Source : Adobe Stock

Open Vario

variomètre multifonction open source pour vol en parapente

Cedric Jimenez (France)

Parfois, les meilleures créations résultent d'une opportunité ! Cédric Jimenez, lauréat du premier prix du concours STM32 Wireless Innovation Design Contest, avait été informé de ce challenge par un ami et développa un projet innovant pour la mesure de l'altitude de vol par un variomètre. En voici le récit complet.

Tout a commencé lorsqu'un ami m'a transmis le lien vers la page du concours " STM32 Wireless Innovation Design Contest " d'Elektor magazine [1] : "Hey ! as-tu vu ce concours ? Il est basé sur l'utilisation d'un chip sans-fil de ST Microelectronics, et ils offrent une carte d'évaluation !" Il savait comment susciter mon attention, car j'adore pouvoir expérimenter les nouveaux microcontrôleurs, et je suis un grand amateur de ces cartes, qui permettent de mettre en œuvre rapidement des projets pour découvrir leurs fonctionnalités.

Choix du matériel

J'ai pris connaissance des cartes d'évaluation proposées : deux cartes Nucleos (NUCLEO-WBA52CG, NUCLEO-WL55JC) et un kit d'expérimentation (STM32WB5MM-DK). Ce dernier était, selon moi, le plus attractif comme base d'un projet, car il intègre de nombreuses possibilités : affichage, mémoire, capteurs, USB, connecteurs d'extensions... Parfait ! j'avais choisi une carte, il me restait à trouver une idée de projet.

Je suis pilote de parapente, et je réfléchissais depuis longtemps à développer mes propres instruments de vol. J'avais même imaginé une conception matérielle, il y a cinq ans, mais je n'ai pas persisté, principalement par manque de temps. J'ai alors considéré ce concours comme une opportunité, pour transformer cette idée en un appareil que je pourrais utiliser durant mes vols.

En vol parapente, le variomètre, également connu sous le nom d'indicateur de vitesse d'ascension et de descente (rate of climb and descent indicator : RCDI) – est l'instrument le plus précieux ! La vitesse ascensionnelle, exprimé en mètres (ou pieds) par seconde, aide à la recherche des courants ascendants, qui permettent au parapentiste de prendre de l'altitude, et ainsi rester en vol durant plusieurs heures. En plus de la fonction principale d'un variomètre, les instruments commerciaux possèdent des fonctions additionnelles comme le système de navigation par satellite GNSS (Global Navigation Satellite System) contrôlé par un smartphone.

Spécifications exigées

En me basant sur les possibilités offertes par la carte STM32WB5MM-DK, j'ai commencé par dresser une liste des fonctions que je souhaitais implémenter (voir **figure 1**) :

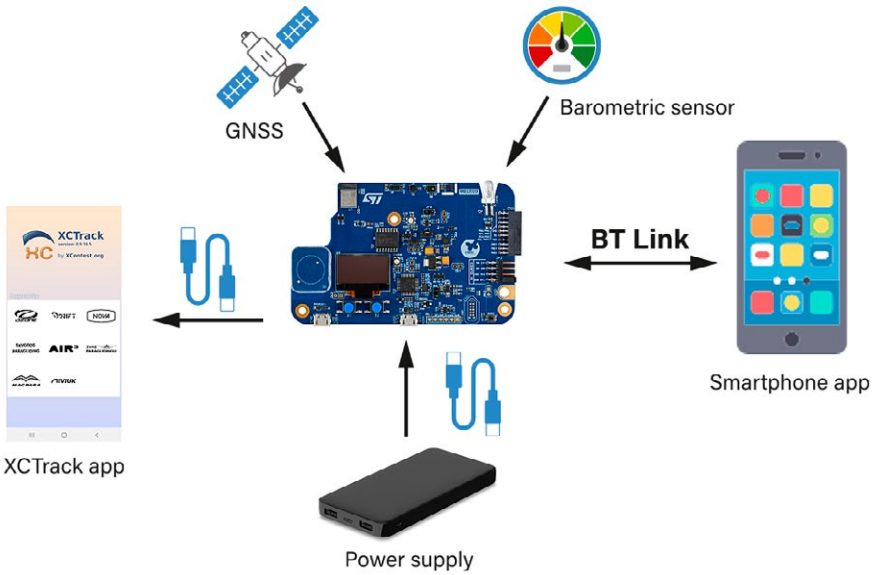

Figure 1. La carte de développement STM32WB5MM-DK est au cœur de ce projet.

- Variomètre
- Positionnement GNSS
- Accéléromètre
- Température
- Calcul du facteur de glissement (i.e., la distance horizontale pouvant être couverte par la perte d'altitude de 1mètre)
- Enregistrement des données de vol (altitude, vitesse, position…)
- Interface de contrôle (IHM) et affichage des données des capteurs, et du vol)
- Lien Bluetooth à faible consommation (BLE), permettant la configuration des paramètres et la lecture des données de vol en temps-réel
- Lien USB OTG permettant de se connecter à l'application smartphone XCTrack (application réputée pouvant être utilisée pour le déport de l'affichage)
- Interface de liaison USB pour se connecter à des utilitaires Python de ma conception, sur PC, permettant l'export des données de vol.

Matériel additionnel nécessaire

Bien que la carte STM32WB5MM-DK puisse remplir la plupart des fonctionnalités requises, il manquait deux capteurs : un récepteur GNSS permettant de recevoir la position de l'appareil et un capteur de pression permettant de calculer la vitesse ascensionnelle.

Le calcul de la vitesse ascensionnelle est réalisé en mesurant la différence d'altitude plusieurs fois par seconde. Un dispositif GNSS ne fournit pas une information d'altitude suffisamment précise, c'est pourquoi un capteur de pression barométrique de haute précision est nécessaire, afin d'obtenir une précision de 15 à 30 cm lors de la mesure de l'attitude.

La nécessité de réaliser une intégration électrique robuste avec la carte STM32WB-5MM-DK, résistante aux vibrations et aux chocs durant le décollage, ou produites par les turbulences et lors de l'atterrissage, introduit des contraintes supplémentaires. Le format de l'ensemble des dispositifs intégrés doit également être aussi compact que possible et facilement transportable. C'est pourquoi j'ai choisi de mettre à profit le connecteur compatible Arduino UNO de la carte STM32 et l'utilisation d'une carte enfichable 'MikroE insérée sur un module (shield) dédié [3], ainsi, aucune soudure ou fil de liaison n'est nécessaire. MikroElektronika propose une grande diversité de modules capteurs (plus de 1 000), j'en ai souvent utilisé en prototypage rapide.

En ce qui concerne le GNSS ? Je possédais un module enfichable MikroE GNSS 4 [4] comprenant le module U-blox AM-M8Q, qui intègre une antenne patch GNSS omnidirectionnelle, et une interface UART.

Pour le capteur de pression barométrique, MikroE propose trois capteurs de haute-précision pouvant convenir à mon projet. J'ai consulté les forums de vol parapente et parcouru la documentation fournie. J'ai finalement décidé d'utiliser le module Altitude 2 Clic [5] basé sur un capteur de pression MS5607 de TE Connectivity, utilisé dans d'autres projets de variomètres. Il est facile à mettre en œuvre, comporte une documentation de qualité, et peut être connecté par un bus I²C ou SPI.

J'ai ensuite relié l'ensemble, afin de voir comment cela se présentait (vous trouverez un schéma synoptique sur la **figure 2**) et… mauvaise surprise : ni le bus I²C, ni le bus SPI n'étaient présents sur les broches de liaison nécessaires au module de pression barométrique ! Bon, ce n'est pas un point bloquant, le protocole du bus I²C n'est pas complexe, j'ai pu contourner le problème en créant un pilote logiciel I²C.

Conception logicielle

Maintenant que j'avais réuni et interconnecté le matériel nécessaire, je pouvais commencer à réfléchir à la partie logicielle. J'aime partager mes codes source avec d'autres concepteurs, j'aime également pouvoir réutiliser le travail des autres personnes. C'est pour cela que je partage la plupart de mes programmes sous licence open source (logiciel libre), et c'est pourquoi ce projet se devait d'être en open source [6]. Pour ce projet, j'ai choisi la licence MIT afin d'offrir la flexibilité maximum pour l'usage par de futurs utilisateurs ou contributeurs, car ce dispositif n'est pas destiné à devenir un produit commercialisé.

Pour la programmation, j'ai utilisé mon langage favori : C++. Il offre toutes les possibilités bas niveau du langage C et offre en plus toutes les caractéristiques d'un langage moderne orienté objet : sécurité des types, héritage, templates… La plupart des gens pensent "qu'utiliser C++ signifie utiliser l'allocation dynamique de mémoire". Est-ce que l'usage de `malloc()` et `free()` est obligatoire dans un programme C ? Non, pas plus que *new* et *delete* en C++. Cela ne dépend que de la

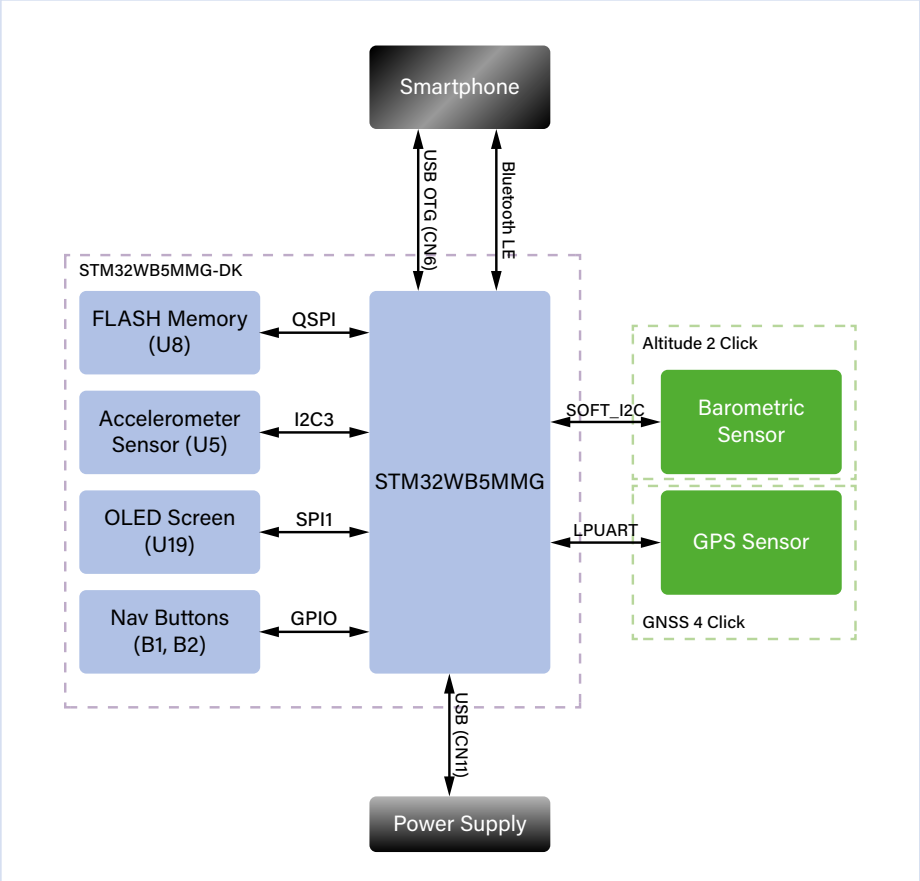

Figure 2. Schéma fonctionnel du système montrant les interfaces physiques aux modules externes.

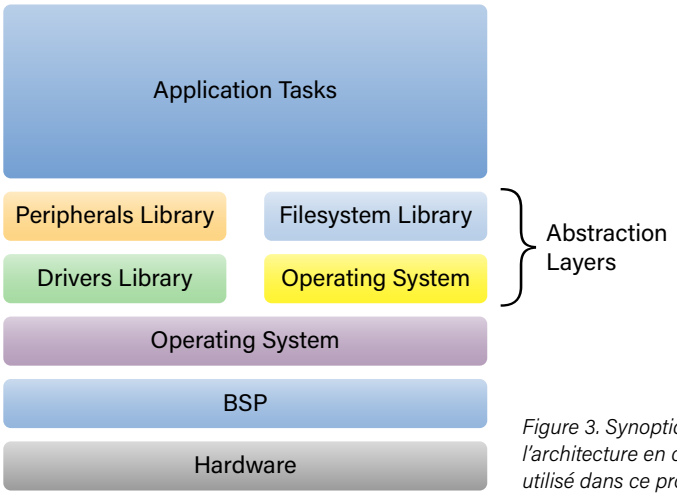

Figure 3. Synoptique fonctionnel de l'architecture en couches du logiciel utilisé dans ce projet.

partie du langage mise en œuvre, de multiples options existent sans utiliser l'allocation dynamique de mémoire.
ST offre une multitude d'outils logiciels pour ses microcontrôleurs (STM32CubeIDE, STM32CubeMX ...) et de nombreux exemples dans la bibliothèque de pilotes ST HAL, ou dans les packages MCU STM32Cube. Bien que j'aime réellement ces outils, qui m'aident à créer rapidement des petits prototypes, en générant une partie du code et en me dirigeant vers les bibliothèques adéquates à utiliser, j'ai préféré utiliser l'environnement de développement intégré (EDI) "Visual Studio Code IDE" avec les extensions C/C++ pour ses possibilités de mise en évidence du code et de navigation, avec en complément, les extensions Cortex-Debug pour la mise au point des périphériques intégrés utilisant les débogueurs ST-Links ou J-Links.
Il offre les mêmes possibilités que STM32CubeIDE, mais il est plus facile à configurer et paramétrer par ses modules attachés, et les fichiers de configuration JSON. J'ai également préféré écrire mes propres systèmes de génération de code exécutable CMake/Make, étant ainsi totalement indépendant de l'EDI utilisé et me permettant d'avoir le contrôle intégral de la configuration du code source et des fichiers de code (aucun code n'est généré par les outils ST).
En plus des bibliothèques *STM32 HAL*, *STM32 USB,* et bien entendu des couches Bluetooth *STM32 WPAN,* j'ai décidé d'utiliser les bibliothèques tierces suivantes (toutes disponibles sur GitHub) :

- FreeRTOS 10.6.2 : Système d'exploitation en temps réel (je l'ai trouvé facile à utiliser par rapport à la conception à boucle unique lorsque les ressources ne sont pas trop contraintes).
- Little-FS 2.81: d'utilisation facile, peu encombrant et insensible aux interruptions d'alimentation, en tant que système de fichier pour la mémoire Flash NOR.
- YACSGL/YACSWL 0.0.1: Bibliothèque graphique peu encombrante adaptée aux afficheurs OLED monochromes et/ou aux écrans E-Ink.

J'ai commencé par créer un module C++ encapsulant l'ensemble de ces bibliothèques afin de faciliter leur intégration au code C++. Par ailleurs, pour certaines parties, comme la classe `mutex`, j'ai simulé le comportement C++ standard de `std::mutex` afin de faciliter l'écriture du code C++. J'ai également décidé d'implémenter une architecture logicielle en couche, "classique" mais efficiente (voir la **figure 3**). Enfin, j'ai utilisé ce que j'appelle un "schéma d'abstraction de carte" qui définit une interface me permettant de retrouver tous mes objets traitant des périphériques et les pilotes, par l'intermédiaire, bien entendu, d'une interface abstraite. Après cette abondance de théorie, regardons maintenant le code du **listage 1**, également disponible sur GitHub [6]. En utilisant cette interface, je peux, dans mes tâches applicatives, accéder facilement aux périphériques matériels et aux pilotes sans me préoccuper de leur implémentation. Ceci me permet, par exemple, de facilement prélever les sorties de mes capteurs pour accélérer le développement des algorithmes des tâches applicatives. Dans un autre projet, j'avais utilisé la même approche pour abstraire le matériel et permettre à un même logiciel, de fonctionner sur une multitude de cartes différentes.

Test

La phase ultime nécessitait de tester mon projet dans les conditions de vol réel. À partir de données simulées, mes algorithmes produisaient des valeurs valides, et se comportaient correctement quand je changeais les paramètres de leurs filtres. Mais en vol parapente, tout est question de ressenti ; j'ai ainsi dû ajuster les paramètres lors de vol réels, afin que le dispositif s'accorde avec ma façon de voler. J'ai également été amené à fournir un moyen de les configurer aisément, afin qu'un autre pilote puisse les ajuster.
Je ne pouvais pas exposer l'appareil à l'air libre en raison de l'impact de l'exposition au soleil (et à l'écoulement d'air) sur le capteur très précis de pression atmosphérique. Au lieu de cela, j'ai utilisé mon smartphone pour visualiser les données et contrôler les paramètres. Il y a deux options pour connecter la carte STM32 à un smartphone, Bluetooth ou la connexion USB OTG ; j'ai expérimenté les deux.
La liaison Bluetooth permet l'accès à tous les paramètres des différents services et caractéristiques (voir le document présent sur la page Elektor Labs de ce projet [7]). Malheureusement, je n'ai pas eu la possibilité de développer une application smartphone dédiée, offrant une interface utilisateur agréable, mais cela reste facilement modifiable en utilisant des

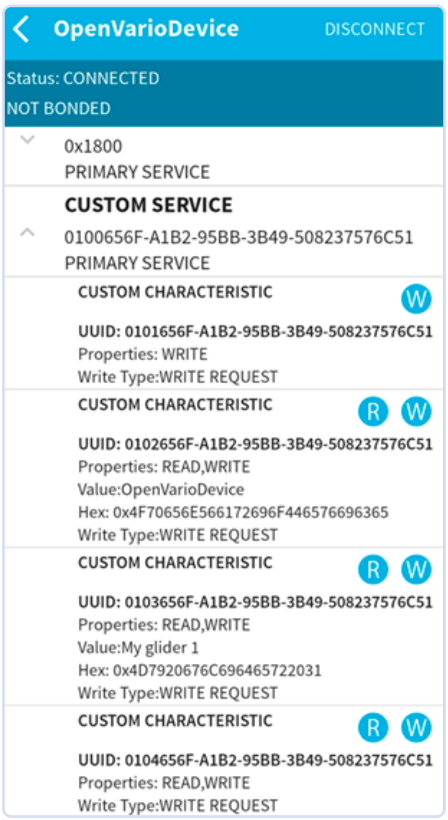

Figure 4. Capture d'écran de l'application BLE Scanner, utilisée pour personnaliser les services d'exécution.

Figure 5. Variomètre commercial utilisé pour comparer les mesures faites par mon Open Vario.

applications BLE génériques gratuites, telle que BLE Scanner (voir la **figure 4**).
Une autre option consiste à utiliser l'affichage sur smartphone durant le vol : l'application XCTrack est communément utilisée par les parapentistes ; elle utilise la liaison OTG du "Discovery Kit". Cela m'a permis de visualiser, sur mon smartphone, les données de vol calculées par mon appareil. J'ai également ajouté à mon installation le variomètre "commercial" que j'utilise habituellement, afin de comparer les valeurs obtenues, comme le montre la **figure 5**.
Durant les vols, j'enregistre périodiquement (le défaut est 1 s.), toutes les sorties de mes capteurs et des algorithmes qui les accompagnent, en plus de ma position GNSS. Chaque vol est enregistré dans un fichier séparé de la mémoire Flash NOR QSPI. Avec les paramètres par défaut, je peux enregistrer environ 24 heures de vol ce qui est largement suffisant. J'ai développé des scripts Python pour récupérer ces fichiers sur un PC utilisant l'interface USB du "Discovery Kit" et les sauvegarder dans un fichier CSV pouvant être facilement analysé à l'aide d'EXCEL (voir **figure 6**), et également visualisables en format KML dans l'application Google Earth pour observer mon vol en 3D, comme illustré sur la **figure 7**.

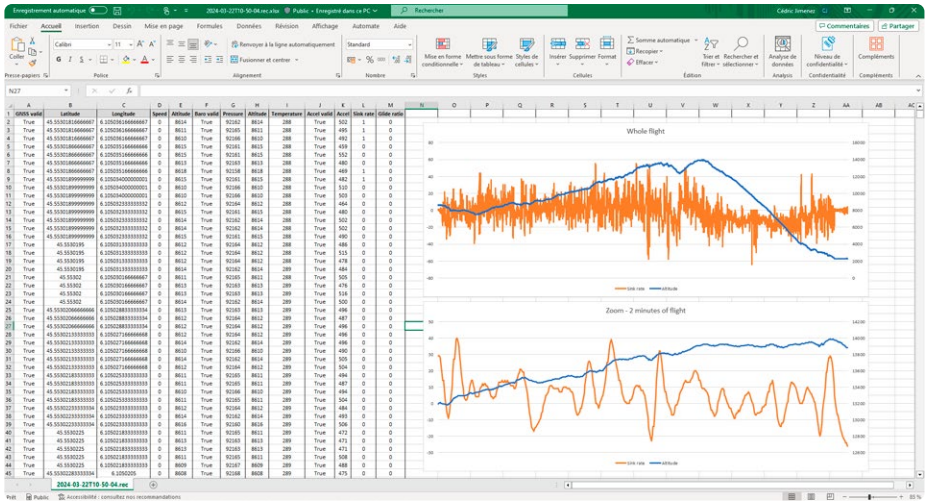

*Figure 6. Données acquises, stockées dans un fichier **.csv**, analysées et visualisées à l'aide d'Excel.*

Nombreuses opportunités

Je pense que vous avez maintenant réalisé que j'ai réellement pris plaisir à participer à ce concours, même si cela était chronophage (5...10 h par semaine) ! Cette occasion m'a offert de multiples opportunités : découverte d'un nouveau microcontrôleur, réalisation d'un projet auquel je n'avais jamais consacré le temps nécessaire à son développement, tout en mélangeant deux de mes passions

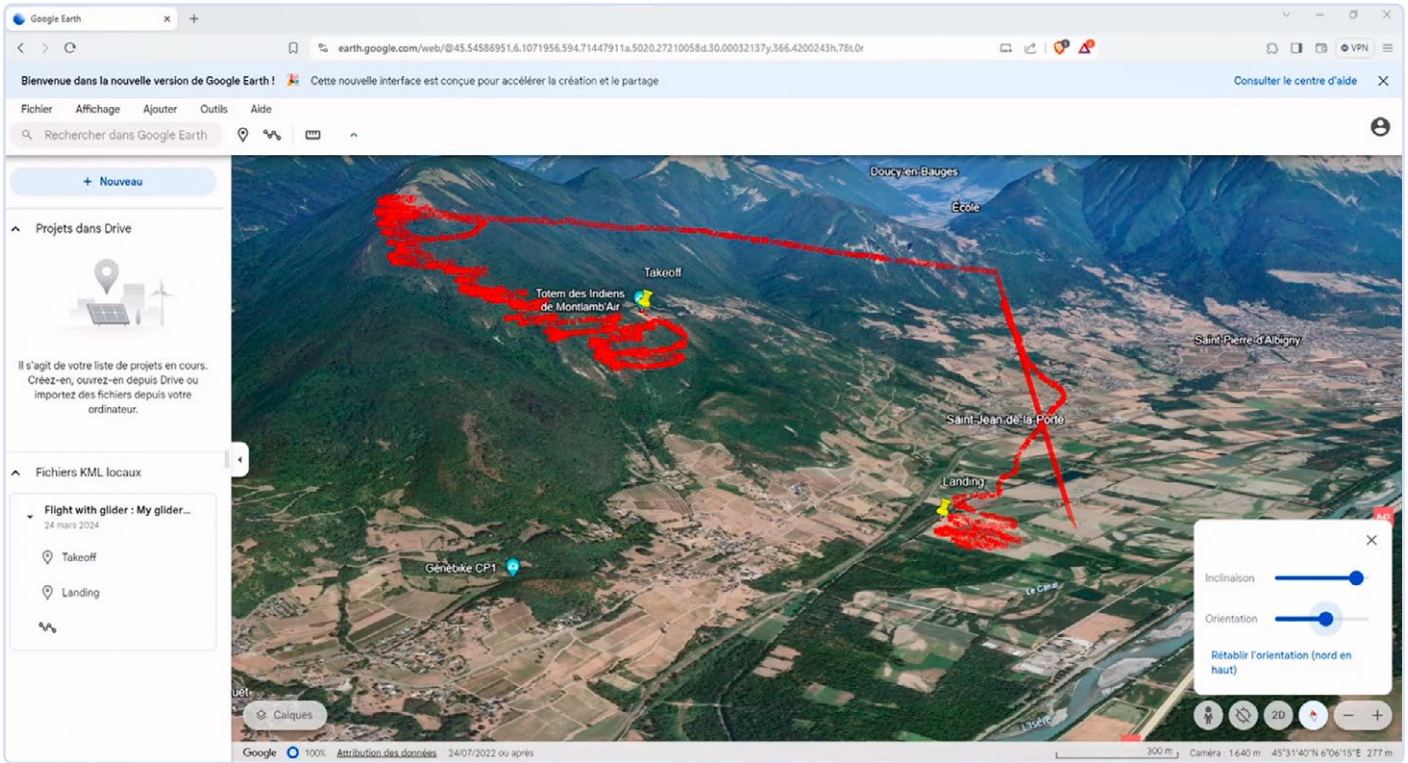

Figure 7. Visualisation 3-D des données de vol (fichier KML) par Google Earth.

favorites, ainsi que la découverte d'autres projets excitants, utilisant le même matériel. La limite de temps imposée pour ce concours (merci à l'extension du délai !) était un véritable challenge : chaque projet comporte de nombreuses difficultés à résoudre, mais doit, en finalité, comporter un nombre important de caractéristiques fonctionnelles. Je pense que ma profession d'Ingénieur logiciel, m'a beaucoup aidé sur cet aspect car ce sont les contraintes que je rencontre de façon habituelle.

Merci au magazine Elektor et à ST Microelectronics d'avoir organisé ce concours. J'espère que d'autres compétitions seront organisées dans le futur !

VF : Jean Boyer — 240325-04

À propos de l'auteur

Cédric Jimenez est âgé de 40 ans, il est l'heureux père d'une fillette de 8 ans. Il vit dans les Alpes françaises, où il se livre à de multiples activités de montagne (ascensions, parapente). Cédric est un grand amateur de développement de logiciel pour les systèmes embarqués. Il a débuté la programmation en langage Basic à l'âge de 9 ans sur un TO7/70 Thomson et s'est orienté vers le langage C++ quelques années plus tard.

Bien qu'il ait appris de nombreux langages de programmation (VB, C#, C, Pascal, FORTRAN, Python, Ruby...), C++ demeure son langage de programmation favori.

Il est titulaire d'un Master en Sciences Informatiques, avec une spécialisation en logiciel temps-réel sur systèmes embarqués. Il a exercé pendant plus de 15 ans comme sous-traitant dans des domaines variés (ferroviaire/métropolitain, grues, services médicaux...). Il est actuellement architecte en logiciel intégré chez Schneider Electric.

Questions ou commentaires ?

Envoyez un courriel à l'auteur (cjz.73fr@gmail.com), ou contactez Elektor (redaction@elektor.fr).

Listage 1. Classe d'abstraction de carte

```
/*
 * Copyright (c) 2023 open-vario
 * SPDX-License-Identifier: MIT
 */
#ifndef OV_I_BOARD_H
#define OV_I_BOARD_H
#include "i_accelerometer_sensor.h"
#include "i_barometric_altimeter.h"
#include "i_ble_stack.h"
#include "i_button.h"
#include "i_display.h"
#include "i_gnss.h"
#include "i_serial.h"
#include "i_storage_memory.h"
#include "i_usb_cdc.h"

namespace ov
{

/** @brief Interface for boards implementations */
class i_board
{

  public:
    /** @brief Destructor */
    virtual ~i_board() { }

    /** @brief Reset the board */
    virtual void reset() = 0;

    /** @brief Get the debug serial port */
    virtual i_serial& get_debug_port() = 0;

    /** @brief Get the USB CDC port */
    virtual i_usb_cdc& get_usb_cdc() = 0;

    /** @brief Get the storage memory */
    virtual i_storage_memory& get_storage_memory() = 0;

    /** @brief Get the display */
    virtual i_display& get_display() = 0;

    /** @brief Get the 'Previous' button */
    virtual i_button& get_previous_button() = 0;

    /** @brief Get the 'Next' button */
    virtual i_button& get_next_button() = 0;

    /** @brief Get the 'Select' button */
    virtual i_button& get_select_button() = 0;

    /** @brief Get the BLE stack */
    virtual i_ble_stack& get_ble_stack() = 0;

    /** @brief Get the GNSS  */
    virtual i_gnss& get_gnss() = 0;
```

```
    /** @brief Get the barometric altimeter */
    virtual i_barometric_altimeter& get_altimeter() = 0;

    /** @brief Get the accelerometer */
    virtual i_accelerometer_sensor& get_accelerometer() = 0;
};

} // namespace ov

#endif // OV_I_BOARD_H
```

LIENS

[1] Page web du concours STM32 : https://tinyurl.com/bdenyy24
[2] Page web des outils d'évaluation STM32WB5MM-DK : https://tinyurl.com/4dp83f8h
[3] MikroElektronika Arduino Uno Click Shield : https://www.mikroe.com/arduino-uno-click-shield
[4] MikroElektronika GNSS 4 Click Board : https://www.mikroe.com/gnss-4-click
[5] MikroElektronika Altitude 2 Click Board : https://www.mikroe.com/altitude-2-click
[6] Page Github de l'auteur pour le projet Vario : https://github.com/open-vario/open-vario
[7] Ce projet sur Elektor Labs : https://tinyurl.com/4rmn5j28

Produits

- **Majid Pakdel, *Advanced Programming with STM32 Microcontrollers*, Elektor, 2020**
  www.elektor.fr/19527