ESAIP
Engineering School
-Saint Barthélémy d'Anjou-

Applicative Project Report

COMPUTER SCIENCE

Specialty : Big DATA

Implementation of an autonomous driving system in simulation software

---

Partner organization : Groupe Sigma

Realized by :
M. Cyprien Barbault
M. Romain Girou

Under the responsability of :
Pr. Pejman Rasti (ESAIP)
Pr. Redouane Djelouah (ESAIP)
M. Armel Kermorvant

Defended on January 13rd, 2021 in front of the jury E110:

Pr. Pejman Rasti :	ESAIP Saint Barthélémy d'Anjou
Pr. Redouane Djelouah :	ESAIP Saint Barthélémy d'Anjou

Year
2020/2021

Contents

Thanks

To Mr. KERMORVANT for his proposal to enroll us in the competition.

To Mr. Pejman, for his disponibility and his goodwill.

To Mr. Crochet for his fantastic LaTeX course.

To Freud MAGUENDJI, our team partner.

To Groupe Sigma for this amazing competition and the professionalism of its speakers.

To ESAIP for the financial support

1 Introduction

The Applicative Project

During the second year of our engineering degree at ESAIP, we had to do what is called the "Projet Applicatif"[1]. This project is the occasion for us to do any kind of project we always wanted to do, but never had the opportunity or time to accomplish.

As we are both Big DATA students, we decided to opte for a project using machine learning and/or deep learning. Our first idea was to try to simulate an autonomous car in a virtual environment[2], but then, mid October Mr. KERMORVANT offered to participate in a tournament: **<IA/> Racing**.

Groupe Sigma and <IA/> Racing

In 2020, the **Groupe Sigma** launched an autonmous RC-Car competition. 6 team of 3 membres (5 teams of students and 1 team of Sigma's employees) are going to compete on a circuit specialy made for the occasion. Each round, 2 cars will drive all alone on the circuit, without any kind of intervention from the teams. The goal of this competition is to initiate student to machine learning and electronic.

You could ask yourself, "But you are only two, how can you participate in the race ?". Well during this Applicative Project we are only two, but for the Sigma Race we're going to be helped by **Freud MAGUENDJI**. He was already in an other project of computer vision when Mr. KERMORVANT offered us to enter the competition. We did the whole machine learning and simulator setup just by ourselves, and Freud is going to help us for the rest of the project as he is an IOT[3] student.

[1] Applicative Project
[2] CARLA to be exact
[3] Internet Of Things

2 Specifications

To be able to participate to the race, our car needed to validate some technical specifications. **Groupe Sigma** gave us a set of rules to follow.

2.1 Technical Specifications

To enter the race, the car have to:

- have 4 wheels without any other ground support
- be 30x25x25cm[4] maximum
- weight 4kg maximum
- have a maximal total battery power of 7800mAh
- be equiped with 2 motors maximum
- be equiped with a circuit-cutter in case of emergency
- be autonomously controlable with the use of captors such as cameras, lidars, ultra-sound...

Figure 1: Example of a "DonkeyCar", an opensource DIY self driving small car

In addition, it's good to know that both thermal motors and cloud computation[5] are forbiddens and that the total budget cannot exceed 500€.

2.2 Start and Finish

The car have to be able to start the race by itself after a luminous countdown done by a traffic light of 10x10cm (red, yellow and green). If the car has not start 2 mn after the start, it will be discalified. Each vehicle has the right to a false start, after it will be penalysed.

The same way, the vehicule must be able to stop itself maximum 5s after crossing the finish line. This line will be yellow and 15cm long accross the wole width.

Any breaches of these rules will result in a penalty:

- 5s for a false start (after the autorized one)
- 5s if the car hasn't stop itself in the 5s limits

[4] Lenght·Width·Height
[5] During the race only, training of the model can be done on the cloud

2.3 Circuit

The circuit will be made of 2cm wide white marking on top of a 4cm wide black marking on each side of the lane, and will be 45 meters long. Obstacle of 10x10x10cm will be set on the track and could change between each race. The speed is limited to 65km/h.[6]
So the car must be able to identify those lane and drive without crossing them.

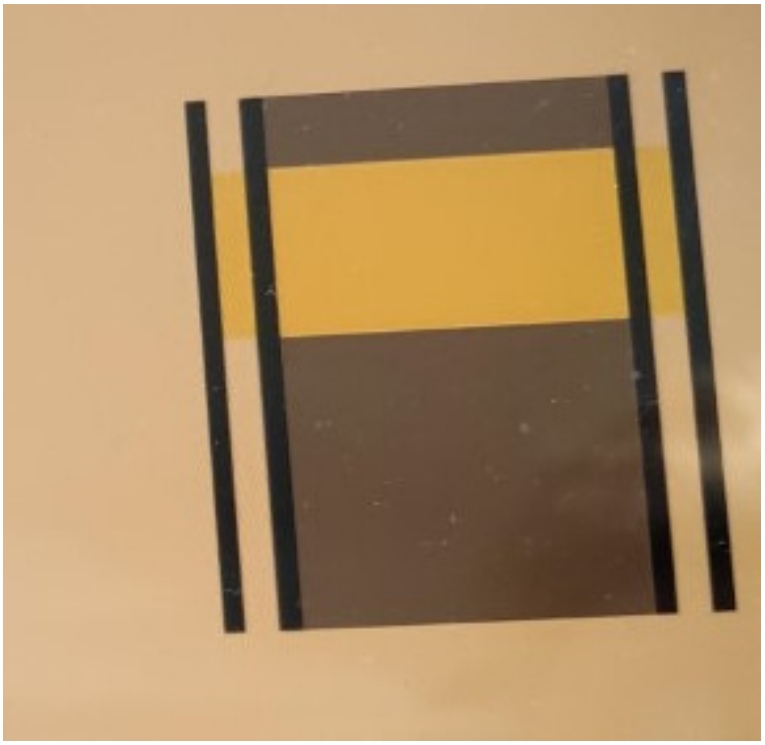


Figure 2: Representation of the Finish line

[6]Even though it's pretty unlikely that any car can reach this speed in autonomous mode.

3 Project Management

3.1 Planning

The applicative projet only last a semester, but the sigma race is only the 3^{rd} of June, so we didn't planned to accomplish all the work only during the **AP**.[7] Instead, we decide to focus first on what was the original goal of our AP, driving autonomously in a simulator.

As we opted for this AP kind of late[8], we really started working on it arround beginning of November. The first thing to do was to documented ourselves about Machines Learning, Deep Learning, existing software on the domain...
We gave ourselves a month to gather information about the state of the art in machine learning in autonomous car, but it's such a big domain that we only scratched the surface.

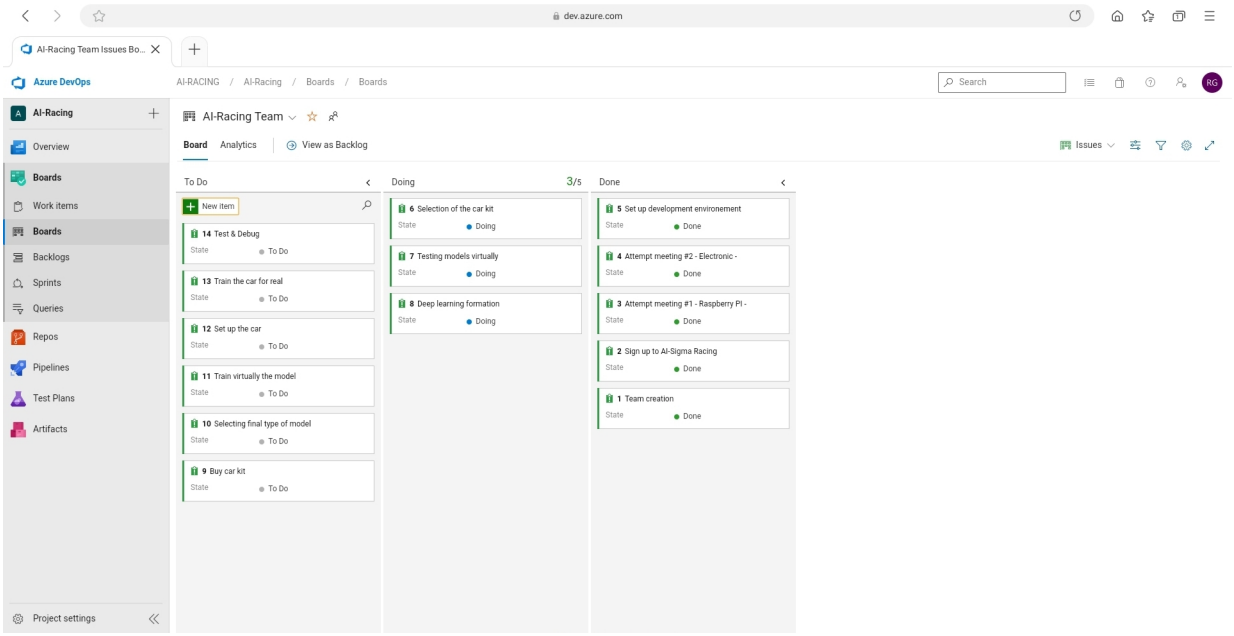


Figure 3: We decided to use **Azure DevOps** to manage our project

3.2 Meetings

As we are flatmates, it was pretty easy to do meetings, we used to do a meeting every week or so, in order to share the result of our research. We both start reading book about Machine Learning and Deep Learning in order to understand how this kind of AI are working[13].

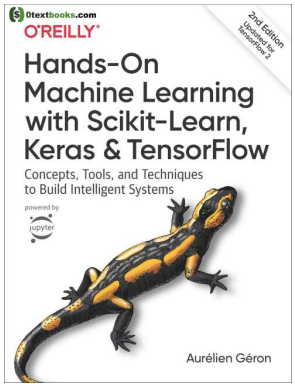
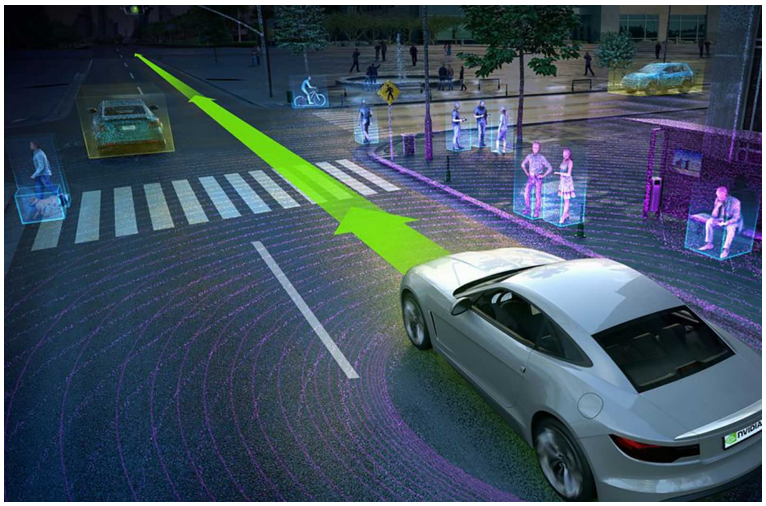


Figure 4: The book Cyprien bought

[7] Short for applicative project

[8] We first heard of it mid October, we had to think about it, and then we had to do all the proceeding for our inscription.



3.3 Division of labor

We divided the work as follow:

- Cyprien had to setup the simulator
- Romain had to design a lane detection algorithm
- We had to do research on our own to compare deep-learning models to find the best for our situation.

4 Technical overview

4.1 DonkeySimulator

When it comes to simulators, we had 2 options:

1. Udacity Simulator
2. DonkeySimulator

Figure 5: Udacity Simulator

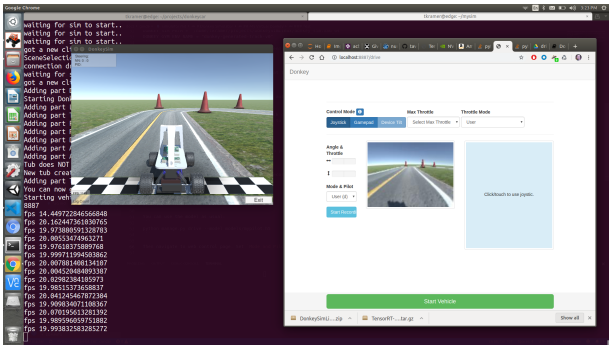


Figure 6: DonkeySimulator

We decided not to choose the udacity simulator, even though closest to the reality than the DonkeySimulator. "Why ?", you may ask. It's because the DonkeySimulator has a built-in track really similar to the one where our car is going to drive the D-day. Moreover, it's easy to implement a second camera to the car, or even simulate the compute power of a **Raspberry pi**. There's even an OpenAi-Gym environnment !

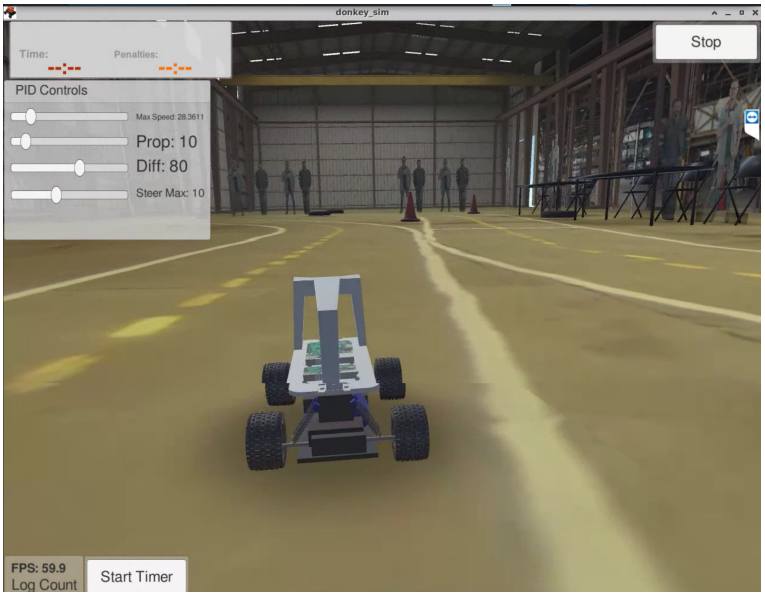


Figure 7: The track on DonkeySim is really similar to the real one

4.2 Dataset

We started the project mid-november and we still, at this day, do not have a physic car. In the other hand, we needed to start training models to choose the most efficient. Our only option was to find online dataset to start training a model, and then, when we'll receive the car, we'll just have to transfer it in the car.

4.3 Our model choice

First we need to plant the basics. What our AI model needs to make predictions on ? Our model will make live predictions on the steering and the throttle of our car using images given by the camera on the car.

What is the best algorithm ? Which one is the most relaible, efficient ?

We have several options and this part, we will present the pros and the cons of the algorithm selection that we have made.

We will be using the Keras (open-source sofware library with a python interface) high level API in combination with a Tensorflow backend.

Keras Categotical

Pros	**Cons**
It has some benefits of showing the confidense as a distribution via the makemovie command	Suffers from some arbitrary limitations of the chosen limits for number of categories, and thottle upper limit
It has been very robust	
In some cases this model has learned thottle control better than other models	
Performs well in a limited compute environment like the Pi3	

Table 1: Keras Categorical Pros/Cons

Pros	Cons
Steers smoothly	May sometimes fail to learn throttle well
It has been very robust	
No arbitrary limits to steering or throttle	
Performs well in a limited compute environment like the Pi3	

Table 2: Keras Linear Pros/Cons

Keras Linear

Keras IMU

Pros	Cons
Steers very smoothly	Driving quality will suffer if noisy imu is used
Performs well in a limited compute environment like the Pi3	
No arbitrary limits to steering or throttle	
Gives additional state to the model, which might help it come to a stop at a stop sign	

Table 3: Keras IMU Pros/Cons

Keras Latent

Pros	Cons
Steers smoothly	Needs more testing to prove theory
Performs well in a limited compute environment like the Pi3	
No arbitrary limits to steering or throttle.	
Image output a measure of what the model has deemed important in the scene	

Table 4: Keras Latent Pros/Cons

Keras RNN

Pros	Cons
Steers very smoothly	Performs worse in a limited compute environment like the Pi3
Can train to a lower loss	Takes longer to train

Table 5: Keras RNN Pros/Cons

Keras Behaviour

Pros	Cons
Can create a model which can perform multiple tasks	Takes more effort to train

Table 6: Keras Behaviour Pros/Cons

Keras Localizer

Pros	Cons
Steers smoothly	May sometimes fail to learn throttle well
Performs well in a limited compute environment like the Pi3	
No arbitrary limits to steering or throttle.	
Location to supply some higher level logic	

Table 7: Keras Localizer Pros/Cons

4.4 Lane Detection algorithm

The first to do was to create a simple lane detection algorithm, by assuming that all line will be straight. We used **OpenCV**, an open-source computer vision library, usable both in Python and C++. This tool include image processing, camera calibration, object detection...

Figure 8: OpenCV is an Open-source Computer Vision Library

The usual methodology in this case is as follow:[9]

[9]Cf. appendix .1 for the complete code and appendix .2 for the pictures

1. Put image in gray in order to emphasize the white lines
2. Blur the immage using a **Gaussian Filter**[10]
3. Use the **Canny** algorithm to find edges on the image
4. Cropp the image only where the lines should be
5. Using the **Hough** algorithm to find lines in the resulted image
6. Select the average left line and right line from the array of lines
7. Finaly display the line on top of the image (or frame in case of a video)

[10] By averaging the values of nearby pixels, the Gaussian Filter reduce the noise in the image

4.5 OpenAI Gym

OpenAI, one of the most famous company in the Deep Learning domain, has developped an open-source framework to help the implementation of a certain type of deep learning algorithm. This kind of algorithm is particularly efficient when it comes to... playing games !

In fact, OpenAi-Gym is best suited for **Reinforcement Learning**, a genre of neural network were the agent is rewarded when taking good actions and penalized when doing bad things. These neural network are complicated to implement in the real world, imagine a car having to crash thousands of time before learning to turn left ! But that doesn't mean they are bad ! In fact, it's kind of the opposite: remember **AlphaGo Zero** ? It's the first algorithm to defeat the world champion of Go[11] in a 1vs1 match, and it was using reinforcment learning !

Reinforcement Learning

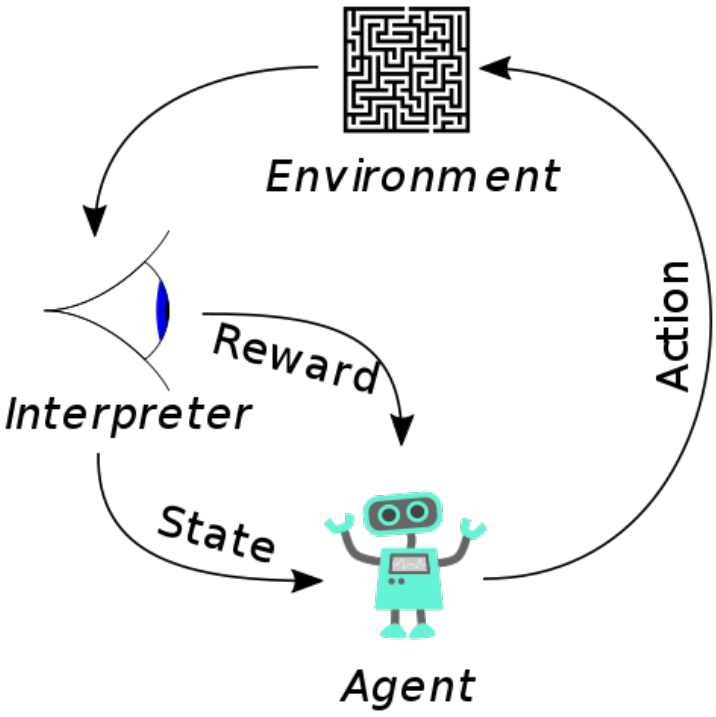


Figure 9: Schematic of the reinforcement learning algorithm

[11] Go is a game with more than 10^{17} possible board configuration[10]

In order to train a Reinforcement Neural Network, we have to provide it 2 differents things at each step:

- A state
- A reward

And the neural network will output an action which is going to have an impact on the environment. Then the loop repeat itself.

In **Python**, OpenAi Gym is implemented as follow:

OpenAi Gym in Python [18]

`Steps` stipulate how long the networks is going to run (it has to be an Integer). At each step, `Observation`, `Reward`, `Done` and `Info` are outputed.

`Observation` represent all the data fed to the agent at a given step, it can be pixel data such as in a picture, speed, memory.[12] The observation differ depending on the task, so it's taking the form of an `object`.

`Reward` is a `float`, positive or not, given to the agent after he took an action. The goal of the agent is to maximise the total reward.

`Done` is a `boolean`. When set to `True`, it indicate that the episode has terminated (either the task has failed, or the task was successfuly handled).

`Info` is a `dictionnary` containing information for the developper (often used for debuging). The agent is not allowed to access these informations.

`Space` is either a `set` or a `box`[13] containing all the possible `Actions` that the agent can take.

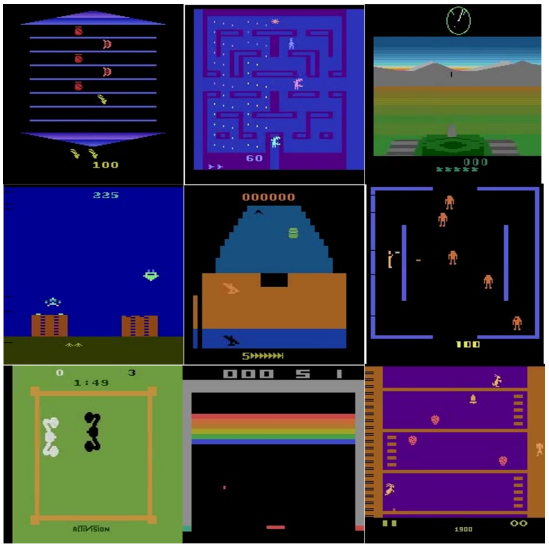


Figure 10: Different kind of Atari Games environment available in OpenAi-Gym

[12] One of the goal of OpenAi Gym was to make an AI able to beat most of the NES game with only the raw memory as observation

[13] Depending if the action space is continous or discrete

This framework makes it really easy to implement RNN in environment well formated. Fortunatly, the community have already build a OpenAI-Gym environment for the DonkeySimulator: Gym-Donkeycar

Gym DonkeyCar

Thanks to Tawn KRAMER[15], we have a pre-made environment following the OpenAI-Gym methodology. It basically runs a TCP connection with the simulator and transform the data received into Python formated data. This way, we don't have to create the TCP client by ourself[14]
The same component are implemented as follow:

`Observation` is a `np.array` with a shape of (120, 160, 3).[15]

`Done` is a `boolean` set to `True` when the car hit an obstacle (`hit != "none"`) or when it's too far from the center of the lane.

`Reward` is an `array` containing `Done`, the **speed**[16] of the vehicule and the **Cross Track error** (cte)[17]

`Info` is an `array` containing the **cte**, the **position** (x,y,z), the **speed** (positive forward, negative backward) and if the car has it an obstacle: **hit** (equal to "none" if all good)[6]
With all these information, we can feed the network with the `Observation` and make it output a steering angle and a throtle value.

It's quite funny to watch the IA train, because it's basically running all over the place at first, but then it start going further and further on the track. Sometimes you can have a big gain from a step to an other, or the AI can even regress ! But in the end, after a few hours of training, the AI is finaly able to drive through the whole track !

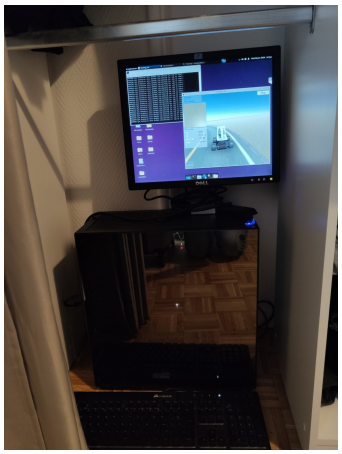


Figure 11: The Linux server we used for our training : Xeon X5650 (6 cores 12 threads) 8Go DDR3 - Nvidia GTX 970 4Go

Layer (type)	Output Shape	Param #	Connected to
img_in (InputLayer)	(None, 120, 160, 3)	0	
conv2d (Conv2D)	(None, 58, 78, 24)	1824	img_in[0][0]
conv2d_1 (Conv2D)	(None, 27, 37, 32)	19232	conv2d[0][0]
conv2d_2 (Conv2D)	(None, 12, 17, 64)	51264	conv2d_1[0][0]
conv2d_3 (Conv2D)	(None, 10, 15, 64)	36928	conv2d_2[0][0]
conv2d_4 (Conv2D)	(None, 8, 13, 64)	36928	conv2d_3[0][0]
behavior_in (InputLayer)	(None, 2)	0	
flattened (Flatten)	(None, 6656)	0	conv2d_4[0][0]
dense_1 (Dense)	(None, 4)	12	behavior_in[0][0]
dense (Dense)	(None, 100)	665700	flattened[0][0]
dense_2 (Dense)	(None, 4)	20	dense_1[0][0]
dropout (Dropout)	(None, 100)	0	dense[0][0]
dense_3 (Dense)	(None, 4)	20	dense_2[0][0]
concatenate (Concatenate)	(None, 104)	0	dropout[0][0] dense_3[0][0]
dense_4 (Dense)	(None, 100)	10500	concatenate[0][0]
dropout_1 (Dropout)	(None, 100)	0	dense_4[0][0]
dense_5 (Dense)	(None, 50)	5050	dropout_1[0][0]
dropout_2 (Dropout)	(None, 50)	0	dense_5[0][0]
angle_out (Dense)	(None, 15)	765	dropout_2[0][0]
throttle_out (Dense)	(None, 20)	1020	dropout_2[0][0]

Total params: 829,263
Trainable params: 829,263
Non-trainable params: 0

Figure 12: example model run

[14] Although we tried doing it cf. appendix .3
[15] Basicaly it's the RGB representation of a 120x160px image
[16] Between max=1 and min=-2
[17] How far the car is from the center of the lane

4.6 Benchmarks

Using the DonkeySimulator, we ran 4 differents algorithms (unfortunatly we were not able to compute neither the imu nor the behaviour model), all in the same conditions, in order to compare their performances :
We used 9143 training images that we split in 7314 training images and 1829 validation images. 57 steps have been run each epoch.

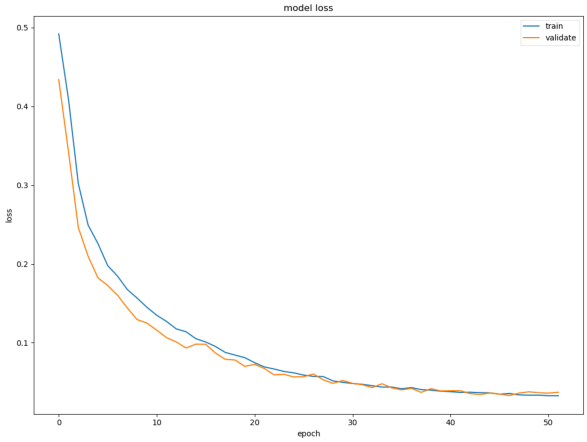


Figure 13: Linear model loss

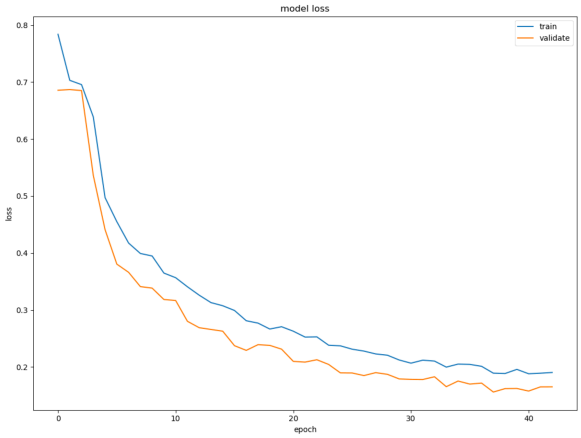


Figure 14: Latent model loss

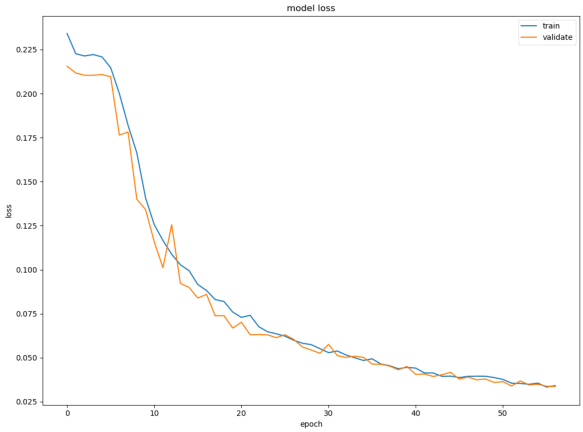


Figure 15: RNN model loss

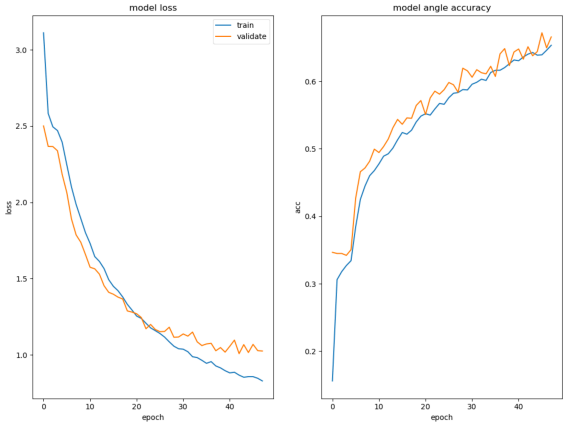


Figure 16: Categorical model loss and model angle loss

We can see that the **Linear** (Fig. 13) and **RNN** (Fig. 15) model we used fit pretty well the data, but in the other hand, the **latent** (Fig. 14) model tend to underfit the data while the **Categorical** (Fig. 16) model seems to overfit it.[18]

Even though the linear model look pretty well suited for the task, in practice, we found that it tends to oscilate way more than the RNN model. A solution to this kind of problem would be to implement a PID controler to smooth the driving, but this will be a way of improvement for the final model.

[18]The categorical model has 2 plot, one for the model angle accuracy and one for the model itself, that's because it's first trained on angle only and then on both throttle and angle.

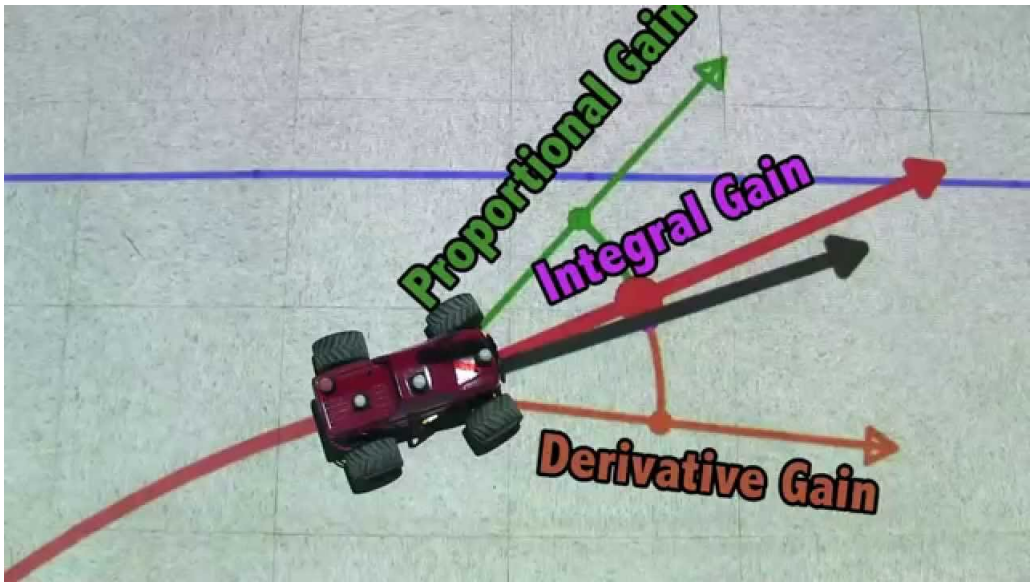


Figure 17: A PID Controler

5 Conclusion

This applicative project and first part of the sigma racing was very interesting.
We learned a lot and we now know that we are really pasionate about image recognition and deep learning.

It is a very complex domain and we are far from mastering it. Nevertheless, we have done our best in order to make a working model. We took deep learning online classes and read books to complete our train. It seemed, at first, very complexe (mostly the mathematicals aspects, and its still is !) but we managed to understand the basics and we are learning more every days.

This project will come to an end on June 2021. The next step is to buy the car and implement all the work that we have done inside it. We also plan to design our own removable track with the same specifications as the official one in order to test the car in different track configurations. Moreover, we will need to get cracking on the obstacles detections (is a Lidar sensor usefull ?) as well as the start and finish behaviors of the car.

We are still struggling with the hardware as we cannot borrow a GPU for the schools servers. It has a direct affect on the duration of the training and some help in this area would be more than welcome.

Figure 18: Romain (left) and Cyprien (right)

6 List of Figure and Tables

List of Figures

List of Tables

7 Bibliography

References

[1] Raffin Antonin and Sokolkov Roma. Learning to drive smoothly in minutes. https://github.com/araffin/learning-to-drive-in-5-minutes/, 2019.

[2] Autorope. https://github.com/autorope/donkeycar/blob/master/donkeycar/parts/transform.py.

[3] Cian B. Donkey car simulator with real rc controller. https://www.hackster.io/wallarug/donkey-car-simulator-with-real-rc-controller-628e77, 05 2020.

[4] Cian B. Donkey car with jetson nano robo hat mm1. https://www.hackster.io/wallarug/donkey-car-with-jetson-nano-robo-hat-mm1-e53e21, 05 2020.

[5] Engin Bozkurt. Self driving car control design. https://github.com/enginBozkurt/SelfDrivingCarsControlDesign.

[6] Donkey Car. Donkey simulator. https://docs.donkeycar.com/guide/simulator/, Unknown.

[7] Fei Cheung. Getting started with jetson nano and donkey car aka autonomous car. https://medium.com/@feicheung2016/getting-started-with-jetson-nano-and-autonomous-donkey-car-d4f25bbd1c83, 04 2019.

[8] Jeremy Cohen. Deep reinforcement learning for self-driving cars — an intro. https://thinkautonomous.medium.com/deep-reinforcement-learning-for-self-driving-cars-an-intro-4c8c08e6d06b, 12 2019.

[9] Roscoe (daduce). How i'd try to win the 36 hour self driving hackathon at aws reinvent. https://www.donkeycar.com/updates/what-id-try-to-win-the-36-hour-self-driving-hackathon-at-aws-reinvent, 11 2017.

[10] DeepMind. Alphago. https://deepmind.com/research/case-studies/alphago-the-story-so-far, Unknown.

[11] Dexlab. Machine learning - algorithm used in autonomous car - a study. https://www.dexlabanalytics.com/blog/machine-learning-algorithms-in-self-driving-cars, 03 2020.

[12] Florian Fuchs Yunlong Song Elia Kaufmann Davide Scaramuzza Peter Durr. Super-human performance in gran turismo sport using deep reinforcement learning. https://arxiv.org/pdf/2008.07971v1.pdf, 2008.

[13] Aurélien Géron. Hands-On-Machine Learning with Scikit-Learn Keras & TensorFlow 2nd Edition. O'Reilly, 2019.

[14] Raj K. Donkey car - imitation learning. https://github.com/RajK853/DonkeyCar.

[15] Tawn Kraner. Openai gym environments for donkey car. https://github.com/tawnkramer/gym-donkeycar, Unknown.

[16] Nvidia. Pilotnet: End to end learning for self-driving cars. https://github.com/lhzlhz/PilotNet.

[17] Yuichi Okuyama. donkey-data. https://github.com/hogenimushi/donkey_data.

[18] OpenAI. Gym. https://gym.openai.com/.

[19] RobocarStore. donkeycar-images. https://github.com/robocarstore/donkeycar-images, Unknown.

[20] Sachin. Self driving car using google colab. https://github.com/sachindroid8/self-driving-car-using-google-colab.

[21] Sentdex. Introduction - self-driving cars with carla and python part 1. https://pythonprogramming.net/introduction-self-driving-autonomous-cars-carla-python/.

[22] Groupe Sigma. Développer la proximite entreprise / étudiants : Ia racing. https://www.sigma.fr/2020/12/03/favoriser-une-intimite-plus-grande-entre-lentreprise-et-les-etudiants/, 2020.

[23] Rayan Slim Amer Sharaf Jad Slim Sarmad Tanveer. The complete self-driving car course - applied deep learning. https://www.udemy.com/course/applied-deep-learningtm-the-complete-self-driving-car-course/, 09 2020.

[24] Udacity. self-driving-car-sim. https://github.com/udacity/self-driving-car-sim, Unknown.

[25] Unknown. Exceed rc truck radio car 1/16 2.4ghz magnet ep electric powered rtr off road truck stripe blue rc remote control car. https://www.nitrorcx.com/51c853-stripeblue-24-ghz.html.

[26] Unknown. Open datasets. https://diyrobocars.com/open-datasets/.

[27] Unknown. Learning to drive smoothly in minutes. https://towardsdatascience.com/learning-to-drive-smoothly-in-minutes-450a7cdb35f4, 01 2019.

[28] Felix Yu. Train donkey car in unity simulator with reinforcement learning. https://flyyufelix.github.io/2018/09/11/donkey-rl-simulation.html, 09 2018.

Appendices

.1 Code for the Lane Detection algorithm

```
import cv2
import numpy as np

def make_points(image, line):
    slope, intercept = line
    y1 = int(image.shape[0])# bottom of the image
    y2 = int(y1*3/5)         # slightly lower than the middle
    x1 = int((y1 - intercept)/slope)
    x2 = int((y2 - intercept)/slope)
    return [[x1, y1, x2, y2]]

def average_slope_intercept(image, lines):
    left_fit    = []
    right_fit   = []
    if lines is None:
        return None
    for line in lines:
        for x1, y1, x2, y2 in line:
            fit = np.polyfit((x1,x2), (y1,y2), 1)
            slope = fit[0]
            intercept = fit[1]
            if slope < 0: # y is reversed in image
                left_fit.append((slope, intercept))
            else:
                right_fit.append((slope, intercept))
    # add more weight to longer lines

    if len(left_fit) and len(right_fit):
    #over-simplified if statement (should give you an idea of why the error
                                                occurs)
       left_fit_average  = np.average(left_fit, axis=0)
       right_fit_average = np.average(right_fit, axis=0)
       left_line  = make_points(image, left_fit_average)
       right_line = make_points(image, right_fit_average)
       averaged_lines = [left_line, right_line]
       return averaged_lines

def canny(img):
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    kernel = 5
    blur = cv2.GaussianBlur(gray,(kernel, kernel),0)
    canny = cv2.Canny(gray, 50, 150)
    return canny

def display_lines(img,lines):
    line_image = np.zeros_like(img)
    if lines is not None:
        for line in lines:
            for x1, y1, x2, y2 in line:
                cv2.line(line_image,(x1,y1),(x2,y2),(0,255,0),10)
    return line_image

def region_of_interest(canny):
    height = canny.shape[0]
    width = canny.shape[1]
    mask = np.zeros_like(canny)

    triangle = np.array([[
```

```
    (200, height),
    (550, 250),
    (1100, height),]], np.int32)

    cv2.fillPoly(mask, triangle, 255)
    masked_image = cv2.bitwise_and(canny, mask)
    return masked_image


# Code for the detection in an image named 'test_image.jpg'
image = cv2.imread('test_image.jpg')
lane_image = np.copy(image)
lane_canny = canny(lane_image)
cropped_canny = region_of_interest(lane_canny)
lines = cv2.HoughLinesP(cropped_canny, 2, np.pi/180, 100, np.array([]),
                                        minLineLength=40,maxLineGap=5)
averaged_lines = average_slope_intercept(image, lines)
line_image = display_lines(lane_image, averaged_lines)
combo_image = cv2.addWeighted(lane_image, 1, line_image, 1, 1)
cv2.imshow("result", combo_image)
cv2.waitKey(0)
cv2.destroyAllWindows()

# Code for the detection in a video named 'test2.mp4'
#cap = cv2.VideoCapture("test2.mp4")
#while(cap.isOpened()):
#    _, frame = cap.read()
#    canny_image = canny(frame)
#    cropped_canny = region_of_interest(canny_image)
#    lines = cv2.HoughLinesP(cropped_canny, 2, np.pi/180, 100, np.array([]),
                                        minLineLength=40,maxLineGap=5)
#    averaged_lines = average_slope_intercept(frame, lines)
#    line_image = display_lines(frame, averaged_lines)
#    combo_image = cv2.addWeighted(frame, 0.8, line_image, 1, 1)
#    cv2.imshow("result", combo_image)
#    if cv2.waitKey(1) & 0xFF == ord('q'):
#        break
#cap.release()
#cv2.destroyAllWindows()
```

.2 Image at each step in the lane detect algorithm

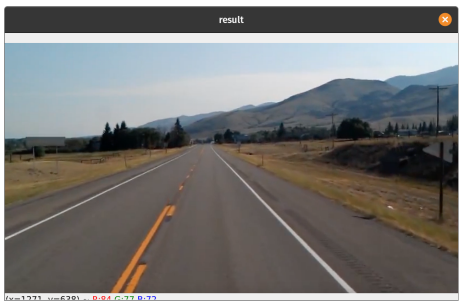


Figure 19: Original Image

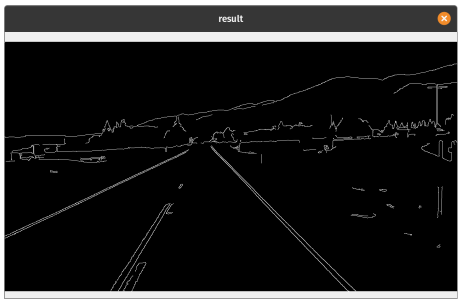


Figure 20: Gray Image after applying blur and canny-edge detection

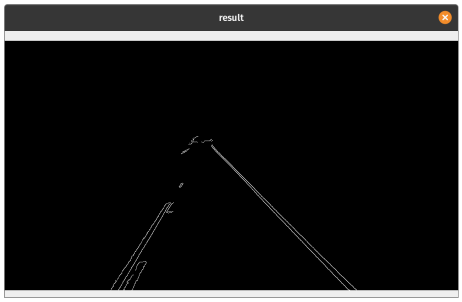


Figure 21: We cropped the image to the center triangle

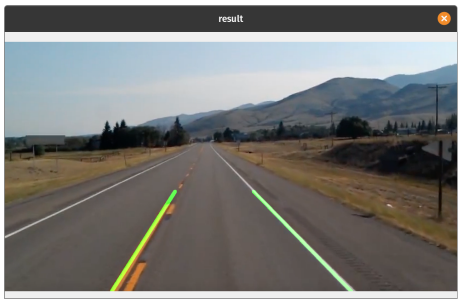


Figure 22: Final image with superposed detected lines (in green)

.3 TCP Client in Python

```
import socket
import base64
import requests
import socketio
import sys
import time
import json


# specify Host and Port
HOST = 'localhost'
PORT = 9091


def send_data(steering, throttle, brake):
    msg = str('{"msg_type": "control", "steering": "' +
              str(steering) + '","throttle": "' + str(throttle) +
              '","brake": "' + str(brake) + '"}').encode('utf-8')
    soc.sendall(msg)


def recv_full():
    """
    Test if the msg_type is telemetry, if yes, return the data
    """
    for _ in range(2):
        data = json.loads(soc.recv(8192).decode('utf-8'))
        if not data:
            break
        if data['msg_type'] == 'car_loaded':
            continue
        return json.loads(soc.recv(5000).decode('utf-8'))


def write_data():
    data = recv_full()
    with open('image.jpg', 'wb') as image:
        image.write(base64.b64decode(data['image']))
    with open("data.json", 'w') as f:
        json.dump(data, f)



with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as soc:
    soc.connect((HOST, PORT))
    write_data()



sio = socketio.Client()
```

.4 Financial Management Report

Next, you can find the financial management report we had to do, inside you will find an overview of the project and the differents approach we took for the budget repartition and the choice of the components of the car.

Gestion Financière – Projet Applicatif

Table des matières

1 Présentation du projet

1.1 Sigma Racing

En 2020, l'entreprise **Sigma** a lancé le concours <**IA**/> **Racing**. L'objectif de ce concours est de construire de toute pièce une voiture miniature autonome. La création et l'entrainnement du modèle d'intelligence artificielle responsable de la conduite de la voiture est également à la charge de l'équipe.

FIGURE 1 – Un exemple de "Donkey car", petite voitures autonome

Les équipes du concours sont à la fois étudiants ingénieurs (6 équipes, dont 2 de l'Esaip) et salariés du groupe sigma.

1.2 Spécificités techniques

Pour que la voiture soit validée le jour de la course [1], elle doit valider quelques spécificités techniques :

- Le véhicule doit posséder 4 roues
- Il doit avoir un gabarit de 30x25x25cm (longueur, largeur, hauteur)
- Peser 4kg maximum
- Avoir une batterie avec puissance maximale de 7800mAh
- Être equipe de 2 moteurs électriques maximum
- Être équipé d'un système d'arrêt à distance (coupe circuit)
- Être équipé d'un système de pilotage autonome (laser, ultra son, imagerie)

Il est également spécifié dans le réglement que le budget maximal autorisé est de 500€TTC [2]

1. À savoir, le 3 juin 2021
2. frais de déplacements non compris

2 Chiffrage

La voiture est constituées de divers éléments essentiel : la structure, le micro-ordinateur et les capteurs. Intéressons nous d'abord à la structure

Deux options s'offrent à nous :

- Acheter un kit **Robocar** ou **Donkey Car**
- Acheter chacune des pièces séparéments

2.1 Achat d'un kit

Les kits déjà constitués offrent une solution "de facilité". En effet les pièces sont déjà toutes réunies pour mettre en place une voiture auntonome basique (pas de capteurs LIDAR [3] ou Ultra-son). Cependant ils offrent moins de liberté dans le choix des pièces et peuvent s'avérer plus chers.

2.1.1 Kit Donkey Car

Le kit Donkey Car est constitué de :

- Une voiture modèle réduit au choix parmis 4 [4]
- Batterie de 6700mAh USB
- kit structure constitués des différents support pour micro-ordinateur et caméra

Pièce	**Coût**
Modèle réduit	90€
Batterie 6700mAh	22€
Kit Structure	82 €
Total	194€

FIGURE 2 – Contenu du kit structure Donkey-Car

3. Capteur de distance par laser
4. À savoir Exceed Magnet, Desert Monster, Blaze, ou Short Course Truck

2.1.2 Kit Robocar

Le kit Robocar est un peu particulier, il inclue la totalité des pièces nécessaire, y compris capteurs et ordinateurs :

- HSP 94186 Brushed RC Car (Voiture modèle réduite)
- Structure imprimée en 3D
- Base découpée au laser
- Raspberry Pi 4 (Cerveau de la voiture, micro-ordinateur)
- Caméra grand angle pour Raspberry
- 16Gb carte micro-SD (pour installer l'OS sur le Raspberry)
- Servo Driver PCA 9685 (pour controller les servos-moteurs des roues)
- DC-DC 5V/2A Voltage Converter (pour alimenter le Raspberry avec la batterie de la voiture)
- Divers autres pièces (vis, fils, ...)

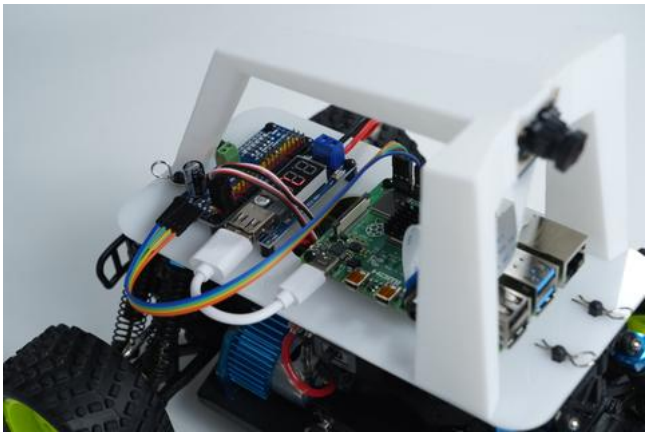


FIGURE 3 – Kit Robocar

Ce kit est disponible pour environ **230€**.

2.1.3 Comparaison

Le kit Donkey Car semble bien inférieur au kit Robocar, tout d'abord par l'absence de certains composant (pas de capteurs ni de micro-ordinateurs) mais également par son prix assez élevé. Comme **Sigma** nous propose d'imprimer nous même notre structure à l'aide de leurs imprimantes 3D, les 82€de structure pourraient être complétement annulés !

2.2 Micro-ordinateur

La pièce la plus importante d'une voiture autonome est évidement son cerveau. Comment traduire en temps réel des images en signaux d'accélération, freinage et direction si la puissance n'est pas suffisante ?
Pour chacune des option évoquée ci-dessous, il faut ajouter le prix d'une carte SD contenant l'OS, il faudra donc ajouter une vingtaine d'euros pour celle-ci.

2.2.1 Raspberry Pi

Le Raspberry Pi n'est plus à présenter, ce micro-ordinateur disponible pour une somme allant de 5 à 85€possède des connectiques essentielles pour gérer les servos-moteurs de direction.

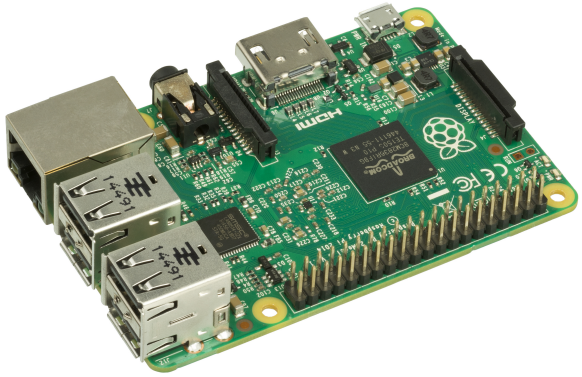


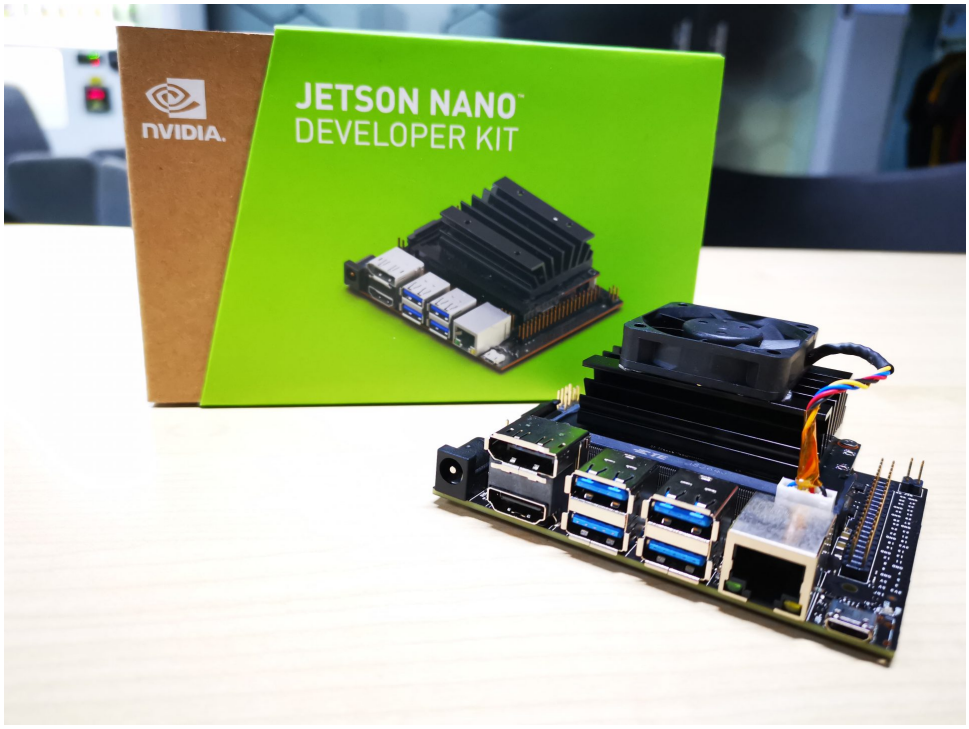
FIGURE 4 – Un Raspberry Pi modèle 3b

Après quelques recherches sur internet, il s'avère qu'il est possible de faire tourner une Donkey Car avec n'importe quel Raspberry Pi, mais que les performance sont très moindre en dessous de la version 3b.

Pour avoir les performance maximale, 2 options s'offrent à nous, la version 4Gb Ram ou la version 8Gb ram. Cette dernière est évidemment plus attrayante, mais également bien plus chère : 83.99€ contre 60.99€ pour la 4Gb. Le gain en performance ne justifie pas une telle augmentation du prix à nos yeux, nous nous contenteront de la version 4Gb si nous optons pour le Raspberry Pi comme micro-ordinateur.

2.2.2 Jetson Nano

Le Jetson nano est un des principaux concurrents du raspberry pi, surtout lorque l'on parle d'IA. Sa puissance réside dans la présence d'un GPU qui accélère grandement le traitement d'image et le calcul IA. Il possède également les même pin GPIO que le Raspberry Pi.



2 modèles de Jetson Nano sont disponnibles : avec 2 ou 4 Gb de ram, respectivement pour 59 et 89 euros.
Le gain de performance est vraiment très important par rapport à un raspberry pi, mais la version 2Gb de Ram n'est peut être pas suffisante, cela impliquera de passer directement à la version 4Gb du Jetson.

2.2.3 Autres options

Si nous nous tournous vers un rapsberry pi, nous avons également la possibilité de choisir d'améliorer ses performance à l'aide d'une "Compute Units". Ces sticks USB sont optimisé pour le calcul de modèle de machine learning et traitent les calculs à la place du Raspberry Pi.

FIGURE 5 – Deux type de compute stick (Intel Neural Compute Stick et Google Coral)

Ce genre de matériel est disponible entre 60 et 80 euros.

2.3 Capteurs

Les capteurs officiels Raspberry sont compatible avec le Jetson Nano, nous avons donc à notre disposition :

- Caméra Basique 8MPX 27.90€
- Lentille grand angle 27.60€
- Caméra 160° 16.90€
- Caméra 5MPX grand angle 34.95€
- Capteur LIDAR 157€
- Capteur ultrason 4.90€

Partir sur un système simple de caméra avec grand angle est l'option la plus économique de très loin. Mais la mise en place d'un système LIDAR est plus proche de la réalité des véhicules autonomes, et bien plus performant. La mise en place d'un capteur ultra reste peu onéreux et très utile dans la detection des obstacles.

3 Bilan Comptable

BILAN	Actif		Passif	
	Actifs	*Montants*	*Ressources*	*Montants*
	Actifs immobilisées	**Montant actifs immobilisées**	**Ressources propres**	**Montant ressources propres**
	caméra	35.00 €		
	micro ordinateur	60.00 €		
	voiture (moteurs, chassis)	90.00 €		
	batterie	30.00 €		
	pièces impressions 3D	20.00 €		
	carte SD	20.00 €		
	Actifs circulants	**Monant actifs circulants**	**Ressources externes**	**Montant ressources externes**
	banque	245.00 €	Budget Esaip	500.00 €
	TOTAL ACTIF :	**500.00 €**	**TOTAL PASSIF :**	**500.00 €**

FIGURE 6 – Bilan financement estimé de la voiture

Une fois la totalité des informations entré dans notre bilan comptable, nous pouvons remarquer qu'il nous reste 245€ de marge. Cet argent pourra être utilisé pour faire face aux imprévus ou alors pour passer à la game supérieur de micro ordinateur, de moteurs, ou encore ajouter un compute stick ou des capteurs suplémentaires.