Grenoble INP - Phelma

Robot serpent qui apprend à compenser une casse

Rapport final du projet

Visuel du robot serpent

Romain BAILLY
Esteban GOUGEON
Colin EPALLE
Olivier VU THANH

2e Année - 25 Avril 2019

Table des matières

I Présentation du projet **2**
- I.1 Contextualisation . . . 2
- I.2 Cahier des charges . . . 2
- I.3 Organisation du projet . . . 3

II Structure **3**
- II.1 Introduction . . . 3
- II.2 Choix de conception . . . 3
  - II.2.1 Les servomoteurs AX-12A . . . 4
  - II.2.2 Choix des roues et conception 3D . . . 4
  - II.2.3 Ensemble bras/support de l'appareil photo . . . 6
- II.3 Conclusion . . . 7

III Pilotage **8**
- III.1 Introduction . . . 8
- III.2 Choix de conception . . . 8
- III.3 Problème d'identification . . . 10
- III.4 Optimisation du mouvement du serpent . . . 10
- III.5 Test temporel de la boucle de gestion du mouvement . . . 10
- III.6 Conclusion . . . 12

IV Traitement d'images **12**
- IV.1 Introduction . . . 12
- IV.2 Choix de l'appareil photo . . . 12
- IV.3 Réalisation . . . 13
  - IV.3.1 Pygame . . . 13
  - IV.3.2 Détection du serpent . . . 13
  - IV.3.3 Prévision de la trajectoire . . . 14
- IV.4 Conclusion . . . 14

V Algorithme génétique **15**
- V.1 Introduction . . . 15
- V.2 Choix de conception . . . 16
  - V.2.1 Fonction d'évaluation d'un individu . . . 16
  - V.2.2 Paramètres de l'algorithme génétique . . . 17
- V.3 Conclusion . . . 17

VI Conclusion **17**
- VI.1 Résultats du projet . . . 17
- VI.2 Pistes d'améliorations . . . 18
- VI.3 Ressenti du projet . . . 18

VIIAnnexes **19**

A Distances parcourue en 1200 ticks **19**

B Diagramme de Gantt prévisionnel **20**

Bibliographie **21**

I Présentation du projet

I.1 Contextualisation

Deep Learning, réseaux neuronaux, algorithmie génétique... Ces mots font de plus en plus écho dans le monde scientifique d'aujourd'hui, en particulier dans l'univers de la robotique. L'engouement autour de la thématique du "Machine Learning" ne fait que s'accroître et les demandes de robots autonomes ne font que gonfler. C'est dans l'idée de construire un robot qui s'adapte à son environnement que le projet a été créé.

Le projet a pour objectif la création d'un robot serpentiforme devant se déplacer à l'aide d'algorithmes ayant des paramètres facilement modifiables. L'algorithme génétique est alors une solution de choix, au coeur de l'apprentissage du robot.

I.2 Cahier des charges

L'objectif de ce projet est dans un premier temps de concevoir un robot serpentiforme capable de se déplacer d'un point A à un point B. Il sera composé de 12 servomoteurs double axe posés sur des roues. Les servomoteurs ainsi que la carte de contrôle seront alimentés sur le secteur. À la manière d'un serpent, le robot se déplacera en suivant une trajectoire quasi sinusoïdale. Le déplacement du serpent sera suivi à l'aide d'une caméra comme capteur servant à mettre en place un asservissement en boucle fermée. Une fois le robot et l'asservissement de son déplacement fonctionnels, l'objectif est de simuler la panne d'un servomoteur et d'entraîner le robot à trouver une nouvelle formule de déplacement par algorithme génétique. À terme le robot doit être capable de s'entraîner de manière autonome dans un espace défini par le cadre de la (des) caméra(s).

Le robot est composé d'une douzaine de servomoteurs sur roues. La première étape consiste à choisir les servomoteurs utilisés puis les tester indépendamment puis en chaîne, afin de pouvoir par la suite être capable d'envoyer la formule de déplacement du robot à chaque servomoteur. En parallèle du contrôle des servomoteurs et de l'établissement du code de déplacement du serpent, la conception de la structure du serpent (base roulante de chaque servomoteur) sera effectuée. La pièce servant de support au servomoteur sera modélisée, vérifiée, corrigée et testée ; si elle ne convient pas, ce processus sera reconduit jusqu'à obtention d'une pièce adaptée. Une fois les différentes parties du robot réalisées, il conviendra de l'assembler afin de le tester dans son ensemble. En ce qui concerne le traitement d'image, une première prise en main des modules de traitement sera effectuée avant de trouver un moyen d'extraire la position du robot en temps réel puis d'en déduire le déplacement. Différents tests avec la maquette du robot pourront alors être mis en place. Enfin l'algorithme génétique sera codé, débogué puis testé sur le robot. Dans un premier temps, il sera obligatoire de rester à côté du robot afin de vérifier que tout se déroule bien mais à terme, il est prévu d'élargir le champ de vision des capteurs d'images et d'automatiser le processus d'apprentissage : détection de sortie de cadre, ré-initialisation de la position avant test, création de la nouvelle génération... Une fois le robot fonctionnel et l'entraînement terminé, un test final permettra de valider l'entièreté du projet. Différents programmes de tests à but uniquement démonstratif seront également créés pour la présentation finale.

I.3 Organisation du projet

Le projet commence le 15 janvier 2019 et donne lieu à une soutenance le 30 avril 2019. Le projet est dirigé par RIVET BERTRAND et l'équipe est composée de BAILLY ROMAIN, EPALLE COLIN, GOUGEON ESTEBAN et VU THANH OLIVIER. Le planning prévisionnel (sous forme d'un diagramme de Gantt) est disponible en Annexes B p.20.

L'organisation du projet s'est articulé autour de 4 grands axes :

1. Le choix des composants et la conception de la **structure** du robot
2. La création des fonctions et les tests permettant le **pilotage** du robot
3. Le traitement de l'image pour la **localisation** du robot et l'évaluation de la performance des individus de l'algorithme génétique
4. La création des fonctions de **l'algorithme génétique** et l'entraînement du serpent

Même si les membres du groupe se sont plus intéressés à certaines parties que d'autres, chaque membre maîtrise le fonctionnement du robot et est capable de debug/modifier les paramètres de certaines fonctions (notamment pour l'algorithme génétique). Les décisions concernant les points majeurs du projet ont été prises par l'ensemble du groupe. Chaque partie étant cependant plus maîtrisée par un membre plus qu'un autre, certains choix ont été effectués de manière autonome, même si une justification ultérieure était quasi-systématiquement effectuée. Enfin, l'ensemble du code écrit et documenté par les membres du groupe a été géré à l'aide d'un gestionnaire disponible à la référence [1] de la bibliographie.

II Structure

II.1 Introduction

La première étape du projet a été de concevoir la structure du robot serpentiforme. Afin d'obtenir un robot dont le mouvement est similaire à celui d'un serpent (*i.e.* quasi-sinusoïdal), le robot doit être composé de plusieurs modules pouvant chacun tourner indépendamment des autres dans le plan orthogonal au mouvement du serpent. La structure de l'année précédente était ainsi composée de plusieurs plaques de bois liées par des servomoteurs. Cette structure n'était pas propice à un déplacement fluide et les contraintes liées au couple exercé sur les moteurs laissa suggérer un changement de structure.

Afin de garantir une structure fiable, modulaire, facile à adapter de manière embarquée et rendant au mieux du mouvement d'un serpent, le choix s'est porté sur une mise en série de servomoteurs sans plaque physique pour les relier.

II.2 Choix de conception

Afin de pouvoir satisfaire à l'envie de modularité du robot (ajout ou suppression de blocs élémentaires du robot) mais aussi afin de pouvoir supporter un couple suffisant tout en ayant une certaine facilité pour le code, ce sont vers des servomoteurs numériques que les recherches se sont portées. En comparaison à leurs homologues analogiques, la commande des servomoteurs numériques se fait par écriture sur registre. Ils sont souvent accompagnés de carte de contrôle et d'alimentation conçues par l'entreprise les réalisant. Disposant à l'école de servomoteurs **Dynamixel AX-12A** de *Robotis* (référence [2]), ce sont ces derniers qui ont été retenus.

II.2.1 Les servomoteurs AX-12A

Les servomoteurs double axe AX-12A (cf Figure 1) sont des servomoteurs à commande numérique. Dans le cadre du projet, les caractéristiques techniques utiles de ces servomoteurs sont les suivantes :

- légèreté : 55g
- pas de résolution faible : 0.29°
- vitesse de rotation élevée : 60° en moins de 0.2s
- couple suffisant : 1.52N.m
- système de communication et alimentation de servos en série déjà existante
- retour de données complet (Position, système de sécurité lors des surcharge)
- mise en série mécanique et électronique aisée

FIGURE 1 – Photo d'un kit servomoteur AX-12A

Les servomoteurs AX-12A ont été conçus pour la conception de robot. Pour la conception du robot serpent, un nombre de douze servomoteurs a été retenu. Ce nombre permet d'avoir un robot suffisamment long pour effectuer un mouvement fluide mécaniquement mais aussi à l'oeil. Cela nécessite cependant de pouvoir alimenter tous ces servomoteurs. Le composant permettant de le faire est un **SMPS2Dynamixel**, composant qu'il est possible d'alimenter à l'aide d'une batterie ou sur secteur. Par souci de praticité, le courant a été délivrée à l'aide d'une alimentation stabilisée à 11.2V et ne pouvant dépasser les 2A (afin de protéger les servomoteurs de la surchauffe en cas de couple trop élevé). La communication entre les servomoteurs est faite à l'aide du composant **USB2Dynamixel** en TTL 3fils et les instructions sont directement envoyées depuis un ordinateur en liaison USB. La structure est maintenant bien établie et il ne reste plus qu'à mettre le serpent en mouvement.

II.2.2 Choix des roues et conception 3D

Il manque cependant au serpent un moyen d'adhérer au sol dans la direction de son déplacement, quelque chose de similaire à la peau écailleuse du serpent. Ne disposant pas de membrane écailleuse, le choix naturel est alors l'ajout de roues en dessous de chaque servomoteur. Les roues ont alors pour rôle d'empêcher au serpent de glisser dans la direction orthogonale à son déplacement (dans le plan de déplacement du serpent).

En ayant dans l'optique de pouvoir facilement remplacer une paire de roues cassée ou un servomoteur défectueux, ce sont des roues montées sur un bloc *LEGO®* qui ont été choisies (cf Figure 2 issue de [3]). Les roues choisies devaient avoir un **diamètre suffisamment petit** devant la taille des servomoteurs (par souci d'encombrement), avoir une **base suffisamment large** afin de permettre le bon maintien du servo (large de 32mm) et être suffisamment épaisse afin d'avoir une meilleure adhérence. Enfin l'avantage du bloc *Lego* est sa facilité à emboîter.

FIGURE 2 – Photo des roues du robot

Afin de pouvoir fixer les roues au servomoteur, il a alors fallu concevoir un support en 3D. L'entièreté de la conception a été réalisée à l'aide de *Autodesk Fusion 360*. Le support permet de faire le lien entre les roues et le servomoteur. Ainsi il a fallu prendre connaissance des dimensions des blocs *Lego* afin de pouvoir assurer une bonne fixation aux roues. Le point le plus délicat fut cependant de trouver un moyen de maintenir efficacement le servomoteur au support. La pièce 3D devant être peu épaisse, un vissage entre le support et le servo n'était pas envisageable au risque de casser le support. Un arc épousant la forme du servomoteur a alors été conçu. Les dimensions de la pièce sont très proches de celle du servo et un encastrement du servo dans le support permet alors de le maintenir tout en conservant un bon maintien. La création de cette pièce a été le fruit de plusieurs tests et impressions afin d'obtenir au final le support présent Figures 3 et 4 :

FIGURE 3 – Visuel du support 3D : vue du dessus

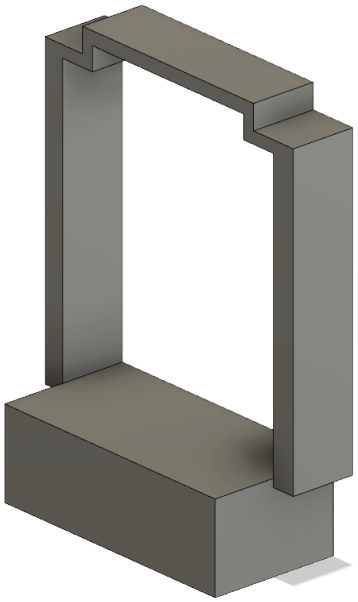
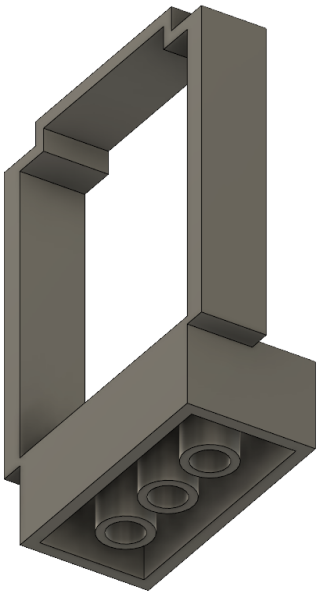


FIGURE 4 – Visuel du support 3D : vue du dessous



La conception de cette pièce a bien fait l'oeuvre de nombreuses remises en question d'un point de vue fonctionnel. Ce n'est qu'au bout de la quatrième impression que cette pièce a pu voir le jour et s'avérer fonctionnel pour le robot. Les premières versions étaient soit inadaptées, soit trop peu solides, soit mal conçues. C'est également cette pièce qui a servi de base pour le support des cartes d'alimentation et de communication. Ce support a tout simplement pour but le bon maintien de ces deux cartes lors du déplacement du serpent, tout en prenant en compte le fait qu'elles doivent être aisément enlevable tout en restant solidement maintenues. Le rendu final de cette pièce (obtenu après "seulement" 2 impressions test) est observable Figure 5 :

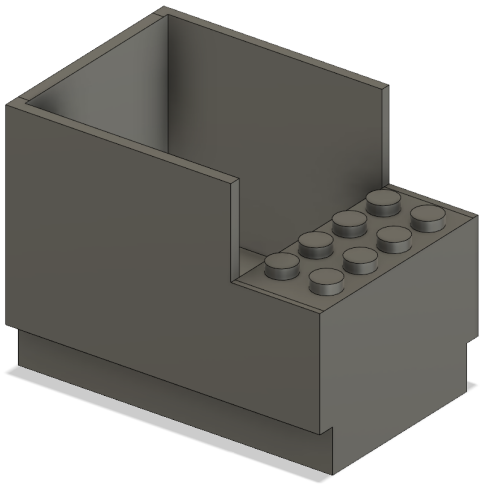


FIGURE 5 – Visuel du support 3D de la tête

Cette pièce consiste tout simplement en une "boîte" dans lequel l'on vient mettre les deux cartes. Les dimensions sont tout de même prévues pour que les cartes ne puissent pas bouger pendant le déplacement du serpent. Tout comme le support classique, il est prévu de mettre des roues sous ce support.

II.2.3 Ensemble bras/support de l'appareil photo

La dernière construction mécanique mise en place dans le projet ne concerne pas le robot. Dans la partie Traitement d'images (développée p.12), il est nécessaire d'avoir un support fixe de capture de l'image. La solution alternative mise en place au début du projet fut d'utiliser une webcam installée au plafond et reliée en USB à l'ordinateur faisant les calculs. Cette solution n'était cependant pas envisageable à long terme car la caméra est tombée à de nombreuses reprises. La construction d'un support stable et proche du plafond a alors été obligatoire. Ce support permettrait alors d'accueillir la caméra et de rester fixe dans la pièce, les marques au sol permettant d'être certain de son bon positionnement. L'assistant à l'éducation TRAVEAUX JULIEN a alors élaboré un bras au bout duquel nous placerions l'appareil photo (ou de quoi le soutenir). Ce bras doit être le plus proche du plafond possible (2.5m de haut) et ses pieds ne doivent pas gêner la zone de déplacement du serpent. Le premier essai n'étant pas stable (cf Figure 6), une consolidation a mené à l'établissement final du bras Figure 7 :

FIGURE 6 – Photo du premier bras

FIGURE 7 – Photo du bras consolidé

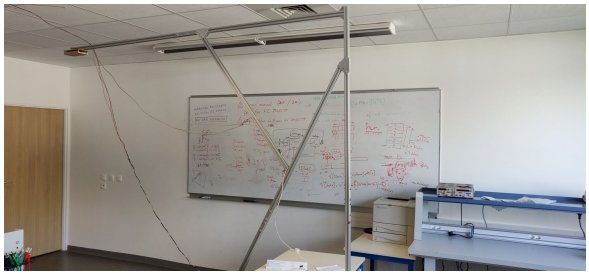
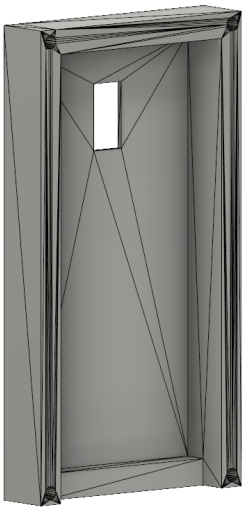


FIGURE 8 – Photo du robot serpent dans son ensemble



Au bout de ce bras se trouve le support de l'appareil photo. Pour des raisons expliquées dans la partie Traitement d'images p.12, la caméra se trouve être un téléphone portable relié en Wi-Fi à l'ordinateur de calcul. Le maintien de ce téléphone au bout du bras se fait alors à l'aide d'une pièce en bois (cf Figure 8) usinée à partir du modèle suivant créé à partir des dimensions du téléphone utilisé (Smartphone OnePlus 6t). Le support accueille le téléphone, le maintient stable et est fixé au bras à l'aide de 4 vis à ses 4 coins.

II.3 Conclusion

L'établissement de la structure mécanique du robot fut longue mais effectuée en parallèle d'autres tâches. Le robot étant maintenant mécaniquement fonctionnel (cf Figure 9), il convient maintenant de le piloter.

FIGURE 9 – Photo du robot serpent assemblé

III Pilotage

III.1 Introduction

Les servomoteurs n'étant pas analogiques mais numériques, une carte **USB2Dynamixel** a été utilisée afin de pouvoir les commander. Le protocole utilisé sera donc celui proposé par Dynamixel, à savoir la version 1.0. L'objectif de cette partie est d'obtenir un déplacement similaire à celui d'un serpent (de forme sinusoïdale), fluide et qui soit optimisé pour parcourir une distance en ligne droite la plus grande possible.

III.2 Choix de conception

Afin de pouvoir piloter les servomoteurs indépendamment les uns des autres, plusieurs paquets de données doivent être envoyés à la suite. La structure d'un paquet de bits est la suivante :

0xFF	0xFF	ID	Length	Instruction	Param 1	...	Param N	CHKSUM

Ainsi, chaque paquet peut être envoyé spécifiquement à un servomoteur via le champ ID. Il existe deux types de paquets : le paquet d'instruction et la paquet de statut. Il sera expliqué par la suite l'influence de cette séparation. Afin de gagner du temps, une librairie Python spécialement conçue pour le pilotage des servomoteurs Dynamixel a été utilisée [4]. Après plusieurs tests infructueux, il a été déduit que cette librairie n'avait pas été testée car dans sa majorité dysfonctionnelle. La bibliothèque *pydynamixel* de PATRICK GOEBEL a tout de même servi de base au développement d'une bibliothèque propre à ce projet. La version modifiée de la librairie *pydynamixel* est disponible sur ce github [1].

Les servomoteurs sont contrôlés de la façon suivante :

- À chaque servomoteur est attribué un ID allant de 1 à 12 en partant de la tête jusqu'à la queue
- À chaque période d'échantillonnage (aussi appelé un 'tick') le servomoteur i reçoit pour instruction de se déplacer à une certaine position θ_i (en degré), et cela à une certaine vitesse.
- $\theta_i(k)$ évolue selon l'équation suivante :

$$\theta_i(k) = \text{amplitude} \cdot \frac{n_{\text{period}}}{n_{\text{servo}}} \cdot cos(2 \cdot \pi \cdot (i \cdot \frac{n_{\text{period}}}{n_{\text{servo}}} + k \cdot \text{res})$$

où

— amplitude : amplitude maximale à laquelle peut osciller le servo

— n_{period} : le nombre de période physique visible sur le serpent. Cette valeur sera fixée à 1.

— n_{servo} : le nombre de servomoteurs utilisés pour faire le serpent. Cette valeur sera fixée à 12.

— k : l'avancement dans la propagation du cosinus, k est incrémenté tous les ticks

— i : ID du servomoteurs

— res : le pas de résolution

Le terme $2 \cdot \pi \cdot (i \cdot n_{period})/n_{servo}$ correspond au déphasage de chaque servomoteurs i par rapport à la tête. C'est grâce à ce terme que le cosinus se "propage" selon les servomoteurs.

L'amplitude ne peut pas être choisie au hasard. Une amplitude trop faible ne permettrait pas au serpent de bien osciller tandis qu'avec une amplitude trop élevée le serpent risque de se "manger la queue".

Soit un polygone régulier à n coins. L'angle entre chaque côté consécutif vaut $\frac{n-2}{n}180°$. Ce qui correspond à un angle θ_i de $\frac{2}{n}180°$ (*cf* Figure 10). Cette valeur de θ_i correspond à la valeur limite de l'amplitude seulement si le serpent peut former un polygone régulier. Or cela est à priori impossible à cause du cosinus. Il faut que la tête ne s'entrechoque pas avec le milieu du serpent. Passé ce milieu, il n'y a plus de risques. L'amplitude peut donc être doublée. Finalement,

$$\theta_{limite} = \frac{2}{n}360°$$

Un cas limite possible est visible sur la Figure 11. Il peut en être déduit que $\theta_i \in \left[-\frac{2}{n}360°; \frac{2}{n}360°\right]$

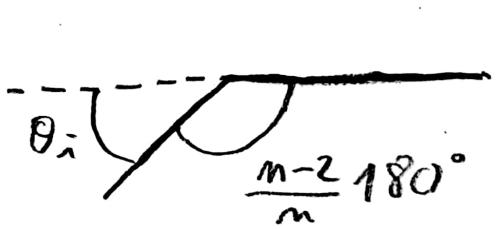


FIGURE 10 – Schéma angle limite

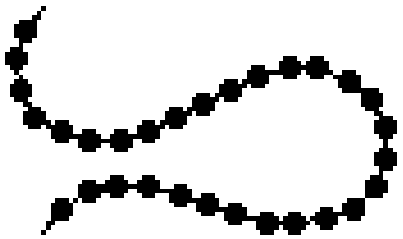


FIGURE 11 – Cas limite pour un serpent composé de 30 servomoteurs

Remarque: La Figure 11 a été obtenue avec une simulation du mouvement du serpent réalisée sur python. Le code est accessible sur github [5]

La formule de θ_i peut être complétée en ajoutant un offset :

$$\theta_i(k) = offset + amplitude \cdot \frac{n_{period}}{n_{servo}} \cdot cos(2 \cdot \pi \cdot (i \cdot \frac{n_{period}}{n_{servo}} + k \cdot res)$$

Il sera vu par la suite l'intérêt de l'ajout de ce paramètre offset.

L'introduction d'un offset change les limites pour lesquelles le serpent risque de se "manger la queue". La nouvelle limite est telle que :

$$\theta_{\text{limite}} = 2\left(\frac{360}{n} - |\text{offset}|\right)$$

Il en résulte aussi que $|offset| \in [0; \frac{2}{n}360]$. Cependant, si $|offset| = \frac{2}{n}360$, le serpent a juste une forme de polygone régulier et ne peut pas serpenter. Il convient alors de choisir arbitrairement que $|offset| < \frac{360}{n}$

L'ajout de cet offset a une conséquence directe sur le déplacement du serpent : son utilité est, comme son nom l'indique, d'ajouter un angle constant au serpent et ainsi de le faire tourner à droite ou à gauche selon le signe.

III.3 Problème d'identification

Lorsqu'un paquet est écrit, il faut faire attention à certains paramètres. Voici un exemple simple s'étant déroulé lors de la phase de prise en main des servomoteurs. Le but était d'assigner à chaque servomoteur un ID différent. Parmi les paramètres du paquet, l'un d'eux a pour but de préciser si l'instruction doit être écrite en mémoire à l'emplacement voulu sur 1 ou sur 2 bits. Une erreur avait été faite lors de l'écriture du paquet car ce paramètre avait été réglé sur 2 bits alors que l'ID est écrit sur 1 bits. Cela a eu pour conséquence d'écrire sur le bloc adjoint en mémoire que des 0. Ce bloc correspondait au baudrate. Toute communication avec le servomoteur était alors impossible. Après recherche, il se trouve que la valeur du baudrate vaut 200000 lorsqu'il n'y a que des zéros écrits en mémoire. Connaissant le baudrate, il a été possible de communiquer avec le servomoteur afin de reconfiguer le bon baudrate. Cet exemple met en valeur l'un des problèmes que nous avons rencontré au début du projet, la précision requise par l'écriture sur registres.

III.4 Optimisation du mouvement du serpent

Les premiers tests sur le serpent étaient très peu concluants. Celui-ci devait juste "serpenter" en ligne droite mais il faisait des mouvements extrêmement saccadé. Cela venait du fait que les servomoteurs étaient contrôlés en écrivant dans la mémoire l'instruction actuelle. Lorsqu'un paquet écrit des données en mémoire, un paquet en retour est envoyé afin de s'assurer qu'il n'y a pas eu d'erreurs lors de l'écriture. C'est le fameux paquet de statut. S'assurer qu'il n'y a pas eu d'erreur consomme donc du temps et c'est pour cela qu'il est difficile d'obtenir un mouvement non saccadé. Il a donc fallu trouver des solutions permettant de mieux échantillonner le mouvement du serpent. Une première piste a été de donner pour instructions au servomoteurs d'effectuer des aller-retours d'une certaine amplitude. Il n'y a alors que 2 instructions de positions en une période. La vitesse à laquelle cette instruction est réalisée est elle échantillonnée. Cette méthode est intéressante mais nécessite de réaliser un travail au préalable minutieux. C'est pour cela que devant le temps imparti, bien que les résultats devenaient presque exploitables, la décision a été prise d'opter pour une autre solution : ne plus attendre de confirmation d'écriture. Pour cela il suffit d'écrire en mémoire de tous les servomoteurs l'arrêt de l'envoi du paquet de statut.

III.5 Test temporel de la boucle de gestion du mouvement

Une période de $10\,ms$ a été choisie pour l'envoi des instructions de mouvement aux servomoteurs. Des tests de temps d'exécution ont été mis en place afin de s'assurer que cette période n'est pas trop faible, et ainsi que l'itération d'une boucle de gestion du mouvement du robot a bien le temps de se terminer dans la période imposée. Une itération d'une boucle de gestion du mouvement comporte 3 parties principales :

— le calcul des positions voulues (angles des servomoteurs) à envoyer dans la prochaine instruction

— La communication avec les servomoteurs, c'est à dire l'envoi de la nouvelle position à atteindre

— La gestion du contrôle du robot interactive par la librairie python `pygame` (cette gestion n'est utilisée que pour le debogage du mouvement du robot)

De plus, il faut que chaque servomoteur atteignent la position souhaitée dans le temps de la période. Pour cela, la vitesse maximale des servomoteurs est fixée à une valeur suffisamment élevée.

Le temps d'exécution d'une itération a été mesuré sur 10000 itérations avec une utilisation active de la gestion du contrôle du robot interactive par pygame comme le montre la Figure 12.

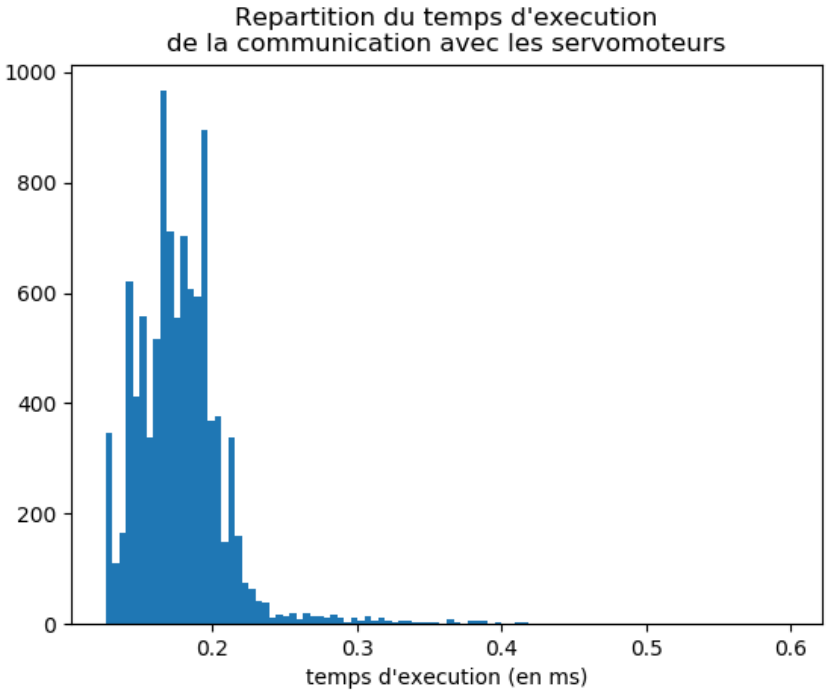


FIGURE 12 – Répartition du temps d'exécution de la communication avec les servomoteurs

Le fait de ne plus attendre la confirmation d'écriture dans les registres permet l'envoi d'instructions en très peu de temps (souvent moins de $0.3ms$ sur une période de $10ms$).

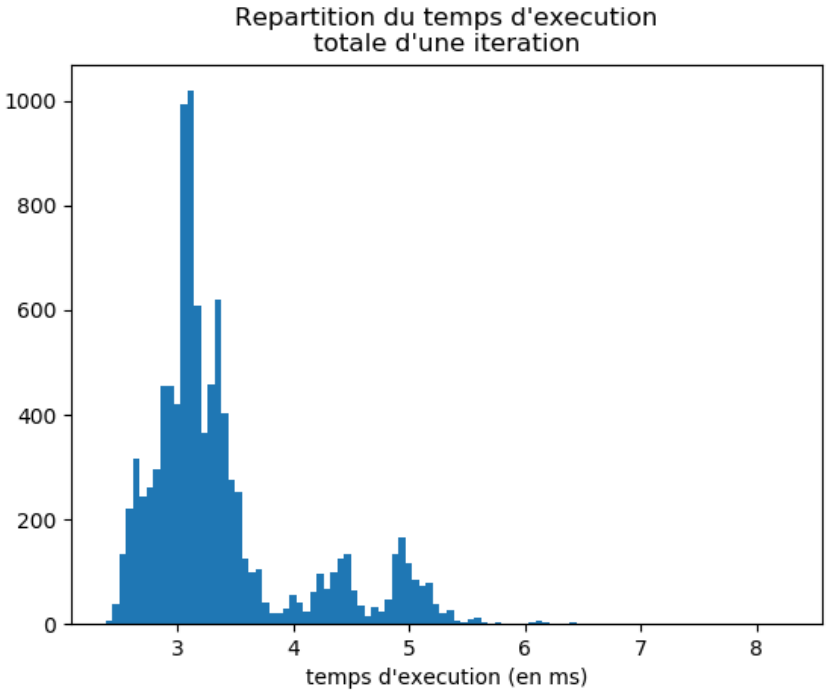


FIGURE 13 – Répartition du temps d'exécution totale d'une itération

Sur l'exécution de 10000 boucles de $10ms$, la plupart des boucles s'exécute entre $1ms$ et $5.5ms$ (cf Figure 13) et le maximum de temps d'exécution se situe vers $8ms$. Une itération de boucle a donc normalement bien le temps de l'exécuter dans le temps imposé par la période. Si jamais une itération s'avérait prendre plus de temps qu'une période, le code est fait de façon à ce que le retard soit rattrapé à la prochaine itération. Il faut juste s'assurer que ce cas est rarissime. C'est pourquoi ces tests n'ont pas besoin d'être plus poussés et qu'il ne faut pas s'alarmer de la proximité entre le temps d'exécution maximal de $8ms$ et la période de $10ms$.

III.6 Conclusion

La prise en main des AX-12 est plutôt agréable, en partie grâce à une documentation très fournie. La difficulté dans l'utilisation de ces servomoteurs a finalement été aussi leur point fort. Le fait qu'ils soient numériques et que le mouvement du serpent soit sinusoïdale nécessitaient d'échantillonner le mouvement des servomoteurs. Finalement deux solutions ont été trouvées : échantillonner en vitesse ou échantillonner en position en désactivant le paquet de retour. L'échantillonnage en position a été retenu pour le déplacement du robot. Maintenant que le robot est pilotable de manière précise et fluide, il est temps de passer à la localisation du robot dans l'espace.

IV Traitement d'images

IV.1 Introduction

Le traitement d'images constitue un point incontournable du projet. En effet, afin pouvoir utiliser un algorithme génétique, il est nécessaire d'être capable d'évaluer chaque individu. Il a été décidé que le moyen le plus simple d'évaluer la performance de ce dernier compte tenu du temps imparti serait d'utiliser une **caméra au dessus du serpent** permettant de détecter sa position initiale puis de comparer le trajet effectivement réalisé au trajet *optimal*.

IV.2 Choix de l'appareil photo

Comme vu précédemment, un support a été construit afin de s'assurer de la stabilité de la position de l'appareil photo pendant l'entièreté du traitement. Afin d'avoir une zone la plus grande possible, l'appareil doit se trouver le plus haut possible mais aussi assez loin de l'ordinateur de calcul. Afin d'éviter un surplus de fils, c'est un téléphone qui fait office d'appareil photo. Ce téléphone (OnePlus 6T) communique en Wi-Fi avec l'ordinateur de calcul. C'est l'application *DroidCam Wireless Webcam* qui permet de faire ce lien. L'application permet d'obtenir l'image issue de la caméra du téléphone en temps réel tout en conservant une qualité correcte à une vitesse d'enregistrement minimale (IPS). L'avantage de cette solution est également de faciliter la mise en parallèle de plusieurs caméras si le besoin s'en fait (ne nécessite que le téléchargement de l'application DroidCam et un réseau Wi-Fi pouvant héberger les connexions).

IV.3 Réalisation

IV.3.1 Pygame

Les servomoteurs étant pilotés grâce à un script Python et en quête d'une homogénéité du développement, le traitement d'image devait également être fait sous Python. La bibliothèque la plus classiquement utilisée sous Python pour le traitement d'images est *OpenCV*. Néanmoins l'un des membres de l'équipe ayant travaillé avec la bibliothèque **Pygame**, c'est cette dernière qui à été choisie.

Elle permet de réaliser des opérations sur les images en temps réel qui était l'une des contraintes fixée par le groupe au début du projet.

IV.3.2 Détection du serpent

Deux solutions ont été envisagées pour la détection du serpent :

— Détecter tous les servomoteurs à l'aide d'une méthode morpho-mathématique

— Détecter uniquement sa tête et sa queue identifiées comme telles afin de déterminer la droite suivie par le serpent ainsi que son orientation

C'est la seconde méthode qui a été adoptée car elle nécessite moins de calculs et les méthodes morpho-mathématiques n'étaient pas un sujet maîtrisé par l'équipe au moment de la décision.

Afin d'identifier la queue et la tête de manières distinctes, une couleur est appliquée sur chacun de ces éléments : **rouge** pour l'un, **vert** pour l'autre. Ces couleurs n'ont pas été choisies au hasard : ce sont des couleurs ayant très peu de chances d'être détectée sur le sol gris de la salle de travail.

Grâce aux fonctions de Pygame, il est alors possible de récupérer un masque de pixels correspondant à la couleur désirée à une sensibilité près. Il a cependant été nécessaire d'étalonner les couleurs exactes des extrémités du serpent car leur perception par la caméra est fortement sensible à la luminosité ambiante.

En effet, la méthode `pygame.mask.from_threshold(display_window, color, sensibility)` renvoie un masque de pixels correspondant à la couleur `color` à la sensibilité `sensibility` près. Il est alors possible de récupérer le centre de masse des masques de pixels rouges et verts grâce à la méthode `centroid()` des objets générés précédemment.

IV.3.3 Prévision de la trajectoire

Une fois le traitement ci-dessus réalisé, il est possible de prévoir le mouvement du serpent pendant les t prochaines secondes. En effet, après 20 essais, l'équipe à remarqué que le serpent se déplaçait selon la droite définie par la position initiale du serpent. Ainsi, la prévision de la position finale de la tête du serpent est estimée à une distance fixée (dépendant du temps de parcours du serpent) selon la droite Queue → Tête.

La Figure 14 résume alors les informations extraites de la photo du serpent.

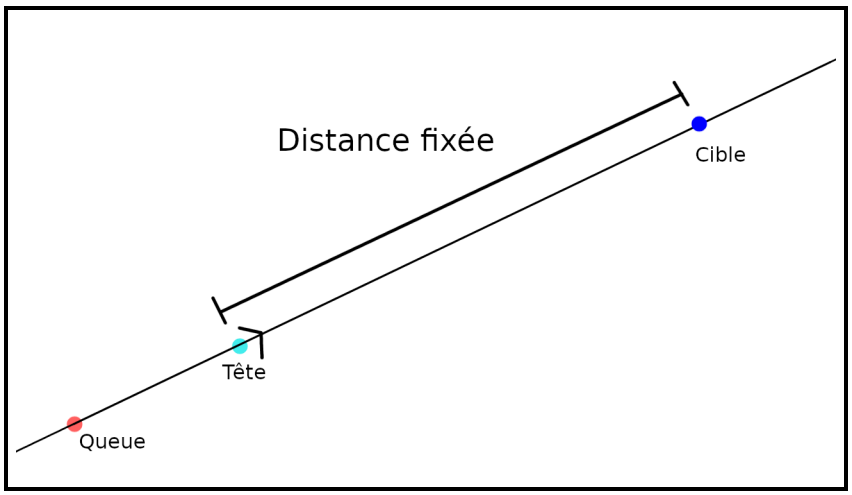


FIGURE 14 – Principe du traitement d'image

IV.4 Conclusion

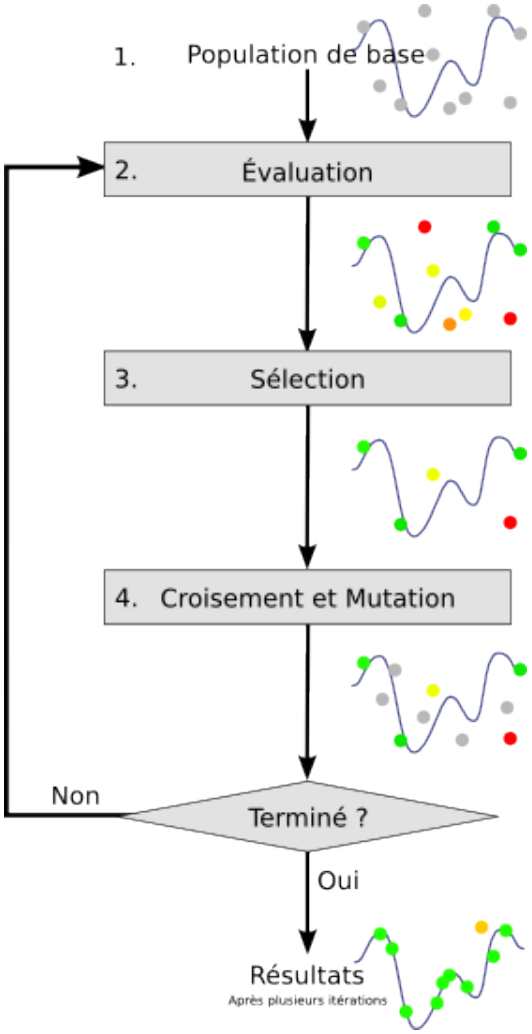
Maintenant que l'on est capable de détecter correctement la direction vers laquelle se déplace le serpent mais aussi de définir une position cible à atteindre, toutes les conditions sont réunis pour passer à l'entraînement par algorithme génétique.

V Algorithme génétique

V.1 Introduction

Un algorithme génétique est un algorithme se basant sur les travaux de Charles Darwin en matière de théorie de l'évolution : des individus concourent pour être le meilleur dans un domaine. Seuls les meilleurs survivent et se reproduisent entre eux transmettant ainsi leurs gènes à leur descendance. Dans le cadre d'un apprentissage évolutif ou bien de l'adaptation à un problème donné, l'algorithmie génétique est la méthode de machine learning la plus simple à mettre en place et la plus intuitive.

FIGURE 15 – Schéma de principe d'un algorithme génétique (Ref [6])



Un algorithme génétique fonctionne alors selon le principe suivant :

1. Génération d'une population aléatoire d'individus
2. Évaluation de chaque individu selon un critère de performance établi
3. Sélection des meilleurs individus et de quelques individus *chanceux*
4. Croisement de ces derniers entre eux et mutation d'une partie des gènes de la descendance ainsi créée
5. Répétition du processus pour N générations ou jusqu'à convergence

Il est important d'introduire les processus aléatoires que sont la sélection d'individus n'étant pas les plus performants de leur génération et la mutation des individus hybridés afin éviter de converger vers un minimum local de la fonction que l'on cherche à minimiser.

Dans le cadre du projet, le robot doit être capable d'adapter son mouvement après la détection d'une panne de l'un des servomoteurs le constituant. L'objectif final est que le serpent soit capable d'adapter son mouvement peu importe le servomoteur en panne, de manière autonome et en conservant une bonne efficacité. Mais il faut dans un premier temps travailler sur un seul servomoteur fixé afin de prendre en main le problème et pouvoir généraliser la correction par la suite.

V.2 Choix de conception

Les individus de chaque génération de serpent vont concourir pour être le plus performant possible, mais selon quel(s) critère(s) ?

V.2.1 Fonction d'évaluation d'un individu

Le premier critère envisagé fut la **distance à la cible** : plus la tête du serpent est loin de la cible, plus son score est élevé. Les meilleurs individus sont alors ceux qui parviennent à obtenir le score le plus faible. Ce critère permet d'évaluer les individus en terme de précision et de les comparer à un individu sain.

Néanmoins, ce n'est pas ce dernier qui à été retenu. En effet l'équipe à remarqué que le principal défaut induit par le blocage de l'un des servomoteurs était la dérive du serpent vers la gauche ou la droite en fonction de la position du servomoteur bloqué. Le Il n'était alors plus question d'arriver au plus proche de la cible mais de se déplacer en ligne droite. C'est pourquoi il à été retenu une fonction d'évaluation **ellipsoïdale** centrée au niveau de la cible usuellement atteinte par un serpent sain et dont l'axe principal est orienté selon la position initiale du serpent. La fonction de score, représentée Figure 16 est alors la suivante :

$$\text{score} = \sqrt{((P_0 - T_0)\cdot C + (P_1 - T_1)\cdot S)^2 + \alpha\cdot((P_0 - T_0)\cdot S - (P_1 - T_1)\cdot C)^2)}$$

P_0, P_1 : l'abscisse, l'ordonnée sur l'image de la tête du serpent

T_0, T_1 : l'abscisse, l'ordonnée sur l'image de la target

$C, S = cos(\theta), sin(\theta)$

α = coefficient de compression de l'ellipse

θ = angle entre l'abscisse de l'image et l'axe de déplacement du serpent

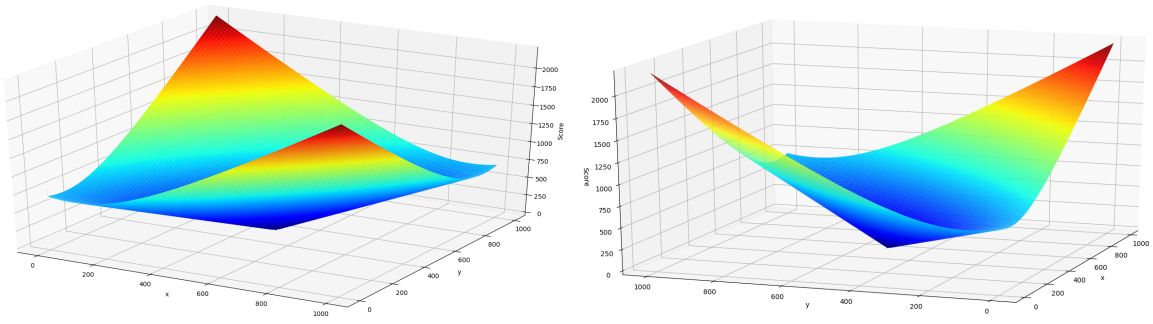


FIGURE 16 – Visualisation de la fonction de score ellipsoïdale pour un serpent orienté selon la première bissectrice

L'ellipse retenue est centrée sur le point d'arrivée estimée, et accorde 10 fois plus d'importance (paramètre α) à la présence du serpent sur l'axe de départ que sur sa normale. Les ellipses créées sont donc un équivalent *applati* des cercles dont le diamètre correspond alors au grand axe de l'ellipse associée.

Le choix de cette fonction aurait également pu prendre en compte l'angle entre l'axe initial du robot et l'axe final du robot. En effet, plus l'angle est réduit, plus le serpent est aligné avec sa direction d'origine. Une multiplication par la valeur absolue de cet angle (à un facteur près) ou le sinus de cet angle. La fonction d'évaluation étant un paramètre essentiel au choix des différents individus, celle choisie semble cependant efficace pour une première approche.

V.2.2 Paramètres de l'algorithme génétique

Certains paramètres propres à l'algorithme génétique ont été imposés par manque de temps. Ainsi le nombre d'individus par génération a été fixé à 12. Chaque génération est créée à l'aide des 5 meilleurs individus ainsi que d'un individu chanceux. Chaque couple engendre alors 4 enfants dont les caractéristiques sont une moyenne pondérée aléatoirement (grâce à la méthode `random.random()`) entre les deux parents. Chaque enfant a en plus une chance de muter de 15%. Si un enfant mute, on modifie alors ses deux caractéristiques aléatoirement autour d'une gaussienne centrée sur la valeur courante et avec une variance arbitrairement fixée à 5. Il aurait pu être intéressant de pondérer les paramètres des enfants en fonction du score des parents mais les solutions finiront tout de même par converger. Ce sont tous ces points particuliers concernant certains paramètres internes à l'algorithme génétique qui n'ont pu être traité.

Après plusieurs tests "*à la main*" afin de déterminer quels paramètres devaient être modifiés afin de corriger la trajectoire du serpent, le choix s'est arrêté sur une modification de l'*offset* global des servo-moteurs et de l'amplitude du serpent. Même si ces deux paramètres ne permettent pas à eux deux de corriger de manière totale le défaut du serpent, ils sont suffisants afin de mettre en place un apprentissage rapide et assez efficace. L'idéal aurait été de travailler sur l'offset de chaque servomoteur, ainsi que sur la variable *tick* permettant "d'améliorer" la vitesse du serpent en dépit de la fluidité de son mouvement.

V.3 Conclusion

L'algorithme génétique est un moyen efficace pour trouver une solution optimisée si l'on dispose de suffisamment de temps. Outre le temps nécessaire à l'établissement de la solution optimale, dépendant intrinsèquement du nombre de paramètres modifiés et de la plage de changement de chacun de ces paramètres, l'entièreté de l'entraînement devait être fait sur le robot. Du fait des calculs de frottements trop complexes à simuler, l'entraînement ne pouvait pas être effectué en simulation. L'évaluation de chaque individu fut alors longue et fastidieuse, ce qui força le groupe à réduire le nombre de paramètres afin de simplifier le problème. L'algorithme génétique, au coeur du sujet initial, s'est donc retrouvé grandement simplifié et son étude n'a pu être que sommaire.

VI Conclusion

VI.1 Résultats du projet

Le cahier des charges construit au début du projet était ambitieux. La base du robot est cependant fonctionnelle et l'entièreté du code a été réalisée afin que chaque partie soit modifiable aisément. Ainsi il est possible d'améliorer le traitement d'image indépendamment des autres parties du code et il en est de même pour chaque paramètre modifiable du code. Au fur et à mesure de l'avancée du projet, c'est avant tout une base solide et des premières pistes de réalisation du projet qui ont été explorées. Au final, l'algorithme génétique n'a pas pu être correctement optimisé et la correction du serpent n'a pas pu être complètement menée à bien. Le résultat est tout de même convaincant. Dans le cadre énoncé précédemment, les individus convergent vers un serpent "type" dont les paramètres d'amplitude et d'offset (valant environ 497 et 280) permettent d'obtenir un score quasi constant légèrement inférieur à 300. À titre informatif, ces individus suivent quasi-systématiquement l'axe principal, seule la distance parcourue est un problème. Les résultats, quoique non optimisés, sont donc convaincants pour une première approche. Le serpent arrive à corriger sa trajectoire pour un servomoteur cassé fixé.

VI.2 Pistes d'améliorations

Ce projet ayant été repris de zéro, c'est avant tout une base de travail qui a été mise en place afin de pouvoir par le futur, travailler de manière plus approfondie sur la partie algorithmie génétique et les résultats à en tirer. Les pistes d'améliorations du projet sont donc les suivantes :

- Le système actuel est totalement filaire. L'alimentation se fait à l'aide d'une alimentation stabilisée et les calculs sont entièrement effectués sur un ordinateur. Une amélioration pourrait être d'alimenter le système à l'aide d'une ou de plusieurs batteries et d'embarquer l'unité calculatoire sur une Raspberry (par exemple).
- Afin d'étendre la zone de déplacement du robot, il est envisageable de mettre en parallèle plusieurs caméras. Cela nécessite alors soit la création d'une image rassemblant les différentes images issues des caméras, soit une triangularisation de la position du serpent, connaissant la position des caméras.
- Actuellement, le traitement d'image n'est pas exécuté en temps réel. Un traitement en temps réel donnerait lieu à une détection instantanée de la sortie de l'image du robot. Un autre avantage serait de pouvoir mettre en place un asservissement du robot vers la position donnée. Le coût de ce traitement en temps réel est l'obligation de parallélisation des calculs, le traitement d'image ayant lieu en même temps que la commande du robot.
- L'apprentissage du robot est manuel. Automatiser le processus d'apprentissage relèverait d'un gain de temps faramineux. L'inconvénient de l'apprentissage automatique est notamment qu'il nécessite une plus grande zone de déplacement (afin de pouvoir manoeuvrer et se remettre en position), un traitement de l'image en continu (afin d'éviter toute sortie du robot de l'aire de déplacement) et le codage de fonctions de réinitialisation du robot. De plus il est nécessaire d'être capable de détecter toute anomalie pouvant survenir (câbles emmêlés, robot renversé...). Ce gain de temps serait néanmoins un atout majeur pour le projet (entraînement possible en l'absence des membres).
- Le temps de mise en place de tout le projet a été plus long que prévu. Ainsi l'algorithme génétique n'a pas été optimisé. Observer l'influence des différents paramètres de l'algorithme, la vitesse de convergence associée, la multiplication des paramètres modifiés, l'influence de la fonction d'évaluation, etc... est l'axe principal vers lequel le projet doit s'orienter. L'analyse effectuée n'est pas suffisante et des résultats plus probants et plus rapides sont possibles.

VI.3 Ressenti du projet

Dans sa globalité, le projet fut instructif et épanouissant. La mise en place de certains points du cours sur un projet réel fut très satisfaisant. Le point noir du projet subsiste dans la perte de temps associée à la prise en main des différents éléments, perte de temps ayant entraîné un retard dans le planning et donc moins de temps passé sur ce qui semblait être le coeur du projet : l'algorithme génétique. Un travail plus approfondi sur ce dernier aurait permis une bonne introduction au monde du machine learning mais aurait également été plus valorisant. Le projet est complexe et de nombreuses améliorations sont possibles, ces dernières apportant une réelle valeur ajoutée au projet. C'est dans l'espoir de voir ce projet repris et transcendé que l'équipe laisse aux années futures le travail qu'elle a fourni.

VII Annexes

A Distances parcourue en 1200 ticks

875	929	890	917	876	909	870	909	851	871
859	859	919	863	863	849	861	837	852	855

Moyenne	875.7	Médiane	866.5

TABLE 1 – Distances parcourue en 1200 ticks

B Diagramme de Gantt prévisionnel

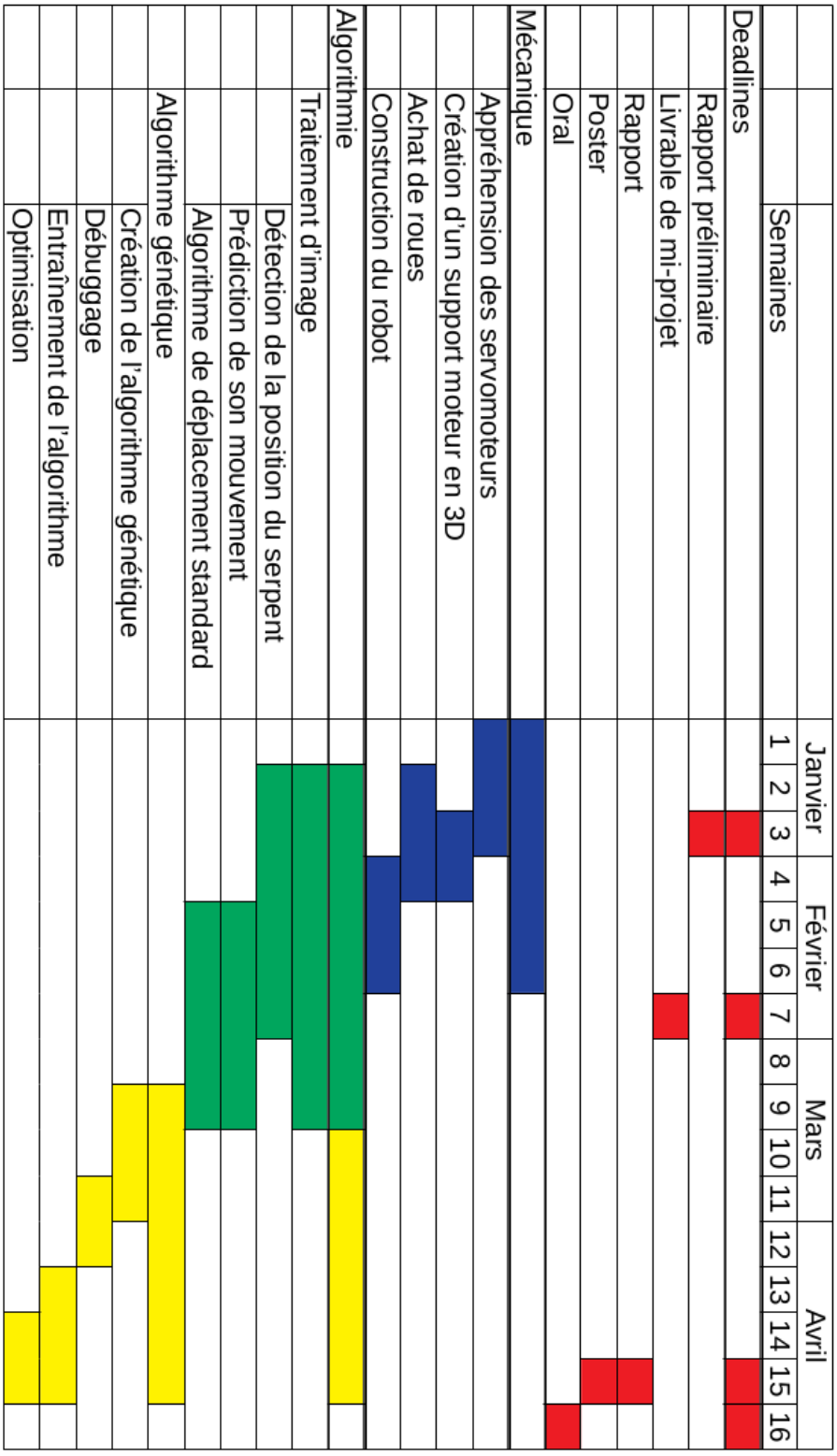


Diagramme de Gantt prévisionnel

Bibliographie

[1] A fork of the python client for dynamixel servos based on patrick goebel's work. URL https://github.com/ShiversVert/Snake_Project/tree/master/servo_controlling/libs/pydynamixel.

[2] ROBOTIS. Robotis e-manual ax-12. URL http://support.robotis.com/en/product/actuator/dynamixel/ax_series/dxl_ax_actuator.htm.

[3] Wheels website. URL https://www.brickowl.fr/catalog/lego-brick-2-x-4-wheels-holder-with-red-

[4] Patrick Goebel. A fork of the python client for dynamixel servos. URL https://github.com/iandanforth/pydynamixel.

[5] A simulation of the snake on python. URL https://github.com/ShiversVert/Snake_Project/blob/master/servo_controlling/simu_snake_mvt.py.

[6] Reference to the genetical algorithm image. URL https://upload.wikimedia.org/wikipedia/commons/4/42/Schema_simple_algorithme_genetique.png.