Grenoble INP - Phelma

Rapport final (attention aux libellules)

Robot qui apprend à marcher tout seul

Bayle Ernest
Bened Violaine
Engel Baptiste
Richard Robin
Stefanos Marie-Ange

2A SICOM - 12 avril 2021

Table des matières

Introduction

L'algorithmie génétique est une méthode imitant le principe de la sélection naturelle pour résoudre une tâche spécifique. L'objectif de ce projet est d'appliquer un algorithme génétique pour apprendre à une voiture à se rendre d'un point A a un point B sans indications préalables.

Notre travail se décompose donc en quelques grandes parties : algorithme génétique, traitement d'images et interface entre le code et la voiture. L'objectif est donc de travailler en parallèle sur ces différents points afin d'optimiser le temps a notre disposition, pour arriver au résultat défini dans le cahier des charges. Nous détaillons dans ce rapport nos objectifs, les moyens mis en place pour les réussir, ainsi que les difficultés rencontrées, tant sur le plan technique que sur le plan organisationnel.

I Cahier des charges

Les premières séances de ce projet nous ont permis de définir, après concertation avec notre tuteur, les objectifs qui nous semblaient les plus réalistes avec le temps et le matériel à notre disposition. Le cahier des charges nous sert de guide pour notre travail aux cours des séances, et à nous recentrer sur les points importants du projet. Nous avons défini le cahier des charges suivant :

1. **La voiture doit être capable de se déplacer d'un point A à un point B le plus efficacement possible, donc en ligne droite, en autonomie et sans entraves.**
   C'est le point clé du projet. Cela nécessite d'avoir un algorithme entraîné, une détection fonctionnelle de la voiture dans l'espace de travail et un câblage/contrôle efficace de cette dernière.
2. **La voiture doit utiliser un algorithme génétique pour apprendre à effectuer les tâches demandées en 1.**
   Cela nécessite donc d'avoir réglé les différents paramètres de sorte à ce que l'algorithme converge vers une solution en un nombre raisonnable de générations. Pour étudier l'influence des paramètres sur l'issue de l'algorithme, il est nécessaire de mettre au point une simulation de la voiture.
3. **Durant l'entraînement de l'algorithme, la voiture doit partir du même point à chaque nouvel essai.**
   Il est donc nécessaire d'établir un algorithme permettant à la voiture de revenir à son point de départ, défini dans le projet comme étant le centre de la zone de travail, à l'issu de chaque test individuel. Cette algorithme doit être robuste, car il est appelée à chaque itération.
4. **L'ensemble des codes écrits doit être documenté et écrit en anglais, en respectant les normes d'usage spécifique à chaque langage.**
   C'est une étape fastidieuse mais nécessaire, même au sein du groupe, à des fins de bonne lisibilité du code et de cohérence entre les différents fichiers.

II Choix de conception permettant de répondre au cahier des charges

Nous détaillons maintenant comment nous avons choisi de répondre au cahier des charges. Pour cela, nous avons identifié les grandes parties de notre travail. Trois grandes lignes de travail se détachent : la partie externe à la voiture, la voiture en elle même, et l'interfaçage entre ces deux éléments. Dans ces trois cas, des solutions physiques et des solutions logicielles doivent être misent en place.

II.1 Matériel a notre disposition et choix des lignes à suivre

Afin de réaliser le projet, nous avons a notre disposition une voiture miniature, composée de 4 roues pouvant être pilotées par une carte Arduino. De plus, nous disposons de Raspberry4, micro-ordinateur disposant de pins programmables. La structure de la voiture est détaillée en figure 1.

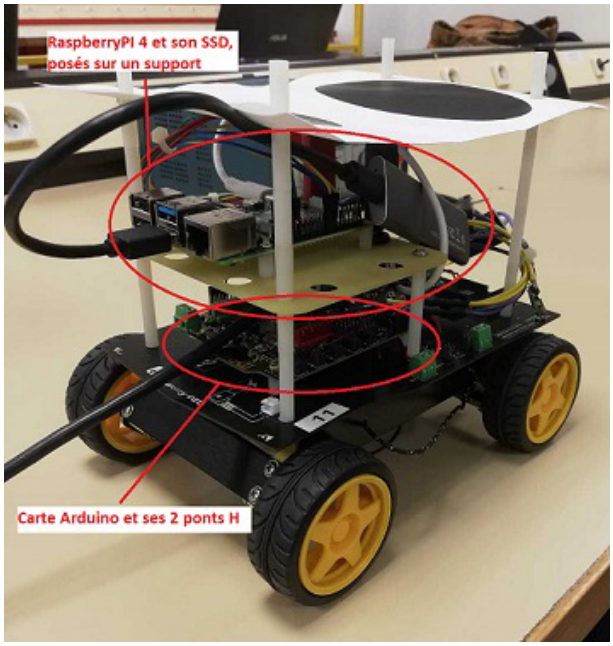


FIGURE 1 – Détails de la voiture

Pour répondre au problème, nous avons choisi de réaliser la structure présentée en figure 2.

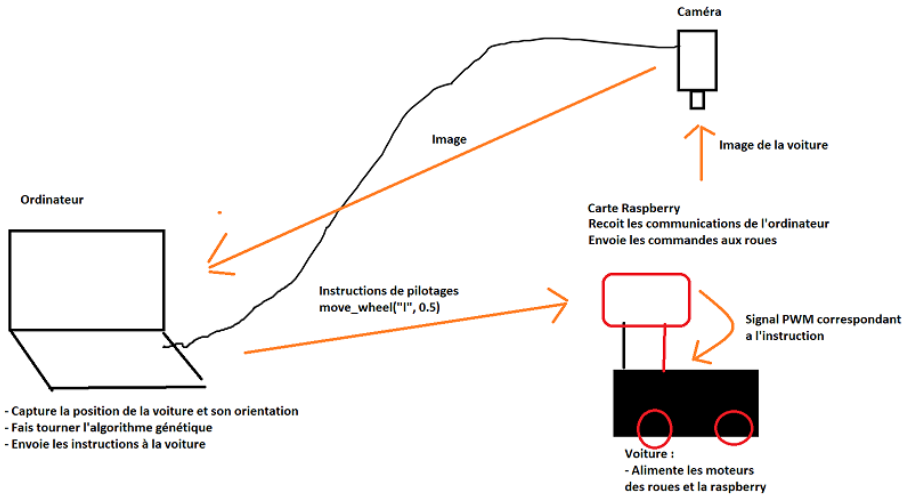


FIGURE 2 – Schéma structurel de notre approche

L'ordinateur pilote l'ensemble de l'exécution. En s'aidant d'une caméra, il récupère la position de la voiture dans l'espace. Cela permet de calculer le score, nécessaire à l'algorithme génétique, mais aussi de ramener la voiture au centre de la zone de travail, zone définie par la largeur de champ de la caméra. L'ordinateur pilote la voiture en envoyant à la Raspberry des instructions du type "faire tourner la roue avant droite à une vitesse x". La Raspberry se charge alors de décoder ce message, et de le transformer en signal PWM destiné à piloter les moteurs.

Nous avons choisi d'utiliser le Python comme langage de programmation pour réaliser l'ensemble des codes nécessaires au projet. C'est un langage simple, dans le sens où un code écrit en Python est aisément compréhensible sans que l'on en soit l'auteur, et puissant, car il permet de réaliser énormément de choses. De plus, tous les membres du projet étant familiers avec le langage, aucun temps d'adaptation ou d'apprentissage n'est nécessaire.

II.2 Matériel physique

II.2.1 Alimentation de la voiture et de la Raspberry

L'objectif est de développer une voiture pouvant se déplacer sans entrave dans une zone de travail. Il est donc nécessaire qu'aucun câblage ne vienne gêner sa mobilité.

Les différentes cartes qui composent la voiture, ainsi que les moteurs, ne doivent pas être alimentés avec les même tensions. L'utilisation de batterie impose un bon rendement électrique pour éviter de baisser considérablement l'autonomie. Notre premier choix s'est porté sur des batteries Lithium-Polymère (LiPo). Ce sont des batteries petites, légères, et à très hautes performances, utilisées en modélisme. Cependant, pour des causes budgétaires, ces batteries n'ont pas pu être commandées. Des batteries Nickel - Hydrure métalliques (NiMH) a 6 éléments nous ont été fournies : nous avons donc travaillé avec 2 batteries NiMH en séries, pour atteindre 14 Volts.

Des abaisseurs de tension permettent ensuite d'alimenter la Raspberry en 5.2V, ainsi que les moteurs. Nous avons commandé des abaisseurs de tensions miniaturisés. Les résistances sont choisies de sorte à avoir 5.2V en sortie de l'abaisseur pour le Raspberry, et 10V pour l'alimentation des moteurs.

Nous avons été confrontés à certains problèmes avec les batteries prêtées par Phelma :

— Les batteries ont une capacité très faible. Nous ne parvenons pas à avoir l'autonomie suffisante pour faire rouler la voiture plus que quelques minutes. Cela est dérangeant dans la mesure où de nombreuses heures sont nécessaires pour réaliser l'ensemble des générations de l'algorithme (voir détails dans les sections suivantes).

— Il y a des problèmes de connexion qui génèrent des faux-contacts et induisent un redémarrage intempestif de la Raspberry

Afin de ne pas être gêné par cela, nous avons réalisé un grand câble banane, permettant à la voiture d'être reliée directement à une alimentation sur secteur, sans être trop gênée dans ses déplacements. Cette solution temporaire peut être remplacée simplement par des batteries plus performantes.

II.3 Partie logicielle

II.3.1 Communication entre le système physique et la carte de commande

Première approche : Raspberry - Arduino - Moteurs

Par souci de simplicité et de compréhension, nous avons commencé par numéroter les roues conformément à la numérotation des moteurs. Les branchements initiaux de la voiture fournie ne permettaient pas de contrôler les quatre roues de manière indépendante puisque seuls deux ponts en H sur quatre présents étaient utilisés (Une seule puce). Cela avait pour impact que nous ne pouvions pas contrôler les roues arrières indépendamment des roues avant. Le circuit contenant deux puces de deux ponts en H chacune,

nous avons observé les signaux de sortie des puces à l'oscilloscope afin d'effectuer les branchements nécessaires à l'utilisation des quatre moteurs indépendamment. Ainsi, les quatre ponts en H sont maintenant utilisés, et les quatres moteurs contrôlables indépendamment les uns des autres.

Nous avons également fait des choix de branchements concernant les PINs de la carte Arduino à utiliser, afin de pouvoir contrôler les moteurs informatiquement. Après étude des possibilités et observation des signaux, nous avons écrit un code de tests, permettant de vérifier que l'on est maintenant capable de mettre en mouvement les quatre roues de manière indépendante, de maîtriser leur vitesse et de choisir leur sens de rotation à partir de code informatique émis depuis la carte Arduino.

Nous avons choisi de communiquer avec l'Arduino en passant par une Raspberry Pi branchée en port série avec celle-ci. Les avantages de cette manoeuvre sont multiples :

— La possibilité de brancher l'Arduino et la Raspberry semble plus simple, stable et fiable que de les connecter en wifi.
— Le principe du port série est d'envoyer des bits, les uns à la suite des autres, ce qui simplifie l'envoi et la réception des données.
— La communication du port série se fait de façon asynchrone : connaissant la taille d'une instruction (2 caractères soit 16 bits), l'utilisation d'une horloge n'est pas nécessaire.

Nous avons fait le choix de gérer le port série à l'aide du module PySerial de Python, qui permet d'envoyer des données directement à l'Arduino.

Seulement, le comportement n'était pas fiable et semblait même un peu aléatoire. C'est la raison pour laquelle nous avons par la suite, totalement arrêté d'utiliser la carte Arduino.

Seconde approche : Raspberry - Moteurs

Passez par l'étape Raspberry - Arduino - Moteurs nous aura permis d'en apprendre plus sur le fonctionnement de la voiture, ainsi que sur la manière de piloter les moteurs. Nous nous sommes de plus aperçus qu'il n'était pas nécessaire de passer par l'Arduino, pour le contrôle des roues, mais que l'on pouvait simplement utiliser les PINs de la carte Raspberry.

C'est le rôle de la classe RaspController du fichier interface.py (voir annexe) d'établir la connexion entre les pins du Raspberry et de la carte de la voiture. A l'initialisation, les PINs du Raspberry sont activés et déclarés comme sorties. On peut alors, avec la méthode RaspController.move_wheel piloter les roues en donnant une vitesse entre -1 et 1, le signe indiquant le sens de rotation, la valeur la vitesse. La vitesse est contrôlée en faisant varier le rapport cyclique du signal de sortie des PINs, qui est un train d'impulsions. La méthode RaspController.move_wheel permet de piloter indépendamment chacune des roues, dans n'importe quel sens. C'est ce que l'on souhaite, puisque les vitesses de chaque roues sont totalement décorrélées au démarrage de l'algorithme génétique.

II.3.2 Communication avec la voiture

L'ordinateur est l'opérateur central de notre système. C'est lui qui produit les résultats de l'algorithme génétique, et qui connaît la séquence d'instructions que doit exécuter la voiture. Ainsi, il doit envoyer à la voiture les mouvements à effectué pour "exécuter" la séquence génétique d'un individu. Plusieurs choix de canaux de communication existent pour faire communiquer un RaspberryPi et un ordinateur.

Le Bluetooth Le Bluetooth est une solution qui semble idéale. Cependant, si le Raspberry dispose bien des systèmes lui permettant de communiquer en Bluetooth avec une autre entité, il est difficile d'exécuter des commandes Python par Bluetooth, qui est un protocole permettant plutôt d'échanger des informations plus que des instructions. De plus, les supports Bluetooth disponibles sur Linux et en Python ne

disposent que de documentation peu claires, datées, et sont parfois dépréciés. Après de nombreux tests, concluants ou non, l'instabilité de la puce Bluetooth et la complexité du problème nous ont poussés à chercher une solution plus simple.

Socket TCP Le protocole TCP est un des premiers protocole développé pour Internet. Un serveur se met en attente (en "écoute") d'une connexion par un client. Une fois celle-ci établie, ils peuvent échanger des informations. C'est un protocole qui a la réputation d'être fiable et d'être aisé à manipuler et à mettre en oeuvre. Nous avons donc choisi d'appliquer cette option à notre cas. Le serveur sera la Raspberry, qui, dès son démarrage, se met en écoute (grâce à la classe Server du fichier "communication.py"). Lorsque l'on appelle la fonction "move_wheel" du fichier "interface.py" depuis un ordinateur, celui-ci scanne le réseau (réseau local créé à partir d'un partage de connexion depuis un téléphone portable) à la recherche d'un serveur en écoute. C'est le rôle de la classe "Client" du fichier "communication.py". Lorsqu'un tel serveur est trouvé (dans notre cas, seul le Raspberry est en écoute sur le réseau local), une connexion permanente entre le client de l'ordinateur et le serveur de la carte Raspberry est instanciée.

A l'allumage de la voiture, le Raspberry s'allume. Nous souhaitons qu'il démarre automatiquement le serveur d'écoute, sans intervention. De plus, à l'allumage, les roues tournent, il faut donc automatiquement les arrêter. Pour cela, nous avons codé un démon lançant au démarrage le fichier "interface.py" qui, si il est lancé depuis un Raspberry, lance un serveur d'écoute et arrête les roues. Ce démon est géré par l'utilitaire "systemctl" du système d'exploitation du Raspberry.

Le Raspberry peut alors recevoir des messages envoyés depuis l'ordinateur, et les exécuter si ce sont des instructions. Afin qu'il n'y ait aucune manipulation à faire pour un éventuel opérateur, tout le processus de connexion est automatisé. A l'allumage de la voiture, le Raspberry se met en écoute. Il suffit alors de poser la voiture au sol, et de lancer le programme "main.py" depuis l'ordinateur. Celui-ci appelle la méthode "simulation" d'un objet de type "Génération", qui s'occupe seule de gérer la caméra (voir section suivante), et de communiquer avec la voiture les informations nécessaire pour l'exécution des gènes de l'algorithme.

II.3.3 Acquisition des données d'entraînement : traitement d'images

Il est nécessaire de connaître la position de la voiture au début et à la fin de l'entraînement de chaque individu afin de pouvoir l'évaluer (C'est à dire lui attribuer un score.). Nous avons choisi pour cette tâche d'utiliser les outils de traitement d'image. Une caméra de type WebCam Logitech C920 fixée au plafond nous permet l'acquisition de l'image du champ dans lequel évolue la voiture. Il s'agit ensuite de traiter l'image correctement afin de déterminer la position et l'orientation de la voiture en temps réel.

Nous avons fait le choix technique d'utiliser Python pour réaliser cette tâche, dans la mesure où nous sommes familiers avec ce langage et des bibliothèques dont il dispose pour le traitement d'image, notamment OpenCV.

1. Détection en utilisant un algorithme de **Template Matching**

   La première idée que nous avons développée est l'utilisation d'un algorithme utilisant SIFT (Scale Invariant Feature Transform) pour détecter la voiture, en comparant l'image acquise à une image de référence (le template). L'avantage de cette technique est qu'elle ne se soucie en théorie ni de la taille ni de l'orientation de la voiture. L'algorithme utilisé renvoie une liste de 'matching point', à partir de laquelle nous pouvons extraire un ensemble de coordonnées appartenant à la voiture. A partir de cela, on peut définir le centroïde de la voiture, et donc sa position.
   En pratique, cette technique présente plusieurs faiblesses :

   — La détection ne fonctionne que relativement bien, dans la mesure où la voiture est petite et que la résolution de la caméra ne permet pas d'en voir tous les détails.

   — Des points n'appartenant pas à la voiture sont détectés comme tels. Le calcul du centroïde est donc faussé.

— L'information concernant l'orientation de la voiture n'est pas évidente à récupérer.

— L'algorithme est gourmand en ressources. Pour améliorer la détection sur une image, augmenter la résolution du template est possible, mais le calcul est alors ralenti. On constate alors une différence très importante entre la réalité et le moment de la détection. La contrainte de temps réel n'est pas remplie.

2. Détection en utilisant de la **reconnaissance de forme**

Une méthode plus simple pour récupérer les informations de position de la voiture est d'utiliser la détection de contours. Comme l'on connaît les dimensions de la voiture, et que l'espace ne contient à priori pas d'objet de dimension similaire, on peut être certain que détecter un contour correspondant permet d'identifier la voiture. L'avantage de cette méthode est que l'on récupère des points qui appartiennent bien au contour, et calculer le barycentre est alors très simple.

Pour détecter l'orientation de la voiture, nous proposons une carrosserie contenant deux formes, l'une à l'avant, l'autre à l'arrière, de taille différente. Une fois ces deux formes détectées, on peut alors connaître le sens de la voiture ainsi que sa direction.

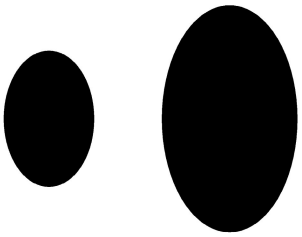


FIGURE 3 – Template de la voiture

Détail du fonctionnement de l'algorithme de détection de la voiture L'exécution se fait dans le fichier "detect.py". Pour repérer la voiture dans l'espace, nous commençons par effectuer un seuillage automatique. Il est nécessaire que le champ de la caméra soit dégagé, et que seule soit présente la voiture. Le seuillage automatique vise à chercher le seuil qui permet d'avoir la bonne proportion de pixels blancs et noirs. Les seuls pixels qui après seuillage sont noirs, sont les pixels des ovales noires de la carrosserie de la voiture et ceux du fond. Connaissant la surface des ovales et la surface capturée par la camera, ce rapport est déterministe et constant. On le fixe de sorte à avoir une valeur moyenne de l'image de 1.05 (voir ImageCapture.init_thresh dans le fichier "detect.py"). Ensuite, à l'aide des fonctions de la librairie Python OpenCV, on récupère les contours présents dans l'image. On peut alors les classer du plus grand au plus petit. Comme l'image est seuillée correctement, on sait que seuls trois contours seront présents : celui de la voiture elle même, celui du grand ovale et celui du petit ovale. En calculant le centre de chacun des deux ovales, on peut alors connaître le barycentre de la voiture, et son angle avec l'axe des abscisses.

Les fonctions utiles pour le traitement d'image et la détection de la voiture sont regroupées au sein de la classe *ImageCapture* du fichier detect.py.

Cet algorithme étant beaucoup moins gourmand en ressource que le premier, c'est celui-ci que nous avons développé pour ce projet. Le code complet est disponible dans le fichier "detect.py".

II.3.4 Algorithme génétique

II.3.4.1 Principe

DÉFINITION D'UN ALGORITHME GÉNÉTIQUE Les algorithmes génétiques font partie de la famille des algorithmes évolutionnistes, c'est-à-dire que leur principe s'inspire de la théorie de l'évolution pour résoudre un problème. L'idée est de faire évoluer un ensemble de solutions à un problème donné, dans l'optique de trouver les meilleurs résultats. Dans le cas spécifique de notre algorithme génétique, le but est de résoudre un problème d'optimisation, c'est-à-dire d'en obtenir une solution approchée en un temps raisonnable. L'algorithme génétique s'inspire de la sélection naturelle : en l'appliquant à une population de solutions potentielles au problème donné, la solution optimale est approchée par « bonds » successifs, d'une population à la suivante.
Dans notre étude, un ensemble de solutions sera représenté par l'objet Generation du fichier "genetic.py", et une solution de cette population est un appelée Individu (cf description détaillée de l'API plus bas).

HYPOTHÈSES DU PROBLÈME Les conditions justifiant le choix des paramètres de l'algorithme génétique sont les suivantes :

— Temps de calcul de la fonction d'évaluation (score) raisonnablement court.
— Nombre important de solutions : c'est lorsque l'espace des solutions possibles est important que l'intérêt de l'algorithme génétique est remarquable. En effet, si le nombre de solutions possibles est relativement faible, on peut parcourir cet espace de manière exhaustive, ce qui donnerait le meilleur résultat en un temps raisonnable. Cependant, lorsque le nombre de solutions est important, on préfère obtenir une solution relativement proche de la solution optimale en un temps raisonnable plutôt que de toutes les parcourir et obtenir la solution optimale en une durée trop grande.

CHOIX DU SCORE D'UN INDIVIDU Chaque individu se voit attribuer un score permettant d'évaluer son niveau par rapport à la solution optimale. Ce score est utilisé pour la sélection des individus d'une génération à la suivante. Sa définition dépend de l'objectif à atteindre que l'on impose à notre population.

Pour commencer, nous avons choisi de définir le score comme la distance parcourue par l'individu pendant une durée fixée. Cela revient à dire que dans cette configuration, l'objectif pour une voiture est d'aller le plus loin possible. Intuitivement, on comprend que si l'individu est toujours au même point de départ mais que son orientation angulaire initiale est aléatoire, les individus qui trouveront la solution optimale (maximisant le score) se retrouveront, à la fin de leur trajectoire, sur un cercle d'un même rayon (valant ce score). Cette hypothèse d'orientation angulaire initiale non imposée est intéressante dans le cadre de notre projet puisque la voiture démarrera toujours avec une position et une orientation initiales prédéfinies mais avec une certain intervalle de tolérance, notamment sur l'angle de la voiture, difficile à maîtriser avec précision.

Notre objectif étant de faire se déplacer la voiture d'un point à un autre le plus efficacement possible (en ligne droite), il est plus cohérent de fixer le point d'arrivée. Le score a alors été adapté à cette situation : on le définit comme l'inverse de la distance entre le point d'arrivée et la position de l'individu à la fin de sa trajectoire.

REPRODUCTION DES INDIVIDUS POUR CONSTRUIRE UNE NOUVELLE GÉNÉRATION

Sélection des individus présents dans une population pour la génération suivante Il existe plusieurs techniques de sélection : uniforme, par rang, par tournoi, ou encore en utilisant la probabilité de sélection proportionnelle à l'adaptation. C'est cette dernière que nous avons utilisée. Afin d'évaluer le niveau d'un individu, c'est-à-dire sa proximité avec la solution optimale, nous avons défini un score à maximiser. Plus le score d'un individu donné est élevé, plus il est proche de la solution optimale. L'idée étant qu'un maximum d'individus atteignent un score élevé, nous avons sélectionné les individus en leur attribuant une probabilité d'apparition dans la génération suivante proportionnelle à leur score. Ainsi, plus un individu a un score faible, c'est-à-dire plus il est éloigné de la solution optimale, moins il a de chances de se reproduire dans la génération suivante.

Les crossovers (enjambement, croisement ou encore recombinaison) De nouveaux individus, que l'on peut assimiler à des chromosomes (soit une succession de gènes, voir description détaillée de l'API dans le paragraphe suivant), peuvent être créés par recombinaison de chromosomes de la génération précédente.

Les mutations Le processus des mutations étant bioinspiré, on définit un taux de mutation (*mutation_factor*), qui indiquera la probabilité qu'un gène mute de façon aléatoire au sein du chromosome associé. On notera que le taux de mutation doit être assez faible (maximum de l'ordre de 0.1) pour conserver le principe de sélection naturelle et éviter de tomber dans une recherche aléatoire.

CONCLUSION SUR LA SÉLECTION DES INDIVIDUS : Dans notre projet, une nouvelle génération n est un ensemble d'individus créés du croisement aléatoire entre 2 chromosomes de la génération $(n-1)$, avec un facteur de cross-over égal au score des individus et un facteur de mutation fixe donné en paramètre.

II.3.4.2 Description de l'API Nous avons créé 3 classes d'objets, dont nous allons expliquer l'utilité et les méthodes principales : Gene, Individual et Generation.

L'OBJET GENE Un Gene correspond à une séquence de mouvements. Ses attributs principaux sont le nombre de portions qui le composent et la durée d'une portion. Ses méthodes consistent à faire des opérations sur les gènes (ajouter deux gènes, affecter un poids à un gène), évaluer un gène (les vitesses et orientations des 4 roues) à un instant t donné et faire muter un gène selon un facteur donné en paramètre du constructeur d'objet Generation.

L'OBJET INDIVIDUAL L'Individual représente un comportement appartenant à une population, c'est à dire un gène possédant un score et une position. Présentons les deux méthodes principales de cet objet : la première met en mouvement la voiture de manière à effectuer le gène de l'individu (pour le cas réel et la simulation), tandis que la seconde permet d'acquérir les coordonnées de la voiture à la fin de la réalisation du gène (grâce à la caméra).
Enfin, afin de laisser faire revenir l'individu à la position de départ (après chaque réalisation d'un gène), nous avons créé une fonction qui déplace la voiture au centre du champ de la caméra et la fait pivoter jusqu'à retrouver sa position initiale.

L'OBJET GENERATION Une Generation est un ensemble d'individus capables d'évoluer au cours du temps grâce à la simulation. Elle est caractérisée par le nombre d'individus et le facteur de mutation. Deux types de simulations ont été mis en place (adaptés pour chacun à la simulation virtuelle et et à l'implémentation réelle de la voiture).

Le premier boucle sur le nombre de générations fixé, ainsi que sur le nombre d'individus par génération (donné en paramètre). Les poids sont choisis aléatoirement à l'étape initiale. Le deuxième type de simulation prend deux paramètres supplémentaires :

- le rayon (*accepted_radius*) autour du point d'arrivée choisi, qui permet d'élargir la zone dans laquelle on considère que les individus sont bien arrivées.
- un pourcentage d'individus (*tolerated_percentage*) qui représente le seuil maximal acceptable d'individus qui ne sont pas parvenus à arriver dans la zone voulue (le disque caractérisé par le point d'arrivée et le rayon défini précédemment).
- un nombre maximal de générations, à partir duquel on considère que l'algorithme diverge (ou que la complexité temporelle est trop grande pour être intéressante et exploitable).

Ainsi, le principe de ce deuxième type de simulation est le suivant : pour chaque génération, tant que le nombre maximal de générations n'est pas atteint et que le pourcentage d'individus qui ne sont pas arrivés à destination (dans le disque centré autour du point d'arrivé et de rayon *accepted_radius*), l'algorithme génétique continue et réalise une nouvelle génération. Sinon, il s'arrête. Deux situations se présentent : ou bien le critère est satisfait (au plus *tolerated_percentage* % d'individus sont arrivés dans le disque d'arrivée,

ou bien on a atteint le seuil maximal de générations et on considère que le critère n'aboutira pas dans un délai convenable.

II.3.5 Retour à la case départ

Lors de l'exécution d'un nouveau gène (nouvel individu) la voiture doit partir d'un endroit fixe, avec une orientation définit préalablement. Afin d'augmenter le nombre de directions possibles nous avons choisis comme point de départ le centre du champ de vision de la caméra (point de coordonnées (320, 230)).

L'algorithme de retour à la case départ procède en 3 temps :

— Rotation de la voiture pour qu'elle soit en direction du point de départ

— La voiture avance en ligne droite (même commande sur les roues de droite et de gauche) jusqu'à atteindre le point de départ

— La voiture tourne à nouveau pour atteindre sa position finale

Des seuils de tolérance ont été choisis pour les coordonnées du point de départ ainsi que pour l'angle d'orientation.

— Erreur sur la position : 50 pixels

— Erreur sur l'orientation : 10°,

Le code de cet algorithme est celui de la méthode "reset_position" de l'objet Individual du fichier "genetic.py", disponible en annexe.

II.3.6 Récupération des données de test

Nous évoquons dans cette partie la récupération des données lors de l'exécution des test réels et des simulations. Pour ce faire, nous avons choisi de créer un dossier dont le nom indique tous les paramètres choisis lors de cette simulation. Ce dossier contient les images représentant les trajectoires de tous les individus de chaque génération (au plus 1 image par génération) ainsi qu'un fichier *data_simu.csv* décrivant la vitesse des roues pendant chaque portion de gène, pour tous les individus, ainsi que leur score, et ce pour toutes les générations.

Génération	Individu	Score	Gene_0roue_0	Gene_0roue_1	Gene_0roue_2	Gene_0roue_3	Gene_1roue_0	Gene_1roue_1	Gene_1roue_2	Gene_1roue_3
0	0	2,537	0,1063	0,4081	0,7469	0,2943	0,9919	0,145	0,520	0,992
0	1	15,725	0,1312	0,3626	0,9456	0,7837	0,7198	0,256	0,771	0,488
0	2	6,305	0,4898	0,8780	0,4841	0,8144	0,9181	0,835	0,543	0,423
0	3	6,059	0,2801	0,1673	0,2259	0,9310	0,5905	0,520	0,826	0,556
0	4	6,359	0,2956	0,5188	0,3168	0,2575	0,8336	0,108	0,091	0,776
1	0	15,725	0,1312	0,3626	0,9456	0,7837	0,7198	0,256	0,771	0,488
1	1	6,458	0,4898	0,8780	0,4948	0,8144	0,9181	0,835	0,543	0,423
1	2	6,737	0,2057	0,2649	0,5857	0,8574	0,6551	0,388	0,799	0,522
1	3	3,927	0,1312	0,3626	0,9456	0,7837	0,7198	0,256	0,771	0,488
1	4	15,725	0,1312	0,3626	0,9456	0,7837	0,7198	0,256	0,771	0,488
2	0	3,938	0,1312	0,3626	0,9456	0,7837	0,7198	0,256	0,771	0,488
2	1	9,064	0,3105	0,6203	0,7202	0,7991	0,8189	0,546	0,657	0,455
2	2	9,546	0,3105	0,6203	0,7202	0,7991	0,8189	0,546	0,657	0,455
2	3	3,938	0,1312	0,3626	0,9456	0,7837	0,7198	0,256	0,771	0,488
2	4	15,725	0,1312	0,3626	0,9456	0,7837	0,7198	0,256	0,771	0,488
3	0	10,635	0,1312	0,3626	0,9456	0,7837	0,7198	0,256	0,771	0,488
3	1	15,013	0,2208	0,4914	0,8329	0,7914	0,769	0,401	0,714	0,472
3	2	4,572	0,2208	0,4914	0,8329	0,7914	0,769	0,401	0,714	0,472
3	3	10,635	0,1312	0,3626	0,9456	0,7837	0,7198	0,256	0,771	0,488
3	4	15,725	0,1312	0,3626	0,9456	0,7837	0,7198	0,256	0,771	0,488

FIGURE 4 – Extrait d'un fichier de sauvegarde des données data_simu.csv

II.3.7 Simulation de la voiture

Pourquoi réaliser une simulation de la voiture ?

Nous allons maintenant aborder la simulation virtuelle du fonctionnement de notre voiture. Le problème est que le temps de développement pour que l'ensemble des systèmes fonctionne nous parait très long. De plus, la voiture étant un système physique, on ne peut exécuter qu'un individu à la fois : si il faut faire 100 générations de 100 individus, les temps de mises en oeuvre risquent d'être excessifs pour obtenir ne serait-ce qu'une seule série de données exploitables.

La décision a été prise, d'un commun accord avec notre tuteur, de réaliser, en parallèle de l'implémentation dans le monde physique de l'algorithme, de réaliser une simulation de la physique de la voiture, à partir de laquelle nous pouvons dicter, de la même manière qu'avec le système réel, les portions de gène à exécuter.

Nous avons donc écrit notre code de façon à ce qu'il n'y ai presque pas de différences entre la simulation et le cas réel. Toute la partie algorithme génétique est exactement la même, mais, là où pour le cas réel, la voiture doit exécuter un gène, la simulation l'exécute numériquement, grâce à une modélisation. La distance à l'objectif est alors obtenue directement, pas besoin du recours au module de détection de la voiture avec une caméra. On en extrait les données sous le même format, seulement ce processus est beaucoup plus rapide pour faire des essais sur plusieurs paramètres.

Fonctionnement de la simulation

Nous avons choisi de modéliser la voiture de sorte à s'appuyer sur un modèle simplifié, peut coûteux en temps de calcul et en technicité d'implémentation. Ce modèle s'appuie sur les hypothèses suivantes :

— La voiture évolue dans un plan, les effets dues à la hauteur de la voiture sont négligeables. Physiquement cela revient a supprimer les mouvements verticaux mais aussi les rotations autour des axes contenus dans le plan.

— Le problème est isostatique : le poids est réparti sur les 4 roues au prorata des distances avec le centre de gravite.
— Les moteurs n'ont pas d'éléments inductif ni résistif, et l'ensemble moto-réducteur n'a pas d'inertie. De ce fait, les roues prennent instantanément la vitesse de consigne.
— Le frottement entre les roues et le sol prend en compte un modèle de frottements secs et un modèle de frottements visqueux.

Ce modèle se résumant à quelques équations différentielles, nous utilisons la méthode d'intégration par trapèzes pour le résoudre. Pour cela, nous procédons par petits pas de temps suffisamment cours pour que l'on puisse dire que les paramètres dérivés (accélérations) n'évoluent pas dans ce petit laps de temps. Chaque incrément suit les étapes suivantes :

1. Calculer les vitesses relatives entre les roues et le sol pour chacun des 2 axes du plan.
2. En déduire les efforts du sol sur chaque roues. Pour calculer ces forces, on utilise le modèle de frottement sec que l'on rend continu au voisinage de 0 de sorte a ne pas faire diverger les solutions.
3. Déduction du tenseur (moment, forces) entre le référentiel du sol et de la voiture.
4. Calcul des accélérations linéaires et angulaire de la voiture.
5. Intégration des accélérations et des vitesses pour en déduire le nouvel état de la voiture.

Paramètres du modèle Il faut ensuite fixer tous les paramètres du modèle afin de simuler au mieux la réalité. Les paramètres sont les suivants :

— La masse de la voiture - *On l'a pesée avec une balance.*
— Les entraxes - *Les dimensions ont été récupérées sur le site de la plateforme roulante.*
— Le coefficient de frottement - *Une simple expérience qui consiste a prendre la tangente de l'angle minimum a partir du quel la voiture commence a glisser permet de l'estimer.*
— Le centre de gravité - *On a tenter de trouver ou il fallait placer le doigt pour pouvoir soulever la voiture avec un seul doit. Le point d'équilibre est un peu approximatif.*
— Le moment d'inertie projeté sur l'axe vertical - *Nous n'avons pas fait d'expérience pour l'estimer. Nous nous sommes dit qu'un gros ballon de football devait avoir le même ordre de grandeur de moment inertiel que notre voiture. Nous avons donc calcule l'inertie d'un ballon de foot.*

Ces valeurs ne sont donc pas toutes exactement fidèles à la voiture réelle, mais nous avons surtout fait attention aux ordres de grandeur. Les résultats des simulations donnaient des valeurs très plausible, nous avons donc estimé que cette modélisation était suffisante pour les besoins du projet.

Il reste à déterminer le pas de temps. Plus ce dernier est court, plus les résultats sont précis. L'approximation linéaire de l'évolution des grandeurs physiques est d'autant plus vrai dans ce cas. Par contre, le temps de simulation est inversement proportionnel à ce pas de temps. Ainsi, un pas de temps trop petit rend les simulations très lente a s'exécuter. De plus, il ne sert a rien de résoudre très finement notre modèle, puisqu'il est déjà biaisé de par sa conception et l'imprécision de ses paramètres. Il faut donc faire un compromis. A force d'essais, nous avons gardé un écart Δ t = 1 ms.

Cette simulation nous permet de mener des batteries de tests avant que le système réel soit totalement au point. Le code est disponible dans le fichier "simulation.py" se trouvant en annexe.

III Tests et résultats

III.1 Simulation virtuelle

Nous allons maintenant exploiter la simulation à ses fins, paramètre par paramètre, de sorte à observer l'influence de chacun d'eux. Cette démarche vise à trouver les meilleurs paramètres permettant d'optimiser notre travail. Dans le cadre d'une hypothétique étape suivante à ce projet, cela ferait gagner beaucoup de temps d'apprentissage sur le cas réel.
Dans un premier temps nous avons codé des simulations avec des vitesses positives et négatives, permettant à voiture d'avancer et reculer. Nous avons pu observer qu'au bout d'un nombre faible de générations (une ou 2 tout au plus), quasiment aucune voiture ne recule si le point d'arrivée est devant elle.
Les figures ci-dessous permettent d'illustrer les trajectoires de 100 individus sur les deux premières générations d'une telle simulation.

Concernant les paramètres gardés constants, nous avons mis en place une phase de tests nous permettant de choisir des valeurs respectant le compromis suivant : obtenir des résultats exploitables et réalistes en un temps raisonnable. Ainsi, les paramètres par défaut sont la durée d'une portion (*portion_duration* = 2s) et le pourcentage maximal d'individus en dehors de la zone d'arrivée toléré (*tolerated_ind_percentage* = 10). Pour ce qui est des autres paramètres, nous les avons fait varier un par un, afin d'étudier leur influence sur la convergence de l'algorithme ainsi que pour trouver les valeurs par défaut à utiliser pour une simulation "classique", la plus réaliste possible.

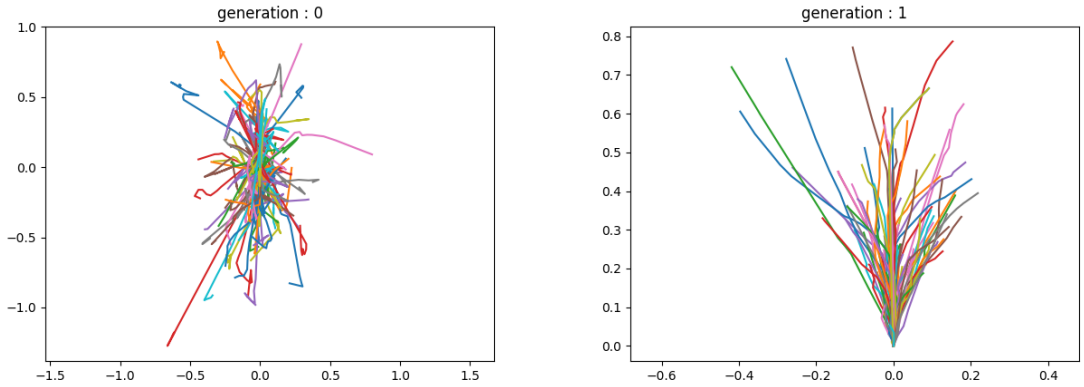


FIGURE 5 – Simulation du mouvement de 100 voitures

1. **Variation de la taille de la population**

   Nous commençons donc avec le paramètre "taille de la population", c'est-à-dire le nombre d'individus considérés à chaque génération. Plus celui-ci est petit, moins il y a de données à entraîner et plus rapide est l'apprentissage. Il est donc intéressant de chercher un compromis entre taille assez faible et une bonne rapidité de convergence.

   Nous observons sur ce graphique le score moyen de la population en fonction du nombre d'individus. Il faut noter que les cas 5 individus et 500 individus ont été enlevés du tableau de données, ceux ci présentant des valeurs trop différentes et rendant le graphique illisible.

   Nous notons donc que la tendance entre ces différents cas est assez similaire. La courbe 10 individus par génération est cependant un peu plus bas ce qui semble logique, c'est assez faible. La courbe 20 individus semble elle très bonne car présentant le meilleur score après 4 générations. Mais ils ne faut pas oublier que tous ces résultats sont issus d'un processus aléatoire et que ces courbes sont donc amenées à varier.

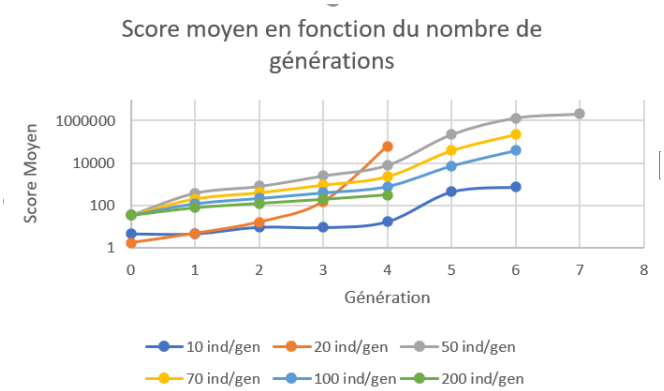


FIGURE 6 – Données de simulation sur l'influence du nombre d'individus par générations

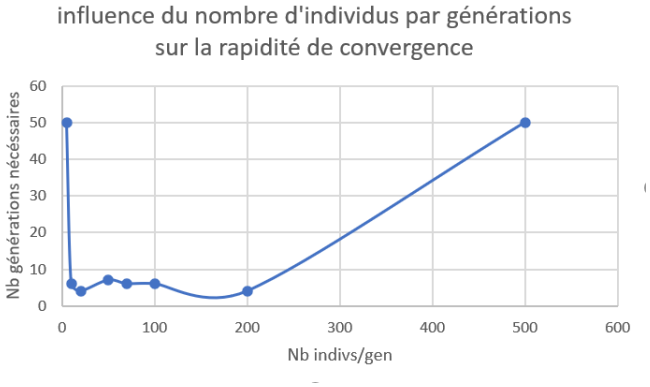


FIGURE 7 – Données de simulation de l'influence du nombre d'individus sur la vitesse de convergence

Nous retenons donc l'intervalle [20;200] individus/générations comme assez satisfaisant d'un point de vue des résultats en fonction de la taille de la population.

2. **Variation du rayon de tolérance autour du point d'arrivée**

Les données de test suivantes nous permettent d'étudier l'influence du rayon de tolérance autour du point d'arriver sur la convergence de l'algorithme. Les données de simulation sont regroupées dans les graphes suivants :

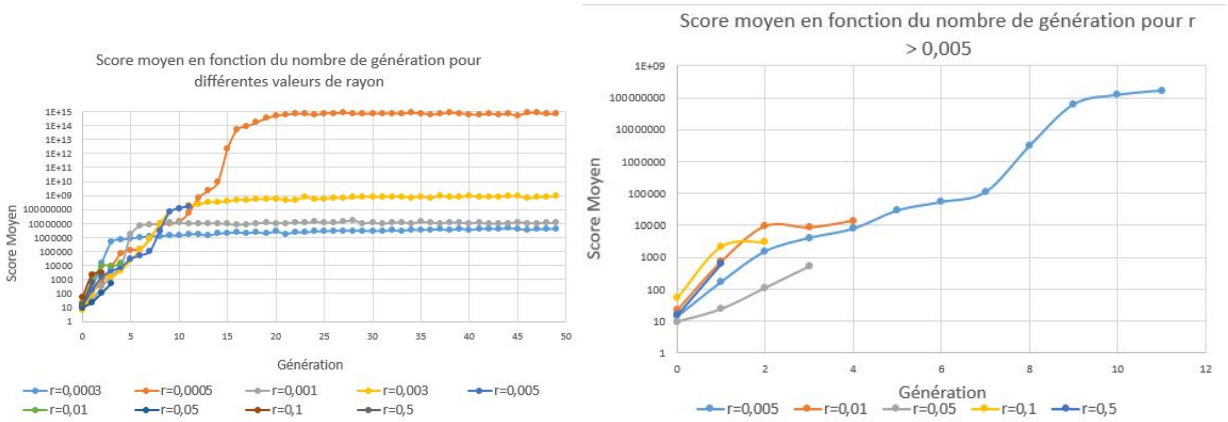


FIGURE 8 – Données de simulation sur l'influence du rayon de tolérance autour du point d'arrivée

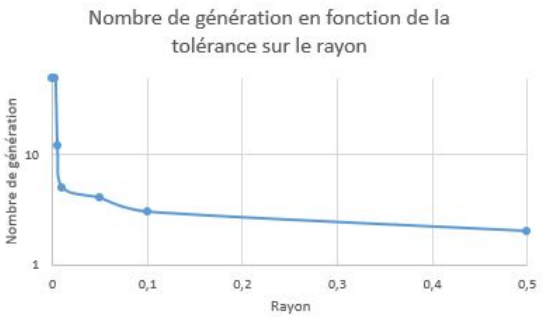


FIGURE 9 – Données de simulation sur l'influence du rayon de tolérance autour du point d'arrivée

Sur le graphique de gauche nous pouvons remarquer que l'on ne parvient pas à la précision souhaitée au bout de 50 générations pour des rayons inférieurs à 0,005, malgré un score élevé. La figure de droite nous pouvons observer plus précisément l'influence du rayon lorsque l'algorithme arrive à converger. On peut notamment remarquer la convergence au bout de deux générations, avec un score relativement faible lorsque le rayon de tolérance est de 0,5. Ce rayon est trop large pour donner un résultat convenable. Avec des rayons compris entre 0,1 et 0,05, le score est encore relativement faible.
Nous pouvons conclure que l'efficacité optimale est obtenue pour des rayons compris dans l'intervalle [0,005 ;0,01].

3. **Variation du facteur de mutation**

Dans cette partie, nous étudions l'influence du facteur de mutation. D'un point de vu théorique, le facteur de mutation peut être vu comme l'aptitude a explorer l'espace des paramètres. Autrement dit plus le facteur de mutation est grand, plus les nouvelles générations testerons de nouveaux mouvements que leurs ancêtres n'auront pas testés. A l'inverse, un petit facteur de mutation traduit des individus peu curieux, qui se contentent d'optimiser ce qu'ils connaissent déjà. Aucun des ces deux extrêmes n'est souhaitable.

— Si le facteur est trop petit, on risque de ne pas suffisement explorer l'ensemble de toutes les solutions possibles. Chaque nouvel individu peut etre vu comme un point aleatoire dans l'espace des parametres. L'etape de la reproduction sans mutation permet d'aller explorer de nouveaux points qui sont contenus a l'interieur du nuage de points forme par la generation precedente. Recursivement, on se rend vite compte qu'il n'est pas possible d'aller explorer l'extérieur du nuage de point initial. Si le nuage de point initial couvre tout l'espace, ce n'est pas genant. Seulement, l'espace dans lequel on se situe est un espace a 40 dimensions (ce qui n'est en soit pas enorme). Il faudrait beaucoup d'individus pour avoir des chances de tout couvrir. Du coup a prendre un facteur de mutation trop petit on risquerait de ne jamais pouvoir trouver une solution vraiment optimale.

— Au contraire, si le facteur de mutation est trop grand, on risque de ne jamais converger. Quand deux individus se reproduisent, ils ameliorent statistiquement les performances de l'anfant, mais cette amelioration est maigre. En effet l'enfant n'herite pas seulement des bons cotes des parents mais aussi des defauts. Il est tres difficile de predire l'amelioration de chaque generation mais ce qui est certain, c'est qu'elle n'est jamais enorme. Or faire muter un individu consiste a le bouger legerement dans une direction aleatoire de l'espace des parametres. Si ce deplacement est en moyenne plus grand que l'amelioration de l'enfant, le caractere aleatoire de la mutation l'emporte. C'est un peu comme si l'enfant etait rebel et se rebifait contres ces parents sans rien vouloir apprendre d'eux. Chaque nouvelle generation se comporte comme la toute premiere, qui n'est pas du tout efficace.

On s'attend donc a trouver un facteur de mutation optimal, qui serait un bon compromis entre ces deux extrêmes. Les données de simulation sont regroupées dans les graphes ci-dessous :

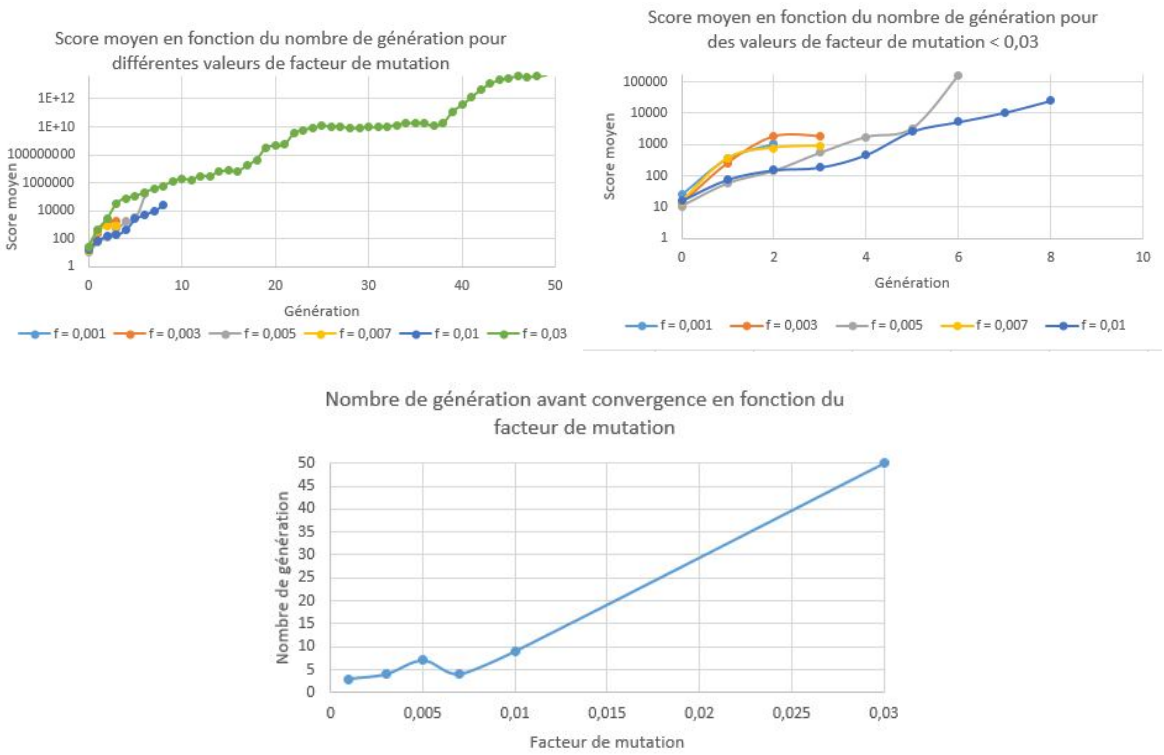
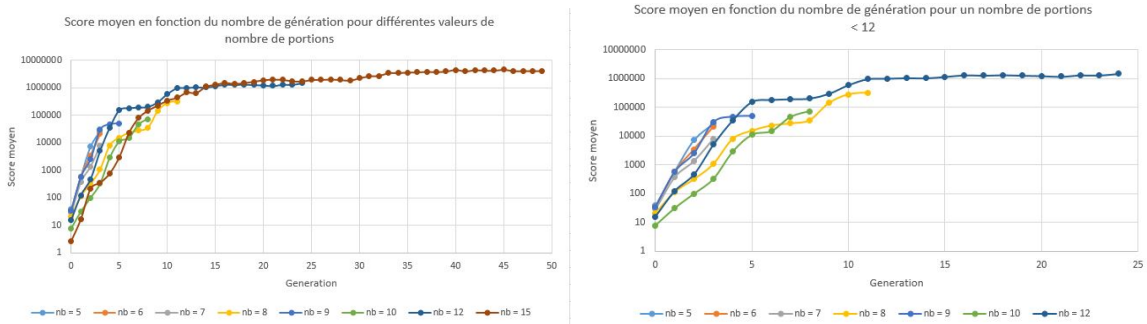


FIGURE 11 – Données de simulation sur l'influence du facteur de mutation

Nous pouvons remarquer sur le graphique de gauche de la figure 11 que l'on ne parvient pas à atteindre la précision souhaitée au bout des 50 générations si le facteur de mutation est supérieur à 0,03. De plus, nous pouvons remarquer qu'au dessous de 0,03 l'influence du facteur de mutation n'est pas significative.
Ces données de simulation nous permettent de conclure qu'il faut prendre un facteur de mutation inférieur à 0,03 afin de faire converger l'algorithme en moins de 50 générations.

4. **Variation du nombre de portions**

Enfin, la dernière simulation effectuée nous permet de tester l'influence du nombre de portions. Le nombre de portion influe sur le nombre de changement de direction que la voiture va effectuer durant la simulation.
La durée d'une portion est constante, le nombre de portion influe donc sur la durée totale de la simulation.



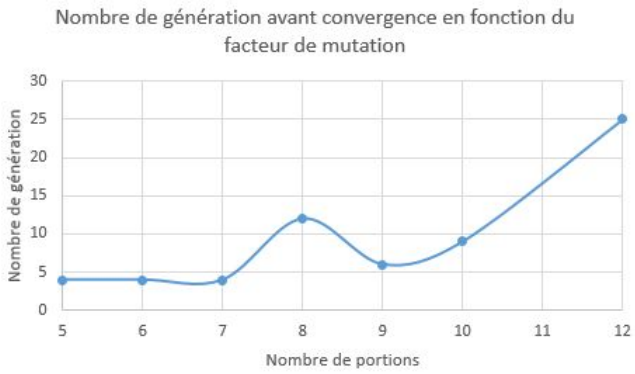


FIGURE 13 – Données de simulation sur l'influence du nombre de portions

Dans un premier temps nous pouvons remarquer la convergence du score autour des valeurs $10^6/10^7$ lorsque le nombre de portions est suffisamment important.

De plus, la voiture n'arrive pas au point souhaité au bout des 50 générations si le nombre de portions est supérieur à 12.

Le graphique de droite permet d'analyser plus précisément l'influence de ce rayon. On remarque que le nombre de générations avant convergence pour 12 portions est bien plus élevé que celui pour 10 portions, alors que le score final est du même ordre de grandeur. Il n'est donc pas nécessaire ni efficace de prendre un nombre de portions supérieur à 10.

Pour un nombre de portion compris entre 5 et 10 les résultats obtenus sont proches en termes de score moyen mais également au niveau du nombre de génération avant d'atteindre le point d'arrivée.

On remarque que le nombre de génération avant d'atteindre le point d'arrivée croit exponentiellement avec le nombre de portions.

On notera néanmoins que la simulation avec 9 portions nécessite un plus grand nombre de générations pour converger que pour 10 portions, ce qui semble aberrant a priori. Cependant, il s'agit de simulations isolées. En effet, réaliser un grand nombre de simulations avec ces mêmes paramètres nous auraient permis de faire une étude statistique encore plus réaliste et précise, en éliminant les éventuels cas éloignés d'une potentielle courbe de modélisation.

Nous retiendrons un nombre de portions compris entre 6 et 10.

III.2 Implémentation réelle avec la voiture

Le manque de temps nous as contraint à ne pas pouvoir mener de test poussés sur la voiture, et à ne pas pouvoir récolté de données exploitables. Cependant, nous avons réussi, au terme de ce projet, à exécuter 5 génération d'une dizaine d'individus en utilisant le système réel. Ainsi, en disposant de plus de temps, nous pourrions confronter la simulation a la réalité.

Conclusion

Nous avons, durant ce projet, essayé au mieux de répondre au cahier des charges. Pour pouvoir avancer le développement de l'algorithme génétique sans être retardés par les problèmes physiques inhérents à la voiture, nous avons conçu une simulation du système. Nous pouvons donc conclure qu'en théorie, chaque individu doit posséder 10 portions, une portion doit durer 2 secondes, avec un nombre d'individus hors zone d'arrivée de 10% avant l'arrêt. Une génération doit contenir environ 100 individus, On considère qu'un individu est arrivé à la bonne destination s'il est dans un rayon de 20cm. Le facteur de mutation doit se situer entre 0.01 et 0.02 pour les paramètres. Dans ce cas, il faut une dizaine de génération pour arriver au résultat souhaité, soit 30000 secondes pour 15 générations (environ 8h30 d'entraînement, sans compter le temps de retour au point de départ entre chaque individu). Il est donc compliqué, avec nos moyens, de pouvoir tester le cas réel en restant près de 10h autour de la voiture pour parer d'éventuels bugs.

Réponse au cahier des charges

1. **La voiture doit être capable de se déplacer d'un point A à un point B le plus efficacement possible,donc en ligne droite, en autonomie et sans entraves.** : c'était le point clé de notre projet. A l'exception des batteries, et donc de la partie "sans entrave", l'ensemble des sous systèmes nécessaire à la réussite de ce point sont fonctionnels et robustes. Malheureusement, par manque de temps, nous n'avons pas pu effectuer un nombre suffisant de tests pour entraîner le système physique jusqu'au bout.
2. **La voiture doit utiliser un algorithme génétique pour apprendre à effectuer les tâches demandées en 1** : C'est bien a l'aide d'un algorithme génétique que la voiture apprend à se déplacer en ligne droite. Nous avons pu tester cet algorithme sur une simulation de la voiture, et voir qu'il convergeait effectivement vers la solution attendue.
3. **Durant l'entraînement de l'algorithme, la voiture doit partir du même point à chaque nouvel essai.** : L'algorithme de retour au point de départ est le dernier élément que nous avons pu déboguer dans son intégralité. Elle est maintenant robuste, et permet à la voiture de rejoindre le centre de la zone de travail en partant de n'importe quel point.
4. **L'ensemble des codes écrits doit être documenté et écrit en anglais, en respectant les normes d'usage spécifique à chaque langage.** : c'est normalement le cas, cependant, certaines parties sont moins bien documentées, notamment à cause de la phase de déboguage.

Pour pousser au plus loin le projet, il serait idéal d'avoir une voiture fonctionnant sur batterie. De plus, étudier la réaction de l'algorithme entrainé face à l'ajout d'un obstacle (temps de réponse au problème, les paramètres importants, etc...) est une démarche intéressante pour mieux comprendre le fonctionnement de cette famille d'algorithme. Si plus de temps nous était proposé pour ce projet, nous fiabiliserions les différentes parties matérielles de la voiture, ainsi que nos algorithme d'acquisition, afin de pouvoir comparer les cas simulés et les cas réels.

Bibliographie

Apprentissage par renforcement, architecture cognitive, adaptation à une panne :

http://www.palais-decouverte.fr/fileadmin/fileadmin_Palais/fichiersContribs/au-programme/activites/1chercheur1manip/Ressources/Des_robots_qui_apprennent/Des_robots_qui_apprennent_1C1M.pdf

Principe des algorithmes génétiques et code :

https://skyduino.wordpress.com/2015/07/16/tutorielpython-les-algorithmes-genetiques-garanti

Le guide des batteries Lipo - RC Team :

https://www.rcteam.fr/fr/blog/le-guide-des-batteries-lipo-n5

Code Arduino et documentation de la voiture : https://wiki.dfrobot.com/Cherokey_4WD_Mobile_Platform__SKU_ROB0102_

Annexes

Illustration d'une simulation

Images extraites de l'étude de la variation de la taille d'une population - trajectoires des individus

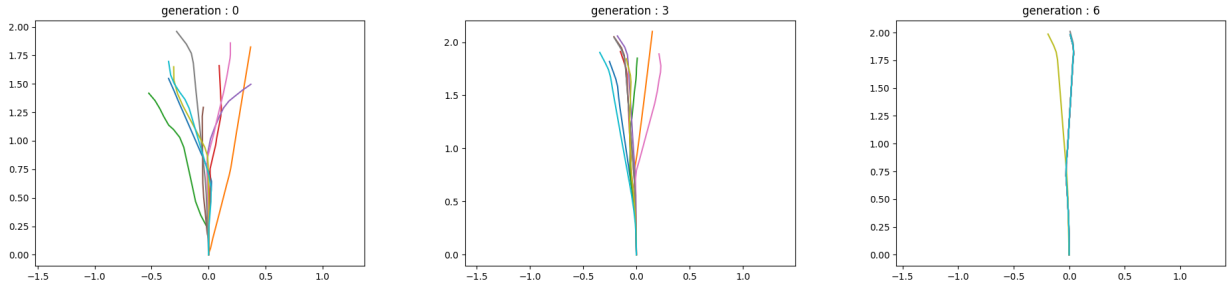


FIGURE 14 – Simulation avec une population de 10 individus

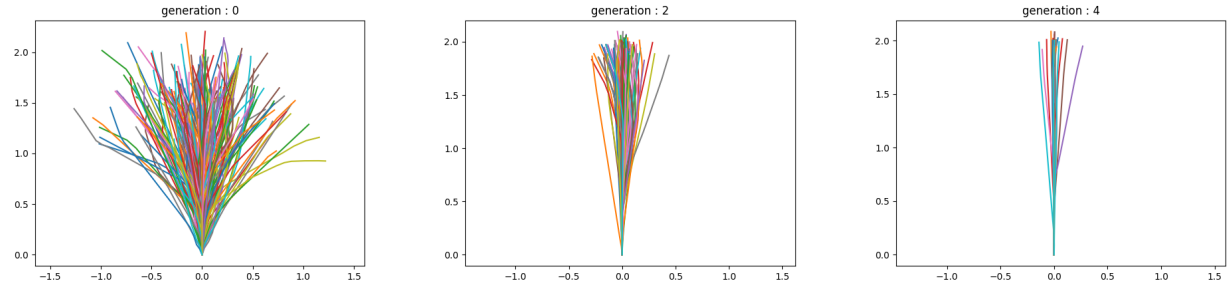


FIGURE 15 – Simulation avec une population de 200 individus

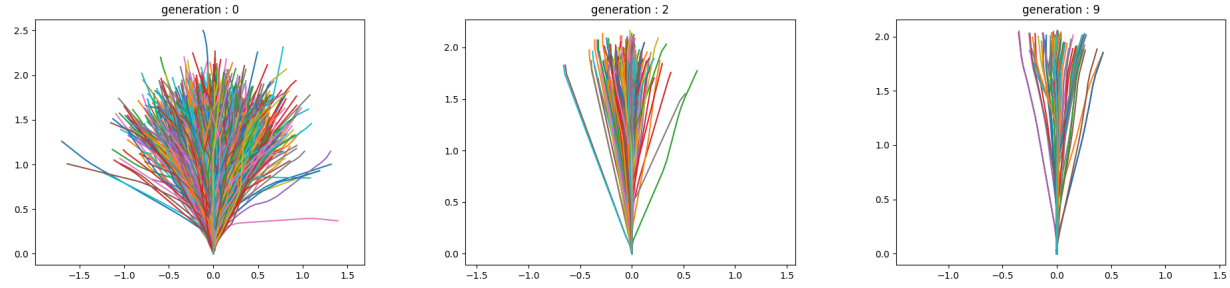


FIGURE 16 – Simulation avec une population de 1000 individus

Code python du projet

Le code du projet et le détail de chaque fichier est a retrouvé sur le GitHub dédié :

`https://github.com/engelba/RobotGenetic`

Calendrier du travail

	Taches	Du 12/01 au 26/01	Du 02/02 au 23/02	Du 23/02 au 02/03	09/03	23/03	06/04	13/04
Elaboration du cahier des charges		Tous						
	Codage de l'interface Roues-Arduino	M & R						
	Codage de l'interface Arduino-Raspberry	M & R						
Interfaces	Communication avec la voiture					B & R	B & R	
	Déterminer position de la voiture	B & V	B & V					
Traitement d'image	Déterminer orientation de la voiture		B & V	V				
	Créer les fonctions de base et la structure	E	M & R	R				
	Tests de l'algorithme génétique avec simulation			M & B	M			
	Tests de l'algorithme génétique sur la voiture							
Algorithme génétique	Fichier de récupération des données de test					E & M & V	E & M & V	
Algorithme déterministe pour ramener la voiture à sa position initiale	Trouver l'algorithme et le rédiger		E	E	E & V			
	Tests			E				
Matériel physique	Alimentation de la voiture				R	R & M		
	Alimentation de la Raspberry				R	R & M		
	Débogage du code assemblé						Tous	B & R
	Rédaction du rapport							E & M & V

FIGURE 17 – Diagramme de Gant