ÉCOLE UNIVERSITAIRE
DE PHYSIQUE ET D'INGÉNIERIE
Université Clermont Auvergne

31/03/2025

Rapport

Rapport du projet de mécatronique

Etablissement : École Universitaire de Physique et d'Ingénierie – Aubière

Élèves : PRADEAU Romaric ; DEKIMECHE **Sarra**
Enseignant : Monsieur Omar AIT-AIDER
Monsieur Fabrice DUMAS

SOMMAIRE

Présentation du projet

Dans le cadre de notre projet, nous avons été amenés à concevoir et réaliser un lanceur balistique à visée automatique. Ce système intelligent a pour objectif de détecter une cible connue à l'aide d'un programme de vision artificielle, et de s'orienter vers elle à l'aide d'une plateforme motorisée, puis de lancer un projectile.

La plateforme en question est de type Pan and Tilt, qui permet au lanceur d'effectuer des mouvements d'orientation horizontale (pan) et verticale (tilt). Ce système assure une visée précise de la cible, quelle que soient sa position et sa hauteur.

Le projet sera structuré autour de trois axes principaux :

- **La conception mécanique**, qui inclut la réalisation de la plateforme motorisée Pan and Tilt ainsi que le développement du système de propulsion à ressort pour le lancement des projectiles.
- **La partie électronique**, qui consiste à mettre en place les différentes connexions sur la carte électronique, notamment la gestion de la carte de puissance pour les moteurs, l'Arduino et les autres composants nécessaires au bon fonctionnement du système.
- **La programmation**, qui vise à piloter les moteurs et la caméra. Cette dernière permet la détection de la cible. L'ensemble du programme assure la synchronisation entre la détection, l'orientation de la plateforme et le déclenchement du lancement, afin de permettre au système d'accomplir sa mission de manière autonome.

Afin de mieux visualiser notre système voici son diagramme fonctionnel :

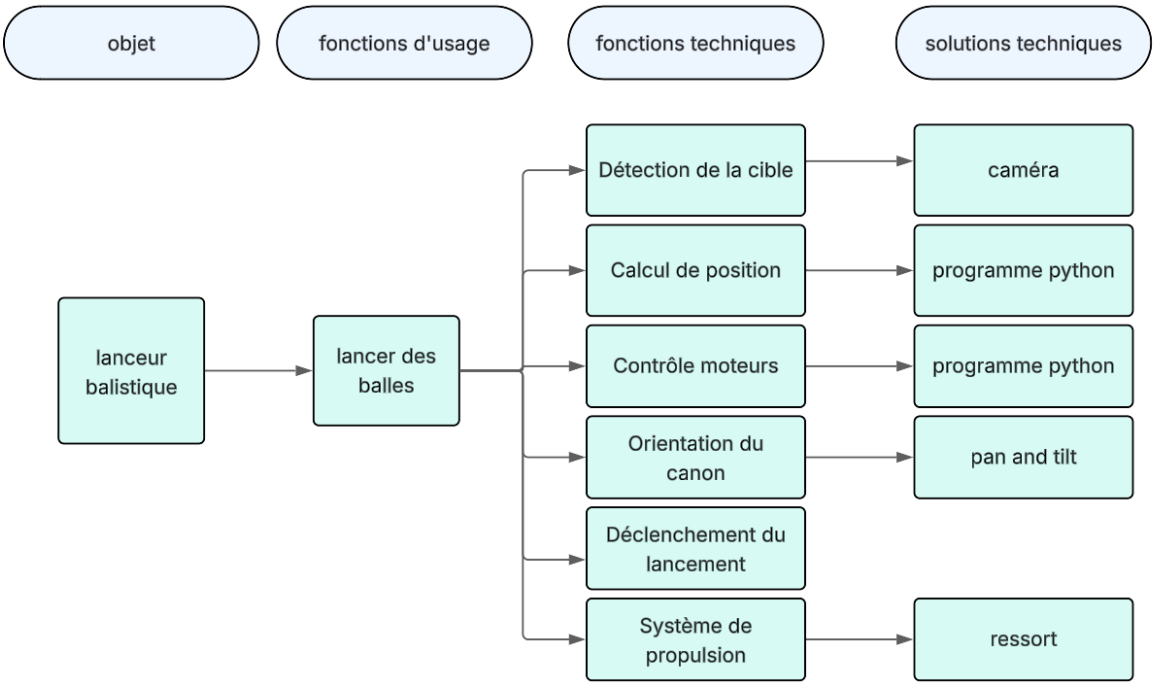


Figure 1:Schéma fonctionnel

Planning et répartition

Planning

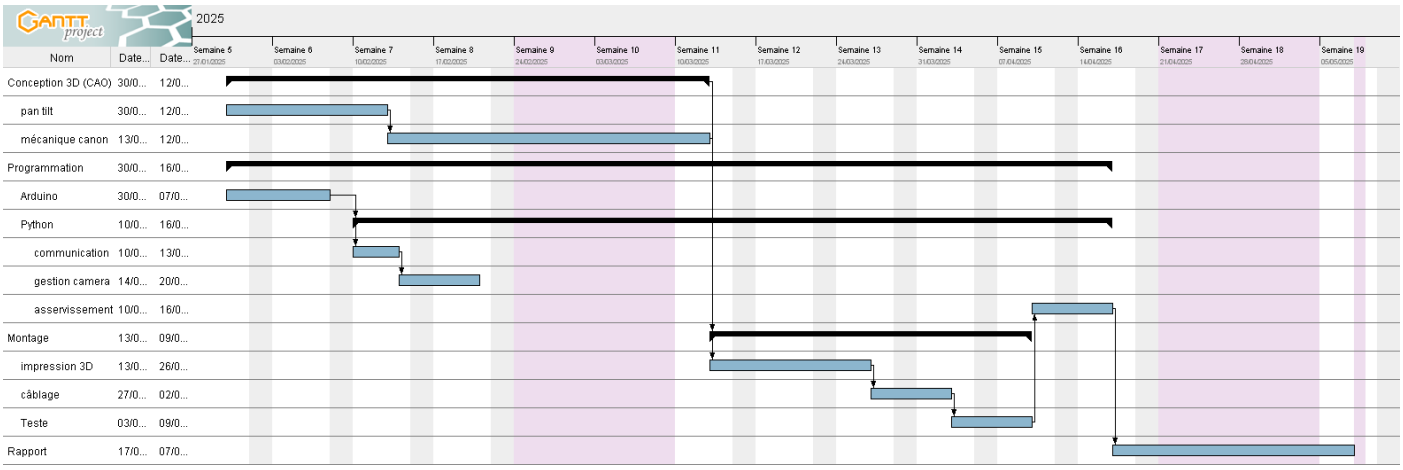


Figure 2: Diagramme de Gantt

Voilà à quoi a ressemblé notre planning, dans l'ensemble il a été respecté. Mais comme toujours parfois des erreurs ou des problèmes non liés à nous nous ont fait prendre plus de temps.

Répartition

Pour la répartition, nous avons fait en sorte de pouvoir manipuler tout un peu mais ensuite on s'est répartie en fonction des facilités et des préférences de chacun.

Pour la partie conception mécanique, Sarra s'est chargé de faire la conception sous Catia 3D expérience pour tout le mécanisme du canon, et rechercher des modèles existants pour les déplacements de la tourelle.

Ensuite pour la partie informatique (soit le programme Arduino et le programme python), Romaric s'est occupé à mettre en place un système de communication entre le programme python et la carte de développement Arduino, et gérer les moteurs pas à pas avec la carte Arduino et leur driver respectif.

Répartition des taches :

Tâche	Description	Responsable(s)
Conception 3D (CAO)	Modélisation de la tourelle + support+systéme de tire	Romaric/Sarra
Fabrication / impression 3D	Assemblage des pièces mécaniques	Romaric/Sarra
câblage/montage	Connexions Arduino, moteurs	Sarra/Romaric
Programmation (python)	Caméra/ moteurs	Romaric
rapport		
test	/	Romaric / Sarra

Le diagramme si dessous reprend l'utilisation des deux personnes tout au long du projet. Ceux sont des chiffres non significatifs, ces données ne représentent pas le réel temps passer dans les différentes tâches.

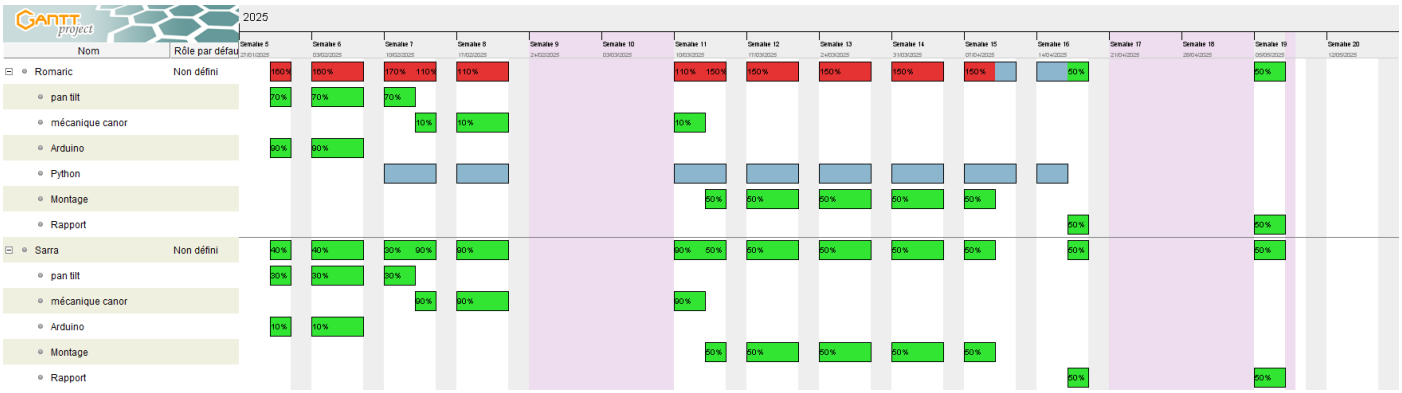


Figure 3: Diagramme des ressources humaines

Coût

Liste du matériel et coût

Désignation	Référence	Photo	Nombre	Prix unitaire	Prix total
Moteur Stepper	17HS15-0404S		3	11,43 €	34,29 €
Driver pour moteur stepper	A4988		3	8,75 €	26,25 €
Carte Arduino	Arduino uno R3		1	23,40 €	23,40 €
Camera Basler	acA1300-200uc		1	479.00 €	479.00 €
Ressort	Sodeman :12070		1	5.73€	5.73 €
Total					568,67 €

Coût de la main d'œuvre

Pour estimer ce cout de la main-d'œuvre, on va partir du temps qu'il faut pour réaliser le prototype

On sépare en deux parties, une partie réalisée par un ingénieur pour la partie théorique et une par techniciens pour la partie câblage et montage mécanique. Ainsi on reprend dans le tableau si dessous le résumé.

Tache	**Temps technicien**	**Temps Ingénieur**
Conception mécanique		**20 h**
Programmation		**45 h**
Montage	**20 h**	
Total	**20 h**	**65 h**

Le salaire horaire pour un technicien est en moyenne 14,84 €

Le salaire horaire pour un ingénieur est en moyenne de 21,98 €

Ainsi le coût de la main-d'œuvre est de 296,8 € pour l'ingénieur et 1428.7 €, soit 1725,5 €

Il faut prendre en compte que ceux sont des valeurs approximatives et ne représente pas la réalité

Coût total

Ainsi pour connaitre le coût total, il suffit d'additionner les différents coûts. Nous n'avons pris en compte que le prix des composants et des mains-d'œuvre, car les coups annexes sont les logiciels utiliser qui sont libre de droit (Arduino et PyCharm pour python) et nous ne connaissons pas le prix du logiciel Catia 3D expérience. Et nous ne prenons pas en compte le prix du plastique utiliser pour la conception par manque de connaissance des produit et la quantité utiliser.

Ainsi notre platine coûterait 2 288,14 € à la fabrication.

Pour commencer, cette partie se divise en deux grandes sections :

- *Le mécanisme Pan Tilt (pivot pan tilt), qui assure l'orientation du système sur deux axes*
- *La partie supérieure, qui intègre le canon ainsi que le système de propulsion destiné au lancement du projectile.*

Conception de la plateforme Pan-Tilt

Dans cette conception, nous avons cherché à réaliser un système Pan-Tilt avec une configuration ou on peut placer les moteurs et les fixer sous la plateforme afin de garantir la stabilité de notre système en abaissant le centre de gravité. Ainsi, l'ensemble des mécanismes d'entraînement (moteurs et engrenages) est situé sous la plaque de support, ce qui libère de l'espace au-dessus pour intégrer plus facilement notre système de propulsion et les moteurs seront moins exposés aux chocs. D'autre part ce type d'architecture facilite une intégration modulaire, où chaque partie (plateforme, canon, caméra) peut être conçue et testée séparément.et bien sûr avec cette configuration on pourra assurer un Double degré de liberté (DOF) ainsi mieux suivre la cible.

- **Description**

Le système est basé sur une transmission par engrenages hélicoïdale, comme illustré ci-dessus. L'objectif était de permettre une double orientation du canon : une rotation horizontale (Pan) ainsi qu'une inclinaison verticale (Tilt). Le mouvement de chaque axe est assuré par un moteur pas à pas qui entraîne un petit pignon. Celui-ci vient engrener avec une roue de plus grand diamètre, ce qui permet de transmettre le mouvement de manière continue sans jeu qui permet d'augmenter au mieux la précision et la réactivité du système (engrenage conique hélicoïdale)

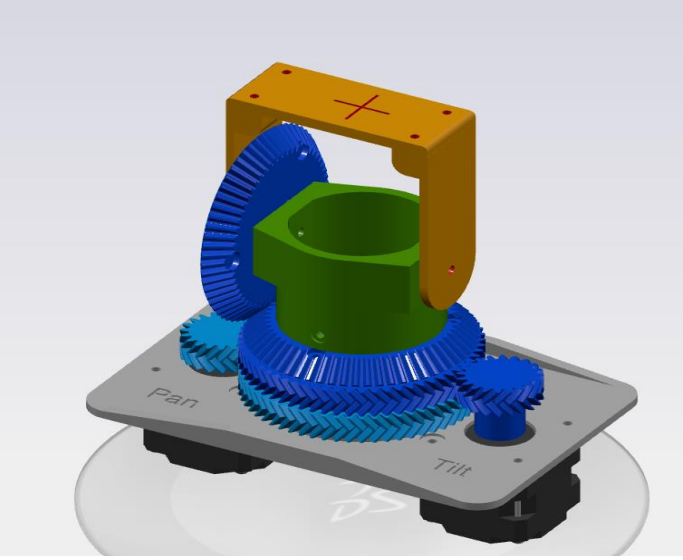


Figure 4: mécanisme Pan Tilt

a) Plaque : Elle sert de support mécanique à l'ensemble du mécanisme. Elle accueille les moteurs en dessous, assurant une fixation stable et une bonne répartition des charges.
b) Tube cylindrique : montée sur les engrenages horizontaux effectue un mouvement de rotation. Elle sert de base à la structure supérieure
c) Bras de support : Il contient les engrenages coniques verticaux qui permettent l'inclinaison (Tilt) de la charge utile (canon/caméra).

Étapes de transmission :

- Le moteur Pan entraîne un petit pignon hélicoïdal, ce pignon engrène avec une couronne hélicoïdale. Lorsque le moteur tourne, le petit pignon fait tourner cette couronne, et donc l'ensemble du tube cylindrique, autour de l'axe vertical.
  Alors la rotation du moteur est transformée en rotation horizontale de toute la partie supérieure du système, ce qui permet d'orienter le système de gauche à droite (Pan).
- Le moteur Tilt, entraîne lui aussi un petit pignon hélicoïdal. Ce pignon engrène avec une roue dentée hélicoïdale elle-même entraine un engrenage conique montée sur l'axe horizontal, qui entrainera a son tour un autre engrenage conique fixée au bras de support et comme ce dernier est solidaire au bras de support: lorsque le pignon tourne, il incline le bras vers l'avant ou vers l'arrière.

Les modifications apportées :

La structure a été adaptée au type de moteur utilisé, à savoir des moteurs pas-à-pas .Ce choix nous permettent de bénéficier d'un meilleur contrôle angulaire, essentiel pour la précision de visée. L'adaptation a nécessité la modification des diamètres pour en engrenage encastrer avec les deux moteurs et l'ajustement des perçages pour assurer un montage optimal. Ensuite, nous avons opté pour des engrenages de module 1, un standard qui offre facilité d'impression 3D.

Rapport de transmission :

- z : nombre de dents
- n : nombre de contacts entre les roues

$r=(-1)^n \times \frac{\prod Z\ roues\ menante}{\prod Z\ roues\ menanée} = -\frac{z_1 \times z_3 \times z_5}{z_2 \times z_6 \times z_4} = -\frac{20 \times 20 \times 60}{60 \times 60 \times 60} = -0.1 < 1$

Alors on a un réducteur de vitesse

Schéma cinématique :

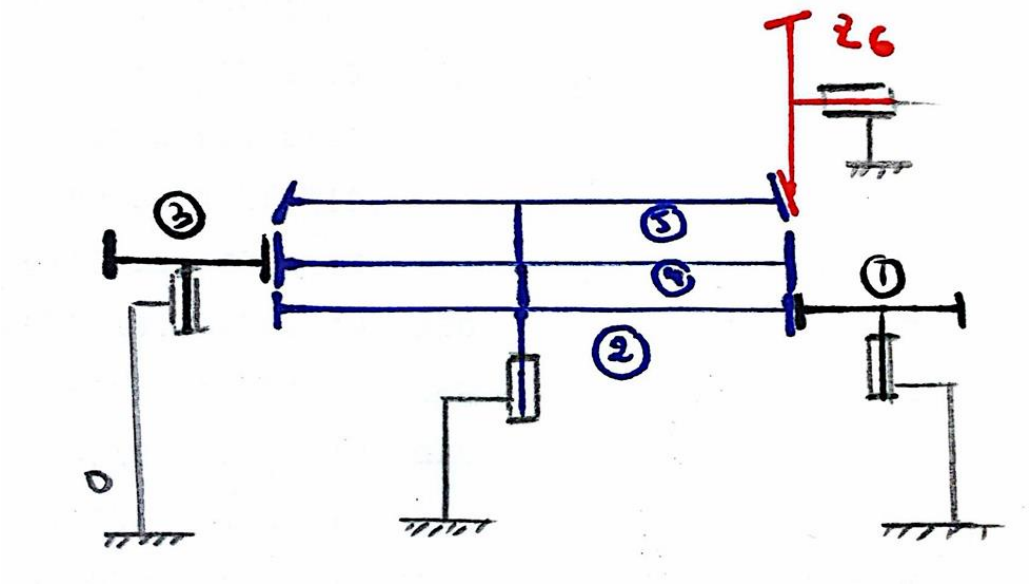


Figure 5: Schéma cinématique

Système de propulsion

La base de ce système est de comprimer un ressort en le tirant en arrière via un système pignon crémaillère en enlevant 15% c'est-à-dire 3 dents du nombre totale des dents du pignon. Cette absence de dents sur une portion du pignon permet de créer une zone de désengagement temporaire avec la crémaillère. Lorsque cette zone atteint la position d'engrènement, le pignon cesse d'entraîner la crémaillère, ce qui provoque une libération instantanée du ressort.

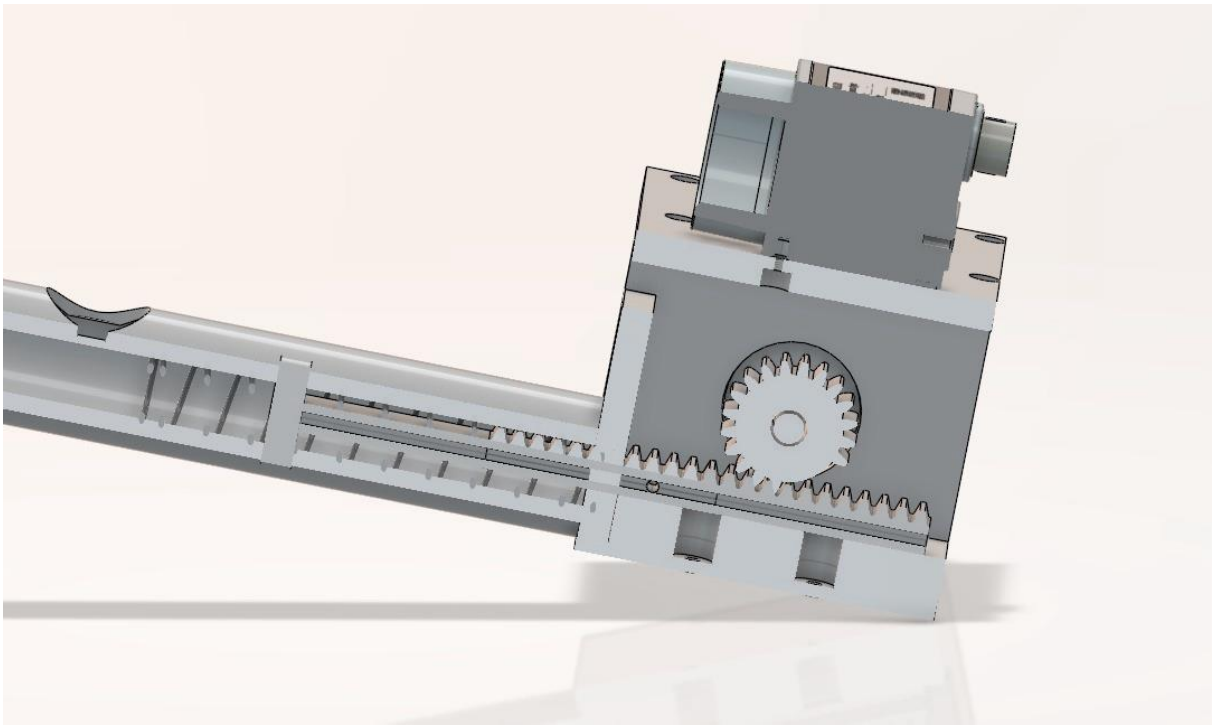


Figure 6: mécanisme du système de propulsion

Afin d'éviter que le ressort et la crémaillère ne soient projetés brusquement lors de la libération du ressort, nous avons apporté deux modifications importantes : d'une part, une vis a été fixée sur la crémaillère pour limiter sa course ; d'autre part, un léger creusage a été réalisé de part et d'autre de la crémaillère, dans le but d'amortir et ralentir le mouvement.
À l'extrémité de la crémaillère, nous avons conçu une forme de disque plat afin améliorer la propulsion de la bille.
Un tube a été conçu pour simuler un canon, intégrant un orifice situé juste après le disque fixé à l'extrémité de la crémaillère, permettant ainsi l'insertion de la bille.
Le moteur est logé dans un boîtier, sur lequel la caméra est fixée.

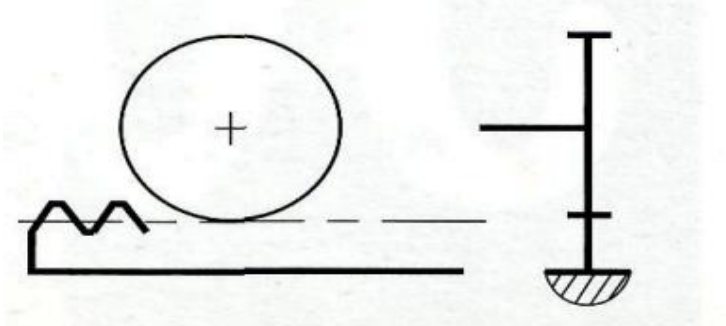


Figure 7: schéma cinématique du système de tire

Programmation Arduino

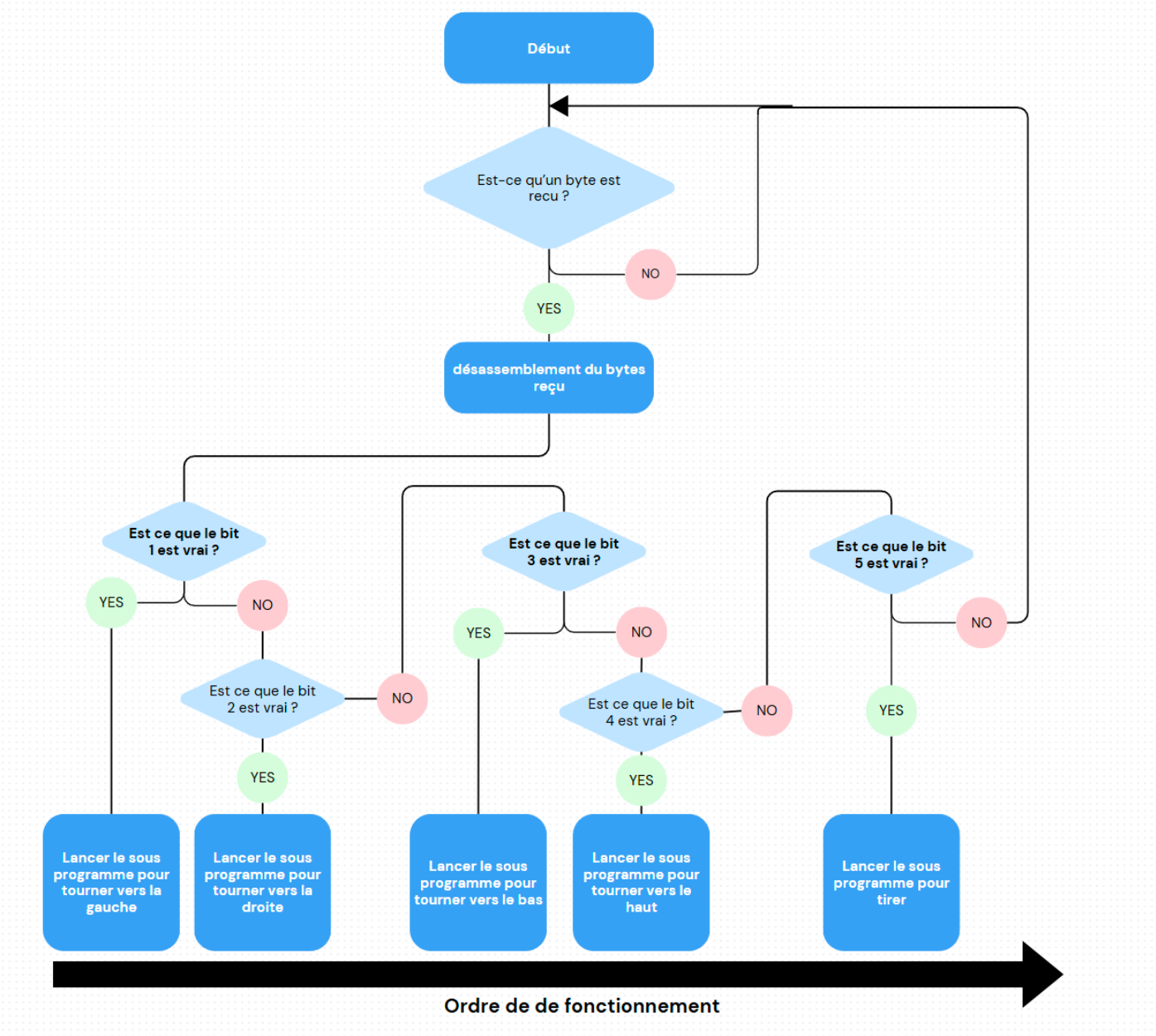


Figure 8: Logigramme du code Arduino

Au niveau de la carte Arduino, nous avons 3 partie principale pour ce programme. Chacune on utilité bien définie et essaie d'etre le plus optimiser pour moins surcharge la carte de développement.

- Partie initialisais : dans cette partie nous allons retrouver la déclaration de la plupart des variables. Nous utilisons beaucoup de fonction faire les divers mouvements, pour cela nous utilisons plusieurs variables globales. Nous utilisons deux types de variable dans notre code :
    - Constante : nous utilisons deux types de constante : *define* et *const*. Les variables déclarer en *define* son les variables qui être uniquement utiliser dans la partie setup de notre programme (nombre de différent pin utiliser). L'avantage de ce type de variable permet de rendre code un peu plus léger et rapide. Les variables déclarer en tant de *const* permet d'avoir plus de sécurité et facilité l'utilisation dans les fonctions. Nous allons y retrouver les variables de vitesse pour les moteurs.
    - Variable quelconque global pour diverses utilités mais nous avons besoin de les modifier en cours de programme.

  Dans cette meme partie nous retrouvons la fonction setup qui est appelé à tout reset de la carte ou remise sous tension.

- Partie de fonction de mouvement : Nous y retrouvons les fonctions qui permet de faire les 4 différent mouvement de la tourelle. La fonction de mouvement pour la droite ou la gauche ont une particularité. Nos deux moteurs n'ont pas le meme déplacement par pas, on fait un réajustement de vite pour combler cette différence.

- Partie communication : cette fonction sera continuellement active dans la fonction loop de notre code, elle permet la communication entre notre programme python et notre carte Arduino. Au départ de notre programme, il lit dans le flux de données entrant un byte pour ensuite le désassemblé en divers bit qui correspondra pour 5 actions possible. Avec notre système de condition, nous avons un ordre de priorité où chaque fonction peut etre exécuter dans un ordre défini qui est le suivant : tourner à droite ; tourner à gauche ; tourner vers le haut ; tourner vers le bas, tirer.

Programmation Python

Le programme Python a pour comme objection de gérer la caméra, traiter ces images pour identifier une cible et y déterminer la distance et son placement par rapport à la caméra. Ensuite il doit gérer un système d'asservissement pour ordonner à la tourelle de se positionner pour puisse y tirer un projectile. Enfin comme dernière tache, il doit crée un byte avec tout ces informations pour l'envoyer par voie USB à la carte de développement Arduino

Code de gestion de camera

Dans cette partie de programmation python, la casi totalité du programme à été fournie par notre professeur mais nous avons juste fait une modification pour que cela soit compatible avec notre projet et programme.
Notre modification impact au niveau des coordonnées x,y et z. Grâce à variable global spécifique qui nous permet de faire circuler des données entre les threads, nous allons tout d'abord vérifier si une cible est détecter. Si ce n'est pas le cas, avoir nous mettons dans une variable Boolean un false qui signifiera qu'il n'y a pas de cible et que si on est en mode automatique, il ne faut pas bouger. Sinon on envoie tous les coordonnées au thread, via des variables globales spécial, qui gère la communication avec la carte Arduino.

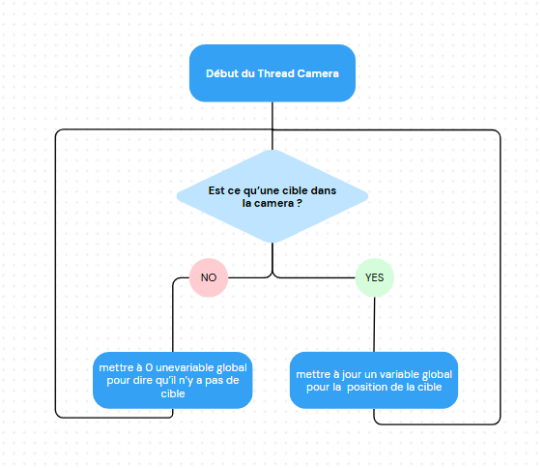


Figure 9: Logigramme de la partie caméra(Python)

Code de gestion du joystick

Dans cette partie, il n'y a pas beaucoup de chose à décrire à par que c'est un programme qui permet de détecter des ordres fait par une manette extérieure pour gérer un système manuel Mais pour éviter de surcharger le programme principal, on le met en temp que second thread.

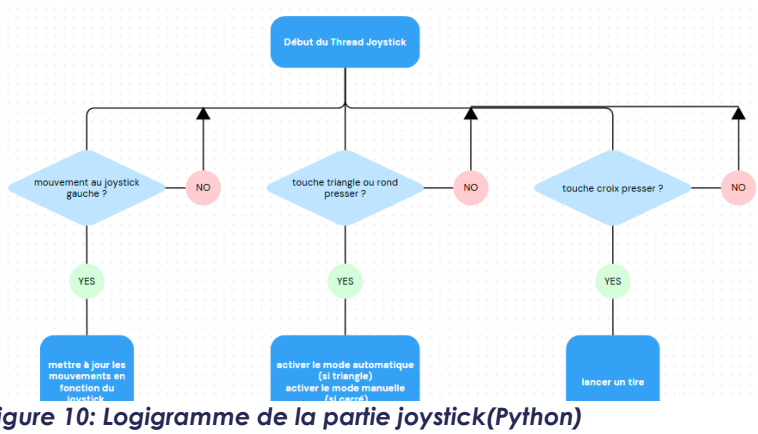


Figure 10: Logigramme de la partie joystick(Python)

Code de communication Arduino

Cette partie de code est la plus important et principal de ce projet Python. Cette partie a comme objectif de récupérer les donnée transmis par le sous-programme de la caméra et du joystick pour pouvoir faire fonction le système d'asservissement pour enfin envoie un byte à la carte Arduino. Tout d'abord, on recherche en quelle mode on est. Le mode automatique fonctionne avec la caméra, le mode manuel par la manette. Par défaut, le mode sélectionner est le mode automatique. En manuelle les commandes de la manette seront converties dans le byte de communication pour contrôler la tourelle.

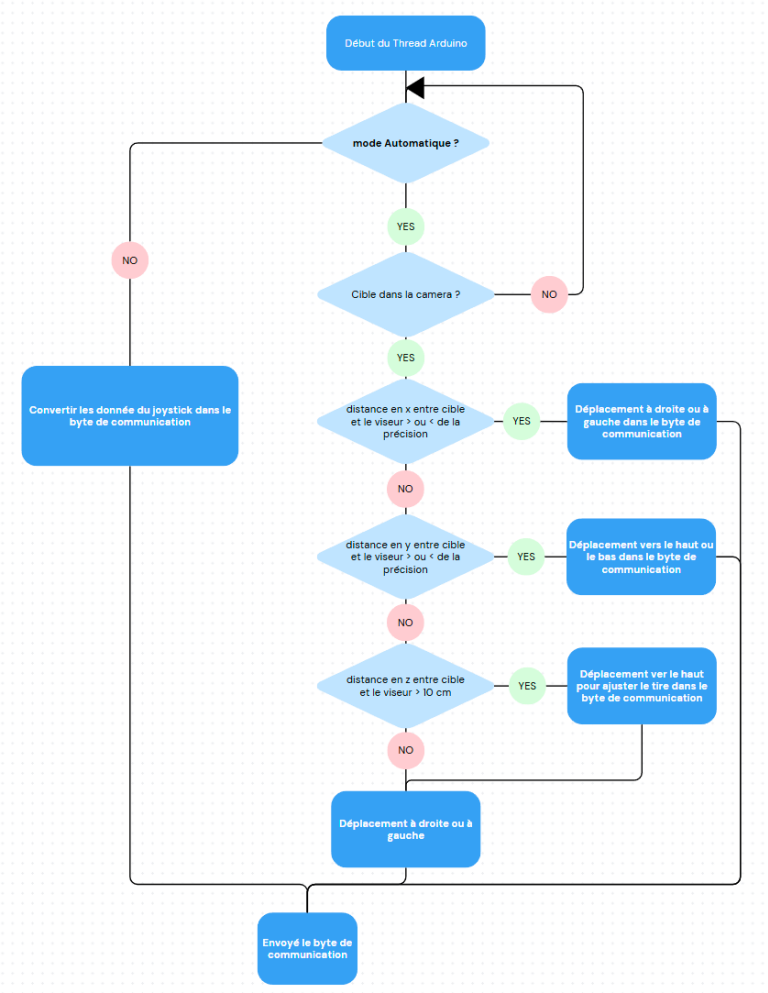


Figure 11: Logigramme de la partie Principal, Comm Arduino (Python)

En automatique, le fonctionnement est méthodique comme pour le code Arduino. Tout d'abord, ont regarde la variable qui permet de dire s'il y a une cible ou non dans la caméra. Pour faire le reste du programme et l'asservissement nous devons avoir une cible dans la caméra. Ensuite on s'occupe des coordonnée X soit on tourne à droite ou à gauche. Ensuite on s'occupe des coordonnées Y. La dernière la partie observe la profondeur de cible. Comme notre puissance feue nous permet de tirer en ligne droite jusqu'à une certaine distance, nous devons réajuster l'angle de tire du canon pour faire un effet de cloche. Un fois bien position, on peut tirer.

Peut import le mode utiliser, à la fin de notre programme, nous envoyons le byte à la carte Arduino via l'USB.

Problème rencontre lors de la conception mécanique

A-Le modèle du système de propulsion :

Au début on était partie pour adapter le système disponible à l'adresse suivante : Spring Ball Launcher - GrabCAD, en fonction des dimensions et caractéristiques de nos balles.

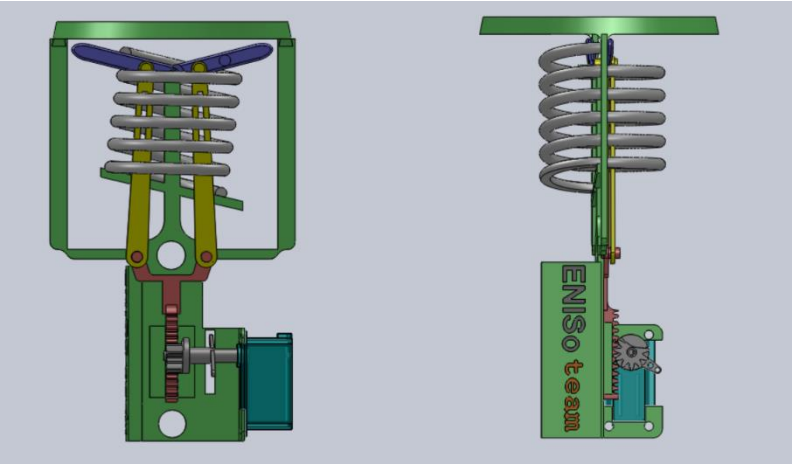


Figure 12: premier système de propulsion

Fonctionnement du système : La partie centrale du mécanisme est le ressort hélicoïdal, qui est comprimé verticalement pour stocker de l'énergie mécanique. Il est relié à une tige de poussée jaune (bielle), qui guide le mouvement du ressort lors de la phase de compression et de détente. Cette tige est fixée à une crémaillère, qui constitue l'élément de conversion du mouvement. En effet, la crémaillère rouge est entraînée par un pignon situé à sa base. Ce pignon est lui-même actionné par un moteur fixé latéralement. Lorsque le moteur tourne, le pignon entraîne la crémaillère vers le bas, comprimant ainsi le ressort. Une fois la compression terminée, un mécanisme de verrouillage (en bleu foncé en haut de la tige) vient bloquer le ressort en position armée. Lors du tir, ce verrou est relâché, permettant au ressort de se détendre brutalement et de propulser la tige jaune vers le haut. Cette dernière transmet alors l'énergie cinétique au projectile situé dans le canon. Tous les composants sont fixés sur une structure, servant de châssis et garantissant la rigidité et l'alignement du système

Les modifications apportées :

Voici la conception 3D de notre système en adaptant le système au-dessus avec le même principe de fonctionnement

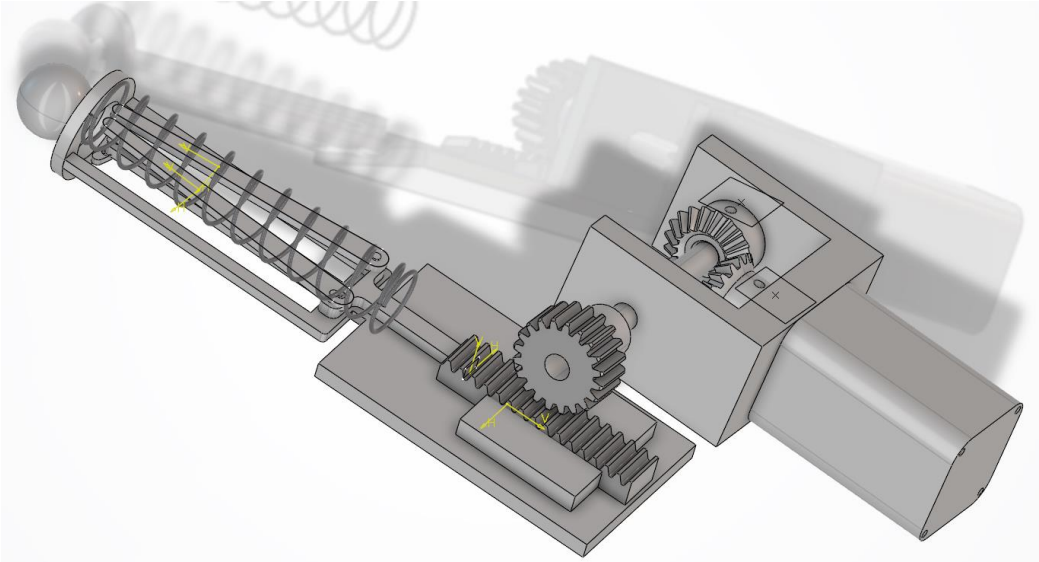
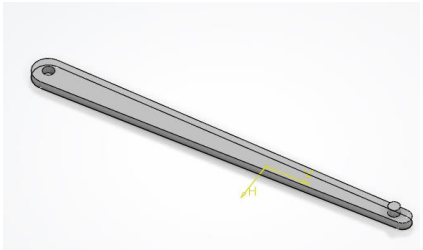
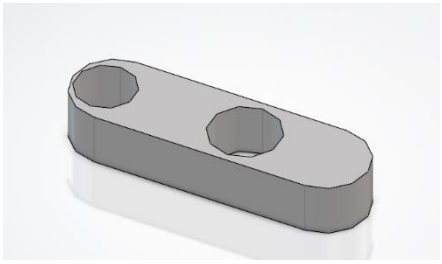
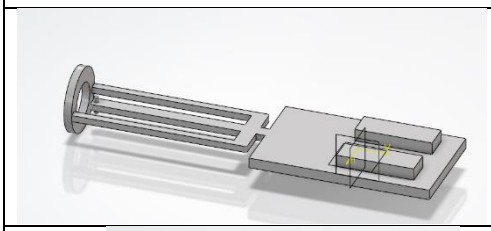
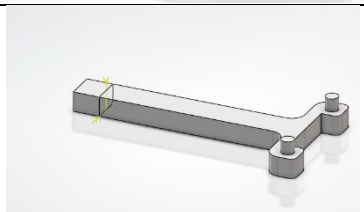
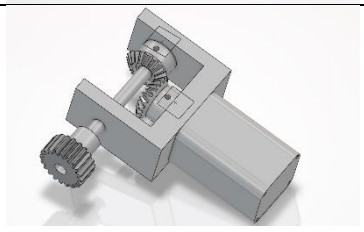


Figure 13: ancien mécanisme du canon

	la bielle permet de tier sur le mécanisme de verrouillage ainsi permettre au ressort de se comprimé en d'autre terme le maintenir en compression
	Mécanisme de verrouillage
	Chassis
	Piston lié a la crémaillère
	Pour le moteur cette modification a pour objectif de changer l'orientation de l'axe de sortie du moteur pour l'adapter aux contraintes mécaniques ou d'encombrement de l'ensemble du système (repartir le poids sur notre tourelle/optimiser l'espace) On utilise deux engrenages coniques montés à 90°. L'arbre moteur est solidaire du premier engrenage conique. Ce dernier transmet le mouvement de rotation à un deuxième engrenage conique

	perpendiculaire, qui lui est monté sur un autre axe orienté à 90° par rapport au moteur. Ce nouvel arbre de sortie est couplé à un pignon cylindrique pour entraîner une crémaillère

Limitation de ce systéme :

Ce système n'a malheureusement pas pu être retenu en raison de plusieurs contraintes liées au dimensionnement. En effet, comme on peut le constater, le ressort ne peut pas être entièrement introduit dans l'espace prévu. Même en réduisant certaines dimensions pour tenter de l'adapter, une nouvelle contrainte est apparue : le système perdait en rigidité mécanique, ce qui compromettait sa fiabilité. De plus, dans cette configuration, le mécanisme présentait des zones peu adaptées à l'impression 3D, que ce soit en termes de stabilité ou de tolérance d'assemblage. Ces limitations nous ont donc conduit à abandonner cette version au profit d'une alternative plus compacte et mieux adaptée à nos contraintes de fabrication.

B- Contraintes d'impression 3D :

Lors de l'impression 3D, les trous prévus dans les pièces n'ont pas été correctement formés. La précision de l'imprimante n'était pas suffisante pour produire des alésages nets et aux bonnes dimensions. Donc il a fallu reprendre manuellement les trous à la perceuse pour permettre l'assemblage des vis.

- Certaines pièces fines (comme les dents d'engrenage ou les supports d'axe) étaient difficiles à imprimer sans déformation.
- Des reprises post-impression (ponçage,) ont été nécessaires.
    - le lien entre l'axe moteur et le pignon n'était pas toujours bien ajusté donc il a fallu modifier le design pour améliorer le serrage c'est-à-dire modifier les diamètres des pignons et le placement des trous pour mettre des vis a 90°.
    - Les frottements donc il a fallu lubrifier à plusieurs reprises

Problèmes de couple Moteur pas à pas qui entraîne le pignon, lequel fait coulisser une crémaillère. Cette crémaillère est soumise à une force de rappel par un ressort (ou ressort de rappel) qui cherche à la ramener à sa position initiale, Lorsque le moteur démarre pour faire avancer la crémaillère, il doit appliquer un couple suffisant sur le pignon pour vaincre la force exercée par le ressort. Cependant dans notre cas on n'a constaté que le couple de maintien de notre moteur n'est pas suffisant pour résister à cette force

Démonstration théorique :

- Couple de moteur vaut 54g.cm²=0.053Nm

On estime la déformation de notre ressort de $\Delta \boldsymbol{x} = \boldsymbol{23.4mm}$

- La force de rappel $\boldsymbol{F_r} = \boldsymbol{k\Delta x}$=0.61× $\boldsymbol{23.4}$=14.3N

T= $\boldsymbol{r} \times \boldsymbol{F_r}$=9.63× $\boldsymbol{19.6} \approx \boldsymbol{137.6}$ Nmm ≈ $\boldsymbol{0.138Nm}$

- r : le rayon du pignon
- T : couple sur le pignon

Alors on a 0.053<0.138 ce qui permet de renforcer notre constat cité ci-dessus

Calcule théorique :

- **Raideur du ressort** : k=0,61 N/mm=610N/m
- **Compression :** x=23,4 mm=0,0234
- **Masse de la bille** : m=2 g=0,002kg
- **Diamètre de la bille** d=16mm

L'énergie emmagasinée par le ressort :

$$\boldsymbol{E_r = \frac{1}{2} k x^2 = 0,5 \times 610 \times 0,00054756 = 0,167j}$$

Energie cinétique :

$$\boldsymbol{E = \frac{1}{2} m v^2 = E_r}$$

La vitesse estimée en ignorant les fortement $\boldsymbol{v = \sqrt{\frac{2E_r}{m}} = 12,92 m/s}$

Etat d'avancement

À ce stade du projet, on a bien pu réaliser les fonctionnalités mentionnées dans le cahier des charges. Tout d'abord, nous avons pu adapter une plateforme Pan-Tilt et la rendre fonctionnelle, capable de s'orienter sur deux axes grâce à l'utilisation de moteurs pas-à-pas et d'un système de transmission par engrenages. Parallèlement, nous avons développé un système de propulsion basé sur un mécanisme pignon-crémaillère, permettant de comprimer un ressort en vue de lancer un projectile. L'ensemble de la mécanique a été intégré de manière cohérente avec la plateforme mobile.

Sur le plan logiciel, la programmation Arduino a permis d'assurer le contrôle des moteurs pour les deux axes de rotation ainsi que le déclenchement du tir. En complément, nous avons développé un script en Python capable de traiter l'image captée par la caméra, d'identifier la cible, puis de transmettre les consignes à la carte Arduino. La synchronisation entre la détection de la cible, l'orientation de la plateforme et le tir a été assez bien faite

Cependant, certains aspects n'ont pas pu être pleinement aboutis. En particulier, la portée du tir reste limitée. Cela est principalement dû à un manque de puissance du moteur utilisé pour comprimer le ressort, ce qui empêche d'atteindre une force suffisante pour propulser la bille à longue distance.

Si nous avions disposé de plus de temps, plusieurs pistes d'amélioration auraient pu être explorées pour optimiser notre lanceur balistique. Sur le plan mécanique, nous aurions souhaité améliorer le système de transmission du tir en intégrant un rapport de réduction de type 1:2, afin d'augmenter le couple transmis à la crémaillère et ainsi permettre une compression plus efficace du ressort. Cela aurait également permis de résoudre partiellement le problème de distance de tir. Enfin, pour améliorer l'expérience utilisateur, la mise en place d'une application ou interface graphique sur ordinateur aurait été envisagée. Celle-ci aurait permis de visualiser en temps réel la détection de la cible, de suivre l'orientation de la plateforme, et d'envoyer des commandes de manière plus intuitive, renforçant ainsi l'interaction homme-machine.

Table des Illustrations

Annexe

Premier partie code Python

```python
import threading
import queue
import time

import pygame

import serial

import numpy as np
import cv2
from pypylon import pylon

# Création des variable de position binaire
bit_right = 0
bit_left = 1
bit_high = 2
bit_low = 3

bit_fire = 4

# Variable de position
target_x = 0
target_y = 0
target_z = 0

# control joystick
# Initialiser pygame et les modules joystick
pygame.init()
pygame.joystick.init()

# Vérifier la présence d'une manette
if pygame.joystick.get_count() == 0:
    print("Aucune manette détectée.")
    exit()

# Se connecter à la première manette détectée
joystick = pygame.joystick.Joystick(0)
joystick.init()
print(f"Manette connectée : {joystick.get_name()}")

# Variable statu
shared_data = {"position_x": 0, "position_y": 0, "position_z": 0, "object":True}

lock = threading.Lock()

shared_data_manette = {"droite_L": False, "gauche_L": False, "haut_L": False, "bas_L":False,
                       "mode_auto":True, "fire":False}

lock_manette = threading.Lock()

ser = serial.Serial(port="COM7", baudrate=9600, timeout=1)

# === Paramètres ArUco ===
aruco_dict_type = cv2.aruco.DICT_6X6_50
marker_length = 0.045  # en mètres (5 cm) coefficient à régler selon cible
```

Seconde partie code Python

```python
# === Paramètres de la caméra (à calibrer selon caméra) ===
camera_matrix = np.array( object: [[1280, 0, 640],
                          [0, 1024, 512],
                          [0, 0, 1]], dtype=np.float32)
dist_coeffs = np.zeros((5, 1))  # si pas de distorsion connue

# Initialisation ArUco
aruco_dict = cv2.aruco.getPredefinedDictionary(aruco_dict_type)
aruco_params = cv2.aruco.DetectorParameters()

# === Initialisation caméra Pylon ===
tlFactory = pylon.TlFactory.GetInstance()
devices = tlFactory.EnumerateDevices()
print(str(len(devices)) + " cameras connected")
if len(devices) == 0:
    raise pylon.RUNTIME_EXCEPTION("No camera present.")
cameras = pylon.InstantCameraArray(len(devices))

for i, cam in enumerate(cameras):
    cam.Attach(tlFactory.CreateDevice(devices[i]))
    print("Using device ", cam.GetDeviceInfo().GetModelName())

camera0 = cameras[0]

converter = pylon.ImageFormatConverter()
converter.OutputPixelFormat = pylon.PixelType_BGR8packed
converter.OutputBitAlignment = pylon.OutputBitAlignment_MsbAligned

cameras.StartGrabbing(pylon.GrabStrategy_LatestImageOnly)

print("Start Code")
1 usage
def loop_comm_arduino():
    distance_x = 0
    distance_y = 0
    target_acting = 0.005
    look = False
    while True:
        if ser.in_waiting:
            byte_data = ser.read(1)  # lire 1 byte brut
            if byte_data:
                value = int.from_bytes(byte_data, byteorder='little')
                print(f"Reçu : {value}")

        data = 0b00000000  # valeur initiale = 0

        # data |= (1 << bit_to_set)
        # data &= ~(1 << bit_to_clear)

        if shared_data_manette["mode_auto"]:
            if shared_data["object"]:
                distance_x = shared_data["position_x"] - target_x
                print(f"x = {shared_data["position_x"]}, distance_y = {distance_x}")
                distance_y = shared_data["position_y"] - target_y
                print(f"y = {shared_data["position_y"]}, distance_y = {distance_y}")
```

Troisième partie code python

```python
            if not look:
                if distance_x < 0:
                    distance_load_x = -distance_x
                else:
                    distance_load_x=distance_x
                if distance_load_x > target_acting:
                    if distance_x < target_acting:
                        data |= (1 << bit_right)
                        data &= ~(1 << bit_left)
                        print("left")
                    elif distance_x > target_acting:
                        data |= (1 << bit_left)
                        data &= ~(1 << bit_right)
                        print("right")

                if distance_y < 0:
                    distance_load_y = -distance_y
                else:
                    distance_load_y=distance_y
                if distance_load_y > target_acting:
                    if distance_y < target_acting:
                        data |= (1 << bit_high)
                        data &= ~(1 << bit_low)
                        print("high")
                    elif distance_y > target_acting:
                        data |= (1 << bit_low)
                        data &= ~(1 << bit_high)
                        print("low")
            else:
                data = 0b00000000  # valeur initiale = 0
        else:
            if shared_data_manette["gauche_L"]:
                data |= (1 << bit_right)
                data &= ~(1 << bit_left)
                print("left")
            elif shared_data_manette["droite_L"]:
                data |= (1 << bit_left)
                data &= ~(1 << bit_right)
                print("right")
            elif shared_data_manette["haut_L"]:
                data |= (1 << bit_high)
                data &= ~(1 << bit_low)
                print("high")
            elif shared_data_manette["bas_L"]:
                data |= (1 << bit_low)
                data &= ~(1 << bit_high)
                print("low")
            elif shared_data_manette["fire"]:
                data |= (1 << bit_fire)
            else:
                data = 0b00000000

        if distance_x < 0:
            distance_x = -distance_x
        if distance_y < 0:
            distance_y = -distance_y

```

Quatrième partie code python

```python
        if (distance_x < target_acting and distance_y < target_acting) and shared_data["object"]:
            look = True
            data = 0b00000000  # valeur initiale = 0
        else:
            look = False


        print(shared_data_manette["mode_auto"])

        print(f"Envoi : {data}")

        ser.write(data.to_bytes( length: 1, byteorder='little'))
        if shared_data_manette["mode_auto"]:
            time.sleep(0.03)
        else:
            time.sleep(0.01)
    ser.close()
1 usage
def loop_target_tracker():
    # === Boucle d'acquisition ===
    while True:
        grab_result0 = camera0.RetrieveResult(5000, pylon.TimeoutHandling_ThrowException)

        if grab_result0.GrabSucceeded():
            image0 = converter.Convert(grab_result0).GetArray()
            gray = cv2.cvtColor(image0, cv2.COLOR_BGR2GRAY)

            # Détection des marqueurs ArUco
            corners, ids, rejected = cv2.aruco.detectMarkers(gray, aruco_dict, parameters=aruco_params)

            if ids is not None:
                # Dessiner les contours
                cv2.aruco.drawDetectedMarkers(image0, corners, ids)

                # Estimer la pose
                rvecs, tvecs, _ = cv2.aruco.estimatePoseSingleMarkers(corners, marker_length, camera_matrix,
                                                                      dist_coeffs)

                for rvec, tvec in zip(rvecs, tvecs):
                    # Dessin des axes sur l'image
                    cv2.drawFrameAxes(image0, camera_matrix, dist_coeffs, rvec, tvec, length: 0.03)  # axe de 3 cm

                    # Position du marqueur dans le repère caméra
                    x, y, z = tvec[0]  # tvec est de forme (1, 3)

                    # Correction : inversion de Y pour qu'il aille vers le haut de l'image
                    y_corrected = -y


                    with lock:
                        shared_data["object"] = True
                        shared_data["position_x"] = x
                        shared_data["position_y"] = y
                        shared_data["position_z"] = z

                    # print(f"Position cible : x={x:.3f} m, y={y_corrected:.3f} m, z={z:.3f} m")
```

Cinquième partie code python

```
            else:
                with lock:
                    shared_data["object"] = False
                    shared_data["position_x"] = 0
                    shared_data["position_y"] = 0
                    shared_data["position_z"] = 0

            cv2.imshow( winname: "Camera ArUco", image0)

        k = cv2.waitKey(1)
        if k % 256 == 27:
            print("Fin d'acquisition...")
            break

    cv2.destroyAllWindows()
1 usage
def read_joystick():
    try:
        while True:
            pygame.event.pump()  # Mise à jour des événements

            if joystick.get_button(3):   # carre
                shared_data_manette["mode_auto"] = True
                print("mode_auto : active")
            if joystick.get_button(2):  # rond
                shared_data_manette["mode_auto"] = False
                print("mode_auto : desactive")

            left_x = joystick.get_axis(0)

            if left_x >0.5:
                shared_data_manette["droite_L"] = True
                shared_data_manette["gauche_L"] = False
            elif left_x <-0.5:
                shared_data_manette["droite_L"] = False
                shared_data_manette["gauche_L"] = True
            else:
                shared_data_manette["droite_L"] = False
                shared_data_manette["gauche_L"] = False

            left_y = joystick.get_axis(1)

            if left_y >0.5:
                shared_data_manette["haut_L"] = False
                shared_data_manette["bas_L"] = True
            elif left_y <-0.5:
                shared_data_manette["haut_L"] = True
                shared_data_manette["bas_L"] = False
            else:
                shared_data_manette["haut_L"] = False
                shared_data_manette["bas_L"] = False

```

Sixième partie code python

```
274
275             if joystick.get_button(0):
276                 shared_data_manette["fire"] = True
277                 print("fire is active  !!!!!!")
278             else:
279                 shared_data_manette["fire"] = False
280
281             time.sleep(0.01)
282
283     except KeyboardInterrupt:
284         print("Arrêt par utilisateur.")
285
286     finally:
287         pygame.quit()
288
289 # Création des threads
290 thread_a = threading.Thread(target=loop_target_tracker)
291 thread_b = threading.Thread(target=loop_comm_arduino)
292 thread_c = threading.Thread(target=read_joystick)
293
294 # Démarrage
295 thread_a.start()
296 thread_b.start()
297 thread_c.start()
298
```

Premier partie code arduino

```
#define STEP_PIN_MOTOR_AXE_2 9  // STEP moteur Axe 1
#define DIR_PIN_MOTOR_AXE_2 8   // DIR moteur Axe 1
#define EN_PIN_MOTOR_AXE_2 10   // DIR moteur Axe 1

#define STEP_PIN_MOTOR_AXE_1 6  // STEP moteur Axe 2
#define DIR_PIN_MOTOR_AXE_1 5   // DIR moteur Axe 2
#define EN_PIN_MOTOR_AXE_1 7   // DIR moteur Axe 2

#define STEP_PIN_MOTOR_FIRE 12  // STEP moteur canon
#define DIR_PIN_MOTOR_FIRE 11   // DIR moteur canon
#define EN_PIN_MOTOR_FIRE 13   // DIR moteur canon

const int speed_delay_MOTOR_FIRE = 2000;

unsigned long lastStepTime1 = 0;
unsigned long lastStepTime2 = 0;

unsigned int speed_delay_MOTOR_AXE_2 = 2000; // vitesse plus lente
unsigned int speed_delay_MOTOR_AXE_1 = 500;

bool stepState1 = false;
bool stepState2 = false;

const int ondulation = 100;

byte data_python = 0;

void setup() {
    pinMode(STEP_PIN_MOTOR_AXE_1, OUTPUT);
    pinMode(DIR_PIN_MOTOR_AXE_1, OUTPUT);
    pinMode(EN_PIN_MOTOR_AXE_1, OUTPUT);

    pinMode(STEP_PIN_MOTOR_AXE_2, OUTPUT);
    pinMode(DIR_PIN_MOTOR_AXE_2, OUTPUT);
    pinMode(EN_PIN_MOTOR_AXE_2, OUTPUT);

    pinMode(STEP_PIN_MOTOR_FIRE, OUTPUT);
    pinMode(DIR_PIN_MOTOR_FIRE, OUTPUT);
    pinMode(EN_PIN_MOTOR_FIRE, OUTPUT);

    pinMode(4, OUTPUT);
    Serial.begin(9600);

    digitalWrite(EN_PIN_MOTOR_AXE_1, LOW);  // Activer le driver
    digitalWrite(EN_PIN_MOTOR_AXE_2, LOW);  // Activer le driver
    digitalWrite(EN_PIN_MOTOR_FIRE, LOW);  // Activer le driver

    digitalWrite(4, HIGH);
    delay(3000);
    digitalWrite(4, LOW);
}


void loop() {
  comm();
}

void comm(){
  bool received = false;

  if (Serial.available()) {
    data_python = Serial.read();
    received = true;
  }
```

Seconde partie code arduino

```
66
67    if (received) {
68      digitalWrite(4, HIGH);
69      bool bit_left = (data_python & (1 << 0));
70      bool bit_right  = (data_python & (1 << 1));
71      bool bit_high  = (data_python & (1 << 2));
72      bool bit_low   = (data_python & (1 << 3));
73      bool bit_fire  = (data_python & (1 << 4));
74
75      if (bit_right)      turn_right();
76      else if (bit_left)  turn_left();
77      else if (bit_high)  turn_high();
78      else if (bit_low)   turn_low();
79      else if (bit_fire)  fire_canon();
80
81      Serial.write(data_python);  // Confirmer la réception
82    }
83    digitalWrite(4, LOW);
84  }
85
86  void turn_right(){
87    digitalWrite(DIR_PIN_MOTOR_AXE_1, HIGH);
88    digitalWrite(DIR_PIN_MOTOR_AXE_2, HIGH);
89    for (int i = 0; i < ondulation; i++) {  // 200 pas = 1 tour si moteur 1.8°/pas
90      rotation();
91    }
92  }
93
94  void turn_left(){
95    digitalWrite(DIR_PIN_MOTOR_AXE_1, LOW);
96    digitalWrite(DIR_PIN_MOTOR_AXE_2, LOW);
97    for (int i = 0; i < ondulation; i++) {  // 200 pas = 1 tour si moteur 1.8°/pas
98      rotation();
99    }
100 }
101
102 void rotation(){
103   unsigned long now = micros();   // plus précis que millis()
104
105   // Moteur 1
106   if (now - lastStepTime1 >= speed_delay_MOTOR_AXE_1)  {
107     lastStepTime1 = now;
108     stepState1 = !stepState1;
109     digitalWrite(STEP_PIN_MOTOR_AXE_1, stepState1);
110   }
111   // Moteur 2
112   if (now - lastStepTime2 >= speed_delay_MOTOR_AXE_2) {
113     lastStepTime2 = now;
114     stepState2 = !stepState2;
115     digitalWrite(STEP_PIN_MOTOR_AXE_2, stepState2);
116   }
117 }
118
119 void turn_low(){
120   digitalWrite(DIR_PIN_MOTOR_AXE_1, LOW);
121
122   digitalWrite(STEP_PIN_MOTOR_AXE_1, HIGH);
123   delayMicroseconds(speed_delay_MOTOR_AXE_1+500);  // Temps entre les pas
124   digitalWrite(STEP_PIN_MOTOR_AXE_1, LOW);
125   delayMicroseconds(speed_delay_MOTOR_AXE_1+500);
126
127 }
128
129 void turn_high(){
130   digitalWrite(DIR_PIN_MOTOR_AXE_1, HIGH);
131
```

Troisième partie code arduino

```
129  void turn_high(){
130    digitalWrite(DIR_PIN_MOTOR_AXE_1, HIGH);
131
132    digitalWrite(STEP_PIN_MOTOR_AXE_1, HIGH);
133    delayMicroseconds(speed_delay_MOTOR_AXE_1-500);  // Temps entre les pas
134    digitalWrite(STEP_PIN_MOTOR_AXE_1, LOW);
135    delayMicroseconds(speed_delay_MOTOR_AXE_1-500);
136
137  }
138
139  void fire_canon(){
140    digitalWrite(DIR_PIN_MOTOR_FIRE, HIGH);
141    for (int i = 0; i < 1600; i++) {  // 200 pas = 1 tour si moteur 1.8°/pas
142      digitalWrite(STEP_PIN_MOTOR_FIRE, HIGH);
143      delayMicroseconds(2000);  // Temps entre les pas
144      digitalWrite(STEP_PIN_MOTOR_FIRE, LOW);
145      delayMicroseconds(2000);
146    }
147  }
148
```