Rapport Projet IMA

Ironcar

(Projet 9)

Réalisé par:
Equipe 1: Raouak Haroun
Equipe 2: Mathlouthi Maher & Verbeke Rémy
Equipe 3: Mertz Thomas & Cremaschi Guillaume

POLYTECH LILLE

2018/2019

I / Contexte

Inspirée de DIYRobocars, née aux Etats-Unis, la compétition Ironcar a été lancée en France en février 2018, sous l'impulsion de Silex ID (magazine de l'innovation), de l'agence Raoul (agence de communication des startups et entreprises innovantes) et de Grégory Renard, fondateur et Chief Artificial Intelligence Officer de xBrain.

L'objectif de cette compétition est d'éduquer le plus grand nombre aux techniques de machine learning / deep learning à travers un projet de voiture autonome.

Plus concrètement, chaque équipe doit proposer une voiture en modèle réduit contrôlée par une IA ayant pour tâche de suivre le tracé d'un circuit le plus rapidement possible et sans sortir des délimitations, à l'aide d'une seule caméra comme capteur.

II / Objectifs et Cahier des charges

L'objectif principal du projet est de participer à la prochaine course Ironcar et de la remporter.
Afin d'y arriver nous avons défini un cahier des charges.

Le cahier des charges se décompose en 3 grandes parties pour ce premier semestre :

- La récupération du code des participants précédents et l'implémentation de chaque programme et librairie nécessaire à leur fonctionnement.

- La mise en place d'un linux temps réel sur une Raspberry Pi sur laquelle l'IA s'exécutera.

- La commande en vitesse et direction de la voiture pour permettre de la diriger à partir d'une carte raspberry pi.

III / Travail réalisé

Equipe 1 : Mise en place d'un Linux temps réel

Dans notre cas, l'information à traiter arrive périodiquement. Le système doit être capable de l'analyser et de la traiter avant qu'une nouvelle valeur ne se présente, d'où la nécessité de mise en place d'un linux temps réel qui va satisfaire les contraintes temporelles strictes du système.

Matériels et logiciels :

- Un Raspberry Pi 3
- Carte micro SD
- Logiciel Etcher
- RealtimePi OS
- Le programme RT-Tests et l'utilitaire cyclictest

Étapes :

1. Installation avec le logiciel Etcher d'une distribution linux intitulé RealtimePi sur carte micro SD qui est déjà patché avec un noyau temps réel.

2. Implémentation du RT-Tests qui fournit un ensemble de programmes qui testent et mesurent divers composants de comportement en temps réel du noyau, tels que le temps de latence.

3. Mesures avec l'utilitaire cyclictest du temps de latence dans le cas où le processeur est non chargé et dans le cas où le processeur est chargé.

Résultats :

Noyau linux non chargé :	Temps de latence maximum mesuré : 39661 µs
Noyau linux chargé :	Temps de latence maximum mesuré : 14 µs

Le test du noyau intégrant le patch RT démontre un comportement « temps réel » car la variation du temps de réponse en temps réel est bien moindre que dans le cas d'un noyau standard, même si les résultats ne sont pas extraordinaires, certainement à cause de la qualité (moyenne) du noyau de la RPI.

Enfin, les mesures données ci-dessus sont approximatives car mesurées sur quelques dizaines de secondes. Pour avoir un temps de latence correspondant au pire cas, il faudrait faire une mesure pendant des heures voire des jours en stressant le système de toutes les façons possibles en même temps.

Conclusion

Nous voyons que le petit Raspberry Pi 3 a un comportement tout à fait honorable en ce qui concerne les traitements temps réel. Naturellement, il n'est pas question de l'utiliser dans un contexte contrôlant la sécurité des personnes mais il peut très bien servir pour réaliser des tâches impliquant des contraintes temporelles.

Bien entendu, il faut être conscient qu'aucune garantie n'est donnée que les valeurs maximales observées ne seront pas dépassées, et que ce risque doit être pris en considération. Les conséquences d'un tel dépassement doivent être soigneusement étudiées avant de choisir un système temps réel.

Equipe 2 : Mise en place de l'environnement logiciel

Matériels et logiciels :

- PC linux
- Python 3

Étapes :

1. Récupérer le code des vainqueurs de l'année dernière.
2. Préparer l'environnement logiciel pour le fonctionnement du code.
3. Récupérer les sorties du réseau de neurones pour pouvoir les exploiter.

Problèmes rencontrés et solutions apportées :

Récupérer le code des équipes précédentes semble être une tâche simple, mais nous nous sommes heurtés tout de même à quelques difficultés.

Nous avons eu le choix entre deux compétiteurs : Patate 42 et Axionaute.

Le code de Patate42 semblait plus simple à première vue, mais nous avons eu de grosses difficultés à installer une des dépendances requise pour exécuter le code. Nous nous sommes donc tournés vers la solution d'Axionaute.

L'installation des dépendances et librairies ne nous a pas posé de problèmes particuliers, tout ou presque était documenté sur le git.

Axionaute avait l'avantage de détailler un peu plus la procédure à suivre pour l'entraînement du modèle, mais laissait complètement au lecteur les explications concernant la récupération des résultats.

Nous avons dû chercher sur internet comment fonctionnait Keras, la surcouche qui est utilisée avec tensorflow pour exploiter le système de neurones, et numpy, une librairie mathématique.

En regardant le code de Patate42, nous avons remarqué qu'ils avaient fourni cette partie. Nous l'avons donc récupérée et adaptée à notre modèle.

Pour l'entraînement de notre modèle, nous avons utilisé le script train.py d'Axionaute qui permet de générer un nouveau profil de réseau de neurones à partir d'un dataset (nous avons choisi celui d'Axionaute également, qui contient 26 000 images sans compter les images augmentées). On peut noter qu'il s'est fini plus rapidement qu'une utilisation normale : le script a considéré que l'erreur de validation arrêtait de s'améliorer et qu'il était donc inutile de poursuivre.

Après avoir fait des tests avec notre modèle entraîné, nous avons vu que les prédictions étaient généralement fausses (on a seulement 35% de bonnes prédictions lorsque l'on compare les labels des images avec lesquelles on s'est entraînés, ce qui est très mauvais car on devrait être aux alentours des 95%).

Le code en annexe détaille la procédure que nous avons utilisé pour récupérer les images du dataset, et comparer les prédictions du modèle avec les prédictions qu'il aurait dû trouver.

On charge d'abord le modèle, puis le dataset d'images et celui des labels. Ensuite, pour chaque image dans le dataset, on la convertit sous deux formes différentes : d'abord sous forme d'image png pour pouvoir la visualiser à l'écran, puis sous une forme d'array numpy, car le modèle est construit de manière à lire des array de dimension 4 (une image est de dimension 3).

A partir de cette structure, on peut demander au réseau de neurone de faire une prédiction sur l'image qui lui est envoyée. Il retourne alors une prédiction brute, constituée de plusieurs probabilités.

On raffine ensuite ces prédictions en ne gardant que la plus grande probabilité du tableau. On a alors la prédiction finale que l'on pourra envoyer au contrôleur des moteurs pour effectuer la correction de trajectoire.

On a remarqué à partir des aperçus des images que l'on envoyait qu'il faisait de mauvaises prédictions dans la plupart des cas. C'est pour cela que nous avons introduit le dataset des labels pour comparer la prédiction du réseau avec ce qu'il aurait dû sortir normalement. C'est avec ces lignes de codes que nous avons vu que le réseau était mauvais.

Conclusion :

Il faut tout de même noter que, de tous les modèles que nous avons pu tester, celui-ci est de loin le meilleur. En effet, le modèle fourni par Axionaute nous renvoyait toujours la même prédiction à quelques décimales près, quelle que soit l'image.

Celui de Patate retournait des résultats aberrants complètement en dehors des limites de fonctionnement de leur script.

Enfin, le premier modèle que nous avons entraîné en utilisant le script d'entraînement d'AxionauteV1 renvoyait systématiquement une réponse sous forme de loi normale (qui privilégie donc la trajectoire "tout droit").

C'est donc avec ce nouveau script d'entraînement AxionauteV2 que nous avons les meilleurs résultats, même s'ils sont loin d'être corrects.

Equipe 3 : Commandes de la voiture en vitesse et direction à partir d'un raspberry pi.

Matériels et logiciels :

- Un Raspberry Pi 3 et accessoires
- Arduino Uno et logiciel correspondant
- Circuit de la voiture (servo-moteurs et ESC)

1. Installation du système d'exploitation Raspbian Lite sur la RaspberryPi

Le choix de la distribution linux, pour le raspberry pi, s'est orienté vers un raspbian lite pour obtenir une rapidité d'exécution (par l'absence d'interface graphique) et une simplicité de mise en oeuvre.

Le port série étant utilisé par la communication Arduino/Raspberry pi il a fallu établir une connexion ssh entre un pc et la carte Raspberry. Cette dernière peut se trouver à l'adresse 172.26.145.69/24. Pour que la carte ait un accès internet il peut être nécessaire d'ajouter le proxy polytech et de configurer le pare-feu du pc en utilisant iptables.

Sur le raspberry pi :

- export https_proxy=https://proxy.polytech-lille.fr:3128
- export http_proxy=http://proxy.polytech-lille.fr:3128

Sur le pc :

- iptables -F & iptables -tnat -F & iptables -PFORWARD ACCEPT

Enfin nous avons installé python, c'est à l'aide de ce langage que nous communiquerons avec l'Arduino. C'est un langage simple d'utilisation et qui propose de nombreuses bibliothèques.

2. Mise en place de la communication entre la Raspberry Pi et l'Arduino Uno.

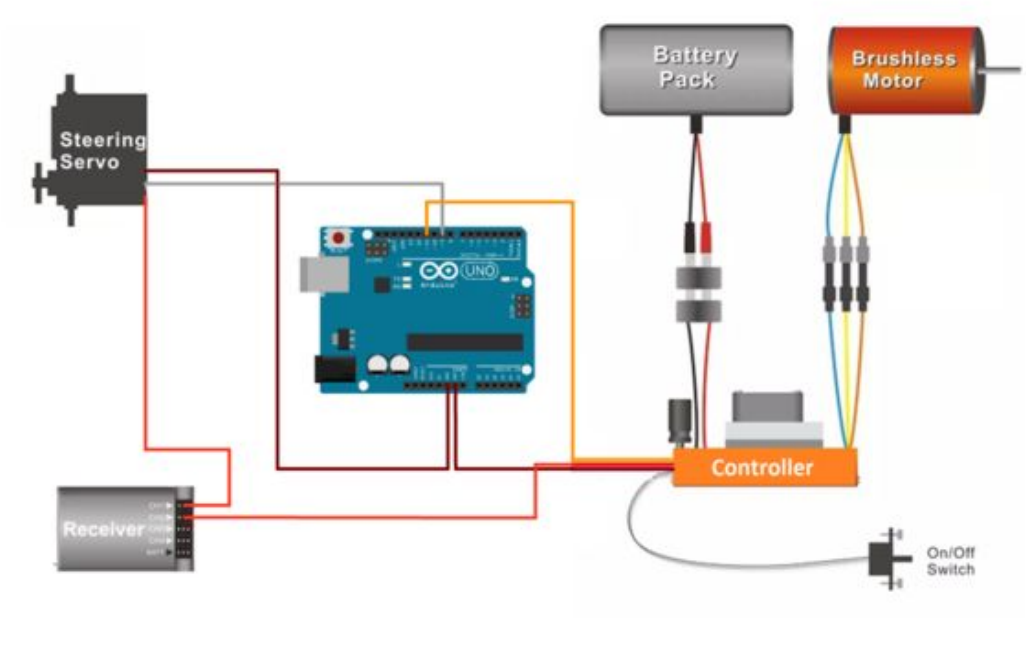
Le Raspberry Pi et la carte Arduino Uno communiquent par liaison série. Nous avons choisi de communiquer à 9600 bauds. La fin d'un message est un retour à la ligne.

Trois messages différents sont envoyés en fonction de la commande souhaitée.

Commande	Avancer	Tourner	Reculer
Valeur	1XXX	2XXX	3XXX

3. <u>Asservissements en vitesse et position des différents moteurs de la voiture.</u>

Les moteurs sont commandés grâce à l'arduino. Celui-ci doit avoir une masse commune avec le circuit de la voiture. Nous avons relié les fils de commandes aux entrées PWM 9 et 11 de l'arduino. Ci-dessous le schéma de câblage.



Le moteur de direction est commandé par le PIN 9. Il accepte des valeurs de 60 à 120°. Pour que les roues soient droites il faut envoyer 90, vers la droite 120 et 60 vers la gauche. La plage de message envoyé depuis le raspberry pi sera donc 2060-2120.

Concernant le contrôle de la vitesse de la voiture (PIN 11), le moteur fonctionne de la même manière que pour la direction grâce à un variateur. Les commandes sont donc les suivantes :

Frein max	Point neutre	Vitesse max
8°	90°	180°

Le frein max est pour une commande de 8° car la calibration du variateur n'est pas parfaite.

Avec l'Arduino il est nécessaire d'effectuer la calibration du moteur à chaque démarrage. Cela revient à envoyer les commandes de point neutre puis vitesse max et enfin de frein max.

Il reste un cas particulier, celui de la marche arrière. Pour commencer il faut envoyer la commande de frein max puis se placer en position neutre. A partir de ce moment on peut envoyer une commande de vitesse. Pour simplifier l'utilisation de cette fonction, il suffit d'envoyer 3XXX à l'arduino. La vitesse de recul allant de 90 à 0. La valeur la plus faible étant la vitesse la plus grande.

Conclusion :

Nous avons réalisé l'entièreté de la commande de la voiture à partir d'une raspberry pi connectée par liaison série à un arduino. Il est donc possible de faire avancer, reculer et tourner la voiture. Il est maintenant nécessaire de récupérer les données de sortie de l'intelligence artificielle et de les transformer en une commande de vitesse et de direction. Il serait également intéressant de créer un support pour tenir les cartes, la batterie sur la voiture.

IV / Bilan et Perspectives

Pour toutes les équipes le bilan est plutôt positif.

Pour la première équipe l'installation du linux RT s'est déroulée sans problème particulier même si nous n'avons pas pu réellement tester si le réseau de neurones s'exécute correctement dessus. Il faudra probablement installer une version du noyau compatible avec les exigences de l'IA.

L'équipe 3 est elle aussi parvenue au bout de son cahier des charges, la voiture est commandée correctement dans toutes les directions par la liaison série avec la Raspberry Pi.

Le seul point négatif concerne l'équipe 2. Les objectifs du cahier des charges sont atteints, mais les prédictions renvoyées par le réseau de neurones ne sont pas régulières.

Il reste bien évidemment plusieurs grandes parties à réaliser avant de pouvoir participer à une quelconque compétition.

Par exemple, il reste à réaliser la jointure entre les prédictions du réseau et la commande de la voiture. Il faudra très certainement adapter le script chargé d'envoyer les données vers le contrôleur des moteurs.

Il faudra également basculer l'IA et la commande sur le système en temps réel pour avoir des réponses plus rapides.

Et enfin, il restera à résoudre le problème des prédictions du réseau, et d'améliorer le programme d'entraînement avec nos propres images. On peut aussi penser à réaliser un script qui capture des images de la caméra pour récupérer plus de données facilement.

Bibliographie :

- https://www.blaess.fr/christophe/
- https://www.linuxembedded.fr
- https://peip-ima.plil.fr/mediawiki/index.php/BE_2018-2019
- https://www.arduino.cc/reference/en/
- https://github.com/Axionable/Axionaut
- https://github.com/EParisot/Patate
- https://github.com/guysoft/RealtimePi

ANNEXES

Programme Arduino :

```
#include <Servo.h>
Servo traction;
Servo direct;

//Initialisation traction
const long interval = 1000;
unsigned long previousMillis = 0;

//Liaison série
boolean messageRecu = false;
String inputString = "";

void setup(){
  //Liaison série
  Serial.begin(9600);
  //Commande Moteur
  traction.attach(11);
  direct.attach(9);
  init_traction();
}

void loop()
{
  if(messageRecu == true)
  {
        int val = inputString.toInt();

        if(val/1000==1)//Tourner
        {
        direct.write(val%1000);
        }
        else if(val/1000==2)//avancer
        {
        traction.write(val%1000);
        }
        else if(val/1000==3)//reculer
        {
        traction.write(8);
        delay(200);
        traction.write(90);
        delay(200);
        traction.write(val%1000);
        }
        messageRecu = false;
        inputString = "";
  }
}
```

```
void init_traction()
{
  for(int i=0;i<3;)
  {
        unsigned long currentMillis = millis();
        if (currentMillis - previousMillis >= interval) {
        previousMillis = currentMillis;
        if(i==0)        traction.write(90);//Point neutre
        else if(i==1) traction.write(180);//Accélération max
        else if(i==2) traction.write(0);//Frein max
        i++;
        }
  }
}

void serialEvent() {
  while (Serial.available()) {
        char inChar = (char)Serial.read();
        inputString += inChar;

        if (inChar == '\n') {
        messageRecu = true;
        }
  }
}
```

Programme test commande Raspberry Pi

```
import sys
import serial
import time
from time import sleep

ser = serial.Serial('/dev/ttyACM0',9600)
vit = 2098
dir = 1060

etat = 0
while 1 :
   ser.write(str(dir) + '\n')
   ser.write(str(vit) + '\n')
   if etat == 0:
            dir +=1
   else
            dir -=1
   if dir%10 == 0 :
            if etat == 0 :
                     vit +=1
            else
                     vit -=1
   if dir >= 1120 :
            etat = 1
   if dir <= 1060 :
            etat =0
   sleep(0.1)
```

Script de test de prédictions

```
#Loads the numpy array of pictures
frames = np.load("x_chicane.npy")
print("Frames loaded")

#Loads the numpy array of labels associed with each picture
real=np.load("y_chicane.npy")

right = 0
for i in range(0, len(frames)):
    #Gets the ith picture in the numpy array
    frame = frames[i]
    print("First frame loaded")

    #Converts the picture to png file to visualize it
    img = Image.fromarray(frame)
    img.save("0.png")
    img.show()

    #Converts the picture to the data structure required from the model
    image = np.array([frame]) / 255
    print("Frame converted into image")

    #Makes a raw prediction of the path to take
    preds_raw = model.predict(image)
    print("Raw prediction generated")
    print(preds_raw)

    #Refines the prediction
    preds = [0, 0, 0, 0, 0]
    maxi = np.argmax(preds_raw[0])
    preds[maxi]=1
    print("Refined array :")
    print(preds)

    #Gets the real label from the picture taken from the dataset and compares it to the refined predictior
    print("Real array : ")
    pyreal = np.array(real[i]).tolist()
    print(pyreal)
    if preds == pyreal:
        right = right +1


percent = right/len(frames)
print("Percentage of correct predictions : ")
print(percent)

#Code used with Patate's model
#preds = [np.argmax(pred, axis=1) for pred in preds_raw]
#print("Predictions generated")
#print(preds)
```