Quantique et drones : La programmation

Pour les curieux, mais pas seulement…

10 février 2023

claude@lemarson.com
https://www.lemarson.com

Sommaire

Quantique et drones : la programmation pour les curieux, mais pas seulement…

- *Des domaines nouveaux et des compétences à acquérir*
- *Les concepts de base du calcul quantique*
- *Quel est l'intérêt des qubits ?*
- *Les algorithmes quantiques*
- *Les langages du quantique : assembleurs et évolués*
- *La double approche des drones : vols automatisés et autonomes*
- *Les connaissances à acquérir*
- *Les exemples des API DroneKit (3D Robotics)*
- *La crédibilité TI du programmeur de drones*

Fact.MR : le marché global des drones passera de 30 milliards $ en 2022 à 279 milliards $ en 2032 (vision très optimiste…).

Des domaines nouveaux
et des compétences à acquérir

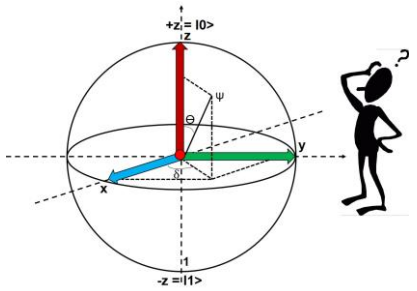
- Il existe des domaines nouveaux qui ont besoin de compétences en programmation.
- Qui nécessitent de passer par des préliminaires :
  - Compréhension minimale des grands principes de la mécanique quantique.
  - Compréhension de ce qu'est la navigation aérienne : on ne peut pas faire n'importe quoi.
- Voire triple difficulté avec la nécessité de maîtriser certains aspects mathématiques.
- Une "respiration" nouvelle pour les développeurs, confinés depuis toujours sur des applications classiques avec les mêmes langages (avec des exceptions : fonctionnel…) : ça change.
- La programmation quantique va satisfaire les scientifiques à la recherche de nouveaux formalismes, tout en gardant leurs langages traditionnels et les codeurs de drones pourront se fonder sur une diversité d'applications que l'on ne fait qu'aborder.
- C'est surtout l'intérêt et la curiosité des développeurs qui est mise en avant : il est plus "fun" de coder une livraison de pièces détachées dans une usine que de développer une Nième gestion des stocks.
- Une différence : le développement du quantique n'est pas assuré alors que les drones vont se répandre, dans un contexte d'IoT : la 4 ème génération du TI.

La programmation quantique…
les mathématiques et la physique

- Le cheminement vers le codage quantique est à peu près le même que celui de la programmation classique : des langages assembleurs qui mettent en musique l'enchaînement des portes logiques, des langages plus évolués pour prendre de la hauteur par rapport à ces éléments physiques et des API pour des fonctions spécifiques quantiques, qu'en principe on n'est pas obligé de comprendre.
- Ce n'est pas l'évolution logique de ce que l'on pratique, même si certains fondamentaux sont les mêmes.
- L'expérience montre que le codage quantique c'est comme la musique : si on ne connaît pas le solfège, on multiplie les obstacles, un certain niveau de mathématiques et de compréhension de la mécanique quantique s'avèrent utiles, voire nécessaires.
- Avec les machines traditionnelles, qui sont un assemblage de portes logiques, combinées par logiciel ou micro-code pour exécuter des opérations booléennes, on peut se contenter de savoir ce qu'elles savent faire. Autrement dit, on ne s'en préoccupe pas…
- En programmation quantique, les obstacles étant déjà importants, on n'est pas obligé de s'en créer un de plus et comprendre par le détail à qui servent les portes logiques… mais ça aide.
- Les mathématiques sont omniprésentes et un minimum de compréhension est indispensable : algèbre linéaire et transformations matricielles.
- Deux approches :
  - Niveau "matériel", avec des assembleurs "proches" des portes logiques.
  - Niveau plus élevé, avec des langages susceptibles de les mettre en œuvre et d'exécuter des séquences opérationnelles sur les qubits.

Des principes (difficiles) à comprendre

- Comme le dit son nom, la programmation quantique exploite certaines caractéristiques de la mécanique quantique
- Principe de superposition de Schrödinger (symbole du chat)
  - Une particule peut présenter simultanément plusieurs états quantiques…
  - Une machine quantique exploite ce principe qu'il traduit dans les qubits.
- Le qubit peut être un "0", un "1" ou une combinaison dite probabiliste de ces états de base : un peu de 0 et beaucoup de 1, par exemple, le tout évidemment en même temps.
- Ce que l'on résume très "clairement" par la formule : **αI0> + βI1>**
  - 0 et 1 étant les états de base et α et β les probabilités d'existence de ces états, reliées par l'équation $\alpha^2 + \beta^2 = 1$. Si α = 1 et β = 0, l'état du qubit sera 0. Les αI0> et βI1> sont appelés ket 0 et ket 1, par analogie avec les bits 0 et 1.
- En résumé : en plus des états de base, 0 et 1, on aura une infinité de superpositions possibles de ces états entre ces 2 extrêmes. Etats que l'on représente souvent dans une sphère dite de Bloch, avec des phases différentes.
- En fait, l'unité quantique qubit est une représentation des probabilités d'existence des états de base et de leur superposition. Ouf !!!



Un état quantique de qubit est représenté par un point ψ sur la sphère dite de Bloch (ou de Poincaré), telle que |ψ> = α|0> + β|1

Quel est l'intérêt des qubits ?

- Avec une machine classique avec des 0 et des 1, un algorithme change ces états au fil des calculs, ce qui demande du temps, car il n'y en a que 2 par bit.
- Avec une machine quantique à 1 qubit, on dispose immédiatement d'un nombre considérable d'états différents, pour chacun d'eux… sans effectuer la moindre transformation.
- Une machine à 2 qubits expose 4 états distincts (00, 01, 10 et 11), plus une infinité de superpositions :

αI00> + βI01> + γI10> + δI11>, cette fois avec $\alpha^2 + \beta^2 + \gamma^2 + \delta = 1$

- Avec 2 qubits on aura donc le double d'états superposables et on admettra que la puissance de la machine aura été multipliée par 2.
- Avec 3 qubits, il y aura 8 états différents : 000, 001, 010, 011, 100, 101, 110, 111 qui se superposent en :

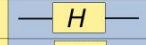
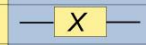
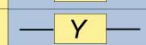
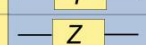
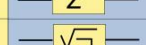
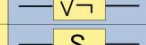
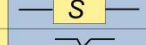
- **αI000> + βI001> + γI010> + δI011 + εI100> + θI101> + λI110> + μI111** toujours avec $\alpha^2 + \beta^2 + \gamma^2 + \delta^2 + \varepsilon^2 + \theta^2 + \lambda^2 + \mu^2 = 1$

- On imagine la puissance d'exécution quand on augmente le nombre de qubits.
- Quant à la conception d'une machine quantique, c'est une autre compétence, dont nous n'avons pas besoin pour le codage.
- Avec un mot binaire de n bits, on ne dispose que de 2n états, avec beaucoup de travail pour passer de l'un à l'autre, alors qu'avec une machine quantique à n qbits on a instantanément un nombre quasi infini d'états. C'est toute la différence.
- Mais encore faut-il y accéder de manière continue sans dégradation…

L'arlequin LeMarson multicolor symbolise une machine quantique par rapport à une "simple" machine binaire.

Les algorithmes quantiques

- L'algorithmique quantique est fondée sur l'usage d'opérateurs spécialisés, des portes logiques, comme en informatique classique, mais avec des portes différentes, dérivées de la mécanique quantique, qui agissent toutes sur 1 ou plusieurs qubits.
- Chacune des portes correspond à une transformation mathématique matricielle ou tensorielle (attention, le calcul tensoriel est particulièrement difficile d'approche).
- Le codage d'une application revient à combiner ces portes logiques en fonction de ce qu'elles savent faire, pour construire un circuit quantique.
- Les portes effectuent des opérations logiques de changement d'états (éventuellement).
- Elles sont représentées par des matrices unitaires de dimensions 2n x 2n ou n est le nombre de qubits utilisé.
- Evidemment, le fait de connaître la différence entre une matrice et un ballon de soccer, peut nous aider un peu…

Porte quantique	Symbole	Signification
Hadamard		Action sur un seul qbit. Version qubit de la transformée de Fourier.
Pauli-X		Agit sur un seul qubit. Transforme l'état |0> en |1> et l'état |1> en |0>. C'est le bit-flip, équivalent du NOT classique. Equivaut à une rotation de π autour de l'axe X dans la sphère de Bloch..
Pauli-Y		Agit sur un seul qubit. Transforme l'état |0> en |1> et l'état |1> en -|0>. Equivaut à une rotation de π autour de l'axe Y dans la sphère de Bloch.
Pauli-Z		Agit sur un seul qubit. Laisse l'état de base |0> inchangé et l'état |1> en -|1>. Equivaut à une rotation de π autour de l'axe Z dans la sphère de Bloch. On l'appelle la porte phase-flip.
Racine carrée de NON		Action sur un seul qbit. Des portes de racibnes carrées peuvent être construites pour toutes les portes. Pour chacune d'elles, c'est une matrice unitaire, qui multipliée par elle-même, donne cette porte.
Changement de phase		Action sur un seul qbit. Laisse l'état |0> inchangé et modifie la phase de l'état |1>. Exemples courants avec des décalages de π/4, π/2 et π.
Porte SWAP		Intervertit 2 qubits.
Contrôlées X, Y, Z		Agissent sur 2 qubits ou plus. 1 ou plusieurs qubits se comportent comme des contrôleurs d'opérations portant sur les qubits restants. La porte NOT agit sur 2 qubits et effectue le NOT qu'après condition.
Toffoli (CCNOT)		Appelée aussi CCNOT, porte sur 3 qubits. Si les 2 premiers sont dans l'état |1>, elle exécute un NOT sur le 3ème. C'est une porte contrôlée, car spécifiée par une table de vérité.
Fredkin (CSWAP)		La porte de Fredkin (CSAP ou cS) est une porte 3 qubits qui effectue un SWAP contrôlé..

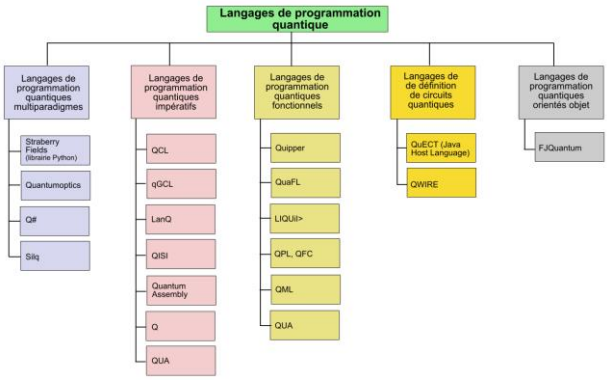
Les langages du quantique

L'assembleur

- Assembleur QASM ("Quantum Assembly Language"), un mélange de formalismes C et assembleur :
- Peu d'instructions, dont la plupart sont compréhensibles et différent peu d'une syntaxe connue.
- Son objectif est d'être en prise directe avec la construction des circuits quantiques, fondés sur des enchaînements de portes
- On utilise un éditeur, celui implémenté par IBM dans sa "Q Experience", étant le plus connu.
- Il permet d'accéder à des registres que l'on remplit avec des qubits par la commande qreg :

```
qreg lemarson[2] ; // le registre quantique lemarson comporte
2 qubits 0 et 1
qreg autre[2] ; //le registre quantique autre contient 2
qubits 2 et 3
```

- A partir de QASM, on a accès à toutes les transformations des portes logiques.
- Il existe plusieurs versions ouvertes de QASM : OpenQASM 3, étant la dernière, que l'on doit toujours à IBM.
- D'autres assembleurs sont envisageables, plus exotiques et très universitaires dans l'esprit, que l'on pourra ignorer.



Les langages quantiques évolués

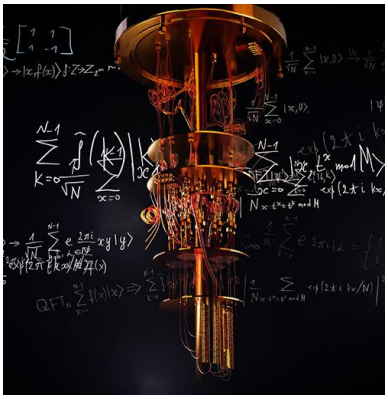
- Langages évolués : marché déjà très fourni.
- **QCL** ("Quantum Computing Language") est l'un des plus connus, que l'on doit à Emanuel Knill.
- Langage de type C, qui manipule aussi bien des données classiques, que spécifiquement quantiques telles que **qureg** (tableau de qubits), **quvoid**, **quconst**, **quscratch**, **qucond**.
- QCL s'appuie sur des fonctions quantiques intégrées, les qufunct, mais aussi sur des opérateurs spécialisés et des procédures telles que measure, dump ou reset.
  Le dump est très intéressant qui permet de connaître l'état des qubits sans les observer, mais en utilisant un simulateur qui ne modifie pas leur état (exemple Wikipedia) :

```
qcl> dump
: STATE: 4 / 32 qubits allocated, 28 / 32 qubits free
0.35355 |0> + 0.35355 |1> + 0.35355 |2> + 0.35355 |3>
+ 0.35355 |8> + 0.35355 |9> + 0.35355 |10> + 0.35355 |11>
```

- Il a l'avantage de supporter des fonctions et opérateurs définis par l'utilisateur.

```
operator diffuse (qureg q) {
  H(q);                 // Hadamard Transform
  Not(q);               // Invert q
  CPhase(pi, q);        // Rotate if q=1111..
  !Not(q);              // undo inversion
  !H(q);                // undo Hadamard Transform
}
```

- On peut mélanger du code QCL avec du C.
- Q, est un autre langage impératif très connu. Conçu comme une extension de C++, il fournit les classes pour les traitements quantiques de base : QHadamard, QFourier, QNot, QSwap, dérivées de la classe Qop.
- Silq, très récent (date de 2020), a été écrit en D et attire de plus en plus d'usagers.
  Conçu à l'université de Lausanne (comme Scala), il est réputé être simple en écriture et lecture et plus compact que ses principaux concurrents, tels que Q#.



Q# de Microsoft

- Q# de Microsoft, a été conçu pour être exploité avec le "Quantum Development Kit".
- Open Source, il comporte des bibliothèques et des simulateurs quantiques.
  Il peut être utilisé de manière autonome ou en s'appuyant sur un langage hôte, comme Python ou C++.

```
@EntryPoint()
operation MeasureOneQubit() : Result {
// The following using block creates a fresh qubit and initializes it
// in the |0⟩ state.
use qubit = Qubit();
// We apply a Hadamard operation H to the state, thereby preparing the
// state 1 / sqrt(2) (|0⟩ + |1⟩).
H(qubit);
// Now we measure the qubit in Z-basis.
let result = M(qubit);
// As the qubit is now in an eigenstate of the measurement operator,
// we reset the qubit before releasing it.
if result == One { X(qubit); }
// Finally, we return the result of the measurement.
return result;
}
```

- Autres solutions :
  - Langage script qScript de Google, fait partie de l'IDE de ce prestataire ("Quantum Computing Playground")
  - Le plus récent QUA de Quantum Machines
    - Présenté comme le seul outil universel, capable de s'adapter à tous les environnements matériels quantiques.
    - Se situe à un niveau d'abstraction plus élevé que ses concurrents et peut être compilé avec XQP, un compilateur maison, qui génère du code assembleur, dit "pulse-impulsion".
    - QUA est disponible sur la plate-forme "Quantum Orchestration Platform" et très adapté à la recherche.

Drones : une double approche

Le drone est un aéronef comme un autre

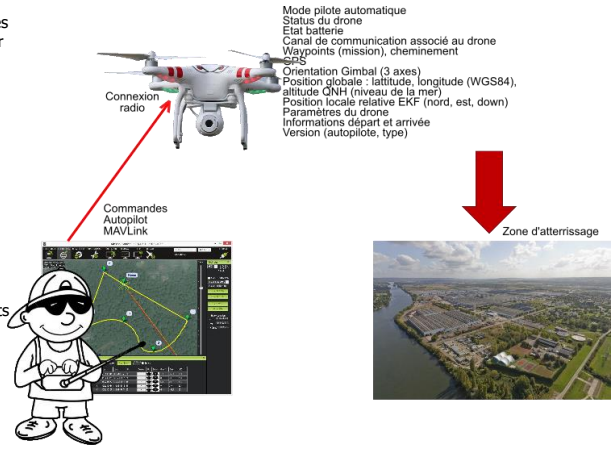
- On achète un kit clé en main, qu'il suffit d'activer : matériel et logiciel de commande, que l'on se contente de paramétrer.
- On achète un drone que l'on va programmer en fonction d'une mission précise :
  - Un métier nouveau apparaît de "pilote" de drone.
- Les drones constituent un danger potentiel grave : choc des objets.
- La règlementation diffère selon les pays. L'espace est partagé entre les tous les aéronefs et les usagers doivent en respecter les contraintes.
- On distingue les vols automatisés des vols autonomes.
  - En France, les vols libres (autonomes) sont interdits, comme au Québec.
  - Seuls des vols automatisés très limités en distance et altitude sont possibles : vue, portée de 100 m, altitude de 150 m (uniquement pour des zones hors contraintes).
- Le concept de vol intelligent :
  - Des fonctions de complément à la programmation et vols, qui peuvent s'avérer utiles en fonction des circonstances.
  - Retour au point de départ sans intervention (ex : perte de contact des radiocommandes).
  - Mode "follow me" : pour suivre de manière automatique le pilote qui émet les commandes.
  - Certains drones sont équipés d'outils pour sélectionner et suivre des personnes précises.
- Un métier nouveau apparaît de "pilote" de drone.

Drones : les connaissances à acquérir

- On peut achèter un kit clé en main, qu'il suffit d'activer : matériel et logiciel de commande, que l'on se contente de paramétrer.
- Un drone professionnel doit comporter un système de navigation, des caméras et des capteurs dédiés pour éviter les obstacles et une logique de traitement, susceptible d'effectuer des calculs temps réel en fonction de l'évolution de l'environnement.
- Les connaissances à acquérir :
  - Règlementation.
  - Principes de météo.
  - Positionnement.
  - Gestion des commandes, avec ou sans assistance.
  - Décollage et atterrissage.
  - Roulis, tangage, lacet.
  - Caps magnétiques et géographiques.
  - Effets du vent (dérive).
  - Vol sur une route balisée (radionavigation).

L'API Dronekit

- **DroneKit :** API Python de 3D Robotics, le premier fabricant américain de drones, dont les applications vont s'exécuter sur la carte mère du drone, pilotées par un contrôleur de vol **ArduPilot**.
- La communication entre les deux éléments est fondée sur le protocole **MAVLink**, très répandu dans le monde des véhicules dits « unmanned », c'est-à-dire non pilotés par un être humain.
- La liaison physique entre le drone et le mobile de commande peut être une simple liaison Wi-Fi, mais plus souvent le système radio 3D Robotics.
- L'API est installée sur une machine Linux, Windows ou Mac OSX, mais l'application finale est portée sous Android ou iOS.
- Le développeur a accès à de nombreuses fonctions :
  - Etat du drone
  - Paramètres de vol
  - Tâches à effectuer : prise de vues, suivi livraison, atterrissage…. Les informations étant encapsulées dans des messages **MAVLink**.
- Le drone peut être placé en pilotage automatique, suivre une route marquée par des points de contrôle GPS que l'on précisera, contrôler les paramètres de décollage et d'atterrissage, éviter les mouvements brusques en prévoyant des trajectoires souples, se mettre en orbite autour d'un point précis, scanner des volumes importants avec la fonction **« Automatic Building Mapper »** et suivre un objet en mouvement, que le drone ne lâchera pas et « mitraillera » avec sa caméra, sous différents angles.
- L'API **DroneKit** est complexe à mettre en oeuvre.
- 2 étapes dans ce développement :
  - Ecriture du code Python sur une machine locale, que l'on connecte à un simulateur **SITL**, sur la même machine ou non, pour vérifier son comportement.
  - La 2 ème étape concerne l'implantation dans l'objet volant du code exécutable et son contrôle réel via **MAVLink** depuis un mobile.
  - Ne pas oublier que le drone peut avoir un comportement erratique et qu'il vaut mieux être prudent

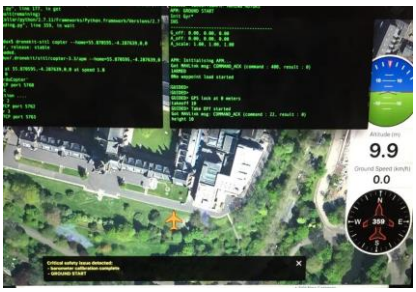


Le codage avec DroneKit

- Le développeur doit traiter 4 phases :
  - la connexion au « **Vehicle** »
  - la récupération des paramètres du « Vehicle » et leur modification pendant les missions
  - le décollage
  - le guidage et le contrôle du vol.
  - La connexion à 1 ou plusieurs « **Vehicles** » passe par un script Python, avec la méthode **connect()**, qui génère un objet **vehicle** :

```
from dronekit import connect
# Connexion au drone, ici par un endpoint UDP
vehicle = connect('127.0.0.01:14550', wait_ready=True)
```

- où le paramètre **wait_ready** précise que la connexion attend qu'un certain nombre d'attributs soient renseignés (la vitesse en bauds, la durée de la connexion, etc).
- Pendant une « mission », le programmeur a accès à une vingtaine de paramètres, à travers la classe **Vehicle** :
  - **velocity**, la vitesse exprimée avec 3 attributs, vx, vy et vz, les vitesses en m/s sur les 3 axes
  - **groundspeed**, **airspeed**, un « **heartbeat** » de messages **MAVLink**, **gimbal**, **attitude**, **location.global_frame**, **location.local_frame**, **battery**, etc.
- Pour le décollage, le code s'assure que le drone peut être « armé », avant de le mettre en mode **GUIDED**.
- Pendant le vol, qui sera « **GUIDED** », le code Python contrôler les paramètres essentiels : la position, la vitesse, etc. Qui seront toujours envoyés à la console de commande par des messages **MAVLink**.




```
# position control
vehicle.mode = VehicleMode("GUIDED")
# Set the target location in global-relative frame
a_location = LocationGlobalRelative(-34.364114, 149.166022, 30)
vehicle.simple_goto(a_location)
# send command to vehicle on 1 Hz cycle
    for x in range(0,duration):
        vehicle.send_mavlink(msg)
        time.sleep(1)
```

D'autres API et SDK sont disponibles : DJI (Chine), basé sur la même idée que DroneKit mais sans être limité à Python et à un OS mobile.
On a accès en temps réel à tous les paramètres de vol : décollage, atterrissage, croisière, altitude, vitesse, point de départ, destination, etc.
DroneCore et nombreux projets Open Source.

La crédibilité TI du codeur de drones

- Les usages des drones vont considérablement se développer :
  - Applications dans les conflits (ex Ukraine).
  - Valorisation du patrimoine.
  - Apprentissage du pilotage d'avions : "flight simulator moderne".
  - Réalisation de films, documentaires et publicités.
  - Surveillance et sécurité, particulièrement pour les zones à risques et difficiles d'accès.
  - Développement de l'agriculture : reconnaissance de l'hétérogénéité du sol, examen de la densité végétale, connaissances des dégâts causés par les eux, les incendies, la sécheresse…
  - Livraison de colis et applications de distribution.
  - Livraison rapide de produits urgents (médical).
  - Surveillance et sécurité sous toutes les formes.
  - Inspection des voies de communication : routes, chemins de fer…
- On aura besoin de spécialistes de drones et le codage jouera un grand rôle.
- Une compétence à ajouter à l'entreprise (pas nécessairement au TI) et à prévoir dans les schémas directeurs.
- Avantages : c'est nouveau, amusant, utile.

Les futurs avions seront sans pilotes : les drones ne font que les préfigurer.

Quantique et drones : La programmation

10 Février 2023

Nos prochains webinaires

17 février 2023	**La vie privée, c'est fini**
24 février 2023	**L'affrontement des processeurs : x86 contre ARM**
3 mars 2023	**Machines virtuelles contre conteneurs**
10 mars 2023	**La distribution de demain, une révolution**
17 mars 2023	**Les grands criminels de la cybersécurité**
24 mars 2023	**L'enseignement de demain, comment abétir les enfants**
31 mars 2023	**Santé et ondes : soyons sérieux**
7 avril 2023	**Oracle, Intel et IBM, colosses aux pieds d'argile**
14 avril 2023	**Ethereum, au cœur de l'Internet de demain**
21 avril 2023	**Backup et restauration des datacenters**
28 avril 2023	**Pourquoi l'IA est-elle stupide ? Elle nous imite**
5 mai 2023	**L'hyperautomation : les temps modernes du TI**
12 mai 2023	**Quand la biométrie sort des sentiers battus**
26 mai 2023	**Productivité, il n'y a pas qu'Office. Ah, bon ?**
2 juin 2023	**Les grandes utopies du TI : capitaliser sur nos erreurs**
9 juin 2023	**Les transports du futur : verts et sans pilotes**
16 juin 2023	**Sécurité : les reproches faits à la suite TCP/IP**
23 juin 2023	**La fédération d'identités : "you will never walk alone..."**
30 juin 2023	**Les grandes figures du TI... dont on parle moins**

claude@lemarson.com
https://www.lemarson.com