Gymnase de Burier

Johan Link
3M5

Système de stabilisation PID

La Tour-de-Peilz, le 13 février 2020

Maître répondant : Alain Salanon

Résumé du travail de maturité

Ce travail consiste en la réalisation d'un système permettant de faire tenir une balle en équilibre sur un plateau qui a la possibilité de s'incliner sur deux axes pour annuler les mouvements de la balle. En d'autres termes, si on lance une balle sur ce plateau, ce dernier s'inclinera pour l'empêcher de tomber et il positionnera la balle en son centre.

Partie matérielle

Le système est composé de deux parties bien distinctes : une base qui contient trois moteurs avec leurs bras sur lesquels se trouve le plateau et un support qui maintient une caméra au-dessus du plateau. La base qui contient les moteurs est une structure en plastique de forme circulaire et de même dimension que le plateau. Cette structure est composée de plusieurs pièces en plastique qui ont été modélisées dans un logiciel puis fabriquées à l'aide d'une imprimante 3D.

Circuit électronique

La base qui contient les moteurs comprend également un circuit imprimé qui permet de contrôler les moteurs et de communiquer avec un ordinateur. J'ai conçu ce circuit spécialement pour ce projet. Je l'ai dessiné à l'aide d'un logiciel, puis il a été fabriqué en Chine. Le circuit contient un microcontrôleur similaire à celui de certaines cartes Arduino, par conséquent j'ai pu programmer mon propre circuit électronique à l'aide de l'environnement Arduino.

Programmation de l'ensemble

Le système fonctionne en trois temps : premièrement, la caméra prend une photo du plateau et de la balle et la transmet à un ordinateur, ensuite un programme Python analyse l'image pour détecter la position et la vitesse de la balle et en déduit que le plateau doit s'incliner d'un certain angle pour annuler les mouvements de la balle. Pour finir, le circuit imprimé fait bouger les moteurs en fonction des informations reçues par l'ordinateur. Le programme Python est la partie la plus importante du système car il traite les images reçues par la caméra, les analyse, puis en déduit à l'aide d'un régulateur PID l'inclinaison qu'il faut donner au plateau.

Utilisation d'un régulateur PID

Le régulateur PID est un système permettant de contrôler la position de la balle. En prenant en compte plusieurs facteurs comme la position et la vitesse de la balle, on peut connaître avec précision l'inclinaison qu'il faut donner au plateau pour que la balle se stabilise à une position voulue.

Remerciements

Je souhaite remercier M. Salanon de m'avoir suivi tout au long de ce TM et pour la relecture de mon travail. Je remercie également M. Bonnet pour m'avoir guidé dans l'amélioration de ce travail.

Table des matières

1 Introduction **5**
1.1 Motivations 5
1.2 Présentation du projet 5
1.3 Les cartes Arduino 6

2 Etat de l'art : robotique parallèle **7**
2.1 Robot Delta 7
2.2 Plateforme de Stewart 8
2.3 Conclusion 9

3 Conception de la partie mécanique **10**
3.1 Fonctionnement général du système mécanique 10
3.2 Conceptualisation géométrique du problème 10
3.3 Choix des moteurs 11
3.4 Conception du design du système 12
3.5 Fabrication des différents éléments 14
3.6 Assemblages du système 16
3.7 Fabrication du support et du boitier de la caméra 16

4 Résolution des équations à l'aide de Python **18**
4.1 Rôle de ce programme 18
4.2 Fonctionnement du programme 18

5 Conception du circuit imprimé **21**
5.1 Pourquoi réaliser un circuit imprimé ? 21
5.2 Cahier des charges 21
5.3 Qu'est ce qu'un microcontrôleur ? 21
5.4 Choix du microcontrôleur 22
5.5 Conception du PCB 22
5.6 Fabrication du PCB 23
5.7 Assemblage des composants 23
5.8 Initialisation du microcontrôleur 24

6 Fonctionnement d'un régulateur PID **26**
6.1 Qu'est ce qu'un régulateur PID ? 26
6.2 La régulation Proportionnelle 26
6.3 La régulation Proportionnelle et Intégrale 27
6.4 La régulation Proportionnelle, Intégrale et Dérivée 28

7 Mise en place du régulateur PID dans le programme Python **29**
7.1 Transformation des équations 29
7.2 Un exemple concret 29
7.2.1 Détermination des composantes I_x et I_y 29
7.2.2 Détermination de l'angle α 31

7.2.3 Détermination de l'angle β 32
7.2.4 Détermination des angles de chaque moteur 34

8 Utilisation de OpenCV avec Python 35
8.1 Qu'est-ce que OpenCV ? 35
8.2 Programme intermédiaire 35
8.3 Reconnaissance d'une couleur 36

9 L'interface graphique 37
9.1 Pourquoi faire une interface graphique ? 37
9.2 Fonctionnalités de l'interface 37

10 Le code Arduino 39
10.1 Communication avec le programme Python 39
10.2 Contrôle des servomoteurs 39

11 Analyse des performances du système selon les coefficients P, I et D 40
11.1 Première expérience : analyse des coefficients P et D 40
11.1.1 Mise en place de l'expérience 40
11.1.2 Réalisation de l'expérience 41
11.1.3 Résultats de l'expérience 42
11.1.4 Discussion des résultats 44
11.2 Deuxième expérience : analyse du coefficient I 46
11.2.1 Résultats pour la balle de ping-pong 47
11.2.2 Résultats pour la balle en céramique 47
11.2.3 Discussion des résultats 48
11.3 Dernière expérience : analyse des coefficients P et D lorsque la balle a une vitesse initiale non nulle 49
11.3.1 Résultats de l'expérience 49
11.3.2 Discussion des résultats 50

12 Discussion des trois expériences 51

13 Faiblesses du système 53
13.1 Temps de détection de la balle 53
13.2 Précision de la détection de la balle 55
13.3 Vitesse des servomoteurs 56
13.4 Effet néfaste de la rotation du plateau sur la vitesse de la balle . 57

14 Conclusion 58

Bibliographie 59

Annexes **62**
A Résultats de la section 11.1.3 63
B Suite des résultats de la section 11.1.3 67
C Résultats de la section 11.3.1 71
D solveEquation.py 75
E programmeIntermediaire.py 78
F interface.py 80
G arduinoCode.ino 92
H Détermination des angles θ des servomoteurs 94
I Quelques plans du projet 97
J Prototypes 104
K Quelques esquisses du projet 105

1 Introduction

1.1 Motivations

L'électronique, la programmation et l'informatique de façon générale sont des domaines qui m'intéressent depuis longtemps, c'est pourquoi j'ai souhaité réaliser un travail de maturité rassemblant toutes ces disciplines afin d'approfondir mes connaissances. J'ai toujours voulu savoir comment les objets ou les machines qui nous entourent fonctionnent c'est pourquoi j'ai l'habitude d'en démonter pour mieux comprendre leur mécanisme. J'ai commencé à m'intéresser à l'électronique il y a de nombreuses années, j'ai commencé par la fabrication de simples circuits électriques puis, plus tard, je me suis également intéressé à la programmation. Il y a quelques années j'ai découvert l'existence des cartes Arduino qui m'ont permis de réaliser des projets de plus en plus intéressants en me permettant de travailler à la fois du côté logiciel et matériel. Les cartes Arduino s'utilisent de manière très intuitive et sont entièrement open-source, de plus une énorme quantité de documentation est présente sur internet. Ce travail de maturité est donc l'occasion de faire un projet concret qui contient une partie hardware et une partie software.

1.2 Présentation du projet

Mon projet consiste en la réalisation d'un système permettant de faire tenir en équilibre une balle sur un plateau. Des servomoteurs permettront au plateau de s'orienter avec un certain angle d'inclinaison pour contrebalancer les mouvements de la bille. Une caméra placée au-dessus du dispositif transmettra en direct des images du plateau à un ordinateur qui sera chargé de les analyser pour en tirer des conclusions telles que la position de la bille, sa vitesse et son accélération. Un programme informatique sera alors chargé de traiter ces données et de les transmettre par USB à l'Arduino qui contrôlera la rotation de chaque servomoteur indépendamment. Le plateau bougera en conséquence des déplacements de la bille, même si cette dernière est déstabilisée par du vent ou une force quelconque, l'Arduino inclinera le plateau pour qu'elle ne tombe jamais.

Ce projet me permettra de concevoir un système contenant une partie mécanique et électronique, je serai également mené à coder un programme capable de traiter des images et d'en déduire la position d'un objet d'une couleur définie.

Dans un premier temps je vais construire et étudier les différents mécanismes qui permettront de faire bouger le plateau. Ensuite je devrai dessiner et fabriquer un circuit électronique contenant un microcontrôleur capable de communiquer avec l'ordinateur et capable de diriger les servomoteurs. Je finirai mon travail par la réalisation du programme qui traitera les images de la caméra et qui en déduira l'inclinaison du plateau nécessaire à l'équilibre de la bille.

1.3 Les cartes Arduino

Les cartes Arduino sont de petites cartes électroniques qui sont principalement composées d'un microcontrôleur autour duquel se trouve de nombreuses entrées et sorties qui permettent à la carte d'interagir avec différents composants électroniques ou quelconques systèmes extérieurs à la carte. Plusieurs modèles de cartes existent, la différence majeure qui les sépare est la puissance du microcontrôleur. Par exemple un microcontrôleur contenu dans un Arduino Mega contiendra une EEPROM d'une capacité de 4 kB alors qu'un Arduino Uno aura une EEPROM limitée à 1 kB. De plus les cartes plus puissantes ont également plus d'entrées et de sorties.
L'Arduino a été fondé en 2005 par un étudiant Italien, par la suite ce simple projet d'étudiant a révolutionné l'univers de l'électronique. Ces cartes permettent à n'importe quelle personne de concevoir un projet à moindre coût. Un Arduino Uno coûte environ 20 euros ce qui rend ces cartes accessibles. Mais l'avantage d'utiliser ces cartes open-source est d'avoir accès à une immense communauté de personnes publiant des tutoriels ou différents projets sur internet. De plus, Arduino fournit également un logiciel de programmation gratuit qui permet de programmer très simplement n'importe quelles cartes Arduino ou d'autres cartes avec un microcontrôleur similaire. En effet le logiciel Arduino peut également servir à programmer une carte d'une autre marque qu'Arduino ou encore une carte fabriquée par soi-même. C'est un exemple d'un des nombreux avantages que les logiciels open-source offrent.

Figure 1 – Photo d'un Arduino Uno
Source : https ://store.arduino.cc/arduino-uno-rev3

2 Etat de l'art : robotique parallèle

Un des éléments essentiel à mon travail de maturité est de trouver un moyen de manipuler une plateforme dans l'espace pour lui donner une orientation recherchée. Différents types de robots répondent déjà à des besoins proche de celui-ci mais ne sont pas toujours parfaitement adaptés à mon projet. Ces derniers font partie de la robotique parallèle : mécanisme en chaîne cinématique fermée dont l'organe terminal est relié à la base par plusieurs chaînes cinématiques indépendantes. [1]
La robotique parallèle à permis de créer des systèmes mécanique plus performant dans certaines applications que les robots sériels inventé plus tôt.

(a) Robot sériel

(b) Robot parallèle

FIGURE 2 – Comparaison entre un robot sériel et parallèle

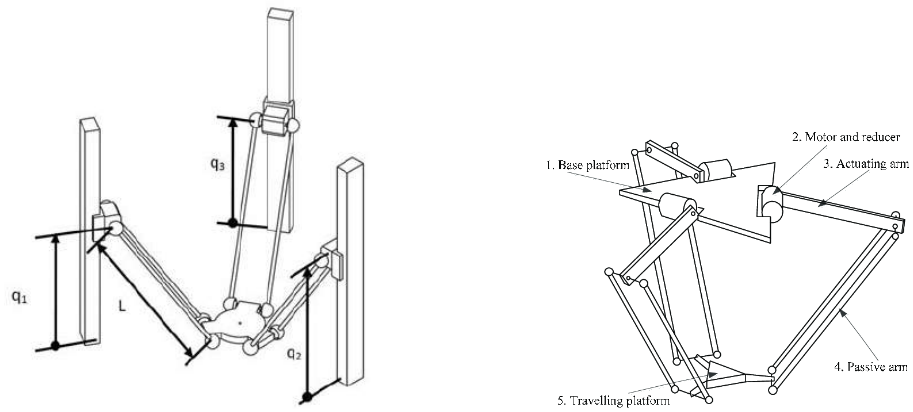
2.1 Robot Delta

Inventé en 1985 par l'ingénieur Reymond Clavel à l'EPFL, le robot Delta (la figure 2b présente un exemple de robot delta) a révolutionné l'industrie. En effet ce système comporte un grand nombre d'avantages en comparaison avec les robots sériels, sa géométrie lui donne une grande rigidité tout en utilisant des matériaux légers et fins. Cette géométrie permet également des déplacements très rapides et précis. En général ce système comporte 3 degrés de liberté ce qui lui permet de soulever des objets pour les translater par exemple. Ces mouvements sont particulièrement adaptés à l'industrie où il faut souvent déplacer les objets fabriqués pour les ranger dans des boites par exemple. Par conséquent,

1. Définition de Jean-Pierre Merlet, Les Robots Parallèles, Paris, Hermes Science Publishing, 21 février 1997, 367 p. (ISBN 978-2866015992)

Source de la figure 2b : https ://new.abb.com/products/robotics/fr/robots-industriels/irb-360-flexpicker

Source de la figure 2a : https ://new.abb.com/products/robotics/de/industrieroboter/SAY dans les robots delta, la plateforme mouvante se déplace toujours de manière parallèle à la plateforme sur laquelle les moteurs sont fixés (voir figure 3b).



(a) Robot delta à actionneurs linéaires

(b) Robot delta à actionneurs rotatifs

FIGURE 3 – Comparaison entre des actionneurs linéaires et rotatifs dans un robot delta

Le robot delta est souvent muni d'actionneurs rotatifs, un des désavantages de ce système est que la plateforme mouvante du système ne peut pas se déplacer dans un très grand espace de travail. Cela ne pose aucun problème lorsque le robot doit juste déplacer de petits objets d'un tapis roulants à l'autre dans une usine par exemple. Cependant il faut parfois un système capable d'évoluer dans un plus grand espace de travail, on peut alors fixer les bras du robot delta directement sur des actionneurs linéaires (voir figure 3a). Ce système est parfois utilisé dans les imprimante 3D car il permet d'augmenter l'espace de travail du robot ce qui permet d'imprimer des pièces de grande taille.

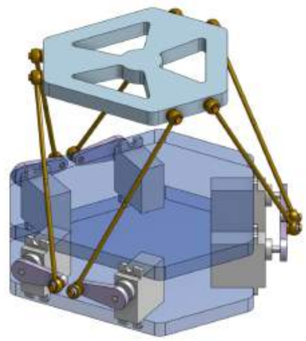
2.2 Plateforme de Stewart

La plateforme de Stewart, inventée en 1965, est également un système qui fait partie de la robotique parallèle. Ce système était destiné à améliorer le réalisme des simulateurs de vols. Le mécanisme est composé d'une plateforme fixée sur six vérins par des articulations mobiles. Les vérins sont indépendants les uns des autres et servent à orienter la plateforme. Cette dernière dispose de six degrés de liberté (les trois coordonnées de translation et les angles de tangage, roulement et lacet). Fonctionnant avec six vérins, le système à la capacité de faire bouger des charges très lourdes cependant les vérins ont le désavantage d'être en général relativement lent. Il existe cependant des versions de la plateforme de

Source de la figure 3a : https ://www.researchgate.net/figure/DELTA-parallel-robot-with-linear-actuators_fig2_4356741

Source de la figure 3b : https ://www.researchgate.net/figure/Scheme-of-the-Delta-robot_fig3_303469087

Stewart utilisant des actionneurs rotatifs ce qui permet d'augmenter la vitesse des mouvements.



(a) Plateforme de Stewart à actionneurs rotatifs

(b) Plateforme de Stewart à actionneurs linéaires

FIGURE 4 – Exemples de plateforme de Stewart

2.3 Conclusion

La robotique parallèle semble être une bonne solution pour ce travail de maturité, en effet nous avons vu que ce type de robotique permet de créer des mécanismes dont la géométrie comporte de nombreux avantages. La plateforme de Stewart et le robot delta permettent tous les deux de déplacer une plateforme avec précision et à grande vitesse. Les pièces mécaniques utilisées étant relativement simples et légères, elles n'ont pas une grande inertie ce qui permet de leur faire subir de grandes accélérations sans problèmes. Malgré les avantages vus précédemment, ces systèmes ne sont pas parfaitement adaptés pour satisfaire les exigences de ce travail. Effectivement il est question d'orienter un plateau selon les angles de roulis et de tangage, or un robot delta permet uniquement de translater sa plateforme dans l'espace sans pouvoir lui faire subir des rotations. Quant à elle, la plateforme de Stewart permet à sa plateforme de se translater dans l'espace mais également de subir des rotations selon les angles de roulis, tangage et lacet. Par conséquent la plateforme de Stewart remplis les exigences de roulis et tangage mais offre également beaucoup d'autres degrés de liberté qui ne sont pas nécessaires dans l'élaboration de ce travail. Il faudra alors trouver un moyen de simplifier la géométrie d'une plateforme de Stewart tout en prenant soins de rester dans la robotique parallèle pour conserver ses propriétés mécaniques intéressantes.

Source de la figure 4a : https ://github.com/NicHub/stewart-platform-esp32
Source de la figure 4b : https ://fr.m.wikipedia.org/wiki/Robot_parallèle

3 Conception de la partie mécanique

3.1 Fonctionnement général du système mécanique

Le plateau sur lequel reposera la bille devra être capable de s'orienter sur deux axes et devra pouvoir s'incliner de 35° au maximum. Étant donné que la plate-forme doit se mouvoir sur deux axes j'ai rapidement commencé à réfléchir à un système comportant deux moteurs. Cependant, après réflexion je me suis rendu compte qu'il était plus simple d'utiliser trois moteurs (fig.5). L'ajout d'un moteur permet également de rajouter un degré de liberté au plateau, en effet avec trois moteurs la plate-forme peut s'incliner sur deux axes et peut également bouger de façon verticale, mais ce degré de liberté ne sera pas utilisé dans ce projet. À présent nous savons que le système comprendra trois moteurs placés à 120° l'un de l'autre et que le mouvement de rotation de ces derniers devra entraîner l'inclinaison du plateau. Pour ce faire, chaque moteur mettra en rotation un bras qui sera composé de deux parties reliées entre elles par une articulation (fig.6). Une bille métallique sera fixée à l'extrémité du bras de chaque moteur et cette bille fera office de rotule entre le bout du bras et le plateau situé en dessus.

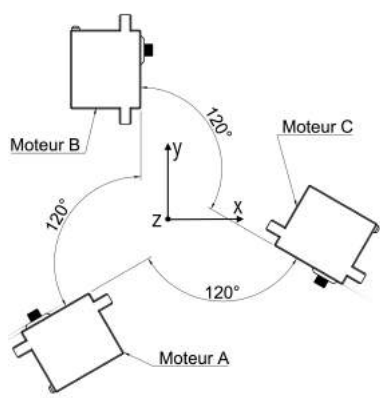


FIGURE 5 – Disposition des moteurs

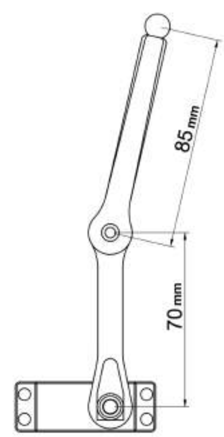


FIGURE 6 – Bras d'un moteur

3.2 Conceptualisation géométrique du problème

Comme dit précédemment, le dispositif comporte trois moteurs, il faut donc trouver un moyen de connaître l'angle du bras de chaque moteur en fonction d'une inclinaison donnée du plateau. Les moteurs ont une position fixe, chaque bras se déplace dans un plan vertical et la distance entre le bout de chaque bras est constante. Dans un premier temps j'ai commencé à réaliser différents calculs permettant de mettre en relation les différentes contraintes du problème. J'ai également établi un système permettant de définir la position du plateau en fonction de deux angles. Pour ce faire j'ai ajouté dans mes calculs un vecteur $\vec{v}$ (fig.7) de norme 1 qui est toujours perpendiculaire au plateau et dont l'origine correspond au centre du plateau. L'orientation de ce vecteur est défini par les angles α et β.

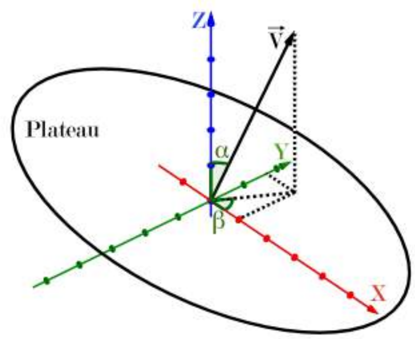


FIGURE 7 – Vecteur $\vec{v}$ perpendiculaire au plateau

Après avoir réalisé plusieurs calculs[2] j'ai pu établir des équations qui permettent de déterminer la position de chaque bille située sur le bout des bras en fonction d'une inclinaison connue du plateau. Après avoir trouvé la position dans l'espace d'une bille on doit encore trouver l'angle du moteur qui permet de positionner le bout du bras avec la bille au bon endroit. L'équation[3] suivante permet de calculer l'angle θ que le bras doit avoir par rapport à l'horizontale :

$$(L - r\cos\theta - \sqrt{X^2 + Y^2})^2 + (r\sin\theta - Z)^2 = l^2 \tag{1}$$

où L est la distance entre l'arbre d'un moteur et l'origine (fig.5), r est la longueur de la première portion du bras de chaque moteur et l est la longueur du deuxième segment du bras (segment sur lequel est attaché la bille). X, Y et Z sont les coordonnées[4] de la bille qui ont été calculées précédemment. L'équation (1) doit donc être utilisée individuellement pour chaque moteur, en d'autres termes on calcule d'abord la position de la bille du bras du moteur A et on utilise l'équation (1) pour déterminer l'angle du moteur A, ensuite on refait les mêmes calculs pour les moteurs B et C.

3.3 Choix des moteurs

Comme vu précédemment, trois bras composés de deux parties chacun devront maintenir et faire bouger le plateau situé en dessus d'eux. Il est donc

2. Ces calculs sont décrits aux pages 94 et 95.
3. L'équation 1 est décrite plus précisément à la page 96.
4. Le repère orthonormé de ces coordonnées se trouve sur le plan qui contient les trois moteurs et se trouve à équidistance de ces derniers. (figure 5, page 10)

évident qu'il est nécessaire d'avoir des moteurs pour mettre en mouvement chaque bras. Cependant mon projet contient plusieurs contraintes qu'il faut prendre en compte pour le choix des moteurs. Premièrement, la masse du plateau repose entièrement sur les moteurs, ils doivent alors être relativement puissants pour le soulever et le maintenir dans une certaine position. Deuxièmement, les moteurs doivent pouvoir incliner le plateau rapidement pour compenser les déplacements parfois rapides de la bille. Troisièmement, il faut que les moteurs soient précis pour que le système fonctionne correctement.
Il existe énormément de types de moteurs différents, cependant je me suis intéressé à deux types de moteurs en particulier : les moteurs pas-à-pas et les servomoteurs. Les servomoteurs sont rapides, puissants et assez précis, ils peuvent cependant effectuer des rotations de seulement 180° ou 360° au maximum mais cela n'est pas un problème pour ce projet. Ils sont très pratiques car ils sont relativement petits et sont très faciles à utiliser avec un Arduino. Je me suis également intéressé aux moteurs pas-à-pas car ils sont plus précis que les servomoteurs, cependant ils comportent quelques désavantages, par exemple, il faut utiliser des drivers [5] pour les faire fonctionner avec un Arduino.
J'ai finalement décidé d'utiliser des servomoteurs [6] (voir figure 8) pour ce projet car ils sont plus simples à utiliser et ont l'avantage de pouvoir maintenir leur position même si une force extérieure est exercée sur l'arbre du moteur. En effet les servomoteurs contiennent un circuit électronique qui régule la position du moteur, par conséquent ces moteurs réajustent leur position en continu et de manière automatique.

FIGURE 8 – Photo d'un servomoteur utilisé pour ce projet

3.4 Conception du design du système

Cette partie consiste en la conception de l'armature qui devra maintenir les différents éléments tels que les servomoteurs, le circuit imprimé ou encore les divers connecteurs (connecteur pour l'alimentation, port USB et connecteur

5. Petits circuits électroniques contrôlant la rotation des moteurs pas-à-pas.
6. Référence des servomoteurs utilisés : 05101748-1
Source de la figure 8 : https ://www.digitec.ch/fr/s1/product/futaba-servo-servos-rc-5335020

pour un joystick). J'ai commencé par modéliser les différentes pièces dans le logiciel *Fusion 360*. Le plateau a un diamètre de 27 cm, par conséquent le socle qui contient les moteurs fait également ce diamètre. La majorité des pièces vont être imprimées en 3D par la suite, malheureusement la majorité des imprimantes 3D ne sont pas capables d'imprimer des pièces aussi grandes, c'est pourquoi j'ai dû diviser plusieurs pièces en plusieurs petites pièces qui peuvent s'assembler à l'aide de boulons et d'écrous.

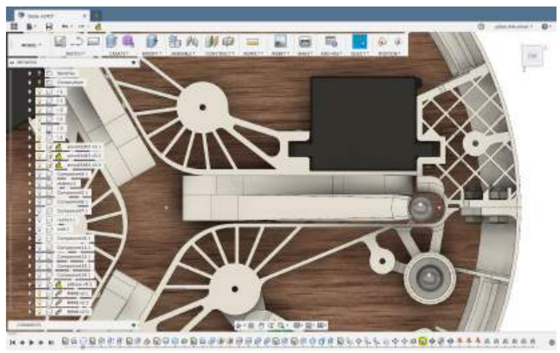


FIGURE 9 – Capture d'écran de mon projet dans *Fusion 360*

La figure (fig.10, page 14) permet de se rendre compte de toutes les différentes pièces qui composent le système. On aperçoit également trois disques, le disque se trouvant tout en haut correspond au plateau sur lequel roulera la bille, les deux autres disques forment respectivement le haut et le bas du boitier qui contient les moteurs, le circuit électronique et les autres composants. Les pièces imprimées en 3D forment le pourtour du boitier et certaines sont également présentes à l'intérieur et servent de support aux moteurs et au circuit.

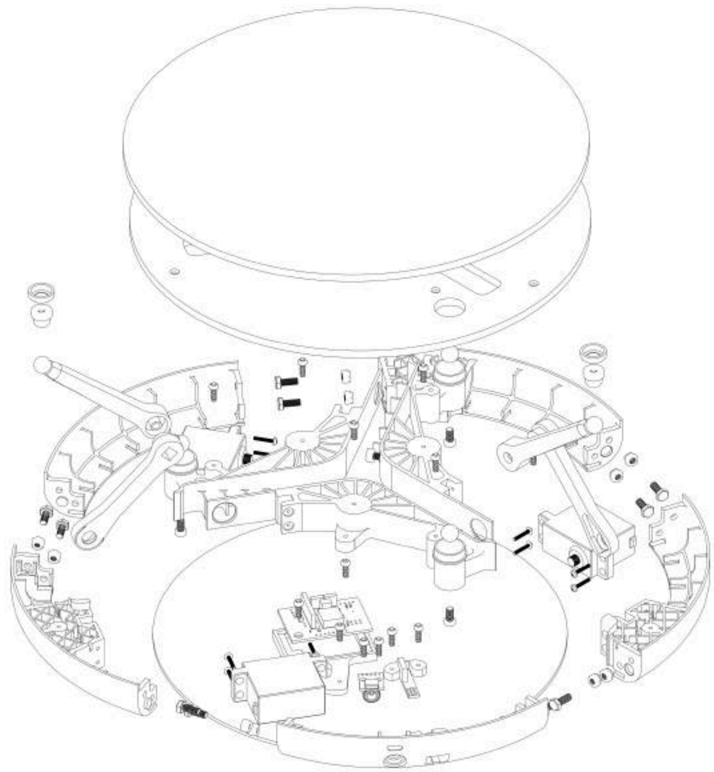


FIGURE 10 – Vue générale de tous les éléments du système

3.5 Fabrication des différents éléments

Étant donné que le système comprend un grand nombre de grands éléments à imprimer en 3D, j'ai décidé de faire appel au site *3D Hubs*. Il m'a suffi de télécharger tous les fichiers de mes pièces sur leur site, puis ils se sont occupés d'imprimer les différents éléments et de les envoyer. De plus, les pièces que j'ai dessinées sont relativement complexes, il fallait donc une imprimante 3D qui dispose d'une grande précision pour pouvoir les fabriquer. Les pièces ont toutes été imprimées en ABS (acrylonitrile butadiène styrène). L'ABS est un plastique très courant dans l'industrie, il est souvent utilisé dans les appareils électro-ménagers ou encore dans les jouets comme par exemple les briques *LEGO*. Ce plastique comporte des caractéristiques intéressantes car il est plutôt léger et est très résistant aux chocs. Un de ses plus grands défauts est d'avoir tendance à jaunir lorsqu'il est exposé à la lumière du soleil.
Les trois disques que l'on voit sur la figure 10 (page 14) ont été fabriqué par l' *Atelier 12Mill*[7]. Il s'agit d'un atelier, basé à Lausanne, qui permet de faire fa-

7. Adresse : Avenue de Sévelin 48 - 1004 Lausanne

briquer toutes sortes de pièces avec des techniques différentes. Deux des disques que j'ai fait fabriquer ont été fraisés dans une plaque en acrylique transparent. L'un de ces deux disques a été peint en blanc. Le troisième disque est le plateau sur lequel repose la bille, il a été fabriqué en carton car il fallait un matériau léger pour que le plateau n'ait pas trop d'inertie. Après avoir testé un plateau en bois, je me suis rendu compte qu'un plateau avec trop d'inertie avait tendance, lors de mouvements brusque, à se décrocher des bras magnétiques qui le supportent.

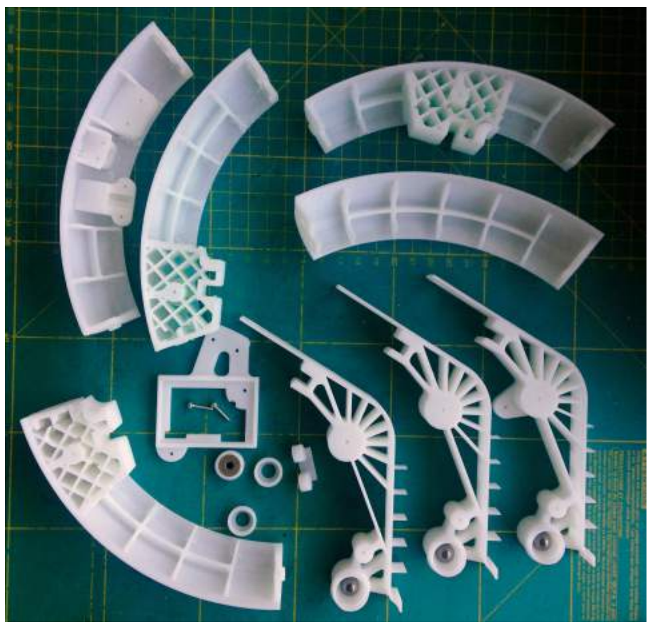


FIGURE 11 – Pièces imprimées en 3D

3.6 Assemblages du système

La dernière étape de la construction consiste en l'assemblage de toutes les pièces du système. Tous les éléments s'emboitent et sont maintenus entre eux par des boulons et des écrous. Cette étape est relativement simple si les pièces ont été correctement dessinées au préalable. Dans mon cas l'assemblage s'est déroulé convenablement, bien que certaines pièces comportaient des défauts qui étaient dus à l'impression 3D. Ces petits défauts ont causé des désalignements, mais heureusement il est assez facile de modifier les pièces à l'aide d'un simple cutter.

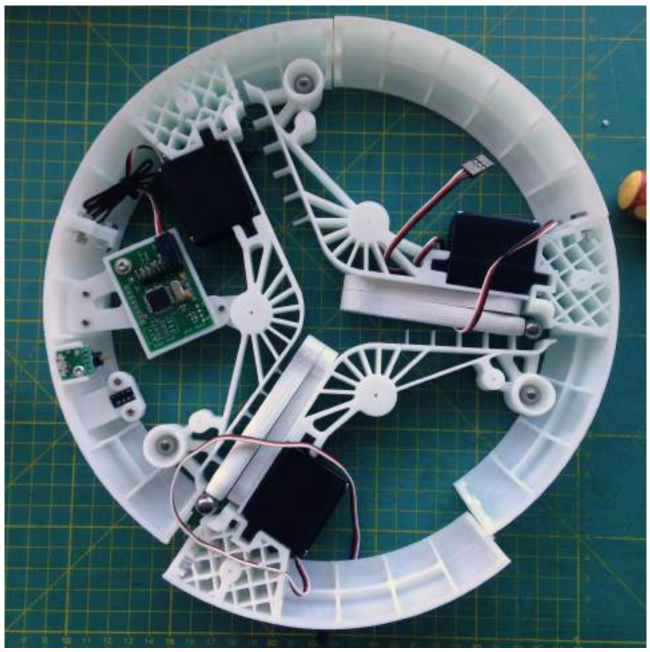


FIGURE 12 – Début de l'assemblage des pièces

3.7 Fabrication du support et du boitier de la caméra

La caméra qui doit filmer le plateau est placé sur un support qui est complètement séparé de la base qui contient les servomoteurs. Le boitier de la caméra est monté sur un tube en aluminium qui lui-même est emboité à un support qui contient trois aimants qui permettent à ce support de se fixer sur des surfaces métalliques. Le boitier de la caméra ainsi que le support ont été imprimés en 3D. Le câble USB de la caméra passe à l'intérieur du tube en aluminium et sort à l'arrière du support aimanté.

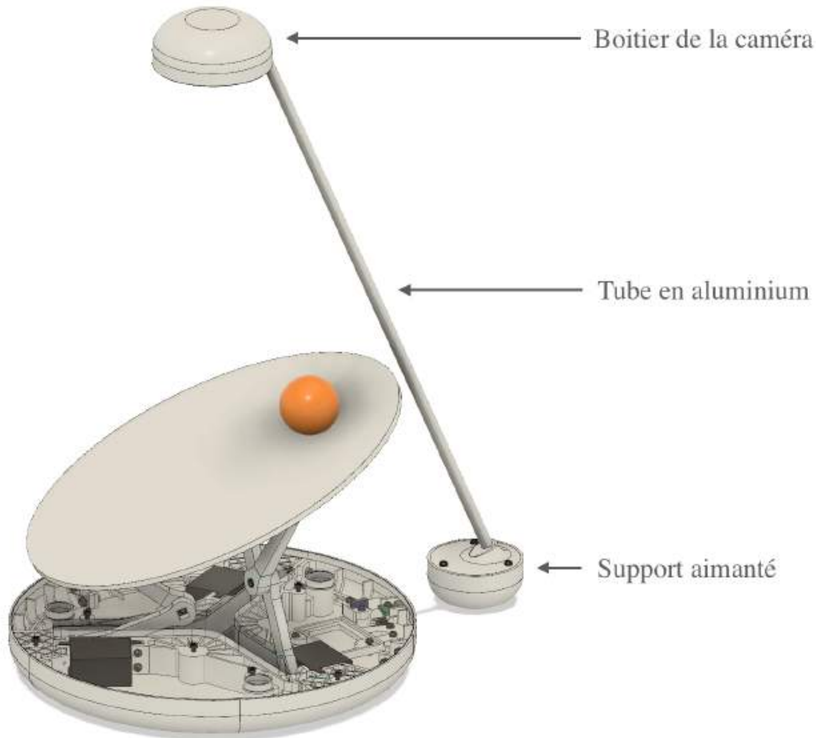


FIGURE 13 – Capture d'écran du support de la caméra dans *Fusion 360*

FIGURE 14 – Caméra utilisée pour ce projet
référence : ELP-USBFHD01M

4 Résolution des équations à l'aide de Python

4.1 Rôle de ce programme

Les équations présentées dans la partie 3.2 de la page 10 sont particulièrement longues et complexes, il est donc très difficile de les manipuler pour en isoler une variable par exemple. Ces équations sont trop compliquées pour être résolues sans l'aide d'un ordinateur, c'est pourquoi j'ai décidé d'écrire un programme python permettant de résoudre les équations. Le programme génère un fichier texte qui contient les résultats de toutes les équations que le programme à résolues pour obtenir l'angle de chaque moteur en fonction des angles α et β qui définissent l'inclinaison du plateau.

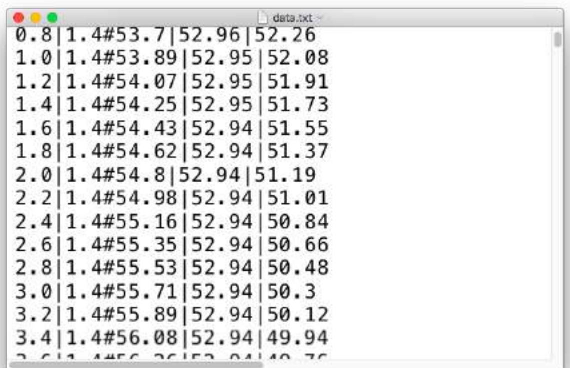


FIGURE 15 – Capture d'écran du fichier texte

Chacune des lignes comporte les informations suivantes : α | β # θ_A | θ_B | θ_C

4.2 Fonctionnement du programme

Ce programme génère un fichier texte en résolvant plusieurs systèmes d'équations, pour ce faire il est nécessaire de fixer certaines constantes au début du code. Voici la liste des différentes valeurs qu'il faut rentrer dans le programme pour qu'il puisse fonctionner :

— Distance entre l'arbre d'un moteur et le centre de la base[8] : L = 8.5cm
— Hauteur du plateau par rapport à la base : d = 11.5cm
— Distance entre l'extrémité de chaque bras : D = 18cm
— Longueur du premier segment de chaque bras : r = 7cm

8. La base correspond au socle sur lequel sont fixés tous les moteurs.

— Longueur du deuxième segment de chaque bras : l = 8.5cm

Le programme doit à présent effectuer plusieurs systèmes d'équations, j'ai donc décidé d'utiliser la fonction *fsolve* issue du module *scipy.optimize*. Cette fonction permet de résoudre des systèmes d'équations en utilisant une méthode de résolution approchée, par conséquent les solutions ne seront pas exactes mais sont suffisamment précises pour ce projet.

```
def equationsKMT(p):
    k, m, t = p
    equation1 = (-cos(alpha)*cos(pi/6)*k)**2 + (-cos(alpha)*m-cos(alpha)*sin(pi/6)*k)**2 + (sin(beta)*sin(alpha)*m+cos(pi/6)*cos(beta)*sin(alpha)*k+sin(beta)*sin(alpha)*sin(pi/6)*k)**2 - 1
    equation2 = (-cos(alpha)*cos(pi/6)*t)**2 + (cos(alpha)*sin(pi/6)*t+cos(alpha)*m)**2 + (cos(beta)*sin(alpha)*cos(pi/6)*t-sin(beta)*sin(alpha)*sin(pi/6)*t-sin(beta)*sin(alpha)*m)**2 - 1
    equation3 = (cos(alpha)*cos(pi/6)*k+cos(alpha)*cos(pi/6)*t)**2 + (cos(alpha)*sin(pi/6)*k-cos(alpha)*sin(pi/6)*t)**2 + (-cos(pi/6)*cos(beta)*sin(alpha)*k-sin(beta)*sin(alpha)*sin(pi/6)*k-cos(beta)*sin(alpha)*cos(pi/6)*t+sin(beta)*sin(alpha)*sin(pi/6)*t)**2 - 1
    return (equation1, equation2, equation3)
```

Ci-dessus se trouve une fonction qui contient un système d'équations [9] à trois inconnues. Dans la suite du code la commande *fsolve(equationsKMT, (1, 1, 1))* est utilisée pour déterminer la valeur des variables [10] k, m et t.
La partie la plus importante de ce code est une boucle dans laquelle le programme va résoudre les systèmes d'équations plusieurs fois en faisant varier les angles α et β. L'angle α varie entre 0° et 35° par pas de 0.2° et β varie entre 0° et 360° par pas de 0.2°également.

```
for beta in range(0, 360*5+1):
    beta = radians(beta)/5
    for alpha in range(0, 35*5+1):
        alpha = radians(alpha)/5
        k,m,t = fsolve(equationsKMT, (1, 1, 1))
        k,m,t = -D*k, -D*m, -D*t
        Xa, Ya, Za = k*cos(alpha)*cos(pi/6), k*cos(alpha)*sin(pi/6), k*(-cos(pi/6)*cos(beta)*sin(alpha)-sin(beta)*sin(alpha)*sin(pi/6))+d
        Xb, Yb, Zb = 0, m*(-cos(alpha)), m*sin(beta)*sin(alpha)+d
        Xc, Yc, Zc = t*(-cos(alpha)*cos(pi/6)), t*cos(alpha)*sin(pi/6), t*(cos(beta)*sin(alpha)*cos(pi/6)-sin(beta)*sin(alpha)*sin(pi/6))+d
        tetaA = degrees(fsolve(equationTeta, 1, args=(Xa, Ya, Za)))
        tetaB = degrees(fsolve(equationTeta, 1, args=(Xb, Yb, Zb)))
        tetaC = degrees(fsolve(equationTeta, 1, args=(Xc, Yc, Zc)))
        tetaA = round(tetaA, 2)
```

9. Ce système d'équations est présenté à la page 95.
10. Les variables k, m et t permettent de calculer la position du bout de chaque bras.

```
        tetaB = round(tetaB, 2)
        tetaC = round(tetaC, 2)

        separateur = "|"
        data = str(round(degrees(alpha),2)) +  separateur + str(round(degrees(beta),2)) + "#" + str(tetaA) + separateur + str(tetaB) + separateur + str(tetaC) + "\n"
        fichier.write(data)
```

Le fichier texte produit par ce programme sera utilisé plus tard par le programme Python *interface.py* qui s'occupera du traitement des images et de la communication avec le circuit imprimé.

5 Conception du circuit imprimé

5.1 Pourquoi réaliser un circuit imprimé ?

Les cartes Arduino sont très pratiques pour réaliser rapidement des prototypes de circuits électroniques. Cependant lorsqu'on utilise ces cartes on se retrouve souvent avec des dizaines de câbles reliant l'Arduino à une platine d'expérimentation [11] qui est souvent elle-même reliée à d'autres composants comme par exemple des servomoteurs. Cela n'est pas un problème lorsqu'on travaille sur un prototype ou lorsqu'on veut tester des circuits, mais si on souhaite fabriquer un système plus fiable et moins encombrant il est préférable de réaliser un circuit imprimé.

5.2 Cahier des charges

C'est souvent très utile de fabriquer un circuit imprimé mais il faut garder en tête qu'il est très difficile de modifier le circuit de ce dernier après l'avoir fabriqué. C'est pourquoi il faut bien réfléchir aux différents composants et fonctions du circuit avant de le faire fabriquer. Le but de cette étape est de dresser une liste des différentes actions que le circuit sera capable d'effectuer. Le circuit dont j'ai besoin pour mon projet est très simple et contient relativement peu de composants, voici la liste des différents éléments qui seront présents dans le futur circuit :

— 1 microcontrôleur ATMEGA32U4-AU
— 1 oscillateur quartz d'une fréquence de 16MHz
— 2 condensateurs céramique de 22pf
— 1 Une diode électroluminescente (led)
— 1 Une résistance de 1kΩ
— 1 Une résistance de 10kΩ
— 2 Une résistance de 22Ω
— 1 condensateur de 0.1μf
— 1 condensateur de 1μf
— 1 condensateur de 10μf
— 3 sorties PWM pour le contrôle des servomoteurs
— 1 entrée pour une alimentation 5 volts destinée aux servomoteurs
— 4 pins destiné à être reliés à un port micro USB
— 4 pins destiné à l'utilisation d'un joystick [12] utilisant l'interface I2C

5.3 Qu'est ce qu'un microcontrôleur ?

Un microcontrôleur se présente sous la forme d'une puce électronique qui contient tous les éléments essentiels au bon fonctionnement d'un ordinateur. En

11. Plus connue sous le nom de "breadboard" en anglais.
12. Pour des raisons techniques, le joystick ne sera finalement pas utilisé dans la suite du projet.

effet on y retrouve les composants suivants : un microprocesseur, une mémoire RAM [13], une mémoire EEPROM [14], une mémoire Flash [15], plusieurs entrées / sorties et un oscillateur.

5.4 Choix du microcontrôleur

Comme présenté dans la partie 5.2, le circuit imprimé sera équipé d'un microcontrôleur ATMEGA32U4-AU. Cette puce est exactement la même qui est utilisée dans les Arduino Leonardo. Son avantage premier est d'avoir la capacité de gérer la communication USB sans l'aide d'un autre processeur. Cela permet de simplifier la conception du circuit imprimé tout en réduisant les coûts car moins de composants sont utilisés.

5.5 Conception du PCB

Pour concevoir le circuit imprimé de ce projet j'ai utilisé le logiciel *EAGLE* qui est un logiciel de conception assistée par ordinateur. La création du circuit s'est déroulé en deux étapes, j'ai commencé par la réalisation du schéma électronique(fig.16), puis j'ai réalisé le routage du circuit(fig.17). Le routage consiste à positionner les différents composants du circuit et à définir les passages par lesquels les connexions vont passer.

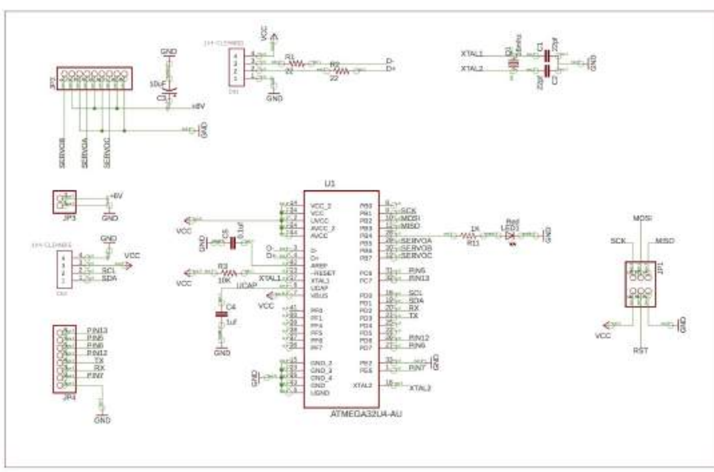


FIGURE 16 – Schéma du circuit imprimé

13. La RAM est une mémoire rapide mais qui stocke les informations de manière provisoire.
14. L'EEPROM est une mémoire non volatile.
15. La mémoire Flash des microcontrôleurs sert à stocker le programme que l'on a téléchargé.

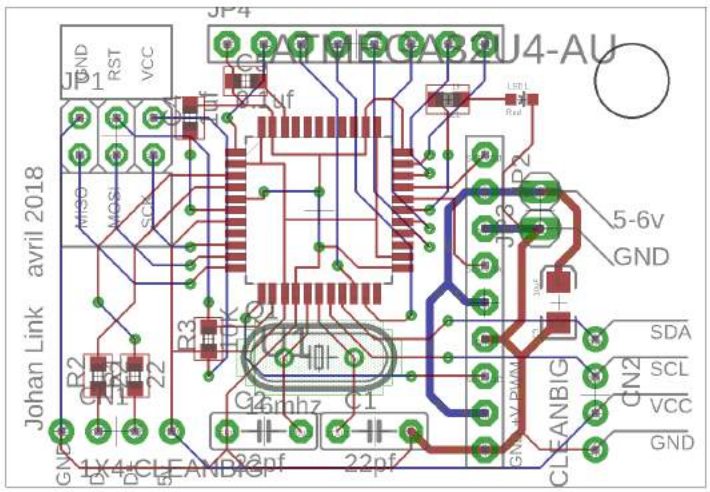


FIGURE 17 – Circuit imprimé

5.6 Fabrication du PCB

Après avoir réalisé le circuit sur le logiciel *EAGLE* il faut le fabriquer. Pour ce faire j'ai utilisé le site `https://www.pcbway.com` qui est un fabricant de circuits imprimés situé à Shenzhen en Chine. Ce site est particulièrement connu pour son rapport qualité-prix. L'utilisation de ce site est très simple, il suffit d'importer les fichiers *Gerber*[16] de son circuit sur le site et de choisir certains paramètres tels que la couleur du circuit ou encore son épaisseur, ensuite la fabrication se déroule sur quelques jours et le circuit est envoyé à sa destination.

5.7 Assemblage des composants

Après avoir reçu le circuit imprimé il faut encore y souder tous les composants. Cette partie est particulièrement compliquée car les pièces utilisées sont très petites[17] et donc difficiles à souder. Malheureusement certains composants tels que le microcontrôleur ne sont disponibles que dans des formats très petits. Je ne suis pas équipé avec les bons outils pour souder ce type de composants, cependant j'ai tout de même réussi à souder le microcontrôleur et les autres composants à l'aide d'un fer à souder standard. J'ai également fabriqué une panne[18] à souder à l'aide d'un fil de cuivre afin d'avoir une panne assez petite pour pouvoir souder les composants. Par conséquent les soudures effectuées ne

16. Le format *Gerber* est un format standard pour stocker les informations d'un circuit imprimé.

17. Les composants utilisés sont soudés à la surface du circuit imprimé. Ce sont des composants SMD (Surface Mounted Device).

18. La panne du fer à souder est le bout du fer. Il s'agit de la partie qui est en contact avec la soudure.

sont pas toujours très propres mais fonctionnent correctement. On peut voir sur la figure 18 que j'ai utilisé la caméra de mon téléphone en guise de loupe pour mieux voir les pattes des composants.

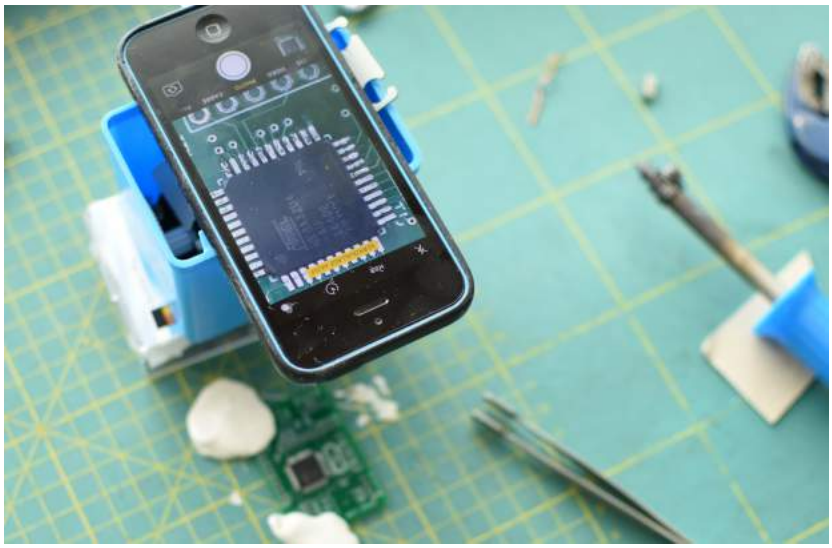


FIGURE 18 – Soudage du microcontrôleur sur le PCB

5.8 Initialisation du microcontrôleur

Après avoir soudé tous les composants on ne peut pas encore utiliser le circuit, en effet le microcontrôleur a besoin de contenir un bootloader. Le bootloader est un microprogramme qui doit être dans le microcontrôleur pour que ce dernier puisse être programmé avec l'environnement Arduino. Il est relativement simple d'installer le bootloader dans le micrcontrôleur, pour commencer il va falloir téléverser l'exemple *ArduinoISP* (Fichier > Exemples > ArduinoISP) dans un arduino UNO. Ensuite il faut faire les connexions suivantes entre l'arduino UNO et le circuit imprimé :

Broches de l'Arduino UNO	Broches du circuit imprimé
10	RST
11	MOSI
12	MISO
13	SCK
5v	Vcc
Gnd	Gnd

Les broches citées dans la colonne de droite du tableau sont toutes regroupées en haut à gauche du circuit imprimé (fig.17). Après avoir réalisé ces branchements il suffit de graver la séquence d'initialisation depuis le logiciel Arduino

(Outils > Graver la séquence d'initialisation). Une fois cette étape terminée le microcontrôleur est prêt à être programmé en branchant le circuit imprimé à l'ordinateur avec un câble USB.

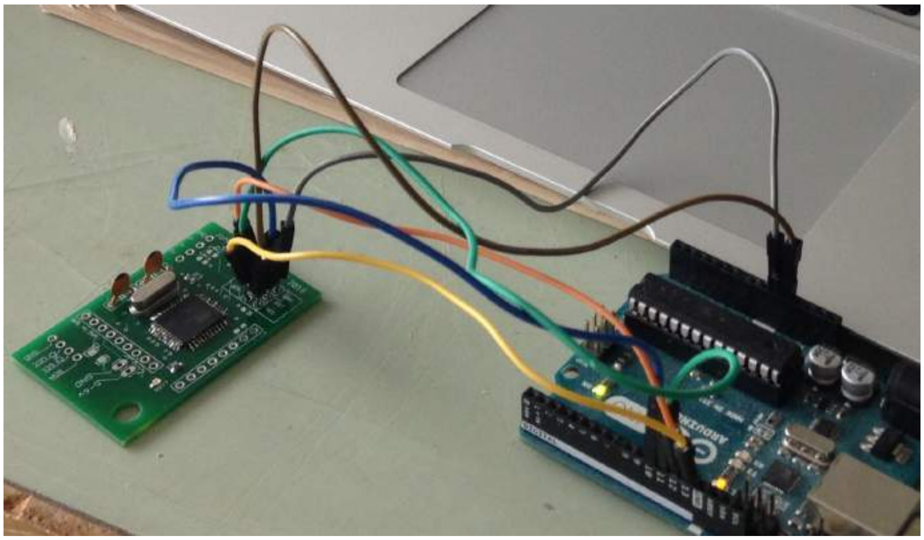


Figure 19 – Initialisation du microcontrôleur

6 Fonctionnement d'un régulateur PID

6.1 Qu'est ce qu'un régulateur PID ?

Un régulateur PID est un système dont l'objectif est de réguler une grandeur réelle pour qu'elle se rapproche au maximum d'une valeur que l'on souhaite atteindre. Le but de ce système est de parvenir à la consigne [19] le plus rapidement possible et cela peu importe les éléments qui peuvent perturber le régulateur. Dans le cadre de ce travail de maturité, le régulateur PID sera un élément-clé car il contrôlera la position de la bille sur le plateau afin de la faire correspondre à la position souhaitée. Le système pourra alors déduire l'inclinaison requise du plateau pour déplacer la bille au bon emplacement.

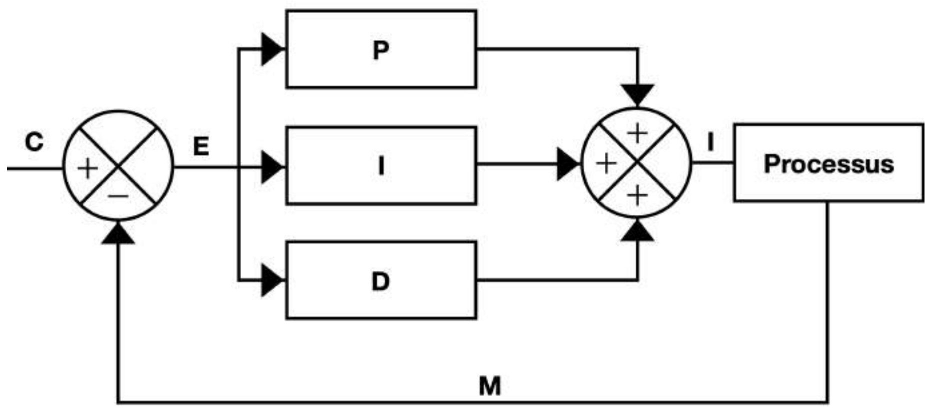


FIGURE 20 – Schéma d'un régulateur PID

Le schéma ci-dessus représente le fonctionnement du régulateur PID qui va être utilisé. Ce schéma se lit de gauche à droite, la lettre C représente la consigne et la lettre M la mesure donnée par un capteur. La différence des deux valeurs va nous donner l'erreur (lettre E sur le schéma) qui va ensuite être traitée par le correcteur PID pour finalement obtenir une instruction (lettre I) qui servira à diminuer l'erreur.

6.2 La régulation Proportionnelle

Un régulateur PID est composé de trois types de régulations, le premier est la régulation proportionnelle. Ce principe de régulation est le plus simple car il met en relation la valeur mesurée avec la consigne de manière linéaire. En d'autres termes, plus la bille sera éloignée de la position souhaitée (consigne), plus le plateau se penchera. Pour un régulateur proportionnel on a la relation suivante :

$$I = K_p(C - M) \tag{2}$$

19. Valeur que l'on souhaite atteindre.

où I est l'instruction, K_p est un coefficient, C est la consigne et M est la valeur réelle mesurée. L'instruction est la valeur qui permet de déterminer les angles des servomoteurs pour qu'ils placent le plateau dans la bonne position. Le coefficient K_p doit être déterminé expérimentalement. Si K_p est grand alors la consigne sera atteinte plus rapidement et avec précision mais le système risque de devenir instable. Il faut donc trouver un compromis entre la stabilité et la rapidité du système.

La régulation proportionnelle ne prend pas en compte la vitesse à laquelle l'écart entre la mesure et la consigne évolue. La durée de l'écart n'est pas non plus pris en compte. De plus ce type de régulation a un autre défaut car, dans certains cas, la consigne n'est jamais atteinte. En effet il se peut que le système se stabilise légèrement en dessus ou en dessous de la consigne et crée une erreur statique. Par exemple, si le système que j'ai fabriqué se trouve sur une surface qui n'est pas parfaitement à l'horizontale, la bille sur le plateau ne se stabilisera jamais à la position voulue mais s'en approchera. L'équation (2) nous montre que l'instruction est égale à 0 si l'erreur entre la consigne et la mesure est nulle. Selon cette équation, le plateau doit avoir une inclinaison de 0° pour maintenir la bille à la bonne place, or si le système se trouve sur une surface qui est légèrement en pente, le plateau sera également penché bien que l'ordinateur a donné l'ordre au système de placer le plateau à l'horizontale. Par conséquent la bille tentera d'atteindre la consigne mais elle se stabilisera un peu à côté. En d'autres termes, lorsque la bille se trouve loin de la consigne le plateau va s'incliner pour faire rouler la bille jusqu'à la consigne, à ce moment l'erreur entre la mesure et la consigne sera nulle, donc l'instruction le sera aussi et par conséquent le plateau se placera à 0° d'inclinaison. Mais comme le système n'est pas sur un sol parfaitement horizontal, la bille va commencer à s'éloigner de la consigne et donc le système penchera le plateau pour refaire parvenir la bille à la position voulue et là le problème recommence. Le système fini donc par se stabiliser en plaçant la bille légèrement à coté de la consigne.

6.3 La régulation Proportionnelle et Intégrale

L'ajout de la régulation intégrale va permettre d'augmenter les performances du système précédent. Le régulateur intégral a pour but de corriger l'erreur statique qui est un des défauts de la régulation proportionnelle. Pour ce faire, le régulateur va prendre en compte la durée de l'écart entre la consigne et la mesure. Par conséquent le sytème va, plusieurs dizaines de fois par seconde, additionner à une variable du programme la valeur de l'écart. Cette variable contient la somme de toutes les erreurs qui ont eu lieu depuis l'allumage du système. Finalement le programme additionne le résultat de la régulation proportionnelle avec le résultat de la régulation intégrale. On peut donc résumer le régulateur proportionnel et intégral avec l'équation suivante :

$$I = K_p(C - M) + K_i \int_0^t (C - M)dt \tag{3}$$

où K_i est un coefficient qui doit être déterminé expérimentalement.

6.4 La régulation Proportionnelle, Intégrale et Dérivée

Le dernier système de régulation fonctionne avec la dérivée de la position de la bille. En effet, les deux régulateurs présentés précédemment ne prennent pas en compte la vitesse de la bille. Pourtant la vitesse de la bille est un élément très important pour pouvoir réguler sa position. Le programme va calculer la vitesse de la bille en temps réel en additionnant la position de la bille avec sa position précédente et en divisant le résultat obtenu par le temps qui s'est écoulé entre deux images envoyées par la caméra qui filme la bille. Ce temps se situe aux alentours de $0.0\overline{3}$ secondes car la caméra filme à une vitesse de 30 images/secondes. On peut résumer le régulateur proportionnelle, intégrale et dérivée par l'équation suivante :

$$I = K_p(C - M) + K_i \int_0^t (C - M)dt + K_d \frac{dr}{dt} \tag{4}$$

où dr est la distance parcourue par la bille dans un intervalle de temps (dt) qui tend vers 0 et K_d est un coefficient qui doit être déterminé expérimentalement.

7 Mise en place du régulateur PID dans le programme Python

7.1 Transformation des équations

Les équations (2), (3) et (4), sont présentées avec des notions d'intégrales et de dérivées. Les équations ont été formulées ainsi pour les présenter de manière générale dans une notation qui est fréquemment utilisée en physique. Dans ce projet, c'est l'équation (4) qui va être utilisée car elle contient les trois formes de régulation présentées. Cette équation va être traitée par un ordinateur, on peut donc légèrement la modifier pour supprimer l'intégrale et la dérivée :

$$I = K_p(C - M) + K_i S_{erreur} + K_d \frac{M_{precedente} - M}{0.0\overline{3}} \tag{5}$$

où S_{erreur} est une variable du programme qui contient la somme de toutes les erreurs [20] qui ont eu lieu depuis l'allumage du système, M est la position actuelle, mesurée par la caméra, de la bille et $M_{precedente}$ est la position précédente de la bille.
Dans le programme Python, l'équation (5) est présente deux fois car les axes X et Y sont traités séparément, on obtient donc les deux équations suivantes :

$$I_x = K_p(C_x - M_x) + K_i S_{erreurX} + K_d \frac{M_{precedenteX} - M_x}{0.0\overline{3}} \tag{6}$$

$$I_y = K_p(C_y - M_y) + K_i S_{erreurY} + K_d \frac{M_{precedenteY} - M_y}{0.0\overline{3}} \tag{7}$$

Grâce aux équations (6) et (7) nous obtenons les composantes du vecteur I qui permettent de savoir dans quelle direction la bille doit se déplacer. Maintenant, nous devons en déduire la valeur des angles α et β (fig.7, page 11) qui définissent l'inclinaison du plateau.

7.2 Un exemple concret

7.2.1 Détermination des composantes I_x et I_y

Cette section permet d'expliquer le fonctionnement du programme Python à travers un exemple concret. Pour ce faire j'ai lancé une balle de ping-pong sur le plateau du système et j'ai récupéré les différentes données traitées par le programme à un moment donné. En voici une partie :

- Position de la balle en x : $M_x = 176px$
- Position de la balle en y : $M_y = 147px$
- Précédente position de la balle en x : $M_{precedenteX} = 184px$
- Précédente position de la balle en y : $M_{precedenteY} = 160px$

20. L'erreur est définie par la soustraction de la consigne par la position mesurée de la bille.

— Somme de toutes les erreurs en x : $S_{erreurX} = 6179px$
— Somme de toutes les erreurs en y : $S_{erreurY} = 2333px$

Il est important de préciser que les valeurs ci-dessus sont définies en pixels car elles ont été mesurées à partir des images de la caméra située au-dessus du plateau. Il faut également rappeler que l'origine du système d'axes se situe en haut à gauche du champ de vision de la caméra, de plus l'axe des y est dirigé vers le bas. La figure 21 (page 31) est une vue schématique de ce que la caméra filme. Dans cet exemple, la balle doit se stabiliser au milieu du plateau par conséquent la consigne sera définie ainsi :

— Position de la consigne en x : $C_x = 240px$
— Position de la consigne en y : $C_y = 240px$

La caméra est réglée pour filmer avec une résolution de $480px$ par $640px$, l'image est ensuite recardée pour avoir une résolution de $480px$ par $480px$. Etant donné que la caméra se situe exactement au-dessus du plateau, le centre du plateau correspond au centre de l'image, donc le centre du plateau se situe à la position ($240px$;$240px$) de l'image (voir figure 21, page 31).

Pour cet exemple, les coefficients K_p, K_i et K_d ont les valeurs suivantes :

— Coefficient proportionnel : $K_p = 5$
— Coefficient intégrale : $K_i = 0.1$
— Coefficient dérivée : $K_d = 4.5$

Ces coefficients ont été déterminés expérimentalement. Ces valeurs permettent au système de stabiliser la balle lorsqu'elle a une vitesse initiale modérée ou nulle. Cependant la détermination et l'étude de ces coefficients sera étudiée de manière plus approfondie dans un futur chapitre de ce travail.

Maintenant le programme peut utiliser les équations (6) et (7) pour calculer I_x et I_y :

$$\begin{aligned} I_x &= K_p(C_x - M_x) + K_i S_{erreurX} + K_d \frac{M_{precedenteX} - M_x}{0.0\overline{3}} \\ &= 5 * (240 - 176) + 0.1 * 6179 + 4.5 * \frac{184 - 176}{0.0333} \\ &\approx 2018.98 \end{aligned} \tag{8}$$

$$\begin{aligned} I_y &= K_p(C_y - M_y) + K_i S_{erreurY} + K_d \frac{M_{precedenteY} - M_y}{0.0\overline{3}} \\ &= 5 * (240 - 147) + 0.1 * 2333 + 4.5 * \frac{160 - 147}{0.0333} \\ &\approx 2455.05 \end{aligned} \tag{9}$$

Les valeurs obtenues sont ensuite divisées par 10000, cette opération a pour but de pouvoir utiliser des coefficients plus petits et donc plus agréables à régler. Il serait donc possible de garder I_x et I_y sans faire la division par 10000, mais les coefficients K_p, K_i et K_d seraient alors 10000 fois plus grands et donc moins agréables à manipuler.

$$I_x = \frac{2018.98}{10000} \approx 0.2019 \tag{10}$$

$$I_y = \frac{2455.05}{10000} \approx 0.2455 \tag{11}$$

7.2.2 Détermination de l'angle α

Nous avons déterminé les composantes I_x et I_y, maintenant nous pouvons facilement en déduire l'angle α qui correspond à l'angle entre un plan horizontal et et le plan qui contient le plateau du système. La section 3.2 (page 10) expliquait qu'un vecteur $\vec{v}$ est créé et que ce vecteur est toujours perpendiculaire au plateau. On peut donc définir l'angle alpha comme étant l'angle entre la verticale et le vecteur $\vec{v}$. La figure 22 ci-dessous permet de mieux comprendre les liens entre les différents éléments. Il est aussi utile de rappeler que $\|\vec{v}\| = 1$. Le point M présent sur les deux figures représente le centre de la balle qui est sur le plateau.

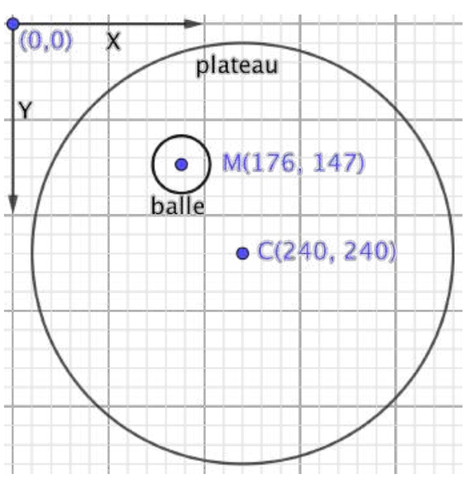


FIGURE 21 – Vu de dessus du problème

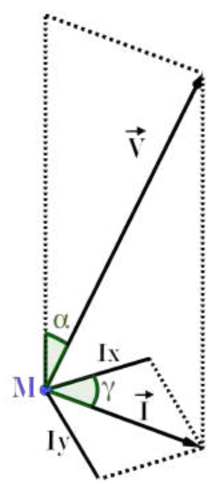


FIGURE 22 – Construction de $\vec{v}$

À l'aide des formules basiques de trigonométrie on peut facilement trouver la relation suivante :

$$sin(\alpha) = \frac{\sqrt{I_x^2 + I_y^2}}{1} \tag{12}$$

Ensuite il suffit d'isoler α :

$$\alpha = arcsin\left(\frac{\sqrt{I_x^2 + I_y^2}}{1}\right) \tag{13}$$

On peut utiliser les valeurs de I_x et I_y trouvées avec les équations (10) et (11) (page 31) :

$$\alpha = arcsin\left(\frac{\sqrt{0.2019^2 + 0.2455^2}}{1}\right) \approx 18.53° \tag{14}$$

Maintenant nous savons que le plateau devra s'incliner de 18.53° par rapport à l'horizontale. Le système fabriqué est capable d'incliner le plateau de 35° au maximum, par conséquent le programme Python contient une condition qui empêche le système de s'incliner de plus de 35°. Autrement dit, l'angle α est fixé par le programme à 35° lorsque l'équation (13) donne une valeur supérieure à 35°. Cet angle est également fixé à 35° lorsque $\sqrt{I_x^2 + I_y^2} > 1$.

7.2.3 Détermination de l'angle β

L'angle β est un peu plus compliqué à trouver que l'angle α. Dans un premier temps nous allons déterminer l'angle γ présent sur la figure 22 (page 31). Cette première étape est assez simple :

$$\gamma = arctan\left(\frac{I_y}{I_x}\right) \tag{15}$$

En utilisant les valeurs de I_x et I_y trouvées avec les équations (10) et (11) (page 31) on obtient :

$$\gamma = arctan\left(\frac{0.2455}{0.2019}\right) \approx 50.56° \tag{16}$$

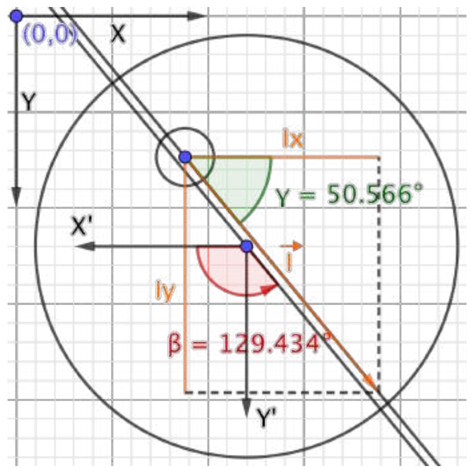


FIGURE 23 – Détermination de l'angle β

Pour des raisons de visibilité, la norme du vecteur $\vec{I}$ a été multiplié par 1000 sur ce schéma.

Les axes *X'* et *Y'* du schéma ci-dessus représentent le repère orthonormé présent sur la figure 5 (page 10).
L'équation (15) donnera toujours un angle compris dans l'intervalle $]-90°; 90°[$, pourtant on a défini dans la section 4.2 (page 18) qu'il faut un angle β compris dans l'intervalle $[0°; 360°[$ pour définir la direction de l'inclinaison du plateau. Il faut alors déduire l'angle β à partir de l'angle γ. Le problème comporte quatre cas de figure, les voici :

— $I_x > 0$ et $I_y \geqslant 0$
— $I_x > 0$ et $I_y \leqslant 0$
— $I_x < 0$ et $I_y \geqslant 0$
— $I_x < 0$ et $I_y \leqslant 0$

Le premier cas de figure correspond à la situation vue dans la figure 23. Les deux composantes sont positives, donc il faut faut faire $180 - \gamma$ pour obtenir l'angle β. Pour les quatre cas de figure il faut faire une opération différente.

— $I_x > 0$ et $I_y \geqslant 0$ $\quad \beta = 180 - |\gamma|$
— $I_x > 0$ et $I_y \leqslant 0$ $\quad \beta = 180 + |\gamma|$
— $I_x < 0$ et $I_y \geqslant 0$ $\quad \beta = |\gamma|$
— $I_x < 0$ et $I_y \leqslant 0$ $\quad \beta = 360 - |\gamma|$

Nous devons ajouter encore deux conditions à la liste ci-dessus car l'équation (15) (page 32) est indéterminée si I_x est nulle :

— $I_x = 0$ et $I_y \geqslant 0$ $\quad \beta = 90$
— $I_x = 0$ et $I_y \leqslant 0$ $\quad \beta = 270$

Avec l'équation (16) (page 32) nous avons calculé que $\gamma = 50.56$. Nous savons grâce aux équations (10) et (11) (page 31) que les composantes du vecteur $\vec{I}$ sont positives. Nous pouvons en déduire que $\beta = 180 - 50.56 = 129.44°$.

7.2.4 Détermination des angles de chaque moteur

Nous avons trouvé les angles α et β qui définissent l'inclinaison et l'orientation du plateau, maintenant le programme doit faire tourner les moteurs pour placer le plateau dans la bonne position. Dans un premier temps, les angles trouvés doivent être arrondis à 0.2 près. La section 4.2 (page 18) expliquait qu'un fichier texte a été créé pour contenir toutes les relations entre l'angle de chaque moteur et les angles α et β. Par conséquent le programme Python qui a déterminé α et β va aller chercher dans le fichier texte la ligne qui commence par "18.6 | 129.4". La ligne trouvée sera la suivante : "18.6|129.4#51.24|37.09|73.18". À gauche du # on retrouve les angles α et β et à droite du # trois nombres correspondent respectivement à l'angle θ de chaque moteur. Pour finir, le programme envoie ces trois angles par USB à l'Arduino qui contrôle le système.

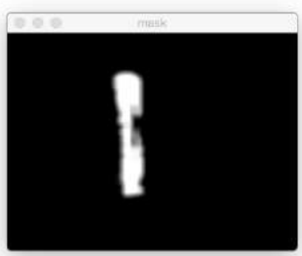
8 Utilisation de OpenCV avec Python

8.1 Qu'est-ce que OpenCV ?

OpenCV (Open Source Computer Vision) est une bibliothèque contenant plus de 2500 d'algorithmes permettant de faire du traitement d'images. OpenCV est la bibliothèque la plus utilisée en ce qui concerne la vision par ordinateur. Cette bibliothèque est disponible pour les langages C, C++ et Python. Pour ce projet nous utiliserons le langage Python car il est particulièrement simple à utiliser.

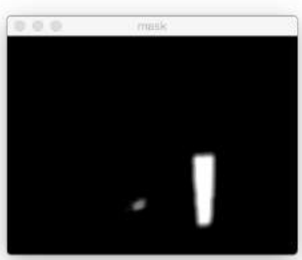
8.2 Programme intermédiaire

Afin de me familiariser avec OpenCV j'ai décidé d'écrire un exemple de programme servant à détecter la position d'un objet d'une certaine couleur. Cet exemple est très important car il sera très utile pour le programme final du projet. L'utilisation de ce code est relativement simple, en effet lorsqu'on lance le programme une fenêtre s'ouvre et affiche en direct les images prises par la webcam branchée à l'ordinateur. Ensuite il suffit de cliquer à l'aide de la souris sur un objet que l'on souhaite et le programme va détecter sa couleur puis entourer cet objet avec un cercle jaune.



(a) Image traitée

(b) Image final



(c) Image traitée

(d) Image final

FIGURE 24 – Capture d'écran du programme intermédiaire en fonctionnement

8.3 Reconnaissance d'une couleur

Grâce à OpenCV, nous pouvons détecter une couleur dans une image relativement facilement. Dans un premier temps il faut savoir que OpenCV utilise l'espace colorimétrique HSV[21], cela veut dire que dans le programme les couleurs seront définies selon trois nombres. Le premier est un nombre compris dans l'intervalle $[0; 179]$ et permet de définir la teinte d'une couleur. Le deuxième et le dernier nombre sont compris dans l'intervalle $[0; 255]$ et permettent de définir respectivement la saturation et la valeur[22] de la couleur. Si par exemple nous voulons que le programme détecte la couleur rouge dans une image, il faut lui donner un intervalle de couleur dans lequel se trouve la couleur que l'on veut isoler. L'intervalle permettra au programme de détecter tous les rouges entre du rouge très clair et du rouge foncé. Pour commencer, nous créons deux variables qui contiennent respectivement le rouge clair et le rouge foncé dans l'espace colorimétrique HSV.

```
rougeClair = np.array([0, 100, 100])
rougeFonce = np.array([0, 250, 250])
```

Ensuite le programme va créer une image en noir et blanc (figure 24) où les zones blanches montrent l'emplacement de la couleur recherchée.

```
imageNoirBlanc = cv2.inRange(imageCouleur, rougeClair, rougeFonce)
```

21. HSV signifie *Hue Saturation Value*.
22. La valeur d'une couleur correspond à sa luminosité.

9 L'interface graphique

9.1 Pourquoi faire une interface graphique ?

Afin d'utiliser le système que j'ai fabriqué avec facilité, j'ai décidé de programmer en Python une interface graphique me permettant de contrôler plusieurs aspects du système.

9.2 Fonctionnalités de l'interface

L'interface est composée de quatre parties bien distinctes :

- La première partie permet de faire des réglages liés à la vidéo prise par la caméra.
- La deuxième partie permet de contrôler les moteurs, et permet également de démarrer le système.
- La troisième partie permet de régler les coefficients K_p, K_i et K_d.
- La dernière partie permet de faire suivre à la balle une forme géométrique.

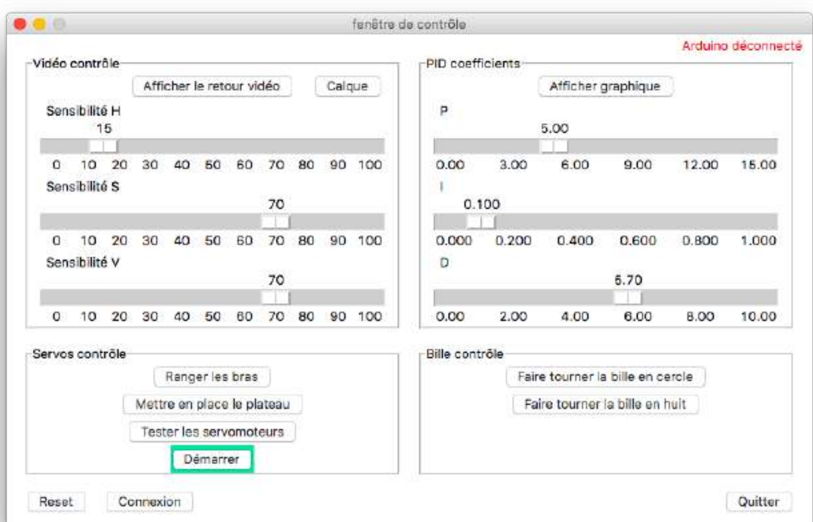


FIGURE 25 – Capture d'écran de l'interface graphique

Les boutons "Afficher le retour vidéo" et "Afficher graphique" permettent d'ouvrir les fenêtres ci-dessous.

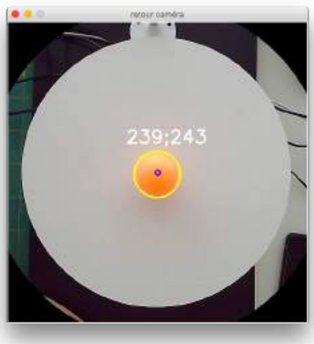


FIGURE 26 – Retour vidéo de la caméra

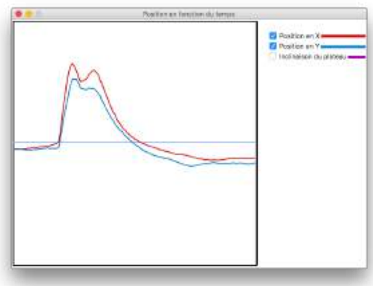


FIGURE 27 – Graphique de la position de la balle en fonction du temps

La fenêtre (figure 26) diffuse les images prises par la caméra. Au centre, nous pouvons y voir la balle de ping-pong orange qui est entouré d'un cercle jaune. Ce cercle permet se rendre compte que le programme détecte le bon objet, de plus le programme rajoute, au-dessus du cercle, les coordonnées de la balle. Au centre du cercle jaune, un point bleu marque l'emplacement de la consigne.
La fenêtre (figure 27) affiche un graphique de la position de la balle en fonction du temps. La ligne rouge représente la position de la balle selon l'axe des x et la ligne bleu représente la position de la balle selon l'axe des y en fonction du temps.

10 Le code Arduino

10.1 Communication avec le programme Python

On peut facilement faire communiquer un programme Python avec un Arduino. La bibliothèque *serial* permet au programme Python d'envoyer des informations à un périphérique USB.

```
ser = serial.Serial(nom_du_port_USB, 9600)
ser.write("information")
```

Dans l'exemple ci-dessus, la commande *write* envoie à l'Arduino le mot "information". De son côté, l'Arduino doit juste lire le port USB de la manière suivante :

```
if (Serial.available() > 0){
        Serial.println(Serial.readString())
}
```

10.2 Contrôle des servomoteurs

Normalement on utilise la fonction *write* pour faire tourner un servomoteur à l'aide d'un Arduino, cependant cette fonction permet de faire tourner le moteur avec une précision de seulement 1°. La section 4.2 (page 18) a permis de calculer les angles que les moteurs doivent avoir pour mettre le plateau dans n'importe quelles positions. Ces angles ont des valeurs précises au centième près et les moteurs doivent avoir ces angles avec précision. Il faut utiliser la fonction *writeMicroseconds* qui permet de faire tourner un servomoteur avec beaucoup de précision. Si nous écrivons *writeMicroseconds(1000)* le servomoteur se placera avec un angle de 0° et si nous écrivons *writeMicroseconds(2000)* le servomoteur se placera avec un angle de 180°. La fonction *writeMicroseconds* est donc moins intuitive à utiliser que la fonction *write* car nous pouvons mettre une valeur en degré dans la fonction *write*. Cependant la fonction *writeMicroseconds* permet d'avoir plus de précision.

11 Analyse des performances du système selon les coefficients P, I et D

11.1 Première expérience : analyse des coefficients P et D

Un des aspects qui permet de définir la qualité d'un régulateur PID est le temps qu'il met à atteindre la consigne souhaitée. Ce temps doit bien entendu être le plus faible possible. L'objectif de cette expérience sera de déterminer le temps qu'il faut au système pour déplacer et stabiliser une balle d'un point sur le plateau jusqu'à son centre en fonction de la masse de la balle et des coefficients utilisés.

11.1.1 Mise en place de l'expérience

Le programme Python du système a été modifié pour réaliser cette expérience, les modifications permettent de contrôler l'expérience et enregistrer des données qui seront analysées après l'expérience. Voici comment se déroule l'expérience :

— Au début de l'expérience le plateau est à l'horizontale et immobile.
— La balle est placée manuellement à un emplacement précis, voir figure 28 page 41.
— En appuyant sur la touche 'a' du clavier de l'ordinateur, le programme Python enclenche le régulateur PID et démarre un chronomètre.
— Le chronomètre s'arrête lorsque la balle est stabilisée au centre du plateau.
— L'expérience s'arrête automatiquement au bout de 10 secondes. Le programme enregistre toutes les coordonnées de la balle qui ont été mesurées pendant l'expérience dans un fichier .xlsx [23]. Ce fichier permettra de faire des graphiques pour analyser les expériences après-coup.
— On peut alors recommencer en modifiant les coefficients P et D et le type de balle.

Une petite cale métallique fixée sur le plateau permet de toujours placer la balle au même endroit au début de chaque essai. La position initiale de la balle correspond toujours aux coordonnées (45 ; 240) (voir figure 28 page 41). En plaçant la balle à la coordonnée 240 on sait que la balle va ensuite se déplacer dans la direction de l'axe X (voir figure 28) ce qui permet de simplifier l'analyse des données à la fin de l'expérience. En effet cela nous permettra de nous concentrer uniquement sur les coordonnées de l'abscisse puisque les coordonnées sur les ordonnées seront toujours égale ou proche de 240.
Pour mener à bien cette expérience il faut également trouver un moyen de définir le moment où on peut affirmer que la balle est stabilisée. Dans tous les cas, le système n'est pas assez précis pour atteindre la consigne sans aucunes erreurs résiduelles c'est pourquoi il a été décidé de déclarer la balle comme étant

23. Le .xlsx est une extension pour tableur.

stabilisée lorsque le système arrive à la maintenir au minimum deux secondes à moins de 20 pixels de la consigne (centre du plateau). Au total, l'expérience dure 10 secondes ce qui est largement suffisant pour stabiliser la balle. En effet si le régulateur PID mettait plus de 10 secondes pour stabiliser la balle, il ne pourrait pas être considéré comme étant efficace pour ce projet.

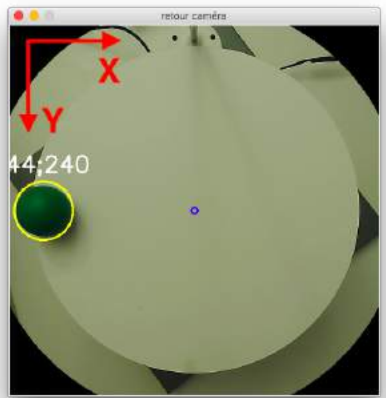


FIGURE 28 – Situation initiale de l'expérience

11.1.2 Réalisation de l'expérience

Pour cette expérience mous allons varier les coefficients P et D mais le coefficient I sera fixé à 0.1. Nous reviendrons sur l'utilité de ce coefficient dans une autre expérience. Le prochain schéma résume les différentes valeurs des coefficients qui vont être utilisées. L'expérience sera réalisée deux fois en utilisant deux balles de masse différentes. La première balle sera une balle de ping-pong de 2.7 grammes et et la deuxième sera une bille en céramique de 19.5 grammes.

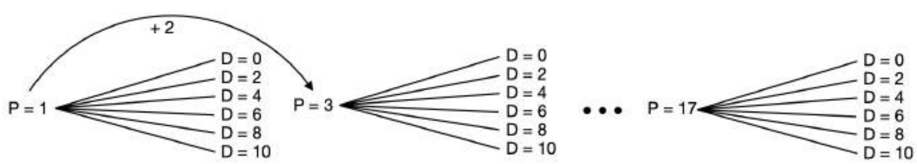


FIGURE 29 – Liste des coefficients utilisés dans l'expérience

A l'issue des ces expériences nous obtiendrons des graphiques comme celui ci-dessous :

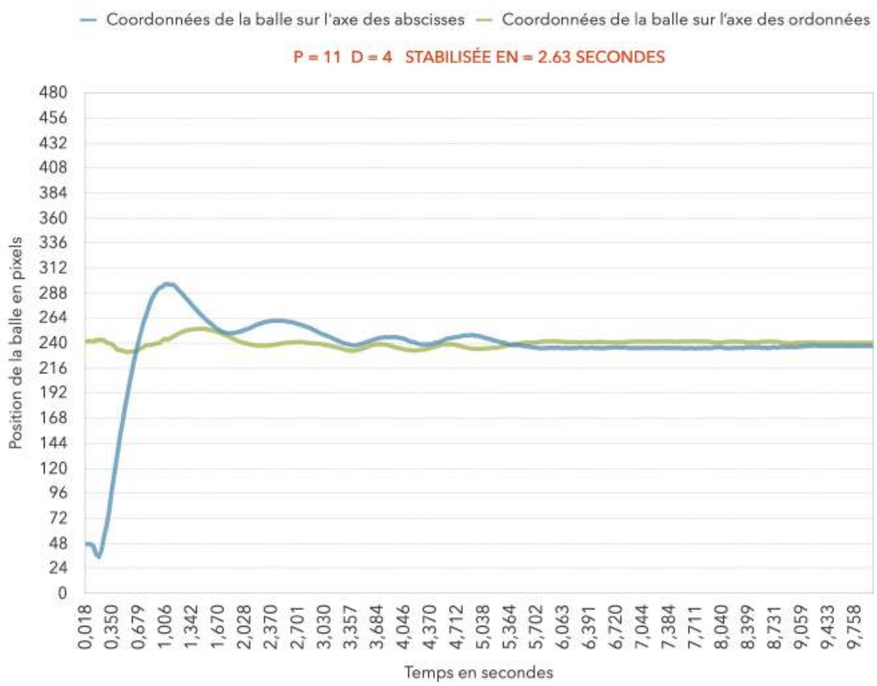


FIGURE 30 – Exemple de graphique obtenu lors de cette expérience

Ce graphique présente la position de la balle en fonction du temps sur une durée de 10 secondes. La courbe verte varie très peu et reste aux alentours de 240 pixels ce qui montre bien que la balle se déplace selon l'axe des abscisses qui correspond à la courbe bleu. Dans les graphiques qui seront présentés dans les résultats, les titres rouges indiqueront que les coefficients utilisés ont permis à la balle de se stabiliser au centre du plateau. Dans cet exemple la balle a été considérée comme étant stable à partir de 2,63 secondes à partir du début de l'expérience.

11.1.3 Résultats de l'expérience

L'ensemble des 54 graphiques résultant des tests effectués en utilisant une balle de ping-pong et différentes combinaisons de coefficients se trouvent dans l'annexe A (page 63). Un extrait des résultats se trouve sur la prochaine page.

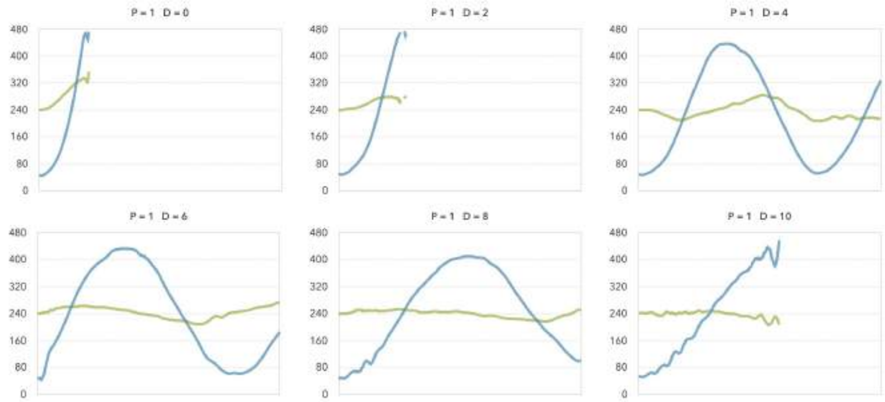


FIGURE 31 – Extrait des résultats de l'expérience avec la balle de ping-pong. L'intégralité des résultats se trouve dans l'annexe A (page 63)

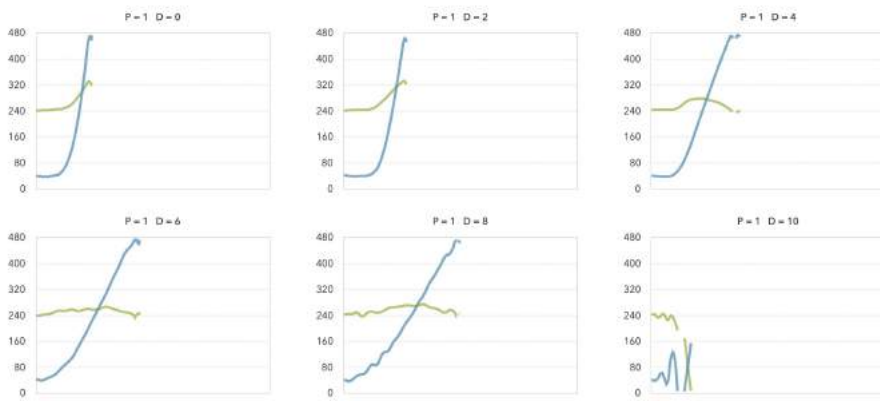


FIGURE 32 – Extrait des résultats de la même expérience mais réalisée avec une balle en céramique d'une masse de 19.5 grammes. L'intégralité des résultats se trouve dans l'annexe B (page 67)

Ci-dessous se trouve la liste des coefficients qui ont permis de stabiliser la balle de ping-pong (liste de gauche) et la balle en céramique (liste de droite). Les trois meilleurs temps de stabilisation T_s de chaque liste sont mis en évidence en couleur.

Coef. P	Coef. D	T_s
5	4	2.73
5	6	6.05
5	8	6.95
7	4	2.16
7	6	3.67
5	8	5.59
5	10	6.91
9	4	1.85
9	6	4.65
11	4	2.63
11	6	0.92
11	8	2.75
11	10	5.74
13	4	1.35
13	6	0.86
13	8	2.69
13	10	4.49
15	4	1.22
15	6	1.94
15	8	2.17
15	10	3.43
17	4	2.38
17	6	1.79
17	8	1.65
17	10	2.41

Coef. P	Coef. D	T_s
5	2	6.03
5	4	5.42
7	2	5.54
7	4	0.99
7	6	6.28
9	2	6.21
9	4	2.44
9	6	4.26
9	8	6.07
11	4	1.35
11	6	0.93
13	4	2.84
13	6	1.02
13	8	2.87
15	4	2.51
15	6	1.65
15	8	2.67
17	4	2.12
17	6	1.62
17	8	2.05

11.1.4 Discussion des résultats

Vingt-six combinaisons de coefficients ont permis de stabiliser la balle de ping-pong mais seulement 20 combinaisons de coefficients ont réussi à stabiliser la balle en céramique. Ce résultat était attendu, en effet la masse plus élevée de la balle en céramique lui confère plus d'inertie que la balle de ping-pong par conséquent il est plus difficile pour le système d'annuler ou de réduire ses déplacements. La balle ayant plus d'énergie, elle est moins affectée par la pente du plateau qui tente de s'opposer à sa vitesse par exemple. Selon ce raisonnement on comprend pourquoi moins de combinaisons de coefficients ont arrêté la balle en céramique.

On observe certaines similarités entre les deux tableaux ci-dessus, en effet on constate que les trois meilleurs temps de chaque tableau ont été réalisés avec un coefficient D valant 4 ou 6 et deux des trois meilleurs temps de chaque tableau sont obtenus avec les coefficients P = 11 ou P = 13. Il même assez étonnant de voir que la combinaison P = 11 et D = 6 a stabilisé les deux balles quasiment avec le même temps. Néanmoins il peut s'agir d'une coïncidence puisqu'il faudrait re-tester toutes les combinaisons de coefficients plusieurs fois pour faire une moyenne des temps de stabilisation et ainsi obtenir des données plus exactes. La combinaison P = 11 et D = 6 a été re-testée cinq fois pour chaque balle dans le tableau ci-dessous :

T_s Balle de ping-pong	T_s Balle en céramique
2.80	1.09
0.89	2.25
4.91	3.07
0.99	1.12
4.82	1.06

Ce tableau montre que les temps obtenus pour une même balle varient de plusieurs secondes lorsqu'on répète l'expérience plusieurs fois de suite. Il est donc impossible de considérer ces valeurs comme étant exploitables car les résultats varient trop d'un essai à l'autre. Il y des écart de plus de quatre secondes entre deux essais avec la même balle. Bien que les temps de stabilisation soient difficilement exploitables, les graphiques semblent beaucoup plus cohérents. Les prochains graphiques correspondent aux essais du tableau ci-dessus.

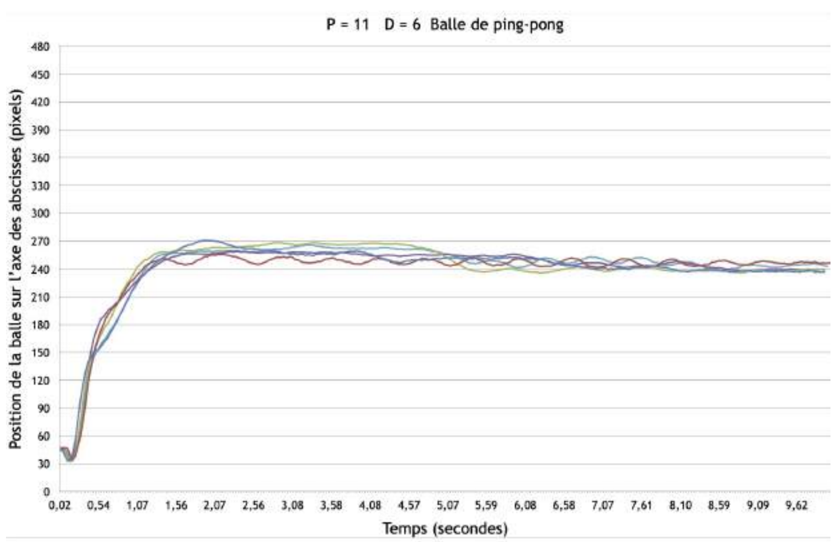


FIGURE 33 – Nouveaux essais de la combinaison P = 11 et D = 6 sur la balle de ping-pong

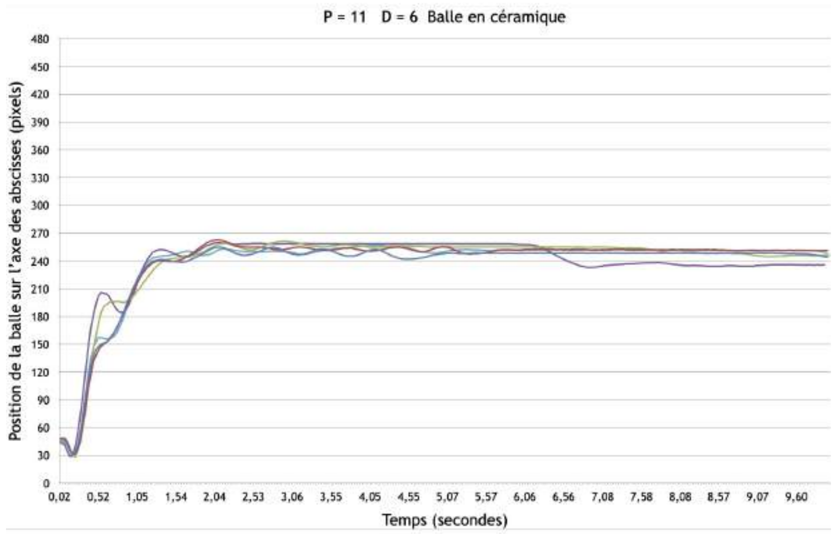


FIGURE 34 – Nouveaux essais de la combinaison P = 11 et D = 6 sur la balle en céramique

Les différentes courbes sont très similaires pourtant nous avons vu que les temps de stabilisation qui en sont déduit sont très variables. On peut remettre en question le critère qui a été défini au début de l'expérience pour affirmer qu'une balle est stabilisée. *" Il a été décidé de déclarer la balle comme étant stabilisée lorsque le système arrive à la maintenir au minimum deux secondes à moins de 20 pixels de la consigne (centre du plateau) ", page 40.*
En conclusion cette expérience a tout de même permis de déterminer les coefficients permettant de stabiliser la balle. Même s'il n'a pas été possible de quantifier précisément le temps de stabilisation de la balle par chaque combinaison de coefficients, les graphiques donnent une bonne idée des meilleurs coefficients à utiliser.

11.2 Deuxième expérience : analyse du coefficient I

Dans l'expérience précédente nous avons fixé le coefficient I à 0.1. Maintenant nous allons faire varier ce paramètre en gardant constant les coefficients P et D. Ainsi nous verrons en quoi il peut participer à l'amélioration du régulateur PID. Cette expérience sera réalisée avec les mêmes balles que l'expérience précédente et nous utiliserons uniquement les coefficients P = 11 et D = 6 car nous avons vu que cette configuration fonctionnait avec les deux types de balle même s'il n'a pas été prouvé qu'il s'agissait de la combinaison la plus efficace. Au début de l'expérience la balle est placée au même endroit que dans l'expérience précédente, à partir du démarrage du régulateur PID le programme enregistre les coordonnées de la balle pendant dix secondes exactement comme dans la première expérience. Pour chaque balle on testera les coefficients I = 0.0, I = 0.1, I = 0.2, I = 0.3, I = 0.4 et I = 0.5 .

11.2.1 Résultats pour la balle de ping-pong

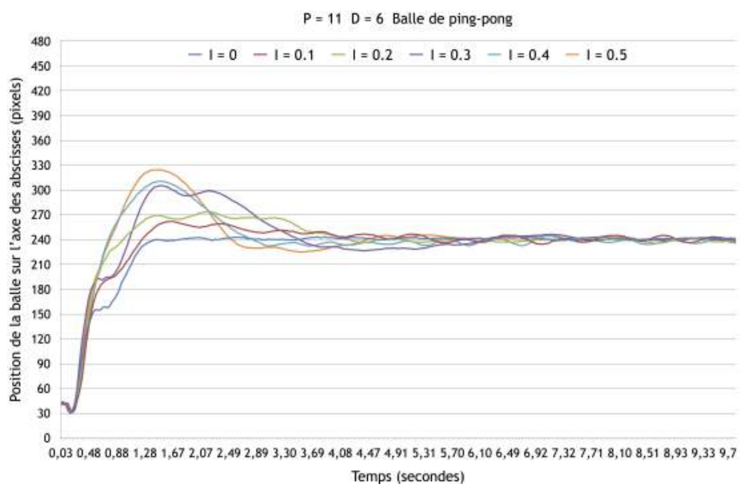


Figure 35 – Expérimentation du coefficient I sur la balle de ping-pong

11.2.2 Résultats pour la balle en céramique

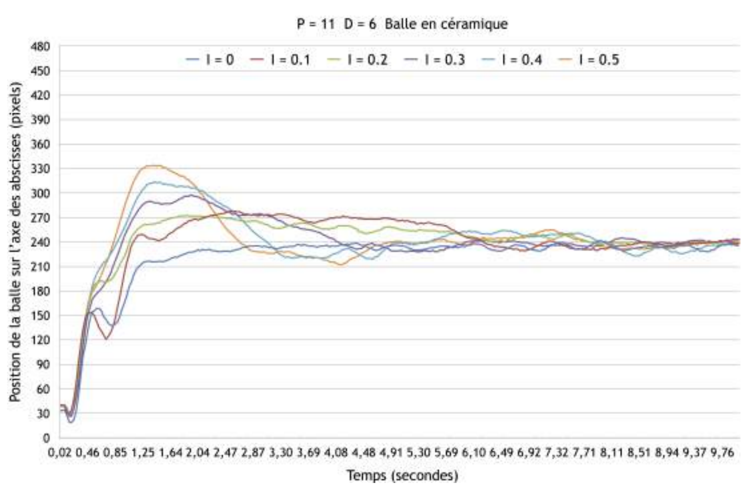


Figure 36 – Expérimentation du coefficient I sur la balle en céramique

11.2.3 Discussion des résultats

Tout comme dans la première expérience on peut voir que le système a toujours un peu plus de peine à stabiliser la balle ayant une masse plus élevée. On voit que les courbes sont plus rapprochées et lissées dans la figure 35 que dans la figure 36. Cependant dans les deux cas, on observe que la courbe qui se rapproche le plus rapidement de la coordonnée 240 est la courbe bleu qui a été produite avec un coefficient I nul. Peu importe la balle, plus ce coefficient est élevé plus la balle dépasse la consigne avant de stabiliser proche de celle-ci. Ces résultats montrent que le coefficient I a un effet négatif sur le régulateur PID puisqu'il augmente le dépassement de la consigne et augmente également le temps de stabilisation de la balle. Toutefois ce coefficient peut améliorer la précision du régulateur dans certaine situation. En effet il permet d'annuler l'erreur statique qui peut apparaître si le système est soumis à certaines contraintes extérieures comme par exemple du vent. On peut également imaginer que le système repose sur une table qui n'est pas parfaitement horizontale ce qui serait également une source d'erreur statique. Pour démontrer l'utilité du coefficient I dans cette situation, le système à été incliné d'environ trois degrés et des tests ont été effectués avec les coefficients I = 0 et I = 0.1 . La figure suivante dévoile très clairement la présence de l'erreur statique lorsque le coefficient I est nul.

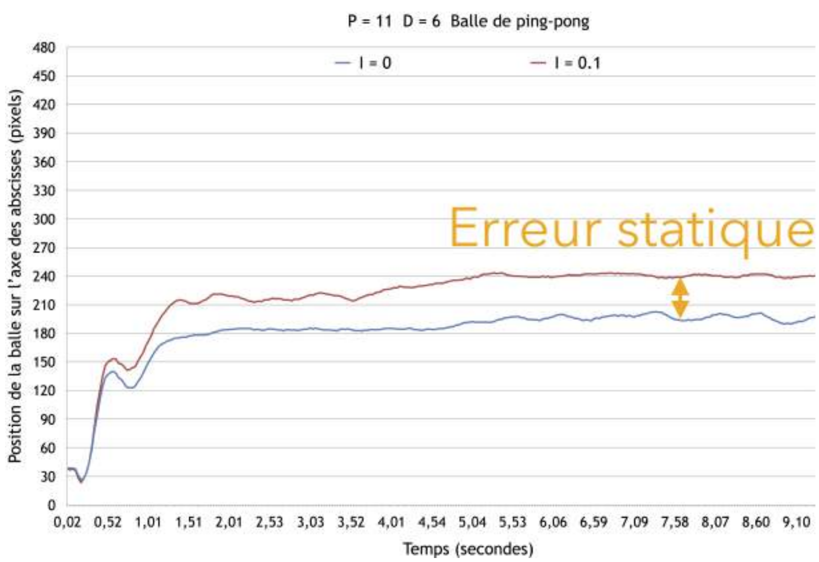


FIGURE 37 – Expérimentation du coefficient I lorsque le système est incliné de 3 degrés

En conclusion il semblerait que le coefficient I diminue les performances du régulateur lorsque le système est placé dans un environnement idéal. Cependant ce coefficient peut se révéler indispensable à la précision du système en présence d'éléments perturbateurs extérieurs comme du vent ou un support penché.

11.3 Dernière expérience : analyse des coefficients P et D lorsque la balle a une vitesse initiale non nulle

Cette expérience sera similaire à la première à la différence près que cette fois la balle ne sera pas posée manuellement sur le plateau mais sur une rampe. La balle descendra cette rampe pour arriver sur le plateau du système avec une certaine vitesse initiale. Les tests seront effectués avec les deux balles habituelles, le coefficient I sera fixé à 0.1 et les combinaisons de coefficients P et D seront les mêmes que dans la première expérience.

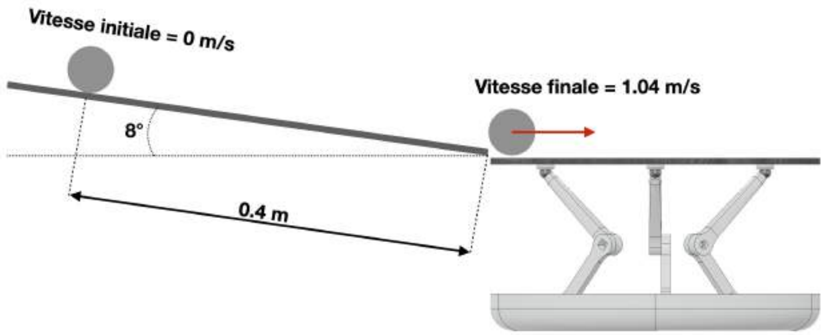


FIGURE 38 – Schéma de l'installation utilisée pour l'expérience

La vitesse de la balle à son arrivée sur le plateau du système a été calculée en sachant que l'énergie potentielle de la balle en haut de la rampe est égale à son énergie cinétique en bas de la rampe.

$$mgsin(8)0.4 = \frac{1}{2}mv^2 \tag{17}$$

$$v = \sqrt{gsin(8)0.8} = \sqrt{9.81sin(8)0.8} = 1.04\,\mathrm{m/s} \tag{18}$$

11.3.1 Résultats de l'expérience

La prochaine page contient un extrait des graphiques résultants des 54 tests effectués avec chaque balle. Il est à noter que les résultats ne sont pas présentés de la même façon que dans la première expérience. Premièrement la courbe qui correspond à la position de la balle sur l'axe des ordonnées n'est pas tracée puisqu'elle est toujours très rapprochée de la coordonnée 240, ce qui est normal puisqu'on fait rouler la balle uniquement dans la direction de l'axe de abscisse du système (voir figure 28, page 41). Les résultats de chaque tests sur la balle de ping-pong et la balle en céramique seront affichés sur le même graphique pour pouvoir comparer les résultats plus facilement. La couleur verte correspond à la balle en céramique et la couleur bleu à la balle de ping-pong.

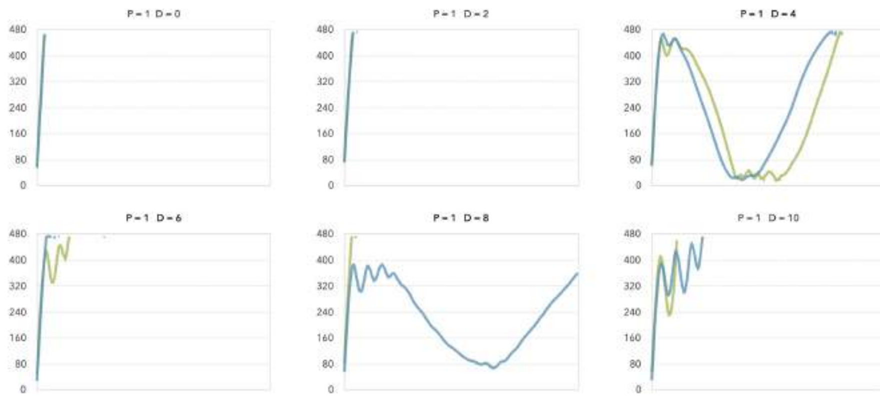


FIGURE 39 – Extrait des résultats de l'expérience. L'intégralité des résultats se trouve dans l'annexe C (page 71)

11.3.2 Discussion des résultats

Comme vous avez pu le constater, les temps de stabilisation des balles n'ont pas été pris en compte pour cette expérience car nous avions montré dans la première expérience que ces temps ne constituaient pas un critère assez précis pour comparer les résultats. Toutefois les résultats des deux balles étant affichés sur le même graphique, on peut facilement déterminer des conclusions intéressantes. Cette expérience montre à nouveau que le régulateur est plus efficace sur la balle la plus légère.

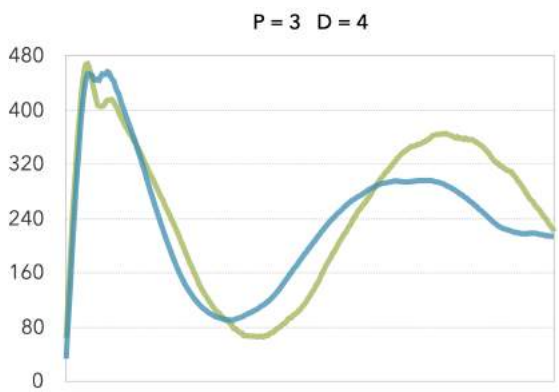


FIGURE 40 – Comparaison de l'efficacité du régulateur selon la balle utilisée.
vert : balle en céramique bleu : balle de ping-pong

Ce graphique présenté dans les résultats de l'annexe C (page 71) est particulièrement parlant puisqu'il montre deux courbes ayant une oscillation très similaire mais légèrement décalée dans le temps. En effet on remarque que la courbe bleu qui correspond à la position de la balle de ping-pong est "en avance" sur la courbe verte. En d'autres termes, les déplacements de la balle de ping-pong sont plus rapidement affectés par les mouvements du plateau du système car l'inertie de cette balle est moindre que l'inertie de la balle en céramique (courbe verte). On constate également que dans l'ensemble des résultats, la balle de ping-pong est tombée du plateau 30 fois alors que la balle en céramique est tombée 45 fois. À nouveau cela prouve le problème d'inertie qui est rencontré avec la balle de masse supérieure.

12 Discussion des trois expériences

Ces trois expériences ont permis d'identifier les différents comportements du régulateur en fonction de la masse de la balle utilisée et des combinaisons de coefficients. Nous avons observé que certaines combinaisons de coefficients étaient plus efficaces que d'autre et que parfois elles permettaient de stabiliser un seul type de balle. La première et la dernière expérience ont permis de montrer que le système réagit différemment en fonction de la vitesse initiale de la balle. Nous avons également expérimenté le coefficient I dont l'utilisation peut être discutée, en effet nous avons vu qu'il abaissait les capacités du régulateur dans un environnement contrôlé mais qu'il pouvait se révéler utile lorsque certains éléments perturbateurs et extérieurs au système sont présents comme par exemple le vent ou un support incliné.
La première expérience a montré qu'il n'a pas été possible d'établir un critère assez précis pour quantifier le temps requis à une combinaison de coefficients pour stabiliser la balle utilisée. Par conséquent aucune expérience n'a permis de déterminer objectivement la combinaison de coefficients optimale à la réussite de l'expérience en question. Malgré tout, les graphiques permettent de déterminer assez facilement les combinaisons qui ont abouties à une stabilisation de la balle. Quoi qu'il en soit, il est très difficile de trouver une combinaison qui satisfait toutes les expériences, en effet même si la combinaison la plus optimale avait été trouvée pour la première expérience, elle n'aurait pas été la plus optimale pour la troisième expérience dans laquelle la balle avait une vitesse initiale non nulle. Par conséquent il n'est pas véritablement possible de trouver une combinaison de coefficients parfaite puisqu'en réalité le réglage des coefficients dépend de l'utilisation du système. Si le système est destiné à stabiliser une balle dont la vitesse initiale est nulle, alors il faudra utiliser la première expérience pour trouver la bonne combinaison de coefficients mais si le système doit être capable de stabiliser la même balle avec un grand nombre de vitesses initiales possibles alors il faudra trouver un compromis entre les résultats de la première et dernière expérience. Selon les expériences réalisée, on remarque que la combinaison $P = 13$ et $D = 6$ a très bien fonctionné dans la première et la dernière expérience. On peut alors choisir cette combinaison si l'on veut que le système soit assez polyvalent pour stabiliser une balle peu importe sa vitesse initiale tant que cette dernière reste en dessous de 1.04 m/s. En effet il faudrait

encore faire des expériences pour vérifier si cette combinaison peut stabiliser une balle lancée à une vitesse supérieure.

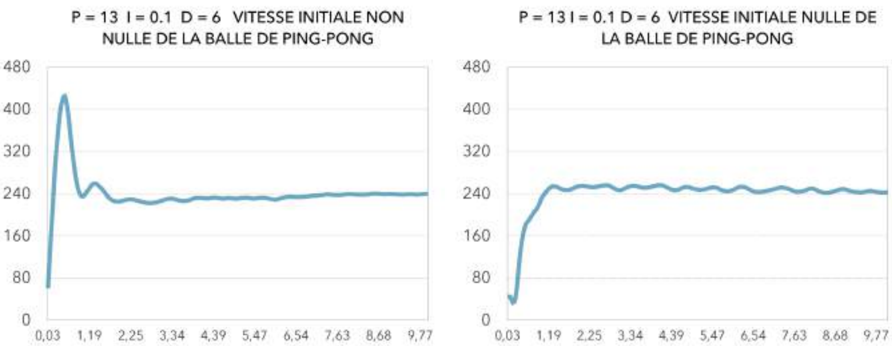
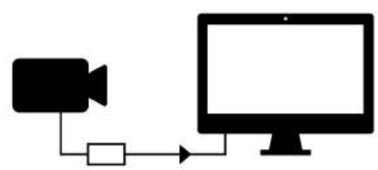


FIGURE 41 – Comparaison des résultats de la première et de la dernière expériences avec la combinaison P = 13 et D = 6

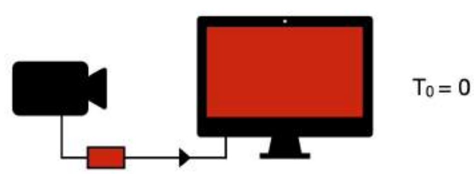
13 Faiblesses du système

13.1 Temps de détection de la balle

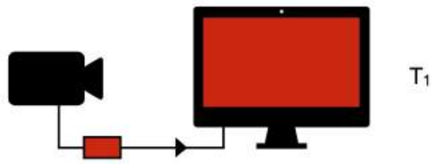
Tout régulateur PID contient un capteur qui permet de mesurer une grandeur réelle. Dans notre cas ce capteur est une caméra qui a pour but de filmer la balle, pour ensuite permettre à un programme d'en déduire les coordonnées de la position de la balle. Etant donné que la régulation se déroule en temps réel, le capteur doit fournir ses mesures dans le plus court laps de temps possible si l'on veut pouvoir corriger l'erreur le plus rapidement possible. Dans l'expérience qui va suivre, l'objectif est de mesurer le temps qui s'écoule entre le moment où la balle apparaît dans le champs de vision de la caméra et le moment où le programme Python déduit de l'image la position de cette balle. Pour exécuter cette expérience il a fallu créer un programme et une installation pour mesurer la latence que l'on cherche. L'expérience fonctionne en plaçant la caméra en face de l'écran de l'ordinateur sur lequel elle est branchée. Ce dernier exécute un programme qui agit en trois temps : au lancement du programme l'écran de l'ordinateur devient entièrement blanc, quelques secondes plus tard l'écran devient spontanément rouge, et finalement le programme qui analyse les images de la caméra en direct détecte la couleur rouge. Le programme peut ensuite facilement mesurer le temps qui s'est écoulé entre le moment ou il a affiché la couleur rouge et le moment où il a détecté cette couleur sur les images reçues par la caméra.



(a) Situation initiale : la caméra filme l'écran blanc et est branchée à l'ordinateur



(b) Le programme affiche du rouge et démarre un chronomètre. L'écran rouge simule l'apparition d'une balle rouge dans le champs de vision de la caméra



(c) Le programme arrête le chronomètre lorsqu'il détecte la couleur rouge sur une image reçue par la caméra

FIGURE 42 – Mesure du temps de détection de la balle par le système

L'expérience a été réalisée dix fois car les résultats semblaient varier. Le tableau ci-dessous présente les résultats obtenus.

Temps [en secondes]
0.1478
0.1920
0.1553
0.1552
0.1881
0.1578
0.1645
0.1557
0.1560
0.1657
Moyenne : 0.1638

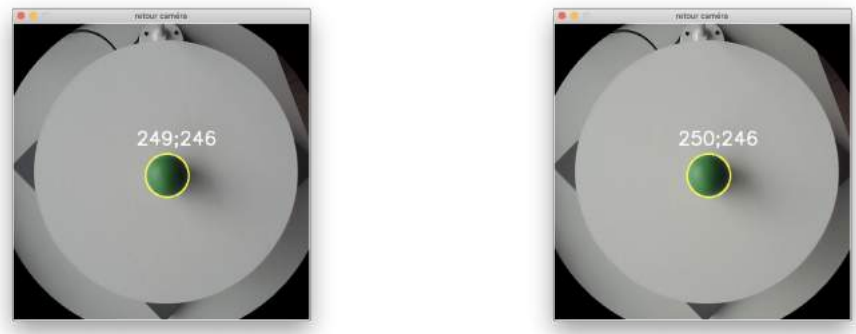
Grâce à cette expérience on observe que le système prend en moyenne 0.1638 secondes pour s'apercevoir de l'apparition de la balle dans le champ de vision de la caméra. Cela veut dire que les coordonnées de la balle mesurées par le programme sont en retard de 0.1638 secondes par rapport à la réalité. Ce délai est causé par deux événements : premièrement le programme doit récupérer l'image prise par la caméra puis dans un deuxième temps il doit la traiter pour y trouver la couleur de la balle et en déduire les coordonnées qui correspondent à sa position. En améliorant le programme utilisé dans cette expérience on peut également mesurer le temps qu'il faut au programme pour traiter l'image et trouver les coordonnées de la balle.

Traitement de l'image [en secondes]
0.0122
0.0590
0.0253
0.0300
0.0272
Moyenne : 0.0307

Ce tableau nous montre que le programme met moins d'un dixième de seconde pour déterminer si une image contient la couleur de la balle. On comprend alors que la latence de 0.1638 secondes vue précédemment n'est presque pas causée par le traitement de l'image, mais plutôt par le temps qui s'écoule entre la prise d'une image par la caméra et le moment où le programme récupère cette même image avant de l'analyser. Pour tenter de diminuer cette latence il faudrait utiliser une autre caméra et améliorer les performances du programme. Cependant les expériences de la section 11 montrent que le système fonctionne malgré l'existence de cette latence.

13.2 Précision de la détection de la balle

Tout régulateur PID a besoin d'un capteur pour fonctionner, dans notre cas il s'agit d'une caméra. Le problème de ce choix est qu'il s'agit d'un capteur très complexe qui est complètement dépendant des conditions de luminosité qui l'entourent. Dans notre cas ce capteur doit pouvoir détecter une balle de couleur intrinsèque uniforme, cependant la couleur réelle de la balle est fortement perturbée par la présence d'ombre et de reflets. Les reflets sont liés à la texture de la balle, en utilisant une couleur mate on peut facilement les réduire. Les ombres proviennent des conditions d'éclairage dans lesquels se trouve le système, il est alors beaucoup plus difficile de les éliminer. Heureusement le programme Python est tout de même capable de déterminer les coordonnées de la balle car on lui donne un intervalle de couleurs à trouver dans l'image. Dans les images de la page suivante cet intervalle va du vert foncé qui correspond aux zones ombragées de la balle jusqu'au vert claire qui correspond aux reflets à la surface de la balle.



(a) Coordonnées mesurées (249 ;246)   (b) Coordonnées mesurées (250 ;246)

FIGURE 43 – Incertitude dans la mesure de la position de la balle à vitesse nulle

Ces deux captures d'écran ont été faites à moins d'une seconde d'intervalle alors que le plateau et la balle était immobile (Le système était hors tension pour être sûr que le plateau reste immobile). On peut voir que le système a mesuré deux coordonnées différentes alors que l'ensemble du système était immobile. Cela peut être dû à l'éclairage naturel qui varie en permanence et qui modifie légèrement les reflets et ombres sur la balle. De plus chaque image prise par la caméra contient du bruit numérique qui varie d'une image à l'autre ce qui perturbe la mesure des coordonnées. Le problème de ces imprécisions est que le système pense voir une balle en mouvement alors qu'elle ne l'est pas, cependant l'imprécision de la mesure est très rarement au dessus de $\pm$ 2 pixels ce qui reste acceptable. Néanmoins comme le système perçoit toujours la balle en mouvement même lorsqu'elle ne l'est pas, le système ne trouve jamais l'équilibre parfait.

13.3 Vitesse des servomoteurs

Le couple et la vitesse des servomoteurs sont directement liés à la tension qu'on applique à leurs bornes. Selon la fiche technique des servomoteurs qui ont été choisis pour ce projet, ils ont besoin d'au minimum 4.8 volts pour fonctionner et atteignent leur capacité maximale à 6 volts. Le système fonctionne actuellement avec une alimentation 5 volts, il serait alors possible de la remplacer par une alimentation de 6 volts pour que les servomoteurs puissent atteindre leur vitesse maximale lors des mouvements rapides du plateau.

13.4 Effet néfaste de la rotation du plateau sur la vitesse de la balle

Lorsque le système se trouve dans la configuration de la figure suivante le plateau doit effectuer une rotation dans le sens horaire pour éviter que la balle roule et tombe du plateau. Le système a été conçu pour que le centre de rotation du plateau soit toujours au centre de celui-ci, malheureusement ce choix a un effet secondaire sur la vitesse de la balle. En effet la figure suivante montre que la rotation du plateau projette la balle vers l'extérieur du système, en d'autres termes elles est accélérée par le plateau en dehors de ce dernier.

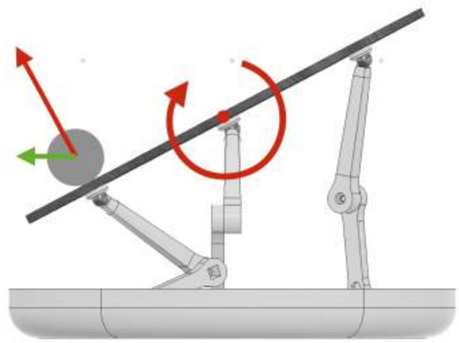


FIGURE 44 – Accélération de la balle dû à la rotation du plateau

La géométrie du mécanisme qui déplace le plateau pourrait permettre de déplacer le centre de rotation du plateau en dessous de la balle, ainsi la balle ne serait pas involontairement accélérée par les rotations du plateau qui doivent justement stabiliser la balle.

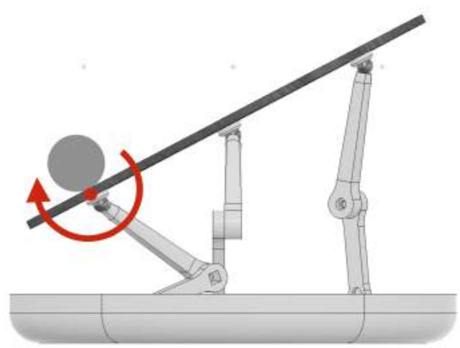


FIGURE 45 – Centre de rotation du plateau sous la balle

14 Conclusion

Pour conclure, ce travail de maturité a eu un impact positif et m'a beaucoup appris. Ce projet a été l'occasion de réaliser un projet concret qui contient une partie hardware et une partie software. Au cours de ce travail j'ai été confronté à plusieurs problèmes techniques que j'ai dû surmonter. Certains de ces problèmes ont eu lieu pendant la construction du système et d'autres problèmes sont survenus pendant la programmation du système. J'ai dû trouver des solutions quitte à devoir faire quelques concessions. La partie matérielle de ce projet est tout aussi importante que la partie informatique, en effet la modélisation et la fabrication du système m'a demandé autant de temps que la programmation de l'ensemble.
Malgré les difficultés rencontrées, le système est capable d'accomplir sa tâche comme les multiples expériences ont permis de le démontrer. La dernière section de ce travail a aussi permis de lister un certain nombre de limitations de ce système. Nous avons vu que par exemple si la balle est lancée trop rapidement le système réagira trop lentement et la bille tombera du plateau. Les expériences ont permis de trouver des combinaisons de coefficients qui ont permis de régler le régulateur PID pour que le système soit polyvalent vis-à-vis de la vitesse initiale de la balle.
Une vidéo de présentation de mon travail est disponible à l'adresse suivante : `https://www.youtube.com/watch?v=57DbEEBF7sE`

Bibliographie

[1] abcclim. Principe de rlation p-pi-pid, 2018 (consult juin 2018). https ://www.abcclim.net/regulation-p-pi-pid.html.

[2] Kar Anirban. Opencv object tracking by colour detection in python, 2017 (consult juin 2018). https ://thecodacus.com/opencv-object-tracking-colour-detection-python/#.W1cKZC30lp-.

[3] maike hao. Linux and python : auto-detect arduino serial port [duplicate], juillet 2012 (consult avril 2018). https ://stackoverflow.com/questions/11364879/linux-and-python-auto-detect-arduino-serial-port.

[4] Mark. A very nonlinear system of three equations, septembre 2013 (consult avril 2018). https ://ask.sagemath.org/question/10506/a-very-nonlinear-system-of-three-equations/.

[5] Ferdinand Piette. Implnter un pid sans faire de calculs !, aot 2011 (consult juillet 2018). http ://www.ferdinandpiette.com/blog/2011/08/implementer-un-pid-sans-faire-de-calculs/.

[6] Adrian Rosebrock. Ball tracking with opencv, septembre 2015 (consult juin 2018). https ://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/.

[7] The scientificsentence. Modes de rlation, 2007 (consult juillet 2018). http ://scientificsentence.net/Regulation/Regulation5.html.

[8] Sparkfun. Sparkfun atmega32u4 breakout, 2015 (consult avril 2018). https ://www.sparkfun.com/products/retired/11117.

[9] user3704293. How to split a string using a specific delimiter in arduino ?, avril 2015 (consult juillet 2018). https ://stackoverflow.com/questions/29671455/how-to-split-a-string-using-a-specific-delimiter-in-arduino.

[10] Wikipedia. Pid controller, juillet 2018 (consult juillet 2018). https ://en.wikipedia.org/wiki/PID_controller.

Table des figures

1 Photo d'un Arduino Uno 6
2 Comparaison entre un robot sériel et parallèle 7
3 Comparaison entre des actionneurs linéaires et rotatifs dans un robot delta 8
4 Exemples de plateforme de Stewart 9
5 Disposition des moteurs 10
6 Bras d'un moteur 10
7 Vecteur $\vec{v}$ perpendiculaire au plateau 11
8 Photo d'un servomoteur utilisé pour ce projet 12
9 Capture d'écran de mon projet dans *Fusion 360* 13
10 Vue générale de tous les éléments du système 14
11 Pièces imprimées en 3D 15
12 Début de l'assemblage des pièces 16
13 Capture d'écran du support de la caméra dans *Fusion 360* 17
14 Caméra utilisée pour ce projet *référence : ELP-USBFHD01M* 17
15 Capture d'écran du fichier texte 18
16 Schéma du circuit imprimé 22
17 Circuit imprimé 23
18 Soudage du microcontrôleur sur le PCB 24
19 Initialisation du microcontrôleur 25
20 Schéma d'un régulateur PID 26
21 Vu de dessus du problème 31
22 Construction de $\vec{v}$ 31
23 Détermination de l'angle β 33
24 Capture d'écran du programme intermédiaire en fonctionnement 35
25 Capture d'écran de l'interface graphique 37
26 Retour vidéo de la caméra 38
27 Graphique de la position de la balle en fonction du temps 38
28 Situation initiale de l'expérience 41
29 Liste des coefficients utilisés dans l'expérience 41
30 Exemple de graphique obtenu lors de cette expérience 42
31 Extrait des résultats de l'expérience avec la balle de ping-pong. L'intégralité des résultats se trouve dans l'annexe A (page 63) . . 43
32 Extrait des résultats de la même expérience mais réalisée avec une balle en céramique d'une masse de 19.5 grammes. L'intégralité des résultats se trouve dans l'annexe B (page 67) 43
33 Nouveaux essais de la combinaison P = 11 et D = 6 sur la balle de ping-pong 45
34 Nouveaux essais de la combinaison P = 11 et D = 6 sur la balle en céramique 46
35 Expérimentation du coefficient I sur la balle de ping-pong 47
36 Expérimentation du coefficient I sur la balle en céramique 47

37 Expérimentation du coefficient I lorsque le système est incliné de 3 degrés 48
38 Schéma de l'installation utilisée pour l'expérience 49
39 Extrait des résultats de l'expérience. L'intégralité des résultats se trouve dans l'annexe C (page 71) 50
40 Comparaison de l'efficacité du régulateur selon la balle utilisée. . . 50
41 Comparaison des résultats de la première et de la dernière expériences avec la combinaison P = 13 et D = 6 52
42 Mesure du temps de détection de la balle par le système 54
43 Incertitude dans la mesure de la position de la balle à vitesse nulle 56
44 Accélération de la balle dû à la rotation du plateau 57
45 Centre de rotation du plateau sous la balle 57

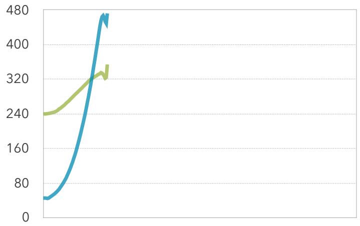
Annexes

A Résultats de la section 11.1.3 63
B Suite des résultats de la section 11.1.3 67
C Résultats de la section 11.3.1 71
D solveEquation.py 75
E programmeIntermediaire.py 78
F interface.py 80
G arduinoCode.ino 92
H Détermination des angles θ des servomoteurs 94
I Quelques plans du projet 97
J Prototypes 104
K Quelques esquisses du projet 105

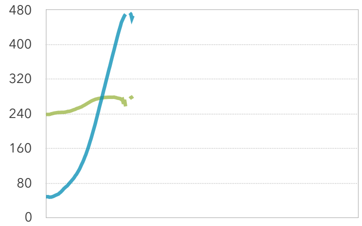
A Résultats de la section 11.1.3

Résultats de l'expérience sur l'effet des coefficients P et D sur la stabilisation d'une balle de ping-pong. Pour rappel les graphiques présentent les coordonnées de la balle en fonction du temps sur un intervalle de dix secondes. La courbe verte correspond aux coordonnées de la balle sur l'axe des ordonnées et la courbe bleu correspond aux coordonnées de la balle sur l'axe des abscisses. Les graphiques dont les courbes s'arrêtent avant 10 secondes montrent que la balle a chuté du plateau.

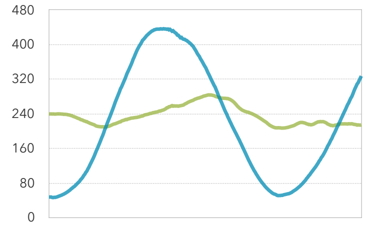
P = 1 D = 0



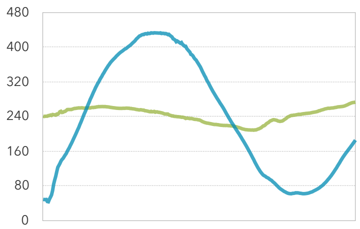
P = 1 D = 2



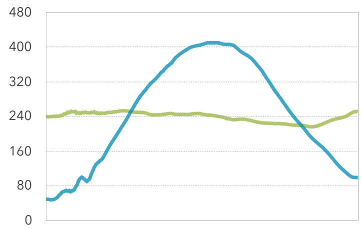
P = 1 D = 4



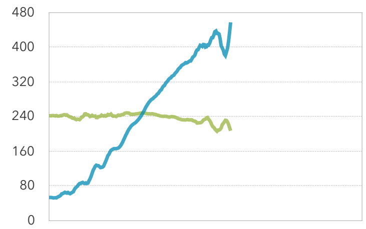
P = 1 D = 6



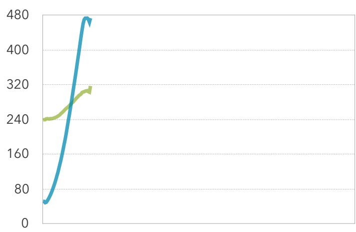
P = 1 D = 8



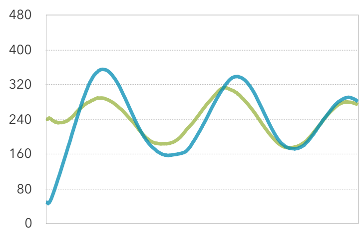
P = 1 D = 10



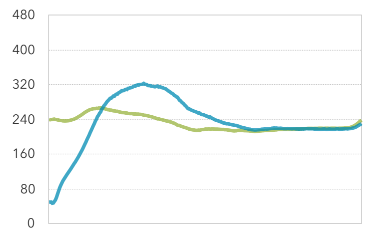
P = 3 D = 0



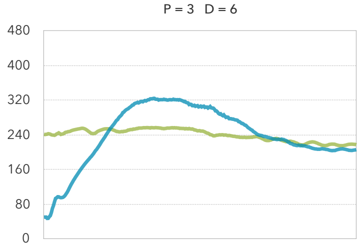
P = 3 D = 2



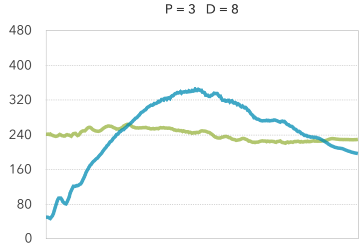
P = 3 D = 4



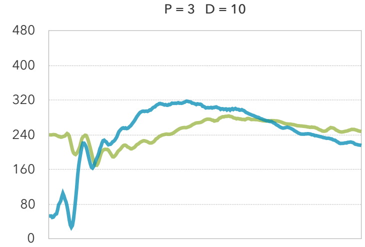
P = 3 D = 6



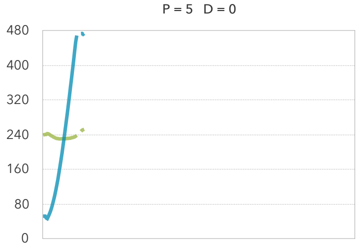
P = 3 D = 8



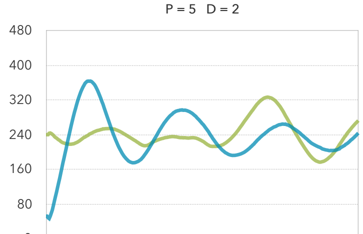
P = 3 D = 10



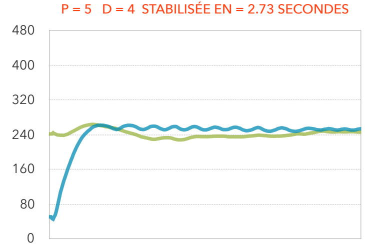
P = 5 D = 0



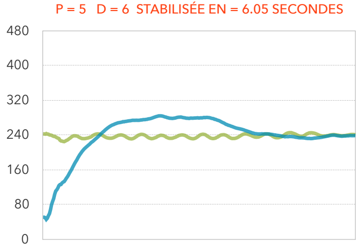
P = 5 D = 2



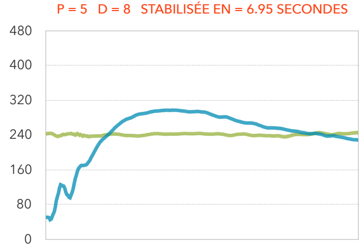
P = 5 D = 4 STABILISÉE EN = 2.73 SECONDES



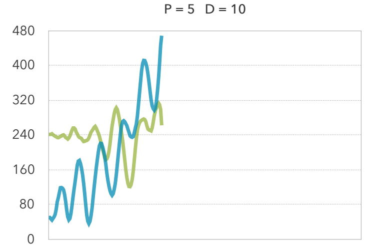
P = 5 D = 6 STABILISÉE EN = 6.05 SECONDES



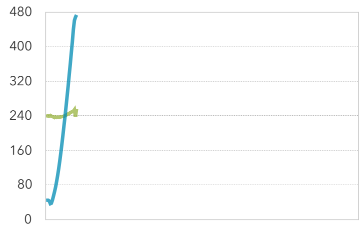
P = 5 D = 8 STABILISÉE EN = 6.95 SECONDES



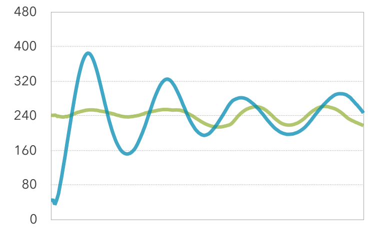
P = 5 D = 10



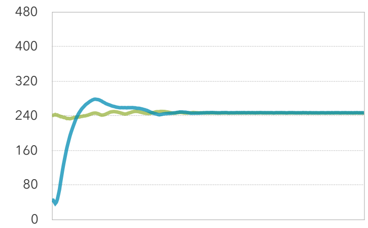
P = 7 D = 0



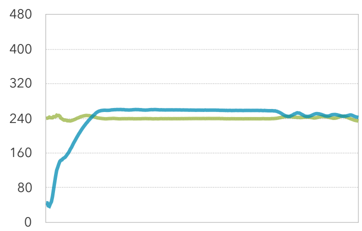
P = 7 D = 2



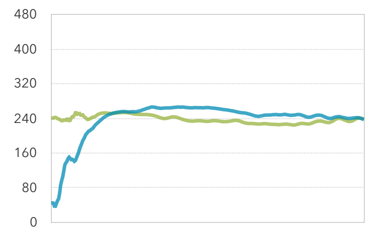
P = 7 D = 4 STABILISÉE EN = 2.16 SECONDES



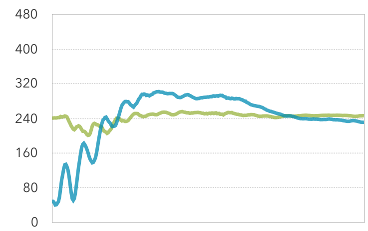
P = 7 D = 6 STABILISÉE EN = 3.67 SECONDES



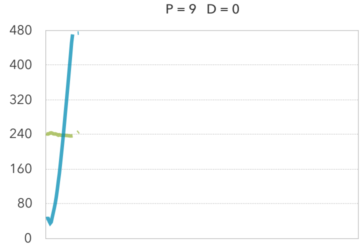
P = 7 D = 8 STABILISÉE EN = 5.59 SECONDES



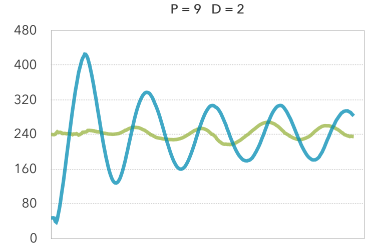
P = 7 D = 10 STABILISÉE EN = 6.91 SECONDES



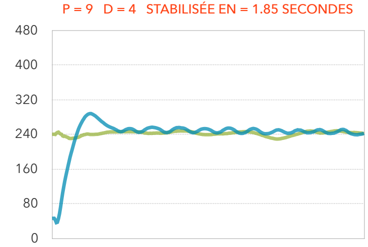
P = 9 D = 0



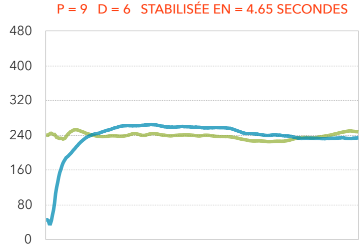
P = 9 D = 2



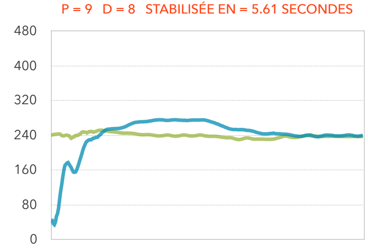
P = 9 D = 4 STABILISÉE EN = 1.85 SECONDES



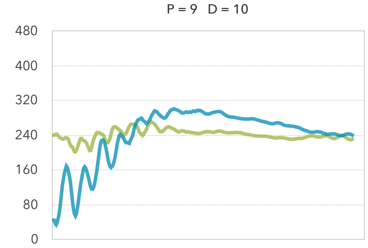
P = 9 D = 6 STABILISÉE EN = 4.65 SECONDES



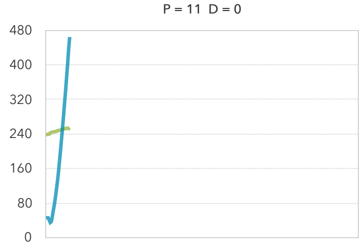
P = 9 D = 8 STABILISÉE EN = 5.61 SECONDES



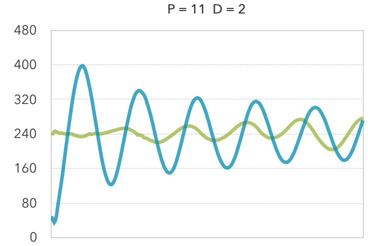
P = 9 D = 10



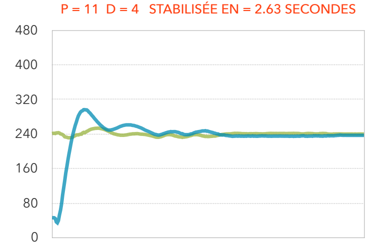
P = 11 D = 0



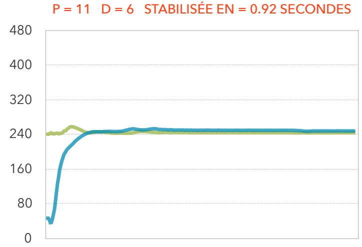
P = 11 D = 2



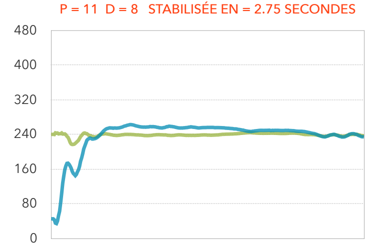
P = 11 D = 4 STABILISÉE EN = 2.63 SECONDES



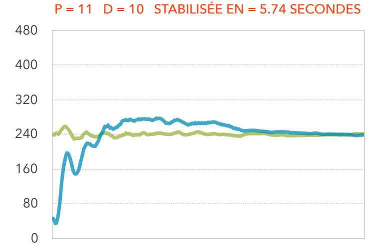
P = 11 D = 6 STABILISÉE EN = 0.92 SECONDES



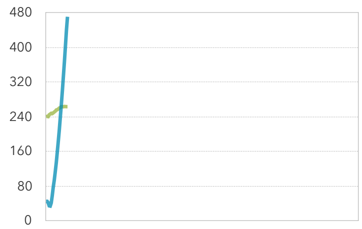
P = 11 D = 8 STABILISÉE EN = 2.75 SECONDES



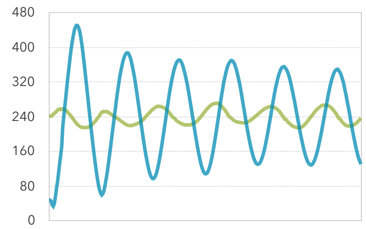
P = 11 D = 10 STABILISÉE EN = 5.74 SECONDES



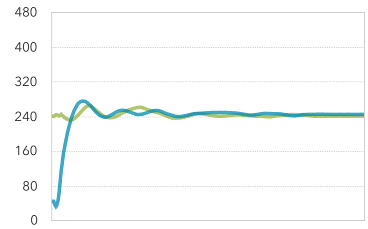
P = 13   D = 0



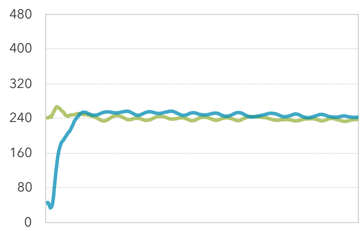
P = 13   D = 2



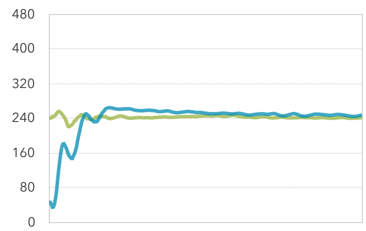
P = 13   D = 4   STABILISÉE EN = 1.35 SECONDES



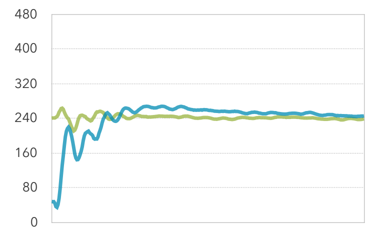
P = 13 D = 6   STABILISÉE EN = 0.86 SECONDES



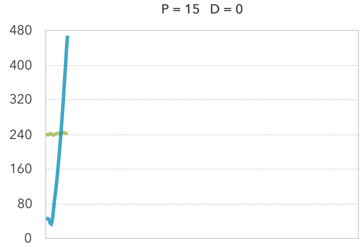
P = 13   D = 8   STABILISÉE EN = 2.69 SECONDES



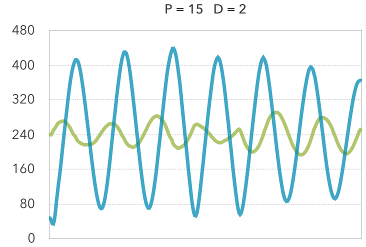
P = 13   D = 10   STABILISÉE EN = 4.49 SECONDES



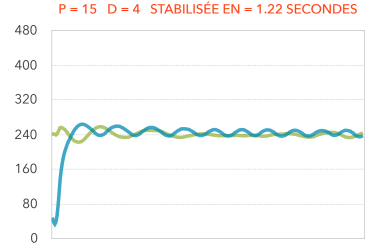
P = 15   D = 0



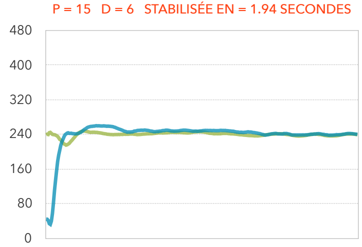
P = 15   D = 2



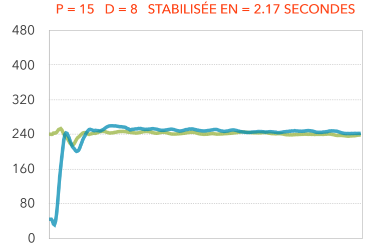
P = 15   D = 4   STABILISÉE EN = 1.22 SECONDES



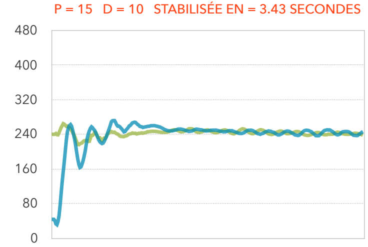
P = 15   D = 6   STABILISÉE EN = 1.94 SECONDES



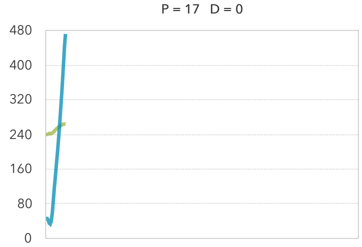
P = 15   D = 8   STABILISÉE EN = 2.17 SECONDES



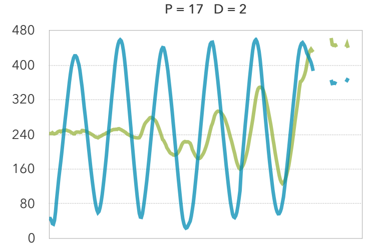
P = 15   D = 10   STABILISÉE EN = 3.43 SECONDES



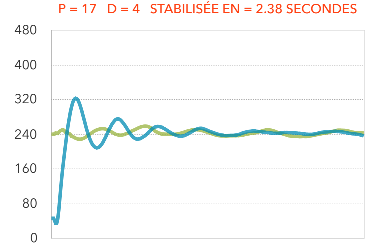
P = 17   D = 0



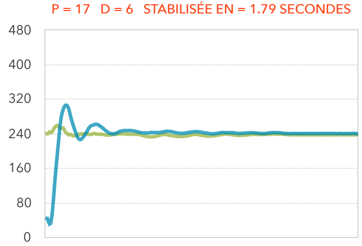
P = 17   D = 2



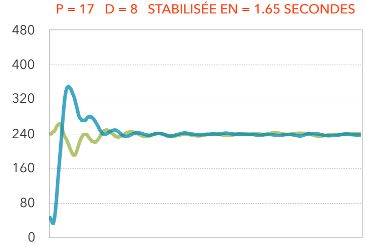
P = 17   D = 4   STABILISÉE EN = 2.38 SECONDES



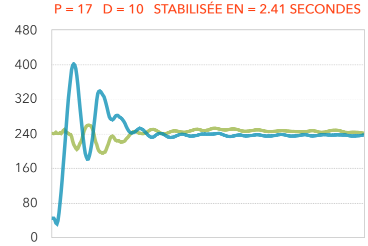
P = 17   D = 6   STABILISÉE EN = 1.79 SECONDES



P = 17   D = 8   STABILISÉE EN = 1.65 SECONDES



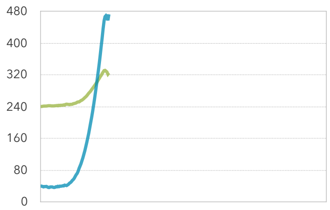
P = 17   D = 10   STABILISÉE EN = 2.41 SECONDES



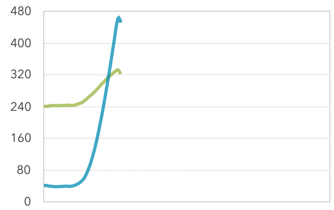
B Suite des résultats de la section 11.1.3

Résultats de l'expérience sur l'effet des coefficients P et D sur la stabilisation d'une balle en céramique. Pour rappel les graphiques présentent les coordonnées de la balle en fonction du temps sur un intervalle de dix secondes. La courbe verte correspond aux coordonnées de la balle sur l'axe des ordonnées et la courbe bleu correspond aux coordonnées de la balle sur l'axe des abscisses. Les graphiques dont les courbes s'arrêtent avant 10 secondes montrent que la balle a chuté du plateau.

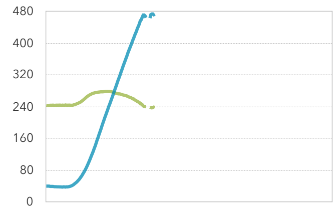
P = 1 D = 0



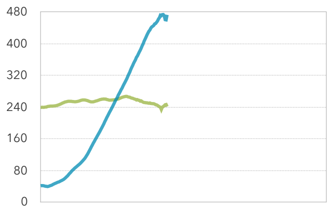
P = 1 D = 2



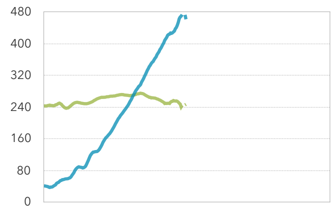
P = 1 D = 4



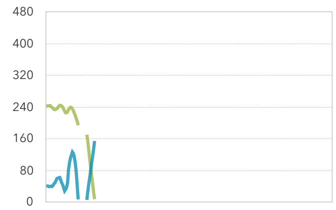
P = 1 D = 6



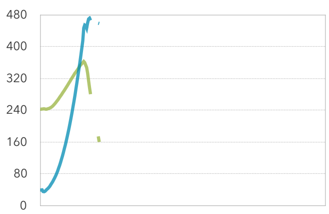
P = 1 D = 8



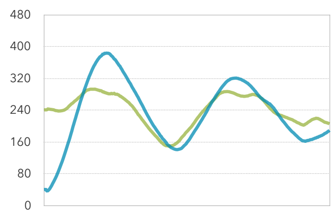
P = 1 D = 10



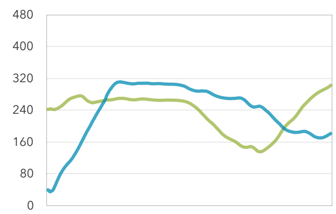
P = 3 D = 0



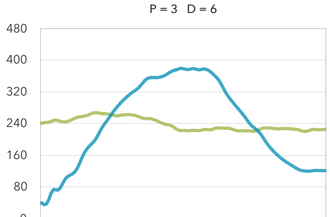
P = 3 D = 2



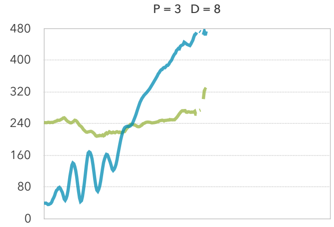
P = 3 D = 4



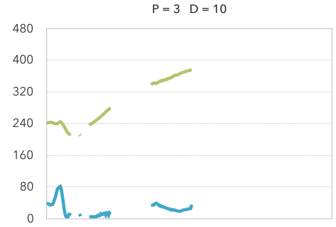
P = 3 D = 6



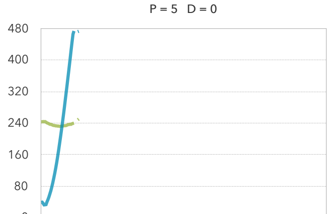
P = 3 D = 8



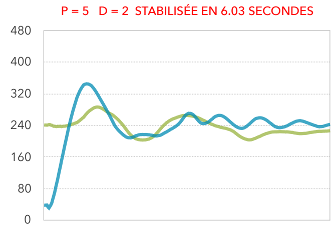
P = 3 D = 10



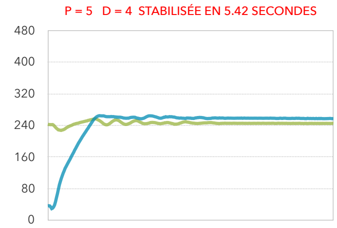
P = 5 D = 0



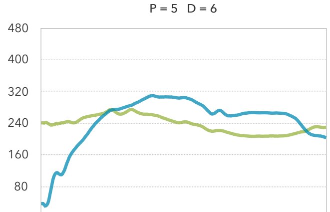
P = 5 D = 2 STABILISÉE EN 6.03 SECONDES



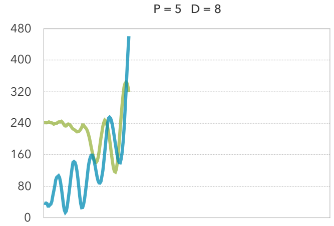
P = 5 D = 4 STABILISÉE EN 5.42 SECONDES



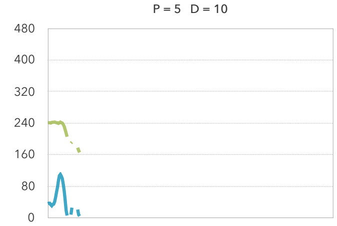
P = 5 D = 6



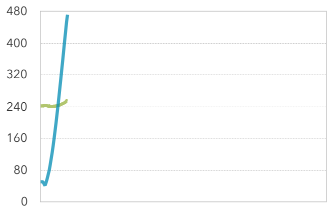
P = 5 D = 8



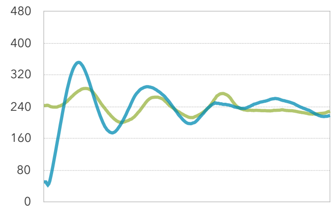
P = 5 D = 10



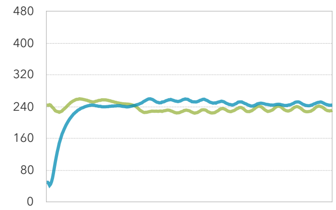
P = 7 D = 0



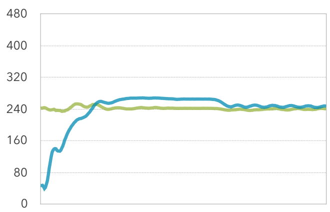
P = 7 D = 2 STABILISÉE EN 5.54 SECONDES



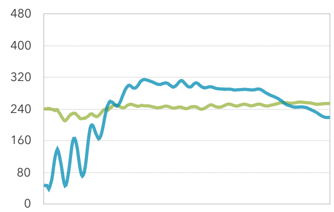
P = 7 D = 4 STABILISÉE EN 0.99 SECONDES



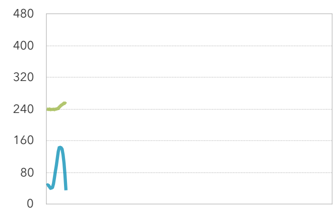
P = 7 D = 6 STABILISÉE EN 6.28 SECONDES



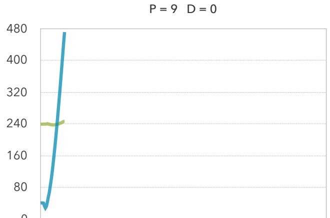
P = 7 D = 8



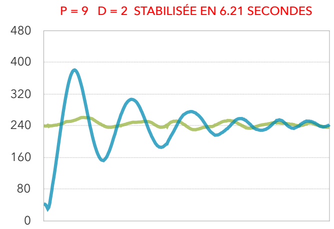
P = 7 D = 10



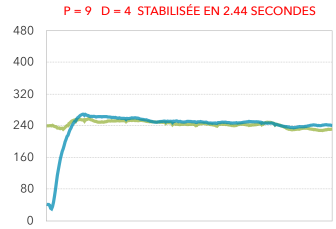
P = 9 D = 0



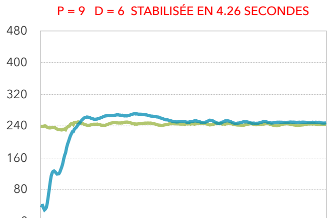
P = 9 D = 2 STABILISÉE EN 6.21 SECONDES



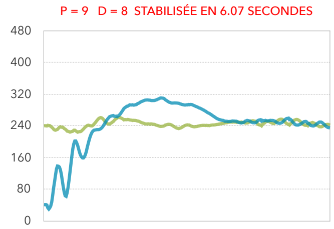
P = 9 D = 4 STABILISÉE EN 2.44 SECONDES



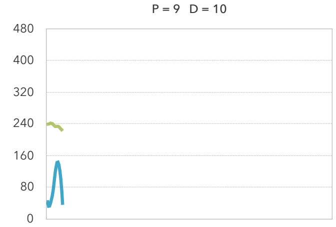
P = 9 D = 6 STABILISÉE EN 4.26 SECONDES



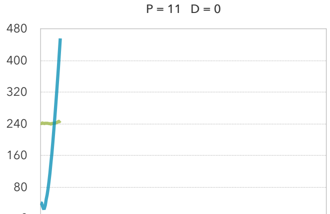
P = 9 D = 8 STABILISÉE EN 6.07 SECONDES



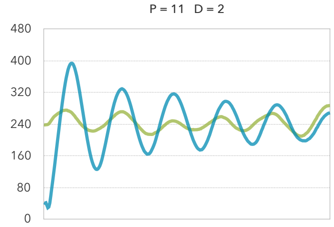
P = 9 D = 10



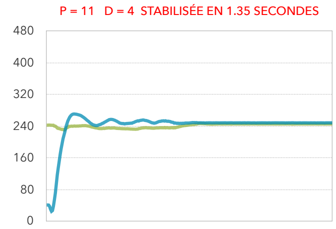
P = 11 D = 0



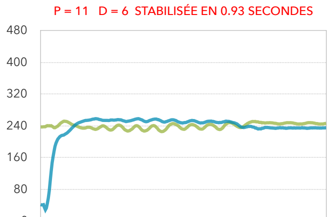
P = 11 D = 2



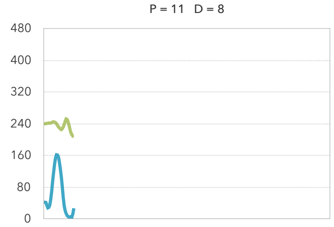
P = 11 D = 4 STABILISÉE EN 1.35 SECONDES



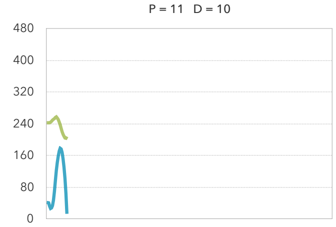
P = 11 D = 6 STABILISÉE EN 0.93 SECONDES



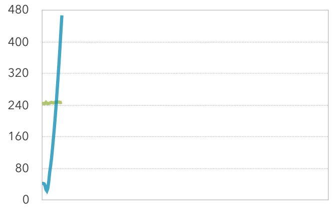
P = 11 D = 8



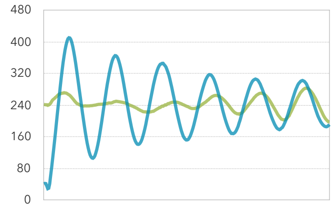
P = 11 D = 10



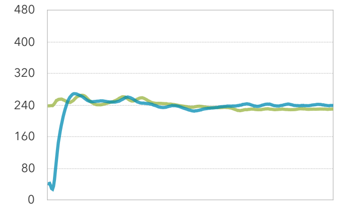
P = 13 D = 0



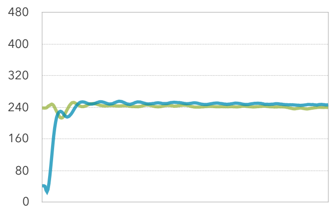
P = 13 D = 2



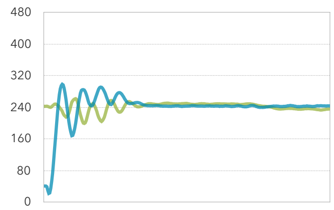
P = 13 D = 4 STABILISÉE EN 2.84 SECONDES



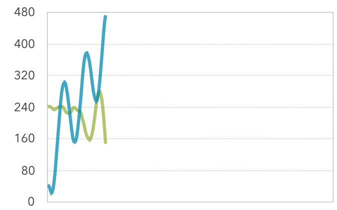
P = 13 D = 6 STABILISÉE EN 1.02 SECONDES



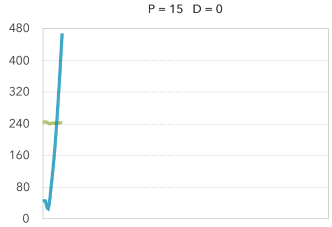
P = 13 D = 8 STABILISÉE EN 2.87 SECONDES



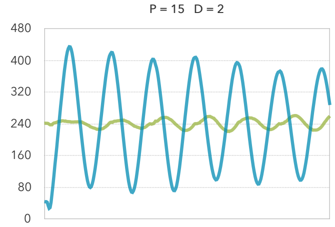
P = 13 D = 10



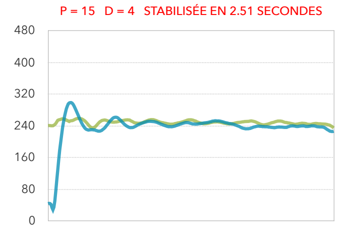
P = 15 D = 0



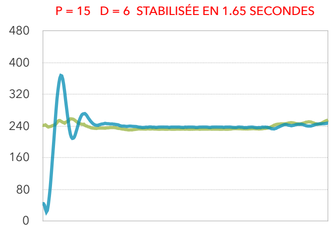
P = 15 D = 2



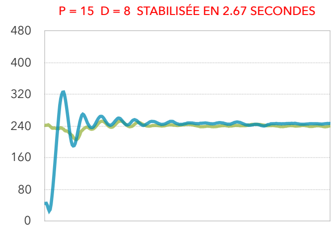
P = 15 D = 4 STABILISÉE EN 2.51 SECONDES



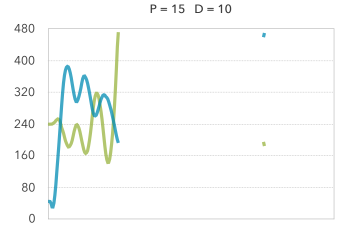
P = 15 D = 6 STABILISÉE EN 1.65 SECONDES



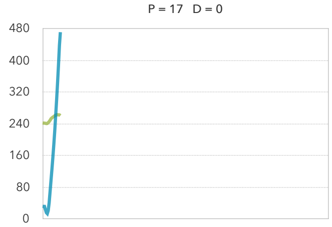
P = 15 D = 8 STABILISÉE EN 2.67 SECONDES



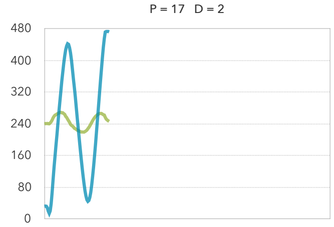
P = 15 D = 10



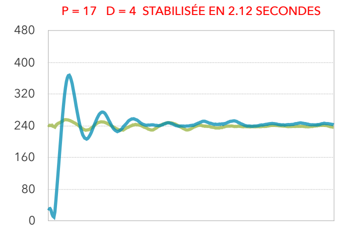
P = 17 D = 0



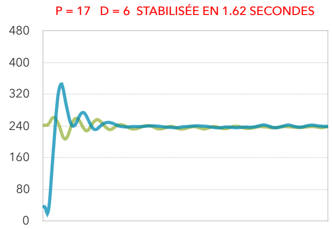
P = 17 D = 2



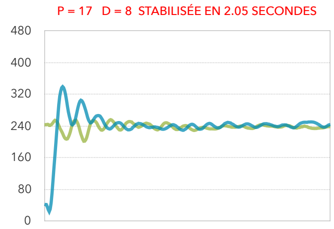
P = 17 D = 4 STABILISÉE EN 2.12 SECONDES



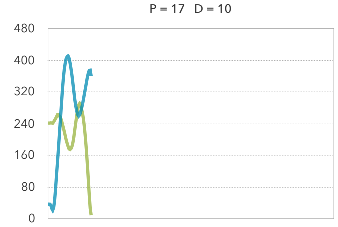
P = 17 D = 6 STABILISÉE EN 1.62 SECONDES



P = 17 D = 8 STABILISÉE EN 2.05 SECONDES

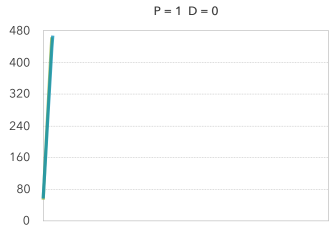


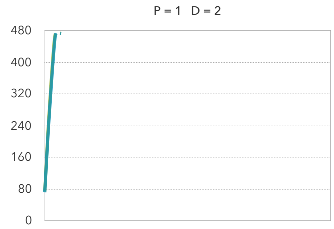
P = 17 D = 10

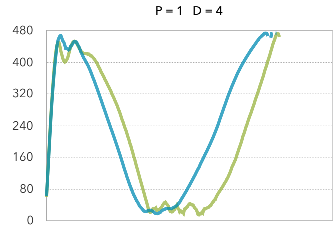


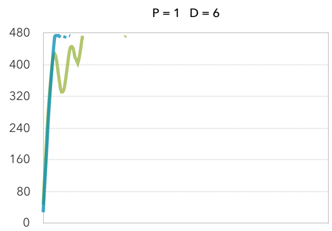
C Résultats de la section 11.3.1

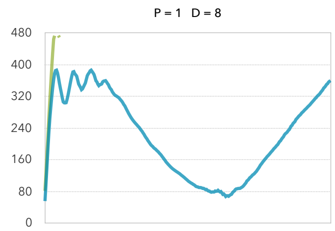
Résultats de l'expérience sur l'effet des coefficients P et D sur la stabilisation d'une balle en céramique et d'une balle de ping-pong dont la vitesse initiale est non nulle. Pour rappel les graphiques présentent les coordonnées de la balle en fonction du temps sur un intervalle de dix secondes. La courbe bleue correspond aux coordonnées de la balle de ping-pong sur l'axe des abscisses. La courbe verte correspond aux coordonnées de la balle en céramique sur l'axe des abscisses.

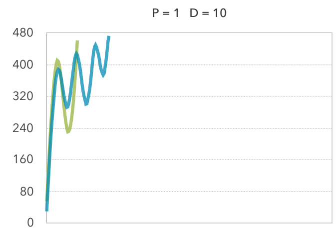

P = 1 D = 0
480
400
320
240
160
80
0

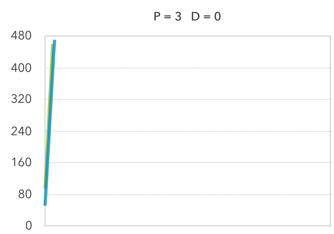

P = 1 D = 2
480
400
320
240
160
80
0

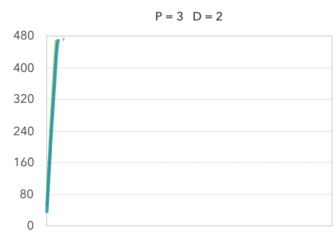

P = 1 D = 4
480
400
320
240
160
80
0

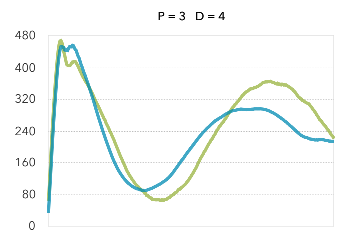

P = 1 D = 6
480
400
320
240
160
80
0

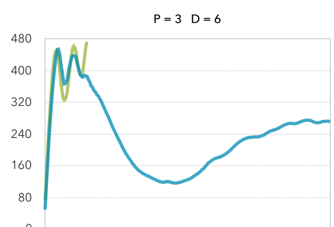

P = 1 D = 8
480
400
320
240
160
80
0

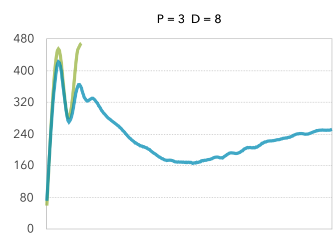

P = 1 D = 10
480
400
320
240
160
80
0

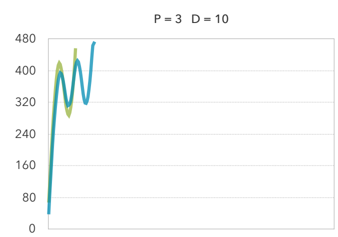

P = 3 D = 0
480
400
320
240
160
80
0

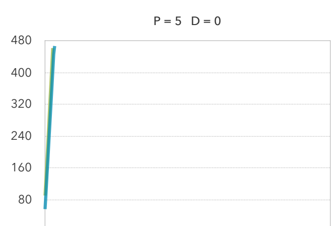

P = 3 D = 2
480
400
320
240
160
80
0

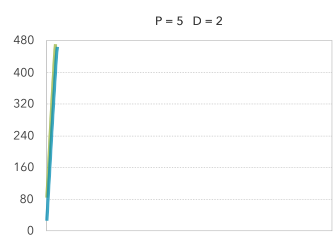

P = 3 D = 4
480
400
320
240
160
80
0

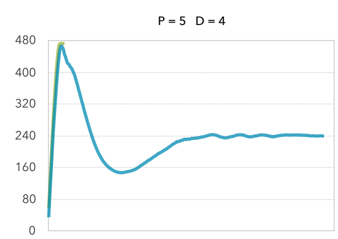

P = 3 D = 6
480
400
320
240
160
80
0

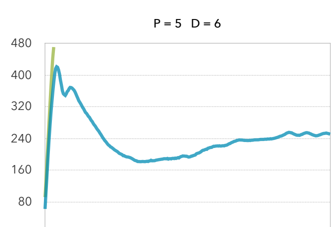

P = 3 D = 8
480
400
320
240
160
80
0

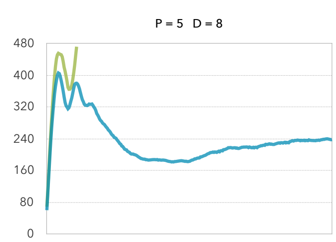

P = 3 D = 10
480
400
320
240
160
80
0

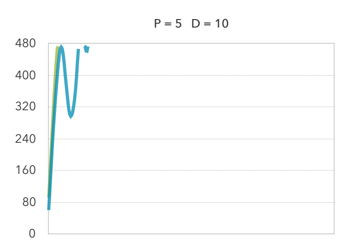

P = 5 D = 0
480
400
320
240
160
80
0

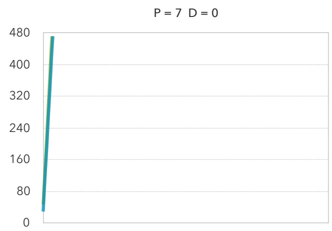
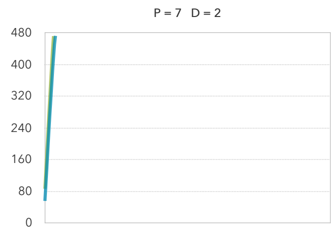
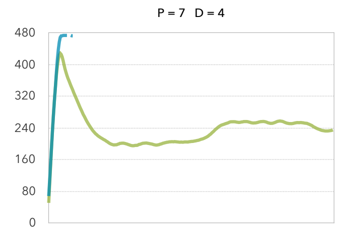
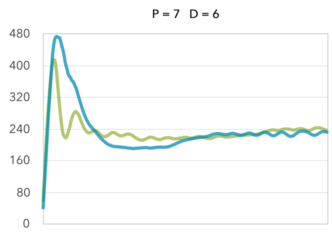
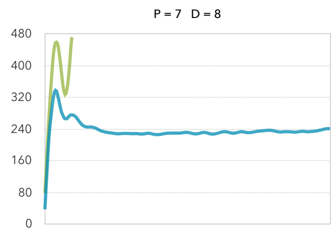
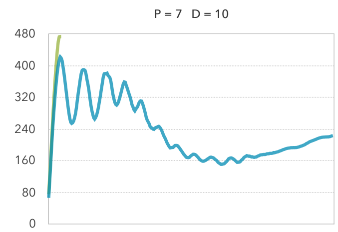
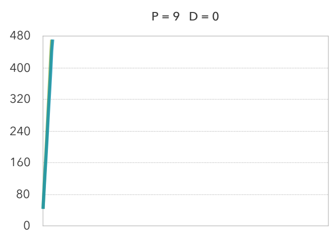
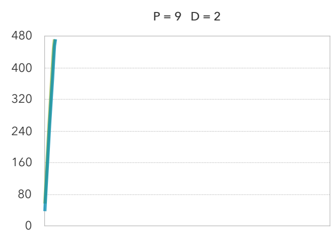
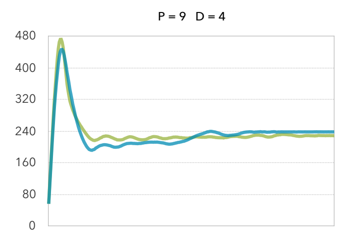
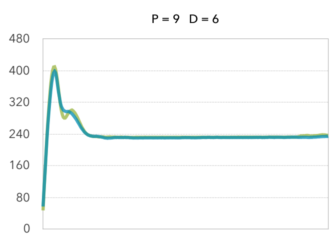
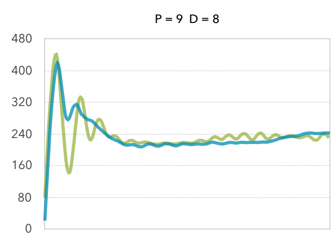
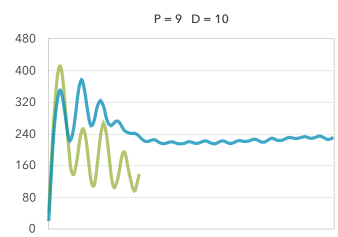
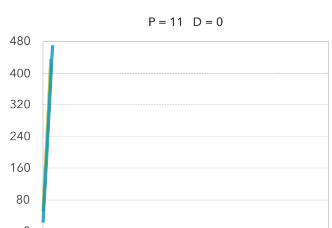
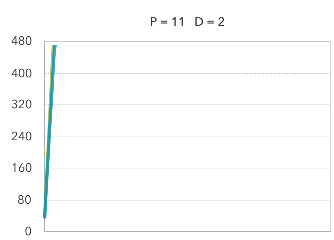
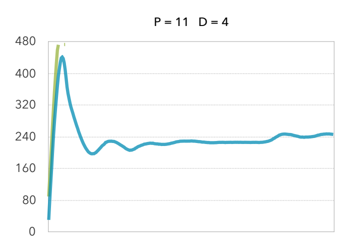
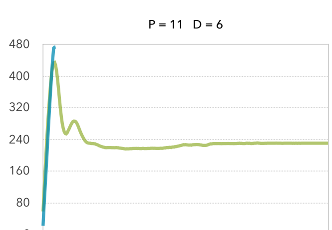
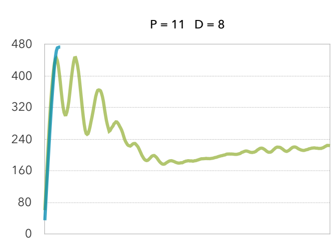
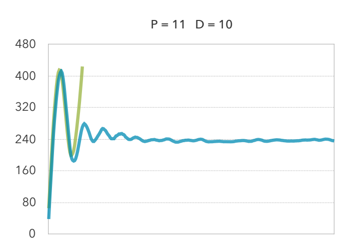

P = 5 D = 2
480
400
320
240
160
80
0

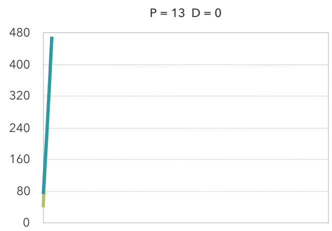

P = 5 D = 4
480
400
320
240
160
80
0

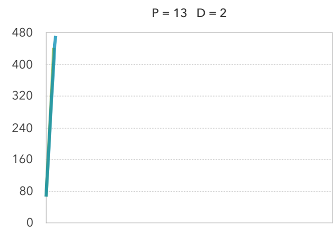

P = 5 D = 6
480
400
320
240
160
80
0

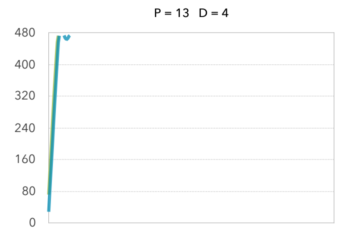

P = 5 D = 8
480
400
320
240
160
80
0

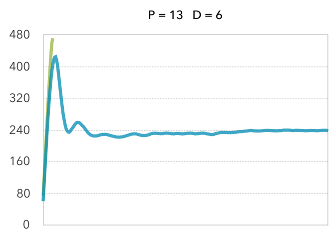

P = 5 D = 10
480
400
320
240
160
80
0

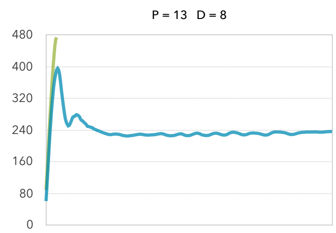

P = 7 D = 0
480
400
320
240
160
80
0

P = 7 D = 2
480
400
320
240
160
80
0

P = 7 D = 4
480
400
320
240
160
80
0

P = 7 D = 6
480
400
320
240
160
80
0

P = 7 D = 8
480
400
320
240
160
80
0

P = 7 D = 10
480
400
320
240
160
80
0

P = 9 D = 0
480
400
320
240
160
80
0

P = 9 D = 2
480
400
320
240
160
80
0

P = 9 D = 4
480
400
320
240
160
80
0

P = 9 D = 6
480
400
320
240
160
80
0

P = 9 D = 8
480
400
320
240
160
80
0

P = 9 D = 10
480
400
320
240
160
80
0

P = 11 D = 0
480
400
320
240
160
80
0

P = 11 D = 2
480
400
320
240
160
80
0

P = 11 D = 4
480
400
320
240
160
80
0

P = 11 D = 6
480
400
320
240
160
80
0

P = 11 D = 8
480
400
320
240
160
80
0

P = 11 D = 10
480
400
320
240
160
80
0

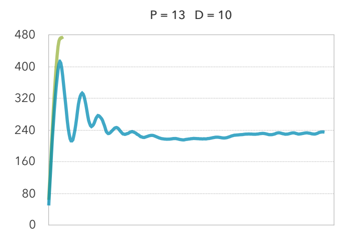

P = 13 D = 0
480
400
320
240
160
80
0

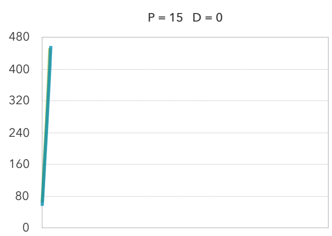

P = 13 D = 2
480
400
320
240
160
80
0

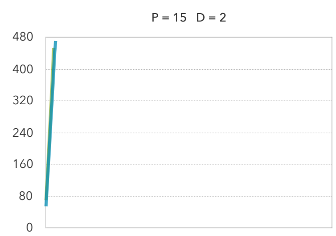

P = 13 D = 4
480
400
320
240
160
80
0

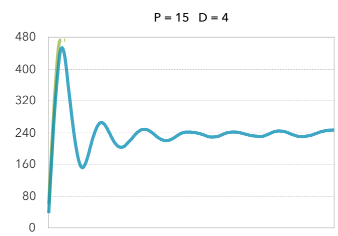

P = 13 D = 6
480
400
320
240
160
80
0

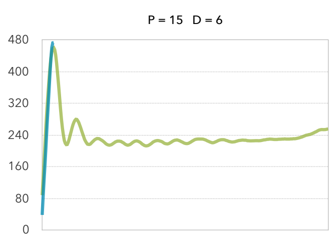

P = 13 D = 8
480
400
320
240
160
80
0

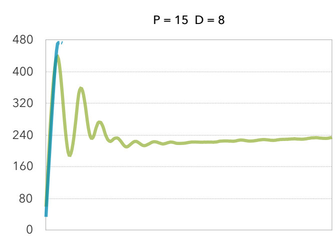

P = 13 D = 10
480
400
320
240
160
80
0

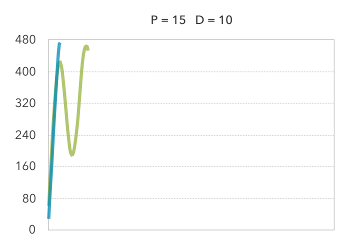

P = 15 D = 0
480
400
320
240
160
80
0

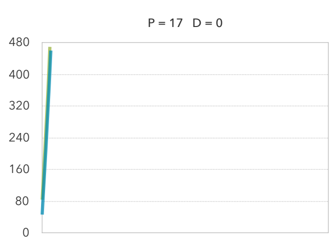

P = 15 D = 2
480
400
320
240
160
80
0

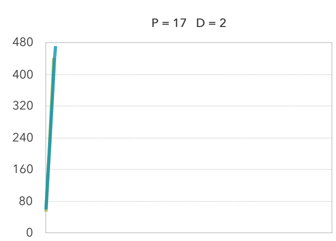

P = 15 D = 4
480
400
320
240
160
80
0

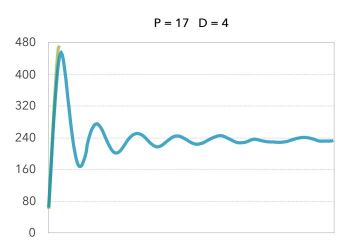

P = 15 D = 6
480
400
320
240
160
80
0

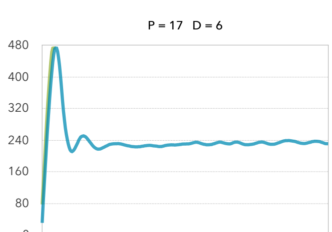

P = 15 D = 8
480
400
320
240
160
80
0

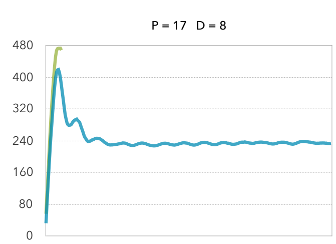

P = 15 D = 10
480
400
320
240
160
80
0

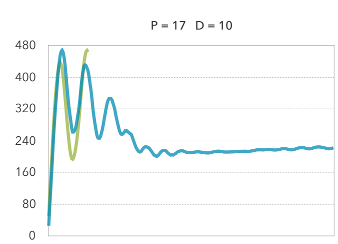

P = 17 D = 0
480
400
320
240
160
80
0


P = 17 D = 2
480
400
320
240
160
80
0


P = 17 D = 4
480
400
320
240
160
80
0


P = 17 D = 6
480
400
320
240
160
80
0


P = 17 D = 8
480
400
320
240
160
80
0


P = 17 D = 10
480
400
320
240
160
80
0

D solveEquation.py

```python
from scipy.optimize import fsolve
from math import *


# valeurs en cm
L = 8.5          # distance entre un moteur et l'origine
d = 11.5         # hauteur a laquelle le plateau se situe par rapport a l'origine
D = 18           # distance entre le bout des trois bras des moteurs
r, l = 7,8.5     # r est la longueur de la premiere partie qui compose un bras, l est la longueur de la deuxieme partie(celle sur laquelle se trouve la bille qui est en contact avec le plateau)


print("...")

alpha = 0
beta = 0
def equationsKMT(p):
    k, m, t = p
    equation1 = (-cos(alpha)*cos(pi/6)*k)**2 + (-cos(alpha)*m-cos(alpha)*sin(pi/6)*k)**2 + (sin(beta)*sin(alpha)*m+cos(pi/6)*cos(beta)*sin(alpha)*k+sin(beta)*sin(alpha)*sin(pi/6)*k)**2 - 1
    equation2 = (-cos(alpha)*cos(pi/6)*t)**2 + (cos(alpha)*sin(pi/6)*t+cos(alpha)*m)**2 + (cos(beta)*sin(alpha)*cos(pi/6)*t-sin(beta)*sin(alpha)*sin(pi/6)*t-sin(beta)*sin(alpha)*m)**2 - 1
    equation3 = (cos(alpha)*cos(pi/6)*k+cos(alpha)*cos(pi/6)*t)**2 + (cos(alpha)*sin(pi/6)*k-cos(alpha)*sin(pi/6)*t)**2 + (-cos(pi/6)*cos(beta)*sin(alpha)*k-sin(beta)*sin(alpha)*sin(pi/6)*k-cos(beta)*sin(alpha)*cos(pi/6)*t+sin(beta)*sin(alpha)*sin(pi/6)*t)**2 - 1
    return (equation1, equation2, equation3)

def getMaxMinPoint(): #retourne le point le plus haut et le point le plus bas
    distanceMax = 0
    distanceMin = 100000
    for beta in range(0, 361):
        beta = radians(beta)
        for alpha in range(0, 36):
            alpha = radians(alpha)
            k,m,t = fsolve(equationsKMT, (1, 1, 1))
            k,m,t = -D*k, -D*m, -D*t
            Xa, Ya, Za = k*cos(alpha)*cos(pi/6), k*cos(alpha)*sin(pi/6), k*(-cos(pi/6)*cos(beta)*sin(alpha)-sin(beta)*sin(alpha)*sin(pi/6))+d

            # ################################################################ trouver le point le plus eloigne et le plus proche du moteur
            if Za >= distanceMax:
```

```python
                distanceMax =  sqrt((sqrt(Xa**2 + Ya**2)-L)**2 + Za**2)
                XaMax = Xa
                YaMax = Ya
                ZaMax = Za

            elif Za <= distanceMin:
                distanceMin =  sqrt((sqrt(Xa**2 + Ya**2)-L)**2 + Za**2)
                XaMin = Xa
                YaMin = Ya
                ZaMin = Za

    return (XaMax, YaMax, ZaMax, XaMin, YaMin, ZaMin)

def equationsRL(p): # systeme d'equation pour trouver les longueurs r et l
    xmax,ymax,zmax,xmin,ymin,zmin = getMaxMinPoint()
    print(p)
    r, l = p
    equation1 = (sqrt((xmin)**2 + (ymin)**2)-L+cos(radians(0))*r)**2 + (zmin-sin(radians(0))*r)**2 - l**2
    equation2 = (sqrt((xmax)**2 + (ymax)**2)-L+cos(radians(90))*r)**2 + (zmax-sin(radians(90))*r)**2 - l**2
    return (equation1, equation2)

def equationTeta(teta, *pointsPosition):
    x, y, z = pointsPosition
    return ((L-cos(teta)*r-sqrt(x**2+y**2))**2 + (sin(teta)*r-z)**2) - l**2       #### L + cos pour bras oriente vers l'exterieur et L - cos pour oriente vers l'interieur

#r, l = fsolve(equationsRL, (9, 8))
#r, l = 9.05625,8.28459

fichier = open("/Users/johanlink/Desktop/TM/python/data.txt", "w") #le fichier txt est ecrase
premiereLigne = "alpha|beta|AngleservoA|AngleservoB|AngleservoC\n"
fichier.write(premiereLigne)


maxPos = 0
minPos = 1000
nbProbleme = 0
minMotorAngle = 100000
maxMotorAngle = 0

for beta in range(0, 360*5+1):
    beta = radians(beta)/5
    for alpha in range(0, 35*5+1):
        alpha = radians(alpha)/5
        k,m,t = fsolve(equationsKMT, (1, 1, 1))
        k,m,t = -D*k, -D*m, -D*t
        Xa, Ya, Za = k*cos(alpha)*cos(pi/6), k*cos(alpha)*sin(pi/6), k*(-cos(pi/6)*cos(beta)*sin(alpha)-sin(beta)*sin(alpha)*sin(pi/6))+d
        Xb, Yb, Zb = 0, m*(-cos(alpha)), m*sin(beta)*sin(alpha)+d
```

```
        Xc, Yc, Zc = t*(-cos(alpha)*cos(pi/6)), t*cos(alpha)*sin(pi/6), t*(cos(beta)*sin(alpha)*cos(pi/6)-sin(beta)*sin(alpha)*sin(pi/6))+d
        tetaA = degrees(fsolve(equationTeta, 1, args=(Xa, Ya, Za)))
        tetaB = degrees(fsolve(equationTeta, 1, args=(Xb, Yb, Zb)))
        tetaC = degrees(fsolve(equationTeta, 1, args=(Xc, Yc, Zc)))
        tetaA = round(tetaA, 2)
        tetaB = round(tetaB, 2)
        tetaC = round(tetaC, 2)

        separateur = "|"
        data = str(round(degrees(alpha),2)) +  separateur + str(round(degrees(beta),2)) + "#" + str(tetaA) + separateur + str(tetaB) + separateur + str(tetaC) + "\n"
        fichier.write(data)

        if tetaA > maxMotorAngle:
            maxMotorAngle = tetaA
            angleMaxPoint = (Xa, Ya, Za)

        if tetaA < minMotorAngle:
            minMotorAngle = tetaA
            angleMinPoint = (Xa, Ya, Za)



infos = "\nDistance L entre servo et origine:"+str(L)+"\nHauteur d du plateau:"+str(d)+"\nDistance D entre les points A,B,C:"+str(D)+"\nLongueur r du bras du servo:"+str(r)+"\nLongueur l du bras du servo:"+str(l)
minmaxAngle = "\nminMotorAngle = " + str(minMotorAngle) + "\nmaxMotorAngle = " + str(maxMotorAngle)
XaMax, YaMax, ZaMax, XaMin, YaMin, ZaMin = getMaxMinPoint()
minmaxPos = "\nle point le plus proche du servo:"+str(XaMin)+"    "+str(YaMin)+"    "+str(ZaMin)+"\nle point le plus eloigne du servo:"+str()+str(XaMax)+"    "+str(YaMax)+"    "+str(ZaMax)
fichier.write("\n")
fichier.write("valeurs en cm")
fichier.write(infos)
fichier.write(minmaxAngle)
fichier.write(minmaxPos)

fichier.close()

print("Termine")
```

E programmeIntermediaire.py

```python
import cv2
import numpy as np
import time
import imutils

cam = cv2.VideoCapture(0)
cv2.namedWindow("image")

cam.set(3,320)
cam.set(4,240)

getPixelColor = False
H,S,V = 0,0,0

mouseX,mouseY = 0,0

def getMousePositionOnClick(event,x,y,flags,param):
    global mouseX,mouseY
    global getPixelColor
    if event == cv2.EVENT_LBUTTONDOWN:
        mouseX,mouseY = x,y
        getPixelColor = True
        print(getPixelColor)
    elif event == cv2.EVENT_LBUTTONUP:
        getPixelColor = False


cv2.setMouseCallback("image",getMousePositionOnClick)

while True:
    start_time = time.time()
    ret, img=cam.read()
    imgHSV= cv2.cvtColor(img,cv2.COLOR_BGR2HSV)

    if getPixelColor == True:
        pixelColorOnClick = img[mouseY,mouseX]
        pixelColorOnClick = np.uint8([[pixelColorOnClick]])
        pixelColorOnClick = cv2.cvtColor(pixelColorOnClick,cv2.COLOR_BGR2HSV)
        H = pixelColorOnClick[0,0,0]
        S = pixelColorOnClick[0,0,1]
        V = pixelColorOnClick[0,0,2]
        print(H, S, V, mouseX, mouseY)

    lowerBound=np.array([H-7,S-70,V-70])
    upperBound=np.array([H+7,S+70,V+70])

    mask=cv2.inRange(imgHSV,lowerBound,upperBound)
    mask = cv2.blur(mask,(7,7))                        # ajoute du flou a l'image
    mask = cv2.erode(mask, None, iterations=2)         # retire les parasites
    mask = cv2.dilate(mask, None, iterations=2)        # retire les parasites

```

```
    cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.
        CHAIN_APPROX_SIMPLE)
    cnts = cnts[0] if imutils.is_cv2() else cnts[1]
    center = None

    if len(cnts) > 0:
        c = max(cnts, key=cv2.contourArea)
        (x, y), radius = cv2.minEnclosingCircle(c)

        if radius > 2:
            cv2.circle(img, (int(x), int(y)), int(radius),(0, 255,
                255), 2)

    cv2.imshow("mask",mask)
    cv2.imshow("image",img)
    cv2.waitKey(10)
    print("FPS: ", 1.0 / (time.time() - start_time))
```

F interface.py

```python
import cv2
import numpy as np
import time
import imutils
import tkinter as tk
import tkinter.messagebox
from PIL import Image, ImageTk
import serial
import serial.tools.list_ports
from math import *


lines = open("/Users/johanlink/Desktop/TM/python/data.txt").read().splitlines()
lines = lines[:-11]      #enleve les 11 dernieres lignes du fichier
lines = lines[1:]        #enleve la premiere ligne du fichier

dataDict = {}

camHeight = 480
camWidth = 640
cam = cv2.VideoCapture(0)
cam.set(3,camWidth)
cam.set(4,camHeight)

getPixelColor = False
H,S,V = 0,0,0

mouseX,mouseY = 0,0

for i in range(0,len(lines)):
    key, value = lines[i].split("#")
    alpha, beta = key.split("|")
    angleA, angleB, angleC = value.split("|")
    dataDict[(float(alpha),float(beta))] = (float(angleA), float(angleB), float(angleC))

controllerWindow = tk.Tk()
controllerWindow.title("fenetre de controle")
controllerWindow.geometry("820x500")
controllerWindow["bg"]="white"
controllerWindow.resizable(0, 0)

videoWindow = tk.Toplevel(controllerWindow)
videoWindow.title("retour camera")
videoWindow.resizable(0, 0)  #empeche de modifier les dimensions de la fenetre
lmain = tk.Label(videoWindow)
lmain.pack()
videoWindow.withdraw()

graphWindow = tk.Toplevel(controllerWindow)
graphWindow.title("Position en fonction du temps")
graphWindow.resizable(0, 0)
graphCanvas = tk.Canvas(graphWindow,width=camHeight+210,height=
```

```
    camHeight)
graphCanvas.pack()
graphWindow.withdraw()

pointsListCircle = []
def createPointsListCircle(rayon):
    global pointsListCircle
    for angle in range(0,360):
        angle=angle-90
        pointsListCircle.append([rayon*cos(radians(angle))+240,
            rayon*sin(radians(angle))+240])
createPointsListCircle(150)

pointsListEight = []
def createPointsListEight(rayon):
    global pointsListEight
    for angle in range(270,270+360):
        pointsListEight.append([rayon*cos(radians(angle))+240,rayon
            *sin(radians(angle))+240+rayon])
    for angle in range(360,0,-1):
        angle=angle+90
        pointsListEight.append([rayon*cos(radians(angle))+240,rayon
            *sin(radians(angle))+240-rayon])
createPointsListEight(80)

drawCircleBool = False
def startDrawCircle():
    global drawCircleBool, drawEightBool, consigneX, consigneY
    if drawCircleBool == False:
        drawCircleBool = True
        BballDrawCircle["text"] = "Centrer la bille"
    else:
        drawCircleBool = False
        consigneX, consigneY = 240, 240
        sliderCoefP.set(sliderCoefPDefault)
        BballDrawCircle["text"] = "Faire tourner la bille en cercle
            "

drawEightBool = False
def startDrawEight():
    global drawEightBool, drawCircleBool, consigneX, consigneY
    if drawEightBool == False:
        drawEightBool = True
        BballDrawEight["text"] = "Centrer la bille"
    else:
        drawEightBool = False
        consigneX, consigneY = 240, 240
        sliderCoefP.set(sliderCoefPDefault)
        BballDrawEight["text"] = "Faire tourner la bille en huit"

pointCounter = 0
def drawWithBall():
    global pointCounter, consigneX, consigneY
    if drawCircleBool == True:
        sliderCoefP.set(15)
        if pointCounter >= len(pointsListCircle):
            pointCounter = 0
```

```python
        point = pointsListCircle[pointCounter]
        consigneX, consigneY = point[0], point[1]
        pointCounter += 7
    if drawEightBool == True:
        sliderCoefP.set(15)
        if pointCounter >= len(pointsListEight):
            pointCounter = 0
        point = pointsListEight[pointCounter]
        consigneX, consigneY = point[0], point[1]
        pointCounter += 7


def setConsigneWithMouse(mousePosition):
    global consigneX, consigneY
    if mousePosition.y > 10:
        refreshGraph()
        consigneX,consigneY = mousePosition.x,mousePosition.y


def getMouseClickPosition(mousePosition):
    global mouseX,mouseY
    global getPixelColor
    mouseX,mouseY = mousePosition.x,mousePosition.y
    getPixelColor = True

showVideoWindow = False
def showCameraFrameWindow():
    global showVideoWindow, showGraph
    global BRetourVideoTxt
    if showVideoWindow == False:
        if showGraph == True:
            graphWindow.withdraw()
            showGraph = False
            BafficherGraph["text"] = "Afficher graphique"
        videoWindow.deiconify()
        showVideoWindow = True
        BRetourVideo["text"] = "Cacher le retour video "
    else:
        videoWindow.withdraw()
        showVideoWindow = False
        BRetourVideo["text"] = "Afficher le retour video"

showCalqueCalibrationBool = False
def showCalqueCalibration():
    global showCalqueCalibrationBool
    showCalqueCalibrationBool = not showCalqueCalibrationBool

showGraph = False
def showGraphWindow():
    global showGraph, showVideoWindow
    global BafficherGraph

    if showGraph == False:
        if showVideoWindow == True:
            videoWindow.withdraw()
            showVideoWindow = False
            BRetourVideo["text"] = "Afficher le retour video"
```

```
        showGraph = True
        BafficherGraph["text"] = "Cacher graphique "
    else:
        showGraph = False
        BafficherGraph["text"] = "Afficher graphique"

t = 480
consigneY = 240
consigneX = 240
def paintGraph():
    global t,consigneY,x,y,prevX,prevY,alpha,prevAlpha
    global showGraphPositionX,showGraphPositionY, showGraphAlpha
    if showGraph == True:
        graphWindow.deiconify()
        if showGraphPositionX.get() == 1:
            graphCanvas.create_line(t-3,prevX,t,x, fill="#b20000", width=2)
        if showGraphPositionY.get() == 1:
            graphCanvas.create_line(t-3,prevY,t,y, fill="#0069b5", width=2)
        if showGraphAlpha.get() == 1:
            graphCanvas.create_line(t-3,240-prevAlpha*3,t,240-alpha*3, fill="#8f0caf", width=2)
        if t >= 480:
            t = 0
            graphCanvas.delete("all")
            graphCanvas.create_line(3,3,480,3,fill="black", width=3)
            graphCanvas.create_line(3,480,480,480,fill="black", width=3)
            graphCanvas.create_line(3,3,3,480,fill="black", width=3)
            graphCanvas.create_line(480,3,480,480,fill="black", width=3)
            graphCanvas.create_line(550,32,740,32,fill="#b20000", width=5)
            graphCanvas.create_line(550,53,740,53,fill="#0069b5", width=5)
            graphCanvas.create_line(550,73,740,73,fill="#8f0caf", width=5)
            if showGraphPositionX.get() == 1:
                graphCanvas.create_line(3,consigneX,480,consigneX, fill="#ff7777", width=2)
            if showGraphPositionY.get() == 1:
                graphCanvas.create_line(3,consigneY,480,consigneY, fill="#6f91f7", width=2)
        t += 3
    else:
        graphWindow.withdraw()

def refreshGraph():
    global t
    t=480

def endProgam():
    controllerWindow.destroy()

```

```

sliderHDefault = 15
sliderSDefault = 70
sliderVDefault = 70
sliderCoefPDefault = 5
sliderCoefIDefault = 0.1
sliderCoefDDefault = 5.7

def resetSlider():
    sliderH.set(sliderHDefault)
    sliderS.set(sliderSDefault)
    sliderV.set(sliderVDefault)
    sliderCoefP.set(sliderCoefPDefault)
    sliderCoefI.set(sliderCoefIDefault)
    sliderCoefD.set(sliderCoefDDefault)

def donothing():
    pass

def rangerPlateau():
    if arduinoIsConnected == True:
        if tkinter.messagebox.askokcancel("Avertissement", "Pensez a retirer le plateau."):
            print("abaissement des bras")
            ser.write(("descendreBras\n").encode())
    else:
        if tkinter.messagebox.askokcancel("Avertissement","L'Arduino n'est pas connecte"):
            donothing()


def eleverPlateau():
    global alpha
    if arduinoIsConnected == True:
        if tkinter.messagebox.askokcancel("Avertissement", "Pensez a retirer le plateau."):
            print("Elevation des bras")
            ser.write((str(dataDict[(0,0)])+"\n").encode())
            alpha = 0
    else:
        if tkinter.messagebox.askokcancel("Avertissement","L'Arduino n'est pas connecte"):
            donothing()

def servosTest():
    if arduinoIsConnected == True:
        if tkinter.messagebox.askokcancel("Avertissement", "Le plateau doit etre en place."):
            for i in range(2):
                beta = 0
                alpha = 35
                while beta < 360:
                    ser.write((str(dataDict[(alpha,beta)])+"\n").encode())
                    ser.flush()
                    time.sleep(0.002)
                    beta = round(beta+0.2,2)
```

```
                    print(alpha,beta)
            time.sleep(1)
            ser.write((str(dataDict[(0,0)])+"\n").encode())
    else:
        if tkinter.messagebox.askokcancel("Avertissement","L'Arduino n'est pas connecte"):
            donothing()


arduinoIsConnected = False
def connectArduino():
    global ser
    global label
    global arduinoIsConnected
    ports = list(serial.tools.list_ports.comports())
    for p in ports:
        if "Arduino" in p.description:
            print(p)
            ser = serial.Serial(p[0], 19200, timeout=1)
            time.sleep(1) #give the connection a second to settle
            label.configure(text="Arduino connecte", fg="#36db8b")
            arduinoIsConnected = True

startBalanceBall = False
def startBalance():
    global startBalanceBall
    if arduinoIsConnected == True:
        if startBalanceBall == False:
            startBalanceBall = True
            BStartBalance["text"] = "Arreter"
        else:
            startBalanceBall = False
            BStartBalance["text"] = "Commencer"
    else:
        if tkinter.messagebox.askokcancel("Avertissement","L'Arduino n'est pas connecte"):
            donothing()

sommeErreurX = 1
sommeErreurY = 1
timeInterval = 1
alpha, beta, prevAlpha, prevBeta = 0,0,0,0
omega = 0.2
def PIDcontrol(ballPosX, ballPosY, prevBallPosX, prevBallPosY, consigneX, consigneY):
    global omega
    global sommeErreurX, sommeErreurY
    global alpha, beta, prevAlpha, prevBeta
    global startBalanceBall, arduinoIsConnected

    Kp = sliderCoefP.get()
    Ki = sliderCoefI.get()
    Kd = sliderCoefD.get()

    Ix = Kp*(consigneX-ballPosX) + Ki*sommeErreurX + Kd*((prevBallPosX-ballPosX)/0.0333)
    Iy = Kp*(consigneY-ballPosY) + Ki*sommeErreurY + Kd*((
```

```
        prevBallPosY-ballPosY)/0.0333)

    Ix = round(Ix/10000, 4)
    Iy = round(Iy/10000, 4)

    if Ix == 0 and Iy == 0:
        alpha = 0
        beta = 0

    elif Ix != 0 and sqrt(Ix**2 + Iy**2) < 1:
        beta = atan(Iy/Ix)
        alpha = asin(sqrt(Ix**2 + Iy**2))
        beta = degrees(beta)
        alpha = degrees(alpha)
        if Ix < 0 and Iy >= 0:
            beta = abs(beta)
        elif Ix > 0 and Iy >= 0:
            beta = 180-abs(beta)
        elif Ix > 0 and Iy <= 0:
            beta = 180+abs(beta)
        elif Ix < 0 and Iy <= 0:
            beta = 360-abs(beta)

    elif Ix == 0 and sqrt(Ix**2 + Iy**2) < 1:
        if Iy > 0:
            beta = 90
            alpha = asin(sqrt(Ix**2 + Iy**2))
        elif Iy < 0:
            beta = 270
            alpha = asin(sqrt(Ix**2 + Iy**2))
        alpha = degrees(alpha)

    elif Ix != 0 and sqrt(Ix**2 + Iy**2) > 1:
        beta = degrees(atan(Iy/Ix))
        alpha = 35
        if Ix < 0 and Iy >= 0:
            beta = abs(beta)
        elif Ix > 0 and Iy >= 0:
            beta = 180-abs(beta)
        elif Ix > 0 and Iy <= 0:
            beta = 180+abs(beta)
        elif Ix < 0 and Iy <= 0:
            beta = 360-abs(beta)

    elif Ix == 0 and sqrt(Ix**2 + Iy**2) > 1:
        alpha = 35
        if Iy > 0:
            beta = 90
        elif Iy < 0:
            beta = 270

    if alpha > 35:
        alpha = 35

    alpha = prevAlpha * omega + (1-omega) * alpha
    beta = prevBeta * omega + (1-omega) * beta

```

```
    alpha = round(round(alpha / 0.2) * 0.2, -int(floor(log10(0.2)))) ## permet d'arrondire avec 0.2 de precision
    beta = round(round(beta / 0.2) * 0.2, -int(floor(log10(0.2))))

    if alpha <= 35 and beta <= 360 and arduinoIsConnected == True and startBalanceBall == True:
        ser.write((str(dataDict[(alpha,beta)])+"\n").encode())

    #print(alpha, beta)
    print(Ix,Iy,alpha,beta,ballPosX,ballPosY,prevBallPosX,prevBallPosY,sommeErreurX,sommeErreurY)

    if startBalanceBall == True:
        sommeErreurX += (consigneX-ballPosX)
        sommeErreurY += (consigneY-ballPosY)


prevX,prevY = 0,0
prevConsigneX, prevConsigneY = 0,0
start_time = 0
def main():
    start_timeFPS = time.time()
    global H,S,V
    global getPixelColor
    global x,y, alpha, beta
    global prevX, prevY, prevAlpha, prevBeta, prevConsigneX, prevConsigneY
    global consigneX, consigneY, sommeErreurX, sommeErreurY
    global camWidth,camHeight
    global timeInterval, start_time
    global showVideoWindow

    _, img=cam.read()
    img = img[0:int(camHeight),int((camWidth-camHeight)/2):int(camWidth-((camWidth-camHeight)/2))] #[Y1:Y2,X1:X2]
    imgCircle = np.zeros(img.shape, dtype=np.uint8)
    cv2.circle(imgCircle, (240,240), 270, (255, 255, 255), -1, 8, 0)
    img = img & imgCircle
    imgHSV = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)

    if getPixelColor == True and mouseY > 0 and mouseY < 480 and mouseX < 480:
        pixelColorOnClick = img[mouseY,mouseX]
        pixelColorOnClick = np.uint8([[pixelColorOnClick]])
        pixelColorOnClick = cv2.cvtColor(pixelColorOnClick,cv2.COLOR_BGR2HSV)
        H = pixelColorOnClick[0,0,0]
        S = pixelColorOnClick[0,0,1]
        V = pixelColorOnClick[0,0,2]
        print(mouseX, mouseY)
        getPixelColor = False

    lowerBound=np.array([H-sliderH.get(),S-sliderS.get(),V-sliderV.get()])
    upperBound=np.array([H+sliderH.get(),S+sliderS.get(),V+sliderV.get()])
```

```
    mask=cv2.inRange(imgHSV,lowerBound,upperBound)
    mask = cv2.blur(mask,(6,6))                        # ajoute du flou a l'image
    mask = cv2.erode(mask, None, iterations=2)         # retire les parasites
    mask = cv2.dilate(mask, None, iterations=2)        # retire les parasites

    cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    cnts = cnts[0] if imutils.is_cv2() else cnts[1]
    center = None

    cv2.circle(img, (int(consigneX), int(consigneY)), int(4),(255, 0, 0), 2)
    if showCalqueCalibrationBool == True:
        cv2.circle(img, (240,240), 220,(255, 0, 0), 2)
        cv2.circle(img, (240,240), 160,(255, 0, 0), 2)
        cv2.line(img, (240, 240), (240, 240+160), (255,0,0), 2)
        cv2.line(img, (240, 240), (240+138, 240-80), (255,0,0), 2)
        cv2.line(img, (240, 240), (240-138, 240-80), (255,0,0), 2)
    if len(cnts) > 0:
        c = max(cnts, key=cv2.contourArea)
        timeInterval = time.time() - start_time
        (x, y), radius = cv2.minEnclosingCircle(c)
        if radius > 10:
            cv2.putText(img,str(int(x)) + ";" + str(int(y)).format(0, 0),(int(x)-50, int(y)-50), cv2.FONT_HERSHEY_SIMPLEX,1, (255, 255, 255), 2)
            cv2.circle(img, (int(x), int(y)), int(radius),(0, 255, 255), 2)
            PIDcontrol(int(x),int(y),prevX,prevY,consigneX,consigneY)
            start_time = time.time()
    else:
        sommeErreurX, sommeErreurY = 0,0


    if showVideoWindow == True:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = Image.fromarray(img)
        imgtk = ImageTk.PhotoImage(image=img)
        lmain.imgtk = imgtk
        lmain.configure(image=imgtk)
    lmain.after(5, main)

    drawWithBall()
    if prevConsigneX != consigneX or prevConsigneY != consigneY:
        sommeErreurX, sommeErreurY = 0,0

    paintGraph()
    prevX,prevY = int(x), int(y)
    prevConsigneX, prevConsigneY = consigneX, consigneY
    prevAlpha = alpha
    prevBeta = beta

```

```python
    #print("FPS: ", 1.0 / (time.time() - start_timeFPS))



FrameVideoControl = tk.LabelFrame(controllerWindow, text="Video controle")
FrameVideoControl.place(x=20,y=20,width=380)
BRetourVideo = tk.Button(FrameVideoControl, text="Afficher le retour video", command=showCameraFrameWindow)
BRetourVideo.pack()
BPositionCalibration = tk.Button(FrameVideoControl, text="Calque", command=showCalqueCalibration)
BPositionCalibration.place(x=290,y=0)

sliderH = tk.Scale(FrameVideoControl, from_=0, to=100, orient="horizontal", label="Sensibilite H", length=350, tickinterval = 10)
sliderH.set(sliderHDefault)
sliderH.pack()
sliderS = tk.Scale(FrameVideoControl, from_=0, to=100, orient="horizontal", label="Sensibilite S", length=350, tickinterval = 10)
sliderS.set(sliderSDefault)
sliderS.pack()
sliderV = tk.Scale(FrameVideoControl, from_=0, to=100, orient="horizontal", label="Sensibilite V", length=350, tickinterval = 10)
sliderV.set(sliderVDefault)
sliderV.pack()



FrameServosControl = tk.LabelFrame(controllerWindow, text="Servos controle")
FrameServosControl.place(x=20,y=315,width=380)
BAbaissementPlateau = tk.Button(FrameServosControl, text="Ranger les bras", command=rangerPlateau)
BAbaissementPlateau.pack()
BElevationBras = tk.Button(FrameServosControl, text="Mettre en place le plateau", command=eleverPlateau)
BElevationBras.pack()
BTesterServos = tk.Button(FrameServosControl, text="Tester les servomoteurs", command=servosTest)
BTesterServos.pack()
BStartBalance = tk.Button(FrameServosControl, text="Demarrer", command=startBalance, highlightbackground = "#36db8b")
BStartBalance.pack()



FramePIDCoef = tk.LabelFrame(controllerWindow, text="PID coefficients")
FramePIDCoef.place(x=420,y=20,width=380)
BafficherGraph = tk.Button(FramePIDCoef, text="Afficher graphique", command=showGraphWindow)
BafficherGraph.pack()
sliderCoefP = tk.Scale(FramePIDCoef, from_=0, to=15, orient="
```

```
horizontal", label="P", length=350, tickinterval = 3, resolution=0.01)
sliderCoefP.set(sliderCoefPDefault)
sliderCoefP.pack()
sliderCoefI = tk.Scale(FramePIDCoef, from_=0, to=1, orient="horizontal", label="I", length=350, tickinterval = 0.2, resolution=0.001)
sliderCoefI.set(sliderCoefIDefault)
sliderCoefI.pack()
sliderCoefD = tk.Scale(FramePIDCoef, from_=0, to=10, orient="horizontal", label="D", length=350, tickinterval = 2, resolution=0.01)
sliderCoefD.set(sliderCoefDDefault)
sliderCoefD.pack()


FrameBallControl = tk.LabelFrame(controllerWindow, text="Bille controle")
FrameBallControl.place(x=420,y=315,width=380, height= 132)
BballDrawCircle = tk.Button(FrameBallControl, text="Faire tourner la bille en cercle", command=startDrawCircle)
BballDrawCircle.pack()
BballDrawEight = tk.Button(FrameBallControl, text="Faire tourner la bille en huit", command=startDrawEight)
BballDrawEight.pack()




label = tk.Label(controllerWindow, text="Arduino deconnecte  ", fg="red", anchor="ne")
label.pack(fill="both")
BReset = tk.Button(controllerWindow, text = "Reset", command = resetSlider)
BReset.place(x=20, y=460)
BConnect = tk.Button(controllerWindow, text = "Connexion", command = connectArduino, background="black")
BConnect.place(x=100, y=460)
BQuit = tk.Button(controllerWindow, text = "Quitter", command = endProgam)
BQuit.place(x=730, y=460)


showGraphPositionX = tk.IntVar()
showGraphPositionX.set(1)
CheckbuttonPositionX = tk.Checkbutton(graphWindow, text="Position en X", variable=showGraphPositionX, command=refreshGraph)
CheckbuttonPositionX.place(x=500,y=20)
showGraphPositionY = tk.IntVar()
showGraphPositionY.set(1)
CheckbuttonPositionY = tk.Checkbutton(graphWindow, text="Position en Y", variable=showGraphPositionY, command=refreshGraph)
CheckbuttonPositionY.place(x=500,y=40)
showGraphAlpha = tk.IntVar()
CheckbuttonAlpha = tk.Checkbutton(graphWindow, text="Inclinaison du plateau", variable=showGraphAlpha, command=refreshGraph)
CheckbuttonAlpha.place(x=500,y=60)
```

```
videoWindow.protocol("WM_DELETE_WINDOW",donothing)
videoWindow.bind("<Button-2>",getMouseClickPosition)
videoWindow.bind("<Button-1>",setConsigneWithMouse)

main()
tk.mainloop()
```

G arduinoCode.ino

```
#include <Servo.h>

Servo servoA;
Servo servoB;
Servo servoC;

int ledTemoin = 8;
float angleA = 5;
float angleB = 5;
float angleC = 5;

void setup() {
  Serial.begin(19200);
  pinMode(ledTemoin, OUTPUT); // led temoin
  digitalWrite(ledTemoin,HIGH);
  servoA.attach(9);     //servo A
  servoB.attach(11);    //servo B
  servoC.attach(10);    //servo C
  delay(1000);
  servoA.writeMicroseconds((-2165+1260)*float(angleA)/90 + 2165);
  servoB.writeMicroseconds((-1975+1130)*float(angleB)/90 + 1975);
  servoC.writeMicroseconds((-1990+1130)*float(angleC)/90 + 1990);
}

int count = 0;

void loop() {
  digitalWrite(ledTemoin , millis() / 500 % 2 ); // led temoin clignotement

  if(Serial.available() > 0) {
    String a = Serial.readStringUntil('\n');
    if(a == "descendreBras"){
      angleA = 5;
      angleB = 5;
      angleC = 5;
    }else{
      a.remove(0,1);
      a.remove(a.length() - 1,1);
      angleA = getValue(a, ',', 0).toFloat();
```

```
      angleB = getValue(a, ',', 1).toFloat();
      angleC = getValue(a, ',', 2).toFloat();
    }
    servoA.writeMicroseconds((-2165+1260)*float(angleA)/90 + 2165);
    servoB.writeMicroseconds((-1975+1130)*float(angleB)/90 + 1975);
    servoC.writeMicroseconds((-1990+1130)*float(angleC)/90 + 1990);
  }

}

//La fonction ci-dessous a ete trouvee telle quelle sur: https://stackoverflow.com/questions/29671455/how-to-split-a-string-using-a-specific-delimiter-in-arduino
String getValue(String data, char separator, int index) {
    int found = 0;
    int strIndex[] = { 0, -1 };
    int maxIndex = data.length() - 1;

    for (int i = 0; i <= maxIndex && found <= index; i++) {
        if (data.charAt(i) == separator || i == maxIndex) {
            found++;
            strIndex[0] = strIndex[1] + 1;
            strIndex[1] = (i == maxIndex) ? i+1 : i;
        }
    }
    return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
}
```

H Détermination des angles θ des servomoteurs

Vecteur $\vec{v}$: $\begin{pmatrix} cos\beta sin\alpha \\ sin\beta sin\alpha \\ cos\alpha \end{pmatrix} \perp \pi$

Plan du plateau (π) : $cos\beta sin\alpha x + sin\beta sin\alpha y + cos\alpha z + d = 0$

Plan vertical contenant le bras du moteur A $(planA)$: $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \wedge \begin{pmatrix} -cos(30) \\ -sin(30) \\ 0 \end{pmatrix} = \begin{pmatrix} sin(30) \\ -cos(30) \\ 0 \end{pmatrix} \Rightarrow sin(30)x - cos(30)y = 0$

Plan vertical contenant le bras du moteur B $(planB)$: $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \wedge \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix} \Rightarrow -x = 0$

Plan vertical contenant le bras du moteur C $(planC)$: $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \wedge \begin{pmatrix} cos(30) \\ -sin(30) \\ 0 \end{pmatrix} = \begin{pmatrix} sin(30) \\ cos(30) \\ 0 \end{pmatrix} \Rightarrow sin(30)x + cos(30)y = 0$

Vecteur $\vec{a}$ se trouvant sur la droite $planA \cap \pi)$: $\begin{pmatrix} cos\beta sin\alpha \\ sin\beta sin\alpha \\ cos\alpha \end{pmatrix} \wedge \begin{pmatrix} sin(30) \\ -cos(30) \\ 0 \end{pmatrix} = \begin{pmatrix} cos\alpha cos(30) \\ cos\alpha sin(30) \\ -cos(30)cos\beta sin\alpha - sin\beta sin\alpha sin(30) \end{pmatrix}$

Vecteur $\vec{b}$ se trouvant sur la droite $planB \cap \pi)$: $\begin{pmatrix} cos\beta sin\alpha \\ sin\beta sin\alpha \\ cos\alpha \end{pmatrix} \wedge \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ -cos\alpha \\ sin\beta sin\alpha \end{pmatrix}$

Vecteur $\vec{c}$ se trouvant sur la droite $planC \cap \pi)$: $\begin{pmatrix} cos\beta sin\alpha \\ sin\beta sin\alpha \\ cos\alpha \end{pmatrix} \wedge \begin{pmatrix} sin(30) \\ cos(30) \\ 0 \end{pmatrix} = \begin{pmatrix} -cos\alpha cos(30) \\ cos\alpha sin(30) \\ cos(30)cos\beta sin\alpha - sin\beta sin\alpha sin(30) \end{pmatrix}$

d est la hauteur du plateau au-dessus des moteurs lorsque le plateau est à plat.

Point A, position de l'extrémité du bras du moteur A : $A = \begin{pmatrix} 0 \\ 0 \\ d \end{pmatrix} + k \begin{pmatrix} cos\alpha cos(30) \\ cos\alpha sin(30) \\ -cos(30)cos\beta sin\alpha - sin\beta sin\alpha sin(30) \end{pmatrix} = \begin{pmatrix} X_a \\ Y_a \\ Z_a \end{pmatrix}$

Point B, position de l'extrémité du bras du moteur B : $B = \begin{pmatrix} 0 \\ 0 \\ d \end{pmatrix} + m \begin{pmatrix} 0 \\ -cos\alpha \\ sin\beta sin\alpha) \end{pmatrix} = \begin{pmatrix} X_b \\ Y_b \\ Z_b \end{pmatrix}$

Point C, position de l'extrémité du bras du moteur C : $C = \begin{pmatrix} 0 \\ 0 \\ d \end{pmatrix} + t \begin{pmatrix} -cos\alpha cos(30) \\ cos\alpha sin(30) \\ cos(30)cos\beta sin\alpha - sin\beta sin\alpha sin(30) \end{pmatrix} = \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix}$

$$\left\|\overrightarrow{AB}\right\| = D \Rightarrow \left\|\overrightarrow{OB} - \overrightarrow{OA}\right\| = D = \left\| \begin{pmatrix} 0 \\ -mcos\alpha \\ msin\beta sin\alpha + d) \end{pmatrix} - \begin{pmatrix} kcos\alpha cos(30) \\ kcos\alpha sin(30) \\ -kcos(30)cos\beta sin\alpha - ksin\beta sin\alpha sin(30) + d \end{pmatrix} \right\|$$

$$\left\|\overrightarrow{BC}\right\| = D \Rightarrow \left\|\overrightarrow{OC} - \overrightarrow{OB}\right\| = D = \left\| \begin{pmatrix} -tcos\alpha cos(30) \\ tcos\alpha sin(30) \\ tcos\beta sin\alpha cos(30) - tsin\beta sin\alpha sin(30) + d \end{pmatrix} - \begin{pmatrix} 0 \\ -mcos\alpha \\ msin\beta sin\alpha + d) \end{pmatrix} \right\|$$

$$\left\|\overrightarrow{CA}\right\| = D \Rightarrow \left\|\overrightarrow{OA} - \overrightarrow{OC}\right\| = D = \left\| \begin{pmatrix} kcos\alpha cos(30) \\ kcos\alpha sin(30) \\ -kcos(30)cos\beta sin\alpha - ksin\beta sin\alpha sin(30) + d \end{pmatrix} - \begin{pmatrix} -tcos\alpha cos(30) \\ tcos\alpha sin(30) \\ tcos\beta sin\alpha cos(30) - tsin\beta sin\alpha sin(30) + d \end{pmatrix} \right\|$$

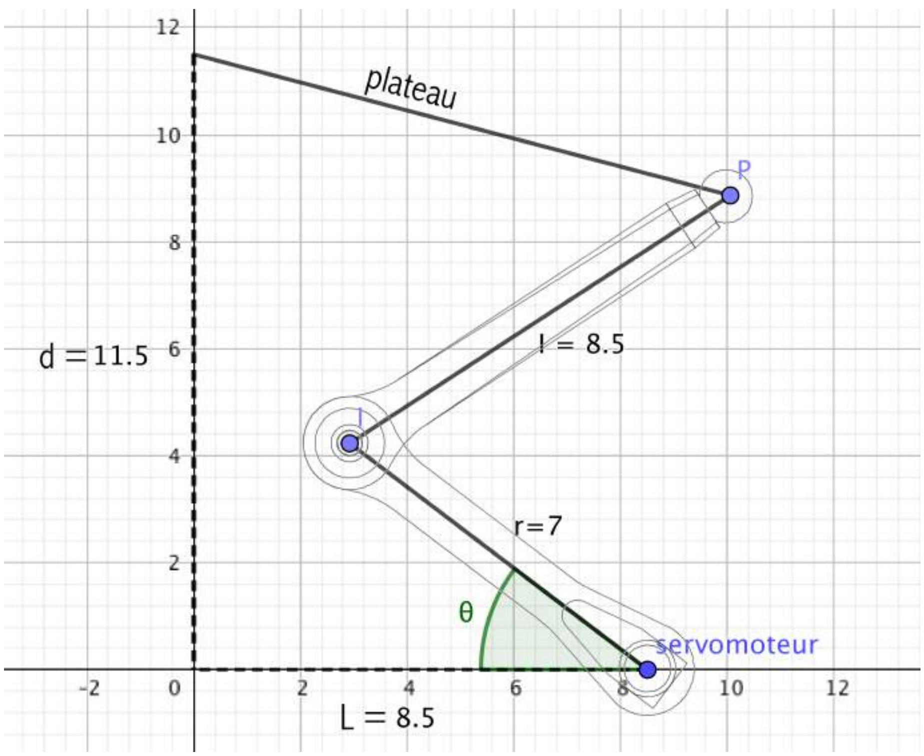
$$\begin{cases} (-kcos\alpha cos(30))^2 + (-mcos\alpha - kcos\alpha sin(30))^2 + (msin\beta sin\alpha + d + kcos(30)cos\beta sin\alpha + ksin\beta sin\alpha sin(30) - d)^2 = D^2 \\ (-tcos\alpha cos(30))^2 + (tcos\alpha sin(30) + mcos\alpha)^2 + (tcos\beta sin\alpha cos(30) - tsin\beta sin\alpha sin(30) + d - msin\beta sin\alpha - d)^2 = D^2 \\ (kcos\alpha cos(30) + tcos\alpha cos(30))^2 + (kcos\alpha sin(30) - tcos\alpha sin(30))^2 + (-kcos(30)cos\beta sin\alpha - ksin\beta sin\alpha sin(30) + d - tcos\beta sin\alpha cos(30) \\ +tsin\beta sin\alpha sin(30) - d)^2 = D^2 \end{cases}$$

Point I, emplacement de l'articulation du bras d'un moteur : $I(L - rcos\theta; rsin\theta)$
Point P, emplacement de l'extrémité du bras d'un moteur : $P(\sqrt{X_p^2 + Y_p^2}; Z_p)$

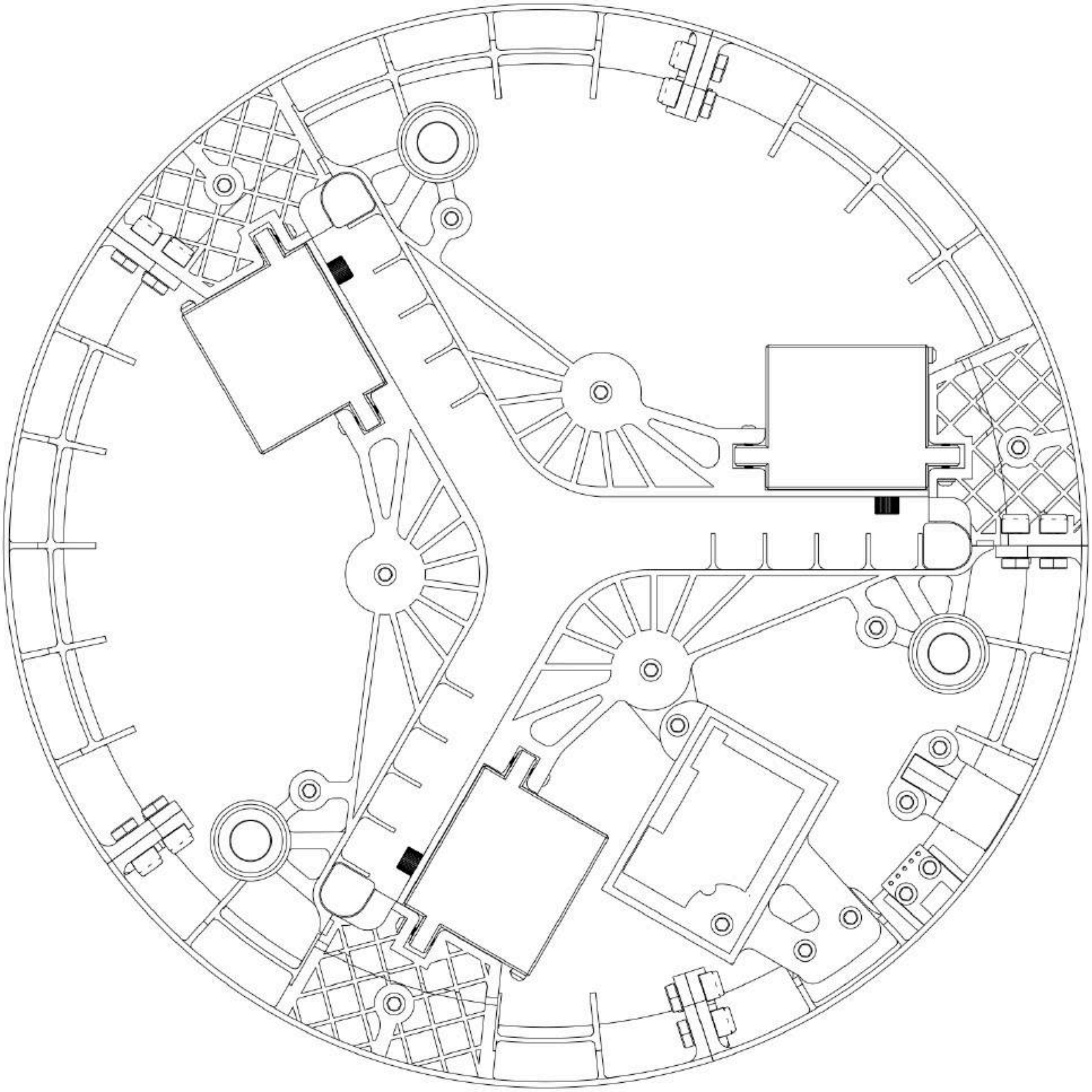
$$\left\|\overrightarrow{PI}\right\| = l$$

$$\overrightarrow{PI} = \begin{pmatrix} L - rcos\theta \\ rsin\theta \end{pmatrix} - \begin{pmatrix} \sqrt{X_p^2 + Y_p^2} \\ Z_p \end{pmatrix}$$

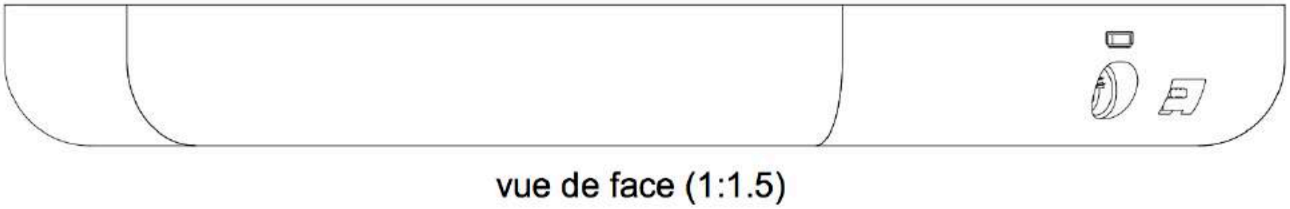
$$(L - rcos\theta - \sqrt{X_p^2 + Y_p^2})^2 + (rsin\theta - Z_p)^2 = l^2$$



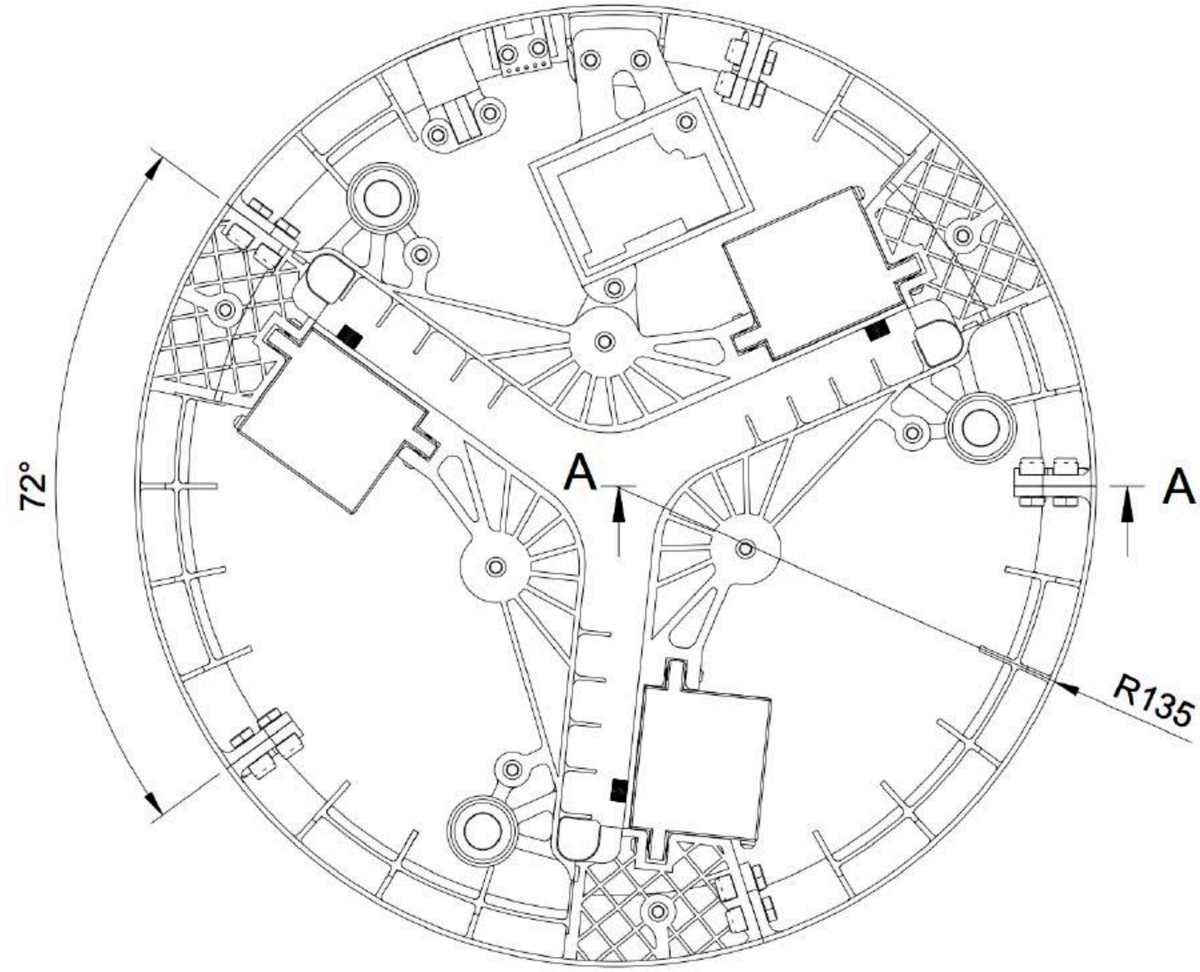
I Quelques plans du projet



vue de dessus (1:1.5)

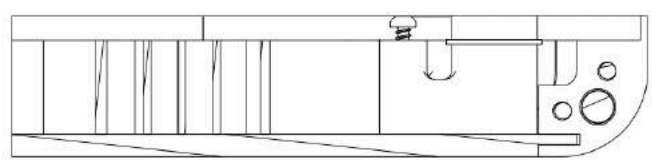


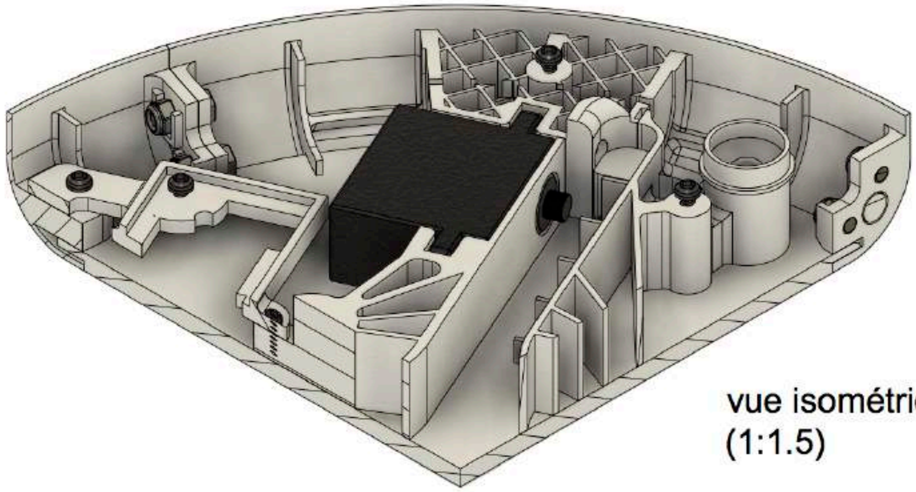
vue de face (1:1.5)



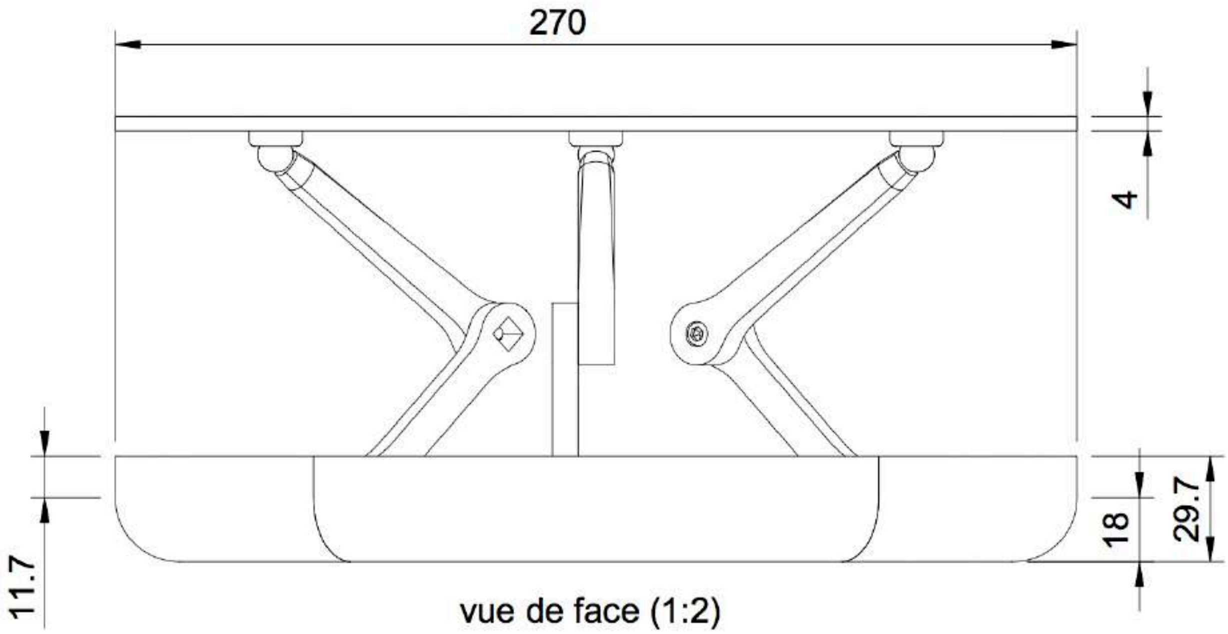
vue de dessus (1:2)

A-A (1:1.5)



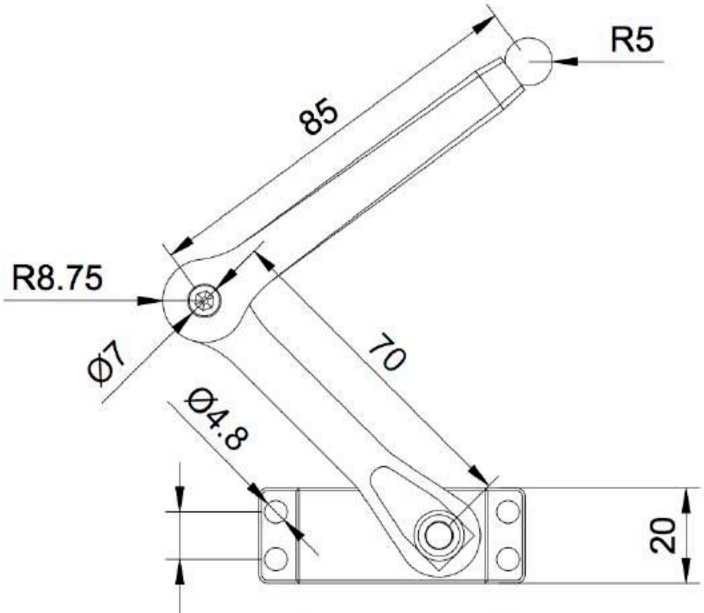


vue isométrique
(1:1.5)

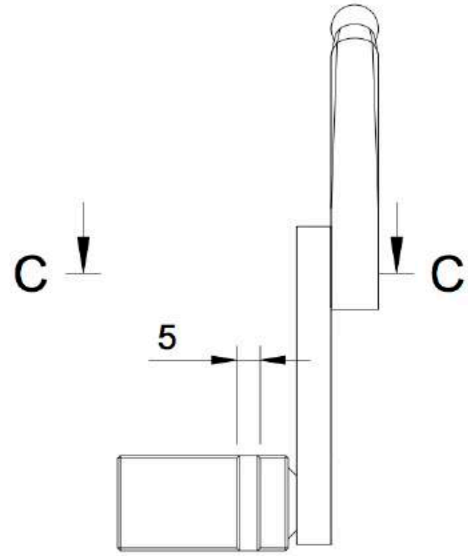


vue de face (1:2)

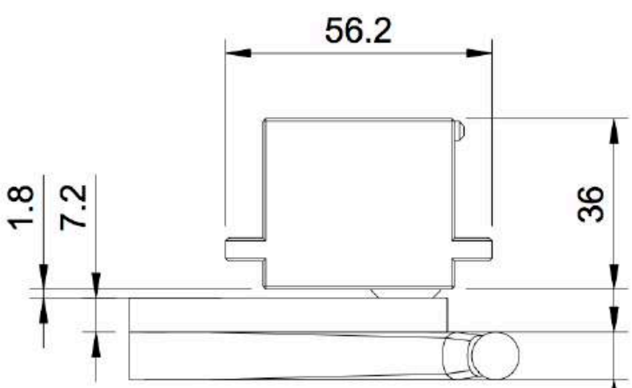

vue isométrique (1:2)



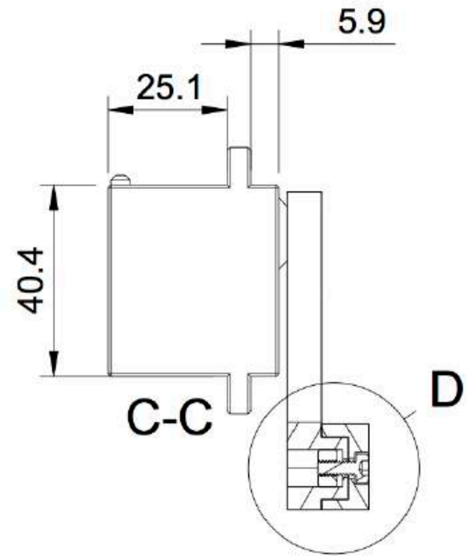
vue de face (1:1.5)



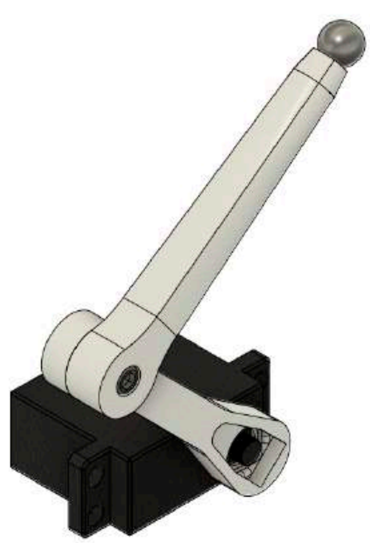
vue de gauche (1:1.5)

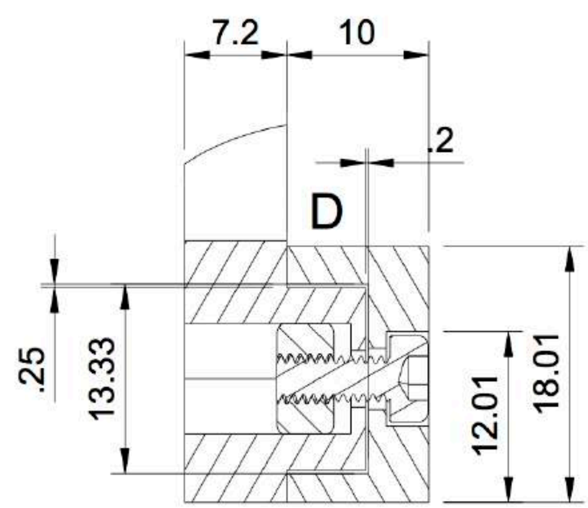


vue de dessus (1:1.5)

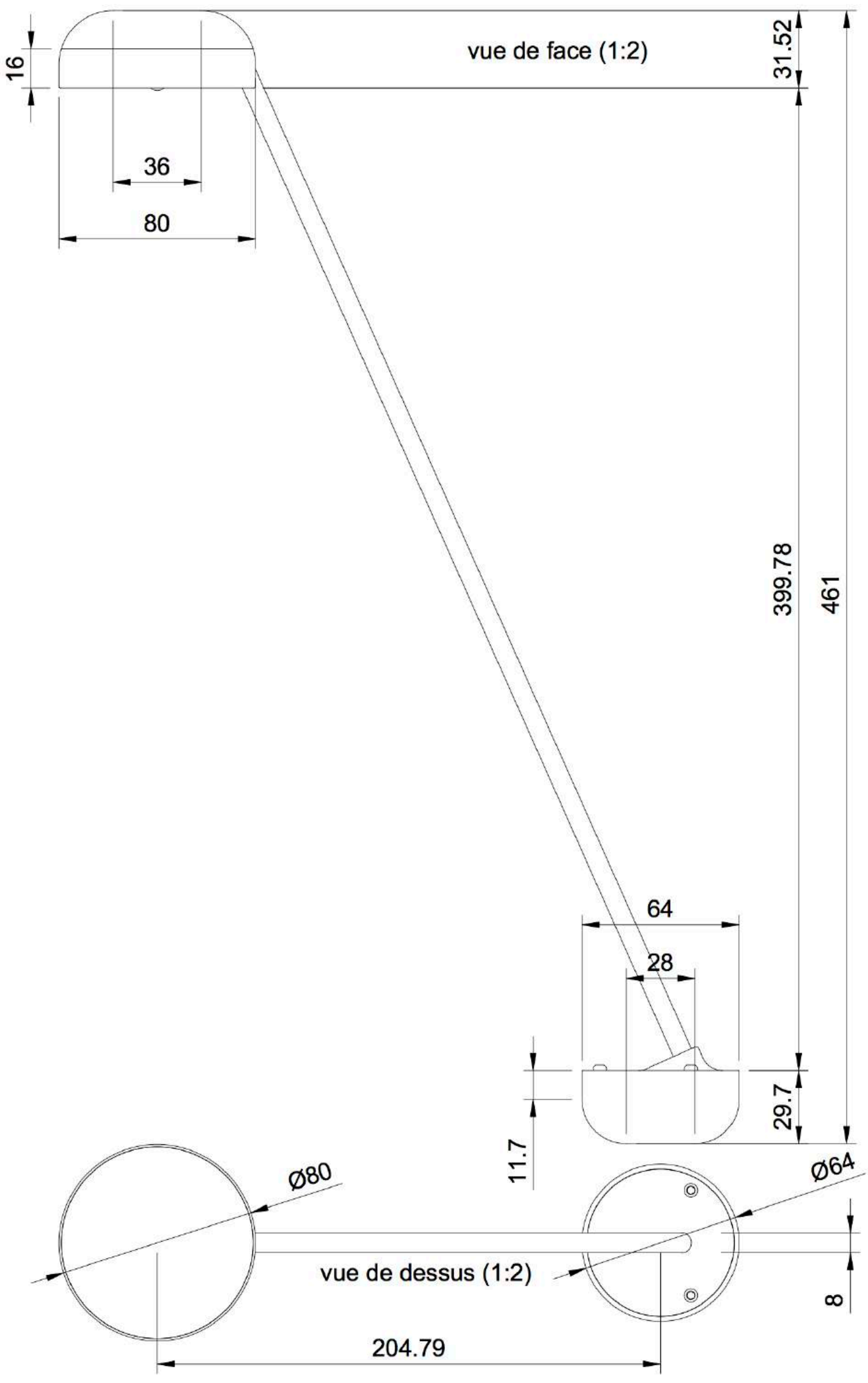


vue de dessus (1:1.5)

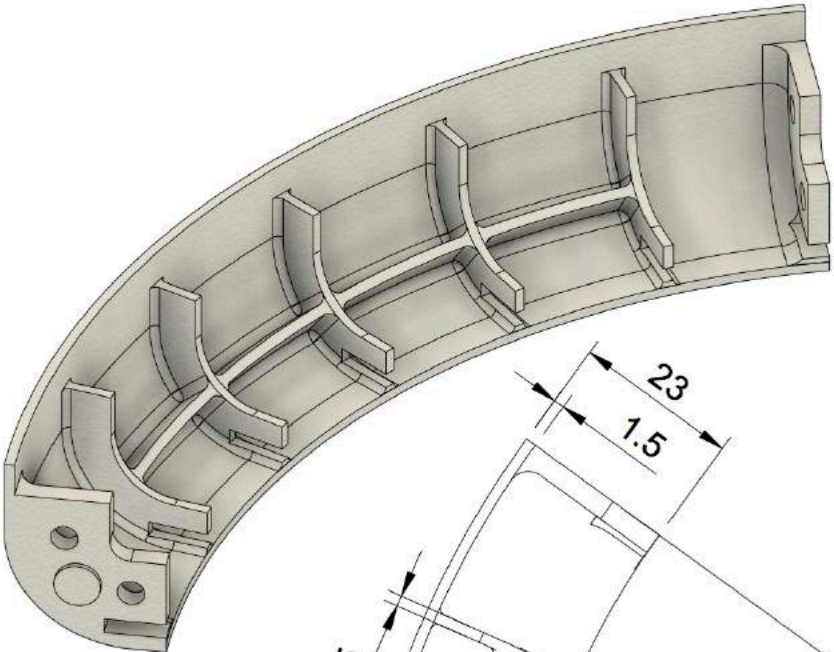

vue isométrique du servomoteur et son bras (1:1.5)



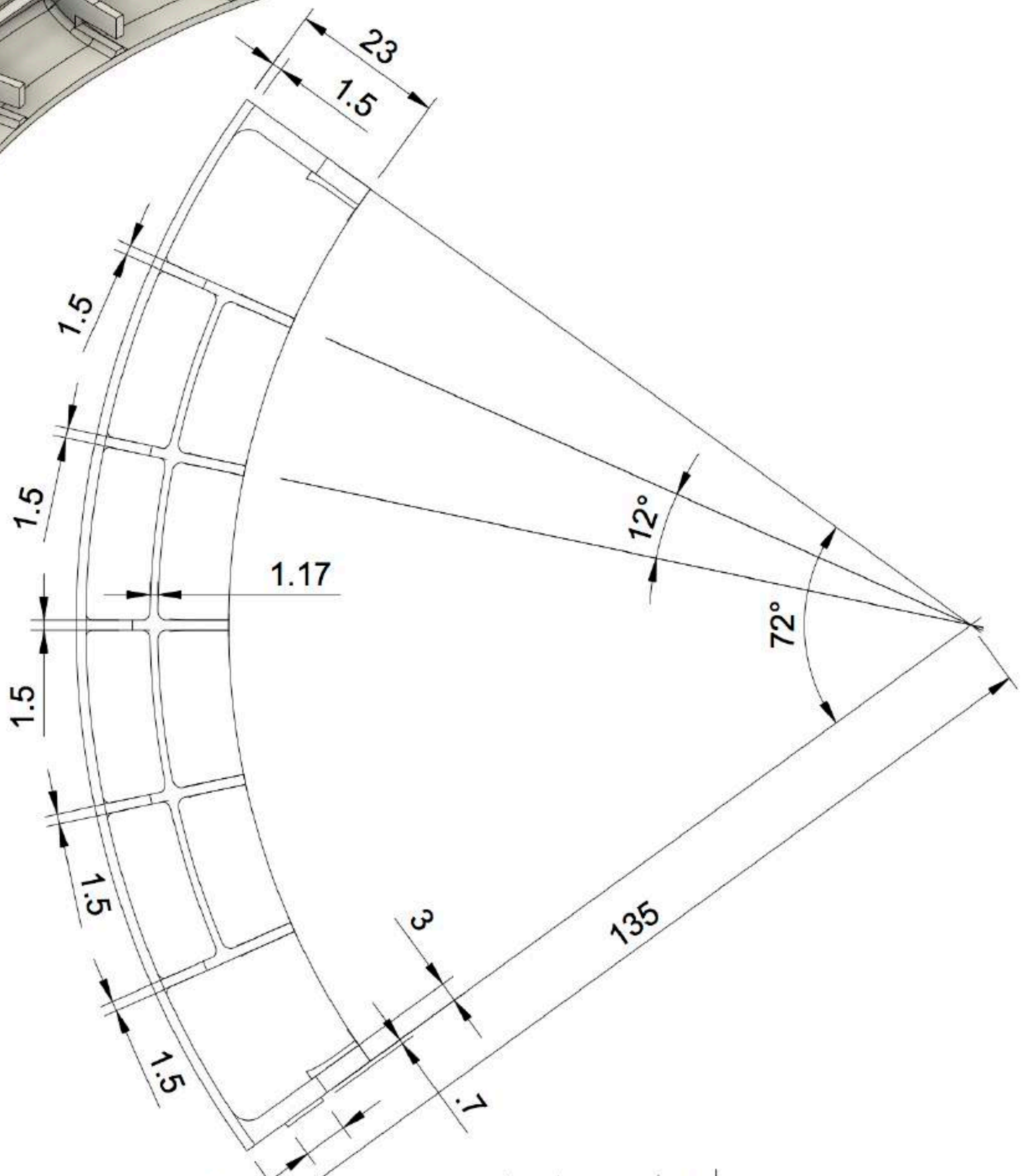
détail de l'articulation du bras (2:1)



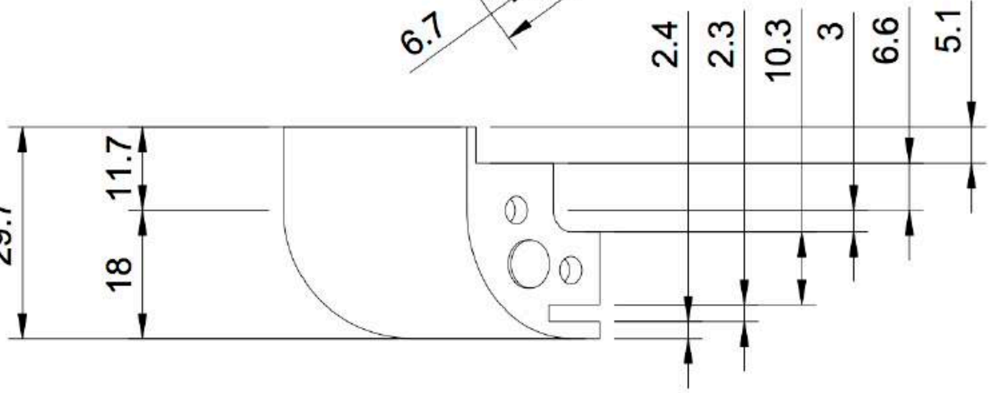
vue de face (1:2)
16
36
80
31.52
399.78
461
64
28
11.7
29.7
Ø80
Ø64
vue de dessus (1:2)
8
204.79



vue isométrique
(1:1)

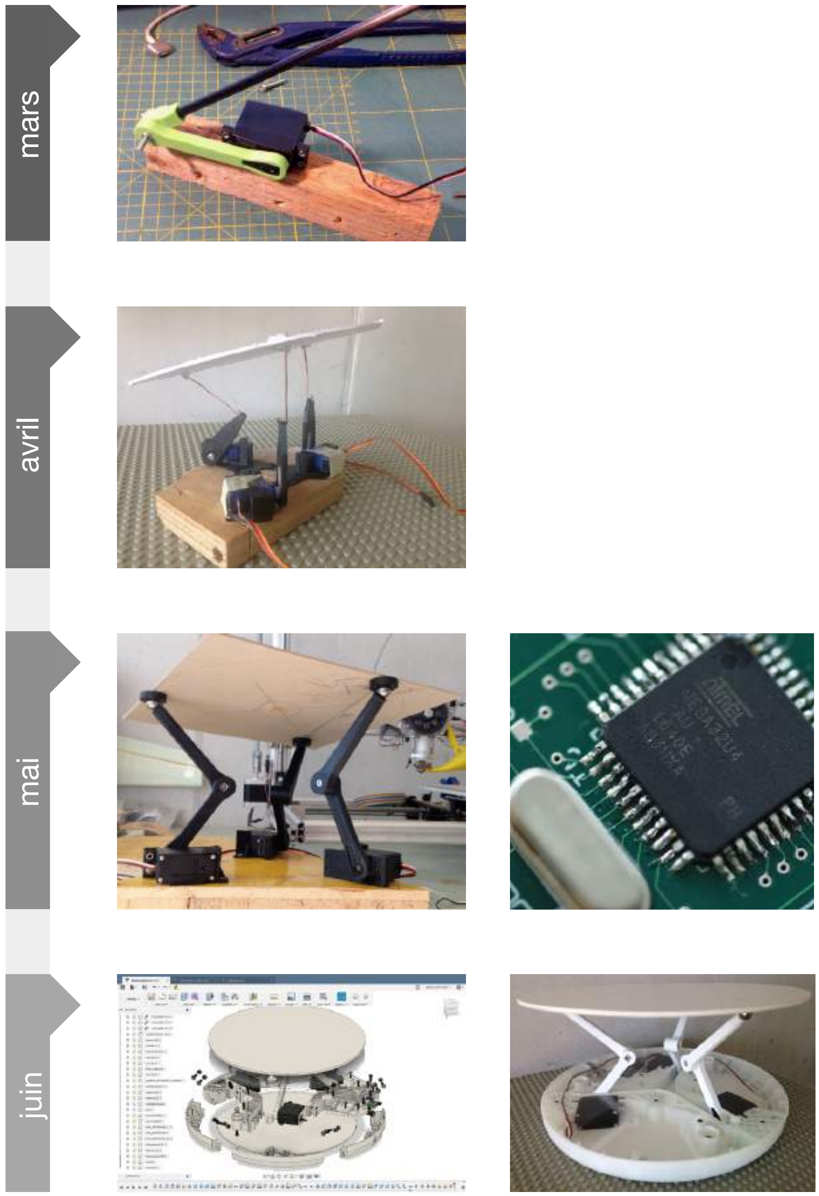


Vue de dessus
(1:1)

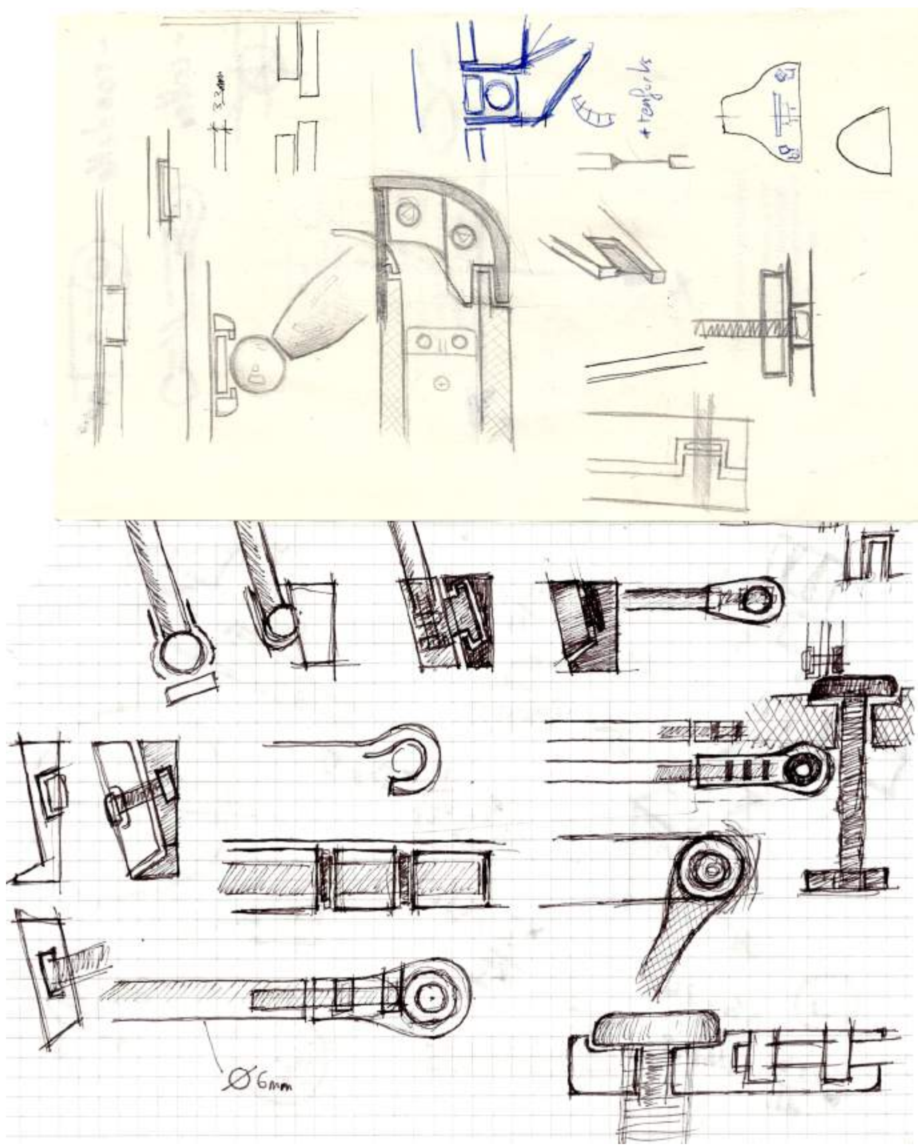


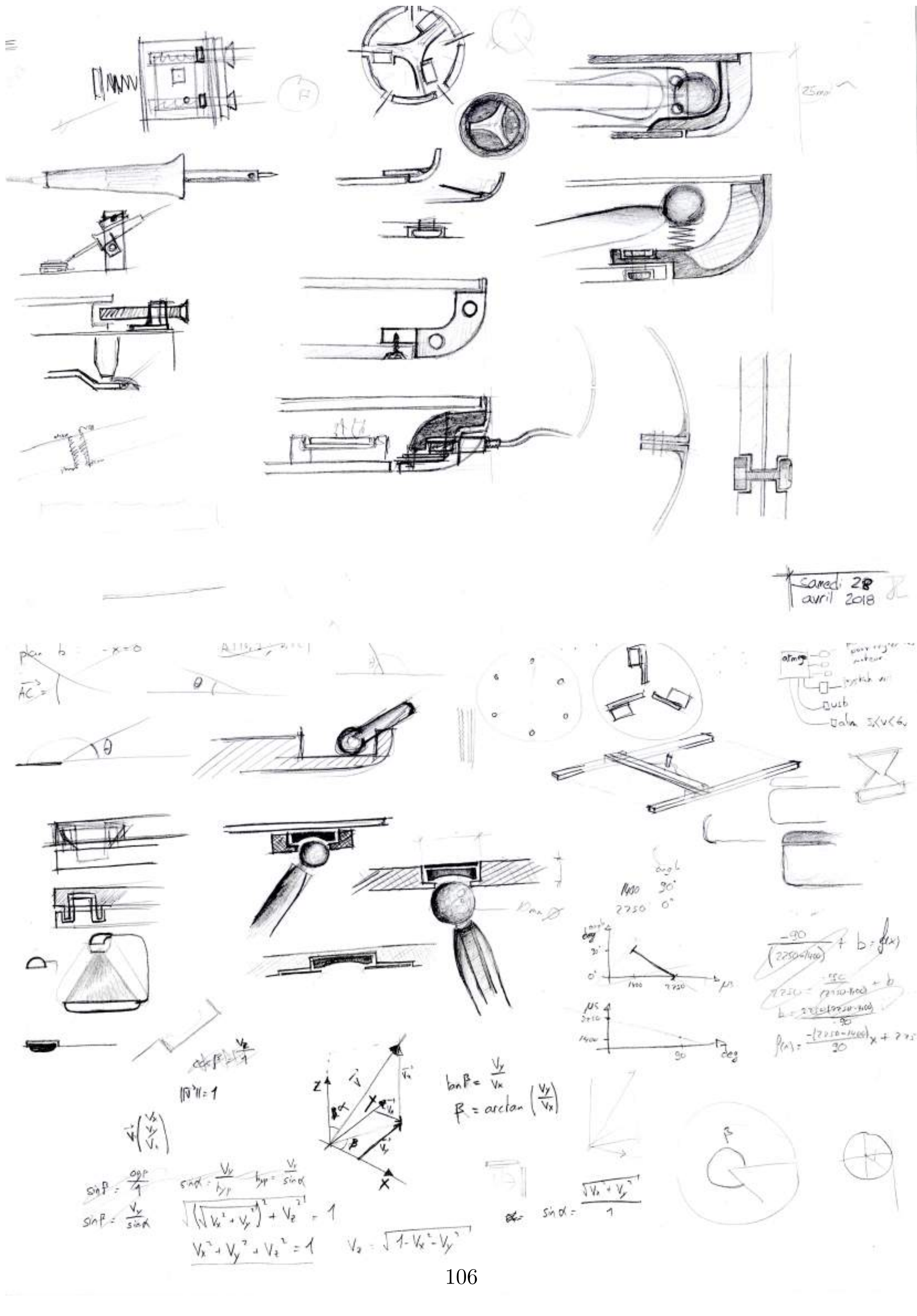
vue de face
(1:1)

J Prototypes



K Quelques esquisses du projet




Samedi 28
avril 2018
$\tan\beta = \frac{V_y}{V_x}$
$\beta = \arctan\left(\frac{V_y}{V_x}\right)$
$\|\vec{V}\| = 1$
$V_x^2 + V_y^2 + V_z^2 = 1$
$V_z = \sqrt{1 - V_x^2 - V_y^2}$