PROJET LAPIN AUTOMATE

Rapport

ONIRIS
Référente : Julie HERVE

IMT ATLANTIQUE
Tuteurs : Cédric DUMAS et Charles PRUD'HOMME

Mayeul DE BUTLER – Gildas PHILIPPE – Julien DALLE - Molly CARTER
[Adresse de courrier]

Table des matières

I - Etat initial

1. Prise en main

Maquette

Lors de la soutenance des étudiants précédents, une démonstration du lapin a eu lieu. Fonctionnel, les solutions techniques du cœur et de l'urée étaient correctes.

Toutefois, la solution retenue pour les poumons ne convenait pas. En effet, le moteur bielle-manivelle donne un mouvement vertical permettant aux coques imprimées en 3D fixées en bout de se déplacer suivant le même mouvement. L'effet gonflement des poumons dans les 3 directions de l'espace n'est donc pas réalisé. Dès lors, on n'observe pas d'effet d'amplitude en volume.

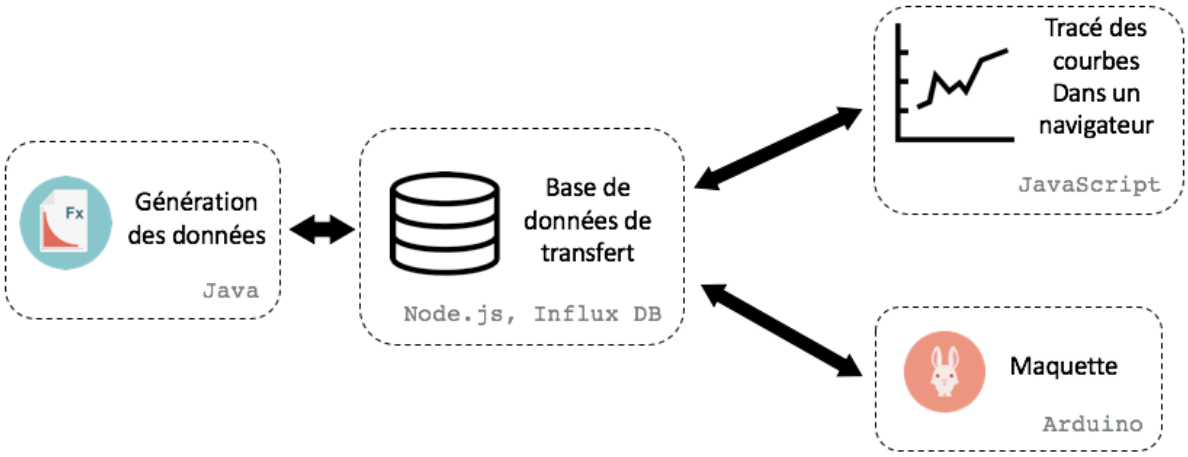
Enfin, ce système engendre beaucoup de bruits causant alors une distance plus importante avec la réalité.

Code Arduino

Par la suite, nous avons tenté de faire fonctionner les codes Arduino sur nos propres ordinateurs. Pour cela nous avons dû suivre un certain protocole : entrée de la bonne adresse IP, configuration de l'Ethernet. A l'issu de cette procédure, nous avons pu activer le lapin mais nous avons alors noté l'impossibilité de lancer automatiquement le code.

Architecture logicielle

Outre la maquette du lapin, un logiciel développé par un groupe de GIPAD et GSI permettait de générer les données et valeurs transmises à la maquette ainsi qu'afficher en temps réel des courbes afin de visualiser les paramètres.
Lors de la prise en main du projet, nous avons qu'un seul paramètre étudié : la pression artérielle.
Puis, nous avons surtout noté une certaine architecture logicielle assez complexe, avec de nombreuses dépendances, langages et bibliothèques. Le Docker permettant de rendre ce dernier aspect transparent n'étant pas encore au point, cela rendait difficile le lancement du logiciel. De plus l'addition de plusieurs éléments consommait beaucoup de ressources.
L'ensemble génération de données, affichage des courbes et communication avec la maquette se présentait de cette manière :



1. Les données sont générées par un programme en Java à partir de formules et de coefficients établis préalablement sur MatLab. Ces données sont stockées via protocole HTTP sur une base de données locale.
2. Le navigateur lit les dernières valeurs de la base de données pour les afficher sur un graphique JavaScript.
3. La maquette interroge la base de données pour récupérer les dernières valeurs ajoutées et mettre à jour son état. En cas d'événement externe (piqûre), la maquette écrit l'événement en base de données pour notifier le programme Java.

Un des enjeux de notre projet aura été de simplifier cette architecture pour améliorer la performance, l'IHM et la prise en main de futurs groupes qui auront à travailler dessus.

2. Entretiens

Julie Hervé

Comme le projet est à l'initiative de l'école Vétérinaire d'Oniris, une rencontre a été organisée avec la référente Julie Hervé, pour entendre ses remarques et attentes pour cette nouvelle phase de développement.

Pour elle, il était important que la maquette soit la plus réelle possible. Très satisfaite des solutions techniques mises en place pour le cœur et l'urée, elle était néanmoins plus perplexe quant aux poumons comme nous avions pu le soulever plus tôt.

De plus, le rappel des objectifs pédagogiques de la séance de Travaux Pratiques sur le lapin montre l'importance de simuler les réactions du lapin et notamment face aux injections. Ainsi les différents états du lapin (par exemple : repos-sous adrénaline-repos) doivent être liés, sans accrocs.

Cédric Dumas et Charles Prud'homme

L'entretien que nous avons eu avec nos tuteurs ayant suivi celui de Julie Hervé, il a été une fois de plus constaté la nécessité d'améliorer la solution technique retenue pour les poumons.

Ensuite, il était crucial pour le projet de faire le lien entre les fréquences (respiratoire, cardiaque et celle de la diurèse) trouvées par la génération des données et celles utilisées par le lapin.

Enfin, un dernier objectif du groupe précédent qui était la génération des données des autres paramètres étudiés, a été mis en attente. Effectivement, nos tuteurs nous ont mis en garde quant au délai du projet et ont mis en avant la difficulté qu'ont eu les étudiants précédents à aboutir à un premier modèle.

II - Cahier des charges - Objectifs

1. Problématique

Une fois la prise en main faite et les différents interviews effectués, les questions suivantes ont été soulevées :

- Comment rendre les poumons plus réels ?
- L'architecture logicielle doit-elle être conservée ?
- Comment permettre l'échange des données en respectant les temps de délai ?
- Comment gérer les différents états du lapin ?

2. Cahier des charges

Le cahier des charges a été défini en fonction des problèmes auxquels il fallait répondre en priorité. Les objectifs ont alors été les suivants :

- Au niveau de la maquette :
  1. Modélisation plus fidèle des poumons
  2. Détection physique d'injection de produits

- Au niveau de la gestion des données
  1. Repenser l'architecture logicielle
  2. Suivi en temps réel des constantes vitales sur ordinateur avec évolution de celles-ci lors d'injection de produits
  3. Rendre la communication des données effective avec synchronisation en temps réel

III - Etat final

1. Maquette

D'un point de vue anatomie, le lapin possède 12 côtes (et 12 vertèbres) dont les 9 premières sont reliées au sternum pour former la cage thoracique. Le tiers restant représente les côtes flottantes.

À la suite de l'entretien effectué avec Johann Hérault nous avons axé notre réflexion en se basant sur l'étude des poumons humains.

La respiration humaine se fait grâce aux poumons, côtes, muscles intercostaux et le diaphragme. Ce dernier muscle est responsable à 75% de la respiration quand les muscles intercostaux permettent la modulation en amplitude.

Structure

Modélisation 3D de la cage thoracique (sous fusion 360)

1. Forme générale

Nous avons décidé de modéliser une cage thoracique de lapin qui s'ouvre et se ferme pour imiter une inspiration et une expiration. L'ouverture et la fermeture seront assurées par la compression de mousse qui va alors pousser les parois extérieures de la cage thoracique.

La forme générale retenue est un cylindre coupé en quartiers comme sur l'image n°1 :

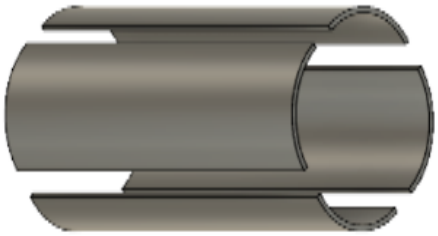


Image n°1

2. Charnière

Afin que celui-ci puisse s'ouvrir nous avons imaginé des charnières pour assurer la rotation des quartiers de cylindre les uns par rapport aux autres et ainsi l'ouverture de la cage thoracique.

Voici le modèle de charnière finalement modélisée :

A noter que celle-ci peut se bloquer avec une goupille. De plus la conception permet de limiter l'amplitude de rotation du gond à 30 ° car une ouverture supérieure à 30° conduirait à une configuration peu vraisemblable pour la cage thoracique. La valeur de 30 ° est arbitraire, elle mérite d'être affinée.

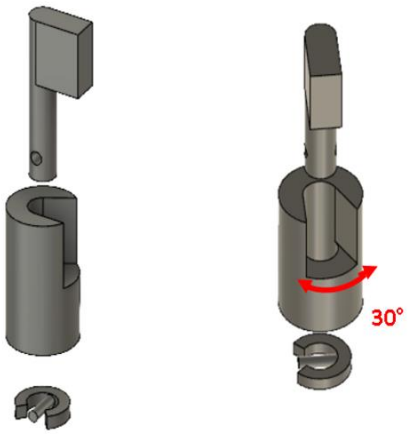


Image n°2

On obtient les “côtes” de notre lapin une fois les quarts de cylindre et les charnières assemblés

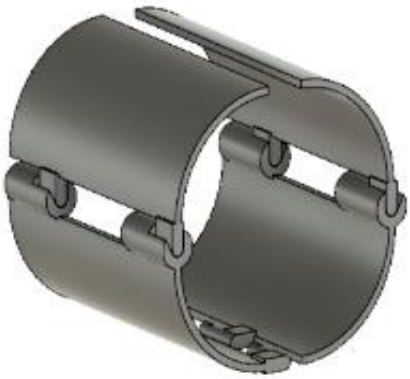


Image n°3

3. Base et piston

L’étape suivante a été de modéliser le piston et la base qui elle va accueillir la cage thoracique (avec les charnières du bas) et le servomoteur dans la zone entourée.

Sur cette base on retrouve également un piston entraîné par un système bielle manivelle directement accroché au servomoteur. Avec cette configuration on va pouvoir transformer un mouvement de rotation en un mouvement de translation afin de comprimer la mousse située dans la cage thoracique et ainsi recréer une pseudo respiration.

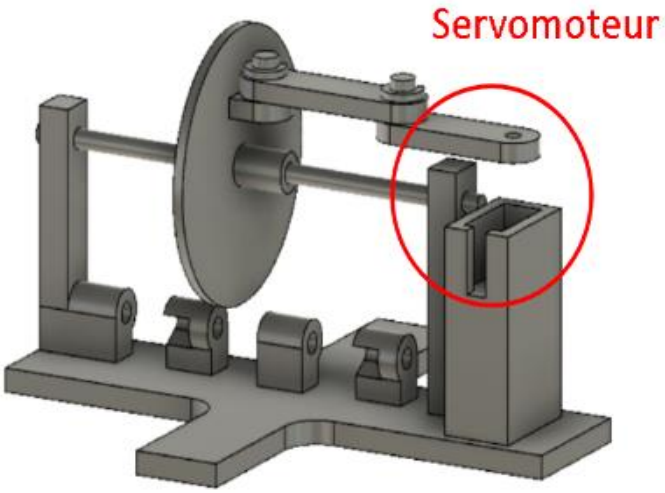
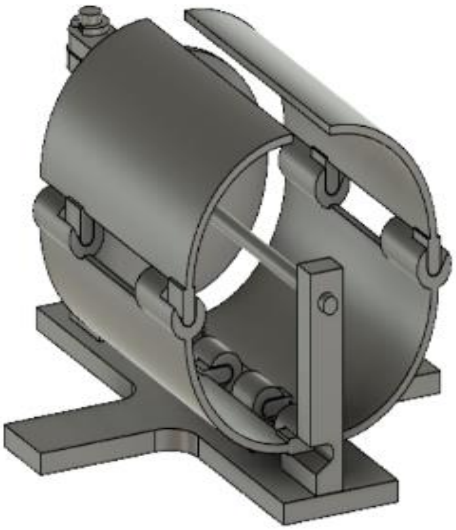


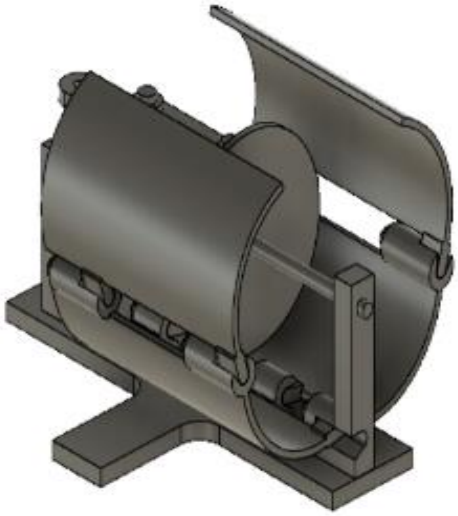
Image n°4

4. Cage thoracique complète

Après avoir assemblé la base et les “côtes” on obtient la cage thoracique complète. Comme on peut le voir ci-dessous, en mettant de la mousse dans la cage thoracique devant le piston, celle-ci va s’ouvrir car la mousse va être comprimée lorsque le piston va être poussé en avant par le servomoteur. On recrée alors un mouvement proche de celui d’une respiration.



Piston reculé



Piston poussé

5. Impression 3D

A la suite de la modélisation 3D de la cage thoracique complète du lapin il a fallu imprimer chacune des pièces en 3D. Les pièces de couleur bleu sont en ABS et ont été imprimées sur l'imprimante de IMT Atlantique : Makerbot Replicator 2x. En ce qui concerne les pièces violettes, elles sont en PLA et ont été imprimées sur une Prusa i3 MK3. Certaines pièces ont eu des difficultés à s'imprimer (notamment la base) compte tenu de leur géométrie. Sur l'image n°6 ci-dessous, les couleurs ne se perçoivent pas très bien.

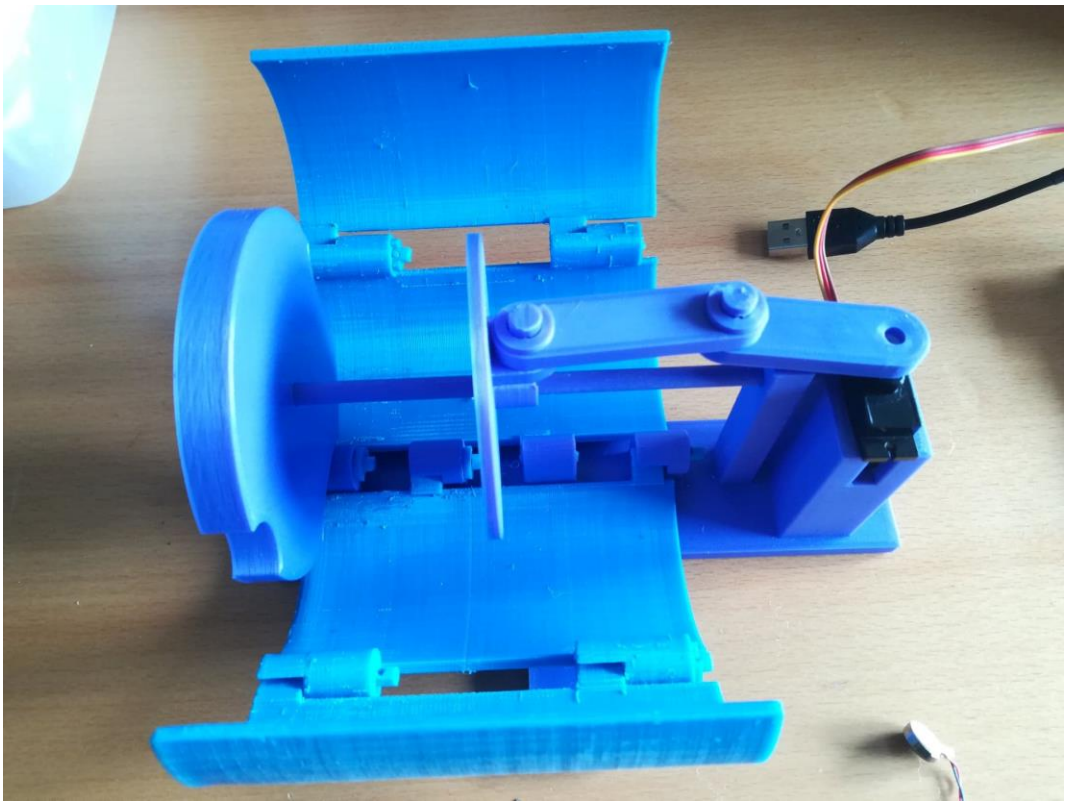
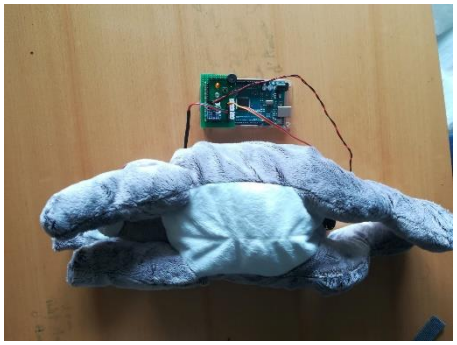


Image n°6

Câblage électronique

Pour une solution compacte nous avons décidé de réaliser un shield à brancher directement sur une carte Arduino Méga.



Voici le tableau récapitulatif de toutes les connexions à l'Arduino Mega 2560 :

	Patte ou fil	Arduino Mega 2560
Led jaune	anode (+)	pin 46
	cathode (-)	Ground
Led rouge	anode (+)	pin 30
	cathode (-)	Ground
Buzzer	patte 1 (indifférent)	pin 50
	patte 2 (indifférent)	Ground
Bouton	patte 1 (indifférent)	pine 40
	patte 2 (indifférent)	Ground
DRV2605L	Vin	5V
	GND	Ground
	SCL	SCL
	SDA	SDA
	IN	Non connecté

Servomoteur poumons	fil marron	Ground
	fil rouge	+5V (autre source d'alimentation que l'Arduino)
	fil jaune	pin 25

Voici la liste des connexions au DRV2605L :

	Patte ou fil	DRV2605L
Moteur haptique	fil rouge	sortie +
	fil noir	sortie -

Malheureusement lors du soudage du DRV2506L sur le shield, celui-ci a pris un coup de chaud, il est donc à changer (ceci est l'analyse la plus probable du dysfonctionnement du contrôleur).

En ce qui concerne le 5V in sur le shield il s'agit en fait d'une entrée 5V (à utiliser avec le câble dupont vers usb) pour alimenter le moteur car l'Arduino ne permet pas de l'alimenter directement sans compromettre le fonctionnement des autres éléments.

2. Arduino

Communication entre PC et Arduino

La communication ne se fait plus via un serveur virtuel mais via une liaison série avec l'ordinateur. Nous avons décidé de modifier le protocole de communication afin de condenser l'architecture logicielle et n'avoir plus que python à lancer sur l'ordinateur.

La communication entre le PC et Arduino se fait d'une manière très simple, découpée en trois phases :

1. Synchronisation des données

Nous utilisons le principe d'accusé de réception. L'ordinateur envoie un booléen à l'Arduino pour lui indiquer qu'il est prêt à envoyer les données. Si le byte lu par Arduino n'est pas ce booléen, Arduino lui répond qu'il n'est pas prêt. Ce schéma boucle jusqu'à ce qu'Arduino reçoive bien le booléen du PC et lui réponde qu'il est lui aussi prêt. L'envoie des données peut alors se faire.

2. Envoi des données

Les données envoyées sont les suivantes :

- l'état du lapin : Il correspond à la dernière substance non assimilée injectée. Il est modélisé par un entier : 0 pour le repos, 1 pour l'adrénaline, 2 pour l'acétylcholine, 3 pour l'ADH et 4 pour l'Angiotensine II

- la période de l'urée. Elle est donnée en déci secondes, codée entre 0 et 255. Ce qui nous laisse une plage de fréquence de 10Hz à 1/25 Hz. Cette valeur est ensuite multipliée par 100 afin d'avoir une période en milliseconde, exploitable par Arduino.

- la période respiratoire. Elle aussi donnée en déci secondes et traitée de la même manière.

- la période cardiaque. Elle aussi donnée en déci secondes et traitée de la même manière.

3. Envoi de la substance captée par le lapin

A chaque envoi des constantes métaboliques au lapin, celui-ci répond par la substance captée. Cela permet à l'ordinateur de réagir à une éventuelle injection. La substance est elle aussi codée sous la forme d'un entier, sur le même principe que l'état : 0 si aucune substance n'est détectée, 1 pour l'adrénaline, 2 pour l'acétylcholine, 3 pour l'ADH et 4 pour l'Angiotensine II. A l'heure actuelle la solution de détection n'est pas mise en place. Nous avons simplement mis un bouton poussoir pour signifier une injection d'adrénaline uniquement. A l'avenir, ce bouton est voué à être remplacé par un capteur de couleur qui captera la couleur d'une seringue afin de déterminer la substance injectée, c'est pour cela que nous avons adopté une architecture qui permet l'ajout de nouvelles substances (jusqu'à 255).
A l'heure actuelle, cette séquence de réception de données est réalisée à chaque tour de boucle de l'Arduino et est bloquante du fait de la phase de synchronisation. A l'avenir, cette séquence n'aura lieu que si le PC cherche effectivement à communiquer avec l'Arduino.

Ce protocole de communication est robuste et permet d'éviter les erreurs dans l'envoi des données.

Mise à jour des paramètres vitaux

Le contrôle des périodes des différents paramètres vitaux se fait grâce à une lecture du temps, via la commande *millis()*. On mesure la durée écoulée depuis la dernière mise à jour et si cette durée est supérieure à la période, on modifie le fonctionnement de l'actionneur, sinon on le laisse finir son mouvement.

Respiration

Nous contrôlons la respiration via la vitesse de rotation du moteur. Idéalement il faudrait un moteur à courant continu, contrôlé par une PWM. Nous étions partis sur un contrôle en position grâce à un moteur pas à pas. La solution décrite ici est donc un contrôle en vitesse de rotation d'un moteur pas à pas via un contrôle en position.

La position du moteur à un instant t est donnée par la formule :

$$\theta(t) = \theta(t_0) + t * sens$$

On contrôle simplement que $\theta(t)$ reste dans la plage admissible ([0°,180 °]). Lorsque l'on arrive à l'une des extrémités, on met à jour t_0, $\theta(t_0)$ et on inverse le sens de rotation.

Débit d'urée

Le débit d'urée est modélisé par un bip sonore, émis à la même fréquence que les gouttes d'urée. Chaque bip a une durée définie et l'espace entre deux bips est contrôlé en mesurant le temps qui s'est écoulé depuis le dernier bip et en le comparant à la période désirée.

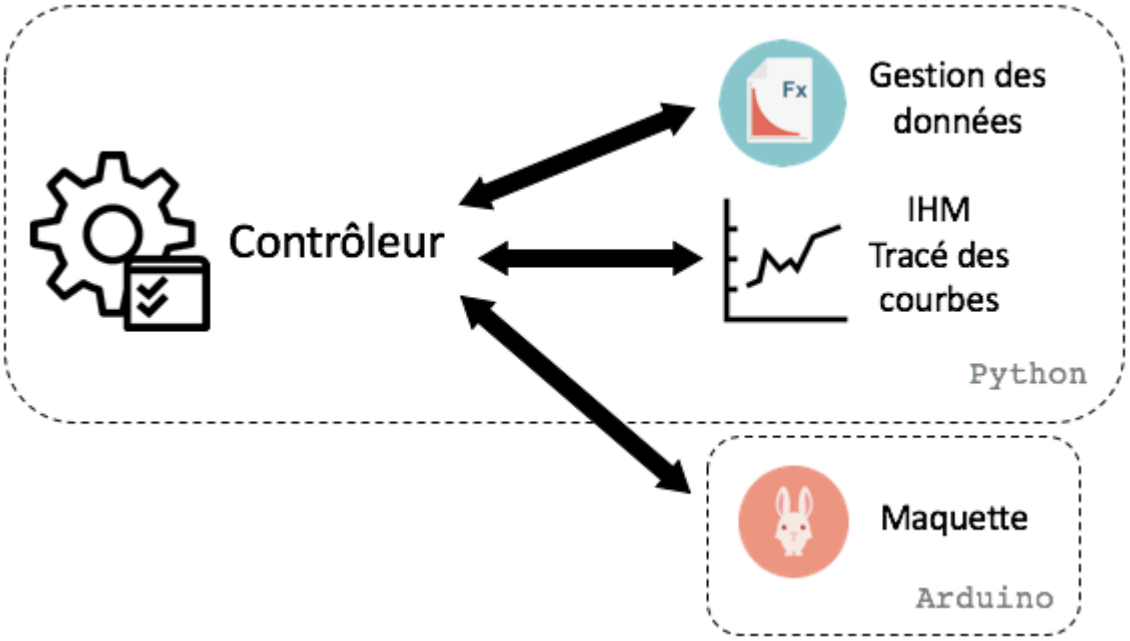
Fréquence cardiaque

La fréquence cardiaque est contrôlée via deux paramètres, la durée du battement et la durée du repos. Ce sont ces deux durées qui sont contrôlées et comparées au temps qui s'écoule.

3. Architecture logicielle

Face aux défauts de l'architecture logicielle existante, et aux difficultés que nous avons eu à faire fonctionner tous les programmes ensemble, nous avons décidé de repartir sur un nouveau logiciel en réalisant un client lourd avec qui manipulerait les données, afficherait les courbes dans une IHM, et transmettrait les données à l'Arduino directement par USB.

Voici un schéma de fonctionnement du logiciel réalisé :



Le contrôleur

Le contrôleur va servir de cerveau du logiciel. C'est lui qui interroge chaque module du code, et leur donne les instructions à réaliser. Il collecte les informations, se fait notifier des changements, vérifie l'état du robot, etc.

La maquette

La communication entre Python et Arduino est établie par la bibliothèque *pySerial* de Python (les détails techniques du mode opératoire sont explicités plus haut).

D'un point de vu macroscopique, le contrôleur va envoyer des paramètres à la maquette (fréquence cardiaque, fréquence respiratoire, etc…) pour que celle-ci puisse évoluer en fonction du temps. Si un événement extérieur (piqûre, simulée par un bouton poussoir) intervient, Arduino notifie le contrôleur.

La gestion des données

Le groupe précédent avait entamé une analyse de données et produit un modèle s'approchant de la réalité, mais seulement pour la pression artérielle moyenne, seul paramètre qui n'est pas nécessaire pour faire fonctionner la maquette.

Nos ressources humaines et temporelles étant limitées, nous avons décidé de laisser de côté pour cette phase du projet la modélisation des données. A la place, nous lisons des données mesurées sur des vrais lapins (données qui ont permis l'analyse). Ces fichiers sont classés par état du lapin (injection d'adrénaline, d'acétylcholine, repos...) et sont tirés au hasard au sein de chaque catégorie et s'enchaînent s'il y a changement de l'état, notifié par le contrôleur (piqûre).

Cela implique quelques problèmes que nous devions prendre en compte.

- *Synchronisation temporelle*

Chaque itération pour afficher un point sur une courbe dure un temps non constant, et ainsi on obtient un pas de temps non constant sur le graphique. Pour que les données restent les plus pertinentes possibles, on calcule le temps de l'expérience à partir de la date de l'itération, et on en déduit ainsi le nombre de lignes, toutes séparées d'une milliseconde, à sauter et lire la ligne représentant l'instant (avec pour écart le temps de calcul, ce qui est de l'ordre de la milliseconde donc négligeable pour une observation humaine).

On effectue donc une synchronisation temporelle à chaque itération pour éviter un décalage dans le temps.

- *Liaison entre les fichiers*

Les fichiers étant tirés au sort, les fichiers tirés ne sont pas nécessairement issus du même lapin ou chronologiquement dans l'expérience. Ceci a pour effet d'avoir des discontinuités aux bords des fichiers. On a donc décidé de calculer un coefficient à partir de valeurs moyennes sur les dernières valeurs du fichier, qui sera ajouté à la courbe suivante pour éliminer ces effets de bord. La pertinence de la méthode n'est cependant pas justifiée.

- *Valeurs manquantes*

Dans certains fichiers, certaines colonnes présentaient des trous (retrait du capteur, problème d'enregistrement…). Ainsi les valeurs envoyées à l'Arduino auraient été nulles, ce qui aurait eu pour effet d'arrêter le cœur ou la respiration du robot. Pour conserver ces fichiers, il faudrait pouvoir déduire ces données à partir des pics de la courbe de la pression artérielle, qui est une valeur stable sur toutes les mesures.

Il existe une fonction extraite de la bibliothèque Python *SciPy* : *find_peaks,* qui prend en entrée une liste d'ordonnées et des paramètres de réglages, et sort la liste des index des pics de la courbe. On note ici deux difficultés.

Analyser tout le fichier en une fois est trop long (autour de 10 secondes), car si le fichier est chargé au milieu de l'expérience (passage du repos à l'adrénaline par exemple), on ne peut pas se permettre de conserver aussi longtemps des valeurs constantes, le lapin doit réagir car c'est l'intérêt du TP. De même analyser les dernières valeurs lues dans le fichier provoquerait un décalage entre la réaction du lapin et son état réel, car transmettrait à la maquette une fréquence respiratoire moyenne sur les 5 dernières secondes par exemple, ce qui ne représente pas les valeurs de l'instant (et le lapin ne respirerait pas pendant les 5 secondes de l'expérience). On a donc décidé d'appliquer le *find_peaks* sur les 5 prochaines secondes, en ayant un

deuxième curseur dans le fichier, ce qui est performant (on ne perd qu'une milliseconde pour le chargement du fichier à peine), bien que sûrement non optimale pour la pertinence de la valeur.

Les paramètres du *find_peaks* ne sont pas triviaux à régler, car il faut que l'algorithme détecte bien les vrais pics de ses maximums locaux liés au bruit. Nous avons eu quelques difficultés avec cela, et pas de solution n'a été apportée pour le moment.

Cette solution a donc été implémentée et est prête à l'emploi, à la condition de régler le *find_peaks.* Comme notre objectif premier était de réussir à faire les liens sur toute la chaîne, du logiciel à la maquette, nous avons décidé de retirer les courbes non pertinentes pour le moment, et de ne lire seulement les quelques fichiers sans défauts apparents, et de continuer. Ce choix a été judicieux, car le temps nous aurait manqué sinon, pour un point qui n'est pas essentiel pour notre objectif à nous.

IHM et affichage des courbes

Après avoir reçu les données à afficher, le contrôleur des transmets à l'IHM pour que celle-ci les affiche.

L'IHM est assez épurée. On a dans la partie supérieure de la fenêtre trois boutons :

- Play : permet de lancer l'expérience
- Stop : permet d'arrêter l'expérience
- Exporter : permet d'exporter les valeurs de la courbe dans un fichier .txt dans un format lu par *LabChart Reader*, le logiciel utilisé à l'école vétérinaire pour visualiser les courbes.

Dans la partie inférieure, on observe les courbes. Nous avons permis aux courbes de défiler plutôt que de recadrer le graphique pour avoir toute la courbe sur les yeux afin d'améliorer la visibilité, défaut de la version précédente qui avait été remonté.

IV - Retours et pistes d'amélioration

Cage thoracique

La mousse choisie pour remplir la cage thoracique ne convient pas et ne permet pas de recréer un mouvement respiratoire. En effet pour un fonctionnement optimal il faut que le matériau retenu exerce une force transversale sur les parois lorsqu'il est comprimé de manière longitudinale par le piston.

De plus il semble que le servomoteur ne permette pas d'appliquer de forces trop importantes sur la mousse avec le piston. C'est un point important à prendre en compte lors du choix du matériau adéquat à placer à l'intérieur de la cage thoracique.

Code Arduino

Lors de l'élaboration du code nous avions décidé d'actualiser les valeurs à chaque pas de temps. C'est-à-dire que cette action se déroule au même titre qu'un déplacement du moteur ou de la durée d'un son. Les valeurs de fréquence sont lues constamment.

Compte tenu de l'architecture choisie, il serait plus intéressant de dissocier la mise à jour des valeurs du reste du code en se basant sur la relation maître/esclave. Effectivement, la notification d'un changement d'au moins 1 bpm devrait ainsi être plus facilement détectée. Cela nécessite un matériel plus important mais nous permettrait d'atteindre des fréquences d'actualisation des données plus hautes. Nous sommes actuellement limités à 100Hz à cause du temps d'envoie des données (10ms environs).

De plus, nous constatons que le moteur ne tourne pas à la bonne fréquence. Nous n'avons pas, à ce jour, établie la raison de cette incohérence. Nous émettons l'hypothèse que l'assemblage des codes (contrôle commande et récupération des valeurs) empêche le bon fonctionnement de ce dernier. Effectivement, les codes pris séparément sont opérationnels.

Enfin, nous aurions souhaité permettre la modélisation des différentes injections.

Architecture logicielle

L'optimisation du logiciel se ferait à travers les points suivants :

- Affichage de toutes les courbes : fréquence respiratoire, fréquence cardiaque, débit d'urée.
- Amélioration de l'IHM : proposer une interface plus complète, proche de celle manipulée par les étudiants lors des séances de TP avec la possibilité d'enregistrer les courbes et données.
- Rendre les valeurs lues/modélisées plus pertinentes : la gestion des données est un point essentiel du projet : plus les données seront pertinentes et proches de la réalité, plus on pourra observer quelque chose d'intéressant d'un point de vue des élèves. C'est un point

qui sera donc important par traiter pour les futurs groupes en charge du projet, que ce soit dans la lecture des fichiers ou dans la modélisation des données.

V - Conclusion

Pour conclure, ce rapport détaille l'aspect technique de notre contribution au projet lapin robot. Nous aimerions ici présenter nos retours personnels sur cette expérience.

Arrivé au terme du temps qui nous a été alloué, nous voulons mettre en exergue la difficulté de reprendre un projet en milieu d'aboutissement. Même si la passation fût complète, l'esprit de conception entre les deux phases de développement reste néanmoins différent. Ceci explique pourquoi nous n'avons pu réellement commencer à mettre en œuvre nos premières idées dès le début.

Aujourd'hui, nous avons proposé une version évolutive du modèle précédent avec une architecture logicielle simplifiée, une prise en main plus rapide et une structure anatomique plus fidèle de la réalité. Nous avons mobilisé nos connaissances en nos options respectives pour concevoir et respecter au mieux les exigences premières du projet. Ces distinctions de spécialité ont permis un travail efficace tout en gardant une cohésion de groupe forte.

Pour finir, nous avons été ravis d'avoir pu participer à l'élaboration de ce lapin automate et espérons que ce dernier sera opérationnel à l'échelle nationale compte tenu des enjeux sociaux et éducatifs que représente une séance de physiologie en école vétérinaire.