NATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY "POLITEHNICA" BUCHAREST
FACULTY OF ENGINEERING IN FOREIGN LANGUAGES
COMPUTERS AND INFORMATION TECHNOLOGY - INFORMATION ENGINEERING

GRADUATION PROJECT

Coordinator
Prof. Dr.Ing Bujorel PĂVĂLOIU

Student:
Petru MUȘAT

Bucharest
2024

NATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY "POLITEHNICA" BUCHAREST
FACULTY OF ENGINEERING IN FOREIGN LANGUAGES
COMPUTERS AND INFORMATION TECHNOLOGY - INFORMATION ENGINEERING

Automated Targeting Device: Remote controls, Mobile application, Web Server

Coordinator

Prof. Dr.Ing Bujorel PĂVĂLOIU

Student:

Petru MUȘAT

Bucharest

2024

NATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY
"POLITEHNICA" BUCHAREST
FACULTY OF ENGINEERING IN FOREIGN LANGUAGES
INFORMATION ENGINEERING

Approved
Dean:
Prof. dr. ing. Cristian DRAGOMIRESCU

DIPLOMA PROJECT THEME FOR:
Petru MUȘAT

1. Theme title (RO/EN):
Dispozitiv automat de ochire – Control de la distanță, aplicație mobilă, server Web
Automated targeting Device – Remote Controls, Mobile Application, Web Server

2. Initial design data:
The goal of this project is to design a compact automated targeting system/ turret, similar to the military and the entertainment ones. A physical device, using an adapted airsoft gun, will be developed as a prototype and operated for testing.

3. Student contribution:
Design and implementation of the mechanical/electronic circuit
Implementation of the remote-control component
Development of the mobile app to control the turret from a distance and watch the camera footage.
Creation of a web server to communicate directly with the device/turret

4. Compulsory graphical material:
Design scheme, Electronic devices layout, Screenshots of the code

5. The paper is based on the knowledge obtained at the following study courses:
Internet of Things, Microprocessor architecture, Neural Networks and Genetic Algorithms, Application Development for Mobile Devices

6. Paper development environment:
Arduino IDE (C) for the device, Visual Studio 2022 (C#) for webserver, Android Studio (Java) for mobile application.

7. The paper serves as:
Research/Development

8. Paper preparation date:
June 2024

Project coordinator	**Student:**
Prof. Dr.Ing Bujor PĂVĂLOIU	**Petru MUȘAT**

DECLARATION OF ORIGINALITY OF THE GRADUATION PROJECT[1)]

I, the undersigned _____[2)],
owner of

B.I./C.I./passport series _____, no. _____, with CNP _____ have carried out the graduation project with the title:[3)] _____ under the supervision of _____, in view of the examination for finalizing the university bachelor level studies, organized by the Faculty of Engineering in Foreign Languages, the Department _____, of the National University of Science and Technology POLITEHNICA Bucharest, session _____, academic year _____.

Taking into consideration the content of art. 143 in the National Education Law no.1/2011, indent. (4) *"The graduation projects/dissertation supervisors are jointly and severely held responsible with the authors for ensuring the originality of the content"* and of indent. (5) *"The selling of scientific projects in view of facilitating the forgery by the buyer of the capacity as an author of a graduation project is forbidden",* I hereby declare on my own responsibility that this project is original, being the result of my own intellectual activity, it does not comprise plagiarized parts, and the bibliographic resources were used with the observance of the legislation in force.

I am aware of the fact that plagiarism or the presentation of a project carried out by another graduate, or taken from the Internet, from course books and books, without the mention of the source, represents a crime (intellectual theft and breaching of the copyright and violation of intellectual property) and it is conducive to the annulment of the bachelor level examination.

Date:_____ Signature:
_____

(in the original – handwritten)

[1)]The declaration shall be completed by handwriting and it represents a document that shall be submitted upon enrolment for the defending of the examination for finalizing of the studies.
[2)]Surname, father's first name initial, and first name, in block caps.
[3)]Title of the graduation project, in block caps.

Table of Contents

1 Introduction

1.1 The Airsoft Sport

Since the first time it was played in 1970 as a military training tool, airsoft matches have become increasingly popular as a recreational sport for people who want to feel the thrill of simulated combat. Unlike any other sport however, airsoft offers a unique mix between strategy and physical activity.

The rise of the internet enabled the emergence of online groups, enabling enthusiasts to freely coordinate events, engage in discussions about equipment, and exchange their personal experiences. Players seeking greater levels of challenge engage in MilSim events, meant to replicate actual military situations. The culture associated with this sport revolves around the players' collective passion for adrenaline and realism, leading them to invest in high-quality, costly equipment, accessories, and authentic outfits to enhance their immersion on the field.

Thanks to the constant advancement of technology, airsoft replicas are gradually becoming more and more realistic. Gas-powered and electric models have been manufactured that faithfully imitate the feel, weight, and functionality of an authentic firearm. Airsoft consistently keeps pace with advancements in military equipment. [1]

1.2 Sentry Turrets

Over the past couple of decades, automated targeting systems have started to be deployed in military applications due to their precision and efficiency. These systems are designed to detect, track, and engage targets with minimal human intervention, offering significant advantages in terms of speed, accuracy, and safety. In military contexts, automated turrets are used for perimeter defense, surveillance, and combat scenarios, where rapid response and precision are critical. Their main advantage is that, whether remotely controlled or fully automated, they are able to scan and defend an area without the risk of any human casualties. The earliest functioning ones are called Close-In Weapons Systems and were used to detect and destroy incoming missiles or aircraft.

Fig 1.1 The Phalanx CIWS

In 2007, the South Korean company Samsung Techwin (now known as Hanwa Vision) created the sentry turret SGR-A1. The objective of this was to completely replace human security personnel in the Korean DMZ and ensure a "perfect guarding operation". It is regarded as the first sentry that brings together surveillance, target tracking, automated firing, and voice recognition. [2]

However, the primary drawback of such a system, and the reason for its limited use in military operations, is its immobility. That renders them susceptible to be destroyed by hostile artillery or mortars. Another notable problem is their high cost, both the turret itself and of the ammunition it requires. Finally, these devices still necessitate human intervention to be reloaded after consuming their ammunition.

1.3 Objective of the thesis

I decided to combine the previous two topics and develop a more compact and easier to use automated targeting device to be used in airsoft matches. The sentry turret is built from an airsoft Taurus P92 handgun as base, controlled using Raspberry Pi Pico WH, 3 servomotors for the movement and firing functions and an ESP32 Cam to provide live video feed all of the mentioned components were encapsulated in a self-designed 3d printed chassis so the device can function properly and seamlessly. The software components it features are a mobile application used to remotely control the device as well as a server to bring the digital components of the project all together.

The mobile application was specifically designed to be user friendly and the hardware components were carefully picked to provide autonomy to the device. The Raspberry Pi was the best solution to remotely control the movement and shooting functions from the virtual joystick and button implemented in the app.

The device was first intended as a project for the 3rd year course we took in Internet of Things, where we were encouraged to come up with automated projects for ourselves. It was driven both by my love of robotics and by my countless hours spent in airsoft matches. When tasked with defending an objective the user would no longer need to expose themselves to enemy fire and be able to operate the device from outside the area that needs to be protected. It would give the opposing team the tough decision of either advancing through the area without being detected by the turret in order to turn it off, or to actively search for the device's operator, thus losing precious time in the round.

As a competitor in this sport, I intended to develop a fresh and unfamiliar challenge for the opponents to engage in and ultimately try to overcome, thus introducing an additional layer of excitement into the already captivating matches.

1.4 Structure of the Thesis

The structure of this theoretical paper is to present in a logical and approachable manner the advantages of my airsoft targeting device, which can autonomously function, as well as how this idea came to existence in a physical, usable form. This project was constructed from the ground up with a specific idea in mind, because of its ability to provide the user with live camera footage and promptly react to their commands makes it highly effective at operating as an area defense system.

The rest of the document is structured in the following order, sectioned in 8 parts:

- An introduction, regarding the proposed application of the device
- The architecture of the microcontrollers
- The state of the art
- The technological choices related to my project
- Details about the implementation of the project
- Testing and trial runs
- Conclusions

1.5 Problem and Purposed Application of the Device

Since we have already talked about airsoft and sentry turrets in general prior to this chapter, I will briefly summarize the purpose of the device I have developed.

As a constant participant in airsoft matches, I got invested into purchasing quality gear to better enhance my experience. From protection equipment to better replicas and better attachments, I wanted to provide myself with an authentic tactical experience. But no matter gear, one is meant to lose when they are the last left standing against multiple enemies.

So I decided to develop a device that would be able to help the last teammate while providing the enemy team with a quite complicated and fun challenge. The automated targeting device, once deployed, would be able to defend its surrounding area, either remotely controlled by the player, or automatically scanning and engaging targets. All of this was possible to achieve with the help of 2 microcontrollers, 3 servomotors and a power supply.

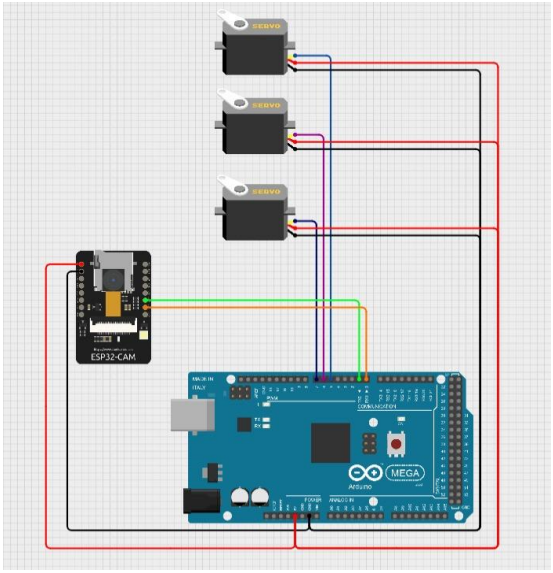

Fig. 1.2 Initial idea of the device, using an Arduino Mega instead of the Raspberry PI

The only thing left to do was to bring everything together and make this project into a reality by picking the required hardware parts and develop the mobile application, the controls and the server.

2 State of the Art

2.1 Arduino Mega 2560

Initially, we made the decision to use an Arduino board in order to control the device's movement and firing functions. We could have done it with an Arduino Uno too, but I already owned an Arduino Mega and decided I will use it for this project.

Built on the ATmega2560, the Arduino Mega 2560 is a microcontroller board. Designed for more complex applications requiring more input/output pins and higher memory capacity than smaller boards such as the Arduino Uno. An external power source ranging in voltage from 7 to 12 volts or a USB connection can both drive it. It run at five volts. The board had 16 analog input pins and 54 digital I/O pins, 15 of which could generate pulse width modulation (PWM). Its clock speed is 16 MHz; it features 256 KB of flash memory, 8 KB of SRAM, and 4 KB of EEPROM.

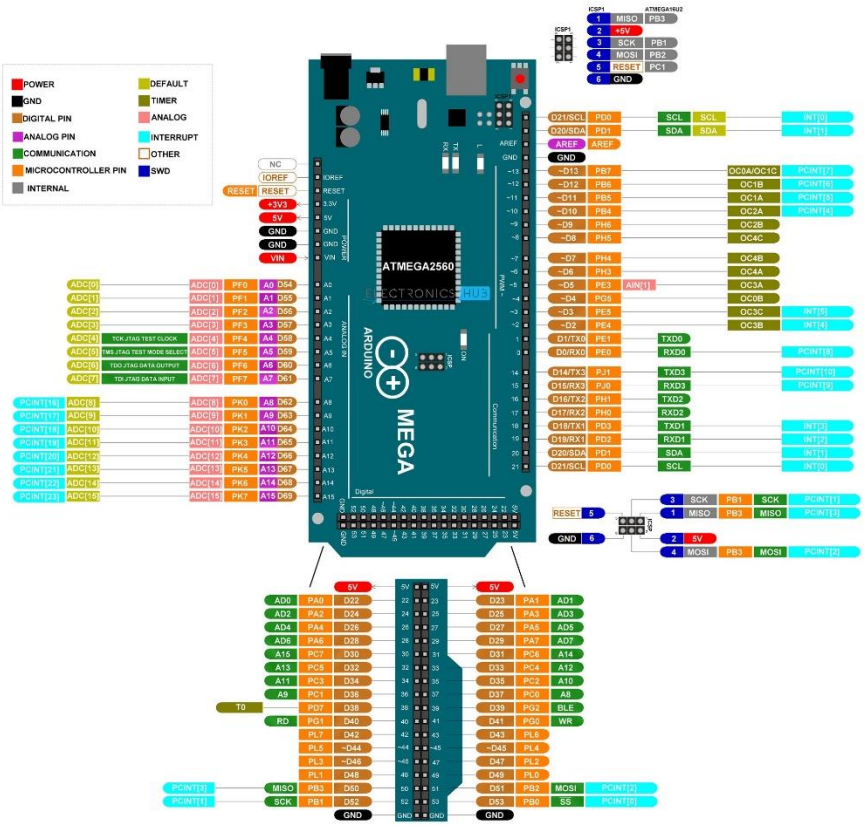

Fig 2.1 Example of Pinout for the Arduino Mega 2560 [3]

Among other noteworthy characteristics of the Arduino Mega 2560 are USB connectivity, a reset button, and an ICSP header. The gadget also has a power port for tying in

outside power sources. Internal embedded LEDs show serial communication methods and power. Along with I2C and SPI interfaces for communication, the board features four hardware serial ports—commonly known as UARTs. It can also manage PWM operations and events at exact timings by supporting several timers, therefore enabling precision timing.

The Arduino Mega 2560 chip is programmed using the C/C++ supporting Arduino Software (IDE). Designed with pre-configured libraries and sample drawings to enable users get started fast, the integrated development environment (IDE) offers a simple-to-use development environment. Users must first download and install the Arduino IDE, then connect the board to their computer via USB cable, choose the suitable board and port inside the IDE, write their code, and lastly upload it to the board in order to efficiently program the board.

Applications for the Arduino Mega 2560 are robotics, home automation, data logging, and interactive installations among others. Applications needing a lot of sensors, actuators, and complex processing operations will find great value in its complete I/O capabilities and significant memory capacity.

2.2 Related work using Arduino:

A similar project to my Automated Targeting Device is the remote controlled Bluetooth Nerf Turret. I stumbled upon this project, right after coming up with an idea myself for my bachelor's thesis.

Fig 2.2 Little_french_kev's Bluetooth Nerf Turret

The person behind this project aimed to create a simple, remote controlled turret that shoots Nerf foam darts and is controlled via either an Android device or a computer.[4]

The tilt and pan motions are controlled by two servos powered by an Arduino Nano. The darts are fired using two rollers powered by two small DC motors. When the shot command is delivered, a MG996r servo is responsible for propelling the darts into the rollers. The majority of the project is 3D Printed featuring a magazine that can hold up to seven darts.

2.3 Raspberry Pi Hardware

The Raspberry Pi Pico WH is equipped with an RP2040 microcontroller using a dual-core ARM Cortex-M0+ processor running at a frequency of 133 MHz. It has 264 KB of SRAM and 2 MB of onboard QSPI Flash memory. This structure enables it to handle fundamental activities such as LED blinking and intricate data processing duties. Along with an extra 3-pin debug port, the "WH" edition features a Wi-Fi module.

The Pico WH operates at 3.3V and can be powered either a micro USB port (5V) or the VSYS pin (1.8-5.5V). It contains 26 multi-function GPIO pins for digital input/output and supports a range of communication protocols such as I2C, SPI, and UART. It also has three analog input ports for interacting with analog sensors and a single user-controllable LED. The board supports USB 1.1 device and host communication, making it excellent for peripheral connections and data transfers. Programming the Raspberry Pi Pico WH is straightforward and can be done using MicroPython.

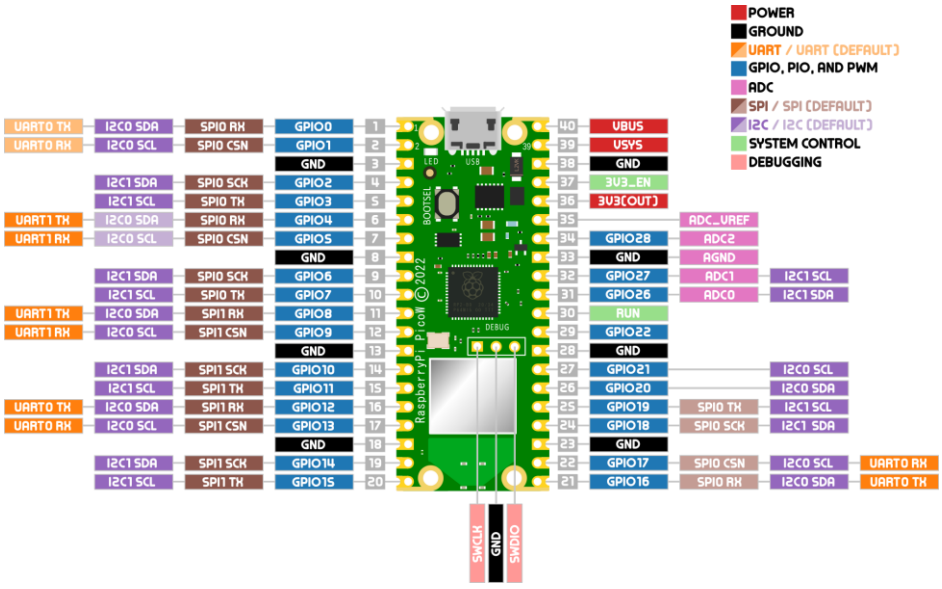

Fig 2.3 Example of Pinout for the Raspberry Pi Pico WH [5]

The Pico WH may be used in several applications, including embedded systems, Internet of Things projects, educational projects, robotics, and rapid prototyping. The system's responsiveness makes it well-suited for controlling the servomotors in the Automated Targeting System and for establishing communication with the server.

Related work using Raspberry Pi:

Inspired by an all-wheel drive toy car from the 1980s called the Stomper, the StoRPer modular DIY robot is a remote controlled rover based on a Raspberry Pi Pico used with custom designed printed circuit board (PCB).[6]

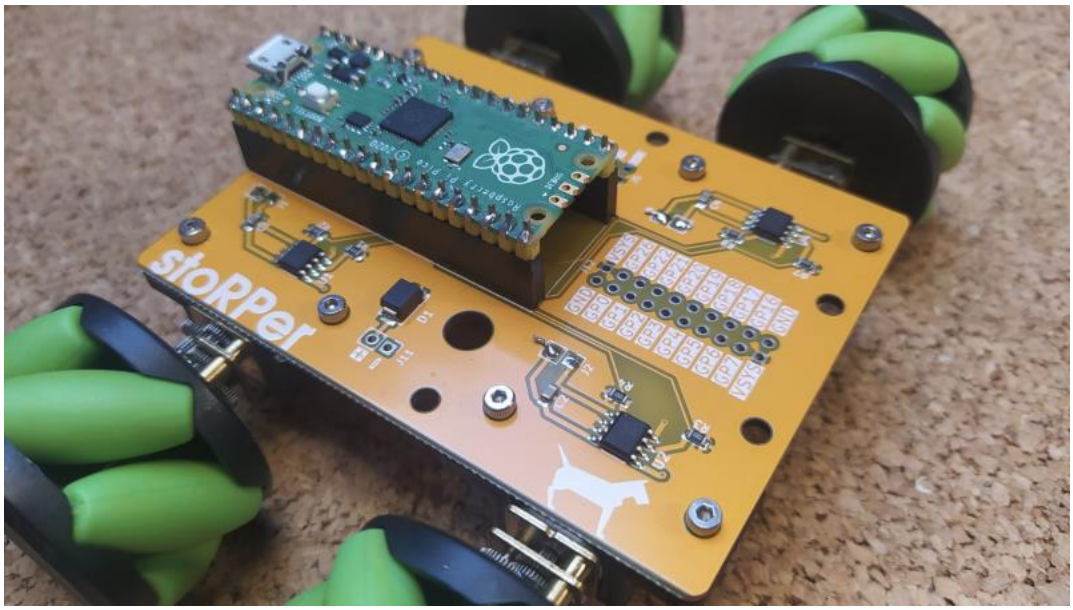

Fig 2.4 The StoRPper modular rover

The chassis of the StoRPer consists of a PCB that has four motor driver circuits. These circuits are specifically developed to work with the L9110S drive integrated circuit, which is both affordable and highly efficient. This machine can be constructed as a typical 4-wheel drive vehicle. However, it is also straightforward to incorporate a mecanum wheel, allowing for the exploration of highly intriguing movements and motions.

2.4 ESP32 hardware

Specifically made for camera-based projects and IoT applications, the ESP32-CAM microcontroller board It is reasonably priced and little in stature. This gadget, which runs the ESP32 chip, has a dual-core Tensilica LX6 CPU, which offers enough of processing to manage demanding chores. It also offers a wide range of connectivity options, including built-in Wi-Fi and Bluetooth.

The board can be useful for effectively managing image processing and other chores demanding a lot of memory. It is equipped with 4 MB of PSRAM and running at 3.3V. The OV2640 camera module fitted into the ESP32-CAM is able to record images with a maximum resolution of 1600x1200 pixels

The ESP32-CAM is well-known for its several networking features. Remote surveillance, wireless imagine transmission, and smart home applications—among other Internet of Things (IoT) uses—all fit the board's integrated Wi-Fi and Bluetooth features. These features let the board interact with other devices and connect with wireless networks. The board features a multitude of GPIO pins and offers support for UART, SPI, and I2C interfaces.

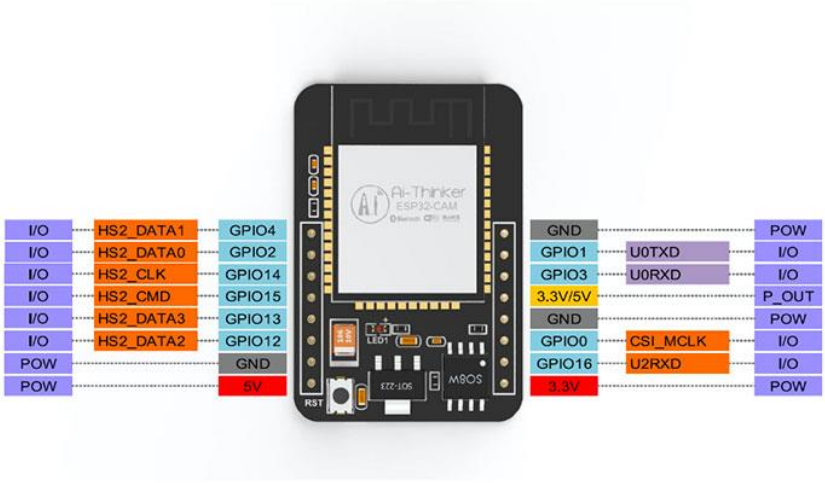

Fig 2.5 Example of Pinout for the ESP32 Cam [7]

The ESP32-CAM supports C/C++ and can be programmed using the Arduino IDE, Espressif's programming environment, or other compatible IDEs. To be able to program it we used using the UART interface with an FTDI232.

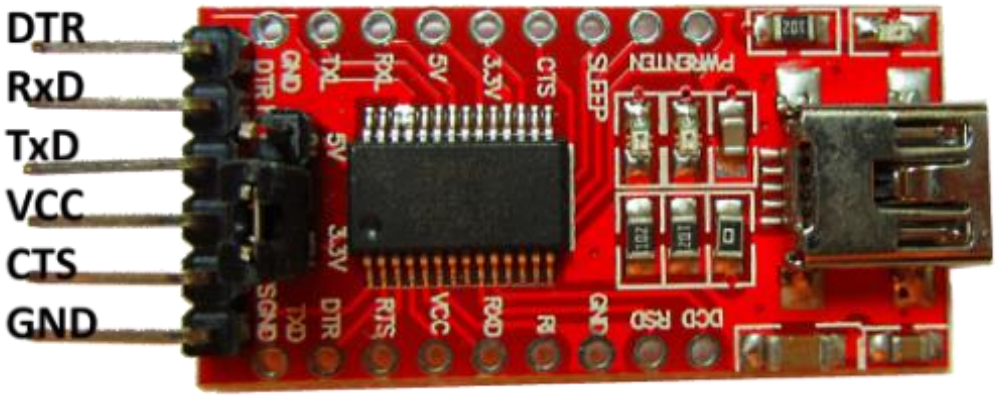

Fig 2.6 The FTIDI232 Programmer

3 Research regarding Current Technologies

So far we have settled on what microcontrollers have been used for this project but now it's time to delve into the software part, specifically what programming languages to use and what frameworks to implement. All this is split into three main categories based on what they have to be used in: the mobile app, the server and the controls. Moreover all three have to be brought together, and for that I had to establish what communication protocols we will have to implement.

3.1 The Mobile Application

The only way to control the Automated Targeting System is through an android application. The user will have to be able to see the camera feed the sentry provides as well as be able to aim and fire at will. I decided to program it using the Jetpack Compose framework, based on the Kotlin programming language.

3.1.1 Kotlin

The Kotlin Multiplatform technology was designed to help cross-platform application development. This approach has several major advantages: it reduces the time needed writing and keeps the same code for different platforms, so preserving the flexibility and benefits of native programming.[9]

One major benefit of this programming language is its flawless interaction with Java which effectively allows developers effectively run Java code from Kotlin and vice versa. Kotlin's syntax is significantly simpler than Java's, meaning that the quantity of repeated code is reduced significantly. Therefore this helps produce maintainable and more easy to read codebases.

Fig 3.1 Kotlin Logo

I decided to choose Kotlin for the mobile application because of the multiple advantages it offers. First of all, it is very easy to learn, especially for Java developers. Overall the user has to write less code which is more reliable, thus saving time when developing and debugging. Moreover, according to Google's internal data, a mobile application developed with Kotlin is 20% less likely to crash.

3.1.2 Jetpack Compose

Jetpack Compose is a declarative user interface framework for Android created by Google and released as open-source software. Unlike the typical imperative approach used in Android development, which requires developers to describe with exacting instructions how the user interface (UI) should evolve, Jetpack Compose allows developers to express what the UI should look like for a given state.

It is compatible with Android versions 5.0 and above. It uses the Kotlin programming language and it provides a reactive programming style similar to that of other user interface frameworks such as React.

Fig 3.2 Jetpack Compose Logo

Using less code, strong tools, and intuitive Kotlin APIs, it accelerates and simplifies UI development on Android. In Compose, a user interface is defined by composable functions, marked with the @Composable annotation, which specify the state of the screen. The Jetpack Compose compiler generates UI boilerplate code using this annotation.[10]

Compose also allows the use of the material design toolkit by offering a set of ready-to-use components following Google's published Material Design guidelines. This includes navigational elements, text boxes, and buttons which are easily mergeable and interchangeable, and so perfect for more complex UI. Moreover, Compose supports animations, allowing for highly satisfying and interactive user experiences with minimal code.

3.1.3 Ktor

The communication between the android application and the web server has been established through WebSockets. While doing research on how to perfectly set it up I stumbled upon the Ktor HTTP Client.

Fig 3.3 Ktor Logo

One of Ktor's main features is its asynchronous architecture. Being built on Kotlin coroutines, Ktor allows for non-blocking and asynchronous programming and so can effectively manage a large number of concurrent connections. This makes Ktor ideal for applications requiring real-time web services and APIs.

The framework, developed with flexibility and efficiency in mind, allows programmers to simply define the server they need to connect to, routing, and more by using a Domain Specific Language (DSL). Its routing mechanism provides a straightforward approach to defining endpoints and managing HTTP requests. Routes can be clearly described using the routing DSL, which also supports parameters, query strings, and other features.[12]

3.2 The Server

The core part of this project is the webserver. It acts as a hub where the application, the ESP Cam and the Raspberry Pi Pico are able to communicate with each other. To be able to access the web server and to establish communication between it, the camera, the Raspberry Pi and the mobile application I decided to use the Spring framework

3.2.1 Java

Built on the four fundamental ideas of Object Oriented Programming (OOP), Java is a well-known, class-based programming language. Its write once, run anywhere (WORA) capability means that any compiled code can be run on any platform supporting the programming language without having to be recompiled.

The Java language was created with five main goals in mind:

- It must be simple, object-oriented, and familiar.
- It must be robust and secure.
- It must be architecture-neutral and portable.
- It must execute with high performance.
- It must be interpreted, threaded, and dynamic.

By interpreting Java bytecode, the Java Virtual Machine (JVM) helps Java applications to run on any device having a JVM hence enabling this cross-platform capabilities.

Fig 3.4 Java Logo

By use of process called garbage collecting, Java implements automatic memory management. The JVM independently controls memory allocation and deallocation, therefore

reducing memory leaks and other programming errors related to human memory management.[13]

3.2.2 Spring

Spring is a flexible framework created for Java that provides a wide range of capabilities and resources for developing applications, particularly those that need a strong and scalable structure. The software is specifically engineered to streamline the development process and optimize the efficiency of programmers. Moreover, it is compatible with Groovy and Kotlin as alternative languages on the JVM and allows for the creation of various architectures based on the specific requirements of a program.

Fig 3.5 Spring Framework Logo

The framework is based on the notion of Inversion of Control (IoC), which separates the configuration and dependency management of application components. The IoC container of Spring maintains the lifespan and dependencies of the beans (objects), facilitating loose coupling and simplifying testability. Dependency Injection (DI) is a design pattern employed by Spring to provide IoC, where the container injects dependencies into beans.[14]

3.3 The Controls

It is not enough to set up the controls on the mobile application and call it a day. The Raspberry Pi Pico needs to be programmed as well to be able to connect the servomotors to the mobile interface. In order do so I used Thonny IDE, coding in MicroPython.

3.3.1 MicroPython

MicroPython is a complete implementation of the Python 3 programming language that operates directly on embedded hardware. An interactive prompt, called the REPL (Read,

Evaluate, Print and Loop), immediately executes commands via USB Serial, and a built-in file system. The Pico port of MicroPython incorporates modules that enable access to low-level, chip-specific hardware.

The read–eval–print loop (REPL) enables developers to input individual lines of code and execute them instantly on a terminal. The REPL facilitates the immediate review of individual parts of a program, enabling developers to execute each section of code and observe the outcomes.

MicroPython's use of hardware abstraction layer (HAL) technology enables the produced code to be easily transferred between other microcontrollers within the same family or platform, as well as on devices that support and can download MicroPython. Programs are commonly created and verified on high-performance microcontrollers and then delivered with the final application for used on lower-performance microcontroller boards.[16]

3.4 Communication Protocols

To bring everything together I had to employ several communication protocols. The Spring boot application receives the control from the mobile application through the use of WebSockets, it forwards them to the Raspberry PI through MQTT and fetches the ESP32's video feed as Server Sent Event.

3.4.1 WebSockets

The mobile application communicates with the Spring app through WebSockets. One on hand it sends the position of the virtual joystick and it the shoot button has or has not been pressed, and on the other it receives the video feed sent by the ESP32 Cam that was sent to the Spring app.

WebSocket is a computer communications method that allows bidirectional communication over a single connection using Transmission Control Protocol (TCP). The WebSocket handshake utilizes the HTTP Upgrade header when transitioning from HTTP to WebSocket, thus it guarantees that the two protocols are compatible.[17]

Unlike existing methods like HTTP polling which only allow half-duplex communication, the WebSocket protocol enables full-duplex interaction between a client program and a web server. This contact is characterized by its reduced operational costs. This

enables the instantaneous transfer of data between the server and the client. Consequently, the server's protocol for transferring content to the client without it initiating a request has been standardized. This facilitates continuous communication by maintaining an open connection and enables bidirectional communication between server and client.

WebSocket, although beginning with an HTTP request and intended to work alongside HTTP, diverges considerably from it in terms of its architecture and application programming models, while also encouraging an event driven design.

Both the HTTP and REST modeling protocols utilize several URLs to symbolize an application. Users engage with the software through a request-response method by accessing the URLs indicated before. Servers route requests to the proper handler based on the HTTP URL, method, and headers.

Moreover, WebSockets typically utilize a single URL for the initial connection. Once the connection is created, all application messages are transmitted using the same TCP connection, leading to the creation of an entirely distinct asynchronous, event-driven messaging architecture.

WebSocket is a transport protocol that operates at a low level and does not impose any specific meaning or structure on the content of messages, unlike HTTP. In order for the message to be routed and processed, it is necessary for the client and server to have a mutual understanding of the message's meaning.

3.4.2 MQTT

I am transmitting the control commands via the MQTT messaging protocol, guaranteeing seamless communication between the mobile app and the Raspberry Pi Pico. After the application sends the control commands to the Spring app on my local device through WebSockets, they are forwarded through a web server, acting as a MQTT broker, before reaching microcontroller.

The MQTT protocol categorizes network elements into two main types. The categories encompass a message broker and several clients. An MQTT broker is a server that receives messages from clients and efficiently directs them to the most appropriate recipients. An "MQTT client" is any device, ranging from a microcontroller to a computer that is capable of running a MQTT library and establish a connection to the MQTT broker across a network.

MQTT categorizes its data using a hierarchical structure of topics. When a publisher acquires data to distribute, they send a control message to the connected broker. Subsequently, the broker will transmit the information to all clients who have subscribed to the pertinent topic for the discussion. The publisher does not require knowledge of the quantity or location of subscribers, and subscribers do not require any information about the publisher.[18]

If a broker gets a message on a topic that has no current recipients, the message will be destroyed unless the publisher explicitly requests it to be retained. In a standard MQTT message, the kept flag is enabled, resulting in the message being retained. The broker stores the latest communication together with the corresponding quality of service (QoS) related to that particular topic. Upon subscribing to a subject that matches the topic of the retained message, the client will promptly get the retained message after the subscription process is finalized. Since the broker only keeps one retained message per topic, new subscribers can promptly access the latest value without having to wait for the next update from the publisher.

During the first connection between a publishing client and the broker, the publishing client has the option to set a default message. This message will be sent to subscribers in the event that the broker detects a sudden disconnection from the publishing client.

3.4.3 SSE

The last protocol we'll talk about is Server Sent Events (SSE). The ESP32 Cam transmits by default the live video feed to its own HTTP server. This means I had to use this protocol to get the feed from its server to the Spring app, and then to transmit it to the mobile application through WebSocket.

SSE is a technology that enables a server to deliver real-time changes to web clients using a single HTTP connection. The initiation of the process occurs when an HTTP connection is established between using a customized endpoint specifically built for transmitting event changes. The connection remains open, enabling the server to promptly deliver updates as soon as they are accessible. The server sends these changes using the text/event-stream format across the open connection. An update, which is often referred to as an event, typically consists of fields such as event, data, id, and retry.[20]

Regarding fundamental applications, the simplicity of SSE is a notable advantage it has over WebSockets. The EventSource API in the browser autonomously manages reconnections and event processing, hence simplifying the development process for developers. SSE is also interoperable with preexisting HTTP infrastructure, including cache and proxy servers. This compatibility can be advantageous for the reasons of deployment and scalability.

4 Building the Project

The idea for the Automated Targeting Device originated during a 3rd-year course titled “Internet of Things” and I decided to further develop this idea into our license project. As I’ve stated before, its objective is to defend an area without disclosing the user's location.

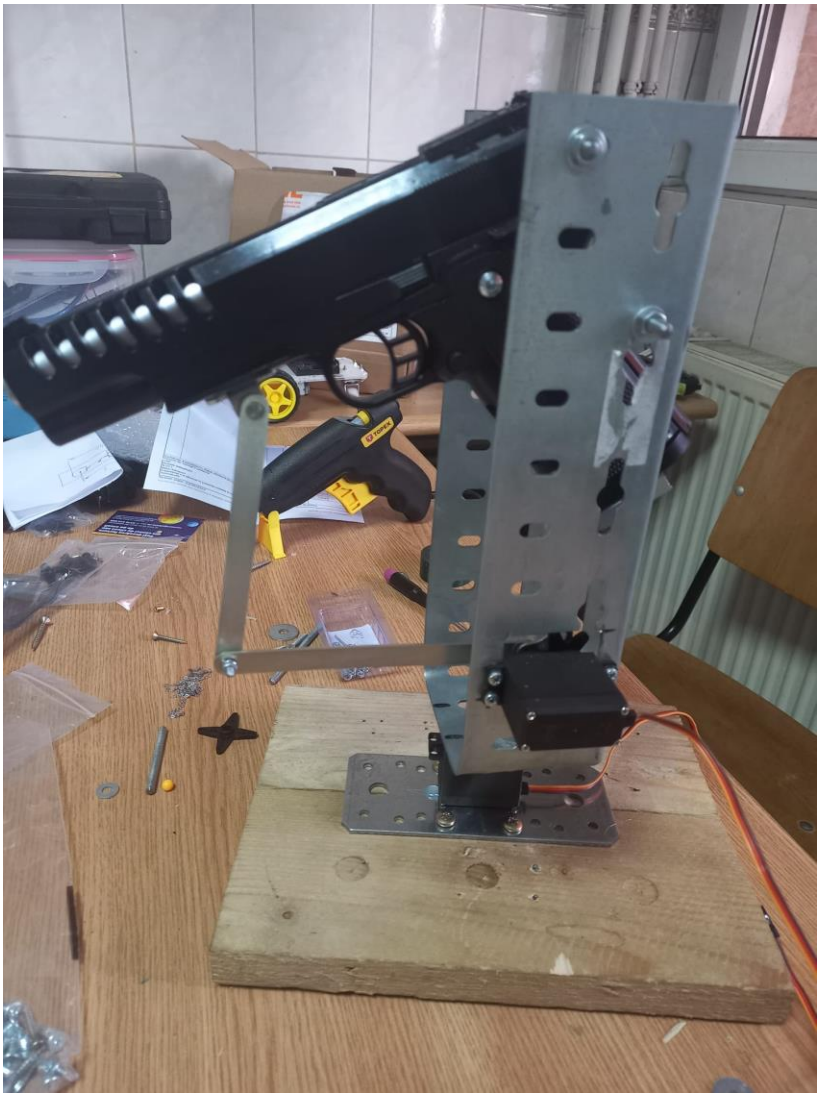

Fig 4.1 Initial design of the Project

Since then, the chassis has been completely redesigned, opting for a 3D printed custom one instead. This offers better stability, allows for seamless, smooth control of the servomotors when it comes to aiming and an overall more pleasant look to the device.

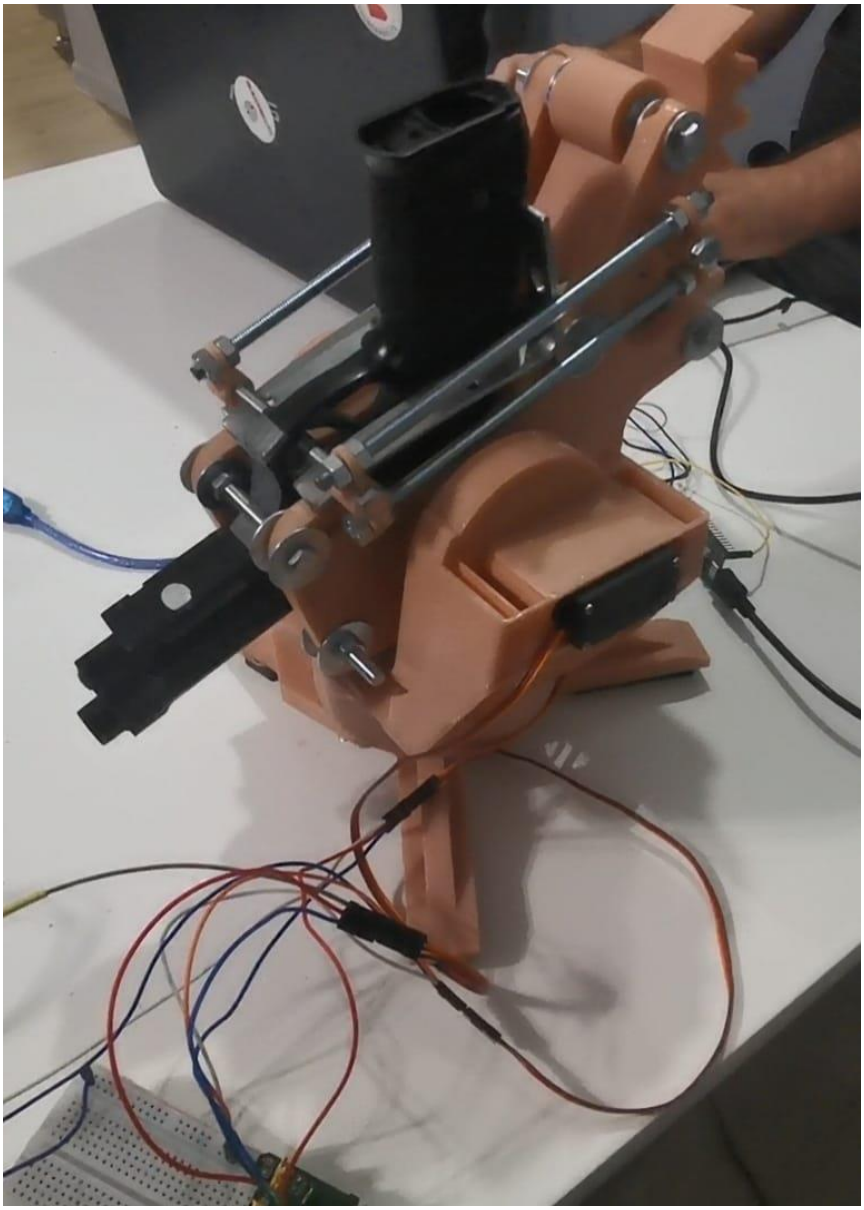

Fig 4.2 The Automated targeting device during testing

When it comes to the controls I have used 2 servo motors MG995 to control the horizontal and the vertical rotation of the pistol, and another one for operating the trigger of the CO_2 pistol. All the components are powered up by a YwRobot Mb-V2 breadboard power supply.

4.1 Planning phase

4.1.1 Gantt Chart

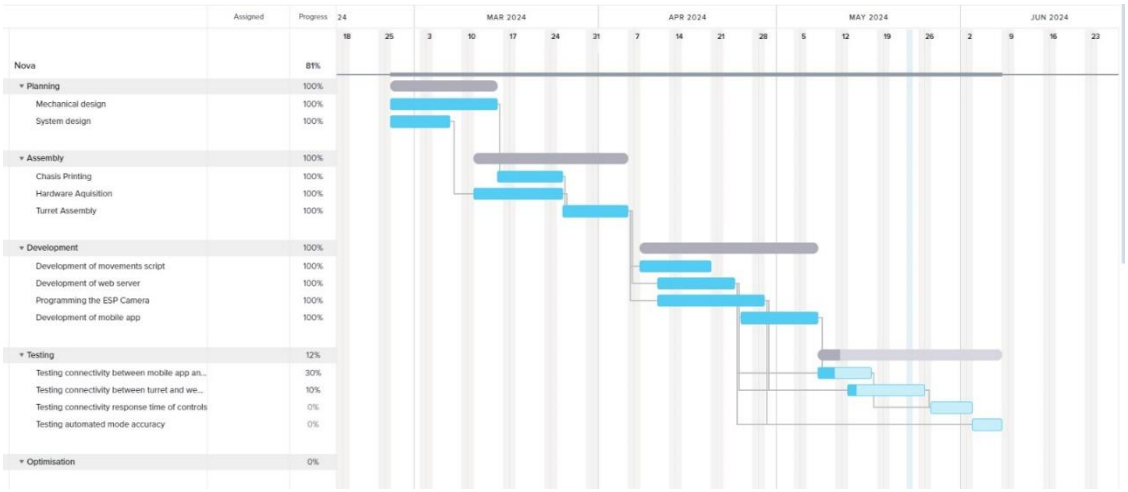

Fig 4.3 Gantt chart

To ensure that any project can be completed before its proposed deadline, a careful thorough schedule has to be established. Thus I have split my work into easy to track subcategories and set up a Gantt chart to keep track of the progress I've made.

4.1.2 SMART Objectives

Develop and Integrate Hardware Components:

- ***Specific:*** Assemble and integrate all hardware components (Raspberry Pi Pico WH, servo motors, ESP32 Cam, CO2 airsoft pistol, and 3D printed chassis) to form a functional airsoft sentry turret.

- ***Measurable:*** Ensure all components are physically connected and operational within the system.

- ***Achievable:*** I have the necessary technical skills and resources.

- ***Relevant:*** This step is crucial for creating the physical foundation of the sentry turret.

- ***Time-bound:*** Complete the hardware integration within 3 months.

Develop Mobile Application for Manual Control:

- ***Specific:*** Create a mobile application using Kotlin and Jetpack Compose for manual control of the turret.
- ***Measurable:*** Ensure the app provides a live video feed, virtual joystick, and firing button, with less than 100ms latency.
- ***Achievable:*** Studied mobile app development in 3rd year
- ***Relevant:*** The mobile app is essential for manual control functionality.
- ***Time-bound:*** Develop and launch the mobile app within 2 months.

4.1.3 SWOT Analysis

Strengths:

- **Technological Integration:** Combines advanced hardware and software, including machine learning for autonomous targeting.
- **Flexibility:** Offers both autonomous and manual control modes, enhancing usability.
- **User Interface:** Mobile application provides intuitive manual control with real-time video feed.
- **Real-time Communication:** Ensures low latency and efficient data transmission between components.

Weaknesses:

- **Complexity:** The integration of various technologies can lead to technical challenges and potential points of failure.
- **Balance Issues:** The back-heavy chassis design can cause stability problems, affecting vertical rotation.
- **Power Consumption:** High power consumption due to multiple components can limit operational time without frequent recharging.

- **Reloading:** Requires to be manually reloaded every 13 shots.
- **Dependency on light levels:** Performance may vary depending on environmental conditions, being harder to see targets in darker areas.

Opportunities:

- **Market Expansion:** Potential for use in other recreational activities or security applications.
- **Technological Advancements:** Continuous improvements in machine learning and hardware components can enhance system performance.
- **Customization:** Opportunity to offer customizable features for different user preferences and scenarios.
- **Collaborations:** Partnerships with airsoft game organizers and tech companies for further development and deployment.

Threats:

- **User Safety:** Risk of injury during use could result in liability issues and damage the project's reputation.
- **Operational Risk:** It is dependent on a stable internet connection for real - time control and little to no latency camera feed

4.1.4 Use Cases:

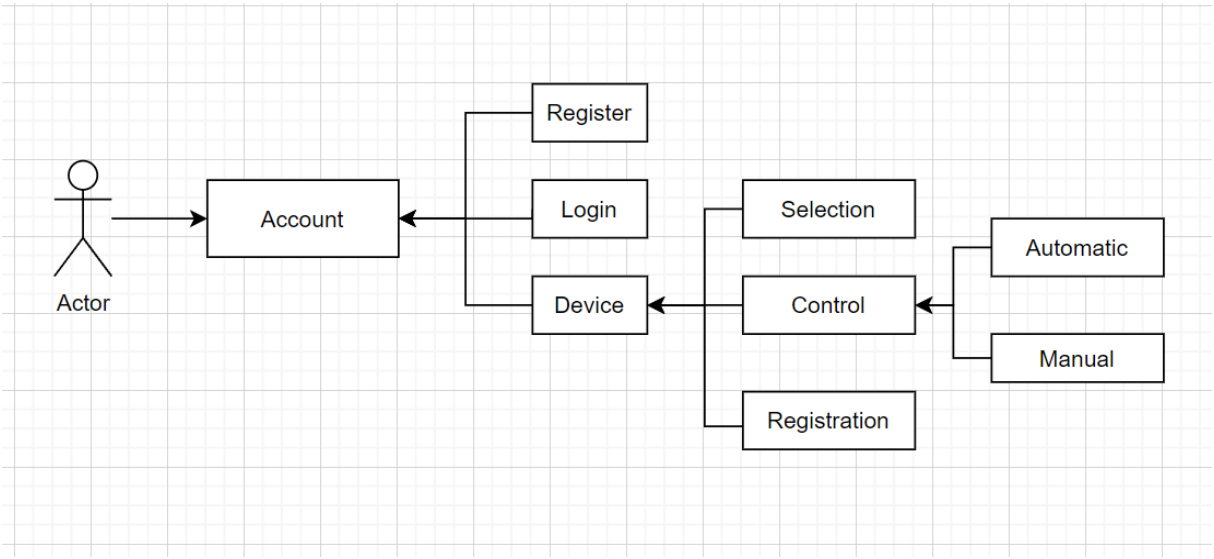

Fig 4.4 Use Case Diagram for my project

Last thing to do before starting building the Automated Targeting System was to get a clear view of what functionalities will I want the project to have. For that I set up use case diagram so I could focus on each individual functionality one at a time.

Descriptions:

Use case: Register (Sign Up)

- Actor: User
- Flow: As soon as the user navigates the register screen, they enter their registration details, choosing a strong password for their account. The system then creates the new account and redirects the user to the Log In screen.

Use case: Log in

- Actor: User
- Flow: Once on the Log In screen, the user will enter their username and password. The system will then verify the provided information and act accordingly. If the credentials are correct then the system logs the user into their account and redirects them to the device selection screen. If the provided information is wrong, the system will notify the user of the invalid credentials, and the user will have re-enter the correct ones and try again.

Use case: Selection

- Actor: User
- Flow: Upon navigating to the device selection screen, the system displays a drop down list of the registered devices. The user will then have to set active the one they'll want to control.

Use case: Control

- Actor: User
- Flow: The user first navigates to the control screen. The system will then display the live video feed from the ESP 32 Cam, the controls for the device as well as a crosshair to help with targeting. The user can use the virtual joystick to aim the turret and press on the button to fire. The system will then send the control commands to the Raspberry Pi Pico via MQTT, and the microcontroller will execute the commands to move the servomotors and fire the CO2 Pistol.

Use case: Registration (Add Device)

- Actor: User
- Flow: Back on the device select screen, the user will be able to register a new device they would like to control Pressing a button at the bottom of the drop down list. They will then have to provide the device's name and its unique id. Once that's done the system will validate the information, register the device and add it to the drop down list.

4.2 The Chassis

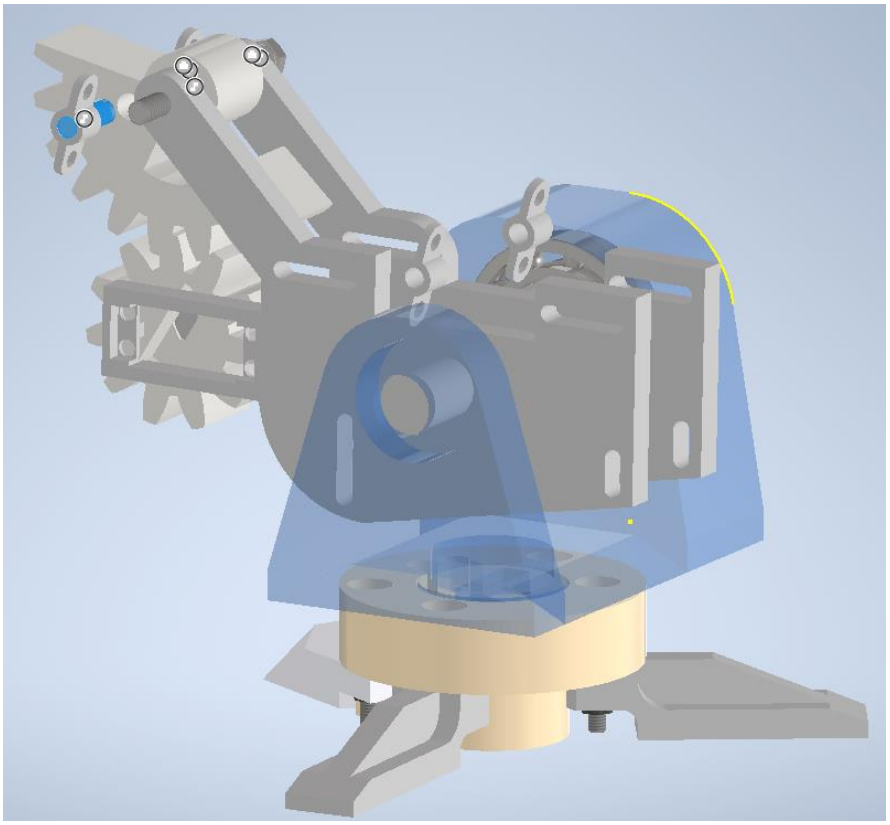

Fig 4.5 Sketch of the 3D printed chassis

Based on the measurements of the motors and the CO2 Pistol, I used Autodesk Inventor, a computer-aided design (CAD) application special for mechanical design. By converting the design sketches to .stl files, I uploaded them to the 3D Printer for manufacturing.

Table 4.1 Chassis dimensions

Dimensions	**Base**	**Holder**	**Gear 1**	**Gear2**	**Overall**
Height	65mm	146mm	110mm	65mm	246mm
Width	223mm	150mm	30mm	30mm	223mm
Length	184mm	190mm	110mm	65mm	264mm

The Printer I used is the Shining 3D AccuFab-L4D. I opted to use a 3D printer that uses resin instead of filament for a smoother look and for less tolerance errors. Due to the dimensions of the individual parts however, I had to do multiple time consuming prints before all the components were brought to shape and ready for assembly.

The chosen resin was the Liqcreate Premium Model, an opaque peach colored photopolymer compatible with the AccuFab-L4D. It provides excellent dimensional stability, low shrinkage during printing and provides great details to the printed elements.

Table 4.2 Resin properties

Polymer properties			
Mechanical properties		**30 minutes high power LED curing at 40°C**	
Description	**ASTM Method**	**Metric**	**Imperial**
Tensile Strength	D638M	53 MPa	7.69 ksi
Tensile Modulus	D638M	1.9 GPa	276 ksi
Elongation at break	D638M	4.1%	
Flexural Strength	D790M	80 MPa	11.6 ksi
Flexural modulus	D2240	2.1 GPa	305 ksi
IZOD Impact (notched)	D256A	17 J/m	0.29 ft-lb/in
Shore D Hardness	D2240	83	
Water sorption	D570-98	0.4 %	
Linear shrinkage during printing	Internal method	0.5 %	
Linear shrinkage during UV-curing	Internal method	0.5 %	

For more precise vertical rotation and in order to induce less stress on the servomotor, the chassis features two radial bearings, fitted on either side of the main body. They help in significantly reducing the friction rate and completely prevent the material from wear down damage.

The trigger mechanism was rather tricky challenge to overcome. Since the MG955 servomotor didn't have enough force to be able to pull the trigger by itself, I had to implement a simple gear down system that would reduce the required force to efficiently perform the firing functionality.

4.3 The Application

As mentioned before, the mobile application, named NovaDefense, was developed using the Jetpack Compose framework. I aimed to make the user interface as visibly pleasing and intuitive as possible. It features a Log In and a Sign Up screen, one for device selection, another one providing account info and, it's main purpose, the control screen.

4.3.1 Start Screen

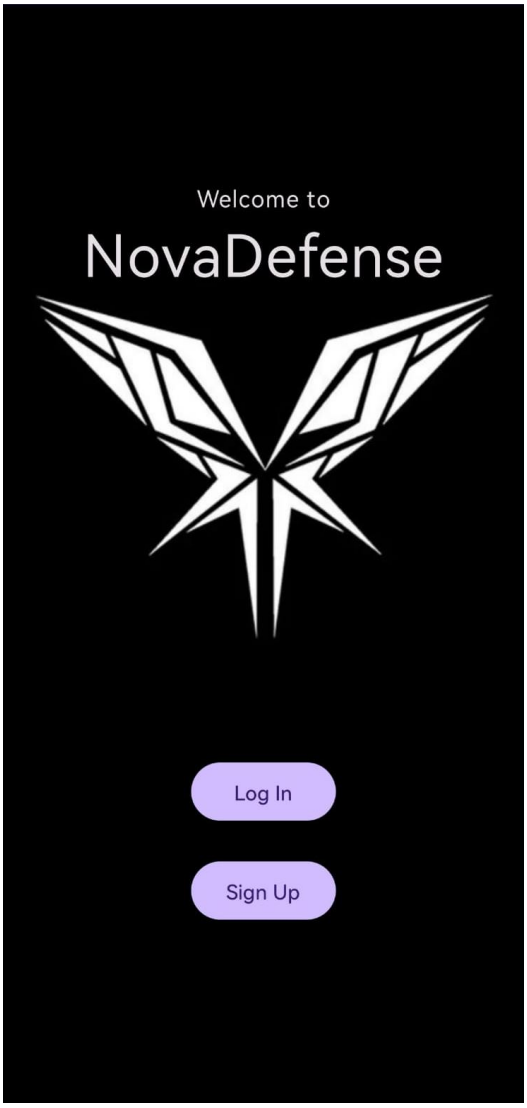

Fig 4.6 Starting page of the Application

When opening the app, the user will be presented with the application's logo and two buttons will fade-in view. The first one is for them to log into their account and the second is meant to allow them to sign up and set up a new account.

4.3.2 Sign up Screen

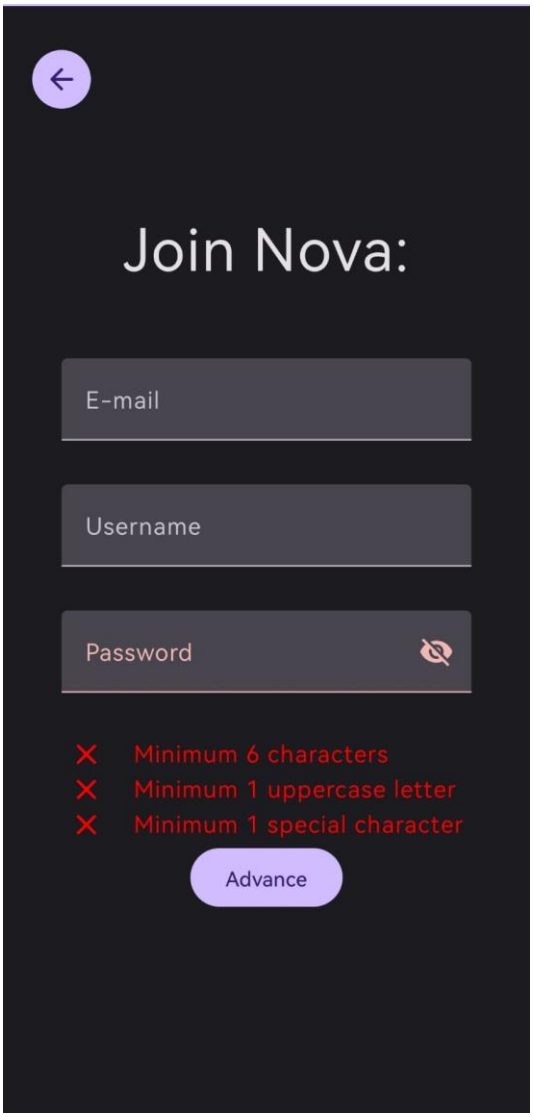

Fig 4.7 Sign Up page of the Application

While setting up their account, the product owners will need to provide their e-mail address, choose a username and set up their password. It is mandatory for the password to have a minimum of six characters, one uppercase letter and one special character. This offers enhanced security, making sure that the sentry turret can only be operated by its owner.

4.3.3 Device Select Screen

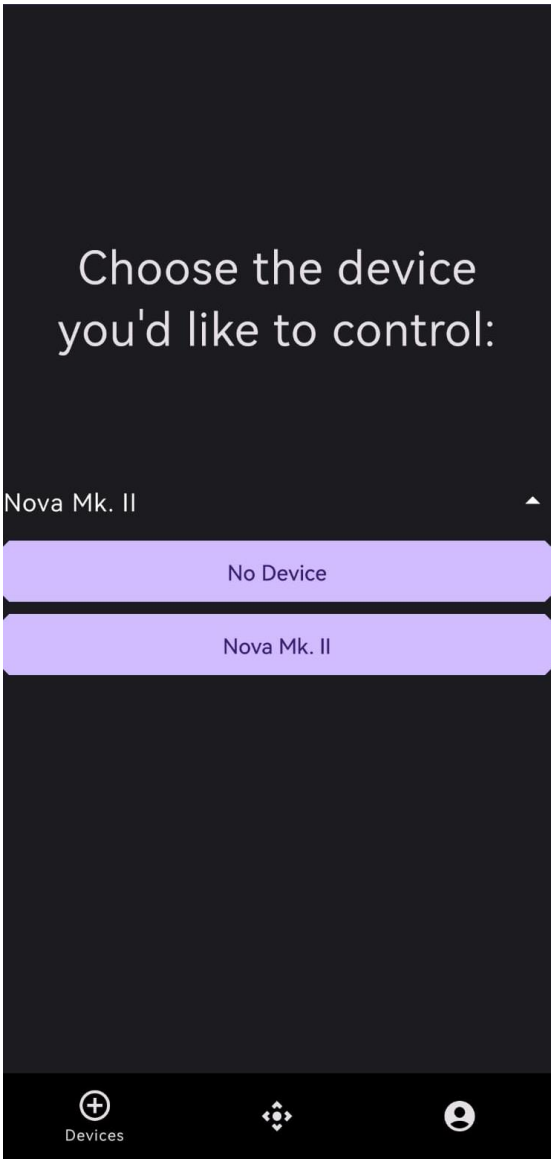

Fig 4.8 Device Select page of the Application

Planning ahead for the future, I also designed a device select screen. This allows the application to be used together with other versions of this product and for other remote-controlled projects that will be developed in the future. It is implemented as a drop down list where the operator can select the device they'd like to control.

4.3.4 Control Screen

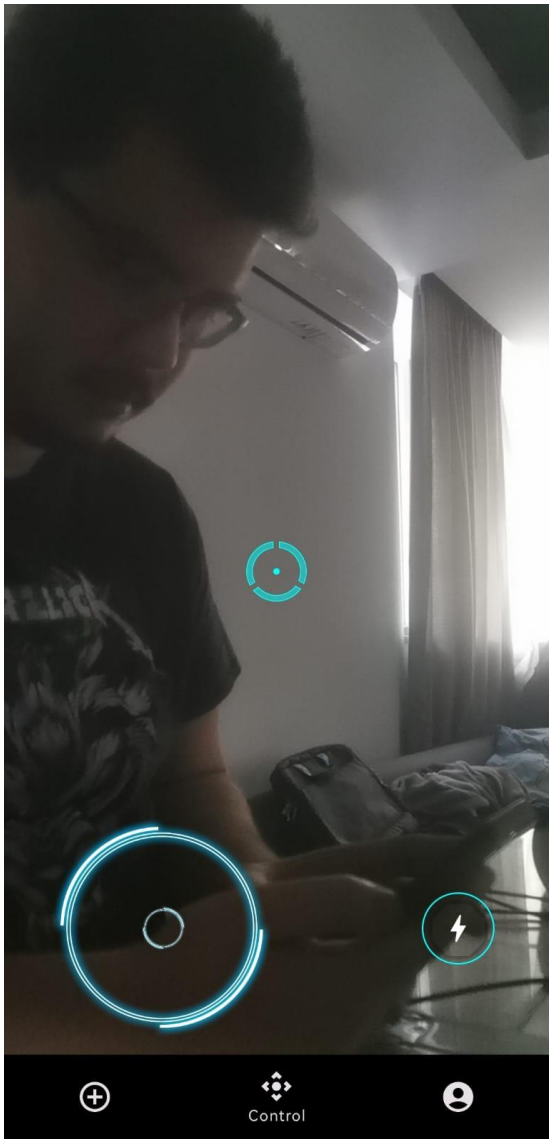

Fig 4.9 Control screen of the Application

Coming up is the main feature of the mobile application: the control screen. Here the ESP32's camera feed is displayed and the user can aim and shoot the turret at will. The controls are simple and intuitive.

On the bottom left I designed and implemented a virtual joystick that controls the two servos responsible for the movement, and on the bottom right there is a button that controls the firing function. In the middle of the screen there is a crosshair for the user to make sure they are aiming at their target

4.3.5 The control code

```
class ControlViewModel : ViewModel() {  Agoriam-1045
    suspend fun sendJoystickPosition(x: Float, y: Float) {  Agoriam-1045
        WebSocketClient.service.sendMessage("{\"x\" : ${x}, \"y\" : ${y}}")
    }

    suspend fun sendButtonClicked() {  Agoriam-1045
        WebSocketClient.service.sendMessage("SHOOT_COMMAND")
    }
}
```

Fig 4.10 The data sent through WebSockets

The controls are sent to the Spring boot application through WebSockets. The button will send a message stating “SHOOT_COMMAND” every time it is pressed, and the joystick will send its x and y position whenever it’s updated

```
ElevatedButton(
    modifier = Modifier
        .padding(40.dp)
        .size(40.dp)
        .border(
            width = 1.dp,
            color = Color.Cyan,
            shape = CircleShape
        ),

    shape = CircleShape,
    onClick = {
        lifecycleOwner.lifecycleScope.launch {
            viewModel.sendButtonClicked();
        }
    },

    colors = ButtonDefaults.buttonColors(
        containerColor = Color.Transparent,
        contentColor = Color.White
    ),
    contentPadding = PaddingValues(0.dp)
) {
    Icon(
        Icons.Rounded.Bolt,
        modifier = Modifier.fillMaxWidth(),
        contentDescription = "shoot_button"
    )
}
```

Fig 4.11 Setting up and displaying the fire button

```
JoyStick(
    Modifier.padding(30.dp),
    size = 150.dp,
    dotSize = 30.dp
){ x: Float, y: Float ->
    lifecycleOwner.lifecycleScope.launch {
        viewModel.sendJoystickPosition(convertJoystickValue(x), convertJoystickValue(y))
    }
}
```

Fig 4.12 Displaying the joystick

The “convertJoystickValue” function is responsible for transforming the x and the y values of the virtual joystick to no longer range between -1 and 1, but from 0 to 180 instead, thus directly transmitting the angles for the motors to turn to the sentry turret

4.4 The Server

To ensure communication between the webservers, the sentry turret and the NovaDefense application, I developed a Spring boot application on my local machine. To bring everything together, I took advantage of three communication protocols.

```
public class NovaMessageHandler extends BinaryWebSocketHandler {
    private final List<WebSocketSession> sessions = new ArrayList<>();  4 usages

    public NovaMessageHandler(SseClientService sseClientService) {  Agoriam-1045
        var flux = sseClientService.getSse( url: "http://192.168.43.3:81/stream");
        flux.doOnNext(lastFrame -> {
            sessions.forEach(session -> {
                try {
                    session.sendMessage(new BinaryMessage(lastFrame));
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            });
        }).subscribe();
    }
```

Fig 4.13 Fetching the camera feed from the ESP’s server

In order to display the live video feed of the ESP32 Cam, I first had to fetch it from its dedicated HTTP server via SSE, and then forward it to the mobile application frame by frame through WebSockets using the “NovaMessageHandler” class in the figure above.

The figure below displays the “AppMessageHandler”, which is responsible for achieving seamless and real-time control of the turret using the NovaDefense application.

```
public class AppMessageHandler extends TextWebSocketHandler {
    private final MqttPublisher mqttPublisher;

    @Override  no usages  Agoriam-1045
    protected void handleTextMessage(@NonNull WebSocketSession session, @NonNull TextMessage message) throws Exception {
        log.info(message.getPayload());
        if (message.getPayload().equals("SHOOT_COMMAND")) {
            mqttPublisher.send(message.getPayload(), MqttTopic.SHOOT);
        }
        else {
            mqttPublisher.send(message.getPayload(), MqttTopic.CONTROL);
        }
    }
}
```

Fig 4.14 Sending the commands through MQTT to the Raspberry Pi

It receives the “SHOOT_COMMAND” and the angles for the pitch and the yaw angles for aiming the sentry through WebSockets and passes them forward to the Raspberry Pi as JSON via the MQTT protocol.

It features two separate transmitting topics, each dedicated for a single role, one for movement, one for firing. An MQTT Topic is a channeling mechanism. Since the protocol features a publish-subscribe model, the topics are especially designed to forward the information only to their subscribed clients, thus making it easier to manage both for the MQTT clients and for the developers themselves.

4.5 The Pi controls

```
# MQTT message callback
def mqtt_callback(topic, msg):
    print((topic, msg))
    if mqtt_control in topic:
        json_payload = ujson.loads(msg)
        print(type(json_payload))
        x = json_payload["x"]
        y = json_payload["y"]
        print("x pos: ", x, " ", "y pos: ", y)
        servo_Angle_Yaw(x)
        servo_Angle_Pitch(y)
    elif mqtt_shoot in topic:
        servo_Shoot()


# Main program
def main():
    connect_wifi(ssid, password)

    client = MQTTClient('pico_client', mqtt_server, port=mqtt_port)
    client.set_callback(mqtt_callback)
    client.connect()
    print('Connected to MQTT broker')

    client.subscribe(mqtt_control)
    print('Subscribed to topic:', mqtt_control)

    client.subscribe(mqtt_shoot)
    print('Subscribed to topic:', mqtt_shoot)

    while True:
        client.wait_msg()

    client.disconnect()
```

Fig 4.15 The code for processing the Received MQTT messages

Last but not least, using MicroPython I was able to program the Raspberry Pi Pico WH to subscribe to the MQTT topics and process the JSON commands received, in order for it to operate the servomotors accordingly.

5 Validation and testing

5.1 Unit Testing

The servo motors were tested individually using at first an Arduino Mega 2560 to which I also connected an analog joystick. The ones responsible for the vertical and horizontal movement were tested by moving the joystick, while the third servo responsible for the shooting function was tested by pressing down on the stick

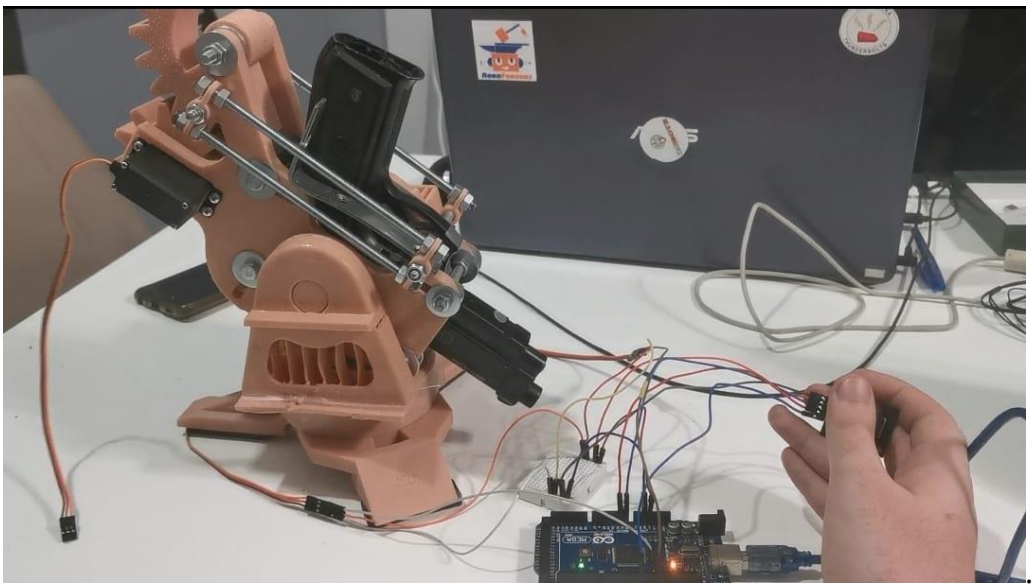

Fig 5.1 Photo taken during the servomotor testing

To ensure the ESP32 camera is able to stream with minimal latency and good video quality I checked its dedicated server. After tweaking with its resolution I managed to obtain satisfying results

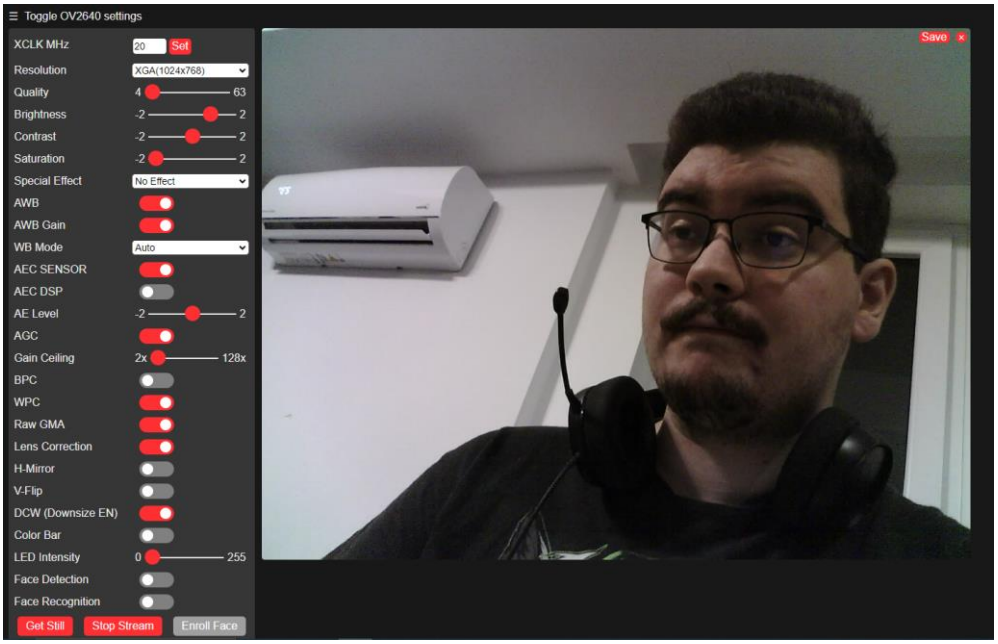

Fig 5.2 Screenshot taken during camera testing

The NovaDefense mobile application was incrementally tested every time a new screen was developed to ensure the navigation flows smoothly and as intended, as well as checking if the UI elements are all in the right place and of the proper size.

5.2 Integration Testing

To verify data integrity, I checked the WebSocket communication with the Spring boot application by consulted the logs from the Spring app.

```
[spring-ws] [nio-1045-exec-2] ws.springws.socket.AppMessageHandler     : {"x" : 149.2, "y" : 158.0}
[spring-ws] [nio-1045-exec-3] ws.springws.socket.AppMessageHandler     : {"x" : 152.8, "y" : 154.4}
[spring-ws] [nio-1045-exec-7] ws.springws.socket.AppMessageHandler     : {"x" : 90.0, "y" : 90.0}
[spring-ws] [nio-1045-exec-5] ws.springws.socket.AppMessageHandler     : SHOOT_COMMAND
[spring-ws] [nio-1045-exec-6] ws.springws.socket.AppMessageHandler     : {"x" : 91.6, "y" : 86.0}
[spring-ws] [nio-1045-exec-8] ws.springws.socket.AppMessageHandler     : {"x" : 96.00001, "y" : 76.0}
[spring-ws] [nio-1045-exec-4] ws.springws.socket.AppMessageHandler     : {"x" : 99.6, "y" : 66.4}
[spring-ws] [nio-1045-exec-9] ws.springws.socket.AppMessageHandler     : {"x" : 101.200005, "y" : 59.999996}
[spring-ws] [io-1045-exec-10] ws.springws.socket.AppMessageHandler     : {"x" : 102.799995, "y" : 54.4}
[spring-ws] [nio-1045-exec-1] ws.springws.socket.AppMessageHandler     : {"x" : 104.399994, "y" : 51.6}
[spring-ws] [nio-1045-exec-2] ws.springws.socket.AppMessageHandler     : {"x" : 90.0, "y" : 90.0}
[spring-ws] [nio-1045-exec-3] ws.springws.socket.AppMessageHandler     : SHOOT_COMMAND
```

Fig 5.3 The logs from the Spring application regarding WebSocket communication

Next I had to ensure that the data was sent in the same order and without any information loss to the Raspberry Pi. For that I took advantage of Postman's testing capabilities and reviewed the responses it provided.

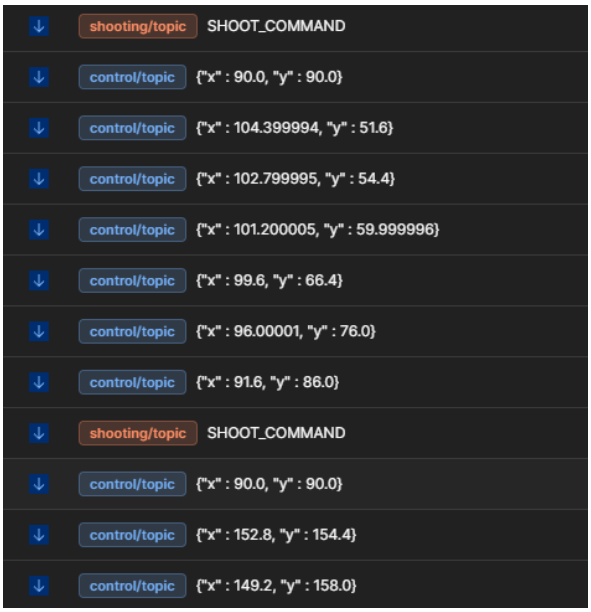

Fig 5.4 The responses from Postman regarding MQTT communication

Once I made sure the Postman responses matched the Spring application's logs it was time to bring everything together.

5.3 System and Performance Testing

After bringing everything together it was time to make sure everything worked as intended. The device was tested by performing typical user interaction such as logging in, aiming and firing the sentry turret directly from the mobile application.

To test the performance of my project we tested to see how the WebSocket and MQTT connections would handle multiple requests sent in quick succession.

5.4 Future Works

The next step to upgrade the sentry turret is to make it completely autonomous. Using a machine learning model designed for human recognition I will be able to implement target tracking ensuring the turret moves accordingly to keep the detected person in the middle, before engaging its target. This would further enhance its purpose to be able to defend an area without disclosing the position of its owner.

Another thing to keep in mind if I were to further chase the idea of transforming the NovaDefense app into a hub for my future remote control projects is a way to register a new device that the user wants to control.

Moreover the ESP32 Cam could be replaced with a more expensive one, offering better quality. In this way the need for the extra HTTP Server the ESP is dependent on would be removed, allowing the video feed to be streamed directly to my own webserver, thus significantly reducing latency.

Another possible improvement for the NovaDefense application would be a dedicated settings screen, where the user could adjust the aim sensitivity, the field of view and the quality of the live video feed to their liking, thus enhancing the user experience.

Tweaks could be made in the chassis' design in order to fit and protect all the electrical components, as well as hide the wiring. This would ultimately offer another layer of protection to the circuit from accidents resulting in critical damage.

Last but not least, a slight problem that is currently present in the sentry turret is the airsoft Taurus P92 handgun's limited magazine size of 13 rounds. Research can be undertaken to find a way to automatically reload the device by pressing a button on the mobile application. The app could also notify the user if the Automated Targeting Device eventually runs out of ammunition and requires manual reloading.

These are just a few of the possible improvements that can be brought to this project, and I most certainly plan on improving the device further, knowing that so far working on bringing this idea to life was a highly enjoyable experience.

6 Conclusions

In conclusion, the development and implementation of the Automated Targeting Device showcased a comprehensive blend of hardware and software integrations. The sentry turret is highly efficient and was developed by integrating multiple technologies such as a Raspberry Pi Pico, an ESP32 Cam, and a custom mobile application developed using Jetpack Compose.

The primary objective of this project was to develop a remotely operated defense system with the aim of enhancing the tactical experience in airsoft. During the development process, significant attention was given to enabling seamless communication between the components through the use of WebSockets, MQTT, and SSE protocols. The final outcome was a system that exhibited high speed and had the ability to broadcast video in real time, all while ensuring minimal delay in control.

Unit testing, integration testing, and performance testing were all necessary components of the validation phase to guarantee the functionality and stability of the system. The device's ability to handle numerous requests while ensuring data integrity across multiple communication channels was proven. The study's successful completion showcases the potential for additional upgrades of this project, including the utilization of machine learning models to enhance target tracking and recognition.

While the automated targeting mode has not yet been implemented, the current achievements lay a solid foundation for future enhancements. The integration of machine learning models for target tracking and autonomous operation will be a logical next step, leveraging the capabilities of Python and OpenCV for real-time image processing and decision-making. Additionally, there are plans to improve the NovaDefense application in order to accommodate a wider variety of remote-controlled devices. This will enhance the app's functionality and adaptability.

This project represents a notable progress in integrating IoT and robotics technology for recreational and security purposes. The technologies developed in this context, along with the knowledge gained, provide a strong basis for future progress in automated defense systems.

References

[1] The evolution of Airsoft: A comprehensive history. (2023). Retrieved from https://www.abbeysupply.com/resources/the-evolution-of-airsoft-from-military-training-to-recreational-sport

[2] Shayotovich, E. (2023). Everything we know about Samsung's machine gun robots. Retrieved from https://www.slashgear.com/825074/everything-we-know-about-samsungs-machine-gun-robots/

[3] Arduino Mega 2560 layout, specifications (Arduino Mega Pinout). (2024). Retrieved from https://www.electronicshub.org/arduino-mega-pinout/

[4] Bluetooth nerf turret. (2019). Retrieved from https://projecthub.arduino.cc/Little_french_kev/bluetooth-nerf-turret-ea2fac

[5] Raspberry Pi documentation - Raspberry Pi Pico and Pico W. (n.d.). Retrieved from https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html

[6] Hill, A. (2024). Raspberry Pi Pico drives Storper modular DIY robot rover. Retrieved from https://www.tomshardware.com/raspberry-pi/raspberry-pi-pico-drives-storper-modular-diy-robot-rover

[7] ESP32-CAM camera module. (2022). Retrieved from https://components101.com/modules/esp32-cam-camera-module

[8] Espressif. (n.d.). ESPRESSIF/ESP32-camera. Retrieved from https://github.com/espressif/esp32-camera

[9] Kotlin for Android: Kotlin. (2023). Retrieved from https://kotlinlang.org/docs/android-overview.html

[10] Get started with Jetpack Compose. (2024). Android Developers. Retrieved from https://developer.android.com/develop/ui/compose/documentation

[11] Jetpack Compose. (2024). Retrieved from https://www.slashgear.com/825074/everything-we-know-about-samsungs-machine-gun-robots/

[12] Create a WebSocket application in Kotlin with KTOR. (2024). Retrieved from https://ktor.io/docs/server-create-websocket-application.html

[13] Java documentation - get started. (2023). Retrieved from https://docs.oracle.com/en/java/

[14] Spring Framework overview. (n.d.). Retrieved from https://docs.spring.io/spring-framework/reference/overview.html

[15] MicroPython. (n.d.). MicroPython Wiki. Retrieved from https://github.com/micropython/micropython/wiki

[16] Halfacree, G. (2024). Get started with MicroPython on Raspberry Pi Pico: The official Raspberry Pi Pico guide. Raspberry Pi Press.

[17] Melnikov, A., & Fette, I. (2011). RFC 6455: The WebSocket protocol. Retrieved from https://datatracker.ietf.org/doc/html/rfc6455

[18] Sengul, C., & Kirby, A. (2023). RFC 9431: Message Queuing Telemetry Transport (MQTT) and Transport Layer Security (TLS) profile of authentication and authorization for constrained environments (ACE) framework. Retrieved from https://datatracker.ietf.org/doc/html/rfc9431

[19] Server-Sent Event. (2024). Retrieved from https://html.spec.whatwg.org/multipage/server-sent-events.html

[20] Server-sent events. (2024). Retrieved from https://www.slashgear.com/825074/everything-we-know-about-samsungs-machine-gun-robots/

[21] Airsoft Romania. (n.d.). Pistol 4 Joules full metal Taurus PT 92 Black Edition CYBERGUN CO2. Retrieved from https://www.airsoftromania.ro/arme-airsoft/460-pistol-4-joules-full-metal-taurus-pt-92-black-edition-cybergun-co2.html

[22] Google. (n.d.). Material Design: Introduction. Retrieved from https://m2.material.io/design/introduction

[23] Project Lombok. (n.d.). Lombok features. Retrieved from https://projectlombok.org/features/

Table of Figures

Table of Tables

Annex

Glossary

The following enumerated abbreviations have been used at least once throughout the theoretical paper:

MilSim – Military simulation

CIWS – Close-In Weapon System

DMZ – Demilitarized zone

USB – Universal Serial Bus

I/O – Input/Output

PWM – Pulse-Width Modulation

MHz – Megahertz

KB – Kilobyte

SRAM – Static Random Access Memory

EEPROM – Electrically Erasable Programmable Read-Only Memory

ICSP – In-Circuit Serial Programming

LED – Light-Emitting Diode

I2C – Inter-Integrated Circuit

SPI – Serial Peripheral Interface

UART – Universal Asynchronous Receiver-Transmitter

IDE – Integrated Development Environment

DC – Direct Current

3D – Three Dimensional

ARM – Advanced RISC Machine

RISC – Reduced Instruction Set Computer

QSPI – Quad Serial Peripheral Interface

WI-FI – Wireless Fidelity

V – Volt

VSYS – Raspberry Pi's External Power Input

GPIO – General-Purpose Input/Output

IoT – Internet of Things

DIY – Do It Yourself

PCB – Printed Circuit Board

CPU – Central Processing Unit

MB - Megabyte

PSRAM – Pseudo static Random Access Memory

UI – User Interface

API – Application Programming Interface

HTTP – Hypertext Transfer Protocol

DSL – Domain Specific Language

OOP – Object Oriented Programming

WORA – Write Once Run Anywhere

JVM – Java Virtual Machine

IoC – Inversion of Control

DI – Dependency Injection

REPL – Read-Eval-Print-Loop

HAL – Hardware Abstraction Layer

TCP – Transmission Control Protocol

REST – Representational State Transfer

URL – Uniform Resource Locator

MQTT – Message Queuing Telemetry Transport

QoS – Quality of Service

SSE – Server Sent Event

CO2 – Carbon Dioxide

SMART – Specific, Measurable, Achievable and Time-bound

SWOT – Strengths, Weaknesses, Opportunities and Threats

CAD – Computer Aided Design

.stl – Stereolithography

mm – Millimeters

°C – Degrees Celsius

ASTM – American Society for Testing and Materials

UV - Ultraviolet

MPa – Megapascal

ksi – Kilo pound per Square Inch

GPa - Gigapascal

J/m – Joule per Meter

ft-lb/in – Foot-pound per Inch

JSON – JavaScript Object Notation