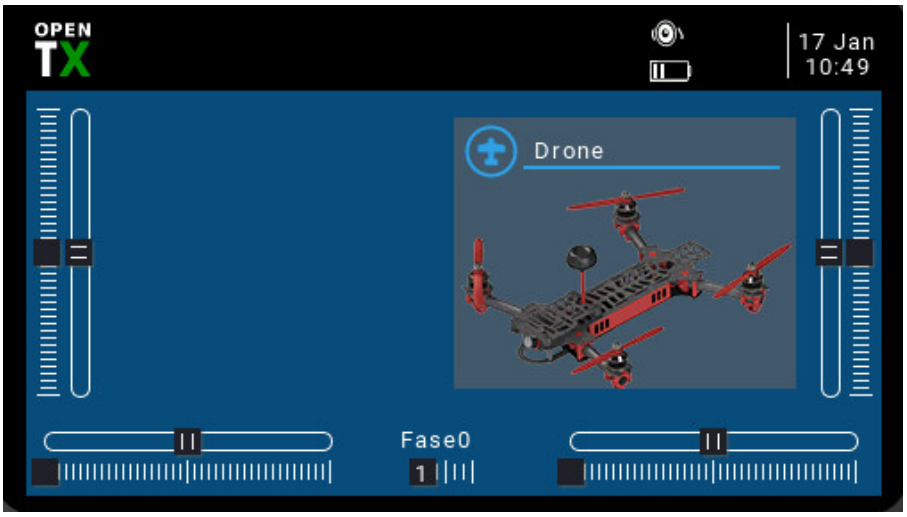

Figura 3.3: Schermata principale del *RadioMaster TX16S*.

Gestione dello schermo mediante le funzioni `lcd`

L'ambiente Lua di OpenTX mette a disposizione dei metodi interni all'oggetto `lcd` in grado di gestire lo schermo LCD del radiocomando, fornendo un'interfaccia tra di esso ed lo script in Lua. Tali metodi possono essere invocati soltanto dai tipi di script per cui è prevista la possibilità di modificare lo schermo, ovvero dagli *one-time script*, dai *telemetry script* e dai *widget* per i radiocomandi che li supportano.

FLAG	**Descrizione**
`0`	Testo normale, nessuna cifra decimale
`DBLSIZE`	Testo a dimensione doppia
`MIDSIZE`	Testo a dimensione intermedia
`SMLSIZE`	Testo a dimensione piccola
`INVERS`	Colore di testo e sfondo invertito
`BLINK`	Testo lampeggiante
`XXLSIZE`	Testo a dimensione molto grande
`LEFT`	Testo giustificato a sinistra
`RIGHT`	Testo giustificato a destra
`PREC1`	Singola cifra decimale per i numeri
`PREC2`	Doppia cifra decimale per i numeri
`GREY_DEFAULT`	Lo sfondo del testo è grigio
`TIMEHOUR`	Mostra le ore

Tabella 3.1: Lista dei possibili flag per i metodi dell'oggetto `lcd`.

I metodi che sono stati adoperati per la gestione dell'interfaccia grafica nella realizzazione degli script sono i seguenti:

- `lcd.clear([color])`: metodo utilizzato negli script di telemetria per resettare la schermata prima di effettuare operazioni di stampa su schermo. Il comando accetta come argomento, opzionalmente, un colore con cui effettuare l'azzeramento della schermata, qualora lo schermo supportasse la modalità a colori. Tale metodo non è necessario negli script widget;

- `lcd.drawText(x, y, text, [flags])`: stampa su schermo la stringa di testo `text`, alla posizione orizzontale `x` e verticale `y`. L'elenco dei `flag` disponibili è indicato in tabella 3.1;

- `lcd.drawTimer(x, y, value, [flags])`: stampa su schermo un timer con valore in secondi `value`, alla posizione orizzontale `x` e verticale `y`. L'elenco dei `flag` disponibili è indicato in tabella 3.1;

- `lcd.drawScreenTitle(title, page, pages)`: stampa su schermo il titolo della schermata `title`, al numero di pagina `page` e con numero totale di pagine `pages`;

- `lcd.setColor(area, color)`: per gli schermi in grado di supportare i colori, assegna un colore `color` alla porzione di schermo indicata dalla costante `area`. In questo elaborato, per il valore di `area` è stata utilizzata la costante `CUSTOM_COLOR`;

- `lcd.drawFilledRectangle(x, y, w, h, [flags])`: disegna un rettangolo pieno, alla posizione orizzontale `x` e verticale `y` e di ampiezza `w` ed altezza `h`. L'elenco dei `flag` disponibili è indicato in tabella 3.1.

3.3 OpenTX Companion

OpenTX Companion è un software di supporto per il firmware OpenTX, sviluppato dal medesimo gruppo di sviluppatori che si occupa del firmware. Esso è un programma scritto in C++ e sviluppato per Linux, macOS e Windows che consente la configurazione del radiocomando direttamente da computer, senza incorrere nelle limitazioni imposte dalla dimensione della schermata della radio e dai suoi pulsanti. OpenTX Companion è infatti adoperato per svolgere agevolmente vari compiti, fra i quali il caricamento del firmware sul radiocomando, eseguire il backup delle impostazioni dei modelli, della scheda SD e del firmware, modificare le configurazioni del radiocomando e condurre delle simulazioni del radiocomando [14]. Quest'ultimo compito è svolto grazie al simulatore *OpenTX Simulator*, incluso nel software Companion.

3.3.1 Specifiche e compilazione del sorgente

Per questo elaborato è stato adoperato il sistema operativo Fedora Linux 33 [15], con la versione 2.3.11 di OpenTX Companion compilata da codice sorgente. La compilazione del software è avvenuta dapprima installando le necessarie dipendenze, successivamente clonando il repository di `opentx` ed infine eseguendo lo script di compilazione `build-companion-release.sh`. Sono stati eseguiti i seguenti comandi (in bash):

```
# Installazione delle dipendenze
sudo dnf -y install sed make cmake git gcc gcc-c++ gcc-arm-linux-gnu
   avr-gcc-c++ avr-gcc xsd bc python python3-pyqt4-sip PyQt4-qsci-api
   python3-PyQt4-devel python3-pillow python3-pillow-devel avr-libc
   qt5-qttools qt5-qttools-devel qt5-qtmultimedia qt5-qtmultimedia-devel
   qt5-linguist qt5-qtsvg qt5-qtsvg-devel dfu-util gcc-plugin-devel fox
   fox-devel fox-utils SDL SDL-devel SDL2 SDL2-devel SDL_sound sdljava
   arm-none-eabi-gcc-cs arm-none-eabi-gcc-cs-c++ arm-none-eabi-newlib
   rpmdevtools rpm-build

# Clone del repository
git clone --recursive -b 2.3 https://github.com/opentx/opentx.git
cd opentx

# Script per la compilazione
./tools/build-companion-release.sh ../ ./build 23
```

Al termine dell'esecuzione dell'ultimo comando, OpenTX sarà correttamente compilato all'interno della cartella `build`. Sarà possibile eseguire il programma con `./build/companion23`.

Alternativamente, è fornito un eseguibile per Windows, Ubuntu o macOS nella relativa pagina web di download [16].

3.3.2 Utilizzo e funzionalità

Al primo avvio di OpenTX Companion è presente una finestra introduttiva, dove viene richiesta una prima configurazione di un profilo di radiocomando. Si seleziona il modello di radiocomando desiderato, nel nostro caso *Taranis QX7S* oppure *RadioMaster TX16S*, e si assegna una breve descrizione al profilo. Successivamente, si include il supporto agli script Lua selezionando la relativa casella e si

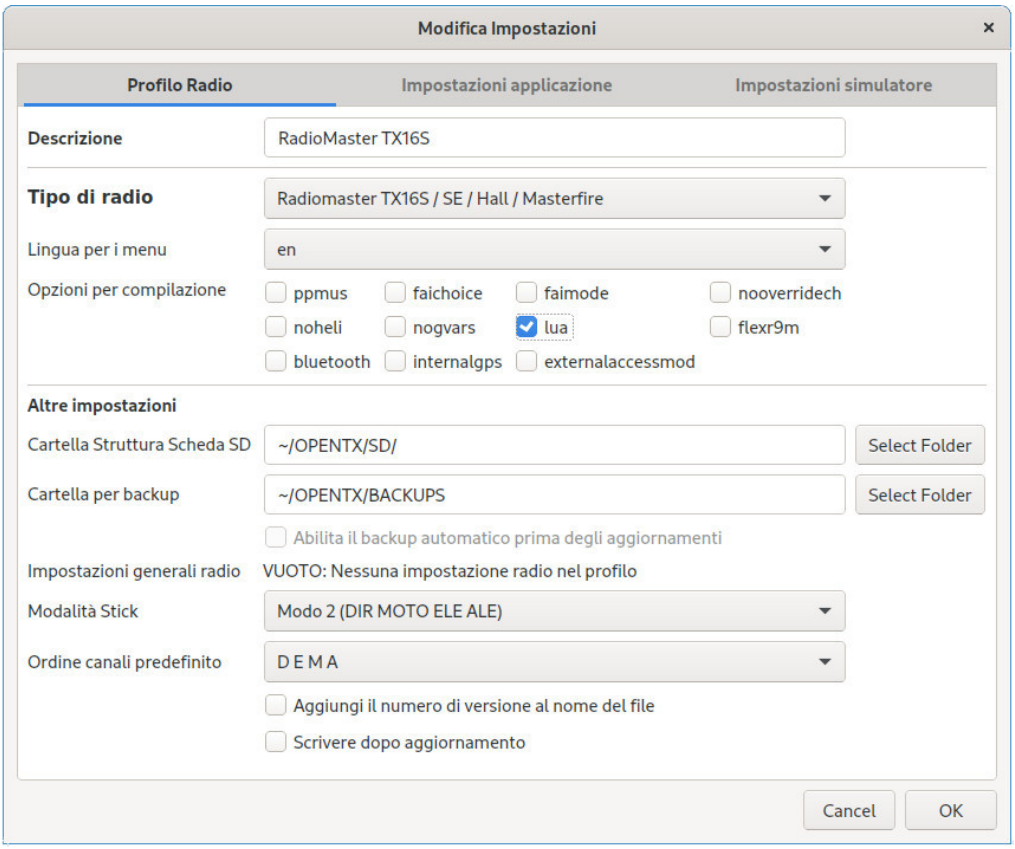

Figura 3.4: Finestra di prima configurazione di OpenTX Companion.

indica il percorso dove verranno collocati i contenuti della scheda SD, ed opzionalmente, il percorso per i backup. Le restanti impostazioni sono lasciate ai loro valori predefiniti.

La schermata principale di OpenTX Companion si presenta come in Figura 3.5. Prima di poter simulare un modello, è necessario scaricare il firmware e i contenuti della scheda SD. Per fare ciò, si naviga sull'icona di "Download" nella barra degli strumenti per procedere con lo scaricamento. I contenuti della scheda SD andranno poi decompressi con un programma di gestione di file `.zip` e collocati nella posizione precedentemente indicata.

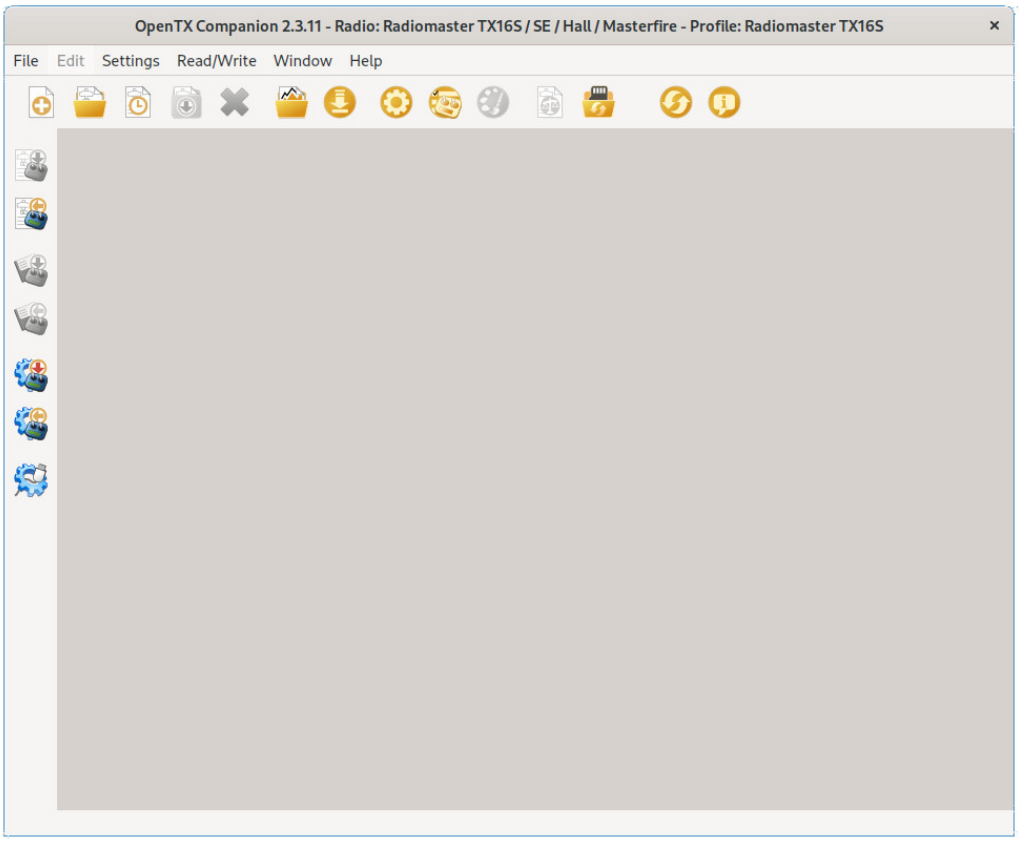

Figura 3.5: Finestra principale di OpenTX Companion.

Prima di simulare il proprio radiocomando è necessario creare un nuovo documento in OpenTX, e collocarvi almeno un modello. Si naviga sull'icona "New" nella barra degli strumenti e si aggiunge un nuovo modello con "Add Model" all'interno della finestra che è stata aperta. Il Wizard di configurazione del modello guiderà l'utente nella selezione delle sue specifiche. È possibile configurare ulteriormente le opzioni della radio in "Edit Radio Settings" prima di lanciare la simulazione. In tale menù è possibile applicare varie configurazioni, calibrazioni e funzionalità specifiche come la possibilità di implementare delle *Global Functions*, azioni da eseguire al verificarsi di un evento (ad esempio, una leva è in una determinata posizione o il pilota aziona un determinato bottone).

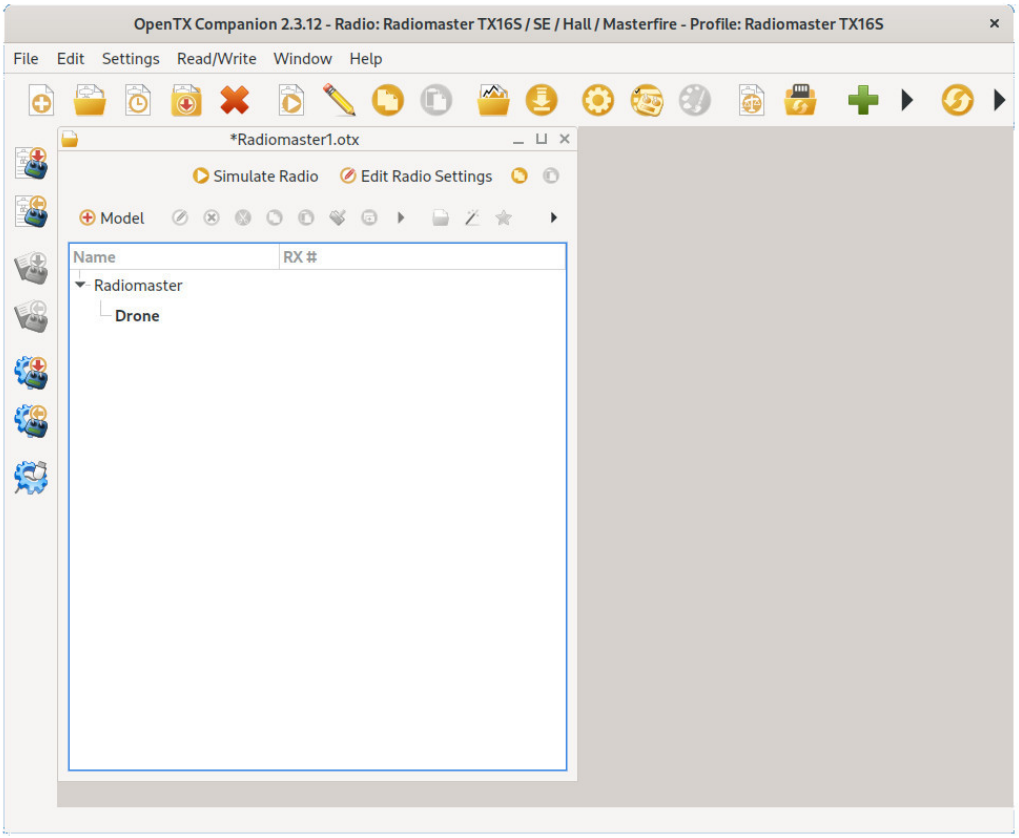

Figura 3.6: Creazione di un documento in OpenTX Companion.

Terminata la configurazione, è possibile eseguire la simulazione con *OpenTX Simulator* navigando su “Simulate Radio”. Una nuova finestra con il simulatore ci mostrerà una rappresentazione del radiocomando con le necessarie leve, pulsanti ed una riproduzione della schermata. La navigazione all’interno dei menù e delle schermate della radio avviene con i medesimi pulsanti utilizzati per il radiocomando reale. Poiché i vari radiocomandi sono rappresentati il più fedelmente possibile, le modalità di navigazione e le schermate presentate possono differire notevolmente fra i vari modelli.

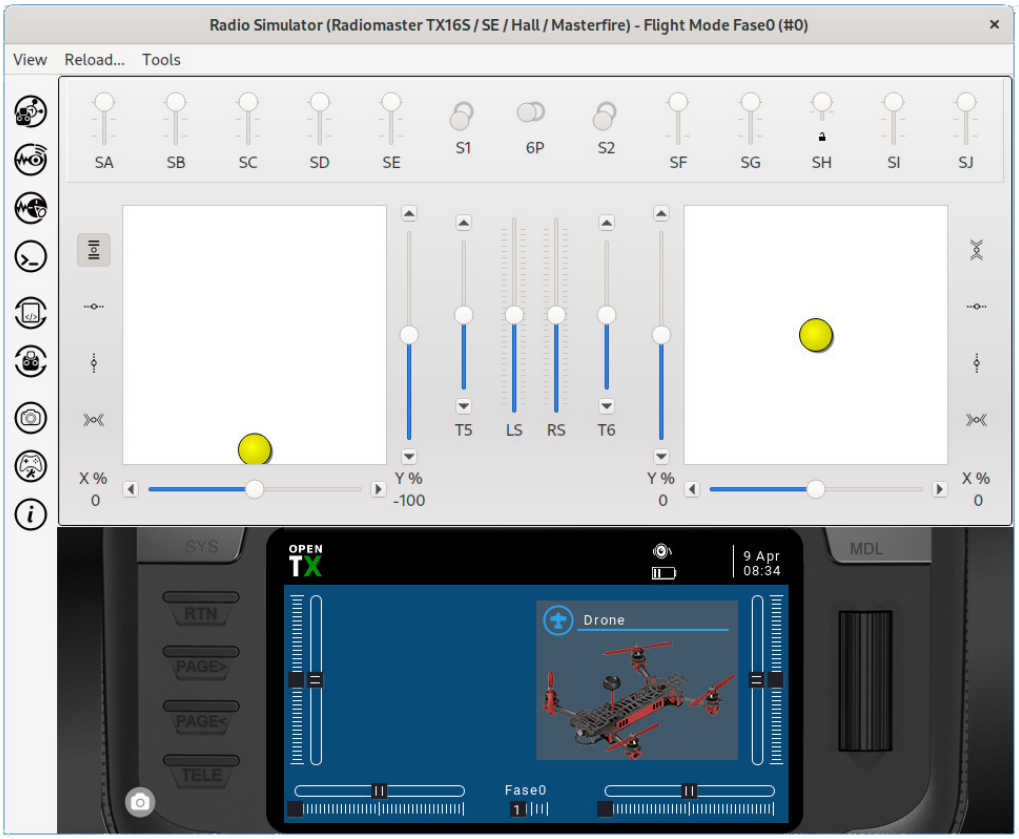

Figura 3.7: Simulazione del radiocomando *RadioMaster TX16S* in OpenTX Simulator.

Degna di nota è la schermata di configurazione dei modelli. Per ciascuno di esso infatti, navigando sul nome del modello, tasto destro del mouse, "Edit Model", è possibile accedere alla configurazione avanzata. È possibile impostare:

- i *timer*, sistemi di radio interni ed esterni, il *trainer*, nome ed immagine di modello;
- le *modalità di volo*;
- gli *input*, i *mixer* e gli *output*;
- le *curve*;
- i *logical switches*, interruttori logici utilizzati per comparare valori e combinare il verificarsi di varie condizioni;
- le *special functions*, per la riproduzione di annunci personalizzati, attivazione della modalità trainer e così via;
- la *telemetria*.

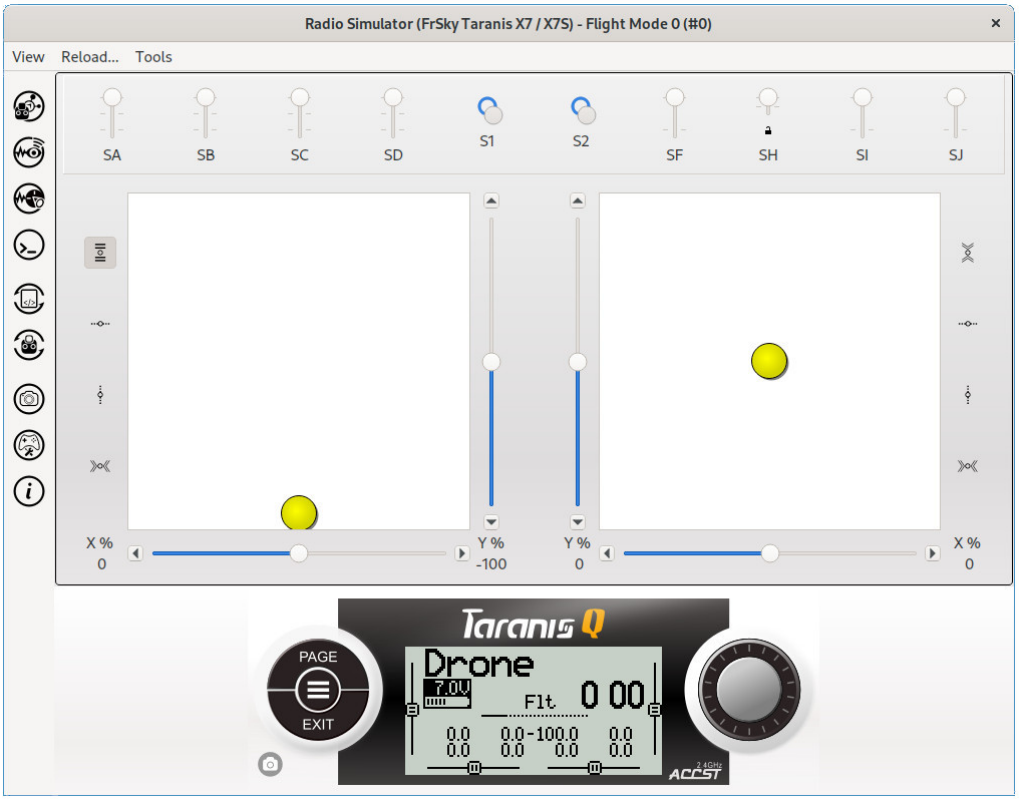

Figura 3.8: Simulazione del radiocomando *Taranis FrSky QX7S* in OpenTX Simulator.

Per l'avvio e l'utilizzo degli script trattati in questo elaborato si è resa necessaria la configurazione della telemetria di entrambi i radiocomandi adottati. Per il *Taranis FrSky QX7S* sono state configurate due schermate di telemetria, una per ciascuno dei due script Lua. Nel caso del *RadioMaster TX16S*, invece, si è fatto uso della configurazione dei widget direttamente dall'interfaccia del radiocomando in fase di simulazione, pertanto non è stato necessario adoperare il programma Companion per l'aggiunta delle interfacce di telemetria.

Il vantaggio legato all'utilizzo di OpenTX Companion è stato quello di poter configurare il radiocomando mediante interfaccia grafica su un sistema operativo per computer desktop. Inoltre, di fondamentale importanza è stata la facoltà di poter personalizzare e simulare i radiocomandi senza richiedere la loro presenza in ogni fase del progetto.

4 RISULTATI E DISCUSSIONE

In questo elaborato di tesi sperimentale sono stati realizzati due script Lua, lo *script di cronometro vocale* (`TmrCnt`) e lo *script di conto alla rovescia vocale* (`CntDwn`), ciascuno dei quali è stato realizzato per ambedue i modelli di radiocomando adoperati, il *Taranis FrSky QX7S* ed il *RadioMaster TX16S*.

4.1 Lo script di cronometro vocale

L'obiettivo dello *script di cronometro vocale* è stato quello di fornire al pilota in tempo reale un'indicazione sia visiva che acustica del tempo trascorso dall'inizio della fase di volo. Le informazioni del cronometro, oltre che visualizzate sull'apposita schermata di telemetria, sono riprodotte dal radiocomando come annunci vocali, consentendo di mantenere un maggiore controllo del tempo totale di volo senza la necessità di interrompere il contatto visivo con il drone per la visualizzazione dell'informazione sul display del radiocomando.

Lo script legge il tempo da un timer a scelta fra quelli messi a disposizione dal radiocomando, e consente di impostare un *tempo di allarme* (`Alert`) in secondi. Esso rappresenta l'intervallo di tempo che intercorre fra un annuncio vocale ed il seguente. Quando il timer raggiunge un valore in secondi multiplo del tempo di allarme, viene riprodotto un annuncio vocale che comunica il tempo attualmente trascorso dall'avvio del timer.

Lo script di cronometro vocale fa ampiamente uso delle funzioni `lcd` in precedenza presentate nella sezione 3.2.4 per sfruttare al meglio le capacità offerte dagli schermi dei due modelli.

4.1.1 Widget `TmrCnt` su RadioMaster TX16S

Lo script è il file `/WIDGETS/TmrCnt/main.lua`.

La parte iniziale dello script è composta dalle variabili locali, dalla table `options`, e dalle funzioni `create()` e `update()`:

```
local played = false

local options = {
      { "Color", COLOR, BLUE },
      { "Timer", VALUE, 1, 1, 3 }, -- Timer 1, 2 o 3
      { "Alert", VALUE, 30, 15, 180 } -- Numero di secondi fra ciascun
         annuncio
}

local function create (zone, options)
      local widget = { zone=zone, options=options }
      return widget
end

local function update(widget, options)
      if (widget ~= nil) then
            widget.options = options
      end
end
```

Ad eccezione della table `options`, l'unica variabile locale dello script è il flag di controllo per la riproduzione `played`, inizialmente con valore `false`.

Nella table delle opzioni sono contenuti tre elementi: `{ "Color", COLOR, BLUE }` che permette di impostare il colore del testo desiderato, `{ "Timer", VALUE, 1, 1, 3 }` che consente la selezione del timer del radiocomando e `{ "Alert", VALUE, 30, 15, 180 }` grazie al quale è possibile assegnare un intervallo personalizzato di secondi fra un annuncio ed il seguente ed il cui valore predefinito è 30.

Successivamente alle variabili dello script sono definite le funzioni `create()` ed `update()`. Esse si occupano, rispettivamente, di generare il widget e di aggiornare le opzioni successivamente alle modifiche effettuate dall'utente.

Altrettanto necessarie al funzionamento del widget sono le funzioni `refresh()` e `background()`, con le elaborazioni opportunamente suddivise nelle funzioni `playTimer()` e

`drawByWidgetSize()` le quali, rispettivamente, hanno il compito di gestire l'esecuzione degli annunci vocali e della stampa su schermo:

```
local function refresh(widget)
      local timer = model.getTimer(widget.options.Timer - 1)
      drawByWidgetSize(widget, timer)
      playTimer(timer, widget.options.Alert)
end
local function background(widget)
      local timer = model.getTimer(widget.options.Timer - 1)
      playTimer(timer, widget.options.Alert)
end
```

Sia la funzione `background()` che `refresh()` eseguono la funzione `playTimer()`. Viene qua riportata:

```
local function playTimer(timer, alert)
      if timer.value % alert == 1 then
            played = false
      end
      if timer.value % alert == 0 and timer.value ~= 0 then
            if played == false then
                  playDuration(timer.value, 0)
                  played = true
            end
      end
end
```

La funzione `playTimer()` riceve come argomenti un timer e il valore `alert`, a cui viene passato il valore `Alert` delle opzioni. Se il valore in secondi del timer è un multiplo di `alert` e contemporaneamente non è pari a 0, viene riprodotto l'annuncio vocale ed il flag `played` assume il valore `true`. Nelle iterazioni successive il flag impedisce all'annuncio vocale di essere riprodotto nuovamente: esso verrà ripristinato al valore `false` soltanto quando il timer passa al secondo successivo, in condizioni nelle quali l'annuncio non può più essere riprodotto. In assenza di un flag di controllo, l'annuncio vocale verrebbe riprodotto ad ogni nuova iterazione della funzione `refresh()` o `background()`, ottenendo la ripetizione indesiderata dell'annuncio vocale.

Le funzioni di stampa su schermo sono interamente contenute all'interno della funzione `drawBywidgetSize()`, la quale accetta come argomenti un widget ed un timer. Inizialmente è estratto il numero del timer nel formato `T%s`, ed il colore scelto nelle opzioni dello script viene assegnato agli elementi dello script quali testo e numeri mediante il metodo `lcd.setColor()`, indicando come primo argomento `CUSTOM_COLOR`:

```
local function drawByWidgetSize(widget, timer)
      local timerInfo = string.format("T%s:", widget.options.Timer)
      lcd.setColor(CUSTOM_COLOR, widget.options.Color)
```

Per la stampa vera e propria si adoperano i metodi `lcd.drawText()` ed `lcd.drawTimer()`. Ad essi vengono passati come argomenti la posizione orizzontale e verticale relativamente all'angolo in alto a sinistra del widget (avente coordinate `widget.zone.x` e `widget.zone.y`), l'informazione da stampare su schermo e flag relativi alla dimensione del testo e al colore. Questa sezione è stata realizzata in modo tale da distinguere le possibili dimensioni del widget supportate dal *RadioMaster TX16S* mediante delle istruzioni `if` che valutano la larghezza del widget (contenuta nella variabile `widget.zone.w`) e la sua altezza (contenuta in `widget.zone.h`). Vi sono in totale 5 diverse dimensioni per i widget del *RadioMaster TX16S*, ottenute empiricamente. In particolare:

- *widget molto grande*: per larghezze maggiori di 380 pixel e altezze maggiori di 165 pixel;
- *widget grande*: per larghezze maggiori di 180 pixel e altezze maggiori di 145 pixel;
- *widget medio*: per larghezze maggiori di 170 pixel e altezze maggiori di 65 pixel;
- *widget piccolo*: per larghezze maggiori di 150 pixel e altezze maggiori di 28 pixel;
- *widget nella barra superiore*: per larghezze maggiori di 65 pixel e altezze maggiori di 35 pixel.

Ad esempio, al *widget molto grande* corrisponde la seguente istruzione `if`:

```
      -- Widget a schermo intero
      if widget.zone.w > 380 and widget.zone.h > 165 then
            lcd.drawText(widget.zone.x, widget.zone.y, timerInfo,
               DBLSIZE + CUSTOM_COLOR)
            lcd.drawTimer(widget.zone.x + 100, widget.zone.y + 40,
               timer.value, XXLSIZE + CUSTOM_COLOR)
      end
```

e similmente vi saranno ulteriori `if` per tutte le altre dimensioni supportate.

Gli offset della posizione orizzontale e verticale dei vari elementi del widget, così come le dimensioni del testo, sono stati anch'essi determinati empiricamente, valutando la leggibilità e la corretta posizione mediante dei tentativi effettuati all'interno del simulatore *OpenTX Simulator*.

Al termine dello script, è collocato lo statement di `return`:

```
return { name="TmrCnt", options=options, create=create, update=update,
   background=background, refresh=refresh }
```

La Figura 4.1 mostra l'aspetto dello script durante la sua esecuzione.

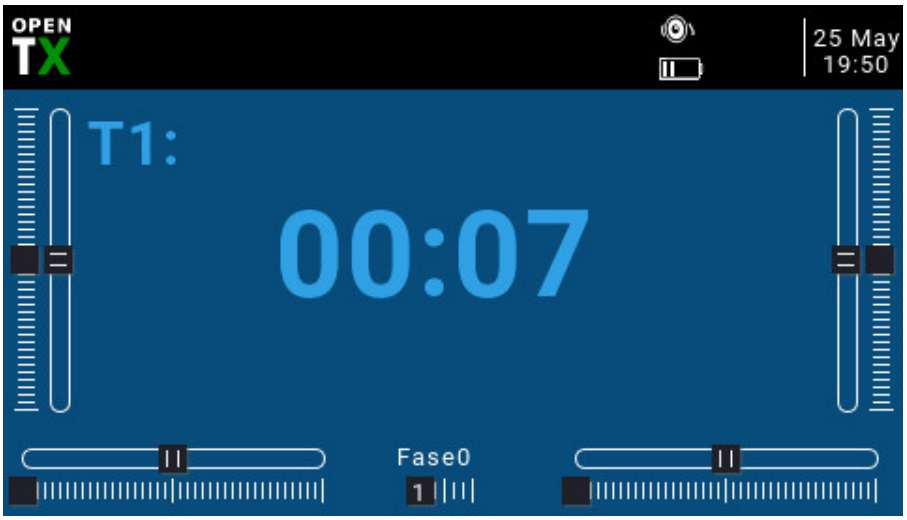

Figura 4.1: Aspetto della schermata del *RadioMaster TX16S* durante l'esecuzione dello script `TmrCnt`. La dimensione del widget è la massima disponibile.

4.1.2 Schermata di telemetria `TmrCnt` su Taranis QX7S

Lo script è il file `/SCRIPTS/TELEMETRY/TmrCnt.lua`.

Diversamente da quanto visto in precedenza per il *TX16S*, il *Taranis QX7S* non supporta i widget e pertanto non fornisce la possibilità di impostare le variabili mediante le opzioni direttamente da radiocomando. Al fine di garantire un'analoga capacità di personalizzazione, sono rese disponibili alcune variabili di configurazione all'inizio dello script, modificabili mediante un editor di testo:

```
local TimerNumber = 1
local Alert = 30
local InversColor = false
```

`TimerNumber` permette la selezione del timer del radiocomando, con possibili valori 1, 2 e 3, mentre l'opzione `"Alert"` presente nella versione per il *TX16S* è stata sostituita dalla variabile `Alert`. La variabile booleana `InversColor`, invece, consente di abilitare l'inversione dei colori su schermo.

Successivamente a tale sezione sono collocate le variabili interne dello script, la cui modifica da parte dell'utente non è prevista poiché potrebbe alterare il funzionamento dello script. Le variabili sono `Timer`, che contiene il timer, `TimerInfo` che contiene la stringa identificativa del timer e il flag `Played`:

```
local Timer
local TimerInfo
local Played
```

Le funzioni principali, `init()`, `run()` e `background()` si occupano, rispettivamente, della configurazione iniziale, della stampa su schermo quando la schermata è visibile, e della riproduzione degli annunci a prescindere dalla visibilità della schermata.

```
local function init()
      Timer = model.getTimer(TimerNumber - 1)
      TimerInfo = string.format("T%s:", TimerNumber)
      Played = false
end

local function refresh()
      drawTelemetry()
end

local function background()
      Timer = model.getTimer(TimerNumber - 1)
      playTimer()
end
```

Le elaborazioni interne sono ancora una volta suddivise in ulteriori funzioni. La funzione `playTimer()` è del tutto analoga a quella per il *TX16S*, e presenta il medesimo algoritmo con flag `Played` per il controllo della riproduzione degli annunci vocali.

In particolare:

```
local function playTimer()
      if Timer.value % Alert == 1 then
            Played = false
      end

      if Timer.value % Alert == 0 and Timer.value ~= 0 then
            if Played == false then
                  playDuration(Timer.value, 0)
                  Played = true
            end
      end
end
```

La stampa su schermo è invece differente, sia poiché è assente il concetto di widget con le varie dimensioni supportate, sia per la presenza di alcune funzioni indispensabili al funzionamento degli script di Telemetria.

```
local function drawTelemetry()
      lcd.clear() -- obbligatorio negli script di Telemetria
      lcd.drawScreenTitle("Timer Counter", 0, 0)
      if InversColor == true then
            lcd.drawFilledRectangle(0, 0, 128, 64)
            lcd.drawText(0, 9, TimerInfo, INVERS)
            lcd.drawTimer(13, 18, Timer.value, XXLSIZE + INVERS)
      else
            lcd.drawText(0, 9, TimerInfo, BIGSIZE)
            lcd.drawTimer(13, 18, Timer.value, XXLSIZE)
      end
end
```

Un'istruzione `if` consente inoltre di determinare se l'utente desidera una stampa a colori invertiti: in tal caso, viene stampato su schermo un opportuno rettangolo di colore scuro, e agli stessi comandi è aggiunto il flag `INVERS` in modo tale da riprodurre le stringhe con caratteri chiari su sfondo scuro.

L'istruzione di return è collocata alla fine dello script:

```
return { run=refresh, background=background, init=init }
```

L'aspetto originale e la versione a colori invertiti sono mostrati, rispettivamente, in Figura 4.2 ed in Figura 4.3.

Figura 4.2: Aspetto della schermata di telemetria del *Taranis QX7S* durante l'esecuzione dello script `TmrCnt`.

Figura 4.3: Aspetto a colori invertiti della schermata di telemetria del *Taranis QX7S* durante l'esecuzione dello script `TmrCnt`.

4.2 Lo script di conto alla rovescia vocale

L'obiettivo dello *script di conto alla rovescia vocale* è stato quello di fornire un conto alla rovescia personalizzabile, garantendo al pilota in tempo reale un'indicazione acustica del tempo residuo. Le informazioni del conto alla rovescia sono riprodotte dal radiocomando come annunci vocali, consentendo di mantenere un maggiore controllo del tempo rimanente senza la necessità di interrompere il contatto visivo con il drone per la visualizzazione dell'informazione sul display del radiocomando.

Lo script legge il tempo da un timer del radiocomando, e consente di impostare un *tempo di partenza* (`Start`) in secondi. Esso rappresenta il numero iniziale di secondi del conto alla rovescia. Quando il timer raggiunge un multiplo del minuto, o un sottomultiplo del minuto se il tempo rimanente è inferiore a 60 secondi, viene riprodotto un annuncio vocale contenente il tempo residuo.

Lo script di conto alla rovescia vocale fa ampiamente uso delle funzioni `lcd` in precedenza presentate nella sezione 3.2.4 per sfruttare al meglio le capacità offerte dagli schermi dei due modelli.

4.2.1 Widget `CntDwn` su RadioMaster TX16S

Lo script è il file `/WIDGETS/CntDwn/main.lua`.

La parte iniziale dello script è composta dalla definizione delle opzioni, dalla dichiarazione di alcune variabili locali e dalle funzioni `create()` ed `update()`:

```
local options = {
      { "Color", COLOR, BLUE },
      { "Timer", VALUE, 1, 1, 3 },
      { "Start", VALUE, 180, 30, 600 }
}

local Total       -- numero totale di secondi
local NextSecond = 11 -- variabile di controllo per gli ultimi 10 secondi
local Played      -- flag di controllo
local Elapsed     -- flag che controlla il termine del conto alla
   rovescia

local function create (zone, options)
      local widget = { zone=zone, options=options }
      Total = widget.options.Start
      Played = false
      Elapsed = false
      return widget
end

local function update(widget, options)
      if (widget ~= nil) then
            widget.options = options
```

```
        end
        Total = widget.options.Start
        Played = false
        Elapsed = false
end
```

La struttura ricalca quella dello script `TmrCnt`, e differenze degne di nota sono la presenza del valore di inizio del cronometro, nelle opzioni `"Start"`, e la presenza di più flag, con aggiornamento dei flag `Played` ed `Elapsed` all'interno della funzione `update()` per riportare il cronometro alle nuove condizioni iniziali al momento della modifica delle opzioni da parte del pilota.

All'interno delle due funzioni principali, `refresh()` e `background()`, viene calcolato il numero di secondi rimanente allo scadere del conto alla rovescia, `countdown`, e successivamente tale valore è stampato su schermo nel caso della funzione `refresh()` e riprodotto acusticamente nel caso di ambedue le funzioni.

Si osserva che le due funzioni svolgono le stesse elaborazioni, eccezion fatta per l'assenza di interazione con lo schermo nella funzione `background()`:

```
local function refresh(widget)
        local timer = model.getTimer(widget.options.Timer - 1)
        local countdown

        if Total - timer.value > 0 then
                countdown = Total - timer.value
        else
                countdown = 0
        end

        drawByWidgetSize(widget, countdown)

        if Elapsed == false then
                playTime(countdown)
        end
end
```

```
local function background(widget)
      local timer = model.getTimer(widget.options.Timer - 1)
      local countdown

      if Total - timer.value > 0 then
            countdown = Total - timer.value
      else
            countdown = 0
      end

      if Elapsed == false then
            playTime(countdown)
      end
end
```

La funzione di stampa su schermo ha funzioni analoghe a quanto osservato per lo script `TmrCnt`: la dimensione e la posizione del testo viene selezionata in base alle dimensioni del widget, e successivamente si utilizza la funzione `lcd.drawNumber()` con gli opportuni argomenti per la stampa del contatore il cui valore è contenuto nella variabile `countdown`.

Differenze sostanziali si trovano nella riproduzione degli annunci vocali. È stato scelto di riprodurre annunci ogni 60 secondi, mentre durante l'ultimo minuto viene comunicato un avviso ai secondi 30, 15, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1. Allo scadere del conto alla rovescia viene riprodotto l'annuncio finale.

L'utilizzo del flag `Played` ha consentito ancora una volta di evitare le riproduzioni multiple che si sarebbero altrimenti verificate.

Allo scadere del tempo, il flag `Elapsed` al valore `true` impedisce la chiamata alla funzione `playTime()`, prevenendo elaborazioni superflue.

Viene riportata la funzione:

```
local function playTime(countdown)

      local seconds = countdown % 60
      local minutes = (countdown - seconds) / 60

      if Played == true then
```

```
            resetPlayedFlag(countdown, NextSecond)
      end

      if Played == false then
            if countdown > 30 then
                  if countdown % 60 == 0 and countdown ~= Total then
                        if minutes ~= 0 and seconds == 0 then
                              playNumber(minutes, 36)
                        else
                              playNumber(minutes, 36)
                              playNumber(seconds, 37)
                        end
                        Played = true
                  end
            elseif countdown == 30 then
                  playNumber(countdown, 37)
                  Played = true
            elseif countdown == 15 then
                  playNumber(countdown, 37)
                  Played = true
            elseif countdown <= 10 and countdown > 0 then
                  playNumber(countdown, UNIT_RAW)
                  Played = true
                  NextSecond = countdown
            elseif countdown == 0 then
                  playFile("/WIDGETS/Cntdwn/timelpsd.wav")
                  Played = true
                  Elapsed = true
            end
      end
end
```

A causa della maggiore complessità nel determinare le condizioni per il ripristino del flag `Played` al valore `false`, tale compito è affidato ad un'ulteriore funzione, `resetPlayedFlag()`, la quale si occupa di stabilire le condizioni opportune per la modifica del flag:

```
local function resetPlayedFlag(countdown, NextSecond)
      if countdown % 60 == 59 then
            Played = false
      elseif countdown == 29 or countdown == 14 then
            Played = false
      elseif countdown <= 10 and Elapsed ~= true and countdown ~=
         NextSecond then
            Played = false
      end
end
```

La variabile `NextSecond` consente di stabilire, per gli ultimi 10 secondi, un criterio per la reimpostazione del flag `Played` al valore `false`. Infatti, `NextSecond` conterrà il valore di `countdown` negli ultimi 10 secondi, ed il ripristino del flag ha luogo soltanto se queste due variabili hanno valore differente, condizione che può verificarsi solo immediatamente dopo il decremento di `countdown`.

Lo script si presenta come in Figura 4.4.

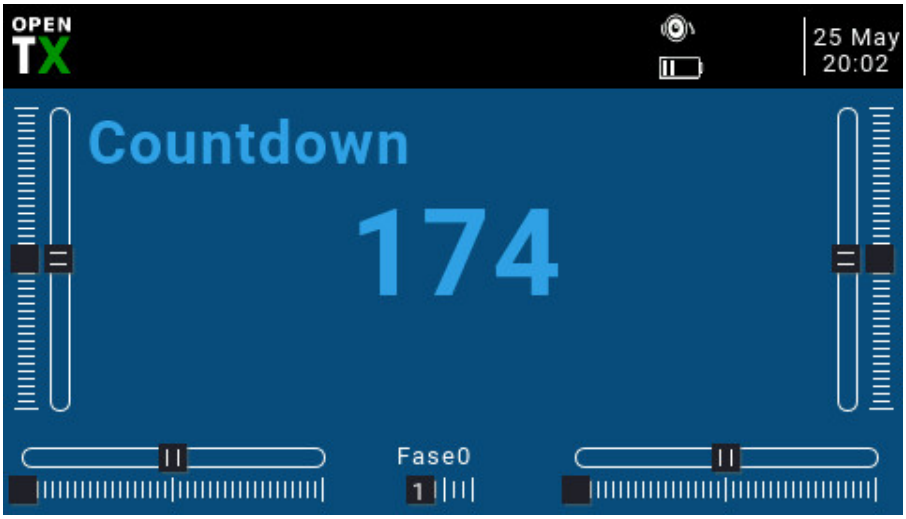

Figura 4.4: Aspetto della schermata del *RadioMaster TX16S* durante l'esecuzione dello script `CntDwn`. La dimensione del widget è la massima disponibile.

4.2.2 Schermata di telemetria `CntDwn` su Taranis QX7S

La struttura per lo script di conto alla rovescia vocale nel *QX7S* presenta poche differenze rispetto alla versione per il *TX16S*.

In sostituzione delle opzioni è fornita una sezione dello script contenente le variabili di configurazione, per cui è possibile configurarne manualmente il valore mediante un editor di testo. Esse sono il numero del timer `TimerNumber`, il totale dei secondi del conto alla rovescia `CountdownTotalSeconds` e l'inversione dei colori `InversColor`:

```
local TimerNumber = 1
local CountdownTotalSeconds = 180
local InversColor = false
```

Le variabili interne dello script sono dichiarate nella sezione immediatamente successiva, e fra di esse, oltre alle medesime variabili della versione per il *TX16S*, troviamo `Timer` e `TimerInfo` che conterranno, rispettivamente, il timer del radiocomando e le relative informazioni.

La gestione della riproduzione degli annunci non presenta nessuna variazione degna di nota rispetto alla versione per il *TX16S*: ancora una volta l'elaborazione è stata suddivisa nelle due funzioni `playTime()` e `resetPlayedFlag()` le quali svolgono le stesse computazioni della variante per il *RadioMaster*.

Come già osservato per lo script `TmrCnt`, a presentare alcune differenze è la funzione di stampa su schermo, che in questa variante prende il nome di `drawTelemetry()`. Nuovamente sono assenti il concetto di widget ed i relativi `if` necessari all'identificazione della dimensione correntemente selezionata dall'utente nell'interfaccia del radiocomando: al loro posto vi è la stampa delle informazioni e la possibilità di invertire i colori modificando il valore della variabile di configurazione `InversColor`,

```
local function drawTelemetry()
      lcd.clear()
      lcd.drawScreenTitle("Countdown Timer", 0, 0)
      if InversColor == true then
            lcd.drawFilledRectangle(0, 0, 128, 64)
            lcd.drawText(0, 9, TimerInfo, INVERS)
            lcd.drawNumber(3, 18, Countdown, XXLSIZE + INVERS)
      else
            lcd.drawText(0, 9, TimerInfo, BIGSIZE)
            lcd.drawNumber(3, 18, Countdown, XXLSIZE)
      end
end
```

Le due funzioni principali, `refresh()` e `background()` svolgono le medesime elaborazioni della variante per il *TX16S*.

L'aspetto originale e la versione a colori invertiti sono mostrati, rispettivamente, in Figura 4.5 ed in Figura 4.6.

Figura 4.5: Aspetto della schermata di telemetria del *Taranis QX7S* durante l'esecuzione dello script `CntDwn`.

Figura 4.6: Aspetto a colori invertiti della schermata di telemetria del *Taranis QX7S* durante l'esecuzione dello script `CntDwn`.

5 CONCLUSIONI

OpenTX mette a disposizione agli sviluppatori software vari strumenti e applicativi di notevole utilità. Lo studio della documentazione, l'apprendimento di *OpenTX Companion* e la fase di test dei due script, avvenuta integralmente tramite il simulatore *OpenTX Simulator*, ha infine portato alla realizzazione di due script di telemetria, in grado di svolgere compiti ed elaborazioni personalizzate.

L'integrazione con il linguaggio di scripting Lua consente allo sviluppatore di potenziare ulteriormente le già ampie possibilità offerte dal firmware all'interno del radiocomando, in particolare dotando i vari modelli di trasmittente di ampie capacità di calcolo e di elaborazione.

La documentazione del progetto, esauriente ed aggiornata, ha permesso uno studio approfondito e dettagliato della materia, con particolare attenzione alla parte relativa al supporto degli script in Lua.

Il software principale per computer desktop, *OpenTX Companion*, ha reso possibile la gestione avanzata e da interfaccia grafica delle impostazioni del radiocomando, della loro personalizzazione e del loro salvataggio anche in assenza di un radiocomando reale. Il secondo programma, il simulatore, ha poi consentito il test delle impostazioni stesse del radiocomando e, con sufficiente realismo, anche degli script in Lua realizzati in questo elaborato di tesi sperimentale.

Al termine della loro stesura e sperimentazione mediante il simulatore, i programmi sono stati sperimentati dal vivo sui due radiocomandi reali, il *Taranis QX7S* ed il *RadioMaster TX16S*. La prova ha dato esito positivo per ambedue gli script, su entrambi i modelli, e sia lo script di cronometro vocale (`TmrCnt`) che lo script di conto alla rovescia (`CntDwn`) hanno manifestato un comportamento idoneo e nel pieno rispetto di quanto era stato inizialmente progettato e testato nel simulatore. Il risultato ha risposto alle aspettative, ed è stato dunque del tutto soddisfacente.

6 SVILUPPI FUTURI

Gli sviluppi futuri potrebbero essere rivolti alla creazione di un maggiormente complesso sistema di telemetria. È infatti possibile ottenere ed elaborare ulteriori dati fra cui l'*RSSI* (Received Signal Strength Indication), la lettura della tensione della batteria del radiocomando, la lettura della tensione della batteria del modello. In particolare, l'RSSI è ottenibile mediante la funzione `getRSSI()`, la quale restituirà un valore intero fra 0 e 99 che indica l'intensità del segnale in ricezione, mentre la lettura della tensione avviene prelevando il dato da una sorgente (`SOURCE`) come definita nella sezione 3.2.1. Il nome della sorgente può presentare variazioni a seconda del radiocomando o del modello adottato. Nei due radiocomandi in esame essa prende il nome di `tx-voltage`, mentre possibili nomi di sorgenti per la tensione di batteria del modello sono `RxBt` oppure `VFAS`. All'ottenimento della tensione del modello potrebbero seguire ulteriori elaborazioni, con particolare attenzione alle problematiche relative alla stima della batteria residua o all'individuazione di eventuali malfunzionamenti.

D'importanza cruciale sarebbe la corretta gestione dei relativi annunci vocali, al fine di sollevare il pilota della responsabilità di mantenere un contatto visivo con lo schermo del radiocomando.

Uno sviluppo più avanzato porterebbe infine al progetto e alla gestione di un complesso sistema di telemetria che presenti coerentemente e su un'unica schermata tutti i dati necessari al pilota (cronometro vocale, tensioni di batterie, coordinate GPS, RSSI e così via), sia come numeri filtrati da opportune elaborazioni con relative stringhe di testo, che in qualità di annunci vocali.

Bibliografia

[1] OpenTX, https://www.open-tx.org/, 2014.

[2] OpenTX, https://www.open-tx.org/radios, 2014.

[3] Repository OpenTX su GitHub, https://github.com/opentx/opentx, 2021.

[4] Burdziakowski, P., Razmjooy, N., Estrela, V., & Hemanath, J. (2020). *"Open-source software (OSS) and hardware (OSH) in UAVs"*. 49-66

[5] ArduPilot Dev Team, https://ardupilot.org/planner/, 2021.

[6] ArduPilot, https://ardupilot.org/, 2016.

[7] OpenDroneMap Authors ODM, *"A command line toolkit to generate maps, point clouds, 3D models and DEMs from drone, balloon or kite images."* https://github.com/OpenDroneMap/ODM, 2020.

[8] UAV4GEO, https://webodm.net/.

[9] Cummings, Anthony R.; McKee, Arlo; Kulkarni, Keyur; Markandey, Nakul, *"The Rise of UAVs"*, Photogrammetric Engineering & Remote Sensing, Volume 83, Number 4, April 2017, pp. 317-325(9)

[10] Andrea Cardamone, *"Implementation of a pilot in the loop simulation environment for UAV development and testing"*, http://hdl.handle.net/10589/135202 2017.

[11] OpenTXU, http://open-txu.org/home/undergraduate-courses/introduction/how-rc-works/, 2014.

[12] OpenTX, https://www.open-tx.org/lua-instructions.html, 2014.

[13] Lua Project, https://www.lua.org/, 2020.

[14] OpenTX, https://doc.open-tx.org/manual-for-opentx-2-2/companion.

[15] Fedora Project, https://getfedora.org/, 2021.

[16] OpenTX, https://www.open-tx.org/downloads, 2014.

Appendice

Script `TmrCnt` per RadioMaster TX16S

```
-- Timer Counter Widget
-- Nome: TmrCnt
--
-- File: "main.lua"
-- Da collocare in "/WIDGETS/TmrCnt/"

local played = false

local options = {
      { "Color", COLOR, BLUE },
      { "Timer", VALUE, 1, 1, 3 }, -- Timer 1, 2 o 3
      { "Alert", VALUE, 30, 15, 180 } -- Numero di secondi fra ciascun
         annuncio
}

local function create (zone, options)
      local widget = { zone=zone, options=options }
      return widget
end

local function update(widget, options)
      if (widget ~= nil) then
            widget.options = options
      end
end
```

```
local function drawByWidgetSize(widget, timer)
      local timerInfo = string.format("T%s:", widget.options.Timer)
      lcd.setColor(CUSTOM_COLOR, widget.options.Color)

      -- Widget a schermo intero
      if widget.zone.w > 380 and widget.zone.h > 165 then
            lcd.drawText(widget.zone.x, widget.zone.y, timerInfo,
               DBLSIZE + CUSTOM_COLOR)
            lcd.drawTimer(widget.zone.x + 100, widget.zone.y + 40,
               timer.value, XXLSIZE + CUSTOM_COLOR)

      -- Widget Grande
      elseif widget.zone.w > 180 and widget.zone.h > 145 then
            lcd.drawText(widget.zone.x, widget.zone.y, timerInfo,
               MIDSIZE + CUSTOM_COLOR)
            lcd.drawTimer(widget.zone.x + 13, widget.zone.y + 25,
               timer.value, XXLSIZE + CUSTOM_COLOR)

      -- Widget Medio
      elseif widget.zone.w > 170 and widget.zone.h > 65 then
            lcd.drawText(widget.zone.x, widget.zone.y, timerInfo,
               SMLSIZE + CUSTOM_COLOR)
            lcd.drawTimer(widget.zone.x + 3, widget.zone.y + 5,
               timer.value, XXLSIZE + CUSTOM_COLOR)

      --Widget Piccolo
      elseif widget.zone.w > 150 and widget.zone.h > 28 then
            lcd.drawText(widget.zone.x, widget.zone.y + 7, timerInfo,
               SMLSIZE + CUSTOM_COLOR)
            lcd.drawTimer(widget.zone.x + 70, widget.zone.y + 1,
               timer.value, MIDSIZE + CUSTOM_COLOR)

      -- Widget nella barra superiore
      elseif widget.zone.w > 65 and widget.zone.h > 35 then
```

```
            lcd.drawText(widget.zone.x, widget.zone.y, timerInfo,
               SMLSIZE + CUSTOM_COLOR)
            lcd.drawTimer(widget.zone.x, widget.zone.y + 10,
               timer.value, MIDSIZE + CUSTOM_COLOR)
      else
            lcd.drawText(widget.zone.x, widget.zone.y, "Insufficient
               space", SMLSIZE + CUSTOM_COLOR)
      end
end

local function playTimer(timer, alert)
      if timer.value % alert == 1 then
            played = false
      end

      if timer.value % alert == 0 and timer.value ~= 0 then
            if played == false then
                  playDuration(timer.value, 0)
                  played = true
            end
      end
end

local function refresh(widget)
      local timer = model.getTimer(widget.options.Timer - 1)
      drawByWidgetSize(widget, timer)
      playTimer(timer, widget.options.Alert)
end

local function background(widget)
      local timer = model.getTimer(widget.options.Timer - 1)
      playTimer(timer, widget.options.Alert)
end
```

```
return { name="TmrCnt", options=options, create=create, update=update,
   background=background, refresh=refresh }
```

Script `TmrCnt` per Taranis QX7S

```
-- File TmrCnt.lua
--
-- Script "Timer Counter"
--
-- Da collocare in "/SCRIPTS/TELEMETRY/"
--
-- impostazioni e variabili modificabili
local TimerNumber = 1
local Alert = 30
local InversColor = false

-- variabili di script
-- Non modificare al di sotto di questa riga
local Timer
local TimerInfo
local Played

-- funzioni dello script
local function playTimer()
      if Timer.value % Alert == 1 then
            Played = false
      end

      if Timer.value % Alert == 0 and Timer.value ~= 0 then
            if Played == false then
                  playDuration(Timer.value, 0)
                  Played = true
            end
      end
end

local function drawTelemetry()
      lcd.clear()
```

```
      lcd.drawScreenTitle("Timer Counter", 0, 0)
      if InversColor == true then
            lcd.drawFilledRectangle(0, 0, 128, 64)
            lcd.drawText(0, 9, TimerInfo, INVERS)
            lcd.drawTimer(13, 18, Timer.value, XXLSIZE + INVERS)
      else
            lcd.drawText(0, 9, TimerInfo, BIGSIZE)
            lcd.drawTimer(13, 18, Timer.value, XXLSIZE)
      end
end

-- funzioni principali
local function init()
      Timer = model.getTimer(TimerNumber - 1)
      TimerInfo = string.format("T%s:", TimerNumber)
      Played = false
end

local function refresh()
      drawTelemetry()
end

local function background()
      Timer = model.getTimer(TimerNumber - 1)
      playTimer()
end

-- istruzione di return
return { run=refresh, background=background, init=init }
```

Script `CntDwn` per RadioMaster TX16S

```
-- File main.lua
-- Script "Countdown Timer"
--
-- Nome: "Cntdwn"
-- Da collocare in "/WIDGETS/Cntdwn/"

local options = {
      { "Color", COLOR, BLUE },
      { "Timer", VALUE, 1, 1, 3 },
      { "Start", VALUE, 180, 30, 600 }
}

local Total
local NextSecond = 11
local Played
local Elapsed

local function create (zone, options)
      local widget = { zone=zone, options=options }
      Total = widget.options.Start
      Played = false
      Elapsed = false
      return widget
end

local function update(widget, options)
      if (widget ~= nil) then
            widget.options = options
      end
      Total = widget.options.Start
      Played = false
      Elapsed = false
end
```

```
local function drawByWidgetSize(widget, countdown)
      lcd.setColor(CUSTOM_COLOR, widget.options.Color)

      -- Widget a schermo intero
      if widget.zone.w > 380 and widget.zone.h > 165 then
            lcd.drawText(widget.zone.x, widget.zone.y, "Countdown",
               DBLSIZE + CUSTOM_COLOR)
            lcd.drawNumber(widget.zone.x + 140, widget.zone.y + 40,
               countdown, XXLSIZE + CUSTOM_COLOR)

      -- Widget Grande
      elseif widget.zone.w > 180 and widget.zone.h > 145 then
            lcd.drawText(widget.zone.x, widget.zone.y, "Countdown",
               MIDSIZE + CUSTOM_COLOR)
            lcd.drawNumber(widget.zone.x + 13, widget.zone.y + 25,
               countdown, XXLSIZE + CUSTOM_COLOR)

      -- Widget Medio
      elseif widget.zone.w > 170 and widget.zone.h > 65 then
            lcd.drawText(widget.zone.x, widget.zone.y, "Countdown",
               SMLSIZE + CUSTOM_COLOR)
            lcd.drawNumber(widget.zone.x + 3, widget.zone.y + 5,
               countdown, XXLSIZE + CUSTOM_COLOR)

      -- Widget Piccolo
      elseif widget.zone.w > 150 and widget.zone.h > 28 then
            lcd.drawText(widget.zone.x, widget.zone.y + 7, "Countdown",
               SMLSIZE + CUSTOM_COLOR)
            lcd.drawNumber(widget.zone.x + 90, widget.zone.y + 1,
               countdown, MIDSIZE + CUSTOM_COLOR)

      -- Widget della barra superiore
      elseif widget.zone.w > 65 and widget.zone.h > 35 then
```

```
            lcd.drawText(widget.zone.x, widget.zone.y, "Countdown",
               SMLSIZE + CUSTOM_COLOR)
            lcd.drawNumber(widget.zone.x, widget.zone.y + 10, countdown,
               MIDSIZE + CUSTOM_COLOR)
      else
            lcd.drawText(widget.zone.x, widget.zone.y, "Insufficient
               space", SMLSIZE + CUSTOM_COLOR)
      end
end

local function resetPlayedFlag(countdown, NextSecond)
      if countdown % 60 == 59 then
            Played = false
      elseif countdown == 29 or countdown == 14 then
            Played = false
      elseif countdown <= 10 and Elapsed ~= true and countdown ~=
         NextSecond then
            Played = false
      end
end

local function playTime(countdown)
      local seconds = countdown % 60
      local minutes = (countdown - seconds) / 60

      if Played == true then
            resetPlayedFlag(countdown, NextSecond)
      end

      if Played == false then
            if countdown > 30 then
                  -- viene riprodotto ogni minuto
                  if countdown % 60 == 0 and countdown ~= Total then
                        if minutes ~= 0 and seconds == 0 then
                              playNumber(minutes, 36)
```

```
                        else
                              playNumber(minutes, 36)
                              playNumber(seconds, 37)
                        end
                        Played = true
                  end
            -- riproduce gli ultimi 30 secondi
            elseif countdown == 30 then
                  playNumber(countdown, 37)
                  Played = true
            elseif countdown == 15 then
                  playNumber(countdown, 37)
                  Played = true
            -- riproduce gli ultimi 10 secondi
            elseif countdown <= 10 and countdown > 0 then
                  playNumber(countdown, UNIT_RAW)
                  Played = true
                  NextSecond = countdown
            -- il conto alla rovescia e' terminato
            elseif countdown == 0 then
                  playFile("/WIDGETS/Cntdwn/timelpsd.wav")
                  Played = true
                  Elapsed = true
            end
      end
end

local function refresh(widget)
      local timer = model.getTimer(widget.options.Timer - 1)
      local countdown

      if Total - timer.value > 0 then
            countdown = Total - timer.value
      else
            countdown = 0
```

```
      end

      drawByWidgetSize(widget, countdown)

      if Elapsed == false then
            playTime(countdown)
      end
end

local function background(widget)
      local timer = model.getTimer(widget.options.Timer - 1)
      local countdown

      if Total - timer.value > 0 then
            countdown = Total - timer.value
      else
            countdown = 0
      end

      if Elapsed == false then
            playTime(countdown)
      end
end

return { name="Cntdwn", options=options, create=create, update=update,
   background=background, refresh=refresh }
```

Script `CntDwn` per Taranis QX7S

```
-- File Cntdwn.lua
--
-- Script "Countdown Timer"
--
-- Da collocare in "/SCRIPTS/TELEMETRY/"
--
-- variabili modificabili
local TimerNumber = 1
local CountdownTotalSeconds = 180
local InversColor = false
--
-- variabili di script
local Timer
local TimerInfo
local Countdown
local NextSecond
local Played
local Elapsed
-- funzioni di script
local function resetPlayedFlag()
      if Countdown % 60 == 59 then
            Played = false
      elseif Countdown == 29 or Countdown == 14 then
            Played = false
      elseif Countdown <= 10 and Countdown >= 0 and Countdown ~=
         NextSecond then
            Played = false
      end
end

local function playTime()

      local Seconds = Countdown % 60
```

```
local Minutes = (Countdown - Seconds) / 60

if Played == true then
      resetPlayedFlag(Countdown, NextSecond)
end

if Played == false then
      if Countdown > 30 then
            if Countdown % 60 == 0 and Countdown ~=
               CountdownTotalSeconds then
                  if Minutes ~= 0 and Seconds == 0 then
                        playNumber(Minutes, 36)
                  elseif Minutes == 0 and Seconds ~= 0 then
                        playNumber(Seconds, 37)
                  else
                        playNumber(Minutes, 36)
                        playNumber(Seconds, 37)
                  end
                  Played = true
            end
      elseif Countdown == 30 then
            playNumber(Countdown, 37)
            Played = true
      elseif Countdown == 15 then
            playNumber(Countdown, 37)
            Played = true
      elseif Countdown <= 10 and Countdown > 0 then
            playNumber(Countdown, 0)
            Played = true
            NextSecond = Countdown
      elseif Countdown == 0 then
            playFile("/SCRIPTS/TELEMETRY/timelpsd.wav")
            Played = true
            Elapsed = true
      end
```

```
	end
end

local function drawTelemetry()
	lcd.clear()
	lcd.drawScreenTitle("Countdown Timer", 0, 0)
	if InversColor == true then
		lcd.drawFilledRectangle(0, 0, 128, 64)
		lcd.drawText(0, 9, TimerInfo, INVERS)
		lcd.drawNumber(3, 18, Countdown, XXLSIZE + INVERS)
	else
		lcd.drawText(0, 9, TimerInfo, BIGSIZE)
		lcd.drawNumber(3, 18, Countdown, XXLSIZE)
	end
end

-- funzioni principali
local function init()
	Timer = model.getTimer(TimerNumber - 1)
	TimerInfo = string.format("T%s:", TimerNumber)
	Countdown = CountdownTotalSeconds
	NextSecond = 11
	Played = false
	Elapsed = false
end

local function refresh()
	drawTelemetry()
end

local function background()
	Timer = model.getTimer(TimerNumber - 1)
	Countdown = CountdownTotalSeconds - Timer.value
	if Countdown < 0 then
		Countdown = 0
```

```
	end

	if Elapsed == false then
		playTime()
	end
end

-- istruzione di return
return { run=refresh, background=background, init=init }
```