Apprentissage par renforcement

Table des matières

1 Modélisation Mathématique

Un système d'apprentissage par renforcement se modélise grâce à un **Processus de décision Markovien (MDP)**.

1.1 Simulateur

Les données présentes sur le simulateur sont différentes de celles de la vie réelle. Voici une représentation du MDP de renforcement dans le simulateur :

On nomme $\mathcal{S}$ l'ensemble des états du simulateur. Ils sont de la forme :

$$\mathcal{S} = \begin{cases} image\ array \\ Pos\ :\ (x, y, z) \\ cte = \text{écart au centre de la piste} \\ speed \\ forward\ velocity \\ hit\ :\ None\ sauf\ si\ collision \\ gyro\ :\ (x_{gyro}, y_{gyro}, z_{gyro}) \\ accel\ :\ (x_{accel}, y_{accel}, z_{accel}) \\ vel\ :\ (x_{vel}, y_{vel}, z_{vel}) \\ last\ lap\ time \\ lap\ count \end{cases}$$

L'ensemble des actions $\mathcal{A}$ est de la forme $\mathcal{A} = \begin{cases} Steering \\ Throttle \end{cases}$

L'observateur $\mathcal{O}bs$ représente la donnée envoyée à l'agent de renforcement en fonction de l'état. Ici on a :
$\mathcal{O}bs = image\ array$

La fonction de Reward $\mathcal{R}$ représente la façon dont est calculée la récompense donnée à l'agent en fontion des choix effectués

$$\mathcal{R} = \begin{cases} -1.0\ si\ episode\ terminé \\ -1.0\ si\ CTE > maxCTE \\ \left[1 - \frac{|CTE|}{maxCTE}\right] * forward\ velocity\ si\ forward\ velocity > 0 \\ forward\ velocity\ sinon \end{cases}$$

La récompense est donnée après chaque step du simulateur

$$\text{Conditions de fin d'épisode : } \begin{cases} |CTE| > maxCTE \\ hit \neq None \\ \text{Sortie de la piste} \end{cases}$$

1.2 Réel

Dans les conditions réelles, beaucoup des informations fournies par le simulateur ne sont pas disponibles
Ainsi le modèle resemble plutot à :
$\mathcal{S} = image\ array$

$$\mathcal{A} = \begin{cases} Steering \\ Throttle \end{cases}$$

Problèmes :

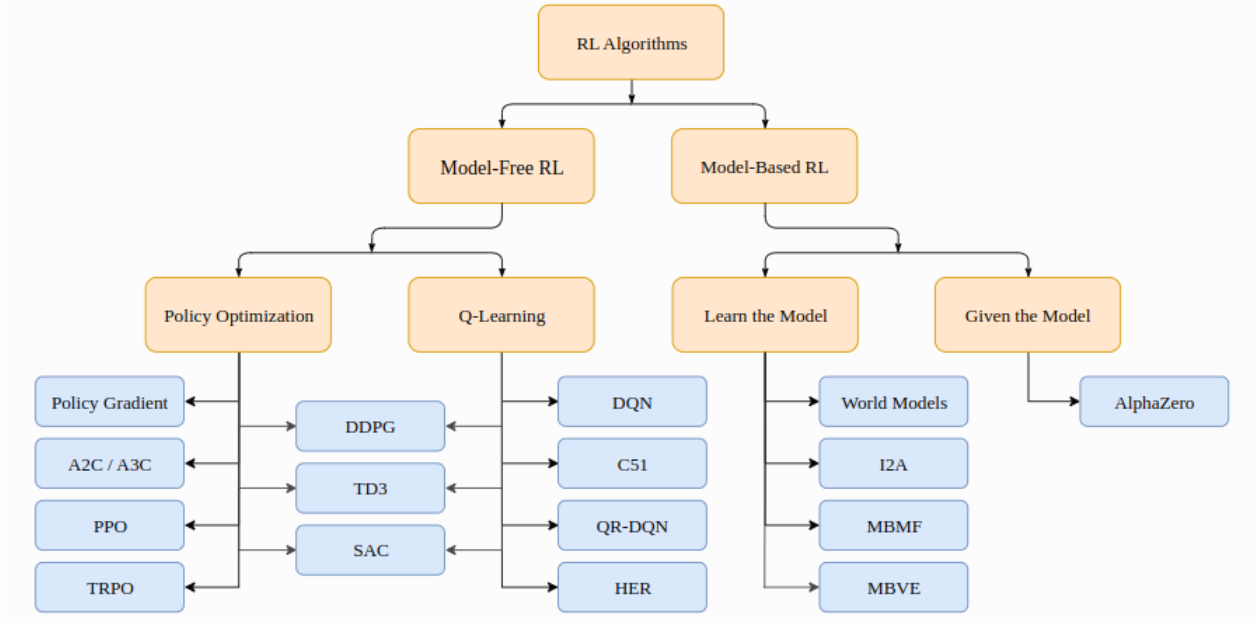
- Pas de CTE
- Fontion de Reward plus adaptée
- Conditions de fin d'épisode
- Quand donner la reward

Solutions :

- Observateur, preprocessing pour extraire les valuers manquantes dans la réalité
- Reward en fonction du temps de survie
- Revoir conditions de fin d'épisode
- Voir quelles valeurs peuvent être extraites de la voiture physique

2 Choix de l'algorithme

2.1 Recherches



En se renseignant, notamment sur ce site nous avons décidé de partir sur des algorithmes "Model-Free" et nous en avons selectionné 4 à tester :

- DQN

- TD3
- SAC
- PPO

2.2 Tests d'algorithmes

Afin de tester les differents algorithmes et choisir celui qui serait le mieux adapté pour notre cas, nous avons lancé un entrainement d'un modèle par renforcement sur le simulateur et avons laissé tourner les deifferents algorithmes pendant 30-45 mins. Nous avons ensuite observé leurs performances sur la piste (distance parcourue avant de sortir de la piste, combien de virages le modèle arrivait à prendre avant de sortir etc).

Cette méthode de test n'est pas très complète et ne nous permet pas forcément d'évaluer les performances des algorithmes à long terme, sur des séances plus longues d'entrainement mais pour 4 algorithmes différents, cela correspondait tout de même à 2h30 d'entrainement/observations.

2.3 Choix final

Suite à cette période de test, nous avons choisi d'utiliser le PPO de la librairie stable-baselines3

3 Entrainement

3.1 Choix de l'entrée

À ce stade du projet, l'équipe qui travaillait la computer vision avait fairt des progrès sur ses filtres et son extraction de données de la route. Ainsi la question s'est posée de savoir quelles informations on allait transmettre au modèle pour son entrainement. On avait donc 3 options :

- Donner l'image brute (On savait que ça marchait mais on voulait faire mieux pour un entrainement plus rapide)
- Donner l'image filtrée (avec les filtres de l'équipe computer vision)
- Donner l'ecart au centre de la piste (Avec la valeur exacte sur simulateur en esperant que le CNN du groupe computer vision pourrait donner la bonne valeur en reel)

3.2 Modifications du simulateur

Pour rendre l'entrainement plus efficace, nous avons rajouté 2 fonctions au simulateur:

- Une fonction qui arrête l'agent s'il roule trop lentement pendant un certain temps. Cela permet d'éviter qu'il roule trop lentement mais aussi la detection de collisions qui parfois ne sont pas reconnues.
- Une fonction avec du traitement d'image pour détecter les sorties de piste en detectant les croisements de lignes blanches.

```
def break_line(self):
    distance_from_reference_x = self.calculate_distance()

    if distance_from_reference_x==float("inf"):
        self.previous_distance = None
        self.previous_sign = None

    if distance_from_reference_x!=float("inf"):
        # Calculate the current sign of the distance
        current_sign = np.sign(distance_from_reference_x)

        # Check for continuity with the history of positions
        if self.line_position_history and not np.isclose(distance_from_reference_x,
        self.line_position_history[-1], atol=self.continuity_threshold):
            pass
        else:
            # If continuous, check for a line break
            if self.previous_distance is not None and current_sign != self.previous_sign:
                if np.abs(distance_from_reference_x) >= self.distance_threshold:

                    return True

            # Update the history
            self.line_position_history.append(distance_from_reference_x)

        # Update the previous distance and sign for the next iteration
        self.previous_distance = distance_from_reference_x
        self.previous_sign = current_sign

    return False
```

Figure 1: Fonction de croisement de ligne

3.3 Implementation

Nous avons utilisé le module stable-baselines3 qui implemente son propre algorithme PPO. Le modèle est généré par le module, ce qui nous a donné du mal à récupérer ses caractéristiques. Le code se situe dans le fichier ppo_train.py. En voici les lignes importantes :

```
import gym_donkeycar
import gym
from stable_baselines3 import PPO
import tensorflow as tf
import numpy as np

# Creation de l'environnement
env = gym.make(args.env_name, conf=conf)

#Chargement du modèle déjà entrainé (ou creation d'un nouveau)
model = PPO.load("ppo_donkey",env)
#model = PPO("CnnPolicy", env, verbose=1)


# Apprentissage sur un nombre total de timesteps
model.learn(total_timesteps=10000)
# On reset l'environnement
obs = env.reset()

# Enregistrement du modèle
model.save("ppo_donkey")
print("done training")

env.close()
```

Figure 2: Extrait de ppo_train.py

Il a aussi fallu modifier les fichiers du simulateur pour la fonction de reward. Celle de base dans le simulateur, que nous avonss utilisé dans un premier temps est celle-ci :

```
def calc_reward(self, done: bool) -> float:
    if done:
        return -1.0
    if self.cte > self.max_cte:
        return -1.0
    # Collision
    if self.hit != "none":
        return -2.0
    # going fast close to the center of lane yeilds best reward
    if self.forward_vel > 0.0:
        return (1.0 - (math.fabs(self.cte) / self.max_cte)) *self.forward_vel
    if self.over: return -1.0
    # in reverse, reward doesn't have centering term as this can result in some exploits
    return self.forward_vel
```

Figure 3: Fonction de reward

3.4 Variations

3.4.1 L'ecart au centre de la piste

Une idée a été de donner à l'agent son écart au centre de la piste comme donnée d'entrée. Cela n'a pas aboutit du tout, on ne détaillera pas cette approche. Le code ce trouve dans ppo_train_cte.py

3.4.2 Les filtres

Pour implémenter les filtres du groupe de computer vision, il a fallu modifier dans le simulateur la fonction d'observation. Ici on applique seulement un inrange :

```
img = self.image_array
TRESH3_MIN = np.array([0, 0, 0],np.uint8)
TRESH3_MAX = np.array([255, 120, 255],np.uint8)
observation=255-cv2.inRange(img, TRESH3_MIN, TRESH3_MAX,)[45:][:]
```

Figure 4: Filtre appliqué sur l'image

Il a aussi fallu modifier l'environnement pris par le modèle car l'image n'avait plus la même taille, ce code se trouve dans ppo_train_filters.py

```
# Creation de l'environnement
env = gym.make(args.env_name, conf=conf)

# Modification de l'espace d'observation
env.observation_space = gym.spaces.Box(0, 255, (75, 160), np.uint8)

#Chargement du modèle déjà entrainé (ou creation d'un nouveau)
model = PPO.load("ppo_donkey_filter",env)
#model = PPO("MlpPolicy", env, verbose=1)


# Apprentissage sur un nombre total de timesteps
model.learn(total_timesteps=10000)
# On reset l'environnement
obs = env.reset()

# Enregistrement du modèle
model.save("ppo_donkey_filter")
print("done training")

env.close()
```

Figure 5: Extrait de ppo_train._filter.py

L'entrainement avec ce filtre s'est avéré plus efficace que l'entrainement classique, on a donc décidé de rester dans cette voie

3.4.3 La fonction de reward

L'entrainement avec la fonction de reward de base était assez efficace. Cependant, étant donné que c'était surtout la vitesse qui était valorisée, l'agent roulait trop vite et avait beaucoup de mal à passer les virages serrés. On a donc décidé de rajouter dans la fonction de reward une valorisation du temps de survie de l'agent.

```
def calc_reward(self, done: bool) -> float:
    if done:
        return -1.0
    if self.cte > self.max_cte:
        return -1.0
    # Collision
    if self.hit != "none":
        return -2.0
    # Modification de la fonction de reward en rajoutant n_steps
    if self.forward_vel > 0.0:
        return (1.0-(math.fabs(self.cte)/self.max_cte))*(self.forward_vel+self.n_steps/25)
    if self.over: return -1.0
    # in reverse, reward doesn't have centering term as this can result in some exploits
    return self.forward_vel
```

Figure 6: Nouvelle fonction de reward

En prenant un agent qui s'était entrainé avec l'ancienne fonction de reward et en l'entrainant avec la nouvelle, il a pu ralentir et passer les virages serrés. Il a quand même fallu qu'on fasse atention car au bout d'un certain temps d'entrainement, l'agent comprenait que s'il allait très lentement, il gagnait plus de reward parce qu'il survivait plus longtemps. Au bout d'un temps d'entrainement nous avons un modèle qui arrive à faire des tours de piste en 15 secondes en moyenne.

4 Portage sur la voiture

Pour porter le modèle sur la voiture, il a fallu rajouter une part custom de donkeycar. Le détail est dans la documentation sur github. Nous avons repris le part créé par le groupe précédant et nous l'avons modifié pour notre modèle.

Nous nous sommes arrété là pour le renforcement, nous n'avons pas réussi à bien porter le modèle, le part custom renvoyait bien des valeurs de throttle et angle mais la communication entre le part et la voiture ne s'effectuait pas bien et nous n'avons pas pu résoudre le problème.