



INSTITUT UNIVERSITAIRE DE TECHNOLOGIE DE BÉZIERS

MÉMOIRE LICENCE PROFESSIONNELLE DE L'INTERNET DES OBJETS

**COMMENT L'IOT, PEUT-IL PERMETTRE UN SUIVI
DÉTAILLÉ ET UNE POTENTIELLE PRÉSERVATION DE
LA FAUNE ET DE LA FLORE EN MILIEU FORESTIER ?**

Par Buddy-Lee CELDRAN-SCHWARZ : Étudiant en Licence professionnelle de l'Internet des Objets

Tuteur académique : M. SEBASTIEN DRUON

**IUT de Béziers 3, place du 14 juillet BP 50438
Béziers cedex Hérault 34505 France**

Année universitaire 2019-2020



Année universitaire 2019-2020

MÉMOIRE LICENCE PROFESSIONNELLE DE L'INTERNET DES OBJETS

COMMENT L'IOT, PEUT-IL PERMETTRE UN SUIVI DÉTAILLÉ ET UNE
POTENTIELLE PRÉSERVATION DE LA FAUNE ET DE LA FLORE EN
MILIEU FORESTIER ?

Présenté par **Buddy-Lee CELDRAN-SCHWARZ**

Numéro d'étudiant : 21700012

Sous la direction de **Sebastien DRUON**, Enseignant chercheur.

Préface :

Ce mémoire rentre dans le cadre de l'obtention du diplôme de Licence professionnelle de l'Internet des Objets. Il étudiera le fonctionnement de L'IOT dans le suivi et dans la préservation de la faune et de la flore dans le milieu forestier. L'idée de ce mémoire est venue du faite que le sujet du stage initialement prévu, portait sur la smartforest qui consistait à un suivi de l'évolution des forêts, de plus la préservation des forêts est un attendu majeurs avec les nouvelles technologies.

En effet, la préservation de la faune et de la flore est une préoccupation majeure, avec l'arrivé de l'Internet des Objets, il est devenu possible de suivre une évolution et d'agir en conséquence s'il y a nécessité.

Ce mémoire se veut être une contribution afin de mettre en avant les avancés faits pour la préservation de la faune et de la flore, tout en apportant une solution ou une amélioration sur des projets existants.

Difficultés :

Les difficultés rencontrées durant ce mémoire sont variées, en particulier au niveau des projets existants qui ne fournissent pas toutes leurs documentations (ce qui est compréhensible).

Au niveau de la partie technique, les difficultés qui se sont fait ressentir sont au niveau de la recherche de solutions, afin d'éviter un maximum de reprendre des solutions déjà existantes dans ce domaine. Mais aussi lors de la conception de notre solution. Car mettre en place une simulation afin de montrer le fonctionnement avec le matériel que l'on a sous la main n'est pas forcément évident pour tous, et aussi créer une solution sans avoir là le matériel adéquat a été un frein. La solution qui est proposée est à moitié mise en pratique et à moitié théorique.

Remerciement :

Je tiens à remercier toutes les personnes qui m'ont aidée lors de la rédaction de ce mémoire.

Je voudrais dans un premier temps remercier, mon tuteur pour ce mémoire M.DRUON, enseignant chercheur à l'IUT de Béziers, pour sa patience, sa disponibilité et surtout ses judicieux conseils, qui ont contribué à alimenter ma réflexion.

Je remercie également toute l'équipe pédagogique de l'IUT de Béziers, pour avoir assuré la partie théorique de la formation.

Résumé :

L'Internet des objets (IDO) dans sa globalité consiste principalement à connecter des objets physiques à l'Internet. L'émergence de ce phénomène permet le développement de nouveaux services et applications permettant la connexion entre le monde physique et le monde virtuel, un processus qui n'était pas possible auparavant. Bien que beaucoup de services et d'application ont vu le jour dont la plupart se retrouvent dans le domaine de la domotique, de la santé, de la Smart city ou encore dans le domaine de l'agriculture et de l'automobile, le domaine forestier est assez peu proposé.

Dans ce mémoire, nous essaierons de présenter, les différents projets existants autour du domaine forestier (faune et flore). Nous essaierons par le biais de la partie technique, de proposer une solution de surveillance, au travers d'objets connectés, afin de préserver la faune et la flore.

Sommaire

Préface :	3
Remerciement :	4
Résumé :	5
Liste des tableaux et graphiques :	7
Liste des abréviations :	8
Glossaires :	9
1. Introduction	11
2. Internet des objets	13
2.1 Sa définition	13
2.2 Son fonctionnement	14
2.3 Les protocoles réseaux utilisés	16
3 Le lien avec la faune et la flore	20
3.1 Les différents projets existants pour la préservation de la faune et de la flore	21
Chapitre 4 : Partie Tech	27
Simulation d'une situation de monitoring dans une forêt	27
Configuration de l'environnement	28
1.1 Le broker MQTT	28
1.2 La base de donnée InfluxDB	29
1.3 Télégraf	30
1.4 Grafana	32
2. Redirection des ports :	34
3. Test MQTT	34
Alertes :	40
Conclusion de la simulation :	43
Partie 2 : Utilisation d'ESP32 pour la récupération des informations des capteurs	43
1. Capteur de températures	44
2. Niveau CO2 dans la forêt	46
3. Capteur d'humidité et pluviométrie	48
4. Niveau rivières ou ruisseaux	49
5. Partage des données vers un serveur	51
6. Alimentations des dispositifs	54
1. Le cas de l'ESP32	54
2. Le cas du Raspberry PI	57
Détection d'éléments avec OpenCV et Yolo	59
Envoi des alertes :	65
Suivi des animaux en forêt	67
Conclusion de la partie technique :	68
Conclusion :	69
Annexes :	70
Sources :	74
Bibliographie :	75
Citations :	75

Liste des tableaux et graphiques :

Index des Figures

Figure 1: Nombre d'objets connectés dans le monde (Cisco Mars 2020).....	11
Figure 2: Fonctionnement objets connectés (ConnectWave).....	14
Figure 3: Cheminement des données (ConnectWave).....	14
Figure 4: Exemple d'architecture réseau ZigBee.....	17
Figure 5: Schéma du principe d'une architecture Wi-fi au niveau domestique.....	19
Figure 6: Architecture réseau du projet SmartForest.....	21
Figure 7: Schéma fonctionnement du projet paru dans IJCTER.....	22
Figure 8: Fonctionnement de la solution (IoTree).....	23
Figure 9: Exemple de l'analyse des sons.....	24
Figure 10: Dispositif Rainforest.....	24
Figure 11: Fonctionnement de la solution chez Rainforest.....	25
Figure 12: Schéma de la simulation.....	27
Figure 13: Paramètres BDD.....	33
Figure 14: Requêtes.....	33
Figure 15: Adresse serveur.....	34
Figure 16: Redirection sur un port spécifique.....	34
Figure 17: Interfaces clé 4G et point d'accès.....	35
Figure 18: Paramètres du point d'accès.....	35
Figure 19: Requête puissance en dBm.....	38
Figure 20: Requête signal Wi-Fi.....	39
Figure 21: Requête niveau de batterie.....	39
Figure 22: Requête niveau de température du dispositif.....	39
Figure 23: Vue globale des différents composants.....	40
Figure 24: WebHook serveur Discord.....	40
Figure 25: Configuration notifications Discord.....	41
Figure 26: Règle pour l'alerte de température.....	41
Figure 27: Paramètre du message.....	42
Figure 28: Message reçu serveur discord.....	42
Figure 29: Branchement DHT11 sur ESP32.....	44
Figure 30: Branchement capteurs CO et CO2.....	47
Figure 31: Branchement capteur d'humidité (Circuito.io).....	48
Figure 32: Branchement capteur ultrasonique (Circuito.io).....	50
Figure 33: Tableaux du choix des panneaux solaires.....	57
Figure 34: Tableaux valeurs Raspberry PI.....	58
Figure 35: Détection dog.jpg.....	59
Figure 36: Chiens.jpg.....	60
Figure 37: Detection chiens.jpg.....	61
Figure 38: Retour image OpenCV et Yolo.....	64
Figure 39: Webhook OpenCV.....	65
Figure 40: Retour messages sur Discord.....	66
Figure 41: Schéma placement des caméras.....	67
Figure 42: ébauche collier GPS.....	68

Index des Tableaux

Tableau 1: Tableau des débits et portées de la technologie Wi-Fi en fonction de la norme.....	18
Tableau 2: Tableau des alertes.....	22
Tableau 3: Niveau de Co2 et impact sur l'Homme (th-industrie).....	23
Tableau 4: Niveau monoxyde de carbone et ses dangers.....	46

Annexe

Illustration 1: Code OpenCV Yolo.....	70
---------------------------------------	----

Liste des abréviations :

IOT : Internet of Things

IDO : Internet des Objets

GSM : Global System for Mobile

GPS : Global Positioning System

M2M : Machine to Machine

Wi-Fi : Wireless Fidelity

4G : LTE/LTE-A

MQTT : MQ Telemetry Transport / Message Queuing Telemetry Transport (IBM)

Glossaires :

Gateways : dispositif permettant de relier deux réseaux distincts présentant une topologie différente.

Routeur : équipement en réseau informatique assurant le routage des paquets. Son rôle est de faire transiter des paquets d'une interface réseau vers une autre.

Objet connecté : objet possédant la capacité d'échanger des données avec d'autres entités physiques ou numériques.

Internet des objets (IDO) : expansion du réseau internet à des objets et/ou des lieux du monde physique. En anglais, on parle d'IoT : Internet of Things.

M2M : machine to machine, échange d'informations entre deux machines sans intervention humaine.

LoraWan : protocole de télécommunication permettant la communication à bas débit, par radio, d'objets à faible consommation électrique communiquant selon la technologie LoRa et connectés à l'Internet via des passerelles, participant ainsi à l'Internet des objets. LoRa est un réseau longue portée à débit compris entre 0,3 et 50Kbps soit un débit plus faible que le 2G.

Zigbee : ZigBee est un protocole de haut niveau permettant la communication d'équipements personnels ou domestiques équipés de petits émetteurs radios à faible consommation. Son débit est de 250 kb/s.

Wi-Fi : ensemble de protocoles de communication sans fil . Un réseau Wi-Fi permet de relier par ondes radio plusieurs appareils informatiques (ordinateur, routeur, smartphone, etc.) au sein d'un réseau informatique afin de permettre la transmission de données entre eux. Les débits peuvent varier en fonction du protocole utilisé (a/b/g/n/ac/ax) on peut avoir des débit compris entre 11 et 1000 Mb/s.

4G : Protocole de 4ème génération des standards de la téléphonie mobile. Ses débits varie en fonction de la catégorie du protocole (4,6,12), on a donc le LTE Cat 4 pour 150 Mbit/s, LTE-A Cat 6 pour 300 Mbit/s, LTE-A Cat 12 pour 600 Mbit/s. Dans ces variations, on a la 4G+ soit LTE-A(Advanced) qui propose des débit nettement supérieurs aux débits 4G classique.

MQTT : Il s'agit d'un protocole de messagerie de publication/abonnement extrêmement simple et léger, conçu pour les appareils à contraintes et les réseaux à faible bande passante, à forte latence ou peu fiables et/ou à faible autonomie.

Dashboard : (Tableau de bord) est un ensemble d'un ou plusieurs panneaux ou graphiques organisés et disposés en une ou plusieurs rangées.

1. Introduction

Depuis plusieurs années, on assiste à une modification de notre environnement du point de vue technologique, car de plus en plus d'objets physiques, sont connectés à l'Internet (montres, lunettes, caméra,...) . Cette innovation technologique a pour nom « l'Internet des objets » (IoT en Anglais), c'est une évolution en marche depuis plus de 20 ans avec la première désignation de ce terme émis par un employé de Procter & Gamble (P&G), Kevin Ashton, qui parlait alors du lien entre la technologie RFID et l'Internet.

Bien que le terme internet des objets a disparu entre 1999 et 2005. C'est en 2003 que le premier objet connecté est commercialisé par la firme Violet. C'est la lampe DAL, équipée de 9 leds qui s'allument en fonction des événements. Deux ans plus tard, c'est en 2005 que Violet lance le lapin Nabaztag.

Aujourd'hui, le nombre d'objets connectés augmente de manière exponentielle. D'après l'entreprise Cisco, on se rapprocherait des 22,5 milliards d'objets connectés en 2020 et 29 milliards d'objets connectés prévu pour 2023, comme le montre *la figure 1* du nombre d'objets connectés dans le monde (Mars 2020) :

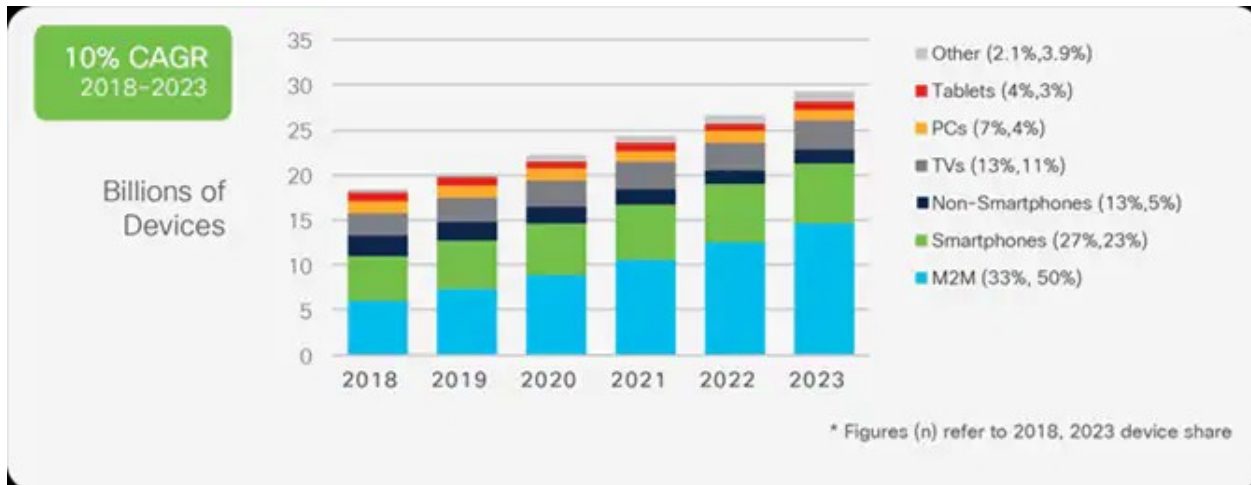


Figure 1: Nombre d'objets connectés dans le monde (Cisco Mars 2020)

Bien que de nombreuses possibilités voient le jour dans plusieurs secteurs, dans ce mémoire nous allons nous intéresser à la préservation de la faune et de la flore dans le domaine forestier, et ce, pour deux raisons.

Premièrement, l'Internet des objets va révolutionner la manière de surveiller les forêts et la manière dont les personnes pourront agir en conséquence. Ne serait-ce que pour la surveillance en cas d'incendies ou d'inondations.

Bien que les promesses soient très grandes, de nombreux challenges sont à surmonter pour pouvoir voir se réaliser les bénéfices espérés, notamment avec la gestion des dispositifs (réseaux, alimentation, ...).

Deuxièmement, même si l'internet des objets est une technologie qui séduit de plus en plus, il ne faut pas oublier, qu'il faut comprendre son fonctionnement et donc il faut se préparer à l'utiliser.

Le problème des nouvelles technologies est un problème assez connu :

Il faut impérativement s'informer et se positionner face à un nouveau phénomène ; cependant, il n'existe pas à notre connaissance d'outil simple pour la mise en place de solutions afin de surveiller la faune et la flore dans les forêts. Dans ce mémoire, nous tenterons d'établir une solution afin de palier à ce problème. Nous allons donc nous concentrer sur la mise en place de solutions possibles quelle que soit la grandeur du secteur à surveiller.

De manière concrète, nous allons mettre en place de façon théorique et de manière méthodologique, une solution applicable par certaines personnes ayant travaillé un minimum dans les réseaux et télécommunications. Nous chercherons le type de matériels à utiliser, l'alimentation et la mise en place d'une architecture réseau pour transmettre les informations souhaitées.

Dans ce mémoire, nous suivrons la logique suivante :

- Premièrement, nous ferons un état de ce qui existe en terme de projets et nous étudierons ces solutions afin de comprendre le fonctionnement d'une surveillance dans le milieu forestier (Chapitre 2 et 3).
- Deuxièmement, avec toutes les informations récoltées au travers des solutions existantes nous essaierons de créer la nôtre, avec la mise en place d'un réseau, du choix des paramètres à prendre en compte pour la surveillance de la faune et de la flore (programmation de capteurs) et de la gestion de l'énergie (Chapitre 4).

2. Internet des objets

Avant de commencer à référencer tous les projets en rapport à la préservation de la faune et de la flore dans le milieu forestier, il est nécessaire de comprendre le concept d'IOT, pour cela, on partira sur une explication en 3 étapes : sa définition, comment fonctionne cette technologie et les protocoles réseaux utilisés pour l'IOT.

2.1 Sa définition

Pour reprendre la citation de M. Han and H. Zhang l'internet des objets serait considéré comme « un réseau qui relie et combine les objets avec l'Internet, en suivant les protocoles qui assurent leurs communications et échange d'informations à travers une variété de dispositifs. » [1]

Pour approfondir cette citation, on peut expliquer que les objets connectés possèdent leurs propre « identité numérique » et qu'ils sont capables de communiquer entre eux. Ce réseau d'objets connectés crée une passerelle entre un monde physique et un monde numérique. Dans certains cas, on parle aussi d'interaction numérique-physique. L'IoT commence ainsi dans le monde physique avec les capteurs qui récupèrent des informations, ces informations sont ensuite transmises grâce à différents protocoles de communication (LoraWan, Zigbee,...), les données sont ensuite traitées et stockées pour être analysées et exploitées par l'utilisateur.

On retrouve cette technologie principalement dans ces 4 grands domaines :

L'industrie : La surveillance et la réduction des dépenses énergétiques, la gestion des alertes,...

La ville : Avec la gestion des places de parking, avec des parkings connectés, la gestion en eau et électricité de la ville.

L'environnement : La surveillance des espaces pleins airs, la gestion des arrosages automatiques

La santé : Suivi des patients et des machines connectés.

2.2 Son fonctionnement

Comme décrit dans la partie précédente, un objet connecté permet la récupération d'informations via des capteurs et permet de transmettre ces informations via l'interconnexion des objets sur le réseau, ainsi son fonctionnement se déroule de la manière suivante :

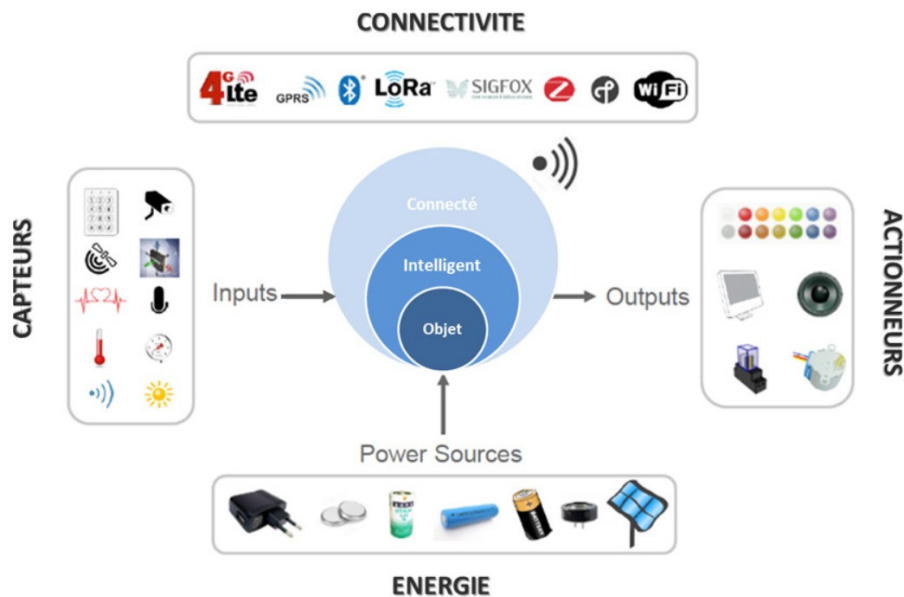


Figure 2: Fonctionnement objets connectés (ConnectWave)

L'objet possède un ou des capteurs qui interagissent avec le monde physique, puis ces informations sont traitées et envoyées via différents protocoles de communication comme LoraWan, Zibgee, Wi-Fi, 4G, ... Bien sûr un objet connectés à besoin d'être alimenté soit par une batterie, soit par une pile bouton dans certains cas. Une fois les données reçues, certains objets peuvent actionner des composants (valve, écran,...)

Ensuite, une fois le traitement effectué (envoi des données,...), on passe sur une plus grande échelles avec d'autres objets qui envoient aussi des données en parallèle :

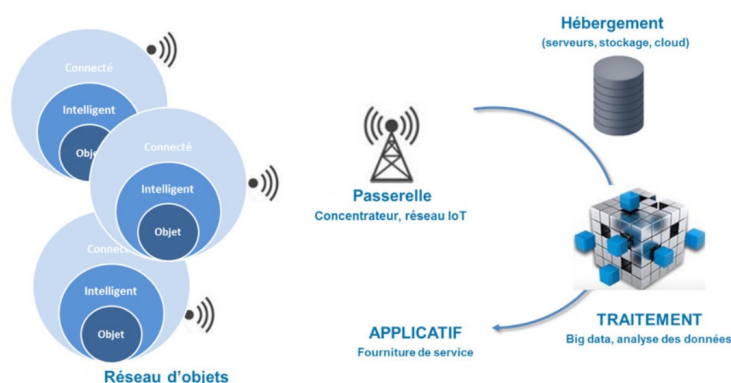
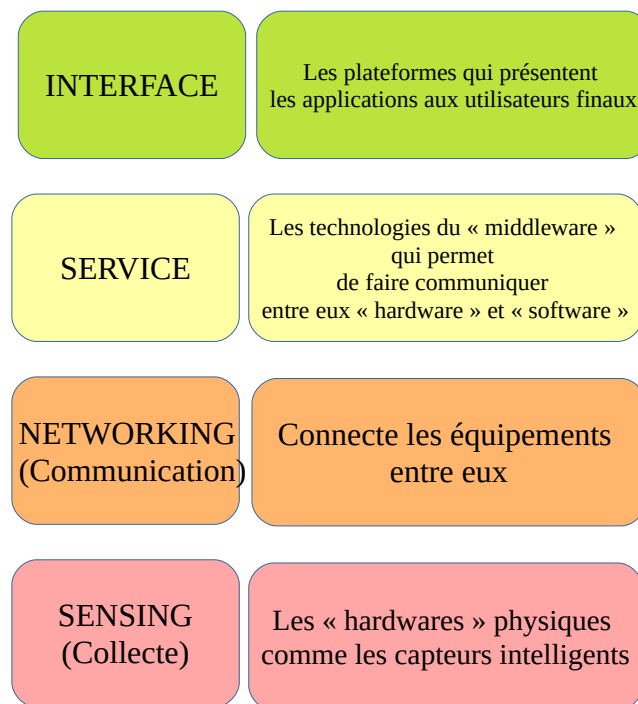


Figure 3: Cheminement des données (ConnectWave)

Chaque objet envoie ces données vers un ou des serveurs en passant par une passerelle qui redirigera ensuite les données vers le serveur adéquat. Ensuite, une analyse des données reçues est effectuée. Ces données peuvent être utilisées pour des utilisateurs (Application et service) ou alors pour de la supervision ou sécurisation.

On peut alors en ressortir 4 couches en rapport à cette technologie :



En ce qui concerne le choix du protocole à utiliser, on peut le choisir en premier lieu en fonction du besoin, si ce sont des données importante alors privilégié des protocoles qui ont un taux de transfert rapide comme le Wi-Fi ou la connexion cellulaire 4G, si au contraire se sont des données de faible taille alors , ZigBee ou encore le Bluetooth Low Energy (BLE) peuvent être utilisés.

On peut aussi choisir le protocole en fonction de la superficie à couvrir dans notre cas, il peut s'agir d'une petite forêt ou d'une réserve naturelle comme celle de la « Réserve Naturelle Nationale de la Forêt de Massane » proche de Banyuls sur Mer. Dans ces cas-là, on peut utiliser si ce sont des petites superficie des réseaux à courte portée comme le Wifi, ZigBee, ou encore le Bluetooth Low Energy, car ils permettent de transférer des données sur de faibles distances. S'il s'agit de grandes distances, on pourra se focaliser sur des réseaux longue portée avec une faible consommation comme Sigfox, LoRa ou encore les technologies cellulaires (GSM, 2G, 3G, 4G...).

2.3 Les protocoles réseaux utilisés

L'établissement d'une connexion consiste à résoudre le principal problème, à savoir établir une méthode de communication, et les différentes méthodes utilisées sont influencées par les contraintes imposées aux appareils connectés au réseau. L'une des contraintes est celle de l'autonomie, et cette contrainte est généralement la plus importante.

Si l'autonomie est assez faible ou limitée, il faut se tourner vers des protocoles dont les débits de transferts sont faibles et dont la consommation est limitée.

Bluetooth Low Energy (BLE)

La technologie Bluetooth Low Energy est une nouvelle implémentation qui n'est pas directement compatible avec un dispositif possédant la norme Bluetooth classique. Elle a été conçue pour des besoins énergétiques faibles et un taux d'échange de données moins fréquent. Un dispositif BLE communique avec un débit correspondant à 1 Mbit/s pour une consommation d'énergie 10 fois inférieur par rapport à un dispositif Bluetooth classique (<15 mA). La portée du BLE est de 50 m.

Le Bluetooth Low Energy cherche donc à s'adresser à des appareils à faible puissance de calcul et dont l'autonomie est une contrainte majeure.

ZigBee

ZigBee est un protocole de communication sans-fil à courte portée et à faible consommation énergétique. Basé sur la norme 802.15.4 qui est un protocole de communication. ZigBee utilise les fréquences 868 Mhz, 915 Mhz et 2,4 Ghz pour établir une connexion. Son débit varie en fonction de la fréquence utilisée (20 kbit/s pour la fréquence 868 Mhz et 250 kbits/s pour la fréquence en 2,4 Ghz).

La portée est assez courte, jusqu'à 100 mètres, mais il peut parcourir de longues distances en passant à travers un réseau maillé avec d'autres appareils ZigBee.

En ce qui concerne l'implémentation réseau, on retrouve 3 topologies différentes (maillée, en étoile et en arbre).

Dans une architecture réseau qui utilise cette technologie, on retrouve 3 modules :

Zigbee Coordinator (ZC) : c'est le module qui permet d'initialiser le réseau, une fois le réseau initialisé, il se comporte comme un routeur.

ZigBee Routeur (ZR) : C'est un module optionnel, cependant s'il est présent il participe au routage des messages.

Zigbee End Device (ZED) : Dispositif qui a pour fonction de communiquer avec son coordinateur.
Il est facultatif et ne peut pas participer au routage des messages.

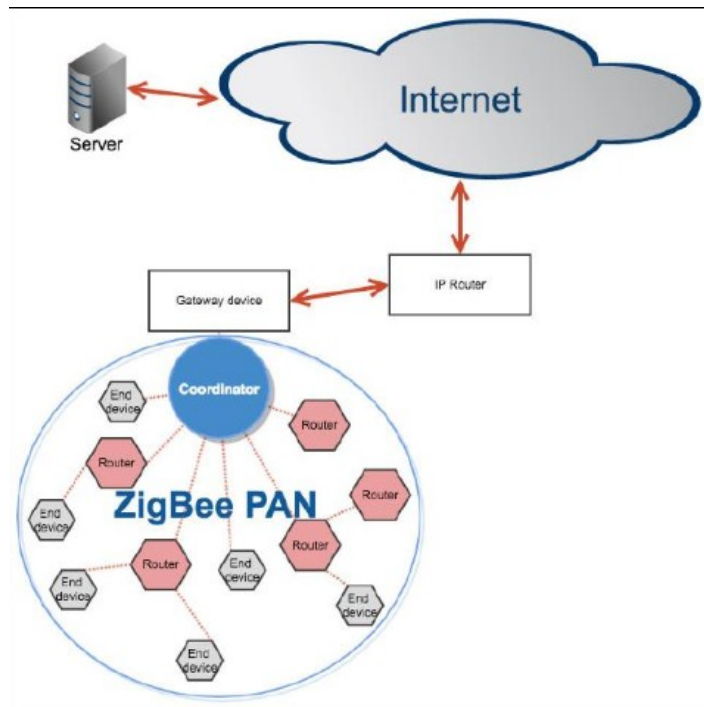


Figure 4: Exemple d'architecture réseau ZigBee

Si l'autonomie n'est pas un problème, on peut donc se tourner vers des protocoles plus énergivores dont le débit de transferts sera plus important.

Wi-Fi

Cette technologie est un ensemble de protocoles de communication sans fil . Un réseau Wi-Fi permet de relier par ondes radio plusieurs appareils informatiques (ordinateur, routeur, smartphone, etc.) au sein d'un réseau informatique afin de permettre la transmission de données entre eux. Le Wi-Fi offre des débits de données très rapides, mais nécessite plus de puissance pour supporter les communications constantes en aller et retour. Les débits peuvent varier en fonction du protocole utilisé (a/b/g/n/ac/ax) on peut avoir des débits compris entre 11 et 1000 Mb/s. Cette technologie est aussi utilisée pour l'internet des objets, car elle permet d'établir une connexion vers « L'internet ».

Sa portée dépend de la norme utilisée :

Standard	Bande de fréquence	Débit	Portée
WiFi a (802.11a)	5 GHz	54 Mbit/s	10 m
WiFi B (802.11b)	2.4 GHz	11 Mbit/s	140 m
WiFi G (802.11g)	2.4 GHz	54 Mbit/s	140 m
WiFi N (802.11n)	2.4 GHz / 5 GHz	450 Mbit/s	250 m

Tableau 1: Tableau des débits et portées de la technologie Wi-Fi en fonction de la norme

Le schéma de fonctionnement est presque identique entre les différents protocoles, c'est-à-dire qu'il s'agit d'un terminal qui va se connecter sur un point d'accès et ce point d'accès est lui même connecté à une passerelle. Dans certains cas comme celui du wi-fi au niveau domestique en général, la « box » ou encore la « box internet » permet d'être à la fois un point d'accès et une passerelle. Cette particularité est due au fait que la box est un matériel qui intègre un routeur, un point d'accès, un switch et un modem. La box est donc une passerelle entre le LAN-réseau local privé et WAN réseau public-internet.

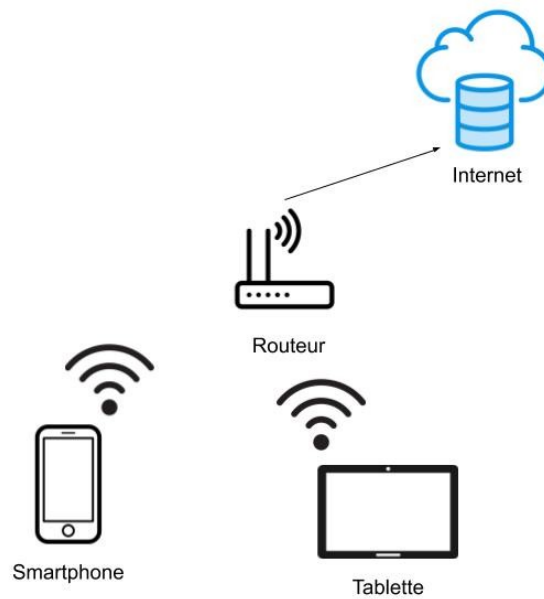


Figure 5: Schéma du principe d'une architecture Wi-fi au niveau domestique

4G (Réseau cellulaire)

Protocole de 4ème génération des standards de la téléphonie mobile. Ses débits varient en fonction de la catégorie du protocole (4,6,12), on a donc pour la catégorie LTE Cat 4 on a 150 Mbit/s, LTE-A Cat 6 pour 300 Mbit/s, LTE-A Cat 12 pour 600 Mbit/s. Dans ces variations, on observe deux types de 4G, nous avons la 4G et la 4G+ soit LTE-A(Advanced) qui propose des débit nettement supérieurs aux débits 4G classique.

Une catégorie a été conçu pour l'internet des objets, il s'agit de la catégorie 0, cette catégorie a un débit de données plus faibles, généralement plafonnés à 1 Mbit/s maximum.

3 Le lien avec la faune et la flore

Sur la période 2007 à 2018, en France métropolitaine, on dénombre en moyenne par an 4 040 feux qui ravagent approximativement 11 117 ha de forêt. La plupart des feux ont eu lieu dans les zones méditerranéennes pas moins de 6 698 ha ont déjà brûlés sur cette période. La plupart de ces incendies ont été provoqués à cause des conditions météorologiques particulières que l'on retrouve sur les côtes méditerranéenne. Les facteurs principaux sont les fortes températures, des vents forts et une sécheresse importante. Bien que les événements naturels surviennent, l'Homme a aussi sa part de responsabilité dans ces incendies, L'activité humaine est une des causes de déclenchement d'incendies dû à une activité économique (chantiers de BTP, activités agricoles...) ou bien d'une activité du quotidien (mégots de cigarettes, barbecues ou feux de camps). Ces incendies sont dû à des imprudences et à des comportements dangereux, aussi bien de touristes que de riverains.

Le problème étant que lorsque qu'un feu est déclaré par une personne, le feu est déjà présent et fait des ravages, c'est là que l'Internet des objets peut intervenir sur ce genre de cas. Des projets comme IoTrees et SmartForest ont vu le jour.

3.1 Les différents projets existants pour la préservation de la faune et de la flore

Le projet SmartForest vise à développer des applications pour les propriétaires de parcs forestiers, pour la surveillance en temps réel de leur propriété, grâce à un réseau de capteurs peu coûteux. Le but est d'anticiper les conditions environnementales propices à l'apparition d'incendies et les détecter dès le début. Une fois, les conditions propices à l'apparition d'un incendie, une alerte est envoyée afin de prévenir les gardes forestiers ou l'organisme qui s'occupe de la forêt pour qu'ils agissent de manière rapide. Les capteurs utilisés sont des capteurs de température, d'humidité et de mesure de niveaux de Co2 et Co.

L'architecture réseau d'un tel projet se compose de la manière suivante :

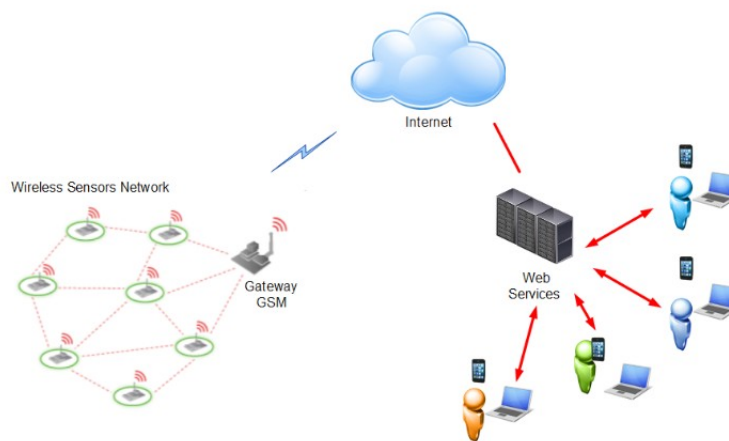


Figure 6: Architecture réseau du projet SmartForest

Les dispositifs sont connectés à une passerelle GSM, afin de faire transiter les informations vers les serveurs, comme il s'agit de données dont la taille est assez petite, un réseau GSM est suffisant. De plus la portée du signal peut varier entre 1 et 30 km. Le gain quant à lui doit être assez élevé car les arbres présents dans une forêt peuvent « interférer » avec le signal. Si le signal n'est pas assez « puissant » et que le dispositif est connecté dessus, mais qu'il n'arrive pas à garder une connexion stable alors on aura une surconsommation d'énergie dû à des tentatives répétées de connexion et de recherche de signal.

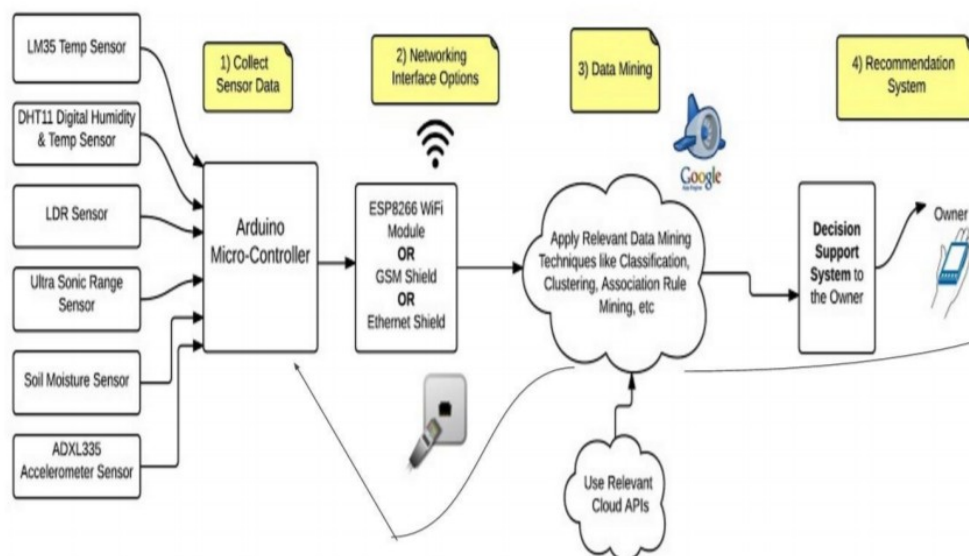
Une fois que les informations sont envoyées vers la passerelle, ces données sont envoyées vers les serveurs qui sont ensuite analysées et affiche l’alerte sur les dispositifs des utilisateurs.

En ce qui concerne l'alimentation des dispositifs, ils sont alimentés par batterie, mais aussi par panneau solaire afin de permettre une certaine autonomie sans que l'Homme ne soit dans l'obligation d'aller vérifier les niveaux des dispositifs.

Un projet semblable à SmartForest a vu le jour, il s'agit d'un projet qui est apparu dans le « International Journal of Current Trends in Engineering & Research (IJCTER) » dans le 3ème volume, numéro 05 datant de Mai 2017.

Ce projet est aussi sur la prévention d'incendies dans le milieu forestier, il utilise plusieurs capteurs à faible coût afin d'alerter l'utilisateur. Ces capteurs sont des capteurs de température LM35 ou encore le DHT11. On a aussi des photorésistances, des capteurs ultra son et des capteurs d'humidité, dont un mesure l'humidité de l'air et l'autre du sol.

BASIC SYSTEM ARCHITECTURE OF THE IoT ENABLED SYSTEM:-



Pour la détection et l'envoi d'alerte, ces dispositifs peuvent se baser sur des conditions

Figure 7: Schéma fonctionnement du projet paru dans IJCTER particulières comme l'exemple ci-dessous qui est un tableau qui est apparu dans un document durant l'événement « The Eleventh International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies » UBICOMM 2017 et qui représente les différents facteurs d'alerte :

Variable	Alert: level 3	Alert: level 2	Alert: level 1
Temperature	$\geq 30^{\circ}\text{C}$	$\geq 37^{\circ}\text{C}$	$\geq 40^{\circ}\text{C}$
Humidity	$\leq 30\%$	$\leq 20\%$	$\leq 10\%$
CO ₂	$\geq 350\text{ ppm}$	$\geq 2000\text{ ppm}$	$\geq 5000\text{ ppm}$
CO	$\geq 10\text{ ppm}$	$\geq 25\text{ ppm}$	$\geq 50\text{ ppm}$

Tableau 2: Tableau des alertes

Pour donner plus d'informations ne serait-ce que pour le taux de Co2 un taux normal de Co2 est compris entre 380 et 480 ppm au-delà de 5 000 ppm cela devient dangereux pour l'Homme.

<u>Concentration</u>	<u>Effet sur l'homme - Seuil</u>
380 - 480 ppm	Taux normal de l'atmosphère
600 - 800 ppm	Taux correct en lieux fermés
1000 - 1100 ppm	Taux tolérable en lieux fermés
5000 ppm	Limite haute pour 8h d'exposition
6000 - 30000 ppm	Exposition très courte
3 à 8 %	Augmentation fréquence respiratoire et cardiaque
Au-delà de 10 %	Nausée, vomissement, évanouissement
Au-delà de 20 %	Evanouissement rapide, décès

Tableau 3: Niveau de Co2 et impact sur l'Homme (th-industrie)

D'autres projets comme le projet IoTrees vise à développer une surveillance en utilisant dendrométrie, pour mesurer le diamètre des arbres et envoyer les informations de mesure de manière transparente à la plateforme qui gère le projet (ForestHQ) où l'utilisateur peut interagir avec les données forestières pour estimer la croissance de la forêt et surveiller la santé des cultures. Cette solution a pour but de réduire le coût de la collecte de données et permettre d'effectuer des mesures forestières plus fréquentes, améliorant ainsi la surveillance durable des ressources forestières. Les nouvelles informations reçues permettront de prendre des décisions plus efficaces sur la gestion de la forêt. Les décisions seront plus opportunes et plus faciles à prendre, par exemple en ce qui concerne le moment de la récolte et celui de l'application de la lutte contre la végétation ou les parasites. Les futures prévisions de la croissance des forêts seront plus précises et plus fiables grâce à des mesures plus fréquentes.

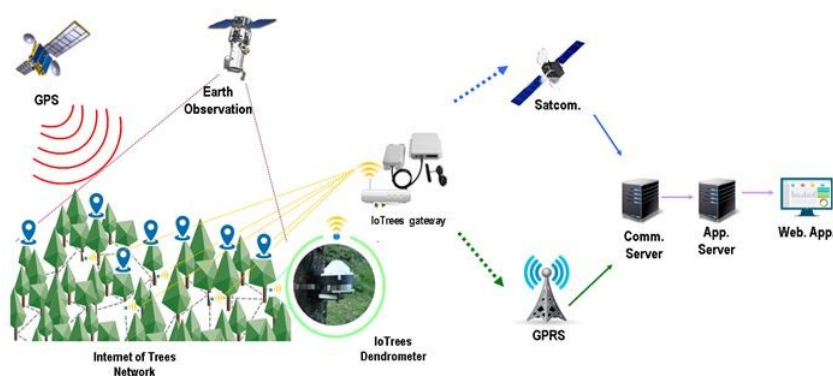


Figure 8: Fonctionnement de la solution (IoTree)

Le projet Rainforest est un projet qui se situe cette fois-ci dans la forêt tropicale. Ce projet a pour but de protéger la forêt de déforestations illégales, au travers d'une analyse de son. Les ingénieurs qui gèrent ce projet réutilisent des téléphones mobiles recyclés, ces systèmes sont rendus étanches et possèdent une autonomie importante. Ces dispositifs sont ensuite placés en forêt. Ces dispositifs envoient en temps réel le son émis par la forêt. Les différentes fréquences sonores reçu sont ensuite analysées et les alertes sont envoyées directement auprès des organismes de surveillance des forêts et de la lutte contre la déforestation criminelle.

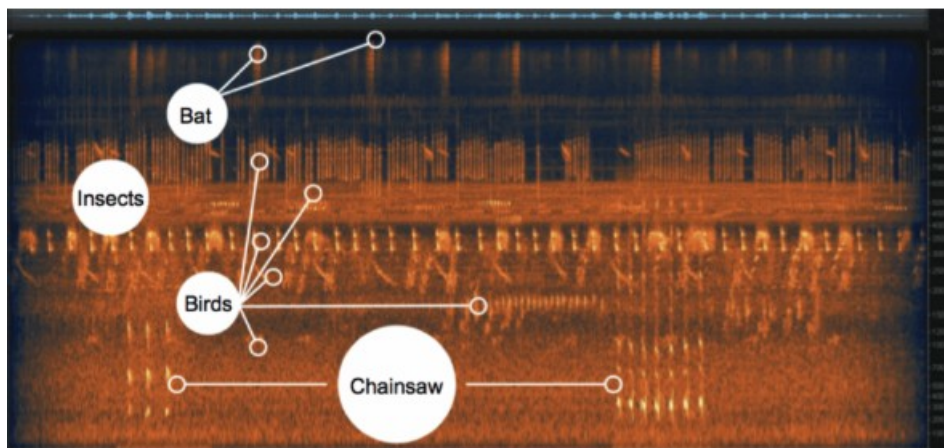


Figure 9: Exemple de l'analyse des sons

Chaque son est analysé et permet de déterminer s'il s'agit ou non d'un animal.

S'il s'agit du bruit d'une tronçonneuse, une alerte est automatiquement envoyée vers les équipes d'intervention afin de stopper l'activité en cours, bien sûr quand un appareil envoi une alerte il envoie ses informations (nom, position,...) afin que l'intervention soit rapide.

Le dispositif permettant de recueillir les sons et de les envoyés vers le cloud de rainforest ressemble à cela :



Figure 10: Dispositif Rainforest

Ce dispositif est alimenté par des panneaux solaires accrochés tout autour du téléphone, ils ont disposé plusieurs bandes de panneaux solaires afin de permettre à l'appareil d'être rechargé tout au long de la journée grâce à une disposition permettant de capter les rayons émis par le soleil quel que soit le moment de la journée. Le dispositif utilise sa batterie la nuit et est rechargé la journée, cela permet une certaine autonomie sans que l'homme ait besoin d'intervenir de manière régulière.

Le schéma ci-dessous montre le fonctionnement du projet :

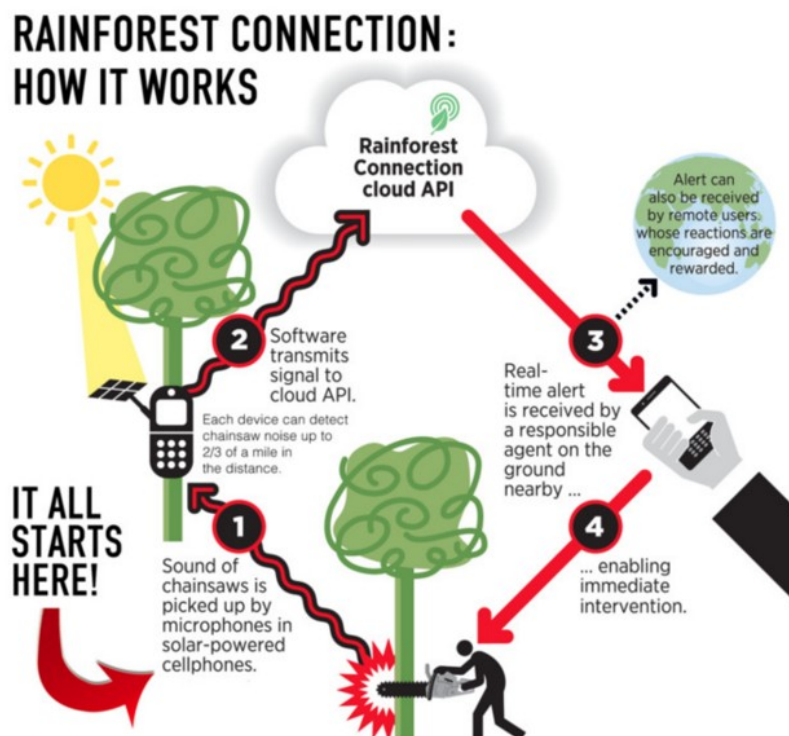


Figure 11: Fonctionnement de la solution chez Rainforest

De plus, ce projet, en plus de gérer la flore, s'occupe aussi de braconnage d'animaux, le but est de faire en sorte que les braconniers ne puissent plus régner librement sur les forêts tropicales humides du monde. Les capteurs qu'ils ont développés permettant d'aider leurs partenaires à reconnaître les schémas d'activité liés au braconnage, notamment les alertes concernant les camions, les voitures et les motos utilisées par les braconniers dans les principales zones protégées. Pour reprendre un de leurs exemples, en Afrique, ils ont pu démontrer, que la protection et la surveillance d'une route clé utilisée par les braconniers pouvaient permettre de protéger une grande partie de la forêt tropicale.

Leurs dispositifs ont permis aux équipes d'intervention d'agir à des moments et des jours clé de chaque mois où les activités de braconnage étaient statistiquement très élevées.

Bien que la flore soit un domaine où l'Internet des Objets peut se développer, il faut aussi prendre en compte que la faune est aussi touchée par cette avancée technologique. On retrouve des objets connectés notamment dans des colliers avec traqueur GPS intégré à l'intérieur du collier. Cela permet aux soigneurs et aux chercheurs de suivre des animaux sans avoir besoin d'intervenir. Cela permet d'évaluer leurs lieux de vie et leurs habitudes, et ainsi intervenir au cas où des braconniers voudraient agir, et donc perturber les habitudes des animaux et l'envoi des données effectué par le collier.

Généralement ce genre de dispositifs est utilisé dans les pays où le taux de braconnage est élevé, pour donner un exemple, au Kenya, la police a installé des micropuces dans les cornes des rhinocéros pour suivre leurs mouvements et ainsi augmenter les chances de poursuivre les braconniers. Au Zimbabwe, le projet Rapid (Real-Time Anti-Poaching Intelligence Device) de l'International Humane Society est déployé sur les rhinocéros avec des capteurs de fréquence cardiaque, des trackers GPS ainsi que des caméras vidéo. En cas de rencontre avec un braconnier, le rythme cardiaque du rhinocéros augmente en raison du stress. Celui-ci déclenche une alarme et transmet ses coordonnées GPS à un centre de contrôle. Le centre de contrôle active alors la caméra du rhinocéros afin d'obtenir une vérification visuelle de ce qui se passe. Ils envoient ensuite un hélicoptère qui arrive sur place avant que les braconniers puissent s'échapper ou récolter les cornes.

Chapitre 4 : Partie Tech.

Dans ce mémoire, nous allons pas seulement parler des projets existants, nous allons aussi tenter de concevoir une solution même s'il s'agit d'une solution théorique.

Le projet autour de cette solution serait de combiner les technologies utilisées pour la supervision de la faune et de la flore et de les faire communiquer entre elles.

Nous allons donc dans une première partie, essayer de faire communiquer un dispositif et un serveur dans lequel le dispositif transmettra des informations vers le serveur qui lui affichera les informations.

Simulation d'une situation de monitoring dans une forêt

Dans cette partie, nous essaierons de faire une maquette d'une simulation sur une situation de monitoring pour une forêt, pour ce faire nous utiliserons un ordinateur portable qui servira de dispositif pour la surveillance, nous utiliserons un autre ordinateur avec une clé 4G qui sera connecté à Internet et servira de point d'accès pour le dispositif.

Nous ferons aussi la configuration d'un point d'accès et d'une passerelle qui permettra la redirection des informations vers le réseau domestique où se trouve le serveur.

Le schéma de la simulation sera présenté de la façon suivante :

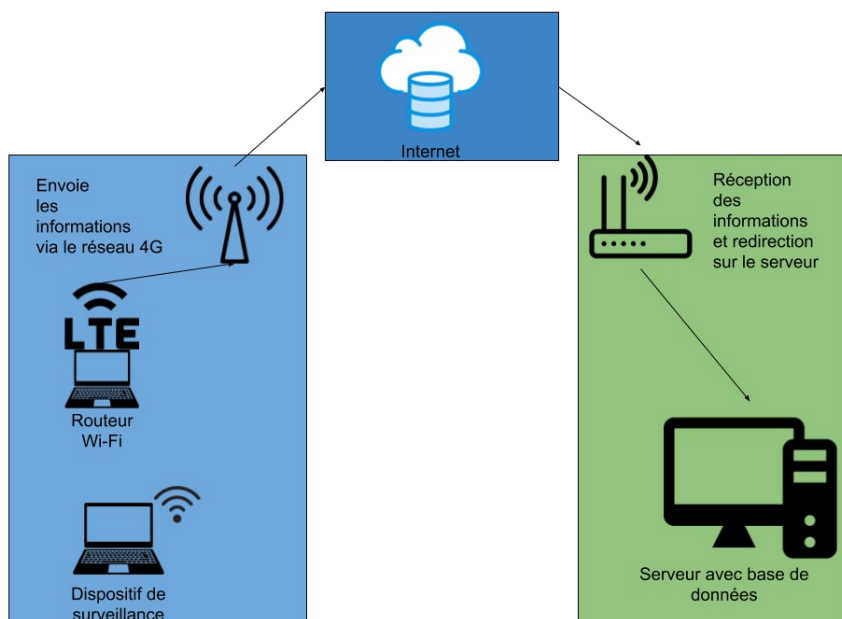


Figure 12: Schéma de la simulation

L'ordinateur qui servira de dispositif de surveillance, enverra des informations « essentiels » via MQTT, il enverra sa température, son niveau de charge pour la batterie, son signal wi-fi (en pourcentage) et son niveau de puissance (en dBm). Pour la transmission des informations en mettra en place un environnement MQTT, pour cela il nous faut mettre en place différents services.

Pour cela on utilisera les programmes suivants :

- Broker MQTT : Mosquitto
- Base de données (InfluxDB)
- Visualiser les données MQTT sur serveur web : Grafana

Configuration de l'environnement

1.1 Le broker MQTT

Pour le Broker qui est la base de l'environnement pour l'envoi des données, on utilisera Mosquitto, celui-ci sera installé sur une VM qui servira de serveur ainsi que notre base de donnée et le serveur web.

Installation de Mosquitto :

Pour installer le paquet on passe par la commande apt , une fois installé on se déplace dans le dossier / etc /mosquitto , pour aller chercher le fichier de configuration mosquitto.conf, afin de configurer certains paramètres :

```
#Partie configurée
log_type debug
#on affichera les logs
allow_anonymous false
#on refuse les connexions anonymes
password_file /etc/mosquitto/user.txt
#on crée un fichier qui contiendra des utilisateurs et mots de passes afin de permettre la connexion au broker
```

Pour « sécuriser » le broker, on désactive les connexions anonymes, on fait en sorte de pouvoir récupérer les logs en cas de problèmes, puis on lui indique que l'on souhaite avoir un liste d'utilisateurs qui pourront se connecter au broker seulement s'ils sont dans ce fichier texte.

Ensuite on utilise la commande mosquitto_passwd afin de prendre en compte le fichier user.txt et on lui attribue un utilisateur ici celui-ci sera buddy :

```
mosquitto_passwd -c /etc/mosquitto/user.txt buddy
```

```
Password:  
Reenter password:
```

On définit un utilisateur afin de permettre l'envoi des données.

1.2 La base de donnée InfluxDB

Pour sauvegarder les messages envoyés sur le broker et donc garder un historique, on configure une base de donnée. Elle recevra les messages via Telegraf, qui sera considéré comme client mqtt sur le topic souhaité. Pour cela on utilisera InfluxDB.

Pour installer influxDB, on procède de la manière suivante :

```
curl -sL https://repos.influxdata.com/influxdb.key | sudo apt-key add -  
echo "deb https://repos.influxdata.com/ubuntu stretch stable" | sudo tee /etc/apt/sources.list.d/influxdb.list  
apt update  
apt install influxdb  
service influxdb start  
influx
```

Afin de configurer notre base de données nous devons modifier son fichier de configuration qui se trouve dans le répertoire `/etc/influxdb`.

On modifie le fichier `.conf` et on recherche la partie qui concerne la connexion via HTTP :

```
[http]  
enabled = true  
bind-address = ":8086"  
auth-enabled = true  
write-tracing = false  
log-enabled = true  
pprof-enabled = true  
debug-pprof-enabled = false  
https-enabled = true
```

Le but de cette manœuvre est d'activer le support HTTP, sur le port 8086, avec l'authentification pour que la base de données ne reçoit que les messages d'utilisateurs enregistrés.

Pour finir la configuration, il nous faut définir un utilisateur avec les privilèges, sur la base de données. On lance influx pour afficher le CLI afin de créer un utilisateur admin :

```
influx  
Connected to http://localhost:8086 version 1.7.9  
InfluxDB shell version: 1.7.9  
> CREATE USER admin WITH PASSWORD 'influx' WITH ALL PRIVILEGES
```

L'utilisateur admin a été créé avec tous les privilèges.

Une fois l'utilisateur créé, il nous faut créer notre base de données, pour ce faire on crée la base de donnée iot :

```
CREATE DATABASE iot
```

On vérifie les valeurs dans la base de donnée iot :

```
> use iot
Using database iot
> SHOW MEASUREMENTS
name: measurements
name
----
cpu
disk
diskio
kernel
mem
processes
swap
system
```

De ce côté la base de données est prête.

1.3 Télégraf

Pour faire le lien entre la base de donnée et le broker on va utiliser Telegraf qui va traduire les informations pour la base de données.

On récupère la clé pour le dépôt, ainsi que le dépôt :

```
wget -qO- https://repos.influxdata.com/influxdb.key | sudo apt-key add -
source /etc/os-release
test $VERSION_ID = "10" && echo "deb https://repos.influxdata.com/debian buster stable" | sudo tee
/etc/apt/sources.list.d/influxdb.list
apt update
apt install telegraf
```

On modifie telegraf.conf pour prendre en compte le serveur MQTT.

```
#####
#           OUTPUT PLUGINS           #
#####
[[outputs.influxdb]]
  urls = ["http://127.0.0.1:8086"]
  database = "iot"
## HTTP Basic Auth
  username = "admin"
  password = "influx"
```

```
[[inputs.mqtt_consumer]]
servers = ["tcp://127.0.0.1:1883"]
topics = ["iot/msg"]
username = "buddy"
password = "test"
data_format = "influx"
```

On lance le service et on effectue des envois via mqtt sur un pc distant ou en local :

```
mosquitto_pub -h 192.168.1.40 -t iot/msg -m "iot,host=buddy value=42" -u buddy -P test
```

Cette requête va se connecter au serveur pour publier dans le topic iot/msg dont le contenu du message est : la table iot, la colonne host= buddy, colonne value=42

Après plusieurs tests, sur la base de données et via mqtt on obtient le résultat suivant via la commande `SHOW MEASUREMENTS` dans influxDB :

```
> SHOW MEASUREMENTS
name: measurements
name
----
cpu
disk
diskio
iot
kernel
mem
message
processes
swap
system
```

On a bien la table iot qui a été créée à la réception du message.

On peut vérifier la table avec la requête `SELECT` :

```
> SELECT * FROM iot
name: iot
time                host    topic    value
----                -
1575117850893600318 buddy  iot/msg  42
1575118597839564563 buddy  iot/msg  69
```

On a bien le nom associé host avec le topic ainsi que la valeur qui avait été envoyée.

1.4 Grafana

Afin de visualiser les données transmises à la base de données, nous devons installer un service capable de lire ces données et d'afficher un rendu clair. Grafana permettant de faire cela et étant facile à configurer et à utiliser, c'est celui-ci que nous allons installer.

Pour installer Grafana plusieurs étapes sont nécessaires tout d'abord il faut ajouter le dépôt sur la machine :

```
sudo add-apt-repository "deb https://packages.grafana.com/oss/deb
stable main"
```

Puis on récupère la clé pour installer grafana :

```
wget -q -O - https://packages.grafana.com/gpg.key | sudo apt-key
add
```

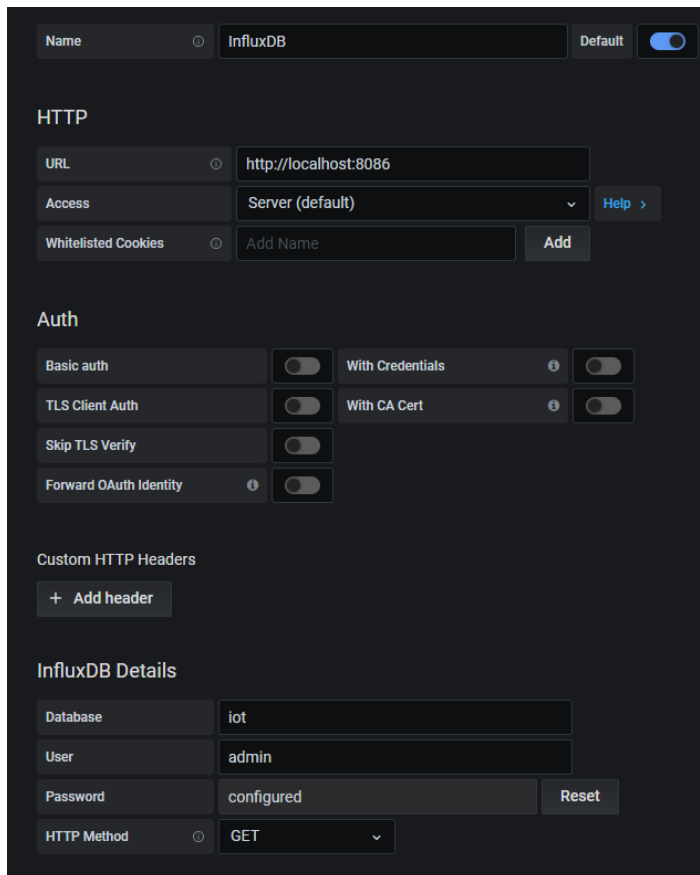
On met à jour les dépôts et on installe Grafana.

On lance le serveur avec les fichiers de configuration par défaut et on vérifie son statut :

```
service grafana-server start
service grafana-server status
• grafana-server.service - Grafana instance
  Loaded: loaded (/usr/lib/systemd/system/grafana-server.service; disabled; vendor preset: enabled)
  Active: active (running) since Thu 2020-06-25 10:52:04 UTC; 1s ago
```

On se connecte sur la page web de Grafana avec l'utilisateur et le mot de passe par défaut soit « admin/admin », ensuite on modifie le mot de passe admin.

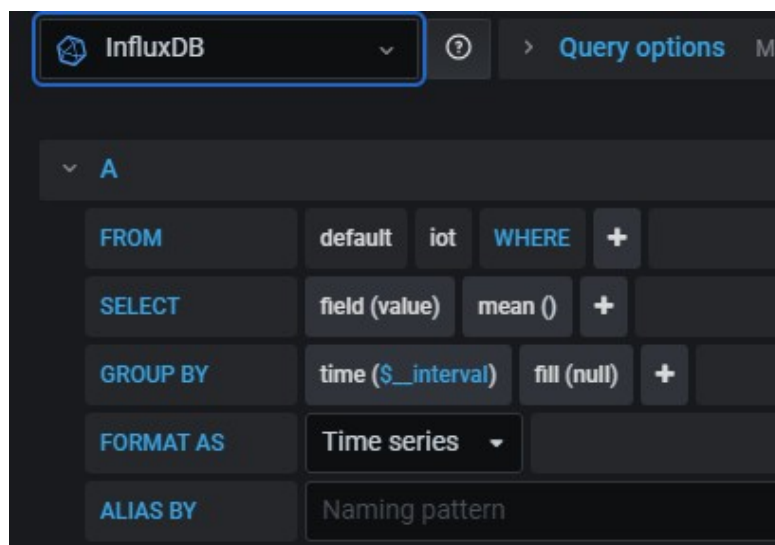
On ajoute une nouvelle base de données source et on lui attribue une base InfluxDB et on lui donne les paramètres suivants :



The screenshot shows the configuration interface for a new InfluxDB data source. The 'Name' field is set to 'InfluxDB' and is marked as the 'Default' source. Under the 'HTTP' section, the 'URL' is 'http://localhost:8086', the 'Access' is 'Server (default)', and there is a 'Whitelisted Cookies' section with an 'Add' button. The 'Auth' section contains several toggle switches: 'Basic auth' (off), 'With Credentials' (off), 'TLS Client Auth' (off), 'With CA Cert' (off), 'Skip TLS Verify' (off), and 'Forward OAuth Identity' (off). Below this is a 'Custom HTTP Headers' section with an 'Add header' button. The 'InfluxDB Details' section includes fields for 'Database' (set to 'iot'), 'User' (set to 'admin'), 'Password' (set to 'configured' with a 'Reset' button), and 'HTTP Method' (set to 'GET').

Figure 13: Paramètres BDD

Une fois configuré, il faut créer un dashboard, on utilisera le dashboard de base. Pour la partie Query (Requêtes) qui va récupérer les informations de la base de donnée, on l'écrit de la façon suivante :



The screenshot shows the InfluxDB Query Editor interface. At the top, a dropdown menu is set to 'InfluxDB'. Below this, there is a query template with several sections: 'FROM' with 'default' and 'iot' as options, 'WHERE' with a '+' button, 'SELECT' with 'field (value)' and 'mean ()' as options, 'GROUP BY' with 'time (\$__interval)' and 'fill (null)' as options, 'FORMAT AS' with a dropdown set to 'Time series', and 'ALIAS BY' with a 'Naming pattern' input field.

Figure 14: Requêtes

2. Redirection des ports :

Pour cette partie on va configurer la redirection des ports afin que les informations soient redirigées vers le serveur. On attribue au serveur une adresse IP fixe :

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:fc:36:1f brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.48/24 brd 192.168.0.255 scope global dynamic eth0
        valid_lft 48186sec preferred_lft 48186sec
```

Figure 15: Adresse serveur

Une fois l'adresse attribué, il faut procéder à la redirection, sur la « box », il faut se rendre dans la section gestion des ports et affecter un port :

Figure 16: Redirection sur un port spécifique

En ce qui concerne la redirection des ports, il n'y a rien de plus à faire.

3. Test MQTT

Pour cette partie, nous allons faire des test d'envois de messages MQTT :

Dans un premier temps, on connecte le dispositif au point d'accès qui se nomme AP 4G :

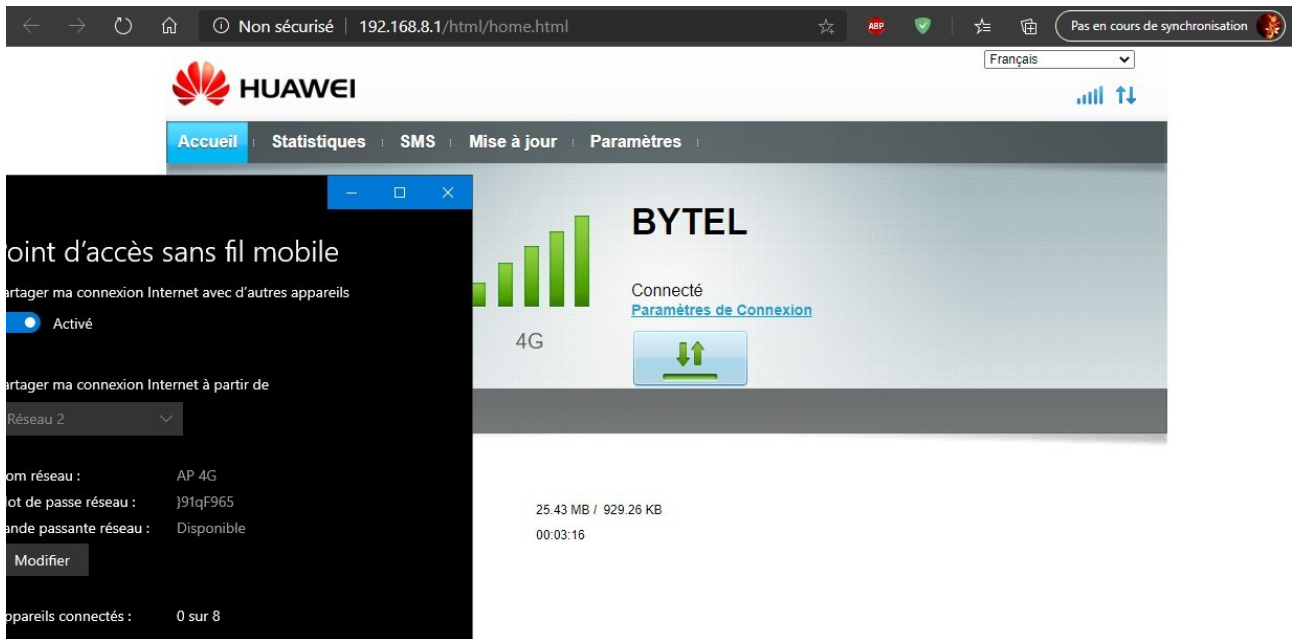


Figure 17: Interfaces clé 4G et point d'accès

Nom réseau :AP 4G

Mot de passe réseau :}91qF965

Bande passante réseau :Disponible

Modifier

Appareils connectés :1 sur 8

Nom de l'appareil	Adresse IP	Adresse physique (MAC)
buddy-G5-5590	192.168.137.9	a4:c3:f0:36:eb:c4

Figure 18: Paramètres du point d'accès

Sur le dispositif on vérifie son adresse IP :

```
3: wlo1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
qlen 1000
link/ether a4:c3:f0:36:eb:c4 brd ff:ff:ff:ff:ff:ff
inet 192.168.137.9/24 brd 192.168.137.255 scope global dynamic noprefixroute wlo1
    valid_lft 604728sec preferred_lft 604728sec
inet6 fe80::4a71:7319:57f9:ebbd/64 scope link noprefixroute
    valid_lft forever preferred_lft forever
```

La commande iwconfig nous donne des informations sur la carte réseau sans-fil et notamment le nom du point d'accès auquel le dispositif s'est connecté :

```
wlo1 IEEE 802.11 ESSID:"AP 4G"  
    Mode:Managed Frequency:2.462 GHz Access Point: 86:C6:3B:B4:FF:61  
    Bit Rate=72.2 Mb/s Tx-Power=22 dBm  
    Retry short limit:7 RTS thr:off Fragment thr:off  
    Power Management:on  
    Link Quality=70/70 Signal level=-33 dBm  
    Rx invalid nwid:0 Rx invalid crypt:0 Rx invalid frag:0  
    Tx excessive retries:21 Invalid misc:1 Missed beacon:0
```

Pour « prouver » que l'on se trouve sur une connexion différente du serveur, on effectue un curl d'un site web qui nous donne notre adresse IP publique :

```
curl ifconfig.me/ip  
176.149.89.77
```

Une fois les étapes de connexion effectuées, le plus simple est d'envoyer un message pour vérifier la connexion, on effectue la requête :

```
mosquitto_pub -h 88.127.243.12 -p 16384 -t iot/msg -m "iot,host=buddy value=54" -u buddy -P  
test
```

On définit l'adresse publique du réseau où se trouve le serveur ainsi que le port à utiliser.

Et on vérifie si le serveur a reçu le message :

```
mosquitto_sub -h localhost -t "iot/msg" -u buddy -P test  
  
iot,host=buddy value=54
```

Deuxième test :

Cette fois-ci on va élaborer un script qui va envoyer la puissance en dBm du signal wifi auquel on est connecté ainsi que sa puissance en pourcentage, on rajoute aussi la température de l'ordinateur ainsi que son niveau de batterie :

Le script :

```
#!/bin/bash

wifi=$(grep "^s*w" /proc/net/wireless | awk '{print $4}');
wifip=$(grep "^s*w" /proc/net/wireless | awk '{ print int($3 * 100 / 70)}');
battery=$(cat /sys/class/power_supply/BAT0/capacity);
temp=$(sensors | awk '/Core 0/ {print $3}' | sed s/+// | sed s/°// | sed s/C//);
while true ;
do mosquitto_pub -h 88.127.243.12 -p 16384 -t iot/msg -m "wifi2,host=buddy puissanced=$wifi" -u buddy -P test ;
mosquitto_pub -h 88.127.243.12 -p 16384 -t iot/msg -m "wifi,host=buddy wifi=$wifip" -u buddy -P test ;
mosquitto_pub -h 88.127.243.12 -p 16384 -t iot/msg -m "bat,host=buddy bat=$battery" -u buddy -P test ;
mosquitto_pub -h 88.127.243.12 -p 16384 -t iot/msg -m "temp,host=buddy cpu=$temp" -u buddy -P test ;
done;
```

Chacune des commande ci-dessus va créer une table dans la base de données, afin de différencier les données reçues. Chaque dispositifs aura un nom différents en fonction du secteur et du numéro attribué (ex : SecA42), ce nom sera mis dans host et on pourra trier les informations par la suite.

On test le script, du côté serveur on réceptionne les messages :

```
wifi2,host=buddy puissanced=-47.
wifi,host=buddy wifi=90
bat,host=buddy bat=100
temp,host=buddy cpu=34.0
```

Une fois la réception des messages effectuée, on se dirige vers la base de données afin de vérifier si les messages ont bien été transmis vers celle-ci.

Sur la table puissance (en dBm):

```
> select * from wifi2
name: wifi2
time          host  puissanced topic
----          -
1592571740056193738 buddy -47    iot/msg
1592571741826097527 buddy -47    iot/msg
```

Sur la table en pourcentage :

```
> select * from wifi
name: wifi
time      host topic  wifi
----      -
1592571750117268791 buddy iot/msg 90
1592571750611939548 buddy iot/msg 90
```

Sur la table batterie :

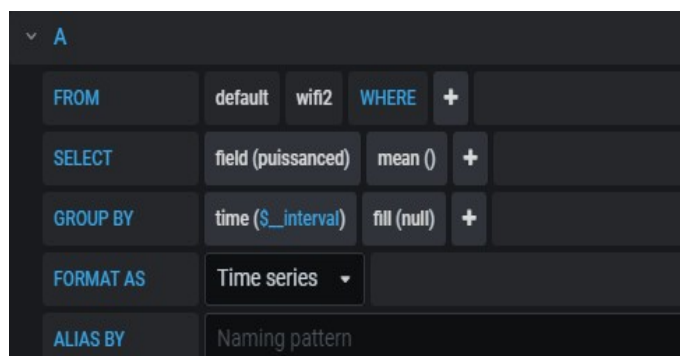
```
> select * from bat
name: bat
time      bat host topic
----      -
1592571750247857057 100 buddy iot/msg
1592571750731875377 100 buddy iot/msg
```

Sur la table température :

```
> select * from temp
name: temp
time      cpu host topic
----      -
1592571750366771664 34 buddy iot/msg
1592571750856544326 34 buddy iot/msg
```

On définit ensuite les graphiques sur grafana :

Requête pour la puissance en dBm :



The screenshot shows the Grafana query editor interface. At the top, there is a dropdown menu with 'A' selected. Below it, the query is built using a table of clauses and values. The clauses are FROM, SELECT, GROUP BY, FORMAT AS, and ALIAS BY. The values are default, wifi2, WHERE, field (puissanced), mean (), time (\$_interval), fill (null), Time series, and Naming pattern. The query is displayed in a dark theme.

Clause	Value
FROM	default wifi2 WHERE +
SELECT	field (puissanced) mean () +
GROUP BY	time (\$_interval) fill (null) +
FORMAT AS	Time series
ALIAS BY	Naming pattern

Figure 19: Requête puissance en dBm

Requête signal en % :

The screenshot shows a query builder interface with a dark theme. At the top, there is a dropdown menu labeled 'A'. Below it, there are five rows of query components:

- FROM:** Contains 'default', 'wifi', 'WHERE', and a '+' button.
- SELECT:** Contains 'field (wifi)', 'last ()', and a '+' button.
- GROUP BY:** Contains 'time (\$__interval)', 'fill (null)', and a '+' button.
- FORMAT AS:** Contains a dropdown menu set to 'Time series'.
- ALIAS BY:** Contains a text input field labeled 'Naming pattern'.

Figure 20: Requête signal Wi-Fi

Requête niveau de batterie:

The screenshot shows a query builder interface with a dark theme. At the top, there is a dropdown menu labeled 'A'. Below it, there are five rows of query components:

- FROM:** Contains 'default', 'bat', 'WHERE', and a '+' button.
- SELECT:** Contains 'field (bat)', 'last ()', and a '+' button.
- GROUP BY:** Contains 'time (\$__interval)', 'fill (null)', and a '+' button.
- FORMAT AS:** Contains a dropdown menu set to 'Time series'.
- ALIAS BY:** Contains a text input field labeled 'Naming pattern'.

Figure 21: Requête niveau de batterie

Requête température :

The screenshot shows a query builder interface with a dark theme. At the top, there is a dropdown menu labeled 'A'. Below it, there are five rows of query components:

- FROM:** Contains 'default', 'temp', 'WHERE', and a '+' button.
- SELECT:** Contains 'field (cpu)', 'last ()', and a '+' button.
- GROUP BY:** Contains 'time (\$__interval)', 'fill (null)', and a '+' button.
- FORMAT AS:** Contains a dropdown menu set to 'Time series'.
- ALIAS BY:** Contains a text input field labeled 'Naming pattern'.

Figure 22: Requête niveau de température du dispositif

On peut aussi obtenir une vue globale du dispositif :



Figure 23: Vue globale des différents composants

Alertes :

En utilisant grafana nous avons la possibilité de créer des alertes en cas de problèmes, ce qui peut être utile en cas de baisse critique de la batterie ou encore un niveau élevé pour la température.

Pour donner un exemple on va configurer une alerte pour le niveau de température, Grafana a ajouté récemment le support de notifications sur Discord, pour faire fonctionner les notifications, nous devons configurer un paramètre nommé WebHook sur un serveur discord.

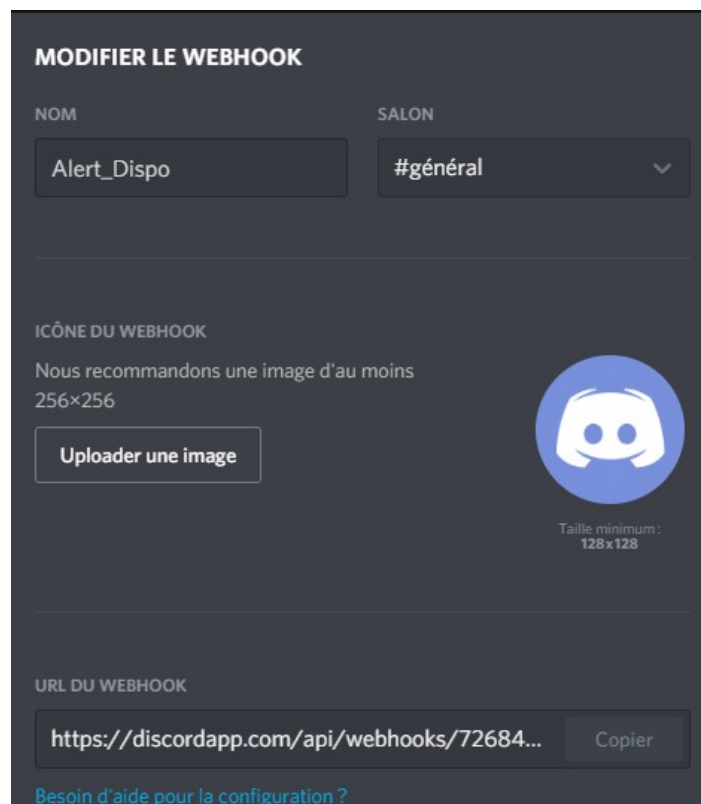
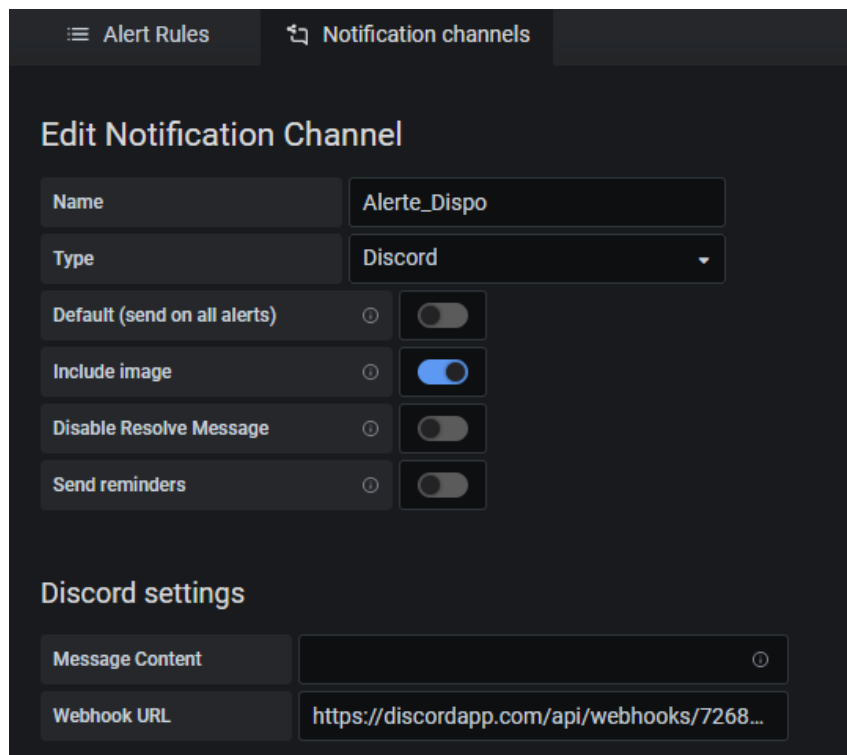


Figure 24: WebHook serveur Discord

Le lien généré par le webhook devrait être retranscrit sur le système d'alerte de Grafana.



The screenshot shows the 'Edit Notification Channel' configuration page in Grafana. At the top, there are tabs for 'Alert Rules' and 'Notification channels'. The page title is 'Edit Notification Channel'. Below this, there are several configuration options:

- Name:** A text input field containing 'Alerte_Dispo'.
- Type:** A dropdown menu set to 'Discord'.
- Default (send on all alerts):** A toggle switch that is currently turned off.
- Include image:** A toggle switch that is currently turned on.
- Disable Resolve Message:** A toggle switch that is currently turned off.
- Send reminders:** A toggle switch that is currently turned off.

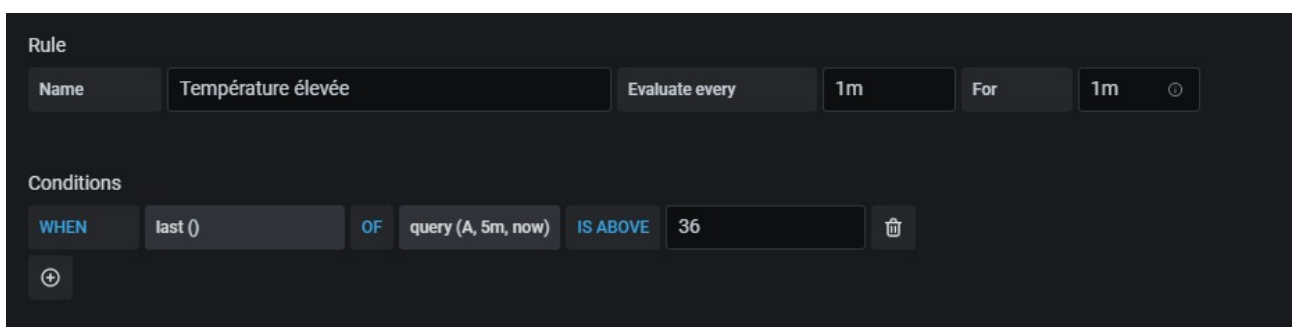
Below these options is a section titled 'Discord settings' with two input fields:

- Message Content:** An empty text input field.
- Webhook URL:** A text input field containing the URL 'https://discordapp.com/api/webhooks/7268...'.

Figure 25: Configuration notifications Discord

Dans la partie notification de grafana, on crée un canal de notification dont les paramètres sont adaptés pour Discord, on lui donne un nom, son type, ainsi que le lien du webhook créé précédemment.

Une fois le canal créé nous devons configurer un dashboard « Graph » car ils prennent en compte le système d'alerte, on lui donne comme requête, la requête du niveau de température, et dans la partie alerte on configure les règles :



The screenshot shows the 'Rule' configuration page in Grafana. At the top, there is a 'Rule' section with the following fields:

- Name:** A text input field containing 'Température élevée'.
- Evaluate every:** A dropdown menu set to '1m'.
- For:** A dropdown menu set to '1m'.

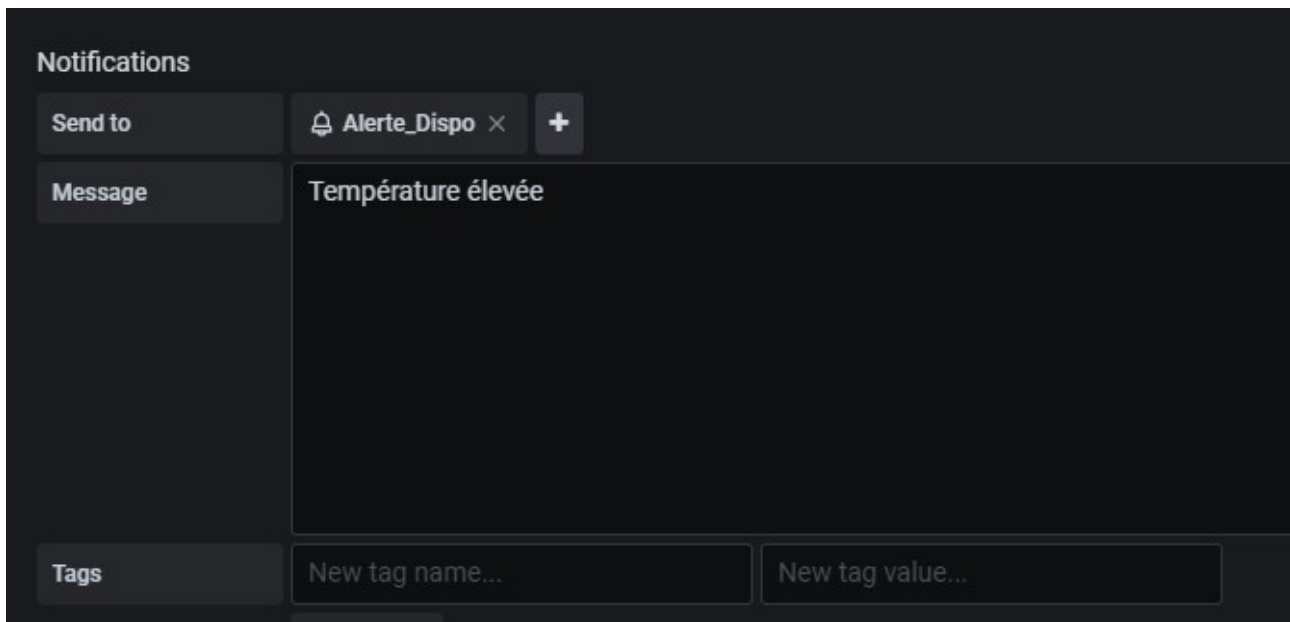
Below this is a 'Conditions' section with a single condition:

- WHEN:** A dropdown menu set to 'last ()'.
- OF:** A dropdown menu set to 'query (A, 5m, now)'.
- IS ABOVE:** A text input field containing '36'.

At the bottom left of the 'Conditions' section, there is a plus sign icon (+) to add more conditions.

Figure 26: Règle pour l'alerte de température

On définit son nom, sa fréquence de vérification, et le moment où l'alerte se déclenche, ici l'alerte se déclenchera dès que la température aura atteint 36 °C.



The screenshot shows the 'Notifications' configuration page in Grafana. On the left, there are three tabs: 'Send to', 'Message', and 'Tags'. The 'Send to' tab is selected, showing a list of notification channels. A channel named 'Alerte_Dispo' is selected, indicated by a bell icon and a close button. The 'Message' tab is also visible, showing the message template 'Température élevée'. The 'Tags' tab is at the bottom, with input fields for 'New tag name...' and 'New tag value...'.

Figure 27: Paramètre du message

Ici on lui attribue le canal de notification créé précédemment avec un message.

Une fois la limite de température atteinte, Grafana envoie une notification sur le serveur :



Figure 28: Message reçu serveur discord

L'utilisateur est donc averti en cas de problème de température.

Conclusion de la simulation :

Cette simulation permet, la visualisation en temps réel, des informations émis par le dispositif qui a été déployé, bien que l'on est utilisé un ordinateur comme dispositif, on pourrait aussi utiliser un Raspberry Pi à la place ou encore un ESP32. En ce qui concerne les requêtes, on pourrait améliorer la répartition des informations, en créant une table par secteur de la forêt, et que l'on pourrait ensuite trier par le nom (« host ») du dispositif. Cela permettrait d'éviter d'avoir trop de tables créée et aussi de rencontrer des problèmes dus à des conflits de nom pour les tables. De plus le support de notification sur des messageries comme discord permet d'avoir un suivi, même en dehors du lieu de travail, car discord est disponible sur plusieurs plateformes (PC, Smartphones, Montres Connectées,...)

Partie 2 : Utilisation d'ESP32 pour la récupération des informations des capteurs

Bien que nous ayons utilisé un ordinateur pour la simulation, nous pouvons aussi utiliser des microcontrôleurs, ces microcontrôleurs sont dépourvus de système d'exploitation (Linux, Windows,...). Ils ont la particularité d'être programmable sur des plateformes comme Arduino. En plus d'être programmable, certains intègrent la technologie Wi-Fi et Bluetooth, deux technologies assez appréciées dans le domaine de l'Internet des Objets.

Dans notre cas nous utiliserons les ESP32, afin d'acheminer des données reçues par des capteurs et de les envoyer vers un serveur. Nous reprendrons le même principe que dans la simulation.

Dans cette partie, nous allons aussi déterminer quels capteurs utiliser. Nous baserons nos critères sur la fréquence d'utilisation du capteur ainsi que sa consommation. Nous en profiterons aussi pour élaborer un programme de manière théorique (sans test) en fonction des capteurs si on en a la possibilité.

1.Capteur de températures

L'un des premiers capteurs que nous allons étudier est celui de température, car dans une forêt, si la température augmente de manière rapide, c'est que l'on peut se trouver dans une situation d'incendie.

L'un des capteurs qui permet d'avoir un retour de la température de manière fiable, avec aussi la possibilité d'avoir un retour sur le niveau d'humidité, c'est le capteur DHT11.

D'après la documentation du capteur, il fonctionne entre 0 et 50 °C avec une précision de 2°C. Nous n'avons pas besoin d'un capteur qui fonctionne avec des températures trop hautes, il suffit que la température approche les 45 °C pour envoyer un alerte. En ce qui concerne sa consommation énergétique, on est entre 0,5 mA et 2,5 mA, l'ESP32 a la particularité de fournir 12 mA sur les pins auxquels on souhaite brancher le capteur.

Son branchement ressemblerait à ceci :

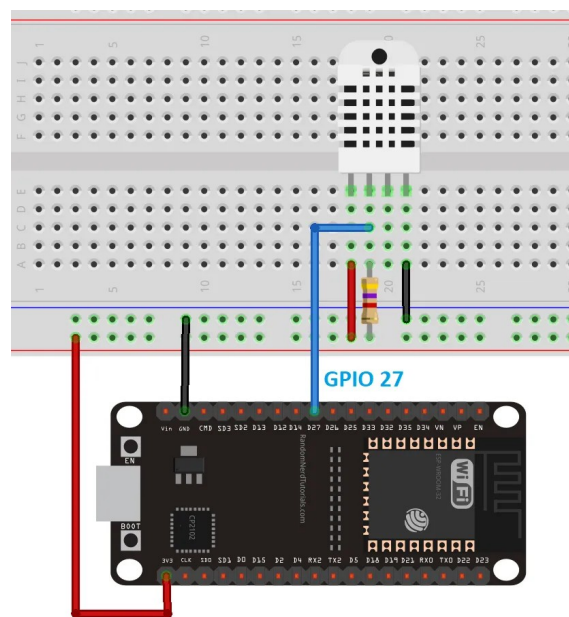


Figure 29: Branchement DHT11 sur ESP32

Un premier branchement se trouve sur le pin 3v3 il permet au capteur d'être alimenté.

Un deuxième branchement se trouve sur le pin gnd (ground) qui permet de relier le capteur à la « terre ».

On applique une résistance de 4,7 k ohm .

La partie qui va récupérer les données est sur le port gpio 27 (General Purpose Input/Output) (on peut le placer sur un autre).

Pour utiliser le capteurs nous aurons besoin de télécharger des librairies. Les librairie étant les suivantes :

- DHT Sensors Library
- DHT Sensors Library for ESPX
- Adafruit Unified Sensor

On élabore un programme permettant de tester le capteur :

```
#include "DHT.h"
#define DHTPIN 25
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(115200);
  Serial.println(F("DHTxx test!"));
  dht.begin();
}

void loop() {
  delay(5000);
  float h = dht.readHumidity();
  float t = dht.readTemperature();
  float f = dht.readTemperature(true);
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
  }
  float hif = dht.computeHeatIndex(f, h);
  float hic = dht.computeHeatIndex(t, h, false);
  Serial.print(F("Humidity: "));
  Serial.print(h);
  Serial.print(F("% Temperature: "));
  Serial.print(t);
  Serial.println(F("°C "));
}
```

2. Niveau CO2 dans la forêt

Pour éviter des fausses alertes envoyer par le capteur de température, qui peuvent être du à des personnes qui mettent une source de chaleur vers le capteur ou qu'il y a une exposition direct en direction du soleil. Nous pouvons mettre en place un ou des capteurs qui surveilleront le niveau de Monoxyde de carbone (CO) et du dioxyde de carbone (CO2) présent dans l'atmosphère environnant . Les modèle que nous pouvons prendre sont le SGP30 et le MICS5524.

Le capteur SGP30 fonctionne sur du 3,3v ainsi sur l'interface I²C (Inter-Integrated Circuit) de l'ESP32 soit les pins SDA (Serial Data Line) et SCL (Serial Clock Line). De plus au niveau de sa consommation, le capteur consomme 48mA (d'après la datasheet). En ce qui concerne sa plage de détection, il commence à 400 ppm, pour donner une comparaison, un taux normal de concentration de CO2 est compris entre 350 et 450 ppm.

Quant au modèle MICS5524, il permet de mesurer le niveau de monoxyde de carbone. Sa plage est comprise entre 1 et 1000 ppm. Une fois alimenté le capteur consomme entre 25 et 35 mA.

Pour un niveau de CO standard sans effets nocifs pour l'Homme, le niveau doit être compris entre 1 et 70 ppm, au-delà cela peut devenir dangereux :

Tableau 4: Niveau monoxyde de carbone et ses dangers

Niveau CO	Effet sur l'homme
50 ppm	pas d'effets nocifs après 8 heures d'exposition.
200 ppm	Léger mal de tête après 2 à 3 heures d'exposition.
400 ppm	Maux de tête et nausées après 1 à 2 heures d'exposition.
800 ppm	Maux de tête, nausées et vertiges après 45 minutes ; effondrement et perte de conscience après 1 heure d'exposition.
1 000 ppm	Perte de conscience après 1 heure d'exposition.
1 600 ppm	Maux de tête, nausées et vertiges après 20 minutes d'exposition.
3 200 ppm	Maux de tête, nausées et vertiges après 5 à 10 minutes ; évanouissement et perte de conscience après 30 minutes d'exposition.
6 400 ppm	Maux de tête et vertiges après 1 à 2 minutes ; perte de conscience et danger de mort après 10 à 15 minutes d'exposition.
12 800 ppm	Effets physiologiques immédiats, perte de conscience et danger de mort après 1-3 minutes d'exposition.

Si le niveau dépasse les 100 ppm on peut commencer à émettre une alerte.

Les branchements des capteurs sont les suivants :

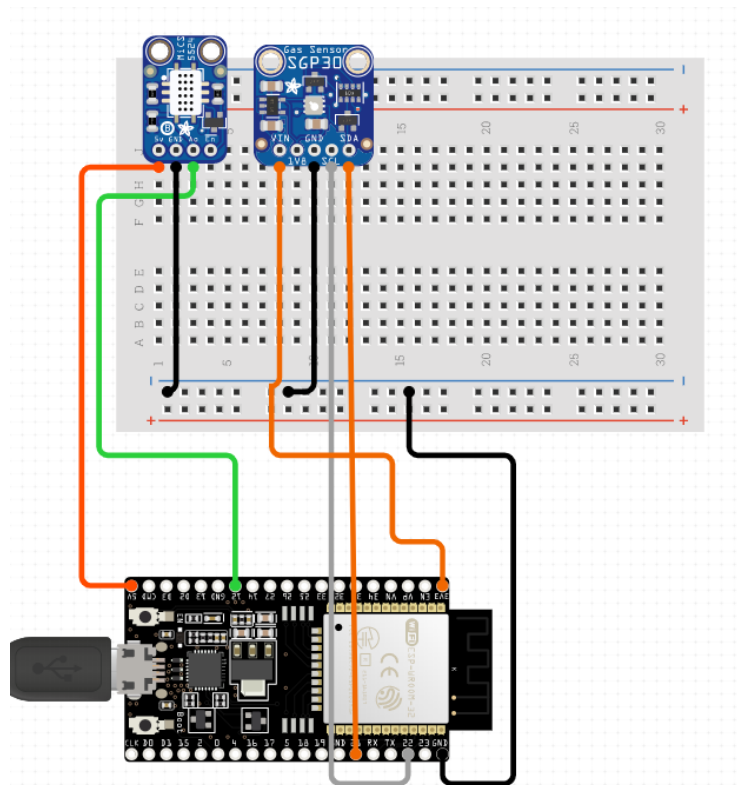


Figure 30: Branchement capteurs CO et CO₂

Pour le capteur SGP30 :

On utilise le pin 3,3v pour l'alimentation.

Pour l'interface I²C, soit les pins SDA (Serial Data Line) et SCL (Serial Clock Line) ce sont les pins 21 et 22 qui seront utilisés.

Pour le capteur MICS5524 :

On branche le capteur sur le pin 5V et le pin A0 du capteur sur le pin 12 de la carte.

Les GND des deux capteurs sont reliés au GND de la carte.

Pour le programme, il faut installer une librairie nommée adafruit sgp30 sensor pour le capteur SGP30. Un exemple est fourni par la librairie.

Pour le capteur MICS5524 on peut faire un programme assez basique pour avoir un retour sur le niveau de CO :

```
void setup() {  
  Serial.begin(9600);  
  Serial.println(" Initializing .....");  
  delay(2000);  
  if (!Serial.available()) {  
    Serial.println("Checking Serial Communication");  
    while (1);  
  }  
  Serial.println("Lecture du niveau de CO");  
  delay(1000);  
}  
void loop() {  
  int co = analogRead(A0);  
  
  Serial.print(" Reading A0 :");  
  Serial.print(co);  
  Serial.println("");  
  delay(100);  
}
```

3. Capteur d'humidité et pluviométrie

Bien que le capteur dht11 à la capacité de mesurer la température, mais aussi l'humidité, on mesure l'humidité dans l'air. Or, on souhaite mesurer le niveau d'humidité dans le sol. On fait cela pour nous permettre d'être avertis s'il y a un risque d'inondation si le niveau est trop élevé.

Notre choix s'est porté sur le capteur moisture de chez sparkfun (SEN-13322), étant facile à mettre en place, il permet un déploiement facile.

Son branchement se constitue de la manière suivante :

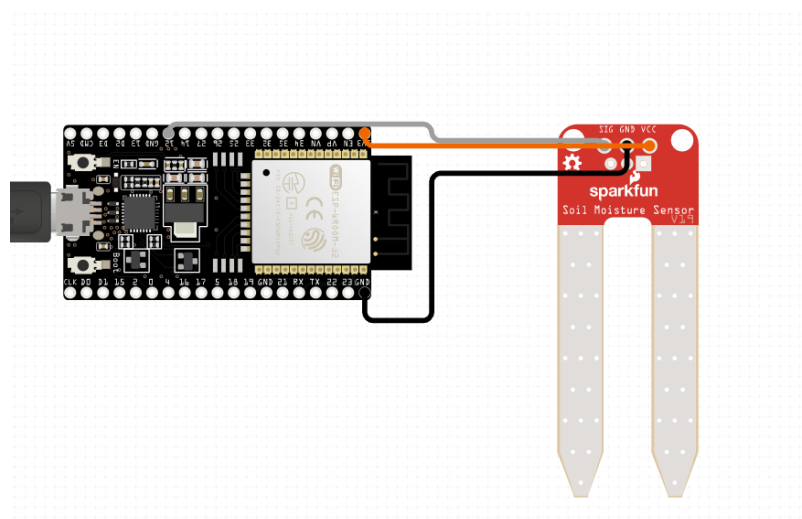


Figure 31: Branchement capteur d'humidité (Circuito.io)

Le pin VCC du capteur sera branché sur le pin 3v3 de la carte.

Le pin GND sur le pin GND de la carte

Le pin qui sert au transfert des données doit être branché sur un pin digital, ici on le branche sur le pin 12 (ou D12 sur certains modèles).

Le programme nous permet d'avoir un retour sur le niveau d'humidité :

```
int val = 0;
int soilPin = 12;
void setup()
{
  Serial.begin(9600);

  pinMode(soilPin, OUTPUT);
  digitalWrite(soilPin, LOW);
}
void loop()
{
  Serial.print("Soil Moisture = ");
  Serial.println(readSoil());
  delay(1000);
}
int readSoil()
{
  digitalWrite(soilPin, HIGH);
  delay(10);
  val = analogRead(soilPin);
  digitalWrite(soilPin, LOW);
  return val;
}
```

4. Niveau rivières ou ruisseaux

Si une ou des rivières ou des ruisseaux sont présent dans la forêt, il est important de vérifier s'il y a un risque de débordement ou d'assèchement, pour cela on peut utiliser un capteur ultrason qui permettra de mesurer la distance entre le capteur et la rivière ou le ruisseau, en envoyant un son à 40kHz vers l'eau et de recevoir un retour via un echo. Pour mesurer la distance, on utilise la relation $d = v \cdot t$. La vitesse du son étant de 340 m/s.

Notre choix pour le capteur se porte sur le capteur HC-SR04, car compatible avec l'ESP32 (après une modification) et dont la durée d'impulsion est en dizaines de μs . Le capteur émet une série de 8 impulsions ultrasoniques à 40 kHz, puis il attend le signal réfléchi. Comme nous sommes en μs il faut convertir 340 m/s soit 34'000 cm/1'000'000 μs . On sait aussi que le son fait un aller-retour. La distance vaut donc la moitié.

La relation finale sera $d = 34'000 \text{ cm} / 1'000'000 \mu s \cdot 10us \cdot \text{valeur} / 2$ en simplifiant on a :

$$d = 17/100 \text{ cm} \cdot \text{valeur}$$

Pour le branchement :

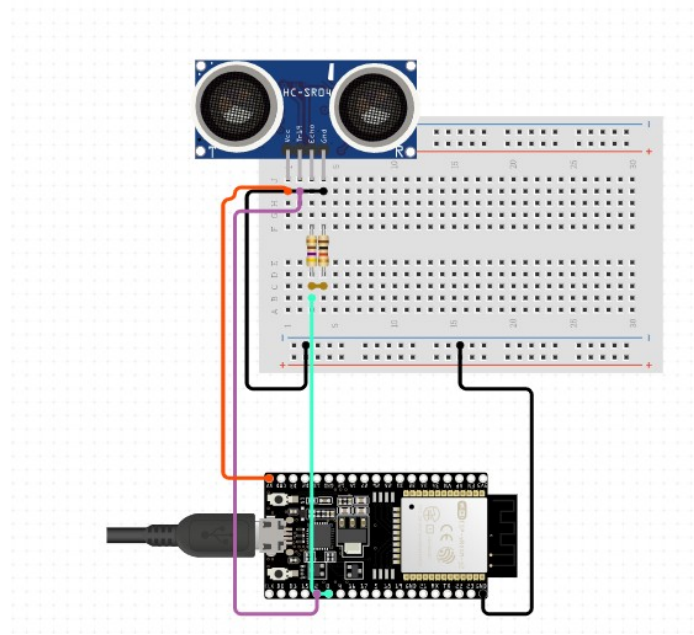


Figure 32: Branchement capteur ultrasonique (Circuito.io)

Comme le capteur fonctionne seulement sur du 5V, on le branche au 5V de la carte.

Le GND sur le GND de la carte.

Le pin Trig qui permet le déclenchement sera sur le pin 2 (ou autres).

Le pin Echo sur 0.

On applique aussi deux résistances en série une de 470 ohm et une autre 1 k ohm, car il s'agit du modèle 5V on peut aussi mettre deux résistances de 10k. Il existe aussi une version 3V3.

En ce qui concerne le programme :

```
const int trigPin = 2;
const int echoPin = 0;

long temps;
int distance;

void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  Serial.begin(9600);
}

void loop() {

  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);

  digitalWrite(trigPin, HIGH);
  // on met trigPin en état HIGH pendant 10 microsecondes
```

```

delayMicroseconds(10);
digitalWrite(trigPin, LOW);

// lecture des valeurs de echoPin
temps = pulseIn(echoPin, HIGH);
//calcul de la distance
distance= temps*0.034/2;

//Affichage de la distance
Serial.print("Distance: ");
Serial.println(distance);
}

```

5.Partage des données vers un serveur

Si l'on souhaite envoyer les données reçus par un capteur vers un serveur, on peut utiliser le protocole MQTT pour ce faire il faut installer la librairie NTPClient afin que l'ESP32 puisse associé ses données avec la date et l'heure.

En ce qui concerne la librairie MQTT, on peut utiliser celle qui est présente sur github :

<https://github.com/knolleary/pubsubclient>

C'est une librairie MQTT compatible avec l'ESP32.

En reprenant l'exemple du capteur de température, nous élaborerons un programme qui permettra de récupérer les données de température, et de les acheminer vers un serveur via MQTT, en étant connecté à un réseau Wi-Fi.

Le programme :

Dans cette partie, on définit les paramètres et librairies qui nous servirons tout au long du code :

```

#include <WiFi.h>
#include <PubSubClient.h>
#include <NTPClient.h>
// Capteur
#include "DHT.h"
#define DHTPIN 25
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
// wifi
const char* ssid = "Foret_SecteurA";
const char* password = "XXXXXXXXXXXXX";
const char* mqtt_server = "XXXXXXXXXXXXX";
//MQTT
#define mqtt_port 12345
#define MQTT_USER "ESPtempA42"

```

```

#define MQTT_PASSWORD "123456789"
#define MQTT_SERIAL_PUBLISH_CH "/Nomdelaforet/Secteur/"
//NTP
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "fr.pool.ntp.org", 3600, 60000);
String formattedDate;

```

Ici on définit les paramètres de connexion Wi-Fi (Foret_SecteurA), l'adresse du broker MQTT ainsi que son port de connexion avec l'utilisateur et le mot de passe.

On définit aussi là où l'ESP32 doit publier le message.

En ce qui concerne la date, on demande à l'ESP32 de se connecter sur un site pour récupérer la date et l'heure.

Ici on initialise la connexion Wi-Fi nécessaire pour l'envoi des données :

```

WiFiClient wifiClient;
PubSubClient client(wifiClient);
void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Connecte à ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connectée");
  Serial.println("Adresse IP : ");

```

Cette partie correspond à la connexion et reconnexion au broker MQTT :

```

void reconnect() {
  while (!client.connected()) {
    Serial.print("En Attente de la connexion MQTT ...");
    String clientId = "ESPtempA42";
    if (client.connect(clientId.c_str(),MQTT_USER,MQTT_PASSWORD)) {
      Serial.println("connecté");
      client.subscribe("/Nomdelaforet/Secteur/");
    }
    else {
      Serial.print("erreur");
      Serial.print(client.state());
      Serial.println(" tentative dans 5 secondes");
      delay(5000);
    }
  }
}

```

De plus ici, on définit le topic auquel l'ESP32 va devoir se connecter, ainsi que la tentative de connexion avec l'utilisateur et le mot de passe qui ont été défini avant, de plus on définit un client ici ESPtempA42 afin de déterminer à quel secteur appartient l'objet connecté.

Par la suite on initialise les variables Température, Wi-fi, MQTT, NTP :

```
void setup() {
  Serial.begin(115200);
  timeClient.begin();
  timeClient.setTimeOffset(3600);
  Serial.setTimeout(500);
  Serial.println(F("DHTxx test!"));
  dht.begin();
  setup_wifi();
  client.setServer(mqtt_server, mqtt_port);
  reconnect();
}
void publishSerialData(char *serialData){
  if (!client.connected()) {
    reconnect();
  }
  client.publish(MQTT_SERIAL_PUBLISH_CH, serialData);
}
```

Cette dernière partie correspond à la boucle loop qui va exécuter le programme, ainsi on utilise le capteur de température, auquel on convertit la donnée reçue en char afin qu'elle puisse être envoyée par MQTT.

```
void loop() {
  while(!timeClient.update()) {
    timeClient.forceUpdate();
    formattedDate = timeClient.getFormattedTime();
    Serial.println(formattedDate);
    delay(5000);
    float h = dht.readHumidity();
    float t = dht.readTemperature();
    float f = dht.readTemperature(true);
    if (isnan(h) || isnan(t) || isnan(f)) {
      Serial.println(F("Failed to read from DHT sensor!"));
      return;
    }
    float hif = dht.computeHeatIndex(f, h);
    float hic = dht.computeHeatIndex(t, h, false);
    Serial.print(F("Humidity: "));
    Serial.print(h);
    Serial.print(F("% Temperature: "));
    Serial.print(t);
    Serial.println(F("°C "));
    //Convertir donnée en char
    char temp[8];
    dtostrf(t, 1, 2, temp);
    client.publish("/Nomdelaforet/Secteur/", temp, formattedDate);
    client.loop();
    if (Serial.available() > 0) {
      char mun[501];
      memset(mun,0, 501);
      Serial.readBytesUntil( '\n',mun,500);
      publishSerialData(mun);
    }
  }
}
```

6.Alimentations des dispositifs

Lors des différentes simulations, l'alimentation des dispositifs n'étaient pas un problème, néanmoins si on veut mettre en pratique les solutions trouvées, il faut choisir la bonne alimentation en fonction des besoins de l'appareil, car en forêt celui-ci ne sera pas brancher à une prise de courant. Dans cette partie on va essayer de choisir la bonne alimentation pour l'ESP32 et le Raspberry PI afin qu'ils puissent fonctionner en total autonomie.

1.Le cas de l'ESP32

L'ESP32 utilise plusieurs modes de fonctionnement :

- mode actif (par défaut) (consommation 160-260 mA et tout fonctionne)
- mode modem (consommation 3-20 mA, les connexions Wi-Fi/Bluetooth et Radio sont réactivés à des moments prédéfinis)
- mode light speed (consommation 3-20 mA, RAM et le CPU où des parties du circuit sont mises hors tension)
- mode deep sleep (consommation 10 μ A, majeure partie de la RAM et tous les périphériques numériques sont mis hors tension)
- mode hibernation (consommation 2,5 μ A, même principe que deep sleep sauf que la mémoire RTC est aussi privée d'alimentation)

Ici, on s'intéressera au mode light speed, qui coupe les communications et les réactivent à des moments prédéfinis en plus de mettre en pause le noyau et la mémoire de l'ESP32.

Certains capteurs notamment le capteur ultrasonique qui permet de mesurer le niveau des ruisseaux , n'a pas besoin d'envoyer ses informations toutes les secondes, on peut donc paramétrer pour qu'il fasse ses mesures, mais qu'il envoie ses valeurs toutes les 6 heures par exemples, en ce qui concerne le capteur de température et les capteurs pour le niveau de CO et eCO2 ces capteurs devront envoyer leurs valeurs plus régulièrement approximativement toutes les 5 minutes.

Le mode modem à lui seul fait une consommation comprise entre 3 et 20 mA. Si on rajoute par exemple le capteur DHT11, sa consommation en mode mesure est de 2,5 mA. En définissant un cycle, il fera sa mesure, enverra l'information puis se remettra en veille. Ces différents modes permettent d'économiser la batterie.

Le code permettant de mettre la carte en mode light_sleep est défini de la manière suivante :

```
void setup() {  
  Serial.begin(115200);  
  Serial.println("setup");  
}  
  
void loop() {
```

```
esp_sleep_enable_timer_wakeup(5000000); //5 secondes car on définit le temps en µs
int ret = esp_light_sleep_start();
Serial.print("light_sleep:");
Serial.println(ret);
}
```

Pour en revenir sur l'alimentation à choisir pour l'ESP32 afin qu'il soit autonome, on peut mettre en place une solution batterie plus panneaux solaires, les panneaux solaires permettront à la carte d'être alimentée la journée et de recharger la batterie et le soir la batterie prendra le relais.

Pour le choix des panneaux solaires, il faut prendre des panneaux dont la sortie pour le voltage soit comprise entre 5 et 6V.

L'un des problèmes que l'on pourrait rencontrer avec une alimentation par panneau solaire et par batterie sera de pouvoir recharger la batterie pour cela, on peut utiliser un module de charge, le module TP4056 permet de recharger des batterie en lithium et possède une protection pour éviter le survoltage et l'inversion de polarité.

Pour la batterie, si on prend une batterie dont le voltage est supérieur à la capacité de l'ESP32, il faudra alors réguler la tension délivrée par la batterie. Pour cela, on peut utiliser un régulateur de tension le MCP1700-3302E permet de descendre la tension à 3,3 V. Le régulateur doit être accompagné d'un condensateur électrolytique de 100µF, et un condensateur céramique de 100nF, connectés en parallèle.

Pour le choix de la capacité de la batterie, il faut prendre en compte la consommation de la carte qui sera la plupart du temps en mode light sleep, mais aussi en mode actif, la consommation des capteurs lors du mode actif et du temps que l'ESP32 doit rester sur batterie (approximativement 12 h).

Si la carte est en mode actif, on est au minimum à 100 mA et au maximum à 260 mA d'après la datasheet. Si on prend 260 mA comme base et que l'on souhaite faire fonctionner l'ESP32 au moins 12 heures.

La formule pour calculer l'autonomie d'une batterie est :

Autonomie batterie = capacité batterie en mAh / courant de charge en mA

Dans notre cas, on a l'autonomie, puisqu'on souhaite qu'elle dure 12 heures, on a le courant de charge (ESP32), on modifie la formule pour trouver la capacité :

Capacité batterie en mAh = Autonomie batterie * courant de charge en mA

Capacité batterie en mAh = 12,00 * 260 = 3120 mAh

Seule la carte aura besoin d'une batterie de 3120 mAh. Or, s'il y a une surconsommation à un moment donné, il est préférable de prendre une batterie de 3500 mAh

En faisant le calcul d'autonomie de batterie, on obtient une autonomie de 13 heures et 27 minutes.

Maintenant si on se base sur la carte plus les capteurs, on a :

- DHT11 : 2,5 mA
- MICS5524 : entre 25 et 35 mA
- SGP30 : 48 mA
- Sen-13322 : 13 mA
- HC-SR04 : 6 mA

Pour la carte et DHT11 :

$$260 + 2,5 = 262,5 \text{ mA}$$

$$\text{Capacité batterie en mAh} = 12,00 * 262,5 = 3150 \text{ mAh}$$

ESP32 + MICS5524+ SGP30 :

$$\text{Capacité batterie en mAh} = 12,00 * 343 = 4116 \text{ mAh}$$

ESP32+Sen-13322 :

$$\text{Capacité batterie en mAh} = 12,00 * 273 = 3276 \text{ mAh}$$

ESP32 + HC-SR04 :

$$\text{Capacité batterie en mAh} = 12,00 * 266 = 3192 \text{ mAh}$$

Bien que les calculs présentés sont pour le mode actifs, les calculs pour le capteur de température et les capteurs pour le niveau de CO et eCO₂, seront presque identique, car on souhaite sortir les cartes de leurs sommeil toutes les 5 minutes. Pour l'ensemble ESP32 + MICS5524 + SGP30 on prendra une batterie de 4200 mAh.

Pour le reste, on veut faire des mesures toutes les heures pour le capteur SEN-13322, et toutes les 6 heures pour le HC-SR04. Une batterie de 3500 mAh sera suffisante pour ces deux capteurs même en mode light sleep.

2. Le cas du Raspberry PI

Pour le Raspberry PI, la solution serait identique, c'est-à-dire que l'on utiliserai des panneaux solaires et une batterie afin que le Raspberry PI soit autonome.

D'après la documentation, un Raspberry PI a besoin de 5V/2A pour fonctionner. Le problème que l'on va rencontrer en plus de trouver une batterie dont la capacité sera suffisante pour supporter le Raspberry PI plus une caméra qui fonctionnera 24h/24, c'est le choix des panneaux solaires qui permettront l'utilisation et la recharge.

Les panneaux solaires PiJuice sont adaptés à tous les modèles de Raspberry PI. Ils fournissent différentes tailles de panneaux en fonction de l'utilisation et des Raspberry PI.

Raspberry Pi Model	Solar Panel Size	Output Voltage	Output Power	Output Current
Pi Zero/Zero W (minimum)	6W	5V	5W	1A (1000mAh)
Pi Zero/Zero W (recommended)	12W	5V	10W	2A (2000mAh)
Pi 3/3B+ (minimum)	12W	5V	10W	2A (2000mAh)
Pi 3/3B+ (recommended)	22W	5V	20W	4A (4000mAh)

Figure 33: Tableaux du choix des panneaux solaires

Dans notre cas, nous faisons tourner un programme de détection assez lourd qui utilise un flux vidéo fournit par une caméra qui filme en continu. Dans l'optique, il faudrait prendre un panneau de 12 W, car en comparaison avec la version 22 W, la version 22 W possède 2 ports USB, mais dont la puissance de sortie, si on utilise qu'un seul port est de 12 W, on serait donc sur une puissance « perdu ».

En ce qui concerne la batterie, comme dit précédemment, nous faisons tourner un programme de détection assez lourd qui utilise un flux vidéo fournit par une caméra, il nous faudra donc une batterie à forte capacité pour durer au moins 12 heures (fonctionnement de nuit).

Si on prend un Raspberry PI 3B+ comme base pour la détection, sa consommation est de 500 mA en fonctionnement.

Product	Recommended PSU current capacity	Maximum total USB peripheral current draw	Typical bare-board active current consumption
Raspberry Pi Model A	700mA	500mA	200mA
Raspberry Pi Model B	1.2A	500mA	500mA
Raspberry Pi Model A+	700mA	500mA	180mA
Raspberry Pi Model B+	1.8A	600mA/1.2A (switchable)	330mA
Raspberry Pi 2 Model B	1.8A	600mA/1.2A (switchable)	350mA
Raspberry Pi 3 Model B	2.5A	1.2A	400mA
Raspberry Pi 3 Model A+	2.5A	Limited by PSU, board, and connector ratings only.	350mA
Raspberry Pi 3 Model B+	2.5A	1.2A	500mA

Figure 34: Tableaux valeurs Raspberry PI

Si on rajoute un module PiCam, on doit rajouter 200 mA, soit 700 mA.

Ensuite, on procède au calcul :

Capacité batterie en mAh = $12,00 \times 700 = \mathbf{8400 \text{ mAh}}$

Il nous faudra au minimum une batterie de 8400 mAh, dans l'optique une batterie de 10000 mAh serait suffisante, au cas où le Raspberry PI a besoins de ressource plus importante.

Détection d'éléments avec OpenCV et Yolo

Afin d'avoir, un suivi des événements présent dans une forêt, nous pouvons utiliser un raspberry Pi avec OpenCV et qui utilisera la bibliothèque Yolo qui permet la reconnaissance de personnes, d'objets et autres au travers d'une caméra.

On utilise ce programme afin de détecter les animaux présent dans la forêt, en plus du collier gps qui peut être accroché. Mais aussi des équipes qui gèrent la supervision de la forêt et dans certains cas des potentiels braconniers.

Pour ce faire, nous devons installer Yolo, Darknet qui est la base du programme, Yolo étant une extension permettant la reconnaissance, ainsi qu'OpenCV qui est une bibliothèque graphique libre, initialement développée par Intel, spécialisée dans le traitement d'images en temps réel.

Pour ce faire on procède de la manière suivante :

On cherche à récupérer le programme darknet sur Github :

```
git clone https://github.com/pjreddie/darknet
cd darknet
make

wget https://pjreddie.com/media/files/yolov3.weights
```

On lance ensuite le programme darknet en prenant en compte Yolo comme fichier de configuration :

```
./darknet detect cfg/yolov3.cfg yolov3.weights data/dog.jpg
```

Le programme va utiliser le fichier de configuration de Yolo et on va lui demander d'aller chercher une image « dog.jpg » dans le dossier data afin qu'il analyse l'image et nous donne son résultat de prédiction :

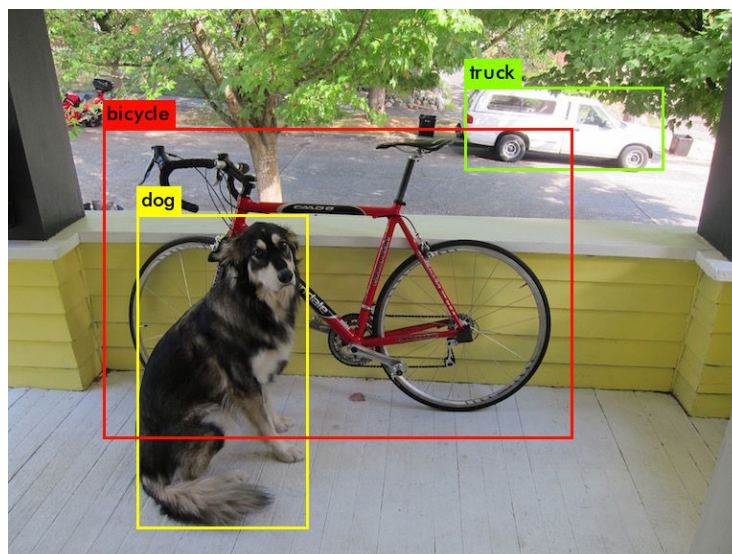


Figure 35: Détection dog.jpg

Ici, il nous détecte un chien, un vélo et un véhicule, cette image est un exemple, en lui fournissant une autre image nous pourrions voir s'il détecte correctement les éléments.

Pour le choix de l'image on restera sur une détection « canines » avec une photo de deux chiens :



Figure 36: Chiens.jpg

On place l'image dans le dossier data pour regrouper toutes les photos qui peuvent être utiles.

Et on lance la commande précédente en modifiant le nom de l'image à analyser.

```
/darknet detect cfg/yolov3.cfg yolov3.weights data/chiens.jpg
```

On obtient le résultat suivant :

```
layer  filters  size      input       output
  0 conv    32  3 x 3 / 1  608 x 608 x 3  ->  608 x 608 x 32  0.639 BFLOPs
  1 conv    64  3 x 3 / 2  608 x 608 x 32  ->  304 x 304 x 64  3.407 BFLOPs
[...]
```

106 yolo
Loading weights from yolov3.weights...Done!
data/chiens.jpg: Predicted in 30.258448 seconds.
dog: 99%
dog: 64%
person: 55%

On obtient une détection des deux chiens ainsi que la détection d'une « personne » :



Figure 37: *Detection chiens.jpg*

Le programme a bien détecté les deux chiens cependant il a détecté une « personne » mais qui correspond au fauteuil sur lequel les chiens ont été disposés.

Ensuite on aimerait essayer de détecter en temps réel afin de pouvoir envoyer une alerte et qu'une équipe puisse intervenir pour ce faire il nous faut installer opencv :

```
sudo apt install libopencv-dev
sudo apt-get install cmake
sudo apt-get install gcc g++
sudo apt-get install python3-dev python3-numpy
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev
sudo apt-get install libgstreamer-plugins-base1.0-dev libgstreamer1.0-dev
sudo apt-get install libgtk-3-dev
sudo apt-get install git
git clone https://github.com/opencv/opencv.git
mkdir build
cd build
cmake ../
make
sudo make install
```

Une fois OpenCV installé, il nous reste à télécharger une librairie très légère pour Yolo, car ce programme demande une grande quantité de ressource, comme il sera sur un raspberry PI il vaut mieux que le programme soit le plus léger possible afin d'éviter toute surconsommation d'énergie :

```
wget https://pjreddie.com/media/files/yolov3-tiny.weights
```

De plus une fois téléchargé il faut modifier le fichier yolov3-tiny.cfg au niveau de la résolution.

```
# Training
# batch=64
# subdivisions=2
width=50
```

```
height=50
channels=3
momentum=0.9
```

On modifie les valeurs de width et height à 50.

On lance ensuite la détection en temps réel :

```
./darknet detector demo cfg/coco.data cfg/yolov3-tiny.cfg yolov3-tiny.weights
```

Maintenant que l'on a un retour, on peut créer un script sous python qui permettra de faire la détection d'éléments et qui permettra d'envoyer une alerte par rapport à l'élément détecté le serveur Discord créé précédemment.

Pour commencer le script, on fait l'import de modules :

```
import cv2
import numpy as np
import argparse
import time
```

On fait l'import d'opencv

En ce qui concerne la partie Yolo, on utilisera les fichiers Yolov3-tiny, c'est pour les mêmes raisons que décrites précédemment.

Il nous faudra aussi un fichier qui se nomme coco.names, ce fichier contient les noms des différents objets que Yolo est capable d'identifier.

On ajoute la fonction load_yolo qui permettra de récupérer les fichiers dont nous avons besoin pour faire la détection :

```
def load_yolo():
    net = cv2.dnn.readNet("yolov3-tiny.weights", "yolov3-tiny.cfg")
    classes = []
    with open("coco.names", "r") as f:
        classes = [line.strip() for line in f.readlines()]

    layers_names = net.getLayerNames()
    output_layers = [layers_names[i][0]-1 for i in net.getUnconnectedOutLayers()]
    colors = np.random.uniform(0, 255, size=(len(classes), 3))
    return net, classes, colors, output_layers
```

Les noms présents dans le fichier coco.names sont triés et classés dans un tableau nommé classes.

Comme avec le programme darknet on peut lui faire analyser des images, on crée alors la fonction load_images :

```
def load_image(img_path):
    # image loading
    img = cv2.imread(img_path)
    img = cv2.resize(img, None, fx=0.4, fy=0.4)
    height, width, channels = img.shape
    return img, height, width, channels
```

La fonction `detect_objets` permet des prédictions correctes, en utilisant les modules fournis par `opencv`, cela nous permet de régler des paramètres de détection :

```
def detect_objects(img, net, outputLayers):
    blob = cv2.dnn.blobFromImage(img, scalefactor=0.00392, size=(320, 320), mean=(0, 0, 0), swapRB=True,
    crop=False)
    net.setInput(blob)
    outputs = net.forward(outputLayers)
    return blob, outputs
```

Dans la fonction `get_box_dimensions()`, une liste appelée `scores` est créée qui stocke la valeur correspondant à chaque objet. On cherche ensuite la valeur classe ayant le plus haut niveau de `scores` en utilisant `np.argmax()`. On obtient le nom de la classe correspondant à l'indice à partir de la liste « classes » de la partie `load_yolo()`.

La fonction `draw_label` est la fonction qui permet d'avoir le cadre lors de la détection, pour créer ce cadre, on utilise encore une fois les modules d'`opencv` `cv2.dnn.NMSBoxes()`, on lui donne aussi la forme et le texte à associer.

```
def draw_labels(boxes, confs, colors, class_ids, classes, img):
    indexes = cv2.dnn.NMSBoxes(boxes, confs, 0.5, 0.4)
    font = cv2.FONT_HERSHEY_PLAIN
    for i in range(len(boxes)):
        if i in indexes:
            x, y, w, h = boxes[i]
            label = str(classes[class_ids[i]])
            color = colors[i]
            cv2.rectangle(img, (x,y), (x+w, y+h), color, 2)
            cv2.putText(img, label, (x, y - 5), font, 1, color, 1)
    cv2.imshow("Image", img)
```

La dernière partie qui nous intéresse est la détection en temps réel au travers d'un flux vidéo fournit par une webcam :

```
def webcam_detect():
    model, classes, colors, output_layers = load_yolo()
    cap = start_webcam()
    label = []
    boxes = []
    while True:
        _, frame = cap.read()
        height, width, channels = frame.shape
        blob, outputs = detect_objects(frame, model, output_layers)
        boxes, confs, class_ids = get_box_dimensions(outputs, height, width)
        draw_labels(boxes, confs, colors, class_ids, classes, frame)
        indexes = cv2.dnn.NMSBoxes(boxes, confs, 0.5, 0.4)
        key = cv2.waitKey(1)
        for i in range(len(boxes)):
            if i in indexes:
                label = str(classes[class_ids[i]])

        if class_ids != []:
            print("un élément a été détecté " + "ID:" + str(class_ids) + " Nom de l'élément: " + str(label))
        if key == 27:
            break
```



```
else:  
    print "je vois rien"
```

La base est identique aux fonctions `image_detect` et `start_video` (voir annexe), néanmoins on doit utiliser la webcam, par défaut OpenCV utilise l'ID `c0`, sur un ordinateur portable par exemple, c'est la webcam intégrée. Ensuite, afin d'avoir un retour texte sur le script, on reprend une partie qui était présent dans `draw_layer` afin de récupérer le nom de l'élément détecté. On définit une condition pour avoir un retour. On utilise le tableau `class_ids` qui contient les id des éléments présent dans le fichier `coco.names` afin d'avoir un retour

Ici, on récupère l'ID de l'élément ainsi que son nom. Ces informations sont essentielles pour la partie « envoi des alertes ».

On lance le programme avec la commande :

```
python yolo.py --webcam c0
```

Et on obtient :

```
buddy@buddy-G5-5590:~/projet_perso/darknet$ python yolo.py --webcam c0  
---- Démarrage de la Webcam pour la surveillance ----  
un élément a été détecté ID:[0] Nom de l'élément: []  
un élément a été détecté ID:[0] Nom de l'élément: person
```

Ainsi qu'un retour d'image :

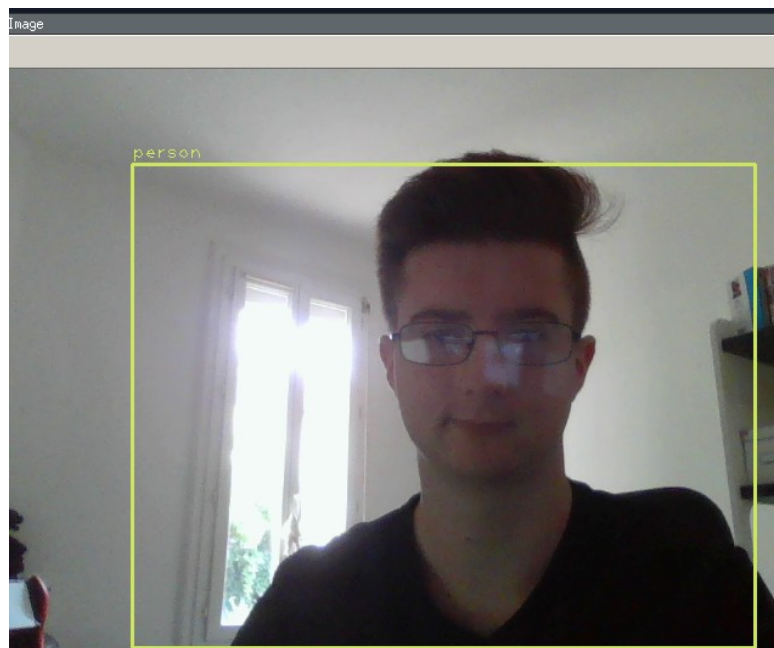


Figure 38: Retour image OpenCV et Yolo

Envoi des alertes :

Pour pouvoir envoyer des alertes sur le serveur discord il faut au préalable installer le module python permettant de le faire. Le nom de ce module est discord-webhook, on l'installe avec la commande suivante :

```
pip install discord-webhook
```

Une fois installé on modifie notre programme afin qu'il puisse envoyer les messages :
On importe les module :

```
from discord_webhook import DiscordWebhook, DiscordEmbed
```

Ensuite on va dans la fonction webcam_detect dans la partie condition et on modifie cette partie :

```
if class_ids != []:
    print("un élément a été détecté "+"ID:"+str(class_ids)+" Nom de l'élément: " +str(label))
    cv2.imwrite("detection.jpg",frame)
    webhook = DiscordWebhook(url='https://discordapp.com/api/webhooks/732905512651456552/
CWJVV0EWuLZoGauFdb4mQ-ZqhmO38KSXujd3QaZ4cY2zNPaoYsxhXuZl6nk2jGJ6q2XM')
    with open("/home/buddy/projet_perso/darknet/detection.jpg", "rb") as f:
        webhook.add_file(file=f.read(), filename='detection.jpg')
    embed = DiscordEmbed(title='Détection', description='Un élément a été détecté son ID est '+str(class_ids)
+' ce qui correspond à '+str(label))
    embed.set_thumbnail(url='attachment://detection.jpg')

    webhook.add_embed(embed)

    response = webhook.execute()
```

Le code présenté permet de capturer une image de l'élément qui a été détecté, de la sauvegarder, et ensuite l'envoyer sur le serveur Discord. Pour ce faire, on définit la variable webhook avec l'URL du serveur, puis on lui demande d'aller chercher l'image capturée précédemment, de la renommer, on peut la renommer en fonction du secteur auquel le dispositif sera rattaché. On lui donne aussi un titre et la possibilité de lui donner une miniature.

En parallèle on crée un webhook sur le serveur qui sera propre à Yolo et OpenCV :

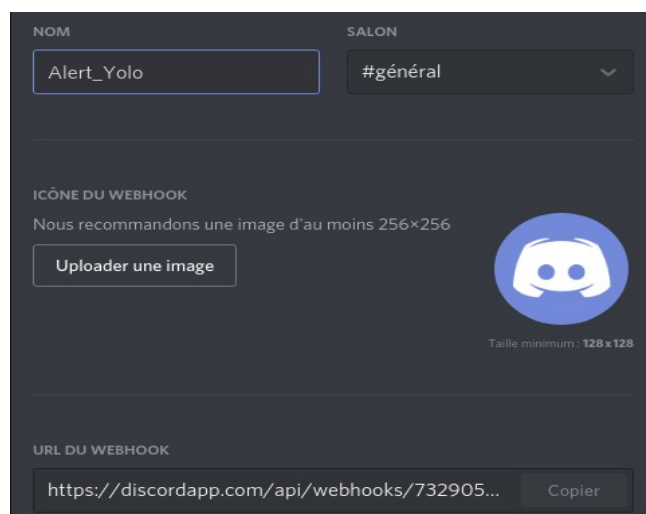


Figure 39: Webhook OpenCV

On lui attribue un nom avec une image (ou non), le salon dans lequel il doit publier des messages, ainsi que l'URL qu'on attribue dans le code.

Une fois les paramètres inscrits, il nous reste plus qu'à tester les modifications :

```
buddy@buddy-G5-5590:~/projet_perso/darknet$ python yolo2.py --webcam c0
---- Démarrage de la Webcam pour la surveillance ----
un élément a été détecté ID:[0] Nom de l'élément: person
un élément a été détecté ID:[0] Nom de l'élément: person
un élément a été détecté ID:[0] Nom de l'élément: person
```

Tout en gardant un retour texte sur le programme, sur Discord les messages ont bien été envoyés :



Figure 40: Retour messages sur Discord

Nous avons un retour de l'application avec la photo prise. On peut remarquer qu'avant que la personne apparaisse, l'ID de l'élément détecté est 56, ce n'est pas une erreur, l'ID 56 correspond à « chair », le programme avait détecté le bout de la chaise qui apparaît dans la première photo.

Les photos sont consultables en cliquant sur l'une d'entre elles et permettent donc un « stockage » des photos avec un historique d'envoi.

N.B la version complète du script sera disponible en annexe.

Suivi des animaux en forêt

Ayant pu proposer des solutions pour la flore, il nous reste à terminer la partie « Tech » sur la faune.

Sur la partie OpenCV et Yolo, nous avons brièvement parlé du sujet du suivi des animaux en forêt.

Bien que le dispositif permet la surveillance de la forêt, il sert aussi de détecteur de mouvements.

Ce dispositif peut être placé dans la forêt ou alors aux extrémités de la forêt pour réguler les entrées et sorties, de n'importe quel éléments animaux ou humains.



Figure 41: Schéma placement des caméras

On peut ajouter à ces dispositifs un module GPS comme ça en plus d'envoyer ce qu'ils ont détectés, on pourra connaître leurs positions, car si le secteur est grand et que l'on met plusieurs caméras, on risque de confondre.

Pour les animaux sauvages qui passent par la forêt, la caméra enverra les informations, néanmoins pour les animaux protégés, on peut mettre en place un collier GPS dont les données seront envoyées toutes les 10 minutes pour conserver une certaine autonomie de la batterie, afin de connaître leurs habitudes (lieux favoris, les différents regroupements, ...) mais aussi pour connaître leurs positions en cas de problèmes (braconnage,...).

Ces colliers GPS, pourront être couplés à des capteurs cardiaques pour savoir s'ils sont stressés ou en danger. Ces données seront envoyées via le protocole GSM afin d'avoir un suivi régulier et d'agir en cas de besoin. On peut aussi mettre en place un microphone pour en savoir plus sur le langage qu'ils utilisent entre eux.

Pour les données du microphone, on peut mettre en place sur le dispositif, un support de stockage (Carte Micro SD) afin de stocker les sons, afin qu'ils soient étudiés ultérieurement.

De plus, si un animal passe devant une caméra et que celle-ci le détecte, on pourra joindre les données GPS du collier avec les données GPS de la caméra afin d'avoir un suivi précis.

Le schéma suivant est une ébauche d'un collier GPS pour les animaux protégés :

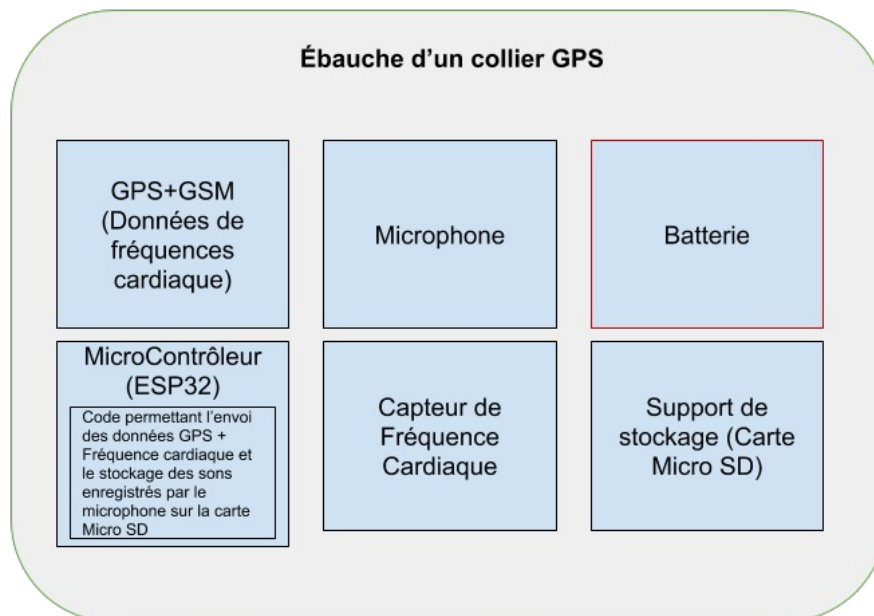


Figure 42: ébauche collier GPS

Conclusion de la partie technique :

Ces différentes sous-parties ont permis de comprendre l'intérêt de la surveillance de la faune et de la flore, tout en permettant le développement de solutions afin de permettre un suivi détaillé des événements présent dans une forêt (température, humidité, ...). De plus, cela a permis de comprendre le fonctionnement du traitement d'image fourni par OpenCV et de comment on peut l'utiliser dans ce domaine et comment permettre l'envoi d'alertes s'il y a une détection.

Les solutions proposées dans cette partie sont dans la plupart à moitié théorique, mais il serait tout à fait envisageable, de proposer ce genre de solution pour une production.

Conclusion :

Pour rappel, la problématique du mémoire était : Comment l'IOT, peut-il permettre un suivi détaillé et une potentielle préservation de la faune et de la flore en milieu forestier ?

Il a fallu dans un premier temps définir la notion d'Internet des objets dans le domaine forestier au travers des projets déjà existants. Cela a permis de réfléchir sur des solutions qui peuvent être envisageables et utilisables par tous.

Il convenait alors de s'intéresser à la mise en place d'une architecture, avec comme problématique pour la partie recherche : Comment avoir un suivi détaillé des différents éléments présent dans une forêt ? Quels composants utiliser ? Quels protocoles réseaux utiliser ? Peut-on avoir un retour vidéo ou d'images de la forêt ? Comment faire si un incident a lieu dans la forêt ? Comment être alerté ? Bien entendu mettre en place telle une architecture, n'est pas sans contraintes. Les contraintes énergétiques sont en soit des contraintes assez conséquentes, pour les dispositifs qui récupèrent les informations, ceux-ci sont auto-alimentés, cependant les gateways (ou passerelle réseaux) ont besoins d'avoir une alimentation constante, ce qui n'est pas évident en pleine forêt. De plus, il faut aussi prendre en compte la couverture réseau. A-t-on une couverture suffisante ? Dans le cas de la simulation nous avons utilisé le réseau GSM.

Au travers de ce mémoire, nous avons pu voir que l'utilisation de l'Internet des Objets dans le domaine forestier, devrait être un indispensable à la préservation de la faune et de la flore.

Si nous utilisons l'IOT, nous avons la possibilité de surveiller et d'agir en conséquence, aussi bien au niveau de la faune avec le suivi GPS, cardiaque,... Mais aussi au niveau de la flore avec les différents capteurs de température, d'humidité et de gaz. Nous pouvons aussi ajouter les dispositifs utilisant OpenCV et la librairie Yolo afin d'avoir un suivi visuel des différents secteurs de la forêt et une détection des différents éléments qui peuvent survenir (animaux, hommes, ...). Si l'IOT était utilisé dans ce domaine, on pourrait éviter ou alors réduire la propagation d'un grand nombre d'incendies, nous prenons l'exemple de l'incendie qui survient à la date du 30 et 31 Juillet 2020 à Anglet, dans la forêt de Chiberta.

Ce travail de mémoire se voulait en partie théorique, à cause, du manque de matériel pour mettre en place une solution pertinente et qui en plus était un remplacement des stages qui n'ont pas eu lieu à cause de la crise sanitaire du Covid-19, néanmoins nous avons pu mettre en place des solutions qui peuvent être utilisées (Exemple : la simulation et OpenCV).

Mais il serait pertinent d'approfondir ce sujet de préservation de la faune et de la flore afin que les solutions abordées ici puissent être, un jour, utilisées dans les forêts.

Annexes :

Illustration 1: Code OpenCV Yolo

```
# -*- coding: utf-8 -*-
import cv2
import numpy as np
import argparse
import time
import socket
from discord_webhook import DiscordWebhook, DiscordEmbed

parser = argparse.ArgumentParser()
parser.add_argument('--webcam', help="True/False", default=False)
parser.add_argument('--play_video', help="True/False", default=False)
parser.add_argument('--image', help="True/False", default=False)
parser.add_argument('--video_path', help="Path of video file", default="videos/car_on_road.mp4")
parser.add_argument('--image_path', help="Path of image to detect objects",
default="data/chiens.jpg")
parser.add_argument('--verbose', help="To print statements", default=True)
args = parser.parse_args()

#Load yolo
def load_yolo():
    net = cv2.dnn.readNet("yolov3-tiny.weights", "yolov3-tiny.cfg")
    classes = []
    with open("coco.names", "r") as f:
        classes = [line.strip() for line in f.readlines()]

    layers_names = net.getLayerNames()
    output_layers = [layers_names[i][0]-1 for i in net.getUnconnectedOutLayers()]
    colors = np.random.uniform(0, 255, size=(len(classes), 3))
    return net, classes, colors, output_layers

def load_image(img_path):
    # image loading
    img = cv2.imread(img_path)
    img = cv2.resize(img, None, fx=0.4, fy=0.4)
    height, width, channels = img.shape
    return img, height, width, channels

def start_webcam():
    cap = cv2.VideoCapture(0)

    return cap

def display_blob(blob):
    """
    Three images each for RED, GREEN, BLUE channel
    """
```

```

'''
    for b in blob:
        for n, imgb in enumerate(b):
            cv2.imshow(str(n), imgb)

def detect_objects(img, net, outputLayers):
    blob = cv2.dnn.blobFromImage(img, scalefactor=0.00392, size=(320, 320), mean=(0, 0, 0),
swapRB=True, crop=False)
    net.setInput(blob)
    outputs = net.forward(outputLayers)
    return blob, outputs

def get_box_dimensions(outputs, height, width):
    boxes = []
    confs = []
    class_ids = []
    for output in outputs:
        for detect in output:
            scores = detect[5:]
            class_id = np.argmax(scores)
            conf = scores[class_id]
            if conf > 0.3:
                center_x = int(detect[0] * width)
                center_y = int(detect[1] * height)
                w = int(detect[2] * width)
                h = int(detect[3] * height)
                x = int(center_x - w/2)
                y = int(center_y - h / 2)
                boxes.append([x, y, w, h])
                confs.append(float(conf))
                class_ids.append(class_id)

    return boxes, confs, class_ids

def draw_labels(boxes, confs, colors, class_ids, classes, img):
    indexes = cv2.dnn.NMSBoxes(boxes, confs, 0.5, 0.4)
    font = cv2.FONT_HERSHEY_PLAIN
    for i in range(len(boxes)):
        if i in indexes:
            x, y, w, h = boxes[i]
            label = str(classes[class_ids[i]])
            color = colors[i]
            cv2.rectangle(img, (x,y), (x+w, y+h), color, 2)
            cv2.putText(img, label, (x, y - 5), font, 1, color, 1)
    cv2.imshow("Image", img)

def image_detect(img_path):
    model, classes, colors, output_layers = load_yolo()
    image, height, width, channels = load_image(img_path)
    blob, outputs = detect_objects(image, model, output_layers)
    boxes, confs, class_ids = get_box_dimensions(outputs, height, width)

```

```

draw_labels(boxes, confs, colors, class_ids, classes, image)
while True:
    key = cv2.waitKey(1)
    if key == 27:
        break

def webcam_detect():
    model, classes, colors, output_layers = load_yolo()
    cap = start_webcam()
    label = []
    boxes = []
    while True:
        _, frame = cap.read()
        height, width, channels = frame.shape
        blob, outputs = detect_objects(frame, model, output_layers)
        boxes, confs, class_ids = get_box_dimensions(outputs, height, width)
        draw_labels(boxes, confs, colors, class_ids, classes, frame)
    indexes = cv2.dnn.NMSBoxes(boxes, confs, 0.5, 0.4)
    key = cv2.waitKey(1)
    for i in range(len(boxes)):
        if i in indexes:
            label = str(classes[class_ids[i]])

    if class_ids != []:
        print("un élément a été détecté "+"ID:"+str(class_ids)+" Nom de l'élément: "
+str(label))
        cv2.imwrite("detection.jpg",frame)
        webhook =
DiscordWebhook(url='https://discordapp.com/api/webhooks/732905512651456552/
CWJVV0EWuLZoGauFdb4mQ-ZqhmO38KSXujd3QaZ4cY2zNP0YsXhXuZl6nk2jGJ6q2XM')
        with open("/home/buddy/projet_perso/darknet/detection.jpg", "rb") as f:
            webhook.add_file(file=f.read(), filename='detection.jpg')
            embed = DiscordEmbed(title='Détection', description='Un élément a été détecté son
ID est '+str(class_ids)+' ce qui correspond à '+str(label))
            embed.set_thumbnail(url='attachment://detection.jpg')

            webhook.add_embed(embed)
            #webhook =
DiscordWebhook(url='https://discordapp.com/api/webhooks/732905512651456552/
CWJVV0EWuLZoGauFdb4mQ-ZqhmO38KSXujd3QaZ4cY2zNP0YsXhXuZl6nk2jGJ6q2XM',
content='Un élément a été détecté son ID est '+str(class_ids)+' ce qui correspond à '+str(label))
            response = webhook.execute()

    if key == 27:
        break

else:
    #print("elements detecte "+"ID:"+str(class_ids)+" Nom de l'element: " +str(label))
    print "je vois rien"

```



```

cap.release()

def start_video(video_path):
    model, classes, colors, output_layers = load_yolo()
    cap = cv2.VideoCapture(video_path)
    while True:
        _, frame = cap.read()
        height, width, channels = frame.shape
        blob, outputs = detect_objects(frame, model, output_layers)
        boxes, confs, class_ids = get_box_dimensions(outputs, height, width)
        draw_labels(boxes, confs, colors, class_ids, classes, frame)
        key = cv2.waitKey(1)
        if key == 27:
            break
    cap.release()

if __name__ == '__main__':
    webcam = args.webcam
    video_play = args.play_video
    image = args.image
    if webcam:
        if args.verbose:
            print('---- Démarrage de la Webcam pour la surveillance ----')
        webcam_detect()
    if video_play:
        video_path = args.video_path
        if args.verbose:
            print('Opening '+video_path+" .... ")
        start_video(video_path)
    if image:
        image_path = args.image_path
        if args.verbose:
            print("Opening "+image_path+" .... ")
        image_detect(image_path)

cv2.destroyAllWindows()

```

Sources :

- Figure 1 : <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>
- Figures 2 et 3 : <https://www.connectwave.fr/techno-appli-iot/iot/reseaux-et-infrastructures-iot/>
- <https://www.synox.io/4-choses-a-savoir-sur-linternet-des-objets/>
- <https://azure.microsoft.com/fr-fr/overview/internet-of-things-iot/iot-technology-protocols/>
- Bluetooth (BLE) : <https://blog.groupe-sii.com/le-ble-bluetooth-low-energy/>
- <https://moodle.didex.fr/moodle/course/view.php?id=654>
- <https://moodle.didex.fr/moodle/course/view.php?id=589>
- <https://www.ecologique-solidaire.gouv.fr/prevention-des-feux-foret>
- <http://smartforest.pt/>
- projet de icjiter : <https://pdfs.semanticscholar.org/dc1e/4a751998713d7e51859876ae2ada849d0cee.pdf>
- Schéma IoTree : <https://business.esa.int/projects/iotrees>
- Rainforest : <https://rfcx.org/home>
- Rainforest (vidéo TED) : https://www.youtube.com/watch?time_continue=376&v=xPK2Ch90xWo&feature=emb_logo
- Rainforest schéma : <https://www.networkworld.com/article/3066880/protecting-the-rainforests-with-iot-and-recycled-phones.html>
- Mesure Co2 : <https://th-industrie.com/content/13-mesure-co2>
- <https://theiotmagazine.com/iot-in-the-wild-59c2525d62ef>
- Dashboard : <https://grafana.com/docs/grafana/latest/features/dashboard/dashboards/>
- DHT11 : <https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf>
- Soil Moisture : <https://www.sparkfun.com/products/13322>
- HC-SR04 : <https://www.adafruit.com/product/3942>
- MICS5524 : <https://boutique.semageek.com/fr/834-capteur-de-gaz-mics5524-co-alcool-voc.html>
- SGP30 : <https://www.adafruit.com/product/3709>
- PiCam : <https://uk.pi-supply.com/products/raspberry-pi-camera-board-v2-1-8mp-1080p?lang=fr>

- Discord webhook : <https://medium.com/@maxwellflitton/free-python-alert-systems-with-discord-6b04d314455c>
- Yolo python : <https://towardsdatascience.com/object-detection-using-yolov3-and-opencv-19ee0792a420>
- Opencv : https://docs.opencv.org/master/d2/de6/tutorial_py_setup_in_ubuntu.html

Bibliographie :

- Analytics for the Internet of Things (IoT) **Minteer, Andrew** 2017
- IoT Projects with Bluetooth Low Energy **Bhargava, Madhur** 2017
- IoT: Building Arduino-Based Projects **Waher, Peter Seneviratne, Pradeeka Russell, Brian** 2016

Citations :

[1] M. Han and H. Zhang, "Business intelligence architecture based on internet of things "