Group 34: Sentry Tracking Attack Turret

Code name: S.T.A.T

Nicholas Martucci	*Electrical Engineer*
Carlos Garcia	*Electrical Engineer*
Devak Sharma	*Computer Engineer*
Christian Solanet	*Computer Engineer*

Review Committee

Dr. Wei Sun	*ECE*	Professor
Dr. Qifeng Li	*ECE*	Associate Professor
Dr. Suboh Suboh	*ECE*	Associate Professor

Mentor

Sreeram Sundaresh

1.0 Executive Summary

The S.T.A.T (Sentry Tracking Attack Turret) is an intelligent, vision-guided tracking system that features dual-microcontroller architecture. It includes AI-assisted target detection and real-time pan-tilt stabilization. This system is developed at the University of Central Florida as part of the senior design effort that integrates and implements embedded control, mechanical actuation, and computer vision to accomplish accurate, quick, low latency tracking of objects in real-time.

The core of S.T.A.T is a primary microcontroller that is responsible for fine tracking the object. This microcontroller processes a bounding-box that contains position data from a HuskyLens and combines it with gyroscopic and acceleration data from an IMU (Inertial Measurement Unit) to predict motion of a target. The MCU will then execute a closed-loop PID control algorithm that runs two servo motors that are configured to the pan-tilt mount. This will ensure the system is stable, has smooth motion, and is responsive in a dynamic environment. The primary microcontroller will also run several other peripherals such as an audible buzzer, haptic indicators, and status LEDs.

A secondary microcontroller serves as the backbone of the wide area detection subsystem. This microcontroller interfaces with two stationary cameras whose captured images are processed using a light machine learning model that runs detection/recognition. The model will output a bounding box with coordinates of several objects in view along with other data that is transmitted to the main MCU via a peripheral interface (e.g. SPI or UART) with a low-latency messaging protocol. This allows for a divided workload between two coordinated microcontrollers to lessen the strain on just one MCU and ensure deterministic behavior.

The S.T.A.T is intended for applications that require autonomous tracking, sensing in multiple directions, or ones that require real-time tracking in dynamic environments, which includes robotics, guidance systems, or surveillance modules. Since this system has modular architectures (software and electrical), it is able to be scaled to include extra cameras, more advanced motors, or heavier machine learning models. The system also supports interoperability with external communication systems, which allows use on a larger scale.

This document describes the design cycle of S.T.A.T in a detailed manner. The details include the part-selection, embedded software architecture, control algorithms for real-time scenarios, and more. The mechanical and electrical constraints are explained, along with their impact on the software, such as how it affects the pan-tilt actuation system. We also discuss parallel tasking with stationary cameras, processing image data with machine learning models. The report includes multiple tables, flowcharts, firmware design information, communication protocols, and testing. The final chapters highlight the results, constraints, and recommendations for further development.

2.0 Project description

Our project is not groundbreaking or something no one has thought of before. This project is to show the wide range and thoughts that engineers must consider. We all need each other to move forward, and one wrong step could mean the difference between keeping our country safe or tumbling down around us.

Our group decided to build a system that can track a target in a set coordinate system and be able to take it out using a turret. This is no easy task as multiple sensors need to be in communication along with needing to know what exactly to look out for. Should it look for a sudden movement or a certain color? Should it be if anything is sensed in this area, we should start tracking it?

How is this accomplished? Let us start with the subsystems we have. First is the power system/access box which will take a 120V/12V AC/DC converter from a wall plug in and then step the voltage down for further use. Voltage will then be distributed from there to the MCU, LEDs, buttons, and all other peripherals. We will have separate access panel which will house the power system along with status LEDs, buttons, kill switches, etc. for this project. All human inputs and status updates will be on this box. We will have LEDs for status such as friendly or enemy target as well as to know if our turret has locked on. We will have switches to turn on our devices as well as emergency kill switches in case something happens that is unexpected. Another subsystem we have is the Turret substation which will house the MCU due to the fact that the turret with its motors and servos will need most of the wiring connected to it. We plan on having this project be able to break down, which will be explained in a later section so having less wired connections externally will help us to stay portable and user friendly. Our last subsystem is the Sentry Camera set up along with our HuskyLens sensor on the turret itself. The Sentry Cameras will be used for early detection of an object and will allow our system plenty of time to read the target, determine if it's friend or foe, and give ample time to respond to this object. The Sentry Cams will send over the target location to the HuskyLens which is attached to the turret. The turret will lock on to the object's position and will first turn on the laser diode to show we have locked on to said target then we will fire a dart at the target.

To help with the setting up of our system, allowing it to be repositioned, we have our systems connected by some external wires. As stated before, our access box will serve as power to the whole system. We will have wires that come out of this box that will connect and power the turret as well as the sentry cameras. We will want to position the turret and cameras, however, to minimize not only latency but also the offset of the camera coordinates from the turret. This can be done by hardware, however, to avoid mistakes with our control, placement of the turret and camera should be closed and allow transition and translation of data to be easy.

2.1 Project background/motivation

The design of our project is set up to mimic how a battlefield would work in a real-life military situation. We have Sentry cameras to sense our target. This could be representative as a pilot giving coordinates to the naval ship which would have the artillery shoot the target in question a thousand miles away. For simplicity, our information is communicated by wires. The pilot would give coordinates in which case the ship would have to adjust to those coordinates. Once the coordinates are locked on by the turret, the ship would be able to shoot, taking out the target. Just like our project, which requires data from the Sentry cameras to get the initial location. Then our turret must lock-on shown by the laser diode and be able to shoot the target as well.

2.2 Current technology/Past projects

2.2.1 - Military technology – C-RAM

The C-Ram stands for Counter-Rocket, Artillery, and Motor. This technology will sense incoming projectiles and be able to counter them using predictive planning and missiles to stop them. This relates to our project in the sense that we will want to be able to see our target and stop them in their tracks by intercepting them. C-RAM and our STAT require the use of sensors to detect and guess how the target is moving to accurately suppress the threat.

2.2.2 - Traffic Stop Watchdog

This project is taking slight inspiration from one of our own engineer's brothers and everyone's mutual interest in this type of technology. A past project called Traffic Stop Watchdog had a camera mounted on a movable base that would be able to scan a room and use image detection to determine if there was an officer present in the room based off the color of their shirt. The camera will then keep the officer in the center frame while they perform their duties. The scanning function of the Traffic Stop Watchdog served as inspiration for the two stationary sentry cameras that we will be using. The tracking functionality served as inspiration for the pan-tilt mount which will ultimately follow our target object. These cameras will work together to detect, locate, and follow an object of our choosing.

2.2.3 - Nerf Dart Turret

Other projects have tried to build automated Nerf style turrets, but most stop short of full autonomy. One example is the Nerf Dart Turret posted on Printables. That design uses a computer and WiFi link to control the turret, letting the user aim and fire remotely. It's more about showing off the mechanical launcher than tracking a target, since the system doesn't detect or move to an object on its own.

Our design builds on this idea but adds several important layers. Instead of relying on manual control, the turret uses two stationary cameras to constantly watch the area. When motion is detected, the turret wakes up and its own onboard camera takes over for precise aiming. IMU provides orientation feedback to smooth turret motion and prevent overshooting. These additions make the system capable of detecting, tracking, and firing automatically, without operator steering.

2.3 Objectives and goals

The main goal is to be able to acknowledge and track a ground target using the 2D coordinate system set up by the sentry cameras on the housing station. We also want the turret to enter a fully "locked on" state. Other goals can be found below.

Main goals

- Build a stationary turret capable of detecting and tracking an object within a 210° arena.
- Turret locks on to target on ground with data from sentry cameras with laser within 2 second of receiving position data.

- Access box signals that an object has been detected.
- Turret hits target with 60%

Advanced goals

- Turret hits target with nerf dart with above 80% accuracy
- Be able to differentiate between friend and foe at the same time

Stretch goals

- Be able to track aerial targets moving in x, y and z axis.
- Be able to lock on to aerial targets based on data from sentry cameras.
- Be able to see laser from diode on aerial targets before firing with in 500ms of receiving position data.
- Be able to hit target in air with nerf dart 75%.

Objectives

Main Objectives

- **3D Printed Housing Station** – Provide a stable base for the turret and secure mounting for the sentry cameras at the required height. The housing will allow detachment of wiring and PCBs for safe storage, while ensuring stable support to prevent camera damage.
- **3D Printed Turret** – Custom design to integrate the shooting mechanism, HuskyLens sensor, and laser diode as a single moving unit. The printed structure allows for optimized motor startup, reliable sensor response, and the ability to tailor hardware/software performance beyond what off-the-shelf kits provide.
- **AC/DC PCB** – This PCB is needed for power. Battery would be sufficient, however, we are focusing on motion tracking and targeting, efficiency is a secondary goal. To allow us to work and worry about power at a later time we are using a wall plug in to power our system. This requires us to take a 120V/12V AC/DC converter wall plug into a PCB on the access board where we will then further regulate the voltage for other uses such as LEDs, buttons, MCU, and motors.
- **Sentry Cameras** – These cameras need to be able to detect that an object has entered its field of view, pinpoint where the object is on the screen, and relay this information to the MCU. Extensive research is needed to choose the correct camera for this operation. We need a camera that can take in the software and also give us back information. They need to be able to communicate with each other as well, if not the theoretical 2D coordinate system will not work as intended.
- **Huskylens sensor** – Motion tracking sensor that allows for full lock on attached to the turret. Using this sensor allows us to track the target after we are given the initial coordinates from the sentry cameras.

- **Laser Diode** – This is a visual verification that the turret has locked on to the target in case any failure happens when it comes to shooting the target.
- **Flywheel Motor Speed** – To obtain a consistent accuracy of 60% the speed for the flywheel brushless motors will be monitored and adjusted to keep a constant speed of 8,000–12,000 RPM

Advanced Objectives

- **Color-tracking system** – To be able to differentiate between friend and foe we would need to preprogram our sentry cameras to only detect an object of a certain size as well as a certain color. For example, if we have a green ballon in front of the camera and a red balloon in front of the camera the camera would only send position data of the red balloon since we would only want to detect enemy targets.
- **Accuracy –** To improve the accuracy of our turret, higher resolution cameras (e.g., Wyze Cam or Pi Camera) paired with a single board computer such as the Raspberry 4 can be used to run more advanced computer vision algorithms (OpenCV-based color/blob detection or lightweight object recognition). This setup would allow more precise detection and tracking while offloading the heavy image processing workload from the ESP32-S3, keeping it focused on real-time control of the mount and turret.

Stretch Objectives

- **(X, Y, Z) Location** - Use the apparent pixel size of the object in one or more camera feeds, combined with the camera's intrinsic parameters (focal length, sensor size, distortion coefficients), to estimate distance. Then calculate the objects X and Y coordinates by mapping the object's image centroid to real-world coordinates via camera calibration. This will need to be done within 20 – 50ms, with the maximum being 100ms.
- **Predictive Tracking -** Use a predictive tracking filter (Kalman / Extended Kalman / Constant-acceleration model) to estimate future target location at projectile impact time. This will need to be done within 10ms.
- **Faster lock on** – Achieving a faster lock on to the target is possible through a few different ways. We could have multiple ESP32 to split up the utilization of our system, allowing for faster processing of data. We could also prioritize lock on using FreeRTOS allowing this task to be at the forefront of data processing. The last way is reducing the camera load and using a lower resolution on the Huskylens could allow for that process to end earlier, allowing the laser diode to act faster as well. The lock on should take no longer than 200ms.
- **Accuracy Improvement –** The system will use predictive tracking algorithms and confirm that the system is locked onto the target for more than 1s. It will then shoot the dart and accurately strike the target 95 times for every 100 attempts.

Software Specific Objectives

- The system must be able to generate a world coordinate system that each camera can transform its local coordinate system to move the pan-tilt mount accurately.

- The two stationary cameras must be sampled one after the other and take precedence over whichever camera detects an object first.
- The two stationary cameras must determine an estimated distance based off the object being a known size
- The two stationary cameras must be able to ignore objects too large or too small to avoid hitting humans or objects too far away
- The system shall store the detected objects with a flag marking if they have been previously detected by the sentry cameras
- The software shall use the distance detected to determine what external indicators and peripherals should be activated or deactivated.

2.3.1 Hardware Basic Goals

The turret operates on a simple principle. Detect a target, track them, and fire safely at close range. Two stationary cameras monitor the area, waking up the turret when motion is detected. These cameras act as the first line of detection; they're always watching but don't drain power since they're not constantly processing. Once triggered, the turret's own camera takes over aiming, making sure it only fires when targets are within about five feet.

The mechanical system needs different motor types working together, a stepper motor handling horizontal rotation, giving smooth sweeping motion across the target area without the jerky movement you'd get from a regular DC motor. Vertical tilting uses a servo motor, which provides enough precision for elevation adjustments without getting too complex. The servo can't rotate continuously like the stepper, but it doesn't need to for simple up and down movement. The projectile system uses dual flywheels driven by a brushless motor, keeping dart speed consistent for better accuracy. The flywheels spin up to speed and stay there, so when a dart gets fed between them, it launches with the same velocity every time.

Power and safety controls sit in a separate access panel, positioned where users can reach it safely without getting in front of the turret. The system runs off a 12-volt wall adapter that is certified to take advantage of its safety benefits like overvoltage and overcurrent protection. Internal circuits then step-down voltages for the electronics, motors, and cameras since they all need different power levels. The control panel has the key safety features: an emergency stop button for instant shutdown, and LEDs showing if the system is powered, armed, ready to fire, or has a fault. A buzzer gives audio alerts too, which helps during demonstrations.

Safety drove most of the design decisions from the start. The launcher only fires soft foam darts like Nerf darts with motor speeds capped and measured, so they don't hit too hard and have higher accuracy. This keeps the demo safe for a classroom setting, which was always the main goal. The emergency stop provides backup protection if something goes wrong.

Splitting the design into two parts (control panel and turret) made everything safer and more reliable. It's easier to build this way since you can work on each part separately, and demonstrations can't accidentally become dangerous because all the safety controls are

physically separated from the shooting mechanism. The modular approach also makes troubleshooting easier when something inevitably goes wrong during development.

2.3.2 Software Basic Goals

The software is built on the ESP-IDF/Arduino framework for ESP32-S3, which provides access to FreeRTOS multitasking, hardware timers, and communication devices. On the stationary (sentry) cameras, the built-in camera driver library is used for low-resolution frame capture, while lightweight OpenCV-style image routines (i.e. frame differencing or blob detection) run locally. Each node uses the ESP-NOW wireless library to send and receive data between devices with low latency and high throughput. The HuskyLens on the pan-tilt mount is accessed through its dedicated HuskyLens library (UART/I2C) to read bounding boxes, object IDs, and lock state. The ESP32-S3 microcontroller uses the FreeRTOS to separate the responsibilities of communication, motor control, safety monitoring, and ensure the turret remains responsive while processing vision data.

The laser and camera are coordinated in a hierarchical fashion. The stationary cameras on the left and right side of the housing station perform coarse detection of motion or color and transmit only target coordinates rather than full images either wirelessly or through UART/I2C. The ESP32-S3 moves the pan-tilt mount to point the main camera at the suspected target region. Once in view, the HuskyLens takes over for fine tracking and object/image recognition. A PID control loop continuously adjusts the turret until the target is centered. When the HuskyLens confirms a stable lock, the laser is enabled through GPIO controlled driver circuit. For safety measures, the software enforces interlocks and mechanical limits before activation.

A PID controller will be used to stabilize the mount's horizontal movements (driven by stepper motor) and vertical movements (driven by the servo motor). This controller utilizes the IMU (gyroscope and accelerometer) to measure the angular velocity and acceleration, while the PID algorithm computes the proportional, integral, and derivative errors to adjust the motor outputs, ensuring stable and accurate orientation of the mount. A Complementary or Kalman Filter is applied to fuse the sensor data, so that the noise from the accelerometer and drift from the gyroscope is reduced, yielding an accurate and stable angle estimate.

The display design provides real-time feedback to the operator. To indicate different stages of operation, multiple LEDs will be used. When the system is powered, a green LED light up to indicate that the system has received power. A blue LED will light up to indicate that the stationary cameras are operating and are currently scanning objects. If an object has been spotted by the stationary cameras, the orange LED will light up. Once the pan-tilt mount locates the object, if it detects an enemy, there are three possible LED outputs. If an enemy is detected but it is beyond the 10-foot range, the orange LED will stay on. If an enemy is detected and it is inside the 10-foot range, the red LED will turn on. If the enemy is within 5 feet of the pan-tilt mount, the red LED will start rapidly blinking.

The HuskyLens camera used for object detection comes equipped with a built-in 2-inch screen that displays the camera visuals in real time. This screen will be used as secondary verification that the camera is correctly detecting and identifying the objects in its FOV.

The primary verification that the camera and mount are working properly will be the laser diode that pinpoints the target when identified.

2.3.2.1 Why didn't we use Arduino or Raspberry Pi?

The ESP32-S3 was specifically chosen because of its dual-core processor, which allows it to handle multiple functions simultaneously. One core can run vision and communication, while the other core handles peripherals and control loops. With 45 programmable GPIO pins, the ESP32 can support many types of interfaces such as displays, cameras, and sensors through UART, SPI, or I2C. The ESP32-S3 is the smallest and lightest microcontroller out of the three options, so the mount won't be weighed down by it, and provides leeway when choosing the cameras and other components. It costs around $10-$15, making it cost effective.

Table 1 Comparison of MCU selection

ESP32-S3	**Raspberry Pi Pico 2 (RP2350)**	**Arduino Uno R3 (ATMega328P)**
Dual Core 32-bit Microprocessor at 240MHz	Dual Core Microprocessor at 150MHz	Single Core 8-bit Microprocessor at 16MHz
512KB SRAM	520KB SRAM	2KB SRAM
45 programmable GPIOs	26 programmable GPIO pins	Supports audio
Low power management	Supports audio	1 USART, 1 SPI, 1 I2C interface options
Supports multiple interface options	Supports multiple interface options	Brown out detection
4MB-64MB of flash memory	4MB of flash memory	39 GPIO pins
Dimensions: 21.7 x 17.8mm, 1.2g	Dimensions: 51 x 21mm, 3g	Dimensions: 68.6 x 53.4mm, 25g
Input voltage of 3.3V	Input voltage of 3.3V	Input voltage of 5V
Ranges from $10-$20	Ranges from $5-$15	Ranges from $20-$30

2.4 Requirement Specifications

Below you will find our Specification requirements for demo purposes (highlighted in orange) along with other requirements we have set for ourselves to further our knowledge in real world engineering aspects during this project.

Table 2: Specification Requirements

Parameter	Description	Target Value and Unit(s)	Market requirement corresponding #
Sentry Range Detection	Sentry cameras must be able to detect an object that has entered its desired range and create a bounding box around said object.	Greater than eight feet	1,3
Success Rate of Detection	This would be the success rate of sentry camera detection of object along with the turret detection and laser diode lock on.	Greater than 90%	1,3,5
Turret Reaction Time	The time that it takes the turret to wake up, position itself towards the object, detect the object, and determine if it should lock onto it must be quick.	Less than or equal to 2 seconds	1,3
Success Rate of shooting dart	After the turret is locked on with the laser diode this would be the percentage of accuracy of being able to hit the target with a nerf dart.	Greater than 60%	1,3
Size	The dimensions of the housing station combined with the dimensions of the turret must be able to fit comfortably on a school desk.	Size of desk is about... Length less than equal to 30 inches Width less than equal to 24 inches	2,4,5
Weight	We want this to be portable so the weight must be manageable by the average person	Less than 25lbs	4,5
Connectivity	The delay between cameras taking a sample and sending it to the MCU which then sends the data to the HuskyLens attached to the turret	Less than or equal to one second	1,3,5
Setup/Tear down time	The device does not need to be set up quickly because it is not	Less than 5 minutes	2,4,5

	used in high-risk scenarios with a strict timing requirement. The device does not need to be taken down as it is designed to be installed and left alone.		
Cost	This is a project design by college students so the cost should not be very high	Less than $1500	all

Table 3 – Marketing Requirements

Below can be found the marketing requirements our group has settled on for this project.

Marketing Requirements
1. Fast detection and fast lock on times 2. Ease of physical set up for user 3. Ease of use for software, only need to change what target to detect 4. We want to make sure this system is small enough to move around to reposition if needed (aka Portable) 5. Cost efficient

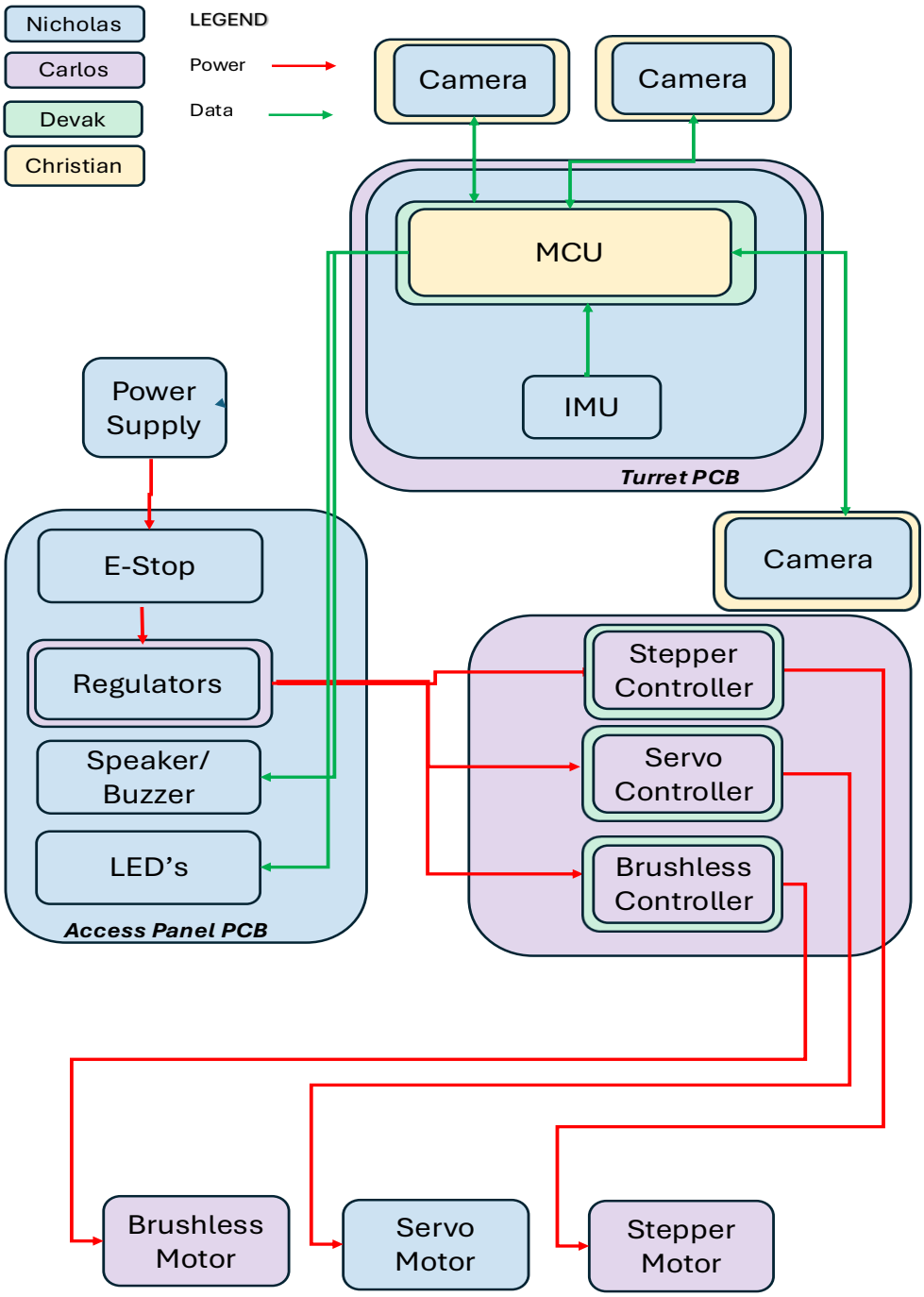
The table below is meant for showing more specifics on both the hardware side and the software side of quantitative values we expect during demonstration and test.

Table 4 - System Requirements

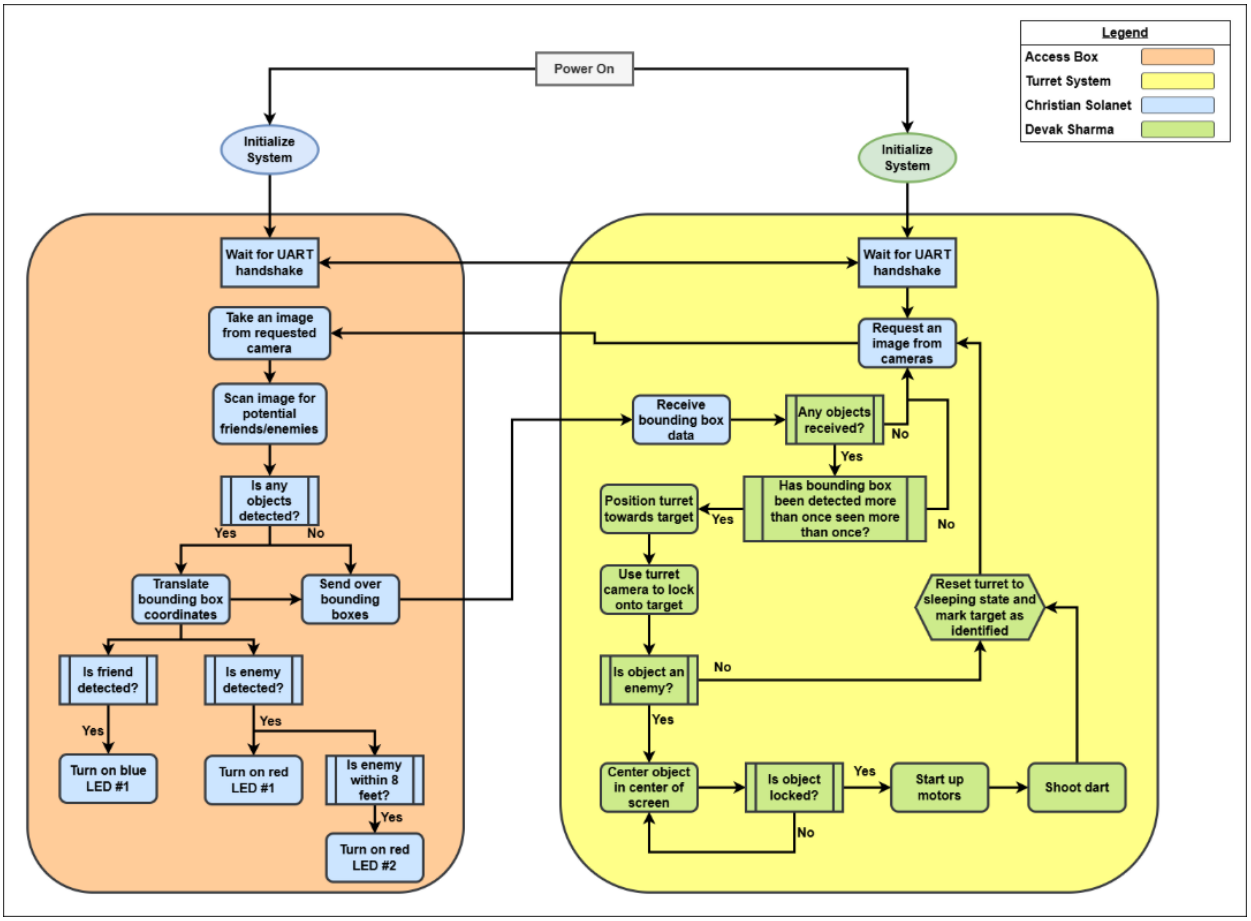
Component	Parameter	Subsystem and description	Target Value and Unit(s)
Cameras	Field of view	Software – The two stationary cameras need to provide an overall FOV that is sufficient to detect objects around the area.	Greater than 90 degrees
Cameras	Sentry Detection Time	Software – Two cameras need to be able to detect an object that has entered the desired range to then send a signal to wake the turret up	Detect target with at least 8 feet from cameras

Cameras	Turret Detection Time	Software – The turret camera needs to be able to detect an object and aim.	Detect/aim in < 2 seconds
Servo Motor / Stepper	Pan/Tilt angles	Hardware- The turret will be able to pan and tilt to the minimum angles that the fixed cameras are capable of detecting.	Pan- 210° Tilt- 90°
IMU (accelerometer / gyro)	Raw data processing	Hardware – Take in raw data to input into PID controller to stabilize signal to send over to the turret for further use.	Stabilize
Access box	User input and status updates	This box will act as the user input for the system. We will have buttons to turn on the system and emergency stop as well. LEDs indicate status such as friendly object or enemy object when a target is detected.	LEDs and buttons turn on with in 500ms of expected turn on

Hardware diagram – Figure 1



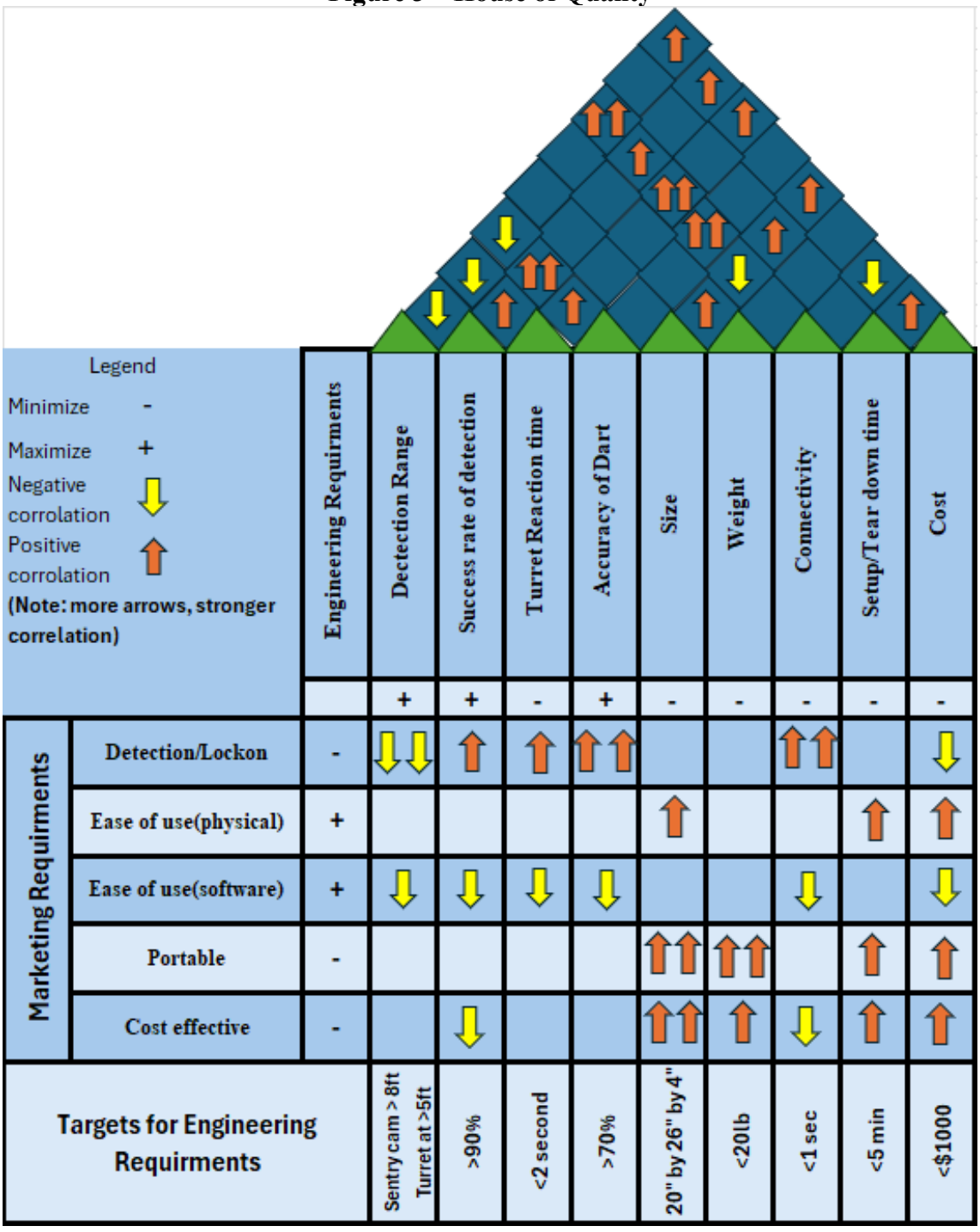
Software diagram – Figure 2



2.5 House of Quality

Below can be found our House of Quality for this project. As we can see there are plenty of interconnecting parts that all depend on each other such as connectivity and detection range of our object. However, cost as we see, can be saved a little from the few changes we have made in this project such as 3D printing when needed.

Figure 3 – House of Quality



3. Chapter 3 - Research and Investigation

3.1 - Components

Now we will talk about the types of components we will be using. We will need to deep dive into the types of IMUs we can use, the types and ways to use LEDs, and some buzzer components to list a few. We will start with a short summary of each component,

dive into the different types, create a table that compares pros and cons, then end the specific section deciding which part we will use.

3.1.1 - Types of LEDs

LEDs have various uses in many industries. In our case, we will be using LEDs as signals for ourselves based on what is happening in our environment. We will have LEDs for on status of our project, the status of the object in front of us, and lock on status just to name a few. We can use the simple amazon through hole design but in the spirit of this report let's look at a few others to see if there are any benefits to using something other than the most common type of LED. The LED we choose we want to be able to have the access box go around the LED with the LED or some kind of light popping out of the box. We also want to be able to remove the top of the box and fix the PCB inside in case anything breaks, and we want to make sure the LED stays in tack during this and make sure if we hit it at all it won't totally break it. If we do, we need to be able to replace this light easily. Knowing all of this let's look at our options

3.1.1.1 - Through-Hole LEDs with headers

This is the most straightforward. Design the PCB with through holes to solder on female headers to the VCC end and one on the GND end. Then we can put the LED in the correct way on these headers and just make sure when designing the 3D box to make it the correct height so that the LED can be near sticking out of where it is on the board. This is the easiest in case anything breaks as well. The LEDs would not be attached to the 3D printed box in anyway in this configuration so swapping what's needed is much easier and is minimally invasive

3.1.1.2 - Panel-Mount LEDs

These are LEDs that come as a module and are expected to be connected by wires rather than from headers. These have about the same difficulty connection wise to the through holes but can extend farther if we need to increase the height of the access box since these are connected via wires instead of headers. The through holes can be just like this if we solder wires onto those but again, we want a small access box, so we probably don't need these types of LEDs but it's something to hold on to incase the issue arises later

3.1.1.3 - Surface-Mount LEDs + Light Tunnels

This method is perhaps the most interesting of the three. This method can allow for any height of the access box again since the light can travel through these tunnels to reach the top. This is often used in industries today, so this would make our project more realistic. We would buy normal surface mount LEDs such as 150060SS75000, and we would also buy PLP2-350 which is a light tunnel that would take the surface mount LED and project it through the entire rod. This would allow for us to solder on the LED to the PCB device and assuming we drill the whole directly where we need it, we would then stick this rod in the whole which would allow the light to travel through rod and outside the box enough for the user to see it. This is minimally invasive similar to our first option since this tube just goes over the light and it won't interfere with it at all. The only issue is the precision that will be needed for this step. Our PCB design can change at any point so need to re-drill a hole in our box will make it look unprofessional and messy.

Table 8: LED technology comparison

LED type	Mount Method	Connection	Height Flexibility	Ease of Replacement	Precision to Implement	Cost
Through- Hole with headers	Female headers soldered to PCB	Pressed on by headers	Fixed height due to header height and LED connection height	Very easy since we just pull out and replace	Low since just need to keep proper distance of headers	$0.10 to $.30 per LED
Panel – Mount LEDs	Wire terminals through panel cutout	Solder wires to PCB pads	Very flexible since we can just leave wire length attached and cut as needed	Easy, just unsolder wires from pads and resolder new part	Medium due to having to drill panel holes which no one on the team has done	$1 to $3 depending on brand/type
Surface mount with light tunnels	SMD soldered to PCB and tunnel inserted in drilled hole on access box	Surface mount soldering	Very flexible since we can just change the tunnel height	Medium – need to know if tunnel or light is the issue first. Tunnel is quick replace, part would require unsoldering and resoldering new part	Harder than rest since hole on access box needs to be exactly over the light on the PCB	$0.50 to $1.50 per LED and tunnel depending on height

3.1.1.4 - Decision on LED technology

We have decided to go with Through hole LEDs that can be attached by headers. This allows for a fixed height, so when designing the access box, we know what height to make it at. If the switches cause any issues with wires, they can be added if needed and won't cause many issues with wire management since they'll be in the access box. This is also the cheapest approach and easiest to change in case anything goes wrong when applying power. PCB layout is easier as well since we just need through holes and solder on headers to make this configuration work.

3.1.2.1 - LED part selection

Something that needs to be considered when selecting the LEDs is the forward voltage drop and the limiting current resistor. LEDs do not operate at a voltage rather can only their brightness is controlled by the rated current that comes in. For example, if an LED is rated at 20mA, and only 10mA comes through it the brightness of the LED will be less than expected. The reverse is true as well, the more current over the rated, the brighter it will be. We need to keep this within a reasonable range; we can't run the LED at a full amp if rated for 20mA rather we can run it at 25mA or 30mA to get a brighter LED if that's within the range of operation for the LED. To solve the resistor needed in front of the LED we can use a simple formula.

R = (Voperating – Vforward)/(Rated current)

So, if we are operating at 3.3V, with a forward voltage of 2V and rated current is 20mA we would end up with a current limiting resistor of 68ohms. Knowing this, let's investigate the types of LEDs we can use.

3.1.2.2 - Kingbright WP7113ID

This first choice of LED is your standard LED type. It runs at 2V and has a forward current of 20mA. It has a 5mm through hole design and costs about $0.20 per part. The part intensity is 200-400mcd which is slightly on the low end but is the most cost effective for our approach

3.1.2.3 - Lite-On LTL-4233

This LED has a forward voltage of 2.1V and has a similar current of 20mA. This is standard for the LEDs we have seen, so this does not affect our power too much when designing. This model is more expensive, around $0.40 per part. Its brightness is where it shines with a range of 2000 to 4000mcd. This is way over what we need but is not a bad idea to have since it still only runs at 20mA and we can adjust the brightness easier if needed.

3.1.2.4 - Vishay TLHG5400

This LED has a forward voltage of 2.2V and still has its forward current of 30mA. The higher current could become an issue when it comes to the power design of our project later on since the motors will take up so much current already. Its intensity is brighter than the Kingbright version of 800 to 1600 mcd. This is more within the range of luminosity we want. It has a through hole footprint of 3mm, and it has an average cost of $0.25 per part. This is a nice model but might be left as a backup if others are not as bright

3.1.2.5 - Cree C503B-RAN

This LED has a forward voltage of 2.1V but has a similar issue to the previous part where rated forward current is at 30mA. It does, however, have the ability to be much brighter than the rest of the models, being at 3000 – 6000mcd. It has a slightly bigger size, being a 5mm through hole. This part is more expensive, with the average cost being $0.60 per part. This part seems to be overkill, but a good premium option to take things up a notch if needed.

3.1.2.6 - Everlight 334-15/T1C1-4WYA

This part is closer to the knightbright in specs. Similar forward voltage and current of 2V and 20mA respectively. Its luminosity is only at 20-40mcd but its viewing angle is 120 degrees whereas the rest were 60 or 30 degrees. It has the bigger size through hole fo 5mm and its cost is around $0.20

Table 9: LED part comparison table

LED Part #	Diameter (mm)	Forward Voltage (V)	Forward Current (A)	Luminous Intensity (mcd)	Viewing Angle	Cost ($)
Kingbright WP7113ID	5	2.0	20	200-400	60	0.15 - 0.25
Lite-On LTL-4233	5	2.1	20	2000-4000	30	0.30 - 0.45
Vishay TLHG5400	3	2.2	30	800-1600	60	0.20 - 0.30
Cree C503B-RAN	5	2.1	30	3000-6000	20	0.50 - 0.70
Everlight 334-15/T1C1-4WYA	5	2.0	20	20-40	120	0.18 - 0.28

3.1.2.7 - LED part decision

We have decided to use the Kingbright WP7113ID due to its average use across all specs. It's a cost-effective part that does what we want at simple ratings that fit well into our design. We can even solder directly on to the PCB if need be but that is just a backup plan if needed. A back up part we want to have is the Vishay TLHG5400 due to its brightness difference between the Kingbright. It does have a higher rated current, so we will need to change our current limiting resistor as well.

3.1.3.1 - Header Parts

Below we will look at two different types of headers, circular or square. The reason we want to have one of each of these or at least look at the options is due to the contact shape of the header. Using a square header with the round LED contact will allow limited contact points compared to using a circular header. The rest of our project can rely on a square header, so selecting a square header for universal use and keeping the circular for anything that may not work covers all of our bases while still looking for a cheap option.

3.1.3.2 - Square header - PPTC021LFBN-RC

This is the first square header we are looking at. It can handle up to 3A at 250V and has an operating temperature of –40 degrees Celsius to 105 degrees Celsius. All of these specs are well within our range of use. It's cheap as well only being about $0.24 per part

we get meaning if we use the Kingbright WP7113ID and this header, in total each part is only ~$0.40 which is again well within budget.

3.1.3.3 - Square header - SSQ-102-03-G-S

This is another square header made by Samtec Inc. Its more expensive compared to the previous option being around $0.75 per part meaning LED and header will be ~$1 per part which is a lot more expensive. Its ratings are 3A for current, 655VDC for voltage, and its operating temperature range is –55 degrees Celsius to 125 degrees Celsius. Again, these are well within range however the price is a huge driving factor to not choose this part,

3.1.3.4 - Circular header - SL-102-T-39

This circular header is from Samtec Inc. And has similar specs to the previous square header. Each contact can handle 3A and can handle up to 212VDC with the same temperature range of –55 degrees Celsius to 125 degrees Celsius. Its cost again is higher than the square headers at closer to $0.90 per contact. Its contact finish is gold, which is part of the reason for the increased price.

3.1.3.5 - Circular Header - 310-13-132-41-001000

This circular header comes with 32 pins to a pack for $7.38. This would give us plenty of contacts for what we need, and we can even replace parts with spare ones we have if something breaks. This is still over what we really need but let's still look at the specs. This gold contact finish header can operate between –55 degrees Celsius to 125 degrees Celsius. It can handle 3A for each contact at a voltage of 150VDC. This would be a good back-up to have due to its wide range of use but 32 pins at one time is a lot to get, especially when the cost of this could go to other components.

Table 10: Header part comparison table

Part #	Connection Type	Pitch(mm)	Current Rating(A)	Cost ($)
PPTC021LFBN-RC	Female Pin header	2.54	3	~$0.24
SSQ-102-03-G-S	Female Pin header	2.54	3	$0.70 - $1
SL-102-T-39	Female Pin header(Round)	2.54	3	~$0.92
310-13-132-41-001000	Machine Pin Socket(Round)	2.54	3	~$0.23/per contact

3.1.3.6 - Decision on Header Parts

We have decided to go with the square header PPTC021LFBN-RC due to its cost. This header paired with the LED chosen gives a simple, clean, cheap approach to status indicators for our system. The lower cost allows for the budget to be shifted else wear such as extra PCB space or extra budget for the power bank. If there are not enough

contact points for the square header we have decided to solder the LED into the header. This defeats the purpose of the “plug and play” approach we had when designing this approach, but it would still be easier to unsolder the LED from a header rather than from a board. The 310-13-132-41-001000 is going to be cheaper per pin but again this locks us into having to buy 32. If we order the PPTC021LFBN-RC we would only need to order what was needed which probably would not be more hen 10 to 15 which would only cost ~$4 instead of $7.38

3.1.4 - Buzzer components

For some extra peripherals, we have a buzzer component. We want this to trigger when the turret has locked onto an enemy target and is ready to fire. We will have some LEDs that indicate similar status, but we wanted an extra component to help with this as well. There are a few different types of buzzers so let's look at each in some details

3.1.4.1 - PCB-Mounted Passive Buzzer

This buzzer component is your basic component with a few functions. It requires a PWM signal from our MCU to operate and has different levels of sounds when buzzing. The passive buzzer is soldered directly to the board and is small, allowing room to be freed up for more complex components and parts. It can generate different tones as well so this would allow us to use the buzzer more for different reasons. For example, we can have a loud sound for when we are locked on and a lower sound for when we are tracking. The issue with this is in general the sound isn't as high as an Active buzzer but also the use of a PWM signal. This could complicate our design both software and hardware.

3.1.4.2 - PCB-Mounted Active Buzzer

An Active Buzzer is simpler compared to a passive buzzer. This component will just need a signal from the MCU that triggers for high and low signals and uses a pull up resistor in parallel to help with current for when it's not in use. It does not have varying sounds like the Passive, but it has a single loud buzzer sound. This component can also be soldered directly on the board and will have a much easier configuration for design as well compared to the Passive.

3.1.4.3 - Panel-Mount Buzzer

These buzzers are an extended version of the Active buzzers. These have wires connected to them and instead of being directly soldered onto the board, they are instead mounted onto the outside of the box. This would be useful if we wanted to see the buzzer; however, this causes a problem with being able to replace parts. We would have to have this part with female headers so we can attach and detach this part from the box whenever we change the PCB if it's mounted onto the box. This could cause more chances of wiring issues arising and messing with traces, which is what we are trying to avoid.

Table 11: Buzzer technology comparison table

Buzzer Type	Signal Required	Tone Flexibility	Volume	Mounting	PCB Complexity	Replacement

PCB-Mounted Passive Buzzer	PWM from MCU	Multiple tones possible	moderate	Soldered directly to PCB	Higher difficulty due to PWM configuration needed	Medium since we need to desolder part from board
PCB – Mounted Active Buzzer	Simple High/Low signal from MCU	Fixed signal tone only	Loud	Soldered directly on to PCB	Simple since just GPIO from MCU and pull up resistor	Medium since we need to desolder part from board
Panel – Mount Buzzer	Simple High/Low signal from MCU	Fixed signal tone only	Very Loud	Mounted to enclosure panel with wires	Medium - GPIO pin but extra difficulty with wire management	Can use headers to make process easier

3.1.4.4 - Decision on Buzzer Technology

We have decided to go with the PCB – Mounted active buzzer. This buzzer only has one tone, but we plan on using LEDs as the main status indicator and plan on using this buzzer for a specific indicator such as lock on to target. Also choosing this allows for minimal wiring issues as addressed in the Panel – Mount buzzer and the Passive buzzer as said has a harder signal to implement into the PCB where the Active is just a pull up resistor and a GPIO pin from an MCU.

3.1.5 - Buzzer Part Selection

For each buzzer we look at, something to consider is the recommended circuit in the datasheet. These components need to be readily available and easy to implement. If the circuit takes up too much space, doesn't use common parts, or in general it is too hard to implement, we can't move forward with this part.

3.1.5.1 - RS Pro 1710866

This through hole active buzzer has a rated voltage of 12V which is much higher than what we wanted to run this buzzer at. It does, however, have an operating voltage range of 3VDC to 24VDC. It has a rated current of 13mA and outputs 77dB. Its resonant Frequency is ~2200Hz. This part is priced at around $1.44 per part. Running at the 12V would not cause much of an issue since it's only 13mA and due to the small current, we could just run this buzzer through holes in the PCB with no extra circuit to help with the incorporation of this part into our system. We can just solder this to a connecting GPIO pin on the ESP32 and a pull up resistor just in case of any issues that arise.

3.1.5.2 - TDK SD1209T5-A1

This buzzer runs at a rated voltage we expected of 5V. It can draw up to 65mA which means we need to consider the circuit that we would need to implement this component. It has a nice sound of 80dB meaning the sound should be clear from a respectable distance. It has a frequency of 2.048kHz

3.1.5.3 - CEM-1206S

This buzzer runs at a cheaper price of around $0.85 per unit. This is good, but this will have additional cost due to it being a magnetic, active buzzer. We need to implement its recommended schematic due to the coil in the buzzer. Its current draw is about 45mA at its rated voltage of 5V at a frequency of 2.4kHz. Its schematic only requires a MOSFET transistor on the back end of the buzzer, and a flywheel diode parallel to the terminals on the buzzer. All parts are easily accessible and have a CAD footprint for easy installation onto a PCB. It has a higher pitches sound at 85dB, allowing for the sound to travel further then both parts mentioned above.

3.1.5.4 - AI-1223-TWT-5V-R

This is similar to the CEM-1206S with a few differences. Its current draw is less at the same rated voltage being less than 30mA. This could help with designing the schematic meaning we may not have to buy extra parts to implement the buzzer. It does also have a similar output sound of around 85dB. This model is very widely used in embedded designs and hobbyist, so reviews are easy to come by. It still does have CAD footprints allowing for easy PCB work, but this is a very expensive product at $3.16 per unit. Even with the extra cost for extra parts in CEM-1206S, the total for that would still be cheaper than this option.

Table 12: Buzzer Part comparison table

Part #	Operating Voltage(V)	Current Draw(mA)	Sound(dB)
RS Pro 1710866	12V	~13	~77
TDK SD1209T5-A1	5V	~65	~80
CEM-1206S	5V	~45	~85
AI-1223-TWT-5V-R	5V	~30	~85

3.1.5.5 - Decision on Buzzer Part

The decision for this part was easy, and we will be going with the CEM-1206S. It does require extra work in the schematics but based on reviews and ease of use in systems we have gone with this model. It has the operating voltage we want of 5V, a small current draw and effective cost for our system. Its sound is also among the higher ones we have seen based off research which we want since we plan on using this buzzer for mission critical status such as lock on to target or enemy identification.

3.1.6 IMU Choices

The IMU will be used for stabilization of the turret movement in your board. Due to our turret needing to be able to move on the x and y axis constantly and wanting to have a consistent straight shot we will need to stabilize our system. The IMU allows for the turret to operate fast but stable to the signals that sentry cameras and the Huskeylens data that will be sent via UART, I2C, or SPI. We want to also have an IMU that can be used across multiple processors and communicate in various ways to allow flexibility when it comes to design changes that might need to happen down the line in cases something

does not go our way. Having this component also be cheap can help push budget to other places such as better motors or servos or high-quality cameras.

3.1.6.1 - IMU with 6DoF

The main difference between IMUs is the degree of freedom. A degree of freedom (DoF) is an axis that the IMU is sensing. A 6 DoF IMU operates on an accelerometer and a gyroscope. The 3 – axis accelerometer is there to measure the linear acceleration of the turret, and the 3 – axis gyroscope is there to measure the angular velocity of the turret. Both will be needed to allow for targeting objects in front of it no matter the size of the object. These are going to have cheaper options compared to the rest but also not be as accurate and still have some noise in the system. It will consume less power, meaning less current will be needed in our system.

3.1.6.2 - IMU with 9DoF

If we want to make our station mobile the use of 9 DoF would need to be considered. This added 3 – axis magnetometer which acts like a compass for our system. This type of IMU is generally more used in robotics and drone projects due to having easier control sets of a set north is declared ahead of time. Since our project is stationary, we do not need an IMU with 9 DoF but if we were to expand on the system and make it mobile this would be something to consider

3.1.6.3 - Industrial grade IMU

These types of IMU can range in DoF and can have six or nine. Generally, these will be more expensive but also consume less power. The increased accuracy is also a nice touch and will be needed if we want to reach our stretch goal of hitting aerial targets.

Table 13: IMU technology comparison table

Option	Features	Pros	Cons
6 DoF IMU	Accel and Gyro	Simpler to implement and cheap	No magnetometer
9 DoF IMU	Accel, Gyro and magnetometer	Better orientation accuracy when system is moving	Can drift with EMI and higher cost
Industrial IMU	Low noise and better stability	Extremely accurate	Expensive and power hungry

3.1.6.4 - Decision of IMU tech

Based on this table, we have decided to go with the 6 DoF IMU. We want to have an accurate stationary system, so we do not need a 9 DoF IMU and we do not want to spend $100 on the IMU alone. The 6 DoF has what we need to stabilize the movement of the turret during location of the target, and we will make sure to choose a cheap option that has good accuracy and lower power consumption to help with our system.

3.1.7 The basic choice – MPU6050

This IMU is as it is listed, basic but efficient. This IMU model operates at 3.3V - 5V and has a default communication of I2C but can also use SPI in some breakout cases. It has six DoF(Degrees of Freedom) with three for gyroscope and three for accelerometer. We can find this cheaply online on Amazon.com, so accessibility is not hard either. An issue that does pop up is due to this model being an older one; many in the industry say this is obsolete and should not be used.

3.1.7.2 Upgraded choice – ISM330DHCX

This IMU has a lot of what we need at a higher price. This IMU operates with 3.3V to 5V, has one of the smaller current ratings of any of the systems talked about above, has I2C and SPI communication, and has better degree ratings depending on the type of board used. This model is used much more in systems today due to the improved accuracy it allows in systems as well as having lower noise which could help with motor or servo issues we could run into later. The only issue as said above is that one of these boards could cost as much as three MPU6050. Taking care of this IMU will have to be high on the priority list to help with budget costs.

3.1.7.3 Expensive but worth it – LSM6DS3

This IMU is another great option for better power management of our system. It has similar features to the previous IMUs with 6 DoF and running in a voltage range of 1.71V - 3.6V. It has two communication protocols of I2C and SPI allowing us to communicate with multiple components at once. It does include motion detection features which correlates with our project very well. It has a lower current draw of around .65mA and even has low power mode. The issue with this type of IMU is it is not precise as other IMUs which in this project is a huge issue

Table 14: IMU part comparison table

IMU	Pros	Cons	Voltage range	Power usage	Interface	Cost
MPU – 6050	Cheap and available	Older and has higher drift then regular IMUs	2.3V-3.4V	Low	I2C, some SPI	About $6 per unit
ISM330DHCX	High stability, low noise	Expensive	1.71-3.6V	Very Low	I2C, SPI	$20 per unit

LSM6DS3	Power efficient, widely used	Not as precise as others	1.71-3.6V	Very Low	I2C, SPI	About $10 per unit

3.1.7.4 - Decision of IMU

We have decided to go with the ISM330DHCX due to its flexible voltage, low power usage, and most of all its higher accuracy. We will be paying the price for this from one unit being $20. However, we feel lots of the benefits will pay off. With the use of I2C communication, we can save this portion to communicate with the ESP-32 and allows for simpler wiring when making the PCB. The SPI communication can be saved for communication with higher difficulty systems such as the Huskeylens we have on the turret.

3.1.8 - Flywheel Motors

The flywheel pair sets response time and shot energy. They need to spin up fast, hold speed during the dart pinch, and behave predictably on a 12V wall supply. The performance of these motors directly affects projectile velocity, response time, and overall system reliability.

3.1.8.1 - Brushed DC Motors

Brushed DC motors wear quickly and create EMI noise at high rpm. The brushes physically rub against the commutator, which generates both mechanical wear and electrical arcing. At the speeds we need of 10,000+ rpm, this is becoming a serious problem. The arcing creates broadband electrical noise that can interfere with the microcontroller, sensors, and PWM signals. We'd end up needing more filtering and shielding, which adds complexity. The brushes also need periodic replacement, and brush dust can contaminate other components. While these motors are cheap and easy to control initially, the long-term maintenance and EMI issues make them less suitable for our application.

3.1.8.2 - Stepper Motors

Stepper motors are inefficient and lose torque at the 10k to 15k rpm range we need. Stepper motors are designed for precise positioning at low speeds, not sustained high speed rotation. As RPM increases, their torque drops rapidly due to inductance limiting how fast current can change in the windings. To get adequate torque at high RPM, you'd need a much larger stepper and a more sophisticated driver, which defeats the purpose. Steppers also have resonance issues at certain speeds where they vibrate badly or even skip steps. The continuous current draws even when holding still also creates unnecessary heat and power consumption.

3.1.8.3 - BLDC Outrunner Motors

BLDC outrunners are efficient, lightweight, and work with off the shelf fixed wing ESCs. The outrunner design puts permanent magnets on the outside rotor, which gives you more torque for a given diameter compared to in-runners. The larger rotor diameter means

higher magnetic flux and better cooling since the magnets are exposed to airflow. Fixed wing ESCs are specifically designed for this type of motor in propeller applications, which is mechanically similar to our flywheel use case.

Mechanically, the 2312 frame fits our mounts, accepts a stiff 4mm adapter, and has enough thermal mass for repeated ramps. The 2312 designation refers to a 23mm stator diameter and 12mm stator height, which is a common size in the hobby market. This means parts are readily available and well documented. The thermal mass matters because each firing cycle heats the motor, and with repeated demos we need the motor to dissipate that heat between shots without overheating.

Table 15: Flywheel Motor Technology Comparison Table

Motor Type	Maintenance	EMI Noise	Torque at High RPM	Control Complexity	Thermal Performance	Cost
Brushed DC	High, brush replacement needed	High, creates interference	Moderate	Low, simple DC control	Poor, brushes generate heat	Low
Stepper	Low maintenance	Low	Poor, drops rapidly above 1000 RPM	High, needs specialized driver	Moderate, continuous current	Medium
BLDC Outrunner	Very low, no brushes	Very low	Excellent at 10-15 krpm	Medium, needs ESC	Excellent, good cooling	Medium

3.1.8.4 - Decision on Flywheel Motor Technology

We have decided to go with BLDC outrunner motors in the 2312 frame size. Within that frame size, KV rating determines the rpm per amp behavior on 12V. We compared 980, 1250, and 1400 KV options. The 1250 KV hits the sweet spot: about 10,500 to 12,000 rpm under load on 12V with quick ramps and manageable current draw. KV is the motor's velocity constant, expressed as RPM per volt with no load. A 1250 KV motor spins at 15,000 rpm unloaded on 12V (1250 × 12), but under the load of spinning a flywheel and launching darts, it slows to around 10,500 to 12,000 rpm, which is right in our target range for adequate rim speed without excessive noise or current.

Table 16: KV Rating Comparison Table

KV Rating	No-Load RPM @12V	Expected Loaded RPM	Torque per Amp	Spin-Up Current	Noise @ Performance
980	11,760	8,200–9,400	High	Low	Low
1250	15,000	10,500–12,000	Medium	Medium	Medium
1400	16,800	11,760–13,440	Lower	Higher	Higher

Table 17: Flywheel Motor Part Selection

Type	Typical Mass	Expected Dart Speed	No-load Current	Continuous/Burst	Why Pick It
Mid-tier 2312–1250 KV	~52–55g		~1–1.5A	18–20A / 28–32A	Good bearings/laminations; 4mm shaft for adapters
Budget 2312–1250 KV	~50–52g		~0.8–1.2A	15–17A / 25–28A	Works if de-rated; more unit-to-unit spread
Premium 2312–1250 KV	~55–58g		~1–1.6A	20–22A / 32–35A	Highest consistency;

For each motor we look at, something to consider is the bearing quality and shaft runout specifications in the datasheet or product reviews. These components need to have minimal vibration and good bearing life since they'll be spinning at high speed during every demonstration. If the motor has excessive vibration or the bearings are known to fail quickly and can provide inconsistencies in the dart accuracy.

3.1.9 - Flywheel Diameter Selection

Diameter sets rim speed for a given rpm, the inertia the motors must spin, and the contact path during the dart pinch. It governs response time, noise, and shot consistency. The wheel diameter is one of the most important mechanical parameters that affects both performance and system behavior.

3.1.9.1 - 65mm Diameter Wheels

The 65mm wheels are the snappiest option but need higher rpm for adequate rim speed. With a smaller diameter, you need to spin faster to achieve the same rim speed, which is what actually matters for dart velocity. Higher RPM means more motor noise, more aerodynamic noise from the spinning wheels, and more sensitive control since small RPM changes translate to bigger velocity changes. The contact time between the dart and the wheels is also shorter with a smaller diameter, which can reduce shot consistency if the dart isn't perfectly centered. The main advantage is the fastest spin up time and lowest inertia, which reduces the load on the power supply during startup.

3.1.9.2 - 70mm Diameter Wheels

The 70mm wheels thread the needle between the 65mm and 80mm options. At 11,800 rpm (our target firing speed), 70mm wheels give us about 43 m/s rim speed, which translates to dart velocities in the safe range for foam projectiles. The inertia is manageable enough that spin up takes under a second, but high enough that the wheels don't bog down noticeably when launching a dart. Testing will tell us if we need to adjust, but 70mm gives us room to move in either direction if the performance isn't quite right. The balanced approach makes this the safest choice for initial testing.

3.1.9.3 - 80mm Diameter Wheels

The 80mm wheels run at lower rpm for the same rim speed and have steadier shots. The larger diameter gives you more contact area and time with the dart, which improves consistency. The flywheels act like energy storage, and larger diameter wheels store more rotational energy for a given RPM. This makes the velocity more stable shot to shot. However, the increased moment of inertia means the motors take longer to accelerate the wheels to speed, and the current draw during acceleration lasts longer. This creates broader pulses on the power supply, which our staged startup choreography has to accommodate.

Table 18: Flywheel Diameter Comparison Table

Diameter	Rim Speed @ 9,800 rpm	Rim Speed @ 11,800 rpm	Relative Inertia	Spin-up Time	Shot Consistency	PSU Transient
65mm	33.40 m/s	40.16 m/s	0.862×	Fastest	Fair (short contact)	Lowest
70mm	35.92 m/s	43.25 m/s	1.000×	Fast	Good	Low
80mm	41.05 m/s	49.43 m/s	1.306×	Slowest	Best	Higher

3.1.9.5 Decision on Flywheel Diameter

We have decided to go with 70mm diameter wheels. The 70mm option gives us adequate rim speed without requiring excessive RPM, keeps spin up time reasonable, and provides good shot consistency without the heavy current draw of 80mm wheels.

3.1.10 - ESC Controllers

Each ESC communicates its BLDC flywheel and controls the ramp, timing, and braking. This directly determines how kind the system is to the 12V wall bus. The ESC selection is critical for managing the power supply choreography and preventing voltage sag during motor startup.

3.1.10.1 - Multirotor ESCs

Multirotor ESCs are battery focused and often lack a gentle soft start. They're designed for quadcopters where you have a large battery that can source hundreds of amps without voltage sag. The control algorithms assume power sources and use aggressive throttle responses for acrobatic flights. Many multirotor ESCs also implement active braking to stop props quickly when throttle is cut, which dumps regenerative current back into the supply. That's fine with a battery that can absorb it, but a wall supply can't handle reverse current and may fault or oscillate. These ESCs would require significant configuration changes to work properly with our wall powered system.

3.1.10.2 - Car ESCs

Car ESCs add bulk and have wheel tuned punch curves that don't help us. They're designed for RC cars where you need high torque at low speeds for acceleration from a standstill, which is a completely different requirement than spinning a flywheel at constant high speed. The throttle curves are nonlinear to give good low speed control, which makes our application harder to tune. Car ESCs are also physically large because they have to handle the shock and vibration of an RC vehicle, and the extra size creates packaging problems in our turret housing.

3.1.10.3 - Fixed Wing ESCs

Fixed wing ESCs provide forward only operation, soft start, brake off capability, adjustable timing, and active freewheel to reduce ripple. Fixed wing aircraft don't need reverse thrust, so these ESCs are simpler and more power supply friendly. Soft start means the ESC gradually increases current during the first few hundred milliseconds of throttle application instead of hitting the motor with full current immediately. This is exactly what we need for staged startup.

The brake off setting prevents regenerative braking, so when throttle is reduced, the motor just coasts instead of trying to push current back. Active freewheel (sometimes called damped light) synchronously rectifies the motor's back EMF instead of letting it ring, which reduces electrical noise on the power rail. Two identical ESCs get programmed the same way, and the endpoint is calibrated once. Using identical units

means we only must figure out the optimal settings once and then copy them to the second ESC.

Table 19: ESC Technology Comparison Table

ESC Type	Soft Start	Brake Behavior	Throttle Curve	Size	Power Source Assumption
Multirotor	Usually no	Active braking (regenerative)	Aggressive, linear	Compact	Large battery
Car	Variable	Active braking	Nonlinear, low speed optimized	Large	Battery
Fixed Wing	Yes, standard feature	Configurable, can disable	Linear, forward only	Compact	Battery but adaptable

3.1.10.4 - Decision on ESC Technology

We have decided to go with Fixed Wing ESCs. These provide the soft start and brake off features we need for power supply friendly operation. Endpoint calibration teaches the ESC what PWM pulse widths correspond to minimum and maximum throttle, ensuring both motors respond identically to the same control signal. Without this calibration, manufacturing tolerances in the ESC's input circuitry could cause the motors to spin at slightly different speeds for the same command, which would make darts curve in flight.

Table 20: ESC Part Selection

ESC Model	Continuous / Burst	BEC	Notes
Hobbywing Skywalker V2 50A	50A / ~70A	5V SBEC	Selected: Soft-Start, Medium timing, active freewheel ON
ZTW Mantis G2 45A	45A / 55A	5.6/7.4V @ ~4A	32-bit, adjustable BEC; compact alternate
Castle Talon 60	60A cont.	8A cont / 20A peak	Premium tuning; larger/heavier (overkill for v1)

For each ESC we look at, something to consider is the programming options and whether they support brake off and soft start modes. These components need to have clear firmware configuration options documented in the manual.

3.1.11 - Gear Ratio & Transmission

The transmission sets slew speed, holding stiffness, and pointing resolution at the turret. The ratio trades speed for torque and finer control, so the turret snaps to target and then holds steady. The gear selection directly affects how quickly we can acquire targets and how accurately we can hold aim during firing.

3.1.11.1 - Direct Drive

Direct drive is simplest but offers no leverage. If you connect the servo output directly to the turret, you get one to one motion. The servo's torque and speed are what you get at the turret. For our application, the servos don't have enough torque to hold the launcher steady against firing impulses and wind up when moving, so direct drive isn't viable. The lack of mechanical advantage means the servo would have to work much harder, drawing more current and potentially overheating during extended operation.

3.1.11.2 - Timing Belts

Timing belts are quiet but add compliance and tension complexity. Belt drives can work well, but the belt itself stretches slightly under load, which creates backlash and reduces pointing accuracy. You also need tensioners and proper alignment of the pulleys, and belts can slip if tension isn't maintained. The belt teeth can also wear over time, increasing backlash. For a demonstration system where we want reliable, repeatable performance without constant adjustment, belts add more complexity than they're worth. The tension system would also add weight and take up space in the turret housing.

3.1.11.3 - Spur Gears

Spur gears are compact, stiff, and provide exact ratios using COTS acetal or steel gear sets. Spur gears mesh directly so there's minimal compliance, and they don't require tensioning. Commercial off the shelf gears are available in standard pressure angles and module sizes, making it easy to get replacement parts or change ratios. Acetal gears are self-lubricating and quiet, while steel gears handle higher loads.

For the tilt axis, we're using 30T:90T (3:1) gearing. The tilt only needs to cover about 90 degrees of travel, so we can sacrifice some of the servo range to gain torque and resolution. The 3:1 ratio triples the servo's torque, giving us enough holding force to resist gravity and firing impulses. It also divides the servo step resolution by 3, so small PWM changes translate to finer angular adjustments. This is particularly important for tilting since the launcher needs to hold steady against the constant downward pull of gravity. Without the gearing, the servo would buzz and hunt trying to maintain elevation angle.

For the pan axis, we're using 1:1 gearing (direct drive through gears for mechanical coupling). Pan needs to cover a wide horizontal sweep to track targets across the full field of view. The SF3218MG servo provides 270° of travel, and we want to use all of that range for panning. Since the pan axis doesn't fight gravity and the horizontal inertia is

manageable, we don't need the torque multiplication that we need for the tilt. The 1:1 ratio preserves the servo's full speed and range while still providing a solid mechanical connection.

Table 21: Gear Transmission Technology Comparison Table

Transmission Type	Stiffness	Backlash	Tensioning Required	Maintenance	Ratio Options	Size
Direct Drive	Maximum	None (servo only)	None	None	1:1 fixed	Minimal
Timing Belt	Low (belt stretch)	Moderate	Yes, critical	Belt wear, tension checks	Flexible with pulley sizes	Medium
Spur Gears	High	Low (gear only)	None	Minimal	Discrete ratios	Compact

3.1.11.4 Decision on Gear Transmission Technology

We have decided to go with spur gears for both axes, but with different ratios. The tilt axis uses 30T:90T (3:1) to provide torque multiplication for holding against gravity. The pan axis uses 1:1 gearing to preserve the servo's full 270° range for maximum coverage. Going to 5:1 on tilt would give more torque and resolution but would make fine adjustments unnecessarily slow.

Table 22: Gear Ratio Part Selection

Gearset	Ratio	Output Stall estimated	No-load Output Speed estimated	Why It Fits
30T:60T	2:1	~4.4 N·m	~188°/s	Fastest slews; least stiffness gain; for very light heads
30T:90T	3:1	~6.6 N·m	~125°/s	Selected: balanced stiffness, finer resolution, moderate speed
24T:120T	5:1	~11.0 N·m	~75°/s	Steadiest aim; slower sweeps and higher inertia

3.1.12 - Pan Motor Selection

The pan motor determines how quickly we acquire a target, how still we hold heading while flywheels spin, and how quickly we settle between shots.

3.1.12.1 - Stepper Motors (NEMA-17)

Stepper motors with NEMA-17 drivers offer precise steps and high hold torque at zero speed, but torque drops with speed and drivers add complexity and resonance. Steppers are great when you need precise positioning and can move slowly, but they have issues at the speeds we need. The torque curve of a stepper drops significantly above a few hundred RPM, and we need faster motion than that for quick target acquisition.

Steppers also draw current continuously even when holding still, which wastes power and generates heat. The drivers can introduce mid band resonance where the motor vibrates badly at certain speeds, and you need tuning or dampers to fix it. More importantly, steppers are open loop, so if the motor skips steps due to a sudden load, you lose position and have no way to know it happened without adding an encoder.

3.1.12.2 - PWM Servos

PWM servos integrate motor, gearbox, and controller and accept simple PWM commands. A hobby servo has everything built in: DC motor, gear reduction, position sensor, and control electronics. You send it a PWM signal to set the desired position, and it handles the rest. The internal controller maintains position against external loads and provides damping to prevent oscillation.

This makes the system much simpler from a firmware and electronics perspective. You just need a PWM output from the microcontroller, no H-bridge, no encoder interface. Pairing with 1:1 gearing for pan gives us the full 270° range while the servo's internal gearbox provides adequate stiffness and torque for the horizontal axis. The servo itself has some backlash in its internal gearbox (typically under a degree), which is acceptable for our demonstration of accuracy requirements.

Table 23: Pan Motor Technology Comparison Table

Motor Type	Control Method	Position Feedback	Torque at Speed	Current Draw at Rest	Setup Complexity	Firmware Complexity	Backlash
Stepper	Open loop steps	None (unless encoder added)	Poor above 300 RPM	High (continuous)	Medium (driver needed)	Low to Medium	None intrinsically
PWM Hobby Servo	Integrated closed loop	Internal potentiometer	Good	Low (only when moving)	Very Low (just PWM)	Very Low (position command)	Low (internal gears)

3.1.12.3 - Decision on Pan Motor Technology

We have decided to go with PWM hobby servos paired with 1:1 spur gears. This combination gives us the simplicity of servo control with the performance we need for stable aiming.

Table 24: Pan Motor Part Selection

Model	Voltage	Speed	Stall Torque	Mass	Notes
SunFounder SF3218MG	4.8–7.2V	0.18→0.14 s/60°	20.5→22.8 kg·cm	~56g	Selected: metal gears, 270° range
Savox SC-1256TG	4.8–6.0V	~0.15 s/60° @ 6V	~20 kg·cm	~52–60g	Proven mid-tier if standardizing at 6V
JX PDI-6221MG	4.8–6.6V	~0.16 s/60° @ 6V	~20 kg·cm	~62–65g	Cost-effective alternative (shorter range)

For each servo we look at, something to consider is the gear material (metal vs plastic) and the feedback of potentiometer quality. These components need to have metal gears for durability under repeated operation and minimal dead band in the potentiometer. If the servo has plastic gears or excessive dead band, we can't move forward with that part.

3.1.13 - Tilt Motor Selection

The tilt motor sets an elevation angle. The tilt axis actually carries heavier loads than pan since it has to support the launcher assembly against gravity. Precision matters - we need crisp moves, quick settling for the vision loop, and steady holding against gravity and firing transients.

3.1.13.1 - Why Use the Same Servo

As with pan, we're avoiding the standstill current and resonance issues of steppers and the tuning plus backlash complexity of DC gearmotors. Servos keep electronics simple and hold position in closed loop. Using the same SF3218MG simplifies parts, wiring, and spares. Having identical motors for both axes means we only need to stock one type of spare, and if we damage a motor during testing or demos, we can swap it with the other axis temporarily to keep the system running.

The control code is also simpler when both motors have identical response characteristics. We don't have to write separate PWM timing or filtering for each axis. The tilt axis actually has heavier loads than pan since it has to support the weight of the launcher assembly and fight gravity constantly. The pan axis just rotates the turret horizontally without having to lift anything.

The tilt axis has to fight gravity, which creates a constant load that varies with elevation angle. When the launcher is pointed horizontally, there's maximum gravitational torque trying to droop the assembly downward. The servo has to continuously provide holding torque to resist this, and if the servo isn't strong enough or the internal gearbox has too much backlash, the aim point will sag. This is why we use 3:1 gearing on tilt but only 1:1 on pan. Using the same SF3218MG servo for both ensures we have plenty of margin, and the 3:1 external gearing on tilt multiplies the servo's torque, providing approximately 60 to 68 kg·cm at the output, which is more than adequate for gravity compensation.

Table 25: Tilt Motor Part Selection

Model	Voltage	Speed	Stall Torque	Mass	Notes
SunFounder SF3218MG	4.8–7.2V	0.18→0.14 s/60°	20.5→22.8 kg·cm	~56g	Selected for tilt too; consistent pairing
Savox SC-1256TG	4.8–6.0V	~0.15 s/60° @ 6V	~20 kg·cm	~52–60g	Lighter mid-tier option
Hitec D955TW	4.8–7.4V	~0.15 s/60° @ 7.4V	~25–29 kg·cm	~66g	Premium margin; heavier/costlier

For each servo considered for tilt, the same criteria apply as for pan: metal gears for durability and good potentiometer quality for minimal dead band and position feedback accuracy.

3.1.14 - Filament Selection for 3D Printed Components

Printed parts include the flywheel cage, motor and servo mounts, camera brackets, covers, and cable guides. Material choice governs impact toughness, heat margin near motors and ESCs, and print risk like warping.

3.1.14.1 - PLA (Polylactic Acid)

PLA offers crisp prints but is brittle and softens around 55 to 60°C. PLA is the easiest filament to print with and gives an excellent surface finish and dimensional accuracy on the first try. It doesn't warp or require an enclosure, and it sticks well to most print surfaces. This makes it ideal for prototyping where you're iterating designs and want fast turnaround.

However, PLA's low glass transition temperature means parts will deform if they get warm. Near the ESCs or motors, ambient temperature can easily reach 50 to 60°C during operation, which is enough to soften PLA and cause brackets or mounts to sag. PLA is also brittle compared to other filaments. It can crack under impact or sustained stress, especially if the part has thin sections or sharp internal corners. This makes PLA unsuitable for structural components that experience loads or heat.

3.1.14.2 - ABS (Acrylonitrile Butadiene Styrene)

ABS has better heat resistance (95 to 105°C) but warps badly and needs an enclosure. ABS can handle the temperatures around motors and electronics without softening, and it's tougher than PLA under impact. However, ABS shrinks significantly as it cools, which causes warping during printing. Large flat parts will lift at the corners, and tall parts can crack as internal stresses build up.

To print ABS successfully, you need a heated enclosure to keep the part warm during printing, which we don't have. The fumes from ABS printing are also unpleasant and potentially hazardous, requiring ventilation. Without proper equipment, ABS prints fail more often than they succeed, which wastes time and material during development.

3.1.14.3 - PETG (Polyethylene Terephthalate Glycol)

PETG is tough with about 75 to 85°C usable heat tolerance and low warp. It's ideal for functional, accurate parts on open printers. PETG bridges the gap between PLA and ABS. It prints almost as easily as PLA on an open printer but has much better mechanical properties. The heat resistance is adequate for our application since we're not expecting sustained temperatures above 70°C anywhere in the system.

PETG has excellent layer adhesion, which makes parts stronger and less likely to delaminate under stress. It's also more ductile than PLA, so it bends before breaking, which is better for parts that might get bumped or dropped during handling. The main downside of PETG is stringing, where thin threads of plastic get dragged between features during printing. This can be minimized with proper retraction settings, and the strings are easy to clean off after printing.

3.1.14.4 - Nylon

Nylon is excellent but hygroscopic and not needed for this build. Nylon has outstanding wear resistance and flexibility, which makes it ideal for parts that slide against each other or need to flex repeatedly. However, nylon absorbs moisture from the air rapidly, and wet filament produces weak, porous parts with poor surface finish. You have to dry the filament in a heated chamber before printing and ideally keep it dry during printing with a dry box.

Nylon also requires higher print temperatures (around 250 to 270°C) and sticks poorly to standard print surfaces. For this project, we don't have any parts that really need nylon's properties like gears or bushings, so the extra hassle isn't justified.

Table 26: Filament Technology Comparison Table

Filament Type	Heat Resistance	Impact Toughness	Warping Risk	Print Difficulty	Layer Adhesion	Dimensional Accuracy	Cost

PLA	Poor (55-60°C)	Low, brittle	Very low	Very easy	Good	Excellent	Low
ABS	Good (95-105°C)	Good	High	Hard, needs enclosure	Moderate	Good	Low to Medium
PETG	Moderate (75-85°C)	Excellent, ductile	Low	Easy on open printer	Excellent	Good	Medium
Nylon	Excellent (90-100°C)	Excellent, flexible	Moderate	Hard, hygroscopic	Excellent	Moderate	High

3.1.14.5 - Decision on Filament Technology

We have decided to go with PETG as the primary filament for structural components. PLA will remain useful for prototyping and non load bearing cosmetic parts. ABS is off the table because the printing challenges aren't worth it for this project, especially since we don't have a reliable enclosure setup.

3.1.15 - Laser Aim Indicator

The laser is a visual alignment aid for labs and demos. It shows the aim point at about 10ft indoors, improving operator and audience understanding. It doesn't measure distance, just provides a visible dot where the turret is aimed.

3.1.15.1 - Red Laser Modules

Red lasers at 650nm wavelength are the cheapest and simplest option. They use basic laser diodes that are widely available and well understood. However, the human eye is much less sensitive to red light compared to green light at the same optical power. A 1mW red laser appears significantly dimmer than a 1mW green laser in typical room lighting. However, since our demonstrations will primarily occur in indoor lab environments with controlled lighting, visibility remains adequate at short distances. A small increase in drive current or beam focus adjustment easily compensates for this reduced visibility.

3.1.15.2 - Green Laser Modules (Class 2)

Green lasers at 520 to 532nm wavelength in Class 2 provide clearly visible dots while staying in the safest laser class. The human eye's peak sensitivity is around 555nm (yellow green), and the sensitivity curve drops off as you move toward red or blue wavelengths. A 532nm green laser at 1mW appears about 3 to 4 times brighter than a 650nm red laser at the same power.

This means we can use lower optical power to achieve the same visual brightness, which improves safety margins and reduces concerns during safety review. Class 2 is considered eye safe under normal circumstances because the blink reflex protects your eye. You'd have to deliberately stare into the beam for several seconds to risk eye damage. The drawback for this component is the cost which is much higher than a red laser diode.

3.1.15.3 - Green Laser Modules (Class 3R)

Green lasers at 520 to 532nm in Class 3R provide brighter dots for well lit spaces. Class 3R can cause eye damage with brief exposure, but it's still low enough that it's allowed without extensive safety controls. If we end up doing demos in brightly lit spaces or outdoors, we might need to step up to Class 3R to maintain visibility, but that would require additional safety briefing and possibly laser safety goggles available for operators.

Table 27: Laser Technology Comparison Table

Laser Type	Wavelength	Class	Visibility in Room Light	Eye Safety	Cost
Red, 5V module	650nm	<5mW	Poor	Moderate risk	Low
Green, Class 2 module	520-532nm	≤1mW	Excellent	Safest, blink reflex protects	Medium
Green, Class 3R module	520-532nm	1-5mW	Excellent	Requires brief safety training	Medium

3.1.15.4 Decision on Laser Technology

After comparing all available laser options, we decided to use a red, 650 nm, 5 V laser module. This choice provides the best balance of cost, simplicity, and safety for our capstone project. The red module is inexpensive, readily available, and requires no special handling procedures or safety reviews. While the red dot is less bright than a green one, it remains visible at the 10ft indoor range where our demonstrations take place. The reduced optical brightness also contributes to a safer system for presentations and classroom environments.

In short, the red laser meets all visibility and functionality requirements without adding unnecessary cost or complexity. It integrates directly into our existing 5 V rail with minimal current draw.

Table 28: Component listing

Component	Supply Voltage	Current	Quantity
Buzzer - CEM-1206S	5V	45mA	2
LED -	Vop = 3.3V or 5V, Vf = 2V	20mA	~7
IMU - ISM330DHCX	1.71V - 3.6V	1.2mA - 1.5mA	1
(Laser diode)	5V	1mA	1
Huskeylens	3.3V-5V	~230mA-320mA	1
Arducam Mega SPI Camera 3MP	3.3V	56mA-136mA	2
ESP-32	3.3V-5V	~50mA-100mA	2
Sunfounder High Torque Digital Servo	4.8V to 7.4V	~2.3A +/- 10%	2
(Brushless DC Motor)	12V	18A start up 10A continuous	2
Total power 3.3V	3.3 * (20mA * 7 + 1.5mA + 136mA + 2*100mA =3.*4775mA = 1.4325W		
Total power 5V	5V * (1mA + 45mA*2+1mA + 230mA) = 5*352mA = 1.76W		
Total power 7V	7*2.3A = 16.1W		
Total power 12V	Startup motors at same time 20A * 2 * 12V = 480W Stagger start motors 20A * 1 * 12V + 10A * 1 * 12V = 360W Worst case power draw is 480W		
Total power in system	**480 + 1.432 + 1.76 = 483.192W**		

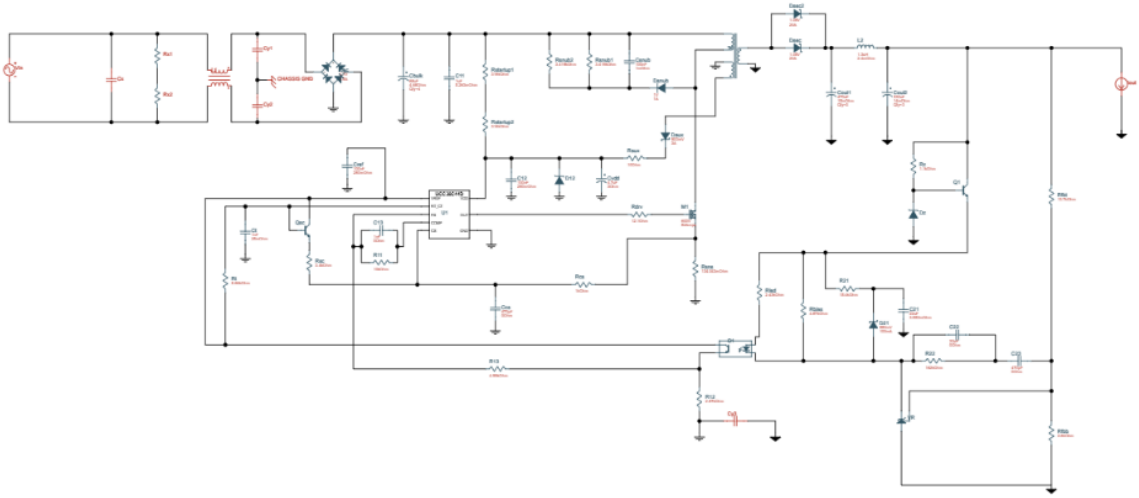
3.2 Power Supplies

For our project we wanted to imitate what a real system would look like in real world situations. Doing this allows us to bring up possible other ways to operate/power our system. Our choices lead us to these three options. Plug in power supply from wall that takes 120V/12V which will plug into access board PCB that then regulates voltage from 12V/5V/3.3V which then distribute to other peripherals, to build our own 120V/12V/5V/3.3V regulator PCB which will take in a plug from the wall into the board and then distribute power to our system, or lastly is use 9V batteries which will be on our access board and step up or down voltage for peripherals such as motors or cameras.

3.2.1 - Design of 120V/12V/5V/3.3V regulator

This design, like the rest of the regulators, will require us to total the current from each component to get the total current coming into our design needed. After adding all of these numbers up we ended up with around 7.5A needed for our system to operate. To account for spikes in current or anything that may cause issues, we decided to up the incoming current to 10A which also helps with the start of up motors or servos we have in our turret.

Figure 4. Power supply schematic for 120VAC/12VDC generated using Texas Instruments WEBENCH® Power Designer (accessed September 18, 2025).



The design has many faults that could occur, and complicated parts that would take multiple trial and error tests to make sure they work before plugging into 120VAC. This is only for taking 120VAC to 12VDC. We still need space to test the other 12V to 5V and 5V to 3.3V regulators. This goes against our requirements of trying to have a small and portable system. To allows us to focus on other specifics such as better coding and extra testing on easier designs with easier parts, we decided to not move forward with this design

3.2.1.2 - Build 12V/5V/3.3V regulators V1

Another option we are deciding on is the use of regulators that will step down the voltage incoming from our 120V/12V AC/DC wall plugin from 12V to 5V then to 3.3V. This would be accomplished by having two cascading regulators of 5V and 3.3V. We will have the output of the 5V design from WEBench feed into the input of our 3.3V which allows for easier design for the components that will need just 5V and the others that will just need 3.3V. With this design, a possible issue is the wires needed to come out of the access box to power the rest of the system. We would need a cord for the motors and servos which would be their own wires of at least two; we would need at least one 5V and 3.3V wire and the final one being a grounding wire on the turret PCB. Having so many wires could cause latency issues and open the design to issues if we connect wires incorrectly and do not check before applying power. We feel this is still better in the long run for possible frying components or shortages if everything were to be on one board. A lot of these designs can be found in common household tools and other electronic devices, so plenty of designs and schematics are available and easy to replicate. For this project we want to make sure we have plenty of time to test such designs and make sure they are cheap in case of any breaks or issues in initial designs; reordering would not cause us to go over budget.

3.2.1.3 - Build 12V/5V/3.3V regulators V2

Originally, the regulator mentioned in the previous paragraph would be made in cascade to help with managing current along with keeping the space between each regulator smaller. However, many problems come into play when we start to cascade regulators instead of running them in parallel. Running in cascade causes drops in voltage which at first doesn't sound like a problem. But when designing these regulators, we are putting an expected incoming voltage into the system. If we design the 12V/5V regulator and we have too much voltage drop due to the components that will need 12V, it could cause the incoming voltage to not be high enough to regulate. This could cause the output voltage to not be high enough either meaning that the components that need to be at 5V won't be able to operate and same for the 3.3V if the voltage is too low there as well. A fix is having the regulators separated from each other and running them in parallel with the incoming voltage. Splitting the voltage across multiple lines so the output can go to multiple devices would allow the components to take the current and voltage needed without the worry of dropping the voltage too low so that other regulators won't work. A drawback to this is that having more signal lines come out of the output of each regulator could allow more problems to happen such as messed up/not connected lines or wrong wire connections all together. Spacing could be expanded as well due to components needing to be spread out to help reduce noise between them as well, which goes against the requirements we have of keeping the access box and turret PCB small.

3.2.2 – Power Supply

The motors that power our turret can take up most of the current needed in this project. The current can be as high as 20A needed to start the motor and another 10A to keep one rotating at the speed we need to shoot. Meaning if we start the two motors at the same time, we need at least 40A coming into our system for the motors alone, not including

other components. A possible way to get around this is having a power supply be at the start of our power cord which will then feed into one of the two versions of our regulator designs for the access box. We would need this power supply to allow for the motors to run at start up and at idle time while also supplying the other components of the system such as the cameras and the MCU. We would have a power cord that goes from the wall into our external power supply. This power supply would either then plug into the turret PCB or plug into the access box. Plugging into the turret PCB could help with wiring issues. Since most of the 12V components will be over by the turret, it would be easier to design circuits and PCBs so that everything is wired over there. From the turret PCB, we would then trace an output wire of 12V into the access box which would then go through one of our regulators to supply power to the other components. This approach would have minimal wire output coming out of the turret but could pose the risk of damaging the turret side of our project. The goal is to only have the access box be the main point of contact, and everything else is connected. Connecting the turret PCB leads to extra faults externally, such as someone tripping over the wire. This could massively affect our system due to how integrated the turret is. If the cord goes into the access box and someone trips on the cord, the access box can be repaired easily by replacing the board if a component gets damaged. If the turret were to get hurt in some way, we lose functionality of the project which would be catastrophic since the project revolves around needing to shoot a target.

3.2.3 - Using Batteries

Using batteries for our power system is also another possibility. These are cheap and readily available and have many parts that we can use in PCB designs. This would allow easier movability for our system as well and not always require the system to be near a wall in order to operate We have plenty of choices such as standard lithium-ion batteries, lithium polymer battery, etc. Many electronic devices use such batteries however we cannot use typical alkaline batteries such as Duracell AA batteries. These will not have enough current rating or proper resistance to handle the current needed to power our motors. If we used batteries, they would have to be recharged and come in packs that allow at least 12V and plenty of amperage to run our motors. Some options we found can be found below.

3.2.3.1 - LG MJ1 18650

These lithium-ion batteries would be meant to be run with at least 4 batteries in series to create an output of at least 12V for our motors and servos. This would allow us to keep a low current while keeping the necessary voltage still to keep the motors on even during start up. Even with doing these batteries in series, the current needed from the motors is still too high and requires a current regulator to step up the current needed for our system. After looking at WEBENCH and needing custom parts to accommodate this, it does not seem feasible to use this type of design even if it would be easier to use compared to a wall plug. The difficulty to implement this design far outweighs the complications of needing to be near a wall outlet

3.2.3.2 - Zippy Compact 4S 5000mAh 25C LiPo Pack

This lithium polymer battery pack is much more within the scope of what we need for this project. It can run automatically at higher voltages, usually around 11V to above or at 12V. It also has a much higher current draw ranging from 10A up to 40A+, which is the range needed for this system due to the motors and servos. These types of battery packs are used for other RC applications such as drones, RC cars and boats, and many other electronic toys/vehicles. This would not require the use of regulators for current or voltage that goes into the turret and can be directly plugged into our motors and servos, which could also help with noise and latency in our system. The issue with these batteries is the stability at long run times and usage. We would have to factor in possible extra safety measures if we implemented this design but also need to adjust the budget due to the unknown issues that could arise from using such batteries in this system

Table 41: Power Technology Options

	Supply Voltage Range	Current capacity	Build difficulty	Repair difficulty	Cost
Wall plug in and 120V to 3.3V regulator	12V to 3.3V	As high as needed	Extremely difficult	Extremely difficult	>$100
Wall plug and build series 12V/5V and 5V/3.3V	12V to 3.3V	Limited to wall plug bought, usually around 20A-30A	Simple, designs come from WEBench	Simple, designs come from WEBench	>$60
Wall plug and parallel 12V/5V and 5V/3.3V	12V to 3.3V	Limited to wall plug bought, usually around 20A-30A	Simple, designs come from WEBench	Simple, designs come from WEBench	>$60
Use of Batteries	At 12V, use of regulators to step down	Rated off battery	Harder due to needing a battery management system	Extra systems needed to be kept an eye on	<$60
External Power supply with parallel regulators	12V use regulators to step down	Rated off Power Supply	Simple, plug from power supply into input of regulators	Higher than batteries and regulators due to higher current	>$100

Looking at this table, we decided to do go with an external power supply that will feed into parallel regulators to power our system. We need high current for our servos, which means we need freedom to choose a wall plugin with extra current. This can be done with an external power supply and then distributing power across the system will be the regulators in parallel. This helps with not having voltage drop between regulators and allows for easier design. The only issue with selecting all of this is the higher cost, but the difficulty is easier and so are repairs since things are more isolated.

3.2.4 - Power Architecture & Supply Selection

For the turret system we need a reliable power source that can handle the demands of spinning up two BLDC flywheel motors, moving servos, and running all the control electronics. The power supply is the only power source for the turret; no batteries involved. It has to maintain a stable 12V rail while two BLDC flywheel motors are ramping up, servos are moving around, and all the logic circuits are running. Wall power supplies aren't designed to absorb the kind of current surges you get when motors start, so we had to design the system to manage loads carefully.

The ESCs will have soft start enabled, there are also bulk capacitors placed at each ESC to buffer the inrush current and smooth out short load steps. Without these measures, the power supply voltage would dip and potentially cause resets or erratic behavior.

3.2.4.1 - Industrial Enclosed, UL Listed Supplies

These supplies have tight voltage regulation, documented overcurrent/ overvoltage/ overtemperature protections, finger safe covers, predictable thermal derating curves, and actual datasheets you can trust. The documentation tells you exactly what happens under fault conditions, which is critical when you're presenting.

These supplies also have proper EMI filtering built in, so they don't create noise on the AC line that could interfere with other lab equipment. The finger safe covers mean you can't accidentally touch live terminals even if someone opens the access panel during a demo, which is a requirement for public demonstrations on campus. With a soft start and the staggered flywheel ramp, we're expecting short peaks around 20 to 30A and steady state current somewhere around 4 to 8A.

3.2.4.2 - Repurposed Server PSUs

Server power supplies pack a lot of current into a small space, but they're loud, the fans run constantly, the connectors are nonstandard, and the documentation is inconsistent. Server PSUs are designed to run in a rack where noise doesn't matter and where they have forced airflow from the server's chassis.

In a standalone application, they're painfully loud, and the high-speed fans add vibration that could affect aiming stability. The connectors also require custom breakout boards or adapter cables, which adds cost and complexity. More importantly, most server PSUs don't have proper overload behavior documentation for standalone use.

3.2.4.3 - Low Cost LED Supply Bricks and Clones

These suppliers have attractive pricing, but their transient response is all over the place and the approval markings are questionable.

The voltage regulation on many of these units is adequate for LEDs, which don't care much about transients, but is inadequate for motor controllers that need stable voltage. Safety markings are also suspected of a lot of cheap supplies. You'll see CE or UL marks that may or may not be legitimate, and there's no reliable way to verify them without expensive testing.

Table 42: Power Supply Part Selection

Type	Typical 12 V Rating	Certifications	Terminals / Enclosure	EMI / Acoustic	Major Cons
Industrial Enclosed PSU	24–50 A (350–600 W class)	UL/CE/CB; documented protections	Covered screw terminals; closed chassis	Low EMI; some fan noise (UHP fanless)	Heavier/pricier than LED; higher-watt models larger
Low-Cost "LED" Supply	Labeled 30 A; practical 15–25 A	Often no UL/CE; sparse docs	Exposed terminals; thin case; open vents	Variable EMI; can buzz	Trips on motor surges; safety/doc concerns
Repurposed Server PSU	60–100 A (750–1200 W common)	UL/CE for server use	Edge connector; needs breakout & enclosure	Loud fans stock; strong internal EMI filtering	Needs enclosure & wiring work; noisy unless modded

3.2.4.4 - Decision on Power Supply Technology

We have decided to go with the Industrial Enclosed, UL Listed supply category. A 12V / 29A unit should cover the choreographed startup profile. If testing shows we want shorter stagger times or longer rapid-fire strings, we might bump up to a 12V / 50A unit with a remote sense to hold 12.0V right at the ESCs. This choice prioritizes safety approval, stable operation, and predictable behavior during demonstrations over cost savings.

Table 29: Power Supply Part Selection

Model	Output Rating	Notable Features	Why It Fits
Mean Well LRS-350-12	12V, 29A (348W)	Enclosed, protections	Selected baseline: covers our staged ramps with ESC bulk capacitors
Mean Well RSP-600-12	12V, 50A (600W)	Remote sense, PFC, remote on/off	Upgrade margin allows shorter staggers/longer strings
Mean Well UHP-350-12	12V, 29.2A (350W)	Slim, fanless	Quiet alternative; ensure airflow for full load

For each power supply we look at, something to consider is the thermal derating curve and protection behavior in the datasheet. These components need to be readily available and provide clear documentation on overcurrent and overvoltage shutdown behavior. If the supply doesn't have verified safety certifications or lacks proper documentation, we can't move forward with that part.

3.2.5 - Regulator Comparison

Below we will explore the different types of regulators we can use for our project. Most of our components can run at 3.3V or 5V with a few exceptions such as the brushless motors. We will need high efficiency and a cheap regulator when deciding which one to use since a lot of our costs are going into better accuracy such as the IMU. The other major cost will be PCB costs due to needing thicker traces and boards due to the high current of our motor. Let's see if we can find a cheap but reliable regulator

3.2.5.1 - TPS564242

This synchronous buck regulator can have an output current of up to 4A and has a voltage range of output between .6V and 7V. It's commonly used thanks to its compact design which is needed on space-constrained boards. This is not a major issue for us right now but later could be useful if we need to increase the copper on our boards. Having a smaller regulator and parts needed for it to operate might help with spacing of the board, bringing down the cost. It has an eco-mode and fast transient response however due to its compact size it does have thermal limits at high continuous current. Since these regulators won't be dealing with more than 2A on their own, we are well within the range of the max 4A. Its efficiency is around where our team needs about 90% to 95%.

3.2.5.2 - LM2679T-ADJ/NOFP

This synchronous buck regulator has a possible output voltage of 1.2V to 37V which has plenty range for what we need. It has a high current capacity, maxing out at 7A but has lower efficiency than the other regulators at around 75%. This regulator is not prone to

not working due to the board layout, which allows extra customization when it comes to building this later. We will need this if we try to enact a “plug and play” build which will be explained later when using a voltage divider.

3.2.5.3 - TPS563200

The main advantage of this type of regulator is the optimal minimized BOM it needs to function. Its small BOM allows it to have a smaller footprint, making it perfect for embedded systems and small systems. It has an output range of .76V to 7V and has a max current of 3A. This max current could be an issue if we want to add more peripherals but for now it's in the running mostly for its size and BOM. It's a six-pin configuration but still has less features compared to other parts on this list

3.2.5.4 - TPS563210A

This buck regulator runs with a max current of 2A or 3A with 8 pins. It has small common power rails that need careful consideration when it comes to set up. It has a good power balance allowing it to be easy to implement. Its small size helps as well to keep its efficiency at around 90%. It has similar thermal constraints but could be harder to maintain due to its lower current capacity

3.2.5.5 - TPS566238

This 6A synchronous buck option has a feature that differs from the rest; its continuous conduction option. This type of regulator is meant to handle multiple loads and high inrush current if it exists in the system. We will be paying for this upgrade due to its higher BOM count, higher price, and higher footprint to build. We wont need something on this size but it's good to consider in case issues come up down the line. Its efficiency range is still among higher ones being at 90% to 95%

3.2.5.6 - LMR60430

This regulator is an automotive-grade buck allowing a much wider range of output from 1V to 20V and has low electromagnetic interference (EMI) in case this comes up with servo or motor issues later. It does have a lower max current of around 3A but efficiency is better, being around 90% to 95%. This model also has better thermal handling, allowing us to run near max current output if needed, whereas many other models could start to deteriorate. The system is bigger than normal models and has a slightly higher cost as well. If EMI isn't critical and VIN isn't changing much, this part would be overkill for what we need to do.

3.2.5.7 - TPS62133

This high switching frequency model of a buck converter is small and compact. It has very high efficiency at lower loads, and the switching frequency enables tiny magnetics in the system if that is something we are looking for. It has a voltage output range of .9V to 6V and runs at a max current of 3A. Its efficiency is the best yet getting up to 100% but again this part seems to be above what our system needs. We need a simple input and one output regulator which this part can do but others can do it in a simpler way.

Table 30: Regulator Part Comparison table

Part Number	Output Voltage Range	Max Current(A)	Efficiency
TPS564242	.6V - 7V	4A	90% - 95%
LM2679T-ADJ	1.2V - 37V	6A	70% - 80%
TPS563200	.76V - 7V	3A	88% - 92%
TPS563210A	.76V - 7V	3A	88% - 92%
TPS566238	.6V - 7V	6A	90% - 98%
LMR60430	1V – 20V	3A	90% - 95%
TPS62133	.9V - 6V	3A	90% - 100%(mode dependent

3.2.5.8 - Decision on Regulators

We have decided to go with the **LM2679T-ADJ** due to its higher max current and less prone to issues with board layout. Its design is simple too for both the 5V and the 3.3V regulator schematics. Both can be small and compact, allowing easier board design and more room for other peripherals. We will keep in mind TPS566238 due to its max current being so high and the fact that it's made for more inrush current type of applications. We may need this down the line if the regulators get affected by the inrush current from our motors, but both systems will be isolated from each other so this should not be an issue.

3.3 - Computer Vision Software

For our project we need computer vision because we need a way to process and understand what is in the images that our stationary cameras output. Computer vision lets us take the raw pixel data and turn it into useful information that can signal our system to initiate. Our main goal with this is to use blob detection, which finds areas in the image that stand out from the background and groups them together. By doing this, our system can pick out objects of interest and treat them as targets for further analysis. This makes computer vision a key part of our project, since it bridges the gap between simply capturing an image and using it for decision making.

3.3.1 – Computer Vision Libraries

For our project, we have two main options when it comes to computer vision software. One option is to port over an existing library, which would allow us to use pre-built functions and tools that are already available. The other option is to develop our own custom library, where we would write the specific algorithms needed for our system from the ground up. Both approaches give us a way to handle image processing and blob detection on our embedded board, and the choice depends on how we want to balance development effort with system requirements.

3.3.1.1 – Open-Source Computer Vision Libraries

There are many libraries that offer blob detection algorithms for free online. These libraries come with built-in functions optimized for different tasks such as image filtering, edge detection, and blob detection. Using libraries will save a lot of development time and ensure that functionality is reliable and efficient. By using a library, more time and energy can be focused on applying the tools to our project, rather than spending time coding the low-level operations. This can help to expand our project in the future, since these libraries come with more than just the required functionality. Using a library will cut down on development time, ensure reliable and efficient algorithms, simplify overall system design, and provide a scalable architecture.

3.3.1.2 – Custom-Made Computer Vision Library

Building our own computer vision library from scratch will allow us to have more control over how the algorithms are written and optimized for our specific use case. A custom library will be designed to only include the necessary functions needed for blob detection, ensuring that memory space is utilized optimally. A custom library will also allow for modifications to the blob detection algorithm, which can be necessary if more vigorous constraints are placed onto the blob detection algorithm. A custom library with the specific hardware components in mind will allow for easy integration onto our MCU. Creating a custom library will cut down on wasted memory, while offering control over the functions that are supported, as well as how those functions operate.

Table 43 - Computer Vision Libraries

Option	Pros	Cons
Open-Source Computer Vision Libraries	Reliable and efficient code, less development time, documentation provided, simple to implement, scalable	Source files take up significant memory, may include unnecessary functions and code, hard to optimize for specific use cases and hardware, hard to manage dependency links
Custom-Made Computer Vision Library	Fully optimized to support specific hardware components, small and manageable memory footprint, can be tailored for speed or efficiency, depending on resource restrictions.	Requires significant development time, increased risk of bugs or errors, limited flexibility with new system requirements

3.3.1.3 - Decision of Computer Vision Libraries

Based on the table, we have decided to create our own custom-made computer vision library. The main driver of this choice was memory restrictions. Embedded boards come with limited memory, and many online libraries require multiple megabytes of flash memory to store the library alone. When a running process links those libraries to the

program code and combines that with the storage space needed for the heap, stack, and runtime data, the overall process size could increase beyond what the MCU can support. The second important factor is that we want to be able to optimize the blob detection algorithm to avoid detecting objects that are either too large or too small, which is a feature we may not be able to guarantee with online libraries. The third factor is that these algorithms are well documented online, giving us many different places to find code and support for building a complete library. The final factor is that it will be easiest to ensure that the input and output of the different functions of our library will work well with the MCU. This means that our input and output will be decided by the limitations of our MCU, and our output will be customized to best be passed down the software pipeline.

3.3.2 – Blob Detection Algorithms

There are two types of object detection algorithms, single-stage object detection and two-stage object detection. Single-stage algorithms detect objects with one pass through the image. This type of algorithm is most suitable for real-time or resource-consumed systems. The downside of this type of object detection is that it is generally less accurate. Two-stage object detection uses two passes of the image to make predictions. The first pass sets up possible predictions for where objects may be, and the second pass refines those predictions. This approach is much more accurate than single stage but ends up being much more computationally expensive.

There are two common metrics used to evaluate the precision of each model, Intersection of Union (IoU) and the Average Precision (AP) metrics. The IoU is calculated by taking the bounding box of the prediction and the bounding box of the ground truth and overlapping the two. Taking the area of the intersection between the two and dividing that the total area covered by the bounding boxes provides a numerical measure of how close the prediction is to the ground truth. The AP metric measures precision and recall. Precision measures the proportion of predicted boxes that correspond to the actual object. Recall measures the fraction of true objects that were correctly detected. High precision means there were few false alarms, and high recall means there were few object misses. We then use IoU and compare it to a threshold, usually 0.5, and if it is greater, then it is a true positive, and if it is less, then it is a false positive. Using the false positive and true positive numbers, we can graph the precision vs recall curve and take the integral of this curve to get our AP for each class, or specific object we are detecting. If there are multiple classes, then take the average of all APs to determine the mean AP (mAP).

3.3.2.1 – YOLO Algorithm

One algorithm is called YOLO, or you only look at it once. YOLO uses 24 convolutional layers followed by 2 fully linked layers within the convolutional neural network. The first 20 layers are pretrained on ImageNet for feature extraction. This model is then converted to perform detection by adding convolutional layers and connected layers. YOLO divides the image into an SxS grid, where if the center of an object falls into a cell, that cell is responsible for detecting the object. Each cell makes multiple predictions, with the one with the highest IoU. This bounding box is now to sole predictor for that object. Then,

non-maximum suppression is used to remove redundant boxes if needed. YOLO V7 can run on images with a max resolution of 608 by 608 pixels, meaning it can detect smaller objects.

3.3.2.2 – Faster R-CNN Algorithm

Another algorithm is called Faster R-CNN. Faster R-CNN is a two-stage object detection algorithm that begins with a backbone network to extract feature maps from the input image. After the feature map is generated, a Region Proposal Network slides over it to predict both the probability that a region contains an object and the coordinates of the proposed bounding boxes. Because the RPN and detection network are trained jointly, Faster R-CNN supports end-to-end training, which improves efficiency and accuracy. The next step is to pool each variable-sized Region of Interest into a fixed-size feature map. Finally, a detection network classifies each proposed region and refines the bounding box coordinates.

3.3.2.3 – Color-Based Blob Detection Algorithm

The final algorithm is a basic color-based blob detection method. This algorithm doesn't rely on any CNNs and just outputs an array of bounding boxes that are based off color thresholds. It works by taking the image as a 2D array of pixels and looping through each pixel, determining if that pixel is part of a blob or not. A second 2D array tracks visited pixels to avoid redundant processing. As the program iterates through the image, any unvisited pixels within the color threshold will trigger the flood fill function. The flood fill function uses depth first search to go through the neighboring pixels that haven't been visited and are within the color threshold, adding them to the current blob.

Table 31: Algorithm Comparison table

Feature	YOLOv5-N (r6.1)	Faster R-CNN	Color-Based Blob Detection
Algorithm Type	Single-Stage Detector	Two-Stage Detector	Classical
Detection Method	CNN	Region Proposal Network and CNN	Pixel color thresholding and flood fill
Accuracy (mAP)	28.0%	> 70%	N/A
FLOPs (Billion)	4.5	~8-9	~0.002
Maximum Resolution	640x640	No maximum	No maximum
Computational Cost	Medium (Runs on CPU and GPU)	High (Runs mostly on GPU)	Low (Runs solely on CPU)
Model Size	7.2 MB	12.6MB (MobileNet)	Tens of Kilobytes
Training Requirement	Requires labeled dataset and training	Requires labeled dataset and training	No training required

Effect from external lighting / background	Low	Low	High
Implementation Complexity	Moderate	High	Simple

3.3.2.4 - Decision of Blob Detection Algorithms

Based on the table, we decided to use a color-based blob detection algorithm that uses thresholding and flood-filling. The main reason for this choice is that the other two algorithms use a CNN, which will take up too much memory space. CNNs require storage for both the model parameters and the intermediate feature maps that are generated during the convolution steps. The running processes, the CNN model size, and the intermediate feature maps will require more resources than what the ESP32 can offer. The second reason that we chose to use the thresholding algorithm is that the stationary cameras act like the first pass in a two-stage detector. The purpose of the cameras is to find regions of interest for further investigation, rather than classifying the objects. The third reason for choosing the thresholding is that the algorithm is much easier to implement, requiring only a few small functions, rather than large CNN models. This will cut down on development time, as no network will need to be trained with labeled data sets, and there are plenty of online resources available to aid in the making of this function.

3.3.3 – Wired and Wireless Communication

Our project uses three cameras to detect and track objects that are nearby. Two of these cameras are responsible for detecting if a threat is nearby, while the third camera is mounted to a pan-tilt mechanism and is used for dealing with the threat. To effectively signal and position the pan-tilt mechanism, it is necessary to transmit position and status data between the cameras and their processing units. Therefore, choosing an appropriate communication method, either wired or wireless, is critical for the operation of our system. This decision will impact speed, reliability, and flexibility of the overall system.

3.3.3.1 – Wired Communication

Wired communication refers to the transmission of data over physical connections such as USB, Ethernet, or serial interfaces between two devices. They generally offer a reliable and stable method for transmitting data because the physical medium prevents interference, noise, and signal loss. Another advantage to wired connections is consistent and high bandwidth. They have their own private communication channels, which allow for maximum data transfer speed. For example, one protocol, USB 3.0, allows a transfer rate of 5 Gbit/s. They also have a smaller power footprint because they do not need to generate radio signals, maintain wireless links, or use amplifiers, antennas, or other RF components.

Wired connections do have limitations. One disadvantage is physical cables. The cables can restrict movement and placement of hardware. If the system is complex enough, cable management can also negatively affect development and debugging. Wired also requires physical ports to connect to. These physical ports limit the number of peripherals that can be connected to the processor.

3.3.3.2 – Wireless Communication

Wireless communication refers to the transmission of data between two devices via electromagnetic waves, eliminating the need for physical wires. This flexibility makes it easier to position hardware and allows for free movement, simplifying installation, and reducing the complexity of wire management. Without wires, reconfiguring and relocating hardware components don't require significant rewiring, saving time when setting up or making changes. Popular wireless technologies, such as Wi-Fi, Bluetooth, and ZigBee, provide communication over a reasonable range depending on the environment, making them necessary for systems where wired connections are impractical. Wireless connections also support multiple devices on the same network, creating scalable setups that can easily add new sensors and peripherals to the network in the future.

There are some major limitations to wireless communication. It is prone to interference, as shared frequency channels and environmental factors reduce bandwidth and introduce errors. Signal quality can also be degraded by the mediums, such as water and air, that the electromagnetic waves travel through. Additionally, these networks are inherently less secure because the signals are broadcast to the world. Wireless systems also use a lot more power to broadcast radio signals and maintain network connections.

Table 32: Communication Technology Comparison

Feature	Wired Communication	Wireless Communication
Reliability and Stability	Highly reliable with minimal interference	Prone to interference, disconnection, and signal degradation.
Bandwidth and Speed	High, consistent speed with maximum bandwidth CPU clock allows	ZigBee: 250 Kbit/s Bluetooth: 50 Mbit/s Wi-Fi: > 100 Mbit/s
Power Consumption	No extra power needed	Requires power to generate signals and maintain connections
Installation and Flexibility	Complex installation with restricted movement and constrained placement	Easy installation with free movement and flexible placement
Range	Limited by cable length	ZigBee: 33 feet Bluetooth: 33 feet

		Wi-Fi: 30 - 50 feet

3.3.3.3 - Decision on Communication Interface

Based on the table, we decided to use wired communication because the problems a wireless connection solves are not present in this project. Although the overall system is made of different subsections, the total area that will be covered is small enough that physical wires won't restrict movement or placement. There will be no need to use a long-range communication protocol if all devices are within a couple of inches of each other. The reduced wire complexity is also not needed because there are only a few peripherals that will be connected. While the added flexibility for movement and placement would be nice, the bandwidth and speed that wired connections offer satisfy the system's needs to be fast and accurate. Power is another metric that cannot be ignored on a small-scale embedded project. Wired connections use less power, which is important when resources are constrained. Although support for wireless connections will be simple because the ESP32 comes with built-in wireless capabilities, it will not be needed for this project, and wired connections will suffice.

3.3.4 – AI Vision Cameras

For this project, an AI vision camera is required to control the pan-tilt mechanism to automatically locate and track objects in real time. An AI vision camera is needed over a traditional camera because traditional cameras output raw image data, but our project requires this data be processed using object detection and recognition. An AI vision camera will help us avoid performing expensive computations by using its built-in processing to identify objects and output bounding boxes directly. These bounding boxes contain data on the objects' position relative to the camera, which can then be used to control the pan-tilt mechanism to orient itself towards the object. This approach allows for reliable and efficient autonomous tracking without needing to implement external computers, making the system design more compact, cheap, and suitable for embedded applications.

3.3.4.1 – OAK-D Series

One popular camera is Luxonis' OAK-D series. This camera is advertised for real-time tasks such as object detection, tracking, and depth estimation. Each camera integrates an Intel Movidius Myriad X VPU which handles the onboard neural networks without relying on an external processor. The OAK-D cameras provide color and depth data simultaneously, allowing for more precise object location predictions and distance measurement. Pretrained deep learning models like YOLO and neural network architectures like MobileNet come installed and can transfer bounding box data and labels directly via a USB 3.0 connection.

However, the OAK-D series comes with a few drawbacks for embedded applications. The modules consume more power compared to other smaller AI vision cameras, and

their higher bandwidth interfaces may require a more powerful host processor if full data streams are needed. The cost of OAK-D devices is also significantly higher than simpler AI cameras. The functions that this camera provides line up with its cost, but it may be too much for this project.

3.3.4.2 - Arduino Nicla Vision

The Arduino Nicla Vision, another AI vision camera. It is designed for low power machine learning and computer vision tasks. It uses a 2-megapixel color camera with an STM32H747 dual-core processor to perform real-time object detection and classification directly on the device. An integrated 6 axis IMU provides accelerometer and gyroscope data that can be used to compensate for tilting and improve stability during object tracking. The camera supports lightweight neural networks that output bounding box coordinates and object labels through UART, SPI, and USB interfaces. This allows for easy integration with the ESP32, where the bounding box data will be converted into signals that control the pan-tilt mechanism.

While the Arduino Nicla Vision offers onboard AI processing in a compact piece of hardware, it does lack the capabilities to perform complex neural networks. There is very little onboard RAM and no built-in accelerator, meaning this camera is only suitable for lightweight models and simple detection tasks, rather than deep network models such as YOLO or SSD. The inference speed is also slow compared to other cameras, at around 10 to 20 frames per second. The camera resolution is also 2 megapixels, which may be restricted for some use cases. Additionally, the integrated 6-axis IMU does not contain a magnetometer, so its orientation estimates will drift over time. Overall, this model is useful for low-power and moderately demanding embedded applications.

3.3.4.3 – HuskyLens AI Machine Vision Sensor

The HuskyLens 2 AI Machine Vision Sensor is an upgraded embedded vision system for more advanced CV (computer vision) applications. Like the HuskyLens 1, it supports built-in algorithms like color recognition and object classification but is more flexible by allowing users to implement their own lightweight deep learning models (e.g., YOLO), making it more suitable for our application that requires higher detection accuracy and target specificity.

The HuskyLens 2 still has a compact design with an onboard display for real-time visualization (similar to HuskyLens 1), which is useful for debugging. Since it also connects via UART or I2C, we can integrate easily into our system with the ESP32-S3. Also, the HuskyLens 2 is not limited to its built-in algorithms, allowing for more customizable vision tasks.

The original HuskyLens 1 was initially considered for this project, due to its simplicity and built-in algorithms. However, after testing it, the algorithms were not sufficient to achieve high detection accuracy. Since we cannot fine tune or retrain the models for our

specific target (red balloon), it limits the effectiveness of the HuskyLens 1 (especially in different lighting conditions).

While the HuskyLens 2 addresses the shortcomings of the HuskyLens 1, it still has some constraints such as limited processing power compared to full-scale platforms. However, its ability to run optimized custom YOLO models significantly improves performance within those constraints.

Table 33: AI Vision Part Comparison Table

Feature	OAK-D	Arduino Nicla Vision	HuskyLens 2 AI Machine Vision Sensor
Processor	Intel Movidius Myriad X VPU	STM32H747AII6 Dual-core Arm Cortex- M7/M4	Kendryte K230
AI Image Resolution	300x300, 416x416, 640x640	96x96, 128x128, 224x224	640x480
Depth Sensing	Yes	Yes	No
DFOV	81°	80°	70°
FPS	~20-30	~5-15	~30
AI Algorithms	Object detection, Facial recognition, Semantic segmentation, Pose estimation, Depth-based 3D localization, Gesture & hand tracking	Image classification, Simple object detection, Color or motion detection, Gesture recognition	Facial recognition, Object tracking, Color recognition, Line tracking, Object classification, supports implementation of your own YOLO model
Onboard Memory	8GB flash, 1GB RAM	2MB flash, 1MB RAM	8MB flash, 6MB RAM
Communication Interfaces	USB 3.0	Wi-Fi, Bluetooth, I2C, UART, SPI, USC 3.0	UART/I2C, SPI
Power Supply	5V	3.5V – 5.5V	3.3V – 5.0V
Current Consumption	1 A	105 mA	230 mA – 320 mA
Additional Peripherals	9-Axis IMU, Stereo depth perception,	6-Axis IMU, Time of Flight sensor, Microphone	2’ IPS display, pre-programmed modes
Output	Image frames, depth maps, bounding boxes, labels, confidence scores,	Image frames, IMU data, bounding boxes, labels, confidence scores	Bounding boxes, labels, confidence scores

	3D coordinates, IMU data		
Price	$299	$80	$85

3.3.4.4 - Decision on AI Vision Camera

Based on the evaluation, the team selected the HuskyLens 2 AI Machine Vision Sensor as the primary vision system. Although the HuskyLens 1 was lower in cost, it ultimately could not meet the required performance threshold due to its reliance on built-in algorithms. It specifically struggled to achieve a higher detection accuracy on our desired target.

The HuskyLens 2 overcomes the limitation of the version 1 model by allowing the implementation of a custom YOLO model tailored to our specific target object (red balloon). This allows the system to focus on relevant features while filtering out noise and irrelevant detections, resulting in improved accuracy and robustness. As a result, the camera can provide more reliable bounding box coordinates, which is required for the tracking system.

The HuskyLens 2 maintains a sufficient frame rate for real-time tracking, ensuring continuous updates of target position for smooth pan-tilt control. Combined with the ability to implement our own custom YOLO model, this made it more suitable for our project than the original HuskyLens. Overall, the upgrade allows the system to meet its performance goals while maintaining a compact design and relatively low cost comparted to other AI vision sensors.

3.3.5 - Stationary Camera Choices

The stationary cameras will be used for target detection and coordinate retrieval. They will provide visual input to detect and classify objects approaching the camera's field of view. Due to the cameras being immobile and stationed on the base of the mount, we must consider factors such as interface compatibility, resolution, overhead processing, and power consumption over things like portability.

3.3.5.1 - Wired Camera Modules

The wired camera modules will connect directly to the microcontroller through the hardware interfaces such as DVP, SPI, I2C, or UART that we implement onto the custom PCB. These modules are typically lower power, allow for direct control, and can provide raw or compressed images that we will use for processing to identify objects for our target and send the coordinates to the main controller using lightweight image detection or TinyML. Some examples of these cameras include the OV2640/5640 DVP modules or Arducam Mega SPI series.

3.3.5.2 - Wireless/Network Cameras

These camera modules connect via Wi-Fi, such as the Wyze or ESP32-CAM modules that internally process video streaming and compression. They send data to the master controller over Wi-Fi using UDP or TCP. These cameras typically offer higher

resolution, but it requires more power and significant amounts of bandwidth and is not MCU-friendly. Thus, it is better suited for tasks like surveillance and not low-latency onboard detection. Since the MCU that we are using only has around 32KB of memory, it becomes difficult to store and process high resolution images, making it not ideal for us to use these types of cameras.

3.3.5.3 - AI-Integrated Cameras

These types of modules include the HuskyLens, which embed AI models directly for tasks such as object detection, line tracking, or face recognition. These modules offload the image processing from the MCU and output simplified data (i.e. bounding boxes or object IDs). Although these modules can greatly reduce the processing load and reaction time, the units are very expensive and provide less access to raw image data.

3.3.5.4 - USB Cameras

USB cameras such as the Wyze Cam are able to stream in high resolution and frame rates. The cost is not too bad at about $15-35, and it is easy to integrate as it is usually plug-and-play. However, our MCU the ESP32-S3 does not support UVC (USB Video Class), so it would require extra hardware to integrate it into our project.

Table 34: Stationary Camera Options Table

Option	Pros	Cons	Features	Cost	TinyML Suitability
Wired Camera Modules	Low power consumption, MCU control	MCU resource heavy, interface limits	Direct MCU interface, raw/compressed image output	$5-30	Strong (scalable resolution, lightweight processing)
Wireless /Network Cameras	Easy integration, high resolution	High bandwidth, high power consumption, not MCU-friendly	Onboard compression and streaming	$20-60	Poor (overhead is too high for the MCU)
AI-Integrated Cameras	Offloads tasks from MCU, easy to use	Expensive, limited ML options to built-in models, no raw data access	Built-in AI models, sends results only	$50-150	Neutral (Does not need or run tinyML)
USB Cameras	Cheap, high frame rates	Our MCU lacks UVC support, requires extra hardware	High-resolution, plug-and-play	$15-35	Poor (not directly usable by MCU)

3.3.5.5 - Decision of Stationary Camera Technology

From the table above, we weighed the pros and cons of each option and decided that it would be best to use a wired camera module that connects via SPI, UART, I2C, or DVP interfaces. Thus, giving us the following options to choose from:|

3.3.6 - Turret Camera

3.3.6.1 - Arducam Mega 3MP SPI Camera

The Arducam Mega 3MP connects via SPI, allowing multiple modules to operate on a single ESP32-S3 MCU by using separate chip-select lines. It provides a 3MP resolution (2048x1536) that can be scaled down and has built-in JPEG compression, which eases strain on the MCU's memory. It features autofocus and IR cut filter capabilities that help ensure accurate detection across various distances and lighting conditions. The scalable resolution allows for tuning speed or detail depending on the application. Even though SPI is slower than DVP connections, making high frame-rate image capturing limited, the trade-off allows for dual-camera operation on one MCU. The JPEG compression and scalable resolution make it ideal for tinyML applications by feeding reduced image sizes into a custom/pre-made model for blob or object detection. The cost is around $35, making it mid-range compared to the other options.

3.3.6.2 - Arducam Mega 5MP SPI Camera

This is the higher resolution version of the 3MP SPI camera mentioned above, as it offers 5MP (2592x1944) resolution. The Arducam Mega 5MP provides greater detail but has slower performance at full resolution. Other than the high resolution, the Arducam Mega 5MP shares the same SPI advantages, flexibility, and compression support as the 3MP version. This camera is moderately suitable for tinyML as the high resolution is overkill for our detection tasks and will need to be scaled down anyways, although it will offer better raw image quality for classification. The cost is around the same as the 3MP version, with it being around $35-45.

3.3.6.3 - OV2640 Camera Module

The OV2640 is a 2MP (1600x1200) camera that connects via a DVP parallel interface. The OV2640 supports multiple image formats including YUV, RGB, and compressed. The resolution can also be scaled down significantly (i.e. 40x30), making it efficient for lightweight blob detection. Using UXGA can achieve 15 fps and 30 fps at SVGA. The problem lies with the ESP32-S3 as it only has one DVP interface, requiring us to use one OV2640 per MCU without additional hardware. The Ov2640 is strongly suitable for TinyML blob detection or simple classification with lower resolution. The cost of one unit is around $5, making it a very low-cost option.

3.3.6.4 - OV5640 Camera Module

The OV5640 is a 5MP (2592x1944) sensor that connects via the DVP interface. It supports a wide range of frame rates (i.e. 30 fps at 1080p, 60 fps at 720p, 120 fps at QVGA) and includes autofocus and adjustable image quality settings. The OV5640 poses the same problem as the OV2640 module with the ESP32-S3 only having a single DVP bus, making multi-camera setups difficult. The OV5640 also has a rolling-shutter which

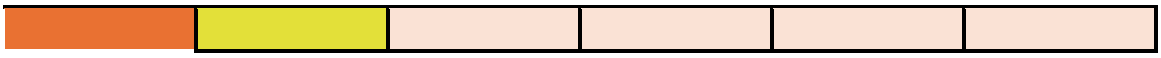
can cause distortion when tracking fast-moving targets. With its higher resolution, it is not as suitable for TinyML as the OV2640, as the resolution would have to be scaled down significantly before processing causing some overhead. The cost is around $15-20, making it more expensive than the OV2640, but cheaper than the Arducam modules.

3.3.6.5 - HuskyLens AI Vision Sensor

The HuskyLens is an AI-enabled vision module that outputs recognition results directly over I2C or UART. It has built-in models for object tracking, color detection, face recognition, and others allowing for minimal programming effort. This module eliminates the need for image processing on the MCU, but because the raw image data is not accessible, the flexibility is restricted. Since the HuskyLens has built-in AI models and version 2 of it allows for deployment of our own YOLO model, meaning that we are able to offload heavy tasks like object recognition from the MCU. The HuskyLens is significantly more expensive than the other options, being around $50-60 and v2 being around $85.

Table 35 - Detailed Camera Options Table

Features	Arducam Mega 3MP	Arducam Mega 5MP	OV2640	OV5640	HuskyLens
Interface	SPI	SPI	DVP	DVP	UART/I2C
Max Resolution	2048x1536	2592x1944	1600x1200	2592x1944	1600x1200
Frame Rate	15 fps at full resolution, scalable	Slower at full resolution, scalable	15 fps at UXGA, 30 fps at SVGA	30 fps at 1080p, 60 fps at 720p, 120 fps at QVGA	Not raw video
Power	185mW-585mW	182mW-650mW	125mW-140mW	18mW-294mW	1.06W-1.15W
Multi-Camera Support on ESP32-S3	Yes	Yes	No	No	Yes
Features	Autofocus, IR cut filter, JPEG compression	Same as Arducam Mega 3MP with greater image detail	Blob detection friendly	Autofocus, image controls, image compression	None
Cost	~$35	~$35-40	~$5	~$15-20	~$50-60



3.3.6.6 - Decision of Camera Technology

After reviewing the options listed above in depth, we have decided to go with the Arducam Mega 3MP SPI for stationary cameras. Even though the OV2640 and OV5640 are lower cost and support higher frame rates, the ESP32-S3 can only support one DVP module per MCU, making a dual-camera setup impractical without increasing the hardware demands. The Arducam Mega 3MP on the other hand uses the SPI bus which the ESP32-S3 can share across multiple devices with separate chip-select lines, allowing us to use two cameras simultaneously on one MCU, helping to reduce complexity, power consumption, and cost.

The 3MP resolution is good enough for detecting approaching targets. It also has built-in JPEG compression and scalable resolution which help save space in the ESP32-S3's limited RAM of 32MB. Even though the SPI connection is slower than DVP, it does not particularly affect our application as we are focusing on detection rather than high-speed video.

The other options, such as USB cameras, were excluded due to compatibility issues with the ESP32-S3. The HuskyLens AI Vision Sensor would ease the workload of the MCU as it has built-in AI models and onboard detection/recognition; it was not chosen due to its cost and the inability to access the raw image data. The Arducam Mega 3MP was chosen over the 5MP version since the resolution would have to be scaled down significantly (lower than 3MP) to process and run lightweight detection on the images. Thus, the Arducam Mega 3MP camera gives the best balance between multi-camera support, flexibility, ease of integration, and resolution for our project's requirements.

3.3.7 - MCU Choices

The Microcontroller Unit (MCU) is responsible for controlling all of the turret functions including the pan-tilt mount servo motors, stationary cameras, main camera, and peripherals (i.e. lights and alarms). An MCU that is capable of handling real-time image detection, wireless communication, and has enough storage to store low resolution images for lightweight detection is required for this project. It should also have sufficient GPIO support in order to control multiple peripherals while drawing less power. Due to the use of multiple cameras and sensors, processing capability and memory capacity were key factors when selecting an MCU.

When choosing the MCU, we needed flexibility to support multiple communication protocols such as UART, I2C, SPI, and more so that we had the ability to connect multiple components to one MCU and allow for a range of options when considering components. The ease of implementation with other components and cost were additional considerations.

3.3.7.1 - MCU vs SBC

When choosing a controller for this project, a Microcontroller Unit (MCU) and a Single-Board Computer (SBC) were considered. The factors that were taken into consideration

included cost, processing requirements in terms of this project, and power constraints. MCUs are more suited for real-time control with low latency, and SBCs are more suitable for applications that require heavier computations like large language models. However, since the project requires performing simple image detection using TinyML, controlling servo motors for a pan-tilt mount, and other real time tasks, then an MCU is a suitable choice for this project.

Table 36: MCU vs SBC

Options	Pros	Cons	Features
MCU	Real-time control, draws low power, low cost (~$5-20)	Limited RAM and CPU power to run high resolution image processing	Supports RTOS, compact, able to run TinyML, supports multiple interfaces
SBC	Powerful CPU, may include GPU, supports heavier computations (i.e. advanced AI)	Requires more power, high cost	Able to run heavy tasks like automation or complex vision tasks, runs full Operating System (i.e., Linux)

From this comparison, an MCU was selected for the turret system. It meets the performance requirements for real-time tasks and is able to process lightweight detection models. The MCU allows for efficient power use and low costs considering budget constraints. Even though an SBC is more powerful than an MCU and would allow us to run advanced image processing with AI (even on live video streams), it is not necessary to implement it in our project. We also have a constraint that we can only use the chip and implement it onto a custom PCB that will include all the connectors, and the controls for the tasks will be flashed into memory.

3.3.7.2 - MCU Options

After determining that an MCU is the best choice for this project, multiple MCU chips were evaluated to determine the most suitable one. Several factors were considered such as memory capacity, support for GPIO, support for communication between devices and peripherals, processor performance, and power consumption. Only the MCU chips are going to be implemented in this project, since we are custom designing the PCBs. This allows us flexibility over the layout and interface design. The MCUs that we evaluated are the ESP32-S3R8V, Arduino Uno R3 (ATMega328p), Arduino Mega 2560 (ATMega2560), and Raspberry Pi Pico 2 (RP2350).

3.3.7.3 - ESP32-S3-WROOM-N32R16V

The ESP32-S3-WROOM-2-N32R16V features a dual-core 240 MHz processor, giving it the ability to handle vision tasks to an extent, control multiple peripherals, and

communication between devices simultaneously. Having a dual core means that we can run tasks in parallel using each core such as having one core manage motor controls and sensors, while the other core can handle lightweight image detection using TinyML. The ESP32-S3 includes 33 GPIO pins that allow interfacing with several devices including servo motors, sensors, and cameras through UART, SPI, DVP, and other interfacing options. Its support for Wi-Fi and Bluetooth 5.0 provides us with a flexible way to communicate and connect without additional hardware. It is a cost-effective option with it being around $5-15. The area where the ESP32-S3-WROOM-2 falls short is memory size (32MB of flash RAM and 16MB of PSRAM), which limits our ability to run heavy machine learning models on high resolution images. However, that can be overlooked with the fact that it can still process low resolution images for simple blob detection or image recognition using TinyML.

3.3.7.4 - Arduino Uno R3 (ATMega328p)

The Arduino Uno R3's ATMega328p chip is an 8-bit microcontroller that runs at 16 MHz with around 32KB of flash RAM and 2KB of SRAM. The memory capacity alone makes this MCU unsuitable for this project as it would not be able to perform real-time image processing and lightweight detection. They can't even hold images of the lowest resolution. The speed of the SPI interface is also too slow with it being about half of the CPU clock speed (~8 MHz), making the data transfer rate low affecting our reaction times and may not meet the project's target goals.

3.3.7.5 - Arduino Mega 2560 (ATMega2560)

The ATMega2560 offers more SRAM than the ATMega328p (~8KB) and more GPIO support, allowing for connections with more devices and peripherals. The 8-bit architecture and 16 MHz processing speed it possesses is not enough for this project's image processing application. The ATMega2560 also lacks support for Wi-Fi and Bluetooth, reducing the options for connectivity. While this MCU is acceptable for applications requiring more I/O tasks, it poses the same problems as the ATMega328p and is not suitable for this project.

3.3.7.6 - Raspberry Pi Pico 2 (RP2350)

The Raspberry Pi Pico 2 features the RP2350 microcontroller that possesses a dual-core ARM Cortex-M33 processor that runs at 150 MHz. The RP2350 also includes an SRAM with the capacity of up to 520KB and 16MB of flash/PSRAM, which is less than the ESP32-S3-WROOM-2, but more than the Arduino microcontrollers. It still provides adequate memory for image processing at low resolution and real-time control. It features a multitude of interfacing options including UART, SPI, I2C, etc. The lack of support for Wi-Fi and Bluetooth limits connectivity and communication between devices (i.e. sending and receiving image data through UDP/TCP), which also requires us to add additional hardware like an external module if we need that ability. Although the RP2350 is a strong option for this project, it still falls short of the features and capabilities that the ESP32-S3-WROOM-2-N32R16V offers.

Table 37: MCU Choices

Features	ESP32-S3-WROOM-2-N32R16V	Arduino Uno R3 (ATMega328p)	Arduino Mega 2560 (ATMega2560)	Raspberry Pi Pico 2 (RP2350)
CPU/Lock Speed	Dual-core Xtensa LX7 at 240MHz	Single core 8-bit AVR RISC-based at 16 MHz	Single core 8-bit AVR at 16 MHz	Dual-core ARM Cortex-M33 at 150 MHz
RAM/Flash Memory	512 KB SRAM and 16 MB of flash memory	2 KB SRAM and 32 KB of flash memory	8 KB SRAM and supports 64,128, or 256 KB of flash memory	520 KB SRAM and up to 16 MB of flash memory
Pros	Dual-core CPU for parallel tasks, able to run TinyML, able to support multiple devices, supports Wi-Fi/Bluetooth	Low power, ease of integration, simple	Same as the ATMega328p, allows for more I/O tasks with high GPIO count	Provides more RAM for processing low resolution images, dual core allows for parallel tasks, and flexible options for GPIO
Cons	Limited RAM for high resolution ML, draws more power	Not enough RAM for any image tasks, slow clock speed	Too slow for real-time image processing, not enough memory storage	Does not support Wi-Fi or Bluetooth, slow USB transfer speeds since it uses USB 1.1
Interface Options	33 GPIO pins, DVP, SPI, UART, I2C	23 GPIO pins, supports UART, SPI, I2C	54 GPIO pins, can support UART, SPI, I2C	30-48 GPIO pins (depending on model), can support UART, I2C,

				SPI, or USB 1.1
Voltage	3.0-3.6V	2.7-5.5V	1.8-5.5V	1.8-5V
Current	Ranges from ~50 to 100mA	~10-20mA	~20m	~15-100mA
Cost	$5 - $15	~$2-5	~$1-5	~$1-10

3.3.7.7 - MCU Decision

After evaluating the MCU options above, the ESP32-S3-WROOM-2 was chosen as the MCU for this project. Given the data above, it was deduced that ESP32 provides the best balance between connectivity, power efficiency, and performance. The dual-core architecture it possesses allows for multiple tasks to be performed simultaneously such as having one core manage the controls for the motors and peripherals, while the other can run lightweight ML models on real-time low-resolution images. Although the ESP32-S3-WROOM-2 has enough RAM to run TinyML applications (i.e. blob detection and tracking), it is limited when trying to perform heavier computations with complex machine learning algorithms, which is what is needed for this project. The RP2350 was a strong contender for the MCU choice, however it falls short of the ESP32 with its lack of wireless communication and slower clock speeds. The ATMega328p and ATMega2560 are seen to be unsuitable for this project as their limited memory and low processing power makes it difficult to handle real-time image data even at the lowest resolution. Their high GPIO pin count, however, makes them ideal for peripheral only controls for smaller systems. Thus, it has been decided that the ESP32-S3-WROOM-2-N32R16V is the best option for this project's requirements.

4.0 Standards and Design Constraints

4.1 IPC-2221/IPC-2222: Generic Standard on Printed Board Design

4.1.1 Standard Overview

IPC-2221 [A23][138] covers the generic standards need to produce printed circuit boards, and IPC-2222 extends that by discussing certain design requirements such as spacing of width tracing and ways to get around EMF. Many standards and requirements listed in this standard are needed for us due to the design of our Access box PCB but also our turret PCB as well. We need these so our boards can perform as expected during the demonstration but to also keep us and people around us safe during testing and when power is applied to our system. These standards also help to ensure the manufacturability of our board.

4.1.2 Conductor Spacing and Width Requirements

When designing our boards, something we need to keep in mind is the conductor spacing and width requirements between traces and pads. The reason we need to look out for this during design is possible shortage of trace, possible arc current, and impedance mismatch. Shortage of trace is especially hard when it comes to the physical implementation of the MCU we have chosen. When designing the traces going out of this MCU, we need to keep in mind which GPIO pins are being used because if we put too many pins too close to each other, this could cause the trace to short. Effects of this could be as simple as peripherals not responding if it's a signal-to-signal trace. For power to ground traces, these need extra care when being made. Issues with this trace short could cause the regulators to not turn on, causing them to heat and break down. Traces could burn as well along the board causing signal to not go through. To mitigate these possible issues, we will make sure each trace in the 5V and 3.3V sections have at least 7 mil spacing between each tracing, allowing for fixes of traces if needed as well. The minimum assigned by IPC-2221 is 5.1 mil between traces, but we will add an overhead to be safe. In the 12V and 7V rails we will use a space of at least 20 mils between traces. This is to ensure the higher current going through does not jump even under extreme conditions that we could experience.

4.1.3 Trace Width for Current Capacity

Not only must we worry about the length between traces, we need to worry about the width of traces themselves, especially with our motors needing an inrush current of 20A. IPC-2221 provides formulas and other materials to help determine what current would be needed and the amount of copper that is needed as well so that the board can perform as expected. What we will need to find the proper trace is the required current; temperature rise above ambient, copper weight, and whether the trace is internal or external. This can be represented by this formula below

$W= (I \wedge (0.744))/(k * \Delta T \wedge 0.44)$

I = current, k = .024 for external layers, .048 for internal layers and ΔT is change in temperature above ambient. Our motors will be running at 20A start up and a continuous 10A. We will need a trace of at least 5mm or 200mil to run the 10A safely with a 1 oz copper pour and a 10°C rise in temperature. For the 20A region, it is recommended that we have a 2 oz copper thickness to help with stability and proper conduction. We will need a trace of at least 150mils for 1 oz copper and around 75 to 80 mils for 2 oz copper. For the rest of our board, such as MCU GPIO, LEDs, and Cameras, we can use a standard of 10 mils which is generally industry standard. The standard and formula say we only need a few mils(1 to 3 mil) but we will run well above to allow for a more robust system but still give us plenty of space to move components around as needed.

4.1.4 Via Design and Thermal Management

Vias are plated through holes connecting different layers of the board. This is important to consider for both signal routing and power distribution between layers of our boards. The via has a current capacity based off the barrel plating thickness, the diameter of the via, and the thickness of the board. This is important, especially in our project due to the servos and motors. Our other peripherals of LEDS, Cameras, and buzzer can all work

with a standard via hole in Fusion360 but won't work with the motors and servos due to the higher current they will draw. Because of this, we will need to focus on routing for this section first and worry about the peripherals later in design. Avoiding vias in the turret section will be the easiest approach. Thermal vias need to be considered as well due to the high heat dissipation some components will output such as the voltage regulators or the motor drivers. We are required to have thermal vias directly under the thermal pad of surface mount components. The via diameter must be .3 to .5mm, and arrays of vias must be utilized when needed. We can also connect to internal copper planes to help with heat spreaders. For our design, when needed, we will create vias connected to the ground plane to help with the heat sink.

4.2 IEC 62368-1: Audio and Video, Information and Communication Technology Equipment

4.2.1 - Standard Overview

This standard (IEC 62368-1 [A24][138]) covers many safety requirements needed when conducting tests in an electrical environment. It breaks up the safety requirements into two parts; the primary circuits, which is the circuit connected directly to the AC source, and the secondary circuits anything powered by the DC outputs of the AC source. Since we will be using a power bank, its connection will be from the wall of 120VAC and the output will be 12VDC, so both requirements must be taken into consideration when designing schematics, PCBs, and wiring of all electronics. It is recommended as well in automated systems that there is emergency stop procedures in place. Our system has automated detection but is not automated firing. To be safe, we will have easily accessible switches to all the power connections we have. Each motor will have its own fuse and switch, and the access box will contain a switch for each voltage rail to help with the insulation of any issues in the system but to also help with stopping anything if an emergency were to occur.

4.2.2 - Primary Circuit Safety requirements

We will not be designing the primary circuit aka the power bank but how to power, proper isolation, and proper handling of this power bank will be needed especially due to the high input and output that we will be handling in this situation. The use of fuses, proper grounding, and physical barriers will be needed to properly protect our input from any issues. Fuses can help detect when too much power is coming into the system. The fuse will trip when too much power comes in, causing the system to shut down a stop operating. The AC wall cord we choose needs to have a proper grounding pin to help with safety of shocks that could occur when plugging and unplugging this device. The physical barrier we are enacting for this power bank is to make it isolated from our system. We will physically have it at least 2 feet away from any active electrical elements to help with EMF and any other transfers that could affect our system. This is in compliance with IEC 62369-1 as it suggests having the AC source and the user interface section not be in close contact with each other.

4.2.3 Secondary Circuit Requirements

The Secondary circuits will be anything connected to the output of the AC source. This includes our motors, ESC, Huskey lens, laser diode, and our access box. Fuses are

recommended to help with the mitigation of high temperature rise in our circuits which could cause harm to the device and user. It also suggests that making bigger traces and paths allows for extra temperate rise without harm to the system. We plan on connecting a fuse to the output of the ESC due to the motors being the most dangerous part of our system. This fuse will serve as a warning that something has gone wrong with the motors, and we must turn off our system and access the issues.

4.3 - ANSI/NFPA 79: Electrical Standard for Industrial Machinery

4.3.1 - Standard Overview

This standard created by the National Fire Protection agency addresses the general industrial electrical safety for industrial machinery. Although this is only a senior design project and we won't be working with industrial machinery, there are still standards we should abide by and look out for during the creation of this project. Specifically we should pay attention to safety regarding the sizing of wires to use going out of our access box, power system protection, and also protection of our system since it has the use of motors and servos.

4.3.2 Wire Sizing and Current capacity

ANSI/NFPA 79 [A25][138] has tables that specify ratings for different sizing of wires and the current they can carry with 60°C or 75°C insulation. Due to heat loss from the current flowing through the wire, proper precautions need to be taken to avoid potential fire starters in the circuit or on the wire itself. Below is a table for common gauge wires with the current carrying capacity.

Table 38: Wire comparison

AWG	Size Cross-Section(mm^2)	Current Carrying capacity
22	.324	3A
20	.519	7A
18	.823	10A
16	1.31	15A
14	2.08	20A
12	3.31	30A

This applies to our approach to design since we need multiple wires coming out of our access box so that we can power our system. 22-guage or 20-gauge wire would suffice for most of our system, but the servo and motors are the biggest issue. These require high amperage that is not recommended to run on PCBs, but it is possible. An alternative we need to be aware of is directly wiring the power bank to our ECS which power the motors. We would need to choose an appropriate wire to do so, and these standards can help decide which wire can sustain 10A for a long period of time and which can handle a quick inrush of current if needed.

Another factor to look at is the derating factor of bundled wires. The more wires that are bundled together, the less current should be carried through these wires. The standard by

NFPA lists different bundles of conducts should only use a certain percentage of the total current carrying capacity. Four to six conductor bundles should have 80% of its base current capacity; seven to nine conductor bundles should have 70% of its base current capacity, and ten to twenty conductor bundles should have 50% of its base current capacity. This is important since bundling wires would help with the presentation and neatness of our project. However, by doing this, the 3.3V and 5V power components would now need a higher current capacity rating of wire.

4.4 IEC 61010-1: Safety Requirements for Electrical Equipment for Measurement, Control, and Laboratory Use

4.4.1 Standard Overview

IEC 61010-1 [A22][137] establishes safety principles for laboratory and control equipment operating from mains or DC power. It focuses on preventing electric shock, overheating, and mechanical injury. Since our turret operates indoors using a 120V AC to 12V DC power supply and includes powered electronics, this standard is directly relevant. The main objective is to ensure user protection through proper insulation, enclosures, and current limiting components.

The standard covers several key areas that apply to our project. First, it defines requirements for basic and supplementary insulation between different voltage levels to prevent users from contacting hazardous voltages. Second, it establishes temperature limits for accessible surfaces and internal components to prevent burns and fire hazards. Third, it addresses mechanical hazards like moving parts, sharp edges, and pinch points that could cause injury during normal operation.

For educational and laboratory equipment like our turret, IEC 61010-1 is particularly important since the users may not have detailed electrical knowledge. The standard helps ensure that even if someone makes a mistake during operation or tries to access internal components, the risk of serious injury remains low. It also provides guidelines for proper labeling, documentation, and emergency shutdown mechanisms.

4.4.2 Application to the Turret

Our access box and power electronics are enclosed in a PETG housing that isolates all live terminals. The flywheel and servo assemblies are mechanically shielded to prevent hand contact with moving parts. All accessible circuits operate below 60V DC, satisfying Safety Extra Low Voltage (SELV) requirements. Power lines include fuses rated below the conductor's thermal limit to prevent overheating or fire during a fault condition.

The enclosure design ensures that users cannot accidentally contact any voltage higher than 12V without using tools to open the access panel. Even with the panel open, the layout positions high current terminals away from the opening, and all connections use insulated terminals or connectors. The 12V distribution uses properly rated wire with appropriate insulation thickness, and all splices are made with insulated crimp connectors rather than exposed solder joints.

Temperature management follows IEC 61010-1 guidelines for accessible surface temperatures. Components that generate significant heat (ESCs, voltage regulators,

motors) are positioned with adequate spacing for natural convection cooling. The ESCs are mounted to metal standoffs that act as heatsinks, and the enclosure design includes ventilation slots positioned to allow airflow without exposing users to moving parts or live circuits.

The mechanical shielding around the flywheels completely encloses the spinning components with only the dart feed opening exposed. This opening is sized to accept foam darts but is too small for fingers to reach the rotating wheels. The servo mechanisms are similarly enclosed, with only the output shafts exposed, and those shafts have smooth surfaces without pinch points or sharp edges.

4.5 IEC 60204-1: Safety of Machinery – Electrical Equipment of Machines

4.5.1 Standard Overview

IEC 60204-1 defines requirements for safe wiring, circuit protection, and emergency stop systems in electrically controlled machines. Though our system is small, the automated motion of servos and flywheels qualifies it as machinery in principle. The standard addresses how to properly implement control circuits, emergency stops, and protective measures for machines with electrical actuation.

Key aspects of IEC 60204-1 include proper circuit separation between control logic and power circuits, requirements for emergency stop functionality, and guidelines for wiring practices that prevent failures. The standard also covers proper labeling of all electrical components and connections, color coding of wires, and strain relief for cables that experience movement.

For our turret project, even though it's a demonstration system rather than industrial machinery, following IEC 60204-1 principles ensures the system behaves predictably and can be safely shut down in an emergency. The standard emphasis on proper wiring practices also helps prevent intermittent failures that could cause unpredictable behavior during demonstrations.

4.5.2 Application to the Turret

The project adheres to IEC 60204-1 by isolating logic and power circuits, labeling all connectors, and providing current protection on each voltage rail. The turret includes a single emergency stop button that immediately cuts power to the ESCs and servos while leaving the low voltage logic powered for diagnostics. All wiring is color coded, strain relieved, and routed away from mechanical motion to prevent abrasion and accidental shorts during operation.

The E-Stop implementation follows the standard requirements for Category 0 emergency stops. When pressed, the E-Stop button breaks power to all motion related circuits (flywheel ESCs, servo regulators) through a mechanical contact that cannot be overridden by software. This ensures the turret stops moving even if the microcontroller has crashed or is running faulty code. The E-Stop button is a latching type that requires deliberate reset, preventing accidental restart after an emergency shutdown.

Circuit separation keeps the 12V power distribution physically separated from the 5V and 3.3V logic circuits on the PCB. Power and signal grounds are connected at a single star point to prevent ground loops while maintaining proper circuit isolation. High current paths use dedicated copper pours or heavy gauge wire, while signal traces are routed away from noisy power switching circuits to minimize electromagnetic interference.

Wire color coding follows standard conventions: red for positive power, black for ground, and various colors for signals with documentation mapping each color to its function. All connectors are labeled on both the board and the cable with their function and voltage level. This makes troubleshooting easier and reduces the risk of connecting power to the wrong circuits during maintenance or modifications.

Strain relief is implemented at all cable entry points using grommets, cable ties, or 3D printed cable guides. Cables that move with the turret (servo power, camera connections) use flexible silicone wire and are routed through cable carriers that limit bending radius and prevent the cables from snagging on moving parts. The routing keeps all cables away from the flywheel area and positions them so they can't be pinched by the rotating turret base.

4.6 IEC 60825-1: Safety of Laser Products

4.6.1 Standard Overview

IEC 60825-1 [A21][137] classifies and regulates all visible light lasers based on optical power, wavelength, and safety precautions. It specifies labeling, emission limits, and user protection requirements for Class 1 through Class 4 laser devices. The classification system helps users understand the potential hazards and what safety measures are required for each class.

Class 1 lasers are completely safe under all conditions of normal use. Class 2 lasers (like ours) emit visible light where eye protection is normally afforded by aversion responses including the blink reflex. Class 3R lasers are low risk, but direct viewing should be avoided. Class 3B and Class 4 lasers are hazardous and require extensive safety controls.

The standard also defines requirements for laser product labeling, including warning labels, aperture labels, and explanatory labels. It specifies the format and content of these labels to ensure consistency across all laser products. For Class 2 lasers, the standard requires specific warning text and the laser radiation symbol but does not require protective housings or interlocks that higher classes need.

4.6.2 Application to the Laser Subsystem

Our turret uses a Class 2 green laser module (Quarton VLM-520-52 LPT, 520nm, 1mW) for aim indication. This class is considered eye safe for brief exposure due to the natural blink reflex. To comply with IEC 60825-1, the module will carry proper labeling, and its mounting includes a shroud that limits beam spread and prevents the laser from crossing the audience area during rotation. The laser is powered from the regulated 5V rail and can be electronically disabled by the microcontroller during idle states.

The laser labeling includes a warning label stating "CAUTION: CLASS 2 LASER PRODUCT" with the standard laser warning symbol. The label also includes the

wavelength (520nm) and maximum output power (1mW). An aperture label near the laser opening warns "LASER APERTURE" to prevent users from looking directly into the opening. These labels are permanent and positioned where they remain visible during normal operation.

The mechanical shroud serves multiple safety functions. First, it restricts the laser beam to a forward-facing cone, preventing the beam from sweeping sideways across observers as the turret pans. The shroud extends approximately 20mm in front of the laser aperture and has an opening angle of about 30 degrees, wide enough for the aiming task but narrow enough to prevent wide angle exposure. Second, the shroud acts as a physical barrier that makes it difficult to position your eye directly in front of the laser aperture.

Software control provides additional safety. The laser only activates when the system is in active tracking mode and has a valid target lock. During idle states, system startup, or fault conditions, the laser remains off. The microcontroller can disable the laser instantly by pulling the enable pin low, which happens automatically if the E-Stop is pressed or if the system detects an error condition. This ensures the laser doesn't remain on if the turret malfunctions or loses control.

The mounting position places the laser near the center of the turret launcher assembly, aimed parallel to the dart trajectory. This positioning ensures the laser dot indicates where the dart will go, but it also means the laser is never pointed upward at steep angles where it could more easily catch someone's eye. The maximum elevation angle is mechanically limited to prevent the turret from pointing the laser at faces of standing observers.

4.7 - Communication Protocol Standards

4.7.1 - Standard Overview

The communication systems implemented in this project are required to transmit data between the two MCUs (master and secondary), main camera (HuskyLens), servo motors, stationary cameras, sensors, and other peripherals. Due to the reliance on various low-level serial communication interfaces (i.e. SPI, UART, etc.) and optional wireless links over Wi-Fi or Bluetooth, using established electrical and communication standards allow for a more reliable and stable operation of the turret system.

The turret system primarily uses SPI, UART, and I^2C serial communication protocols, as each performs specific actions based on distance, data rate, required position, and protocol support (how many connections are available on the MCU). There is Wi-Fi and Bluetooth compatibility available on the MCU that can be used for communication between the two MCUs (i.e. send coordinate data to master MCU for fine tracking), however for a more reliable and low-latency option we will connect the MCUs via a wired connection. These component-level serial communication standards are established by several organizations/companies including Motorola, EIA/TIA, and NXP that define safe operating voltage levels, timing of data transmission, and bus configurations in embedded systems.

Standards applied to this project design:

EIA/TIA-232 [A26][138] and EIA/TIA-485 [A26][138]:

This standard defines the characteristics for using UART, which is an asynchronous serial communication.

Motorola SPI Standard:

This standard describes a synchronous serial communication protocol (SPI) with high-speed data transfer between master and slave devices that is suitable for peripherals like cameras in embedded systems.

NXP I^2C Standard (UM10204 [A27][138]):

This standard defines the two-wire serial communication protocol (I^2C) that is primarily used for sensors and peripherals. Following the above standards will allow for reduced noise disturbance, prevent signal contention, and maintain integrity on custom designed PCB traces. With the combination of standards, proper line termination, isolation, and other factors, we can ensure that the system has robust, stable, and reliable communication between components.

4.7.2 - Wired Communication Protocol and Implementation

4.7.2.1 - I^2C Communication

The Inter-Integrated Circuit (I^2C)communication protocol allows for low-speed sensor data transmission, primarily from the IMU readings. I^2C provides a two-wire interface Serial Clock Line (SCL) and Serial Data Line (SDA) and is compatible with multiple devices interfacing using a shared bus. Pull up resistors (between I^2C) are connected on both lines to prevent signal interference between devices. The cable length can be adjusted to fit the proportions of our system and be minimized to ensure clock stability by reducing the line capacitance to around 400 pF and below, which is recommended according to the standard. With its ability to support multiple devices, there are bound to be collisions when transmitting data, so to prevent that the protocol implements a mechanism that monitors the bus for any incoming traffic and if a transmission is detected it will withdraw. This protocol was chosen for its ease of use and efficiency, making it ideal for short distance communication.

4.7.2.2 - SPI Communication

The Serial Peripheral Interface (SPI) communication protocol allows for high-speed data transfer between stationary cameras and MCU to perform tasks like capturing real-time image data. For multiple device use, each slave device will connect to a dedicated chip-select (CS) line to prevent data collision and will have a pull-up resistor attached to prevent unwanted activation when MCU is waking up. It operates using a 4-wire interface that is for the CS, MOSI (master out slave in), MISO (master in slave out), and SCLK (serial clock) lines which are routed directly on the PCB to help maintain clock integrity when operating at higher frequencies. The SPI bus will operate up to 8 MHz (maximum clock speed of Arducam) with 3.3V operating voltage, balancing data throughput and signal integrity. When switching between cameras, it is recommended to use decoupling capacitors near the power source for noise filtering. To prevent noise

disturbance/interference caused by high-current motor lines, proper grounding and resistance will be integrated into our custom PCB. Given SPI's ability for high-speed data transmission, support of half and full duplex data exchange between a master and multiple slave devices, and since the master controls the data flows which means that it does not require separate mechanisms to handle it which makes SPI protocol ideal for handling devices like the stationary cameras.

4.7.2.3 - UART Communication

UART will be used for serial communication between the master MCU and the HuskyLens AI vision sensor. This connection allows for streaming data continuously at varying baud rates using independent TX and RX lines. The standard recommends including series resistors near MCU pins to reduce EMI/RFI. Its ease of use, reliable point to point communication, debugging capabilities, and ability for asynchronous operation (helps timing among multiple devices) are the reasons UART communication was chosen.

4.7.3 - Safety Measures

Mechanisms are deployed in this project to detect, correct, or recover certain errors that can occur when communicating between multiple devices simultaneously. The SPI and UART data packets should include some kind of CRC or checksum validation in case any bit-level errors occur due to either noise or data corruption and ensure the integrity of the data during transmission. The MCU has a watchdog timer that will act as a fail-safe mechanism if the system becomes unresponsive. For cases like bus hang when using I^2C where the device on the bus holds one of the lines (SDA or SCL) in a low state, which prevents other devices from communicating on the bus, a toggle on the clock line can be placed to reset the communication without having to do a full reboot, saving a lot of time. Defaults could also be implemented in case of communication loss with critical peripherals where the system can enter a default state to prevent damage or uncontrolled processes.

4.8 ISO/IEC/IEEE 12207:2017 Systems and software engineering — Software life cycle processes

4.8.1 Standard Overview

This standard establishes a comprehensive framework for defining, managing, and implementing software throughout the entire project life cycle. It outlines the processes required for obtaining third party software, developing software, system maintenance, and retirement of the system. There is a strong emphasis on quality assurance, configuration management, and verification of functionality. This standard provides a common structure for organizations and project-level software processes, ensuring that there is consistency and traceability between all stages of the life cycle, from initial requirements to deployment. By following this standard, our team can be certain that the development of our project will be fluid, adhere to industry's best practices, and the final output will meet the defined requirements for the system. This standard is essential for

projects that demand reliability, maintainability, and formal documentation of software engineering practices.

4.8.2 Primary Life Cycle Processes

This standard organizes development into a clearly defined life cycle process: acquisition, development, operation, and maintenance. Acquisition and development help with deciding what the requirements are, overall design of the system, as well as creating the actual code, then testing the code, and then integrating the code. This is also when it is decided what third-party software will be used. The acquisition and development stage ensure that the software meets system requirements and performs in a safe and expected manner. The second stage, operation, involves monitoring the software as it is deployed in its intended environment. This stage ensures that it is reliable and user-friendly. The last stage is maintenance. This stage deals with bug fixes and requirements changes.

Having this architecture for our project will help to ensure that each software change is documented and easily reversible if a critical error is produced. By following this life cycle of development, it helps us to focus our time effectively, such as only writing software once it is decided exactly what the software needs to do.

4.8.3 Supporting Processes

Another process defined by this standard is the tools, methods, and assurance that are activated that make the life cycle effective and reliable. This includes documentation and configuration management, verification and validation, quality assurance, and problem resolution with risk management. Documentation and configuration management provide clear and verbose documents explaining what changes were made and by who. This helps track versions and software updates. Verification and validation check if the software matches the determined design and specifications, such as its speed and accuracy, as well as if it does what it is supposed to do. Quality assurance ensures that the system meets established quality standards, such as safety measurements. Lastly, problem resolution with risk management helps identify, track, and resolve bugs while weighing the amount of risk that each bug produces. This helps determine how likely a bug is to happen and if it is worth fixing the bug or not.

This supporting process will help to iron out how to go about each stage of the development life cycle. Having clear configuration management will ensure that each change is accounted for, as well as the supporting documentation to explain why it was needed. Performing tests that confirm that we have a quality product that meets the defined requirements will help reduce future errors and offer predictable performance across all stages of the life cycle. The standard also explains how to format code so that it is structured, readable, and maintainable, which is important as all developers will be working together to edit files and peer review each other’s changes before committing a change.

4.9 ISO/IEC 14882:2024 Programming languages — C++

4.9.1 Standard Overview

The standard discusses the specifications and requirements for the C++ programming language, and ensures consistency, reliability, and portability over all software and hardware platforms. In this standard, a formal definition of C++ language syntax, standard libraries, and core features such as object-oriented programming and memory management are established. There are distinct differences laid out between the core language and its associated data types, operators, and control flow and those of the standard library, which help to provide reusable components for IO, concurrency, and common functions. Because our system involves low-level hardware interaction with high performance image processing, it is important that we adhere to these standards to ensure that our C++ code behaves in a predictable manner and is compatible among different compilers for embedded environments. This standard will also ensure safe programming practices by using type safety, defined behaviors, and error handling when handling live data from external devices. Implementing this standard will help us to create reliable software that is easily maintained and compliant with the official C++ language.

4.10 ISO 10218-1 Robotics — Safety requirements

4.10.1 Standard Overview

This standard covers the safety requirements for industrial robots, focusing on the design and construction of the robotic system to protect operators and nearby personnel. It defines limits for robot movement, force, and speed, as well as adding protective measures such as emergency stops and guards. For our system, this standard will be relevant because of the pan-tilt mechanism and attached turret that involve automated motion. This motion can become a risk if not properly controlled. By following ISO 10218-1, we can implement safeguards to make sure that all moving parts stop on command if there was an error or emergency, and the system operates within safe mechanical limits. These safeguards include switches that can isolate power to the motors and design motion control software to limit the maximum speed and power. Adhering to this standard will help keep the operators, system, and surrounding environment safe during testing and normal operation.

4.10.2 Safe Motion and Operator Safety

For our project, ISO 10218-1 will guide the design and operation of the pan-tilt mechanism and turret by enforcing strict motion limits with emergency stop mechanisms and clear outlines of where the robot can and cannot reach. When programming the motors and actuators, we will ensure that the movement stays within the defined speed and accepted force limits. There will also be accessible switches that can immediately cut power to the system in case of an emergency. This standard will force us to use software to create safety zones, which means our algorithms need to be intelligent enough to distinguish humans from potential threats. This will avoid any accidental harm to the surrounding people. By following this standard, we will create a motion control system that is effective at detecting only the necessary objects, as well as creating a safe environment for operators and nearby personnel.

4.11 ISO/IEC 61966-2-1 sRGB Color Space

4.11.1 Standard Overview

This standard defines sRGB, which is a dedicated color space for red, green, and blue images that ensure consistent color representation among cameras, displays, and other devices. ISO/IEC 61699-2-1 specifies the chromaticity of the primary colors, the white point, and the gamma curve used to map linear RGB values to the non-linear format typically used in digital imaging. For our project, this is important because the stationary cameras output RGB frames, and we need the color data to be interpreted consistently when performing image processing and object detection. Adhering to the sRGB standard ensures that the numeric RGB values correspond accurately to perceived colors, which is crucial when bounding boxes are generated based on detected objects or when additional visual indicators like LEDs are used to represent distances or states. By following this standard, we can reduce errors caused by inconsistent color interpretation between cameras and processing modules.

4.11.2 RGB Data Representation and Processing

When handling RGB images from the cameras, a key consideration is how pixel values are represented and processed. According to the sRGB standard, each pixel is composed of three channels, each with values typically ranging from 0 to 255 for 8-bit images. The standard defines a gamma-corrected transfer function, which ensures that the perceived brightness of colors is consistent and proportional to the numeric values. For our image processing pipeline, this means that the software can reliably convert the raw RGB frames into the formats needed for object detection algorithms, such as normalizing pixel values or converting them to alternative color spaces if required. Following sRGB also helps maintain accuracy when overlaying bounding boxes or sending visual cues to external devices, such as LED indicators or display outputs, ensuring that the processed information accurately reflects the camera data.

4.12 - Time Constraints

4.12.1 - Overview

This project has two timed constraints, with one of them being the development process that includes the research, design, testing, and implementation of the turret system over the course of two semesters. The other time constraint is the real-time system that requires the turret system to respond to real-time events within a certain time frame. To successfully build this turret system, a schedule should be followed closely and be able to accommodate any issues that may arise.

4.12.2 - Development Time Constraints

This project is taking place over two consecutive 16-week semesters. This should give us adequate time to plan and develop our system with some headroom for any problems that may occur. The first semester will focus on research, design, and prototyping, while the second semester focuses on implementing, testing, and finalizing the turret system. Proper scheduling of our time is key to completing a functional turret system by having time constraints that will help with prioritizing tasks and deciding on designs.

The turret system consists of multiple components, including two microcontrollers, two to three PCBs that we are custom designing, three cameras, two servo motors for a pan-tilt mount, and several other peripherals making it a complex system as we need proper coordination between software and hardware. Given the complexity, problems with designing the PCB schematics and implementing the firmware are bound to happen which will lead to delays in completing this project. Therefore, it is important that we communicate with each other and set deadlines to ensure that no bottlenecks occur, causing delays later in the project.

A 32-week limit will restrict our ability to optimize the system. Ideally, we would have more time for testing, tuning, and optimization, however we need to focus more on achieving a functional system that meets the requirements we set in the proposal. By the end of the first semester, a working prototype should have been developed. By the end of the second semester, the turret system should be stable and demonstrable even if certain features must be removed or postponed due to the time constraints.

The use of custom PCBs can significantly impact the time we need to complete this project. The time it takes to draw up schematics, ordering, and fabricating the PCB introduces a challenge when scheduling as each part can take several weeks. That does not include the time it would take in cases of design errors, which could set back progress as it would require redesigning and ordering a new PCB. Therefore, it is important that the schematic is reviewed thoroughly and prototyping is completed early, so we can mitigate any risks.

Other factors that need to be considered include part shortages, shipping delays, or lab availability (limited space). The ways to counter some of these problems are to source components early on (during the first semester) and to have backup plans in case a component cannot be sourced. This is discussed with the group on a regular basis to identify the problems that could potentially cause the schedule to be thrown off course. Unexpected problems such as bugs in the software, mismatches of communication protocols, or failure of components are issues that must be addressed as quickly as possible. When integrating the subsystems, we must make sure that each part is operating correctly and cohesively, as this poses a great risk for time delays. This is combatted through carefully planning and testing each subsystem before integrating into the final turret system, avoiding the need to rework the whole system.

The effect of all these time constraints dictates that a disciplined approach to time management is key. To stay on track, the team will create a schedule, divide labor, and spend a great deal of time researching early in the process. Maintaining communication throughout both semesters, dynamically planning, and continually testing the system, we will be able to successfully complete the project within the timeframe while developing a robust system.

4.12.3 - Real-Time Systems Constraints

In addition to the time constraints related to development, there are also real-time system time constraints that must be considered to ensure the system is responsive and operates smoothly. The feedback and response of the system is important for safety, reliability, and performance. This includes visual and auditory feedback from LEDs and buzzers to

activate within 500ms of an event trigger (i.e. target detection). The synchronization of alert mechanisms with real-time events prevents any delays that may occur.

The stationary cameras and vision sensor module also require timing constraints when performing their roles. To detect targets and respond effectively, the turret system has to process the images received from the stationary cameras, send the data to the main controller, and react accordingly within 2 seconds of detecting the target movement. The response time takes the image processing using a light machine learning model (e.g. blob detection or color recognition), data transmission to servo motors on pan-tilt mount, and peripherals. A delay in this process could cause inaccuracy when tracking and detecting.

To address these requirements, task prioritization, communication between components, and rate of transferring data within the system (between MCUs and peripherals) must be taken into account when developing the software. To achieve these goals, the ESP32-S3 MCU was chosen for its dual-core capability allowing for task separation, such as controlling the servo motors, operating the stationary cameras, processing image data, tracking, and LED signaling using FreeRTOS (Real-Time Operating System) allowing for task prioritization of time-sensitive tasks from less time sensitive ones. Thus, we can minimize the number of interruptions or bottlenecks in data transfer across communication protocols (i.e. SPI or UART) while maintaining a deterministic performance. The real-time system time constraints significantly affect decisions regarding the design involving optimization, communication protocols, and MCU selection to ensure the system is able to operate consistently and reliably under several conditions.

4.13 Economic Constraints

This project also faces a financial constraint which involves how the overall budget is managed and ensures that all components and materials required for the turret system can be obtained without going over the agreed-upon budget. As the four of us are all undergraduate students, and this project is not sponsored, the cost of all materials comes out of our own pockets, therefore we have limited our budget to about $1000 USD.

The economic constraint will significantly influence the development cycle as well as the overall specifications of our system, as every decision must be weighed by performance and by cost, which in turn could lower the overall quality of the product to fit within the budget. Hardware will need to be chosen based on affordability, rather than pure performance, and software will need to be optimized and written for systems that offer fewer computing resources. Limited funding may restrict the use of high-end components and sensors, which could have been used to increase accuracy, speed, and durability. This constraint will require us to mark careful tradeoffs between quality and affordability. Additionally, it may also limit the number of prototypes we can build and test, which will slow down our ability to detect and fix early design flaws.

Resource allocation will need to be carefully managed, ensuring that the essential elements such as microcontrollers, motors, and cameras are prioritized and given the most flexibility when it comes to their price. By prioritizing these elements, we can spend more money getting the core functionality of the system working, while potentially losing

some of the less important functions. Additionally, the team may need to look at repurposing existing parts and technologies that would be free of charge and integrating those parts where needed. By using technologies that the team already possesses, such as old motors from past projects, we can reduce our overheads, but with the added cost of implementing technology that may not have been manufactured to integrate with a project like ours.

One way this constraint has affected our team is when making the decision of what to use for our color-based blob detection algorithms. The most efficient algorithms rely on deep neural networks, with millions of parameters and billions of floating-point operations. The amount of memory to hold the network, as well as the processing power to compute those floating-point operations would have required an integrated accelerator such as a GPU or TPU, as well as hundreds of megabytes, potentially even gigabytes of flash storage and RAM. The added performance and quality that a deep neural network provides for image recognition and object detection would development time and drastically increase accuracy and speed, but each of those added parts can cost anywhere from 20 – 30 dollars for additional RAM, or hundreds of dollars for a mediocre GPU. Because of the added costs of these parts, it was decided to go with a simpler color-based blob detection algorithm that doesn't require a deep neural network, sacrificing quality for a cheaper price.

4.14 Design Constraints

When designing our system, we are constrained by the internal mechanisms and performance of the electronics we choose. One such limitation is for the Arducam 3MP camera. This camera can capture high quality images, but this will result in large data files that exceed the data handling capabilities of the ESP32 MCU. The camera's SPI interface has a limited data transfer speed due to its internal clock speed of 8 MHz. This can create a bottleneck during image transmission, which will drastically slow down our system. For the maximum resolution of 3MP and a maximum transfer rate of 1MB/s, an image would take roughly 9 seconds to transfer from the Arducam to the ESP32. This limitation affects how quickly the ESP32 can receive and process new images, potentially causing unwanted latency between image capture and system response. This limitation causes us to decrease the image quality from the Arducam, resulting in less data per image, potentially decreasing accuracy, and worse overall performance.

The ESP32-S3 microcontroller also presents another hardware constraint. Due to its restricted computational power and available memory, processing large images for object detection can quickly use up all 512 KB of onboard RAM. This may cause the device to struggle with performing real-time object detection while also controlling the rest of the system. This limitation could result in dropped frames, slower decision time, or delays in system responses. To help mediate this constraint, an ESP32 board with a dedicated "external" RAM module was chosen. This module includes 16MBs of PSRAM, connected to the main RAM through an SPI interface, running on a 120MHz clock. This allows for 15MB/s of data transfer within main RAM and the PSRAM. All operations are performed on main RAM, therefore we are limited by how fast the PSRAM and RAM

can communicate, however 15MB/s should be more than enough to perform the necessary calculations.

Chapter 5: Application of ChatGPT with Similar Platforms

Since the release of ChatGPT in 2022, it has evolved into a useful tool using ChatGPT 5.0 that uses advanced reasoning and logic to solve problems and answer prompts. Since that time, there has been a spark of evolution in the AI field and multiple large language models (LLMs) have been created and released by Google, DeepSeek, Claude, etc. With all these LLMs available, it should simplify the amount of research needed as they search multiple sources and sites to formulate a logical and reasonable solution/answer. This has certainly been our experience when using multiple LLMs. However, each LLM has a limit to the amount of information they can output accurately. In my experience, ChatGPT has difficulty handling high level math involving circuits and such, but it excels at basic research by providing potential outcomes, several options, and tradeoffs without requiring too much detail in the prompt. The other LLMs such as Google AI were also able to provide sufficient information for a given prompt, however some required more elaboration such as my goal and what my expected outcome would be while some were able to deduce what I wanted and gave me more than what was asked by providing examples and possibilities for several applications. In most cases, the LLMs were able to provide sources for their information, but some struggled as they did not have access due to copyright issues, making an LLM like Google AI ideal for research since it has more access to a wide range of sources. The use cases ranged from member to member as can be seen in the case studies below, but in general each member used multiple large language models to perform different tasks for consistency and for accuracy we double checked the information and did follow up research.

5.1 - Case Study 1

Please refer to Appendix E, [1a], [1b] and [1c] for the Case Study 1 given prompt and outcomes from ChatGPT and Google AI.

ChatGPT [1b] [140] gave a brief but detailed list of recommendations, along with reasons for the recommendations and a list of potential architectures that could be considered when designing this project. The amount of detail is appreciated as it allows us to see the pros and cons of each MCU option, making it easier to choose one. As I did give the large language models some project specific items such as being able to use an AI vision sensor, it was general enough to provide me with various options that will excel in multiple areas of this project. Even though the options given by the ChatGPT large language model are detailed enough to get a rough overview of the MCUs, they do not provide datasheet information which would be an in-depth look at each option. As I did not specify an architecture for the system, the large language model gave a few recommendations as to how the system can be designed. This allowed for more insight into what MCU to choose, depending on factors like ease of use, cost, control, etc. The information that ChatGPT did provide for each MCU seems to be factual and sufficient to meet industry standards. It provided an answer with a detailed face-level overview of the

options that required extra research, which matched the specificity of the prompt provided.

When Google AI [1c] [143] was given the same prompt, it provided an answer similar to ChatGPT's large language model. The AI model included a few options ranging from SoCs (System-on-Chip) to SBCs (Single-board computer) with pros and cons, as well as architecture options similar to ChatGPT. The information that Google AI provides is not as detailed as it does not provide details about each options feature (i.e. processor cores, types of memory, ability to run machine learning models, etc.) that ChatGPT offers despite being given the exact same prompts and its access to a larger amount of sources. Although both options were given the same prompts, each one gave similar information on the types of MCU to choose for our application, however, the ChatGPT large language model is preferred over Google AI since it offered more detailed information and insight into the path we could take to build this system depending on what we want the MCU to prioritize.

5.2 - Case Study 2

Please refer to Appendix E, [2a], [2b] and [2c] for the Case Study 1 given prompt and outcomes from ChatGPT and Claude AI.

ChatGPT [2b] [145] and Claude AI [2c] [148] were both given the same prompt above, and the results between the large language models are vastly different. ChatGPT gave a much more detailed list of recommended cameras, types of cameras, specific modules, and tradeoffs. The Claude AI large language model gave a very direct answer to the prompt. It gave a brief list of camera modules with communication protocols.

ChatGPT did a great job of elaborating on the prompt it was given by thinking of several possible paths that the user could take and providing relevant sources for the user to do additional research on the topic. Given the broad scope of the prompt, ChatGPT was able to build on it and provide lots of useful information that would typically be found after researching deeper on the topic. It gave examples of camera modules using different communication protocols with explanations of why they would be good (i.e. compatibility, throughput, etc.). ChatGPT also provided tradeoffs to consider such as memory for image processing, protocol choices, or bandwidth vs latency. The information provided is much more detailed than the information output by Claude AI.

Claude AI gave a direct response to the prompt above, by giving information pertaining only to the cameras. This was unlike ChatGPT, whose large language model provided detailed information on the camera modules and possible uses for them given the application. The Claude AI model did provide relevant sources for further research on the camera module recommendations but only gives a simple bullet point list of the surface level information that requires more research. Unless I specifically ask for certain information, Claude will not expand on the options it recommended.

Although both provide adequate answers to the prompt above, ChatGPT is the preferred choice. Its ability to provide a detailed list of recommendations with considerations for several different applications/scenarios while also providing relevant sources of information makes it easier to choose an option. While Claude gives sufficient

information, it only provides a surface level view of the options requiring much more research compared to ChatGPT.

5.3 - Case Study 3

Please refer to Appendix E, [3a], [3b] and [3c] for the Case Study 1 given prompt and outcomes from ChatGPT and Google AI.

Both large language models were able to provide feasible solutions to problems pertaining to task prioritization in a 2 MCU architecture. ChatGPT [3b] [150] gave a general list of communication protocols that could be used for intercommunication, gave math to calculate throughput depending on the communication protocol, a list of tasks that are listed from high to low priority, an example table showing a potential architecture with timing, a possible implementation method, and the recommended split. Even though the amount of information is sufficient, it required further research to check the if the math provided was correct (seems correct at first glance but need to double check) and since no relevant sources were cited, all the information needs to be checked in depth. This is different from the solution offered by Google's LLM.

Google AI [3c] [157] provided a similar answer and tailored it specifically to the ESP32-S3 which was given in the prompt. It was able to pull information from relevant sources directly such as MCU capabilities from Espressif (company that makes ESP32-S3) and formulate a potential solution for how the tasks can be split between the MCUs. This makes it easier to visualize a possible path that we could take when designing the software. Aside from the numerous sources (i.e. datasheets and videos), Google's LLM did not take into account the types of peripherals and the communication protocols they use, which can significantly affect the throughput and latency of the system. So, Google AI requires more specific instructions to get additional information and compared to ChatGPT it focuses more on the technical aspects of the MCU, while ChatGPT's LLM was able to provide more information than the question asked such as showing math to calculate rate of data flow using SPI as well as the throughput which can significantly impact the time requirements, however the accuracy of it is still questionable.

Given its ability to provide numerous relevant sources and helpful information pertaining to the prompt, Google AI's answer is preferable over the one provided by ChatGPT. Although the ChatGPT LLM provides extra information beyond the scope of the question, it usually lacks any source of material for information like the math to calculate latency or ESP32 capabilities. Meanwhile, the Google LLM was capable of providing relevant information about the capabilities of the MCU and how the tasks could be split, while providing sources where further research could be performed should the user wish to delve deeper into the topic.

5.4 - Case Study 4

Please refer to Appendix E, [4a], [4b] and [4c] for the Case Study 1 given prompt and outcomes from ChatGPT and Copilot.

ChatGPT [4b] [159] and Copilot [4c] [163] were both given the above prompt, and the answers were similar, with both leading to the same answer. However, ChatGPT's

answer included a more detailed explanation for each part of the prompt, while Copilot provided a small table that briefly went over the given options. ChatGPT created bullet points for each algorithm and split the bullet points into sections based on why I would use the algorithm, why it may not be possible, and added a section on potential workarounds. Copilot provided the mentioned table, with about one sentence worth of data for the description, the pros, and the cons. Copilot also produced a detailed paragraph explain why some algorithms cannot be used and another paragraph explaining why an algorithm can be used.

ChatGPT gave an informative answer that explained in detail what each algorithm did. This is important because it helps the user to understand why the algorithm might have certain pros or cons. When explaining the reasons to use each algorithm, ChatGPT listed multiple bullet points of high-level reasons. This is beneficial when trying to understand what the algorithm does and what is needed for it, but the answers lacked any real statistics to back main points, such as accuracy. Regardless, the listed points give a blunt explanation of what each algorithm has to offer. The next set of bullet points provided go over the limitations of each algorithm. The bullets provided by ChatGPT quickly explain why something like Faster-RCNN will not be possible on an ESP32 board. This is beneficial to this project because it avoids wasted time learning and implementing an algorithm that ultimately will not work. The bullets also include numbers, such as minimum memory requirements, like the need for tens of megabytes of RAM to store the YOLO model. ChatGPT also included potential workarounds, which were not a part of the prompt. This bonus gives us ideas to research if the algorithm in question might be necessary, even if it cannot run natively on the ESP32. After all this, ChatGPT then provided a small table that summarized each of the points from the bullet list. This table alone is beneficial to group characteristics together for each algorithm. Overall, the output from ChatGPT was much more useful than that provided by Copilot.

The output from Copilot consisted of a small table, then two paragraphs explain its choice. The provided table lacked a useful description of each algorithm, forcing us to look more into algorithms that may not be used. The pros and cons of the table were sufficient but lacked any in-depth reasoning. For example, the cons for threshold blob detection only describe its lack of robustness, without mentioning other important qualities such as accuracy, or explaining why it lacks robustness. Copilot also did not mention any direct workarounds; however, it did offer a hybrid approach that combines threshold object detection with one of the CNNs. After the table, Copilot provided some bullet points as to why YOLO and Faster-RCNN won't work. These bullet points went into more detail but lacked any raw numbers for the algorithms and why they may not work. The reasons for using threshold blob detection were well written and gave some new information, such as image qualities that will work with it. The information given was sufficient to explain the final verdict, but it lacked any deep reasons and statistics.

Although both AIs provided the same result, ChatGPT provided a much more elaborate response to the prompt. Both results lead us to the answer that the prompt was asking for, but the ChatGPT one explained in greater detail the pros and cons of each option. The addition of workarounds was also very helpful, because it gave the foundation for a

follow up question if the workaround was something worth trying. The Copilot answer was sufficient but would not suffice if more information was needed, or I ultimately wanted to make the decision instead of the AI.

Ch 6. Hardware Design

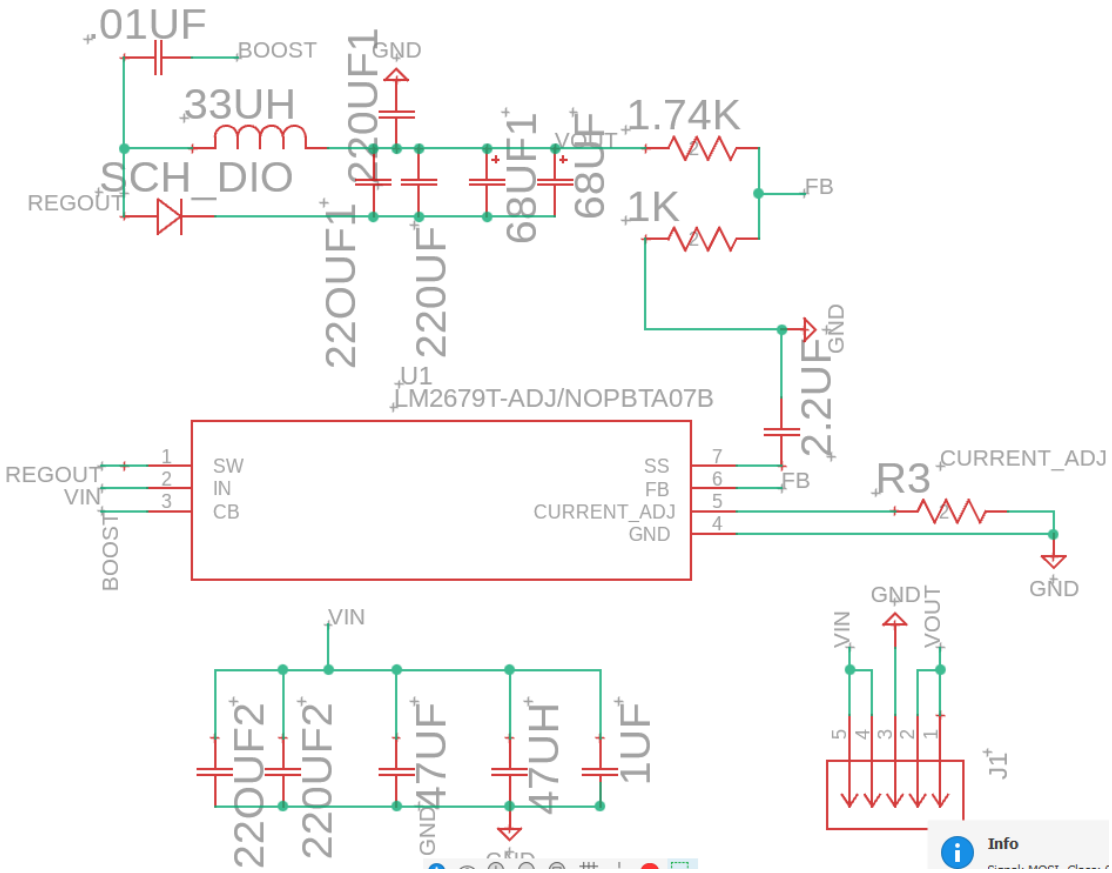
6.1 - Overview of regulators

We decided to create a unique design and try to be cost effective when it comes to needing order regulators. We decided to go with a “plug and play” approach when it comes to the output of our boards. When ordering on websites such as JLCPCB there is a

minimum count of 5 per order. It is always good to have backups which we plan on having anyway; however, we will only be using no more than two regulators at each time depending on how we build our access box and our turret PCB which will be discussed later in this section. If we need to go through a second iteration/design as well, we have to order all three sets again for the 3.3V, 5V, and 7V rails. This could start to get expensive quickly, so we went with different approaches. We decided to have all the possible combinations of resistors, capacitors, and inductors each regulator would need all on one board. The output of our regulator is dependent on the voltage divider created by the two resistors at the output of our LM2679-ADJ IC. For the 7V, it requires a different output inductor, and capacitor is the only major difference between the 3.3V and 5V. This way when we do our initial order to test, we only need to order the minimum amount of 5 boards. We can then test and see if our design worked for each output and if it does order more to complete what we need and to have extras. If we need to redo the circuit, we can place a single order again to test. If none of this works, we can of course go to the normal circuit of just the required components on board. This use of plug and play does have a huge benefit however we could have some major cost if this doesn't work out. We are hoping this approach works out the way we want which would allow us to save on cost and difficulty when it comes to board design. We will see each section highlighted below for each rail. Everything between each regulator rail will have the same layout and components except for the orange boxes which indicate what changes are changing between circuits.

6.1.1 - 3.3V Rail

Figure 4 – Regulator Design for 3.3V rail



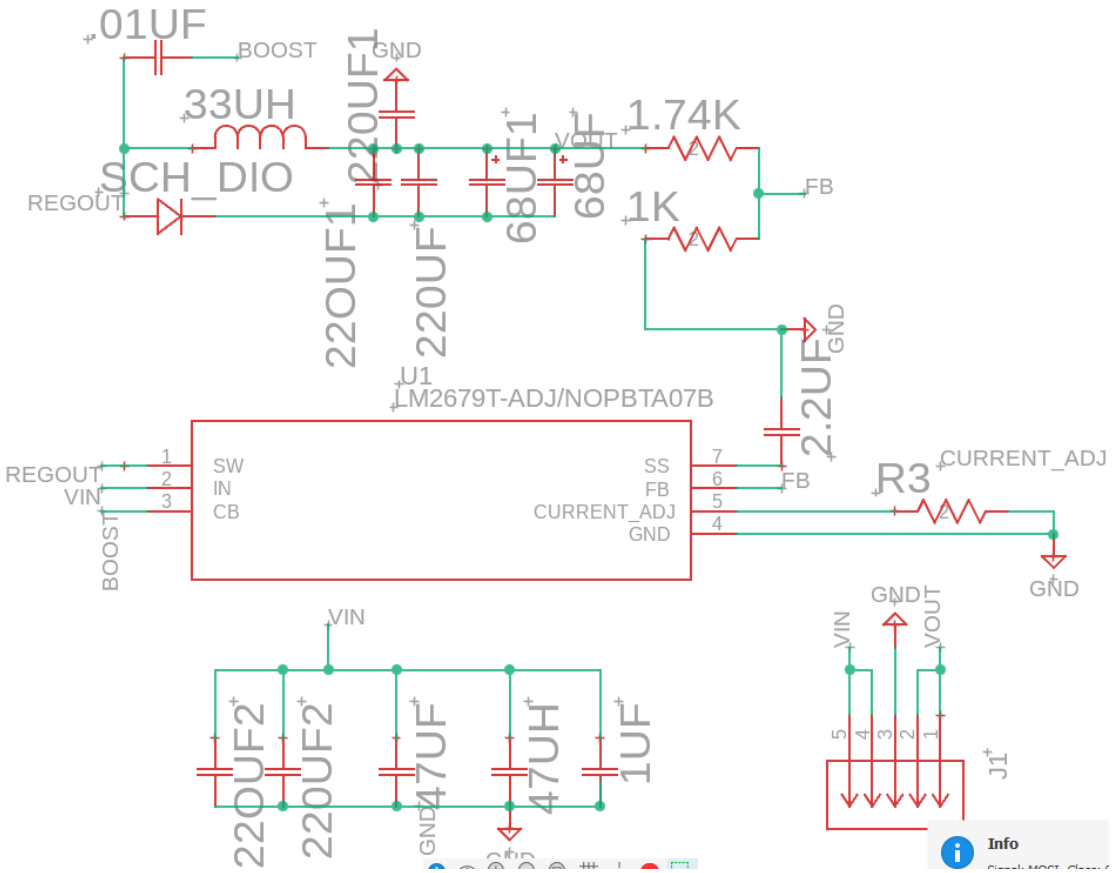
We are using the formula of Vout = Vref * (1+Rout/Rin) which is extracted from our datasheet for the LM2679-ADJ to find the ratio of each R1 and R2 that will be needed at our output to achieve our desired voltage. We are using 3.3uH inductor, 3 220uF aluminum capacitors for output capacitors, 2 220uF aluminum capacitors, 2 47uF ceramic capacitors, and 1 1uF capacitor for input capacitors.

*Vout = 1.21V * (1+(1.8k/1k)) = 3.38V*

As we see, we have a proper 3.3V expected output. Our data sheet calls for a varying amount of output. To help with stabilization of our circuit, we decided to have a mix of ceramic and aluminium capacitors. This will help keep a steady frequency for the voltage coming along with the current. The output will have a better transient response and be more stable around the 3.3V line instead of varying with the help of the extra capacitors. We added these in every circuit since there are more benefits than cost to adding these in.

6.1.2 - 5V Rail

Figure 5 – Regulator design for 5V rail

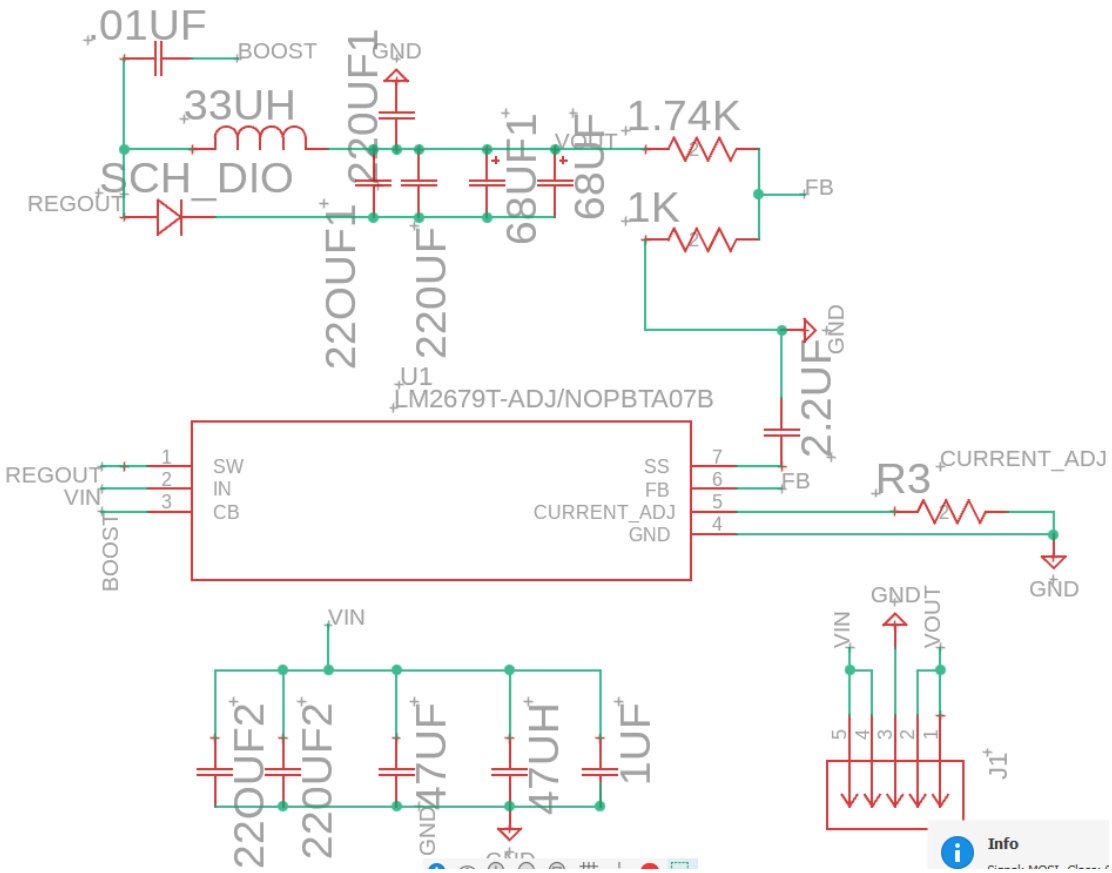


We are using the formula of Vout = Vref * (1+Rout/Rin) which is extracted from our datasheet for the LM2679-ADJ to find the ratio of each R1 and R2 that will be needed at our output to achieve our desired voltage. We are using 3.3uH inductor, 2 68uF aluminum capacitors for output capacitors. We are using 2 220uF aluminum capacitors, 2 47uF ceramic capacitors, and 1 1uF capacitor for input capacitors.

*Vout = 1.21V * (1+(3k/1k)) = 4.9V.*

6.1.3 - 7V rail

Figure 6 – Regulator design for 7V rail



We are using the formula of Vout = Vref * (1+Rout/Rin) which is extracted from our datasheet for the LM2679-ADJ to find the ratio of each R1 and R2 that will be needed at our output to achieve our desired voltage. We are using 3.3uH inductor, 1 68uF aluminum capacitors for output capacitors, 2 220uF aluminum capacitors, 2 47uF ceramic capacitors, and 1 1uF capacitor for input capacitors.

*Vout = 1.21V * (1+(4.99k/1k)) = 7.2V*

6.2 - Access Box

The use of our access box is meant to act as the brain of our system, transporting information, and relaying information to the user. We plan on having our main power switch and arm/disarm switch on here as well. This way our system is on and can alert us to objects in the area, but we won't track the objects until the arm switch is flipped on. Now we plan on having two MCUs, one on the turret PCB and another on the access box. The reason for this is the motion tracking algorithm and the data space needed for the cameras we are using will be extensive. To help with CPU utilization and latency, we have split the MCU functions. The access box will house the camera data MCU along with user peripherals such as LEDs, switches etc. The other MCU on the turret PCB will handle taking the camera data from the first MCU and we will use the turret PCB to

power and control the turret along with configuring the data for our Husky Lens sensor. Now the question is do we want to 3D-print this access box or do we want to buy a premade box of some kind. Let's look at the two choices in detail.

6.2.1 - Pelican box

A premade box option we are looking at is a pelican box. This box comes with some built in padding and is made to transfer electronic equipment over long distances due to allowing foam to be in this box to hold such elements nice and tight and has a hard exterior. A hard exterior would be much more realistic as this configuration would allow for the more portable features of our system to happen with increased protection when transporting the system. We can house multiple layers of PCBs and only have the user interface PCB on top to help with wiring/display issues. We would need to take extra precautions when making the PCBs as well in this case. The first issue is making sure the PCBs are small enough to fit in the dimensions of the box. We want our system to be small as part of our requirements, but the box limits would be extra requirements not originally accounted for. This could cause some issues down the line when it comes to design. We also need to consider spacing more in this case with external wire connections. With the thick cover of the external part of the box, making holes big enough for wires but also small enough that not too much foreign debris gets in there both when the wires are plugged in and when they are not such as when we are transporting the system. Having covers for these holes when nothing being used is something that needs to be considered in the price since finding specific hole coverings for this would be no easy challenge. Making the PCBs small enough to fit on one side of the box also allows us to house the turret and the Sentry station in the box when it's not being used as well. This way this pelican box can house everything; allowing the perfect portable box that can set up and tear down our system quickly and keep it nice and organized all in one place with extra protection in case issues arise during transportation.

6.2.2 - 3D printed box

With this option a lot gets flipped in terms of pros and cons. Here our issue is transportation and protection of the system, but freedom of use for the system is increased. Using the pelican box, we can either choose the route of deciding on the box first then fitting the PCBs to this size, or we can make the PCBs then buy a box big enough to fit what we need. Using a 3D printed box allows for the ifs to be taken care of. Designing a box would happen after PCB creation allowing more freedom of design spacing to be utilized with the PCBs. This box would be one dimensional only housing the main PCB. This main PCB would contain the user peripherals along with the regulators needed for each component and the main MCU. This allows the box to be small, and we would have minimal wires coming out of this box as well. The only wires coming out of the box should be the wires needed to power the Sentry cameras along with sending data and the power wire for the turret PCB. The turret PCB would be with the turret which would help with all the wires needed to power this portion of the system. Doing the pelican box, we will need multiple wires to come out of the box if we try to do the stacking design as mentioned where in the 3D printed version, complication of wires would be cut down significantly. The issue with these choices comes down to the

protection and transportation of the system. We plan on having all of our elements, the access box, Housing station, and turret, be separated, which again will help with designing. But transporting this system could cause significant issues to function if this is not done correctly. Wires could come undone, or something could be bent on the housing station that was not meant to be.

6.2.3 - Pelican box mixed with 3D printed box

A possible solution to our problem is mixing both scenarios described above. We could design the access box and our system to the fullest, not worrying about room for now. Once the initial testing is done, we can go back into our system and scale it down to make sure it is well below our requirements to fit in our dimensions. From there we can then look online to see if we can find some pelican boxes that fit all our parts of the system into this container. This solution could help cover each con in the scenarios explained above. The protection of transportation would be there, assuming we use foam to separate the parts. The wires coming out of the box would not be an issue since everything is still housed in the access box and turret. We would not need to worry about foreign debris coming into holes we do not expect while transporting either since the pelican box will be tightly sealed. The issue with this choice is cost. We can stay in our design requirements of being small fairly easily but redesigning to fit into a box is not a priority but a luxury. We can transport everything “unsafely” but just make sure to be careful when we are transporting each part. This choice will only be exercised later if we are well within budget and lots of this project has gone smoothly and accordingly to the plan.

6.2.4 - Decision on Access box

We have decided on 3D printing for our access box. This option provides the easiest way to implement our project with minimal hassle. We can make changes as needed and any modifications that need to happen are cheap to implement. We need room to change especially when it comes to switches and LEDs since these are the hardest to get correct when moving them from a surface mount component to a through-hole component that must be seen up top to be controlled by us.

6.3 - Overview of setup for Access Box

Below you will see the block diagram for our access box. This box is meant as the brain of our system, giving us feedback in real time based off what the cameras are seeing and is in charge of distributing power along with sending data from the cameras to the turret PCB so the turret can start its process of locking on to the enemy if there is one in range. The access box will take 12V from the power bank then distribute 12V to the turret PCB. The reason for this is we want a readily accessible switch that is near the controls for the user. The turret is not planned to be very far away from our system, but we plan on having the access box be user friendly and keeping controls in one place and everything else should be automated. We will have a switch for each regulator to help with the controls of our system and to try to isolate portions of the circuits when we can in case any rise in temperature occurs that is not expected or if the motors produce too much EMF, we have a quick way to protect the system from further harm. The access box is broken into a few different parts. The switch section meant to control power for the entire board, the regulator section which is where our regulator circuits will connect via through hole female headers, the MCU which is where the esp32 is along with its reprogramming

circuit. The buzzer circuit will be activated during mission critical times and finally the status LED section which will have multiple LEDs for any type of situation our system will encounter that should be known by the user. Below you will find the overview of the entire schematic and the rest of this section will look at each circuit individually and explain our choices.

Figure 7: Access Block Diagram with Legend

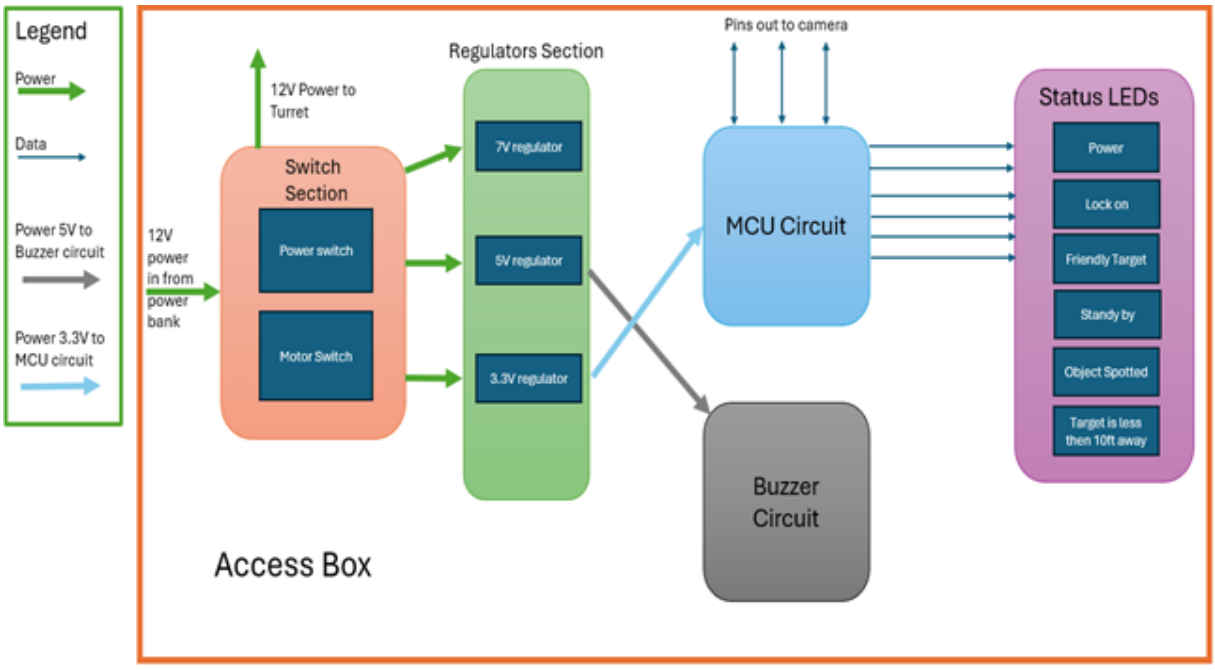
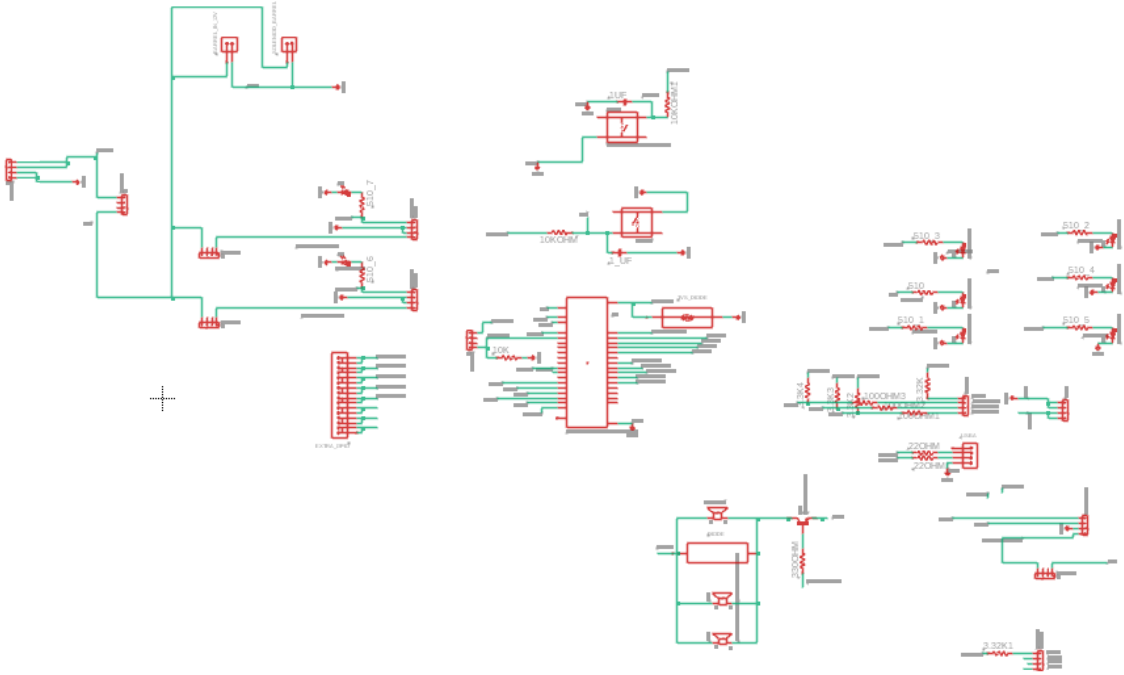


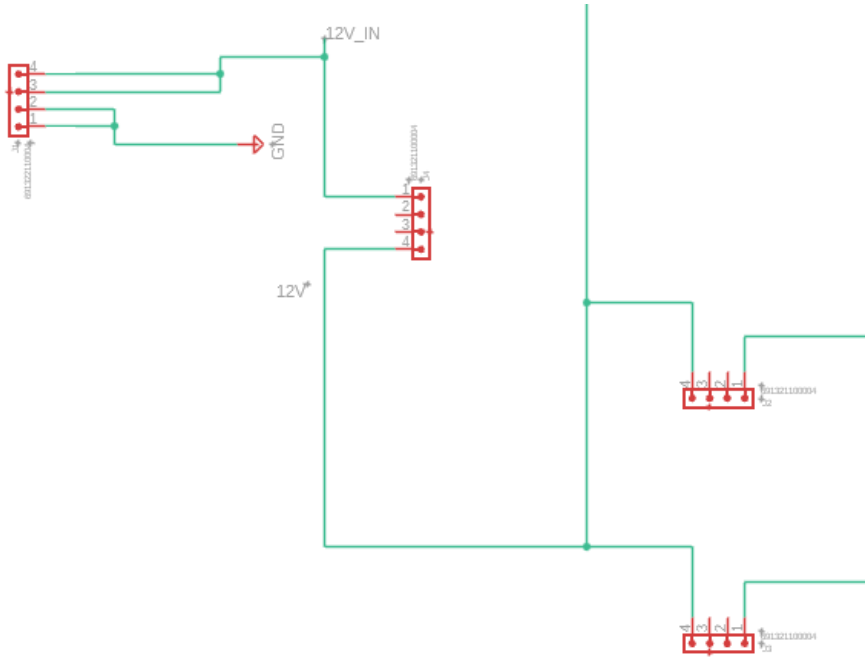
Figure 8 - Full schematic of access box



6.3.1 - Switch section

Let's start with the switch section for our first circuit. This circuit is meant to simply control the flow of power into the system. Referring to figure 9 below, the two right holders will hold the switches to the regulator. This new addition uses through hole snap in connectors so these are the female headers soldered on to the board. This is meant to help with test cases and shielding for debugging in the future. We may only want to test certain sections at one time or another and not power the rest of the system. So, we split up power to help with scenarios such as those. We also have multiple switches in case of needing to shut off the system we can do this quickly and only turn off portions of the circuits if needed. We also split up the switches to help with the handling of current into our system. Each switch is rated for 5A, and that is just about the current draw of access box. To help with the headroom and to stay safe, breaking up the regulator's power with its own dividual switches made more sense. A more robust and cleaner way is to only use one switch to power them all, but we are taking a safer, more sectional way of things as we have with the entirety of this project. The top two switches will be controlling the motors over at the turret. We plan on having the switches on the access box again so we can have a centralized control box instead of needing to have one person at our access box and one person at the turret. Due to the higher current pulled power the motors, servos, and ESC again it makes more sense to break up the switches so we can stay closer to the rating of the switch of 5A. We will need relays as well, which will be discussed further in the turret PCB section of this chapter.

Figure 9 – Switch circuit

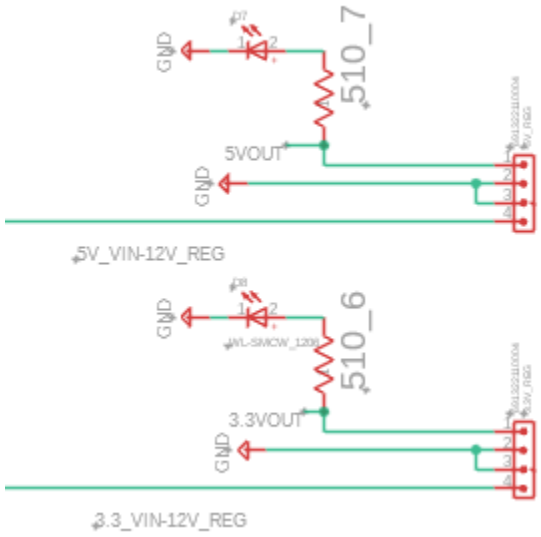


6.3.2 - Regulator Section

As discussed in the regulator section, we separated the regulators from our board to help with costs in case issues were to arise. In this portion of our access box, we have through holes which will have female headers attached so we can attach the male headers on the

regulators to our access box PCB. Pin one will take the 12V from its respective switch, pin 2 and 3 will be connected to ground, and pin 4 will be our expected output voltage for each regulator. Below you can see a more zoomed in version

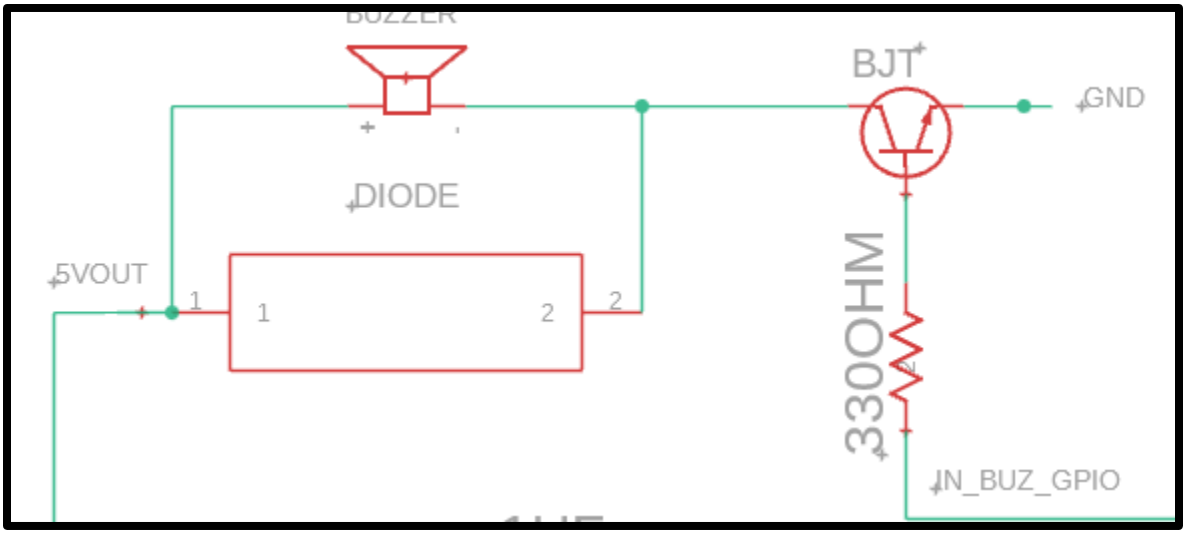
Figure 10 – Regulator section of Access box



6.3.3 - Buzzer Circuit

Our buzzer is here for mission critical alerts to the user. This will be saved for actions such as enemy lock on achieved or when a new target has entered the area. This buzzer produces a magnetic field so due to this we must use the circuit that is given in the datasheet based on the buzzer we choose. It has a few components such as the buzzer itself, a diode, a MOSFET transistor, and a resistor of 330 ohms. The reason for the buzzer is to produce the sound we want during these alerts as explained before. We have what's called a flywheel diode which is there to help counteract the magnetic field that is created by the buzzer when it switches from the on to off position. The transistor acts as a switch, and the 330-ohm resistor is to help protect the transistor from current being too high. 5V is fed into the positive terminal of the buzzer and a GPIO pin is fed into the 330ohm resistor which is connected to the base of our transistor. The emitter is then connected to ground, completing our circuit. Our input must be a square wave, aka we must turn on the GPIO that our buzzer is connected to, add a delay, then turn it low and loop to get the buzzer sound. We cannot just apply a GPIO pin and put it in a constant high position like an LED can handle; this signal must be in the form of a square wave.

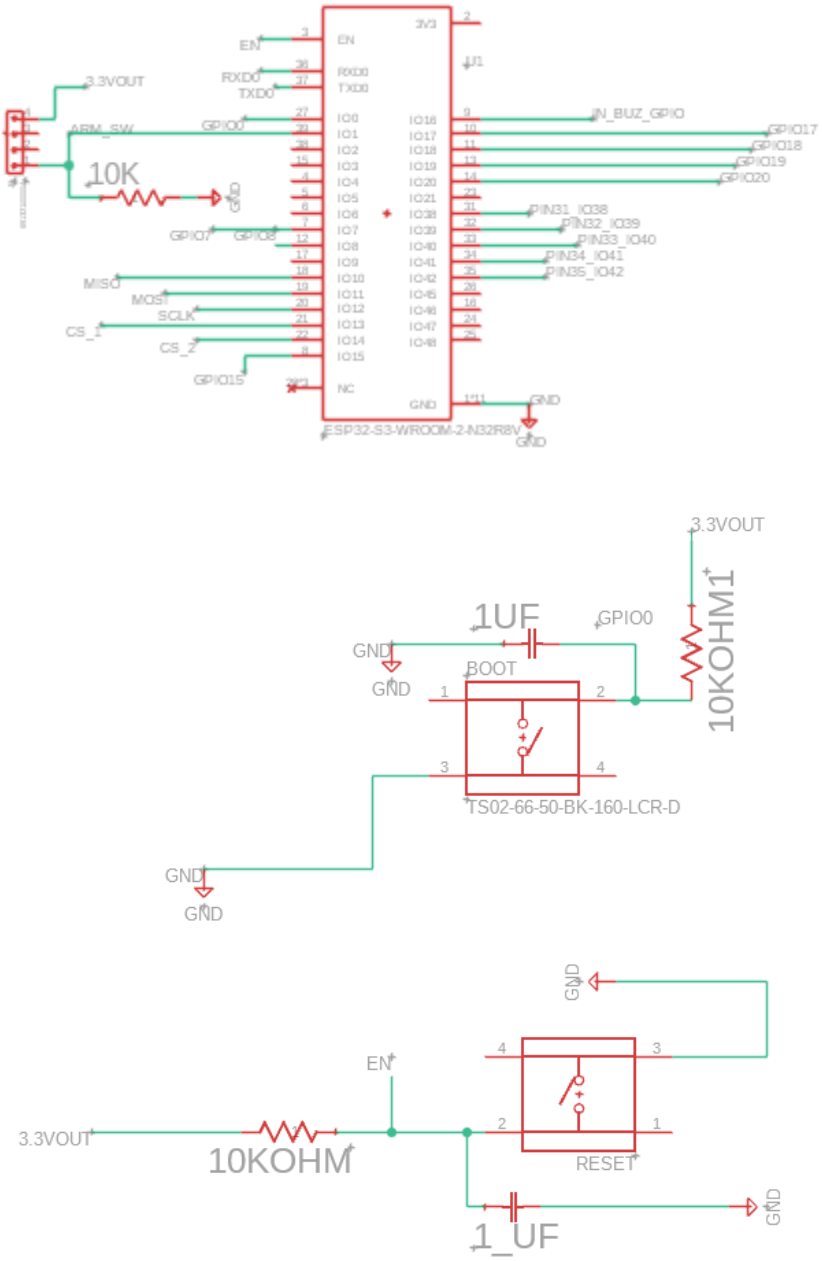
Figure 11 – Buzzer Circuit



6.3.4 - MCU circuit

In our project, we are not allowed to use a development board which means we need a specific way to program our chips. The ESP32 we have chosen has a few different ways to achieve this however we have decided to use this circuit to do that. As we can see, our circuit has switches on the ESP32 along with some LEDs. These switches are here so we can set the ESP32 into programming mode, which will allow us to flash code from the development board to the bare chip so we can update our design as time goes on. We need two switches in this case, one for the enable signal and one for the GPIO0 pin. The GPIO0 pin will act as our BOOT button which in tandem with the enable signal will set the esp32 into programming mode. It will stay in this mode until code has been sent to the bare chip through RXD0 and TXD0 which will be connected to similar pins on the development board of the ESP32. We have pulled the switches high using a pull up resistor and have a status LED to show that these have been pulled up. We will be using surface mount LEDs here instead of through holes since we don't need to see the lights when the cover of the access box is on. We will take the board out to reprogram it when needed as the point of the project is to just flip a switch and it be self-sufficient from there with predetermined targets. To put our ESP32 into programming mode, we will hold the enable in a low position and flip the switch for GPIO0 from high to low and back to high which puts it into our programming mode so we can change the behavior of the system as needed.

Figure 12 – MCU circuit

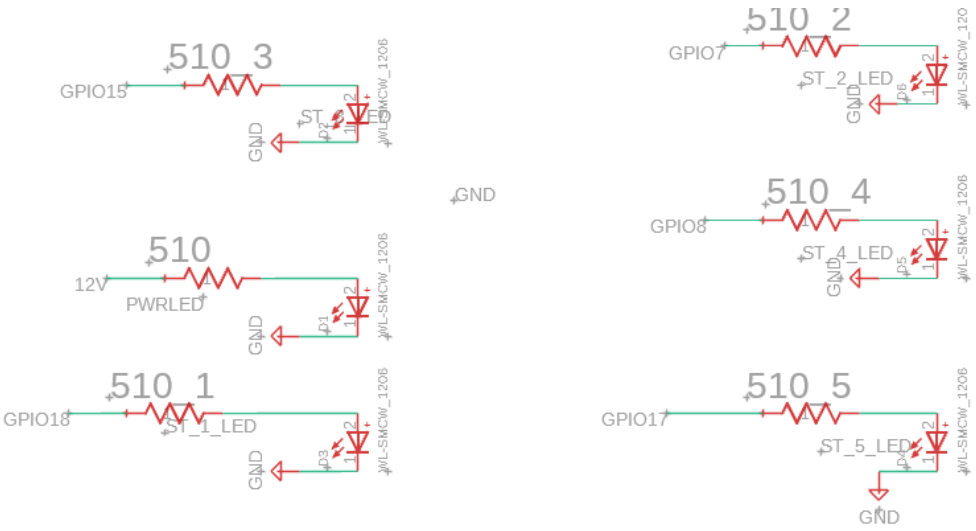


6.3.5 - Status LED section

Our last section is the status LED section. This section is meant to serve as the main way to alert the user as to what is happening. We plan on having at least 6 LEDs to show a few different scenarios. One will be for power and knowing the power bank voltage is received before our switches have even been turned on. One LED will be for when our turret has been locked on to a target. At this moment our buzzer circuit will trigger as well. We want to be able to distinguish between enemies and friendly targets, so we will have another LED for if the camera is tracking a friendly target or an enemy target. We

will break this up into a few LEDs if an enemy is spotted. First, we will turn on an LED if an object is in range of detection, next we will then identify a friend or enemy. If it's an enemy, no LED will turn on until the target is within 10 feet, at which point the camera will then send data to the turret to start its lock on process. Our last LED is a standby signal to show that power is in the system; everything is operational, but no objects are within the cameras field of view. Our LEDs are plugged into our two-pin female header and have a current limiting resistor attached to GPIO pins from the ESP32.

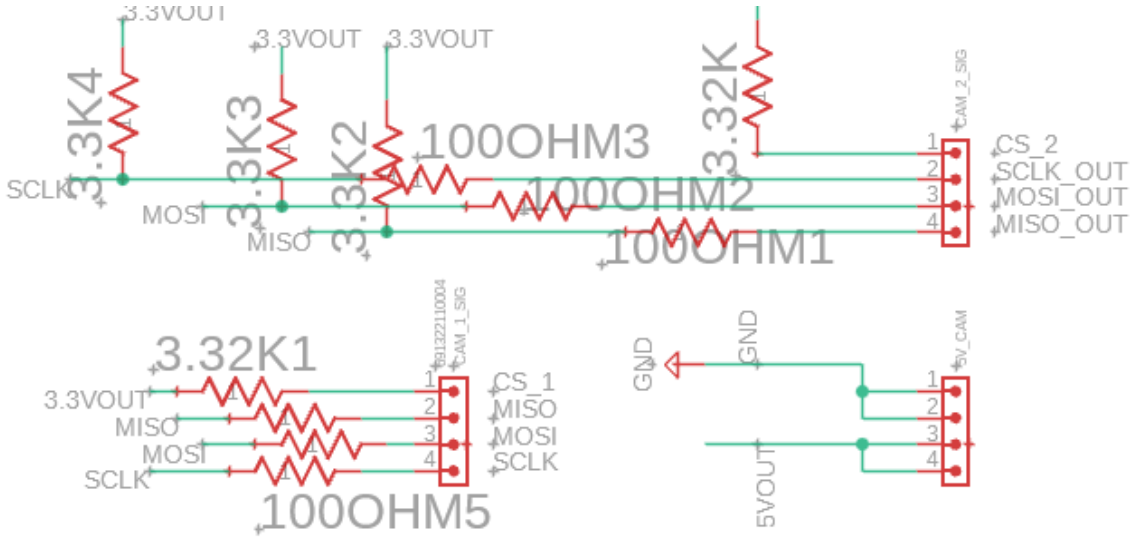
Figure 13 – Status LED



6.3.6 – Camera Section

The Sentry Cameras are the first line of detection in this project. They will utilize SPI communication using MOSI, MISO, SCLK and chip select. It will also need power of 5V and GND connected totaling 6 pins for each camera. Each line will need a pull up resistor of 3.32k ohms up to 3.3V. They will also need series resistors of 100ohms to help with noise which could degrade the max speed of operation for the cameras on each of the signal lines.

Figure 14 – Camera Pin outs



6.4 Power Delivery / Electrical Power System

The turret is powered exclusively from mains via a 12V, 50A industrial enclosed supply. Eliminating batteries simplifies compliance and removes charging risks while giving ample headroom for simultaneous flywheel spin ups. The 12V bus enters the access box through a fused inlet and lands on a relay power board that performs high current switching, branching, and protection. From there, the bus splits into three paths: 1. two independent branches to the BLDC ESCs each with its own inline fuse and bulk capacitors, and to a regulator branch for auxiliary rails.

Two point of load regulators derive the low voltage rails. A synchronous buck converts to 7.0 to 7.4V for the high torque servos and another synchronous buck provides 5V for logic, camera, and the aim laser. Rails are monitored with simple indicator circuits so that a technician can verify power state at a glance. To preserve bus stability during motor events, each ESC input is locally decoupled with 2x 2200 to 3300 µF low ESR capacitors and the 12V trunk is clamped with a TVS diode near the supply entry. This arrangement keeps the 12V rail stiff during simultaneous ramps while isolating digital loads from motor transients..

6.4.1 - Bring-up behavior

On power up, only logic rails energize. Once the access box MCU finishes self checks, it asserts the main relay to feed the ESC/servo bus. Emergency stop will open the relay coil supply and removing power from all actuators. With Soft Start enabled in the ESCs and local bulk capacitance, measured currents are expected to be around 24 to 36A during the first 0.2 to 0.4 seconds when both flywheels spin up simultaneously, and 6 to 10A steady, leaving margin for dart contact spikes and servo corrections.

The sequencing is important because if we tried to power everything at once, the inrush current from the ESCs could cause the power supply to fault or the voltage to dip enough that the microcontroller resets. By bringing up logic first and waiting for it to stabilize, we ensure the system can properly control the relay before high current loads come

online. This also gives the microcontroller time to run its startup checks and make sure everything is in a safe state before motors get power.

6.5 Turret PCB

The Turret MCU board is the main controller for the turret. It's built around the ESP32-S3 and handles the local low-voltage I/O including flywheel ESC PWM signals, pan and tilt servos, HuskyLens camera, laser control, IMU, solenoid actuation, and communication with the access box controller. The access box has its own PCB and microcontroller, and the two boards communicate over UART through the main harness.

For programming and debugging, a USB-A connection was included for ease of use so the turret PCB can connect directly to a computer. Standard boot circuitry was still kept on the board with 10 kΩ pull-ups on EN and GPIO0, along with BOOT and RESET buttons for firmware loading and troubleshooting when needed.

For sensing, the HuskyLens and IMU share the I2C bus. Pull-up resistors on SDA and SCL were required so the bus would have a cleaner and more reliable signal. Since the ESP32-S3, I2C bus, and camera interface all need clean routing, ground pours and stitching vias were used to improve return paths and reduce interference. Higher-power wiring was also kept separate from signal wiring to reduce noise coupling into the control section.

The actuation breakouts are arranged as dedicated labeled headers. The two ESC throttle ports each get an MCU PWM signal through a small series resistor, and a pull-down is used at the connector so the motors stay idle on boot. The pan and tilt servo signals follow the same signal treatment, while the laser control line also includes series resistance and a pull-down so it stays off by default. These safe defaults make sure the turret doesn't unintentionally activate during startup or reset.

The board gets 12V from the access box and generates its local supply rails onboard. These include two separate 7V regulators, one for each servo, along with one 5V regulator and one 3.3V regulator for the remaining electronics. This keeps the servo loads isolated from the logic rails and helps reduce voltage droop when the servos move under load. Regulator status indicators were also included so rail conditions can be checked more easily during testing.

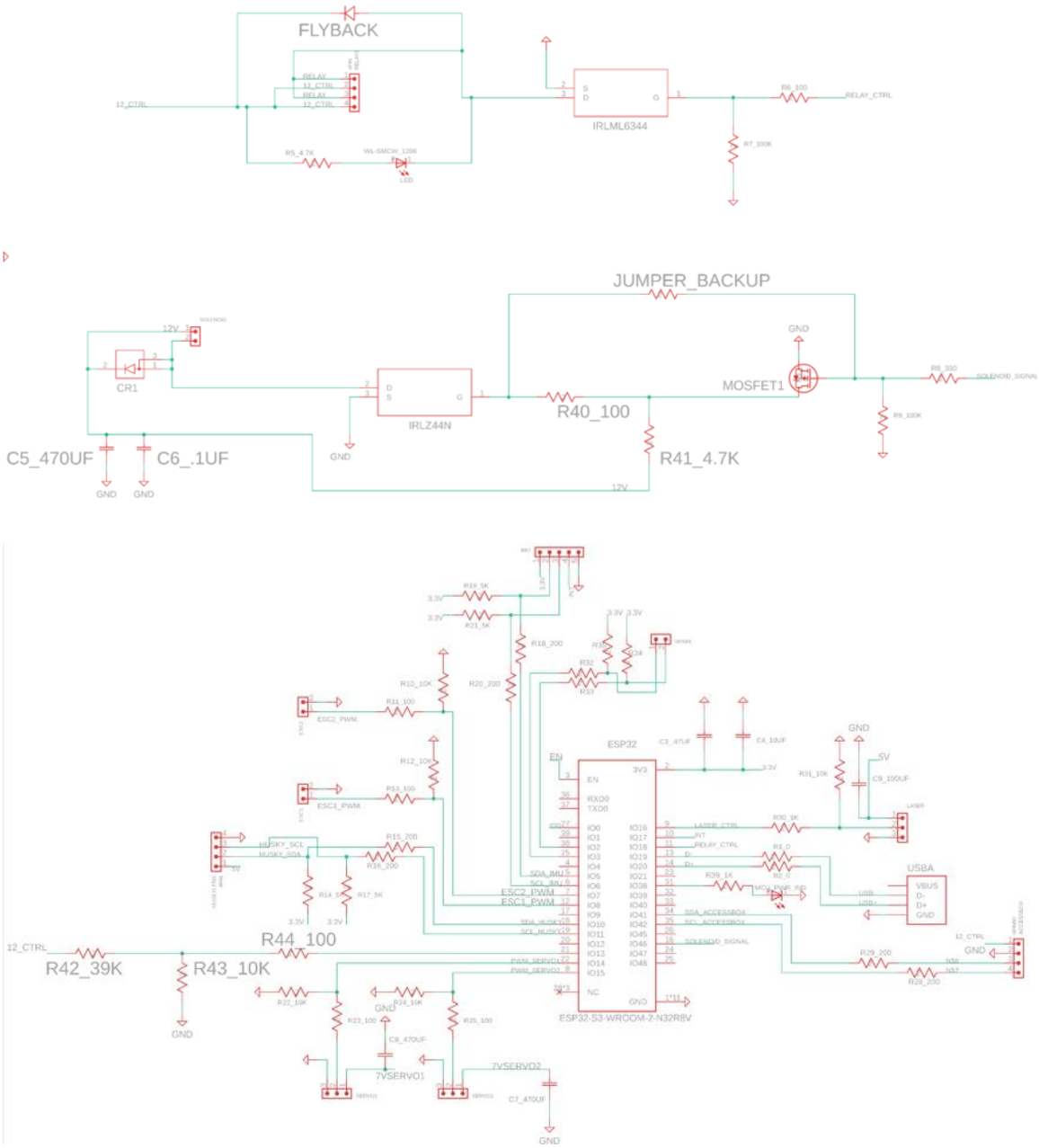
The solenoid used to actuate the darts is also connected directly to the turret PCB. Since it can draw up to about 8A, it's driven by an IRLZ44Z MOSFET, and that path was done using copper pours instead of small signal traces so the board could safely handle the current.

Two separate 12V lines are routed from the access box to the turret. One line provides the main turret power input. The second 12V line is switched by the emergency stop in the access box and is used for relay control. That switched line is also stepped down and brought into the turret PCB so the controller can monitor and use the state of relay-side

power for other peripheral control. The turret PCB only handles the logic-level control of the relays through ESP32 GPIO pins and MOSFET drivers. The high-current relay power path itself is kept separate from the PCB.

For debugging purposes, test pads and vias were added so signals can be probed more easily during bring-up and troubleshooting. Mounting holes line up with the 3D-printed standoffs in the turret so the PCB is mechanically supported and the connectors don't carry structural load while the turret moves.

Figure 15



6.6 Rail Monitoring and Thermal Considerations

Each regulator includes a status LED so rail conditions are immediately visible during testing. Components with notable heat dissipation, especially the 7 V regulators, were given thermal consideration during layout and placement. High-current runs were kept short, and the 12 V input wiring was arranged with proper strain relief to reduce losses and improve reliability.

The ESCs are not mounted inside the turret enclosure. Instead, they're mounted outside the turret for better ventilation and thermal performance. This improves airflow around the heatsinks and prevents heat from building up inside the turret structure. Keeping the ESCs outside the turret also separates a major heat source from the PCB and other sensitive electronics.

Thermal management was especially important because the ESCs switch high current at high frequency and the servo regulators have to handle dynamic load changes. By separating the higher-dissipation components and maintaining cleaner airflow around them, the system stays more stable during extended operation.

6.7 BLDC Flywheel Drive

The launcher uses two BLDC outrunners, one per flywheel, each driven by its own Hobbywing Skywalker 40A ESC. The controller inputs are standard RC PWM signals from the turret MCU with a shared ground. Signal lines are routed with small series resistors at the MCU pins to reduce ringing and improve signal quality over the harness.

The ESCs are configured with the LED program card using Brake OFF, Soft/Medium Start, Timing Low, Governor OFF, and LVC disabled for PSU operation. These settings minimize inrush current, avoid regenerative braking, and keep throttle response predictable when running from the power supply instead of a battery.

The main goal was to keep the motor current loops close to the ESCs. Each ESC branch includes its own fuse, and the wiring from ESC to motor uses locking connections for reliability. The ESCs are mounted outside the turret for ventilation, while their PWM control still comes directly from the turret MCU. This arrangement keeps the high-current motor hardware physically separated from the main control board while still maintaining direct control from the PCB.

The series resistors on the PWM lines help reduce ringing on the signal edges. Without them, the wire runs to the ESCs can reflect edges and introduce noise on the PWM signal. Using small series resistance improves signal integrity without affecting the control levels the ESCs expect.

6.8 Pan/Tilt Servo Actuation

A pair of digital servos provide pan and tilt. Pan is geared 1:1 to preserve a full 270° sweep while maintaining control authority. Tilt uses a 30T:90T gear ratio to multiply torque and hold angle against gravity with reduced buzz. Both servos are commanded directly from the turret MCU timer PWM outputs. Cabling to the moving head is dressed with a service loop and kept clear of the flywheels.

Each servo is powered from its own dedicated 7 V regulator. This helps isolate load transients and reduce voltage droop when the servos move under load. If both servos were tied to the same rail, their current spikes could disturb the logic supply and create resets or unstable sensor behavior. By giving each servo its own regulated supply, the higher-current motion loads stay more isolated from the sensitive digital electronics.

The 3:1 gearing on the tilt axis was chosen after testing different ratios. Direct drive didn't provide enough torque to hold the launcher assembly steady, especially at shallow elevation angles where gravity creates the highest moment. The selected ratio gives enough mechanical advantage for stable positioning without making motion too slow.

The service loop in the servo cabling is important for reliability. As the turret pans back and forth, the cables need to flex without placing stress on solder joints or connectors. The extra cable length arranged in a gentle curve helps absorb that motion without sharp bending.

6.9 Relay Logic Interface

The turret PCB only handles the logic-level interface for relay control. The relays themselves are physically mounted off-board on the 3D-printed turret, while the high-current relay power path stays separate from the PCB. This keeps the main power switching hardware out of the control board layout while still letting the ESP32-S3 control relay behavior.

For the relay driver stage, a small logic-level N-MOSFET is used on the low side of each relay coil. These MOSFETs fully turn on with 3.3 V gate drive, have low on-resistance compared to the relay coil, and are rated well above the coil current and control voltage. This lets the microcontroller switch the relay coils reliably without stressing the GPIO pins. Each relay coil has its own clamp diode to handle the energy from the collapsing magnetic field and prevent high-voltage spikes across the MOSFET when the coil is switched off.

The MOSFET gate network uses a 100 Ω series resistor and a 100 kΩ pull-down. The series resistor limits inrush current into the gate and damps fast edges, while the pull-down keeps the MOSFET turned off when the microcontroller is in reset or the pin is floating. This prevents the relay from turning on accidentally during power-up.

A switched 12 V control line from the access box is used to represent the emergency-stop-controlled relay side. That line is stepped down and brought into the turret PCB so the controller can sense the state of the emergency button and disable other components if needed. In this way, the turret PCB doesn't carry the relay power path itself, but it's still aware of relay-side power state and can respond accordingly.

Chapter 7: Software Design

There is one system for this project that handles all automated tasks and requires no human-machine interface (HMI). This system runs on code flashed to both ESP32s, each of them working together autonomously. The software on these MCUs is the brain that controls all the hardware components, ensuring that the system functions as desired. It is responsible for configuring external sensors, processing data, and controlling motors, LEDs, and the laser diode. The software itself controls multiple different components; therefore, it is important to split the whole system apart into smaller subsystems that are easier to manage and integrate. This project utilizes a two MCU architecture, therefore the software will be split between the two ESP32-S3 microcontrollers as each MCU will have its own tasks to perform. Each MCU is responsible for a separate set of tasks to ensure the performance of the turret system is reliable and consistent under real-time conditions. The first ESP32-S3 (master) microcontroller will be responsible for motor control (pan-tilt mount), HuskyLens AI vision sensor for fine-tracking of target, and communication between the two controllers. The second MCU is primarily for image capturing and processing through lightweight image detection but also controls peripherals (LEDs and buzzers).

Integrating both software components together forms the turret system to ensure that it has a quick response time with accurate tracking abilities, provides timely visual and auditory feedback through several LEDs and buzzers, and each component in the system has the ability to work together seamlessly. This section will outline how each system was designed and integrated into our project.

7.1 - Primary MCU

The primary ESP32-S3 is the central controller of the system as it handles most of the control tasks. Its tasks involve handling servo motors for a stable pan-tilt mount, processing high-level image data incoming from the secondary MCU and HuskyLens, as well as managing the feedback mechanisms. It will also undertake control of safety mechanisms and state transitions (idle, tracking, etc.).

Goals:

1. Control the pan and tilt servo motors using PWM signals and a PID algorithm for smooth and stable motion.
2. Fine tracking using positional data retrieved from the HuskyLens via the I2C communication protocol.

3. Auditory and visual feedback mechanisms that are able to activate within 500ms of detecting a target or an error.
4. Receive target data from the secondary MCU that takes input from two stationary cameras via the SPI interface and incorporates it into the tracking algorithm for the HuskyLens to use.
5. Handle real-time user input for cases such as emergency shutdown.

The tasks are divided between cores (dual-core CPU) using FreeRTOS (Real Time Operating System). It allows for parallel task operation by separating control, feedback, and communication into individual tasks. The tasks are prioritized accordingly with servo motor control and safety monitoring at the highest priority to maintain deterministic timing, and the auditory and visual feedback mechanisms such as the LEDs and buzzers along with I2C communication (HuskyLens) operate at a lower priority but maintains responsiveness. External triggers such as target detection, loss of signal, or lock on handled immediately using interrupt routines. This setup allows for tasks to be performed parallelly while maintaining the stability of the system and operating in real time consistently.

7.1.1 HuskyLens 2 YOLO Model Implementation

S.T.A.T uses a custom YOLOv8n object detection model deployed on the HuskyLens 2 to identify red balloon targets in real time. The HuskyLens 2 is powered by the Kendryte K230, a dual-core RISC-V processor with a 6 TOPS neural processing unit, capable of running quantized inference at 15–25 FPS. The custom model was developed to replace the HuskyLens 2's built-in color recognition algorithm, which produced excessive false positives from any red object in the scene — including clothing, vehicles, and signage. By leveraging the YOLOv8n architecture, the model incorporates both shape and color features simultaneously, allowing it to target red balloons from other red objects with significantly higher reliability in uncontrolled outdoor environments.

The dataset used for training consisted of 775 labeled images split 87% train, 8% validation, and 4% test. Images were sourced from a combination of public balloon datasets on Roboflow Universe and field-captured photos taken at 2–8-meter engagement range under varied lighting conditions, angles, and backgrounds. All images were preprocessed with auto-orientation correction and resized to fit within 320 × 320 pixels to match the K230 deployment resolution. To improve generalization, each training image was augmented three times using horizontal flipping, brightness jitter of ±20%, exposure jitter of ±10%, and up to 1 px of blur — simulating the lighting variation and minor motion blur expected in real outdoor deployment conditions.

Training was performed on Google Colab using a T4 GPU, running the YOLOv8n architecture for 100 epochs at 320 × 320 input resolution with a batch size of 16. After training, the best-performing checkpoint was exported to ONNX format (opset 11) and then converted to the K230 KPU deployment format using DFRobot's onnx2kmodel tool, which performs uint8 quantization using 50–100 held-out calibration images. The quantized model was packaged and installed directly to the HuskyLens 2 via USB. The final model achieved a mAP@0.5 of 0.95–0.96, well above the 0.80 minimum

deployment threshold, indicating strong detection accuracy and generalization across the validation set.

Once deployed, the HuskyLens 2 continuously streams bounding box detections — center X, center Y, width, and height — to the ESP32 master MCU over I²C at address 0x50. The ESP32 outer control loop, running at 30 Hz, reads these blocks and computes the pixel error between the detected balloon center and the camera's optical center. This error is fed into a PD control loop that incrementally drives pan and tilt servo setpoints until the balloon is centered within the aim tolerance. A minimum bounding box size filter rejects detections that are too small, eliminating false positives from balloons outside the engagement range. Lock-on is confirmed after two consecutive frames within the aim tolerance, at which point the brushless motors ramp up and the solenoid fires.

Table 39: Balloon Detection YOLO Model Dataset

Parameter	Value
Target Class	Balloon (Single Class)
Total Images	775
Labeling Tool	Roboflow (bounding box, YOLOv8 export format)
Annotation Format	YOLOv8 (.txt per image, normalized coordinates)

Table 40: Balloon Detection YOLO Model Dataset Split

Split	Images	Percentage
Train	678	87%
Validation	65	8%
Test	32	4%

Augmentation was applied only to the training images, with 3 augmented outputs per source image (effective training set: ~2,034 images):

Table 41: Balloon Detection YOLO Model Augmentation

Augmentation	Parameters
Horizontal Flip	Applied
Brightness jitter	±20%
Exposure jitter	±10%
Blur	Up to 1px

Augmentations were added to simulate realistic deployment variation: changing ambient light levels (brightness/exposure), camera motion blur at acquisition, and left/right symmetry of ballon targets.

Table 42: Balloon Detection YOLO Model Performance Metrics

Parameter	Value
mAP@0.5	0.95-0.96
Input Resolution	320x320
Inference Rate	15-25 fps

An mAP@0.5 of 0.95-0.96 significantly exceeds the 0.80 deployment threshold, indicating strong generalization across the validation set. The result is consistent with the augmentation – brightness, exposure, and blur variation during training reduces overfitting to controlled lighting conditions, which is the primary source of mAP degradation in outdoor deployments.

7.1.2 System Controller

The SystemController class acts as the top-level orchestrator for the entire system. On startup it initializes all hardware components — servos, IMU, HuskyLens, brushless motors, UART, and GPIO monitor — and arms the ESCs with a 5-second stop signal before creating the main control task. The control task runs a continuous loop that non-blocking polls three FreeRTOS queues for key status updates, emergency shutoff events, and handshake requests from the access box, then blocks for up to 100 ms waiting for a bounding box packet from the camera service. When a packet arrives it is forwarded to

the BoundingBoxAggregator for confidence filtering before any tracking decision is made. The SystemController also handles the handoff between the coarse acquisition phase (ArduCam cameras) and the fine tracking phase (HuskyLens 2), stopping the camera service and starting the tracking service once a confident target has been identified.

7.1.3 Coarse Target Acquisition – Arducam and Bounding Box Aggregator

Initial target detection is handled by stationary ArduCam cameras whose bounding box outputs are processed by the CameraService and forwarded to the SystemController via a FreeRTOS queue. To prevent false triggers from single spurious detections, all incoming bounding boxes pass through the BoundingBoxAggregator before any tracking action is taken. The aggregator maintains a list of tracked boxes across frames, matching new observations to existing tracks using a spatial distance threshold of 20 pixels. A box must appear consistently for a minimum of 2 frames (MIN_NUMBER_FRAMES) before it is considered a confident target. Boxes that have timed out after 5 frames of absence are pruned automatically. Once a confident target is confirmed, the aggregator selects the largest persistent box by area, and that box's 3D position data (center X, center Y, depth) is used to compute an initial pan and tilt servo setpoint via inverse tangent. Friend detections are discarded at this stage — if the target box is flagged as friendly, the aggregator resets and acquisition resumes. Boxes that have recently been tracked are placed on a 30-frame cooldown to prevent the system from re-acquiring and re-firing at the same target immediately.

7.1.4 HuskyLens 2 Fine Tracking – Outer Loop

Once the SystemController hands off to the TrackingService, the outer control loop takes over at 30 Hz. This loop queries the HuskyLens 2 over I²C each frame using the custom balloon YOLO model (algorithm ID 0x80) and receives bounding box detections as pixel coordinates on a 640 × 480 frame. The pixel error between the detected balloon center and the optical center (320, 240) is converted to degrees using the HuskyLens 2's ~65° horizontal field of view. An exponential moving average filter with $\alpha = 0.6$ smooths the raw pixel coordinates each frame to reduce detection jitter, with the filter snapping directly to the raw value after three or more consecutive missed frames to avoid convergence lag on re-acquisition. The filtered error is fed into a proportional control law (PAN_OUTER_KP = 0.1, TILT_OUTER_KP = 0.15) that produces an incremental update to the pan and tilt setpoints each frame. A deadband of 1.0° suppresses setpoint updates when the error is below the HuskyLens jitter floor, preventing random-walk

accumulation on a stationary target. Each frame's step is clamped to ±7° to prevent large jumps that would cause overshoot. If a servo axis saturates at its travel limit for 15 consecutive frames while error still pushes in the same direction, the target is declared out of range and acquisition restarts. If no detection is received for 1000 ms, a software I²C recovery routine is attempted once before the tracking session is abandoned.

7.1.5 Lock-On and Firing Sequence

Lock-on is confirmed when the filtered pixel error falls within LOCK_ON_AIM_TOL_DEG (1.75°) of the aim point for LOCK_ON_FRAMES (2) consecutive frames, and the key switch is active. The aim point includes empirically tuned pan and tilt offset constants (PAN_AIM_OFFSET_DEG, TILT_AIM_OFFSET_DEG) that compensate for physical misalignment between the HuskyLens 2 optical axis and the dart barrel trajectory. Once lock-on is confirmed, the outer loop spawns a lockOnTarget task that immediately ramps both brushless flywheel motors to operating speed (MAX_MOTOR_SPEED = 45%), sets a g_firing flag that freezes the inner loop servo position to prevent solenoid-induced jitter from disturbing the aim, waits for the settle delay (LOCK_ON_TIMEOUT_MS), and then fires the solenoid for 150 ms (SOLENOID_PULSE_MS). The dart counter decrements on each shot. After firing, the g_firing flag clears, motors ramp down, servos return to center, and the camera service restarts to begin a new acquisition cycle.

7.1.6 IMU Stabilization – Inner Control Loop

The inner control loop runs at 200 Hz on a dedicated FreeRTOS task pinned to Core 1, independent of the outer loop. It reads the ISM330DHCX IMU over I²C at 100 kHz and computes stabilization corrections that compensate for platform disturbances such as vibration from the solenoid and brushless motors. Pan (yaw) is tracked by pure gyroscope integration of the Z-axis at 0.00875 °/LSB sensitivity, while tilt (pitch) uses a complementary filter blending the gyroscope ($\alpha = 0.95$) with an accelerometer anchor derived from the X and Z axes, preventing long-term tilt drift. At the start of each tracking session, the integrated pan angle is reset to the servo center position (PAN_ANGLE_CENT = 105°) to eliminate accumulated gyro drift from previous shots. The inner PID controllers (PAN_INNER_KP = 0.2, TILT_INNER_KP = 0.2) output a correction in degrees that is summed with the outer loop's setpoint as a feedforward-plus-correction drive to the servo. During the solenoid firing window, the inner loop skips all IMU reads and servo writes entirely, holding the last LEDC duty value to prevent mechanical shock from corrupting the gyro integration or disturbing the aim.

7.1.7 Servo Control

Pan and tilt servos (SF3218MG) are driven by the ESP32-S3's LEDC peripheral at 50 Hz with 14-bit resolution, providing a duty cycle range of 500–2055 µs mapped across a 210° travel range. Servo angles are set by converting a degree value directly to a PWM pulse width. The inner loop writes new servo positions every 5 ms (200 Hz), though the LEDC peripheral holds the last duty cycle autonomously between updates, ensuring no glitching if a task is delayed. Angle outputs from the inner PID are clamped to the physical travel limits (PAN: 0–210°, TILT: 0–180°) before being written to prevent mechanical binding.

7.1.8 Brushless Motor Control

The two brushless flywheel motors are driven by ESCs controlled through a second LEDC timer at 50 Hz with 14-bit resolution, sharing the same PWM protocol as hobby RC ESCs (stop at ~640 counts, full throttle at ~2000 counts). Both ESCs are armed at startup with a 5-second stop signal. Motor speed changes are handled asynchronously via FreeRTOS tasks: rampUp and rampDown each spawn a dedicated task that steps the speed by 5% every MOTOR_RAMP_UP_DELAY_MS / MOTOR_RAMP_DOWN_DELAY_MS milliseconds, allowing the main control logic to continue running without blocking. Operating speed is set to 45% (MAX_MOTOR_SPEED) to balance dart velocity and noise. Motors ramp down automatically when the key switch is off, when tracking completes, or when the emergency shutoff is triggered.

7.1.9 Safety Systems

Two independent safety mechanisms prevent unauthorized firing. The key switch, monitored via UART from the access box, gates the lockedOnAllowed flag in the tracking service — lock-on confirmation is blocked entirely when the key is off, and the motors ramp down immediately on key removal. The key status message also reloads the dart counter to the full magazine count (MAX_DART_COUNT = 8). The emergency shutoff switch is monitored by the GpioMonitor component via a hardware GPIO interrupt. When the shutoff is disengaged, all services stop immediately and a SYSTEM_DOWN packet is sent to the access box over UART. Re-engaging the shutoff sends a SYSTEM_RESTART packet and triggers esp_restart() to perform a full system reboot, ensuring all state is cleanly reinitialized. A dart counter prevents firing when the

magazine is empty, logging a warning and signaling target lost rather than attempting a shot.

7.1.10 UART Communication – Access Box Interface

The ESP32-S3 communicates with the access box over UART at 230,400 baud using a fixed 4-byte packet protocol (0x55, 0x00, <CMD> , 0xAA). The UartMonitor component runs a dedicated receive task that parses incoming packets and routes them to typed FreeRTOS queues — key status to keyStatusQueue, shutoff events to shutoffStatusQueue, and handshake requests to handshakeStatusQueue — which the SystemController drains each cycle. Outgoing packets are written directly via UartDriver and include system state notifications such as handshake acknowledgment, lock-on achieved, shot fired, buzzer on/off, and system down/restart. A handshake exchange is performed at startup to confirm the access box is online before the system enters its acquisition loop.

Figure 16 - Primary MCU Outer Loop Flowchart

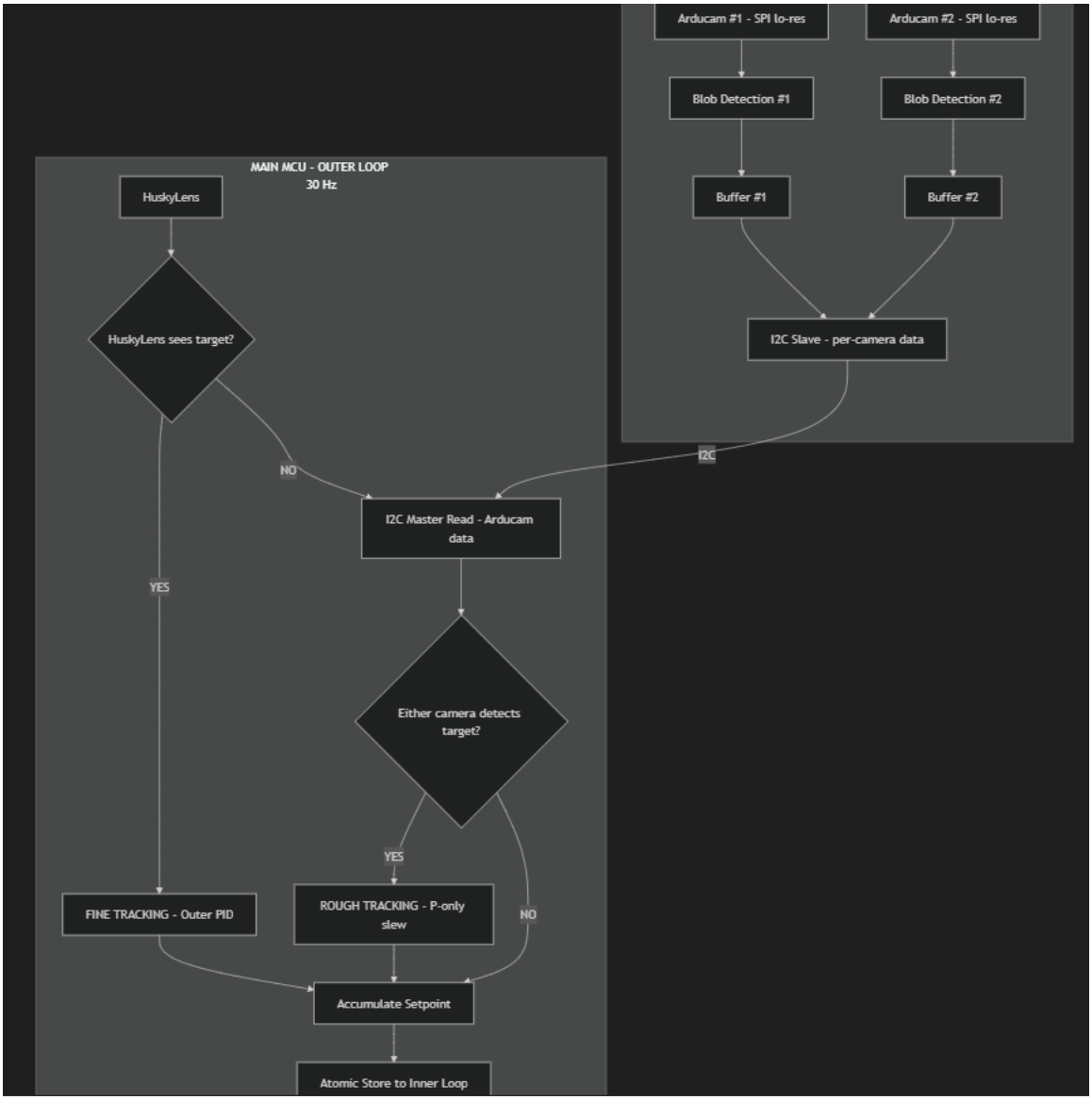


Figure 16 - Primary MCU Inner Loop Flowchart

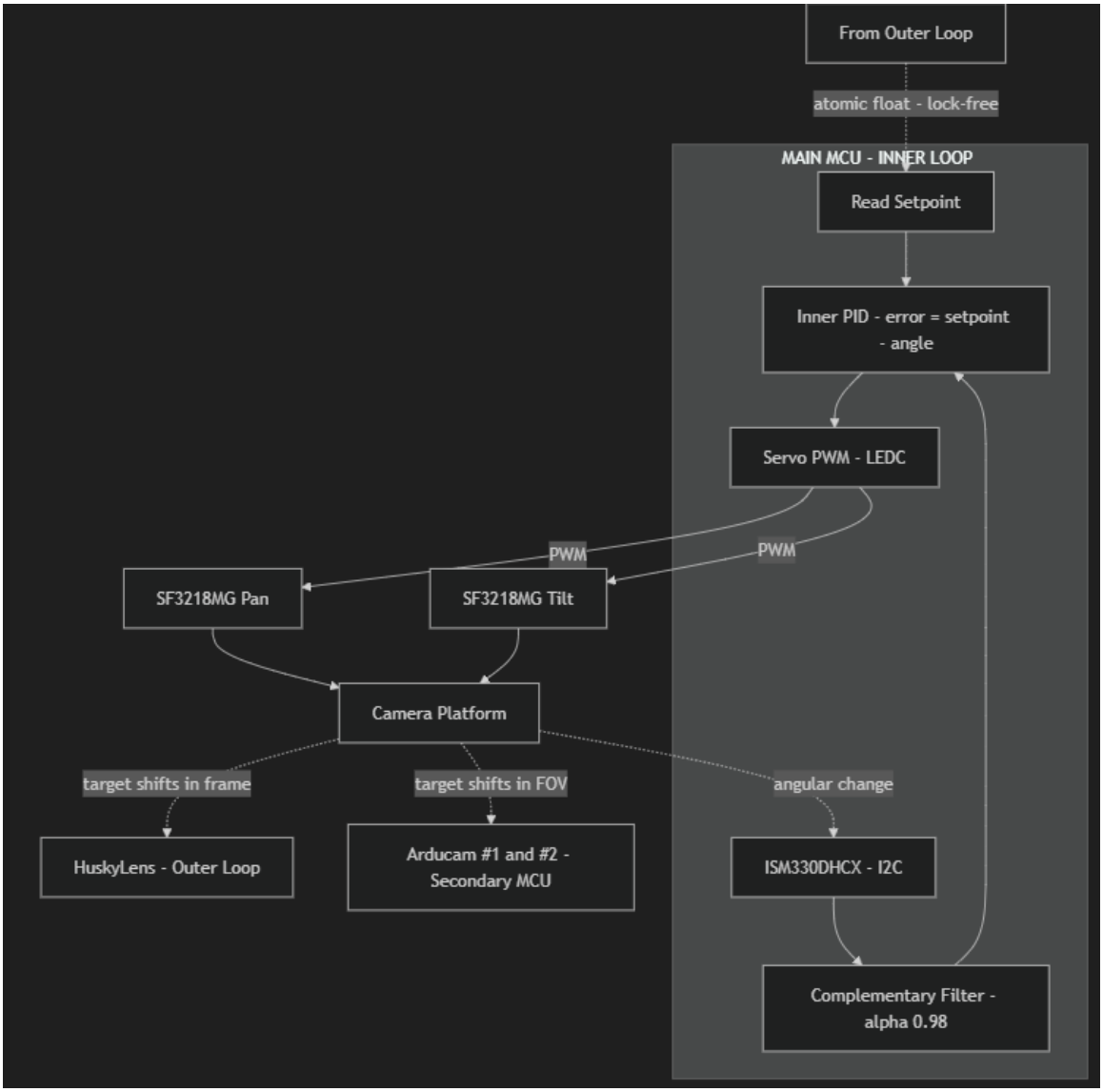
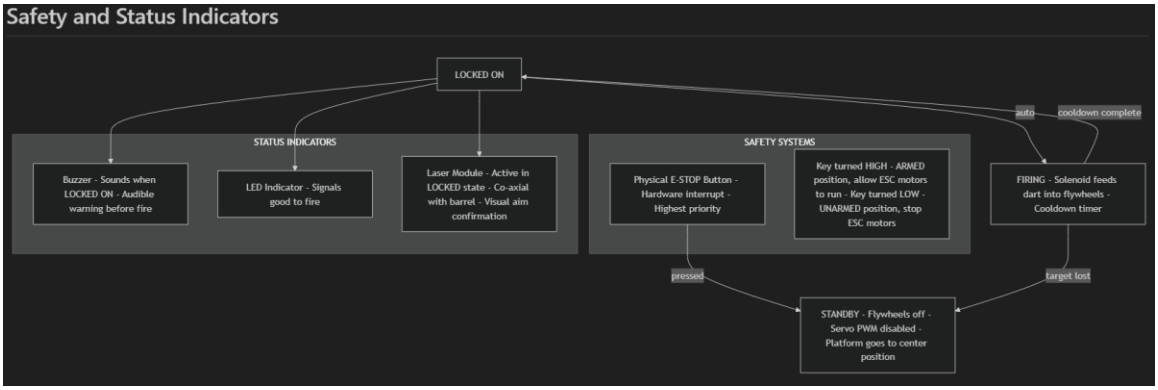


Figure 18 – Safety and Status Indicators



7.2 Secondary MCU

The secondary ESP32-S3 will primarily handle image processing from the two stationary cameras that connect via the SPI interface. The primary function of these cameras is to detect any incoming objects and localize (i.e. obtain coordinates) the targets for processing. These images are run through lightweight blob detection algorithm that is specifically designed for embedded systems, as MCUs don't typically have the memory capacity to support heavy machine learning models. Once the images are processed, the coordinate data will be transmitted to the primary MCU for fine tracking and actuation.

Goals:

1. Acquire image data from stationary cameras via SPI communication protocol
2. Scale image resolution or grayscale the image to conserve memory resources.
3. Perform blob detection and color recognition to identify potential targets.
4. Get the results (i.e. bounding box size, coordinates, confidence levels) and send them to the master MCU using SPI, UART, or over Wi-Fi using ESP-NOW depending on which connection gives the quickest data transmission rate with low latency
5. Capture images and process them in under 2 seconds per frame.

The secondary ESP32-S3 is solely dedicated to vision processing by running asynchronous (cannot capture images simultaneously) image capturing and inference tasks. Image data is captured efficiently using dynamic memory allocated (DMA) SPI data transfers. Since the image retrieval time and processing tasks are quite heavy, the MCU is not able to perform too many other tasks simultaneously, therefore it focuses primarily on processing images. It accomplishes this by running the model in a processing loop, while transmitting the data to the primary MCU only when new, valid detections are made.

This significantly reduces the workload of the master ESP32-S3 by offloading the heavy processing tasks to the secondary ESP32. This allows the master controller to focus on maintaining stable and precise motion control and feedback from multiple peripherals.

7.3 - Inter-MCU Communication

To communicate between the master and secondary ESP32s, there are several options for implementation such as communicating via SPI, UART, or ESP-NOW (wireless). Each option has its advantages and disadvantages and is chosen based on speed and latency. SPI is the preferred interface for its speed and low-latency capabilities and would have the primary ESP32 as the master and the secondary ESP32 as the slave device. The data packets sent between MCUs are structured in a frame format to maintain integrity and synchronizes them:

Field	Description
Header Bytes	Start of frame

camNumber	Which camera sent image (1, 2)
imageWidth	Width of image in pixels (320)
imageHeight	Height of image in pixels (320)
size	Number of bounding boxes found
minX	Targets leftmost X coordinate
minY	Targets leftmost Y coordinate
centerX	Targets center X coordinate (Real world space)
centerY	Targets center Y coordinate (Real world space)
width	Width of bounding box
height	Height of bounding box
size	Size of bounding box
distance	Estimated distance to object in inches
depth	Targets depth coordinate (Real world space)
isFriend	True if object detected is friend, false if enemy
Header Bytes	End of frame

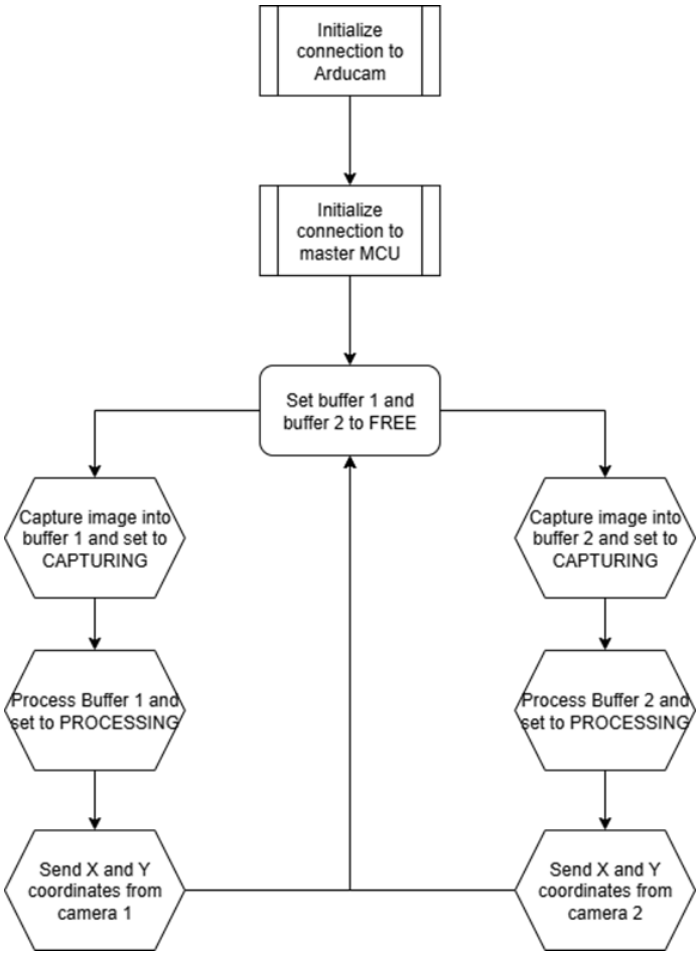
The main ESP32 will continuously poll the interface for new data incoming from the secondary MCU. If a valid detection is received, then they are implemented into the loop to track the target. However, if no detection is received before a timeout, then the system will search or go idle until a data transmission is available to be received. This design is reliable and has low overhead when transmitting data between MCUs, allowing for minimized latency.

7.4 Stationary Camera Sensor Software

The stationary camera sensors are used to actively scan the surroundings for potential threats. The cameras connect to the ESP32 via SPI pins, which gives the software full control over the operation of these cameras. After initializing both cameras, the MCU requests an image from a camera. Once it receives this image, the image is processed and outputs an array of bounding box data structures. These data structuress are then processed through different math functions to convert into real world coordinates. The

data is then sent over serial communication to the main ESP32, which will handle the data on its own. After sending the processed image over, the system resets and requests an image from the second camera. The process repeats again, only stopping when the system is shut down or when the huskylens takes over. The system relies on strict timing, making sure that only one camera is accessed at a time. By following a pipelined architecture, we can avoid idling the CPU while it waits on IO by processing an image while the second camera begins taking its image. To ensure real-time performance, there will need to be efficient timing control.

Figure 16 - Camera Flow



7.4.1 Camera Interface Module

The software for the camera interface handles configuration, initialization, and communication. To establish a connection to the cameras, we use the specific ESP32 SPI pins. We can then utilize ESP-IDFs SPI library to easily define what pins will be used for SCLK, MISO, MOSI, and CS. After writing a byte to the cameras test register, in our case 0x00, we can verify the connection by reading the register and confirming it contains the sent value.

To configure and initialize the cameras, a custom library was built that uses SPI to send specific bytes to specific registers. These registers control standard camera configuration options such as exposure, saturation, and sharpness, but also constrain other registers like image output type (RGB, JPEG, YUV), resolution, and image size.

After the connection has been established and the camera is ready to take a photo, the software will take a photo, then transfer the raw bytes into one large buffer. Because the image is 320x320, the size of the buffer is only 307KB, well within the available 32 MB of PSRAM. This buffer is fed into an algorithm that goes pixel by pixel, looking for either red or blue. Once a red or blue pixel is found, it will use flood fill to search the neighboring pixels to see how deep the color goes. If enough pixels of the acceptable color are found, a bounding box is generated. Another array stores the coordinates of visited pixels, effectively stopping the same pixel from being checked multiple times.

7.4.2 Image Processing Module

The image processing module performs threshold-based color blob detection. The input to module will be one image buffer as well as meta data about the image (image width and height). The image will be looped over going pixel by pixel, comparing it to a predefined threshold for what constitutes red and blue. Then, using flood fill, we will go over every neighboring pixel and check to see if it is above that threshold. If it is, then we can recursively flood that pixel. Once there are no more pixels above the threshold, the bounding box data structure is made by taking the top left coordinates and bottom right coordinates. With this, the size of the box can be found to be determined. Using the size of the box, we can filter out smaller boxes that are mostly noise. We can also use the height of the box to get an idea of the distance to the object because the object will be of a known size. This will be important later on.

It is important to note that the bounding boxes will be relative to the image taken. Because of this, we need to convert those coordinates to be relative to the camera itself and then convert those coordinates to be relative to a global center. To do this, take the physical width and divide it by the image width in pixels, then do the same with the height, and this will give you the scale of mm/pixel.

$$\text{Scalex} = \text{Physical Width} / \text{Image Width in Pixels}$$

$$Scaley = Physical\ Height / Image\ Height\ in\ Pixels$$

Then convert this to X and Y coordinates by multiplying the X and Y of the bounding box by the scale. This will be relative to the center of the image, rather than the top left corner.

$$Xcam = (x- width / 2) * Scalex$$
$$Ycam = (y – height / 2) * Scaley$$

We can then get a global X and Y by using a 2D Rigid Body transformation.

$$Xglobal = Xcam * cos(theta) – Ycam * sin(theta) + Xcam\_origin$$
$$Yglobal = Xcam * sin(theta) – Ycam * cos(theta) + Ycam\_origin$$

These coordinates will be sent over to the main MCU, which will be able to process and use it.

7.4.3 Communication Module

To send the data between the two MCUs, a standard protocol must be followed. Because only the bounding box data is sent, only 44 bytes of raw data will be sent per bounding box. To start the transmission, a 3-byte header will be sent that lets the main MCU know that it is receiving packets. The next 16 bytes will be used to send data about the image taken and number of bounding boxes. Then the 44 bytes of bounding box data are sent over per bounding box, followed by 2 bytes for a final header to notify the reciever that transmission is complete.

Because of how often transmissions occur, time will not be wasted to ensure that the receiver gets the correct data. If the receiver gets corrupt data or there is data loss, then it will simply wait for the next frame to come through, as the next frame will include any updated data. It determines this by checking if each value received is abnormally large or abnormally small.

7.4.4 System Scheduler

To ensure that the system flows smoothly, images will only be taken when requested. To maximize CPU usage, the ESP32 will queue image requests, and will schedule tasks to handle each queue. Once an image is taken, it will be added to a processing queue. The task can then take another photo will another task processes the first image. This architecture works because the time it takes to transfer the image from the camera is limited by the 8 MHz SPI clock. This pipeline efficiently uses the ESP32. Once the image is processed, it sends the data over uart and begins processing the next frame, if available. This effectively cuts out 70ms of wasted time between image requests and images being sent.

7.4.5 Software Flow

The first step is to initialize the entire system. This will be handled by the camera interface module, and it will set up the system. Once all communications between devices are established, the system will go into its active state. Inside the active state, there will be one main control loop that loops indefinitely. This loop will check for image requests over UART. When a request comes in, it will be added to a command queue. The command queue will then be read and handled. If the command sent is to capture an image, the request will be added to a separate queue which will take an image and add the buffer to another queue. This queue is read from and then processed. Finally, the updated data is sent to the other ESP32.

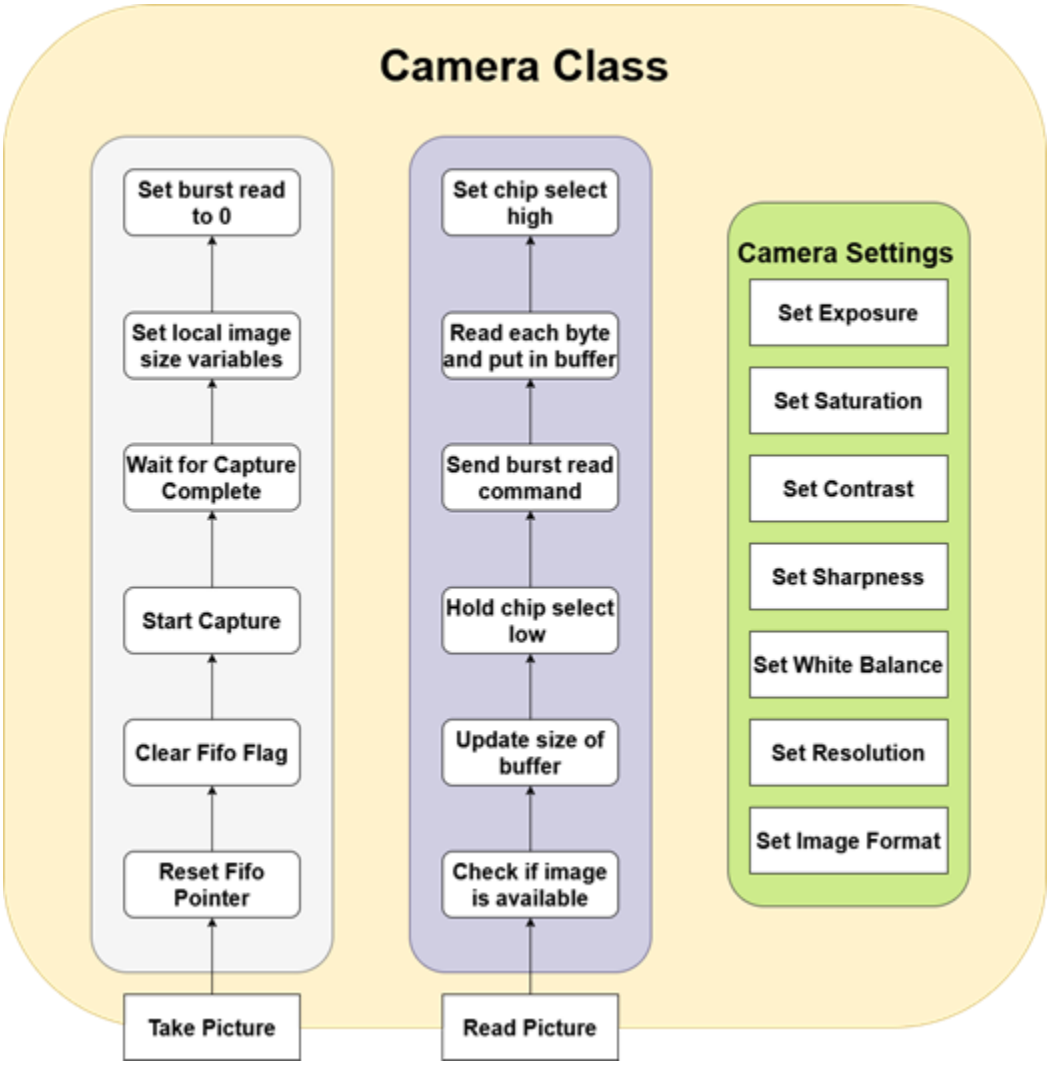
This flow will not be broken and cannot go in different paths. Additionally, because of the nature of the pipeline, it is not guaranteed that the images will be sent from different cameras in every frame. This is expected, and the main MCU won't care where the coordinates came from.

7.4.6 Camera Class

The camera class is used to interface with each stationary camera. It is fully independent of any other class or module. The camera can easily be configured using the different setters that can change its exposure, saturation, sharpness, contrast, white balance, resolution, and image format. When creating a camera object, the only required input is the chip select pin the SPI camera is attached to. Apart from the configuration functions, the two public functions that will be called is takePicture and readPicture. The takePicture function will set internal camera registers to the appropriate values and wait until an image has been captured. There will be no input or output to this function as everything is stored on the camera hardware. The readPicture function will take in a buffer and its size. It will then write the exact number of bytes to the buffer and increment the internal buffer read pointer. The output from this function will be the number of bytes written to the buffer, as this number can sometimes be less than the size of the buffer.

This class is fully independent from the rest of the code, meaning if requirements change for the system, this class will be unaffected. The sole purpose of this class is to take an image and read the image. It is the responsibility of different classes to determine what to do with that image and how to manipulate the output data.

Figure 17 – Block diagram of Camera Class



7.4.7 Blob Detection Algorithm

The blob detection algorithm operates by scanning the image pixel by pixel (nested loops: for each y, for each x) while maintaining a visited array and a vector of detected blobs. As each pixel is processed, the algorithm first checks if it has already been visited. If not, it applies a color threshold function (isRed or isBlue) to determine whether the pixel belongs to a target color region.

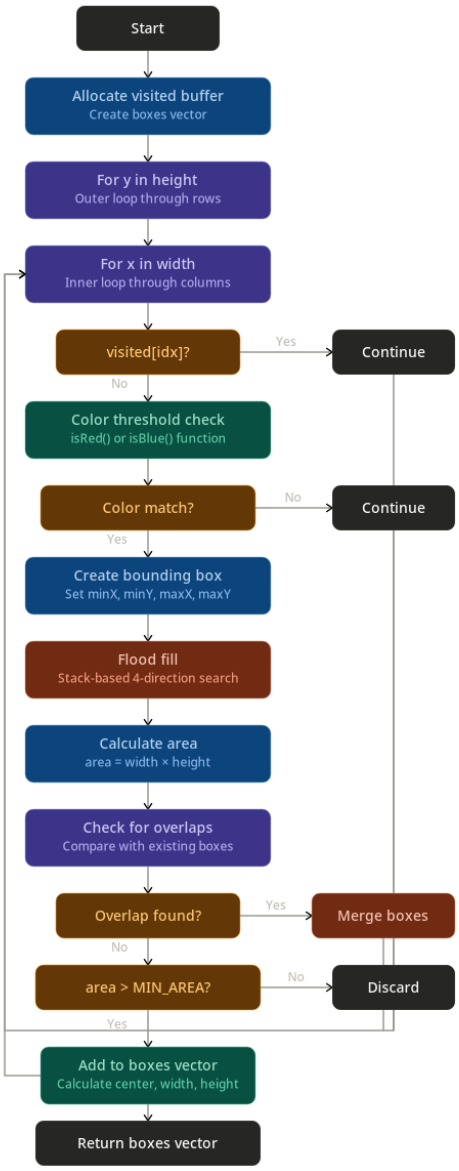
The color threshold functions check RGB values against specific criteria. For red detection, the pixel's red channel must be significantly higher than both green and blue channels using linear thresholds. For blue detection, the blue channel must dominate. These functions return true or false based on whether the pixel matches the target color profile.

When a matching unvisited pixel is found, the algorithm creates a new temporary bounding box initialized with that pixel's coordinates. It then initiates a flood fill operation starting from this pixel. The flood fill uses a stack-based approach to explore all connected pixels of the same color in four directions (right, left, down, up). As it processes each neighboring pixel, it marks them as visited and expands the bounding box boundaries (minX, maxX, minY, maxY) to encompass all connected pixels. This continues until no more connected pixels of the target color remain.

After the flood fill completes, the algorithm calculates the area of the detected blob. It then checks if this new blob overlaps with any existing blobs of the same color using a delta-based overlap detection. If an overlap is found, the two bounding boxes are merged into one by taking the minimum and maximum coordinates, and the area is recalculated. If no overlap exists and the blob area exceeds the minimum threshold (MIN_AREA), the blob is added to the results vector.

This process continues for every pixel in the image. The visited array ensures each pixel is only processed once, preventing duplicate blob detection. The final output is a vector of bounding boxes, each containing position, dimensions, center point, size, color type, and friend/enemy classification.

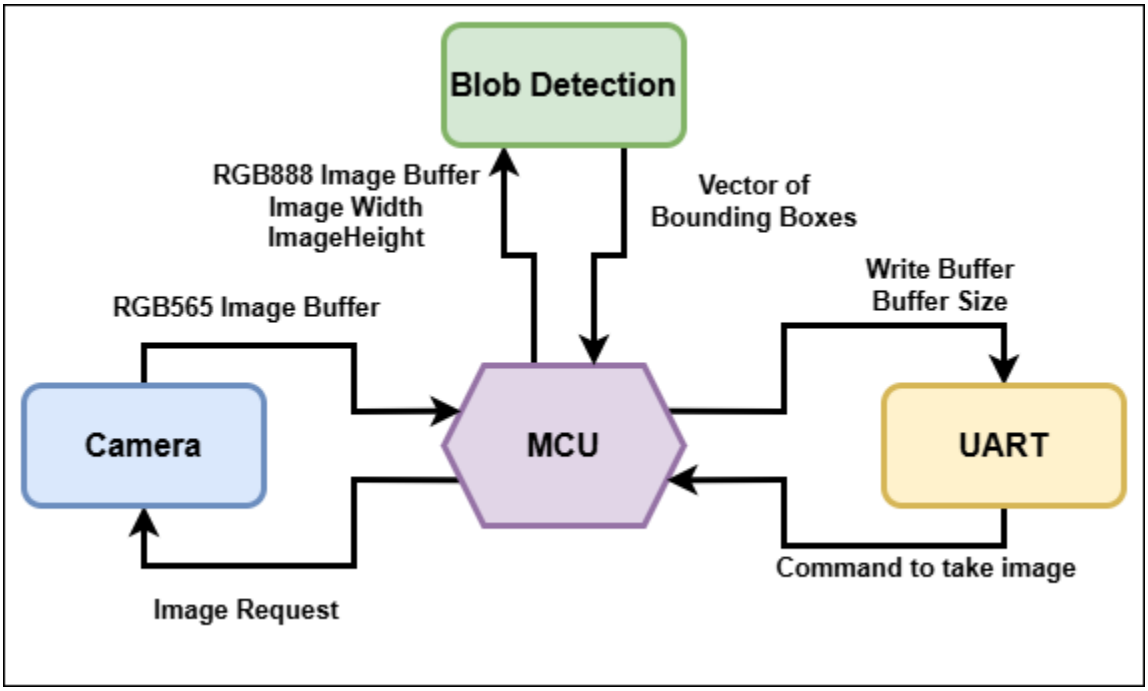
Figure 18 - Blob detection Algorithm



7.4.8 System Architecture

The whole system relies on the center MCU to perform the functions. Each module is standalone and has no dependency on any neighboring module. The MCU is the only component that is linked to all. This central hub will be responsible for sending requests to take a picture, read an image buffer, convert the image buffer to rgb888, then pass that image buffer into the blob detection algorithm, then finally output the bounding boxes through UART to any machine ready to receive them.

Figure 19 – Block diagram of System Architecture



7.5 - Pan-Tilt Motor Control Software

This system will include a 3D-printed pan-tilt mount that will house the HuskyLens AI Vision Sensor that performs fine tracking, and the laser diode that points at the target. Two servo motors will be used to control the pan-tilt mount. One motor control the horizontal movement (pan) and the other will control the vertical movement (tilt). The maximum range of the pan movement should be around 270 degrees, and the maximum range of the tilt movement should be 90 degrees. The servo motors we are using in this system are the Sunfounder SM3218MG which have an operating voltage range of about 4.8V-7.2V, an angle range of 0 to 270 degrees, and a torque of 20kg/cm with a speed (no load) of 0.18sec/60 degrees. These specifications give us more than enough leeway when designing the pan-tilt mount as it is able to handle heavier loads without affecting the speed, helping to meet the design requirements.

7.5.1 - Mechanical Design Considerations

The mechanical design of the pan-tilt mount is critical when building the system to ensure that the movement when tracking the target is accurate, stable, and repeatable. The structural rigidity, range of motion, and load distribution of the mount has to be taken into consideration since it houses the HuskyLens and laser diode which can affect the tracking ability and reliability of the system. These characteristics will directly affect the software responsible for controlling the mount/system. It has to account for the physical limitations, dynamic motion, and behavior when responding to certain situations.

An important aspect of the software to consider is mapping the angle range to software constraints. Although the pan (horizontal movement) supports up to 270 degrees and the

tilt (vertical movement) supports up to 90 degrees, the usable range may have to be reduced depending on the wiring, mechanical stops, or center of mass of the components (could be heavier on one side). The software for controlling motion has to identify and initialize safe boundaries, usually by setting angle limits, to safeguard the servos from exceeding its physical constraints. These limits are enforced at the firmware level to ignore commands that exceed the range or saturate them.

Another aspect to consider is the mechanical load and inertia of the components on the mount. Since the mount must carry sensors and wiring, the mechanical movement is affected and will show signs of overshoot, settling time, and oscillations. This can be mitigated by implementing a PID-control algorithm for smoothing and speed limiting. These help to prevent sudden jumps in position, decrease vibrations, and improve stability when tracking fast moving objects that are detected by the HuskyLens.

The software should account for the feedback characteristics of the mount. Since the mount itself does not provide positional feedback to the MCU, the system has to rely on internal modeling, timing assumptions, and feedback from the IMU and cameras. These are implemented through incremental movement tuning, loop timing, and PID parameters for predictable movements. It will be a linear control system, so if nonlinearities arise from the mechanical design (i.e. increased load at certain angles), the PID algorithm will be adjusted to ensure performance across the full range of motion maintains stable.

These design considerations can significantly impact the performance of the pan-tilt mount. Using the software designs above, we can mitigate the constraints of the mechanical components to control the pan-tilt mount reliably, stably, and ensure the fine-tracking of targets is accurate.

7.5.2 - Servo Motor Electrical Considerations

The electrical characteristics of the servo motors will have a direct impact on the software by determining how the control algorithms will be designed and implemented. The electrical system will be physically separate from the software, but the firmware will be programmed to operate within the electrical limits of the hardware for responsive and stable performance of the motors.

One responsibility of the software is to ensure proper PWM (Pulse-Width Modulation) signal generation with accurate timing. Each servo motor requires a 50 Hz PWM signal where each pulse width ranges from around $500-2500\mu s$ which depends on the desired angle. The primary MCU will generate PWM signals using hardware timers to prevent jittering. Blocks in the code, interrupts, or delays in tasks could cause jittering in pulse width which leads to instability in the servo motors. That is why the PWM signal generation will be controlled through dedicated timer peripherals, and the software will be designed to avoid long blocking routines in motor control tasks.

Another factor to consider is electrical isolation of components to prevent electrical noise and brownout awareness. This is handled in hardware, but the software has to be robust enough to handle transient behavior. An example would be if the servo draws high current during quick acceleration, there could be dips in voltage which could cause logic-level glitches or instability in the MCU. The firmware will implement brownout-detection and reset-recovery routines; therefore, the system will have a fresh start in case

of power disturbance. The PID algorithm will also account for such noise by implementing a filter (Complementary or Kalmann; e.g., smoothing HuskyLens position error to avoid unnecessary movement of servo motor) on the input signals.

To prevent current spikes or mechanical stress caused by something like instantaneous positioning, the software implements a command slew-rate limiter to keep the PID controller from commanding sudden large angle changes. It will help to keep the current draw predictable and decrease the chance of electrical interference with other components. Since the servo motors share the system ground and rails with other components, the MCU firmware implements task prioritization and management system for interrupts to make sure that time-critical PWM tasks are run reliably alongside other high and low priority tasks such as communication, image-processing, etc. Taking all these into account will create a deterministic system that ensures reliable and stable behavior in the servo motors.

7.5.3 - Pan-Tilt Software Architecture

The software architecture for the pan-tilt system is built to achieve and maintain stable, accurate, and smooth movement of the HuskyLens, laser diode, and solenoid while tracking targets. To achieve this, the subsystem integrates sensor feedback (from HuskyLens and IMU), motion control algorithms, servo commands, and communication with the MCU. The characteristics of this architecture include smooth actuation, deterministic timing, and the ability to robustly handle sudden changes.

1. High-Level Control Flow
    a. Target Acquisition Layer (High-Level)
        i. Receives coordinates of target from HuskyLens (i.e. Bounding box - (x, y)).
        ii. Convert visual offsets into angular tracking errors (horizontal and vertical movements).
        iii. Pass normalized error signal to the main MCU to run through motion tracking and PID algorithms.
    b. Fine Tracking Layer (Mid-Level)
        i. Runs the PID control algorithm for the pan-tilt
        ii. Compute the servo command increments required for minimal position errors.
        iii. Include measures for better stability.
    c. Servo Command Layer (Low-Level)
        i. Convert MCU outputs into PWM signals.
        ii. Add measures to handle any vibrations or jerks in the motors.
        iii. Keep the commands within the bounds of the motor’s mechanical limits.

    This hierarchy allows for a modular and maintainable system where the tracking algorithms can be tuned and modified without significantly affecting the servo motors. It also allows multiple cameras to be installed while minimizing the changes to the upper layer of the system.

2. Control Loop Timing

The pan-tilt system has timing constraints to maintain stable tracking:

a. Control Loop Frequency: 50 – 100 Hz
    a. Has to be fast enough for smooth compensation, but slow enough for updates to the servo.
b. Servo update rate should be around 50 Hz or higher.
c. HuskyLens update rate should be around 20 – 30 Hz
    i. The system will use the most recent available frame, and the control loops will interpolate between frames.
d. Reaction Time Requirements: < 2 seconds to complete tracking, the controller should be tuned to allow the pan-tilt mount to re-center the target within 150-500ms (depends on displacement/offset)

The control loop will implement timer interrupts to maintain constant updates. This will allow us to avoid jittering and maintain deterministic behavior while running multiple peripherals simultaneously.

3. Error Processing

The software will convert positional errors into servo angles by:

a. Calculate pixel offset (where t is the target coordinate)

$$e_x = x_t - x_{center},\ e_y = y_t - y_{center}$$

b. Normalize the offset
    i. Map the pixel errors on a scale based on the camera's resolution.
c. Convert positional errors to angular errors using the camera's field of view

$$\theta_{horizontal} = e_x \cdot \frac{FOV_x}{2},\ \theta_{vertical} = e_y \cdot \frac{FOV_y}{2}$$

d. Feed the above errors into the PID algorithm for smoothening, and each axis (x, y) will have its own PID loop.
e. Servo PWM: The controller outputs will be mapped to pulse widths with a range of about 500 – 2500 microseconds.

4. PID Control Algorithm Implementation

The system will use two independent PID loops, one for the pan and one for the tilt. The loops will include:

a. Proportional (P) term that reduces immediate error.
b. Integral (I) term that removes the steady state offset.
c. Derivative (D) term to damp oscillations.
d. Anti-windup limits stopping integral runaway.
e. Output clamping that stops motors from exceeding mechanical limits.
f. Rate limits to ensure smooth transitions at high speeds.

The gains for each term will have to be tuned on a trial-and-error basis based on the load inertia of pan-tilt mount as well as the abilities of the Sunfounder SM3218MG servo motors.

5. Safety Measures and Fault Handling
    a. Bound Checking: Prevent servo motors from exceeding their mechanical limits. The pan servo should operate between 0 and 270 degrees, while the tilt operates between 0 and 90 degrees.
    b. Stall Detection: In case of failed updates to servo motors after repeating commands.
    c. Loss-of-target Mode: The mount will return to a neutral position if the HuskyLens loses sight of the target.
    d. Low-Voltage Protection: In case of voltage drops under heavy load.

Figure 20 – Pan-Tilt System Flowchart

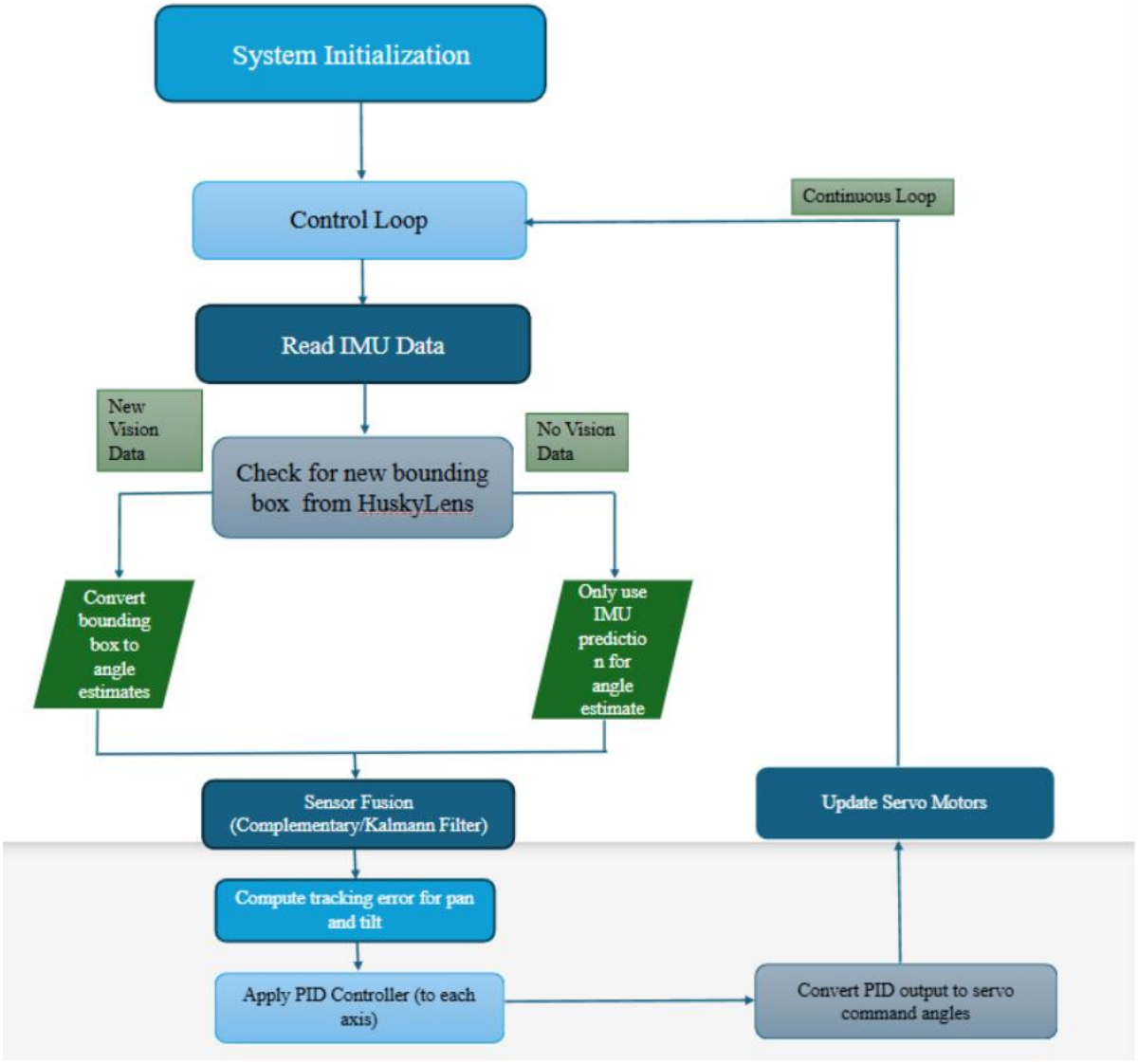
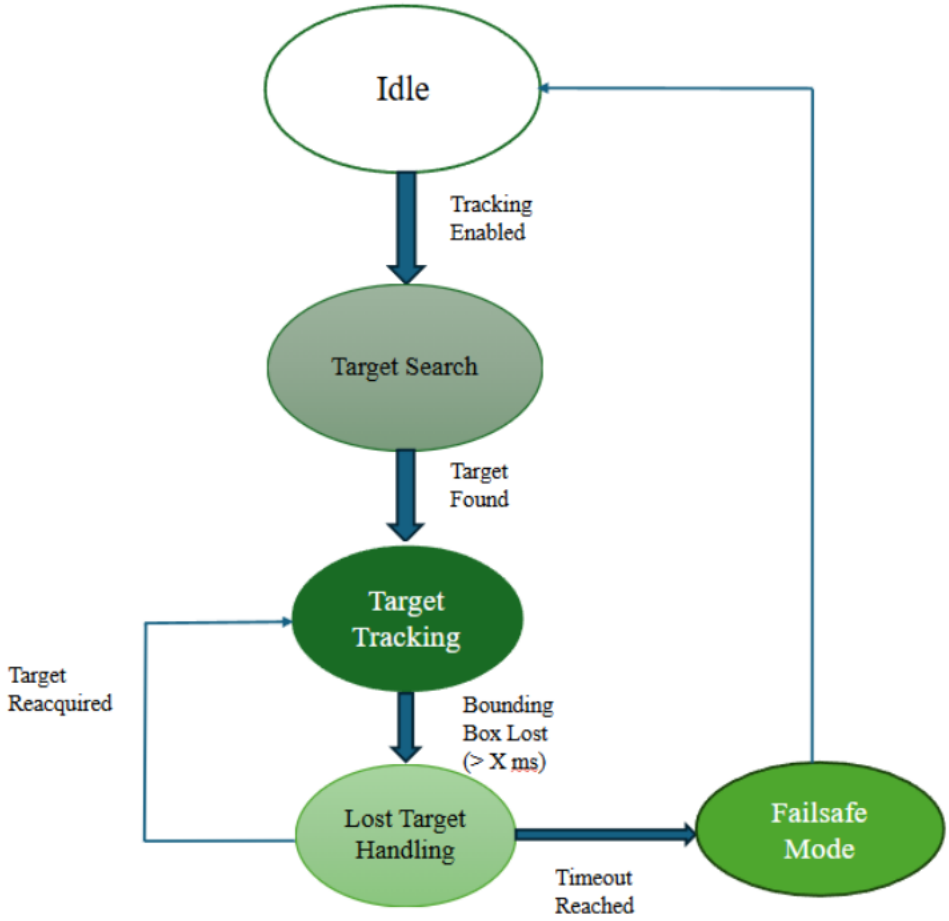


Figure 21 – Pan-Tilt Tracking State Machine



Chapter 8 - System Fabrication/Prototype Construction

This section will take a look at the layout of how we wire our PCB board for the turret, access box and regulator PCBs. In each section you will notice we want to have ample space between parts. The reason for this is in case we need to hardwire new traces through anything; we want to have room to achieve this if needed. Further refinement of the layout will be needed, but the idea will remain the same in later stages. Each picture below will show a certain section composed from the block diagram and schematics from chapter 6. The only major difference is we moved

the Power LED over to the through holes that will take in our 12V. This way we know the board has routing power and from there we can start turning on the regulator switches and motor switches so our system will be fully turned on.

Figure 22 - Switch section

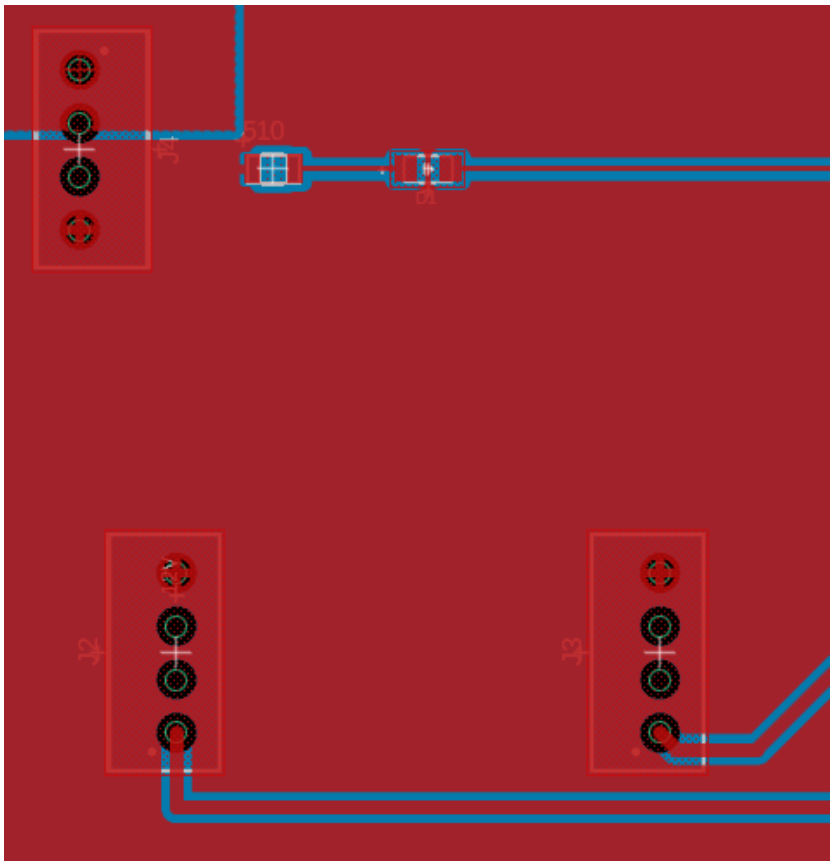


Figure 23 – Regulator Pin holes

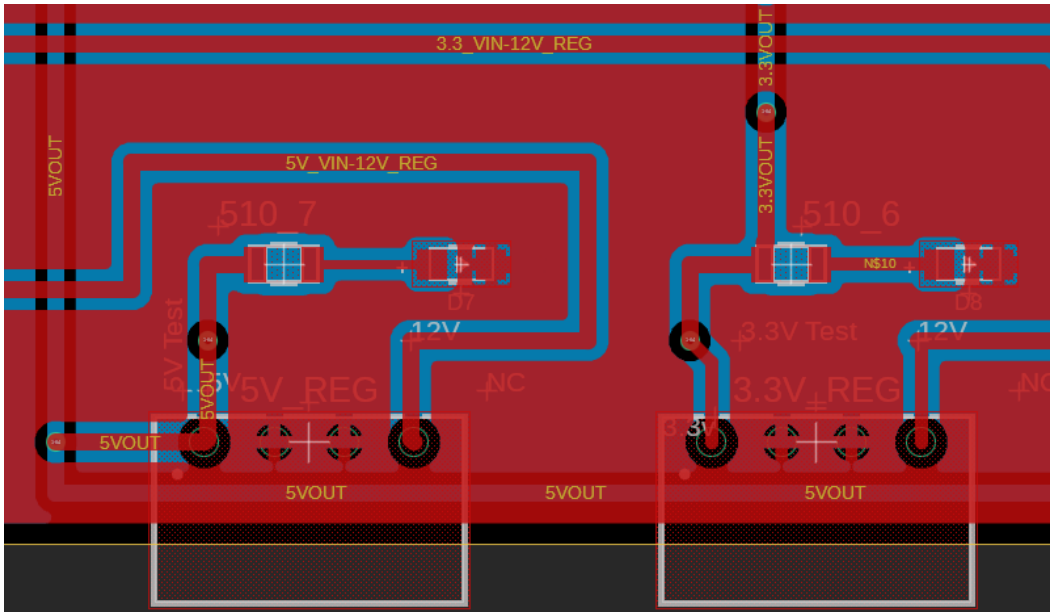


Figure 24 - Buzzer Circuit

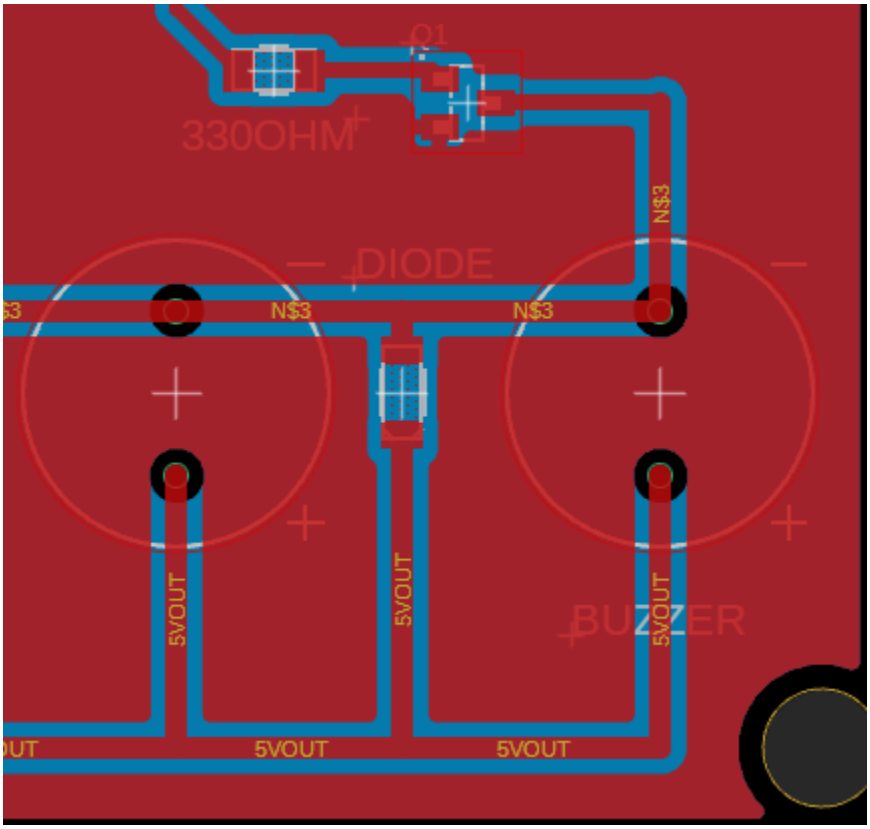


Figure 25 - Led Status section

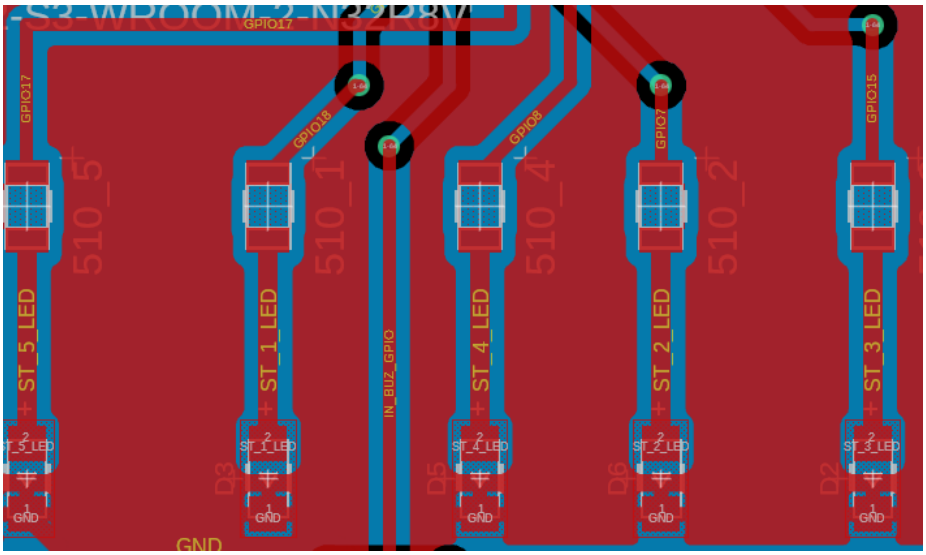


Figure 26 – MCU circuit and pinout for cameras

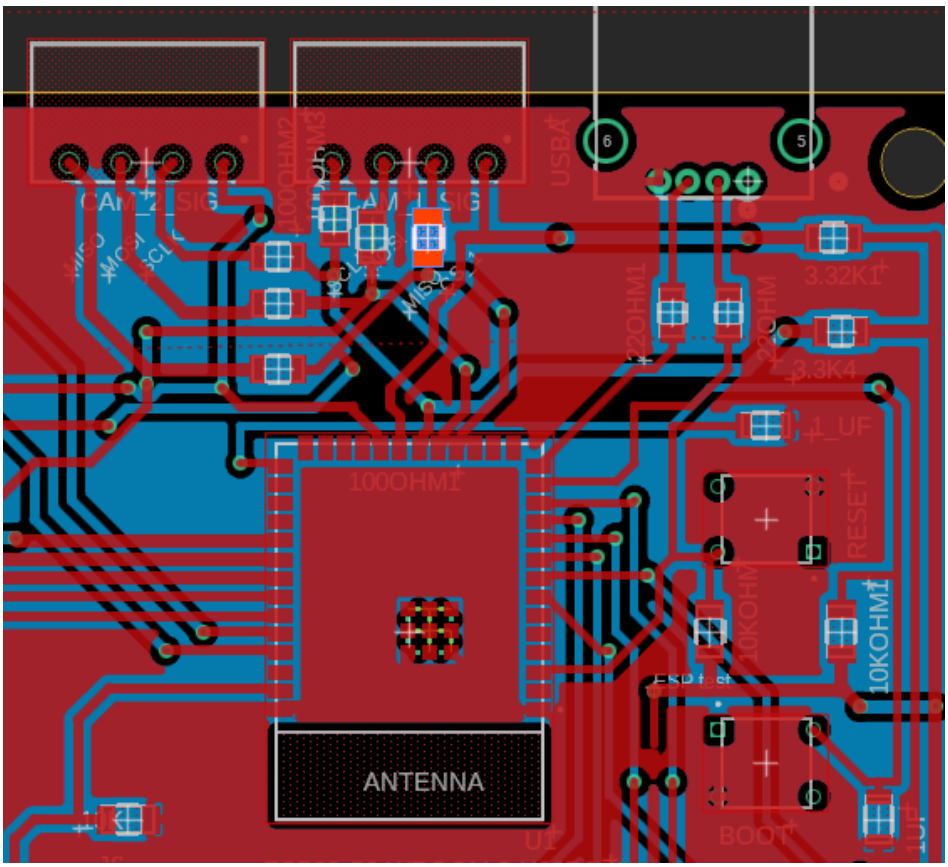
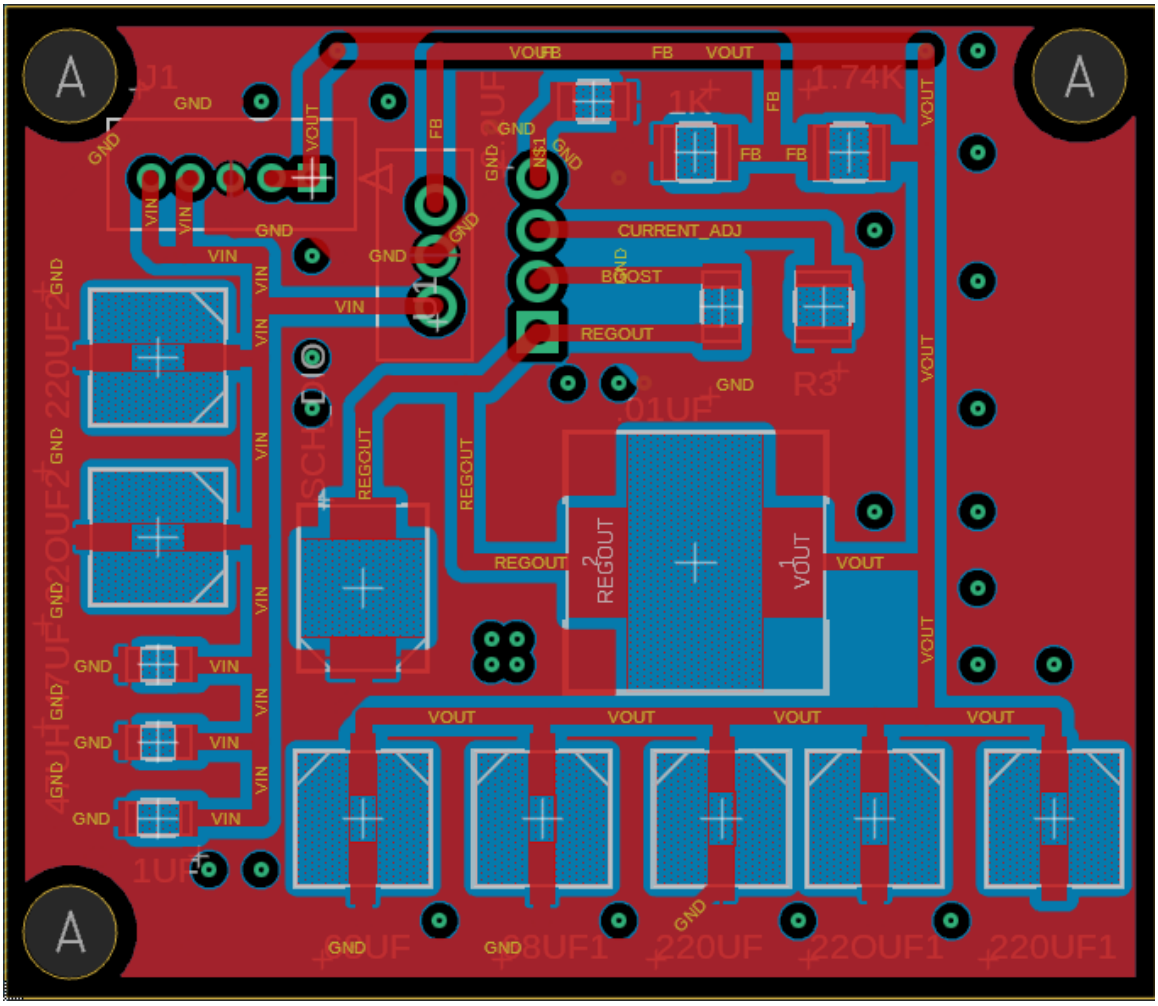


Figure 27 Regulator



8.1 Turret MCU Board Layout

The Turret MCU board is the main controller for the turret. It has the ESP32-S3 module and handles all the local I/O including flywheel ESCs, pan/tilt servos, HuskyLens camera, laser control, IMU, solenoid actuation, and other sensors.

Programming the ESP32-S3 without taking the board off every time was important, so a USB-A connection was included for ease of use, allowing the turret PCB to connect directly to a computer for programming and debugging. This made testing a lot easier since the board could stay mounted.

The board has dedicated headers for everything: two ESC PWM outputs, pan and tilt servos, the HuskyLens camera, laser enable, and the main harness connection. These are grouped by what they do and labeled so assembly is easier and wiring mistakes are less likely. Power and ground pins are arranged the same way across connectors, and the main harness connector is placed near the edge of the board so the cable can route cleanly into the turret structure.

Noise was a problem that had to be dealt with since the ESP32-S3, I2C bus, and camera interface all need clean signals. Ground pours and stitching vias were used to improve return paths and reduce interference. Higher power wires were kept separate from signal wiring to avoid coupling noise between the power and control sections. Pull-up resistors were also required on the SDA and SCL lines so the I2C bus would have a cleaner and more reliable signal.

For debugging purposes, test pads and vias were added. This way signals can be probed or traces can be cut without desoldering parts, which is really helpful when something breaks or needs to be isolated from the rest of the circuit. Mounting holes on the board line up with 3D-printed standoffs in the turret, so the MCU board is physically supported and the connectors are not carrying mechanical load when the turret moves.

This board also has its own regulators. There's a single 12V input coming from the access box that powers the PCB and generates the local supply rails onboard. These include two 7V regulators, one 5V regulator, and one 3.3V regulator for the different subsystems. TVS protection was added to help protect against voltage transients, and there's also regulator status monitoring built in.

The ESCs aren't powered through the PCB traces directly. ESC power is routed from the power supply through relays that are mounted on the 3D-printed turret. The PCB controls these relays using GPIO pins driving MOSFETs, which switch the relay coils at logic level. The ESP32 also generates the ESC PWM control signals. The emergency stop cuts power to the relays, which shuts down the ESCs. A stepped-down version of that relay line is sent back to the controller so the ESP32 can detect whether relay power is active.

The solenoid that actuates the darts is connected directly to this PCB. It draws high current, so it's driven by an IRLZ44Z MOSFET, and the power path was done with wide copper pours instead of small traces so the board could handle the load safely.

Figure – 27.1 Shows the Solenoid/Relay Part of the PCB

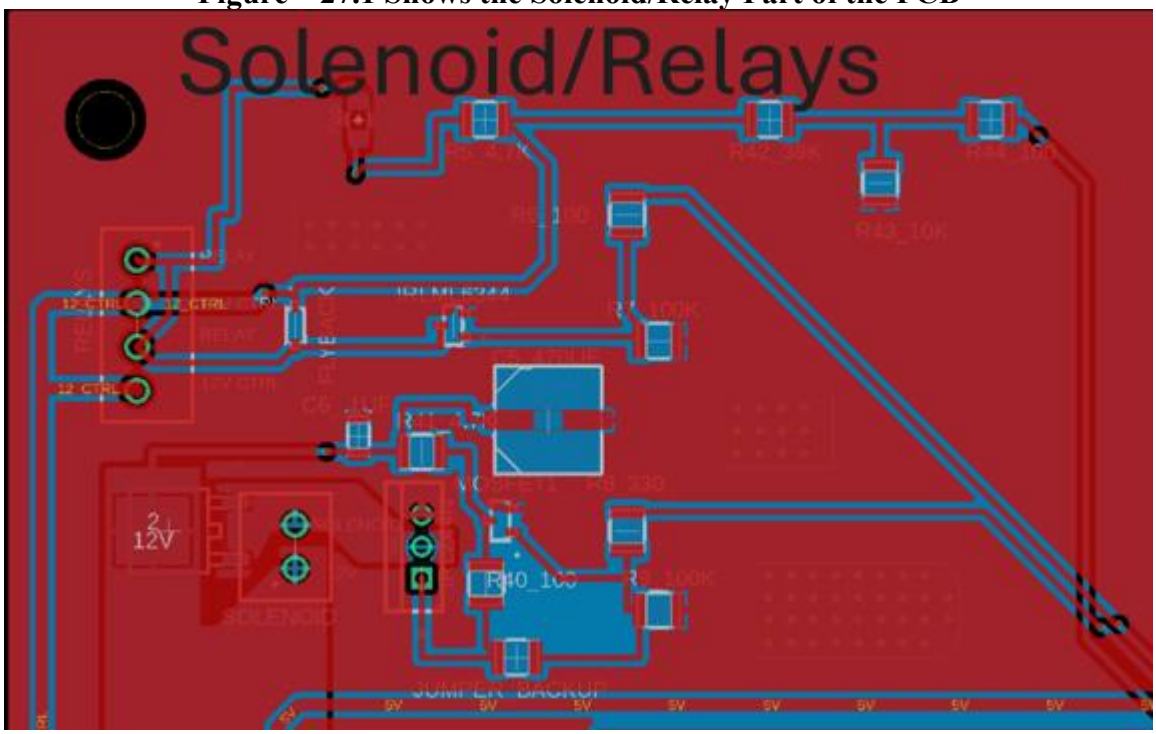


Figure – 27.2 Shows the peripherals part of the PCB

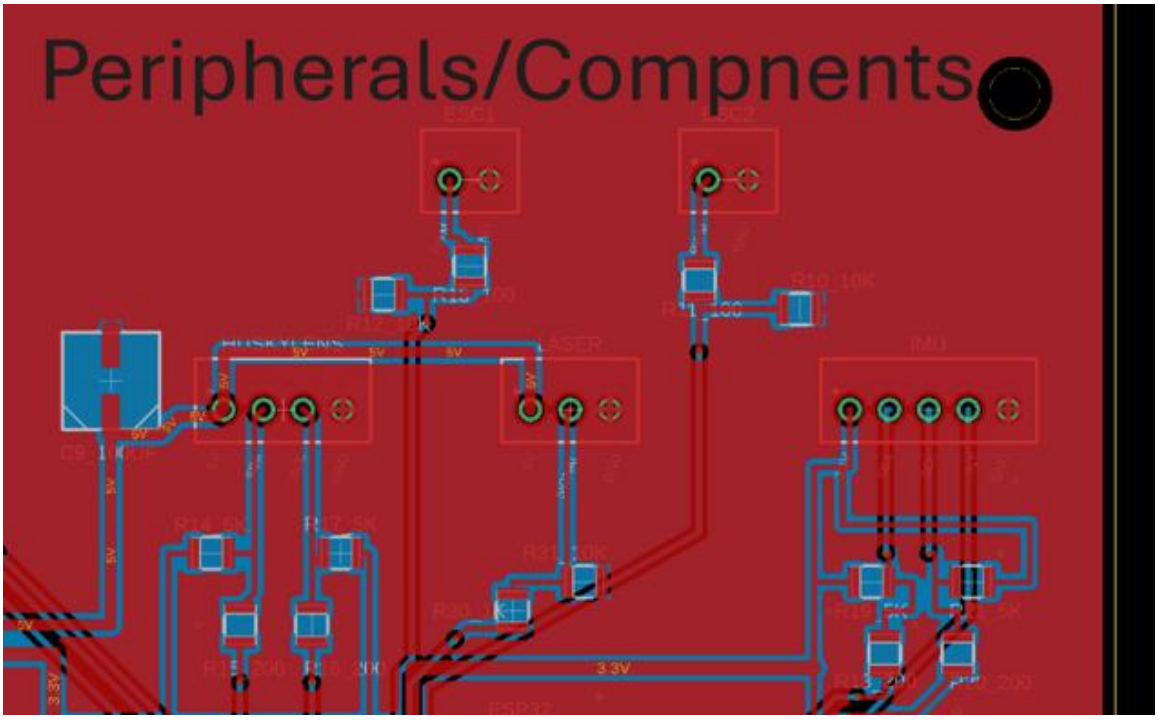
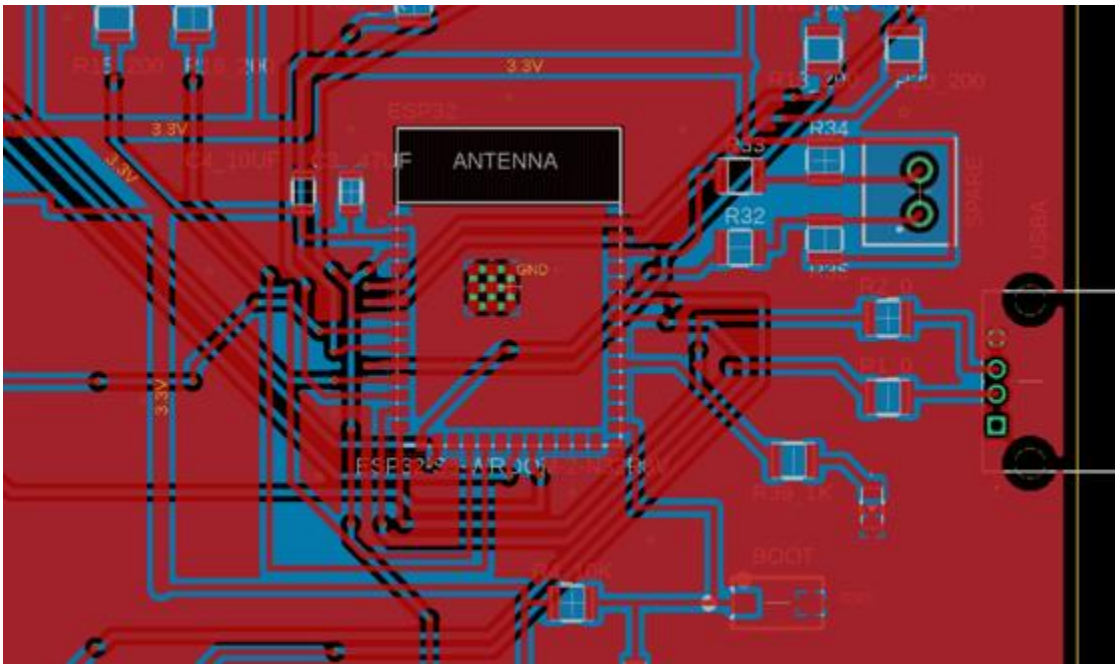


Figure – 27.3 Shows the pinout of the ESP32



Ch 9 – Testing

9.1 - Overview

This section will focus on our key test of sub-systems that will be integrated together to get our fully functional prototype. We will be testing how the regulator can operate under normal loads such as our buzzer as well as seeing how it can handle the servos on the pan tilt. We need to see if any issues arise since this will be our power for the system. Lastly for our testing section we will have our blob detection on multiple Cameras. Our only source of knowledge for knowing whether an enemy is being detected is the use of our cameras with blob detection, and we want to make sure this design is at the forefront of operation and important as without it, the entire system could not work.

9.2 - Regulator powering load

First is the regulator which will power the pan tilt on our mini demo video. We will have the voltage output by 7V in our video, but we have also tested and got a 3.3V and 5V output. The buzzer circuit was made before as well to make sure it works with different

values and parts as the parts that are suggested in the buzzer datasheet are not manufactured anymore. All regulators provide a stable output with little to no variance even underload. All can handle the extra transient and stable response for our system which is why we designed them with the capacitor choices we did and the extra head room as well.

9.3 Relay Circuit Breadboard Testing

The relay circuit was first tested on a breadboard using the same core components planned for the Relay Power PCB being a 40 A automotive SPST relay, a logic-level N-MOSFET on the low side of the coil, a flyback diode across the coil, a coil on indicator LED with series resistor, and an E-Stop/enable switch in series with the coil supply. The circuit was powered from a 12 V bench supply and controlled with a 3.3 V logic signal.

The objectives of this test were to verify that a 3.3 V logic output can reliably drive the MOSFET and turn on the relay coil, confirm that the relay cleanly switches a 12 V load on and off without chatter, check that the flyback diode protects the driver when the relay turns off, verify that the coil on LED works as expected, and show the E-Stop idea by breaking coil power and confirming that the relay cannot turn on when that path is open.

The 12 V bench supply fed the relay common and coil. The MOSFET source was tied to ground, the drain connected to the coil return, and the gate driven from a 3.3 V digital output. A 1N4007-type diode was wired directly across the relay coil for flyback protection. The indicator LED and 4.7 kΩ resistor were connected from 12 V to the MOSFET drain so that the LED is on only when coil current is flowing. The E-Stop function was just a simple switch in series with the coil's 12 V feed. A small dummy load on the relay contacts confirmed that the switched side was opening and closing correctly, high-current ESCs were not connected at this stage.

With the E-Stop switch closed, driving the MOSFET gate high caused the relay to turn on with a solid audible click and the coil on LED to light. Driving the gate low dropped the relay out immediately and turned the LED off. The relay did not chatter and stayed on as long as the gate stayed high, showing that the MOSFET and gate network can supply the required coil current from a 3.3 V control signal. Opening the E-Stop switch while the relay was on caused the relay to drop out even though the gate signal was still high. With the switch left open the relay could not turn on at all no matter what the gate state was. This showed that interrupting the coil supply is an effective hardware-level emergency stop, if the coil path is open firmware cannot keep the relay on.

During repeated on/off cycling there were no obvious problems on the 12 V rail and no signs of stress or overheating on the MOSFET or diode. The circuit continued to work normally after many cycles which means the flyback diode is clamping the inductive energy as intended and that the chosen components are right for the relay coil current.

9.4 Servo and Pan/Tilt Prototype Testing

In addition to the relay circuit, the pan and tilt servos were tested together with the 3D-printed pan/tilt assembly and the prototype 7 V regulator that will later become the Servo Power board. The two servos were mounted into the 3D-printed pan/tilt brackets in their intended final positions. Instead of using the final turret software, a simple demo program was written that reads a two-axis joystick and maps the X and Y positions to the pan and tilt servo PWM signals. This allowed manual control of the turret head to move it around its workspace during the demonstration. The servos were powered from the 7 V regulator prototype which was built on a breadboard and powered from a 12 V source. The regulator followed the planned buck converter, but with jumper wires and breadboard connections instead of a PCB with short, wide traces and proper decoupling. By moving the joystick both servos could be driven independently or together, including fast motion at the same time to stress the power rail.

Mechanically, the 3D-printed pan/tilt assembly worked as expected. Both servos were able to move the structure smoothly through the intended travel without obvious binding or gear skipping. The joystick control made it easy to check the full range of motion in both axes and to spot any mechanical interference or cable snagging. Electrically, the test showed that the prototype 7 V regulator output sagged a lot under combined servo load. With light or single-servo motion the output stayed close to the intended 7 V. However, when both servos moved at the same time, especially during rapid joystick movements, the measured output dropped down to around 4.5 V. The servos continued to move but this level of sag is not acceptable for a final design and would reduce torque and responsiveness.

Because the regulator and wiring were assembled on a solderless breadboard with long jumper wires and loose contacts, the conclusion was that the sag was mostly caused by the breadboard setup rather than the regulator design itself. The additional series resistance and inductance from the breadboard and wiring increase the voltage drop during the high current pulses that occur when both servos accelerate together.

This test did two things. It confirmed that the mechanical pan/tilt design and servo selection are working, the servos can move the assembled structure through its required range under joystick control. It also showed a clear requirement for the final 7 V rail must use low-impedance routing, solid ground, and enough output capacitance so that the voltage does not sag from 7 V down toward around 4.5 V when both servos move at the same time. More detailed measurements of angle accuracy, response time, and final regulator performance will be done once the Servo Power and Relay Power PCBs are fabricated and installed, but this prototype test was enough to confirm the overall design and to show that the breadboard power setup is the limiting factor in the current test not the core design.

9.5 - Stationary Camera Testing

Testing the stationary cameras was broken into multiple parts, each verifying that an important milestone was reached. The different milestones started with verifying

connection to the camera, followed by verifying that an image can be taken. After taking an image, the next step was to verify that the image can be sent over UART. Then, verify that the blob detection algorithm worked, and port all code onto MCU. Finally, establish a UART connection between two MCUs and transfer the bounding box data between them. The milestones were broken up this way because each step required that the previous be completed and working well. This type of testing also helped build the system from the ground up, making the overall system more modular and minimizing time wasted on debugging.

The first milestone is to establish a connection to the cameras. This is important to get settled first because it eliminates the doubt that later if there is an issue, it is because the connection to the cameras is faulty. To test this, much online documentation was read to determine which pins on esp32 were SPI, how to begin an SPI transaction using Arduino code, and which registers to write to and read from. The pins for SPI for our specific module were 11, 12, 13, and any chip select pin. Two functions were then written to write to a register and to read from a register. Finally, the register to read and write to was provided in the documentation for the Arducam cameras, register 0x00. This is a dummy register made specifically to verify reading and writing capabilities. After all the setup was finalized, data was sent to the register, then read from the register to verify the value had changed. Once tested and proven, the camera can start taking pictures.

The next milestone was to take a picture with the camera. Much more work was needed to verify that a picture could be taken, and it started by reading more documentation on how Arducam takes its pictures. Once the code was written to set up the camera to take the picture, such as clearing the image buffer and resetting any flags, the picture was taken and printed out to a console. The code was written using Arduino IDE, so the serial monitor was used to verify that the image was taken. Arducam supports multiple image formats, one of them including JPEG. By testing the image format in JPEG, it was possible to look at the first two bytes and last two bytes. After tweaking the order of functions, delays, and adding necessary flag resets, finally the output started with 0xFFD8 and ended with 0xFFD9, signaling that the image was successfully taken.

After taking the image, the next milestone was to send it over UART. This test step is not necessary for the functions of the system, but rather it is included to make debugging easier and faster. It is impossible to look at the binary from a JPEG and construct an image from it without using software, so a way to display the images needed to be created. This helped us to further tweak any image taking and reading functions. A python script was written to read from UART and write the output to a file. Once the script and camera code was tweaked to send in a reasonable format, a jpeg file was created and could be used to verify the image was taken and it was not corrupted or filled with garbage. This step exposed many issues with the code for taking pictures. The first few rounds of output images showed that the camera was taking pictures, but the image appeared dark with barely any detail, just enough to see outlines if looking very closely. After reaching this step in testing and seeing how corrupt each image was, alternative ways to take images were explored. Eventually, an example code from Arducam was found, and when using the Arducam Mega application with the code, an image was taken

and displayed on a testing laptop. This proved that the camera worked and was not broken, rather the code was missing some key steps. After digging through the example code, many of the functions were ported over to our code base. After tweaking the code, an output image was read and perfectly showed the room that the camera was pointing at, confirming that the image taking functions and reading functions worked as expected.

The next milestone was to verify that the blob detection algorithm worked. This was done separate from the MCU code, as it does not have any dependencies with the camera or UART connection. C++ code was written that reads a ppm file, converts it to an array, then outputs an array of bounding boxes as well as a ppm file with the bounding boxes on the image. This image file was purely for testing and will not be needed in final product. The idea of testing this was that the image will be taken from the camera and stored as an array of RGB values, which can be input into the code through a function. The code worked well in testing and bounding boxes were successfully added to the images, confirming that the blob detection algorithm worked.

With the bounding box code working, it was time to test the code on the board rather than off the board. Importing the code was as easy as just copying and pasting the code into the project directory. To test it, the images were sent over UART as RGB values and then added to a file on a laptop to be viewed. The bounding box data was also sent over and displayed on a console to verify that they make sense and send correctly. Testing these exposed problems with the detection of the color red. The first issue appeared when taking images in different areas. The different lightings meant that the camera settings for exposure, saturation, contrast, and white balance needed to be updated depending on the environment. The other alteration made was how to define what was red and what is not. At first, a pixel was red if its R value was greater than 160, and the B and G value was less than 80. This worked for very bright reds but failed when lighting was dim or too bright. The next idea was to check if the red channel was greater than the blue and green channel combined. This worked well with darker images but failed when the image was brighter and included pink. There were also issues where darker colors such as browns were falsely identified as red. Finally, a line was drawn on the color graph, where anything greater than the line was deemed red, and nothing less than the line was not red. To work better in dark lighting, a piecewise function was used to skew the darker red to be accepted. The final RGB ranges are as follows: 255 130 160 → 50 0 0. Using the slope formula to find R with respect to G and with respect to B, R must be greater than 1.58 * G + 50, and R must be greater than 1.28 * B + 50. For the darker colors where 50 < R < 100, the R channel must be greater than the G or B channel by 50. The testing in this step verified that these values gave consistent results the matched what was needed.

Figure 28 – RGB Classification Graph for Red

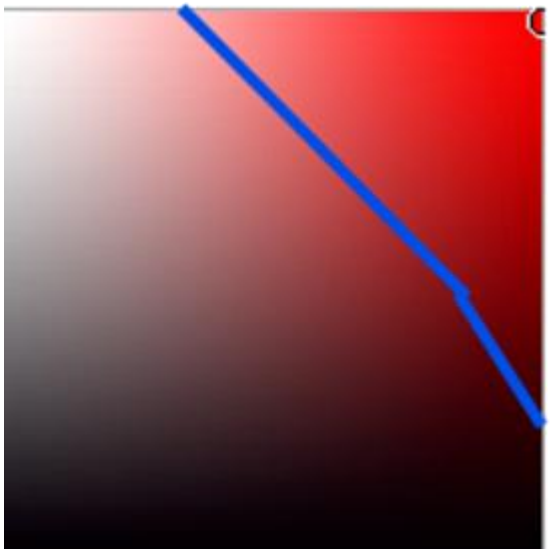
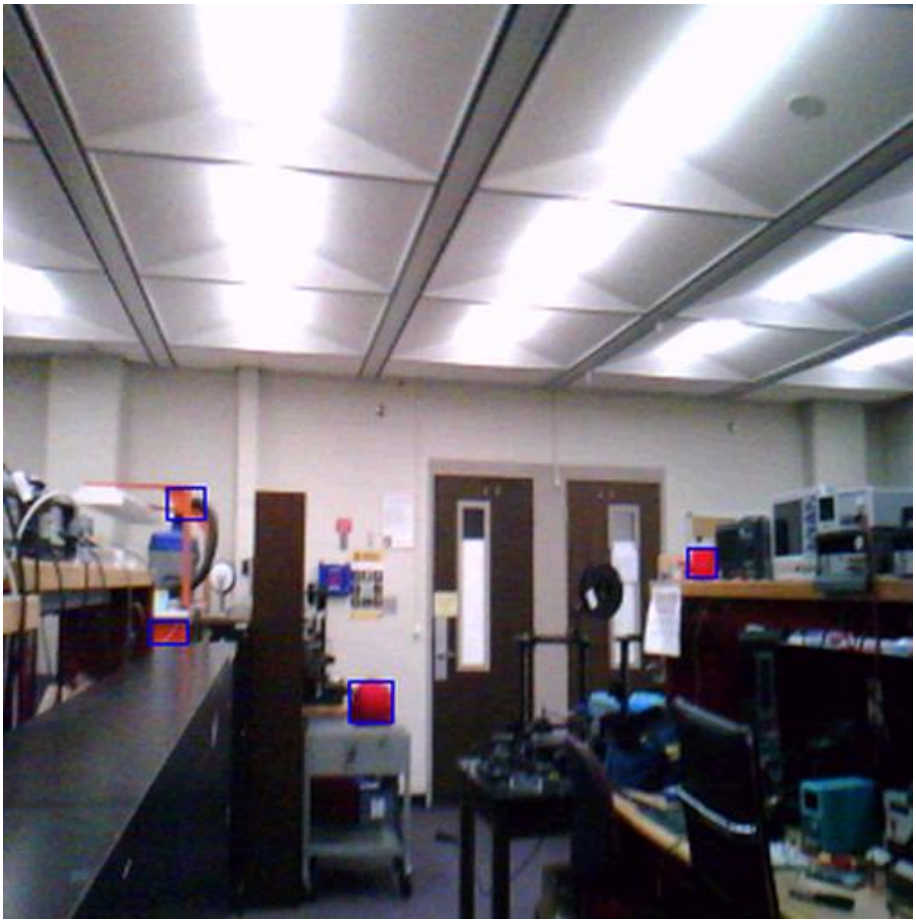


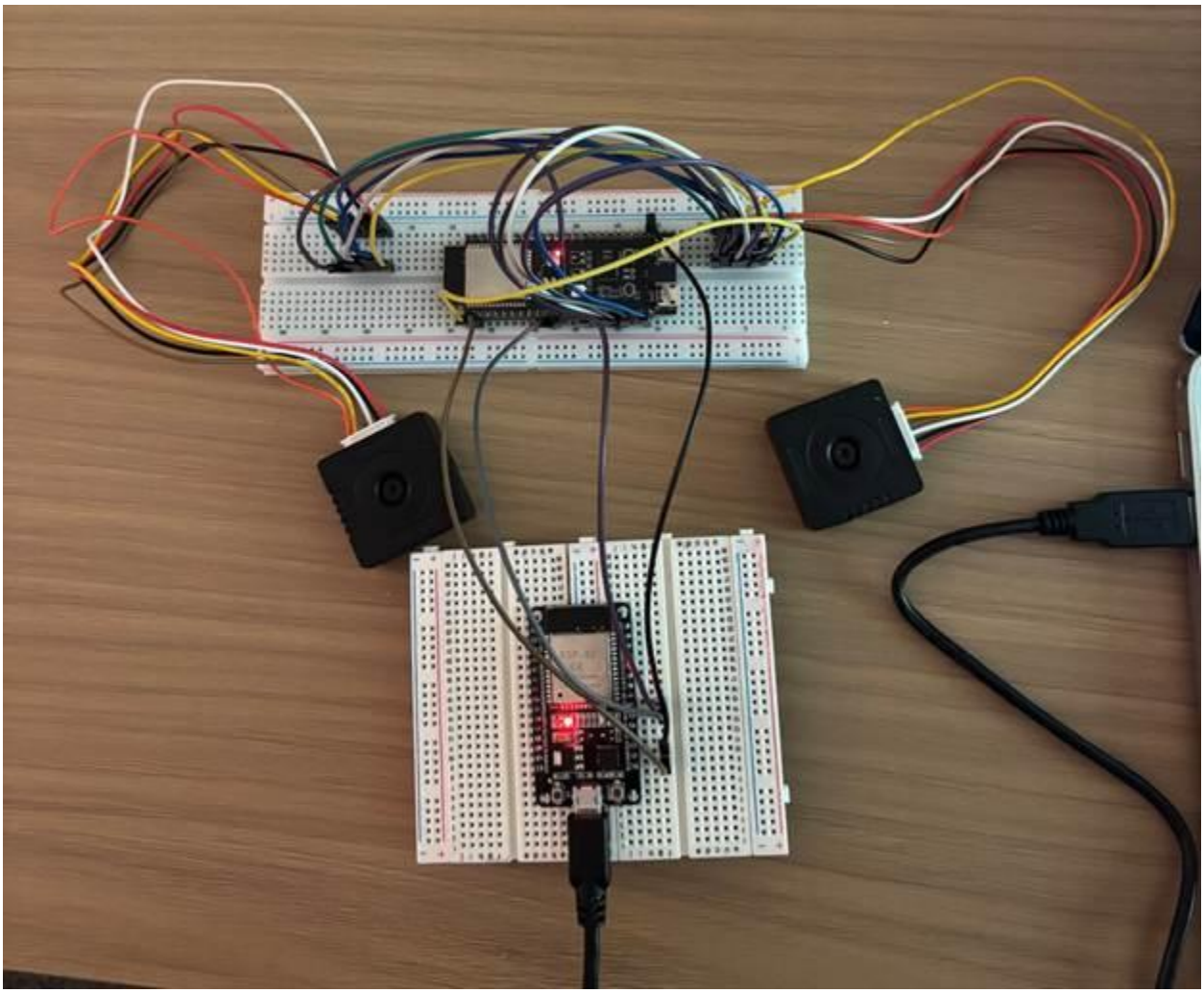
Figure 29 – Blob Detection Testing



Finally, once the color detection has been tweaked and the images are successfully being sent over UART to a laptop, the last milestone was to add a second MCU to send

commands to the MCU connected to the cameras. This mimicked how our system works. To test this system, the MCU connected to the laptop acts as a middleman sending over commands from the laptop to the other ESP32. This test verified that the implementation would work on the esp32 and the blob detection module is completed and will not need to be tweaked anymore.

Figure 30 – Final Board Layout with Cameras



9.6 - HuskyLens AI Vision Sensor Testing

The initial testing of the HuskyLens AI Vision Sensor was primarily focused on verifying its ability to perform tasks like object detection and output useful and usable position data for a fine-tracking system. The HuskyLens is able to integrate multiple machine learning models that are built-in and include face detection, object recognition, and color tracking which allows for image processing tasks to be offloaded from the primary ESP32 microcontroller. By reducing the computational burden by a significant amount on the ESP32, it enables the microcontroller to dedicate extra processing time to real-time motor control, servo motor actuation, and later on the closed-loop PID controller required for fine tracking.

Due to limited hardware availability at home, no breadboard was available during early testing. This was solved with a direct connection between the HuskyLens and ESP32 development board using male-to-female jumper wires. These connections include ground, regulated input voltage, SCLK (Serial Clock), and SDA (Serial Data Line), which are standard for I2C interfacing. Extra attention was paid to verify that both devices shared a common reference ground to maintain the integrity of I2C. Despite the improvised wiring setup, the communication between the devices was stable, and no I2C errors occurred.

The first machine learning model that was tested was color recognition. The HuskyLens interface (2-inch screen on the back) allows the user to select a model with the onboard dial, aiming the crosshair at a specific target, and pressing the learn button to capture a sample of the feature. Once the sensor learns the object, it will assign an ID to the object and is able to continuously output the results in the form of bounding-box coordinates (width, height, x, y) over the I2C interface. The values of these results were read and printed through the ESP32's serial monitor (in Arduino IDE) to verify that the HuskyLens protocol is working correctly.

The bounding box positional data was the main interest when testing, as these are the inputs that are fed into the PID control algorithm that is responsible for the stable motion of the pan-tilt mount in dynamic environments. Early tests validated the stream of coordinate data were updated consistently with low latency. These results show the promising potential for real-time feedback control. The HuskyLens was also able to successfully demonstrate its ability to learn and track multiple IDs simultaneously, which will be useful in situations where multiple objects need identification in the future.

Testing for accuracy revealed several performance characteristics that will need to be further investigated in Senior Design 2. Under controlled lighting settings indoors, the color-based detection performed relatively reliably, and the reported coordinates were stable with minimal jittering. However, the test results also revealed decreased confidence in object recognition and bounding box stability when tested under poor light conditions, shadows, reflections, or overexposure. These insights are consistent with the limitations of simple built-in models and reinforce the need for additional calibration and behavior testing under several environmental conditions.

Future testing will include evaluating the HuskyLens under several different lighting settings, color temperatures, and distances. We will also be testing the HuskyLens's ability to retain object IDs during sessions to be consistent. Additional testing will also be required to measure the frame rate latency and noise characteristics of the output coordinate data to ensure the sensor provides the necessary temporal resolution for smooth fine tracking with a PID controller.

Overall, the conclusion we have drawn from early testing of the HuskyLens is that it is a capable and lightweight sensing solution for this project. With its built-in machine learning models, we are able to simplify the software architecture and give the primary ESP32 the ability to focus on high priority real-time operations. Although additional environmental testing and tuning are required, the results produced from preliminary

testing confirm that the HuskyLens is able to integrate successfully into the fine-tracking subsystem planned for Senior Design 2.

In Senior Design 2, development shifted from the original HuskyLens to the HuskyLens 2, which features the Kendryte K230 processor with a dedicated 6 TOPS neural processing unit capable of running custom-trained models. The color recognition model from Senior Design 1 was replaced with a custom YOLOv8n model trained specifically on red balloons, addressing the false positive problem identified in earlier testing. The model was trained on a 775-image dataset using Google Colab, exported to ONNX format, and converted to a K230-compatible uint8 quantized format for on-device deployment. The final model achieved a mAP@0.5 of 0.95–0.96, substantially exceeding the 0.80 minimum threshold set for deployment, and demonstrated strong discrimination between red balloons and other red objects such as clothing and vehicles.

Integration of the HuskyLens 2 into the full fine-tracking system revealed a key hardware reliability issue: the I2C bus connecting the HuskyLens 2 to the ESP32 became intermittently unresponsive when the brushless flywheel motors were active, likely due to electromagnetic interference from motor wiring routed near the I2C cable. This was mitigated in firmware with a software recovery routine that performs a 9-clock GPIO bus recovery sequence followed by driver reinstallation when the sensor goes silent for more than 1000 ms. Additionally, the I2C clock speed was reduced from 400 kHz to 100 kHz, which improved bus stability under motor noise. These measures resolved most dropout occurrences, though the recovery procedure adds approximately 1–1.5 seconds of latency to the acquisition-to-lock cycle when triggered.

End-to-end system testing measured the time from target acquisition to confirmed lock-on, which requires the HuskyLens 2 bounding box center to fall within 1.75° of the aim point for two consecutive frames at 30 Hz. Under nominal conditions without I2C recovery events, acquisition-to-lock times ranged from approximately 1.9 to 3.0 seconds depending on the initial angular offset between the turret and the target. The outer PD control loop, running at 30 Hz, incrementally drives the pan and tilt servo setpoints toward the detected balloon center each frame. Larger initial offsets result in longer convergence times due to the incremental nature of the control loop. Once lock-on is confirmed, the brushless motors ramp to operating speed and the solenoid fires, with total acquisition-to-fire times typically in the range of 2–3.5 seconds across test sessions.

Accuracy testing showed consistent hits on a standard balloon target at ranges of 2–5 meters following aim offset calibration. Physical misalignment between the HuskyLens 2 optical axis and the dart barrel trajectory was compensated through an empirically tuned pan and tilt aim offset constants in firmware. Results confirmed that the HuskyLens 2

custom YOLO model, combined with the dual-loop PID control architecture, provides sufficient detection reliability and positional accuracy to support the fine-tracking and firing requirements of the S.T.A.T. system. Remaining areas for improvement include reducing acquisition-to-lock time under I2C recovery events and further characterizing detection performance at extended ranges beyond 5 meters.

9.7 - Servo Motor Actuation

The testing of the servo motors was done to validate our choice of using the Sunfounder SM3218MG servos for the pan-tilt mount as it should be able to reliably meet the turret system's mechanical and timing requirements. Due to hardware limitations, I only had access to one servo motor, and testing was done using a lab bench setup, allowing the servos to be driven under controlled conditions while monitoring its accuracy, response times, and repeatability.

The first part of testing focused primarily on basic actuation. The servo motor was connected to the ESP32 development board by using a breadboard, jumper wires to connect to the ESP32 ground and servo ground, also connected a wire to a GPIO pin, and the servo motor was powered by an external power supply to get a constant voltage source of around 7V. This was because the ESP32 was not able to provide sufficient power to the servo motor and will be powered by a regulator in the final product. The servo motor was driven using standard PWM signals to confirm correct rotational behavior across expected ranges. The pan axis should span approximately 270 degrees, and the tilt axis should be around 90 degrees. The movements were increased in increments of around 10-20 degrees to ensure there were no jitters or dead zones present. The testing also verified that the servo was able to respond reliably at a stable operating voltage and was able to achieve consistent positioning without stalling.

The second stage of testing evaluated the motor's dynamic responsiveness. The system simulated rapid directional changes similar to what will take place during the fine tracking of objects. They were commanded to sweep from one extreme position to another while the time it takes to complete the full cycle was measured. Although there was no load, the speed it took to complete the task helped to confirm the actuators are able to meet the system's real-time tracking demands.

Preliminary work on the closed-loop control architecture was performed during this phase. While the complete PID algorithm could not be implemented due to limited hardware availability, early testing of the servo motors helped to verify that the chosen motors were able to produce predictable and repeatable movements that are able to be mapped to future control-loop inputs. The observations pertaining to their positional consistency during repeated movements served as confirmation of the motor's deterministic behavior that is required to establish an effective error-based feedback system once the pan-tilt mount is fully assembled and sensor inputs (i.e. IMU data) become available.

Overall, the testing of the servo motors helped to demonstrate their ability to meet speed, performance, and torque requirements that will be needed for stable and precise operation of the pan-tilt mount. The results revealed that once the motors are properly integrated

into the final subsystem with the HuskyLens positional data and IMU (gyroscope and accelerometer) feedback, they will be capable of supporting a robust PID controlled fine-tracking system.

9.8 Turret Accuracy testing

Below is the table for Turret Accuracy in a few different conditions. We have used a stationary target for all of testing. A factor that changed how accurately our system was, is the lighting of the room and the shadow cast on the target.

In Figure 43 showing six and seven feet were tested outside during midmorning/afternoon ranging between 11am to about 1 pm. Our test location was a fellow groupmates backyard porch which had a covering over top and was screened in. This room had plenty of light but was dim in certain areas. For example, if we had the turret move to the left it would be in a shaded space where the house is. If we pointed to the right of the turret, it was in more light, where more of the backyard was. This caused different shadows or no shadows to occur, affecting the angle and distance sent by the Sentry Cameras which affects HuskeyLens lock on time. When the HuskeyLens took over, it did not always see the target as well. If it was too dark it may not be able to get a solid read on our target causing it to go back to center and wait for more data to be sent by the Arudcams. This would happen until the ballon was moved either closer to the turret or at the same distance but with better lighting.

The testing for eight and nine feet was done in early evening, around 5pm. At this point the sun was in the back yard, shining into the porch area we were on. This created extra shadows and made the target a different color than what it actually was. The Huskylens could not always see the target due to it being too bright at this point. We either needed to move it closer to the area of the porch closer to the house or make our own shade by putting an object that would project shade onto the target. This would allow enough light for cameras to operate but leave enough darkness that the Huskeylens contrast was not affected in being able to see the ballon.

Some testing not seen in Figure 43 is night time setting with no sunlight and just porchlights. Even with plenty of light that you can see around you with no issue, it was still not enough for the Sentry cameras to pick up the target, not allowing the HuskeyLens to take over. In order for our system to work during a late evening setting, someone had to be about 2 feet from the target shining a phone flashlight on it. With this flashlight, the system functioned like it was in the day time. We do notice slight inaccuracy compared to daylight however because we can control where the shadow is created and how big it is, its much easier to move light to increase accuracy which we were not able to do in sunlight other then moving the object to know places.

Table 44 – Turret Accuracy

Distance	Hit	Miss	Lock time S1	Lock time S2	Lock time S3	Lock time S4	Lock time S5	Lock time S6	Average hit time	Accuracy
6ft	5	1	0.9	1.6	1.7	1	1	1.6	1.3	83.33333
6ft	5	1	1.7	1.7	1.6	1.7	1.5	0.98	1.53	83.33333
6ft	4	2	1.6	1	0.98	0.77	0.77	1.7	1.136666667	66.66667
6ft	5	1	1	0.98	1.5	1.9	0.98	1.6	1.326666667	83.33333
6ft	6	0	1.6	0.98	1.7	1.6	1.7	1.5	1.513333333	100
7ft	3	3	0.74	0.99	1.1	0.78	1.6	0.94	1.025	50
7ft	4	2	0.99	0.78	0.7	0.9	0.9	0.9	0.861666667	66.66667
7ft	6	0	2.5	2.1	1.3	1.4	1.9	1.8	1.833333333	100
7ft	6	0	2	1.3	1.8	1.8	1.3	1.3	1.583333333	100
7ft	6	0	1.4	1.4	2	1.7	1.7	1.3	1.583333333	100
8ft	5	1	1.3	2.2	2.1	1.3	2.1	1.3	1.716666667	83.33333
8ft	6	0	1.4	2.1	2.2	2.1	1.5	1.4	1.783333333	100
8ft	6	0	1.5	1.5	1.5	1.3	1.5	1.4	1.45	100
8ft	5	1	1.4	1.3	1.3	1.3	1.4	2.1	1.466666667	83.33333
8ft	6	0	2.2	2.1	2.1	1.3	1.3	1.3	1.716666667	100
9ft	6	0	1.3	2.15	2.2	2.4	2.3	2.2	2.091666667	100
9ft	6	0	1.3	2.25	2.2	2.3	1.4	1.3	1.791666667	100
9ft	6	0	1.4	1.8	2.2	1.9	2.4	1.8	1.916666667	100
9ft	5	1	1.3	2.1	1.27	1.3	2.1	1.4	1.578333333	83.33333
9ft	6	0	1.4	2.2	1.4	1.3	2.3	2.2	1.8	100

Ch 10 Administrative content

This section serves as guidance on how to navigate the roles of this project and the cost of the project as well. This is important for such a project as to see how the balance of workload, not just for the individual person but for the whole team, will look and fair out.

10.1 Bill of Materials

Below is our Bill of Materials for this project and the expected expenses for each item. We plan on the majority of our costs coming from PCB ordering. The reason for this is due to the high tariffs at the moment and the size of our boards. We are including a cost buffer to also allow for failure of parts or misaligning issues that could appear during this project.

Table 5 – BOM

Subsystem	Part Name	Quantity	Unit	Estimated Price per [$]
Hardware	Microcontroller	1	1	$5.06
Hardware	BLDC Motor	2	1	$57.26
Hardware	High Torque Servo	3	1	$17.99
Hardware	Electronic Speed Controller	2	1	$24.00
All	IMU	1	1	$12.95
Hardware	LEDs	1	50	$7.10

Hardware	Buzzer	1	1	$0.54
Hardware	Laser Diode	1	1	$18.95
Hardware	SG90 Servo	1	1	$3.62
Hardware	Resistors			
Hardware	PSU (Mean-Well 600 12v)	1	1	$80.60
Hardware	Emergency Button	1	1	$5.90
Hardware	5V Regulator	1	1	$0.44
Hardware	Mechanical Misc (bearings,springs)	1	1	$131.00
Hardware	3.3V Regulator	1	1	$0.44
Hardware	PLA Filament	1	1	$17.99
Software	Husky Lens	1	1	$54.90
Software	Cameras (OV5640)	2	1	$11.99
Hardware	PCBs	3	1	$150.00
All	20% Cost Buffer	1		$120.15
Total				**$1,150.11**

10.2 Project job description per person

Table 6 – Job Descriptions

Description	Name
Modeling of power PCB and access box controls	*Nick Martucci*
Modeling of 3D printed turret and turret PCB	*Carlos Garcia*
Functionality and compatibility of PCBs working	*Nick Martucci and Carlos Garcia*
Choosing MCU	*Devak Sharma*
Data transfer and saving between sentry cameras and HuskyLens	*Devak Sharma and Christian Solanet*
Motion tracking algorithm	*Devak Sharma and Christian Solanet*
Encoding of LEDs and button on access box and any other peripherals	*Christian Solanet*
Modeling of hardware to hold cameras	*Nick Martucci and Carlos Garcia*
Modeling of access box	*Nick Martucci and Carlos Garcia*
Motor Controls	*Devak Sharma*

10.3 Milestones for SD1 and SD2

Table 7 – SD1 milestones

Project idea and initial research	Fall Week 1 to Week 3
Divide and Conquer	Fall Week 2 to Week 3
Deep Research into parts and specs	Fall Week 3 to Week 5
Start drawing schematics and working on initial code generation	Fall Week 5 to Week 8
Midterm report	Fall Week 3 to Week 8
Start Testing physical breadboard and initial code	Fall Week 8 to week 12
Start PCB design work	Fall Week 12 to Week 14
Final Report	Fall Week 8 to Week 16

Table 39 – SD2 milestones

Fully working Breadboard prototype with at least initial laser diode detection of target	Spring Week 1
Order initial PCBs	Spring Week 1
Start soldering on components	Spring Week 3
Test all stress cases see how far range can work and redesign PCBs if needed	Spring Week 4 to Week 8
Final PCB Design	By Spring Week 10
Final Demo	Spring Week 16

Chapter 11 - Conclusion

This project, although interesting, took lots of time and work to put together. Two sides need to come together for this project to work, one being the hardware and one being the software. Even though it gets broken down further into task that each member must be able to create and leave room for adjustments, as many revisions will be needed when final integration and testing will happen. We have the hardware side which will consist of the access box and turret. The access box and power design were given to one EE and the turret along with its needed peripherals, and PCB was given to the other. Two CE made up the software side with one working on being able to have initial detection of our target while the other focused on fine tracking the object once it had been spotted. Each step would not have been possible without each other's hard work to initial testing to see where we are at and see what needs to be done for the future. Let's look at some achievements and struggles on each side.

The access box in the block diagram shows it being nicely sectioned and zoned out. After building the PCB and schematic, to save in cost and space, it seems it would be easier to combine all the sections. We will have to place LEDs in the proper place compared to the

ESP32 on the board to avoid too many crossing lines. The buzzer circuit will need to be isolated as well in case any adjustments need to be made, such as adding more buzzers or needing to reroute shorted pins. We are saving the printing of the access box last since we do not know the size of the PCB yet since the final designs still need to be made. The switches is what the height of the access box needs to be based on and since the switches are only about 2 inches high, the LEDs will probably stick further then the switches which means adjustments to how we implement the LEDs might be needed at a later time. The initial buzzer ordered for testing also does not work with a constant GPIO pin input. We must create a square wave by having the GPIO pin active for some time then turning it off for some time then repeating this cycle. This causes a beeping buzzer rather than the constant we wanted but if the delay is small enough, we can then create a constant buzz of sorts

From the turret hardware side most of my work this semester has been a mix of learning new things and small but important wins. One big challenge was getting the power-switching and safety feature off the page and into a working prototype. I spent a good amount of time breadboarding the relay driver stage, chasing down wiring mistakes and coil issues until the MOSFET, flyback diodes, TVS placement, and E-Stop wiring all worked together the way we wanted. A clean click when the system arms, a clear LED indication, and guaranteed hardware shut-off when the E-Stop is pressed. On the motion side I assembled and tested the 3D-printed pan/tilt with the SunFounder servos, driving them from a joystick demo because the final control software wasn't ready yet. That testing showed real problems like the servo rail sagging down to about 4.5 V when both axes moved at once on the breadboard setup, but it also showed that the mechanical design itself was good. Learning SolidWorks well enough to model the pan/tilt geometry, servo mounts, and clearances right was another big challenge but in the end it paid off: we now have a pan/tilt kit that fits our hardware, leaves room for wiring and future sensors, and gives the rest of the team a solid base to keep building on in Senior Design II.

For the vision portion of the project, the color-based blob detection proved to be quite challenging. Initially, the plan was to use a pretrained neural network, but this proved impossible without the use of a more powerful MCU. When AI was ruled out, software libraries were considered, but this was not possible due to memory restrictions on ESP32 boards. The only way to implement this feature was to build it from scratch. The first implementation attempted to use recursion to visit the neighboring pixels of any red pixel, then add them to a blob if necessary. This solution was too slow and did not function properly. The idea was scrapped, and a stack was used instead to go to each of the four neighboring pixels instead. This worked much better and was a huge achievement in the creation of the color-based blob detection code.

The second half of the vision portion of the project was the setting up of the Arducam cameras. Getting the cameras to connect properly to the board through the dedicated SPI pins was a struggle due to the lack of documentation on the specific board used for testing. Searching online for the exact SPI pins never yielded a decisive answer for the board used, rather a “guess” from online AI tools or generic pin numbers from

documentation for other similar boards. Eventually the right pins were found after much trial and error, but it still wasted much time and prevented testing of other important camera features. Taking a picture with the camera also proved difficult for the same reason. There was little to no documentation on what registers need to be written to start an image capture, and where the image buffer is stored after capture. This was done using an example code found online that included many useless functions for this project and was not exactly derived for the specific camera used. Because of this struggle, a huge achievement was seeing a clear image taken from the camera appear on a laptop for the first time. It was then possible to edit the image and process it using the previously made color detection code. After solving each issue, the time it took to solve each remaining issue slowly decreased, until the current state of development, which involves tweaking configuration values. The fully integrated software can now successfully accomplish the task of red object detection, with only minor improvements left on the to do list.

Appendix

Appendix A - References

General Sources / Web Articles / Research / Blogs

[1] "Project Convergence: Revolutionizing Targeting in Large-Scale Combat Operations," C-RAM Research. [Online]. Available: Project Convergence: Revolutionizing Targeting in Large-Scale Combat Operations

[2] "UCF Logo, Identity, University, Crest, Emblem," PNG Clipart HD. [Online]. Available: Ucf Logo, Identity, University, Crest, Emblem PNG Clipart HD

[3] "HuskyLens AI Machine Vision Sensor," DFRobot Wiki. [Online]. Available: https://wiki.dfrobot.com/HUSKYLENS_V1.0_SKU_SEN0305_SEN0336

[4] "Nerf Dart Turret – Brushless – WiFi – WIP," Printables. [Online]. Available: https://www.printables.com/model/338574-nerf-dart-turret-brushless-wifi-wip

[5] "Stepper Motor vs. Servo Motor: Which Is Right for Your Application?," KEB America Blog, 2023. [Online]. Available: https://www.kebamerica.com/blog/stepper-motor-vs-servo-motor-which-is-right-for-your-application/

[6] "Brushless DC Motors vs. Stepper Motors," StepperOnline Support. [Online]. Available: https://www.omc-stepperonline.com/support/brushless-dc-motors-vs-stepper-motors

[7] "Top 3 Motor Technologies for Compact Machines: Stepper vs. Servo vs. BLDC," Anaheim Automation Blog, 2025. [Online]. Available: https://anaheimautomation.com/blog/post/top-3-motor-technologies-for-compact-machines-stepper-vs-servo-vs-bldc

[8] "WEBENCH® Power Designer Tool," Texas Instruments. [Online]. Available: https://www.ti.com/design-resources/design-tools-simulation/webench-power-designer.html

[9] "AI Answers About LEDs and Buzzer Components," Microsoft Copilot. [Online]. Available: https://copilot.microsoft.com

[10] "Choosing Between Brush and Brushless DC Motors," Allied Motion. [Online]. Available: https://www.alliedmotion.com/choosing-between-brush-and-brushless-dc-motors/

[11] "Advantages of Brushless DC Motors," Aspina Group. [Online]. Available: https://eu.aspina-group.com/en/learning-zone/columns/what-is/019/

[12] "The Difference Between Analog and Digital RC Servos," RadioControlInfo. [Online]. Available: https://www.radiocontrolinfo.com/the-difference-between-analog-and-digital-rc-servos/

[13] “Analog vs Digital Servos – Which Is the Better Option?,” Industrial Automation Co. [Online]. Available: https://industrialautomationco.com/blogs/news/analog-vs-digital-servos-which-is-the-better-option

[14] “Brushed vs Brushless DC Motor: What’s the Difference?,” JouAV. [Online]. Available: https://www.jouav.com/blog/brushed-vs-brushless-motor.html

[15] “Installing Arduino-ESP32 Support,” Espressif Systems. [Online]. Available: https://docs.espressif.com/projects/arduino-esp32/en/latest/installing.html

[16] “Getting Started With HuskyLens – Full Tutorial,” YouTube. [Online]. Available: https://www.youtube.com/watch?v=HZxFj2u9k00

[17] “HuskyLens Vision Sensor Overview,” IGNITE Pathways. [Online]. Available: https://www.ignitepathways.org/post/huskylens-vision-sensor

[18] “Using Servo with ESP32,” Arduino Forum. [Online]. Available: https://forum.arduino.cc/t/using-servo-with-esp32/1215968

[19] “ESP32 Servo Motor Control Guide,” DroneBot Workshop. [Online]. Available: https://dronebotworkshop.com/esp32-servo/

[20] “ESP32 Pinout Reference,” LastMinuteEngineers. [Online]. Available: https://lastminuteengineers.com/esp32-pinout-reference

Standards

[21] “IEC 60825-1: Safety of Laser Products – Equipment Classification and Requirements,” IEC. [Online]. Available: https://downloads.regulations.gov/FDA-2017-D-7011-0010/attachment_1.pdf

[22] “Laser Products – Conformance with IEC 60825-1 and IEC 60601-2-22,” U.S. FDA. [Online]. Available: https://www.fda.gov/media/110120/download

[23] “IPC-2221: Generic Standard on Printed Board Design,” IPC. [Online]. Available: https://www.tortai-tech.com/wp-content/uploads/2023/09/IPC-2221-Generic-Standard-on-Printed-Board-Design.pdf

[24] “IEC 62368-1: Audio/Video, Information and Communication Technology Equipment – Part 1: Safety Requirements,” IEC. [Online]. Available: https://www.intertek.com/ict/iec-62368-1-certification

[25] “NFPA 79: Electrical Standard for Industrial Machinery,” NFPA. [Online]. Available: https://www.nfpa.org/codes-and-standards/nfpa-79-standard-development/79

[26] “EIA/TIA-232, 422, and 485 Networks,” *Control.com Digital Data Acquisition & Networks*, [Online]. Available: https://control.com/textbook/digital-data-acquisition-and-networks/eiatia-232-422-and-485-networks

[27] “UM10204 I²C-bus Specification and User Manual,” NXP Semiconductors. Rev. 7.0 – 1 October 2021. [Online]. Available: https://www.alldatasheet.com/html-pdf/467563/NXP/UM10204/414/7/UM10204.html

Datasheets and Technical Documentation

[28] “ESP32-S3 Series Datasheet,” Espressif Systems. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf

[29] “ATmega328P 8-bit AVR Microcontroller Datasheet,” Microchip Technology. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

[30] “RP2350 Microcontroller Datasheet,” Raspberry Pi. [Online]. Available: https://datasheets.raspberrypi.com/rp2350/rp2350-datasheet.pdf

[31] “TPS56424x 4-A Synchronous Buck Converter,” Texas Instruments. [Online]. Available: https://www.ti.com/product/TPS56424

[32] “TPS56524x 5-A Synchronous Buck Converter (Rev. A),” Texas Instruments. [Online]. Available: https://www.ti.com/product/TPS56524

[33] “TPS56x210A Step-Down Regulator Datasheet,” Texas Instruments. [Online]. Available: https://www.ti.com/product/TPS56210A

[34] “TPS56623x 6-A Synchronous Step-Down Converter,” Texas Instruments. [Online]. Available: https://www.ti.com/product/TPS56623

[35] “LMR60440 4-A Synchronous Buck Converter,” Texas Instruments. [Online]. Available: https://www.ti.com/product/LMR60440

[36] “TPS6213x 3-A Step-Down Converter Datasheet,” Texas Instruments. [Online]. Available: https://www.ti.com/product/TPS62130

[37] “TPS56x200 Step-Down Regulator Datasheet,” Texas Instruments. [Online]. Available: https://www.ti.com/product/TPS56200

[38] “ISM330DHCX Inertial Module Datasheet,” STMicroelectronics. [Online]. Available: https://www.st.com/resource/en/datasheet/ism330dhcx.pdf

[39] “PPTC021LFBN-RC Connector Datasheet,” Sullins Connector Solutions. [Online]. Available: https://www.digikey.com

[40] “SSQ-102-03-G-S Header Datasheet,” Samtec Inc. [Online]. Available: https://www.digikey.com

[41] “SL-102-T-39 Header Datasheet,” Samtec Inc. [Online]. Available: https://www.digikey.com

[42] “310-13-132-41-001000 Connector Datasheet,” Mill-Max. [Online]. Available: https://www.digikey.com

[43] “Piezo Buzzer, 3–24 VDC,” RS PRO. [Online]. Available: 66b5745f84cf99d25b4e2e20f736f74e.pdf

[44] “AI-1223-TWT Buzzer Datasheet,” PUI Audio. [Online]. Available: specs-AI-1223-TWT-5V-R.pdf

[45] “CEM-1206S Transducer Datasheet,” SameSky. [Online]. Available: CEM-1206S Datasheet - Audio Transducers | Buzzers | Same Sky

[46] “KSC1815YTA Transistor Datasheet,” onsemi. [Online]. Available: https://www.onsemi.com

[47] “CB1-12V Automotive Relay Technical Summary,” Panasonic. [Online]. Available: https://www.digikey.com

[48] “V23134 Series Power Relay Datasheet,” TE Connectivity. [Online]. Available: https://www.mouser.com/datasheet/2/418/5/NG_DS_V23134-X0000-A001_0314_134_0314-844411.pdf

[49] “40-40 Relay Specifications V2.0,” MGI Speedware. [Online]. Available: https://mgispeedware.com/wp-content/uploads/2018/07/40-40-Relay-Specifications-V2.0.pdf

[50] “650 nm 5 mW Laser Diode Module,” Adafruit via Digi-Key. [Online]. Available: https://www.digikey.com/en/products/detail/adafruit-industries-llc/1056/7035009

[51] “LRS-600 Series 600 W Power Supply Datasheet,” Mean Well. [Online]. Available: https://files.trcelectronics.com/datasheets/lrs600.pdf

[52] “ESP32-S3-WROOM-2-N32R16V Module Datasheet,” Espressif Systems via Digi-Key. [Online]. Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/6644/ESP32-S3-WROOM-2-N32R16V.pdf

[53] “Standard Multilayer Ceramic Capacitors,” Manufacturer Datasheet. [Online]. Available: *various manufacturers*

Other References

[54] “ChatGPT.” Accessed: Sept. 2025. [Online]. Available: https://chatgpt.com

[55] “Claude AI.” Accessed: Sept. 2025. [Online]. Available: https://claude.ai/new

[56] “Google AI.” Accessed: Sept. 2025. [Online]. Available: https://ai.google

[57]“YOLOv5 GitHub Repository,” Ultralytics. [Online]. Available: https://github.com/ultralytics/yolov5?tab=readme-ov-file

[58] “Faster R-CNN – Machine Learning,” GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/faster-r-cnn-ml/

[59] “How SSD Object Detection Works,” ArcGIS Developers. [Online]. Available: https://developers.arcgis.com/python/latest/guide/how-ssd-works/

[60] “YOLO Object Detection Explained,” DataCamp. [Online]. Available: https://www.datacamp.com/blog/yolo-object-detection-explained

[61] “SPI Master Driver,” Espressif Systems. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/peripherals/spi_master.html

[62] “External RAM on ESP32,” Espressif Systems. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/external-ram.html

[63] “ESP32-S3 Datasheet,” Espressif Systems. [Online]. Available: https://documentation.espressif.com/esp32-s3_datasheet_en.pdf

[64] “ESP32-S3-WROOM-2-N32R16V Datasheet,” Digi-Key Electronics. [Online]. Available:

https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/6644/ESP32-S3-WROOM-2-N32R16V.pdf

[65] “MEGA 3MP Camera Module Product Page,” Arducam. [Online]. Available: https://www.arducam.com/presale-mega-3mp-color-rolling-shutter-camera-module-with-solid-camera-case-for-any-microcontroller.html

Appendix B – Code

MainBoard.ino

```
// ESP32 #1: Forwards bytes between Python <-> ESP32 #2

#define RX1 16
#define TX1 17

void setup() {
  Serial.begin(115200);               // USB to Python
  Serial1.begin(115200 , SERIAL_8N1, RX1, TX1);  // UART1 to ESP32 #2
}

void loop() {
  // From laptop → ESP32 #2
  while (Serial.available()) {
    uint8_t b = Serial.read();
    Serial1.write(b);
  }

  // From ESP32 #2 → laptop
  while (Serial1.available()) {
    uint8_t b = Serial1.read();
    Serial.write(b);
  }
}
```

Main.ino

```
#include "camera.h"
#include "uart.h"
#include "color_detection.h"
#include <math.h>

uint8_t temp;
Camera cam1(10, 1);
Camera cam2(9, 2);
```

```
// DEBUG ONLY
// void writeImageToMonitor() {
//   uint32_t bytesRead = 0;
//   uint8_t imageBuff[IMAGE_BUFFER_SIZE] = {0};
//   uint32_t size;
//   Serial1.print("total length: ");
//   Serial1.println(cam.getTotalLength());

//   while (bytesRead < cam.getTotalLength()) {
//     size = cam.readBuffer(imageBuff, IMAGE_BUFFER_SIZE);
//     bytesRead += size;

//     for (int i = 0; i < size; i++) {
//       Serial1.print(imageBuff[i], HEX);
//       Serial1.print(' ');

//       if (imageBuff[i] == 0xD9 && imageBuff[i-1] == 0xFF) {
//         Serial1.println(bytesRead);
//         return;
//       }
//     }
//     Serial1.println();
//   }
// }

void convertRGB565toRGB888(const uint8_t* rgb565, uint8_t* rgb888,
uint32_t numPixels) {
  for (uint32_t i = 0; i < numPixels; i++) {
    uint16_t pixel = (rgb565[2 * i] << 8) | rgb565[2 * i + 1];
    uint8_t r = ((pixel >> 11) & 0x1F);
    uint8_t g = ((pixel >> 5) & 0x3F);
    uint8_t b = (pixel & 0x1F);

    rgb888[i* 3] = r * 255 / 31;
    rgb888[i * 3 + 1] = g * 255 / 63;
    rgb888[i * 3 + 2] = b * 255 / 31;
  }
}

void updateImageWithBoundingBoxes(uint8_t *image, uint32_t width, uint32_t
height, std::vector<BoundingBox> boundingBoxes) {
  for (const auto &blob : boundingBoxes) {
```

```
    // Draw rectangles around blobs
    for (int x = blob.minX; x <= blob.maxX; ++x) {
      image[(blob.minY * width + x) * 3] = 0;
      image[(blob.minY * width + x) * 3 + 1] = 0;
      image[(blob.minY * width + x) * 3 + 2] = 255;
      image[(blob.maxY * width + x) * 3] = 0;
      image[(blob.maxY * width + x) * 3 + 1] = 0;
      image[(blob.maxY * width + x) * 3 + 2] = 255;
    }
    for (int y = blob.minY; y <= blob.maxY; ++y) {
      image[(y * width + blob.minX) * 3] = 0;
      image[(y * width + blob.minX) * 3 + 1] = 0;
      image[(y * width + blob.minX) * 3 + 2] = 255;
      image[(y * width + blob.maxX) * 3] = 0;
      image[(y * width + blob.maxX) * 3 + 1] = 0;
      image[(y * width + blob.maxX) * 3 + 2] = 255;
    }
  }
}

void testPhotoTime(Camera& cam) {
  unsigned long start = millis();

  cam.takePicture();

  unsigned long duration = millis() - start;

  Serial1.print("Full image capture: ");
  Serial1.println(duration);
}

void sendPic(int wholeImage, Camera& cam) {
  cam.takePicture();

  uint32_t totalSize = cam.getTotalLength();
  uint32_t pixelBufferSize = cam.getImageWidth() *
pow(cam.getImageHeight(), wholeImage);
  uint8_t* rgb565Buffer = (uint8_t*)calloc(pixelBufferSize * 2,
sizeof(uint8_t));
  uint8_t* rgb888Buffer = (uint8_t*)calloc(pixelBufferSize * 3,
sizeof(uint8_t));

  writeHeader();
```

```
  writeInt(totalSize * 3 / 2);

  int bytesRead = 0;
  int size = 0;
  std::vector<BoundingBox> boundingBoxes;

  while (bytesRead < totalSize) {
    size = cam.readBuffer(rgb565Buffer, pixelBufferSize * 2); // read
buffer and set size to number of bytes read

    convertRGB565toRGB888(rgb565Buffer, rgb888Buffer, size / 2); // size
is now twice the size of pixels to read
    free(rgb565Buffer); // Clear up memory so can be used for visited
buffer

    boundingBoxes = detectRedBlobs(rgb888Buffer, cam.getImageWidth(),
cam.getImageHeight());

    updateImageWithBoundingBoxes(rgb888Buffer, cam.getImageWidth(),
cam.getImageHeight(), boundingBoxes);
    sendBufferUart(rgb888Buffer, size * 3 / 2);
    bytesRead += size;
  }

  free(rgb888Buffer);

  writeInt(boundingBoxes.size());

  for (BoundingBox& box : boundingBoxes) {
    writeInt(box.minX);
    writeInt(box.maxX);
    writeInt(box.minY);
    writeInt(box.maxY);
    writeInt(box.size);
    writeInt(cam.getCamNumber());
  }

  writeEndTransferHeader();
}

void performAction(uint8_t *commandBuffer) {
  Camera& cam = (commandBuffer[0] == 0x01) ? cam1 : cam2;
```

```
  switch (commandBuffer[1]) {
    case 0x00:
      sendPic(1, cam);
      break;
    case 0x01:
      cam.setResolution(commandBuffer[2]);
      break;
    case 0x02:
      cam.setExposure(commandBuffer[2]);
      break;
    case 0x03:
      cam.setSaturation(commandBuffer[2]);
      break;
    case 0x04:
      cam.setSharpness(commandBuffer[2]);
      break;
    case 0x05:
      cam.setContrast(commandBuffer[2]);
      break;
    case 0x06:
      cam.setWhiteBalance(commandBuffer[2]);
      break;
  }
}

void bedroomSetup(Camera& cam) {
  cam.setImageFormat(CAM_IMAGE_PIX_FMT_RGB565);
  //cam.setResolution(CAM_IMAGE_MODE_QVGA); // 153 ms 153kb real: 205ms
  //cam.setResolution(CAM_IMAGE_MODE_VGA); // 614 ms 614kb real: 820ms
  //cam.setResolution(CAM_IMAGE_MODE_96X96); // 18 ms 18kb real: 24ms
  //cam.setResolution(CAM_IMAGE_MODE_128X128); // 32 ms 32kb real: 44ms
  cam.setResolution(CAM_IMAGE_MODE_320X320); // 204 ms 204kb real: 273ms
  cam.setExposure(CAM_EV_LEVEL_3);
  cam.setSaturation(CAM_STAURATION_LEVEL_1);
  cam.setSharpness(CAM_SHARPNESS_LEVEL_AUTO);
  cam.setContrast(CAM_CONTRAST_LEVEL_MINUS_2);
  cam.setWhiteBalance(CAM_WHITE_BALANCE_MODE_OFFICE);
}

void setup() {
  beginUart(115200);
```

```
  bedroomSetup(cam1);
  bedroomSetup(cam2);

  //testPhotoTime();

  // Serial1.println("Before");
  // sendPic(1);
  // Serial1.println("After");
}

void loop() {
  int commandLength = 0;
  uint8_t commandBuffer[4] = {0};
  if (Serial1.available()) {
    temp = Serial1.read();

    if (temp == 0x55) {
      while (Serial1.available()) {
        commandBuffer[commandLength] = Serial1.read();

        if (commandBuffer[commandLength] == 0xAA) {
          break;
        }
        commandLength++;
      }
      writeUart(commandBuffer[0]);
      writeUart(commandBuffer[1]);
      writeUart(commandBuffer[2]);
      performAction(commandBuffer);
    }
  }
}
```

Camera.cpp

```
#include "camera.h"
#include <SPI.h>
#include <Arduino.h>

#define ARDUCHIP_FIFO 0x04
#define ARDUCHIP_FIFO_2 0x07
#define ARDUCHIP_TRIG 0x44
```

```
#define CAP_DONE_MASK 0x04
#define SINGLE_FIFO_READ 0x3D
#define BURST_FIFO_READ 0x3C

#define FIFO_CLEAR_ID_MASK 0x01
#define FIFO_SIZE1 0x45
#define FIFO_SIZE2 0x46
#define FIFO_SIZE3 0x47
#define FIFO_CLEAR_MASK 0x80
#define FIFO_START_MASK 0x02

#define CAM_REG_SENSOR_STATE_IDLE (1 << 1)
#define CAM_SENSOR_RESET_ENABLE (1 << 6)
#define CAM_REG_SENSOR_RESET 0x07
#define CAM_REG_SENSOR_STATE 0x44
#define CAM_REG_FORMAT 0X20
#define CAM_REG_CAPTURE_RESOLUTION 0X21
#define CAM_SET_CAPTURE_MODE (0 << 7)
#define CAM_REG_CONTRAST_CONTROL 0X23
#define CAM_REG_SATURATION_CONTROL 0X24
#define CAM_REG_EV_CONTROL 0X25
#define CAM_REG_WHILEBALANCE_MODE_CONTROL 0X26
#define CAM_REG_COLOR_EFFECT_CONTROL 0X27
#define CAM_REG_SHARPNESS_CONTROL 0X28

Camera::Camera(int csPin, int camNumber) {
    setCsPin(csPin);
    this->camNumber = camNumber;
    totalLength = 0;
    remainingLength = 0;
    burstFirstFlag = 0;
    imageWidth = 0;
    imageHeight = 0;

    SPI.begin(12, 11, 13, csPin);
    pinMode(csPin, OUTPUT);
    csLow();

    begin();

    setImageFormat(CAM_IMAGE_PIX_FMT_RGB565);
    setResolution(CAM_IMAGE_MODE_VGA);
    setExposure(CAM_EV_LEVEL_DEFAULT);
```

```
    setSaturation(CAM_STAURATION_LEVEL_DEFAULT);
    setSharpness(CAM_SHARPNESS_LEVEL_AUTO);
    setWhiteBalance(CAM_WHITE_BALANCE_MODE_DEFAULT);
    setContrast(CAM_CONTRAST_LEVEL_DEFAULT);
    setColorEffect(CAM_COLOR_FX_NONE);
    csHigh();
  }

Camera::~Camera() {}

int Camera::getCsPin() { return csPin; }
int Camera::getCamNumber() { return camNumber; }
uint32_t Camera::getImageWidth() { return imageWidth; }
uint32_t Camera::getImageHeight() { return imageHeight; }
uint32_t Camera::getTotalLength() { return totalLength; }
uint32_t Camera::getRemainingLength() { return remainingLength; }
uint8_t Camera::getBurstFirstFlag() {return burstFirstFlag; }
uint8_t Camera::getImageFormat() { return imageFormat; }
uint8_t Camera::getResolution() { return resolution; }
uint8_t Camera::getExposure() { return exposure; }
uint8_t Camera::getSaturation() { return saturation; }
uint8_t Camera::getSharpness() { return sharpness; }
uint8_t Camera::getwhiteBalance() { return whiteBalance; }
uint8_t Camera::getContrast() { return contrast; }
uint8_t Camera::getColorEffect() { return colorEffect; }

void Camera::setCsPin(int csPin) { this->csPin = csPin; }

void Camera::setImageFormat(uint8_t imageFormat) {
  this->imageFormat = imageFormat;
  writeReg(CAM_REG_FORMAT, imageFormat); // set the data format
  waitI2cIdle();
}

void Camera::setResolution(uint8_t resolution) {
  this->resolution = resolution;
  writeReg(CAM_REG_CAPTURE_RESOLUTION, CAM_SET_CAPTURE_MODE | resolution);
  waitI2cIdle(); // Wait I2c Idle
  switch (this->resolution) {
    case CAM_IMAGE_MODE_QQVGA:
      imageWidth = 160;
      imageHeight = 120;
      break;
```

```
case CAM_IMAGE_MODE_320X320:
  imageWidth = 320;
  imageHeight = 320;
  break;
case CAM_IMAGE_MODE_QVGA:
  imageWidth = 320;
  imageHeight = 240;
  break;
case CAM_IMAGE_MODE_VGA:
  imageWidth = 640;
  imageHeight = 480;
  break;
case CAM_IMAGE_MODE_SVGA:
  imageWidth = 800;
  imageHeight = 600;
  break;
case CAM_IMAGE_MODE_HD:
  imageHeight = 1280;
  imageHeight = 720;
  break;
case CAM_IMAGE_MODE_SXGAM:
  imageWidth = 1280;
  imageHeight = 960;
  break;
case CAM_IMAGE_MODE_UXGA:
  imageWidth = 1600;
  imageHeight = 1200;
  break;
case CAM_IMAGE_MODE_FHD:
  imageWidth = 1920;
  imageHeight = 1080;
  break;
 case CAM_IMAGE_MODE_QXGA:
  imageWidth = 2048;
  imageHeight = 1536;
  break;
case CAM_IMAGE_MODE_96X96:
  imageWidth = 96;
  imageHeight = 96;
  break;
case CAM_IMAGE_MODE_128X128:
  imageWidth = 128;
  imageHeight = 128;
  break;
```

```
  }
}

void Camera::setExposure(uint8_t exposure) {
  this->exposure = exposure;
  writeReg(CAM_REG_EV_CONTROL, exposure);
  waitI2cIdle(); // Wait I2c Idle
}

void Camera::setSaturation(uint8_t saturation) {
  this->saturation = saturation;
  writeReg(CAM_REG_SATURATION_CONTROL, saturation); // set Saturation
Level
  waitI2cIdle();
}

void Camera::setSharpness(uint8_t sharpness) {
  this->sharpness = sharpness;
  writeReg(CAM_REG_SHARPNESS_CONTROL, sharpness); // set Brightness Level
  waitI2cIdle();
}

void Camera::setWhiteBalance(uint8_t whiteBalance) {
  this->whiteBalance = whiteBalance;
  writeReg(CAM_REG_WHILEBALANCE_MODE_CONTROL, whiteBalance); // set Light
Mode
  waitI2cIdle();
}

void Camera::setContrast(uint8_t contrast) {
  this->contrast = contrast;
  writeReg(CAM_REG_CONTRAST_CONTROL, contrast); // set Contrast Level
  waitI2cIdle();
}

void Camera::setColorEffect(uint8_t colorEffect) {
  this->colorEffect = colorEffect;
  writeReg(CAM_REG_COLOR_EFFECT_CONTROL, colorEffect); // set effect
  waitI2cIdle();
}

CamStatus Camera::reset() {
  writeReg(CAM_REG_SENSOR_RESET, CAM_SENSOR_RESET_ENABLE);
```

```
  waitI2cIdle();
  imageFormat = CAM_IMAGE_PIX_FMT_JPG;
  resolution = CAM_IMAGE_MODE_VGA;
  return CAM_SUCCESS;
}

CamStatus Camera::begin() {
  // // reset cpld and camera
  writeReg(CAM_REG_SENSOR_RESET, CAM_SENSOR_RESET_ENABLE);
  waitI2cIdle(); // Wait I2c Idle
  return CAM_SUCCESS;
}

CamStatus Camera::takePicture() {
  flushFifo(); // reset fifo read pointer to 0
  clearFifoFlag(); // clear capture complete flag
  writeReg(ARDUCHIP_FIFO, FIFO_START_MASK); // start capture (PUT IN
FUNCTION)

  while (getBit(ARDUCHIP_TRIG, CAP_DONE_MASK) == 0); // wait for image
capture complete
  remainingLength = readFifoLength();
  totalLength = remainingLength;
  burstFirstFlag = 0; // set to 0 to do send dummy byte before doing first
burst read
  return CAM_SUCCESS;
}

void Camera::csHigh() {
  digitalWrite(csPin, HIGH);
}

void Camera::csLow() {
  digitalWrite(csPin, LOW);
}

uint8_t Camera::cameraSpiTransfer(uint8_t data) {
  return SPI.transfer(data);
}

void Camera::cameraSpiTransferBlock(uint8_t *buff,uint16_t len){
    SPI.transfer(buff,len);
}
```

```
uint32_t Camera::imageAvailable() {
    return remainingLength;
}

uint32_t Camera::readBuffer(uint8_t* buffer, uint32_t size) {
  if (imageAvailable() == 0 || (size == 0)) {
      return 0;
  }

  if (remainingLength < size) {
      size = remainingLength;
  }

  SPI.beginTransaction(SPISettings(8000000, MSBFIRST, SPI_MODE0));
  csLow();
  setFifoBurst();
  if (burstFirstFlag == 0) {
      burstFirstFlag = 1;
      cameraSpiTransfer(0x00);
  }

  size_t idx = 0;
  const size_t block = 8192;
  while (idx < size) {
      size_t n = block < (size - idx) ? block : size - idx;
      SPI.transferBytes(NULL, buffer + idx, n);
      idx += n;
  }

  csHigh();
  SPI.endTransaction();
  remainingLength -= size;
  return size;
}

uint8_t Camera::readByte() {
  uint8_t data = 0;
  csLow();
  cameraSpiTransfer(SINGLE_FIFO_READ);
  cameraSpiTransfer(0x00);
  data = cameraSpiTransfer(0x00);
  csHigh();
```

```
  remainingLength -= 1;
  return data;
}

void Camera::writeReg(int reg, int data) {
  csLow();
  cameraSpiTransfer(reg | 0x80); // 8th bit is the write bit (1)
  cameraSpiTransfer(data);
  csHigh();
}

uint8_t Camera::readReg(uint8_t reg) {
  uint8_t value;
  csLow();
  cameraSpiTransfer(reg & 0x7F); // 8th bit is read bit (0)
  value = cameraSpiTransfer(0x00); // Dummy byte
  value = cameraSpiTransfer(0x00); // Dummy byte
  csHigh();
  return value;
}

void Camera::flushFifo() {
  writeReg(ARDUCHIP_FIFO_2, FIFO_CLEAR_MASK);
}

void Camera::clearFifoFlag() {
  writeReg(ARDUCHIP_FIFO, FIFO_CLEAR_ID_MASK);
}

uint32_t Camera::readFifoLength() {
  uint32_t len1, len2, len3, length = 0;
  len1 = readReg(FIFO_SIZE1);
  len2 = readReg(FIFO_SIZE2);
  len3 = readReg(FIFO_SIZE3);
  length = ((len3 << 16) | (len2 << 8) | len1) & 0xffffff;
  return length;
}

uint8_t Camera::getBit(uint8_t addr, uint8_t bit) {
  uint8_t temp;
  temp = readReg(addr);
  temp = temp & bit;
  return temp;
```

```
}

void Camera::setFifoBurst() {
  cameraSpiTransfer(BURST_FIFO_READ);
}

void Camera::waitI2cIdle() {
  while ((readReg(CAM_REG_SENSOR_STATE) & 0X03) !=
CAM_REG_SENSOR_STATE_IDLE);
}
```

Color_detection.cpp

```
// To Run: .\blobV2.exe INPUT_FILE OUTPUT_FILE (1 for smoothener,
otherwise leave blank)
#include <iostream>
#include <stack>
#include <fstream>
#include "color_detection.h"
#include <Arduino.h>

#define MIN_AREA 100
#define DELTA 10

// RBG ranges: 255 130 160 --> 50, 0, 0
// using range from above, creates a line where any values above will be
red, and any values less will not be red.
bool isRed(uint8_t r, uint8_t g, uint8_t b) {
    return (r > 100 && r > 1.58 * g + 50 && r > 1.28 * b + 50) || (r > 50
&& r < 100 && r > g + 50 && r > b + 50); // Works well in bedroom (Might
have to change depending on environment)
}

/*
when a red pixel is found, check all neighboring pixels and update
bounding boxes
x and y are original pixel coordinates
visited is list of pixels that have been accounted for
image is the rgb888 image
box is the new bounding box these pixels are assigned to
width and height are  for image width and height
*/
void floodFill(int x, int y, std::vector<uint8_t>& visited, uint8_t
*image, BoundingBox& box, int width, int height) {
```

```
    std::stack<std::pair<int, int>> s;
    s.push({x, y});

    // direction offsets (right, left, down, up)
    const int dx[4] = {1, -1, 0, 0};
    const int dy[4] = {0, 0, 1, -1};

    // while there are pixels to check
    while (!s.empty()) {
        auto [cx, cy] = s.top();
        s.pop();

        // boundary check
        if (cx < 0 || cx >= width || cy < 0 || cy >= height)
            continue;

        // check if pixel has been visited before or is not red and skip
by if so
        int idx = cy * width + cx;
        if (visited[idx])
            continue;

        if (!isRed(image[idx * 3], image[idx * 3 + 1], image[idx * 3 +
2]))
            continue;

        // mark visited
        visited[idx] = 1;

        // update bounding box
        if (cx < box.minX) box.minX = cx;
        if (cx > box.maxX) box.maxX = cx;
        if (cy < box.minY) box.minY = cy;
        if (cy > box.maxY) box.maxY = cy;

        // add neighbors to stack
        for (int i = 0; i < 4; i++) {
            int nx = cx + dx[i];
            int ny = cy + dy[i];
            if (nx >= 0 && nx < width && ny >= 0 && ny < height)
                s.push({nx, ny});
        }
    }
```

```
}

/*
given two bounding boxes, check if they overlap
a will be existing bounding box, b will be new bounding box
*/
bool overlaps(const BoundingBox& a, const BoundingBox& b) {
    return !((a.maxX + DELTA) < b.minX ||
             (a.minX - DELTA) > b.maxX ||
             (a.maxY + DELTA) < b.minY ||
             (a.minY - DELTA) > b.maxY);
}

// bool overlaps(const BoundingBox& a, const BoundingBox& b) {
//     // If there is a separating gap larger than eps on any axis, they
do NOT overlap.
//     if (a.maxX < b.minX - DELTA) return false;
//     if (a.minX > b.maxX + DELTA) return false;
//     if (a.maxY < b.minY - DELTA) return false;
//     if (a.minY > b.maxY + DELTA) return false;
//     return true;
// }



/*
given two bounding boxes, combine them into one
a will be old bounding box, b will be new one
*/
BoundingBox mergeBoxes(const BoundingBox& a, const BoundingBox& b) {
    BoundingBox out;
    out.minX = std::min(a.minX, b.minX);
    out.minY = std::min(a.minY, b.minY);
    out.maxX = std::max(a.maxX, b.maxX);
    out.maxY = std::max(a.maxY, b.maxY);
    out.size = (out.maxX - out.minX + 1) * (out.maxY - out.minY + 1);
    return out;
}

/*
given an image and its height and width, detect blobs in it
*/
std::vector<BoundingBox> detectRedBlobs(uint8_t *image, int width, int
height) {
```

```
    std::vector<uint8_t> visited(width * height, 0); // list of pixels
that have been visited and dealt with
    std::vector<BoundingBox> boxes; // list of all bounding boxes in image

    // loop through every pixel
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            int idx = y * width + x;

            // if pixel is new and red
            if (!visited[idx] && isRed(image[idx * 3], image[idx * 3 + 1],
image[idx * 3 + 2])) {
                BoundingBox box = {x, y, x, y}; // create temporary
bounding box

                // flood fill will update visited[] and expand box
                floodFill(x, y, visited, image, box, width, height);

                // eliminate small bounding boxes
                int area = (box.maxX - box.minX + 1) * (box.maxY -
box.minY + 1);
                box.size = area;
                int merged = 0;

                // merge boxes, and if no merge, check if size is valid
                for (BoundingBox& oldBox : boxes) {
                    if (overlaps(oldBox, box)) {
                        // merge newBox into existing box
                        oldBox = mergeBoxes(oldBox, box);
                        merged = 1;  // do NOT add newBox
                    }
                }
                if (!merged) {
                    if (area > MIN_AREA) {
                        // No overlap → add newBox
                        boxes.push_back(box);
                    }
                }
            }
        }
    }

    return boxes;
```

```
}
```

Uart.cpp

```
#include "uart.h"
#include <Arduino.h>

#define IMAGE_BUFFER_SIZE 1024
#define RX1 18
#define TX1 17

// being UART connection (Probably need to update when UART is connected
via pins)
void beginUart(uint32_t baudRate) {
  Serial1.begin(baudRate, SERIAL_8N1, RX1, TX1);
}

// Headers to let listener know a picture is being sent
void writeHeader() {
    writeUart(0xff);
    writeUart(0xAA);
    writeUart(0x01);
}

// header to let listener know data transfer is over
void writeEndTransferHeader() {
  writeUart(0xff);
  writeUart(0xbb);
}

// given size of data to be sent, send it over little endian
void writeInt(uint32_t size) {
  writeUart(size & 0xff);
  writeUart((size >> 8) & 0xff);
  writeUart((size >> 16) & 0xff);
  writeUart((size >> 24) & 0xff);
}

// loop over buffer and write data
void sendBufferUart(uint8_t *buffer, uint32_t totalSize) {
  for (int i = 0; i < totalSize; i++) {
    writeUart(buffer[i]);
  }
}
```

```
// Clear out UART buffer
void flushUart() {
  while (Serial1.available()) {
      Serial1.read();
  }
}

// write data and wait 12 microseconds
void writeUart(uint8_t data) {
  Serial1.write(data);
  delayMicroseconds(12);
}
```

Color_detection.h

```
#ifndef COLOR_DETECTION_H
#define COLOR_DETECTION_H

#include <vector>
#include <cstdint>

#ifdef __cplusplus
extern "C" {
#endif

struct BoundingBox {
    int minX, minY, maxX, maxY;
    int size;
};

bool isRed(uint8_t r, uint8_t g, uint8_t b);
void floodFill(int x, int y, std::vector<uint8_t>& visited, uint8_t
*image, BoundingBox& box, int width, int height);
bool overlaps(const BoundingBox& a, const BoundingBox& b);
BoundingBox mergeBoxes(const BoundingBox& a, const BoundingBox& b);
std::vector<BoundingBox> detectRedBlobs(uint8_t *image, int width, int
height);

#ifdef __cplusplus
}
#endif
```

```
#endif
```

Uart.h

```
#ifndef UART_H
#define UART_H

#include <stdint.h>

#ifdef __cplusplus
extern "C" {
#endif

void beginUart(uint32_t buadRate);
void sendBufferUart(uint8_t *buffer, uint32_t size);
void flushUart();
void writeUart(uint8_t data);
void writeHeader();
void writeInt(uint32_t size);
void writeEndTransferHeader();

#ifdef __cplusplus
}
#endif

#endif
```

Camera.h

```
#ifndef CAMERA_H
#define CAMERA_H

#include <stdint.h>

#ifdef __cplusplus
extern "C" {
#endif

/* Image Formats */
#define CAM_IMAGE_PIX_FMT_NONE 0x00
#define CAM_IMAGE_PIX_FMT_JPG 0x01
#define CAM_IMAGE_PIX_FMT_RGB565 0x02
#define CAM_IMAGE_PIX_FMT_YUV 0x03
```

```
/* Resolutions */
#define CAM_IMAGE_MODE_QQVGA 0x00 /**DOES NOT WORK <160x120 */
#define CAM_IMAGE_MODE_QVGA 0x01 /**YES <320x240*/
#define CAM_IMAGE_MODE_VGA 0x02 /**YES <640x480*/
#define CAM_IMAGE_MODE_SVGA 0x03 /**DOES NOT WORK <800x600*/
#define CAM_IMAGE_MODE_HD 0x04 /**<1280x720*/
#define CAM_IMAGE_MODE_SXGAM 0x05 /**<1280x960*/
#define CAM_IMAGE_MODE_UXGA 0x06 /**<1600x1200*/
#define CAM_IMAGE_MODE_FHD 0x07 /**<1920x1080*/
#define CAM_IMAGE_MODE_QXGA 0x08 /**<2048x1536*/
#define CAM_IMAGE_MODE_96X96 0x0a /**YES <96x96*/
#define CAM_IMAGE_MODE_128X128 0x0b /**YES <128x128*/
#define CAM_IMAGE_MODE_320X320 0x0c /** YES <320x320*/

/* Exposure Levels */
#define CAM_EV_LEVEL_MINUS_3 0x06 /**<Level -3 */
#define CAM_EV_LEVEL_MINUS_2 0x04 /**<Level -2 */
#define CAM_EV_LEVEL_MINUS_1 0x02 /**<Level -1 */
#define CAM_EV_LEVEL_DEFAULT 0x00 /**<Level Default*/
#define CAM_EV_LEVEL_1 0x01 /**<Level +1 */
#define CAM_EV_LEVEL_2 0x03 /**<Level +2 */
#define CAM_EV_LEVEL_3 0x05 /**<Level +3 */

/* Saturation Levels */
#define CAM_STAURATION_LEVEL_MINUS_3 0x06 /**<Level -3 */
#define CAM_STAURATION_LEVEL_MINUS_2 0x04 /**<Level -2 */
#define CAM_STAURATION_LEVEL_MINUS_1 0x02 /**<Level -1 */
#define CAM_STAURATION_LEVEL_DEFAULT 0x00 /**<Level Default*/
#define CAM_STAURATION_LEVEL_1 0x01 /**<Level +1 */
#define CAM_STAURATION_LEVEL_2 0x03 /**<Level +2 */
#define CAM_STAURATION_LEVEL_3 0x05 /**<Level +3 */

/* Sharpness Levels */
#define CAM_SHARPNESS_LEVEL_AUTO 0x00 /**<Sharpness Auto */
#define CAM_SHARPNESS_LEVEL_1 0x01 /**<Sharpness Level 1 */
#define CAM_SHARPNESS_LEVEL_2 0x02 /**<Sharpness Level 2 */
#define CAM_SHARPNESS_LEVEL_3 0x03 /**<Sharpness Level 3 */
#define CAM_SHARPNESS_LEVEL_4 0x04 /**<Sharpness Level 4 */
#define CAM_SHARPNESS_LEVEL_5 0x05 /**<Sharpness Level 5 */
#define CAM_SHARPNESS_LEVEL_6 0x06 /**<Sharpness Level 6 */
#define CAM_SHARPNESS_LEVEL_7 0x07 /**<Sharpness Level 7 */
#define CAM_SHARPNESS_LEVEL_8 0x08 /**<Sharpness Level 8 */
```

```
/* White Balance */
#define CAM_WHITE_BALANCE_MODE_DEFAULT 0x00 /**< Auto */
#define CAM_WHITE_BALANCE_MODE_SUNNY 0x01 /**< Sunny */
#define CAM_WHITE_BALANCE_MODE_OFFICE 0x02 /**< Office */
#define CAM_WHITE_BALANCE_MODE_CLOUDY 0x03 /**< Cloudy */
#define CAM_WHITE_BALANCE_MODE_HOME 0x04 /**< Home */

/* Contrast Level */
#define CAM_CONTRAST_LEVEL_MINUS_3 0x06 /**<Level -3 */
#define CAM_CONTRAST_LEVEL_MINUS_2 0x04 /**<Level -2 */
#define CAM_CONTRAST_LEVEL_MINUS_1 0x02 /**<Level -1 */
#define CAM_CONTRAST_LEVEL_DEFAULT 0x00 /**<Level Default*/
#define CAM_CONTRAST_LEVEL_1 0x01 /**<Level +1 */
#define CAM_CONTRAST_LEVEL_2 0x03 /**<Level +2 */
#define CAM_CONTRAST_LEVEL_3 0x05 /**<Level +3 */

/* Color Effect */
#define CAM_COLOR_FX_NONE 0x00 /**< no effect   */
#define CAM_COLOR_FX_BLUEISH 0x01 /**< cool light   */
#define CAM_COLOR_FX_REDISH 0x02 /**< warm   */
#define CAM_COLOR_FX_BW 0x03 /**< Black/white   */
#define CAM_COLOR_FX_SEPIA 0x04 /**<Sepia   */
#define CAM_COLOR_FX_NEGATIVE 0x05 /**<positive/negative inversion  */
#define CAM_COLOR_FX_GRASS_GREEN 0x06 /**<Grass green */
#define CAM_COLOR_FX_OVER_EXPOSURE 0x07 /**<Over exposure*/
#define CAM_COLOR_FX_SOLARIZE 0x08 /**< Solarize   */

typedef enum {
    CAM_SUCCESS = 0,  /* Operation succeeded */
    CAM_ERR = -1, /* Operation failed */
} CamStatus;

class Camera {
  private:
    int csPin; // Chip select pin
    int camNumber; // Camera number (1 or 2)
    uint32_t totalLength; // Total Length of image taken in bytes
    uint32_t remainingLength; // Remaining number of bytes to process in
image
    uint8_t burstFirstFlag; // Flag bit for reading data in burst mode
    uint8_t imageFormat; // Current image format (JPEG, RGB, YUV)
    uint8_t resolution; // Current image resolution
    uint8_t exposure; // Current exposure value
```

```
  uint8_t saturation; // Current image saturation
  uint8_t sharpness; // Current image sharpness
  uint8_t whiteBalance; // Current image white balance
  uint8_t contrast; // Current image contrast
  uint8_t colorEffect; // Current image color effect
  uint32_t imageWidth; // Width of image
  uint32_t imageHeight; // Height of image

public:
  Camera(int csPin, int camNumber); // Constructor

  ~Camera(); // Destructor

  // Getters
  int getCsPin();
  int getCamNumber();
  uint32_t getImageWidth();
  uint32_t getImageHeight();
  uint32_t getTotalLength();
  uint32_t getRemainingLength();
  uint8_t getBurstFirstFlag();
  uint8_t getImageFormat();
  uint8_t getResolution();
  uint8_t getExposure();
  uint8_t getSaturation();
  uint8_t getSharpness();
  uint8_t getwhiteBalance();
  uint8_t getContrast();
  uint8_t getColorEffect();

  // Setters
  void setCsPin(int csPin);
  void setImageFormat(uint8_t imageFormat);
  void setResolution(uint8_t resolution);
  void setExposure(uint8_t exposure);
  void setSaturation(uint8_t saturation);
  void setSharpness(uint8_t sharpness);
  void setWhiteBalance(uint8_t whiteBalance);
  void setContrast(uint8_t contrast);
  void setColorEffect(uint8_t colorEffect);

  // Methods
  CamStatus reset();
```

```
    CamStatus begin();
    CamStatus takePicture();
    void csHigh();
    void csLow();
    uint8_t cameraSpiTransfer(uint8_t data);
    void cameraSpiTransferBlock(uint8_t *buff,uint16_t len);
    uint32_t imageAvailable();
    uint32_t readBuffer(uint8_t* buffer, uint32_t size);
    uint8_t readByte();
    void writeReg(int reg, int data);
    uint8_t readReg(uint8_t reg);
    void flushFifo();
    void clearFifoFlag();
    uint32_t readFifoLength();
    uint8_t getBit(uint8_t addr, uint8_t bit);
    void setFifoBurst();
    void waitI2cIdle();
};

#ifdef __cplusplus
}
#endif
#endif // camera.h
```

Appendix C – ChatGPT Prompts and Outcomes

[1a] ***Case Study 1 Prompt:*** *"Given the workload of controlling two servo motors, two cameras whose image data will be processed using light machine learning model (i.e. blob detection), an main camera for fine tracking (AI vision sensor), several peripherals (LEDs, buzzers, etc.), and other controls, what MCU will be best for this architecture."*

[1b] ***Case Study 1 ChatGPT Outcome:*** *"Nice — you've got a real-time control + tiny-vision workload, so the right MCU choice comes down to splitting hard real-time I/O (servos/timers/interrupts) from vision/ML work (camera frames, blob detection, NN inference, bigger memory needs). My recommendation: use a 2-MCU architecture and here are concrete, practical options + why.*

Short answer / recommended setup (best balance)

Primary "vision / ML" MCU: ESP32-S3 (dev board with PSRAM / OV2640 support)
Secondary "real-time control" MCU: Teensy 4.1 (or RP2040/Pico W if you want cheaper)

This combo gives you hardware ML acceleration and camera support on the vision side, and a very fast, deterministic MCU with plenty of PWM/timers for servo control on the control side. Espressif+1

Why this split works

Vision node (ESP32-S3):

ESP32-S3 includes vector instruction support and libraries (ESP-NN / ESP-DSP) that accelerate tiny neural nets / signal processing — good for blob detection and running lightweight NN inference on-device. It also has many dev boards with PSRAM and camera (OV2640) support which are useful for buffering frames. Espressif+1

ESP32 boards are also convenient for Wi-Fi/BLE telemetry and talking to AI vision sensors (HuskyLens) over UART / I²C / serial. (You mentioned HuskyLens before — ESP32 is widely used with such sensors.) Makerguides.com+1

Control node (Teensy 4.1 / RP2040):

Servo control requires low-jitter PWM, many timers, and fast interrupt handling. The Teensy 4.1's NXP i.MX RT (600 MHz class) gives deterministic, high-performance control loops and lots of I/O (including hardware PWM, timers, and real-time libraries). It's ideal when you need stable PID loops and sub-millisecond timing. PJRC+1

If cost/size matters, the RP2040 (Pico / Pico W) is a strong lower-cost choice: dual-core M0+ (good for splitting control tasks), many PWM channels, and unique PIO for custom peripherals — but earlier Pico variants lack the same CPU horsepower or hardware float performance as Teensy. Use Pico W if you need built-in Wi-Fi but not heavy ML. Raspberry Pi+1

Alternatives and when to pick them

Single-board SBC (Raspberry Pi Zero 2 W / Pi 4) + MCU for servos — if your blob detection gets heavier (larger models, more frames, or you want OpenCV), an SBC handles vision far more easily. Use the SBC for vision and a small MCU (RP2040 or Teensy) for real-time servo control. (SBCs are less deterministic for real-time PWM without an MCU.) Hackster

STM32 with on-chip NPU (newer MCUs like STM32N6 / M55 series) — if you want single-MCU with stronger on-chip ML acceleration and deterministic control, the STM32 family with NPUs is attractive, but it's a heavier development path than ESP32 + Teensy for hobby/prototype projects. Consider this if you plan production and want a single-chip solution. STMicroelectronics - STM32 AI+1

Practical parts / boards to consider

Vision MCU (ESP32-S3):

Espressif ESP32-S3 dev kits / modules with 8MB PSRAM + OV2640 camera connectors (look for "Sense" or camera dev kits). Good for buffering frames and running ESP-NN. Espressif+1

Control MCU (Teensy 4.1):

Teensy 4.1 (i.MX RT1062) — lots of PWM channels, high CPU for control loops, fast USB/host if you need logging or a microSD. PJRC+1

Low-cost alternative (RP2040):

Raspberry Pi Pico / Pico W (RP2040) — dual core for splitting tasks, many PWM channels, PIO for custom I/O. Use Pico W if you want Wi-Fi on the control board. Raspberry Pi+1

Engineering notes / gotchas (important)

Separate power for servos — always power servos from a dedicated supply (not the MCU 3.3V USB rail). Use a common ground. Servos produce noise and brownout risk. (This is essential for reliability.)

Use PSRAM on the vision MCU if you want to buffer images or run slightly larger models — boards without PSRAM will be memory constrained. Adafruit

Inter-MCU comms: use a simple high-speed link (UART, SPI, or I²C) between vision and control nodes. Let the vision node send concise steering/target data (centroid x/y, distance, confidence); keep heavy data on the vision node.

Real-time tasks stay on the control MCU. Don't let Wi-Fi or ML inference block your servo loops — that's the whole point of the split.

Timing and interrupts: pick a control MCU with enough timers and low interrupt latency (Teensy/RP2040 are excellent here).

Final recommmendation (one-line)

If you want the easiest, most reliable path for your described workload: ESP32-S3 (vision + ML + HuskyLens/OV2640) + Teensy 4.1 (real-time servo & peripherals). That combo

gives you on-device tiny-ML acceleration, camera support and plenty of deterministic control horsepower for servos and peripherals. Espressif+1"

[1c] ***Case Study 1 GoogleAI Outcome:*** *"For a workload involving real-time servo control, light machine learning for blob detection, and camera management, a two-MCU architecture provides the best balance of deterministic performance and processing power.*

The recommended setup is an Espressif ESP32-S3 for vision/ML tasks and a Teensy 4.1 (or Raspberry Pi Pico W as a lower-cost alternative) for real-time motor control.

Why a Two-MCU Architecture?

Splitting the workload is crucial for reliability. Machine learning inference and camera frame buffering require significant, non-deterministic processing time and memory, which can interfere with the precise, low-jitter timing needed for servo motor control.

Vision/ML Node: Handles camera frames, runs lightweight neural networks/blob detection, and communicates higher-level commands (e.g., target coordinates) to the control node.

Control Node: Focuses solely on executing real-time control loops, managing PWM signals for servos, and handling immediate peripheral I/O without interruption.

Recommended MCU Choices

Role , Recommended MCU, Key Features & Rationale

Vision & ML, Espressif ESP32-S3, Includes vector instruction support (ESP-NN, ESP-DSP libraries) for hardware-accelerated tiny ML inference. Widely supported development boards include PSRAM for buffering camera frames (e.g., OV2640 sensors) and built-in Wi-Fi/BLE for telemetry. It easily interfaces with AI vision sensors like the HuskyLens via UART/I²C.

Real-Time Control, Teensy 4.1, The NXP i.MX RT (600 MHz class) processor provides highly deterministic, high-performance control loops. It has abundant timers, low interrupt latency, and many reliable hardware PWM channels essential for smooth, stable servo control (e.g., PID loops).

Control (Budget), Raspberry Pi Pico W, A strong alternative to the Teensy 4.1. Its dual-core ARM Cortex M0+ allows for dedicated handling of I/O tasks. The unique Programmable I/O (PIO) state machines are excellent for custom peripheral interfaces, and the "W" variant adds Wi-Fi connectivity.

Summary of the Best Path

This combination provides powerful, on-device tiny-ML acceleration and camera support on the ESP32-S3, paired with the necessary deterministic horsepower and I/O for real-

time control on the Teensy 4.1 or Pico W. This separation ensures reliability and performance for all parts of the workload.

AI can make mistakes, so double-check responses

What libraries or frameworks are recommended"

[2a] ***Case Study 2 Prompt:*** *"What stationary cameras are compatible with multiple MCUs that will be able to connect via a standard communication protocol and whose image data is able to be processed through a tinyML model."*

[2b] ***Case Study 2 ChatGPT Outcome:*** *"Great question — picking the camera is as important as picking the MCU for a tiny-ML pipeline. Below I list camera modules that are widely used with microcontrollers (not full SBCs), the standard hardware interface they expose (SPI, parallel/DVP, MIPI-CSI, etc.), and whether they're a good fit for tinyML on MCUs (memory/throughput/power tradeoffs). I also give quick, practical recommendations for which camera to pick depending on your MCU (ESP32-S3, RP2040 / Pico, STM32, K210 / Sipeed, etc.).*

Short summary / top picks

Arducam SPI camera family — best universal choice for "works with almost any MCU (SPI + I²C)". Good for RP2040, STM32, ESP32 variants when you don't have a parallel DCMI interface. Arducam Documentation+1

OV2640 / ESP32-CAM — easy plug-and-play on ESP32 boards (DVP interface supported by ESP32 camera drivers). Good for low-to-mid resolution tinyML on Espressif chips. Elecrow+1

Himax HM01B0 — ultra low-power monochrome QVGA sensor designed for always-on CV / tinyML tasks (works with MCU host via parallel interface). Excellent for low-power, low-resolution ML. himax.com.tw+1

OV5640 / OV5642 (5MP) and OV5640 + DCMI — higher resolution for STM32 boards that support DCMI (parallel) or via Arducam SPI adapters. Use if you need higher image detail but expect more memory/processing cost. Arducam Blog+1

MT9V034 / MT9V022 (global shutter) — grayscale, high-fps, global-shutter modules used in embedded vision; often paired with OpenMV/MCUs for motion/robotics. Good for fast moving scenes. OpenMV+1

Camera types, interfaces, and MCU compatibility (concise)

1) SPI camera modules (Arducam SPI series)

Interface: SPI for pixel data + I²C for camera registers (many modules provide a simplified interface).

Why use them: Work with any MCU that has SPI (RP2040, STM32, AVR, ESP32, etc.). Arducam provides libraries and "Mega-SPI" / mini modules specifically targeted to microcontrollers. Great when your MCU lacks a parallel DCMI interface or when you want simpler wiring.

tinyML suitability: Good for QVGA/2–3MP, can stream small frames or snapshots for on-device inference. Performance depends on SPI clock and MCU CPU. Use when frame rate can be modest. Arducam Documentation+1

2) DVP / Parallel cameras (OV2640, OV5640, OV5642, GC0328, OV7670)

Interface: Parallel DVP (commonly called "DCMI" on STM32) — wide 8/10/12-bit bus.

Why use them: ESP32 (ESP32-CAM / ESP32-S3 camera devkits) and many STM32 MCUs support directly attaching these sensors for higher throughput than SPI. Many tinyML demos use OV2640 or OV7670.

tinyML suitability: Good for real-time tinyML when paired with a capable MCU (ESP32-S3 with PSRAM, STM32H7 series, etc.). Memory requirements grow quickly with resolution. Elecrow+1

3) Low-power vision sensors (Himax HM01B0, GC0328 etc.)

Interface: Parallel or simple serial control; many optimized for low power and grayscale output.

Why use them: Designed for "always on" CV and very power efficient — ideal for event detection, gesture, or centroid/blob extraction on constrained MCUs.

tinyML suitability: Excellent for tinyML tasks that accept low resolution (QVGA or less) and grayscale input — lower preprocessing and model size. himax.com.tw+1

4) Global-shutter / machine-vision sensors (MT9V034, MT9V022)

Interface: Parallel (and often easy to integrate with OpenMV or STM32 DCMI).

Why use them: If motion blur is a problem (fast robots / conveyors), global shutter is preferred. Often used with OpenMV cameras (which are MCU-based and expose simple APIs).

tinyML suitability: Great for robotics vision where you need crisp frames at high fps; models still must be small for MCU inference. OpenMV+1

Practical matchups: which camera for which MCU

ESP32 / ESP32-S3 (with PSRAM)

Use OV2640 (ESP32-CAM) or ESP32-S3 camera devkits; or Arducam DVP modules. Good for on-device tinyML (with PSRAM for frame buffers). Elecrow+1

RP2040 (Pico / Pico W)

Best with Arducam SPI cameras (Arducam supports Pico and has plug-n-play kits). RP2040 lacks a standard DCMI peripheral, so SPI is easiest. For more advanced AI you can use RP2040 paired with a KPU board (e.g., Maix/RISC-V K210). Arducam+1

STM32 (F4 / H7 / M7 with DCMI)

Use OV5640/OV5642/OV3640 via STM32 DCMI for higher resolution; or Arducam SPI if you prefer SPI approach. STM32 can be a solid single-MCU option if you pick a model with enough RAM and DMA support. Arducam Blog+1

Kendryte K210 / Sipeed boards

Many K210 dev boards are sold with GC0328 / OV2640 and are aimed at tinyML — these boards include a neural accelerator (KPU) to run models much faster than general MCU cores. Good option when you want small NN inference without extra hardware. DFRobot Electronics

OpenMV / MicroPython camera boards

OpenMV Cam + MT9V034 / OV7725 modules are MCU-level solutions with existing CV pipeline and are easy to prototype with if you want immediate blob detection capabilities. OpenMV

Important engineering notes (don't skip)

Interface matters more than sensor: pick a camera that your MCU can physically talk to (SPI vs DVP vs MIPI/CSI). Many MCUs don't have MIPI-CSI — that interface is typical for SBCs (Raspberry Pi) not bare MCUs. Arducam Blog

Memory & PSRAM: tinyML image models need memory for frame buffers — boards with PSRAM (ESP32-S3 devkits) or external RAM are much easier to work with. Elecrow

Preprocess on camera vs host: some camera modules let you downsample / crop / produce grayscale on the sensor or via Arducam's hardware — reduce MCU load by doing as much preprocessing close to the sensor as possible. Arducam Documentation

Power & size: low-power sensors like Himax HM01B0 are great for battery-powered always-on applications. himax.com.tw"

[2c] ***Case Study 2 Claude AI Outcome:*** *"Based on my research, here are the key stationary camera options compatible with multiple MCUs for tinyML applications:*

Popular Camera Sensors

OV2640 (2MP)

Features hardware JPEG encoding that reduces a typical 1600×1200 image from 3.66MB to around 150KB, making it ideal for MCU processing Arducam

Uses I2C (SCCB interface) for configuration and DVP (Digital Video Port) for image data transfer Uctronics

Compatible with platforms including ESP32, STM32, Arduino, and ARM-based MCUs Seeed Studio

Available with SPI interface through Arducam modules, supporting 4-8Mbps transfer speeds Arducam

OV5640/OV5642 (5MP)

5MP sensors with on-chip ISP and JPEG compression capabilities Arducam

Similar interface protocols to OV2640

OV7670 (VGA)

Lower resolution alternative with similar MCU compatibility

Uses I2C and parallel interface, starts outputting video immediately after power-up OurPCB

Standard Communication Protocols

The key protocols used are:

I2C/SCCB - For sensor configuration and register settings

SPI - For image data transfer (when using modules like Arducam)

DVP (Digital Video Port) - Standard parallel interface for direct MCU connection

UART - Less common for camera data but supported by some modules

TinyML-Ready Platforms

Popular development boards for tinyML camera applications include the XIAO ESP32S3 Sense with OV2640 camera and the Arduino Nicla Vision Seeed Studio. These platforms

support TensorFlow Lite and Edge Impulse frameworks for deploying ML models Seeed Studio.

The OV2640 is particularly well-suited for tinyML because its on-chip JPEG encoder significantly reduces the memory footprint, allowing resource-constrained MCUs to process image data effectively."

[3a] ***Case Study 3 Prompt:*** *"for a 2 MCU architecture using ESP32-S3, how are the tasks split between both MCUs that will allow us to meet specific time requirements"*

[3b] ***Case Study 3 ChatGPT Outcome:*** *"MCU A — ESP32-S3 (Vision/AI MCU)*

MCU B — Real-Time Control MCU (Teensy 4.1 / RP2040 / STM32, etc.)

This architecture ensures you meet tight timing requirements (servo update rates, camera processing deadlines, ML inference frame rate, communication deadlines) by cleanly separating non-deterministic tasks (vision, Wi-Fi, ML) from deterministic tasks (PWM, control loops, sensors).

☑ *High-level Split*

ESP32-S3 ("Vision Brain")

Handles:

Camera capture (OV2640 or similar)

Frame preprocessing (resize, grayscale, cropping)

TinyML inference (blob detection, centroid, classification)

AI sensor handling (HuskyLens / I2C / UART)

Feature extraction (x, y centroid, bounding box, confidence)

Non-real-time tasks (Wi-Fi, Bluetooth, logging, UI)

Communication to Control MCU (UART or SPI)

NOT ALLOWED on ESP32-S3:

Servo PWM

Hard real-time loops

Timing-critical processes
(ESP-IDF tasks + Wi-Fi stack add jitter.)

Control MCU ("Motion + Timing MCU")

Handles:

Servo PWM generation (20–200 Hz, low jitter)

PID control loops (100–200 Hz typical)

Position calculation

Motor/servo smoothing + stability

Debounced digital inputs

Safety states (stop, limit, watchdog)

LED/Buzzer/peripheral control

High-precision timers + interrupts

NOT ALLOWED on Control MCU:

Image capture

TinyML inference

Wi-Fi / BLE stacks

Any blocking communication

🔧 *Why split this way?*

Because:

1. Vision tasks are unpredictable (variable time)

Frame capture + inference on ESP32-S3 can vary from 20–70 ms depending on image size.
This would destroy servo timing if run on the same MCU.

2. Servo and PID timing must be deterministic

Servo update = 20 ms (minimum)
Motor PID loop = 5–10 ms

If you miss these deadlines → jitter, instability, oscillation.

3. ESP32-S3 has Wi-Fi/BLE stack → interrupts → jitter

Even disabled, internal tasks still run.

4. Control MCUs (Teensy, STM32, RP2040) have predictable timers

Perfect for servo PWMs and fast control loops.

📡 Data Flow Between the MCUs

Use SPI (best), or UART (simple) for inter-MCU messaging.

Vision MCU → Control MCU (Every frame, ~20–60 ms)

Send compact data only:

```
{
  target_x: 0–1 normalized,
  target_y: 0–1 normalized,
  target_size: 0–1,
  confidence: 0–1,
  tracking_state: TRACKING/LOST
}
```

Payload size: < 16 bytes
Latency: < 1 ms (UART), < 0.2 ms (SPI)

Control MCU → Vision MCU

Send commands:

```
{
  mode: IDLE / TRACK / SCAN,
  request_frame: YES/NO,
  servo_states,
  error_flags
}
```

⏱ Meets Time Requirements Because…

ESP32-S3 timing (non-deterministic tasks):

Task, Typical Time

Capture & preprocess QVGA frame, 8–15 ms

TinyML inference, 5–25 ms

Serial transmission to MCU B, <1 ms

Wi-Fi/BLE events, unpredictable

Not used for real-time.

Control MCU timing (deterministic tasks):

Task, Rate, Notes

Servo PWM generation, 20 ms, Hardware timer-controlled

PID loop, 5–10 ms, Never late

Position smoothing, every loop,

Emergency stop, <1 ms, ISR

Communication read, <500 µs, Non-blocking

This guarantees the servos always update on time, regardless of camera/AI load.

🩹 *Concrete Example Timeline (Real System)*

ESP32-S3:

t=0 ms → capture frame

t=10 ms → preprocess

t=20 ms → run tinyML model

t=30 ms → extract centroid

t=31 ms → send to Control MCU

(next frame begins)

Control MCU:

t=0 → PID loop iteration

t=5 → update servo PWM

t=10 → receive new centroid (asynchronous)

t=15 → update target position

t=20 → next servo frame

t=25 → next PID loop

(never waits for ESP32-S3)

The Control MCU never depends on ESP32-S3 timing → stability guaranteed.

🎯 *Final Recommended Split (Summary)*

ESP32-S3 (Vision/ML)

Camera capture

Image preprocessing

TinyML inference

HuskyLens / peripheral vision

Object detection / centroid / confidence

Wi-Fi/Bluetooth

Transmits small data packets

High-level state machine (track, search, idle)

Control MCU (Real-Time Control)

PID control for Pan/Tilt

Servo PWM signals

Limit switches / input interrupts

LED/Buzzer

Safety watchdog

Receives vision data & reacts immediately

Smooth trajectory generation

No blocking tasks

This architecture is robust enough for robotics, gimbals, AI tracking, UAV pointing systems, and target-tracking pan-tilt platforms."

[3c] ***Case Study 3 GoogleAI Outcome:*** *"In a two-MCU architecture using an ESP32-S3 and a control-oriented MCU (like the*

Teensy 4.1

or

Raspberry Pi Pico W

), tasks are split to ensure deterministic, low-latency performance for motor control by isolating it from the variable timing of vision processing and wireless communication.

Task Split and Communication

The goal is for the ESP32-S3 to act as the "eyes" and "brain" for high-level decisions, and the control MCU to act as the "muscles" for precise, real-time movements. A high-speed, simple communication link is used between them.

MCU, Primary Tasks, Secondary/Peripheral Tasks, Time Requirement Focus

*Vision MCU (ESP32-S3), * Camera frame acquisition (from two cameras), * Wi-Fi/BLE communication (telemetry), Variable latency (acceptable up to ~100ms for visual feedback/updates)*

** Running lightweight ML models (blob detection, object tracking), * Interfacing with the main AI vision sensor (HuskyLens) via UART/I²C,*

** Processing high-level commands and generating target data (e.g., target X/Y, distance, confidence), * General network communication,*

*Control MCU (Teensy 4.1 or Pico W), * Real-time control loops, * Controlling LEDs/buzzers (simple I/O), Deterministic, sub-millisecond timing for servo control (low jitter PWM)*

** Generating low-jitter PWM signals for two servo motors, * Reading any non-critical sensors,*

** Handling high-priority interrupts, ,*

** Receiving concise command data from the Vision MCU (e.g., "move to X angle at speed Y"), ,*

Meeting Specific Time Requirements

Minimizing Control Latency: The control MCU runs a tight, dedicated loop for servo control, often on a specific core if it is a dual-core device (like the

Pico W

), or using a high-speed, interrupt-driven architecture (Teensy 4.1). No time-variable tasks (like image processing or network stacks) can block this loop.

Managing Vision Latency: Vision processing inherently has variable latency depending on image content or model complexity. The ESP32-S3 uses its hardware acceleration features (vector instructions) and PSRAM to process this as quickly as possible. The key is that this latency does not impact motor stability.

Effective Communication: Use a simple, high-speed communication protocol like UART, SPI, or I²C to send concise, processed data (e.g., just the final X/Y coordinate and confidence score) between the two MCUs. This minimizes inter-MCU communication overhead.

Practical Engineering Note

Crucially, all servos must be powered by a dedicated power supply, with a common ground connected to both MCUs, to avoid electrical noise and brownout issues that disrupt real-time stability.

These guides illustrate inter-MCU communication strategies and servo power considerations for your ESP32-S3 dual-core project:"

[4a] ***Case Study 4 Prompt:*** *"Given that I am using an ESP32, what is the most feasible to do object detection on objects of a certain color: YOLO, Faster-RCNN, or threshold blob-detection. Give me the reasons to use each, as well as why they may or may not be possible"*

[4b] ***Case Study 4 ChatGPT Outcome:*** *"Great question — this is a key design decision when doing computer vision on an ESP32, which is a low-power microcontroller (not a high-performance processor like a Raspberry Pi or Jetson). Let's compare YOLO, Faster R-CNN, and color-based blob detection (thresholding) in that context:*

⚙ *1. YOLO (You Only Look Once)*

What it is:
A deep-learning–based object detector that uses a convolutional neural network (CNN) to detect multiple object types in real-time.

✅ *Why You'd Use It:*

Very accurate for general-purpose object detection.

Can detect multiple classes and handle complex scenes.

Pretrained models are available (YOLOv3, YOLOv5, etc.).

❌ *Why It's NOT Feasible on an ESP32:*

Requires high compute power (GPU or at least a multi-core CPU with hundreds of MB of RAM).

The ESP32 only has 320–520 KB RAM and a dual-core Tensilica LX6 CPU, which is orders of magnitude too small.

Even tiny YOLO models (like YOLOv5n or YOLOv3-tiny) need tens of MB of RAM and hundreds of MFLOPS, far beyond what ESP32 can handle.

✅ *Possible workaround:*

Use an external co-processor (e.g., ESP32 + EdgeTPU or ESP32 + Raspberry Pi) where the ESP32 sends frames or triggers detection handled by the other device.

Use TinyML models quantized for ESP32-S3 with AI acceleration, but that's only for very small neural nets, not full YOLO.

👉 *Verdict: Not feasible on its own on ESP32.*

⚙️ *2. Faster R-CNN*

What it is:
A two-stage CNN object detector (Region Proposal + Classification) — slower but often more accurate than YOLO.

✅ *Why You'd Use It:*

Great for research or when accuracy is critical.

Can precisely locate and classify objects in cluttered images.

❌ *Why It's NOT Feasible on ESP32:*

Heavier than YOLO. Needs hundreds of MB of memory and a GPU to run in reasonable time.

Even desktop CPUs struggle to run it in real-time.

Cannot be optimized for microcontrollers easily.

✅ *Possible workaround:*
Same as above — offload processing to a server, and use ESP32 as a camera streamer (send frames via WiFi/Bluetooth to a machine running Faster R-CNN).

👉 *Verdict: Completely infeasible on ESP32.*

⚙️ *3. Color-based Thresholding / Blob Detection*

What it is:
A classic computer vision technique that filters image pixels by color range (HSV/RGB thresholds) and finds "blobs" (regions) of that color.

✅ *Why You'd Use It:*

Lightweight and fast — can easily run on ESP32 at several frames per second.

No neural network required — just math on pixel values.

Perfect for detecting objects of a specific color (e.g., red ball, blue marker, yellow cube).

Can be implemented even with low-res images (e.g., 160×120).

Works well with ArduCAM or OV2640 sensors connected via SPI or I2C.

❌ *Limitations:*

Only works for distinct color targets.

Sensitive to lighting changes, reflections, shadows, and color similarity.

Can't distinguish between different shapes or textures of the same color.

✅ *Optimization options:*

Convert image to HSV color space (better color segmentation).

Use bounding box around the largest blob.

Implement simple motion or size filters to track specific objects.

👉 *Verdict:*
✅ *Most feasible and reliable option on ESP32 for color-based object detection.*
Feasible in real time, uses almost no RAM, and doesn't need external hardware.

🪀 *Summary Table*

Method, Feasible on ESP32?, Speed, Accuracy, Complexity, Best For

YOLO, ❌ No, Slow (unusable), High, Very high, External processing (server)

Faster R-CNN, ❌ No, Very slow, Very high, Extremely high, Research on GPU

Color Blob Detection, ✅ Yes, Real-time, Low–Medium, Very low, Tracking colored objects

💡 *Final Recommendation*

If your goal is detecting or tracking an object of a certain color (e.g., a red ball or green tag), use color threshold + blob detection.
You can run it entirely on the ESP32-CAM or ESP32-S3, possibly with OpenMV or a lightweight MicroPython script.

If you later want to expand to shape or object category detection, you can:

Stream frames to a server or Raspberry Pi running YOLO/Faster R-CNN.

Keep ESP32 as the low-level vision node and use the server for higher-level analysis."

[4c] ***Case Study 4 Copilot Outcome:*** *"Great question — let's break this down carefully, because the ESP32 is a very resource-constrained microcontroller (low RAM, limited CPU, no GPU). That makes the choice of algorithm critical.*

⚡ *Option 1: YOLO (You Only Look Once)*

Why use it:

State-of-the-art real-time object detection.

Can detect multiple classes and complex shapes.

Widely supported in embedded AI frameworks (TinyML, TensorFlow Lite Micro).

Why it may not be possible on ESP32:

YOLO models are large and computationally heavy. Even the smallest YOLO variants (YOLOv5n, YOLOv8n) require tens of MBs of memory and floating-point acceleration.

ESP32 typically has < 520 KB RAM and no GPU, so running YOLO directly is infeasible.

You'd need external hardware (e.g., ESP32 + AI accelerator like Kendryte K210 or Coral TPU).

✅ *Verdict: Not feasible standalone on ESP32. Possible only if paired with external accelerators or if you offload inference to a server.*

⚡ *Option 2: Faster-RCNN*

Why use it:

Very accurate for object detection, especially with complex scenes.

Good for research or high-precision tasks.

Why it may not be possible on ESP32:

Even heavier than YOLO. Faster-RCNN uses a two-stage pipeline (region proposals + classification), which is computationally expensive.

Requires large memory and compute resources (GPU/desktop-class CPU).

No realistic way to run this on ESP32, even with quantization or pruning.

✅ *Verdict: Impossible on ESP32. This is strictly a desktop/server-class algorithm.*

⚡ *Option 3: Threshold Blob-Detection (Color-based segmentation)*

Why use it:

Extremely lightweight — just compares pixel values against a color threshold.

Can be implemented with simple math (RGB/HSV thresholding + connected component labeling).

Fits perfectly within ESP32's constraints.

Real-time performance possible even at modest resolutions (e.g., 160×120).

Why it may not be possible (limitations):

Only works for objects with distinct colors.

Sensitive to lighting changes, shadows, and camera quality.

Cannot distinguish between objects of the same color.

✅ *Verdict: Most feasible and practical on ESP32. Ideal for detecting colored markers, balls, or simple objects.*

🎯 *Final Recommendation*

For an ESP32, threshold blob-detection is the only realistic option if you want on-device detection. YOLO and Faster-RCNN are far too heavy without external accelerators.

👉 *If your task is simply "detect objects of a certain color," then blob-detection is not only feasible but optimal. If you need more complex detection (shapes, multiple classes), you'd need to either:*

Use a more powerful board (e.g., Raspberry Pi, Jetson Nano).

Or offload detection to a server/cloud and just stream images from ESP32."