Amaury Colin
Jhon Munoz
Malick Ba

Romain Tessier
Pierre Gaillard
Wyatt Marin

Document de passation
Conduite autonome

Objectifs du document

Nous sommes 2 équipes de trois étudiants ayant participé à l'édition 2023 de la course IA Racing en mettant en place une voiture de course autonome s'appuyant sur une conduite par apprentissage supervisé et pas renforcement.

Ce document a pour but de décrire notre organisation et notre démarche scientifique lors du développement de nos voitures autonomes. La partie technique et intégration est disponible sur github.

Objectifs du document **1**
I - Contexte du projet et organisation **3**
Course de voiture portée par Sigma 3
Gestion du temps 3
Ce qu'il y a à faire 3
Lots 4
Outils mis en place 5
II - Start **7**
Formulation du problème 7
Choix 7
Implémentation (Schéma) 9
Expériences 9
Observations, solutions et pistes 9
III - Stop **11**
Formulation du problème 11
Choix 13
Implémentation 13
Expériences 14
Observations, solutions et pistes 15
IV - Drive Supervisé **17**
Formulation du problème 17
Choix 19
Implémentation 19
Expériences 20
Observations, solutions et pistes 22
V - Drive Renforcement **25**
Formulation du problème 25
Choix 26
Implémentation 28
Expériences 30
Observations, solutions et pistes 31
Limites 32

I - Contexte du projet et organisation

Course de voiture portée par Sigma

Ce projet s'inscrit dans une course de voitures autonomes organisée par le groupe Sigma. Lors du tournoi, plusieurs équipes de 3 personnes s'affrontent. Les équipes sont issues d'écoles d'ingénieurs de la région et d'employés de Sigma.

Gestion du temps

L'emploi du temps de la course est assez différent de l'emploi du temps école.
La course se déroule en juin alors que le projet se termine en mars. De fait, l'emploi du temps mis en place par Sigma est aligné sur cette date butoir.
Sigma dispense des formations et des ateliers, mais ils commencent un peu tard. Il est conseillé de bien prendre de l'avance et de rentrer dès que possible dans la réflexion des grandes étapes et leur mise en place.
Les formations Sigma n'apportent pas beaucoup d'éléments supplémentaires. Par conséquent, il est d'autant plus encouragé de commencer à travailler tôt.

Ce qu'il y a à faire

Pour IMT Atlantique

Pour IMT Atlantique, il y a plusieurs livrables à réaliser qui sont susceptibles d'évoluer selon les années. Certains sont individuels, d'autres collectifs. Dans tous les cas, l'école cherche à évaluer une organisation en mode projet. Il faut donc mettre en avant son aptitude à mettre en place une démarche projet avec une planification des tâches, un suivi, de la communication, des reportings…

Ainsi, il sera demandé de réaliser un Gantt pour fixer les jalons et faire un rétroplanning. Les différentes tâches devront être détaillées et leurs temps d'exécution devront être estimés en amont puis comparés aux temps effectivement réalisés une fois complétées. Un suivi des retards devra être fait. La charge de chaque membre des groupes devra aussi être estimée. Des points méthodologiques sont régulièrement organisés avec un intervenant extérieur, ce qui permet de remettre régulièrement à jour le Gantt et de s'aligner avec les attendus école.

Les encadrants du projet attendront également une formalisation du problème d'optimisation qui correspond à l'apprentissage. Pas d'inquiétude, ils sont là pour accompagner sa rédaction.

Le livrable final est un poster pour le forum ProCom (à destination des étudiants en 2A et 1A) au format A0. Attention, les délais d'impression sont de 1 semaine et nous avons été prévenus très tard de ce fait. Il faut donc bien anticiper ce livrable.

Pour Sigma

Nous pensons que Sigma organise cette course dans le but de recruter et d'obtenir de la visibilité sur les réseaux sociaux. En outre, la compétition se déroulera au Web2day, un événement important dans la communauté informatique. Cela donne donc de la visibilité et de l'attractivité à Sigma.

L'événement est sympathique et il est correct de rendre la pareille en étant présents aux différents temps organisés par l'entreprise. De plus, l'entreprise met à disposition une salle pour réaliser des essais sur piste, ce qui peut s'avérer pratique étant donné que les délais sont courts.

Il faudra envoyer à Sigma le nom des participants de chaque équipe de 3 personnes maximum.

Pour la course

Le jour de la course est pendant le stage de dernière année. S'engager dans le projet c'est donc faire le nécessaire pour participer le jour de la course (prendre un congé ou une disponibilité et être au clair avec l'entreprise de stage sur ce point).

Lors de la course les véhicules devront :

1) démarrer
2) rouler
3) s'arrêter après X tours

Des pénalités de temps sont accordées si le véhicule sort de la piste, s'il fait un faux départ ou ne s'arrête pas à la fin. Le démarrage au feu et l'arrêt ont été rendus facultatifs pour cette édition (et peut-être pour les suivantes) pour se concentrer sur la partie modèle de conduite supervisée.

Lots

Pour notre édition, nous étions 2 équipes de 3. Nous avons donc déterminé les phases communes à chaque équipe et les phases propres à chacune. Nous nous sommes organisés en 3 lots (qui correspondent aux 3 temps de la course) et avons partagé nos informations. Chaque lot a été intégré sous forme de part au framework donkeycar (voir partie II).

Bien que certains lots aient un caractère facultatif, nous avons été prévenus de la non-nécessité de les mettre en place seulement une fois nos travaux bien engagés (écart dans les projections temporelles Sigma vs IMT Atlantique).

Lot START

Pour le lot start, il a été choisi de détecter le feu vert grâce à une plage de couleur HSV. L'image de la caméra est récupérée puis un masque est appliqué en fonction d'une plage de couleur (de vert clair à vert foncé) afin de trouver les zones vertes.

Lorsqu'on considère une zone de vert assez grande pour être le feu vert, on considère que la voiture est autorisée à démarrer.

Difficultés du lot :

- Trouver la plage de valeurs qui comprend le vert du feu
- Le feu n'est pas mis assez tôt à disposition par Sigma, ce qui ne permet pas de tester en condition réelles

Lot DRIVE

Le lot drive est celui qui diffère selon les équipes mais la compréhension de l'environnement dans lequel le modèle de conduite évoluera peut être réalisée en commun.

Lorsque les 2 équipes se sont séparées, nous avons choisi de mettre en place de la conduite supervisée (ce qui est aussi encouragé par Sigma) dans une équipe, et l'apprentissage par renforcement dans une autre équipe.

L'apprentissage supervisé consiste à entraîner la voiture sur les données prises en conduisant. Concrètement, une personne conduit, on enregistre, on entraîne le modèle avec les données de la conduite et on fait rouler la voiture sur le circuit. Le modèle apprend donc dans la situation où il évoluera par la suite.

L'apprentissage par renforcement entraîne le modèle à optimiser sa politique d'actions grâce à un des récompenses. L'apprentissage se base sur un grand nombre d'expérimentations, l'utilisation d'un simulateur est donc importante. En théorie, au terme de l'entraînement, le modèle pourra s'adapter à des situations diverses et nouvelles.

Lot STOP

Pour le lot stop, nous avons d'abord mis en place un modèle de classification (réseau de neurones) pour détecter la présence de la ligne de stop (qui marque la complétion d'un tour). Lorsque la ligne n'est plus détectée après N images, on considère qu'on vient de finir un tour. Un compteur de tours et une condition d'arrêt ont été ajoutés.

Cependant le lot a été repris avant la fin du projet pour adopter une détection similaire à la partie start car le modèle était trop lourd et demandait trop de ressources. Un meilleur choix de modèle pourrait éviter cela. Un des problèmes a aussi été que la détection de stop a été développée avec Pytorch tandis que les modèles de conduite l'ont été avec Keras (car le framework donkeycar inclut des modèles tensorflow) dans le cas de la conduite supervisée.

Difficultés :

- Réunir un dataset et labelliser (très très chronophage)
- Trouver la plage de couleur sur le modèle de plages de couleur

Outils mis en place

Gestion de projet

- **Gantt** (avec Gantt Project) : planification des tâches sur le long terme
- **Trello** : suivi de l'avancée des tâches

Partage

- **Google Drive** : réalisation des livrables et partage de fichiers
- **GitHub** : partage des codes et des fichiers

Communication

- **Messenger** : Messagerie instantanée permettant de garder le contact en dehors des rendez-vous journaliers et de communiquer sur nos avancement
- **Discord** : Messagerie qui nous a permis de communiquer avec les encadrants ainsi que l'autre équipe et de stocker des ressources utiles au projet. Sigma utilise aussi un serveur Discord pour échanger avec nous.
- **Mail** : communication formelle

II - Start

Formulation du problème

Le but de l'algorithme est de détecter lorsque le feux de départ passe au vert après avoir été rouge puis orange, et de démarrer la voiture à ce moment-là.

Choix

Comme le feu de départ sera un panneau LED, nous commençons par admettre qu'il ne sera pas trop soumis aux reflets, d'autant plus que la salle choisie le jour de la course sera, normalement, sans grandes baies vitrées ou fenêtres.

Par ailleurs, ce feu de départ est assez grand et en hauteur, d'autant plus que nous pouvons placer notre voiture à l'endroit où on le souhaite avant le début de la course, tant qu'elle reste derrière la ligne d'arrivée. Ainsi, il est considéré qu'en ne gardant que la moitié supérieure des images fournies par la caméra, le feu de départ occupera la majeure partie de celle-ci.

Avec toutes ses hypothèses et paramètres, nous avons décidé de ne pas simplement détecter le feu de départ vert mais de le détecter dès que celui-ci est rouge puis de vérifier quand il passe au vert. Ainsi, nous nous assurons une plus grande sécurité vis-à-vis du départ car un carré vert détecté dans le décor ne poussera pas la voiture à démarrer. La seule difficulté résultant de ce choix est le choix de la plage de couleur pour le rouge et le vert qui nécessitera un réglage fin.

Finalement, le programme va donc regarder chaque image fournie par la caméra et va tenter de trouver sur celle-ci le plus grand carré rouge possible et si celui-ci est le plus gros jamais vu, il est sauvegardé. Ensuite, toujours avec la même image, il va regarder le plus grand carré vert visible et si celui-ci correspond à quelques détails près au plus grand carré rouge trouvé alors c'est que c'est le feu qui est passé de rouge à vert et donc il faut démarrer la voiture.

Pour savoir si deux carrés sont les mêmes, on compare chacun de leurs 4 paramètres: leurs positions x et y, leurs hauteurs et leurs largeurs, en faisant ces calculs:

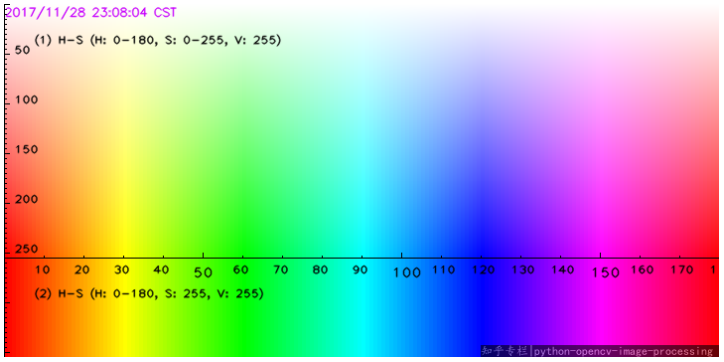
$$\frac{|x_2-x_1|}{img_{width}}, \frac{|y_2-y_1|}{img_{height}}, \frac{|w_2-w_1|}{img_{width}}, \frac{|h_2-h_1|}{img_{height}}$$

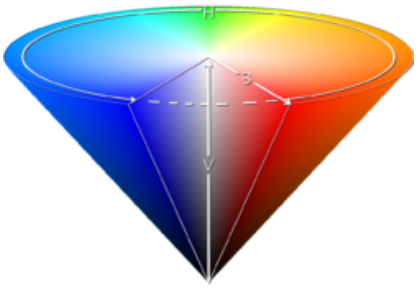
Avec:

- img_width la largeur de l'image
- img_height la hauteur de l'image
- xi, yi, wi et hi respectivement la position en x et y, la largeur et la hauteur du rectangle i

Par la suite, on calcule le maximum de ces 4 quotients et on regarde si celui-ci est bien inférieur à un seuil que nous avons prédéfini à 10% et si c'est le cas alors on considère que ce sont les mêmes rectangles.

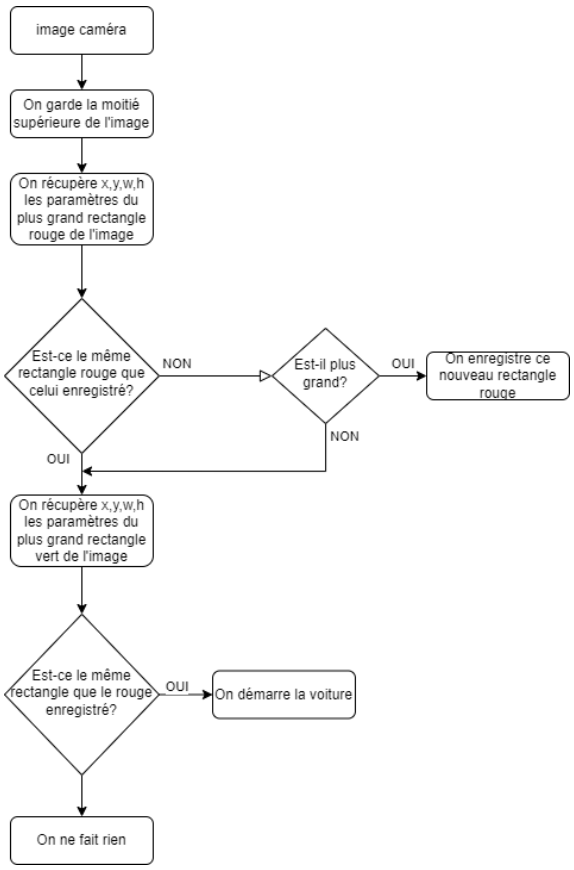
Pour détecter les rectangles de couleurs, on utilise le filtre inRange de la bibliothèque cv2 (OpenCV) qui prend en argument l'image et les valeurs minimales et maximales, exprimées en code HSV, de la bande de couleur que l'on cherche. Le code HSV est composé de 3 paramètres: la teinte (H), la saturation (S) et la valeur (V). Si on regarde sur internet, on peut trouver le type de diagramme suivant donnant le code HSV pour chaque teinte de couleur.





Ainsi, dans le cas de la couleur verte typiquement, nous avons une teinte qui est comprise entre 50 et 60. Comme nous pourrions avoir des reflets, la couleur pourrait tourner au blanc, ainsi, on pose une valeur minimale de saturation assez faible (50) et une valeur maximale de 255 pour avoir les couleurs les plus vives. Enfin, nous n'aurons pas de couleurs sombres puisque ce seront des panneaux LED, ainsi, le dernier paramètre, la valeur, est gardée élevée, entre 160 et 255.

Implémentation (Schéma)



Expériences

Il était difficile de tester notre programme de START puisque nous n'avions pas à disposition un feu de départ comme à la course, et ce même dans les locaux de SIGMA. Nous avons donc tout d'abord testé le programme avec la caméra de l'ordinateur en lui présentant une vidéo simulant le passage du rouge à l'orange puis au vert du feu de départ. Malheureusement, tout d'abord, l'écran de téléphone est très sensible aux reflets donc cela compliquait les tests même si, par conséquent, nous nous sommes mis loin des fenêtres et proche de la caméra. Ensuite, la caméra de l'ordinateur avait beaucoup de difficultés à détecter la couleur rouge contrairement à la couleur verte, surtout que les codes HSV que nous avions trouvés pour les deux couleurs, n'étaient pas ceux qu'il fallait utiliser réellement.

Nous avons, par la suite, testé directement le programme sur la voiture Donkey Car avec sa caméra pour vérifier que nous n'avions pas de problème du même type. Le mieux, évidemment, aurait été de tester dans cette condition mais cette fois-ci avec un vrai feu de départ et pas un téléphone.

Observations, solutions et pistes

Globalement, le programme semble bien marcher. Nous mettons "semble" car, comme dit précédemment, nous n'avons pas pu tester avec un vrai feu de départ. Cependant, en limitant les reflets et en mettant l'écran de téléphone en assez grand devant

la caméra, celle-ci réussit à bien détecter les rectangles de couleurs or, on sait que le feu de départ sera en très grand sur les images.

Bien évidemment, un axe d'amélioration serait le test en lui-même en obtenant un vrai feu de départ et notamment pour pouvoir bien ajuster les filtres pour détecter les couleurs. En effet, lors de nos tests, ces valeurs de filtres, notamment pour le rouge, ont été beaucoup modifiées pour au moins détecter les rectangles mais sans pour autant détecter trop de couleur. Ainsi, il faudrait tester dans les conditions réelles pour être sûr que ces filtres aient la bonne valeur.

III - Stop

Formulation du problème

Le but de ce lot est de détecter la présence d'une ligne jaune ou avec des caractéristiques particulières dans l'environnement. Cette section vise à formaliser en détail la première approche que nous avons eu, puis d'introduire la solution palliative que que nous avons adoptée.

Données :

- I : ensemble des images, on notera $I = I_s \cup I_{ns}$, où I_s est l'ensemble des images avec une ligne de stop et I_{ns} l'ensemble des images sans ligne.
- N : nombre d'images (cardinal de I)
- I_i : image i de dimensions (w, h, 3) où w est la largeur, h la hauteur, pour $i \in [[0, N-1]]$. Par défaut, w = 160 et h = 120.
- y_i : entier binaire (0 ou 1) associé I_i pour $i \in [[0, N-1]]$.

$$y_i = \begin{cases} 1 & \text{si } I_i \text{ a une ligne de stop} \\ 0 & \text{sinon.} \end{cases}$$

- f : la fonction indicatrice de I_s telle que $\forall i \in [[0, N-1]], f(I_i) = y_i$.

Choix des données :

A priori, notre objectif est seulement de déterminer la présence d'une ligne sans intérêt porté aux observations passées ($\forall i \in [[0, N-1]], f(I_i, I_{i-1}) = f(I_i)$). En ce sens, nous pouvons choisir les images aléatoirement dans notre dataset pour entraîner notre modèle.
Une autre version du modèle pourrait être de déterminer l'apparition de ligne, auquel cas le modèle serait récursif.

A chaque epoch, on choisit de constituer N_b batches de B_s images à l'aide de permutations aléatoires. On note I_j^k l'image $j \in [[0, B_s]]$ du batch k, $k \in [[0, Nb-1]]$.

On considère I équilibré, c'est à dire que $card(I_s) \approx card(I_{ns})$. Si I n'est pas équilibré, on se propose d'augmenter le nombre d'images de l'ensemble I_s ou I_{ns} ayant le cardinal le plus faible par oversampling, c'est-à-dire en lui ajoutant des images transformées du même ensemble (data augmentation). On peut aussi réaliser de l'undersampling en diminuant la taille de l'ensemble de plus grand cardinal.

Objectif :

On cherche à approximer la fonction f. On note x_i^k la prédiction de f(Ii) par le modèle.

f est à valeur dans {0,1}. Par conséquent, le choix d'une fonction de perte Binary Cross Entropy semble appropriée. On notera cette fonction de perte *l*. Ainsi, en notant X^k l'ensemble des prédictions sur les images d'un batch *k* et Y^k l'ensemble des valeurs attendues, l'objectif du modèle est :

$$\forall k \in [\![0, Nb-1]\!],\ min(l(X^k, Y^k)) = \frac{1}{B_s} \sum_{i=0}^{B_s-1} -\omega_i \cdot (y_i^k \cdot log(x_i^k) + (1-y_i^k) \cdot log(1-y_i^k))$$

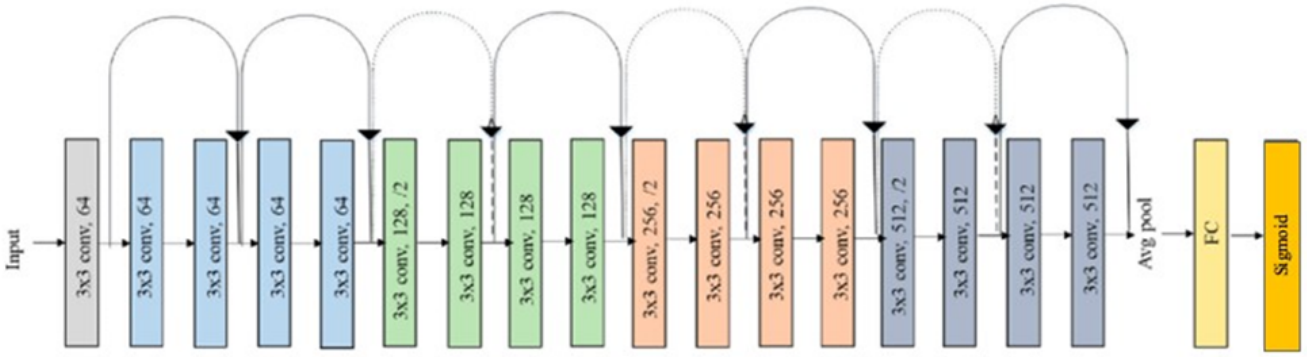
A chaque batch, on essaye d'optimiser la fonction de coût et on applique une rétropropagation pour ajuster les poids du modèle.

On notera ω_i dans l'expression de la fonction qui représente un poids accordé à chaque classe qui est calculé en fonction de la densité de classe afin de rééquilibrer l'ensemble *I* s'il n'est pas équilibré. Simplement, on peut définir les poids pour toutes les images sans ligne comme étant $card(I_n s)/card(I)$. On peut définir similairement les poids des images avec ligne.

Construction du réseau :

Le problème actuel s'inscrit dans les problèmes de type computer vision. On choisit donc une architecture ResNet, typique pour ce genre de sujet (car elle possède de nombreuses couches de convolution).

On souhaite que les prédictions soient comprises dans l'intervalle [0, 1]. On ajoute donc une couche dense après le average pooling de sorte à n'avoir qu'un seul neurone et on lui renseigne une fonction d'activation de type sigmoïde.



Architecture du réseau

On choisit d'entraîner l'ensemble du réseau à chaque entraînement.

Prise de décision :

Pour prendre une décision, on se fixe un α correspondant à la marge de certitude que l'on souhaite se fixer pour les prédictions.

Ainsi, pour une prédiction *x*, la décision *d* est définie telle que :

$$d \quad \begin{cases} 1 & \text{si } x > 1 - \alpha \\ = & \\ 0 & \text{sinon.} \end{cases}$$

Si *d* = 1, on considère qu'il y a une ligne de stop sur l'image, sinon on admet l'absence de ligne.

Choix

Cette méthode permettait de s'initier à l'intelligence artificielle en vue d'aller plus loin dans le sujet avec la partie Drive.

Le choix des données d'entrée est imposé par la seule information que nous puissions facilement récupérer : l'image de la caméra implémentée sur la voiture. Il a été choisi de considérer le problème comme un problème de classification binaire car l'information que nous souhaitons extraire est binaire : présence de ligne (VRAI) vs absence de ligne (FAUX).

Dès lors, le choix de la fonction de perte Binary Cross Entropy est évident étant donnée la nature du problème.

L'architecture ResNet a été choisie suite à la lecture de résolution de plusieurs problèmes de classification binaire sur le web. Elle semble être commune pour les problèmes de type computer vision. Bien que le choix de l'architecture doit être indépendant de la technologie de mise en œuvre, on notera qu'avec le framework PyTorch, il est très simple d'importer un modèle de type Resnet.

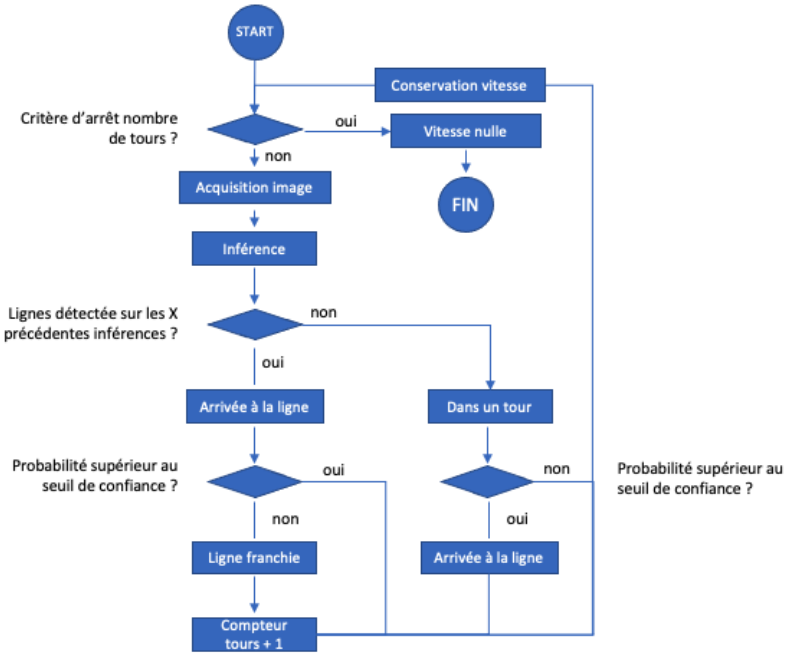
Le choix d'ajouter une fonction d'activation de type Sigmoïde en sortie résulte de la classification binaire attendue.

Implémentation

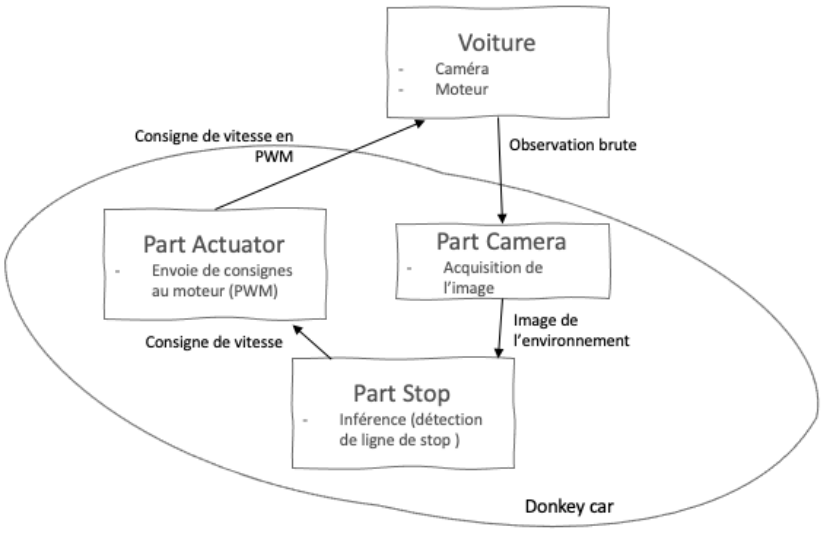
Comme nous l'avons introduit, le modèle réalise une inférence à partir d'une observation de l'environnement acquise par la caméra. Il retourne une probabilité sur la présence d'une ligne de stop dans l'acquisition de l'environnement. Alors, en fonction d'un seuil de décision, la voiture pourra conclure sur la présence ou non d'une ligne.

Lorsque la ligne n'est plus détectée alors qu'elle l'était avant, on peut considérer que la voiture a franchi la ligne. Elle vient donc de conclure un tour.

Le nombre de tours étant le critère d'arrêt du véhicule pendant la course, on peut donc considérer qu'il y a une seconde décision pour arrêter le véhicule. En réalité, cette prise de décision peut être réalisée avant celle de la détection de ligne. En effet, si le critère d'arrêt n'est pas atteint, il faut savoir s'il y a une ligne ou non. Sinon, on peut arrêter la décision.



Dans le contexte du projet, l'algorithme prendrait la forme d'une part qui s'implémenterait tel qu'illustré ci-après.



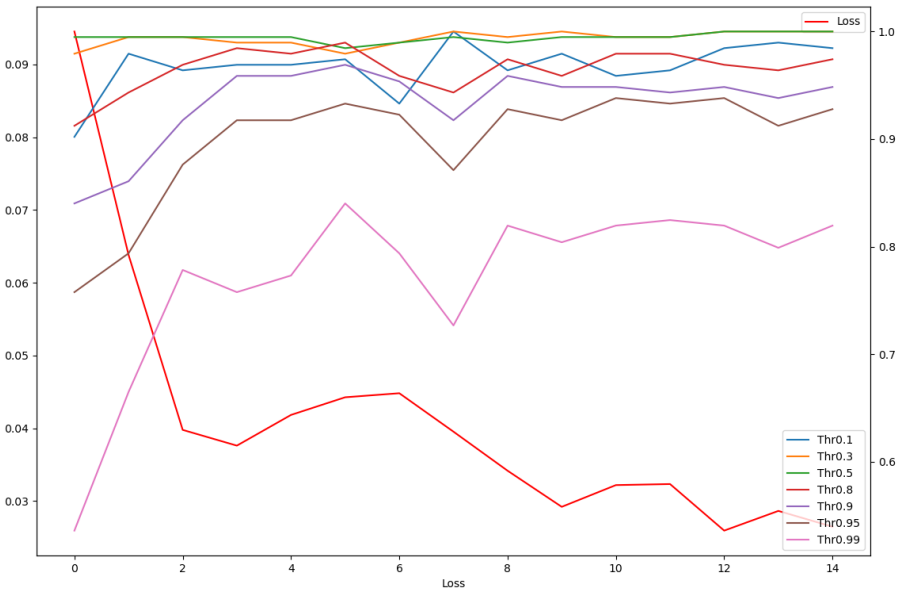
Expériences

Lors des entraînements, nous avons utilisé des images prises en conditions réelles. Les résultats obtenus sont très satisfaisants.

Les expériences réalisées ont été faites sur un dataset d'images réelles équilibré au regard de la présence ou non d'une ligne de stop.

La figure suivante montre le taux de succès dans la détection de la présence et l'absence de lignes en fonction de différents niveaux de confiance en fonction des epochs d'entraînement. La courbe descendante représente la valeur de la fonction de coût à chaque

epoch. Les autres courbes représentent le taux de succès à détecter la présence et l'absence d'une ligne sur un dataset de validation indépendant de celui de l'entraînement.



Observations, solutions et pistes

Observations

Nous avons observé que le rognage de l'image pour éliminer les données au-dessus de l'horizon peut améliorer légèrement la détection de la ligne de stop. En effet, puisque la ligne de stop se trouve sous l'horizon, il peut être bénéfique de retirer des données inutiles.

On observe que le taux de succès croît avec le nombre d'epochs d'entraînement. L'observation de la fonction de coût semble indiquer que nous n'avons pas encore atteint l'overfitting.

Ensuite, on observe que le taux de succès est presque parfait pour des seuils de confiance faibles et diminue lorsque le taux de confiance augmente. En effet, plus le taux de confiance est élevé, plus on demande une probabilité d'avoir une ligne élevée pour conclure à sa présence. Ainsi, on préfère dire qu'il n'y a pas de ligne plutôt qu'il y en a une. À l'inverse, un taux de confiance faible consiste à dire que si on n'est pas sûr de l'absence d'une ligne, alors on pourra considérer qu'il y en a une. Un taux de confiance à 50% correspond à l'entre-deux et semble afficher les meilleurs résultats. Un taux de confiance à 30% présente les mêmes résultats qu'à 50%.

Peu importe le seuil de confiance choisi, on observe des résultats corrects à plus de 90% des inférences, exception faite du seuil de confiance à 99% (qui est très haut) qui obtient des résultats corrects à environ 80% des inférences.

Conclusion sur les observations

Bien que les résultats semblent optimaux pour un seuil de confiance à 0.5, on aura tendance à choisir un seuil de confiance plus élevé pour s'assurer de la présence d'une ligne. Ainsi, on pourra éviter les faux positifs et on ne risquera pas de compter des tours en trop et de s'arrêter trop tôt.

Problème constaté

Malgré les bonnes performances de la solution proposée, il ne nous a pas été possible de la mettre en place. En effet, lors de son implémentation et de son exécution sur la voiture, nous avons observé une chute des performances de cette dernière dans ses tâches essentielles (avancer, tourner…).

Cette approche est possiblement trop gourmande en ressources pour la Raspberry Pi et le choix d'un ResNet est peut-être trop lourd pour un système embarqué. L'exécution de l'inférence est aussi possiblement trop longue, ce qui résulterait en un non-respect de la fréquence d'exécution de la voiture.

Solution proposée

Pour pallier ce problème de ressources, une solution déterministe peut être mise en œuvre similaire au lot START, avec la recherche d'une plage de couleur sur une image.

Cette recherche de couleur sur une image remplacerait l'inférence du modèle et son implémentation serait la même que celle présentée en amont pour le lot stop.

Après avoir mis en place cette solution, nous pouvons conclure que les résultats sont plus mitigés car ils dépendent des conditions ambiantes (par exemple la luminosité). Le réglage de la plage de couleur d'un jour ne sera pas forcément le même que celui d'un autre jour. En outre, régler la plage de couleur peut être laborieux. En effet, s'il n'est pas trop difficile de trouver une plage de couleur dans laquelle se trouve le jaune de la ligne de stop, il est plus complexe de s'assurer que la plage n'est pas trop large. Le cas échéant, nous pourrions compter des franchissements de ligne lorsqu'il n'y en a pas, et donc risquer de nous arrêter trop tôt.

L'expérience a montré qu'en utilisant des outils adaptés (permettant de tester la plage de couleur sur l'ensemble de la course) on peut trouver une plage de couleur cohérente et suffisamment précise pour ne détecter que la ligne de stop (plus d'infos sur le Github du projet). Ces réglages doivent être fait avant la course pour coïncider le plus possible avec les conditions ambiantes de la course.

IV - Drive Supervisé

Formulation du problème

Contexte :

Le but de l'algorithme est de déterminer l'angle et la vitesse à adopter à l'approche d'une situation donnée. Dans la situation actuelle, on considère un apprentissage supervisé, c'est-à-dire qu'initialement, en amont de tout apprentissage, il y a un humain qui pilote la voiture. Nous appellerons cet humain conducteur. Lors de la conduite de la voiture par le conducteur, des données sont enregistrées. On considérera donc que les comportements enregistrés dans des situations données sont à l'initiative du conducteur, même si les paramètres enregistrés sont ceux de la voiture (vitesse, angle. . .).

Données :

- H : dernière période de l'horizon d'enregistrement. H dépend du temps d'enregistrement total D et de la fréquence f d'images enregistrées par seconde par la formule $H = D \cdot f$.
- T : ensemble des périodes temporelles d'enregistrement. On a donc $T = [\![0, H-1]\!]$
- I : ensemble des images enregistrées. $I = I_e \cup I_v$
- I_e : ensemble des images qui serviront pour l'entrainement du modèle.
- I_v : ensemble des images qui serviront pour la validation du modèle.
- I_t : image à l'instant $t \in T$ de dimensions (w, h, 3) où w est la largeur et h est la hauteur. Par défaut, w = 160 et h = 120.
- v_t : vitesse adoptée dans la situation de l'image I_t par le conducteur, $v_t \in [-1, 1]$ pour $t \in T$. Une vitesse négative correspond à un recul et une vitesse positive à une avancée.
- $\hat{v}_t$: vitesse à adopter dans la situation de l'image I_t, $\hat{v}_t \in [-1, 1]$ pour $t \in T$. Les $\hat{v}_t$ suivent les mêmes conventions d'orientation que les v_t.
- θ_t : angle adopté dans la situation de l'image I_t par le conducteur, $\theta_t \in [-1, 1]$ pour $t \in T$. Un angle positif correspond à un virage à droite et un angle négatif à gauche.
- $\hat{\theta}_t$: angle à adopter dans la situation de l'image I_t, $\hat{\theta}_t \in [-1, 1]$ pour $t \in T$. Les $\hat{\theta}_t$ suivent les mêmes conventions d'orientation que les θ_t.
- γ : fonction renvoyant le comportement adopté par le conducteur dans une situation donnée.

$$\begin{array}{rccc} \gamma & : & I & \to & [-1,1] \times [-1,1] \\ & & I_t & \mapsto & (v_t, \theta_t) \end{array}$$

C'est la fonction que l'on cherche à approximer avec notre modèle dans ce problème.

Note : l'angle et la vitesse sont normés par rapport aux amplitudes maximales du véhicule.

Choix des données :

Notre objectif est de déterminer le comportement à adopter dans une situation donnée. Dans un premier temps, on décide de ne pas porter d'intérêt aux comportements passés car on s'intéresse à la valeur de la consigne passée aux servomoteurs par la voiture à l'instant $t \in T$. Par exemple, si la voiture est en train de tourner à l'instant $t-1 \in T$ avec un angle θ_{t-1} et que l'on souhaite continuer la rotation avec le même angle, alors il faut conserver une consigne $\theta_t = \theta_{t-1}$ à $t \in T$.
En posant cette hypothèse de prédiction sans historique, nous pouvons alors choisir les images aléatoirement dans notre dataset pour entraîner notre modèle.

On choisit de constituer aléatoirement des batches de B_s images (ce qui forme alors N_b batches). On note I_j^k l'image $j \in [\![0, B_s - 1]\!]$ du batch k, $k \in [\![0, N_b - 1]\!]$.

Objectif :

On cherche à approximer la fonction γ. On note $\hat{y}_j^k = (\hat{v}_j^k, \hat{\theta}_j^k)$ la prédiction de $\gamma(I_j^k)$ par le modèle. On notera $y_j^k = \gamma(I_j^k)$ le comportement réel adopté par le conducteur dans le contexte de l'image I_j^k.

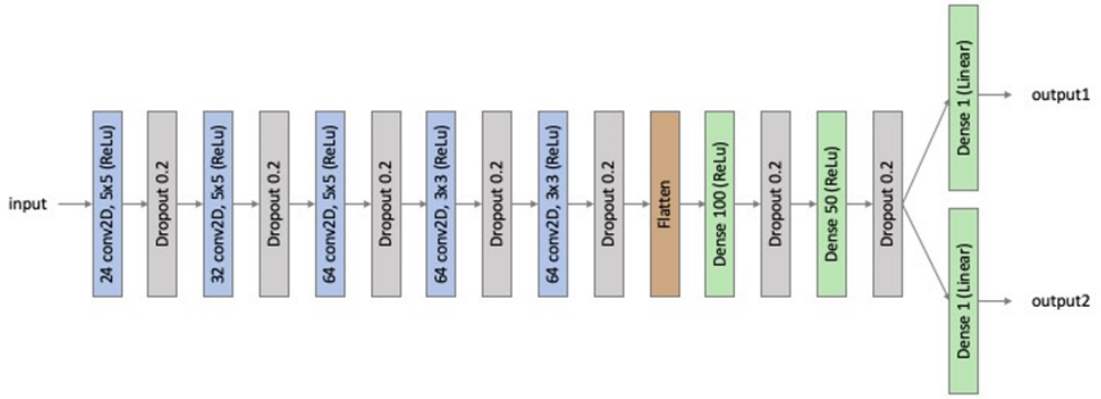
γ est à valeur dans $[-1, 1] \times [-1, 1]$. Par conséquent, le choix de l'erreur quadratique moyenne (mean squared error) comme fonction de perte semble appropriée. On notera cette fonction de perte l. Ainsi, en notant $\hat{Y}^k$ l'ensemble des prédictions sur les images d'un batch k et Y^k l'ensemble des comportements réels pour ces mêmes images, l'objectif du modèle est :

$$\forall k \in [\![0, Nb - 1]\!],\ min(l(\hat{Y}^k, Y^k)) = min(\frac{1}{B_s} \sum_{i=0}^{B_s - 1} \frac{(v_i^k - \hat{v}_i^k)^2 + (\theta_i^k - \hat{\theta}_i^k)^2}{2})$$

On appelle epoch le parcours de l'ensemble de nos batchs constitués par I_e ou I_v. À l'issue de chaque epoch d'entrainement, on a parcouru l'ensemble de nos images d'entraînement et on essaye d'optimiser la fonction de coût par rétro-propagation pour ajuster les poids du modèle.
Afin d'éviter l'overfitting ou surapprentissage (c'est-à-dire que le modèle soit trop spécifique au problème pour les données d'entrainement seulement), on considère que l'entrainement s'arrête lorsque la valeur moyenne de l sur une epoch est identique aux précédentes (ou très similaire). À l'issue de chaque epoch d'entraînement, on parcours l'ensemble des images de validation I_v et on calcule la fonction de perte. Il n'y a pas de retropropagation mais cela permet de se rendre compte de l'overfitting lorsque les valeurs de la fonction de perte se dégradent significativement sur l'epoch (et sur les suivantes).

Construction du réseau :

Pour supporter le problème actuel, on utilise la structure de réseau présenté ci-après. On choisit d'entraîner l'ensemble du réseau à chaque entraînement.



Architecture du réseau du Lot Drive supervisé

Remarque : nous ne l'avons pas fait, mais il aurait été intéressant de mettre une fonction d'activation à valeurs dans [-1, 1] pour les neurones de sortie du réseau.

Prise de décision :

On considère que pour une image I_t, $t \in T$, la valeur de l'un des neurones de sortie du modèle (arbitrairement (tant que c'est le même choix pour tous les entrainements et inférences) le premier) correspond à la vitesse prédite $\hat{v}_t$ et la valeur de l'autre correspond à l'angle prédit $\hat{\theta}_t$.

Si la valeur de sortie d'un des neurones est supérieure à 1, on considère que sa valeur est 1. Si la valeur de sortie d'un des neurones est inférieure à -1, on considère que sa valeur est -1. Sinon, on considère la valeur telle qu'elle est pour accélérer ou tourner.

Choix

Initialement, nous avions décidé de créer un modèle maison pour répondre au lot drive supervisé. Après plusieurs tests sur simulateur, il s'est avéré que le modèle ne présentait pas de bonnes performances (défaut de direction, le véhicule tournait trop tôt). Bien que pour ce premier modèle l'architecture était différente de celle présentée ci-dessus, les hypothèses et choix de données étaient similaires.

Finalement, pour le lot Drive supervisé, nous avons choisi de ne pas créer le modèle depuis zéro mais de partir sur une base existante dans le framework python donkeycar. Nous avons préféré montrer notre bonne compréhension du problème d'optimisation plutôt que de passer du temps à le développer.

Le choix a donc été fait de principalement travailler sur les données d'entrées du modèle en cherchant à optimiser la perception de l'environnement, notamment à travers du preprocessing des images. Alors, nous avons testé un ensemble de preprocessings et réalisé plusieurs tests afin d'observer les différences et similarités et conserver celui qui permet au modèle d'être le plus pertinent sur circuit.

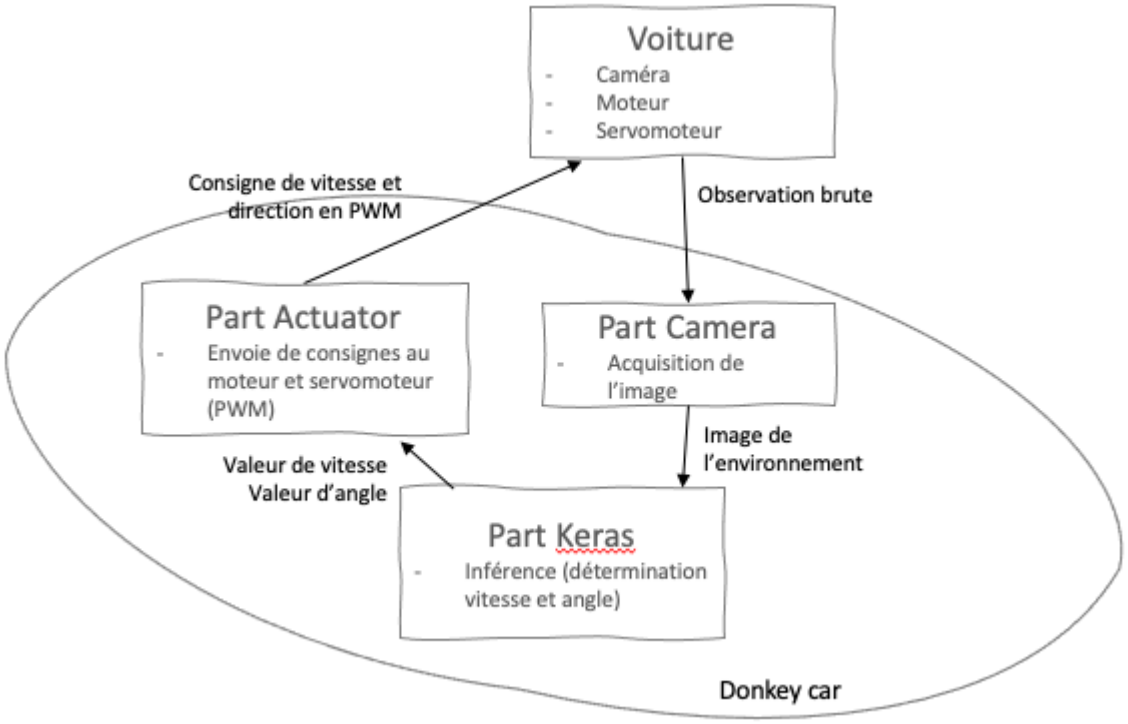
Implémentation

Le modèle réalise une inférence à partir d'une observation de l'environnement acquise par la caméra. Il retourne alors une valeur sur 2 neurones, l'une correspondant à la vitesse et l'autre à la direction. Ces valeurs seront utilisées pour transmettre les consignes aux moteur et servomoteur (en PWM).

La capture de l'image depuis la caméra est réalisée par la part Camera du framework python donkeycar qui repose sur le package picamera.

L'inférence est réalisée par la part Keras du framework donkeycar.

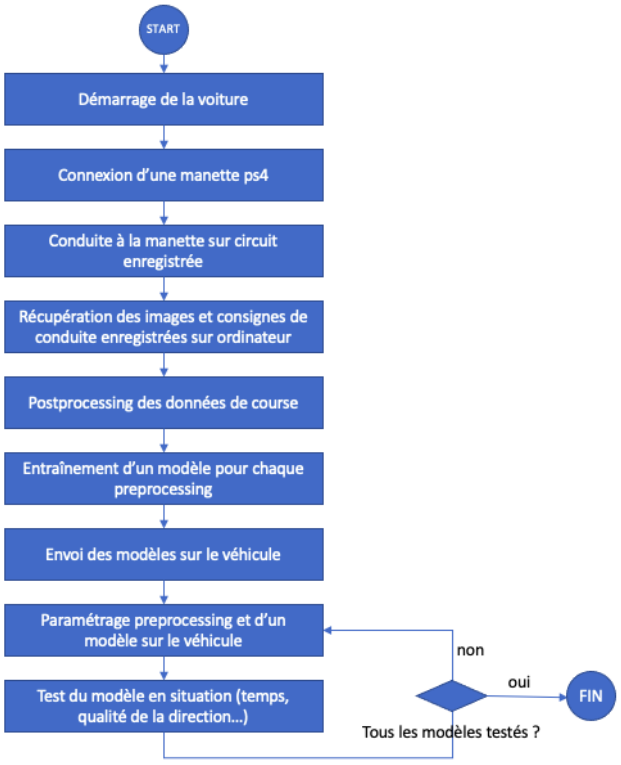
La transmission des consignes aux moteur et servomoteur est réalisée grâce à la part Actuator du framework donkeycar.



Expériences

Pour réaliser nos expériences, nous sommes allés sur site pour tester les modèles en conditions quasi-réelles (car le circuit de test est plus petit que le circuit de la course).

À chaque expérience, nous avons répliqué le processus suivant.

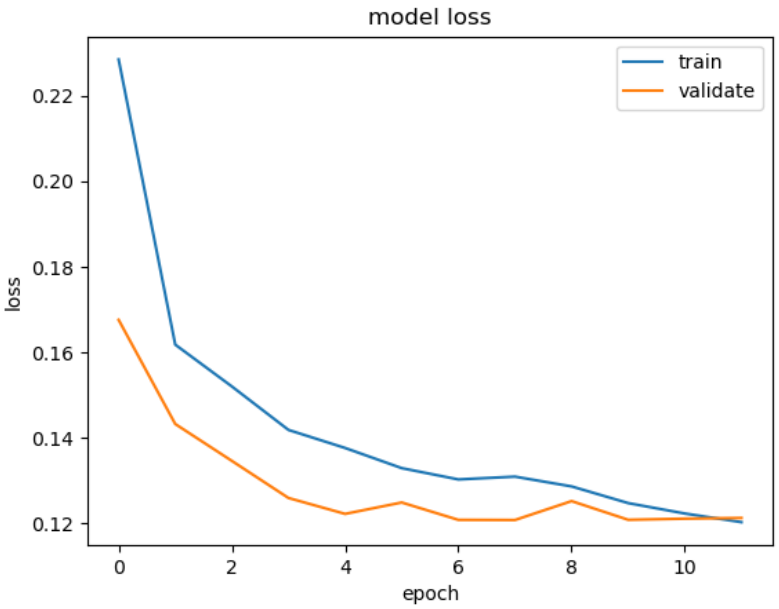


Ainsi les données utilisées pour les tests sont celles du jour du test. Donc on peut conclure sur la qualité de notre preprocessing et modèle indépendamment (ou avec moins de dépendance) des variations de luminosité entre le moment de collecte des images d'entraînement et lors de la course. Par ailleurs, le jour de la course, nous aurons accès au circuit avant la course pour réaliser ces prises d'images.

Pour optimiser notre conduite, nous connectons une manette de ps4 pour mieux conduire qu'avec la manette fournie par défaut ou le contrôleur web. Lors de la course nous essayons d'être le plus fluide possible et de ne pas trop sortir des lignes. Nous avons également réalisé des tests sur des conduites déplorables, des mélanges de bonnes et mauvaises conduites, etc. pour tester les limites du modèle. Chaque course est automatiquement enregistrée dans un répertoire propre trié en fonction de la date (qu'on appelle tub).

Les données que nous récupérons de la course sont les images et les vitesses et directions à chaque prise d'image. Ensuite, nous les traitons ensuite sur nos terminaux avec les preprocessing que nous sommes susceptibles de tester par la suite. Ainsi, nous conservons la donnée brute intacte. Le choix de faire du post-processing les images vient du risque que traiter une image sur la Raspberry Pi lors de la conduite avec plusieurs preprocessing prennent plus de temps que celui alloué entre la prise de deux images, et donc crée une latence dans la conduite du véhicule qui se répercuterait sur les vitesses et directions, et par conséquent, finalement, sur l'entraînement.

Nous entraînons donc la conduite sur les images, vitesses et directions pour chaque preprocessing. L'entraînement nous génère une figure telle que présentée sur la figure suivante représentant les valeurs de la fonction de coût en fonction du nombre d'epochs sur l'entraînement.



Nous renvoyons le modèle sur la Raspberry Pi et nous faisons un test.

Observations, solutions et pistes

Pour chaque test, nous évaluons la vitesse, le suivi de trajectoire et le nombre de tours effectués. Nous classons les résultats dans une matrice.

	Entrainements						Résultats DRIVE			
Nom du modèle	Nom du Preprocessing	Taille images	Coach	Nbs de batchs	Nbs epochs	Nbs images	Vitesse (/5)	Suivi de trajectoire (/5)	Nombre de tours	Cor
linear_160x80_v2-1-02_03+09_03.tflite	None	160*80	Malick	20	18	1400	4	2	/	Trop bo
linear_160x80_lines_50_50_v1.tflite	Lines	160*80	Amaury	50	50	16800	3	3	2+	Dépen

Lors de l'évaluation, on observe sans surprise que les résultats du modèle dépendent de la qualité de la conduite dont les données d'entraînement sont issues. Cependant, il existe des observations générales que l'on retrouve quelque soit la conduite.

Observations sur les résultats de l'entraînement

Sur la figure illustrant les valeurs de la fonction de coût sur les données d'entraînement et de validation en fonction du nombre d'epochs de l'entraînement, nous remarquons que le coût diminue. Ainsi le modèle apprend de manière cohérente avec le temps. Étant donné le caractère assez variable des décisions qu'on peut prendre en conduisant, une erreur moyenne de 12% semble acceptable. En effet, en mettant un humain 2 ou 3 fois dans la même situation de conduite, il est peu probable qu'il fasse exactement les mêmes choix au même moment.

On observe également qu'on évite l'overfitting car le coût de perte sur le dataset de validation ne se dégrade pas. Cela est dû à l'early stopping qui fait que lorsque la fonction de coût n'évolue presque plus, l'entraînement s'arrête.

Observation sur l'adaptation de la vitesse

Dans l'ensemble des modèles, on observe que le véhicule retranscrit bien la conduite d'un humain : il accélère dans les lignes droites ou dans la seconde moitié des virages lorsque nous avons adopté ce comportement (car on peut choisir de rouler à pleine vitesse tout le temps ou bien de ralentir à l'approche des situations difficiles). En particulier, à l'approche des virages serrés, le modèle est capable de ralentir. Cependant, le modèle ralentit parfois trop pour pouvoir prendre les virages. Dans ce cas, il faudrait post-traiter la commande de vitesse et la forcer à être supérieure à un certain seuil, sauf pour l'arrêt final.

Les résultats sont donc satisfaisants pour la vitesse même s'ils ne sont pas incroyables pour autant. Les allures que nous avons réussi à avoir restent modérées et ne dépassent pas celles que nous avions lors de la conduite.

Observations sur la direction

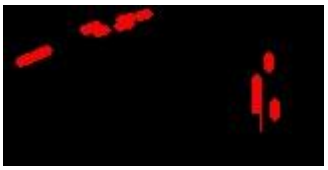
Dans l'ensemble le véhicule adopte une direction plutôt certaine (peu d'hésitation) et cohérente au regard des situations quel que soit le preprocessing. En revanche, nous avons noté un certain nombre de sorties de piste dont les causes semblent être les conditions ambiantes de luminosité. En effet, par moment, l'image renvoyée par la caméra est blanche à cause de reflets trop importants et difficiles à traiter. Alors, même pour un humain, il serait parfois impossible de percevoir certains virages.

Nous avons essayé d'entraîner le modèle à revenir sur la piste mais les performances constatées sont médiocres. Il est possible que cela soit dû au fait que les couloirs de course sont proches sur le circuit, si bien que lorsque le modèle sort, il ne sait pas lequel rejoindre ou bien s'il est entre les lignes d'un couloir ou bien entre deux couloirs.

Ainsi, ces observations remettent en question certaines hypothèses que nous avons faites lors de la formalisation. En effet, il est compliqué pour le modèle de déterminer sans mémoire de ses décisions passées où aller lorsqu'il est aveugle (perception de son environnement avec une image blanche) ou quel comportement adopter pour revenir sur la piste. Il pourrait donc être intéressant de remettre en perspective l'hypothèse d'indépendance des situations lors de l'entraînement du modèle. L'introduction de récurrence pour pallier ce problème semble être une hypothèse à évaluer.

Observation sur le preprocessing

La principale observation que nous avons faite sur les preprocessings est un peu décevante. En effet, nous avons constaté qu'un preprocessing qui nous semblait bon car synthétique (affichage des lignes seulement) n'était pas nécessairement synonyme de meilleures performances.



En effet, il est probable que lors du preprocessing, on injecte un biais. Ainsi, à l'approche de la ligne de stop, on remarque que le modèle entraîné avec le processing ci-dessus ralentit car il doit percevoir des lignes horizontales qui pourraient indiquer un virage à angle droit sans le contexte global (couleur, épaisseur de la ligne…). Bien évidemment, nous ne pouvons que faire des hypothèses et des interprétations du comportement du modèle.

En outre, nous avons remarqué que les modèles peu preprocessés (flou gaussien et rognage seulement) présentent des résultats supérieurs à nos modèles entraînés sur des images preprocessées, et ce pour tous les preprocessings.

Conclusion

Pour conclure, la principale limite de la stratégie conduite supervisée est qu'elle dépend principalement de la conduite fournie à l'entraînement. Il faut donc un bon pilote pour générer de bonnes données et entraîner un modèle performant.

En outre, le modèle ne connaît que les situations sur lesquelles il a été entraîné. Il arrive à s'adapter à certaines situations légèrement nouvelles mais si le modèle n'a été entraîné qu'à tourner à gauche, alors il ne comprendra pas un virage à droite (observé sur la piste ovale du forum Projet 3A).

Par ailleurs, puisque la seule source de perception que l'on a de l'environnement est la caméra, on est limité par la donnée entrante qui peut être de mauvaise qualité à cause des conditions ambiantes (reflets). Les conditions de la course devraient être plus acceptables. Entraîner le modèle avec des données enregistrées dans les conditions ambiantes permet de partiellement pallier ce problème. En outre, l'introduction de récurrence dans la définition du modèle semble être une bonne idée pour améliorer le comportement du véhicule à l'approche de situations perçues de manière dégradée.

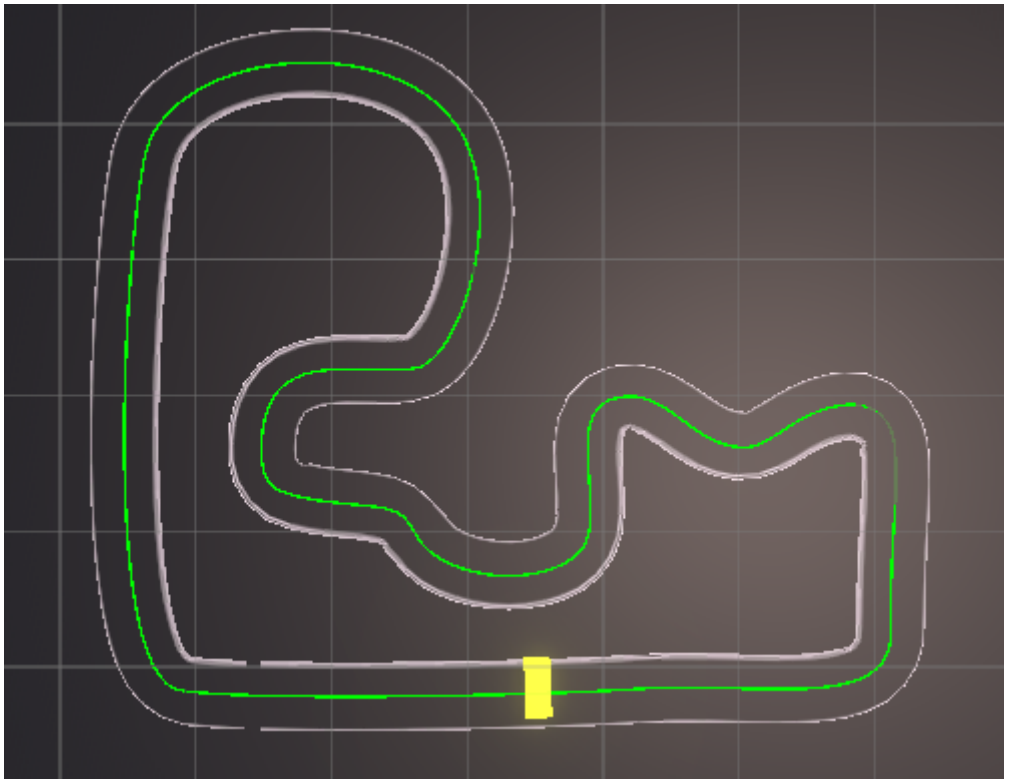
Ainsi, si les conditions sont bonnes le jour de la course, et avec le temps de mise à disposition du circuit en amont de la course pour entraîner son modèle, on pourra avoir de bons résultats.

Enfin, un preprocessing ne fait pas tout et rogner l'image apporte parfois de meilleures performances. Il faut être bien conscient que preprocesser ses images, c'est ajouter du biais aux données, sans savoir lequel et l'impact qu'il aura.

V - Drive Renforcement

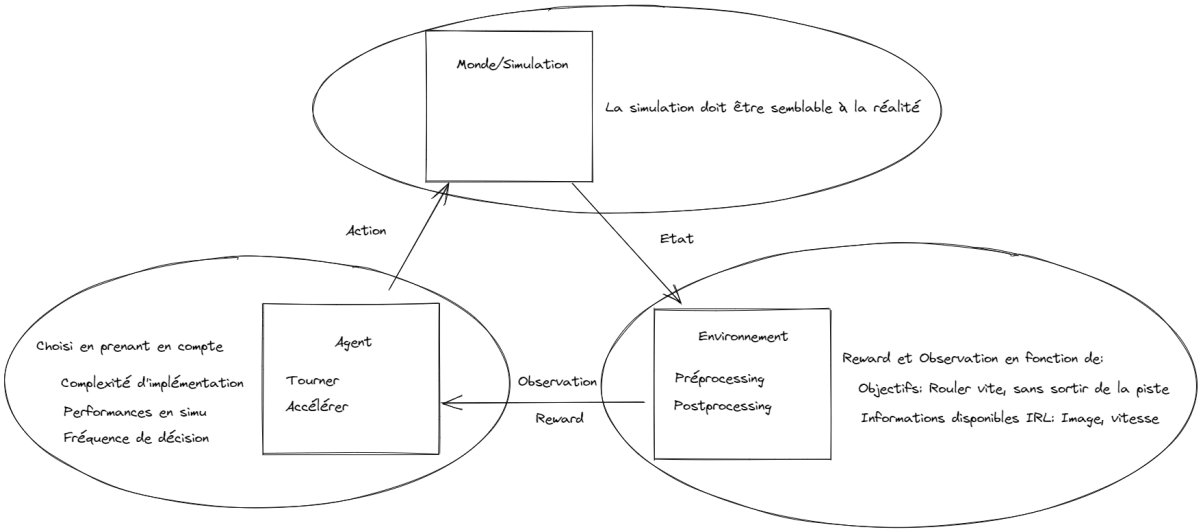
Formulation du problème

Entraîner un modèle de DeepLearning à conduire une voiture DonkeyCar sur la piste de course suivante:



modélisation de la piste de course (vue de dessus)

Nous avons fait le choix de l'apprentissage par renforcement pour traiter ce problème. Il se formule comme suit.



Dans la méthode d'apprentissage par renforcement, on n'a pas besoin de générer de données d'entraînement:
Nous plaçons un Agent dans un monde (réel ou simulation).

- L'environnement extrait du monde les données présentées à l'agent pour sa décision.
- L'agent effectue des actions en suivant sa Policy. Les actions de l'agent modifient ce monde (ici les actions modifient la position et la vitesse de la voiture).
- L'environnement calcule la récompense liée à l'action de l'agent. Cette récompense est utilisée pour corriger la Policy de l'agent (son comportement).

Le Monde dans lequel la voiture évolue est fixé par Sigma. Nous avons donc eu à faire des choix sur l'agent, ainsi que sur l'environnement, mais avions peu de choix à faire concernant le Monde dans lequel l'agent évolue.

Choix

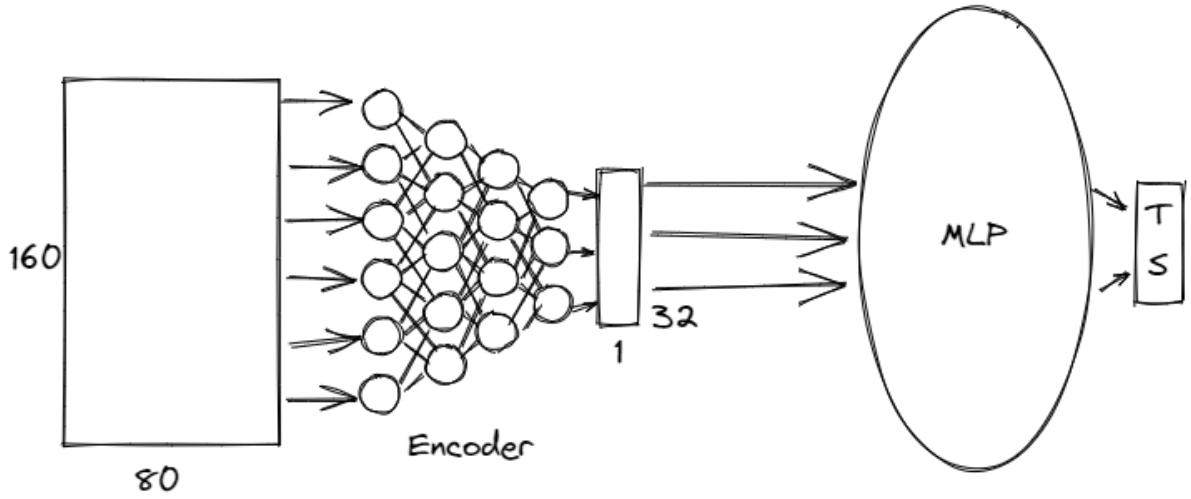
Choix concernant l'agent

Notre but est de faire une preuve que la méthode par renforcement est viable pour une conduite dans la réalité, et non d'améliorer l'état de l'art de la conduite sur simulateur.

Nous avons donc fait le choix de la simplicité d'implémentation, en prenant un travail déjà effectué sur simulateur, avec des performances largement satisfaisantes. Les principales sont la librairie StableBaseline3 et les tutoriels d'Antonin Raffin.

- *https://stable-baselines3.readthedocs.io/en/master/index.html*
- *https://www.youtube.com/watch?v=ngK33h00iBE&list=PL42jkf1t1F7dFXE7f0VTeFLhW0ZEQ4XJV*

Les algorithmes ont été entraînés dans le simulateur, avec pour but d'être directement utilisables dans la réalité. Voici les différents éléments qui ont pu composer le modèle:



Autoencoder :

L'Encoder et le Decoder sont des simples réseaux de convolutions symétriques, à 4 couches chacun. L'espace latent utilisé était un vecteur unidimensionnel de 32 valeurs.

Policy: CNN/MLP

La CNN Policy est plus adaptée sur des inputs de grandes dimensions, comme les images. Avec l'Autoencoder et un input de basse dimension, la MLP policy a été privilégiée.

Dans tous les cas, on a un Acteur et un Critique. L'acteur correspond à la policy, et le critique estime la valeur des actions. L'Acteur corrige sa policy en fonction de l'output du Critique. Le critique est corrigé avec la reward.

Algorithme: TQC

L'algorithme TQC est une amélioration du DQN visant à diminuer la surestimation de la Qvalue.

En des termes plus simples, on utilise plusieurs critiques et on prend en compte l'ensemble de leurs prédiction pour éviter les erreurs de surestimation des valeurs des actions.

Sources :

- *https://research.samsung.com/blog/Fixing-overestimation-bias-in-continuous-reinforcement-learning*
- *https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f*

Choix concernant l'environnement

Nous avons pu jouer sur le préprocessing (modification de l'image, estimation de la vitesse…), sur le post-processing (simulation de la zone morte du moteur réel), ainsi que sur la reward.

Préprocessing :

Nous avons testé plusieurs sortes de préprocessing, que vous pourrez retrouver dans la partie dédiée du repo. Le but était à chaque fois de diminuer le bruit (reflets lumineux), et de gommer les différences entre le simulateur et la réalité (couleur de piste, des lignes…).

Postprocessing :

Le post processing a consisté principalement à interdire une zone du throttle, en mettant sa valeur à 0. Cette zone correspond à la zone morte du moteur dans la réalité.

Un lissage des valeurs de steering consécutives a aussi été testé en postprocessing.

Reward :

Les Rewards étaient calculées comme suit:

- Punir la distance au centre de la route
  - (1.0 - (D / Dmax) ** 2) D=Distance au centre de la route
- Récompenser la vitesse
  - (V / Vmax) V=Vitesse de la voiture

Ces deux éléments sont multipliés dans le calcul de la reward (1.0 - (D / Dmax) ** 2) *(V / Vmax).

- Punir les crash (par détection des collisions, ou des arrêts trop longs)
    - -2
- Punir les sorties de route (D>Dmax)
    - -1

Dans le cas d'un steering smoother(lissage des actions de steering consécutives), la Reward suivante a pu être ajoutée et testée :

- Punir la différence entre des actions successives
    - R'(n) = R(n) - abs[(S(n) - S(n-1)) / S(n)]

Avec

- R(n) la reward de l'action n telle que définie précédemment
- R'(n) la reward de l'action n avec steering smoother
- S(n) le steering de l'action n

Choix concernant le Monde (simulation/réalité)

Nous ne pouvions pas choisir la piste dans laquelle la course aurait lieu, nous avons donc dû adapter notre simulateur pour qu'il corresponde à la réalité.

Le guide d'utilisation du simulateur est disponible [dans la partie dédiée].

Les paramètres du simulateur et de la voiture réelle que nous avons pu modifier sont :

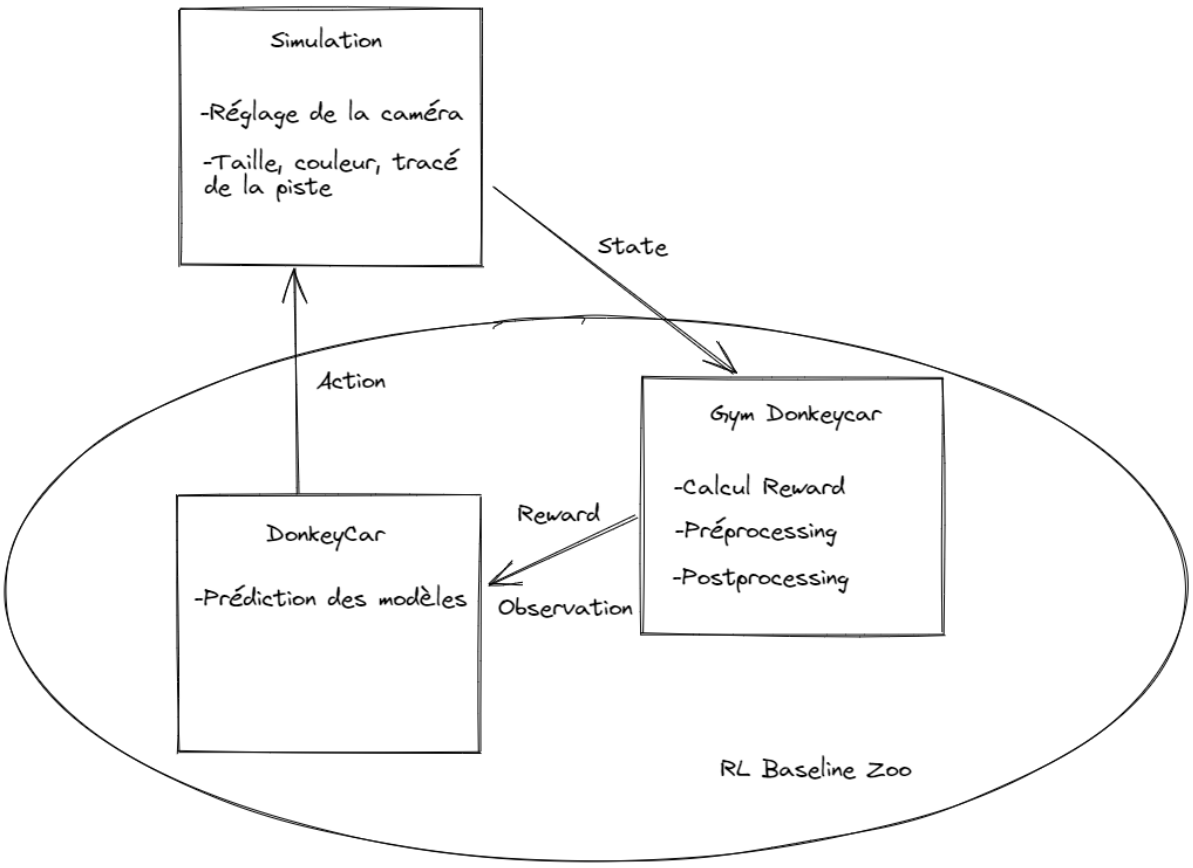
- Tracé piste
- Couleur piste/sol
- Luminosité/soleil
- Objets/Obstacles
- Fish-eye (modifié dans la réalité)
- Position et orientation de la caméra

Réalité :

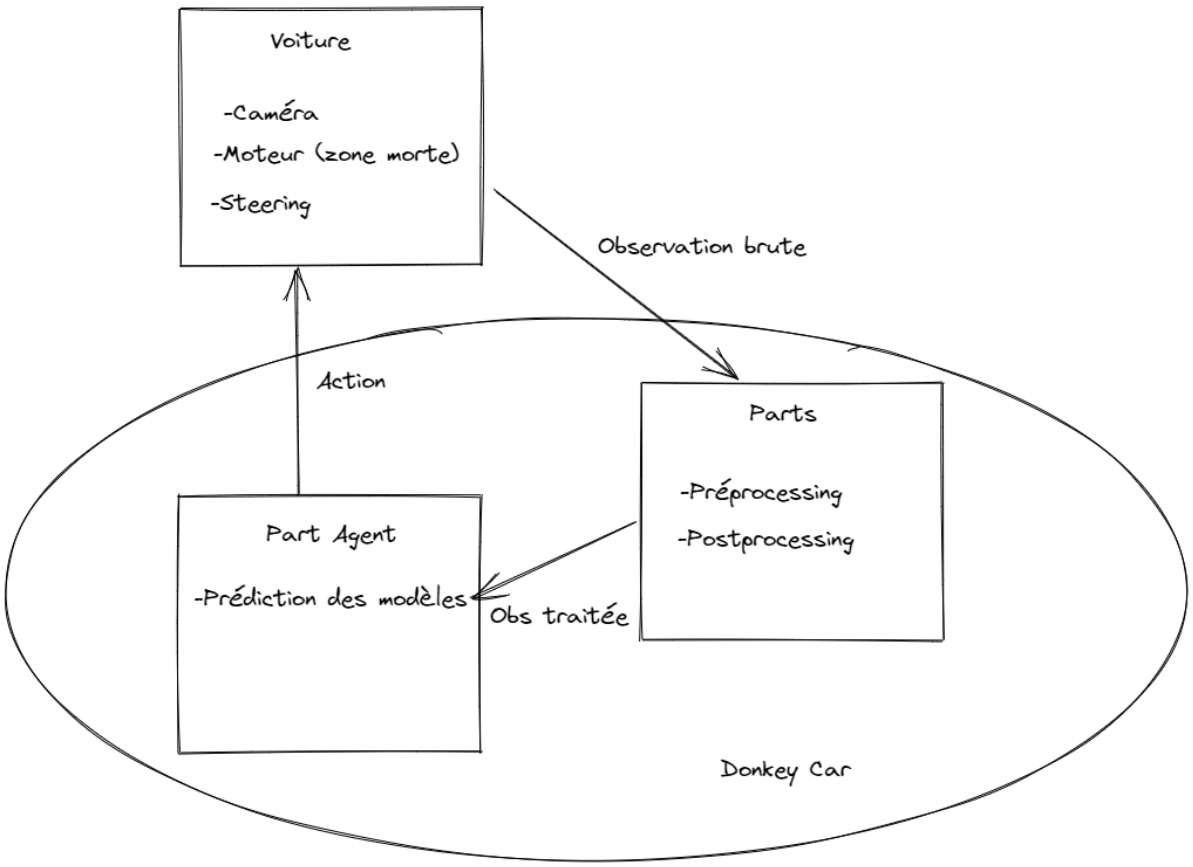
- Position et orientation de la caméra

Implémentation

Dans la réalité, le Monde correspond à la simulation, l'environnement est géré par GymDonkeycar, et l'Agent est géré par le framework Donkeycar. RL Baseline Zoo permet de gérer tous les hyperparamètres, les algorithmes et les modèles utilisés.



Dans la réalité, le Monde correspond bien au monde réel, puis tout se passe dans le framework Donkeycar (préprocessing, postprocessing et prédictions). Ici pas d'entrainement, donc pas de reward.



Expériences

Durant nos expériences nous avons donc joué avec les paramètres suivants :

- Différents préprocessing
  - otsu
  - line
  - edge
- Autoencoder
  - Avec/sans Data Augmentation
    - Différentes augmentations (pluie, sunflare, trous…)
    - intensité des augmentations
- Drive
  - vitesse max
  - steering max
  - zone morte
  - Avec/sans Data Augmentation
    - Différentes Data Augmentation
    - Intensités des augmentations
  - Différentes pistes
    - taille de la piste
    - virages plus ou moins serrés
    - couleur de la piste
    - Avec ou sans décor

    - Avec ou sans délai pour simuler les performances de la Rasp
    - Différente position et orientation de la caméra dans le simulateur
    - Avec/sans estimation de la vitesse en input (donnée non accessible IRL)
- Test réel
    - Différente orientation de la caméra
    - Vitesse max

Observations, solutions et pistes

Observation : Arrêt dans les virages (IRL) :

La zone morte est importante à simuler pour éviter les arrêts dans les virages. La prise en compte des derniers throttle n'a rien donné pour éviter ces arrêts.

Observation : Roule sur la ligne (IRL) :

La voiture a tendance à mordre et osciller autour des lignes lors des essais en situation réelle.

Une explication est la position différente de la caméra dans le simulateur et dans la réalité. Dans le simulateur, on a une vue plus haute, avec la possibilité de voir les lignes à droite et à gauche pour savoir qu'on est au milieu de la piste.

Dans la réalité, on ne voit que la ligne la plus proche, et la voiture peut interpréter ça comme une sortie de piste.

Il est possible de changer la position et l'orientation de la caméra dans la simulation (ou en bricolant dans la réalité), il faut trouver le moyen de faire correspondre les deux.

Observation : Sortie de piste dans les virages serrés (IRL) :

La piste de simulation n'avait pas de virages aussi serrés que dans la réalité, avec la présence d'un autre segment de la piste en face du coude.

Nous avons adapté notre piste de simulation à cela.

Observation : disparition des lignes dans la réalité, ou présence de bruit (IRL) :

Nous avons répondu à ces observations de deux manières différentes :

- OTSU + Augmentation ⇒ Non concluant, le bruit était trop présent
    - Augmentation trop forte avec la pluie ⇒ Entraînement trop impacté
    - Les autres augmentations ne permettent pas de rendre le modèle plus robuste au bruit.
- Ligne/Edge ⇒ Le bruit est quasiment éliminé, mais on perd parfois les lignes. Cela semble acceptable (augmenter le flou de l'image peut permettre de grossir les lignes et de moins les perdre)

Observation : tendance à osciller (IRL) :

Peut être lié à l'observation "Roule sur les lignes". Une solution serait d'utiliser un smoother. Nous avons testé :

- Modification de la reward qui punit les coups de volant brusques ⇒ Non concluant
- Modification du postprocessing pour empêcher les coups de volant brusques ⇒ Non concluant

Il faudrait réessayer après avoir résolu le problème de position de caméra (voir observation “Roule sur les lignes”).

Observation : modèles trop entrainés ne fonctionnent pas mieux (IRL) :

On soupçonne un overfitting du modèle drive, il faut donc faire attention lors des tests d’avoir bien des modèles entraînés à différents steps.

Observation : temps d’une boucle supérieur à taux de raffraichissement (IRL) :

On peut jouer sur deux paramètres pour résoudre ce problème qui semble avoir de sérieuses conséquences sur les performances (après test en simulation avec de la latence artificielle, on obtient les mêmes résultats que la réalité).

Solutions possibles:

⇒ améliorer la Raspberry Pi (overclock)

⇒ entraîner en simulateur avec de la latence artificielle (difficile de savoir où mettre quelle latence)

⇒ entraîner un modèle moins long à l’inférence

⇒ améliorer les performances de lu postprocessing

Limites

Nous portons un regard très critique sur nos tests et observations en conditions réelles, car notre voiture a eu un problème de servomoteur qui a affecté son steering jusqu’à rendre impossible le fait de tourner les roues.

Nous ne savons pas quand ce problème a commencé, mais il a sûrement affecté la plupart de nos tests, en s’amplifiant au fil du temps.

Lors de nos derniers tests en conditions réelles, nous avons testé nos modèles avec une autre voiture, qui a effectué plusieurs tours sans sortir de la piste, et a même pu faire un tour avec une vitesse de 0.25, deux résultats que nous n’avions jamais eu précédemment.

Cela constitue donc une grande frustration, et limite les leçons que l’on peut tirer de nos tests, mais c’est aussi une source d’espoir pour la reprise du projet, car la voiture a déjà des résultats très satisfaisants.