



HAUTE ECOLE DE NAMUR-LIEGE-LUXEMBOURG
DEPARTEMENT INGENIEUR INDUSTRIEL
PIERRARD - VIRTON

**SYSTEME INTELLIGENT : RECONNAISSANCE D'UNE PLAQUE
D'IMMATRICULATION ET CONTROLE D'UNE BARRIERE
D'ENTREE**

Année Académique
2018 – 2019

DENIS Jérémy
KRAUS Geoffrey
M1A

Table des matières

1.	Introduction :	4
2.	Tableau des composants	5
3.	Time line final	6
4.	Partie 1 : Arduino	7
1.	Composants :	7
2.	Programme :	9
3.	Modélisation 3D	11
5.	Partie Raspberry (reconnaissance)	12
1.	Installation de TensorFlow	12
2.	Installation d'OpenCV	14
3.	Installation de Protobuf	14
4.	Installation des modèles TensorFlow	15
5.	PYTHONPATH	15
6.	Premier test de reconnaissance :	15
7.	Collecte de photos	16
8.	Labellmg	17
9.	Conversion du format .xml en .csv	18
10.	Conversion du format .csv en .TFRecord	18
11.	Fichier de configuration du modèle :	19
12.	Entrainement du modèle :	19
6.	Conclusion :	23
7.	Bibliographie :	24

Table des images et figures :

Images :

Image 3-1 - Schéma de câblage de l'arduino.....	7
Image 3-2 - Micro servomoteur [1]	8
Image 3-3 - Graphe temporel des impulsions PWM [2].....	8
Image 3-4 - Barrière.....	11
Image 3-5 - Supports du servomoteur	11
Image 4-1 - Test de TensorFlow sur le Raspberry	13
Image 4-2 - Test de Protobuf sur le Raspberry.....	15
Image 4-3 - Premier test sur Raspberry	16
Image 4-4 - Exemple de photo prise pour l'entrainement.....	17
Image 4-5 - Logiciel LabellImg	17
Image 4-6 - Abandon du processus d'entrainement sur Raspberry.....	20
Image 4-7 - Fin de l'apprentissage du modèle sur Lubuntu	20
Image 4-8 - Modèle entrainé utilisé sur Lubuntu.....	21
Image 4-9 - Test final de la détection de plaque	22

Figure :

Figure 3-1 - Ordinogramme du code arduino.....	10
--	----

1. Introduction :

Dans le cadre du projet du cours de microcontrôleur et système intelligent, nous avons décidé de créer un système d'ouverture de barrière à la détection d'une plaque d'immatriculation d'un véhicule. Ce projet se divise donc en deux grandes parties : Raspberry et Arduino.

En effet, toute la partie commande de la barrière avec un écran et une lampe sera gérée par l'Arduino qui est un microcontrôleur. Et toute la partie reconnaissance de plaque est gérée par le Raspberry avec sa caméra pi.

Chaque partie sera réalisée séparément pour ensuite réaliser une communication entre les deux systèmes par le biais de un ou plusieurs câbles.

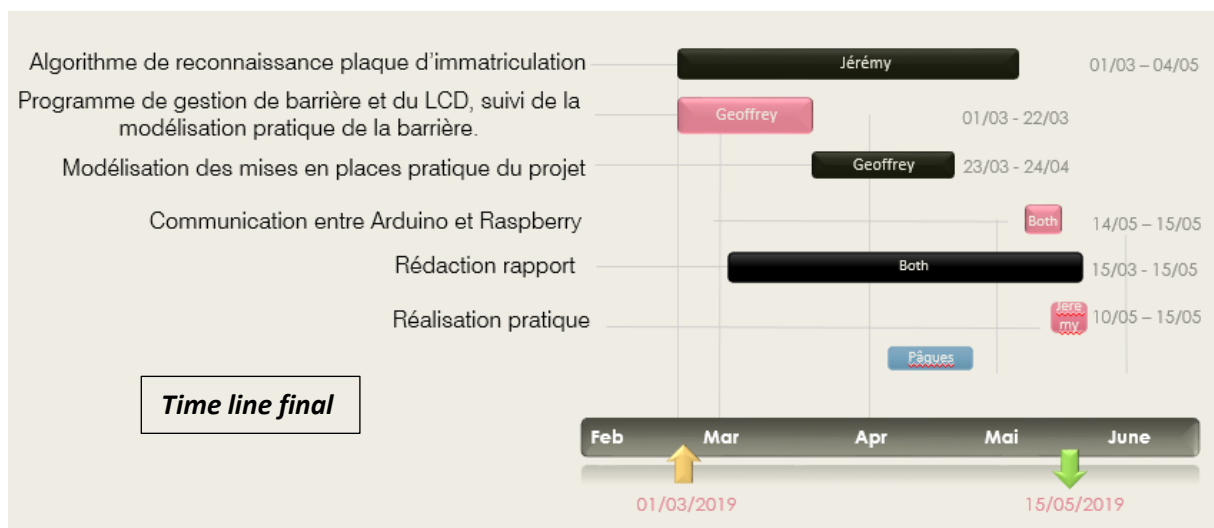
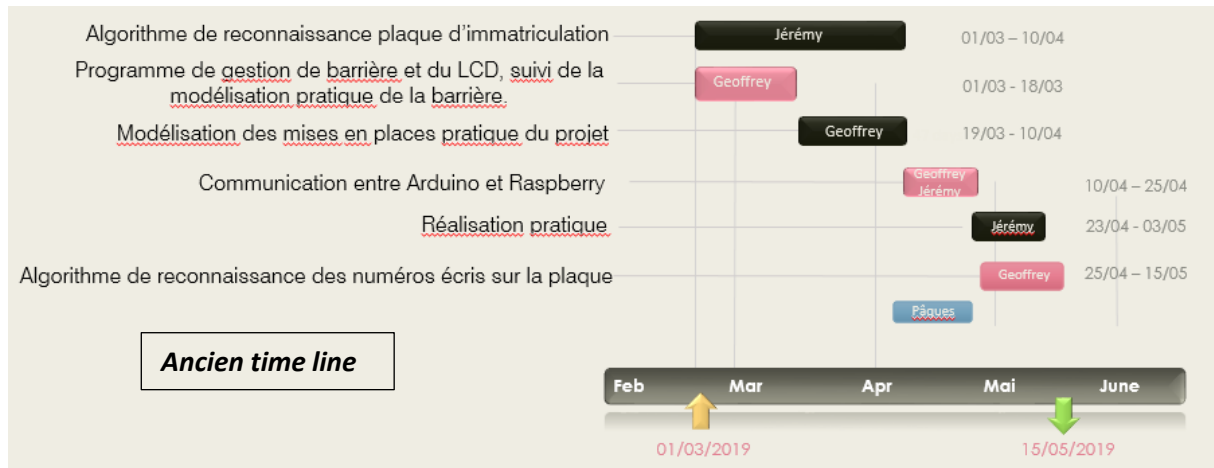
2. Tableau des composants

Partie	Composants	Prix unit.	Qte	Prix tot.	Fournisseur	Réf site	Lien	Etat
Arduino	Arduino Uno	20,75 €	1	20,75 €	Digi-Key	A000066	https://www.digikey.com	Fourni par l'école
	LCD	7,57 €	1	7,57 €	Amazon	DLM-B00XOYIXIO	https://www.amazon.fr	Fourni par l'école
	Servomoteur	4,75 €	1	4,75 €	KAPshop	/	http://www.kapshop.com	Fourni par l'école
	Potentiomètre 10K	0,25 €	2	0,50 €	Gotronic	04601	https://www.gotronic.com	Fourni par l'école
	Led rouge	0,15 €	2	0,30 €	Gotronic	03030	https://www.gotronic.com	Fourni par l'école
	Résistance 220ohms Pack de 10	0,26 €	1	0,26 €	RS Components	135-500	https://bepfr.rs-components.com	Fourni par l'école
Sous-total		33,73 €						
Raspberry	Raspberry Pi 3 B+	63,70 €	1	63,70 €	RS Components	174-7510	https://bepfr.rs-components.com	Fourni par l'école
	Pi caméra V2	23,09 €	1		RS Components	913-2664	https://bepfr.rs-components.com	Fourni par l'école
Sous-total		86,79 €						
Communication	Câble mâle-mâle	2,41 €	1	2,41 €	RS Components	791-6463	https://bepfr.rs-components.com	Fourni par l'école
	Câble mâle-femelle	2,41 €	2	4,82 €	RS Components	791-6454	https://bepfr.rs-components.com	Fourni par l'école
	Jumper Cable Kit Fil	2,50 €	1	2,50 €	Amazon	14731	https://www.amazon.fr	Fourni par l'école
Sous-total		7,32 €						
Montage pratique	Bois	20 €	/	/	Récupération	/	/	/
	Impression 3D	(€/g)	(m)	(g)	Nombre d'essai	Prix total		
	Barrière	0,035	0,8	6	2	0,420 €		Fourni par l'école
	Support haut du servo	0,035	0,49	3	1	0,105 €		Fourni par l'école
	Support bas du servo	0,035	1,2	8	1	0,280 €		Fourni par l'école
Sous-total		0,81 €						
Total général		128,65 €						

En ce qui concerne l'impression 3D, on prend, ici, seulement le prix de la bobine en compte (soit 35 € du kg). Mais il faudrait également prendre en compte l'amortissement de la machine ainsi que sa consommation électrique. Données que nous ne possédons pas.

3. Time line final

Comme on peut remarqué, le time line a un peu bougé entre le début et la fin du projet. Et notamment parce que certaine tâche on prit plus de temps qu'espéré. Nous avons également ajouté la partie « Rédaction du rapport » qui est tout de même une partie importante de ce projet.



4. Partie 1 : Arduino

1. Composants :

Comme vu précédemment, le microcontrôleur Arduino va contrôler la barrière, l'écran et le clignotement d'une lampe. La barrière est réalisée avec un servomoteur, l'écran avec un LCD et lampe avec une led rouge.

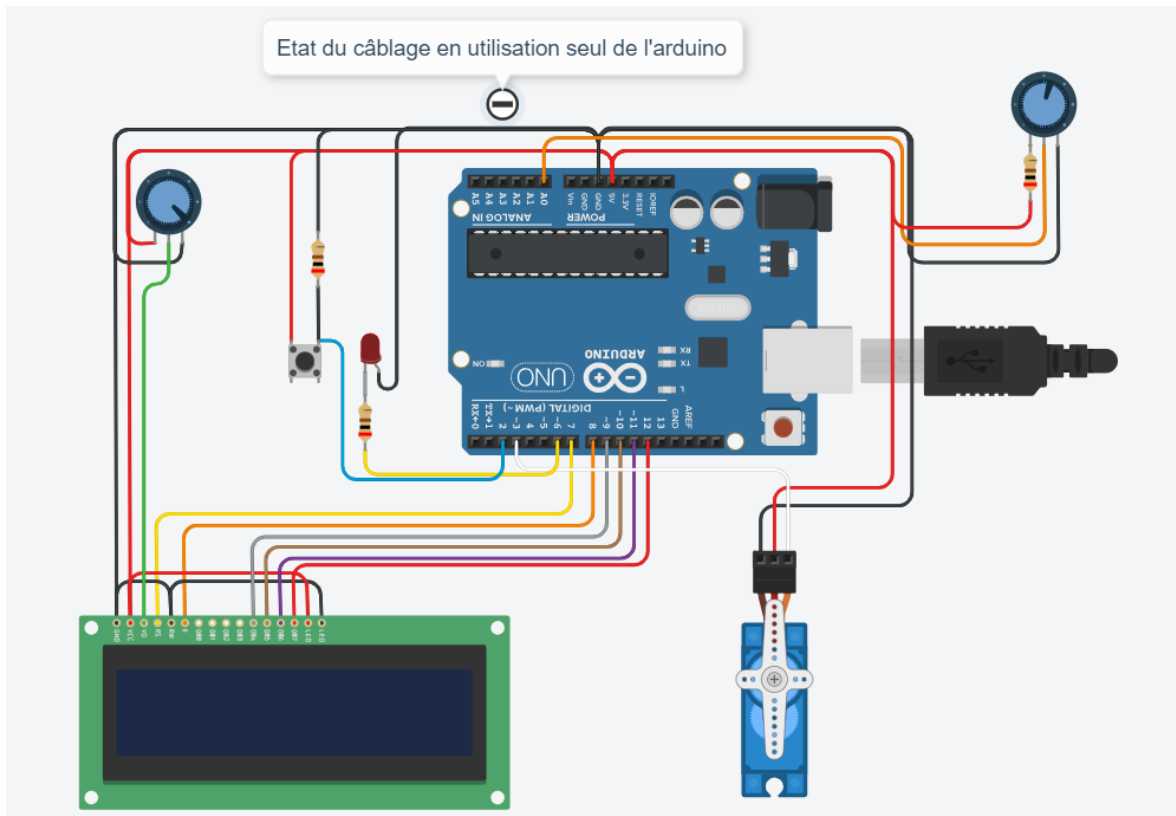


Image 4-1 - Schéma de câblage de l'arduino

Nous pouvons voir sur ce schéma tout le circuit de la partie Arduino. Il faut prêter attention aux pins que l'on ne peut pas utiliser sur le microcontrôleur. Les pins 0 et 1 sont deux pins utilisées par le protocole RX et TX, utilisées comme communication série ainsi que pour l'écriture sur l'interface moniteur série. Ensuite c'est la pin 13 à laquelle il faut prêter attention car elle contrôle en même temps la petite LED « L » se trouvant sur le microcontrôleur. Etant donné que nous disposons d'assez de pin digital, nous utilisons cette LED pour s'assurer que le « void loop () » fonctionne bien.

En bas à droite, nous retrouvons le servomoteur qui comporte 3 fils. Un noir pour la masse, un rouge pour l'alimentation 5 volts et un blanc/brun clair pour la commande. Cette commande se fait sur la pin 3 (PWM). En effet, pour commander le servomoteur, il faut lui envoyer des impulsions (d'où l'utilité d'une pin PWM) et c'est le temps de durée de pulsation qui déterminera l'angle que le servomoteur doit prendre (voir image suivante). Une fois cette angle réglé, il s'efforcera au mieux possible de garder cette position jusqu'à la commande suivante.



Image 4-2 - Micro servomoteur [1]

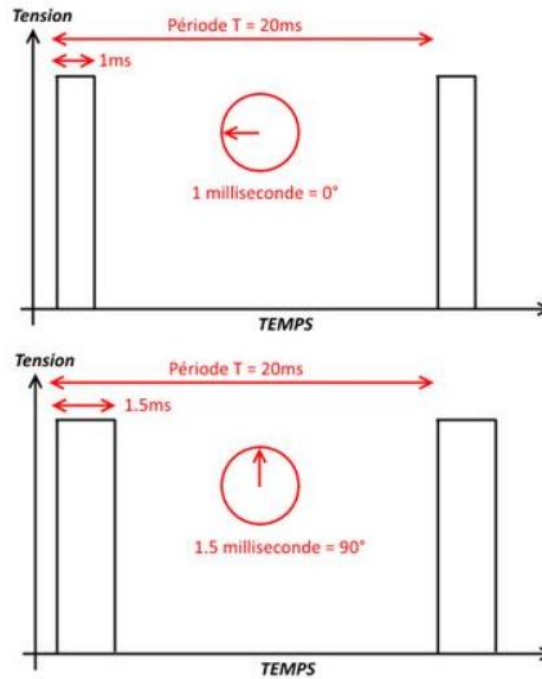


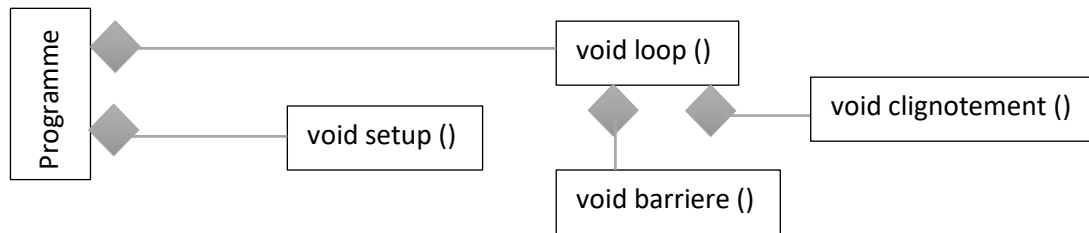
Image 4-3 - Graphe temporel des impulsions PWM [2]

Ensuite, on retrouve l'écran LCD (liquid crystal display) dont le branchement peut rapidement devenir un casse-tête si l'on n'y met pas un peu de rigueur. On retrouve 2 fils d'alimentation 5 volts (rouge) et 3 fils de masse (noir). Les 6 autres pins sont reliés à l'Arduino sur les pins 7 à 12. 7 et 8 pour la communication et les 4 autres pour les données. Il y a aussi une pin (n°3 en vert) provenant d'un potentiomètre pour régler le contraste de l'écran.

Les trois derniers éléments sont plus simples. On retrouve un bouton poussoir simulant la communication avec le raspberry qui sera, bien entendu, remplacé par un fil de communication. Ensuite une LED rouge clignotant à une certaine fréquence lorsque la barrière est en mouvement ou en l'air, et finalement, un potentiomètre branché sur la pin analogique A0 qui fournira une fréquence compris entre 1 et 6 Hz pour le clignotement de la LED.

2. Programme :

La structure même du code de l'arduino est très simple.



En regardant ce diagramme, on peut facilement comprendre que le programme fait appel au setup et au loop, et le loop fait lui-même appel au sous-programme du clignotement de la LED et au sous-programme du mouvement de la barrière.

Un peu plus loin, on retrouve un ordinogramme qui permet de comprendre facilement comment il tourne mais il n'est pas fort complet. En fait, avant même le void setup, on retrouve l'insertion des bibliothèques nécessaires (pour le servomoteur ainsi que pour l'écran LCD) suivi ensuite de toutes les déclarations de variables bool, int et float (pour les temporisations) et les déclarations de numéro de pin. Pour terminer, on déclare le nom du servomoteur et les 6 pins utiles à l'écran.

Ensuite vient le void setup () dans lequel on définit chaque pin en entrée ou sortie, puis on attache un pin au servomoteur et on le positionne. Et pour finir on paramètre la taille du LCD avant d'attendre pour une durée de 2 secondes. Ce temps d'attente n'est pas une obligation mais il permet de s'assurer que les mouvements engendrés par le void setup soient bien terminés.

Et enfin vient le void loop qui, on le rappelle, est l'endroit dans lequel le programme va tourner en permanence. Dans celui-ci, on va tout d'abord aller lire la valeur de l'entrée analogique provenant du potentiomètre et la convertir grâce à un tableau en fréquence de 1 à 6 Hz. Ensuite, cette fréquence va donner un temps entre chaque clignotement de la LED. Avant d'utiliser les sous-programmes, il nous manque encore la communication avec le raspberry. On lit la valeur et on vérifie qu'elle reste active plus d'une seconde pour éliminer les petites imperfections que pourrait engendrer la reconnaissance de plaque.

Grâce à cette vérification, nous pouvons enfin accéder aux deux sous-programmes dont le premier est le clignotement de la LED. Dans celui-ci, on vérifie le temps d'écoulement depuis le dernier changement d'état de la LED. Si celui-ci est supérieur ou égal à ce que l'on souhaite (c'est-à-dire le temps induit par la fréquence), alors on lit l'état actuel de la LED et on l'inverse. De cette manière, on crée un clignotement.

Le sous-programme suivant (c'est-à-dire le mouvement de la barrière) écrit dans un premier temps sur l'écran LCD et se divise ensuite en 4 parties conditionnelles :

- La première vérifie que la barrière n'a pas encore bougé et la fait monter de 1 degré jusqu'à un angle de 89°.
- La deuxième vérifie que la barrière est bien en haut et qu'il n'y a plus de plaque détectée. Ensuite, elle attend 2 secondes.
- La troisième vérifie que la barrière est toujours en haut qu'elle a bien attendu pour la faire redescendre à l'horizontal.
- Et enfin, la dernière vérifie que toutes les autres actions ont bien été effectuées pour arrêter le clignotement, effacer l'écran et permettre le réarmement du futur mouvement.

Le code a été programmé de façon à ne jamais rester bloqué à un endroit. Dans le cas où la barrière est montée de 1 degré, le programme fait d'abord tout le tour des autres conditions avant de revenir à celle de la montée de la barrière.

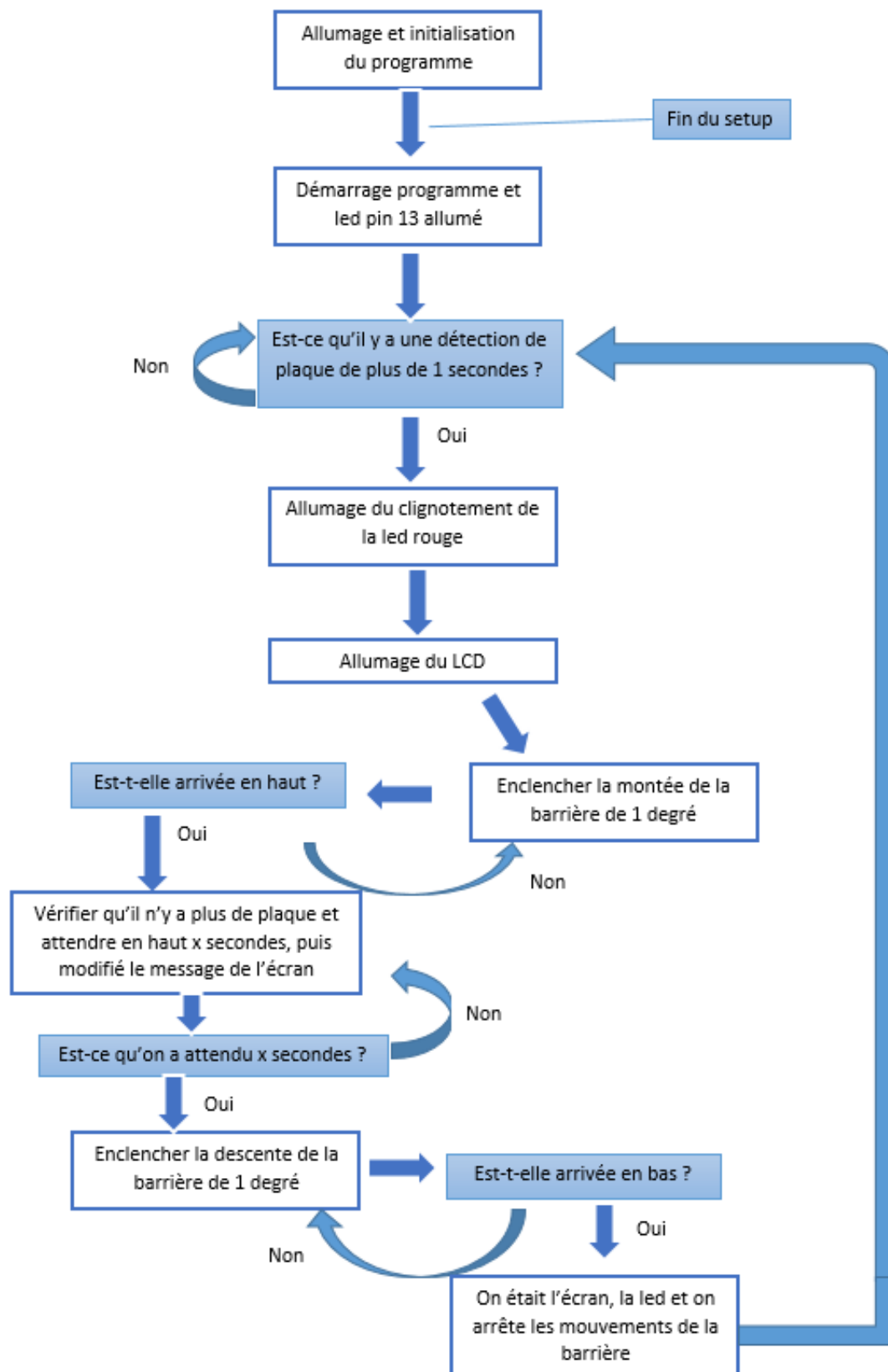


Figure 4-1 - Ordinoigramme du code arduino

Bien entendu, ce genre de programme ne s'écrit pas en une seule fois et nécessite des tests et du temps. Par exemple, les premières ébauches de code ne ressemblaient pas à celle que nous avons maintenant. On peut notamment voir les tests du servomoteur.

https://github.com/GPro97/Projet_gestion_camion/commit/5b1bba8dc9cf58819158919282b81bce4895565b

Ensuite est seulement venu le contrôle de vitesse du servo.

https://github.com/GPro97/Projet_gestion_camion/commit/5b1bba8dc9cf58819158919282b81bce4895565b

Tout cela pour en arriver à un programme final, complet et sans erreur.

3. Modélisation 3D

Pour donner un peu plus de réalisme au projet, 3 dessins 3D ont été réalisés puis imprimés avec une imprimante 3D pour créer un support au servomoteur et une barrière à lui fixer.

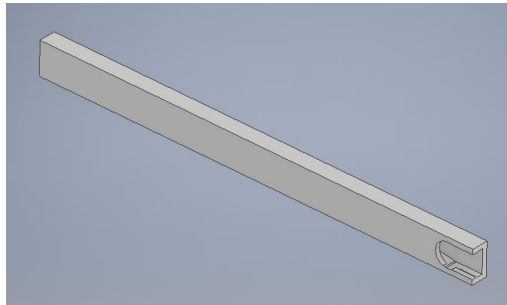


Image 4-4 - Barrière

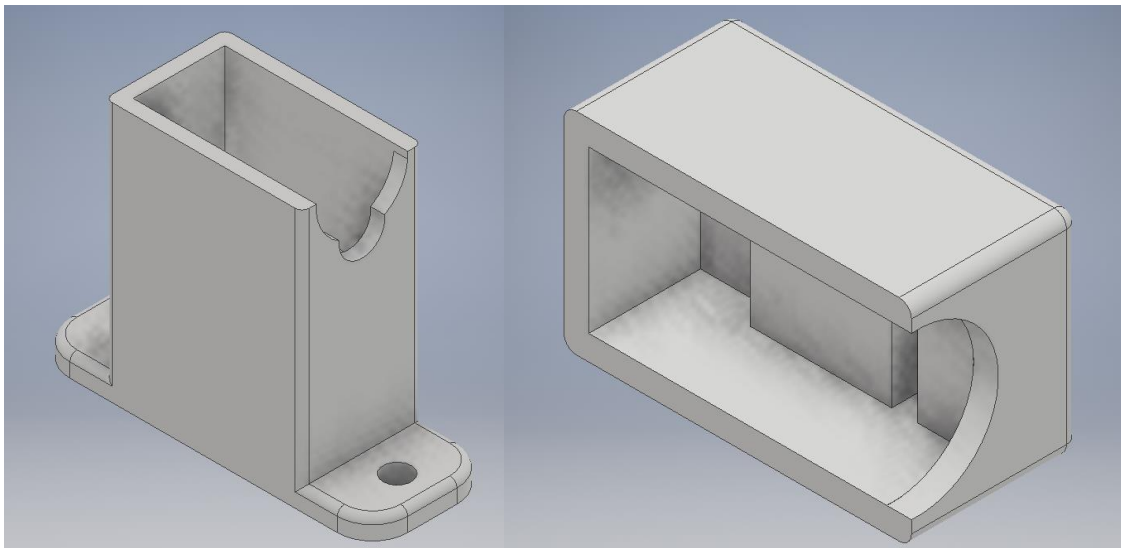


Image 4-5 - Supports du servomoteur

https://github.com/GPro97/Projet_gestion_camion/commit/5b1bba8dc9cf58819158919282b81bce4895565b

5. Partie Raspberry (reconnaissance)

Notre projet vise à détecter les plaques d'immatriculations. Nous avons donc rejeté l'idée du traitement d'images car cela ne convient pas spécialement pour cette tâche. C'est principalement utilisé pour les couleurs, les textures, etc. Nous sommes donc partis sur du Deep Learning.

L'apprentissage profond (Deep Learning) fait partie d'une famille de méthodes d'apprentissage automatique fondées sur l'apprentissage de modèles de données. Une observation (une image) peut être représentée de différentes façons par un vecteur de données, notamment en fonction de :

- L'intensité des pixels dont elle est constituée ;
- Ses différentes arêtes ;
- Ses différentes régions, aux formes particulières.

Cette méthode est idéale car on cherche une région dans l'image qui est plaque d'immatriculation.

Notre cas d'utilisation spécifique est la détection d'objets. On peut donc utiliser Tensorflow à différents niveaux :

Niveau 1 : Utilisez un modèle pré-entraîné et l'appliquez-le directement sur nos données pour reconnaître les plaques. Cette méthode est la moins complexe et la plus rapide car nous n'avons pas besoin de construire ou d'entraîner un modèle.

Niveau 2 : Entraîner un modèle sur nos propres données, afin que le modèle puisse apprendre et connaître précisément les plaques. Ce niveau nécessite d'entraîner un modèle sur les données que nous avez labellisées au préalable. Cela prend plus de temps mais apporte des résultats plus pertinents car le modèle reconnaîtra les références exactes des objets choisis.

Niveau 3 : Créez notre propre modèle à partir de zéro. Dans ce niveau, nous n'entraînons pas seulement un modèle, nous le construisons dès le début. Il peut s'agir de l'ajustement de paramètres sur un modèle existant, ainsi que du développement d'une nouvelle architecture de réseau neuronal. Ce niveau est plus dédié à la recherche, car il s'agit d'une approche qui prend beaucoup de temps.

Dans le cadre du cours, nous allons donc partir sur le deuxième niveau.

1. Installation de TensorFlow

En premier lieu, on met à jour notre Raspberry pi. Cela est vivement conseillé avant de travailler avec divers programmes.

```
sudo apt-get update
```

```
sudo apt-get dist-upgrade
```

Maintenant que le Raspberry est à jour, nous allons commencer par installer le logiciel de machine Learning TensorFlow.

TensorFlow est une bibliothèque de logiciels open source pour le calcul numérique à l'aide de graphiques de flux de données. Il a été développé à l'origine par Google Brain Team au sein de l'organisation de recherche Machine Intelligence de Google pour l'apprentissage par la machine et la recherche sur les réseaux neuronaux profonds, mais le système est suffisamment général pour pouvoir s'appliquer à une grande variété d'autres domaines également.

TensorFlow est multiplateforme. Il fonctionne sur presque tout : les processeurs graphiques et les processeurs, y compris les plates-formes mobiles et intégrées, et même les unités de traitement de tenseurs (TPU), qui sont du matériel spécialisé permettant d'effectuer des calculs de tenseurs.

https://github.com/lhelontra/tensorflow-on-arm/releases/download/v1.13.1/tensorflow-1.13.1-cp35-none-linux_armv7l.whl

TensorFlow a aussi besoin d'une librairie LibAtlas pour fonctionner.

```
sudo apt-get install libatlas-base-dev
```

```
sudo pip3 install pillow lxml jupyter matplotlib cython
```

```
sudo apt-get install python-tk
```

Maintenant, on peut lancer l'installation de TensorFlow

```
sudo pip3 install « chemin du fichier whl TensorFlow »
```

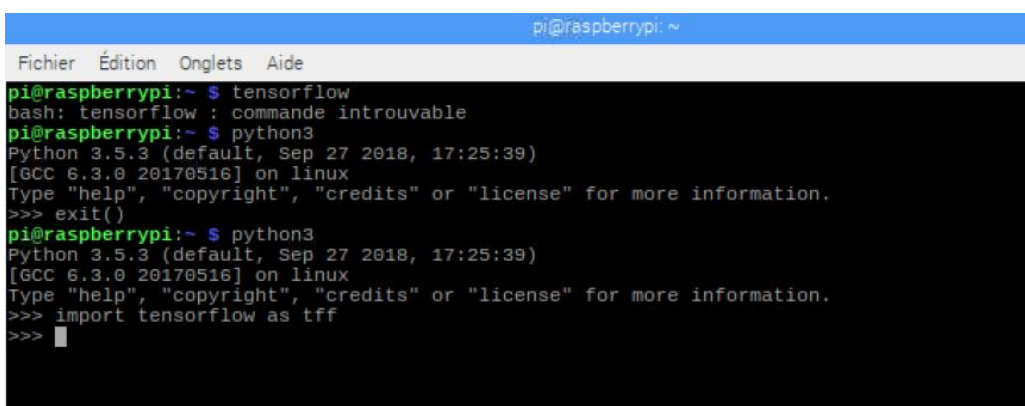
Ayant la version 3.5.3 de python, nous avons pu remarquer que la version 1.13.1 de TensorFlow n'était pas compatible. Il a donc fallu supprimer cette version de TensorFlow. Divers forums ont confirmé ce résultat. A noter que changer la version de python3 déjà installé dans le raspberry n'aurait rien changé car les versions 3.6 sont instables et les versions 3.7 ne prennent pas encore en charge TensorFlow.

Par la suite nous avons installé une version plus ancienne.

https://github.com/lhelontra/tensorflow-on-arm/releases/download/v1.8.0/tensorflow-1.8.0-cp35-none-linux_armv7l.whl

En désinstallant TensorFlow 1.13.1 et en installant l'ancienne version 1.8.0, nous avons subi bons nombres de problèmes de compatibilité de certaines dépendances encore présente de la version 1.13.1. Par question de facilité, nous avons donc réinitialisé le Raspberry. Nous en avons profité pour installer Raspbian Stretch sur une carte SD de plus grande capacité (16Gb).

Après avoir réinstallé la version 1.8.0. Nous l'avons testé dans un script python3. On peut voir que la fonction import fonctionne, ce qui indique que TensorFlow est bien installé.



```
pi@raspberrypi: ~  
Fichier  Édition  Onglets  Aide  
pi@raspberrypi:~ $ tensorflow  
bash: tensorflow : commande introuvable  
pi@raspberrypi:~ $ python3  
Python 3.5.3 (default, Sep 27 2018, 17:25:39)  
[GCC 6.3.0 20170516] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> exit()  
pi@raspberrypi:~ $ python3  
Python 3.5.3 (default, Sep 27 2018, 17:25:39)  
[GCC 6.3.0 20170516] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import tensorflow as tff  
>>>
```

Image 5-1 - Test de TensorFlow sur le Raspberry

2. Installation d'OpenCV

Maintenant que le TensorFlow est installé, il faut installer une bibliothèque graphique spécialisé dans le traitement d'images en temps réel. Nous allons utiliser OpenCV, c'est la bibliothèque de référence pour la vision par ordinateur.

OpenCV (Open Source Computer Vision) est une bibliothèque proposant un ensemble de plus de 2500 algorithmes de vision par ordinateur, accessibles au travers d'API pour les langages C, C++, et Python. Elle est distribuée pour les plateformes Windows, GNU/Linux, Android et MacOS.

Initialement écrite en C il y a 10 ans par des chercheurs de la société Intel, OpenCV est aujourd'hui développée, maintenue, documentée et utilisée par une communauté de plus de 40 000 membres actifs.

Avant de l'installer, nous avons besoin de différentes dépendances.

```
sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
sudo apt-get install libxvidcore-dev libx264-dev
sudo apt-get install qt4-dev-tools
```

Maintenant, nous pouvons installer OpenCV :

```
pip3 install opencv-python
```

3. Installation de Protobuf

L'API (application programming interface) de détection d'objets TensorFlow utilise Protobuf, un package qui implémente le format de données. Protocol Buffers est un format de sérialisation avec un langage de description d'interface développé par Google. L'implémentation d'origine publiée par Google pour C++, Java et Python est disponible sous une licence libre. Malheureusement, il n'existe actuellement aucun moyen facile d'installer Protobuf sur le Raspberry Pi. Nous devons le compiler nous-mêmes à partir des sources, puis l'installer.

En premier lieu, on installe les packages pour compiler Protobuf :

```
sudo apt-get install autoconf automake libtool curl
```

Ensuite, on le télécharge de la source :

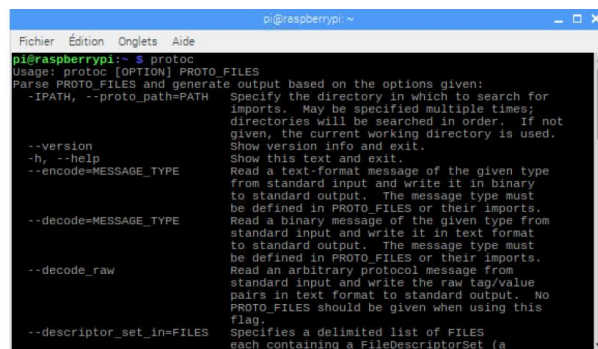
```
wget https://github.com/google/protobuf/releases/download/v3.5.1/protobuf-all-3.5.1.tar.gz
```

On entre ensuite dans la directory pour exécuter divers commandes :

```
./configure
make check
sudo make install
cd python
export LD_LIBRARY_PATH=./src/.libs
python3 setup.py build --cpp_implementation
```

```
python3 setup.py test --cpp_implementation
sudo python3 setup.py install --cpp_implementation
export PROTOCOL_BUFFERS_PYTHON_IMPLEMENTATION=cpp
export PROTOCOL_BUFFERS_PYTHON_IMPLEMENTATION_VERSION=3
sudo ldconfig
```

Avec la faible puissance du processeur du Raspberry, il aura fallu approximativement 3 heures pour exécuter toutes les commandes. Pour vérifier que Protobuf est bien installé, il suffit de vérifier si la commande `protoc` nous amène bien l'aide. Cela est le cas, nous pouvons passer à la nouvelle étape.



```

pi@raspberrypi:~$ protoc
Usage: protoc [OPTION] PROTO_FILES
Parse PROTO_FILES and generate output based on the options given:
  -IPATH, --proto_path=PATH  Specify the directory in which to search for
                              imports. May be specified multiple times;
                              directories will be searched in order. If not
                              given, the current working directory is used.
                              Show version info and exit.
  --version                  Show this text and exit.
  -h, --help                 Read a text-format message of the given type
                              from standard input and write it in binary
                              to standard output. The message type must
                              be defined in PROTO_FILES or their imports.
  --encode=MESSAGE_TYPE     Read a binary message of the given type from
                              standard input and write it in text format
                              to standard output. The message type must
                              be defined in PROTO_FILES or their imports.
  --decode=MESSAGE_TYPE     Read an arbitrary protocol message from
                              standard input and write the raw tag/value
                              pairs in text format to standard output. No
                              PROTO_FILES should be given when using this
                              flag.
  --descriptor_set_in=FILES  Specifies a delimited list of FILES
                              each containing a FileDescriptorSet (a

```

Image 5-2 - Test de Protobuf sur le Raspberry

4. Installation des modèles TensorFlow

Vu que nous allons entrainer un modèle à partir d'un qui est déjà existant, il va nous falloir télécharger les différents modèles TensorFlow

<https://github.com/tensorflow/models.git>

La partie qui nous intéresse est l'« Object_détection ».

5. PYTHONPATH

Ensuite, nous devons modifier le PYTHONPATH. Le pythonpath indique à python quels dossiers il doit prendre en compte pour sa recherche de modules. Il va devoir pointer sur certains répertoires de TensorFlow que nous venons de télécharger. Nous allons donc modifier le fichier `~/.bashrc` pour que PYTHONPATH soit défini chaque fois que nous ouvrons un terminal. On y ajoute donc simplement:

```
export
PYTHONPATH=$PYTHONPATH:/home/pi/tensorflow1/models/research:/home/pi/tensorflow1/models/research/slim
```

A noter que sur le Raspberry, notre dossier `models` se trouve dans un répertoire préalablement créé `tensorflow1`.

6. Premier test de reconnaissance :

Maintenant nous allons télécharger un programme .py qui va utiliser la Picamera (ou une caméra usb) ainsi qu'un modèle pour nous permettre de réaliser de la reconnaissance d'objet.

https://github.com/GPro97/Projet_gestion_camion/commit/9bae59446e5cc9bba70614b4980a79f71368cdca

Comme on peut le voir à la ligne 55, ce programme utilise le modèle `ssd_mobilenet_v2_coco_2018_03_29` Il va donc falloir le télécharger.

http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v2_coco_2018_03_29.tar.gz

Une fois extrait et placé dans le même directory, on peut tester le programme.

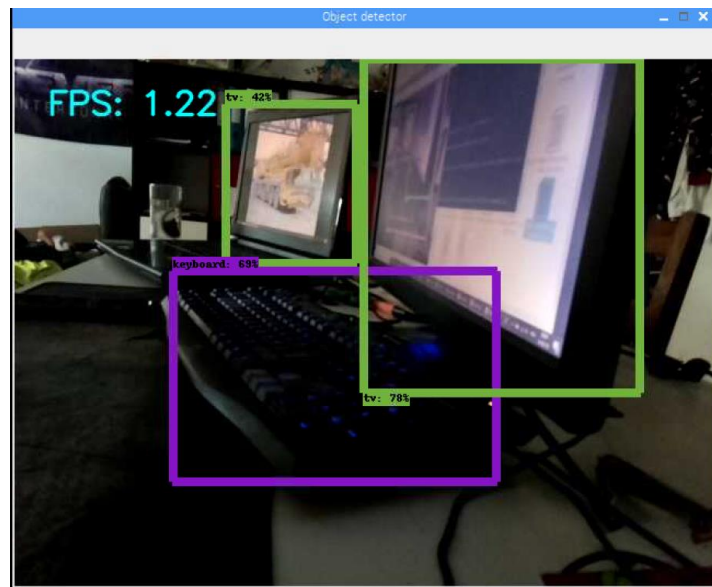


Image 5-3 - Premier test sur Raspberry

On s'aperçoit que le programme met environ 30 secondes pour se lancer. Le temps de charger le modèle ainsi que les autres dépendances. Il fonctionne correctement avec néanmoins une grande latence. Nous tournons aux alentours de 1.2 images par seconde. Un second test a été réalisé en diminuant la qualité de la caméra à 480p (720p lors du premier test) mais cela n'a rien changé au niveau des images par seconde. Une hypothèse a été émise : Le processeur du Raspberry n'est pas assez puissant et donc n'a pas le temps de scanner l'image à trop haute vitesse.

Maintenant que tout est en place, nous allons nous attaquer à la grosse partie de ce projet, entrainer un modèle pour qu'il nous permette de détecter les plaques d'immatriculation qui passeront devant la Picamera.

Nous allons procéder par différentes étapes :

- Collecter des photos pour l'entraînement
- Transformer les photos .jpeg en fichiers .xml
- Transformer les fichiers .xml en .csv
- Transformer les fichiers .csv en .TFRecord
- Entraîner notre modèle

7. Collecte de photos

Pour entrainer notre modèle nous allons devoir collecter une série de photos de véhicules en situations réelles de véhicule où l'on peut voir la plaque d'immatriculation.



Image 5-4 - Exemple de photo prise pour l'entrainement

Pour un entrainement correct il faut au minimum 100 images par classe (objet à détecter) mais idéalement 500 images ou plus encore. Pour notre projet nous avons utilisé 60 photos.

NB : Les photos ont été prises en 4k ($4\,096 \times 2\,160$) ce qui donne 8 847 360 pixels. Durant l'entrainement plus tard, nous avons observé que chaque étape prenait 120 secondes ce qui est beaucoup trop important vu le grand nombre à réaliser. Nous avons donc diminué la qualité à 480p (1280×720) ce qui donne 921 600 pixels soit 10 fois moins.

8. LabelImg

Toutes nos photos sont prêtes. Nous allons devoir nous attaquer au programme LabelImg

<https://github.com/tzutalin/labelImg.git>

Ce programme nous permet de dessiner des encadrements de façon aisée autour des éléments qui nous intéressent. On peut appeler les encadrements comme on le souhaite pour différentes catégories d'objet par exemple (dans notre cas on a un seul type nommé « licenceplate »). Par la suite, LabelMap créé un fichier .xml du même nom que le fichier .jpg auquel il se rapporte.

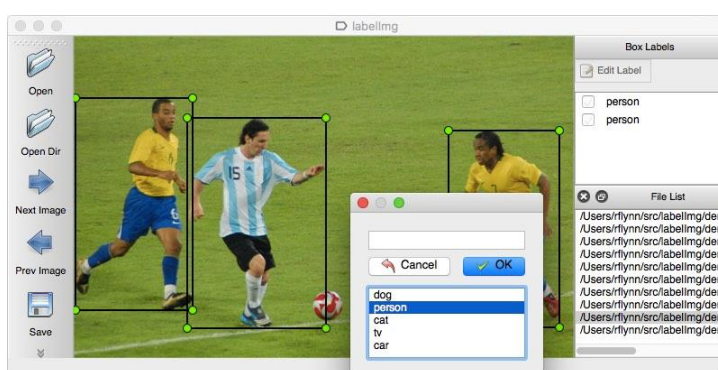


Image 5-5 - Logiciel LabelImg

Dans ce fichier .xml, on retrouve différents éléments :

- La taille de l'image
- Le fichier .jpg auquel il se rapporte
- Le nom de la catégorie de chaque encadrement
- Les positions (xmin,ymin et xmax,ymax) de chaque encadrement

Une fois que tous les fichiers .xml sont réalisés, on va les séparer en 2 fichiers. Un fichier « train » qui comportera 90% des images et de leur fichier .xml associé. Et le second « test » comportera le reste (10%).

9. Conversion du format .xml en .csv

Pour pouvoir créer nos fichiers TFRecord, il faut d'abord passer sous un format .csv.

Un fichier CSV est un fichier texte, par opposition aux formats dits « binaires ». Chaque ligne du texte correspond à une ligne du tableau et les virgules correspondent aux séparations entre les colonnes. Les portions de texte séparées par une virgule correspondent ainsi aux contenus des cellules du tableau.

Il existe un programme xml_to_csv.py prévu à cet effet :

https://github.com/datitran/raccoon_dataset/blob/master/xml_to_csv.py

Il faut appliquer quelques changements pour pointer dans les bons dossiers.

https://github.com/GPro97/Projet_gestion_camion/commit/fed5f08fe0103ca523ff79ec0fe85702b3f18821

Ensuite on peut exécuter le fichier : `python3 xml_to_csv.py`

On obtient:

https://github.com/GPro97/Projet_gestion_camion/commit/393379d2bd82f0926299fc8610a063c2937966a0

10. Conversion du format .csv en .TFRecord

Comme le point précédent un programme est prévu à cet effet (`generate_tfrecord`). La seule modification à faire se trouve au niveau des classes à détecter.

https://github.com/GPro97/Projet_gestion_camion/commit/4888c238af3a3033873ac5db94b4647dc66e9b84

Un format TFRecord stock les données en une chaîne de binaire. Cela facilite l'utilisation des données car un programme gère beaucoup mieux une suite d'informations binaires que de changer de fichier à chaque fois.

On peut l'exécuter de la manière suivante :

```
python3 generate_tfrecord.py --csv_input=data/train_labels.csv --output_path=data/train.record --image_dir=images/
```

```
python3 generate_tfrecord.py --csv_input=data/test_labels.csv --output_path=data/test.record --image_dir=images/
```

Voici les résultats :

https://github.com/GPro97/Projet_gestion_camion/commit/f64c59f585ad08c738fd5c9e7b408bd0dac36e2

Le `train.record` ne se trouve pas sur GitHub car il fait 105Mo ce qui dépasse la limite autorisée par git.

11. Fichier de configuration du modèle :

Maintenant il nous reste à paramétrer le fichier de configuration du modèle que nous allons entraîner. Pour l'entraînement, nous nous baserons sur le modèle utilisé lors du premier essai (ssd_mobilenet_v2_coco_2018_03_29).

On crée un répertoire « training » se trouvant dans le même dossier que le modèle. Dans celui-ci, on y ajoute le fichier de configuration présent dans le fichier du modèle. De base, il est appelé pipeline.config mais par facilité il sera renommé : ssd_mobilenet_v2_coco_2018_03_29.config

Il y a quelques éléments à codifier, des annotations sont présente sur le git.

https://github.com/GPro97/Projet_gestion_camion/commit/07b22c779fdd414eeb8ca3cbc6fa2cf96ea5e328

NB : on peut voir que le nombre d'images utilisés ainsi que le nombre d'étapes pour l'entraînement ne sont pas cohérent avec les valeurs de base. Cela signifiera que notre modèle sera moins bon que le modèle sur lequel on se base.

Dans le dossier training, il nous reste à ajouter un fichier object-detection.pbtxt

Celui-ci indique quelle id correspond à quel objet. Dans notre cas, c'est très simple car nous n'avons que « licenceplate » qui prendra l'id 1.

12. Entraînement du modèle :

Il est maintenant temps d'entraîner le modèle. Pour ce faire il y a un programme train.py prévu à cet effet. De base il se trouve dans l'onglet legacy mais par facilité, on le déplace au même niveau que training et que le modèle (ssd_mobilenet_v2_coco_2018_03_29).

```
python3 train.py --logtostderr --train_dir=training/ --pipeline_config_path=training/
ssd_mobilenet_v2_coco_2018_03_29.config
```

Ici, le Raspberry commence l'entraînement. Cependant à partir de quelques étapes d'apprentissage ou avant, le Raspberry plante ou abandonne le processus. Cela est due au manque de mémoire (erreur : « std: :bad_alloc ») et à la puissance du processeur. Après plusieurs essais, nous retrouvons toujours le même diagnostic.

```
pi@raspberrypi: ~/tensorflow1/models/research/object_detection
Fichier Édition Onglets Aide
WARNING:root:Variable [FeatureExtractor/MobilenetV2/layer_19_2_Conv2d_4_3x3_s2_256_depthwise/depthwise_weights/RMSProp_1] is not available in checkpoint
WARNING:root:Variable [FeatureExtractor/MobilenetV2/layer_19_2_Conv2d_5_3x3_s2_128/BatchNorm/beta/ExponentialMovingAverage] is not available in checkpoint
WARNING:root:Variable [FeatureExtractor/MobilenetV2/layer_19_2_Conv2d_5_3x3_s2_128/BatchNorm/beta/RMSProp] is not available in checkpoint
WARNING:root:Variable [FeatureExtractor/MobilenetV2/layer_19_2_Conv2d_5_3x3_s2_128/BatchNorm/beta/RMSProp_1] is not available in checkpoint
WARNING:root:Variable [FeatureExtractor/MobilenetV2/layer_19_2_Conv2d_5_3x3_s2_128/BatchNorm/gamma/ExponentialMovingAverage] is not available in checkpoint
WARNING:root:Variable [FeatureExtractor/MobilenetV2/layer_19_2_Conv2d_5_3x3_s2_128/BatchNorm/gamma/RMSProp] is not available in checkpoint
WARNING:root:Variable [FeatureExtractor/MobilenetV2/layer_19_2_Conv2d_5_3x3_s2_128/BatchNorm/gamma/RMSProp_1] is not available in checkpoint
WARNING:root:Variable [FeatureExtractor/MobilenetV2/layer_19_2_Conv2d_5_3x3_s2_128/weights/ExponentialMovingAverage] is not available in checkpoint
WARNING:root:Variable [FeatureExtractor/MobilenetV2/layer_19_2_Conv2d_5_3x3_s2_128/weights/RMSProp] is not available in checkpoint
WARNING:root:Variable [FeatureExtractor/MobilenetV2/layer_19_2_Conv2d_5_3x3_s2_128/weights/RMSProp_1] is not available in checkpoint
WARNING:root:Variable [FeatureExtractor/MobilenetV2/layer_19_2_Conv2d_5_3x3_s2_128_depthwise/BatchNorm/beta/ExponentialMovingAverage] is not available in checkpoint
WARNING:root:Variable [FeatureExtractor/MobilenetV2/layer_19_2_Conv2d_5_3x3_s2_128_depthwise/BatchNorm/beta/RMSProp] is not available in checkpoint
WARNING:root:Variable [FeatureExtractor/MobilenetV2/layer_19_2_Conv2d_5_3x3_s2_128_depthwise/BatchNorm/beta/RMSProp_1] is not available in checkpoint
WARNING:root:Variable [FeatureExtractor/MobilenetV2/layer_19_2_Conv2d_5_3x3_s2_128_depthwise/BatchNorm/gamma/ExponentialMovingAverage] is not available in checkpoint
WARNING:root:Variable [FeatureExtractor/MobilenetV2/layer_19_2_Conv2d_5_3x3_s2_128_depthwise/BatchNorm/gamma/RMSProp] is not available in checkpoint
WARNING:root:Variable [FeatureExtractor/MobilenetV2/layer_19_2_Conv2d_5_3x3_s2_128_depthwise/BatchNorm/gamma/RMSProp_1] is not available in checkpoint
WARNING:root:Variable [FeatureExtractor/MobilenetV2/layer_19_2_Conv2d_5_3x3_s2_128_depthwise/depthwise_weights/ExponentialMovingAverage] is not available in checkpoint
WARNING:root:Variable [FeatureExtractor/MobilenetV2/layer_19_2_Conv2d_5_3x3_s2_128_depthwise/depthwise_weights/RMSProp] is not available in checkpoint
WARNING:root:Variable [FeatureExtractor/MobilenetV2/layer_19_2_Conv2d_5_3x3_s2_128_depthwise/depthwise_weights/RMSProp_1] is not available in checkpoint
WARNING:tensorflow:From /home/pi/.local/lib/python3.5/site-packages/tensorflow/contrib/slim/python/slim/learning.py:736: Supervisor.__init__ (from tensorflow.python.training.supervisor) is deprecated and will be removed in a future version.
Instructions for updating:
Please switch to tf.train.MonitoredTrainingSession
WARNING:tensorflow:From /home/pi/.local/lib/python3.5/site-packages/tensorflow/contrib/slim/python/slim/learning.py:736: Supervisor.__init__ (from tensorflow.python.training.supervisor) is deprecated and will be removed in a future version.
Instructions for updating:
Please switch to tf.train.MonitoredTrainingSession
INFO:tensorflow:Restoring parameters from ssd_mobilenet_v2_coco_2018_03_29/model.ckpt
INFO:tensorflow:Restoring parameters from ssd_mobilenet_v2_coco_2018_03_29/model.ckpt
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Starting Session.
INFO:tensorflow:Starting Session.
INFO:tensorflow:Saving checkpoint to path training/model.ckpt
INFO:tensorflow:Saving checkpoint to path training/model.ckpt
INFO:tensorflow:Starting Queues.
INFO:tensorflow:Starting Queues.
terminate called after throwing an instance of 'std::bad_alloc'
terminate called recursively
Abandon
pi@raspberrypi:~/tensorflow1/models/research/object_detection $
```

Image 5-6 - Abandon du processus d'entrainement sur Raspberry

Pour pouvoir contourner le problème, nous avons décidé de réaliser l'entrainement sur un ordinateur possédant Ubuntu (une version allégée d'Ubuntu) comme système d'exploitation. Nous avons donc effectué les mêmes étapes que vu précédemment mais sur l'ordinateur. Nous avons lancé l'entrainement et après environ 20 heures, l'apprentissage c'est terminé correctement comme on peut le voir ci-dessous.

```
INFO:tensorflow:Recording summary at step 2990.
INFO:tensorflow:Recording summary at step 2990.
INFO:tensorflow:global step 2991: loss = 1.5591 (16.827 sec/step)
INFO:tensorflow:global step 2991: loss = 1.5591 (16.827 sec/step)
INFO:tensorflow:global step 2992: loss = 2.1601 (13.944 sec/step)
INFO:tensorflow:global step 2992: loss = 2.1601 (13.944 sec/step)
INFO:tensorflow:global step 2993: loss = 2.9852 (14.948 sec/step)
INFO:tensorflow:global step 2993: loss = 2.9852 (14.948 sec/step)
INFO:tensorflow:global step 2994: loss = 1.9782 (13.247 sec/step)
INFO:tensorflow:global step 2994: loss = 1.9782 (13.247 sec/step)
INFO:tensorflow:global step 2995: loss = 1.1204 (14.720 sec/step)
INFO:tensorflow:global step 2995: loss = 1.1204 (14.720 sec/step)
INFO:tensorflow:global step 2996: loss = 1.5973 (13.998 sec/step)
INFO:tensorflow:global step 2996: loss = 1.5973 (13.998 sec/step)
INFO:tensorflow:global step 2997: loss = 1.5222 (13.413 sec/step)
INFO:tensorflow:global step 2997: loss = 1.5222 (13.413 sec/step)
INFO:tensorflow:global step 2998: loss = 1.7433 (14.796 sec/step)
INFO:tensorflow:global step 2998: loss = 1.7433 (14.796 sec/step)
INFO:tensorflow:Saving checkpoint to path training/model.ckpt
INFO:tensorflow:Saving checkpoint to path training/model.ckpt
INFO:tensorflow:global step 2999: loss = 1.6548 (17.943 sec/step)
INFO:tensorflow:global step 2999: loss = 1.6548 (17.943 sec/step)
INFO:tensorflow:Recording summary at step 2999.
INFO:tensorflow:Recording summary at step 2999.
INFO:tensorflow:global step 3000: loss = 1.5576 (14.272 sec/step)
INFO:tensorflow:global step 3000: loss = 1.5576 (14.272 sec/step)
INFO:tensorflow:Stopping Training.
INFO:tensorflow:Stopping Training.
INFO:tensorflow:Finished training! Saving model to disk.
INFO:tensorflow:Finished training! Saving model to disk.
/home/pwprp/.local/lib/python3.5/site-packages/tensorflow/python/summary/writer/writer.py:386: UserWarning: Attempting to use a closed
iter. The operation will be a noop unless the FileWriter is explicitly reopened.
warnings.warn("Attempting to use a closed FileWriter.")
pwprp@pcdev:~/tensorflow/models/research/object_detection$
```

Image 5-7 - Fin de l'apprentissage du modèle sur Ubuntu

On peut voir que le train.py c'est bien arrêté à la 3000ème étape.

L'entrainement terminé, on aperçoit différents éléments ajoutés dans le dossier training. Ils seront utilisés juste après.

Pour obtenir notre modèle final, il ne reste plus qu'à utiliser le programme `export_inference_graph.py` :

```
python3 export_inference_graph.py \
--input_type image_tensor \
--pipeline_config_path training/ssd_mobilenet_v1_pets.config \ #directory du fichier de config
--trained_checkpoint_prefix training/model.ckpt-3000 \ #cela dépend du nombre d'étape réalisé
--output_directory licenceplate_inference_graph #nom du modèle créé
```

Le modèle ne se trouve pas sur GitHub car il est trop volumineux.

On peut maintenant modifier le programme utilisant la Picamera pour y ajouter le nouveau modèle. On obtient:

https://github.com/GPro97/Projet_gestion_camion/commit/9022c1c2871b559961292d36e81eb9dcf0279930

En testant le programme sur Raspberry, pour une raison inconnue, il ne démarre pas. Pour vérifier le bon fonctionnement du modèle, nous avons décidé de l'essayer sur l'ordinateur. On peut voir le résultat ci-dessous :

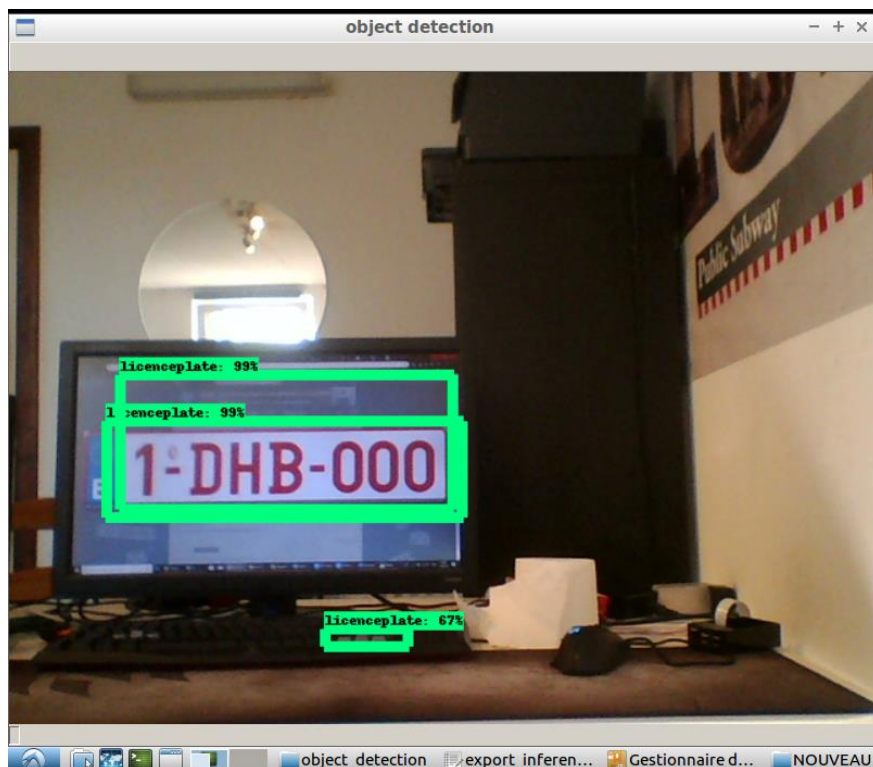


Image 5-8 - Modèle entraîné utilisé sur Ubuntu

Le programme fonctionne mais il a du mal à détecter les plaques quand l'image couvre des surfaces carrées comme un écran ou autres. Cela dit nous considérons qu'il est fonctionnel car il dysfonctionne peu lorsque la plaque se situe sur un véhicule à l'extérieur.

Pour mener à bien notre projet sur Raspberry, vu que notre modèle est fonctionnel mais n'est pas compatible sur Raspberry, nous avons décidé de partir sur un modèle détectant les plaques se trouvant sur GitHub.

https://github.com/GPro97/Projet_gestion_camion/commit/e12c5ea732a488a98f68a8f201118f8cea5bb9824

Nous avons remodifié notre programme utilisant la Picamera. De plus, nous avons ajouté quelques modifications du programme. Quand une plaque (ou plus) est détectée, un bit se met à 1. L'Arduino pourra alors lire le bit pour lever la barrière.

https://github.com/GPro97/Projet_gestion_camion/commit/44e014954b1d14c477c2a74f9651eca66fffbde9e

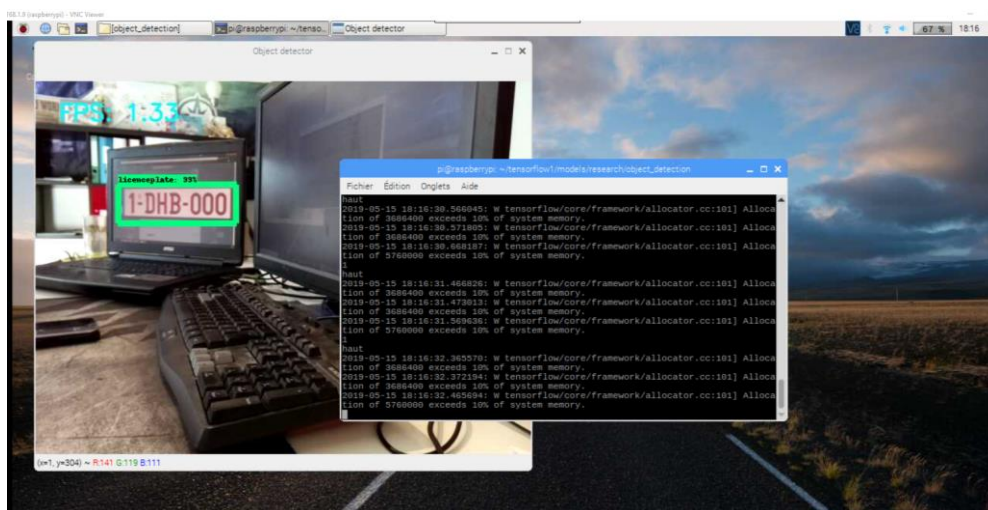


Image 5-9 - Test final de la détection de plaque

Le système fonctionne maintenant correctement. Le bit se met bien à 1 quand on détecte la plaque et les plaques sont détectées correctement. Cela met fin à notre projet de reconnaissance de plaque d'immatriculation.

6. Conclusion :

Nous avons pu constater que le DEEP LEARNING est quelque chose de d'extrêmement complexe et de fort abstrait. Nous avons rencontré de nombreux problèmes comme des incompatibilités, un manque de librairies ou encore un manque de mémoire et puissance processeur.

Nous avons pu voir que le modèle installé d'internet et beaucoup plus performant. Malgré tous les objets présents dans la pièce, il est capable de détecter une plaque alors que notre modèle entraîné dysfonctionne à de nombreuses reprises. On se rend compte de l'importance d'utiliser un grand nombre de photos et d'étapes pour être précis.

Pour améliorer ce système, il faudrait ajouter un capteur de présence en dessous de la barrière, utiliser une deuxième caméra pour le passage dans l'autre sens. Ensuite, ce serait de pouvoir lire les lettres de la plaque et définir des heures d'arrivées et de départs d'une certaine plaque et ainsi pouvoir faire des statistiques.

7. Bibliographie :

<https://pythonprogramming.net/testing-custom-object-detector-tensorflow-object-detection-api-tutorial/?completed=/training-custom-objects-tensorflow-object-detection-api-tutorial/>

<https://www.framboise314.fr/i-a-realisez-un-systeme-de-reconnaissance-dobjets-avec-raspberry-pi/>

<https://github.com/EdjeElectronics/TensorFlow-Object-Detection-on-the-Raspberry-Pi#5-set-up-tensorflow-directory-structure-and-pythonpath-variable>

<https://www.youtube.com/watch?v=npZ-8Nj1YwY>

<https://retroetgeek.com/wp-content/uploads/2018/02/GpioRPIRetroetgeek.png>

<https://artefact.com/fr-fr/news/comment-utiliser-tensorflow-et-ses-ressources-open-source-pour-construire-un-modele-de-reconnaissance-de-produit-sur-mesure/>

<https://opensource.com/article/17/11/intro-tensorflow>

<https://www.arduino.cc/en/Main/Software> (Utilisé dans le cadre de plusieurs recherches de programmation propre à arduino)

<https://www.locoduino.org/> (Celui-ci fût également beaucoup consulté)

Référence :

[1] Voir PDF micro servomoteur

https://github.com/GPro97/Projet_gestion_camion/commit/3dcc848742897b9e47f678c52700cee1d833aa73

[2] <https://www.carnetdumaker.net/articles/controler-un-servomoteur-avec-une-carte-arduino-genuino/>