UNDERGRADUATE RESEARCH EXPERIENCE (UREx) AY24/25

Technical Project Report

Developing a Dart System for the RoboMaster University Championship

Supervised by:
Dr. Idris Lim Li Hong
Mr. Chew Wanlong Nicholas

No.	Name	Matriculation No.
1	Lim Ji Yong	
2	Lin Hong Yi	
3	Teh Wei Sheng	
4	Cai Jiali	

Abstract

This research centres on the development of a high-precision dart-launching system for the NUS Calibur Robotics team's participation in the RoboMaster University Championship (RMUC). The project aims to design a system capable of accurately launching darts to hit designated targets within the competition's rigorous specifications. Key objectives include creating a flywheel-based launcher, designing both passive and active dart prototypes, integrating robust electrical systems for control and automation, and implementing computer vision for real-time target tracking and trajectory adjustment.

The flywheel mechanism was selected for its reliability and consistency, paired with a revolver-style reloading system for efficient and compact dart handling. Passive darts emphasise aerodynamic stability and modularity, while active darts incorporate servo-controlled fins for precise directional adjustments. Innovations in design and testing include vibration analysis to enhance stability, modular components for iterative prototyping, and an efficient electrical system using vibration sensors and stepper motors.

Extensive testing ensured the system met RMUC's requirements for accuracy, range, and operational consistency. The computer vision system, implemented on a Raspberry Pi Zero 2W with an IMX219 camera, successfully tracks targets using green light detection algorithms. This integration enables real-time guidance adjustments to improve accuracy and reliability under competition conditions.

To summarise, this project contributes to the advancement of robotics by demonstrating the integration of mechanical, electrical, and computer vision systems. It provides valuable insights into achieving precision in dynamic environments, offering a scalable approach to solving similar engineering challenges. The findings underscore the potential of modular and data-driven design for optimising robotic performance. Future improvements could include stronger materials for dart components, advanced aerodynamic modelling, and refined vision algorithms for dynamic target tracking.

Preface and Acknowledgments

We extend our heartfelt gratitude to Prof. Idris, Nicholas, and Graham for their invaluable guidance to the UREx Robomaster Dart team. Special thanks to Veejay, the project director of the Robomaster Dart team, for his contributions in managing resources and passing down knowledge from the previous team. We are also grateful to the staff from iDP Hub and the Central Workshop for their expert advice and logistical support. Finally, we thank the Provost's Office for providing the opportunity to bring this research project to life. Their support has been instrumental in our success, and we deeply appreciate each contribution.

Table of Contents

Chapter 1 | Introduction

1.1 Project Scope

This research project focuses on developing a reliable dart-launching system for NUS Calibur Robotics' entry in the RoboMaster University Championship (RMUC). The scope is to research and develop:

- **Dart Launcher**: Responsible for loading and propelling the darts over a minimum distance of 5 metres.
- **Dart**: Equipped with its own vision system to navigate and strike a dart detection module.
- **Electrical System**: Integrated features to enable the dart launcher to operate smoothly within specifications and according to the designed mechanism.
- **Computer Vision**: Implement functionality for the dart to track the dart detection module and control the servo motors accordingly.

Key research and engineering areas include mechanical design for the launcher, dart design (both passive and active variants), electrical system integration, and machine vision for trajectory monitoring. The development process encompasses concept generation, feasibility assessment, prototyping, and iterative refinement, all while adhering to RMUC technical specifications. Extensive testing and analysis are conducted to optimise dart launch accuracy and consistency.

Specifically, The URExx group members will focus on various research areas and project scopes, as shown in the table below:

Lim Ji Yong	Electrical System and Vibration Testing
Lin Hong Yi	Dart Designs
Teh Wei Sheng	Dart Launcher Designs
Cai Jiali	Computer Vision

Table 1.1a: Research Areas and Project Scope

Overall, the project entails building a dart-launching system, designing passive and active dart prototypes, and implementing a machine vision system for real-time trajectory control and enhanced accuracy.

1.2 Project Objectives

1. **Dart Launcher**
    a. Design and develop a dart launcher capable of firing four darts continuously within a 30-second timeframe.
    b. Implement mechanisms for automated dart reloading to streamline the launch process and improve efficiency.
    c. Ensure the dart launcher is compatible with both passive and active darts through thoughtful design and engineering considerations.
    d. Achieve a launch range of 15–30 metres with precision sufficient to hit a 1 m² target area.

2. **Dart**
    a. Develop a lightweight and aerodynamically stable dart using optimal materials and structural design.
    b. Engineer the dart to be impact-resistant, capable of withstanding repeated collisions without damage.

3. **Electrical System**
    a. Integrate a sensor and detect vibrations detected during launching - using the previous launcher prototype and the new UREx prototype - to compare values as a measure of structural integrity and stability during launch.
    b. Integrate an electrical system to control the elements of the operating mechanism for the launcher that the team will be developing. This includes actuation and feedback controls.

4. **Computer Vision**
    a. Integrate a vision system into the dart for precise location tracking and navigation toward the dart detection module.
    b. Design a vision system that provides real-time feedback to the team regarding the dart's landing location.

Chapter 2 | Design Specifications

2.1 RoboMaster University Championship (RMUC) Requirements

The dart projectiles are launched by a dart launcher designed to achieve a range of over 15 metres and accurately target one of two towers in the RoboMaster University Championship (RMUC) arena map. The requirements are as follows:

Dart Launcher:

- Pitch angle range: 25°–45°
- Maximum power supply capacity: 265 Wh
- Maximum supply voltage: 30 V
- Maximum weight: 25 kg
- Maximum expansion dimensions: 1000 mm x 600 mm x 1000 mm (L x W x H)
- Shall not use compressed air for propulsion

Dart:

- Maximum power supply: 4 Wh
- Maximum voltage: 8.4 V
- Maximum weight: 0.35 kg
- Maximum expansion dimensions: 250 mm x 250 mm x 150 mm (L x W x H)
- Thrust-to-weight ratio must remain less than one

Figure 2.1a: Annotated Arena Map (Top View)

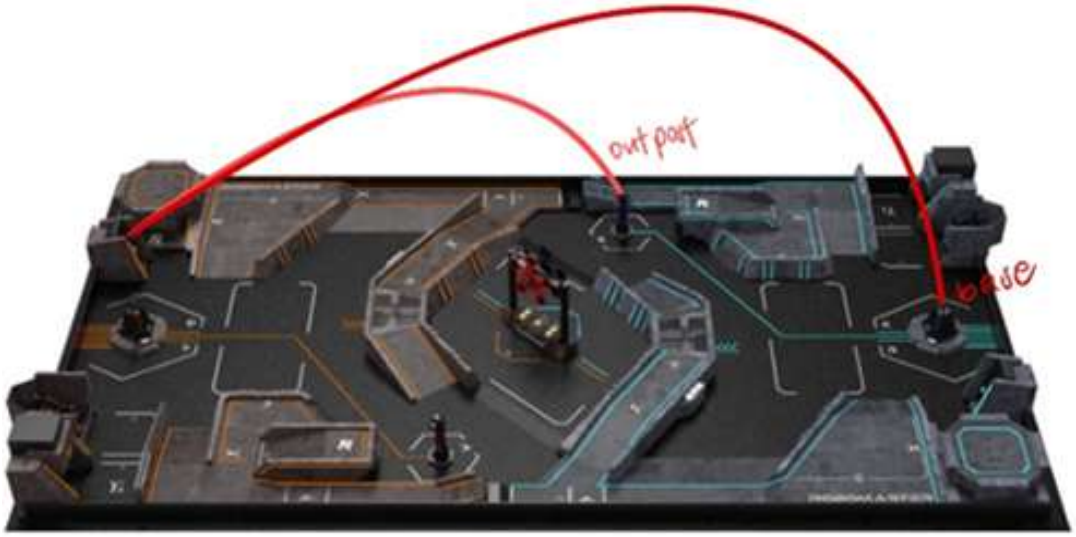


Figure 2.1b: Arena Map (Side View) with Trajectory Annotations

Figure 2.1c: Arena map with dimensions

Chapter 3 | Literature Review

3.1 Competitive Analysis (RMUC)

An in-depth analysis was conducted on the top-performing teams in past RMUC competitions, focusing on their successful dart systems. The findings were studied to inspire concept generation for this project. This section presents the analysis and a comparative assessment of different mechanisms.

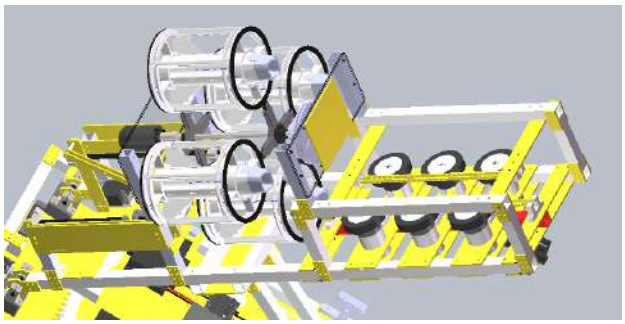
<table>
<tr><th colspan="2">Dart Launcher</th></tr>
<tr><td>Beijing University of Science and Technology</td><td>


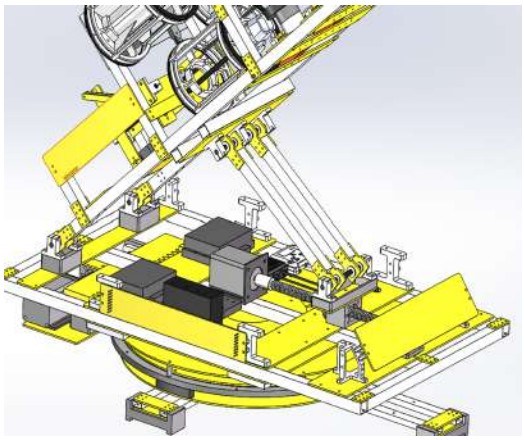
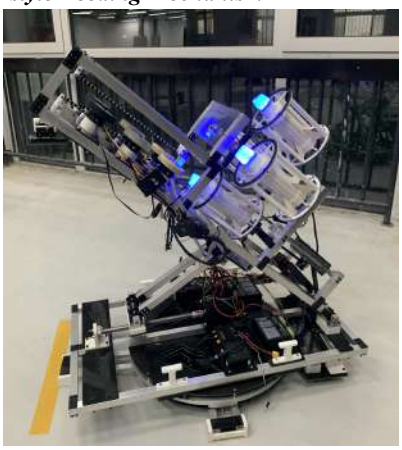
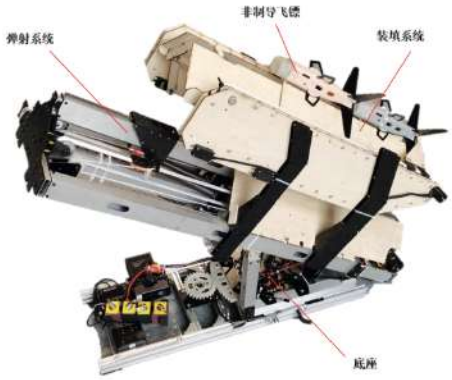
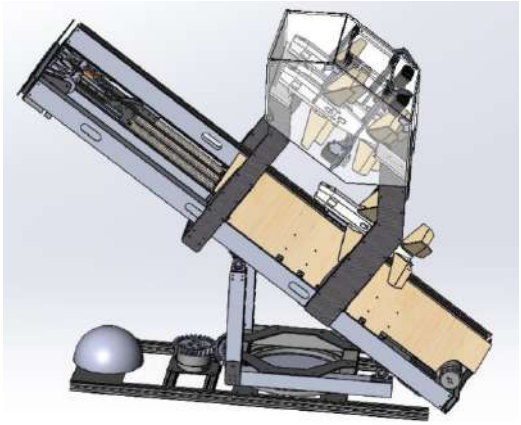
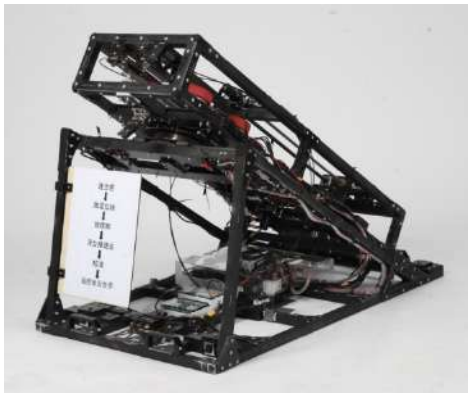
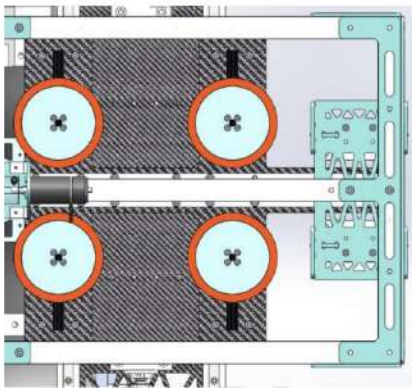
Figure 3.1a: 6-Flywheels with Revolver-style Feeding Mechanism



Figure 3.1b: Pitch & Yaw System Figure 3.1c: Dart System Overview

(Beijing University of Science and Technology, 2021)

The design incorporates a six-flywheel launching system integrated with a revolver-style feeding mechanism and a lever system to accurately position the dart within the launch zone. The revolver rotates 90 degrees for each loading cycle, facilitating the transfer of the dart into the flywheel region for launching. This assembly is affixed to an adjustable pitch and yaw base, employing an unsupported ball screw system for precise pitch adjustments and a "Lazy Susan" configuration for effective yaw control.
</td></tr>
</table>

Nanjing University of Aeronautics and Astronautics	  ***Figures 3.1d, 3.1e: Tensioning Mechanism Launcher (Nanjing University of Aeronautics and Astronautics, 2022)*** The dart system integrates a tensioning mechanism driven by dual motors for slide retraction, a trigger mechanism operated by servo motors, and an anti-torque design to ensure stability during firing. The impact damping is achieved through MGN9 sliders paired with TPU components and nylon ropes, while the feeding mechanism employs a vertical loading system with a parallelogram design for dart transfer, utilising fibreglass and laminated wood for optimal structural integrity, along with a gravity-driven stop to secure darts during movement.
South China University of Technology	  ***Figures 3.1f, 3.1g: Two-stage Acceleration Flywheels Launcher (South China University of Technology, 2023)*** The dart launcher employs a two-stage acceleration mechanism to enhance stability, utilising a larger polyurethane flywheel for increased rotational inertia. Precision linear guides minimise resistance during the launch process, while magnetic positioning ensures accurate alignment for loading. A direct-push mechanism allows for the rapid firing of two darts within eight seconds. The overall structure features an aluminium frame with a yaw adjustment system that enables a ±9° range of motion and a rectangular base that enhances stability through magnetic mounts securing the launcher to the firing platform.

Computer Vision		
Nanjing University of Aeronautics and Astronautics	This dart employs an OpenMV camera positioned at the front, which tracks the location of the detection module by leveraging colour modules in the OpenMV library. The distance between the dart and the target is calculated, and this data is used to guide the continuous control of servo motors that adjust the dart's wings accordingly. The 2022 design had a detection range of only three metres. However, with the same OpenMV camera, an optimised algorithm has extended this range to 16-20 metres in 2024 (RoboMaster赛务君, 2024).	 ***Figure 3.3h: Vision system based on color (Nanjing University of Aeronautics and Astronautics, 2022)***
Northwestern Polytechnical University	This vision system utilises two cameras: one mounted on the launcher and one positioned at the front of the dart to track the detection module's position. Additionally, a gyroscope is used to determine the dart's orientation. Based on the calculated angle and distance between the dart and the detection module, the servo motors adjust the wings dynamically, enabling precise control.	 ***Figure 3.3i: Vision system with gyroscope (Northwestern Polytechnical University, 2024)***

Table 3.1a: Analysis of Top-Performing Teams in RMUC

3.2 Relevant Research on Dart Launcher

3.2.1 Launching Mechanism

Launching mechanism is a critical component in designing the dart launcher system that requires precise and consistent propulsion. Three common mechanisms explored in the literature include flywheel systems, tension springs, and elastic bands, each offering unique advantages and challenges. Comparisons between the three mechanisms are presented in the table below:

Mechanism	Advantages	Challenges	Performance Considerations
Flywheel (Post, 1973)	-Mechanically straightforward to implement.	- Requires refinement on the electrical and software sides, including designing a PID control system and precise angular speed calibration. - Sensitive to fluctuations in speed.	- Produces consistent outputs when rotational speed is steady. - Allows easy adjustment of launch distance via speed control.
Tension Spring (Du et al., 2019)	- Reduces reliance on electrical and software systems. - Potential for greater launching force and distance compared to flywheel mechanisms.	- Mechanically complex and challenging to implement. - Higher failure risk due to overloading or stress in the spring. - Difficult to control and estimate launch distance accurately.	- Capable of achieving greater force and distance. - Less precision in controlling launch distance compared to flywheels.
Elastic Band (Connolly, 2024)	- Potential for a high launching force.	- Mechanically challenging due to non-linear tension characteristics. - Prone to material fatigue and performance degradation. - Calibration is difficult due to variability in tension and material wear.	- Less consistent output compared to flywheels. - Less predictable and reliable over time due to wear and fatigue.

Table 3.2.1a: Comparison of Launching Mechanisms

Flywheel (VEX Forum, 2015)

Flywheel-based launch mechanisms are commonly utilised in projectile systems due to their mechanical simplicity and ability to produce consistent launch velocities when calibrated effectively. The fundamental principles of projectile motion suggest that the optimal launch angle for maximum range in the absence of air resistance is 45°, with range determined by the equation

$$d = \frac{v^2 \sin(2\theta)}{g}$$

where v is the initial velocity and θ is the launch angle. However, the presence of air drag significantly influences the trajectory, particularly for lightweight projectiles with low ballistic coefficients.

In flywheel mechanisms, energy is transferred to the projectile through rotational momentum, with factors such as flywheel speed, surface uniformity, and friction at the contact points playing critical roles in determining launch accuracy and range. Some notable findings regarding flywheel mechanism include:

1. Interaction between the projectile and flywheels results in both linear acceleration and potential spin.

2. Maintaining consistent rotational speeds and balanced flywheels can minimise energy loss and improve trajectory predictability.

3. In ideal setups where both flywheels operate at equal speeds, the projectile exits with minimal spin. Deviations such as unequal speeds or irregular wheel surfaces can introduce spin and directional inaccuracy.

4. Lightweight projectiles such as foam or plastic balls experience significant air resistance and may exhibit lift or drag variations due to spin.

Tension Spring (KB Delta, 2017)

A tension spring launching mechanism relies on the principles of energy storage and release, governed by Hooke's Law, which states that the restoring force of a spring is proportional to its displacement:

$$F = -kx$$

where k is the spring constant. As the spring is stretched, elastic potential energy is stored:

$$U = \frac{1}{2}kx^2$$

which is subsequently converted into kinetic energy to propel an object:

$$KE = \frac{1}{2}mv^2$$

The efficiency of this energy transfer depends on factors such as the spring's material, environmental conditions, and the magnitude of applied stress. Proper selection of spring parameters including the spring constant and material ensures optimal performance and safety.

Elastic Band (Tru Physics, 2023)

The elastic band launching mechanism operates similarly to the tension spring, utilising elastic potential energy. When stretched, the elastic band stores potential energy which is then converted into kinetic energy when released. However, compared to a tension spring, the elastic band's material properties such as its stretchability and resilience play a more prominent role. The efficiency of energy transfer in an elastic band mechanism also depends on factors like the band’s thickness, stretch limits, and environmental conditions, ensuring optimal performance for launching.

Besides, elastic bands are more prone to material fatigue and performance degradation over time. With repeated use, the band may lose its elasticity, reducing the efficiency of energy transfer. Calibration of the launcher can be challenging due to variability in the tension and wear of the material. These factors make it difficult to maintain consistent performance as the material properties change and stress accumulation may cause unpredictable results during launches.

3.2.2 Reloading Mechanism

In addition to the launching mechanism, the dart reloading mechanism plays a critical role in the system's performance, as it directly impacts the ability to launch four darts consecutively within the competition's 30-second timeframe. The choice of mechanism must strike a balance between mechanical complexity, reloading speed, and compatibility with the chosen launching system. The comparisons between three reloading mechanisms are presented in the table below:

Mechanism	Key Features	Advantages	Challenges	Performance Considerations
Top-loading	-Involves multiple mechanical and electrical components. -Adds an additional layer above the launching area.	-Raises the centre of gravity, allowing compatibility with vertical designs.	-Higher complexity due to mechanical and electrical integration.	-Suitable for **tension spring** and **rubber band** mechanisms; incompatible with flywheel mechanisms.
Revolver	-Requires precise alignment for accurate dart positioning.	-Efficient reloading process.	-High precision needed for alignment.	-Compatible only with **flywheel** mechanisms
Linear (Jang et al., 2019)	-Simplest design with a rack and pinion track to hold darts.	-Straightforward implementation.	-Requires an extended track for holding darts.	-Compatible only with **flywheel** mechanisms.

Table 3.2.2a: Comparison of Reloading Mechanisms

3.3 Relevant Research on Aerodynamics

Aerodynamics is very relevant to the design and process of Dart, both Passive and Active systems require a robust, consistent, and reliable aerodynamic Dart for tuning and iteration. Below, outlined are the Aerodynamic principles behind the Dart design.

In the context of a small projectile like Dart developed from RMUC, stability and flight path of a small projectile, such as a dart, are tied to the interplay between its centre of gravity (CG), centre of pressure (CP), and angle of attack (AoA). These parameters collectively determine the aerodynamic behaviour, stability, and control characteristics of the dart during its trajectory.

Center of Gravity:

The CG is the effective point where the dart's mass is concentrated. Its position impacts stability through the moment arm it creates relative to the CP. The pitching moment MMM about the CG is given by:

$$M = (X_{CP} - X_{CG})L$$

where X_{CP} and X_{CG} are the longitudinal positions of the CP and CG, respectively, and L is the aerodynamic lift force. For stability, $X_{CP}>X_{CG}$ ensures a restoring moment when the projectile is perturbed.

To quantify the effect of the CG's position, the static margin SM is defined as:

$$SM = \frac{X_{CP} - X_{CG}}{c}$$

where c is the chord length. A larger SM improves stability but decreases manoeuvrability, while a smaller SM may lead to instability (Caughey, 2011).

Centre of Pressure:

The Center of pressure is the reference point where aerodynamic forces act upon, the Aerodynamic moment about the aerodynamic centre is given by:

$$M_{AC} = C_{M,AC}QSc$$

Where $C_{M,AC}$ is the moment coefficient about the AC, Q is the dynamic pressure, S is the reference area and c is the chord length. The Center of pressure location can be derived by balancing moments:

$$X_{CP} = X_{AC} - \frac{C_{M,AC}}{C_L}$$

As C_L, the coeff of lift increases with ant angle of attack, X_{CP} moves forwards, this corresponds with a destabilising effect on the projectile. This further shows us the importance of ensuring the centre of pressure remains behind the CG of the projectile for optimal flight. (Nelson, R.C. , 1998)

Dynamic Stability and Restoring Moments:

The dynamic stability of the dart is highly dependant on the damping characteristics, which depends on the relationship between CG and CP, the restoring moment can be expressed as:
(Etkin, B., & Reid, L. D. ,1996)

$$\Delta M = -\rho V^2 S \frac{\Delta \alpha}{\Delta t} l_{eff}$$

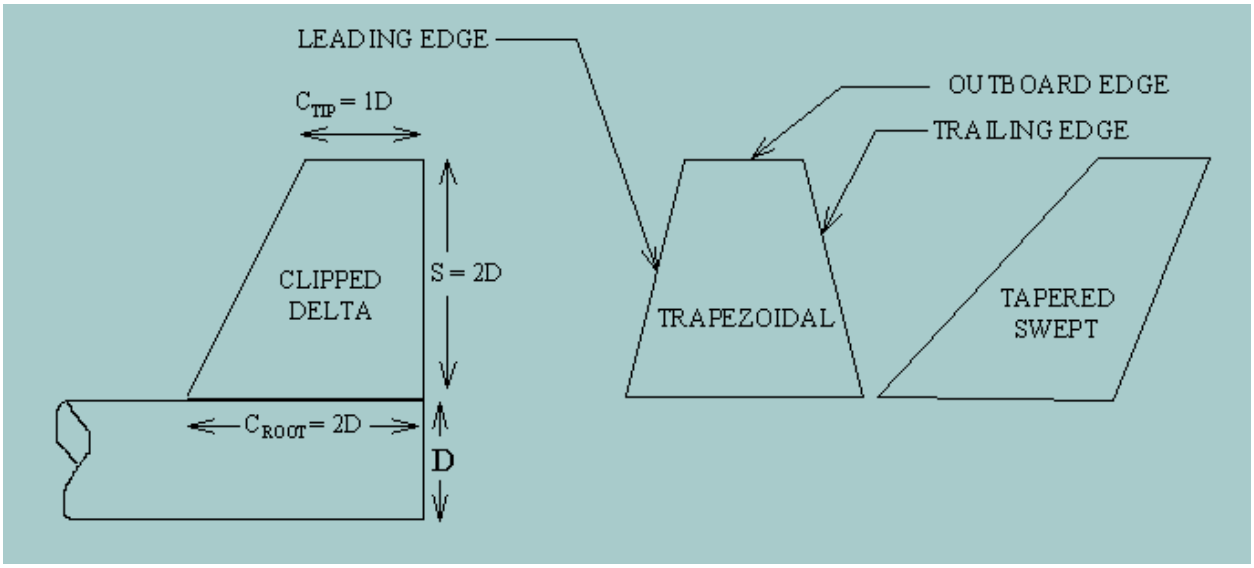
Where l_{eff} is the effective lever arm between the CG and CP. The restoring moment becomes oscillatory if the damping ratio ζ satisfies:

$$\zeta = \frac{c_d}{2\sqrt{km}} \quad \text{where} \quad k = \frac{\partial M}{\partial \alpha}$$

Where C_d is the damping coefficient, m is the mass and k is the stiffness derived from aerodynamic forces.

Fin Design and its Aerodynamic properties

Fin design is crucial for stabilising small projectiles, ensuring aerodynamic stability, this is because the fin design affects the centre of gravity and centre of pressure. The primary purpose of the fins are to generate the restoring force that counters the disturbances during flight and during the launch. As noted by Nakka (n.d.), the exact shape of the fins is less critical than their span and ability to position the CP appropriately behind the CG. However, some planform designs are better suited for specific conditions, offering trade-offs between aerodynamic performance and durability.



Trapezoidal and Clipped Delta Planforms

1. **Trapezoidal Planform:**
    - The trapezoidal fin has straight leading and trailing edges. It strikes a balance between aerodynamic efficiency and durability.
    - One notable advantage is its structural resilience during landing. The trailing edge is forward of the body tube's end, reducing the likelihood of damage upon impact (Nakka, n.d.).
2. **Clipped Delta Planform:**
    - This design, which features a forward-swept trailing edge, combines lower drag with ease of construction. However, its trailing edge's proximity to the body tube increases the risk of bending or damage(Nakka, n.d.).
3. **Tapered Swept Planform:**
    - The tapered swept fin is particularly well-suited for high-performance small rockets due to its aerodynamic and stability benefits. It is essentially a combination of the 2 with the added benefit of moving the CP even further back, allowing for maximum stability.

3.4 Relevant Research on Electrical Controls

3.4.1 Selection of MCU Board

Board	ESP32	STM32 blue pill	Arduino UNO
Processors	32-bit, Dual-core Xtensa LX6	32-bit ARM Cortex M3	8-bit ATmega328P
CPU Clock	80MHz - 240MHz	72MHz - 168 MHz	16 MHz
Timers	16-bit Prescaler 64-bit counters	4x 15-bit	1 x 8-bit, 2 x 16-bit
Memory	520 KB RAM and 448 KB ROM	20KB RAM	2 KB SRAM
Flash Size	4MB	64KB	32 KB (0.5 KB for bootloader)
Peripherals	34 programmable GPIOs, 12-bit SAR ADC (up to 18 channels), 2× 8-bit DACs, 10 touch sensors, and multiple communication interfaces (SPI, I²S, I²C, UART, CAN)	USB 2.0 OTG HS and FS, CAN 2.0B, SPI, I²S, I²C, USART, UART, SDIO, timers, watchdog timers, temperature sensor, ADCs, DACs, GPIOs, DMA, RTC, CRC engine, and RNG engine.	ADC, UART, SPI, I2C, PWM, GPIO
i//p Board Power /alternative b.p.	5V USB, Micro-B 3.3V (V_IN)	5V USB Micro-B 2.0V-3.6V	5V USB power
Program Loading	UART/USB, Over-the-Air (OTA) updates	Serial Wire Debug (SWD) ST-LINK/V2, UART	USB bootloader
Wireless	Wi-Fi Trans Receiver 802b/g/m Bluetooth 4.2 BLE	NA	NA

Table 3.4a: Comparison of potential MCUs

3.4.2 Vibrations

The existing dart launcher system developed was suspected to suffer from inconsistencies, due to potentially high levels of vibration. If this hypothesis is correct, vibration indicators can be instrumental in diagnosing the issue.

By accurately measuring vibrations and identifying the primary axis along which vibrations occur, we can pinpoint the root cause of instability in the launcher, as well as verify if our new launcher system is quantitatively better. This insight will help in refining the mechanical design and ensuring more consistent and reliable performance. The team started with exploring different vibration sensors and accelerometers to explore suitable sensor options for our use case, and reviewed the working principles of both types of sensors below.

Feature	Vibration Sensors	Accelerometers
Measured Quantity	Vibration/Displacement	Acceleration
Sensitivity	Moderate	High
Frequency Range	Specific frequencies	Wide frequency range
Measurement Principle	Piezoelectric/Resistive	Capacitive/Piezoelectric
Output Signal	Analog or Digital	Analog or Digital
MCU Interfacing	May provide simple digital output or analog signal	Requires analog-to-digital conversion or digital interface; may need signal conditioning
Application	Detecting machine faults, equipment monitoring	General-purpose sensing (motion, tilt, impact)
Typical Units	Velocity/Displacement (mm/s, m/s^2)	g (acceleration due to gravity)
Integration Complexity	Simpler, often single-axis	Higher (often 3-axis)
Cost	Can be much less expensive	Generally more expensive

Table 3.4b: Comparison of (in General) Accelerometers against Vibration Sensors

3.4.2.1 Vibration sensors

Vibration sensors encompass a broader range of devices designed to measure different aspects of vibration, such as acceleration, velocity, or displacement, but mostly referring to the latter two.

Vibration sensors, also known as vibration transducers, are often used to measure the vibration of machinery or structures. A common type is the piezoelectric vibration sensor, which operates based on the piezoelectric effect: certain materials generate an electric charge when mechanically stressed. In piezoelectric vibration sensors, mechanical vibrations cause stress on a piezoelectric material (such as quartz or certain ceramics), producing an electrical charge proportional to the force of the vibration. This charge is typically converted into a voltage signal through a charge amplifier. The voltage signal represents the vibration's amplitude and frequency, which allows for detailed analyses of the source of vibration.

Alternatively, some vibration sensors use microelectromechanical systems (MEMS) technology similar to accelerometers but are optimised for vibration detection over specific frequency ranges. These sensors can provide high-resolution measurements of vibration parameters and are suitable for integration into electronic systems.

The output from vibration sensors can be analog or digital. Analog signals often require signal conditioning before interfacing with an MCU, while digital sensors can communicate using standard protocols.

A few potential vibration sensors for consideration are explored below:

Model	Measurement Range	Output Type	Voltage Range	Sensitivity	Interface	Notable Features
Sencera 801S	0 - 50g	Digital	5 V	N/A	Digital Output	Sensitivity Adjustment
SW-420	Threshold-based	Digital (Switch)	3.3 V – 5 V	N/A	Digital Output	Simple vibration detection
Dytran 3035B	±500 g	Analog	18 V – 30 V	10 mV/g	Coaxial Connector	Wide frequency range
PCB 352C33	±50 g	Analog	18 V – 30 V	100 mV/g	Coaxial Connector	High sensitivity, industrial applications

Table 3.4d: Comparison of Potential Vibration Sensors

3.4.2.2 Accelerometers

Accelerometers are electromechanical devices that measure acceleration forces. These forces may be static, like the constant force of gravity, or dynamic, caused by moving or vibrating an object. Modern accelerometers commonly use Micro-Electro-Mechanical Systems (MEMS) technology, integrating microscopic mechanical structures onto silicon chips. The fundamental working principle involves a proof mass suspended within the sensor. When acceleration occurs, the proof mass experiences inertia, causing it to displace relative to the sensor housing. This displacement alters an electrical property, such as capacitance, piezoresistive value, or the piezoelectric effect, depending on the accelerometer type.

- **Capacitive** Accelerometers: Measure changes in capacitance caused by the displacement of the proof mass between capacitor plates. The variation in capacitance is proportional to the acceleration. This method is widely used in MEMS accelerometers due to its sensitivity and ease of integration.
- **Piezoresistive** Accelerometers: Utilise materials whose electrical resistance changes under mechanical stress. The deformation of the proof mass under acceleration alters the resistance, which is measured and converted into an acceleration value.
- **Piezoelectric** Accelerometers: Employ piezoelectric materials that generate an electric charge when subjected to mechanical stress. However, they are generally not suitable for measuring static acceleration (like gravity) because the charge dissipates over time.

The output signals from accelerometers can be analog voltages or digital data. Analog outputs require analog-to-digital conversion when interfacing with a Microcontroller Unit (MCU), while digital outputs can communicate via protocols like I²C or SPI.

A few potential accelerometers for consideration are explored below:

Model	Axes	Output Type	Voltage Range	Measurement Range	Interface	Notable Features
ADXL335	3-axis	Analog	1.8 V – 3.6 V	±3 g	Analog Output	Analog Output
ADXL345	3-axis	Digital	2.0 V – 3.6 V	±2/4/8/16 g	I²C/SPI	High resolution (13-bit), tap detection
MMA7361	3-axis	Analog	2.2 V – 3.6 V	±1.5/6 g	Analog Output	Selectable sensitivity, sleep mode
BMA220	3-axis	Digital	1.62 V – 3.6 V	±2/4/8 g	I²C	Ultra-small package, low noise

Table 3.4c: Comparison of Potential Accelerometers

3.5 Relevant Research on Computer Vision

Based on the project objectives, extensive research conducted on the selection of cameras and boards, tracking algorithms with the implementation of a real-time system.

Selection of camera

Considering the context of the RoboMaster Competition and the requirement of real-time processing, the camera is expected to meet the requirements below:

- High resolution to detect the target at 15-30 metres away
- Fast frame rate for real-time tracking
- Small size to fit within the dart's constraints
- Software development kit (SDK) or application programming interface (API) availability for easier integration with software

Detailed comparison of six small cameras satisfying the requirements is outlined in the Table 3.2a:

Camera	Resolution (pixels)	Frame Rate (FPS)	Size (mm)	Connectivity	**Pros**	**Cons**
IMX219 Camera 160 FOV	3280 × 2464	30	25 × 24	CSI	- High resolution - Widely supported - Large field of view	- Moderate frame rate for fast-moving objects
Raspberry Pi Camera 5MP	2592 × 1944	Up to 60	25 × 24	CSI	- Good balance of resolution and frame rate - Widely supported	- Requires Raspberry Pi - Fixed lens
Raspberry Pi Camera Module V2	3280 × 2464	30	25 × 24	CSI	- High resolution - Widely supported	- Requires Raspberry Pi - Moderate frame rate
Pixy2	640 × 400	Up to 60	50 × 40	USB, UART, SPI, I2C	- Advanced tracking algorithm based on colour - Built-in image processing	- Moderate resolution - Large size

OpenMV H7	640x480	Up to 150	45 × 36 × 32	USB, UART, SPI, I2C	- Built-in machine vision capabilities - Easy programming - Fast frame rate	- Moderate resolution - Large size
OpenMV H7 Plus	1920 × 1080	Up to 60	36 × 36	USB, UART, I2C, SPI	- Built-in machine vision capabilities - Easy programming	- Large size

Table 3.5a: Comparison of potential cameras

Selection of boards

With relevant research on the selection of boards, the requirements of boards is summarised as below:

- Camera interface compatibility
- Small size to fit dart dimensions
- Sufficient processing power for CV algorithm
- Supported programming languages and libraries

Detailed comparison of four boards meeting the requirements is outlined in the Table 3.2a:

Board	Interface	Size (mm)	Pros	Cons
Raspberry Pi Zero 2W	CSI, USB	65 x 30	- Compact - Raspberry Pi ecosystem - Good camera interface compatibility	Lower processing power
Raspberry Pi 3B+	CSI, USB	85 x 56	- High performance - Raspberry Pi ecosystem - Good camera interface compatibility	Larger size
Raspberry Pi 3A+	CSI, USB	65 x 56.5	- Balanced size and performance - Raspberry Pi ecosystem	Slightly larger than Zero 2W

			- Good camera interface compatibility	
Arduino Nano 33 BLE	I2C, SPI	45 x 18	- Compact - Arduino compatibility	- Limited processing power - Limited interface

Table 3.5b: Comparison of potential boards

Tracking algorithm

A real-time system that needs to deal with live video streams requires a high performance tracking algorithm. To design a tracking algorithm, the first step is to decide the description of the object to track such as a shape, texture and colour (Yang et al., 2011). Given there is a green guiding light on the detection module in this project's context, the green light would be considered as the most straightforward approach to locate the target. Therefore, the main focus of the research on tracking algorithms was colour tracking.

Colour tracking is a conventional tracking process for systems with high-moving objects like dart, because colour information represents the global feature of objects, which are relatively independent of the viewing angle, translation, and rotation of the objects (Yang et al., 2011). There are two main colour models to represent colour information - RGB colour model and HSV colour model (Gajbhiye & Gundewar, 2015) as shown in Figure 3.2a.

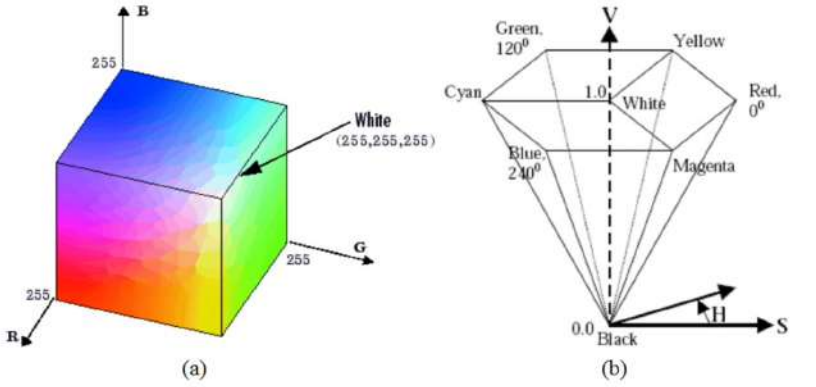


Figure 3.5a: Colour models (Gajbhiye & Gundewar, 2015)

RGB colour model is denoted by the volume of red, green and blue that exists in the colour space. RGB information is represented in three 0-255 numbers, each present one primary colour. For example, the RGB representation of purely green is (0, 255, 0). By comparison, HSV colour model uses the volume of hue, saturation and value exists in colour space to denote, where hue stands for the property and type of colour range from 0 to 360, saturation stands for purity of colour range from 0 to 100, and value stands for brightness of colour range from 0 to 255. The conversion between RGB and HSV can be implemented easily by some equations (Gajbhiye & Gundewar, 2015).

Thresholding is a widely used image-processing operation that utilises colour information to segment an image into two or more parts - target objects, known as foreground, and the rest objects, known as background (Goh et al., 2017). Predefined colour boundaries are established for the target objects, enabling every pixel in the image to be analysed and classified according to these thresholds, which may include specific colour or grayscale values. This segmentation process divides the image into distinct segments, allowing for effective identification and localization of the foreground. Morphological operation is another basic image-processing operation which is usually applied after thresholding. It is used to adjust object structure by either thinning (erosion) or thickening (dilation) of the object. Noise can be removed and holes can be filled by this operation (Gajbhiye & Gundewar, 2015).

A common technique to do foreground segmentation named background subtraction is an example of effective combination of thresholding and morphological operation as shown in figure 3.2b. In this method, the thresholding can be easily achieved by comparing the current image with a reference background image. The pixels where the referenced image does not match are considered as the targeted object.

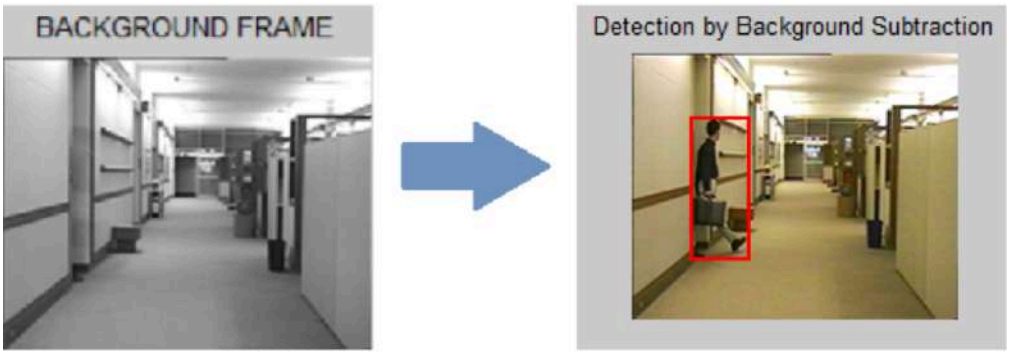


Figure 3.5b: Detection Using Background Subtraction Modeling (Goh et al., 2017)

In real-time object tracking, several challenges may arise due to factors such as significant partial or full occlusion, camera motion, variations in object and environment appearance, shadows, and the presence of multiple objects with similar colours. To address these complexities, several potential solutions are identified and summarised:

- **Handling Occlusion**: One approach is to use a particle filter method with colour features, which tracks the object's colour distribution over successive frames, even if the object is not fully visible. The grey world assumption combined with particle filtering using easy-MOSSE or CSR-DCF can further help by assuming that the average colour of an image scene is grey, thus improving robustness under varying lighting conditions (Liu et al., 2020).

- **Improving precision with discriminative methods**: Discriminative methods which incorporate inputs like motion models, feature extraction, and observation models, allow for more complex decision-making processes. These methods adjust their model updates based on observed changes, leading to more precise outputs and reducing false positives in tracking (Liu et al., 2021). Although this method is robust, the processing power needed might be high.

- **Handling multiple object detection**: One efficient approach is the optical flow method. This method uses the motion vectors of the moving object across time to help locate the target object. If the targeted object has been "remembered", there will be its presence in the next frame somewhere with almost the same colour information (Jansari et al., 2013). The detailed procedure is outlined in the flowchart below (Figure 3.2b).

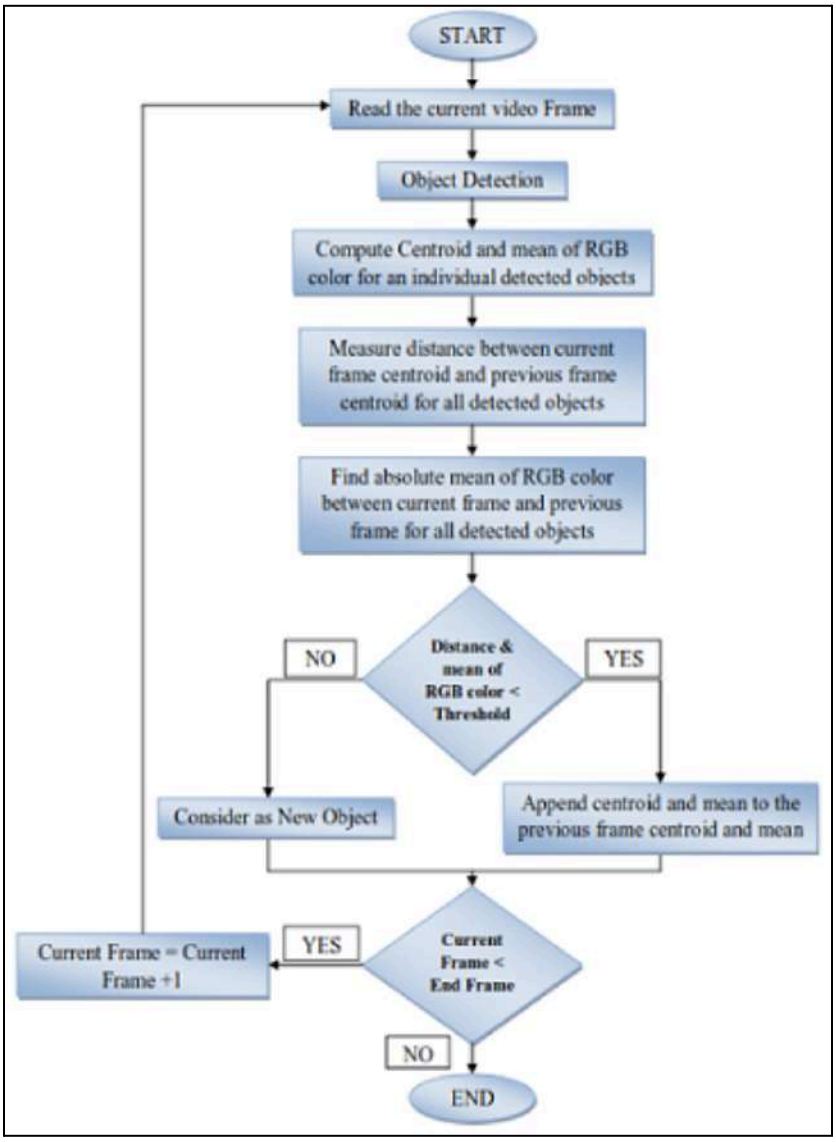
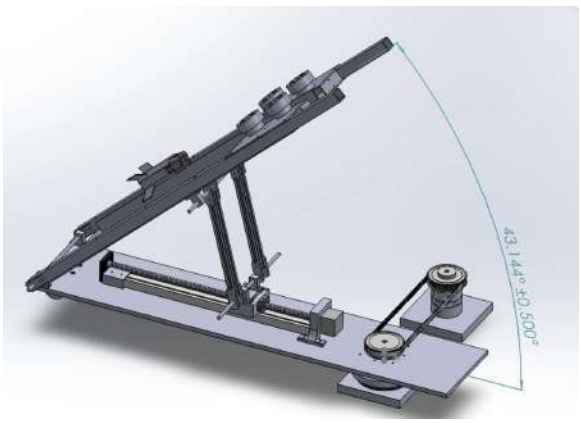


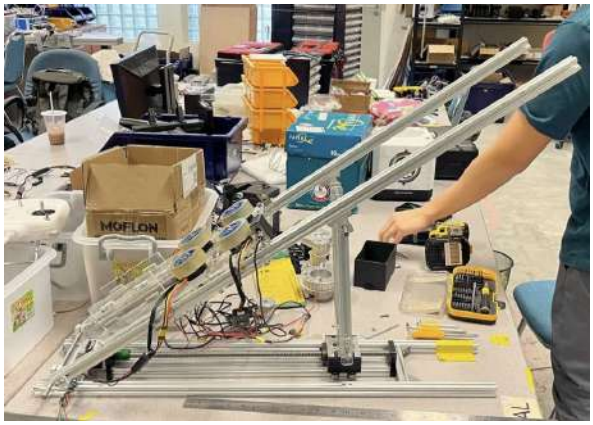
Figure 3.5c: Flowchart of colour tracking algorithm using optical flow method (Jansari et al., 2013)

3.6 Review of Existing Dart System

Dart Launcher

The initial iteration of the dart launcher developed by the NUS Calibur Robotics dart team has exhibited significant limitations in achieving consistent flight paths for successive darts, primarily attributed to structural instability and inconsistencies in the flywheels during the launching process.





Figures 3.6a, 3.6b: First Iteration of Dart launcher

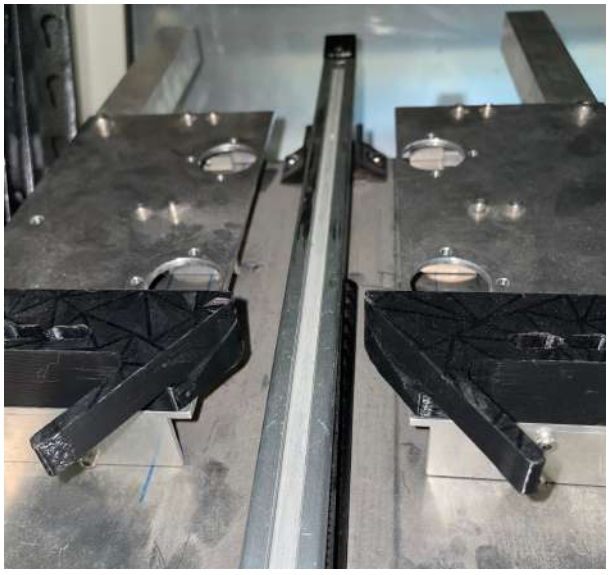


Figure 3.6c: Small Gap of Old Launcher

Figure 3.6d: Damaged Flywheel Rubber

Figure 3.6e: Inaccurate Launch due to inconsistent Flywheels Angular Speed

Figure 3.6f: Imbalance Base

(NUS Robomaster Archives)

Comprehensive testing and analysis have revealed several key factors contributing to the suboptimal performance of the dart launcher:

- The opening gap through which the dart must pass is insufficiently sized and does not accommodate the fins of the dart, leading to collisions during launch (see Figure 3.6c).
- The rubber surface of the flywheel has sustained damage during testing, resulting in an uneven surface characterised by dents and imperfections (see Figure 3.6d).
- The angular speeds of the four flywheels are not properly calibrated, causing variations in rotational speeds and resulting in inconsistent launch dynamics between the left and right sides of the dart (see Figure 3.6e).
- The base of the dart launcher fails to maintain a flat orientation on the surface due to protruding bolts, which can compromise stability (see Figure 3.6f).

These deficiencies will be addressed in the project to enhance the accuracy and consistency of the dart system in subsequent iterations.

Dart

The initial vision system for the NUS Calibur Robotics dart team was a prototype developed using the OpenMV H7 Camera and OpenMV board, designed for colour tracking

Thorough testing and analysis, several critical limitations of this prototype became evident:

- The OpenMV H7 Camera's resolution proved insufficient for the team's requirements. Specifically, the camera was unable to accurately detect targets beyond a distance of 2

metres, far short of the team’s goal of enabling the dart to aim at targets 15–30 metres away.

- The OpenMV H7 Camera was too large (45 × 36 mm) to integrate within the dimensions of the existing dart design (26 × 26 mm).
- There were no actuators to adjust the dart's direction, making the vision system incomplete.

Chapter 4 | Engineering Design Process

4.1 Concept Generation and Selection

Based on the literature review, several potential concepts for the dart launcher design, dart design, electrical system, and computer vision system have been identified and analysed. This section performs a comparison between each concept and provides the rationale behind the final selection for each mechanism that the team will adopt moving forward.

4.1.1 Dart Launcher

Launching Mechanism:

Mechanism	Difficulty	Consistency	Cost	Risk of failure	System lifespan	Experience
Flywheel	Medium	8/10	High	Low	Long	Yes
Tension Spring	Hard	5/10	Medium	Medium	Short	No
Elastic Band	Hard	6/10	Low	Medium	Short	No

Table 4.1.1a: Evaluation of Launching Mechanism Choices

Final Selection: Flywheel Mechanism

Justification: Flywheel mechanism offers the highest reliability and greater margin for adjustments within the project scope. Evidence from testing by other university teams indicates that the flywheel mechanism delivers more consistent launch performance and avoids issues related to material fatigue and the non-linear tension characteristics inherent in spring and rubber band mechanisms. Additionally, the team possesses prior experience working with flywheel systems, as the previous iteration of the dart launcher also employed this mechanism. This familiarity provides the team with a deeper understanding of the design, operation, and optimization of flywheel-based systems.

Feeding Mechanism:

Mechanism	Difficulty	Consistency	Cost	Risk of failure	Efficiency	Space Consumption
Top-loading	Hard	7/10	High	High	Medium	Large
Revolver	Hard	6/10	Medium	Medium	High	Medium
Linear	Easy	9/10	Medium	Low	Low	Large

Table 4.1.1b: Evaluation of Feeding Mechanism Choices

Final Selection: Revolver Feeding Mechanism

Justification: The revolver feeding mechanism provides an efficient reloading process without the need for an extended track. While it demands high precision for dart alignment, this challenge can be mitigated through the use of a stepper motor. Additionally, the revolver feeding mechanism is compatible with the flywheel launching system, which has been selected as the preferred mechanism for this project. Lastly, the revolver feeding mechanism requires the least amount of space among the three options, making it the most compact choice for the dart launcher design.

4.1.2 Dart

Mechanical Design of Dart:

The 2 main considerations of the Dart Design are to firstly create an Aerodynamically Viable Dart design, Secondly, the design should be modular to allow for easy iteration of the design for future prototyping (eg. different fin configurations, Active vs. Passive Dart systems, different types of interference fit for flywheels).

The following subsections will detail the main factors that have led to the current design.

Aerodynamic Considerations:

Based off initial tests of the previous prototype, it was noted that the stability and hence predictability of the flight path was largely based off the initial trajectory (potential rotational forces imparted onto the dart by the old flywheels and untuned motors) as well as the CG of the dart.

To combat potential instability, we concluded we have to optimise the pitching, yawing and rolling coefficients. These coefficients directly correspond to the magnitude of the restoring force when perturbed in pitch or yaw; extreme values would correspond to either over or under corrections to such disturbances.

To iteratively test these parameters, we have developed a modular dart whereby the aerodynamic surfaces on the body as well as the fins of the dart can be swapped out for different geometries and their respective aerodynamic properties.

It was also postulated that a CG at the front of the Dart is optimal for stable flight. By placing the CG further to the front, the distance between the centre of pressure (COP) and the centre of gravity. During the flight path, there is a change in the angle of attack in the parabolic flight, the restoring moment generated by the fins at the back of the Dart (COP); By moving the CG forward, we can hence strengthen the stabilising effect. Additionally by shifting the CG forwards, we also reduce the Dart's sensitivity towards angular disturbances by increasing the moment of inertia.

The effect of CG can be studied by varying the CG iteratively using taped on masses in the internal structure of the Dart, allowing us to iteratively find the optimal placement of CG.

It is also notable that during the initial design process, we attempted to develop a model based on aerodynamic simulations on Matlab. However, we were quick to face difficulty in getting the equations to converge. This was often due to the unique angle of attack of the Dart which does not resemble a conventional "plane" which has much larger aerodynamic and control surfaces, rendering most assumptions invalid.

Final Aerodynamic Property of the Dart:

To provide the Dart with sufficient restoring focus without introducing instability together with providing sufficiently low drag we concluded with these final design choices.

- Front heavy Dart, (meaning most of the 3d Print mass has to be at the front)
- No camber on the fins resulting in no pressure difference between top and bottom surfaces of the fins
- There is a net 0 lift provided by Fins
- Swept back trapezoidal Fin design

Production and Assembly Considerations:

Material Considerations:
Previous iterations were mostly a single construction printed out of TPU for its high flexibility and strength, however its relatively higher density 1.2g cm^-3 made initial prototypes relatively heavy for the size.

- High Density of 1.2g cm^-3, similar to PLA
- High flexibility and strength
- Intricate parts are harder to print due to stringing of 3d print

Similar to other teams, we experimented using PLA Aero with a density of 0.54g cm^-3, which allowed us to drastically decrease the weight of the body, allowing us to vary CG to a greater degree as well as dedicate more weight to the internal components which are even more significant in the case of an active dart.

Strength Requirements:
The Darts are made to be Disposable after a few launches, we expect parts like the fins to account for majority of the breaks due to their relatively thinner profile, and high likelihood of bending during impacts.
Material selection wise, this aligns with the choice of PLA Aero, with it being less brittle than PLA but less flexible than TPU.

To increase strength of the 3d print, print orientation was considered, with the vertical print orientation for the body being much stronger for the "head first" collisions that come with head heavy dart in a parabolic path. Similarly, the fins are printed flat.

3D Printing Specifics:
It is notable that PLA Aero has dynamic densities based off print temperature, this is due to the foaming property of PLA Aero

Printing Temperature	Minimum Flow Rate Ratio	Maximum Volumetric Expansion Ratio
190 °C	~ 0.95	~ 100%
200 °C	0.89	110%
210 °C	0.85	115%
220 °C	0.76	129%
230 °C	0.62	158%
240 °C	0.45	217%
250 °C	0.38	258%
260 °C	0.35	280%
270 °C	0.37	265%

For the prototypes, we are printing at a density of 0.62, trying to strike a balance between strength, density and ease of printing.

Assembly Considerations:

- Assembly has to work for both Active and Passive designs
- Easy access to electronics
- Ability to Vary CG easily without a “redesign”
- Easily replace broken parts
- Easy access to charging

Future Considerations:

- Potentially consider stronger materials for the fins such as carbon fibre (I think one of the team’s active dart has carbon fibre fins)
- Making assembly of the dart easier and require less pieces / screws, achieving less points of failure

Active Dart Construction Considerations:

Methods Considered to Steer the Dart:

- Using Roll to affect the trajectory / steer the dart
    - Potentially control using 1 servo but also require a more complex internal design
- Using the back of the fins as a “rudder”
    - 4 independent servos, 1 per fin
    - 4 servos, with 2 servos “coupled” using software
    - 2 independent servos, coupling the fin into “one long fin” mechanically

It was decided to use 2 independent servos to save on weight and allow for a more compact construction. This is in line with the prototypes seen from the Chinese teams, which controlled their fins with external servo rods.

4.1.3 Electrical System

In building a new dart launcher, we were aware that we had to integrate a rotary motor and two servo motors.

- A rotary motor is required to spin the revolver mechanism structure by 90 degrees accurately.
- A servo motor is required to push a dart out, via a dart rail, from a dart holder to the launching platform where the dart gets sent flying by the flywheels' propulsion.
- Another servo to implement a final alignment and locking mechanism of the revolver mechanism frame in every 90-degree rotation

The Robomasters Development Board C (Dev-C), which runs on STM32f407IGH6, is generally used by Robomaster students to operate robots. Likewise, in the old dart launcher design, the flywheels have been integrated with the Dev-C board to propel the darts. Hence, the initial plan for electrical controls of the entire new launcher design was to integrate the above components with the Dev-C as well, along with a vibration sensing component.

4.1.3.1 Vibration Testing

<u>Accelerometer ADXL335:</u>

At the beginning, the team acquired an ADXL335 accelerometer from NUS Electronics Workshop (E2A #02-02). It has a resonant frequency of 5500Hz. The team noted the difference in frequency bandwidth for the three axes of operation but proceeded to integrate it since the mounting orientation can be changed afterward if necessary.

Preliminary testing of the ADXL335 using Arduino UNO shows that the three axes of the ADXL335 accelerometer change in value when shaken lightly (approximately 3Hz, 1 cm displacement). In Figure 4.2a - 4.1c below, the serial monitor graphs show the fluctuating accelerometer values against time.

Figure 4.1.3a: x-axis vibration

Figure 4.1.3b: y-axis vibration

Figure 4.1.3c: z-axis vibration

Although the accelerometer works, the team faced difficulty integrating it with the Dev-C.

The ADXL335 only outputs analog data - but the customizable IO ports of Dev-C are not compatible with analog. A digital-to-analog converter (ADC) is required to process the data output from the accelerometer because Dev-C can only read digital data.

More specifically, the team could not initialise any ADC peripherals from STM32Cube IDE because it was used by internal peripherals or functions, such as timers or communication interfaces. None of the user-accessible pins were able to access an ADC. *[ref: RoboMaster Development Board Type C Schematic Diagram]*

Vibration Sensor: Cytron 801S (Final decision)

Given that the ADXL335 accelerometer cannot output digital signals to integrate with the STM32 Dev-C board, the team switched to using the Cytron 801S vibration sensor which outputs digital signal. It is also easier to integrate with only 3 pins.

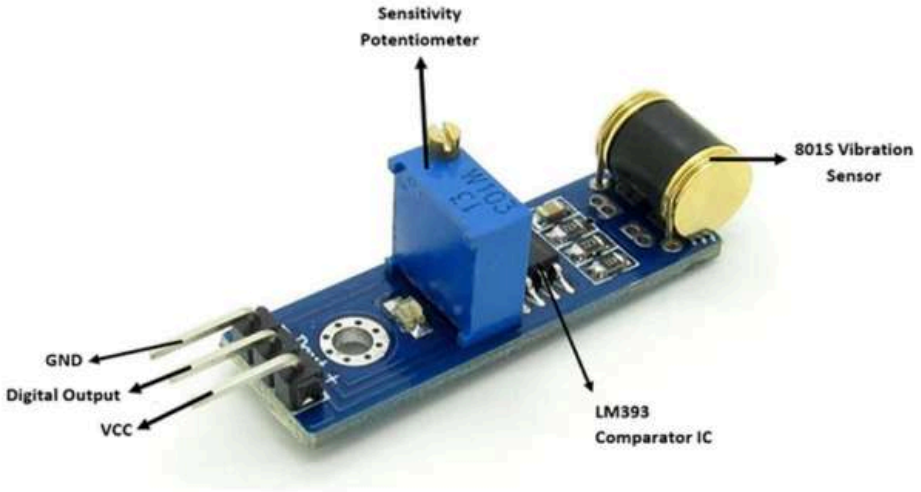


Figure 4.1.3d: 801S Vibration Sensor Pinout

Two high-precision Cytron 801S vibration sensors, that output digital data, were ordered to be compatible with the Dev-C. More specific characteristics for this vibration sensor has been discussed in the Literature Review Section.

It works by changing its resistance drastically when subjected to vibrations. The sensor is internally connected to a voltage divider circuit to get a voltage output, similar to many resistance-varying devices.

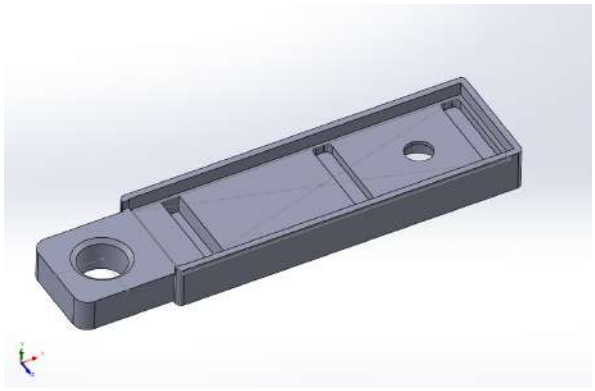


Figure 4.2.3e: Vibration Sensor Mount

The team noticed that the vibration sensor's breakout board had conductive terminals protruding from the bottom of the breakout board, and would short circuit if mounted directly onto the dart launcher frame. Three iterations of vibration sensor mounts were designed and 3D-printed in PLA to act as a compact insulator. The on-board M2 screw hole allows the 801S to be screwed tightly onto the mount above; the mount can be screwed onto the dart launcher frame's M5 screw holes. In the end, epoxy was used instead to replace the screws to ensure that the mounting was as secure as possible so that in the long run (screws loosen over time especially with vibrations). The team then used this model to test for vibrations, and the results will be covered in a later section.

4.1.3.2 Rotary Motor for Revolver Mechanism

Brushless DC Gear Motor: DJI M3508 P19

Initially assuming that the team would develop the launcher entirely on Dev-C firmware, the team planned to use the existing RoboMaster motor controlled via Controller Area Network (CAN) communication. CAN offers robust, noise-resistant, and error-checked data transmission and is the communication protocol used for controlling the M3508 motor in RoboMaster robots.

The M3508 motor is attached to a gearbox with a gear reduction ratio of 3591/187 *[Page 18 of DJI M3508 datasheet]*. This ratio can be read as a fraction representing the factor to which the speed of the BLDC is reduced, and also the factor by which the torque of the motor is increased.

Using CAN protocol introduced some difficulty in implementing the gear motor. While CAN communication is robust and suitable for industrial applications, it introduced a steep learning curve for those unfamiliar with it. In addition, BLDC motors are inherently designed for continuous rotation and are excellent for applications requiring high speed and torque. They are not ideal for applications requiring precisely fixed positional control without additional components. Achieving exact angular movements, such as rotating exactly 90 degrees, is more complex with BLDC motors because it lacks inherent position-holding capabilities and requires more sophisticated closed-loop control systems to achieve precise positioning.

Stepper Motor: NEMA 23 (57BYG250B-8)

The team switched to stepper motors which are designed for precise positional control. Moving in discrete steps that allow for exact positioning without the need for complex feedback systems.

They can rotate a specific angle per input pulse, making them highly suitable for applications requiring accurate and repeatable movements and lowers the barrier to achieving reliable motor control. The team expected that implementing stepper motor control would be more intuitive and better aligned with their expertise.

This specific stepper motor model has the following characteristics:

- 200 steps per revolution × 1.8-degree step angle
- holding torque of 19 kg-cm
- 12kg torque (to rotate revolver)

Despite referencing datasheets for this stepper motor driver model (TB6600) and tutorials on similar stepper motor drivers, there was still some difficulty in getting the stepper motor to move as intended. Due to incomplete and unclear documentation on online datasheets, the team decided to test out the motor driver's operational settings in a hands-on manner. The conclusions are summarised below:

The STEPPER DRIVER has the following pins and their purposes:

Terminal on Driver TB6600	Function and Connection
ENA-	Enable pin (negative)
ENA+	Enable pin (positive)
DIR-	Direction pin (negative) (GND)
DIR+	Direction pin (positive) (DC 5V)
PUL-	Pulse Pin (negative) (GND)
PUL+	PWM Pulse Pin (positive)
B-	Connection to one coil of the stepper motor (negative)
B+	Connection to one coil of the stepper motor (positive)
A-	Connection to the other coil of the stepper motor (negative)
A+	Connection to the other coil of the stepper motor (positive)
GND	Ground
Vcc	Connect to the power supply (rated 3.36V, but we are connecting it to the 22.8V Robomasters TB48S battery)

Table 4.1.3a: Conclusion of Wiring DM6600 Terminals

Key highlights from reading and testing:

- Confirms that DIR- and PUL- are connected to the common ground (GND) of the Arduino UNO.
- How to use the enable pins (ENA+/-):
    - the motor is disallowed to move (even if PWM is inputted) when:
        - ENA- to GND, and
        - ENA+ to 5V
    - the motor is allowed to move when:
        - ENA- to 5V, and
        - ENA+ to GND;

        or

        - ENA- to 5V, and
        - ENA+ to 5V;

        or

        - ENA- to GND, and
        - ENA+ to GND;

On STM32Cube IDE, the team wrote a code to spin the M3508 motor in Dev-C. The team effectively programmed a manual switching between HI (5V) and LOW (0V) to create PWM signal while keeping it non-blocking. Non-blocking code ensures that all the interrupts required by the flywheels-controlling PID functions can run.

However, in testing, the stepper motor fails to move 90 degrees with every rotation, and consistently moves approximately 75 degrees. This is despite receiving the correct number of steps required. The team had checked the PWM signal using an oscilloscope as shown below.

Figure 4.1.3f: Oscilloscope showing PWM signal of 500.1 Hz, in an attempt to move it by 100 steps with a PWM frequency of 500 Hz. (Motor driver settings: 400 steps per revolution.)

The team tried other methods in attempt to move it by 90 degrees, such as toggling the enable pins such that power to the stepper motor is cut off for a set duration - but the angle fails to be precise and consistent. The team suspect a time lag in the power cuts, causing the motor to have some residue power to spin more than 90 degrees for a small fraction of time after the power can be cut.

The team also tried to reprogramme the PWM signal to be generated only for alternative multiples of 90 increments in step_count, and generate a DC for the duration during which the motor is meant to be static. This achieves better accuracy, but it only works when StepperMotor_OutputPulse(100), and is still not perfect. Deviation is seen after multiple revolutions are made.

Decision Point:

The team ultimately decided to switch to using an Arduino UNO instead of using Dev-C.

The following lists a few useful implications of using Arduino:

- The Arduino will be able to control electronics without interfering with the flywheel functions in STM32 since they are two distinct systems now. Therefore there is no more need to ensure the code is non-blocking.
- The implementation of all components are simplified. There is no need for configurations like setting up clock sources, timers, and there is also a wider community utilising the UNO board with our intended peripherals. This resource is helpful in debugging and for referencing.
- Servo motors will be easier to programme by accessing the in-built library via <servo.h>.

For easy sub-system integration testing, the team added a push button. The team obtained a four-pin push button but it was unlabelled. The team took it apart to explore the internal mechanism and saw the mechanical configuration. By pressing the button, two pins each pin on two halves of the button connect together. Hence, by wiring any two opposite pins, a press of the button connects the two pins, allowing current to flow.

The resultant integrated system is presented in the next section as our final prototype.

4.1.4 Computer Vision

Selection of Boards:

Board	Ecosystem	Camera Compatibility	Processing Power	Size
Raspberry Pi Zero 2W	Easy to use with Raspberry Pi ecosystem	High (almost all cameras)	High	Second smallest
Raspberry Pi 3B+	Easy to use with Raspberry Pi ecosystem	High (almost all cameras)	Higher than Zero 2W	Larger
Raspberry Pi 3A+	Easy to use with Raspberry Pi ecosystem	High (almost all cameras)	Higher than Zero 2W	Larger
Arduino Nano 33 BLE	Easy to use with many open sources	Limited	Lowest	Smallest

Table 4.1.4a: Evaluation of board choices

Final Selection: Raspberry Pi Zero 2W

Justification: Raspberry Pi Zero 2W takes the advantages of Raspberry Pi ecosystem and wide camera interface availability. It also balances well between the size and performance - the size is small and it has been proved to perform well with AI models like computer vision algorithms. Even if it failed to provide enough power, there is a potential solution - compile the code and generate a lower-level code executable file on laptop; save the file on the board to direct use without compile. This will reduce the power consumption (Gajbhiye & Gundewar, 2015)

Selection of Camera:

Camera	Resolution	Frame Rate	Field of View	Compatibility	Built-in Processing	Size
IMX219 Cam 160 FOV	Highest	Low	160° (Largest)	Good with RPi Zero 2W	No	Standard
Raspberry Pi Camera 5MP	Medium	Medium	Standard	Good with RPi Zero 2W	No	Standard

Raspberry Pi Camera Module V2	Highest	Low	Standard	Good with RPi Zero 2W	No	Standard
Pixy2	Medium	Medium	Standard	Compatible with RPi Zero 2W	Yes (tracking algorithms)	Largest
OpenMV H7	Low	Fastest	Standard	Not specified	Yes (tracking algorithms)	Large
OpenMV H7 Plus	HIgh	Medium	Standard	Not specified	Yes (tracking algorithms)	Large

Table 4.1.4b: Evaluation of camera choices

Final Selection: IMX219 Cam 160 FOV

Justification: IMX219 Cam 160 FOV performs great in most areas: good compatibility with the chosen board Raspberry Pi Zero 2W, high resolution rate and large field of view. Although the frame rate is slow, this can be solved by lowering the resolution rate. With testing and adjustments, the most balanced configuration could be found.

Tracking Algorithm:

Algorithm	Implementation Complexity	Power Consumption	Accuracy	Processing Time
Thresholding	Simplest	Lowest	Good for simple tasks	Fastest
Thresholding with Morphological Operations	Simple	Low	Solid for classic techniques	Fast
Enhanced Thresholding with Advanced Operations	Complex	High	Higher and more consistent	Longest

Table 4.1.4c: Evaluation of Tracking Algorithm

Final Selection: Thresholding with Morphological Operations

Justification: Given the high velocity of the darts after launch, the vision system operates under strict time constraints. Thresholding with basic morphological operations offers a balance of accuracy and speed, ensuring quick response times without heavy computational demands. Algorithms can be further enhanced for specific purposes or be further simplified.

4.2 Design and Prototype

This section outlines the design and prototyping process of the dart system, segmented into four main components: the dart launcher, the dart, the electrical system, and the computer vision system.

4.2.1 Dart Launcher

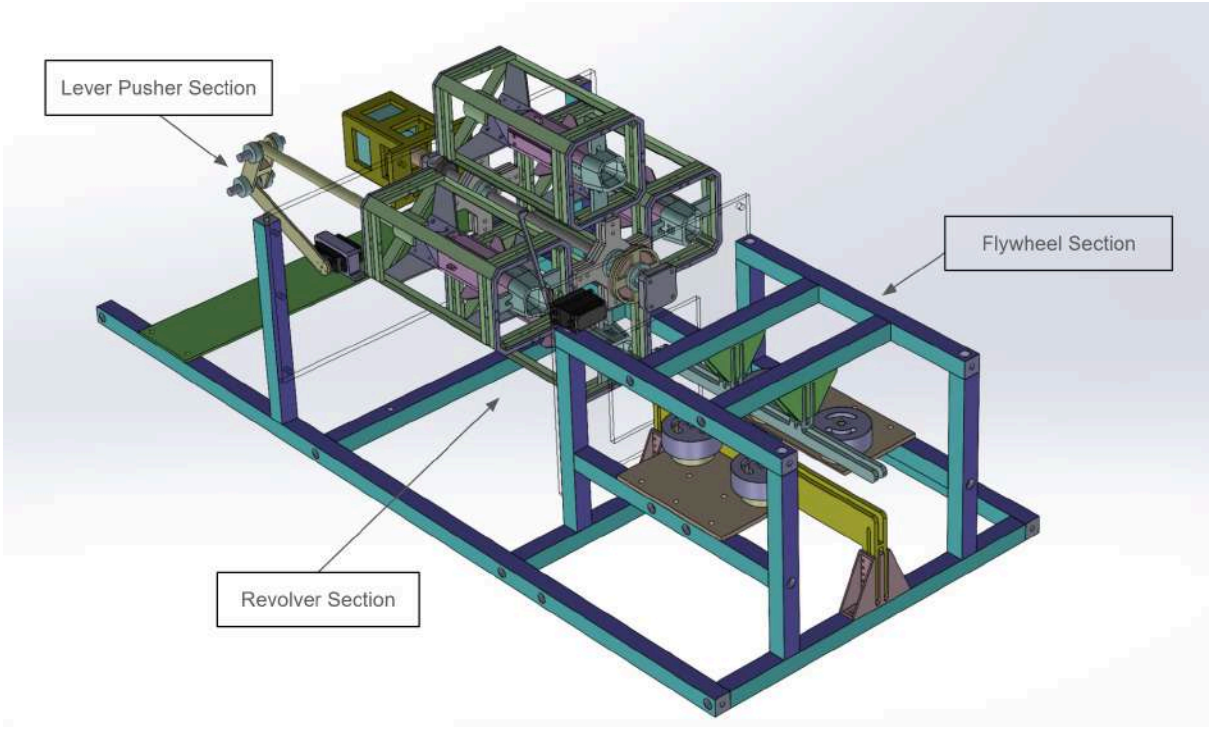


Figure 4.2.1a: Isometric View of CAD design

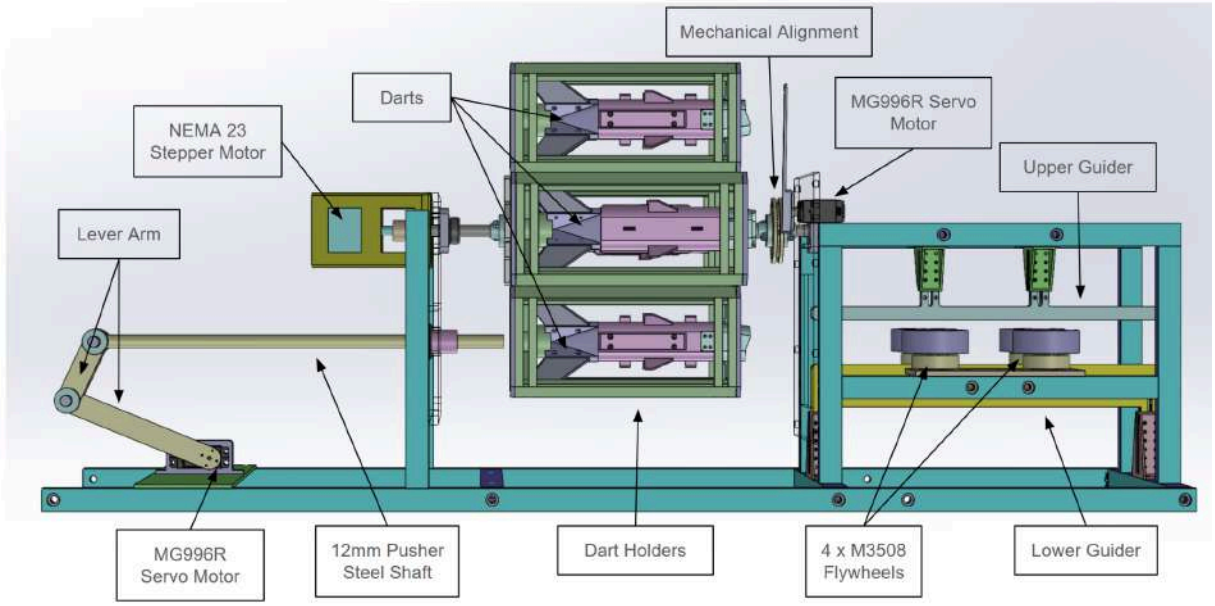


Figure 4.2.1b: Side View of CAD design

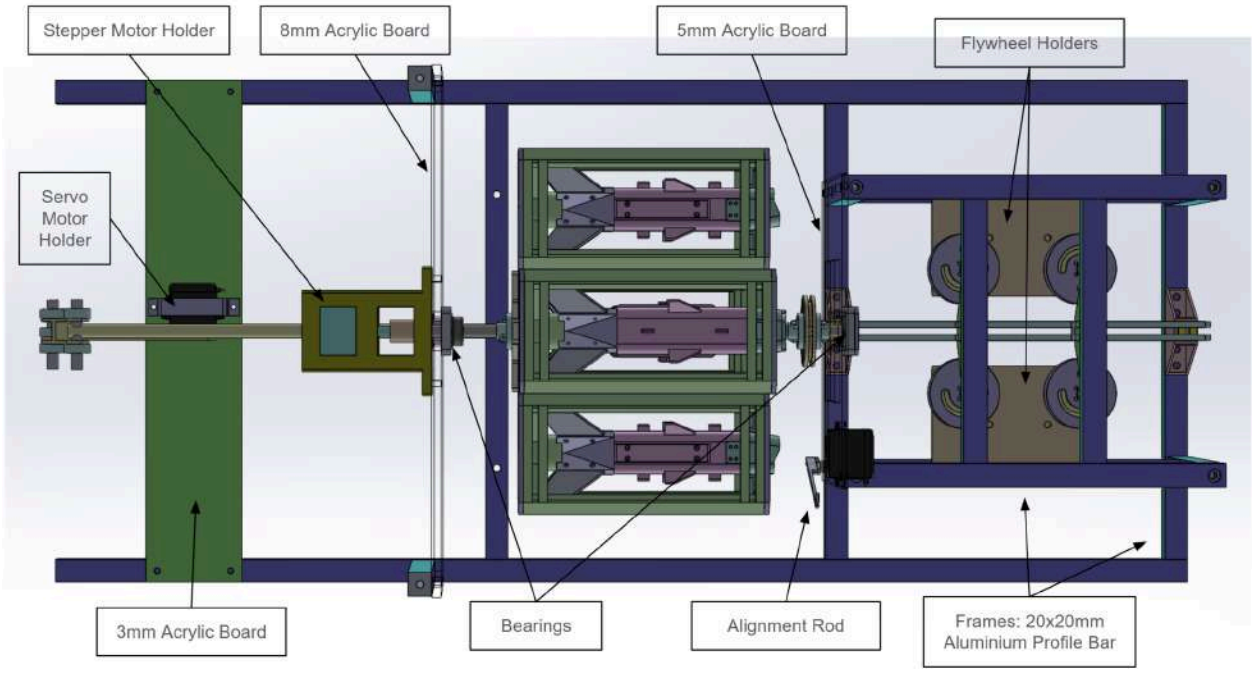


Figure 4.2.1c: Plan View of CAD design

The screenshots provide a comprehensive overview of the CAD design for the dart launcher, which consists of three main sections: the flywheel assembly, the revolver holder, and the lever pusher. This current design does not incorporate a pitch and yaw mechanism. The overall dimensions of the launcher, excluding the pitch and yaw assembly, measure 1000mm x 440mm x 396mm (L x W x H).

The first prototype was constructed by individually manufacturing the components using a combination of methods, including sourcing off-the-shelf parts, 3D printing, acrylic laser cutting, and CNC machining. The assembled prototype was then tested for structural rigidity and precision.

The overall structure of the launcher is constructed using 20mm x 20mm aluminium profile bars of varying lengths for a robust and modular frame.

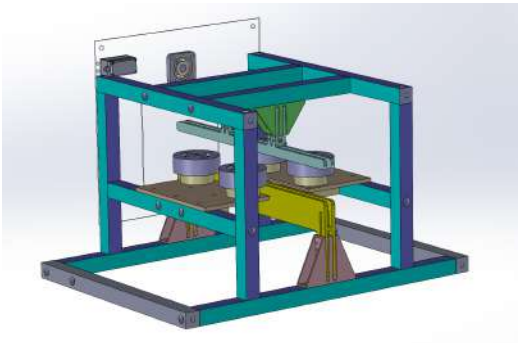


Figure 4.2.1d: Isometric view of Flywheels Section

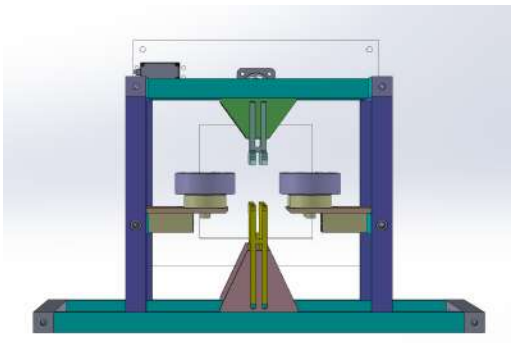


Figure 4.2.1e: Front view of Flywheels Section

The flywheel section consists of four flywheels driven by M3508 gear motors without gearboxes, supported by two aluminium cantilever plates to secure the flywheels in place. Additionally, two 3D-printed guide bars are integrated to ensure precise alignment and accurate dart launching.

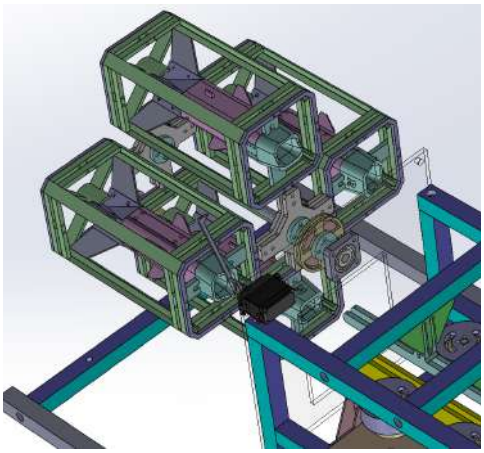


Figure 4.2.1f: Isometric view of Revolver Section

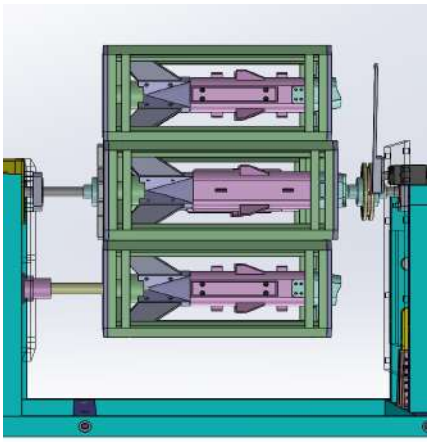


Figure 4.2.1g: Side view of Revolver Section

The revolver section is primarily composed of 3D-printed parts and laser-cut acrylic components. Darts are secured within dart holders mounted on a central rotation axis. With each 90-degree rotation of the revolver, a new dart aligns with the flywheel section for launching. Key mechanical components include bearings and an aluminium shaft, while the rotation and precise positioning of the revolver are achieved using a MG996R servo motor and a NEMA 23 stepper motor.

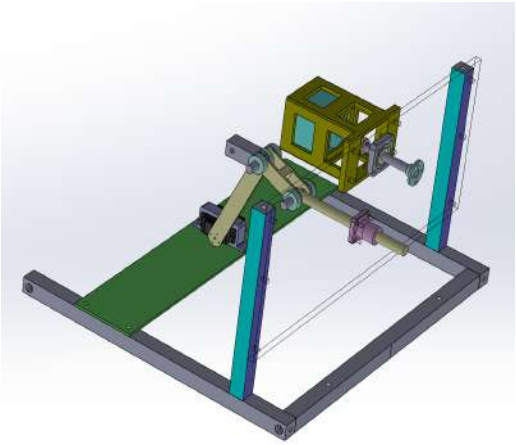


Figure 4.2.1h: Isometric view of Lever Pusher

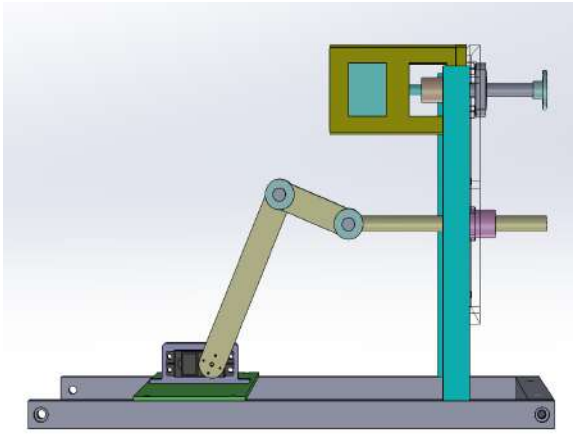


Figure 4.2.1i: Side view of Lever Pusher

The lever pusher mechanism consists of a 2-joint lever structure that pushes the dart into the flywheel section using a 380mm long aluminium shaft. The shaft's movement is restricted to a horizontal path by a linear bushing mounted on an acrylic plate, ensuring stable motion. The system's actuation is controlled by an MG996R servo motor for precise operation.

Images of the physical prototype have been included in the appendix for reference.

4.2.2 Dart

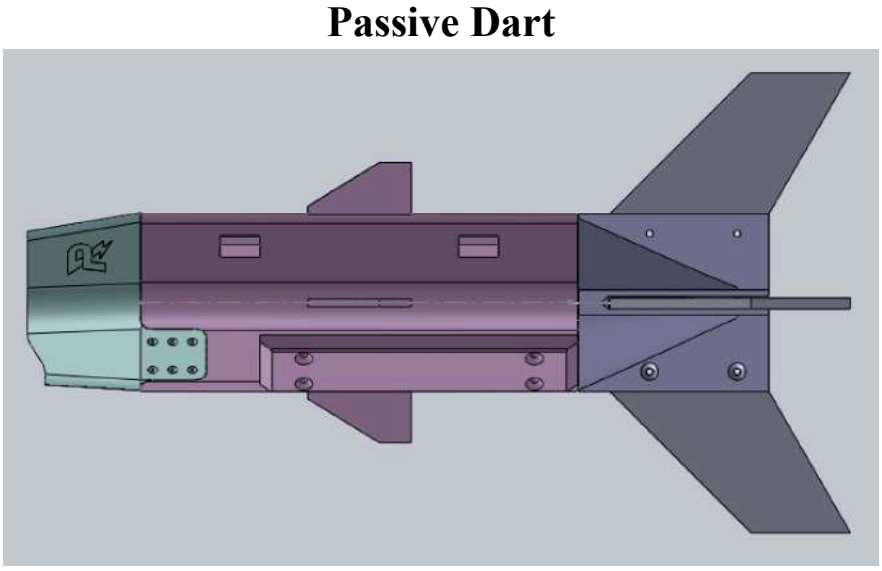


Figure 4.2.2a: Side view Passive Dart CAD model

The Passive Dart is made out of 12 main parts inclusive of the dart trigger (BOM to be included in appendix). During the Design process, there were a few main considerations

1. Attaching the replaceable tail piece to the main body
2. Attaching the side plates to the body
3. Easily replaceable fins

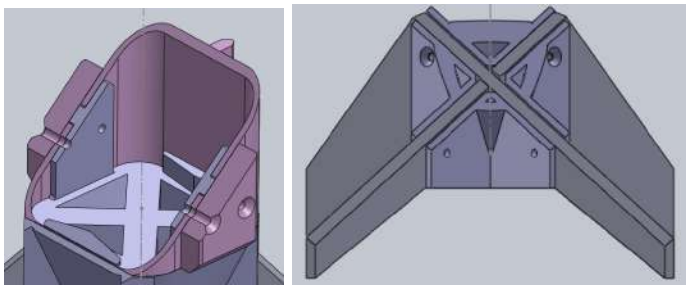


Figure 4.2.2b: Internal structure of removable pieces

To address the issue of securing these pieces together, we decided to use a clam shell design, with 3 plates screwed together. The purple tail piece serves both as a plate for the screws as well as geometrically constraining the movement of the upper and lower pieces of the body. M1.2 countersunk screws were used due to the geometrical limitation of the side plate being rather small, a larger size would require a wider side plate which would have to have a curved surface in contact with the Dart, which would likely result in higher manufacturing defects and inaccuracies when 3d printing. Another important thing to note is the use of countersunk screws on the side plates, this helped address the problem of the old design whereby the larger socket head screws sticking out of the dart created indentations along the flywheels and resulted in a much higher rate of wear and tear, likely leading to much higher inaccuracies.

As for attaching the fins, they are also made up of separate planar pieces, with slots cut out to ensure that they can slide into each other (do note that they are not tightly fitted). They are then held in place by similar m1.2 countersunk screws.

Active Dart

Following the original pedagogy, the active dart is a modified passive dart, with the interchangeable parts, the fins and the body upper being updated with internal supports to house the electronics.

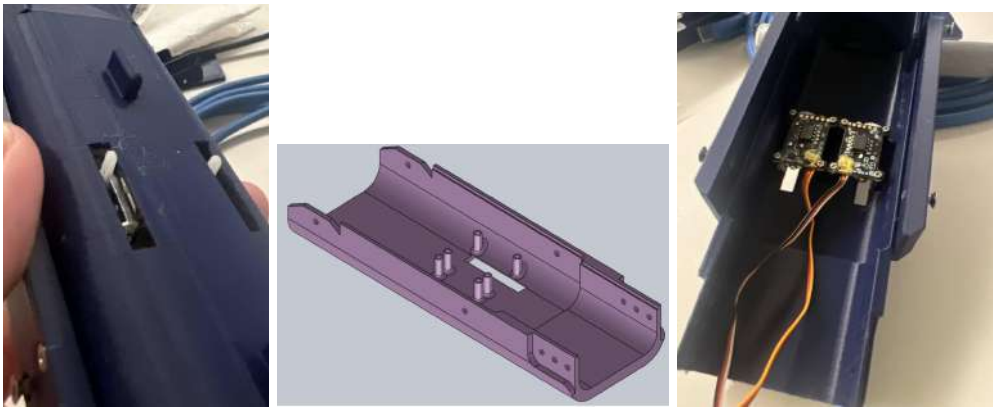


Figure 4.2.2c: Internal structure of servo support

Given the rounded internal geometry, we approached the selection of servos from mostly a size perspective. It is also important to note the left and right handedness of the servo, only then does it fit in the body. It is also notable that the M1.2 mounting holes are a potential cause of failure and are relatively weak.

The top of the servos horns are connected to servo rods which are directly connected to the active surfaces on the fin, which are on a hinge joint with the main fin surface.

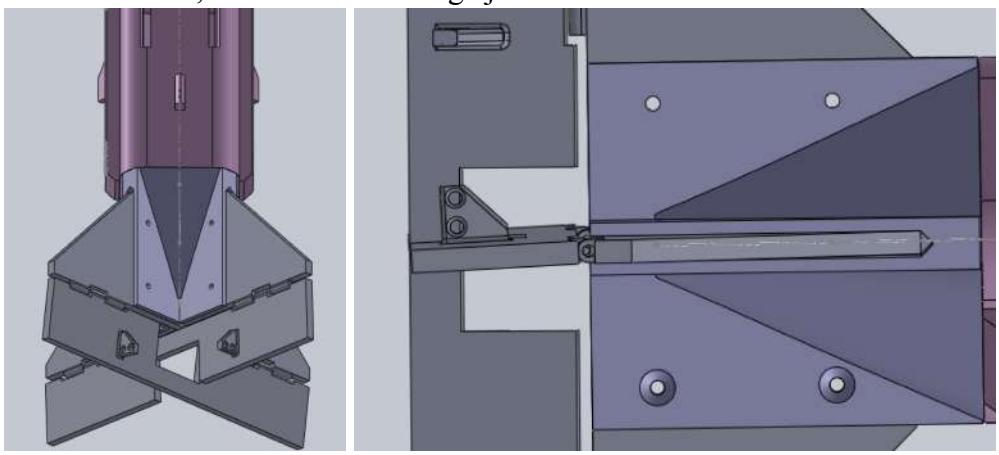


Figure 4.2.2d: Active surfaces and hinge joint

The hinge joint is secured using a section of M1.2 servo rod. The degree of freedom of the active surface is largely influenced by the height of the hinge structure. There is obstruction when the horn goes under the plane of the fin.

Manufacturing Details

3d printing for the dart body can be specifically difficult given the odd geometry. This is specifically relevant for the upper part of the body and the internal standoffs. The automatically

generated supports are often not ideal.

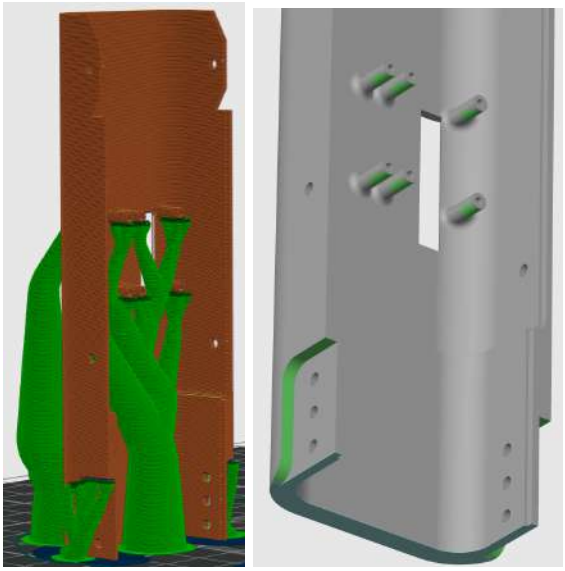


Figure 4.2.2e: Support painting for body upper

3D print settings:

- Support settings:
  All supports were painted on for the body, with only the upper part of the active dart requiring support.
- Seam settings:
  Seams are painted on to ensure consistency in the layers, when random seems were used, the dimensional accuracy decreased
- PLA Aero:
  When printing PLA Aero, we used the setting suggested by
  Bambu Lab. (n.d.). *Studio settings for RC models*. Bambu Lab Wiki. Retrieved November 14, 2024, specifically, retraction was tuned to reduce the amount of stringing while 3d printing. It was also noted that prints are generally much better performing when the filament is dried beforehand due to the hygroscopic nature of PLA Aero

Using of Servo rods:

4.2.3 Electrical System

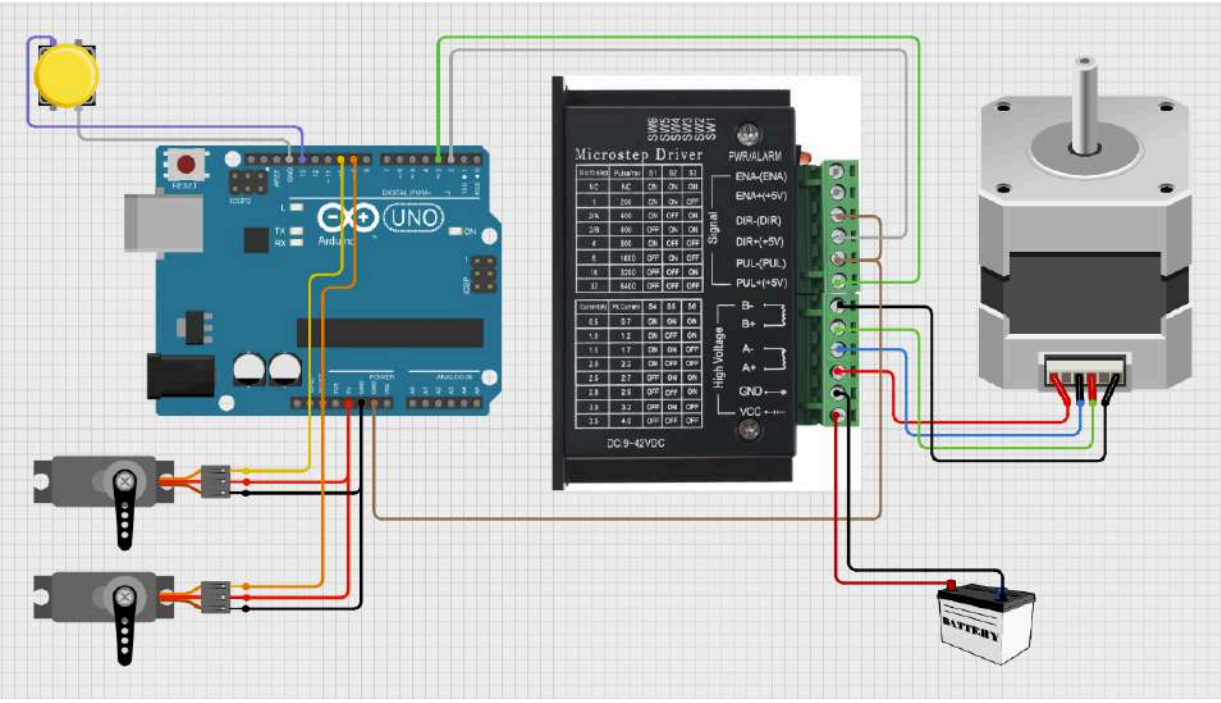


Figure 4.2.3e: Arduino circuit diagram - NEMA+MG996R+PB

The team presents the above electrical system integrating the NEMA 23 stepper motor with a TB6600 driver and powered by an RM battery, as well as two MG996R servo motors. A push button has been added for this sub-system testing for the ease of triggering a new cycle of rotations.

Successful results of the above sub-system testing can be viewed here: https://www.youtube.com/watch?v=qLhBmzX3Jrg.

The DC gear motor + CAN code, vibration sensor code and flywheel control code can be found in the appendix.

4.2.4 Computer Vision

Software

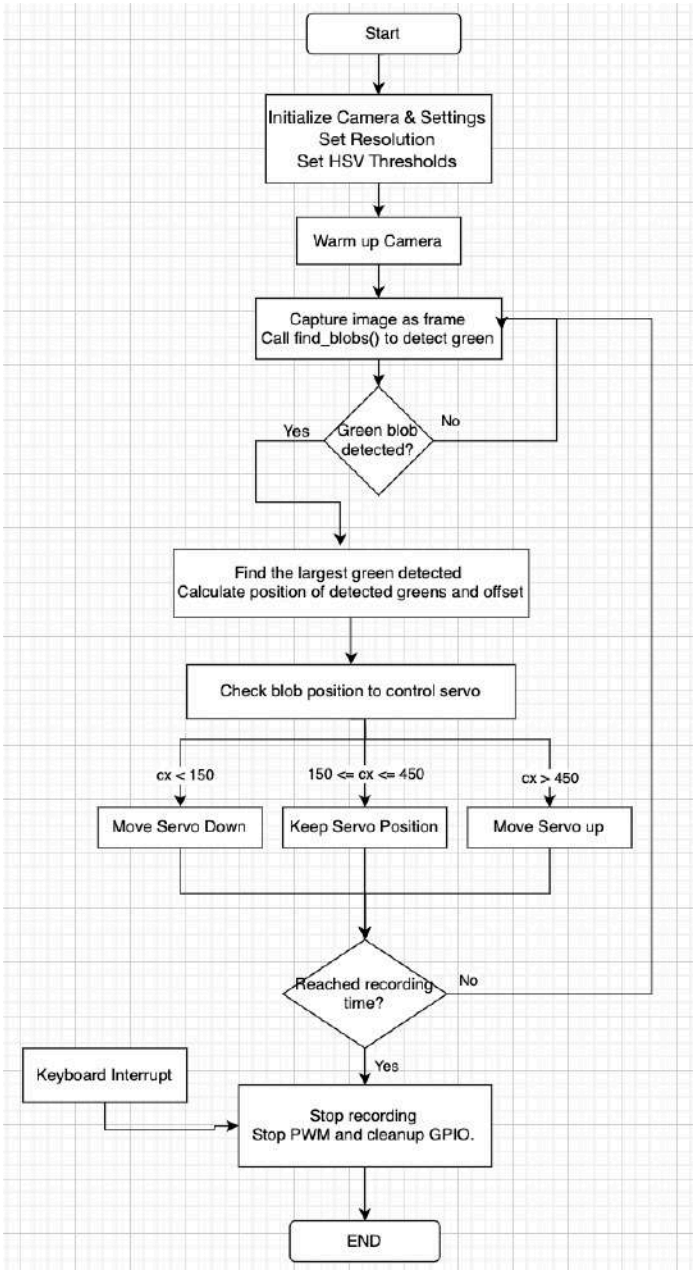


Figure 4.2.4a: flowchart of tracking algorithm

The prototype vision system implements a vision-guided control system, integrating a micro servo motor to demonstrate adaptive motion adjustment based on visual feedback. Figure 4.3X presents the vision system's operational flowchart, illustrating the following key processes:

- Initialization
    - Configuration of the camera module at 640x480 resolution
    - Establishment of RGB threshold parameters for green detection:
      LOWER_THRESHOLD = np.array([0, 100, 0])

      UPPER_THRESHOLD = np.array([100, 255, 100])
- Processing Sequence
    - Image processing via the find_blobs() algorithm to detect green blobs
    - Upon successful green blob detection, the system performs:

      a) Identification of the largest detected region

      b) Position calculation relative to the frame coordinates
    - Servo actuation is determined by the blob's centre x-coordinate (cx):
        - Downward movement when $cx < 150$
        - Position maintenance within $150 \leq cx \leq 450$
        - Upward movement when $cx > 450$
- System Control Architecture
    - The operational duration is determined by the pre-established recording parameters
    - The system is equipped with manually keyboard interrupt

In the find_blob function, thresholding and morphological operation are used. Details are shown in the flowchart below:

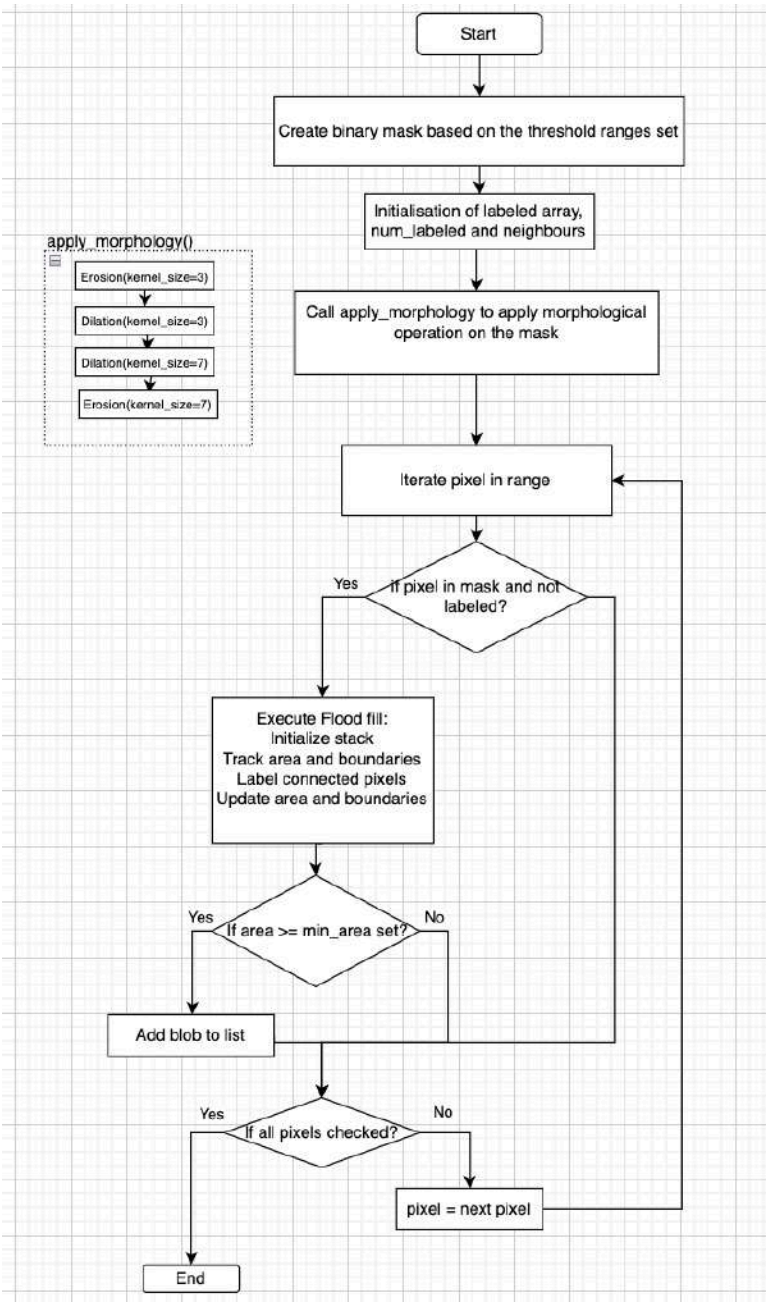


Figure 4.2.4b: flowchart of colour detecting algorithm

The initial algorithm design incorporated both thresholding and morphological operations, as illustrated in the flowchart above. However, the results of our testing revealed unacceptable latency in the system's response with morphological operations. Consequently, the implemented green detecting algorithm removes morphological operation to ensure a short processing time. Details of code can be found in appendix **X**.

Electronics

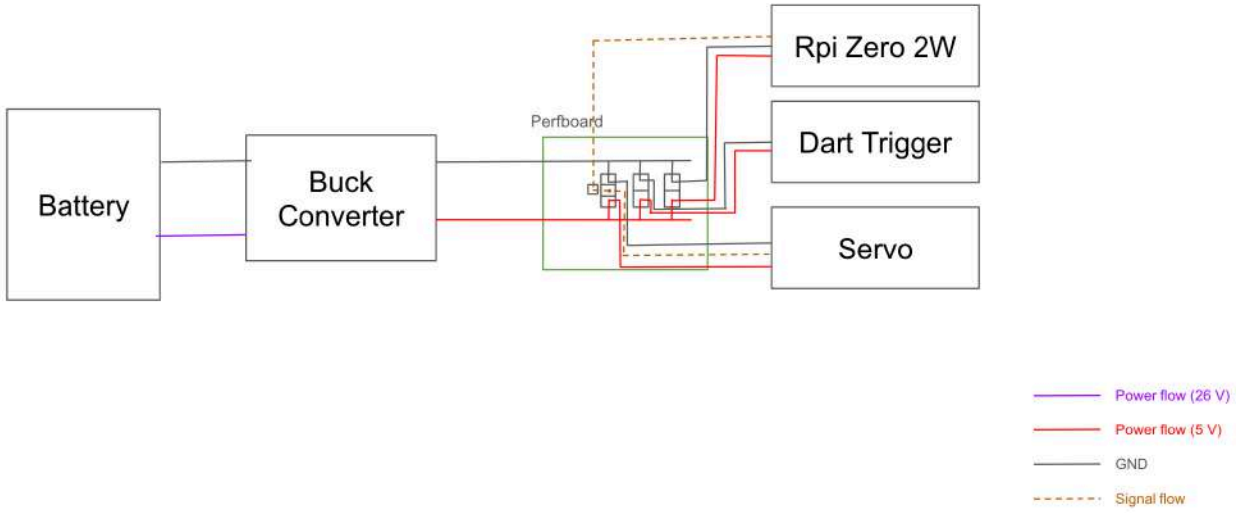


Figure 4.2.4c: Electronic Architect of Vision System

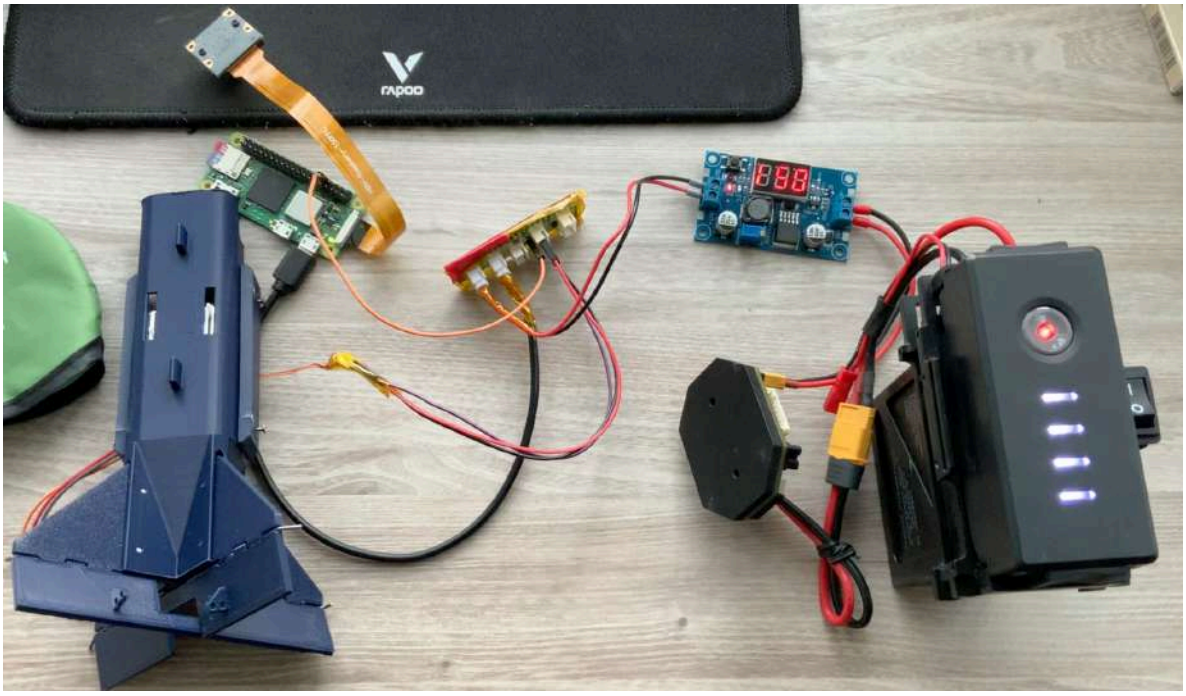


Figure 4.2.4d: Physical Setup of Vision System

As shown in figure 4.3X and figure 4.3X, the vision system employs a distributed power architecture with centralised control. The key components are:

- Power Management:
    - Battery serves as the primary power source
    - Buck converter steps down 26V to 5V for system components

    - Perfboard distributes power efficiently while maintaining signal integrity

- Control Integration:
    - Raspberry Pi Zero 2W functions as the central controller
    - IMX219 camera connects via CSI
    - D1015PRO linear servo receives control signals via GPIO 12 on Raspberry Pi

This architecture ensures reliable operation while maintaining system responsiveness for real-time tracking applications. The perfboard implementation provides a robust platform for prototyping and the use of JST connectors can help maintain connection stability.

Chapter 5 | Testing and Validation

5.1 Description of Testing Methodologies

We have developed a comprehensive testing plan for the dart system, addressing key aspects including the mechanical design of the dart launcher, the electrical and software systems, the dart's robustness and stability, the computer vision system, and, most importantly, the overall accuracy of the dart-launching system. This section outlines our testing methodologies for each subsystem.

5.1.1 Dart Launcher

For the dart launcher, three primary criteria must be met and rigorously tested:

1. The dart should launch in a straight path upon contact with the flywheels.
2. The revolving mechanism should reliably position each dart into the flywheel section.
3. The lever pusher must function smoothly and accurately to ensure consistent dart placement.

The team will employ a mix of observation and data collection through dart launches. Below is the approach for testing each criterion:

1. **Flywheel Launches**: The team will set up an overhead camera above the flywheel section to capture slow-motion video of dart launches. Video playback will allow us to review frame-by-frame screenshots to assess if the dart launch is straight. Additionally, this footage will reveal whether the interference between the dart's body and flywheel provides enough friction for consistent launches reaching a target distance of at least 15 metres.

2. **Revolving Mechanism Accuracy**: The team will incorporate a stepper motor to rotate the revolver 90 degrees each time a button is pressed. To verify rotation accuracy, the team will use visual inspection alongside a spirit level to check if the top dart holder returns to a horizontal alignment after each turn. The team will also ensure that the dart aligns correctly with the flywheel section after each rotation.

3. **Lever Pusher Efficiency**: The team will test the lever pusher mechanism by cycling it back and forth with an attached servo motor, checking for smooth operation and minimal resistance in the rotation joints. The team will also confirm that the pusher shaft is of sufficient length to fully deliver the dart into the flywheel section, establishing contact with the flywheels. Lastly, the team will ensure the shaft fully retracts without obstructing the dart holders for subsequent rotations.

5.1.2 Dart

For the Dart,

1. Common points of failure
2. Consistency of flight and its ability to "correct" for small perturbations from launch or in flight

3. Mass distribution of the Dart and its effect on Launch Distance

5.1.3 Electrical System

Project progress in the electrical aspect is tested with reference to 3 main criteria:

1. **Vibration:** Vibration sensor values should show the difference in values between using the old pre-UERx launcher vs the new UREx launcher during launching. The sensor is mounted on the main launcher body frame, in close proximity to the flywheels.

   A properly integrated vibration sensor detects the variation in movement. It should be able to verify the previous hypothesis that the old launcher could not achieve accuracy due to high vibrations during launch.

2. **Stepper motor:** We aim to achieve precise controls in the form of angular position for the stepper motor controlling the revolver mechanism.

   Positional control for the stepper motor is deemed successfully accurate if the motor spins exactly 90 degrees and the dart rails on the revolver mechanism align with the dart rails on the launching platform.

3. **Servo motors:** We aim to achieve timely and effective actuation of servo motors (one to lock the revolver frame and another to push a single dart out of the revolving frame to feed into the flywheels for launch).

   Servo control is deemed successful if the revolving mechanism cycles smoothly according to the intended sequence of events without jamming itself. In addition, the servo motor has to successfully feed a dart far enough on the rails of the launching platform such that the dart flies out.

5.1.4 Computer Vision

For computer vision, the effectiveness, efficiency and reliability should be tested:

Aspect	Criteria of Success	Test Method
Effectiveness	The system must achieve the following functions: - Identify green targets - Execute servo control based on target positioning	Position green target at predetermined coordinates, test if the system can react as expected.
Efficiency	The system should respond in a sensitive manner: the tracking algorithm should be capable of detecting the green once it enters the	Record the response time and do evaluation.

	camera frame, and the servo motor must then react accordingly.	
Reliability	System should maintain consistent performance during the whole operation.	Test continuously to analyse the consistency.

Table 5.1.4a: Computer Vision Testing

5.1.5 Launching Accuracy

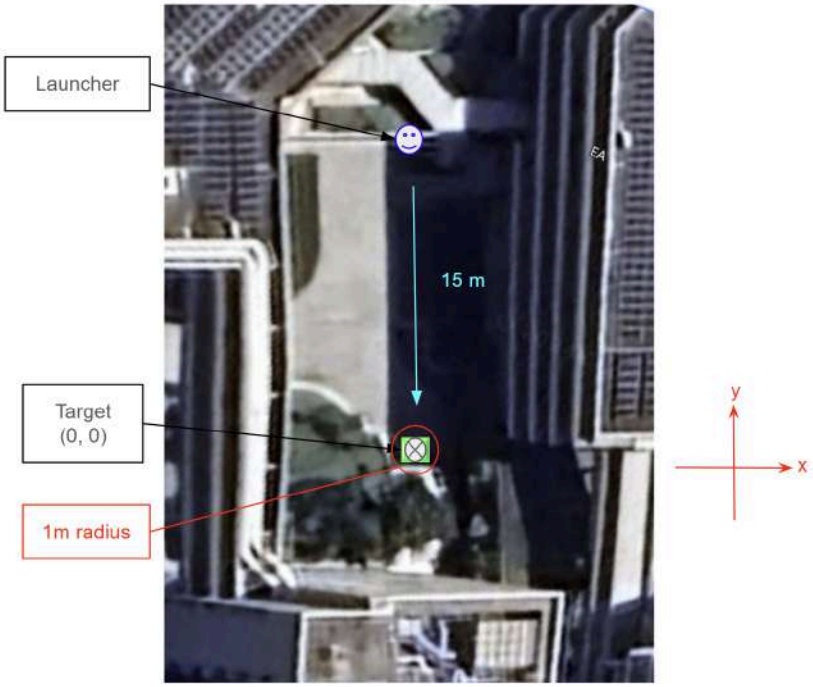


Figure 5.1.5a: Launch Test Map (Engineering Auditorium)

To test the launch accuracy, the team conducted a series of launch tests in the open area outside the Engineering Auditorium (EA) on level 3. A large white paper, composed of six A4 sheets (approximate dimensions: 0.63m x 0.58m), are placed on the ground with a cross marked at the centre as the origin. To record each landing point, the dart heads are coloured with crayons so that contact with the paper leaves a visible mark.

The open test environment featured variable conditions, including occasional light drafts due to wind flow, which will be absent in the controlled, draft-free RMUC arena. This outdoor setup introduces potential deviations in trajectory caused by environmental factors, such as uneven wind resistance or air turbulence, that would not affect the darts in the competition environment. While this testing environment differs from the RMUC arena, it provides a preliminary stress test for the system's robustness in less predictable conditions. The large open space also allows for testing the darts' stability over the entire intended range, something critical to evaluate before transitioning to competition-level controlled trials.

The team performed multiple launches with darts of identical configurations, repeating the process for several different darts. For each launch, the team measured and recorded the distance from the origin to the landing point. Calculating the average deviation of all launches allows the team to assess the overall accuracy of the system and compare it with data from the previous dart launcher. This data also gives an estimate of the improvements needed to achieve consistent accuracy in hitting the target tower within the RMUC arena.

5.2 Final Launch Results

The following data was collected from a series of launch tests. For all tests, the flywheels were set to an angular speed of 10,000 rpm, and the launcher was angled at 45 degrees. While not all darts hit the target paper directly, the team tracked each dart's landing point using visible marks left on the ground and measured the distances (in metres) from the origin.

Launch No.	Dart Type	Landing Point, (x, y), in metres	Absolute Distance to Target, in metres
1	Full PLA	(0.67, -0.81)	1.05
2	Full PLA	(-0.33, 0.52)	0.62
3	Full PLA	(-0.32, 1.45)	1.48
4	PLA body + PLA Aero fins	(1.22, -1.80)	2.17
5	PLA body + PLA Aero fins	(-0.24, -1.34)	1.36
6	PLA body + PLA Aero fins	(0.49, -1.03)	1.14
7	Full PLA Aero	(-1.02, 3.36)	3.51
8	Full PLA Aero	(0.65, 2.23)	2.32
9	Full PLA Aero	(0.23, 2.75)	2.76

Table 5.2a: Test Results

*Data collected are subjected to ~2% errors

Please refer to the videos in this link:

PLA body + PLA Aero fins: https://youtube.com/shorts/bvZN9Jkd8aI?si=PNlMrtP9aqnK6-N0
Full PLA: https://youtube.com/shorts/37MI46xF4Rw?si=mi15Q_Ab14RbUDqv
Full PLA 2: https://youtube.com/shorts/SThk3FEggHc?si=nHwTnP7lkRfCTRPg
Full PLA Aero: https://youtube.com/shorts/2nYlhDYRY6s?si=PAv1aS1_gkjait2S
PLA body + PLA Aero fins: (Different view):
https://youtu.be/vplNyxLJ6pc?si=eddIMXW6KwLa-VEf

Chapter 6 | Results and Discussion

6.1 Analysis of Test Data

This section will address the testing criteria outlined in Section 5.1. Using the data collected in Section 5.2, we will analyse observed trends and provide validation, discussion, and justification for the testing conducted.

6.1.1 Dart Launcher

1. **Flywheels Launches:** The data collected in Section 5.2 shows that the deviation in the x-direction is significantly smaller than that in the y-direction. The average absolute deviation in the x-direction is ±0.57m, while in the y-direction, it is ±1.70m. This indicates that the launches were relatively straight and consistent along the x-axis but less consistent along the y-axis. The average distance to the target is 1.82 metres, with the closest shot at 0.62 metres and the farthest at 3.51 metres, indicating significant room for improvement. Further refinement of the launching mechanism is necessary to achieve better consistency and accuracy. This enhancement will be crucial for ensuring the system's reliability and performance in the upcoming RMUC.

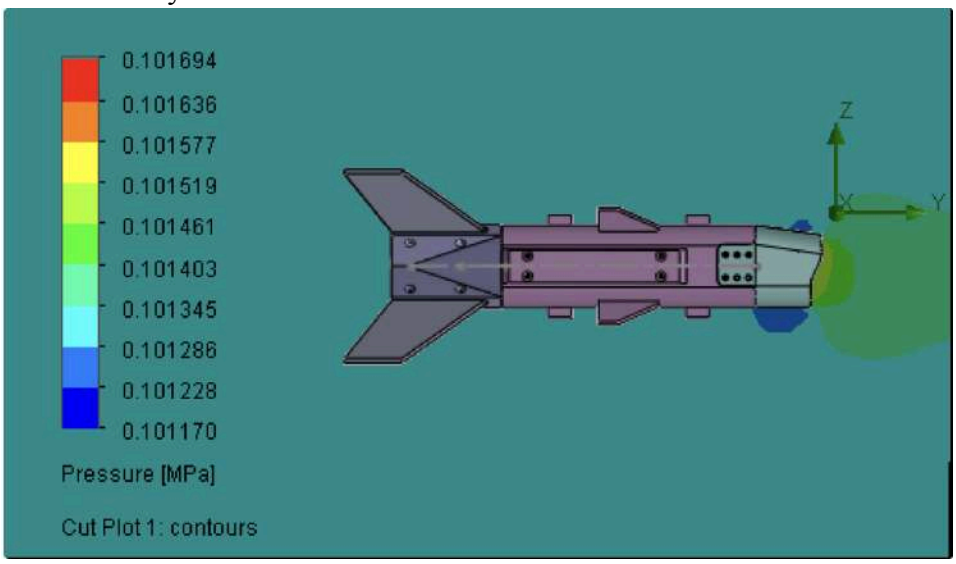
2. **Revolver Mechanism Accuracy:** Upon integrating the first prototype of the revolver feeding mechanism with a NEMA-23 stepper motor, the team programmed the motor to rotate 90 degrees per button press. Initial tests of the motor when it is not connected to the mechanical system showed accurate 90-degree rotations with minimal deviation. However, once connected to the revolver system, each rotation deviated by up to ±30 degrees. The team suspects that the added mass of the revolver system, combined with friction between mechanical components such as bearings, shafts, and the motor coupler, may contribute to the observed discrepancies.

3. **Lever Pusher Efficiency**: The team tested the lever pusher mechanism by activating the MG996R servo motor connected to the lever pusher shaft. However, due to poor design of the joints between the shaft and servo motor arm, the lever pusher failed to function properly and often became stuck when attempting to push forward. Additionally, the shaft could not fully retract from the dart holder, likely due to the shaft's length and joint configuration. The angled dart launcher and gravitational force further hindered the system, and the lack of bearings at the joints contributed to excessive rotational friction.

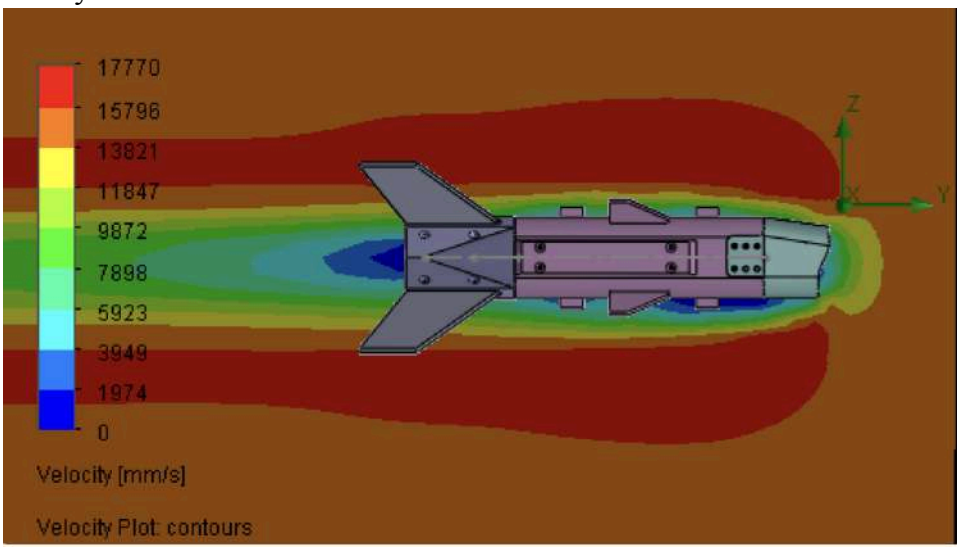
6.1.2 Dart

CFD Simulations of Dart:

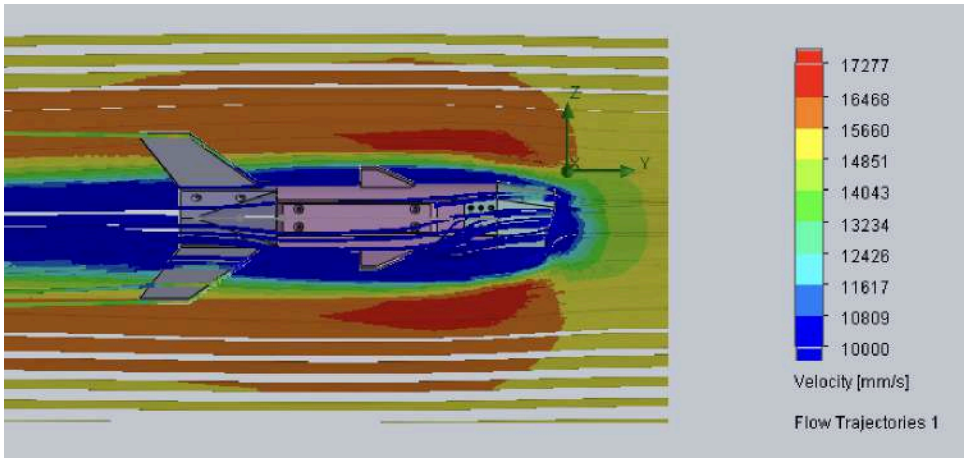
Given that the dart was Designed to have no camber and have all aerodynamic surfaces to have no lift, we expect there to be little variance in pressure across the top and bottom surfaces of the dart. This is reflected in the CFD performed in SolidWorks, the simulations were performed at an estimated launch Velocity of 16.5ms^ -1.



There is a slight difference in pressure at the top and bottom of the dart due to the asymmetrical shape of the Dart Head. This is likely from the more aerodynamically shaped top, resulting in a higher air velocity.



Velocity along the centre plane of the Dart



Velocity across all surfaces of the Dart

This also suggests there is a very slight lift being generated, Below are the simulated values for Lift and Drag.

Goal Name	Unit	Value	Averaged Value	Minimum Value	Maximum Value	Progress [%]	Use In Convergence	Delta	Criteria
GG Normal Force 1	[N]	0.184	0.184	0.184	0.184	100	Yes	3.450e-04	0.018
GG Normal Force (Z) 2	[N]	0.003	0.003	0.002	0.004	100	Yes	0.001	0.001
GG Normal Force (Y) 1	[N]	-0.184	-0.184	-0.184	-0.184	100	Yes	3.468e-04	0.018

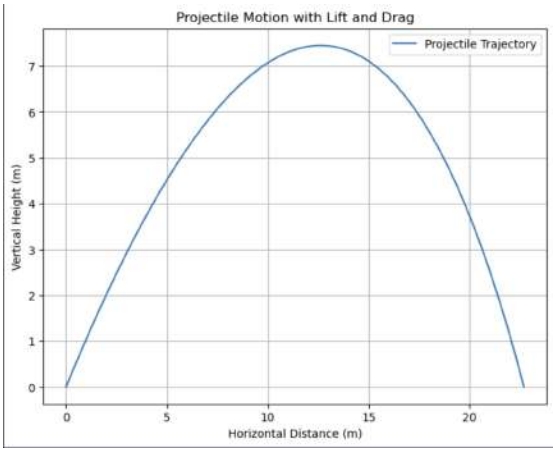
Launch Simulations:

Based on the Lift and Drag Values, the launch angle of 47° (as per experiment), and mass of 110g (full PLA dart) a simple model based on Parabolic motion affected by lift and drag was developed (Python code in appendix). Note that the coefficient of lift and drag are non consistent over different velocities, this is assumed to be constant in this simulation and a source of error.

$$\frac{m \cdot v_0 \cdot (-2.0 \cdot D \cdot \sin(\theta) + 2 \cdot (-L + g \cdot m) \cdot \cos(\theta)) \cdot \sin(\theta)}{(L - g \cdot m)^2}$$

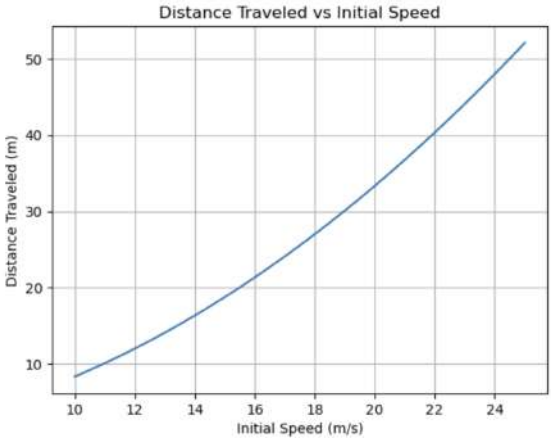
Symbolically solved expression for Distance travelled based off the parameters

Simulation based on launch speed of 16.5 m^-2 the expected distance travelled was calculated.



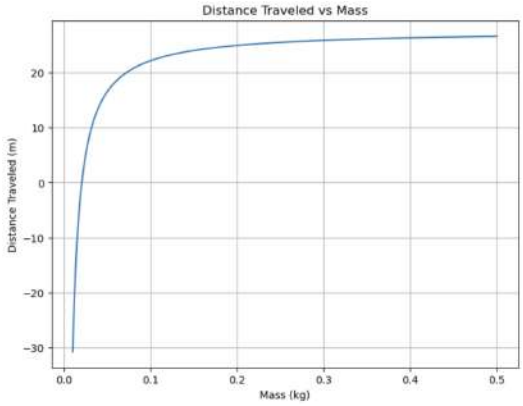
Launch trajectory given 16.5 m^-1 launch, final distance travelled is 22.69m.

The effect of launch speed on distance can also be calculated.



Effect of launch speed on distance

The effect of increasing mass on distance was also simulated. It is observed that the optimal weight is above 0.9kg, it is likely due to the dart with greater mass being less affected by drag due to it carrying more momentum.



Effect of mass on distance

Although not exactly accurate due to variation in launch velocity and assumptions made in the model, these simulations provide a decent approximation to real world results.

6.1.3 Electrical System

Motor Controls

The integration of the electrical subsystem (stepper and servo motor control) with the mechanical system (physical launcher) have been described in 6.1.1 Dart Launcher section.

Watch a slow-movement recording of the integrated system testing here:
Integrated electrical sub-system with mechanical sub-system - backsliding observed

To reiterate with a focus on the electrical controls, the integration of a motor controls subsystem which works independently could not be successfully integrated with the revolver mechanism due to the added weight of the mechanical structure. The team suspects that the NEMA 23 stepper motor does not have enough torque to rotate the shaft to the agility as during sub-system testing. It is also open-loop and not compatible with encoders, and thus does not have positional feedback to realise that it has skipped steps, so those steps are therefore not accounted for and compensated back.

Vibration Monitoring

In terms of quantifying the amount of vibrations during launch - for the old dart launcher and the new dart launcher the team has built - the 801S vibration sensor was adhered to each of the launcher frames with epoxy and integrated with Arduino UNO. The serial plots below show the results:

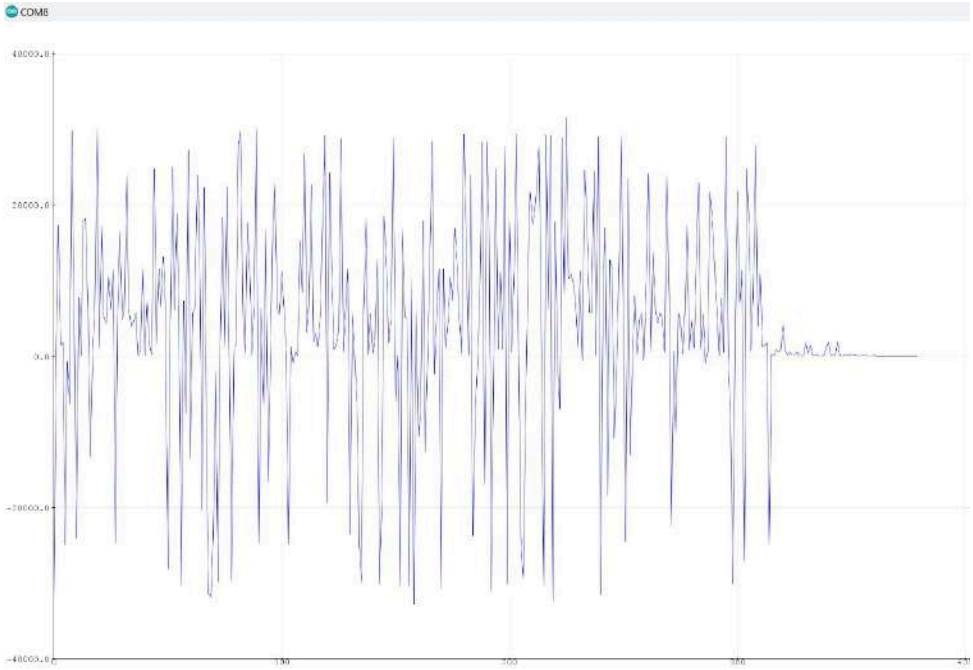


Table 6.1.3a: Vibration Data for the Old Launcher

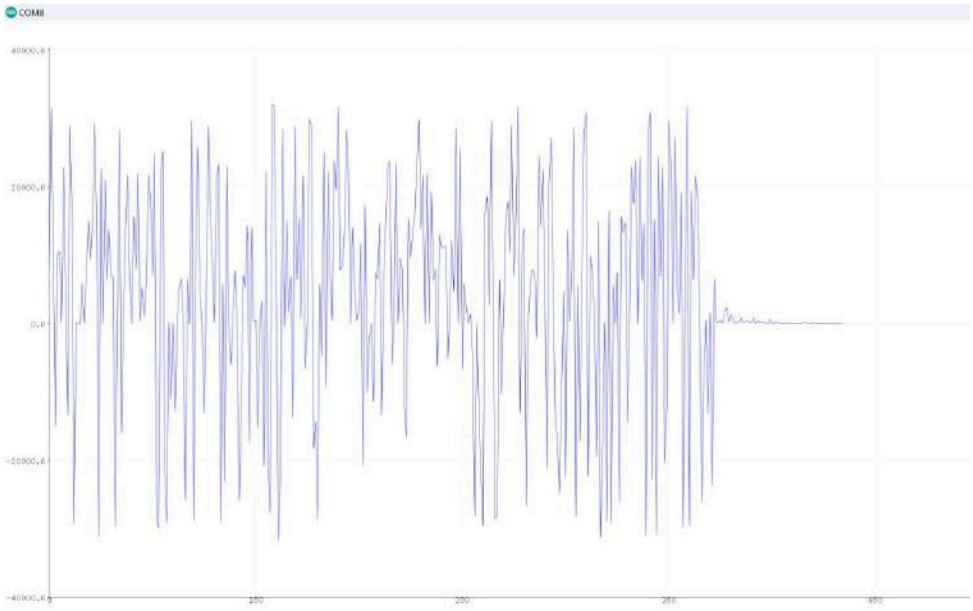


Table 6.1.2b: Vibration Data for the New Launcher

Comparing the results from the vibration sensor tests, the peak vibration values and graph shapes for both the old and new dart launchers are comparable, suggesting that the old launcher likely did not suffer from significant vibration-related issues. The results challenge the initial hypothesis that vibrations were a primary cause of the poor performance observed in the old launcher.

However, these results do not conclusively indicate that the new launching system is not superior. The inconsistencies in the old launching system may stem from other factors, such as the quality of dart design. Therefore, while vibration may not have been a significant issue, the new launching system design should still be further evaluated for overall performance improvements. Additional testing and diagnostics are recommended to identify and address other potential sources of instability to ensure consistent and reliable operation.

6.1.4 Computer Vision

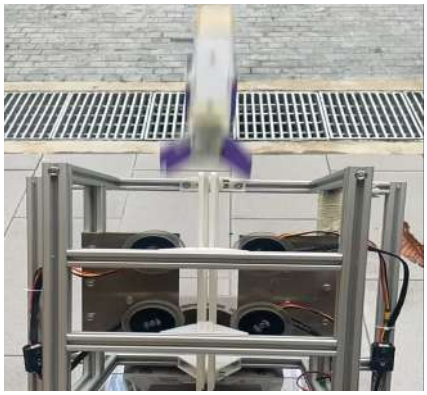
The vision system was tested separately in a static approach rather than launching out due to the constraints of the size of the dart. Test results are summarised in the table below:

Aspect	Test Result	Analysis
Effectiveness	- Servo moved upwards when the target is at the right; - Servo moved downwards when the target is at the left; - Servo maintained position when the target is at the centre.	The system successfully demonstrated all required positioning responses with high accuracy
Efficiency	- In all testing, the camera could react very fast to the change of target. - Servo responded promptly but exhibited cumulative delay over extended operation	While initial response times met requirements, the cumulative servo delay requires attention for extended operations
Reliability	Performance maintained consistent performance across 10+ test iterations, each test ran for 90 seconds without degradation	Long-term stability validated under controlled conditions

Table 6.1.4a: Vision System Testing Results

The vision system testing demonstrated consistent green target identification and accurate servo motor control based on target positioning. However, the servo's response delay may impact system efficiency in dynamic situations. While static testing validated the core functionality, real competition scenarios with moving darts may present additional challenges for vision accuracy and tracking performance that weren't evaluated in these controlled tests.

6.2 Potential Improvements Based on Test Results

Section	Potential Improvements
Dart Launcher	1. The significant deviation in both x- and y-axis distances may result from uncalibrated flywheel motors, which appear to operate at slightly different angular speeds. This discrepancy impacts the distance consistency (y-axis) more than the straightness (x-axis) of the trajectory. The data suggests that our dart system maintains better control over trajectory alignment than flight distance. From slow-motion overhead video screenshots, the dart is observed to travel straight upon exiting the launcher.  ***Figure 6.2a: Screenshots of Dart Position at Launching*** To address this, the dart team could implement PID control to regulate and calibrate the flywheel rotation speeds, ensuring uniform speed across all four flywheels. Additionally, testing various angular speeds may help identify an optimal speed for achieving consistent, accurate hits at a 15m range. 2. To enhance the revolver feeding mechanism, the team could position a servo motor beneath the dart holder. After each rotation, the servo would activate to stop the dart holder at the 90-degree mark. This provides a more reliable mechanical alignment method compared to relying solely on the stepper motor's accuracy. Alternatively, implementing PID control on the stepper motor, with positional feedback, could improve the accuracy and ensure consistent performance even with the added mass of the revolver system. 3. To improve the current lever pusher mechanism, a linear actuator (as shown in Figure 6.2b) could replace the existing complex pusher system to simplify the design. The next iteration would focus on designing mounts for the linear actuator and selecting an appropriate off-the-shelf actuator to serve as the pusher shaft. This solution would streamline the mechanical system, ensuring better

	efficiency and reducing the likelihood of failure.  ***Figure 6.2b: Example of a linear actuator***
Dart	1. The Tail piece of the Dart is susceptible to breaking, especially when the tail of the dart collides with the ground, the mechanism for attaching the 2 parts can be modified to provide greater strength in the vertical direction so that the part is not sheared off. 2. The possible angles for deflection of the aerodynamic surfaces on the active dart have to be iteratively experimented on together with the control policy. 3. Side structure of both the active and passive dart can also be tested to see how it influences the speed of the launch. This current round of tests uses 4mm height. Different interference fits are likely to deform the flywheels differently resulting in variance in speeds. 4. The Active Dart requires a custom PCB in order to fit the buck boost, Rpi as well as the camera. Further testing is required on the weight distribution for such a set up.
Electrical System	1. The NEMA 23 stepper motor does not have enough torque to rotate the shaft to required precision. This is because it is open-loop and will not realise when it skips steps, and so those steps cannot be compensated for. To counter this issue, it is possible to: a. Find a stepper motor with even **higher torque** and add an encoder with closed-loop feedback signals, tracking the motor shaft's position and speed. b. Add a stopping mechanism. (Optional: switch to a DC gear motor and) Integrate a pair of **opto sensors** to indicate the angular position of the revolver and stop the gear motor when it triggers the 2nd sensor at the position suitable for

	dart loading. Alternatively, a servo motor can be added to introduce a **mechanical barrier** to force the revolver frame to stop rotating when it reaches a certain precise limit. This forced stop happens mechanically while the stepper or gear motor is still running. It releases momentarily after the dart has been launched. 2. Electrical subsystems may have to be converted to be hosted by Dev-C to be used in the competition. We will need to use wireless control on the system to control its activity during the match. While the remote control is an existing skill set of many competition team members, it is still work to be done.
Computer Vision	1. The test results revealed a noticeable **cumulative delay in servo response**. This delay affects the system's ability to track targets smoothly and could be more serious during actual dart launches. To address this, implementing a more efficient servo control algorithm could help reduce response time. Testing different PWM frequencies and optimising the control loop timing could minimise the cumulative delay effect. In addition, higher-performance with shorter response time could be considered. 2. Current testing was **limited to static conditions**, which may not accurately represent the system's performance during actual dart launches where rapid target tracking is crucial. The vision system's ability to maintain accuracy during high-speed scenarios remains unverified. To improve this limitation, a dynamic testing setup should be developed and further improvement on algorithms could be considered based on the testing results.

Table 6.2a: Potential Improvements

Chapter 7 | Budget Analysis

7.1 Bill of Materials

Type	Expense / SGD$
Electronics	719.47
Mechanical Parts	145.36
Manufacturing	487.90
Testing Materials	88.58
Total	**1531.31**

Table 7.1a: Summarised BOM

Electronics costs encompass all components and hardware associated with the dart launcher and the dart itself, including boards, cameras, actuators, batteries, and cables. **Mechanical parts** purchased include bearings, couplers, shafts, clamps, and fasteners for the dart launcher, as well as weights and nuts for the dart. **Manufacturing** costs covered fabrication services from the NUS Professional Workshop, based on provided drawings, and specialty 3D-printing services from the Innovation & Design Hub (EDIC). **Testing materials** primarily included Aero PLA and standard PLA spools used for 3D-printing dart bodies. A detailed Bill of Materials (BOM) is attached in the appendix.

Chapter 8 | Conclusion

8.1 Summary of Key Findings

The first prototype of the dart launcher (excluding the base, pitch, and yaw system) has been designed and fabricated as a proof of concept. Several issues were identified, including the imbalance of the flywheel's angular velocity, inaccuracies in the revolver feeding mechanism, and design flaws in the lever pusher mechanism. Despite these challenges, the team was able to pinpoint the sources of error and propose solutions, such as using a mechanical alignment to mitigate the inaccuracies of stepper motor, and replacing the lever pusher mechanism with a linear actuator. Further iterations are necessary to refine the system, as is typical in engineering projects. Throughout this project, the team gained valuable experience in mechanical design and fabrication techniques, and have significantly improved their ability to discern the reasons behind successful and unsuccessful designs.

The redesigned dart has proven that a large contributing factor to the inconsistencies in the previous design was the weight distribution and the suboptimal aerodynamic profile of the old system. We have also validated the use of PLA Aero together with PLA for more complex designs that would not be possible using the previous TPU 95A. This was done in conjunction with a modular design to increase the ease of repair due to the more fragile nature of PLA and PLA Aero. Lastly, the modular structure was adapted to work with an actively controlled dart, It was then proven that servos can be used to control the active control surfaces at the back of the dart, however, there are design limitations in assembly due to the requirement of a customised PCB to house all the electronics within the housing.

The Cytron 801S vibration sensor successfully recorded vibration data from both the old and new dart launcher systems, showing no significant differences in vibration levels, which ruled out vibrations as the primary cause of the old launcher’s inconsistencies. The NEMA 23 stepper motor provided accurate 90-degree rotations during sub-system testing, but struggled under full-load conditions due to insufficient torque and the absence of positional feedback, causing misalignments in the revolver mechanism. Servo motor integration was not attempted due to expected jams in the mechanical structure, preventing effective testing of dart feeding and frame locking. Transitioning to an Arduino UNO for electrical controls improved development ease and reduced subsystem interference, but further development is needed to enable wireless control for competition readiness.

For computer vision, although it is not implemented, the system successfully achieved its core functionalities in static testing conditions. Using the Raspberry Pi Zero 2W and IMX219 camera (160° FOV), the system demonstrated consistent green target detection and appropriate servo control responses. Testing showed rapid target detection capabilities, though a notable cumulative delay in servo response was observed. The hardware integration provided adequate processing power, and the system maintained reliable performance across multiple test cycles. However, the limitation to static testing means performance under dynamic competition conditions remains unverified.

8.2 Future Plans

In the future, additional test launches should be conducted to assess the improvements made to the dart system as a whole. Refinements are necessary for the existing subsystems, and new subsystems will need to be designed and integrated.

The current dart launcher prototype lacks a base with pitch and yaw mechanisms due to the project time constraints. These should be implemented and integrated with the existing system in future iterations, and tested in a similar manner. In terms of structural rigidity, the current prototype performs well, as most components are fabricated from aluminium profile bars, aluminium plates, and off-the-shelf mechanical parts. However, some parts were 3D printed or laser-cut from acrylic for prototyping. Once the design is finalised and all systems are thoroughly tested, these parts should be replaced with CNC-machined metal or stronger materials, such as carbon fibre. Overall, the dart launcher system requires further testing, refinement, and improvements to ensure optimal performance for the upcoming RMUC competition.

The Dart can potentially gain even more from more rigorous testing on the weight distribution in the dart, this would be even more so important in the passive dart to ensure that the Dart trigger is at the optimal angle when it hits the target. More experimentation is also possible on the effect of different profiles and shapes of the modular fins and their respective aerodynamic properties. The side plates can also be experimented on, determining the optimal interference fit for more efficient launches could potentially result in more stable and consistent launch angles as well.
On the active Dart, more experimentation is also possible on the optimal size of the control surface and the optimal control policy for good aerodynamic response in the final active Dart. This would be possible upon the completion of a customised PCB to house all the electronics needed by an actively controlled system.

To address positional inaccuracies, a higher-torque stepper motor with encoder-based closed-loop control should be explored, else a DC gear motor with opto sensors for precise stopping could be implemented. Servo motor integration will be revisited once mechanical jamming is resolved, ensuring smooth dart feeding and locking mechanisms. The electrical subsystem may have to transition to the Dev-C to enable wireless operation during competition. PID parameters for flywheel controls will be tuned to enhance propulsion accuracy and consistency under RMUC conditions.

The computer vision system improvements can be prioritised alongside other subsystem enhancements. The current thresholding algorithm requires upgrades through morphological operations to improve target detection reliability. After dynamic testing is completed, more sophisticated image processing methods like background subtraction and optical flow tracking could be implemented for better moving target detection. While leveraging OpenCV libraries would provide access to optimised algorithms, careful consideration must be given to processing overhead on the Raspberry Pi Zero 2W platform.

Hardware improvements for the vision system should focus on reducing system latency. The existing servo motor might need to be upgraded to a higher-performance model to address the cumulative delay issues. Additionally, like other subsystems, the vision components require testing in competition-representative conditions. This would enable proper calibration of

thresholding parameters under RMUC-specific lighting and validate performance across the required 15-30 metre target distances.

The overall system testing should also be planned to be tested in a draft-free setting. If the competition team intends to use the current location for preliminary stress tests, as an intermediate measure, wind speed can be intermittently recorded during testing to note potential correlations with dart deviations. This is to mitigate the influence of drafts and better approximate competition conditions. Ultimately, however, sport halls in the university should be booked prior to the competition to validate the performance of future dart systems in conditions closest to the RMUC setting.

Chapter 9 | References

Technical Manuals and Official Documents

RoboMaster. (2024). RoboMaster 2024 University Championship Rules Manual V1.2.
RoboMaster. (2024). RoboMaster 2024 University Series Robot Building Specifications Manual V1.2.

Academic Journal Articles and Books

Caughhey, D. A. (2011). Introduction to aircraft stability and control: Course notes for M&AE 5070. Sibley School of Mechanical & Aerospace Engineering, Cornell University.

Chen, J., Chong, E. K., & Luk, B. L. (2006). Investigation of MEMS accelerometers for angle measurement. Journal of Micromechanics and Microengineering, 16(5), 1038–1043. https://doi.org/10.1088/0960-1317/16/5/024

Etkin, B., & Reid, L. D. (1996). Dynamics of flight: Stability and control (3rd ed.). Wiley.

Ewins, D. J. (2000). Modal testing: Theory, practice and application (2nd ed.). Research Studies Press.

Gajbhiye, S. D., & Gundewar, P. P. (2015). A real-time color-based object tracking and occlusion handling using ARM cortex-A7. IEEE, 1–6. https://doi.org/10.1109/indicon.2015.7443641

Gevers, T., & Smeulders, A. W. (1999). Color-based object recognition. Pattern Recognition, 32(3), 453-464.

Goh, T. Y., Basah, S. N., Yazid, H., Safar, M. J. A., & Saad, F. S. A. (2017). Performance analysis of image thresholding: Otsu technique. Measurement, 114, 298–307. https://doi.org/10.1016/j.measurement.2017.09.052

Jang, G., Kim, C., Seo, S., Shin, K., Yoon, I., & Choi, J. (2019). Torque characteristic analysis and measurement of magnetic Rack–Pinion gear based on analytical method. IEEE Transactions on Magnetics, 55(7), 1–5. https://doi.org/10.1109/tmag.2019.2900447

Jansari, D., Parmar, S., & Saha, G. (2013). Real-time object tracking using color-based probability matching. IEEE, 1, 1–6. https://doi.org/10.1109/ispcc.2013.6663399

Liu, C. (2006). Foundations of MEMS. Prentice Hall.

Liu, S., Liu, D., Srivastava, G., Połap, D., & Woźniak, M. (2020). Overview and methods of correlation filter algorithms in object tracking. Complex & Intelligent Systems. https://doi.org/10.1007/s40747-020-00161-4

McConnell, K. G., & Varoto, P. S. (2008). Vibration testing: Theory and practice (2nd ed.). John Wiley & Sons.

Nelson, R. C. (1998). Flight stability and automatic control (2nd ed.). McGraw-Hill.

Post, R. F., & Post, S. F. (1973). Flywheels. Scientific American, 229(6), 17–23. http://www.jstor.org/stable/24923262

Wang, L. (2020). PID control system design and automatic tuning using MATLAB/Simulink. John Wiley & Sons.

Wilson, J. S. (2005). Sensor technology handbook. Elsevier.

Yang, H., Shao, L., Zheng, F., Wang, L., & Song, Z. (2011). Recent advances and trends in visual tracking: A review. Neurocomputing, 74(18), 3823–3831. https://doi.org/10.1016/j.neucom.2011.07.024

Yazdi, N., Ayazi, F., & Najafi, K. (1998). Micromachined inertial sensors. Proceedings of the IEEE, 86(8), 1640–1659. https://doi.org/10.1109/5.704270

Technical Standards
International Organization for Standardization. (2009). ISO 2041:2009: Mechanical vibration, shock and condition monitoring—Vocabulary.

Online Resources and Websites

Bambu Lab. (n.d.). Studio settings for RC models. Bambu Lab Wiki. Retrieved November 14, 2024, from https://wiki.bambulab.com/en/knowledge-sharing/studio-settings-for-rc-models

Connolly, M. (2024). Rubber band cannons: Harnessing elastic energy for fun projectile experiments. LearningMole. https://learningmole.com/rubber-band-cannons/

KB Delta. (2017). How does a tension spring work? https://kbdelta.com/blog/tension-spring-work

Nakka, R. (n.d.). Fin aerodynamics. Retrieved November 18, 2024, from https://www.nakka-rocketry.net/fins.html

Tru Physics. (2023, February 22). Projectile motion—Rubber bands and hex nuts. https://tru-physics.org/2023/02/22/projectile-motion-rubber-bands-and-hex-nuts/

VEX Forum. (2015). Physics of the flywheel launcher. https://www.vexforum.com/t/physics-of-the-flywheel-launcher/29357

Video Resources

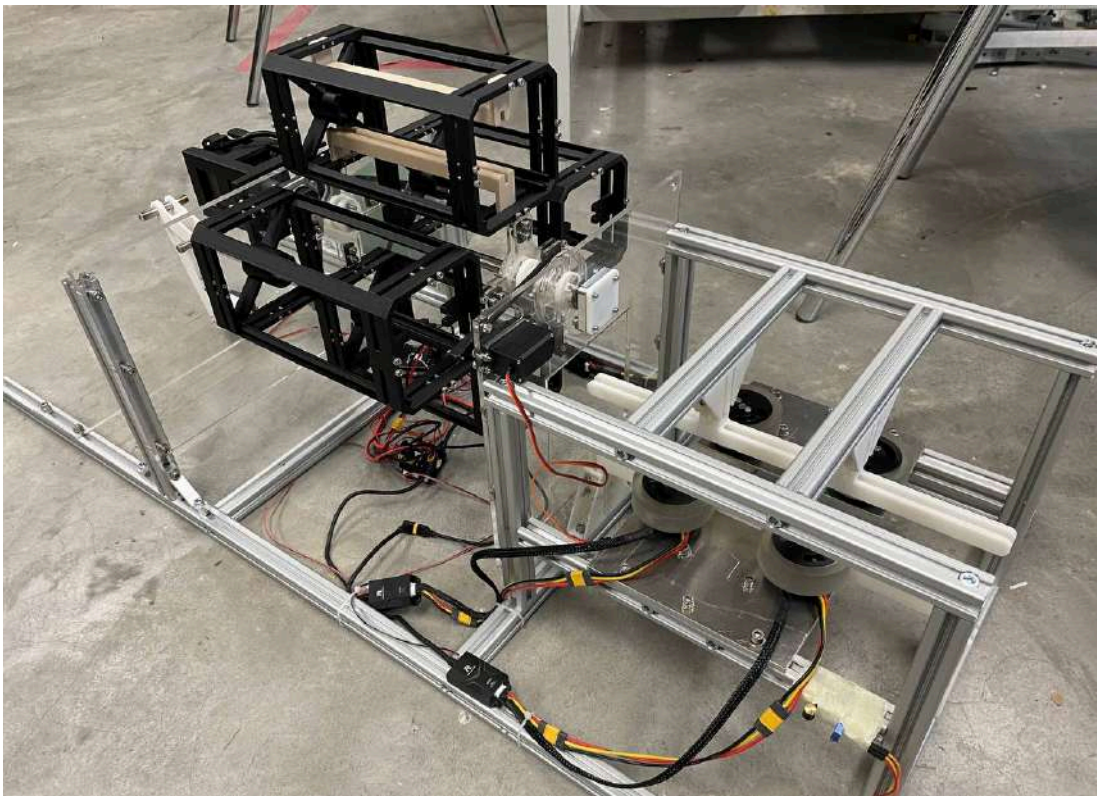
RoboMaster机甲大师. (2024, May 17). 24赛季首个随机靶命中！｜西北工业大学制导飞镖展示 [Video]. Bilibili. https://www.bilibili.com/video/BV1YU411o7b5/

RoboMaster赛务君. (2024, August 14). RMUC 2024青工会技术分享 南京航空航天大学 飞镖机械设计 [Video]. Bilibili. https://www.bilibili.com/video/BV1ox4y1W783/

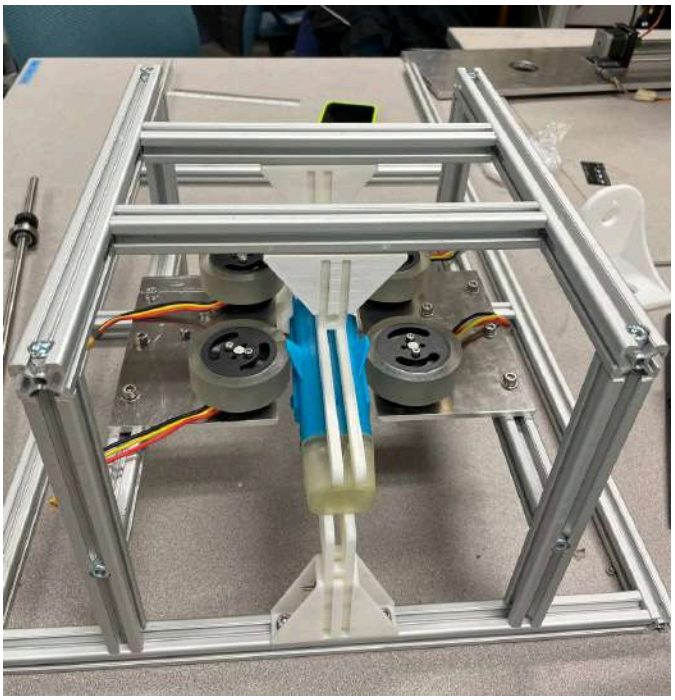
Open Source Materials and Archives

Beijing University of Science and Technology. (2021). Open source.
Nanjing University of Aeronautics and Astronautics. (2022). Open source.
National University of Singapore Robomaster Archives. (2023).
South China University of Technology. (2023). Open source.

Chapter 10 | Appendices



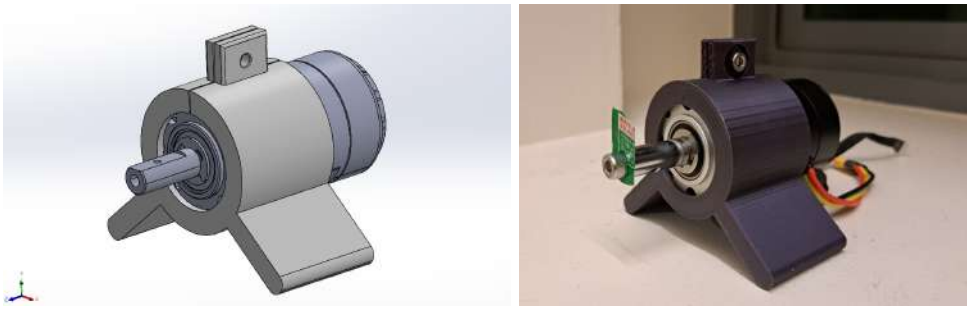
Appendix A: Dart Launcher Prototype



Appendix B: Flywheel Section

Appendix C: Revolver Section

Appendix D: Lever Pusher Section



Appendix E: M3508 mount

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
int16_t motor_rpm1;
int16_t motor_rpm2;
/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
CAN_HandleTypeDef hcan1;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_CAN1_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
```

```c
/* USER CODE BEGIN 0 */

void setup_can(){
 CAN_FilterTypeDef can_filter_st = {0};
 can_filter_st.FilterActivation = ENABLE;
 can_filter_st.FilterMode = CAN_FILTERMODE_IDMASK;
 can_filter_st.FilterScale = CAN_FILTERSCALE_32BIT;
 can_filter_st.FilterIdHigh = 0;
 can_filter_st.FilterIdLow = 0;
 can_filter_st.FilterMaskIdHigh = 0;
 can_filter_st.FilterMaskIdLow = 0;
 can_filter_st.FilterBank=0;
 can_filter_st.FilterFIFOAssignment = CAN_RX_FIFO0;
 HAL_CAN_ConfigFilter(&hcan1, &can_filter_st);
 HAL_CAN_Start(&hcan1);
 HAL_CAN_ActivateNotification(&hcan1,CAN_IT_RX_FIFO0_MSG_PENDING | CAN_IT_RX_FIFO0_FULL| CAN_IT_RX_FIFO0_OVERRUN);
}



void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan){
CAN_RxHeaderTypeDef rx_header;
uint8_t rx_buffer[8];
HAL_CAN_DeactivateNotification(hcan,
  CAN_IT_RX_FIFO0_MSG_PENDING | CAN_IT_RX_FIFO0_FULL| CAN_IT_RX_FIFO0_OVERRUN);
HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO0, &rx_header, rx_buffer);

if (rx_header.StdId == 0x203)
{
           motor_rpm1 = (rx_buffer[2] << 8) + rx_buffer[3];
}

if (rx_header.StdId == 0x202)
{
           motor_rpm2 = (rx_buffer[2] << 8) + rx_buffer[3];
}

HAL_CAN_ActivateNotification(hcan,
  CAN_IT_RX_FIFO0_MSG_PENDING | CAN_IT_RX_FIFO0_FULL| CAN_IT_RX_FIFO0_OVERRUN);
}



void ctrl_motor(int16_t torque1,int16_t torque2){
 uint8_t tx_msg[8];
 CAN_TxHeaderTypeDef CAN_tx_message;
 uint32_t send_mail_box;
 CAN_tx_message.IDE = CAN_ID_STD;
 CAN_tx_message.RTR = CAN_RTR_DATA;
 CAN_tx_message.DLC = 0x08;
 CAN_tx_message.StdId = 0x200;
 tx_msg[4] = torque1>>8;
 tx_msg[5] = torque1;
 tx_msg[2] = torque2>>8;
 tx_msg[3] = torque2;
 HAL_CAN_AddTxMessage(&hcan1, &CAN_tx_message, tx_msg, &send_mail_box);
}


#define KP 1
#define KI 0.01
#define KD 0
#define INT_MAX 5000
int16_t pid_lol(int16_t setpt, int16_t curr_pt){
           static float integral;
           static float prev_error;
           float error = setpt - curr_pt;
           integral += error * KI;
           integral = (integral > INT_MAX) ? INT_MAX : (integral < -INT_MAX) ? -INT_MAX : integral;
           float diff = (prev_error - error) * KD;
           prev_error = error;
           return (error * KP) + integral + diff;

}

int16_t pid_lol_2(int16_t setpt, int16_t curr_pt){
           static float integral;
           static float prev_error;
```

```
            float error = setpt - curr_pt;
            integral += error * KI;
            integral = (integral > INT_MAX) ? INT_MAX : (integral < -INT_MAX) ? -INT_MAX : integral;
            float diff = (prev_error - error) * KD;
            prev_error = error;
            return (error * KP) + integral + diff;

}

#define SPEED 557
/* USER CODE END 0 */


/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_CAN1_Init();
  /* USER CODE BEGIN 2 */
  setup_can();

  //uint32_t last_button_time = HAL_GetTick();
  //int16_t spd = 1;
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  int counter=0;
  while (1)
  {


            ctrl_motor(pid_lol(SPEED,motor_rpm1) , pid_lol_2(SPEED,motor_rpm2));
            //ctrl_motor(pid_lol(SPEED,motor_rpm1) , pid_lol(SPEED,motor_rpm2));
            //ctrl_motor(SPEED);
            HAL_Delay(1); //milliseconds

            counter += 1;
            if (counter==3000){  //1000 counts for 1 cycle
                      ctrl_motor(0, 0);
                      break;
            }


            /*
            if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == 0){
                      if (HAL_GetTick() - last_button_time > 500){
                                spd = (spd == 1) ? 0 : (spd== 0) ? -1 : (spd == -1) ? 1 : 0;
                                last_button_time = HAL_GetTick();
                      }
            }
            */
```

```
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 6;
  RCC_OscInitStruct.PLL.PLLN = 168;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief CAN1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_CAN1_Init(void)
{

  /* USER CODE BEGIN CAN1_Init 0 */

  /* USER CODE END CAN1_Init 0 */

  /* USER CODE BEGIN CAN1_Init 1 */

  /* USER CODE END CAN1_Init 1 */
  hcan1.Instance = CAN1;
  hcan1.Init.Prescaler = 3;
  hcan1.Init.Mode = CAN_MODE_NORMAL;
  hcan1.Init.SyncJumpWidth = CAN_SJW_1TQ;
  hcan1.Init.TimeSeg1 = CAN_BS1_9TQ;
  hcan1.Init.TimeSeg2 = CAN_BS2_4TQ;
  hcan1.Init.TimeTriggeredMode = DISABLE;
  hcan1.Init.AutoBusOff = DISABLE;
  hcan1.Init.AutoWakeUp = DISABLE;
  hcan1.Init.AutoRetransmission = DISABLE;
  hcan1.Init.ReceiveFifoLocked = DISABLE;
```

```
  hcan1.Init.TransmitFifoPriority = DISABLE;
  if (HAL_CAN_Init(&hcan1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN CAN1_Init 2 */

  /* USER CODE END CAN1_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{

  /* GPIO Ports Clock Enable */

  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Appendix F: Dev-C (STM32) Code for Spinning Motors

```
#include <Servo.h>

// Define motor driver pins
const int stepPin = 3;   // PULSE+ pin
const int enaPin = 4;    // ENA+ pin
const int ena2Pin = 5;    // ENA- pin
const int stepsPerRevolution = 400;  // based on your motor driver's specification (steps per 360 degrees)

// Servo configuration
Servo servoMg996R;           // Create Servo object
const int servoPin = 9;  // Pin to servo signal
// Button configuration
const int buttonPin = 13;   // Pin connected to the button
bool buttonPressed = false; // Variable to store button state
```

```
// Movement control
int rotationCount = 0;   // To track number of rotations
int totalRotations = 40;  // Total number of 90-degree rotations
int delayTime = 1000;    // User-defined delay time (1 second by default) - adjust as needed


void setup() {
  // Initialize the motor control pins as outputs
  pinMode(stepPin, OUTPUT);
  pinMode(enaPin, OUTPUT);
  pinMode(ena2Pin, OUTPUT);

  //digitalWrite(dirPin, HIGH); // any direction of stepper works, no need
  digitalWrite(enaPin, LOW); // ENABLE pin
  digitalWrite(ena2Pin, HIGH); // ENABLE pin

  // Initialize servo
  servoMg996R.attach(servoPin);  // Attach servo to specified pin
  servoMg996R.write(0);          // Set servo to initial position

  // Initialize the button pin as input
  pinMode(buttonPin, INPUT_PULLUP);  // Use internal pull-up resistor

  Serial.begin(9600);

  // Hold position
  servoMg996R.write(90);
  Serial.println("0 degrees.");
}

void loop() {

  if (digitalRead(buttonPin) == LOW) { // LOW means button is pressed due to INPUT_PULLUP
    if (!buttonPressed) {
      // Set buttonPressed to true to prevent multiple triggers
      buttonPressed = true;

      // Perform the motor and servo movements
      moveStepper();
      delayMicroseconds(2000);
      moveServo();
      rotationCount++;
    }
  } else {
    // Reset the button state when it is released
    buttonPressed = false;
  }
}

void moveStepper() {
  // Calculate steps for 90-degree rotation (quarter of a full rotation)
  int steps = stepsPerRevolution / 4;

  // generate PWM
  for (int i = 0; i < steps; i++) {
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(1000);  // affects PWM period. affects for motor speed? idk
    digitalWrite(stepPin, LOW);
    delayMicroseconds(2000);
  }

  Serial.print("Rotation ");
  Serial.print(rotationCount + 1);
  Serial.println(" completed.");
}



void moveServo() {
  // Move the servo to +90 degrees
  servoMg996R.write(0);
  Serial.println("+90 degrees.");
  delay(600);  // clockwise for x ms second

  // Hold position
  servoMg996R.write(90);
  Serial.println("0 degrees.");
  delay(250);  // Hold position for x ms second

  // servo to wait
  servoMg996R.write(180);
  Serial.println("-90 degrees.");
  delay(600);  // anticlockwise for x ms second

  // Hold position
  servoMg996R.write(90);
  Serial.println("0 degrees.");
  // delay(1000);  // Hold position for x ms second
}
```

Appendix G: Arduino Code for Running Motors Sub-System (with Push Button)

```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
#define VIBRATION_PIN GPIO_PIN_9  // Replace XX with the actual pin number
#define VIBRATION_GPIO_PORT GPIOE  // Replace X with the actual GPIO Port
/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
TIM_HandleTypeDef htim1;
uint32_t IC_Value1 = 0;
uint32_t IC_Value2 = 0;
uint32_t Difference = 0;
uint8_t Is_First_Captured = 0;  // Flag to indicate if the first capture is done
uint32_t pulseDuration = 0;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM1_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */


/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{

  /* USER CODE BEGIN 1 */
```

```
  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM1_Init();
  /* USER CODE BEGIN 2 */

  // Start the Input Capture in interrupt mode
  HAL_TIM_IC_Start_IT(&htim1, TIM_CHANNEL_1);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 6;
  RCC_OscInitStruct.PLL.PLLN = 84;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
```

```
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief TIM1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM1_Init(void)
{

  /* USER CODE BEGIN TIM1_Init 0 */

  /* USER CODE END TIM1_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_IC_InitTypeDef sConfigIC = {0};

  /* USER CODE BEGIN TIM1_Init 1 */

  /* USER CODE END TIM1_Init 1 */
  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 72-1;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 65535;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_IC_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
  sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
  sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
  sConfigIC.ICFilter = 0;
  if (HAL_TIM_IC_ConfigChannel(&htim1, &sConfigIC, TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM1_Init 2 */

  /* USER CODE END TIM1_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOE_CLK_ENABLE();

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
/* Input Capture Interrupt Callback Function */
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
```

```
    if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)  // Check if interrupt is from Channel 1
    {
        if (Is_First_Captured == 0)  // First edge detected (rising edge)
        {
            IC_Value1 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);  // Capture first value
            Is_First_Captured = 1;  // Set the flag for first capture
            __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_1, TIM_INPUTCHANNELPOLARITY_FALLING);  // Switch to falling edge
        }
        else if (Is_First_Captured == 1)  // Second edge detected (falling edge)
        {
            IC_Value2 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);  // Capture second value

            if (IC_Value2 > IC_Value1)
            {
                Difference = IC_Value2 - IC_Value1;  // Calculate the difference (pulse width)
            }
            else
            {
                Difference = (0xFFFF - IC_Value1) + IC_Value2;  // Handle overflow
            }

            // Pulse width is in timer ticks; convert it to microseconds
            pulseDuration = Difference;  // Timer configured for 1 µs resolution

            // Process the pulse duration (e.g., log or use it)
            Is_First_Captured = 0;  // Reset the flag for the next capture
            __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_1, TIM_INPUTCHANNELPOLARITY_RISING);  // Switch back to rising edge

        }
    }
}
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Appendix H: Dev-C (STM32) Code for Running Vibration Sensor

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
```

```
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
int16_t motor_rpm1;
int16_t motor_rpm2;
int16_t motor_rpm3;
int16_t motor_rpm4;
/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
CAN_HandleTypeDef hcan1;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_CAN1_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

void setup_can(){
 CAN_FilterTypeDef can_filter_st = {0};
 can_filter_st.FilterActivation = ENABLE;
 can_filter_st.FilterMode = CAN_FILTERMODE_IDMASK;
 can_filter_st.FilterScale = CAN_FILTERSCALE_32BIT;
 can_filter_st.FilterIdHigh = 0;
 can_filter_st.FilterIdLow = 0;
 can_filter_st.FilterMaskIdHigh = 0;
 can_filter_st.FilterMaskIdLow = 0;
 can_filter_st.FilterBank=0;
 can_filter_st.FilterFIFOAssignment = CAN_RX_FIFO0;
 HAL_CAN_ConfigFilter(&hcan1, &can_filter_st);
 HAL_CAN_Start(&hcan1);
 HAL_CAN_ActivateNotification(&hcan1,CAN_IT_RX_FIFO0_MSG_PENDING | CAN_IT_RX_FIFO0_FULL| CAN_IT_RX_FIFO0_OVERRUN);
}



void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan){
CAN_RxHeaderTypeDef rx_header;
uint8_t rx_buffer[8];
HAL_CAN_DeactivateNotification(hcan,
  CAN_IT_RX_FIFO0_MSG_PENDING | CAN_IT_RX_FIFO0_FULL| CAN_IT_RX_FIFO0_OVERRUN);
HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO0, &rx_header, rx_buffer);

if (rx_header.StdId == 0x201)
{
  motor_rpm1 = (rx_buffer[2] << 8) + rx_buffer[3];
}

if (rx_header.StdId == 0x202)
{
  motor_rpm2 = (rx_buffer[2] << 8) + rx_buffer[3];
}
if (rx_header.StdId == 0x203)
{
  motor_rpm3 = (rx_buffer[2] << 8) + rx_buffer[3];
```

```
}

if (rx_header.StdId == 0x204)
{
  motor_rpm4 = (rx_buffer[2] << 8) + rx_buffer[3];
}

HAL_CAN_ActivateNotification(hcan,
  CAN_IT_RX_FIFO0_MSG_PENDING | CAN_IT_RX_FIFO0_FULL| CAN_IT_RX_FIFO0_OVERRUN);
}



void ctrl_motor(int16_t torque1,int16_t torque2,int16_t torque3,int16_t torque4){
 uint8_t tx_msg[8];
 CAN_TxHeaderTypeDef CAN_tx_message;
 uint32_t send_mail_box;
 CAN_tx_message.IDE = CAN_ID_STD;
 CAN_tx_message.RTR = CAN_RTR_DATA;
 CAN_tx_message.DLC = 0x08;
 CAN_tx_message.StdId = 0x200;
 tx_msg[0] = torque1>>8;
 tx_msg[1] = torque1;
 tx_msg[2] = torque2>>8;
 tx_msg[3] = torque2;
 tx_msg[4] = torque3>>8;
 tx_msg[5] = torque3;
 tx_msg[6] = torque4>>8;
 tx_msg[7] = torque4;
 HAL_CAN_AddTxMessage(&hcan1, &CAN_tx_message, tx_msg, &send_mail_box);
}


#define KP 1
#define KI 0.01
#define KD 0
#define INT_MAX 5000
int16_t pid_lol(int16_t setpt, int16_t curr_pt){
  static float integral;
  static float prev_error;
  float error = setpt - curr_pt;
  integral += error * KI;
  integral = (integral > INT_MAX) ? INT_MAX : (integral < -INT_MAX) ? -INT_MAX : integral;
  float diff = (prev_error - error) * KD;
  prev_error = error;
  return (error * KP) + integral + diff;

}

int16_t pid_lol_2(int16_t setpt, int16_t curr_pt){
  static float integral;
  static float prev_error;
  float error = setpt - curr_pt;
  integral += error * KI;
  integral = (integral > INT_MAX) ? INT_MAX : (integral < -INT_MAX) ? -INT_MAX : integral;
  float diff = (prev_error - error) * KD;
  prev_error = error;
  return (error * KP) + integral + diff;

}

int16_t pid_lol_3(int16_t setpt, int16_t curr_pt){
  static float integral;
  static float prev_error;
  float error = setpt - curr_pt;
  integral += error * KI;
  integral = (integral > INT_MAX) ? INT_MAX : (integral < -INT_MAX) ? -INT_MAX : integral;
  float diff = (prev_error - error) * KD;
  prev_error = error;
  return (error * KP) + integral + diff;

}

int16_t pid_lol_4(int16_t setpt, int16_t curr_pt){
  static float integral;
  static float prev_error;
  float error = setpt - curr_pt;
  integral += error * KI;
  integral = (integral > INT_MAX) ? INT_MAX : (integral < -INT_MAX) ? -INT_MAX : integral;
  float diff = (prev_error - error) * KD;
  prev_error = error;
  return (error * KP) + integral + diff;

}

#define SPEED 3501
/* USER CODE END 0 */


/**
```

```
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_CAN1_Init();
  /* USER CODE BEGIN 2 */
  setup_can();

  //uint32_t last_button_time = HAL_GetTick();
  //int16_t spd = 1;
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {


    ctrl_motor(pid_lol(-SPEED,motor_rpm1) , pid_lol_2(SPEED,motor_rpm2)
        , pid_lol_3(-SPEED,motor_rpm3), pid_lol_4(SPEED,motor_rpm4));
    //ctrl_motor(pid_lol(SPEED,motor_rpm1) , pid_lol(SPEED,motor_rpm2));
    //ctrl_motor(SPEED);
    HAL_Delay(1);


    /*
    if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == 0){
      if (HAL_GetTick() - last_button_time > 500){
        spd = (spd == 1) ? 0 : (spd== 0) ? -1 : (spd == -1) ? 1 : 0;
        last_button_time = HAL_GetTick();
      }
    }
    */


    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 6;
  RCC_OscInitStruct.PLL.PLLN = 168;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
```

```
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief CAN1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_CAN1_Init(void)
{

  /* USER CODE BEGIN CAN1_Init 0 */

  /* USER CODE END CAN1_Init 0 */

  /* USER CODE BEGIN CAN1_Init 1 */

  /* USER CODE END CAN1_Init 1 */
  hcan1.Instance = CAN1;
  hcan1.Init.Prescaler = 3;
  hcan1.Init.Mode = CAN_MODE_NORMAL;
  hcan1.Init.SyncJumpWidth = CAN_SJW_1TQ;
  hcan1.Init.TimeSeg1 = CAN_BS1_9TQ;
  hcan1.Init.TimeSeg2 = CAN_BS2_4TQ;
  hcan1.Init.TimeTriggeredMode = DISABLE;
  hcan1.Init.AutoBusOff = DISABLE;
  hcan1.Init.AutoWakeUp = DISABLE;
  hcan1.Init.AutoRetransmission = DISABLE;
  hcan1.Init.ReceiveFifoLocked = DISABLE;
  hcan1.Init.TransmitFifoPriority = DISABLE;
  if (HAL_CAN_Init(&hcan1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN CAN1_Init 2 */

  /* USER CODE END CAN1_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{

  /* GPIO Ports Clock Enable */

  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}
```

```c
#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Appendix I: Dev-C (STM32) Code for Spinning Flywheels

```python
import numpy as np
import time
from picamera2 import Picamera2
from picamera2.encoders import H264Encoder
from picamera2.outputs import FfmpegOutput
from libcamera import controls
import RPi.GPIO as GPIO

# Color Tracking Thresholds (R, G, B) - need to be changed based on enviroment
LOWER_THRESHOLD = np.array([0, 100, 0])  # Lower green threshold
UPPER_THRESHOLD = np.array([100, 255, 100])  # Upper green threshold

# Set up camera
CAPTURE_SIZE = (640, 480)
picam2 = Picamera2()
video_config = picam2.create_video_configuration(main={"size": CAPTURE_SIZE, "format": "RGB
# 24-bit color (8 bits per channel)
picam2.configure(video_config)

# Enable auto exposure and auto white balance
picam2.set_controls({"AeEnable": True, "AwbEnable": True})
encoder = H264Encoder(bitrate=10000000) # 10 Mbps for standard recording
output = FfmpegOutput('test.mp4')

picam2.start_recording(encoder, output)
print("Warming up the camera...")
time.sleep(2)

#Servo Motor Control
# Set up
GPIO.setmode(GPIO.BCM)
SERVO_PIN = 12

# Setup pin
GPIO.setup(SERVO_PIN, GPIO.OUT)
pwm = GPIO.PWM(SERVO_PIN, 50)  # 50Hz (20ms) for servos - industry standard
pwm.start(0)

max_duty = 10; # fully extend - ~9mm
min_duty = 4; # fully retracted - 0mm
# the actual relationship between change in duty cycle and length is not linear,
# average changeLength per changeDuty is 1.5mm

def test_duty_cycle(duty):
   GPIO.output(SERVO_PIN, True)
   pwm.ChangeDutyCycle(duty)
   time.sleep(0.2) # Can shorten in real competition


"""
Detect and analyze colored regions in the image.
```

```
Args:
   img (numpy.ndarray): Input RGB image
   lower_thresh (numpy.ndarray): Lower RGB threshold values
   upper_thresh (numpy.ndarray): Upper RGB threshold values
   min_area (int): Minimum pixel area to be considered a valid blob

Returns:
   list: List of tuples (x, y, w, h, cx, cy) for each detected blob where:
         x, y: Top-left corner coordinates
         w, h: Width and height of bounding box
         cx, cy: Center coordinates of the blob
"""
def find_blobs(img, lower_thresh, upper_thresh, min_area=100):
   # Create a binary mask using the color thresholds
   mask = np.all((img >= lower_thresh) & (img <= upper_thresh), axis=2)

   # Find connected components (blobs)
   labeled, num_labels = np.zeros_like(mask), 0
   neighbors = [(0,1), (0,-1), (1,0), (-1,0)]

   blobs = []
   for i in range(mask.shape[0]):
       for j in range(mask.shape[1]):
           if mask[i, j] and not labeled[i, j]:
               num_labels += 1
               stack, area = [(i, j)], 0
               min_x, min_y, max_x, max_y = j, i, j, i

               while stack:
                   y, x = stack.pop()
                      if 0 <= y < mask.shape[0] and 0 <= x < mask.shape[1] and mask[y, x] a
labeled[y, x]:
                       labeled[y, x] = num_labels
                       area += 1
                       min_x, max_x = min(min_x, x), max(max_x, x)
                       min_y, max_y = min(min_y, y), max(max_y, y)
                       stack.extend((y+dy, x+dx) for dy, dx in neighbors)

               if area >= min_area:
                   cx = (min_x + max_x) // 2
                   cy = (min_y + max_y) // 2
                   blobs.append((min_x, min_y, max_x-min_x, max_y-min_y, cx, cy))

   return blobs


print("Starting color tracking and video recording. Press Ctrl+C to quit.")
start_time = time.time()
try:
   while True:
       frame = picam2.capture_array()
       blobs = find_blobs(frame, LOWER_THRESHOLD, UPPER_THRESHOLD)

       if blobs:
           print("\nDetected blobs:")
           for i, blob in enumerate(blobs, 1):
               x, y, w, h, cx, cy = blob
               print(f"Blob {i}: Center ({cx}, {cy}), Size {w}x{h}")
           largest_blob = max(blobs, key=lambda b: b[2] * b[3])
           x, y, w, h, cx, cy = largest_blob  # Correctly unpack all values
           print(f"Largest Blob: Center ({cx}, {cy}), Size {w}x{h}")

           # Servo control with more precise positioning
           if cx > 450:
               print("Target is at the right, servo move up")
               test_duty_cycle(max_duty-2)
           elif cx < 150:
               print("Target is at the left, servo move down")
               test_duty_cycle(min_duty)
           else:
```

```
                print("Target is at center, no need to change")
        else:
            print("No blobs detected", end='\r')

        if time.time() - start_time > 60:  # Stop after 60 seconds
            break

        time.sleep(2)  # Delay to reduce CPU usage

except KeyboardInterrupt:
    print("\nInterrupted by user. Exiting...")

finally:
    picam2.stop_recording()
    pwm.stop()
    GPIO.cleanup()
    print("Video recording stopped.")
```

Appendix J: Vision system code (final choice)

```
import sympy as sp

# Define symbolic variables
v0, theta = sp.symbols('v0 theta')        # Initial speed and launch angle
m, g = sp.symbols('m g')                  # Mass and gravity
L, D = sp.symbols('L D')                  # Lift and drag forces
t = sp.symbols('t')                       # Time variable

# Compute accelerations
ax = -D / m
ay = -g + L / m

# Initial velocity components
v0x = v0 * sp.cos(theta)
v0y = v0 * sp.sin(theta)

# Time of flight (t_total) when vertical displacement is zero
# Equation: y = v0y * t_total + 0.5 * ay * t_total^2 = 0
# Solve for t_total (excluding t=0)
t_total = (-2 * v0y) / ay

# Horizontal and vertical positions as functions of time
x_t = v0x * t + 0.5 * ax * t**2
y_t = v0y * t + 0.5 * ay * t**2

# Simplify expressions
x_t_simplified = sp.simplify(x_t)
y_t_simplified = sp.simplify(y_t)

# Now, define numerical values for the parameters
numerical_values = {
    m: 0.11,              # Mass in kg
    g: 9.8,              # Gravity in m/s^2
    L: 0.003,            # Lift force in N
    D: 0.184,            # Drag force in N
    theta: sp.rad(47),   # Convert 47 degrees to radians
```

```
}

# Substitute numerical values into accelerations and initial velocities
ax_num = ax.subs(numerical_values)
ay_num = ay.subs(numerical_values)
v0x_num = v0x.subs(numerical_values)
v0y_num = v0y.subs(numerical_values)
t_total_num = t_total.subs(numerical_values)

# Create numerical functions for positions
x_t_func = sp.lambdify((v0, t), x_t.subs(numerical_values), modules='numpy')
y_t_func = sp.lambdify((v0, t), y_t.subs(numerical_values), modules='numpy')

# Define the initial speed (from previous calculation)
v0_specific = 16.5  # Initial speed in m/s

# Compute time values from 0 to t_total_num
import numpy as np
import matplotlib.pyplot as plt

# Evaluate t_total_num numerically
t_total_specific = t_total_num.subs(v0, v0_specific)
t_total_specific = float(t_total_specific)

# Generate time values
t_values = np.linspace(0, t_total_specific, 500)

# Compute x and y positions over time
x_values = x_t_func(v0_specific, t_values)
y_values = y_t_func(v0_specific, t_values)

# Plot the trajectory
plt.figure(figsize=(8, 6))
plt.plot(x_values, y_values, label='Projectile Trajectory')
plt.title('Projectile Motion with Lift and Drag')
plt.xlabel('Horizontal Distance (m)')
plt.ylabel('Vertical Height (m)')
plt.legend()
plt.grid(True)
plt.show()

# Plot distance to initial speed

# Plot Range vs Initial Speed
plt.plot(v0_values, R_values)
plt.title('Distance Traveled vs Initial Speed')
plt.xlabel('Initial Speed (m/s)')
plt.ylabel('Distance Traveled (m)')
plt.grid(True)
plt.show()

# Additionally, display the symbolic expression for Range (optional)
```

```
R = x_t.subs(t, t_total)
R_simplified = sp.simplify(R)
print("Symbolic expression for Range (R):")
sp.pprint(R_simplified)

# Calculate the distance traveled at the specific initial speed
R_numeric = R_simplified.subs(numerical_values).subs(v0, v0_specific)
distance = float(R_numeric)

print(f"\nDistance traveled at initial speed {v0_specific} m/s: {distance:.2f} meters")
```

Appendix K: General Simulation for Launch

```
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt

# Define symbolic variables
v0, theta = sp.symbols('v0 theta')        # Initial speed and launch angle
m, g = sp.symbols('m g')                  # Mass and gravity
L, D = sp.symbols('L D')                  # Lift and drag forces
t = sp.symbols('t')                       # Time variable

# Compute accelerations
ax = -D / m
ay = -g + L / m

# Initial velocity components
v0x = v0 * sp.cos(theta)
v0y = v0 * sp.sin(theta)

# Time of flight (t_total) when vertical displacement is zero
# Equation: y = v0y * t_total + 0.5 * ay * t_total^2 = 0
# Solve for t_total (excluding t=0)
t_total = (-2 * v0y) / ay

# Horizontal distance (Range)
R = v0x * t_total + 0.5 * ax * t_total**2
R_simplified = sp.simplify(R)

# Define numerical values for the parameters (excluding mass)
numerical_values = {
   g: 9.8,              # Gravity in m/s^2
   L: 0.003,            # Lift force in N
   D: 0.184,            # Drag force in N
   theta: np.deg2rad(47),  # Convert 47 degrees to radians
   v0: 16.5,           # Initial speed in m/s (from previous calculation)
}

# Create a function to compute distance as a function of mass
m_values = np.linspace(0.01, 0.5, 500)  # Mass values from 0.01 kg to 0.5 kg
distance_values = []
```

```
for mass in m_values:
    # Update mass in numerical values
    numerical_values[m] = mass
    # Substitute numerical values into R_simplified
    R_numeric = R_simplified.subs(numerical_values)
    # Evaluate R_numeric
    distance = float(R_numeric)
    distance_values.append(distance)

# Plot Distance Traveled vs Mass
plt.figure(figsize=(8, 6))
plt.plot(m_values, distance_values)
plt.title('Distance Traveled vs Mass')
plt.xlabel('Mass (kg)')
plt.ylabel('Distance Traveled (m)')
plt.grid(True)
plt.show()
```

Appendix L: Simulation for Distance Travelled against Mass

Item Name	Qty	Price	+ shipping	Reason to purchase	Website	Remarks
OpenMV RT1062	1	120 USD	$ 213.41	Board in dart	OpenMV	
Camera Module Extension cable - Open	1	10 USD		Extend camera from OpenMV board	OpenMV	
Rpi zero 2w	1	21.91 SGD	$ 32.80	Possible board in dart(we would like to explore	Element 14	
IMX219 Camera	1	21.20 SGD	$ 21.20	Rpi camera	Shopee	
7.4V 450 mAh Lipo Battery	2	12.02 SGD (After con	$ 13.07	Powering the boards	TaoBao	
Buck Convertor output 5V	2	2.60 SGD	$ 4.09	Step down to 5V to power the dart trigger and	Shopee	Select the first option mini56
Foaming PLA	1	44.99 USD	$ 67.86	Print Dart Structure	BambuLab	Select white I guess
L50 HI Green LED with 20mm Copper M	3	33.78 SGD	$ 41.90	Mimic greenlight at base/outpost for CV/Dart t	AliExpress	Select the 20mm Copper MC
LM2596 Stepdown	5	11.32 SGD		24V dji battery stepdown to power the green L	AliExpress	Select the one with 5 boards
Vibration Sensor	2	3.60 SGD	$ 15.22	Vibration Sensor	Cytron.io	If dh then select any vibratio
Micro USB stripped	1	1.73 SGD	$ 5.76	Buck convertor to rpi power	Shopee	Select 2-pin micro male
Type -C cable Stripped	1	2.50 SGD		Buck convertor to openMV power	Shopee	Select 2pin type-c male
7.4V lipo battery charger	1	2.57 SGD (After conv	$ 2.87	Charge Lipo battery from wallplug	TaoBao	
Sandisk 64GB MicroSD card	2	24.60 SGD	$ 24.10	Storage and to run OS in rpi and OpenMV	Shopee	Select 64GB Up to 140MB/s
7.4V 500mAh LipoBattery	3	27.3	$ 42.60	Battery inside dart	Lazada	delivered, Deliver by 4-10 Oc
Lipo Battery charger	1	14.3		Charge the lipo battery	Lazada	delivered, Deliver by 4 - 10 O
7.5cm diameter frosted acrylic	1	4.92	$ 8.23	DART test bench	Aliexpress	Delivered, by 13 Oct
Mini HDMI Male to HDMI Female Cable	1	4.47	$ 5.96	Get display from Rpi zero 2w	shoppee	Delivered
20pcs 2 Pin connector male female jst p	10	1.9	$ 3.39		Shoppee	Delivered
RS PRO 6000-2RS Single Row Deep Groo	3	2.57 SGD	$ 9.20	bearings for rotation of revolver	shopee	delivered 10oct, by 15-17 oc
Servo Motor MG996R 180 360 degrees	1	13.4 SGD	$ 15.39	Launcher Feeding + locking	Shopee	delivered - 360 deg, by 9 Oct
Nema 23 Stepper motor (8mm shaft 12k	1	25.6	$ 36.59	motor for revolver mechanism	kuriosity.sg	delivered, by 12 oct, BUY 8m
Stepper motor shaft coupler (6mm * 8mr	1	6		Attach stepper motor shaft to rod	kuriosity.sg	delivered, by 12 oct, BUY 8m
Stepper motor shaft coupler (8mm * 10r	1	6	$ 10.99	correct size	kuriosity.sg	Delivered, by 17 oct
MINI HDMI Male to HDMI Female Cable	1	4.47	$ 6.40	Get display from rpi zero2w	Shopee	Delivered, by 12-15 oct, BUY
Stepper motor driver	1	7.63	$ 9.12	make stepper motor move	Shopee	Delivered, by 14-15 oct
Passive Dart battery and charger (5baer	1	28.23	$ 28.23	Lighter batteries for passive dart	Shopee	delivered 22Oct, by **28 oct** -
Lipo Rider Plus Board	3	21.48	$ 42.34	Step-up 3.7V to 5V to power the dart trigger	Element 14	Delivered, by 15 oct
~~HONGI YI FILAMENT (first purchase of~~	~~1~~	~~25.98~~	~~$ 26.97~~	~~3d printing filament~~	~~shopee~~	~~Cancelled bc parcel missing,~~
1.1g micro linear digital servo mini servo	3		$ 24.35	micro linear digital servo	TaoBao	Delivered 18 Oct
AFRC-D1015PRO large stroke linear serv	4		$ 27.29	2L + 2R micro linear	TaoBao	Delivered 18 Oct
AFRC linear motor delivery	1	19 yuan	$ 3.55	delivery		
2g servo 2.2g ultra-micro digital servo P	3	2.8 SGD	$ 12.71	servo	TaoBao	delivered on 25Oct
2g servo delivery	1	20.40 yuan	$ 3.81	delivery		
Servo JST Ultra Nano Sub C017CLS 0.050	3		$ 26.26	servo	aliexpress	Delivered, by Oct 20
10Sets JST1.25 Connector 3P Plug With	2	3.16 SGD	$ 7.17	connector wires female jst1.25	shopee	Delivered, by 14 Oct - 19 Oct
10PCS 1.25mm Connector SMD Horizon	2	1.09 SGD	$ 5.73	connectors male jst1.25	aliexpress	delivered? Oct 15 - 22
Servo Motor MG996R 180 360 degrees	1	13.4 SGD	$ 15.39	Launcher Feeding + locking (same as row 22)	Shopee	delivered, by 15 Oct - 17 Oct
Two Piece Split Retaining Ring Collar Cla	3	4.77	$ 14.68		Shopee	Delivered, 17 Oct - 19 Oct, 1
Flange Linear Motion Bushing Ball Beari	1	6	$ 7.99	flange for launcher rotary axle	Shopee	delivered, 16 Oct - 17 Oct, bu
304 Rod Stainless Steel Shaft SS304 Rou	1	7.09	$ 9.08	shaft, ie launcher rotary axle	Shopee	delivered 17oct, 17 Oct - 18
Two Piece Split Retaining Ring Collar Cla	7	4.77	$ 30.75	same as previously purchased	Shopee	delivered 23oct, by 23 Oct -
Linear Bearing Shaft Rod Bar Hardened	1	4.91	$6.90	shaft, for pusher	Shopee	delivered, by 22 Oct - 25 Oct
M1.2 screws	3	1.83	$ 5.51	mounting of servos	Aliexpress	delivered 5nov, by 4nov, m1
M2 M1.2 nuts	2	1.83	$ 4.57	mounting of servo / active dart	Aliexpress	delivered 5nov, by 4nov, M1
Small weights		?	$ 21.59	small weights for dart CG?	Shoppee	delivered 25oct, by 25-28 oc
Thermal paste		1	$ 10.30	Dart target : cooling of green led	Shoppee	delivered 30oct, by 28oct -
Radiator block		6	$ 3.94	Dart target : cooling of green led	Shoppee	delivered 28nov, by 30oct -
Filament		1	$ 13.68	Print stuff?	TaoBao	at last mile delivery as of 1/11, by 28-31 oct
delivey		38 ¥	$ 7.04			
Servo Rod	1	1.31	$ 2.47	Active dart	Aliexpress	delivered 6nov, by 3nov, M1
Servo Rod nut	1	1.92	$ 6.74	Active dart	Aliexpress	delivered 6nov, by 2nov, 1.5
Servo Rod Attachment piece	1	4.85	$ 7.10	Active dart	Aliexpress	by 5nov, M1.2
JST PH2.0 Connector Pair (set of 50)	1	$4.81	$ 6.30	3.7V battery to Boost convertor port	Shoppee	delivered 1 nov, by 2-5nov,
304 Rod Stainless Steel Shaft SS304 Rou	1	$9.80	$7.79	Launcher pusher shaft	Shopee	delivered 1nov, by 15 nov
rudder small chuck	40	¥ 0.40	$4.53	Servo Rod Attachment piece	TaoBao	
sum			$ 984.91			

Appendix M: Overall Bill of Materials