I’ve Got the Power: OpenTX

An open source operating system with the Lua progamming language.

Michael Shellim

Some OpenTX-compatible transmitters. Left to right: FrSky TX12S, X9D, X9 Lite, RadioMaster TX12.

Few topics generate more noise and hot air than OpenTX, the popular open source operating system. Some love it. Others hate it, and for the rest OpenTX remains a mystery. In this article I’m going to explain the background to OpenTX, as well as its unique approach to programming.

I will also show how, with the aid of the built in Lua facility, you can do some pretty advanced stuff — like a continuously variable trim system!

Humble Beginnings

OpenTX’s popularity is remarkable when you consider it started life as an obscure operating system for cheap Chinese transmitters. It finally hit the big time in 2013, when FrSky chose it for their first transmitter, the Taranis X9D. Thanks in large part to OpenTX, the X9D became a huge hit.

Part of the appeal of OpenTX is the free Companion software. This provides a model editor, and a simulator for testing your setups. It's also a great way to familiarise with the OpenTX before purchasing a transmitter.

OpenTX also incorporates a Lua interpreter. This allows users to create powerful extensions in the form of *scripts* — we'll look at an example later in this article.

Designing a Setup

As with any serious pursuit, knowing the lingo is key. Architects use lines and curves. Mathematicians use symbols. And OpenTX'ers use *interactions*. An interaction expresses the relationship between one stick and one servo.

Here's how you might write an interaction representing a simple elevator function.

```
Elevator stick → CH3/elevator (pitch)
```

The first step in designing a setup is simply to make a list of all the interactions required. Here's an example, for a 4-servo glider with twin aileron servos. Note that the aileron stick generates two interactions, one for each servo:

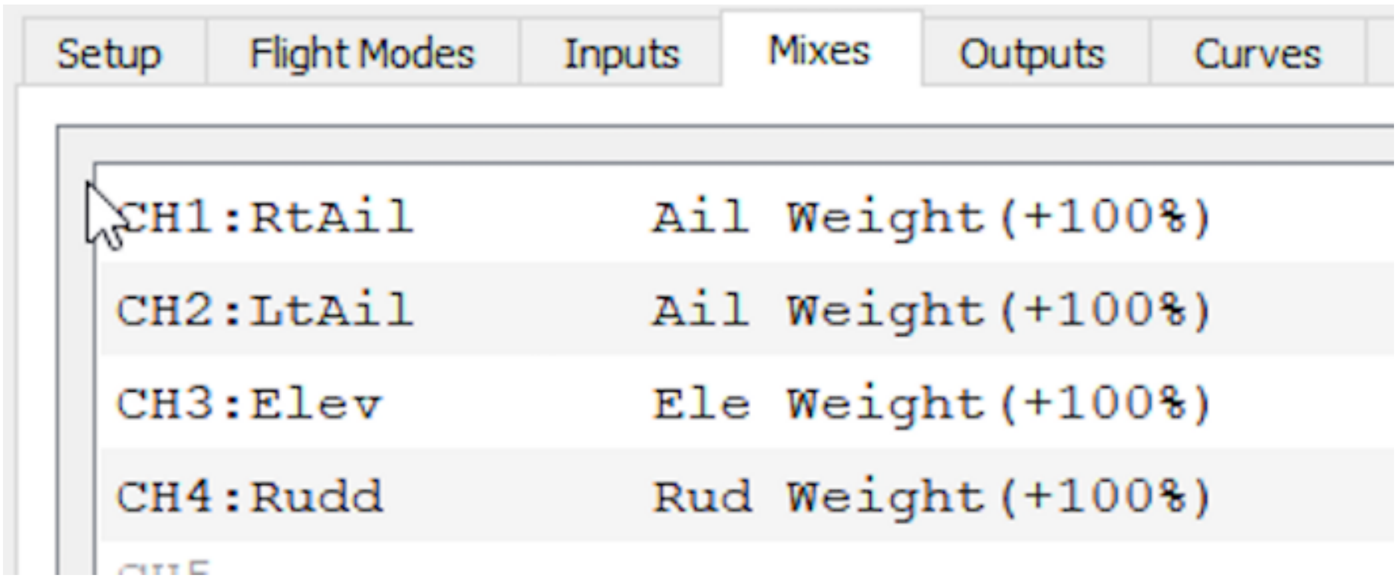
```
Aileron stick → CH1/RtAil (roll)
Aileron stick → CH2/LtAil (roll)
Elevator stick → CH3/Elev (pitch)
Rudder stick → CH4/Rudd (yaw)
```

The beauty of this approach is its simplicity: this four-line table completely describes the basic operation of the model.

Implementing the Design

While the design is concerned with interactions, the implementation is all about *mixers*. And a mixer in OpenTX is very simple — it takes an input, applies a transformation, and assigns the result to a channel.

The similarity with interactions is obvious, but it gets better... to implement your design, simply reverse the interactions and enter them into the Mixes menu:



```
CH1:RtAil      Ail Weight(+100%)
CH2:LtAil      Ail Weight(+100%)
CH3:Elev       Ele Weight(+100%)
CH4:Rudd       Rud Weight(+100%)
```

Figure 1: Mixers for a 4-servo glider, shown in OpenTX Companion software.

Want to add snapflap? No problem — just add two interactions:

```
Elevator stick → CH1/RtAil (snapflap)
Elevator stick → CH2/LtAil (snapflap)
```

... and again, those translate into a couple of extra mixes.

The 1:1 relationship between interactions and mixes makes it easy to design a setup. And because your work can be expressed as a list, it's easy to browse and document.

The Beauty of Lists

Mixers are just one aspect of OpenTX. Others elements include:

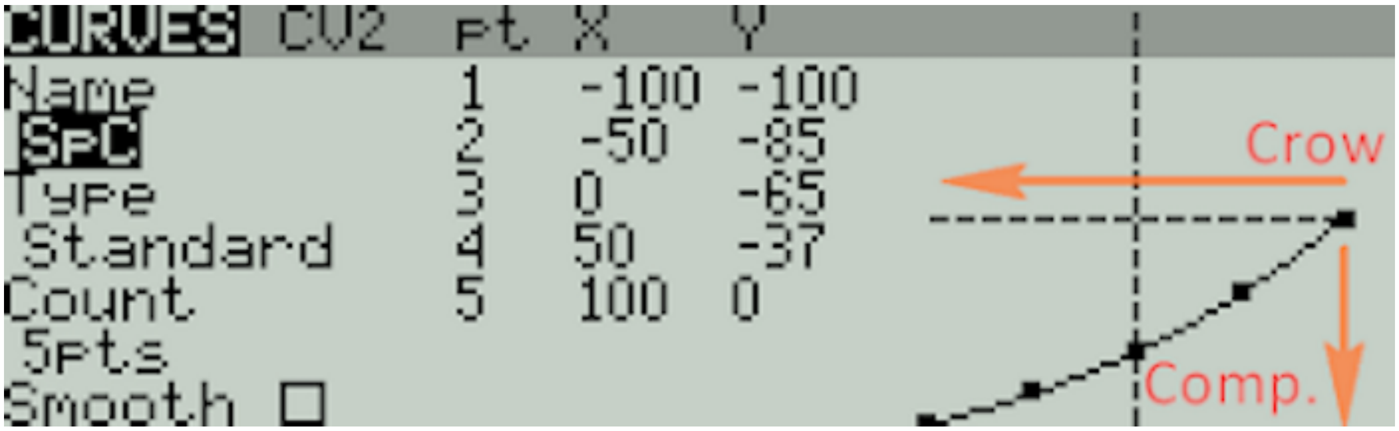
- *Inputs* — flight controls with rates and expo
- *Logical switches* — combine switches using AND/OR operators… and much more.
- *Outputs* — set servo end points and centres
- *Gvars* — define integer variables
- *Special functions* — trigger actions like telemetry logging, timers

All these elements are shown as simple lists.

It should be clear that OpenTX is very simple at heart. Yet it's also extremely powerful. Part of this is due to its simplicity — building complex systems is easier if you start with small building blocks. However, a large part is also due to an embedded Lua interpreter. What follows is an example of a powerful Lua based extension for a continuous trim system.

The Power of Lua: A Crow Aware Trim System

Anyone who has used crow brakes will know that they can wreak havoc with pitch trim. The usual antidote is a crow-to-elevator mix with a *compensation curve* to deal with any non linear behaviour.



```
CURVES CV2   pt  X     Y
Name         1   -100  -100
 SPC         2   -50   -85
Type         3   0     -65
 Standard    4   50    -37
Count        5   100   0
 5pts
Smooth ☐
```

Figure 2: Crow compensation curve on author's Stribog F3F model.

The problem is that adjusting the curve can take several flights to get right. But hey, why not just *use the trim to bend the compensation curve*? For such a system to be useful, it would need to be completely transparent.

Turns out that with the help of Lua, a practical solution is possible. I call it the 'crow aware adaptive elevator trim'. A bit of a mouthful, I admit, for what is essentially a continuously variable trim system!

The solution, in three parts

There are three parts to the solution. The first is to decouple the elevator trim from the elevator stick — easily achieved in OpenTX. Once freed from its normal duties, the trim behaves like a dumb (double-throw) switch.

The second part is getting the trim to bend the compensation curve. This is where Lua comes in…

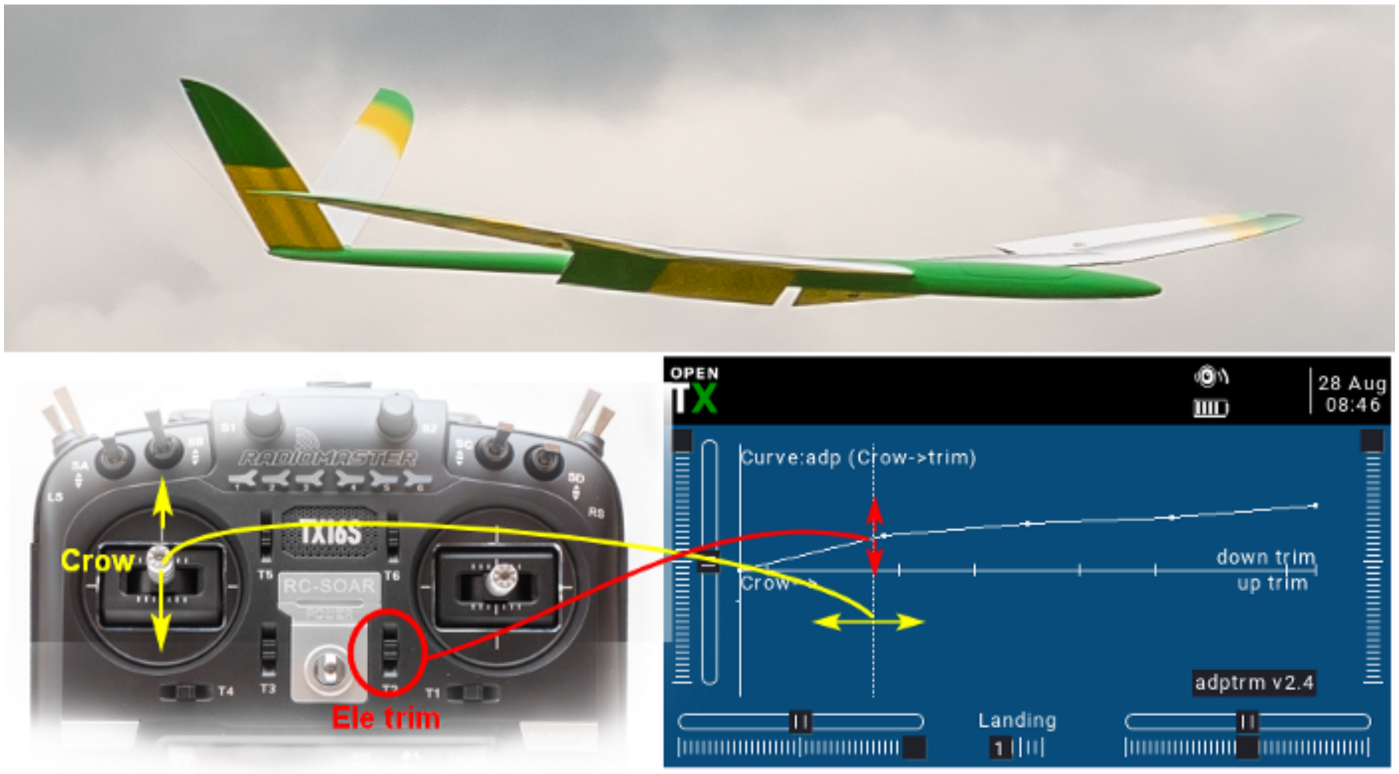


Figure 3: Crow aware trim system. Bottom right shows live view of compensation curve.

The script runs in the background under the control of OpenTX. When a click is detected, the main part of the script springs into action; it determines the crow setting, recalculates the compensation curve, then writes back the updated curve. The key API call is:

```
model.setCurve (cv_idx, {name=cv.name, smooth=1, y=pt_y})
```

As far as I'm aware, there's no equivalent to the `Model.setCurve()`function in other systems, so this is an application where OpenTX has a unique advantage.

The third and final part is to assign the compensation curve to a crow-to-elevator mix. The output of the mix is the crow compensation.

Using this script it's possible to optimise the crow trim before a new model's first landing. The script in use by several pilots and is available for download from my website OpenTX Clinic (see links).

Summing Up

I hope this has given you a flavour of OpenTX — both its simplicity and its power. I won't pretend that it's all roses — the overview I've presented has of necessity been greatly simplified. However, if you're technically inclined, you'll find it a uniquely capable and satisfying operating system.

Finally, I'm eternally grateful to the developers for their continued commitment to this project. OpenTX has been the mainstay of my soaring activity for almost eight years — as it has been for thousands of modelers around the world controlling everything from drones to competition F3X models. Long may it continue!

Additional Resources

- OpenTX.org — home of OpenTX
- OpenTX Clinic — author's site, with sailplane templates and tutorials:
- Crow Aware Trim Script

- List of Interactions for an F3F Model

All photos and figures by Michael Shellim. Read the ***next article*** *in this issue, return to the* ***previous article*** *or go to the* ***table of contents***.