

## Rapport de Stage technique

Réalisé au sein de l'entreprise Innovatel



**Sous le thème :**

# Supervision d'un parking intelligent à l'aide des caméras de surveillance.

**Réalisé par :**

Ahbar Abdellah  
Sikou Moahmed

**Encadré par :**

Errami Ahmed

**Encadrant d'entreprise :**

Coulibaly Moussa

**Soutenu le : 17/10/2022**

# Résumé

**Pour améliorer la qualité de son parking intelligent, Innovatel propose plusieurs solutions pour atteindre son objectif. Dans ce sens, une étude de la possibilité d'optimiser le coût et améliorer les résultats des solutions proposées était faite.**

**Le présent rapport consiste à une élaboration de plusieurs outils de l'amélioration continue pour éliminer les anciennes méthodes basées sur l'alimentation des batteries à durée de vie très petite.**

**Pour ce faire, nous avons jugé utile de commencer par un diagnostic de l'existant, par la suite nous avons réalisé une analyse pertinente des contraintes des solutions existantes, avant de se lancer dans la recherche de nouvelles solutions.**

**Ces actions ont porté principalement sur la vision par ordinateur en utilisant plusieurs outils pour pouvoir superviser les voitures et aussi suivre l'emplacement des voitures en temps réelle en utilisant des caméras de surveillance déjà installées dans le parking.**

# **Abstract**

**To improve the quality of its intelligent parking, Innovatel proposes several solutions to achieve its objective. In this sense, a study of the possibility to optimize the cost and improve the results of the proposed solutions.**

**The present report consists of an elaboration of several tools of continuous improvement to eliminate the old methods based on the supply of batteries with very small life.**

**To do this, we found it useful to start with a diagnosis of the existing, then we made a relevant analysis of the constraints of existing solutions, before embarking on the search for new solutions.**

**These actions were mainly focused on computer vision using several tools to be able to supervise the cars and also to follow the location of the cars in real time using surveillance cameras already installed in the parking lot.**

# Table des Matières

|  |    |
|--|----|
| Résumé   | 2  |
| Abstract   | 3  |
| Table des Matières   | 4  |
| Liste des figures  | 5  |
| Table des abréviations                                     | 7  |
| Introduction générale                                      | 8  |
| Contexte général du projet                                 | 14 |
| 1. Les Parkings intelligents :                             | 15 |
| 3. Problématique et Solution existante                     | 16 |
| 3.1 Description du parking Innovatel Engineering S.A.R.L : | 16 |
| 3.2 Solution existante                                     | 19 |
| 3.3 Description de la problématique :                      | 19 |
| 4. Cahier des charges                                      | 20 |
| 5. Solution proposée et démarches à suivre                 | 20 |
| Vision assistée par ordinateur                             | 22 |
| 1. Généralités sur la vision par ordinateur                | 23 |
| 1.1 Fonctionnements  | 23 |
| 1.2 Applications   | 24 |
| 2. Algorithme de détection des objets « YOLO V4 » :        | 27 |
| 3. Les outils utilisés :                                   | 27 |
| 1. Implémentation de la solution                           | 32 |
| II.1 Test des résultats :                                  | 41 |
| Conclusion générale :                                      | 66 |
| Bibliographies   | 68 |

# Liste des figures

|   |    |
|---|----|
| Figure 1: Le parking d'Innovatel Engineering SARL .....   | 15 |
| Figure 2 : Photo satellitaire du parking .....  | 16 |
| Figure 3 :Plan architecte du parking AutoCAD .....  | 17 |
| Figure 4 : les capteurs installés sur places.....   | 17 |
| Figure5 : Capteur de présence .....   | 18 |
| Figure 6 : architecture de capteurs à distance lier par antennes .....                              | 18 |
| Figure 7: Panneaux du parking et places réservées .....   | 19 |
| Figure 8 : Détecteur de présence .....  | 19 |
| Figure 9 : Exemple de vision par ordinateur .....   | 23 |
| Figure 10: Exemple de classification .....  | 25 |
| Figure 11 : Exemple de détection des objets.....  | 26 |
| Figure 12: Exemple de suivi d'objet.....  | 26 |
| Figure 13 : Exemple de recherche d'image .....  | 27 |
| Figure 14: architecture de YOLO - v4 .....  | 27 |
| Figure 15 : logo python .....   | 28 |
| Figure 16 : Logo OpenCV.....  | 29 |
| Figure 17 : Logo Visual studio code .....   | 30 |
| Figure 18: Les flux vidéo collectés .....   | 32 |
| Figure 19: Les fichiers nécessaires pour travailler avec YOLO v4 .....                              | 32 |
| Figure 20 : installation des bibliothèques sur la terminale de V.S code .....                       | 33 |
| Figure 21: Une partie du code de tracking sur V.S code .....  | 33 |
| Figure 22 : Organigramme du détection des voitures .....  | 33 |
| Figure 23: un échantillon de la détection .....   | 34 |
| Figure 25: Frame 235 d'une vidéo échantillonnée .....   | 34 |
| Figure 24: Frame 595 d'une vidéo échantillonnée .....   | 34 |
| Figure 26 : l'algorithme mise en place pour le tracking .....                                       | 34 |
| Figure 27: l'algorithme de tracking .....   | 35 |
| Figure 28 : Une partie du code du suivi .....   | 36 |
| Figure 29 : data enregistrer sous format CSV .....  | 36 |
| Figure 30 : Code pour annulation de la perspective.....   | 36 |
| Figure 31 : frame avec annulation d'effet de perspective .....                                      | 37 |
| Figure 32 : frame avec effet de perspective du parking de l'ENSEM .....                             | 37 |
| Figure 33 : le programme responsable de la restitution des points des corners du parking .....      | 38 |
| Figure 34: vue après annulation de la perspective .....   | 38 |
| Figure 35 : vue directe .....   | 38 |
| Figure 36 : le programme parkingSpacePicker.py qui permet de sélectionner les spots de parking..... | 39 |
| Figure 37: prototype du parking avec un flux de caméra en temps réel .....                          | 39 |
| Figure 38 : RIO dans le parking de l'ENSEM .....  | 40 |



# Table des abréviations

\*par ordre alphabétique

| Abréviation | Désignation                                     |
|-------------|---|
| IBM         | International Business Machines Corporation     |
| IOT         | Internet des objets                             |
| API         | Les interfaces de programmation d'applications  |
| IA          | L'intelligence artificielle                     |
| CBD         | Central Business District                       |
| YOLO        | Real-time Object Detection (YOU ONLY LOOK ONCE) |
| OCR         | Reconnaissance optique des caractères           |
| OSOD        | Détection d'objets One-Shot                     |

# Introduction générale

Plusieurs définitions de ce qui rend une ville "intelligente". IBM définit une ville intelligente comme « une ville qui fait un usage optimal de toutes les informations interconnectées disponibles aujourd'hui pour mieux comprendre et contrôler ses opérations et optimiser l'utilisation des ressources limitées ». En résumé, une ville intelligente utilise un cadre de technologies de l'information et de la communication pour créer, déployer et promouvoir des pratiques de développement permettant de relever les défis urbains et de créer une infrastructure technologique et durable commune.

Les villes intelligentes utilisent une variété de logiciels, d'interfaces utilisateur et de réseaux de communication aux côtés de l'Internet des objets (IoT) pour offrir des solutions connectées au public. Parmi ceux-ci, l'IoT est le plus important. C'est un réseau d'appareils connectés qui communiquent et échangent des données. Il peut s'agir de véhicules, d'appareils ménagers ou de capteurs installés dans les rues. Les données collectées à partir de ces appareils sont stockées dans le Cloud, ou sur des serveurs pour permettre d'améliorer l'efficacité des secteurs public et privé, d'avoir des avantages économiques et d'améliorer la vie des citoyens.

De nombreux dispositifs IoT utilisent l'Edge computing, qui garantit que seules les données les plus pertinentes et les plus importantes sont transmises sur le réseau de communication. En outre, un système de sécurité est mis en œuvre pour protéger, surveiller et contrôler la transmission des données du réseau de la ville intelligente et empêcher tout accès non autorisé au réseau IoT de la ville.

Parallèlement aux solutions IoT, les villes intelligentes utilisent également des technologies, notamment :

- Les interfaces de programmation d'applications (API)
- L'intelligence artificielle (IA)
- Services d'informatique en nuage
- Tableaux de bord
- Communications entre machines
- Réseaux maillés
- Computer vision
- Edge computing

- Communication agent
- Le forecasting

Toute ville souffre du problème de parking à cause de l'augmentation de l'utilisation des voitures personnelles. D'où la nécessité de créer des parkings intelligents adaptés aux besoins des citoyens. De plus ces parkings doivent contribuer à la collecte de données pour d'autres utilisations au niveau de l'économie ou de sécurité.

Consciente de l'importance de l'IoT dans le domaine de la gestion des parking, *Innovatel*, a su suivre le rythme du progrès industriel, continuer d'apporter constamment ces savoir-faire afin qu'elle puisse améliorer leurs performances pour s'adapter à la concurrence actuelle. Dans cette optique, *Innovatel*, nous a confié la mission de la mise en place d'une solution optimale pour un parking intelligent.

Ce parking est parmi les premiers parking intelligent au Maroc, il est en cours de développement. Dans ce cadre, la société nous a proposé un stage dans pour l'amélioration du parking en utilisant des techniques de la vision par ordinateur pour minimiser le coût et faciliter le déploiement de ce genre de solution sur les infrastructures du parking existants.

Ce rapport décrit les différentes étapes de la réalisation de ce projet, il est divisé en trois chapitres :

Le premier chapitre est dédié pour le contexte générale du projet, le deuxième chapitre présentera les différentes notions abordées ainsi que la démarche suivie pour réaliser notre projet, en ce qui concerne le dernier chapitre, il présentera la phase de réalisation du prototype, les résultats et des tests du système proposé. Enfin, le rapport sera clôturé par une conclusion générale et les différentes perspectives du projet.

# Présentation de La société Innovatel Engineering d'étude et d'ingénierie

La société Innovatel Engineering est un bureau d'étude et d'ingénierie spécialisé en courant faible (CFA) au service de la Smart City ainsi de l'Industrie 4.0 dans le cadre des projets tertiaires et industriels ainsi que le transfert technologique et Recherche et développement

Prestation de service :

Le bureau d'étude et d'ingénierie a pour objet tant au Maroc qu'à l'étranger, ce qui suit:

- Etude et ingénierie du lot de Courant Faible (CFA)
  - i. Etude et ingénierie de la discipline de la sécurité électronique :
    - a. Système intelligent de la Vidéosurveillance IP
    - b. Conception de la salle de télésurveillance à la norme mondiale
    - c. Système de Contrôle d'accès physique
    - d. Système de la protection pérимétrique des sites sensibles
    - e. Système de détection d'incendie et HVAC
  - ii. Etude, ingénierie et réalisation de la Digitalisation industrielle et l'Industrie 4.0
  - iii. Conseil, Consulting, Expertise et Audit de la sûreté et de la sécurité électronique des sites industriels, tertiaires et militaires
    - i. Etude, ingénierie et réalisation de mise en place de Smart Building :
      - a. Livrable d'un système Smart Parking
      - b. Livrable d'un système Smart Grid et Smart Lighting
      - c. Livrable d'un système Smart Mobilité électrique et durable
      - d. Livrable d'un système de gestion de qualité de l'air et de l'eau
    - iv. Etude et ingénierie de la discipline de la communication et informatique :
      - a. Réseau informatique passif / actif et la télécommunication Urbaine FTTH
      - b. Ingénierie de la conception de la Datacenter
      - c. Système Téléphonique Industrielle, PA système et Sonorisation
      - d. Multimédia et Audiovisuel (Affichage dynamique & IPTV) (Hôtelier, Banque et Tertiaire)
    - v. Etude et ingénierie de la discipline d'Infrastructure Electrique :
      - a. Electricité industriel (BT, HT, THT)
      - b. Bâtiment Intelligent (Electricité Général & Domotique, GTC & GTB)
      - c. Efficacité Energétique (Energie Renouvelable et Isolation Thermique)
    - vi. Etude et ingénierie d'Automatisation et de Gestion Intelligente de Flux des véhicules

- vii. Etude et ingénierie de la discipline d'instrumentation & Commande/Contrôle/Mesure :
  - a. Capteurs industriels intelligents connectés
  - b. Automate Programmable Industriel et SCADA
- viii. Harmonisation et Alignement des Offres Techniques des soumissionnaires
- ix. Revue Technique des lots de courants faible pour les différentes phases d'études (APS, APD et DCE).
- x. Suivi, accompagnement et gestion de vos projets de bout en bout jusqu'à la réception
- xi. Consulting pour le Transfert Technique et Technologique
- xii. Réalisation des plans d'Ingénierie de Formation technique et technologique

**Liste non exhaustive de nos clients :**

| Client  | Prestation de service   |
|---|---|
| LAFARGEHOLCIM MAROC   | <b>Etude et ingénierie de la Vidéosurveillance IP au sein de l'usine Bouskoura et de l'infrastructure réseau informatique et télécoms</b>   |
| LAFARGEHOLCIM MAROC   | <b>Etude d'une Prestation d'installation d'un système Pyroscan (Camera pyromètre de visualisation de la flamme du Four 2) au sein de l'usine Bouskoura</b>  |
| ONCF (projet en cours)  | <b>Etude et ingénierie d'un projet de la digitalisation de process ferroviaire</b>  |
| Conseil, Ingénierie et Développement (CID)                            | <b>Etude et ingénierie d'un Projet de la digitalisation des flux des camions de chargement de poisson au compte de l'ONP</b>  |
| Jacobs Engineering SA   | <b>Etude et ingénierie de la vidéoprotection du Siège Social OCP et hors site</b><br><b>Etude et ingénierie de la Data Center du Groupe OCP à la Ville Benguerir</b><br><b>Etude et ingénierie VDI d'une usine de chargement d'engrais OCP à RWANDA</b><br><b>Etude et ingénierie d'un système de contrôle d'un nouvel accès du Siège Social OCP</b><br><b>Etude et ingénierie de la vidéosurveillance IP du Pipeline Jorf OCP</b><br><b>Etude et ingénierie du Courant faible de Technopole Foum Oued OCP Laayoune géré par la Fondation OCP</b><br><b>Revue technique de la Tour Banque Populaire</b><br><b>Revue technique du CHU Cheikh Zayed à Bouskoura</b><br><b>Etude et ingénierie d'automatisation de gestion des flux des camions à Jorf Lasfar OCP</b><br><b>Etude relative aux travaux d'aménagement de la zone d'activité économique via Réseau FTTH de Khouribga OCP</b> |
| Ministère des Affaires Etrangères et de la Coopération Internationale | <b>Etude et ingénierie d'un système de Détection d'Incendie et extinction automatique et l'ingénierie d'un système de contrôle d'accès physique via les Antennes RFID</b>   |

|  |  |
|--|--|
| <b>Fondation de la Recherche Scientifique de l'ENSEM</b>                       | <b>Etude et ingénierie de la Data Center du centre d'Innovation et de transfert de technologie à l'université Hassan II à Casablanca</b><br><b>Etude et ingénierie d'un projet Smart Parking de 51 places</b><br><b>Et l'ingénierie d'un Bâtiment Intelligent : domotique Et de la Sécurité électronique</b> |
| <b>Ministère de la Justice</b>   | <b>Etude et ingénierie d'un système de Détection d'Incendie et extinction automatique au sein de 04 tribunaux à Tanger</b>   |
| <b>Siège OFPPT</b>   | <b>Rapport d'audit et d'expertise sur la sûreté et la sécurité électronique instaurée au siège OFPPT</b>   |
| <b>Groupe KITEA</b>  | <b>Etude d'Installation de 370 caméras de surveillance et de système de contrôle d'accès</b>   |
| <b>Compagnie d'aménagement agricole et de développement industriel (A.D.I)</b> | <b>Etude et ingénierie d'un système de sûreté et de la sécurité électronique de la Centrale Thermique SAFI (SAFIEC)</b>  |
| <b>ATOMAS Ingénierie</b>   | <b>Etude et ingénierie d'un système de CFA et VDI de la Tour Mohamed VI Salé</b>   |

**Les Exigences : Normes et Standards :**



## Nos Références :



JACOBS Enqineering SA.



ATOMAS  
INGENIERIE

LafargeHolcim  
Maroc

Marsa  
Maroc



Royaume du Maroc  
Ministère des Affaires Etrangère  
et de la Coopération Internationale

SAFIEC  
SAFI ENERGY COMPANY

ONP  
الموكب الوطني للصياد  
OFFICE NATIONAL DES PECHEES

IRESSEN  
Institut de Recherche en Energie  
Solaire et en Energies Nouvelles

KITEA

ENSEM

OFPPt  
La voie de l'avenir

ETI  
المدرسة العليا للتقنيولوجيا الدار البيضاء  
Ecole Supérieure de Technologie Casablanca



Chapitre I :

## Contexte général du projet

## 1. Les Parkings intelligents :

Le stationnement est limité dans presque toutes les grandes villes du monde, ce qui cause le problème d'embouteillage, la pollution atmosphérique et la frustration des conducteurs. Par exemple, le Central Business District (CBD) de Manhattan compte 109 222 places de stationnement public hors voirie, soit un ratio d'environ une place de stationnement public hors voirie pour 16 travailleurs du CBD. Pourtant, les places de stationnement sont souvent non utilisées. Dans les grands parcs de stationnement, un conducteur ne peut savoir si de nouvelles places viennent de se libérer. La longue durée de recherche d'une place de stationnement vide peut également entraîner une frustration du conducteur si une autre voiture prend la place avant que le conducteur puisse l'atteindre.

Il est donc nécessaire de mettre en place des systèmes de stationnement innovants pour répondre à la demande. Grâce aux communications sans fil, les technologies informatiques, l'électronique, et l'intelligence artificielle on peut avoir une gestion intelligente du stationnement par un axé sur les services afin d'améliorer l'utilisation de l'espace de stationnement ainsi que l'expérience du conducteur.



Figure 1: Le parking d'Innovatel Engineering SARL

### 3. Problématique et Solution existante

#### 3.1 Description du parking Innovatel Engineering S.A.R.L:

Le parking de Innovatel est un smart parking qui se situe au Maroc à Casablanca au Boulevard Abdellah Ibrahim (Ex. Route d'El Jadida) à côté de l'École Nationale Supérieure d'Électricité et Mécanique, exactement au centre d'innovation et de transfert technologique (CITT).

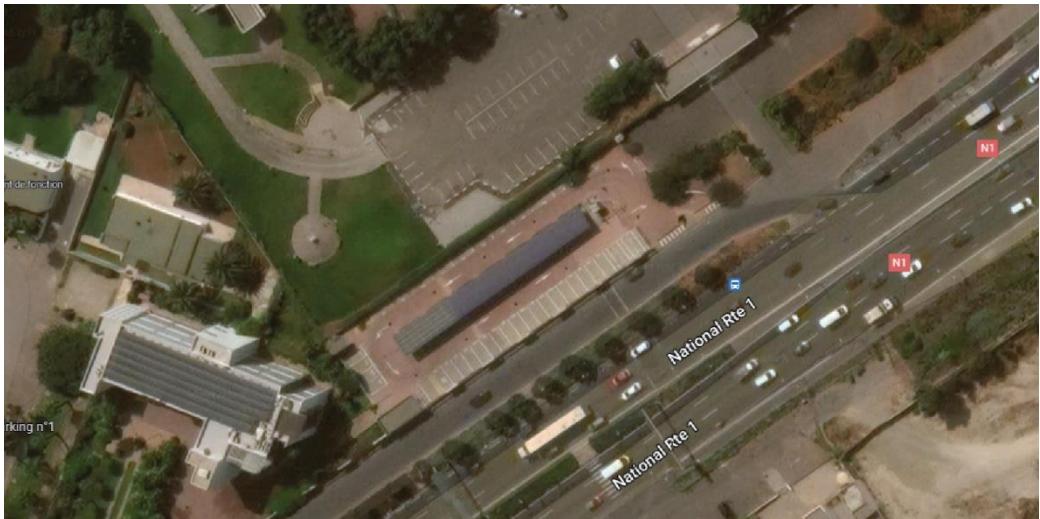


Figure 2 : Photo satellitaire du parking

Le parking contient 52 places, divisées en trois zones, une est toujours réservée pour le STAFF de la société. Ce parking est surveillé par 7 caméras qui couvrent son intégralité plus une caméra occupée d'un module OCR (*Optical Character Recognition*) pour détecter les matricules des voitures automatiquement.



Figure 3 : caméra OCR sur place



Figure 4 : Place réservé aux staffs

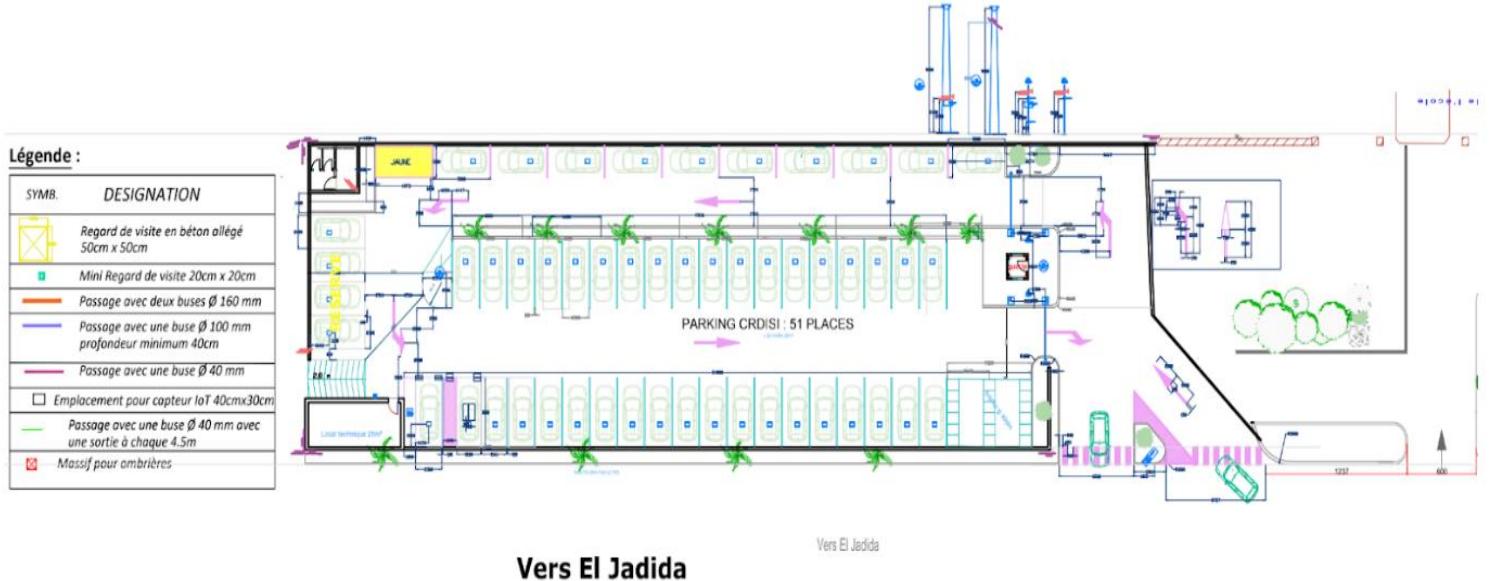


Figure 5 : Plan architecte du parking AutoCAD

Plusieurs solutions de parking intelligent sont mises en place, comme le parking avec capteurs de présence installés au sol :

#### INNOVATEL :



Figure 6 : les capteurs installés sur places

Cette solution se base sur des capteurs de présence installés au sol, ces derniers se communiquent avec l'unité de traitement (ordinateur - serveur ou microcontrôleurs ...) avec une liaison filaire ou sans fils. Enfin l'information sur le nombre de place vide et la capacité totale du parking s'affiche sur un afficheur LED.

Parmi les avantages de cette solution la facilité en termes de communication puisque la quantité des données transférée est très petite (présence ou non + l'identifiant du capteur).



Figure 7 : Capteur de présence

D'autre part la liaison filaire nécessite d'installer un réseau d'alimentation pour les capteurs, et si on utilise la liaison sans fil on est obligé d'installer des antennes pour transférer les données des capteurs vers l'unité de traitement. En plus, l'autonomie des batteries des capteurs est le problème majeur de cette solution.

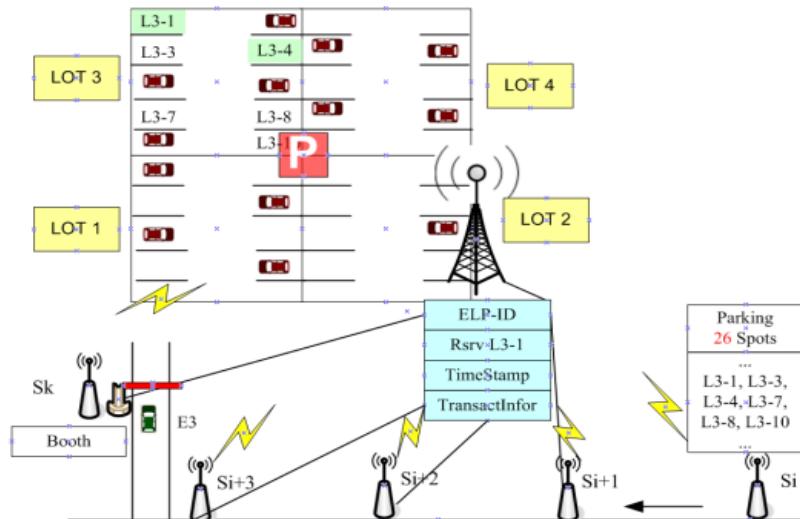


Figure 8 : architecture de capteurs à distance lier par antennes

Dans le parking de la CITT les batteries ne durent pas plus qu'un mois ce qui rend cette solution très coûteuse en termes de changement des batteries des capteurs ainsi que la mise en place de ces capteurs.

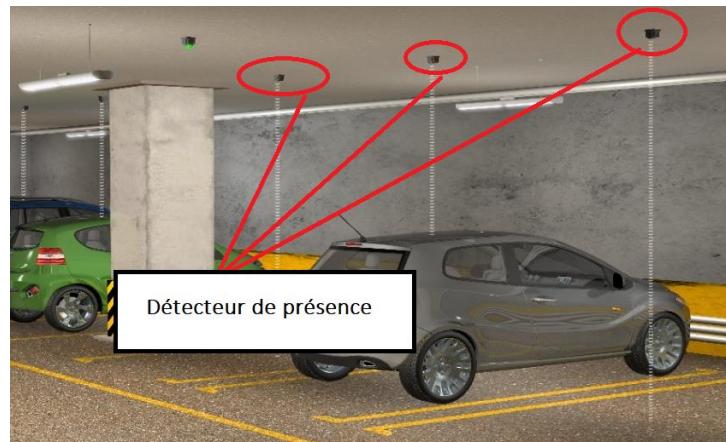


*Figure 9: Panneaux du parking et places réservées*

Dans le cadre de ce projet de fin d'année, l'organisme **Innovatel Engineering Sarl** pose la problématique de la durée de vie des batteries des capteurs de présence. Le problème qui fait que le nombre indiqué par le panneau est toujours 51 places.

### 3.2 Solution existante

Il existe un alternatif de cette solution qui se base sur le même principe sauf que les capteurs sont installés au plafond du parking. Mais les problèmes restent toujours présents.



*Figure 10 : DéTECTEUR DE PRÉSENCE*

### 3.3 Description de la problématique :

L'organisme *Innovatel Engineering Sarl* cherche à résoudre le problème des capteurs de présence qui se décharge rapidement, Ce qui donnent de faux résultat concernant le nombre de places libres et celles occupées, ainsi que la nécessité de la maintenance fréquente.

## **4. Cahier des charges**

Ce projet vise à créer une solution moins coûteuse, plus performante et surtout qui ne s'appuie pas sur un capteur de présence. Il a pour objectif :

- Détecter les emplacements vides et ceux occupés dans le parking.
- Proposer un spot de parking optimal pour les conducteurs.
- Avoir une base de données des clients visiteurs du parking.

## **5. Solution proposée et démarches à suivre**

Chaque parking contient des caméras de surveillance qui couvrent le parking. Sachant que pour surveiller les parkings, il est important d'installer des caméras haute définition, permettant de reconnaître les personnes qui commettent une infraction.

Les caméras doivent être anti-vandales, résistantes aux chocs, elles émettent une alarme en cas de choc. Des mini-dômes avec LED infra-rouges permettent de visualiser le jour, comme la nuit dans le parking.

Donc nous allons se baser sur cette infrastructure déjà installée sur le parking et nous allons traiter ces images en utilisant les techniques de vision par ordinateur, ces techniques nous permettront de superviser les voitures et aussi de suivre l'emplacement des voitures en temps réelle, cette solution est la solution la plus optimale et la moins couteuse, en se basant sur les caméras de surveillance déjà installées sur le site.



Chapitre II :

**Vision assistée par ordinateur**

**– COMPUTER VISION –**

## 1. Généralités sur la vision par ordinateur

La vision par ordinateur est un domaine de l'intelligence artificielle (IA) qui permet aux ordinateurs et aux systèmes de dériver des informations significatives à partir d'images numériques, de vidéos et d'autres entrées visuelles - et de prendre des mesures ou de faire des recommandations sur la base de ces informations. Si l'IA permet aux ordinateurs d'analyser, la vision par ordinateur leur permet de voir, d'observer et de comprendre.

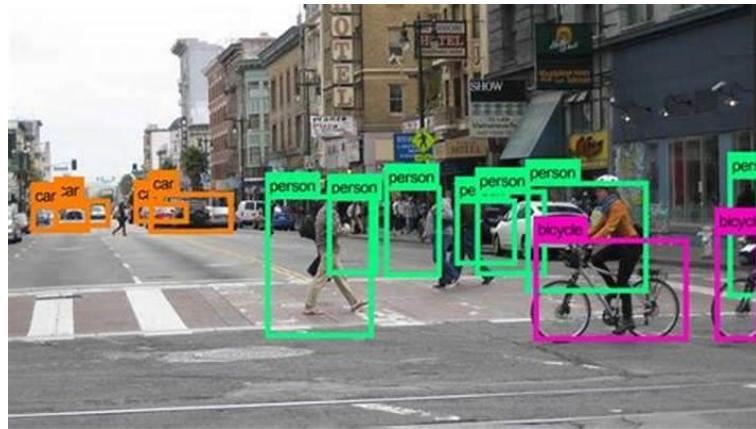


Figure 11 : Exemple de vision par ordinateur

La vision par ordinateur fonctionne à peu près comme la vision humaine, sauf que les humains ont une longueur de visualisation plus grande. La vision humaine a l'avantage de pouvoir s'entraîner, grâce à des vues entières de contexte, à distinguer les objets, à déterminer leur distance, à savoir s'ils sont en mouvement et s'ils excitent un problème dans une image.

La vision par ordinateur forme des machines à exécuter ces fonctions, mais elle doit le faire en beaucoup moins de temps avec des caméras, des données et des algorithmes plutôt qu'avec des rétines, des nerfs optiques et un cortex visuel. Comme un système formé à l'inspection de produits ou à la surveillance d'un actif de production peut analyser des milliers de produits ou de processus par minute, en remarquant des défauts ou des problèmes imperceptibles, il peut rapidement dépasser les capacités humaines.

La vision par ordinateur est utilisée dans des secteurs allant de l'énergie et des services publics à la fabrication et à l'automobile - et le marché continue à croître. Il devrait atteindre 48,6 milliards USD en 2022.

### 1.1 Fonctionnements

La vision par ordinateur a besoin de beaucoup de données. Elle effectue beaucoup d'analyses de données jusqu'à ce qu'elle discerne des distinctions et fini par reconnaître les images. Par exemple, pour entraîner un ordinateur à reconnaître des pneus de voiture, il faut lui

fournir plusieurs images de pneus et d'éléments liés aux pneus pour qu'il arrive à reconnaître un pneu et en particulier un pneu sans défaut.

Deux technologies essentielles sont utilisées à cette fin : un type d'apprentissage automatique appelé apprentissage profond et un réseau neuronal convolutif (CNN).

L'apprentissage automatique utilise des modèles algorithmiques qui permettent à un ordinateur d'apprendre à connaître le contexte des données visuelles. Si le modèle est alimenté par suffisamment de données, l'ordinateur "regarde" les données et apprend à distinguer une image d'une autre. Les algorithmes permettent à la machine d'apprendre par elle-même, plutôt que d'être programmée par quelqu'un pour reconnaître une image.

Un CNN aide un modèle d'apprentissage automatique ou d'apprentissage profond à "regarder" en décomposant les images en pixels auxquels sont attribuées des étiquettes ou des labels. Il utilise les étiquettes pour effectuer des convolutions (une opération mathématique sur deux fonctions pour produire une troisième fonction) et faire des prédictions sur ce qu'il "voit". Le réseau neuronal exécute des convolutions et vérifie l'exactitude de ses prédictions dans une série d'itérations jusqu'à ce que les prédictions commencent à se réaliser. Il reconnaît ou voit alors les images d'une manière similaire à celle des humains.

Tout comme un être humain qui distingue une image à distance, un CNN discerne d'abord les bords durs et les formes simples, puis complète les informations au fur et à mesure des itérations de ses prédictions. Un CNN est utilisé pour comprendre des images uniques. Un réseau neuronal récurrent (RNN) est utilisé de manière similaire pour les applications vidéo afin d'aider les ordinateurs à comprendre comment les images d'une série de cadres sont liées les unes aux autres.

## 1.2 Applications

De nombreuses recherches sont menées dans le domaine de la vision par ordinateur, mais il ne s'agit pas seulement de recherche. Les applications du monde réel démontrent l'importance de la vision par ordinateur pour les entreprises, les loisirs, les transports, les soins de santé et la vie quotidienne. L'un des principaux moteurs de la croissance de ces applications est le flot d'informations visuelles provenant des smartphones, des systèmes de sécurité, des caméras de circulation et d'autres dispositifs à instrumentation visuelle. Ces données pourraient jouer un rôle majeur dans les opérations de tous les secteurs, mais elles sont aujourd'hui inutilisées. Ces informations constituent un banc d'essai pour la formation des applications de vision par ordinateur et une rampe de lancement pour leur intégration dans toute une série

d'activités humaines :

- Des sociétés de recherche comme IBM ont utilisé la vision par ordinateur pour créer « My Moments » pour le tournoi de golf Masters 2018. IBM Watson a visionné des centaines d'heures de séquences du Masters et a pu identifier les vues (et les sons) des coups significatifs. Il a conservé ces moments clés et les a livrés aux fans sous forme de bobines d'extraits personnalisés.
- Google Translate permet aux utilisateurs de pointer la caméra d'un smartphone sur un panneau dans une autre langue et d'obtenir presque immédiatement une traduction du panneau dans leur langue préférée.

Le développement des véhicules à conduite autonome repose sur la vision par ordinateur, qui permet de donner un sens aux données visuelles fournies par les caméras et autres capteurs de la voiture. Elle est essentielle pour identifier les autres voitures, les panneaux de signalisation, les marqueurs de voie, les piétons, les vélos et toutes les autres informations visuelles rencontrées sur la route.

IBM applique la technologie de vision par ordinateur avec des partenaires comme Verizon afin d'amener l'intelligente à la périphérie et d'aider les constructeurs automobiles à identifier les défauts de qualité avant qu'un véhicule ne quitte l'usine.

On distingue donc que le domaine de la vision par ordinateur est un domaine très vaste qui comprend :

- **Classification d'images** : voir une image et pouvoir la classer (un chien, une pomme, un visage d'une personne). Plus précisément, elle est capable de prédire avec précision qu'une image donnée appartient à une certaine classe. Par exemple, une société de médias sociaux pourrait vouloir l'utiliser pour identifier et séparer automatiquement les images répréhensibles téléchargées par les utilisateurs.

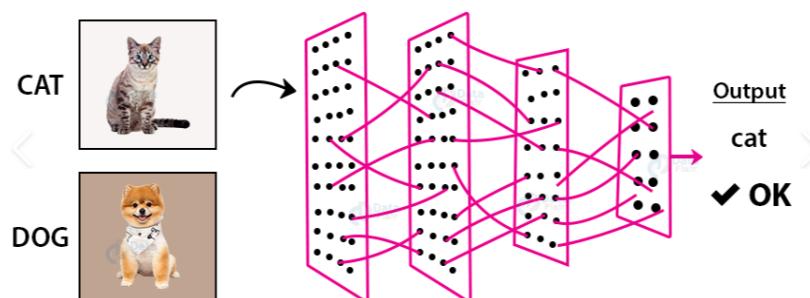


Figure 12: Exemple de classification

- **Détection des objets** : pouvoir de l'utilisation de la classification d'images pour identifier une certaine classe d'images, puis la détection et la classification de leur

apparition dans une image ou une vidéo. Il s'agit par exemple de détecter les dommages sur une chaîne de montage ou d'identifier les machines qui nécessitent un entretien.

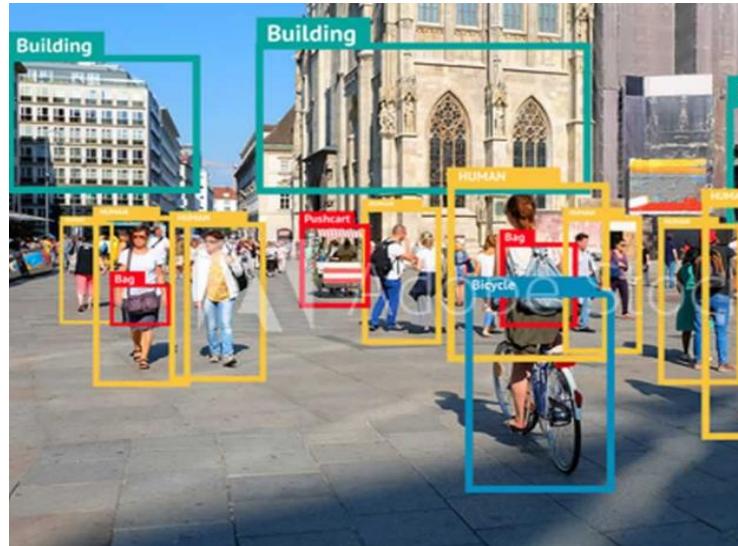


Figure 13 : Exemple de détection des objets

- **Suivi d'objet :** suivi d'un objet une fois qu'il est détecté. Cette tâche est souvent exécutée avec des images capturées en séquence ou des flux vidéo en temps réel. Les véhicules autonomes, par exemple, doivent non seulement classer et détecter les objets tels que les piétons, les autres voitures et les infrastructures routières, mais aussi les suivre en mouvement pour éviter les collisions et respecter le code de la route.

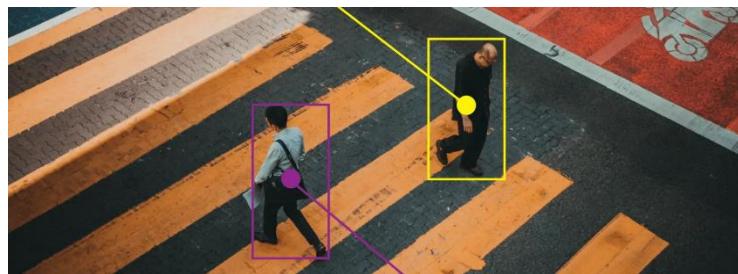


Figure 14: Exemple de suivi d'objet

- **Recherche d'images :** basée sur le contenu utilise la vision par ordinateur pour parcourir, rechercher et récupérer des images dans de grandes magasins de données, en se basant sur le contenu des images plutôt que sur les balises de métadonnées qui leur sont associées. Cette tâche peut intégrer l'annotation automatique des images qui remplace le marquage manuel des images. Ces tâches peuvent être utilisées pour les systèmes de gestion des actifs numériques et peuvent augmenter la précision de la recherche et de l'extraction.

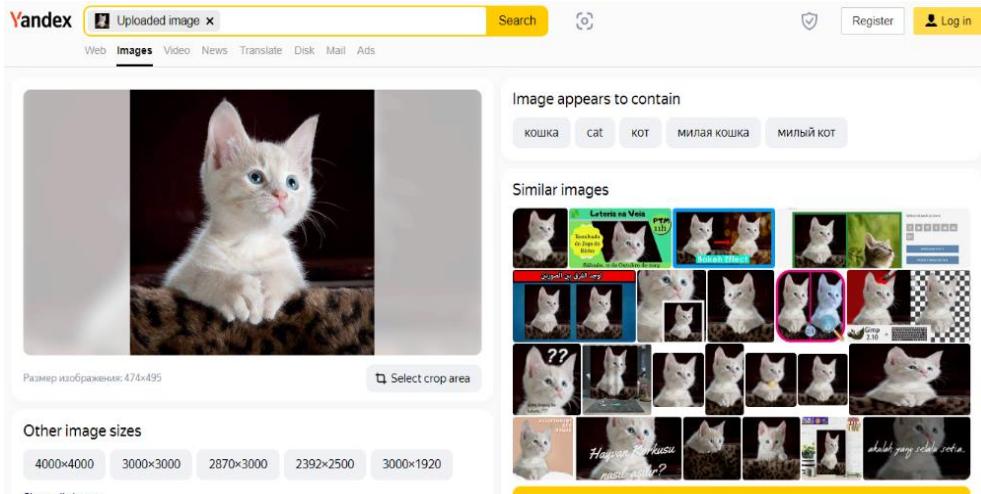


Figure 15 : Exemple de recherche d'image

## 2. Algorithme de détection des objets « YOLO V4 » :

L’objectif principal de ce projet est d’identifier si les emplacements de parking des voitures sont vides ou non, autrement dit est ce que dans un emplacement existe-t-elle une voiture ou non donc identifier la voiture en premier lieu. Pour cela on va utiliser l’architecture de YOLO V4 [*You Only Look Once Version 4*].

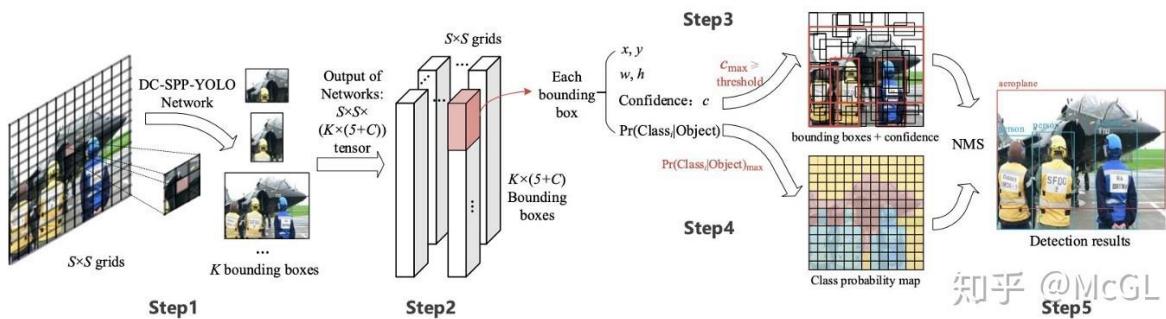


Figure 16: architecture de YOLO - v4

C’est une architecture pré-entraînée générale qui peut détecter plus de 80 classes comprises les voitures, les personnes, les vélos, et d’autres.

## 3. Les outils utilisés :

Avant de commencer le développement de notre projet, on doit faire le choix de la technologie qu’on va utiliser pour la programmation. C’est à dire choisir le langage de programmation et le Framework ou la bibliothèque de traitement.

Nous avons choisi Python comme langage de programmation, car lorsqu'il s'agit d'IA ou la vision par ordinateur, il existe de nombreux outils possibles qu'on peut opter pour (et par "outils", nous entendons le langage de programmation). Java, C#, LISP, Python, et d'autres. Cependant, l'un d'entre eux reste toujours le plus populaire: Python.



Figure 17 : logo python

Python est un langage de programmation interprété de haut niveau, basé sur la POO. Sa première version date de 29 ans. Le langage est principalement axé sur le RAD (Rapid Application Development) et le DRY (Don't Repeat Yourself), qui vise à mettre fin aux répétitions des modèles logiciels. Python est également loué pour sa capacité à connecter les composants existants. Ce phénomène est appelé Glue Language. L'adaptabilité, l'évolutivité et la facilité d'apprentissage sont les principales raisons pour lesquelles il développe une base d'utilisateurs plus importante. Les bibliothèques que possède Python sont l'une des raisons pour lesquelles il peut être utilisé pour les applications web, la science des données et même l'IA.

Les bibliothèques de Python sont également une autre raison essentielle de sa popularité parmi les programmeurs et les développeurs d'IA. Une bibliothèque est un groupe de modules ou un module unique compilé par différentes sources comme PyPi et comprend également un morceau de code pré-écrit qui permet aux utilisateurs d'atteindre certaines fonctionnalités ou d'effectuer différentes actions.

L'apprentissage automatique, qui est un domaine impératif de l'IA, nécessite un traitement continu des données. Les bibliothèques Python vous permettent ici d'accéder aux données et de les transformer. Voici quelques bibliothèques réputées pour l'IA :

- **Scikit-Learn** : Utilisé principalement pour gérer les algorithmes de base, tels que les régressions linéaires et logistiques, les régressions, les classifications, le clustering, et autres.
- **Keras**: Utilisée pour l'apprentissage profond
- **Scikit-image**: Utilisé pour le traitement d'images
- **PyBrain** : utilisé pour les réseaux neuronaux, l'apprentissage non supervisé et renforcé
- **StatsModel**: Utilisé pour l'exploration des données et les algorithmes statistiques.
- **Pandas** : Utilisé pour les structures de données de haut niveau et leur analyse.
- **Dépôt PyPI** : utilisé pour découvrir et comparer d'autres bibliothèques Python.

Le computer vision ou la vision par ordinateur nécessite une couche de base de traitement d'image et pour cette couche nous utiliserons OpenCV.

OpenCV ou Open Source Computer Vision a été construit pour fournir une infrastructure commune pour les applications de vision par ordinateur et pour accélérer l'utilisation de la perception artificielle dans les produits commerciaux. Étant un produit sous licence BSD, OpenCV permet aux entreprises d'utiliser et de modifier facilement le code.

OpenCV est une bibliothèque de fonctions de programmation principalement destinée à la vision par ordinateur en temps réel. En langage simple, c'est une bibliothèque utilisée pour le traitement des images. Elle est principalement utilisée pour effectuer toutes les opérations liées aux images.

Ce logiciel permet de :

- Lire et écrire des images.
- Déetecter les visages et de leurs caractéristiques.
- Déetecter les formes comme le cercle, le rectangle.
- Reconnaissance du texte dans les images (lecture de plaques d'immatriculation).
- Modifier la qualité et les couleurs de l'image.
- Développer des applications de réalité augmentée.

Il supporte des langages de programmation comme :

- C++
- Android SDK
- Java
- Python
- C (non recommandé)

Et parmi les avantages d'utilisation en trouvent :

- Sa simplicité à apprendre en plus, nombreux tutoriels sont disponibles.
- Son fonctionnement avec les différents langages de programmation connus.
- Son utilisation gratuite.



Figure 18 : Logo OpenCV

Pour exécuter le code et le debugger on aura besoin d'un environnement de développement on a choisi Visual Studio Code. C'est un éditeur de code extensible développé par Microsoft pour Windows, Linux et MacOs. Les fonctionnalités incluent la prise en charge du débogage, la mise en évidence de la syntaxe, la complétion intelligente du code, les snippets, la refactorisation du code et Git intégrer. Les utilisateurs peuvent modifier le thème, les raccourcis clavier, les préférences et installer des extensions qui ajoutent des fonctionnalités supplémentaires.



Figure 19 : Logo Visual studio code

Chapitre III :  
Réalisation d'un prototype

## 1. Implémentation de la solution

Comme tout projet nous commençons par une démonstration de faisabilité (poc : proof of concept), dans cette partie nous allons travailler en temps différer c'est-à-dire au lieu de prendre le flux vidéo des caméras directement et faire le traitement, nous travaillerons sur des échantillons vidéo déjà enregistrés avec les caméras de surveillance du parking.



Figure 20: Les flux vidéo collectés

Pour travailler avec YOLO-V4 le model pré-entraîner, il faut télécharger les fichiers suivants à partir du projet open source sur GitHub nommé darkNet . Nous téléchargerons 3 fichiers.

| Nom            | Type                 | Taille     | Modifié le       |
|----------------|----------------------|------------|------------------|
| classes.txt    | Document texte       | 1 Ko       | 01/02/2021 14:21 |
| yolov4.cfg     | Fichier source Co... | 12 Ko      | 10/06/2020 21:48 |
| yolov4.weights | Fichier WEIGHTS      | 251 678 Ko | 30/04/2021 17:37 |

Figure 21: Les fichiers nécessaires pour travailler avec YOLO v4

Le fichier classes.txt contient les noms des classes que le modèle peut détecter. Le deuxième fichier yolov4.cfg contient la configuration des couches du réseau de neurone utilisé. Le troisième fichier contient les poids du réseau de neurone utilisé.

Après téléchargement des fichiers nécessaires et enregistrement des flux vidéo on installe les bibliothèque nécessaire NumPy, OpenCv et Matplotlib.

```

File Edit Selection View Go Run Terminal Help
creation de video.py - smart parking - Visual Studio Code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
Copyright (C) Microsoft Corporation. Tous droits réservés.

PS C:\smart parking> pip install numpy
Collecting numpy
  Downloading numpy-1.23.0-cp310-cp310-win_amd64.whl (14.6 MB)
----- 14.6/14.6 MB 7.4 MB/s eta 0:00:00
Installing collected packages: numpy
Successfully installed numpy-1.23.0
WARNING: You are using pip version 22.0.4; however, version 22.1.2 is available.
You should consider upgrading via the 'C:\Users\ENSEM\AppData\Local\Programs\Python\Python310\python.exe -m pip install --upgrade pip' command.
PS C:\smart parking> pip install opencv-python
Collecting opencv-python
  Using cached opencv_python-4.6.0.66-cp36-abi3-win_amd64.whl (35.6 MB)
Requirement already satisfied: numpy>=1.21.2 in c:\users\ensem\appdata\local\programs\python\python310\lib\site-packages (from opencv-python) (1.23.0)
Installing collected packages: opencv-python
Successfully installed opencv-python-4.6.0.66

```

Figure 22 : installation des bibliothèques sur la terminale de V.S code

Ensuite, on commence les étapes de développement :

- La première étape c'est d'identifier les voitures dans un flux vidéo, donc on appliquera YOLO V4 sur une vidéo :

```

data.csv object_detection.py Preview 'data.csv'
object_detection.py > ObjectDetection
1 import cv2
2 import numpy as np
3
4 class ObjectDetection:
5     # constructeur de la classe de la detection des objets
6     def __init__(self, weights_path="dnn_model/yolov4.weights", cfg_path="dnn_model/yolov4.cfg"):
7         print("Loading Object Detection")
8         print("Running opencv dnn with YOLOv4")
9         self.nmsThreshold = 0.4
10        self.confThreshold = 0.5
11        self.image_size = 608
12
13        # charger le reseaux de neurone et ces poids
14        net = cv2.dnn.readNet(weights_path, cfg_path)
15
16        # activer l'acceleration de GPU
17        net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
18        net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)

```

Figure 23: Une partie du code de traking sur V.S code

L'organigramme de détection des voitures est le suivant :

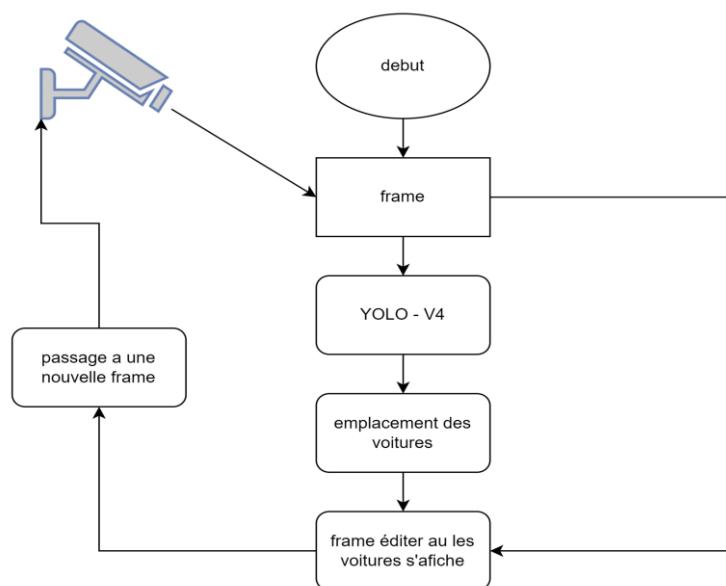


Figure 24 : Organigramme du détection des voitures

Nous testons ce code sur une série de trames vidéos et voici les résultats qu'on obtient :



Figure 25: un échantillon de la détection

L'algorithme a pu identifier toutes les voitures présentes sur la trame. Maintenant on crée un algorithme capable de suivre les véhicules dans deux frames successifs

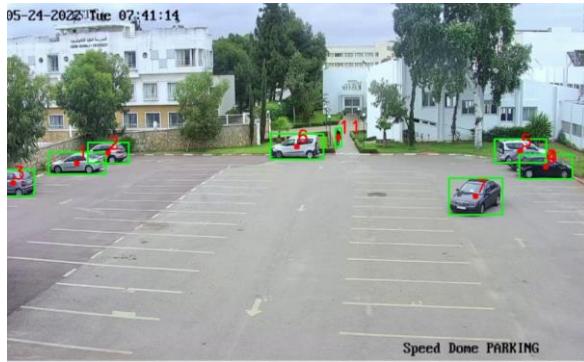


Figure 26 : Frame 235 d'une vidéo échantillonnée

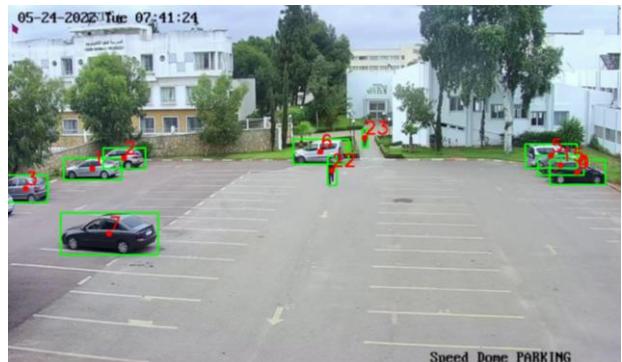


Figure 27 : Frame 595 d'une vidéo échantillonnée

Comme on voit ici l'algorithme est capable de suivre les voitures avec une très grande précision, pour arriver à ce résultat on ajoute une partie de suivi (tracking) dans l'organigramme suivant :

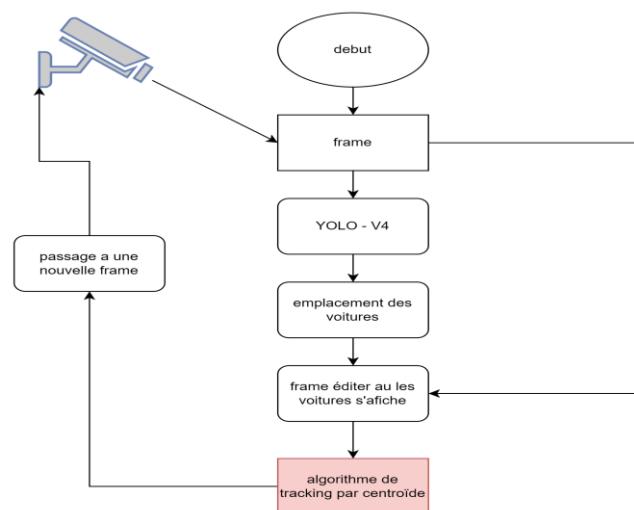


Figure 26 : l'algorithme mise en place pour le Traking

La méthode utilisée pour le suivi est résumée dans le bloc en rouge, elle s'agit de la méthode de suivi par centroïde l'algorithme fonctionne comme suit :

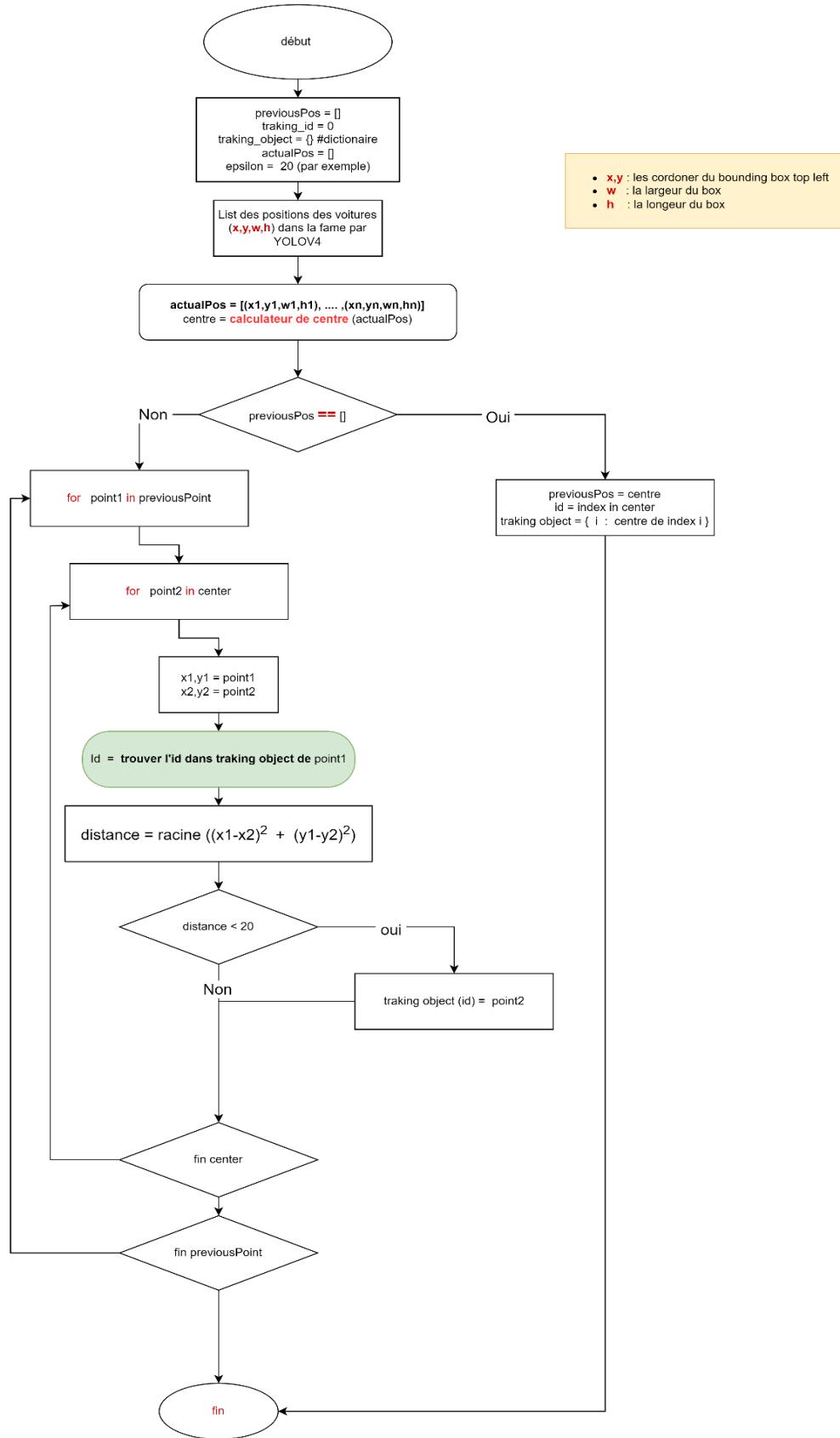


Figure 27: l'algorithme de Traking

```

(class_ids, scores, boxes) = od.detect(frame)
for box in boxes:
    (x, y, w, h) = box
    cx = int((x + x + w) / 2)
    cy = int((y + y + h) / 2)
    center_points_cur_frame.append((cx, cy))
    #print("FRAME N°", count, " ", x, y, w, h)

    # cv2.circle(frame, (cx, cy), 5, (0, 0, 255), -1)
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

# Only at the beginning we compare previous and current frame
if count <= 2:
    for pt in center_points_cur_frame:
        for pt2 in center_points_prev_frame:
            distance = math.hypot(pt2[0] - pt[0], pt2[1] - pt[1])

            if distance < 20:
                tracking_objects[track_id] = pt
                track_id += 1
else:
    tracking_objects_copy = tracking_objects.copy()
    center_points_cur_frame_copy = center_points_cur_frame.copy()

```

Figure 28 : Une partie du code du suivi

|    | A  | B  | C  | D             |
|----|----|----|----|---------------|
| 1  |    |    |    |               |
| 2  | 1  | 2  | 2  | 0 (1182, 276) |
| 3  | 2  | 3  | 3  | 0 (1181, 275) |
| 4  | 3  | 4  | 4  | 0 (1179, 274) |
| 5  | 4  | 5  | 5  | 0 (1177, 271) |
| 6  | 5  | 6  | 6  | 0 (1176, 270) |
| 7  | 6  | 7  | 7  | 0 (1174, 269) |
| 8  | 7  | 8  | 8  | 0 (1169, 267) |
| 9  | 8  | 9  | 9  | 0 (1163, 267) |
| 10 | 9  | 10 | 10 | 0 (1160, 266) |
| 11 | 10 | 11 | 11 | 0 (1156, 264) |
| 12 | 11 | 12 | 12 | 0 (1151, 263) |

Figure 29 : data enregistrer sous format CSV

Maintenant on peut suivre les voitures, l'étape suivante est d'enregistrer ces positions dans une data set avec le temps d'identification de la voiture et les positions du centroïde en coordonnées pixel dans un fichier csv (comma separated values).

```

# do perspective transformation setting area outside input to black
imgOutput = cv2.warpPerspective(img,
                                 matrix,
                                 (width,height),
                                 cv2.INTER_LINEAR,
                                 borderMode=cv2.BORDER_CONSTANT,
                                 borderValue=(0,0,0)
                                )
print(imgOutput.shape)

# save the warped output
cv2.imwrite("sudoku_warped.jpg", imgOutput)

```

Figure 30 : Code pour annulation de la perspective



Figure 31 : frame avec annulation d'effet de perspective

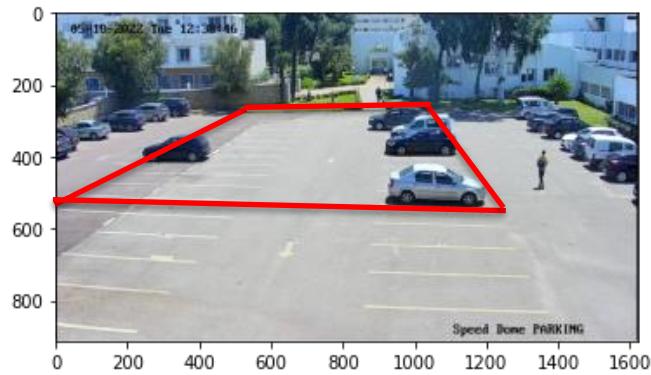


Figure 32 : frame avec effet de perspective du parking de l'ENSEM

Il existe deux problèmes majeurs dans l'utilisation de cette méthode :

- Le premier est lié à la méthode avec laquelle la fonction a été écrite car si on veut la transformation d'un point exacte on ne peut pas l'obtenir. Puisque le retour de cette fonction est une nouvelle matrice transformée avec de nouveaux adresses mémoire.
- Le deuxième problème vient du model de YOLO v4, cette transformation qu'on applique sur l'image transforme la forme des voitures et donc le model sera incapable de détecter avec une très grande efficacité la position de la voiture

Maintenant on résoudra ce problème en suivant les étapes suivantes :

- Avoir les corner du parking ou des zones d'intérêt pour chaque camera.
- Faire la transformation géométrique
- Déclarer les zones du parking/

D'abord la calibration et l'annulation des effets de perspectives. Dans ce sens on a créé un programme qui a pour objectif de détecter les points qui représentent les régions d'intérêt.

```

C:\> Users > AHBAR ABDELLAH > Documents > Project celec > exposition > park detect
 1 import cv2
 2 import numpy as np
 3 import matplotlib.pyplot as plt
 4
 5
 6
 7 posList = []
 8
 9
10 def mouseClick(events, x, y, flags, params):
11     if events == cv2.EVENT_LBUTTONDOWN:
12         posList.append((x, y))
13
14
15
16 while True:
17     img = cv2.imread('frame.png')
18     for pos in posList:
19         cv2.circle(img, pos, 5, (0, 0, 255), -1)
20
21     cv2.imshow("Image", img)
22     cv2.setMouseCallback("Image", mouseClick)
23     print(posList)
24     cv2.waitKey(1)
25
26

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```

[[268, 56], (492, 58), (232, 379), (566, 391)]
[(268, 56), (492, 58), (232, 379), (566, 391)]
[(268, 56), (492, 58), (232, 379), (566, 391)]
[(268, 56), (492, 58), (232, 379), (566, 391)]
[(268, 56), (492, 58), (232, 379), (566, 391)]
[(268, 56), (492, 58), (232, 379), (566, 391)]
[(268, 56), (492, 58), (232, 379), (566, 391)]
[(268, 56), (492, 58), (232, 379), (566, 391)]
[(268, 56), (492, 58), (232, 379), (566, 391)]
[(268, 56), (492, 58), (232, 379), (566, 391)]

```

Figure 33 : le programme responsable de la restitution des points du corners du parking

- Donc un fichier de calibrage est créé pour annuler ces effets de la caméra. Ses entrées seront les points donnés par le premier programme, et l'image ou le flux vidéo, ce qu'est représenté au-dessous :



Figure 34: vue après annulation de la perspective

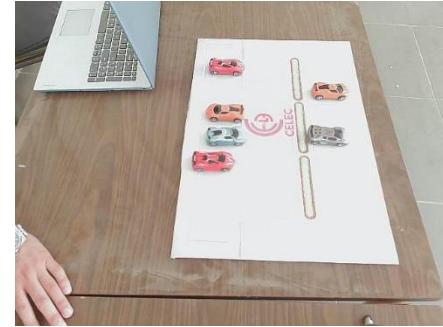


Figure 35 : vue directe du camera

### REMARQUE :

La différence des couleurs est due à la différence entre les couleurs de la caméra et les couleurs obtenu par open-CV RGB et BGR.

D'autres par il faut sélectionner les spots du parking au les voitures Park, ce travaille peut mieux être sélectionnées manuellement car il ne s'effectue qu'une seule fois. Donc on crée un programme « parkingSpacePicker.py » qui peut faire ceci :

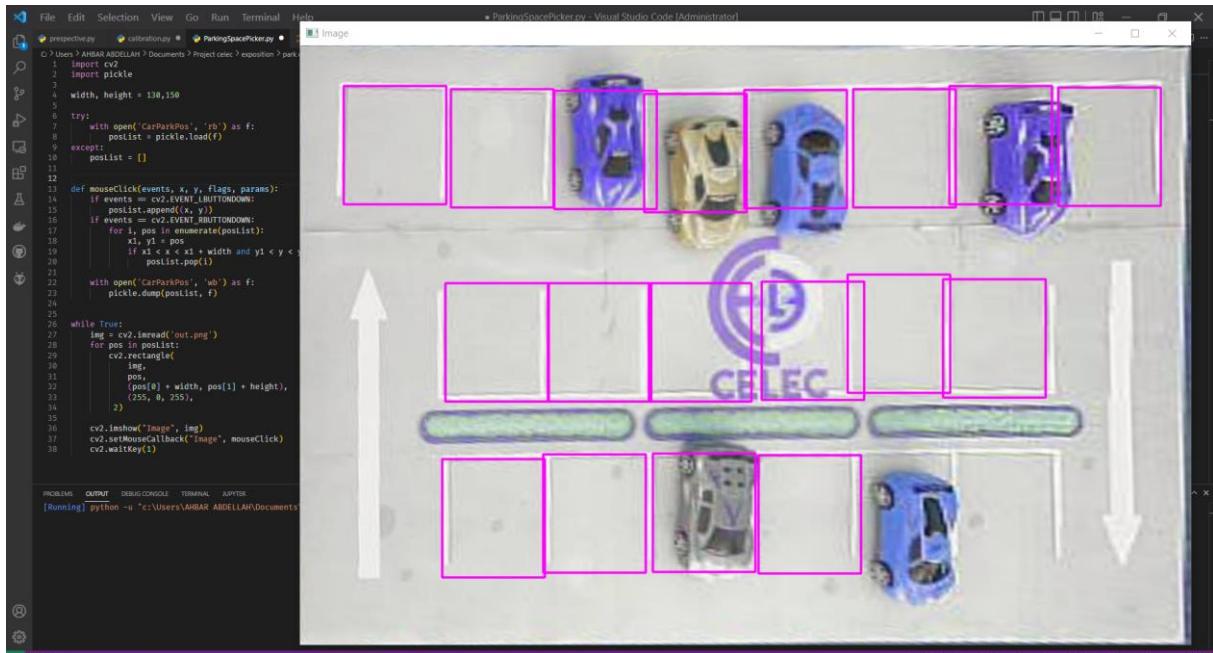


Figure 36 : le programme parkingSpacePicker.py qui permet de sélectionnée les spots de parking

- En fin on va rassembler ces unités dans un seul programme, et voici les résultats.

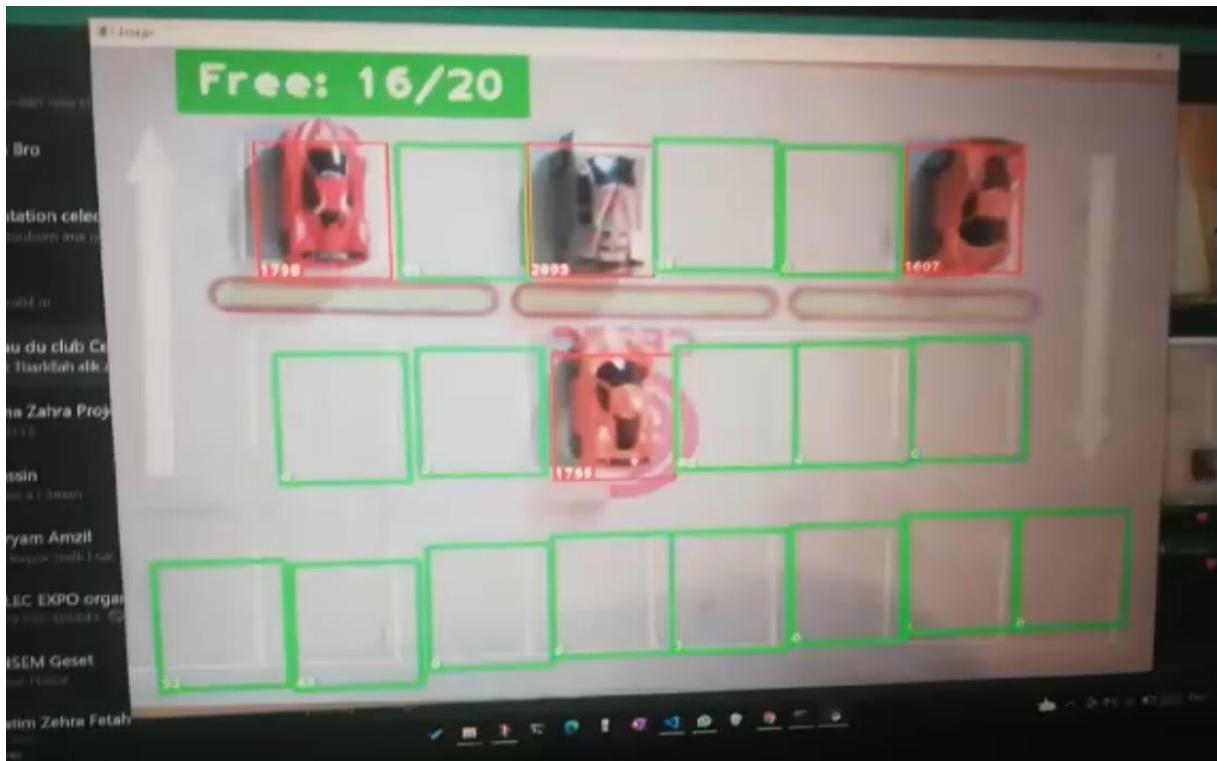


Figure 37: prototype du parking avec un flux de camera en temps réel

Des améliorations peuvent être porté à ce travaille comme l'usage des régions d'intérêts (RIO) au lieu de l'usage de toute l'image.

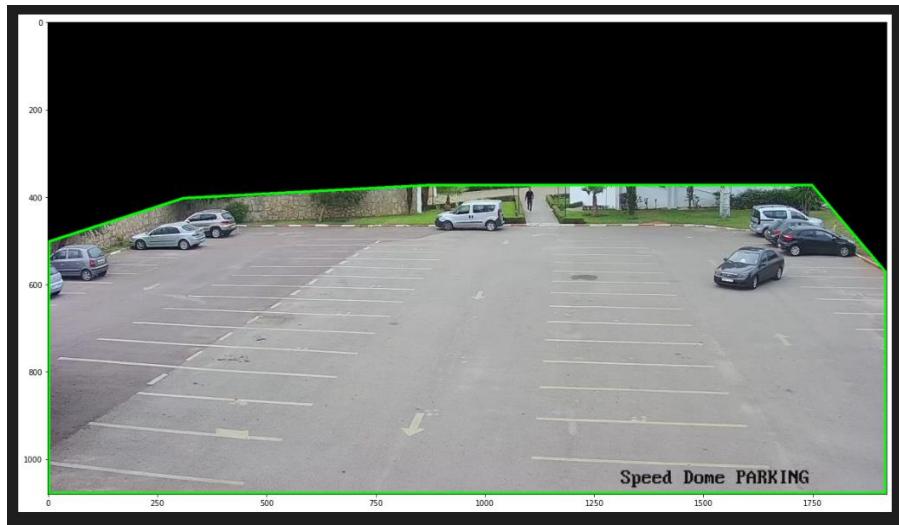


Figure 38 : RIO dans le parking de l'ENSEM

## II.1 Test des résultats :

Dans cette partie on va présenter les résultats des tests qu'on a fait et les remarques à améliorer dans la version prochaine :

### 1- La détection des Object (véhicules) :

L'algorithme de YOLO-V4 doit être appliqué avec les camera de très grandes définitions surtout en termes de mémoire et de temps d'exécution. Pour vous donner une image on propose cette série de test qu'on a fait :

*Tableau 1 : série de test sur YOLO V4*

| Définition | Temps d'exécution | Précession |
|------------|-------------------|------------|
| 144p       | 0.2s              | 30.4%      |
| 240p       | 0.5s              | 65.6%      |
| 720p       | 1s                | 81%        |
| 1080p HD   | 1.5s              | 99.8%      |
| 2040p 2K   | 3s                | 100%       |

Ces tests représentés ci-dessus sont faite par les mêmes frames d'une vidéo et la même configuration de l'ordinateur utilisée (processeur Intel core i7 de la 7eme génération et de 16 GB de mémoire RAM).

Les résultats de ces tests représentent un problème réel d'où la nécessité de passer a une architecture plus rapide, soit en utilisant un modèle qui offre plus de rapidité ou bien créé le nôtre en entraînant un modèle de réseaux de neurone de convolution.

On va représenter ici les étapes nécessaires pour créer notre modèle de détection des véhicules :

- Collecte des images labéliser : il faut collecter des photos et les enregistrer en deux groupes, le premier sera dédié aux images représentant les véhicules et le deuxième dédié aux images négatives.
- La deuxième étape sera spécifiée à la création du modèle : le modèle qu'on va utiliser est un réseau neurone de convolution (CNN), nous utiliserons une stratégie de Transfer Learning où on va ajouter quelques couches de neurones à un modèle généralisé déjà entraîné.
- La troisième étape consiste à tester les performances du modèle, la complexité temporelle, spatiale et le temps d'exécution.

Sinon on utilisera l'architecture YOLO-V6-nano c'est une architecture de la série YOLO, récemment introduite à l'utilisation public dès le début de Juillet 2022, en utilisant le jeu de données MS COCO (Microsoft Common Objects in Context) il a été démontré que YOLO-v6n est plus performant que ces antécédents.



Figure 39: YOLO-v6n vs YOLO-v5n

## 2- Utilisation des régions d'Intérêt ROI

Des améliorations peuvent être portées à ce travail comme l'usage des régions d'intérêts (RIO) au lieu de l'usage de toute l'image.

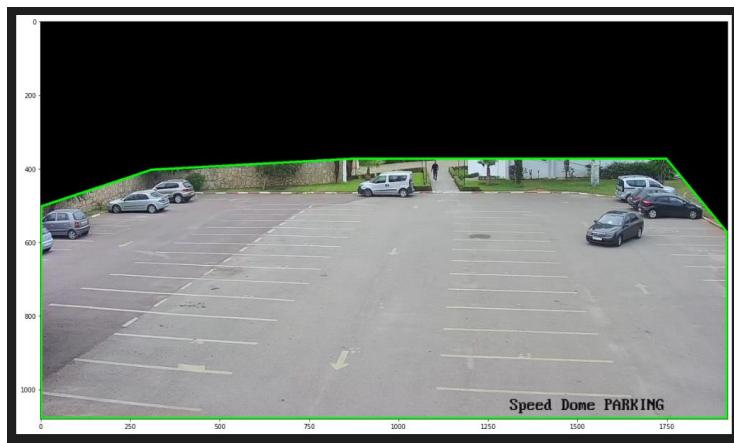
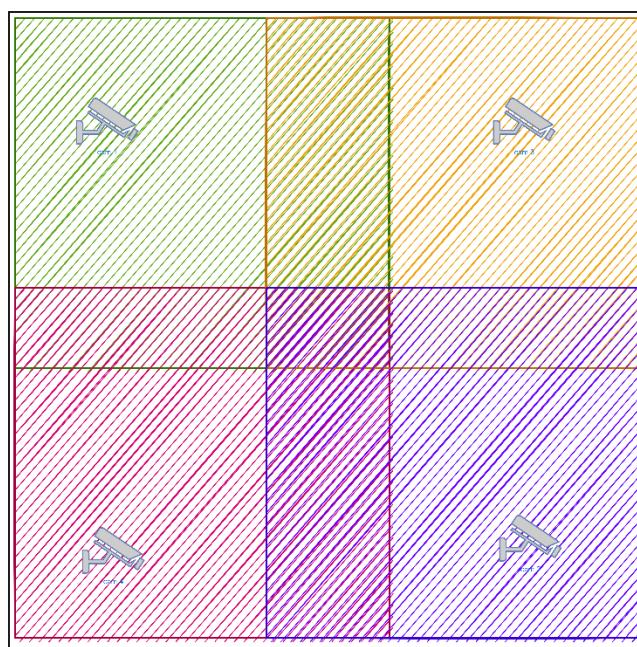


Figure 40 : RIO dans le parking de l'ENSEM

### **3- Multiplexage des caméras :**

Dans tout ce qui précède nous avons utilisé qu'une seule caméra, mais en réalité une caméra ne peut jamais superviser la totalité d'un parking ce qui fait on aura besoin de plusieurs caméras pour la supervision.

La question qui se pose : Comment peut-on faire la gestion de l'ensemble de ces caméras ?



*Figure 41 : cas réel de la supervision des caméras*

**Remarque :** Il est nécessaire de ne pas avoir des points aveugle. Pour avoir une supervision générale de la totalité du parking.

On propose les trois méthodes suivantes :

### Générer une photo globale à partir des photos obtenues par chaque camera

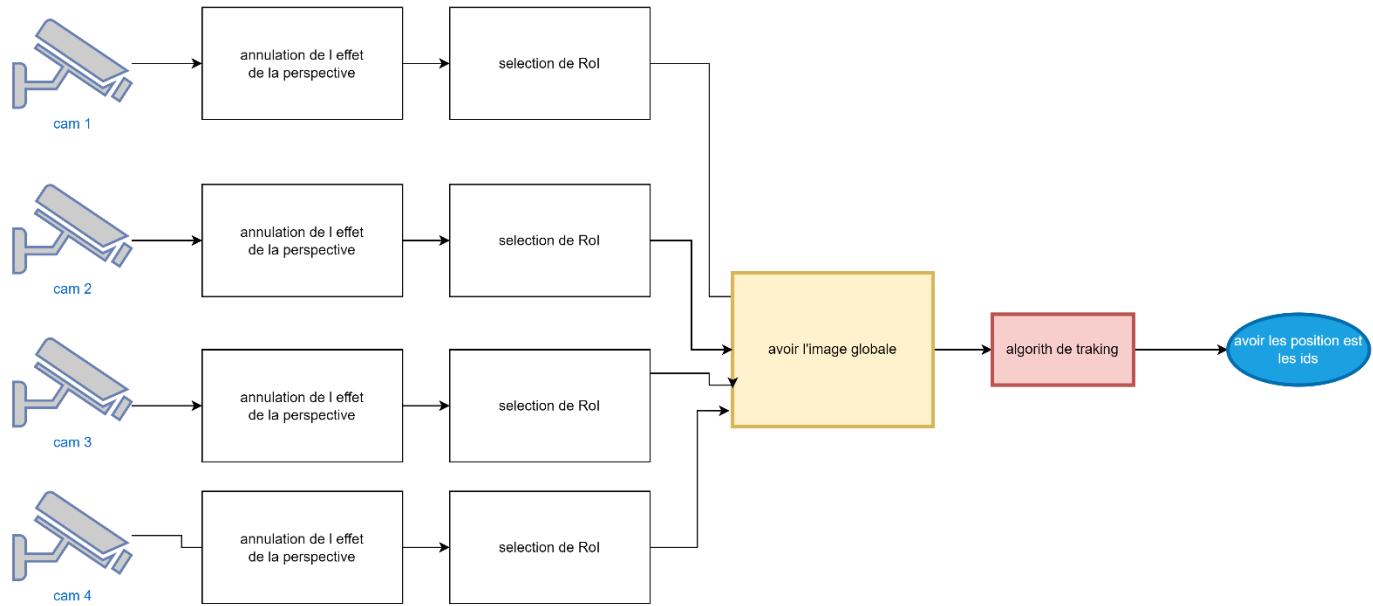


Figure 42 : méthode de la génération d'une grande image

Même la représentation ici peut apparaître parallèle mais en effet c'est séquentiel, chaque camera donne c'est résultat après la fin du traitement de l'autres.

Le problème de cette méthode que la l'algorithme de détection YOLO a une complexité qui dépend de la taille de l'image.

**Comptions parallèle :** chaque caméra s'occupe d'une région indépendamment et à la fin les images seront envoyées à un système d'affichage,

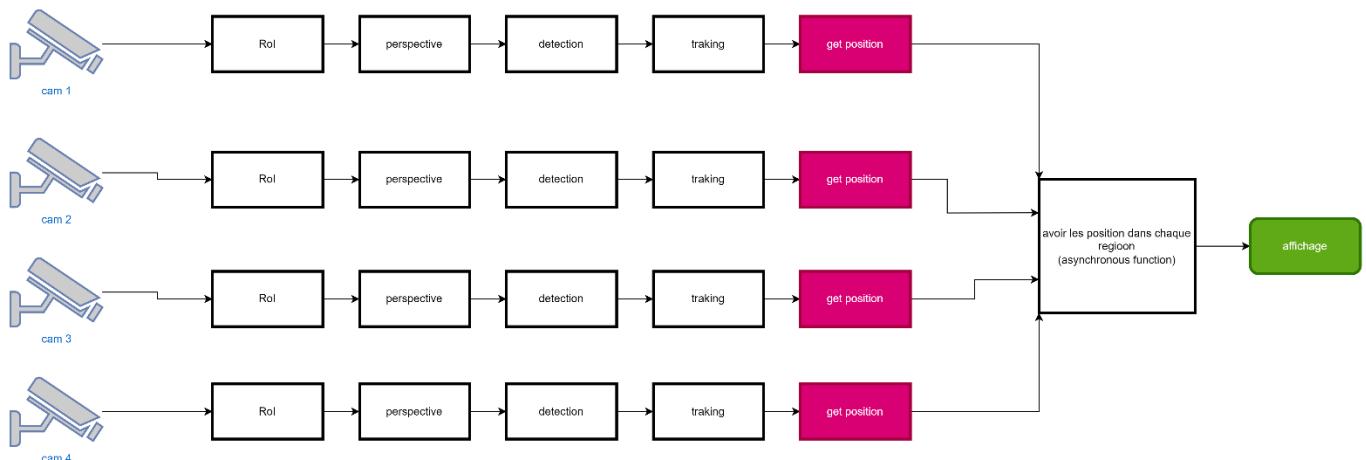


Figure 43 : parallèle computing

Chaque Core (cœur) de processeur doit s'occuper d'une région (ou d'une caméra). Cette méthode nécessite l'utilisation des serveurs qui peuvent supporter le multithreading (parallelisme)

### Multiplexage temporel :

Cette solution se base sur le principe que chaque instant, le système lit une image à partir d'une seule caméra. Après on passe à la région suivante puis on lit l'image mais de l'instant prochain et ainsi de suite...

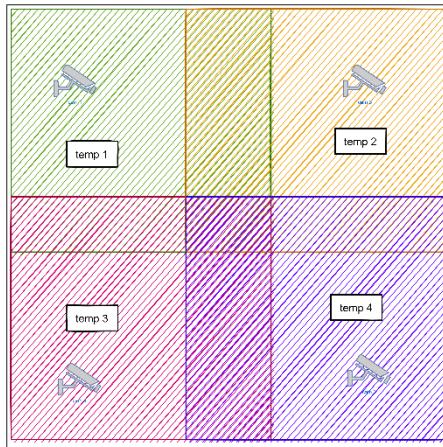


Figure 44 : multiplexage temporelle

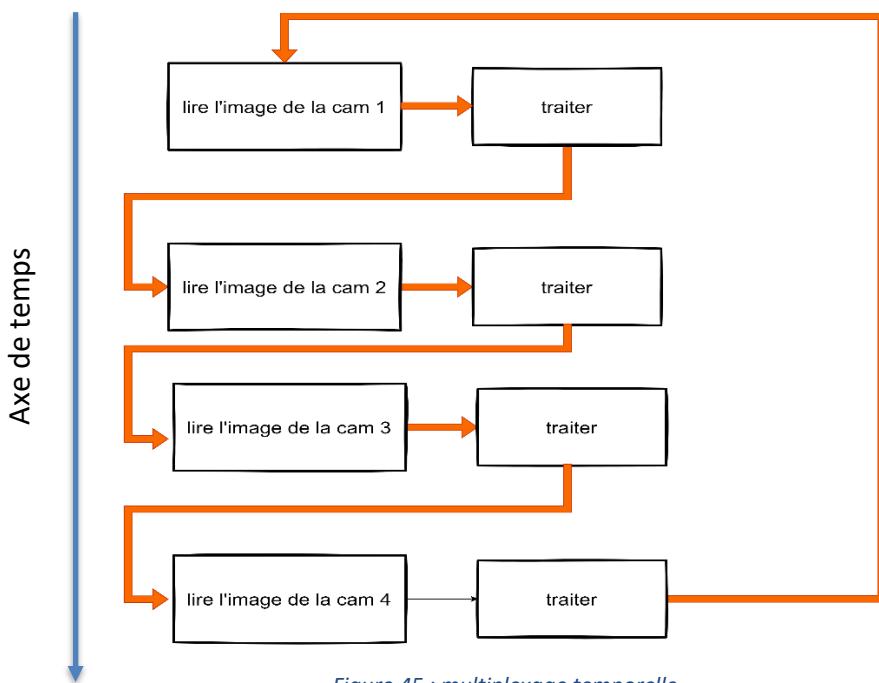


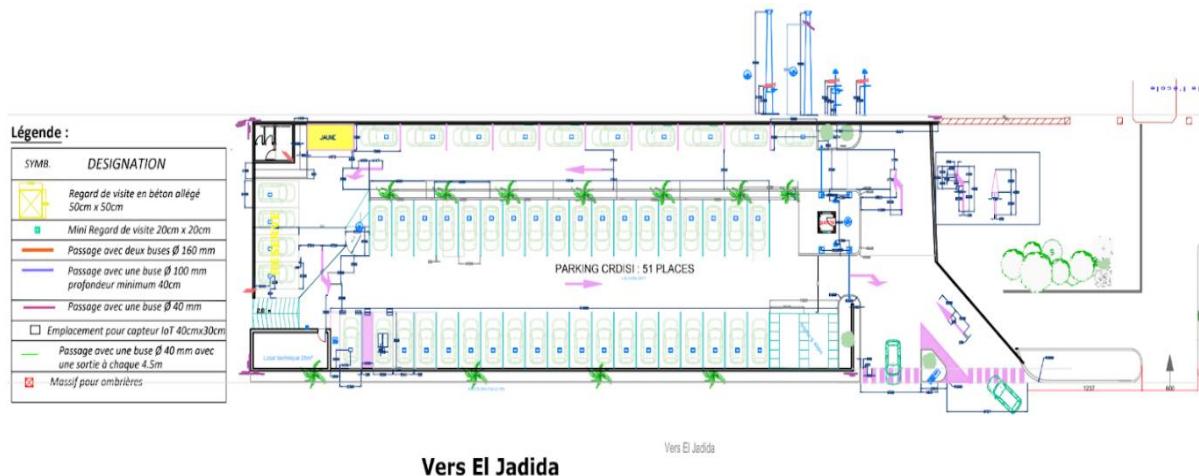
Figure 45 : multiplexage temporelle

## La carte virtuelle et réalité augmenter :

Le système maintenant est capable de nous donner les positions de chaque voiture après élimination des effets de perspective.

On transforme ces coordonnées pixels en coordonnées réelles en liant des points spatiaux à ceux données par la carte.

Ces points doivent être fixes dans la période de calibrage.



Voici l'exemple de la carte qu'on a pu créer (les valeurs sont données manuellement) :

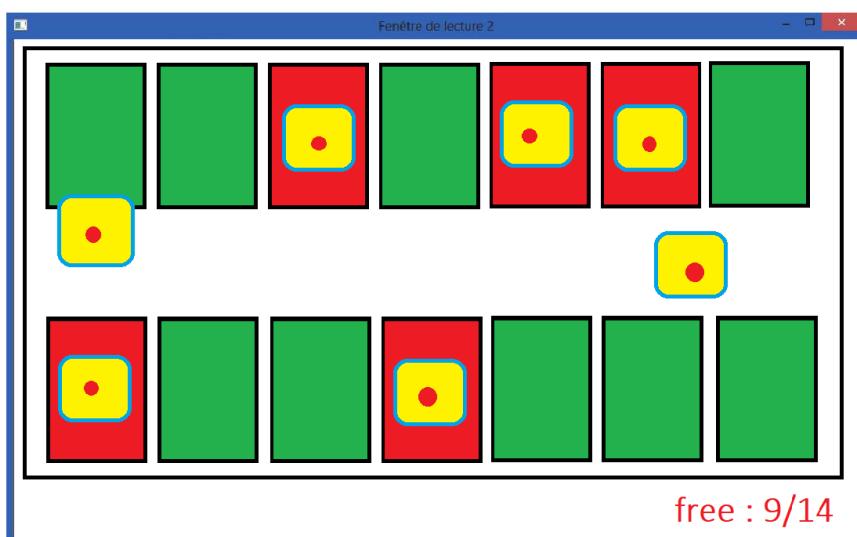


Figure 46 : carte OpenCv

On peut aussi se baser sur la réalité, On peut augmenter en ajoutant des illustrations qui montre que les états des places :

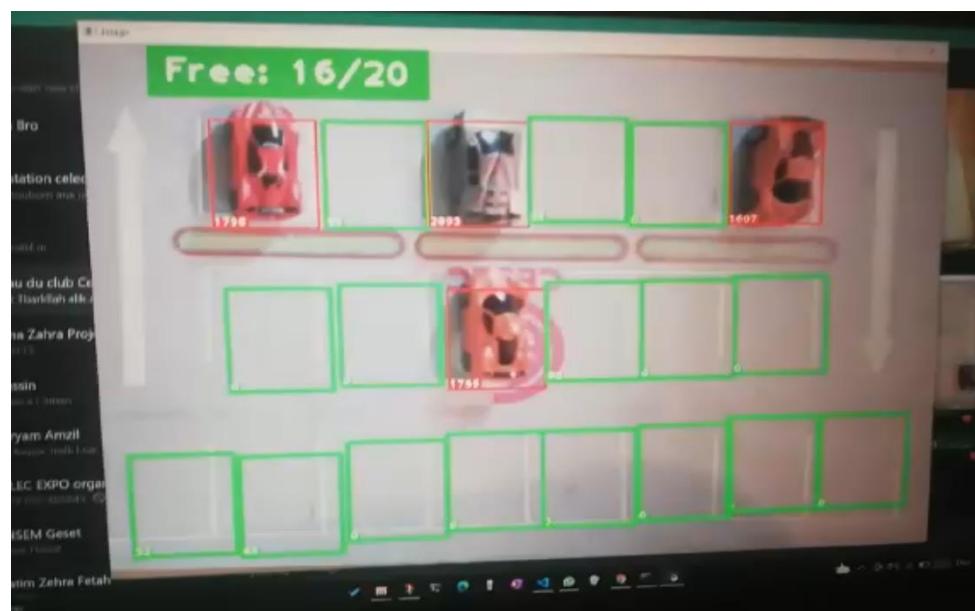


Figure 47 : A.R parking

Chapitre IV :  
Implémentation dans un  
parking - cas réelle -

### Objectives :

L'objectif de cette partie c'est d'implémenter ce qu'on a fait dans la partie précédente dans un parking réel il s'agit du parking de l'CITT. Et pour commencer il faut tout d'abord accéder aux cameras du parking. Le parking est couvert par plusieurs cameras, l'objectif est de lire les différentes cameras, puis on exploite les différentes techniques de multiplexage afin de faire les traitements nécessaires. Puis avoir les informations sur une carte.

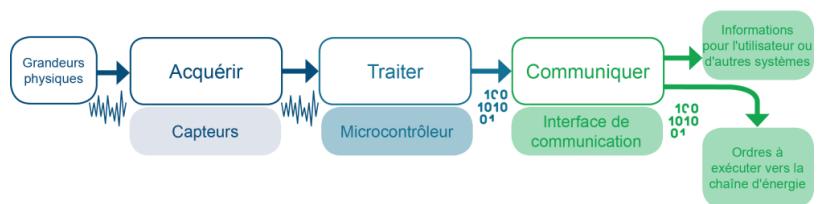


Figure 48: chaîne d'information dans tout système

#### A. Implémentation de la détection.

Dans cette partie, on se concentre sur les zones C et D du parking sachant que c'est le même principe sur tous les zones.

##### 1. Lire les flux vidéo des cameras :

Les cameras du parking sont des caméras de type IP de la société Hikvision. Elles communiquent à travers des câbles à connecteur RJ45 POE.



Figure 49: HikVision camera

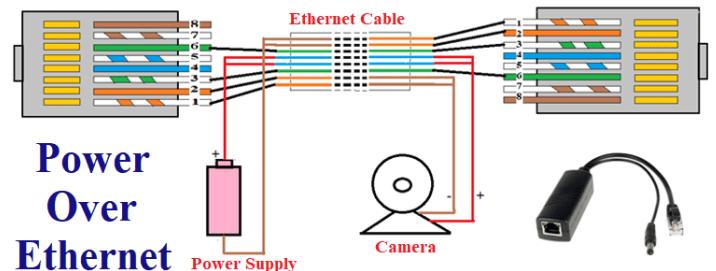
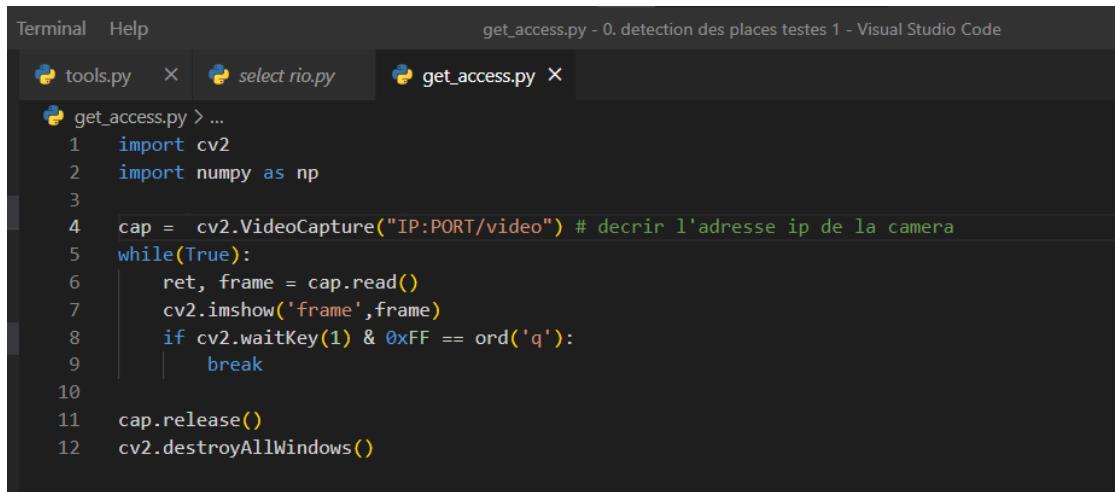


Figure 50: Cable POE

La bibliothèque OpenCV de Python nous offre une méthode d'accès à ces caméras connectées localement sur les switches de la société (réseaux LAN). Il suffit seulement de connecter l'unité de traitement aux réseaux contenant les camera. Chaque camera a son propre adresse IP, Donc on doit impérativement avoir cette adresse et on doit configurer le réseaux de telle manière qu'on ait l'accès à ces derniers.

Le Code suivant nous permet à ce connecter au camera :



```
Terminal Help get_access.py - 0. detection des places testes 1 - Visual Studio Code
tools.py × select_rio.py × get_access.py ×
get_access.py > ...
1 import cv2
2 import numpy as np
3
4 cap = cv2.VideoCapture("IP:PORT/video") # definir l'adresse ip de la camera
5 while(True):
6     ret, frame = cap.read()
7     cv2.imshow('frame',frame)
8     if cv2.waitKey(1) & 0xFF == ord('q'):
9         break
10
11 cap.release()
12 cv2.destroyAllWindows()
```

Figure 51: code responsable à se connecter à la caméra IP

## 2. Exploiter les techniques de multiplexages :

Le problème qui se génère lors de la lecture de plusieurs cameras, est la saturation des ressources de l'ordinateur ce qui nécessite des puissances énormes.

Tous les tests ici sont faite sur un ordinateur de processeur Intel I7 de la 8 génération (Intel® Core™ i7-8559U Processor) avec 4 cœurs avec 20 Gb de ram DDR4 et une unité de stockage SSD.



On a exploité deux techniques de multiplexage :

### 1) Big Image :

Cette technique se base sur la génération d'un flux vidéo qui contient la concaténation des vidéos générées par chaque camera. En suivant les étapes suivantes :

- Connecter les camera :

```
import cv2 #opencv
from tools import *
## zone c
cap_C01_C06 = cv2.VideoCapture(r'video croper 10s zones d et c\zones c\C01 - C06.mp4')
cap_C13_C07 = cv2.VideoCapture(r'video croper 10s zones d et c\zones c\C13 - C07.mp4')
cap_C19_C13 = cv2.VideoCapture(r'video croper 10s zones d et c\zones c\C19 - C13.mp4')
# zone d
cap_D01_C06 = cv2.VideoCapture(r'video croper 10s zones d et c\zones d\D01 - D04.mp4')
cap_D07_C04 = cv2.VideoCapture(r'video croper 10s zones d et c\zones d\D07 - D04.mp4')
cap_D11_D08 = cv2.VideoCapture(r'video croper 10s zones d et c\zones d\D11 - D08.mp4')
cap_D15_D12 = cv2.VideoCapture(r'video croper 10s zones d et c\zones d\D15 - D12.mp4')
cap_D18_D15 = cv2.VideoCapture(r'video croper 10s zones d et c\zones d\D18 - D15.mp4')
cap_D19_D16 = cv2.VideoCapture(r'video croper 10s zones d et c\zones d\D19 - D16.mp4')
```

Figure 52: Connecter plusieurs cameras

- Lire le flux vidéo de chaque camera :

- Pour lire les flux vidéo de chaque camera on crée une liste qui contient les camera.

$f_{\Delta}(x)$

```
15
16
17    # liste de camera pour la zone c et d
18    list_of_captures = [cap_D01_C06 ,cap_D07_C04 ,cap_D11_D08 ,cap_D15_D12 ,cap_D19_D16, cap_C01_C06 ,cap_C13_C07, cap_C19_C13] #cameras
19
20
21
```

Figure 53: liste de flux de la part des cameras

- Puis on boucle sur chaque camera et on lit les frames.

- Sélectionner les régions d'intérêt de chaque camera & obtenir le vidéo global :

On doit sélectionner la région d'intérêt de chaque camera, cela suppose que les cameras ne contiennent aucune zone aveugle pour faire les traitements. Une fonction sous le nom « `select_RIO` » qui va s'occuper de ce traitement.

```

22
23 def select_RoI (list_of_captures):
24     rets,frames = [],[]
25     for cap in list_of_captures :
26         ret , frame = cap.read()
27         rets.append(ret)
28         frames.append(cv2.resize(frame, (720,480)))
29
30     posList = []
31
32     posList = getPositions(frames)
33
34     perfectPos = []
35     for i in posList:
36         pos = i[4]
37         perfectPos.append(pos)
38     cv2.destroyAllWindows()
39
40     print(np.array(perfectPos))
41
42 #loop non terminé

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER  
File "c:\Users\ahbar\OneDrive\Documents\Stage technique\my old pc code\multiplix

Figure 54: code responsable sur la sélection des régions d'intérêt

- Sélection des régions dans chaque camera :



Figure 55: résultat de la sélection des régions d'intérêt

Manuellement, on indique les points rouges qui indiquent les limites de la région d'intérêt dans cette caméra, puis on refait le même travail dans les nouveaux frames jusqu'à finir avec tous les caméras.

- **Obtenir la vidéo globale :**

```

ret, frame = cap.read()
rets.append(ret)
frames.append(cv2.resize(frame, (720,480)))

List
List
effect
i j
pos = D01, D02, D03, D04, D05, D06, D07, D08, D09, D10, D11, D12, D13, D14, D15, D16, D17, D18, D19
perfectPos.append(pos)
.destroyAllWindows()

```

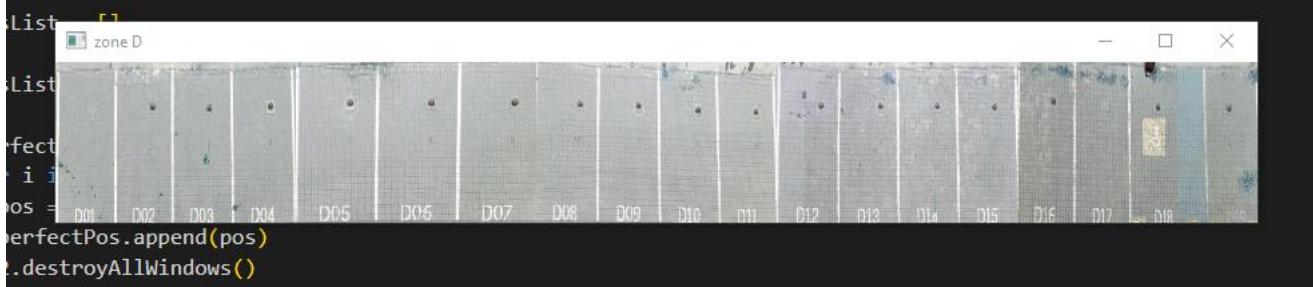


Figure 56: BIG IMAGE

- **Sélectionner les emplacements de stationnement :**

La sélection des spots de stationnement peut se faire avec deux manières soit en utilisant le computer vision, ou bien juste par les techniques de traitement d'image.

- **Par Deep Learning :**

Le principe se base sur un algorithme qui détecte les voitures dans la vidéo globale puis on détermine approximativement le centroïde de l'Object détecté et après on vérifie dans quelle partie la voiture est détectée ? C'est-à-dire dans quel domaine de stationnement le point centroïde de cordonnés ( $x_c, y_c$ ) appartient. C'est-à-dire répondre à la question est-ce-que le centroïde de l'appartient ou domaine D01, ou D02, ..., ou bien D20 ?

- **La démonstration géométrique sur cette question est la suivante :**

Soit ABCD un Quadrilatère dans le plan géométrique noté  $P(o, x, y)$  et  $x$  un point de  $P$  :

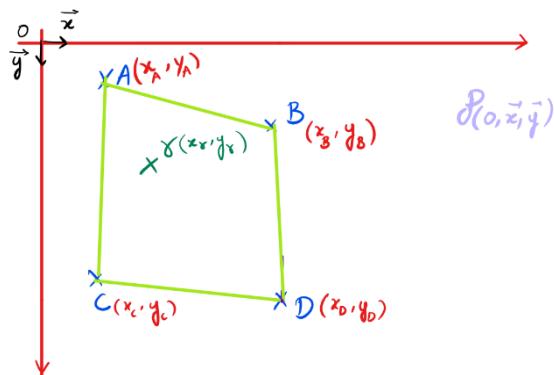
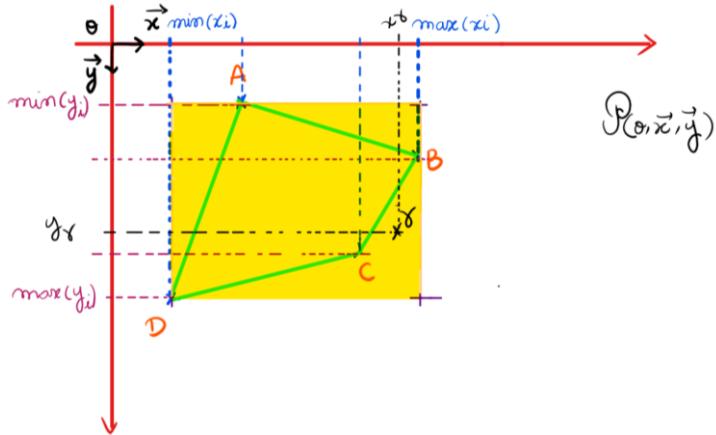


Figure 57: abstraction en problème géométrique

⇒ **Quelles sont les conditions pour que  $x$  soit dans le quadrilatère ABCD ?**

- Soit  $\Omega = \{A, B, C, D\}$  on note  $x_{i \in \Omega}$  et  $y_{i \in \Omega}$  les coordonnées du point  $i \in \Omega$ . Et donc pour que  $\gamma \in ABCD$  il faut que  $x_\gamma \in [\min(x_i), \max(x_i)]$  et de même  $y_\gamma \in [\min(y_i), \max(y_i)]$  telle que  $i \in \Omega$  cette condition est nécessaire mais pas suffisante.



Et donc :

- On note les intervalles  $[\min(x_i), \max(x_i)]_{i \in \Omega}$  et  $[\min(y_i), \max(y_i)]_{i \in \Omega}$  respectivement  $J$  et  $K$ .
- Si  $x_\gamma \in J$  et  $y_\gamma \in K$  alors :

On doit obtenir les équations des quatre droites  $(AB)$ ,  $(CD)$ ,  $(AD)$ , et  $(BC)$ .

Pour  $(AB)$  : pour toute  $M \in (AB)$  et  $M$  de coordonnées  $(x_m, y_m)$  il existe  $\alpha, \beta$  telle que :  $y_m = \alpha x_m + \beta$

Déterminant  $\alpha$ , et  $\beta$  :

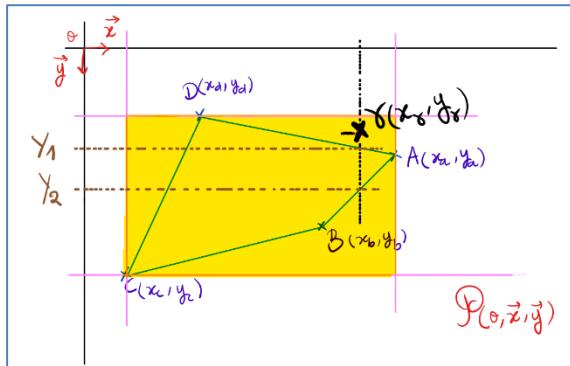
$$\begin{cases} y_a = x_a \cdot \alpha + \beta \\ y_b = x_b \cdot \alpha + \beta \end{cases} \Rightarrow \begin{cases} \beta = y_a - x_a \frac{y_b - y_a}{x_b - x_a} \\ \alpha = 1 \frac{y_b - y_a}{x_b - x_a} \end{cases}$$

- De même pour  $(CD)$ ,  $(AD)$ , et  $(BC)$  :

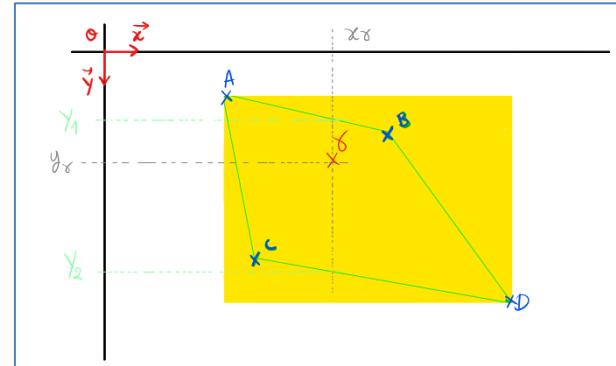
On appelle  $f_\Delta(x) = x \cdot \alpha + \beta$  la fonction de la droite  $(\Delta)$ . Avec  $\alpha$ , et  $\beta$  sont déterminées par la manière précédente. Si  $x_\gamma \in J$  on doit déterminer  $Y_1$  et  $Y_2$  tel que  $\gamma$  soit dans le quadrilatère  $ABCD$ , il faut que  $y_\gamma \in [Y_1, Y_2]$  :

- Avec  $Y_1$  et  $Y_2$  sont l'intersection de la droite  $(D)$  :  $x_\gamma$  Avec les deux droite  $(AB)$  et  $(CD)$ .

C'est-à-dire  $Y_1 = f_{(AB)}(x_\gamma)$  et  $Y_2 = f_{(CD)}(x_\gamma)$ .



1er cas :  $\gamma$  n'appartient pas à  $ABCD$



2ème cas :  $\gamma$  appartient à  $ABCD$

- En fait la même chose pour les axes des  $x$ , par la même procédure on obtient  $X_1$  et  $X_2$

- Par conclusion :

*Deux conditions nécessaires et suffisantes pour qu'un point  $M$  appartient au Quadrilatère  $ABCD$  :*

1.  $x_m \in [\min(x_i), \max(x_i)]$  et  $y_m \in [\min(y_i), \max(y_i)]$  telle que  $i \in \{a, b, c, d\}$
2. Pour  $x_m$  et  $Y_1 = f_{(AB)}(x_\gamma)$  et  $Y_2 = f_{(CD)}(x_\gamma)$ . Il faut que :  
 $y_m \in [\min(Y_1, Y_2), \max(Y_1, Y_2)]$ , telle que  $f_\Delta(x) = x \cdot \alpha_\Delta + \beta_\Delta$
3. Pour  $y_m$  et  $X_1 = f_{(AD)}^{-1}(y_\gamma)$  et  $X_2 = f_{(CB)}^{-1}(y_\gamma)$ . Il faut que :  
 $x_m \in [\min(X_1, X_2), \max(X_1, X_2)]$ , telle que  $f_\Delta^{-1}(y)$  est la fonction inverse de  $f_\Delta(x)$

## 2) Par traitement d'image :

- 1) Pour chaque place de stationnement dans la vidéo global obtenue, on filtre et on applique des transformations sur cette zone afin de savoir si la place est vide ou remplie.
- 2) Remarque : Cette méthode prend plus de ressources et même avec la configuration puissante de notre ordinateur, il n'arrive pas à gérer c'est traitement.



Figure 58: Pour une seule zone

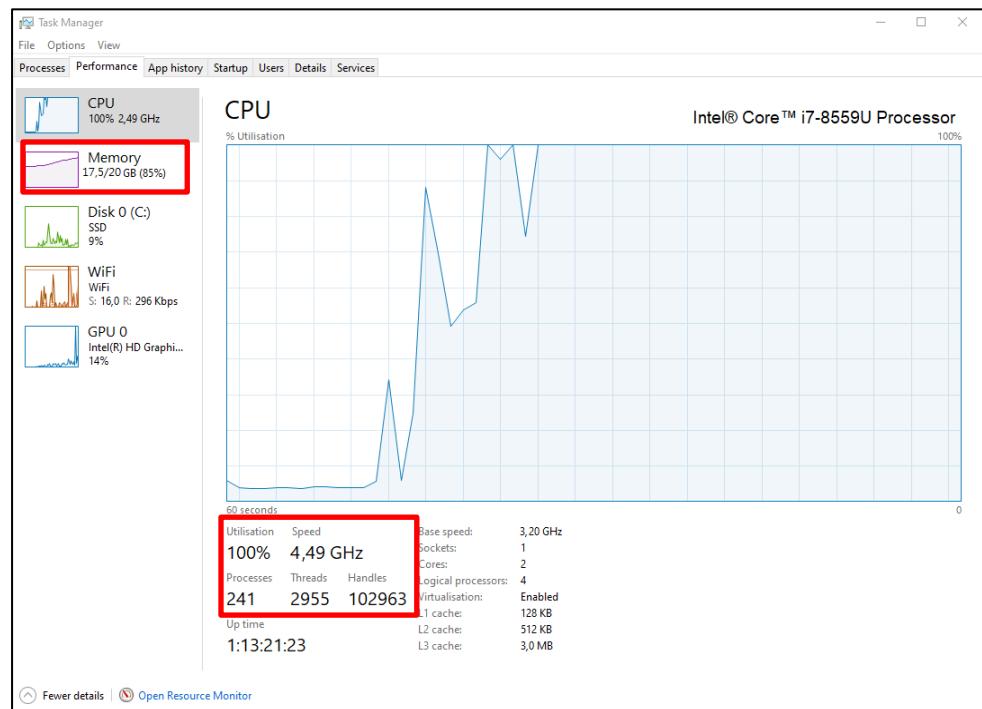


Figure 59: pour plusieurs zones

3) Indiquer les places remplis des places vides dans chaque zone :

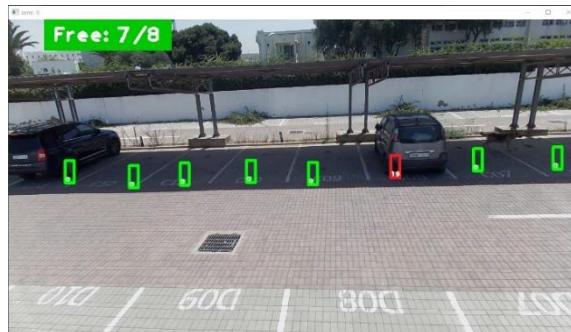


Figure 60: indication sur les places vides et remplis

A la fin, le programme indique le nombre total de places vides et remplis dans le vidéo et ceci peut être importé sur une carte virtuelle.

### Multiple vidéo :

Le principe de cette méthode se base sur le traitement en série des flux prévenant du camera. On connecte les caméras et on sélectionne pour chaque caméra la région d'intérêt. On sélectionne les places de stationnement et on sauvegarde ces données dans un fichier de type Picle qu'on appelle chaque fois lors du démarrage le programme de surveillance.

#### *1. Sélection de la région d'intérêt dans chaque camera :*

Code responsable sur la sélection des places :

```

EXPLORER
...
OPEN EDITORS
  main_.py
  WIN_20220805_12_31_30_Pro.mp4
1. MULTIPLE CAMERA
  vedio implementation
    autreface.mp4
    CarParkPos0
    CarParkPos1
    CarParkPos2
    CarParkPos3
    main_.py
    test.py
  WIN_20220805_12_18_31_Pro.mp4

main_.py  WIN_20220805_12_31_30_Pro.mp4
main_.py > [paths_or_Cameralds
1
2 This code is under development for INNOVATEL ENGINEERING :
3
4 >>> Developer : Ahbar Abdellah & Sikou Mohamed
5 >>> Adress : 47, rue des Aït Ba Amrane 20300 Casablanca - Maroc
6 >>> Project name : Smart Parking INNOVATEL
7 >>> requirement : under development
8 >>> last modification : 09-august-2022 02:43 pm
9
10 ...
11
12 import cv2
13 import numpy as np
14 import pickle
15 import cvzone
16
17 #camera id :
18 paths_or_Cameralds = [r'autreface.mp4', 'vedio implementation\WIN_20220805_12_21_06_Pro.mp4'
19
20
21 def mouseClickGenerator(PosList):
22     def mouseClick(events, x, y, flags, params):
23         if events == cv2.EVENT_LBUTTONDOWN:
24             PosList.append((x, y))
25     return mouseClick
26
27 def getPositions(img):
28     posList = []
29     fun = mouseClickGenerator(posList)

```

Figure 61: Code responsable sur la sélection des places

## **2. L'étape de la sélection :**

En cliquant sur le bouton gauche de la souris, ceci permet de faire une sélection, et le bouton droit permet de supprimer la sélection.

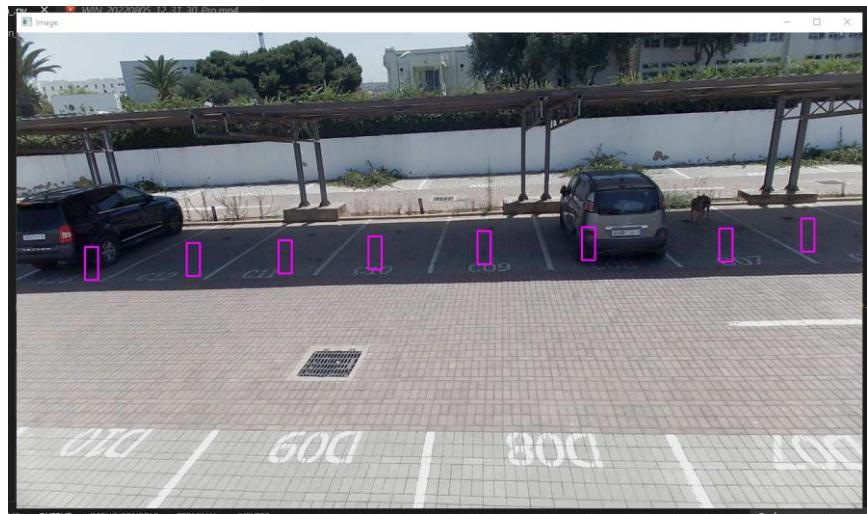


Figure 62: sélections des places

Après la sélection, cette dernière est stockée dans des fichiers afin d'éviter la répétition de la sélection chaque fois on démarre le programme. Chaque zone couverte par une caméra possède un fichier comme montré ci-dessous :

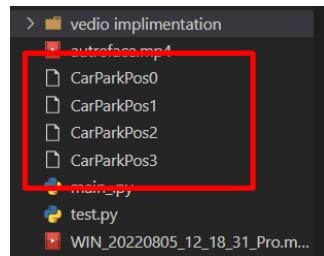


Figure 63: fichier d'annotation sauvegarder

### 3. Surveillance :



Figure 64: chaque vue sur une caméra séparer



Figure 65: détection en temps réelle



Figure 66: multiple vue sur une seules fenêtre

## B. Implémentation de Suivie des voitures (Object Traking) :

### - Remarque :

Cette partie nous a pris plus d'un mois de développement mais malheureusement on n'a pas pu arriver à grand-chose. Je cite ici les essais qu'on a fait et les problèmes qu'on a rencontrés.

### B.1 suivie à l'aide de l'algorithme de centroïde :

#### - Présentation de la méthode :

Cette méthode est déjà expliquée dans le chapitre 2 (Réalisation d'un prototype). En gros cette méthode consiste de détecter l'objet puis on suppose qu'entre deux frames de la vidéo, la voiture ne dépasse pas une région qu'on définit par un rayon epsilon de confiance.

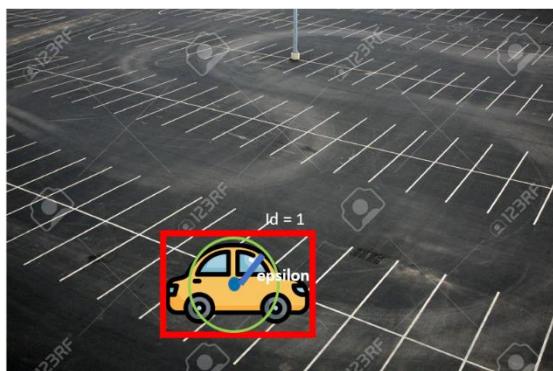


Figure 67: frame 1



Figure 68 : frame 3

On a testé l'algorithme comme il est représenté dans les figures 26 et 27.

#### - Problème rencontrer :

Dans le cas d'une implémentation réelle l'algorithme a trois désavantages majeurs.

##### 1. La Fiabilité :

Cet algorithme dépend à l'algorithme de la détection des voitures et donc si pour un seul frame l'algorithme n'arrive pas à détecter la voiture et après il a détecté il considère que c'est une nouvelle voiture ce qui rend la dernière places ou l'algorithme à perdre la détection comme occupé. Dans la part de la base de données.

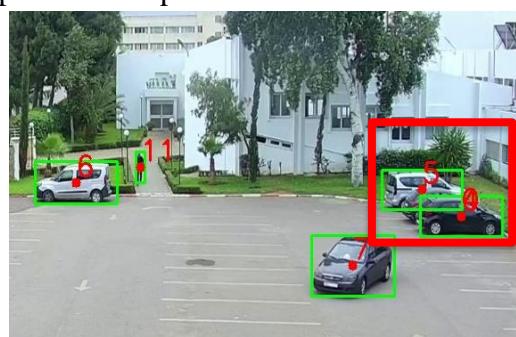


Figure 69: 3 voitures mais que 2 détecter

## 2. Interaction avec les obstacles

Même on a essayé de créer un rayon adapter et on a joué sur tous les hyperparamètres (rayons de confiance epsilon, le box de détection, le centroïde, ...).

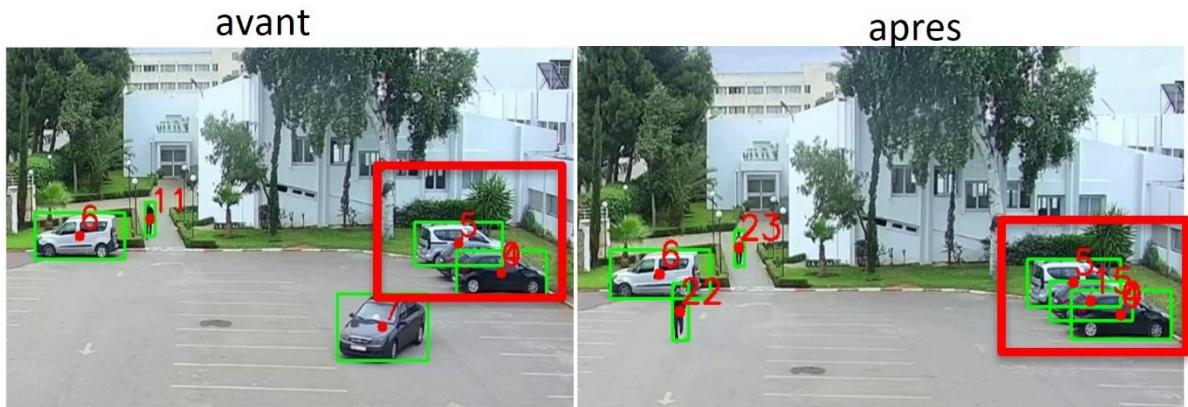


Figure 70: l'algorithme n'arrive pas à interagir avec les obstacles

## 3. Il n'est pas adapté pour plusieurs vues :

S'il y a une zone non couverte par les cameras cette méthode n'arrivera pas à synthétiser l'emplacement de la véhicule.

### - Analyse de la complexité temporelle :

#### Détection des objets :

```
object_detection.py ×
object_detection.py > ObjectDetection > __init__
1 import cv2
2 import numpy as np
3
4
5 class ObjectDetection:
6     def __init__(self, weights_path="dnn_model/yolov4.weights", cfg_path="dnn_model/yolov4.cfg"):
7         print("Loading Object Detection")
8         print("Running opencv dnn with YOLOv4")
9         self.nmsThreshold = 0.4
10        self.confThreshold = 0.5
11        self.image_size = 608
12
13        # Load Network
14        net = cv2.dnn.readNet(weights_path, cfg_path)
15
16        # Enable GPU CUDA
17        net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
18        net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)
19        self.model = cv2.dnn_DetectionModel(net)
20
21        self.classes = []
22        self.load_class_names()
23        self.colors = np.random.uniform(0, 255, size=(80, 3))
24
25        self.model.setInputParams(size=(self.image_size, self.image_size), scale=1/255)
26
27    def load_class_names(self, classes_path="dnn_model/classes.txt"):
28
29        with open(classes_path, "r") as file_object:
30            for class_name in file_object.readlines():
31                class_name = class_name.strip()
32                self.classes.append(class_name)
33
34        self.colors = np.random.uniform(0, 255, size=(80, 3))
35        return self.classes
36
37    def detect(self, frame):
38        return self.model.detect(frame, nmsThreshold=self.nmsThreshold, confThreshold=self.confThreshold)
```

Complexité =  $O(T)$

Temps fixe mais qui dépend de la taille de l'image de l'input

Figure 71: code de la détection des objets

## Algorithme de suivi :

```

object_tracking.py X
object_tracking.py > [o] box
1 import cv2
2 import numpy as np
3 from object_detection import ObjectDetection
4 import math
5
6 # Initialize Object Detection
7 od = ObjectDetection()
8
9 cap = cv2.VideoCapture("test.mp4")
10 i = 0
11 # Initialize count
12 count = 0
13 center_points_prev_frame = []
14
15 tracking_objects = {}
16 track_id = 0
17 fourcc = cv2.VideoWriter_fourcc(*'XVID')
18 video_writer = cv2.VideoWriter('outringe.mp4', fourcc, 20, (640, 480))
19
20
21 while True:
22     ret, fram = cap.read()
23     frame = cv2.resize(fram,(1280,720),fx=0,fy=0, interpolation = cv2.INTER_CUBIC) # 1280,720
24     count += 1
25     i += 1
26     if not ret:
27         break
28
29     # Point current frame
30     center_points_cur_frame = []
31
32     # Detect objects on frame
33     (class_ids, scores, boxes) = od.detect(frame)
34     for box in boxes:
35         (x, y, w, h) = box
36         cx = int((x + x + w) / 2)
37         cy = int((y + y + h) / 2)
38         center_points_cur_frame.append((cx, cy))
39         ##print("FRAME N°", count, " ", x, y, w, h)
40
41         # cv2.circle(frame, (cx, cy), 5, (0, 0, 255), -1)
42         cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
43
44     # Only at the beginning we compare previous and current frame
45     if count <= 2:
46         for pt in center_points_cur_frame:
47             for pt2 in center_points_prev_frame:
48                 distance = math.hypot(pt2[0] - pt[0], pt2[1] - pt[1])
49
50                 if distance < 20:
51                     tracking_objects[track_id] = pt
52                     track_id += 1
53     else:
54         tracking_objects_copy = tracking_objects.copy()
55         center_points_cur_frame_copy = center_points_cur_frame.copy()
56
57         for object_id, pt2 in tracking_objects_copy.items():
58             object_exists = False
59             for pt in center_points_cur_frame_copy:
60                 distance = math.hypot(pt2[0] - pt[0], pt2[1] - pt[1])
61                 # Update IDs position
62                 if distance < 20:
63                     tracking_objects[object_id] = pt
64                     object_exists = True
65                     if pt in center_points_cur_frame:
66                         center_points_cur_frame.remove(pt)
67                     continue
68
69                     # Remove IDs lost
70                     if not object_exists:
71                         tracking_objects.pop(object_id)
72
73                     # Add new IDs found
74                     for pt in center_points_cur_frame:
75                         tracking_objects[track_id] = pt
76                         track_id += 1
77
78                     for object_id, pt in tracking_objects.items():
79                         cv2.circle(frame, pt, 5, (0, 0, 255), -1)
80                         cv2.putText(frame, str(object_id), (pt[0], pt[1] - 7), 0, 1, (0, 0, 255), 2)
81
82         cv2.imshow("Frame", frame)
83
84         # Make a copy of the points
85         center_points_prev_frame = center_points_cur_frame.copy()
86         video_writer.write(frame)
87         print(i)
88
89         # Wait for a key press
90         key = cv2.waitKey(1) & 0xFF
91         if key == 27: break
92
93         cap.release()
94         video_writer.release()
95         cv2.destroyAllWindows()
96
97
98
99

```

- Pour chaque frame (24 ~ 30) / second

# N : nombre de frame

-Pour chaque véhicule détecter dans chaque frame

# V : nombre de véhicule

$\Rightarrow o(V^2)$

Donc : complexité de  $o(N.V^2)$ .

La complexité de l'algorithme est : quadratique

## B.2 suivie à l'aide de l'algorithme « One Shot Problème » :

### - Présentation de la méthode :

Cette méthode est basée sur un algorithme de Deep Learning, appelé one shot détection.

Définition :

‘La détection d’objets One-Shot (OSOD) consiste à détecter un objet à partir d’un seul exemple par catégorie. Contrairement au détecteur d’objets qui nécessite de nombreux exemples variés d’objets dans le monde réel, le détecteur d’objets One-Shot nécessite un très petit (parfois même un seul) exemple canonique de l’objet.’

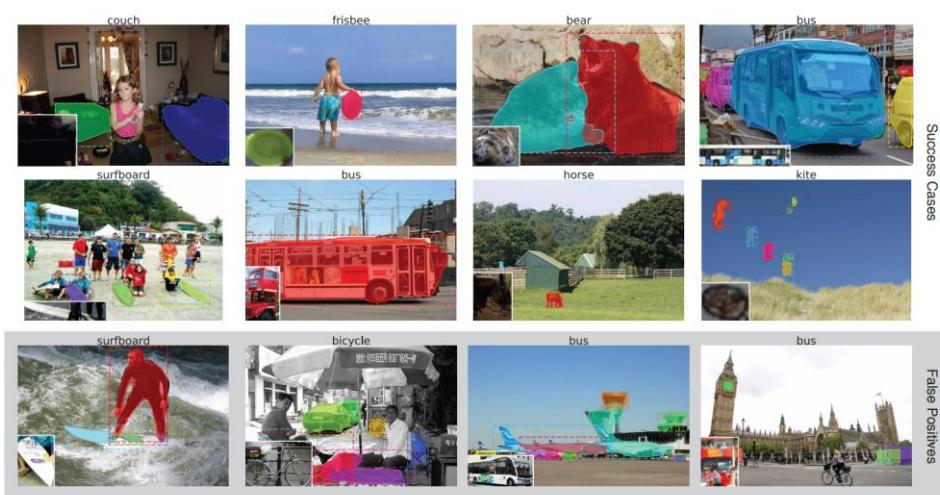


Figure 72: segmentation et détection à l'aide de OSOD.

### - Comment on va utiliser cette méthode :

L’utilisation de cette méthode consiste à entraîner un modèle OSOD à chaque fois on détecte un nouveau véhicule pour l’identifier directement. Et à chaque fois que le véhicule sort du parking on supprime le modèle.

### - Problème rencontrer :

Le problème avec cette méthode qu’il est très coûteux dans notre cas :

Car l’utilisation vas consister à entraîner et utiliser en même temps dans un processus itératif. Et l’entraînement prend un temps important entre 30 secondes et 15 minutes pour chaque nouveau véhicule. Ce qui rend le système de surveillance infonctionnelle pendant la période d’entraînement.

### B.3 extraction de factures :

#### - Présentation de la méthode :

Cette méthode va consister à extraire les paramètres des dernières couches du réseaux de neurone qui est responsable de la détection et après à travers un autre model de détecter est ce qu'il s'agit du même véhicule ou non.

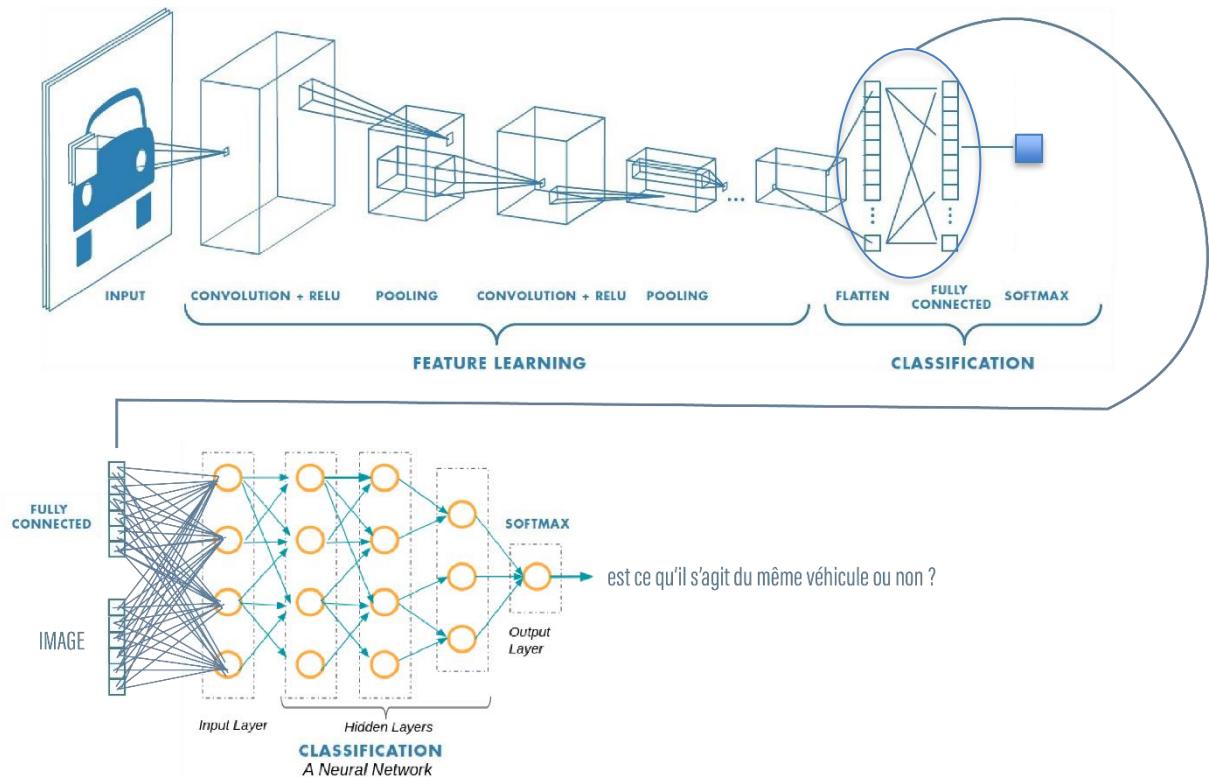


Figure 73: extraction de paramètres et détection de similarités

#### - Problème rencontré :

La fonction d'optimisation dans YOLO est entraînée pour détecter les véhicules donc quelques soit la voitures les paramètres des dernières couches sont très similaires dans le cas d'une voiture ce qui rend l'algorithme incapable de détecter la différence de plus ils ne sont pas très significants et donc l'algorithme est toujours tombe dans le UNDERFITTING.

De plus la complexité sera très grand  $O(N \cdot V^2)$  où N est nombre de frame, et V le nombre de véhicule

#### - Résultat :

```
[ ] history = model.fit(X_train_res, y_train_enc, epochs=500, validation_split=0.2)
Epoch 1/500
199/199 [=====] - 1s 3ms/step - loss: 1.1967 - accuracy: 0.5039 - val_loss: 2.1967 - val_accuracy: 0.4902
Epoch 2/500
199/199 [=====] - 0s 2ms/step - loss: 0.8853 - accuracy: 0.6433 - val_loss: 2.1967 - val_accuracy: 0.4902
Epoch 3/500
199/199 [=====] - 0s 2ms/step - loss: 0.7651 - accuracy: 0.6930 - val_loss: 2.1967 - val_accuracy: 0.4902
Epoch 4/500
199/199 [=====] - 0s 2ms/step - loss: 0.7028 - accuracy: 0.7263 - val_loss: 2.1967 - val_accuracy: 0.4902
Epoch 5/500
199/199 [=====] - 0s 2ms/step - loss: 0.6818 - accuracy: 0.7308 - val_loss: 2.1967 - val_accuracy: 0.4902
Epoch 6/500
199/199 [=====] - 0s 2ms/step - loss: 0.6344 - accuracy: 0.7469 - val_loss: 2.1967 - val_accuracy: 0.4902
Epoch 7/500
199/199 [=====] - 0s 2ms/step - loss: 0.6189 - accuracy: 0.7498 - val_loss: 2.1967 - val_accuracy: 0.4902
Epoch 8/500
199/199 [=====] - 0s 2ms/step - loss: 0.6067 - accuracy: 0.7565 - val_loss: 2.1967 - val_accuracy: 0.4902
Epoch 9/500
199/199 [=====] - 0s 2ms/step - loss: 0.5848 - accuracy: 0.7666 - val_loss: 2.1967 - val_accuracy: 0.4902
Epoch 10/500
199/199 [=====] - 0s 2ms/step - loss: 0.5782 - accuracy: 0.7703 - val_loss: 2.1967 - val_accuracy: 0.4902
Epoch 11/500
199/199 [=====] - 0s 2ms/step - loss: 0.5677 - accuracy: 0.7821 - val_loss: 2.1967 - val_accuracy: 0.4902
```

Figure 75 : entrainement

```
[ ] print(accuracy_score(y_test_enc, y_pred_enc))
0.7829457364341085
[ ] print(classification_report(y_test_enc, y_pred_enc))

precision    recall   f1-score   support
          0       0.26      0.56      0.36       18
          1       0.36      0.68      0.47       53
          2       0.94      0.82      0.88      661
          3       0.29      0.43      0.34       42
```

Figure 74 : mauvais résultat de test

- En remarque que pendant l'entraînement tout vas bien mais dans l'évaluation le model ne fonctionne pas correctement et donne des mauvais indices.

### - *Solution proposer :*

On crée une architecture mais dans la période d'entraînement en lie les deux fonctions d'optimisation et de mise ajour des poids dans le même temps. Malheureusement on n'a pas pu tester cette solution.

### C. Est-ce que on ait bousions au Traking ?

Dans cette partie je vais proposer d'autre méthode plus simple qui peuvent remplacer le Traking.

- Détection de matricule : pour avoir une traçabilité sur l'ensemble des voitures stationner sur les emplacements dans le parking.

The screenshot shows a Visual Studio Code interface with two code snippets. The top snippet is titled '4. Use Easy OCR To Read Text' and contains the following Python code:

```
1 reader = easyocr.Reader(['en'])
2 result = reader.readtext(cropped_image)
3 result
... [[0, 0], [44, 0], [44, 53], [0, 53], 'H982 FKL', 0.736311316490173]]
```

The bottom snippet is titled '5. Render Result' and contains the following Python code:

```
1 text = result[0][3]
2 font = cv2.FONT_HERSHEY_SIMPLEX
3 res = cv2.putText(img, text=text, org=(approx[0][0]), approx[0][0][1]+40), fontFace=font, fontScale=1, color=(0,255,0), thickness=2, lineType=cv2.LINE_AA)
4 res = cv2.rectangle(img, tuple(approx[0][0]), tuple(approx[2][0]), (0,255,0), 2)
5 pic.imshow(cv2.cvtColor(res, cv2.COLOR_BGR2RGB))
... <ipython-input-11-458f858be0>
```

Below the code, there is a small image of a car's license plate with the text 'H982 FKL' highlighted in green.

Figure 76: OCR

Le problème dans cette méthode qu'il n'existe pas de modèle pour les caractères arabe mais a licence commerciale. Ce qui rend nécessaire de crée un model OCR pour les caractères arabe

- Collecter la data : rassembler plusieurs échantillons des lettres arabes
- Entrainer un réseau de neurone : pour classifier les lettres
- Teste les performances.
- Pour mètres un niveau de redondance on peut enregistre OCR et l'image du matricule.

# **Conclusion générale :**

Pour conclure, ce projet a été effectué au sein de l'entreprise Innovatel, et parmi plusieurs sujets qui présente une nécessité, nous avons choisi le sujet Supervision d'un parking intelligent à l'aide des caméras de surveillance.

L'objectif était d'effectuer une analyse sur l'état actuelle du parking et de trouver une solution pour éliminer les capteurs de présence afin d'optimiser les coûts ainsi qu'améliorer la qualité des résultats.

Après tout un tas d'analyse et de recherche, on a pu mettre proposer des solutions amélioratives avec un minimum d'investissement engendrent des résultats de bonnes qualités en se basant sur la vision par ordinateur pour pouvoir superviser les voitures et suivre leur emplacement en utilisant des caméras de surveillance.

# Bibliographies

[1]: Build your own Vehicle Detection Model using OpenCV and **Python**:

- <https://www.analyticsvidhya.com/blog/2020/04/vehicle-detection-opencv-python/>

[2]: A tutorial on Centroid Tracker & Counter System :

- <https://www.analyticsvidhya.com/blog/2022/05/a-tutorial-on-centroid-tracker-counter-system/>

[3]: Implementing Real-time Object Detection System using PyTorch and OpenCV :

- <https://towardsdatascience.com/implementing-real-time-object-detection-system-using-pytorch-and-opencv-70bac41148f7>

[4]: Robotic and computer vision:

- <https://www.youtube.com/c/MurtazasWorkshopRoboticsandAI>

[5]: one shot learning :

- <https://datahacker.rs/deep-learning-one-shot-learning/>
- One-Shot Instance Segmentation:
  - Claudio Michaelis - Ivan Ustyuzhaninov Matthias Bethge - Alexander S. Ecker
  - University of Tübingen
  - <https://arxiv.org/pdf/1811.11507v2.pdf>

[6]: you only look ones

- <https://arxiv.org/pdf/1506.02640.pdf>

[7]: Object Tracking in Computer Vision (Complete Guide):

- <https://viso.ai/deep-learning/object-tracking/>