MILITECH

Autonomous Tank

R&D Project

Youssef Miri
Nicolas Consalvi
Luca Boninsegna

14 April 2025

1 Improvement of Obstacle Detection and Motorization System

1.1 Introduction

The goal of this two-month project phase was to improve the **reliability, precision, and modularity of a mobile robotic platform** by addressing key software and hardware limitations.

The work focused on two main areas:

- **Software Development**:
  - Rewrote the codebase with **modular functions for motor and sensor control**.
  - Integrated **quadrature encoders for precise, closed-loop motion control** (e.g., 90° turns).
  - Adapted the system to function with **fewer ultrasonic sensors**.
- **Hardware Redesign**:
  - Replaced tracks with **independent wheels to reduce slippage**.
  - Installed **encoder-equipped DC motors for accurate movement tracking**.
  - **Designed and 3D-printed custom shaft adapters and motor mounts**.
  - Modified the chassis to **improve sensor positioning and cable management**.

1.2 Software Development

1.2.1 Modularization and Functional Expansion

At the beginning of this project phase, the existing codebase inherited from the previous year **lacked modular structure no functions were implemented**. All logic was embedded directly within the `loop()` and `setup()` functions, making the code **difficult to read, debug, and scale**. To address this, a **complete restructuring of the software architecture** was undertaken. Dedicated functions were designed and implemented for key tasks such as **motor control, sensor measurement, encoder setup, motion control, and emergency stops**. The control system was rewritten to modularize key functions such as:

- `controleMoteur(...)` – controls motor direction via logic-level signals.

```
void controleMoteur(bool avantGauche, bool avantDroit, bool arriereGauche, bool arriereDroit) {
  digitalWrite(pin1AvantGauche, avantGauche);
  digitalWrite(pin2AvantGauche, !avantGauche);
  ...
}
```

  - **Purpose**: Controls the direction of all four motors based on boolean flags.
  - **Useful for**: Simplifies directional logic (forward, reverse, turn) by abstracting motor control into a single function.

- `measureDistance(...)` – standardizes ultrasonic distance measurements.

```
unsigned long mesureDistance(int trigPin, int echoPin) {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  return pulseIn(echoPin, HIGH) * 0.017;
}
```

  - **Purpose**: Initializes the pins connected to quadrature encoders and sets up interrupts for real-time tracking of wheel rotation.
  - **Useful for**: Measuring distance traveled and direction of each wheel precisely — enabling closed-loop motion control.

- `arretMoteurs()` – halts all motor activity for safety and control.

```
void arretMoteurs() {
  digitalWrite(pin1AvantGauche, LOW);
  digitalWrite(pin2AvantGauche, LOW);
  ...
  analogWrite(moteurArriereGauche, 0);
}
```

  - **Purpose**: Immediately stops all motors.
  - **Useful for**: Safety, halting the robot when an obstacle is too close or an error occurs.

- `encodeurSetup()` and `resetEncoderPositions()` – manage the initialization and feedback of quadrature encoders.

```
void encodeurSetup() {
  pinMode(encArriereDroitA, INPUT_PULLUP);
  pinMode(encArriereDroitB, INPUT_PULLUP);
  ...
  attachInterrupt(digitalPinToInterrupt(encArriereDroitA), [](){
    posArriereDroit += digitalRead(encArriereDroitB) ? -1 : 1;
  }, RISING);
}
```

  - **Purpose**: Initializes the pins connected to quadrature encoders and sets up interrupts for real-time tracking of wheel rotation.
  - **Useful for**: Measuring distance traveled and direction of each wheel precisely — enabling closed-loop motion control.

```
void resetEncoderPositions() {
  posArriereDroit = 0;
  posArriereGauche = 0;
  ...
}
```

  - **Purpose**: Resets the encoder position counters to zero.
  - **Useful for**: Starting a new movement or rotation with a clean baseline, crucial for accurate rotations.

- `tournerAGauche()` and `tournerADroite()` – enable precise 90-degree rotations using real-time encoder data.

```
void tournerADroite() {
  resetEncoderPositions();
  controleMoteur(HIGH, LOW, HIGH, LOW);
  while (true) {
    long sum = abs(posArriereDroit) + abs(posAvantGauche) + ...;
    if (sum > threshold) break;
  }
}
```

  - **Purpose**: Executes a right or left turn based on encoder feedback.
  - **Useful for**: Controlled, precise in-place turning using real-time encoder data — critical for navigation in tight spaces.
  - (`void tournerAGauche()` is the same but for turning the robot to the left).

Full Code Name in Github: `OverhaulDetectionObstacle.ino` See Video on Github of the obstacle avoidance : `DetectionObstacle2025Demo.mp4`

1.2.2 Precision Motion Using Encoders

A specific code was developed to allow the robot to rotate exactly 90 degrees, using real-time data from quadrature encoders. This was tested and refined to allow successive 90° turns with high repeatability, a key capability for grid-based navigation or structured indoor environments. Code name : `EncodeurRotationSuccessive.ino` See video on Github : `encodeursrotationDemo.mp4`

1.3 Hardware Improvements

1.3.1 Transition from Tracks to Wheels

Originally, the robot was equipped with continuous tracks, which exhibited significant slippage, especially during rotational motion and uneven terrain transitions. This negatively affected path accuracy and obstacle avoidance. Modifications Implemented:

- Replaced tracks with independent wheels for improved traction and reduced inertial drift.
- Introduced differential drive mechanics to allow for precise in-place rotation.

1.3.2 Motor Replacement and Encoder Integration

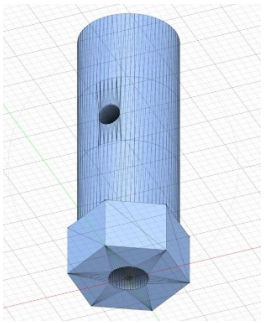
To enhance motion precision:

- Existing motors were replaced with DC motors equipped with quadrature encoders.
- Encoder feedback is now used in the software loop to compute displacement and heading.

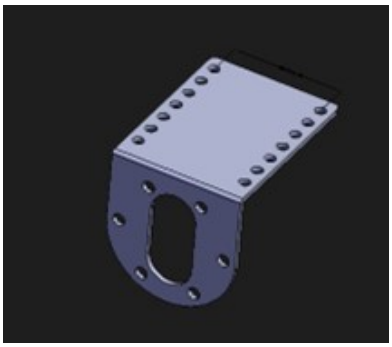
1.3.3 Custom Mechanical Adaptation

To support the hardware transition:

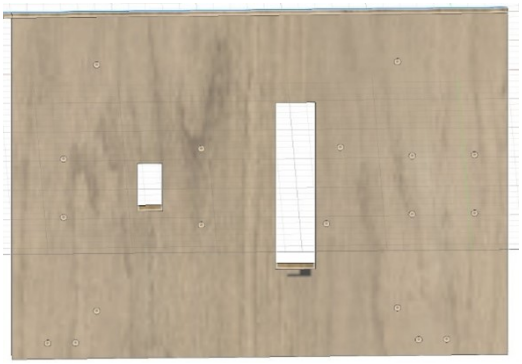
- 3D-printed shaft adapters were designed to connect the new motors with the wheels.



- Motor mounts and support brackets were printed to secure the motors firmly to the chassis.



- The chassis layout was redesigned:
  - Repositioned internal components to optimize cable routing and minimize electromagnetic interference.
  - Ensured better weight distribution for mechanical stability.



1.4 Outcome

- The integration of encoders allows closed-loop control, reducing drift and overshooting during navigation.
- The robot can now reliably turn 90 degrees with repeatable accuracy, essential for grid navigation and autonomous pathfinding.
- Hardware modifications have significantly increased system robustness and reduced maintenance complexity.

1.5 Challenges Encountered and Engineering Solutions

During development, several issues emerged that affected the robot's performance and durability. The main problems and solutions are summarized below.

1.5.1 Sensor Reduction and Repositioning

Reducing the number of ultrasonic sensors from 7 to 5 led to decreased environmental awareness and detection blind spots. The switch to wheels also forced suboptimal sensor placement. **Solution**: A chassis redesign was proposed to reposition wheels and relocate sensors for better coverage and flexibility.

1.5.2 Wear of 3D-Printed Parts

Over time, 3D-printed shaft adapters wore out, causing motor shafts to spin without driving the wheels. Motor mounts also deformed, leading to misalignment and desynchronization. **Solution**: Short-term reinforcement of printed parts; long-term plan to replace key components with machined metal for durability and precision.

1.5.3 Ultrasonic Sensor Limitations

Ultrasonic sensors showed poor angular resolution and inconsistent readings, particularly in tight or soft-surfaced environments. **Solution**: Consider upgrading to Time-of-Flight (ToF) sensors for higher accuracy, faster response, and better obstacle detection.

1.5.4 Cable Management

Loose and tangled cables caused frequent disconnections and debugging difficulties. **Solution**: Redesign the wiring layout using sleeves, locking connectors, and routing guides. A custom PCB is proposed to improve reliability and reduce wiring complexity.

1.6 Resource Investment and Cost Assessment

1.6.1 Financial Costs

As part of the redesign and hardware integration phase, the following components were purchased or fabricated to support the transition from the previous tracked design to a wheel-based encoder-driven configuration:

Item	Quantity	Unit Price (€)	Total (€)
DC Motors with Integrated Encoders	4	60	240
Wooden Components for Chassis Redesign	–	–	10
Total	–	–	250

These costs reflect the hardware expenses specifically related to your contribution to the robotic system.

1.6.2 Time Investment

Over the course of this two-month period, the project required substantial time commitment, both during supervised lab sessions and through personal development work:

- **In-class development time**: 9 sessions $\times$ 4 hours = 36 hours
- **Personal work and experimentation** (coding, hardware adaptation, 3D design, testing): $\approx$ 100 hours
- **Total time investment**: $\sim$136 hours

This time was allocated across multiple domains:

- Hardware integration (mechanical design, 3D printing, assembly)
- Software development (motor control, encoder logic, obstacle avoidance)
- Troubleshooting and performance validation

The time and cost reflect a comprehensive, hands-on engineering effort, combining both design and implementation responsibilities.

Github link

`https://github.com/YoussefMiriXX/Militech-Project`

2 Gears' system and turret

2.1 Introduction

This report details the design, implementation, and ongoing development of an automated robot tank equipped with a rotating turret and laser-based targeting system. The project represents an integration of mechanical engineering principles, custom 3D-printed components, and electronic systems to create a functional mobile platform with target acquisition capabilities.

The robot features a wheeled chassis for mobility, a rotating turret system driven by a step motor through a custom gear mechanism, and a laser-based targeting system that serves as

a safer alternative to an actual electronic cannon. This document will cover the mechanical challenges encountered during development, the solutions implemented, and the remaining issues to be addressed in future iterations.

2.2 System Overview

2.2.1 Base Platform

The foundation of the project is a tank-style chassis equipped with wheels for mobility. This platform serves as the mounting point for all other components, including the drive system, electronics, and turret assembly. The base is designed to provide stability during movement and targeting operations while accommodating the necessary electronic components.

2.2.2 Turret Assembly

The turret represents the most complex mechanical component of the system. It features: A custom 3D-printed housing, Rotation capabilities via a gear system, An integrated laser sensor for target detection, Mounting provisions for potential future weaponry.

The turret connects to the base through a gear mechanism that allows for precise rotational control, enabling the system to track and engage targets within its operational range.

2.2.3 Electronic Components

The electronic subsystem consists of: Arduino microcontroller board for system control, Step motor for driving the turret rotation, Laser sensor for target acquisition, Associated wiring and power management components

The Arduino serves as the central control unit, processing sensor inputs and controlling the step motor to position the turret as required.

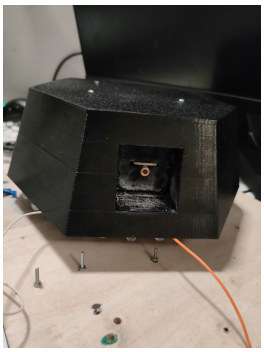


Figure 2.1: Robot tank prototype with rotating turret and laser targeting system

2.3 Mechanical Design Challenges and Solutions

2.3.1 Gear System Design

The rotational mechanism for the turret presented significant mechanical challenges. The system employs two primary gears:

1. **Small Gear (Driver):** Directly connected to the step motor, responsible for transferring rotational force to the large gear, initially experienced issues with axial rotation relative to the motor shaft

2. **Large Gear (Driven):** Connected to the turret assembly, requires smooth rotation with minimal friction, bears the weight and operational forces of the turret components

2.3.2 Small Gear Adapter Solution

The initial configuration of the small gear presented a significant challenge, as it rotated freely with the motor axis rather than transmitting torque effectively. This rotation compromised the precision required for accurate turret positioning.

To address this issue, a custom bearing adapter was designed and 3D-printed with the following features:

- Gear-shaped perimeter with notched edges that securely interface with the small gear
- Central mounting mechanism to attach firmly to the motor shaft
- Appropriate tolerance design to ensure proper mechanical coupling

This adapter effectively eliminated the undesired independent movement, ensuring that the gear rotates coaxially with the motor shaft. This modification significantly improved the system's ability to precisely control turret position.

Figure 2.2: Custom 3D-printed adapter for the small gear

2.3.3 Large Gear and Rotational Axis Implementation

For the large gear component, the design objective differed significantly. Rather than restricting rotation, the goal was to facilitate smooth movement while maintaining proper alignment with the drive system.

The implemented solution included:

- A custom-designed rotational axis for mounting the large gear
- A specifically engineered bearing adapter to accommodate a standard bearing assembly
- Proper alignment features to ensure concentricity with the rotation axis

The integration of a standard bearing into this custom housing significantly reduced rotational friction, allowing the turret to move smoothly even under varying load conditions. The wear patterns observed on the bearing surfaces indicate proper function and load distribution, validating the design approach.

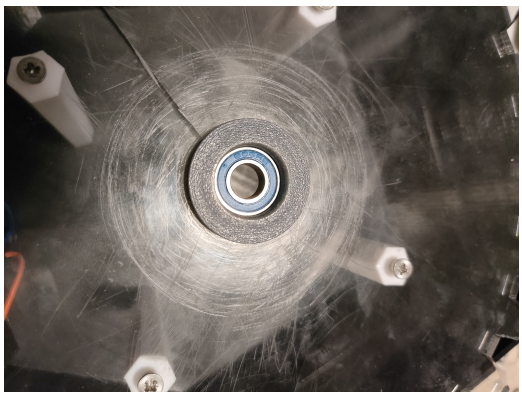


Figure 2.3: Bearing assembly for the large gear

Figure 2.4: Rotation axis for the big gear

Custom 3D-printed components for the big gear

2.4 Turret Integration and Target Acquisition System

2.4.1 Turret Design Considerations

The turret assembly was designed with several critical requirements in mind:

- Stable platform for the laser targeting system
- Sufficient internal space for potential future electronic components
- Balanced weight distribution to minimize strain on the rotation mechanism
- Appropriate apertures for the laser sensor

Through iterative design and testing, the final turret configuration successfully balances these requirements while maintaining functional compatibility with the gear system.

2.4.2 Laser System Implementation

Instead of implementing an actual electronic cannon, which would present safety concerns, the project utilizes a laser-based targeting system. This approach offers several advantages:

- Enhanced safety during development and demonstration
- Reduced power requirements
- Simplified mechanical integration
- Comparable targeting accuracy for demonstration purposes

The laser sensor is precisely positioned within a custom-designed housing in the turret, allowing for accurate target detection within the system's operational range.

2.4.3 System Integration

The integration of the mechanical components with the electronic systems presented its own set of challenges. Particular attention was paid to:

- Mechanical tolerances to ensure proper fit between components
- Stability of mounting connections to prevent unwanted movement
- Appropriate clearances for operational movement

- Access provisions for maintenance and modifications

The current implementation successfully addresses these integration considerations, resulting in a functional prototype capable of demonstrating the core capabilities of the system.

2.5 Current Limitations and Future Work

2.5.1 Cable Management Challenges

A significant ongoing challenge relates to cable management for the turret's electronic components. Currently, the laser system in the turret requires a cable connection to the Arduino board, which is mounted on the fixed part of the robot. This arrangement presents several issues:

- Potential for cable tangling during rotation
- Limitations on continuous rotation capabilities
- Risk of connection failures due to repeated movement stress
- Aesthetic and functional compromises

2.5.2 Proposed Solutions for Cable Management

Several approaches are being considered to address the cable management issue:

1. **Slip Ring Implementation:** Installing a slip ring would allow for continuous rotation while maintaining electrical connections.
2. **Onboard Electronics:** Relocating certain electronic components to the turret itself could reduce the number of required connections.
3. **Wireless Communication:** Implementing wireless data transmission between the turret and base could eliminate some cable requirements.
4. **Cable Routing Optimization:** Designing a more effective cable path with appropriate strain relief could mitigate some current limitations.

2.5.3 Additional Future Improvements

Beyond addressing the cable management issues, several other enhancements are being considered for future iterations:

- Improved power management system
- Enhanced targeting algorithm implementation
- Addition of distance sensing capabilities
- Implementation of autonomous navigation features
- Further refinement of the 3D-printed components for increased durability

2.6 Conclusion

The robot tank project has successfully integrated mechanical design, 3D printing technology, and electronic systems to create a functional prototype with target acquisition capabilities. The custom-designed bearing adapters and rotational components have effectively solved the initial challenges related to gear rotation and system integration.

The current implementation provides a solid foundation for future enhancements, particularly in addressing the cable management challenges. The modular approach to design allows for iterative improvements without requiring complete system redesign.

This project demonstrates the effectiveness of targeted mechanical solutions in addressing specific operational challenges. The combination of standard components with custom-designed interfaces illustrates a practical approach to prototype development that balances functionality, cost, and implementation complexity.

3 The Main Objective of the Jetson

As part of this project, **object detection was carried out using the Jetson Nano and the YOLOv8 algorithm**. The Jetson Nano, developed by NVIDIA, is a small but powerful computing board designed specifically for artificial intelligence (AI) applications and real-time image processing. YOLOv8, short for "You Only Look Once," is a real-time object detection algorithm known for its speed and efficiency. It enables the detection and classification of various objects in a scene in a single pass, thus offering optimal performance for applications such as video surveillance, autonomous driving, and many others.

The main objective of using the Jetson Nano combined with YOLOv8 was to perform **real-time detection of a specific target in each environment**. Once the target was detected, the Jetson Nano was expected to be able to communicate with other devices to trigger specific actions based on the detection results.

In this project, the Jetson Nano was configured to communicate with an **Arduino Uno board via a serial communication protocol**. The Arduino Uno, a popular electronic prototyping platform, was used to control various peripherals and perform actions in response to signals from the Jetson Nano.

For example, once the target was detected by the Jetson Nano, the coordinates or other relevant information could be transmitted to the Arduino Uno via the serial link. The Arduino Uno could then use this data to activate actuators such as motors, lights, or other devices, depending on the specific application scenario.

This integration between the Jetson Nano and the Arduino Uno enables **intelligent and responsive interaction between object detection and device control**, paving the way for a multitude of practical applications.

3.1 Machine State

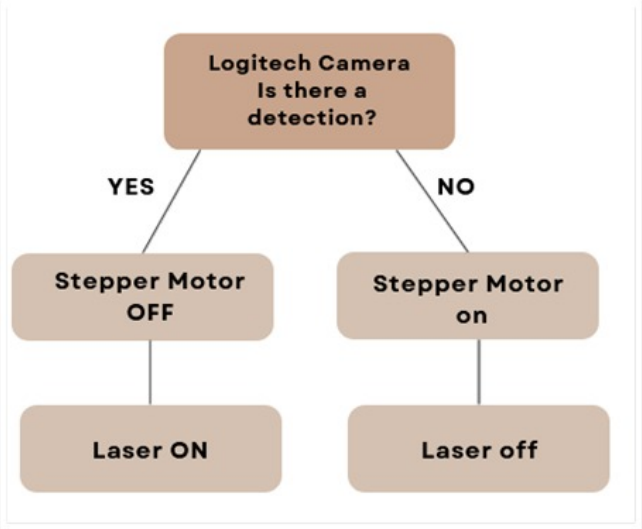
3.1.1 Explanation of the Algorithm

The system begins with the Logitech camera, which attempts to detect a target. If no target is detected, the stepper motor continues to rotate in search of the target.

When a target is detected, the stepper motor stops, and the laser and the two LEDs turn on.

After detection, the system resets: the LEDs and laser turn off, ready for a new detection and launch cycle.

In summary, **continuous communication exists between the Jetson, the camera,**



and the Arduino to ensure accurate target detection and activation of the laser.

3.2 Progress of Part 2

3.2.1 Preparation of the Working Environment

Download and Installation of the System Image

To begin, I downloaded a system image optimized for the Jetson from Q-Engineering. This image uses Ubuntu 20.04 and supports several libraries such as PyTorch and OpenCV, compiled with CUDA to utilize the Jetson's GPU, which allows for good performance.

```
Platform
 Machine: aarch64
 System: Linux
 Distribution: Ubuntu 20.04 focal
 Release: 4.9.253-tegra
 Python: 3.8.10

Libraries
 CUDA: 10.2.300
 cuDNN: 8.2.1.32
 TensorRT: 8.0.1.6
 VPI: 1.1.15
 Vulkan: 1.2.141
 OpenCV: 4.8.0 with CUDA: YES
```

Initial Performance Test

After the installation, I tested the performance using YOLOv8n, the fastest version of YOLO, and achieved a rate of 13 FPS (frames per second). To improve this performance, I decided to export the model in FP32 and FP16 formats, and I found that the FP16 format provided better performance, reaching 17 FPS.

Installation of Required Libraries

To do this, I downloaded and installed Ultralytics using the command `pip3 install ultralytics`. Then, I removed the default version of OpenCV because the new version was not compiled with CUDA, which is crucial for performance on the Jetson.

3.2.2 Data Preparation

Data Collection and Preparation

To train my own model, I found a dataset containing several targets, already annotated, which greatly simplifies the training process. The presence of annotations is essential as it clearly defines the objects to be detected in each image, thereby optimizing the training process. **Roboflow Universe Link:** `https://universe.roboflow.com/`

3.2.3 Model Training

Using Google Collab

To train the model quickly, I used Google Collab, which provides access to powerful GPUs such as the Tesla A100. This GPU, which costs around 8000 euros, significantly accelerated the training process. Here is the link to the Collab used: `https://colab.research.google.com/github/roboflow-ai/notebooks/blob/main/notebooks/train-yolov8-object-detection-on-custo` `ipynb`

Training Parameters

I trained the model on **800 images, with the number of epochs set to 100**. This means the model went through the 800 images 100 times. It's important not to increase the number of epochs excessively to avoid overfitting. Overfitting happens when the model learns the details and noise of the training set too well, which harms its performance on new data.

Training Result

At the end of the training, a file named `best.pt` was generated. This file contains the model's optimized weights and was exported in **FP16 engine format for optimal performance on the Jetson**.

3.2.4 Use of the Trained Model

Implementation and Testing on Jetson

With the model training completed, I used the `best.pt` file exported in FP16 format for final tests on the Jetson. The model demonstrated satisfactory performance, confirming its effectiveness in object detection tasks.

3.2.5 Integration with Arduino

Using PyFirmata for Communication

To facilitate communication with the Arduino, I chose to use Python as the programming language. This is made possible through the **PyFirmata library**. PyFirmata allows an Arduino board to be controlled directly from Python, greatly simplifying the integration process. PyFirmata is easy to install and use, with examples available in the library's installation files.

Arduino Configuration

I configured the Arduino using PyFirmata to enable efficient serial communication with the Jetson. On the Arduino, I uploaded the **Firmata firmware**, which allows the Arduino to receive and execute commands sent from a Python script.

Development of Python Scripts

On the Jetson, I created two Python files: `main.py` and `control.py`.

- **main.py**: This file initializes the AI model and starts the target detection process. When objects are detected with sufficient confidence, the information is sent to `control.py`.
- **control.py**: This file receives the data from `main.py` and controls the external components connected to the Arduino, such as LEDs, laser, and stepper motor.

Establishing Serial Communication

I set up **serial communication between the Jetson and the Arduino via a USB port**. This ensures fast and reliable data transmission, which is essential for real-time control of external components.

This part has reached a **100% maturity level**, with a fully ready and operational solution. All planned activities were successfully completed within the given deadlines, demonstrating a generally satisfactory level of progress.

3.3 Year-over-Year Project Improvements

The system has seen notable advancements compared to last year's version. The object detection model was **upgraded from the standard YOLO to YOLOv8 trained on a custom dataset, significantly improving detection accuracy**. In terms of motor control, the old step-based movement was replaced by **continuous rotation, offering smoother and more precise tracking of targets**.

On the safety front, the shift from an electromagnetic cannon to a **laser pointer marks an improvement in reducing risk and enhancing operational safety**. Finally, the dataset used for training was greatly enhanced—transitioning from a small, general dataset to a large and diverse one, complemented by data augmentation, which boosted model performance and reliability.

3.4 Planned System Improvements for Next Year

To enhance performance and system efficiency, the following upgrades are planned for the upcoming year:

- **Switch to Orin Nano**: Replacing the Jetson Nano with the Orin Nano, offering 30x more processing power, will allow faster AI model inference and better real-time performance.

- **Dual Arduino Setup**: Introducing two Arduino units—one for motor control and one for laser control—will reduce system complexity and improve cable management.
- **LIDAR Integration with ROS**: Integrating LIDAR with the Robot Operating System (ROS) will enable more accurate object detection, mapping, and improved environmental awareness.
- **Sensor Fusion**: A sensor fusion system will be developed to combine inputs from lidar, camera, and ultrasonic sensors, enhancing perception and decision-making.

These improvements aim to significantly boost system reliability, intelligence, and responsiveness.