Development of a Range Estimator, through enhancing log data analysis

Auteur:
Martin GALLIOT

Superviseur:
Victor DELAFONTAINE

RigiTech
Prilly, Suisse

Octobre, 2023

RIGITECH

Abstract

Development of a Range Estimator, through enhancing log data analysis

by Martin GALLIOT

L'industrie des drones connaît une expansion rapide, avec un nombre croissant d'entreprises se spécialisant dans un large éventail d'applications. RigiTech, une start-up suisse, est à la pointe d'un écosystème complet axé sur des livraisons efficaces et autonomes dotées de capacités opérationnelles étendues. L'autonomie des drones est maintenue grâce à différents processus, mais dans l'objectif de faciliter la tâche de l'opérateur. L'objectif ultime est de permettre à un seul opérateur de superviser efficacement plusieurs itinéraires de livraison simultanément.

J'ai principalement travaillé dans le but de développer un Range Estimator. Ce projet a été divisé en deux parties. Tout d'abord, j'ai corrigé et amélioré Flight Review, un outil d'analyse de logs PX4, afin d'obtenir des valeurs de consommation cohérentes. Cela a fourni une base solide pour le développement du Range Estimator, d'abord sur RigiCloud et plus tard pour l'estimation en direct sur le drone.

Enfin, on m'a confié des responsabilités supplémentaires, notamment la correction de bogues, l'intégration de nouvelles fonctionnalités et la participation aux efforts de recherche et de développement. Cet effort collectif a joué un rôle essentiel dans l'amélioration continue du drone et de l'interface cloud qui lui est associée.

The drone industry is experiencing rapid expansion, with an increasing number of companies specializing in a wide range of action. RigiTech, a Swiss startup, is pioneering a comprehensive ecosystem centered on efficient and autonomous delivery with extensive operational capabilities. The drones' self-reliance is upheld through different process but all working together to make the drone operator's job easier. The ultimate goal is to empower a single operator to efficiently oversee multiple delivery routes simultaneously.

I primarily worked with the aim of developing a Range Estimator. This project was divided into two parts. Firstly, I corrected and improved Flight Review, a ULog analysis tool, to obtain consistent consumption values. This, in turn, provided a solid foundation for developing the Range Estimator, initially on RigiCloud and later for live estimation directly on the drone.

Finally, I was tasked with additional responsibilities, encompassing bug fixes, the integration of new features, and engagement in Research and Development efforts. This collective effort has played a pivotal role in the ongoing enhancement of both the drone and its associated cloud interface.

Acknowledgements

I would like to thank here all the people that made this internship possible. I wish to extend my heartfelt gratitude to Victor Delafontaine for his unwavering support, patience in addressing my myriad of questions, and the wealth of answers he shared with me.

I also wish to thank Jonas Perolini and Etienne Meunier, who took me under their wing from the very beginning, greatly enhancing my efficiency and enabling me to achieve as much as possible during this short four-month internship, through excellent advice.

Furthermore, I would like to express my appreciation to the entire RigiTech team, who have created a welcoming and dedicated work environment. Their invaluable experience and knowledge have been instrumental during this journey.

Contents

List of Figures

Chapter 1

Introduction

1.1 Presentation of RigiTech

The internship and its activities that will be presented here took place at RigiTech, a Switzerland start-up, based at Prilly, near Lausanne. Named after the mount Rigi, RigiTech's goal is to create a fully integrated, inter-city drone delivery solution, flying over crowded cities, rivers, lakes and mountains.

RigiTech is a Swiss air logistics company founded in 2018 by 3 founders : Adam Klaptocz (CEO), David Rovira (CBO) and Oriol López (CTO). In less than five years, they have gone from the drawing board to selling an innovative delivery solution on five continents. Being among the first on the VTOL drone market, the needs and therefore the size of the drone. So it's only natural that their offer has varied and improved. From the RigiOne (middle) to the Eiger (right), via the Minto (left), RigiTech has adapted to the needs of its different customers.

Figure 1.1: The RigiTeam

1.2 What they offer

Contrary to what one might think, RigiTech does not sell drones. That's not its objective. So as not to limit themselves in what they offer, they want to put forward a customizable delivery service, for a set period of time. This includes a drone (Eiger), an Edge Node for communication / internet and a Cloud for mission planning and execution.

Figure 1.2: Edge Node

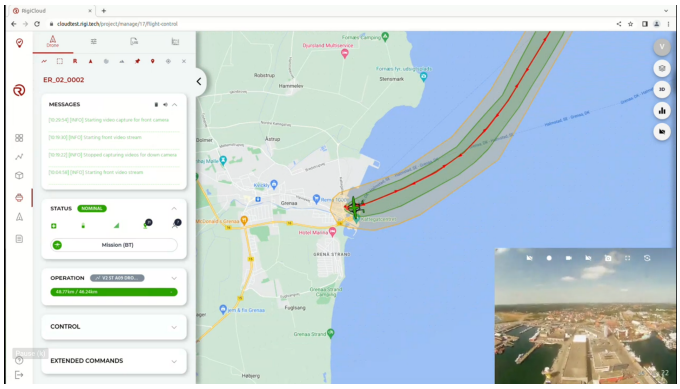


Figure 1.3: RigiCloud

1.3 The Eiger

Currently, RigiTech exclusively offers the Eiger drone, which possesses the ability to transport a payload weighing up to three kilograms within a radius of one hundred kilometers in just one hour. This remarkable technological achievement is made achievable through the utilization of vertical take-off and landing (VTOL) technology, granting the drone dual operational modes. In its quadcopter mode, the Eiger takes off vertically using four propellers. After becoming airborne, it transitions to fixed-wing mode, with the rear propeller taking control and propelling the drone forward, allowing it to navigate like a traditional aircraft thanks to its wings providing the necessary lift. This innovative design consumes significantly less battery power compared to conventional quadcopters, resulting in a tenfold increase in its operational range,while also notably increasing its speed.

Figure 1.4: The Eiger

The Eiger is also proficient in executing Beyond Visual Line Of Sight (BVLOS) missions, which, as per their definition, do not necessitate the presence of a pilot on-site or visual supervision. RigiTech's primary focus presently lies in the delivery of medical samples to remote regions. Utilizing the Eiger drone expedites laboratory analysis and contributes to a reduction in carbon emissions when juxtaposed with conventional transportation methods such as trucks or automobiles.

1.4 Purpose of the internship

RigiTech embraces the Agile methodology and follows the Scrum framework to ensure efficient and iterative project management. During each development sprint, which spans a duration of one month, all code enhancements are rolled out onto a dedicated test server. This server exclusively houses code that has reached a reasonably stable state and has undergone thorough testing on a development server. At three-month intervals, following comprehensive testing and the necessary bug fixes, the contents of the test server are migrated to the production server, ensuring that customers can benefit from the latest improvements and additions. As a result, short- and medium-term objectives change rapidly, and the work I had to do was not defined at the start of the internship.

When I arrived, one of the main needs was to improve robustness, both on the drone and on the Cloud. In particular, there was room for improvement in terms of estimating consumption and battery life, and therefore in terms of estimating the distance that can be covered. I also worked on one-off tasks such as debugging important tasks that needed hot fixes on the production server.

Chapter 2

PX4 and Flight Review

RigiTech uses and adapts PX4 [3], a versatile open-source autopilot software suite widely employed for controlling autonomous drones and robotic vehicles. It records flight data in ULog files, which capture crucial telemetry and system information. To analyze and optimize vehicle performance, users rely on Flight Review [4], a web-based tool that visualizes ULog data, offering insights into flight behavior and aiding in troubleshooting and configuration adjustments.

RigiCloud includes an improved version of Flight Review. It stands out for its more numerous and useful data for operators and customers. In the following, PX4 and Flight Review refer to their RigiTech versions.

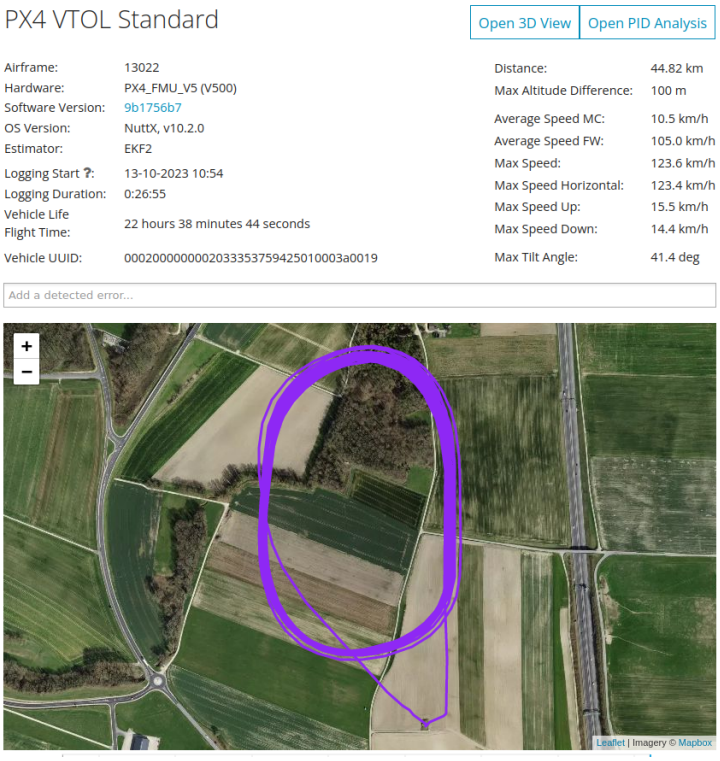
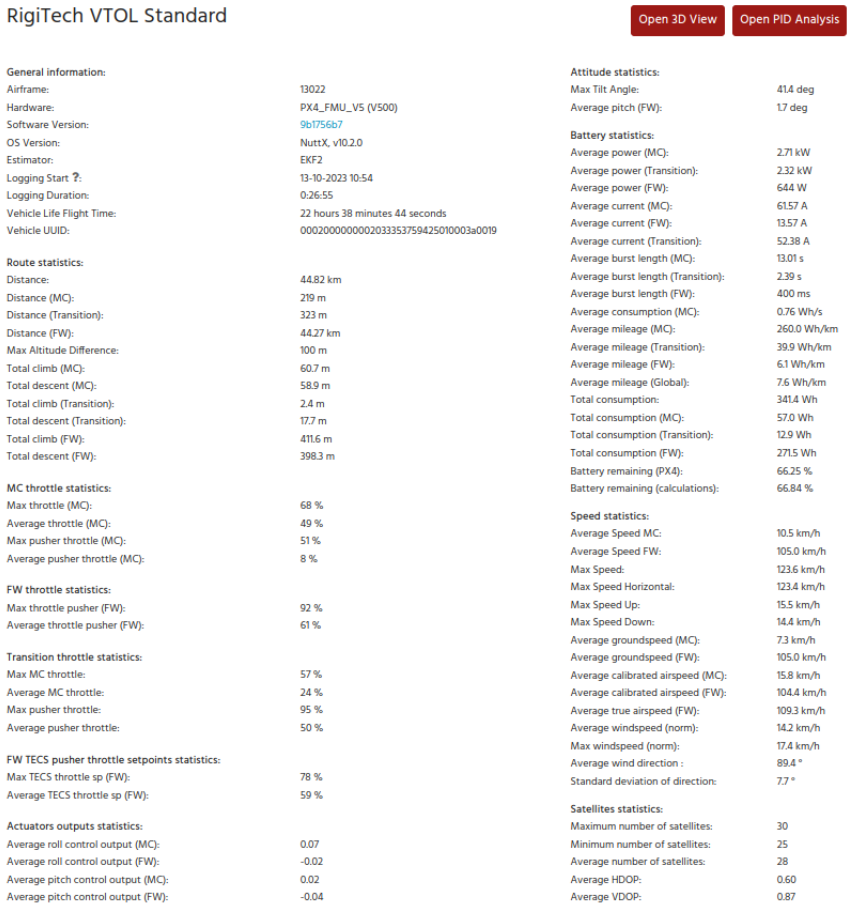


Figure 2.1: Some data from PX4's Flight Review



RigiTech VTOL Standard

Open 3D View | Open PID Analysis

General information:

Airframe:	13022
Hardware:	PX4_FMU_V5 (V500)
Software Version:	9b1756b7
OS Version:	NuttX, v10.2.0
Estimator:	EKF2
Logging Start :	13-10-2023 10:54
Logging Duration:	0:26:55
Vehicle Life Flight Time:	22 hours 38 minutes 44 seconds
Vehicle UUID:	0002000000000203335375942501003a0019

Route statistics:

Distance:	44.82 km
Distance (MC):	219 m
Distance (Transition):	323 m
Distance (FW):	44.27 km
Max Altitude Difference:	100 m
Total climb (MC):	60.7 m
Total descent (MC):	58.9 m
Total climb (Transition):	2.4 m
Total descent (Transition):	17.7 m
Total climb (FW):	411.6 m
Total descent (FW):	398.3 m

MC throttle statistics:

Max throttle (MC):	68 %
Average throttle (MC):	49 %
Max pusher throttle (MC):	51 %
Average pusher throttle (MC):	8 %

FW throttle statistics:

Max throttle pusher (FW):	92 %
Average throttle pusher (FW):	61 %

Transition throttle statistics:

Max MC throttle:	57 %
Average MC throttle:	24 %
Max pusher throttle:	95 %
Average pusher throttle:	50 %

FW TECS pusher throttle setpoints statistics:

Max TECS throttle sp (FW):	78 %
Average TECS throttle sp (FW):	59 %

Actuators outputs statistics:

Average roll control output (MC):	0.07
Average roll control output (FW):	-0.02
Average pitch control output (MC):	0.02
Average pitch control output (FW):	-0.04

Attitude statistics:

Max Tilt Angle:	41.4 deg
Average pitch (FW):	1.7 deg

Battery statistics:

Average power (MC):	2.71 kW
Average power (Transition):	2.32 kW
Average power (FW):	644 W
Average current (MC):	61.57 A
Average current (FW):	13.57 A
Average current (Transition):	52.38 A
Average burst length (MC):	13.01 s
Average burst length (Transition):	2.39 s
Average burst length (FW):	400 ms
Average consumption (MC):	0.76 Wh/s
Average mileage (MC):	260.0 Wh/km
Average mileage (Transition):	39.9 Wh/km
Average mileage (FW):	6.1 Wh/km
Average mileage (Global):	7.6 Wh/km
Total consumption:	341.4 Wh
Total consumption (MC):	57.0 Wh
Total consumption (Transition):	12.9 Wh
Total consumption (FW):	271.5 Wh
Battery remaining (PX4):	66.25 %
Battery remaining (calculations):	66.84 %

Speed statistics:

Average Speed MC:	10.5 km/h
Average Speed FW:	105.0 km/h
Max Speed:	123.6 km/h
Max Speed Horizontal:	123.4 km/h
Max Speed Up:	15.5 km/h
Max Speed Down:	14.4 km/h
Average groundspeed (MC):	7.3 km/h
Average groundspeed (FW):	105.0 km/h
Average calibrated airspeed (MC):	15.8 km/h
Average calibrated airspeed (FW):	104.4 km/h
Average true airspeed (FW):	109.3 km/h
Average windspeed (norm):	14.2 km/h
Max windspeed (norm):	17.4 km/h
Average wind direction :	89.4 °
Standard deviation of direction:	7.7 °

Satellites statistics:

Maximum number of satellites:	30
Minimum number of satellites:	25
Average number of satellites:	28
Average HDOP:	0.60
Average VDOP:	0.87

Figure 2.2: RigiTech's Flight Review for the same log

2.1 Fix mileage consumption

Upon my arrival, I noticed that the calculation of average consumption per kilometer exhibited inconsistency. The overall consumption in one mode, such as the multicopter mode, didn't align with the result of multiplying the consumption per kilometer by the distance traveled.

As depicted in the image above, there are two methods for estimating the remaining battery capacity at the end of the flight ; *PX4* and *calculations*. Given that the drone employs Li-Ion batteries, accurately gauging the remaining battery power can be challenging. PX4 handles this calculation internally, as we will explore later. This calculation approach serves as a means to verify the accuracy and consistency of the mileage calculations.

Energy (E) is the integral of power (P) with respect to time (t):

$$E = \int P \, dt$$

Power is the rate at which energy is transferred or converted, and it can be calculated as the product of voltage (V) and current (I):

$$P = V \cdot I$$

To approximate the energy using numerical methods, such as the rectangle method, trapezoidal rule, and Simpson's rule, we discretize the time interval into n subintervals (Δt).

2.1.1 Rectangle Method

In the rectangle method, we approximate the integral as a sum of rectangles with heights equal to the values of P at specific time points:

$$E \approx \sum_{i=1}^{n} P_i \cdot \Delta t$$

2.1.2 Trapezoidal Rule

The trapezoidal rule is a refinement of the rectangle method and provides a more accurate approximation by averaging the heights of consecutive rectangles:

$$E \approx \frac{1}{2} \sum_{i=1}^{n} (P_i + P_{i+1}) \cdot \Delta t$$

2.1.3 Simpson's Rule

Simpson's rule is even more accurate and approximates the curve using quadratic functions. It takes three consecutive points at a time:

$$E \approx \frac{1}{3} \sum_{i=1}^{n/2} (P_{2i-2} + 4P_{2i-1} + P_{2i}) \cdot \Delta t$$

2.1.4 Comparison and conclusion

To evaluate each approach, I analyzed three flight scenarios: one involving solely a transit in multicopter, another simulating a typical flight sequence (take-off, transitions, and landing), and a third featuring a higher frequency of transitions to simulate scenarios like payload drops.

	Flight 1	Flight 2	Flight 3
Rectangle Method	123.00	745.22	632.30
Trapezoidal Rule	122.95	745.20	632.31
Simpson's Rule	122.96	745.15	632.37

Table 2.1: Comparison of Energy Calculation Methods (in Wh)

I opted to use the rectangle method for energy calculation due to several compelling reasons. Firstly, when comparing the results obtained with the rectangle method against the more complex trapezoidal and Simpson's rules, I found that the differences were negligible, typically less than 1%. This level of accuracy was deemed sufficient for my specific application. Secondly, the rectangle method demonstrated a significant advantage in terms of computational efficiency, making it considerably faster in execution compared to the other methods. On Flight Review, the speed difference was substantial. Indeed, the calculations were carried out in C++, which is known to be computationally efficient, and on a relatively powerful computer. However, as the long-term objective was to code an on-board Range Estimator, the calculations had to be the same everywhere and as fast and simple as possible, even if the error was a little greater.

Consequently, the combination of acceptable accuracy and significantly improved computational speed made the rectangle method the preferred choice for my energy calculations.

I have hence corrected the consumption data. The disparity between the PX4-provided value and the calculated one is now minuscule, typically falling below 1% of the battery capacity. As a result, I have established consistent mileage figures. This dataset serves a valuable purpose, especially in discerning whether the drone's energy consumption exceeds that of other instances or even surpasses its own historical patterns and therefore reveal unusual behaviour such as bad weather, battery wear or motors.

2.2 PX4 parameters

We assumed that PX4 did a good estimation of the remaining battery. But this assumption needed to be confronted.

2.2.1 A Li-Ion battery

Li-ion (Lithium Ion) batteries are a type of rechargeable battery known for their high energy density, lightweight, and compact design. Li-ion batteries have become popular in the drone industry due to their ability to provide consistent power, relatively long flight times, and a favorable balance between energy capacity and weight. Li-ion batteries are characterized by three key parameters.

The first is the capacity, which measures the total energy storage of the battery in milliampere-hours (mAh). The second parameter is the "number of cells," denoted by the "S" value (e.g., 4S, 6S). Each cell has a nominal voltage of 3.7 volts, and the number of cells determines the overall voltage of the battery pack and thus the drone's performance. The third parameter, "C-rating," represents the battery's discharge rate capability in relation to its capacity. It specifies how quickly the battery can deliver power in amperes (A). A higher C-rating indicates the ability to provide more current.

Let's take a look at the battery of an Eiger :

Figure 2.3: Eiger's Li-ion battery : Foxtech's Diamound HV

This 12S Foxtech Diamond HV Series Li-ion Battery has a capacity of 22Ah. It thus can provide 22A of current for one hour. The 15 C-rating indicates that this battery can safely deliver a maximum continuous discharge current of 15 * 22 = 330A. This high C-Rating indicates that the battery can handle relatively high power demands, for instance while in multicopter mode.

Traditional Li-ion batteries typically have a discharge range for each cell, spanning from 3.2 volts to 4.2 volts. In contrast, a High-Voltage (HV) battery operates within a range of 2.7 volts to 4.35 volts per cell. This difference in voltage range translates to a substantial variance in energy capacity. For instance, a classic 12S 22Ah battery offers 976.8Wh of energy, while the HV variant provides a higher capacity of 1042Wh. With an average power consumption during transit of 700 watts, this translates to an extension of nearly 6 minutes of flight that represents 10 kilometers.

2.2.2 How to estimate Li-ion's battery percentage

Reading a Li-ion discharge voltage curve is essential for monitoring the battery's state of charge and understanding its performance during use. Here's one :

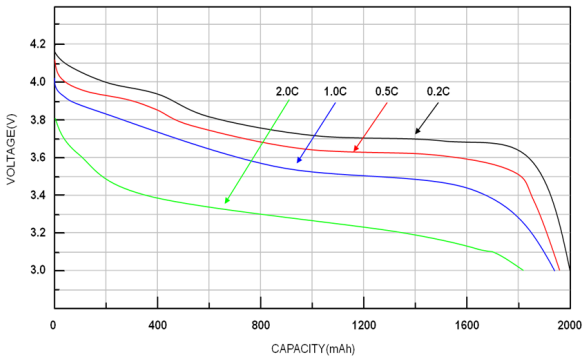


Figure 2.4: Typical Discharge Curves of a 2000mAh Li-Ion Battery [1]

The discharge voltage curve of a Li-ion battery can vary with the load current or the rate at which energy is drawn from the battery. High current draws, such as those required for acceleration or power-hungry applications, can result in voltage sag, causing temporary drops in voltage. This effect can lead to variations in the shape of the curve, especially during high-demand scenarios.

The voltage behavior of a Li-ion battery also changes with its state of charge (SoC). As the battery discharges, the voltage gradually decreases. Different parts of the curve correspond to different SoC levels. For example, the flat region in the middle of the curve represents the battery's usable capacity, while the steep drop towards the end signifies the battery reaching a critically low SoC. Measuring the battery's voltage provides a relatively quick and straightforward way to gauge its state of charge. However, it does not provide accurate SoC readings if it is not associated with a current measure to know the load current. This is why the Eiger is equipped with a voltage sensor as well as a current sensor.

2.2.3 PX4 estimation

PX4 has differents ways to estimate the SoC [5]:

- ***Basic Battery Settings*** : In the default settings, the raw measured voltage of the battery is compared to a range defined by "empty" and "full" voltages. However, this approach provides only rough estimates because the measured voltage and its corresponding capacity tend to fluctuate under load conditions.
- ***Voltage-based Estimation with Load Compensation***: This method is designed to mitigate the impact of load variations on capacity calculations. It adjusts the voltage-based estimation to account for the effects of different loads.
- ***Voltage-based Estimation with Current Integration***: In this approach, the load-compensated voltage-based estimation is combined with a current-based calculation of the consumed charge. This is the most accurate way to measure relative battery consumption and theoretically allow for accurate remaining flight time estimation.

Since the Eiger is equipped with a current sensor, we can use the latter. This approach assesses the remaining battery capacity by combining the voltage-based estimation of available capacity with a current-based calculation of the consumed charge.

I had to take care of converting the power board calibration data to make them consistent with the following PX4 parameters:

- BAT1_CAPACITY : Battery 1 capacity
- BAT1_R_INTERNAL : Defines the per cell internal resistance for battery 1
- BAT1_A_PER_V : Battery 1 current per volt (A/V)
- BAT1_V_DIV : Battery 1 voltage divider (V divider)

With the *Voltage-based Estimation Fused with Current Integration* method, the estimation of the consumed charge over time is derived by mathematically integrating the measured current, a method known for delivering highly precise energy consumption estimations.

Upon system startup, PX4 initially employs a voltage-based estimation to assess the initial battery charge. This estimation is then combined with the result of current integration, resulting in a more accurate combined estimate. The weight assigned to each estimate in the fused outcome varies with the state of the battery. As the battery's charge diminishes, a greater proportion of the voltage-based estimate is incorporated, which serves as a safeguard against deep discharge (e.g., due to incorrect capacity configuration or initial values).

If you consistently begin with a fully charged and healthy battery, this approach closely resembles the strategy used by smart batteries and that's the next RigiTech's step to improve the estimation of the SoC.

2.3 Other improvements

2.3.1 Sprint process

As Flight Review became increasingly subject to change, a single structure became obsolete. We therefore had to copy the structure that existed for the rest of the code, i.e. a development, test and production version on GitLab.

This raised a new issue : knowing which version of Flight Review was being called up depending on the server where the log was saved. I modified then the Dockerfile to call and print it when the Docker is launched.

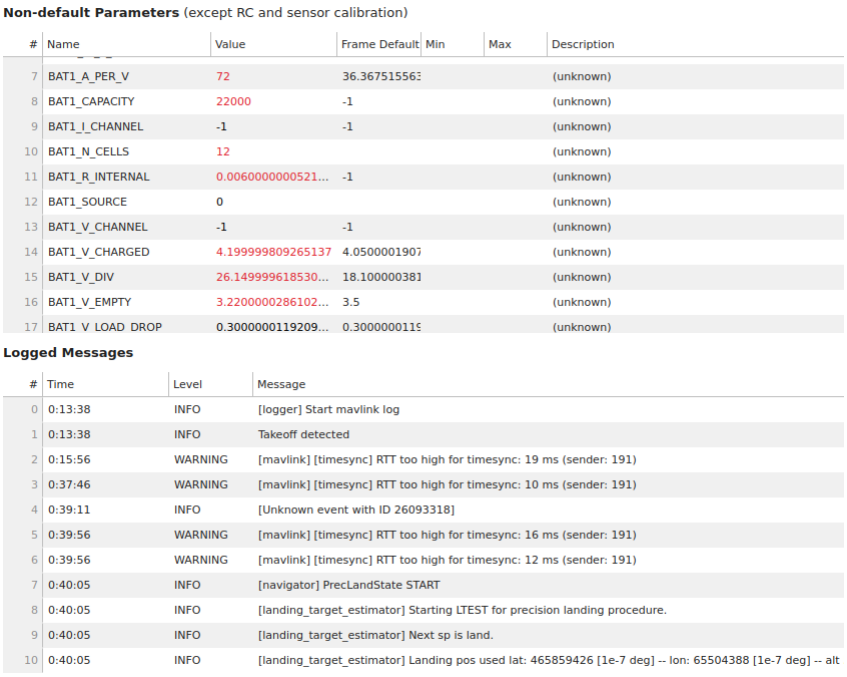
2.3.2 Failsafe parameters and logged messages

In the event of an emergency, such as the loss of the battery or leaving the flight zone, there is an independent failsafe that will trigger emergency actions, including the deployment of a reserve parachute.

Flight Review is one way of analysing a ULog file. Plot Juggler [6] is an other that will extract more raw data. This is why RigiTech has used the ULog type for its failsafe logs. Initially, it didn't seem appropriate to use Flight Review to analyse this data because this was more of a back-end job (not for customers) and PlotJuggler was sufficient and faster.

However, with the ongoing improvements to failsafe, it was thought useful for the client to also have an analysis of these logs. Unfortunately, as it was too different from original ULog files, these were not tolerated by Flight Review. I therefore had to develop a parallel version that could read this data.

An especially valuable aspect of a ULog file is the inclusion of recorded PX4 parameters from the flight, with a focus on identifying any parameters that have been altered.



Non-default Parameters (except RC and sensor calibration)

#	Name	Value	Frame Default	Min	Max	Description
7	BAT1_A_PER_V	72	36.367515563			(unknown)
8	BAT1_CAPACITY	22000	-1			(unknown)
9	BAT1_I_CHANNEL	-1	-1			(unknown)
10	BAT1_N_CELLS	12				(unknown)
11	BAT1_R_INTERNAL	0.0060000000521...	-1			(unknown)
12	BAT1_SOURCE	0				(unknown)
13	BAT1_V_CHANNEL	-1	-1			(unknown)
14	BAT1_V_CHARGED	4.199999809265137	4.0500001907			(unknown)
15	BAT1_V_DIV	26.149999618530...	18.100000381			(unknown)
16	BAT1_V_EMPTY	3.2200000286102...	3.5			(unknown)
17	BAT1_V_LOAD_DROP	0.3000000119209...	0.300000011			(unknown)

Logged Messages

#	Time	Level	Message
0	0:13:38	INFO	[logger] Start mavlink log
1	0:13:38	INFO	Takeoff detected
2	0:15:56	WARNING	[mavlink] [timesync] RTT too high for timesync: 19 ms (sender: 191)
3	0:37:46	WARNING	[mavlink] [timesync] RTT too high for timesync: 10 ms (sender: 191)
4	0:39:11	INFO	[Unknown event with ID 26093318]
5	0:39:56	WARNING	[mavlink] [timesync] RTT too high for timesync: 16 ms (sender: 191)
6	0:39:56	WARNING	[mavlink] [timesync] RTT too high for timesync: 12 ms (sender: 191)
7	0:40:05	INFO	[navigator] PrecLandState START
8	0:40:05	INFO	[landing_target_estimator] Starting LTEST for precision landing procedure.
9	0:40:05	INFO	[landing_target_estimator] Next sp is land.
10	0:40:05	INFO	[landing_target_estimator] Landing pos used lat: 465859426 [1e-7 deg] -- lon: 65504388 [1e-7 deg] -- alt

Figure 2.5: PX4 parameters in Flight Review

As you can see in red, the previously mentioned PX4 parameters have been updated for a more accurate SoC estimation.

The problem was that the parameters were only defined to be int, whereas the failsafe parameters were mostly string. I therefore sent the parameters to channel: 'Logged Messages', and then displayed them in an array corresponding to the failsafe parameters.

Chapter 3

Range Estimator

As mentioned above, RigiTech is starting to win more and more customers, and is present on several continents. The company has made a pretty huge start in proving its capabilities, but now needs to improve the robustness, resilience and reliability of its systems.

This means knowing whether the drone will be able to carry out a mission, both at the planning stage and in real-time. For example, if weather conditions have changed and the drone needs to turn towards a rally point to land safely. As you have seen, calculating the remaining capacity of a Li-Ion battery is no simple matter, so checking that PX4 gives a good estimate of the battery SoC was also part of this quest for reliability.

At some point, there was a huge debate about where to develop this Range Estimator. Either directly on the Navigation Node, or on the Cloud. The chosen solution was to develop a common library, where the linked mathematics could be used both on the Cloud and by the drone.

3.1 Principle

The Range Estimator has two goals :

- To know if the drone has enough battery to accomplish the mission (the route), whether it's during mission planning or in live.
- Calculate the maximum distance that can be flown before landing. You can then consider which Rally Point (defined further) you can go to if you do not have enough battery.

In either case, you need to know :

- The remaining route, its distance in multicopter, in fixed wings, its course, if you have to drop a payload...
- Some drone parameters : its speed, consumption, battery capacity...

Let's start by studying the route.

3.1.1 The route

A route is defined in PX4 by a JSON file. Here is an example of a route :

Figure 3.1: Take off waypoint from a JSON route

Each stage of the route (take off, transition, waypoint, landing) is defined by an item, characterized by parameters. Thus, each parameter can be retrieved and implemented in a C++ structure called *Waypoint* defined as follows:

```
struct Waypoint
{
    float x_lat;
    float y_long;
    float z_alt;                    // AMSL
    float terrain_altitude;         // AMSL
    cmd command;                    // Type of Waypoint (Take-off, Landing...)
    vtol_mission_state param1;      // Multicopter / Fixed Wings
    int param2;                     // Remaining loop(s) to do
    float course_rad;               // course from the WP to the next Waypoint
    float loop_course_rad;          // temporary course when doing a loop
};
```

We can thus represent a route with a vector of *Waypoint*

3.1.2 Computation of energy and time

By knowing certain parameters of the drone, such as its speed and power consumption, we can calculate the energy and time it will need to cover a certain distance.

$$\text{Duration (s)} = \frac{\text{Distance (m)}}{\text{Speed (m/s)}}$$

$$\text{Energy (Wh)} = \text{Power\_consumption (W)} \cdot \left(\frac{\text{Duration (s)}}{3600} \right)$$

Understanding the energy in Watt-hours (Wh) is valuable as it provides insight into the available energy stored in the battery using this particular unit of measurement.

3.1.3 The idea : flying backwards

Continuously calculating the remaining route to landing and deducting the energy and time needed to complete the mission seems very costly and not at all optimized. A much more intuitive way would be to say that the remaining route is equal to the route from the next waypoint plus the route to that waypoint. For each waypoint, we will calculate a *flight_dist* structure defined by :

```
struct flight_dist
{
    float MC_dist_m;          // remaining distance (m) in multicopter
    float MC_climb_m;         // remaining climb (m) in multicopter
    float MC_descent_m;       // remaining descent (m) in multicopter
    float FW_dist_m;          // remaining distance (m) in fixed wings
    float FW_climb_m;         // remaining climb (m) in fixed wings
    float FW_descent_m;       // remaining descent (m) in fixed wings
    float flight_time_s;      // remaining time (s) before landed
    float mean_course_rad;    // course angle from the waypoint to the last one
    vtol_mission_state VTOL_expected_state; // MC or FW
    int associated_wplist_idx; // index in the Waypoint vector (the route)
};
```

A vector of *flight_dist* vectors would suffice to calculate the remaining distances for each flight mode and subsequently determine the energy required to complete the mission.

An intuitive approach to calculate these structures would be to say that the *MC_dist_m*, which represents the remaining multicopter distance from the *associated_wp_list_idx* index waypoint, is equal to the *MC_dist_m* of the previous *flight_dist*, minus the distance between these two waypoints.

But I realized that this raised an issue, as we need to calculate the *MC_dist_m* of the previous waypoint. The problem is that initialization doesn't start at 0, but at the total distance to be covered, and therefore calculated. I had to start where the initialization would be equal to 0, i.e. the landing waypoint. Thus, the remaining route from a waypoint is the remaining route from the next waypoint, minus the distance between the last waypoint and the next one.

Basically, it's like flying backwards, starting with the landing. Of course, this is only possible and relevant during mission planning. In flight, the drone will already have recorded these estimates.

Subsequently, by applying the equations in 3.1.2, we can obtain a reasonably accurate estimate of the remaining battery capacity after the mission and assess whether it is sufficient.

The estimation outcomes were relatively accurate, with an approximate 10% error observed on a mission utilizing 50% of the battery capacity, which translates to a 20% margin of error. We'll see later how we were able to improve this estimate.

3.1.4 Rally points

If the battery is not sufficient, or if a problem has been encountered during the flight, RigiTech has developed the possibility of reaching a Rally Point defined during the mission planning. It may therefore be useful to look at things the other way round, and instead calculate the maximum distance that can be covered knowing our remaining capacity in order to determine which Rally Point the drone could reach.

So I coded an intermediate function that calculated the budget needed to make a back transition and therefore return by multicopter, make the distance needed by multicopter to go over the Rally Point and land. With this budget, I was able to know the maximum range in fixed wing that the drone was able to cover.

3.1.5 Approximations and improvements

For all these calculations, I used the equations in paragraph 3.1.2, which require knowledge of speed. The speed of the multicopter was held constant at 5 meters per second, a velocity that was effortlessly attained thanks to the drone's utilization of its pusher propeller in conjunction with its four other propellers. This was more complicated in fixed wing. For a number of reasons linked to RigiTech's different choices in the past, the drone's fixed-wing speed was set at 29.5 metres per second. However, this is an airspeed. The ground speed, needed for the calculus of the energy, was obviously different [7].

$$v_g = \sqrt{v_a^2 + v_w^2 - 2v_a v_w \cos(\delta - \omega + \alpha)}$$

Here we can identify:

- The ground speed v_g, the speed of the aircraft relative to the ground;
- The true airspeed v_a, the speed of the aircraft relative to the air it's traveling in;
- The wind speed v_w;
- The course δ, the planned direction of the plane;
- The wind direction ω;
- The wind correction angle α, the correction to the course to remain on route.

This required the calculation of the course between two waypoints, hence the addition of *course_rad* to the *Waypoint* structure. I also needed the average heading for the route in each *flight_dist* to calculate the budgets required for the maximum distance that could be covered with the remaining battery. By taking the wind into account in a very simple way, this has reduced estimation errors to around 5% on the same missions as before (in roughly similar conditions). The error has therefore been halved.

However, the 10% error rate is still too high, and here are some of the reasons why:

- There wasn't much available data on transition times and their energy consumption. When I attempted to collect data, the introduction of new drone batches featuring different and more powerful motors posed a potential source of data distortion.
- We observed that ground speed varied with the wind and factored that into our calculations. However, I assumed that the power consumption remained constant, which is, of course, an inaccurate assumption.
- I empirically gathered average consumption data by examining historical logs, probably even before the consumption sensors were calibrated.

There are several ways to address these issues and improve the Range Estimator:

- The development of an ongoing project at RigiTech is the *Batch Log Analysis*, which enables the simultaneous analysis of multiple logs, either for an individual drone or an entire fleet. This analysis allows for the derivation of consumption statistics and other debugging data.
- Taking into account the actual power consumption during the mission and subsequently updating the initial parameters would allow us to determine whether the drone consumes more or less under various conditions, such as rain or wind, or even if a motor is starting to age. This approach would also consider the payload weight, a factor that has not been considered so far.

3.2 Transfer to RigiCloud

Any C++ insider wouldn't encounter any specific issues when coding this Range Estimator using the standard library. However, the primary goal of this Range Estimator was to serve as a shared library between the drone and the Cloud, at least for the time being.

3.2.1 The solution : WebAssembly

"WebAssembly is a new type of code that can be run in modern web browsers — it is a low-level assembly-like language with a compact binary format that runs with near-native performance and provides languages such as C/C++, C# and Rust with a compilation target so that they can run on the web. It is also designed to run alongside JavaScript, allowing both to work together." [2]

Basically, WebAssembly (Wasm) enables the development of complex, high-performance web applications that were previously challenging to implement purely with JavaScript. It allows to code in C++ and run it into web applications like RigiCloud.

This is particularly interesting in the context of the Range Estimator because, in chronological order, I had already implemented a real-time version within the drone, coded in C++.

3.2.2 Emscripten

Emscripten is a compiler that takes your existing C++ code and compiles it into WebAssembly.

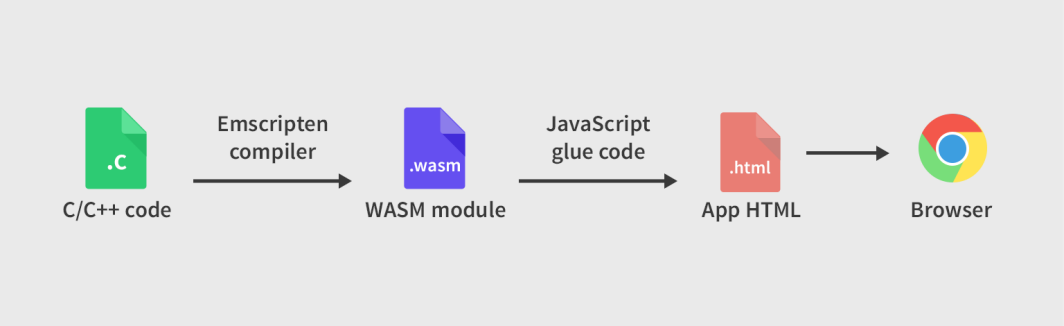


Figure 3.2: WebAssembly through Emscripten [2]

In my case, Emscripten will generate a .wasm module that will be imported into RigiCloud's JavaScript project. This module will define the functions I've chosen to export. For instance, my function *float remaining_battery_given_route*, which utilizes C++ data structures like vectors, will still be able to return the float value representing the remaining battery after the given route as an argument.

3.2.3 Spécifications

WebAssembly is powerful because it enables harnessing the performance of C++ code, but it does have its limitations. The imported module will have been defined as a library or will directly include the exported functions that one wishes to us. As I only needed a few functions, I didn't create any libraries.

However, it seems logical that not all functions can be exported. Those returning a type of structure not recognized in JavaScript (vectors, lists, strings...) are obviously part of this category. So, I had to modify the code to export only what interested me: a *float* representing the remaining battery.

Handling strings was also complicated because the wasm module couldn't directly receive a JSON representing the route. So, I had to delve into how strings was intrinsically represented in JavaScript to send the route to the Range Estimator.

It was necessary to allocate memory to be able to store the JSON's size, convert the string to UTF8, and send the address of the first character to the Wasm module.

On the C++ side, this involved an intermediary function that took this pointer to retrieve the JSON by translating the UTF8 back into a string. Then, I converted this JSON into a vector of *Waypoint* as defined in 3.1.1 to represent this route.

Finally, I successfully managed, with some modifications, to export my C++ code to be used on RigiCloud. What's even better is that with this Wasm library, each exported function can now be used wherever it seems useful and necessary. It was quite laborious, especially for testing my code because nobody at RigiTech was familiar with Wasm, and it required cross-functional skills: C++, JavaScript, and HTML, and no one possessed all these skills.

Chapter 4

Conclusion

This internship at RigiTech has improved the drone's robustness, accuracy and reliability. PX4 battery estimation has become more accurate. Flight Review log analysis has become more relevant for both developers and customers.

I'm proud to have been given responsibility for concepts as important as flight safety. Indeed, except in certain cases, the operator is still the main actor in emergency triggers. Once the Range Estimator has been properly tested and improved, the drone will be able to rely on it and take appropriate action.

The estimate still has errors, but these are taken into account. It was deployed on the test server when I left and should be deployed in the next few weeks on the production server.

I consider myself fortunate to have had the opportunity to work in a burgeoning startup, alongside a top-tier product that enjoys international success, and with a team of highly skilled and dedicated individuals. The process of integrating into an established team proved to be an intriguing experience, allowing me to appreciate the significance of coding with proper documentation. Naturally, the use of Git is both pertinent and essential.

Nonetheless, communicating with overseas developers in a language distinct from my native tongue presented a significant obstacle to the efficiency of my work. This challenge prompted a transformation in my perspective and comprehension of collaborative telecommuting.

Bibliography

[1] R. Technology, "Designing applications with li-ion batteries." https://www.richtek.com/battery-management/en/designing-liion.html.

[2] Tutorialzine, "Getting started with webassembly." https://tutorialzine.com/2017/06/getting-started-with-web-assembly.

[3] PX4, "Open source autopilot." https://px4.io/.

[4] PX4, "Flight review :fon open source autopilot." https://review.px4.io/.

[5] PX4, "Battery and power module setup." https://docs.px4.io/main/en/config/battery.html.

[6] PlotJuggler, "Fast, intuitive and extensible time series visualization tool." https://plotjuggler.io/.

[7] D. Borchia, "Ground speed calculator." https://www.calctool.org/kinetics/ground-speed.

Appendix A

range_estimator.hpp

```
/**
 * @file range_estimator.hpp
 * @brief Basic range estimator
 * @author Thomas Stauber, Martin Galliot
 */

#include "json.hpp"
#include <cmath>
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <list>
#include <stdint.h>
#include <string>
#include <tuple>
#include <vector>

#ifndef RANGE_ESTIMATOR_HPP
#define RANGE_ESTIMATOR_HPP

#define square(a) ((a) * (a))

typedef enum
{
    MC_state = 3,
    FW_state = 4,
    INIT = -1

} vtol_mission_state;

typedef enum
{
    NAV_WAYPOINT = 16,
    NAV_LAND = 21,
    NAV_TAKEOFF = 22,
    DO_VTOL_TRANSITION = 3000,
    DO_JUMP = 177,

} cmd;
struct flight_dist
{
    float MC_dist_m;        // remaining distance (m) in multicopter
    float MC_climb_m;       // remaining climb (m) in multicopter
    float MC_descent_m;     // remaining descent (m) in multicopter
```

```
    float FW_dist_m;          // remaining distance (m) in fixed wings
    float FW_climb_m;         // remaining climb (m) in fixed wings
    float FW_descent_m;       // remaining descent (m) in fixed wings
    float flight_time_s;      // remaining time (s) before landed
    float mean_course_rad;    // course angle from the waypoint to the last one
    vtol_mission_state VTOL_expected_state; // MC or FW
    int associated_wplist_idx; // index in the Waypoint vector (the route)
};



struct Waypoint
{
    float x_lat;
    float y_long;
    float z_alt;                  // AMSL
    float terrain_altitude;       // AMSL
    cmd command;                  // Type of Waypoint (Take-off, Landing...)
    vtol_mission_state param1;    // Multicopter / Fixed Wings
    int param2;                   // Remaining loop(s) to do
    float course_rad;             // course from the WP to the next Waypoint
    float loop_course_rad;        // temporary course when doing a loop
};

// JSON function
/** @brief Transform a json (pointer) to a vector of Waypoint
 *
 * @param json_ptr pointer to a json string
 * @return std::vector<Waypoint>
 */
std::vector<Waypoint> json_to_wpvector(const char *);

/** @brief Update drone's parameters with those in the json
 *
 * @param json_param
 */
void updateParams(const char *);

// Computations functions

/** @brief Compute ground speed in FW, in m/s
 *
 * @param wind_m_s wind in m/s. Positive means tailwind
 * @param wind_direction_rad North->South=pi, South->North=0
 * @param air_speed_m_s airspeed in m/s, mainly g_FW_cruise_airspeed_m_s
 * @param course_rad direction of the course
 * @return float
 */
float compute_fw_ground_speed_m_s(const float, const float,
                                  const float, const float);

/** @brief Compute the direction between two GPS points
 *
 * @param lat1d
 * @param lon1d
 * @param lat2d
 * @param lon2d
```

```
 * @return float
 */
float compute_course_angle_rad(const double, const double, const double,
const double);

/** @brief Compute the distance between two GPS points
 *
 * @param lat1d
 * @param lon1d
 * @param lat2d
 * @param lon2d
 * @return float
 */
float distanceEarthMeters(const double, const double, const double,
                          const double);

/** @brief Calculate the time (s) and consumption (Wh)
 *
 * @param dist in meters
 * @param rate in m/s
 * @param power in W
 * @return std::tuple<float, float>
 */
std::tuple<float, float> compute_time_s_and_consumption_Wh(
                               const float, const float, const float);

// Functions used for Planner
/** @brief Create a flight_dist from the route given
 *
 * @param json_route pointer to the json route
 * @param wind_m_s wind in m/s. Positive means tailwind (not use for now)
 * @param wind_direction_rad North->South=pi, South->North=0
 * @return flight_dist
 */
flight_dist initRemainingRouteMatrix(const char *, const float,
                                                        const float);

/** @brief Calculates takeoff, land, transition, mc climb, fw climb
    (ie all vertical budget needed for the r_route) fixed time and budget
 *
 * @param r_route
 * @return std::tuple<float, float>
 */
std::tuple<float, float> compute_vertical_budget_s_and_Wh(flight_dist);

/** @brief Calculates takeoff, land, transition, mc climb, fw climb
(ie all horizontal budget needed for the r_route) fixed time and budget
 *
 * @param r_route
 * @return std::tuple<float, float>
 */
std::tuple<float, float> compute_horizontal_budget_s_and_Wh(float,
                                                    float, flight_dist);

/** @brief Calculate the time and consumption in multicopter mode (MC)
needed for the r_route
 *
```

```
 * @param r_route
 * @return std::tuple<float, float>
 */
std::tuple<float, float> compute_mc_climb_and_descent_s_and_wh(flight_dist);

/** @brief Calculates the extra energy consumed to climb in fixed
                                    wing (FW) >0 in climb, <0 in descent
 *
 * @param r_route
 * @return float
 */
float compute_fw_total_extra_energy(flight_dist);

// Functions used for live estimation

/** @brief Computes the budget (s and Wh) to land, including a back
                                                    transition if in FW
 *
 * @param altitude
 * @param is_fw
 * @return std::tuple<float, float>
 */
std::tuple<float, float> compute_landing_budget_s_and_Wh(const float,
                                                         const bool);

/** @brief Gives the maximum range in FW including all transitions
                            and MC left and including the vertical budget
                            (eg to reach a landing point higher)
 *
 * @param usable_energy_Wh
 * @param r_route
 * @return std::tuple<float, float>
 */
std::tuple<float, float> max_fw_range_km_and_time_minutes_given_battery
                    (const float, const float, const float, flight_dist);

/** @brief Gives the maximum range in FW including only a back
                transition and the MC needed to land from the altitude)
 *
 * @param usable_energy_Wh
 * @param altitude
 * @param r_route
 * @return std::tuple<float, float>
 */
std::tuple<float, float>
max_fw_range_km_and_time_minutes_given_battery_and_alt_just_landing
        (const float, const float, const float, const float, flight_dist);

/** @brief Gives the maximum range in MC including only
        a back transition and the MC needed to land from the altitude)
 *
 * @param usable_energy_Wh
 * @param r_route
 * @return std::tuple<float, float>
 */
std::tuple<float, float> max_mc_range_km_and_time_minutes_given_battery
                                            (const float, flight_dist);
```

```
/** @brief Gives the maximum range in MC including the landing
from the altitude)
 *
 * @param usable_energy_Wh
 * @param altitude
 * @return std::tuple<float, float>
 */
std::tuple<float, float>
max_mc_range_km_and_time_minutes_given_battery_and_alt_just_landing
                                        (const float, const float);

// define functions as external, needed for Wasm
#ifdef __cplusplus
extern "C"
{
#endif

    /** @brief Computes the % of battery and the time needed to
                    do a route represented by a mavros waypoint list
     *
     * @param json_param char* pointer pointing to drone parameters
     * @param json_route string of the json route
     * @param wind_m_s float windspeed in m per s
     * @param wind_direction_rad float wind direction : North->South=pi,
                                                        South->North=0
     * @return a pointer to an array of two float
     */
    float remaining_battery_given_remaining_route(const char *,
                                sconst char *, const float, const float);

#ifdef __cplusplus
}
#endif

#endif // !RANGE_ESTIMATOR_HPP
```