Sentry Tracking Attack Turret a.k.a S.T.A.T. Project

Nicholas Martucci, Carlos Garcia, Devak Sharm, and Christian Solanet

University of Central Florida, Department of Computer and Electrical Engineering, Orlando, Florida, 32816, U.S.A.

Abstract — S.T.A.T is a 3D printed turret controlled by servos, motors, sensors and custom-made PCBs. It takes information from sentry cameras using blob detection to identify a target. It will then send these captures to an access box near the user. The box will display signal LEDs as well as a buzzer depending on certain status. It will then transfer said data to the main turret PCB which will wake the Huskey lens to initiate lock-on to the target ahead. From there servos, motors, and a solenoid will activate to take out the pre-determined object which has been uploaded to our Huskey Lens sensor from our own generated database.

Index Terms — Embedded systems, Object detection, Computer vision, Servo control, PID control, Machine learning, Real-time systems, Microcontrollers, Sensors, Autonomous systems

I. Introduction

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.

Our project is not a first of its kind. Many current technologies are present both in military as well as other school projects. Take the C-RAM for example, which stands for Counter Rocket Artillery and Missile system. This system along with many others that are similar, focus on using cameras and sensors to know when a foreign target has entered friendly air space and will use live ammunition or lasers to take out this threat before any harm can be done to the user.

Some senior design Projects that inspired us was a past project called Traffic Stop Watchdog. This project used a camera mounted on a movable base that would scan a room and use image detection to determine if an officer was present in a room and continuously track said officer. Our sentry cameras and Huskey Lens sensor take inspiration from this project as well as the movable base. The addition we have added is being able to shoot at a target rather than track a friendly.

II. System Overview

The S.T.A.T is built around three main subsystems that work together to accomplish this. 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. 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.

The second subsystem 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. The third 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.

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 architecture (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.

III. Hardware Subsystems

Each subsystem is vital to making our project operate. The selection of an MCU required extra research due to the extra memory and ram that was needed from this project. A specific tracking sensor was needed and took a great amount of time to pick a correct and working one. Power distribution from access box to turret took time to consider due to the needed voltage and current rating for all of our components.

A. Power Distribution

Our system will be supplied by a 600W power bank that will feed directly into the access box. This power bank takes in 120VDC from a wall plug in and steps it down to 12V. From here it is fed directly into the access box and is controlled by a variety of switches and buttons in case something happens that is not expected. 12V will feed into a 5V regulator and a 3.3V regulator located on the access board and we will directly feed 12V into the turret PCB. The voltage for the turret PCB will be stepped down into 7V, 5V, and 3.3V for many of the functions that PCB must handle. The reason for stepping down the voltage at the turret PCB rather then the access box then feeding into the turret was due to loss and wiring connections. Wiring is already something that needs to be addressed due to the motors and servos already present in the system. The system also already has lots of power draw so to help minimize all of the listed issues, we will have one wire feeding 12V into the turret PCB from the access box and make sure proper 3D printed housing stations are done to ensure minimal wire exposure.

B. Access Box

The use of our access box is meant to act as the brain of our system, transporting data, 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 two 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.

C. MCU Operations

The ESP32-S3-WROOM-2 was chosen as the MCU for this project. 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. This is why we decided to specifically go with the ESP32-S3-WROOM-2-N32R16V due to its sufficient storage capacity and RAM size and left the harder computations to the Huskey Lens for harder, lock-on detection algorithms.

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.

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.

D. Sentry Cameras

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 structs are 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. 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.

F.. 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 as twisted pairs with small series resistors at the MCU pins to reduce ringing. The ESCs are configured with the LED program card: Brake OFF, Soft/Medium Start, Timing Low, Governor OFF, and LVC disabled for PSU operation. These settings minimize inrush, avoid regenerative braking, and keep throttle predictable.

Electrical hygiene focuses on keeping motor current loops local to the ESCs. Each ESC branch includes a 20 to 25A slow blow fuse, and the wiring from ESC to motor uses a compact three conductor connector with locking retention. The system runs both motors simultaneously. The chosen PSU provides plenty of headroom to handle the combined startup current without issues.

H. 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 30T:90T to multiply torque and hold angle against gravity with reduced buzz. Both servos run from the dedicated 7.0 to 7.2V rail and are commanded directly from the turret MCU's timer PWM outputs. To reduce steady current and oscillation, motion profiles use gentle acceleration/deceleration, and the tilt axis is mechanically balanced about its pivot. Cabling to the moving head is dressed with a service loop and kept clear of flywheels.

The servo power rail is separate from the logic rail for a good reason. When servos move under load, they can draw several amps in short bursts, which would cause significant voltage drop on a shared 5V rail. That drop could reset the microcontroller or cause cameras and sensors to behave erratically. By giving the servos their own regulated 7V rail, we isolate their current spikes from the sensitive digital circuits.

The 3:1 gearing on the tilt axis was chosen after testing a few different ratios. Direct drive didn't have enough torque to hold the launcher assembly steady, especially at shallow elevation angles where gravity creates maximum moment. The 3:1 gives enough mechanical advantage that the servo can hold position without constantly drawing current to fight gravity.

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

I. HuskeyLens AI Sensor

The HuskyLens AI Machine Vision Sensor is designed for computer vision applications for embedded systems. It comes with built-in face recognition, object classification, color detection, line tracking, and object following. The camera also features a 2-inch IPS display that provides real-time visuals from the camera, which is highly useful for debugging and demonstrations. The HuskyLens comes with support for UART and I2C communications, which can be easily configured using the cameras for custom GUI. The camera also comes with the ability to train models directly on the device, simplifying the process for learning custom objects by removing external hardware and datasets.

The HuskyLens can be expanded to creating your own model and uploading it to the device. This is useful for our project since using color recognition on a general setting could be glitchy depending on the time of day and shades of color we are testing. Uploading a database allows us to choose a specific object and have consistent results across multiple tests and cases, allowing the upscale and use of this project to be expanded.

IV. PCB Design

A. Regulator

The regulator design was meant to be a plug and play approach to help when it comes to the cost of ordering. We will only be using two regulators for each rail and due to minimum order numbers from websites like JLCPCB, we decided to be able to change the output of our regulators by changing the resistor on the voltage divider rather than needing to make a whole new design itself. Using the LM2679T-ADJ achieves this. Keeping most of the design the same, except for R2 from (1)

$$\text{Vout} = 1.21 * (1 + \frac{R2}{R1}) \quad (1)$$

and some output capacitors, we can safely order a big quantity of boards with plenty of extra to spare in case we need to make more. From Figure 1 seen below we would change the 1.74k to the respective resistor to have a proper output for each rail. We would also change the capacitors on the bottom of the board depending on the rail as well.

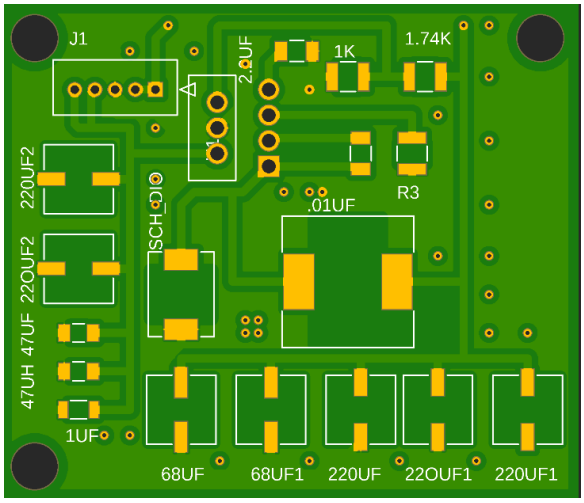

Figure 1. PCB regulator showing layout and where to change components

B. Access Box Design

The Access Box has many functions and distributes most of the power to the turret PCB. It has a total of 3 power switches and one emergency stop button. One switch is for main power and one for each regulator in case we quickly want to shut off a regulator rather than the whole system. The emergency stop button is meant to stop the motors in case they start to go in an expected way. We have the MCU circuit which will utilize LEDs and a buzzer circuit to help with user interaction and show the different status of the turret in real time such as when lock-on occurs or when an enemy is with in range of detection from the sentry cameras. It also houses all connections for the century cameras including the 5V, GND, and the SPI signals. Cameras and regulators will be connected via connectors rather than mounted onto the board to help with cost. It is easier to replace parts rather than to replace the entire board if anything were to go wrong.

C. Turret PCB Design

The turret side control PCB is built around an module and handles all the low voltage I/O for sensors and actuators at the head. The module runs from a local 3.3V regulator and uses standard boot control with 10 kΩ pull-ups on both EN and GPIO0. We added momentary RESET and BOOT buttons so you can drop into the ROM downloader when you need to flash new firmware.

For sensing, everything is shared on a 3.3V I2C bus. The HuskyLens vision module connects to SDA and SCL while getting its 5V on the power pin. The IMU ties into the same bus with CS strapped high for I²C mode, and SA0 sets the address. Keeping sensors on one bus simplifies the wiring and lets us add more devices later without running out of GPIO pins.

The actuation breakouts are arranged as dedicated labeled headers. The two ESC throttle ports each get an MCU PWM signal that runs through about 220 Ω series resistance, and there's a 10 kΩ pulldown at the connector to guarantee the motors stay idle on boot. The pan and tilt servos follow the same signal treatment and pull their power from the external servo rail. The laser module gets input driven through 1 kΩ series with a 10 kΩ pulldown to keep it off by default.

For coordination between boards, we added a MASTER link that carries TX, RX, and GND across a short harness so the access box controller can exchange commands and status with the turret MCU. This keeps the system modular and lets us test each board independently.

D. Relay Power Control PCB

For the relay driver stage, a small logic-level N-MOSFET is used on the low side of the coil. These parts 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-rail voltage. This allows the microcontroller to switch the relay coil reliably without stressing the GPIO pins. Each relay coil has its own clamp diode, either a 1N4007 or MMSD4148. These diodes are connected directly across the coil and conduct when the coil is switched off, handling the energy from the collapsing magnetic field and preventing high-voltage spikes across the MOSFET.

A unidirectional SMBJ18A TVS diode is placed on the 12 V control rail at the access board connector. This device is selected for 12 V systems and clamps any larger transients on the control rail, protecting the coil driver, indicator LED, and other components tied to that node. The MOSFET gate network uses a 100 Ω series resistor and a 100 kΩ pulldown. The series resistor limits the rush of current into the gate and damps any ringing on fast edges while the pulldown keeps the MOSFET turned off when the microcontroller is in reset or the pin is floating, preventing the relay from turning on by accident during power-up.

V. Software Design

A. Primary MCU: Turret Controller

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.).

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 maintain 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 in parallel while maintaining the stability of the system and operating in real time consistently.

B. Secondary MCU: Image Processing

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 TinyML (Tiny Machine Learning) models that are 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.

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 TinyML models are memory 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.

C. Inter-MCU Communication

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.

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.

D. Blob Detection Algorithm

The blob detection algorithm as defined earlier will go row by row, updating a global registry of all blobs as it passes through the image. There is a static table that tracks all blobs in the image, assigning them a unique integer label. Before assigning the label, a row must first run through the threshold function. Then the pixels that passed the threshold will be compared to the previous row's pixels. Once blobs have been assigned to each necessary pixel, any merge conflicts will be resolved. Then the previous row will be set to the current row and the process can continue again.

After each red pixel has been identified, the algorithm will loop through the current encoded row and compare it to the previous. The previous row will not be made up of 1s and 0s, but it will be made up of integer labels and 0s. These integer labels correspond to that pixel to a blob. The first step is to check if the current pixel is 1, if so, check if the adjacent pixels in previous row have a label. If they do, then set the

current pixel to that label. If no neighbors have a label, then create a new label and assign it to the current pixel. Then continue with the next pixel and repeat. Each time a pixel is added to a blob, the blob updates its size.

If two blobs are adjacent, then they will need to be merged. This can be accomplished by imagining that each blob is a set. When two sets need to be merged, a union operation can be performed. This operation chooses a set to become the parent of the other set and then updates its size accordingly. This way, when two blobs are combined, the labels don't need to be updated, just the parent of the blob needs to be updated. After the full image is processed, only the parent blobs need to be accounted for as they will contain the updated dimensions.

Lastly, the previous row of labeled pixels needs to be updated to be the current row. The next buffer can then come in and repeat the process.

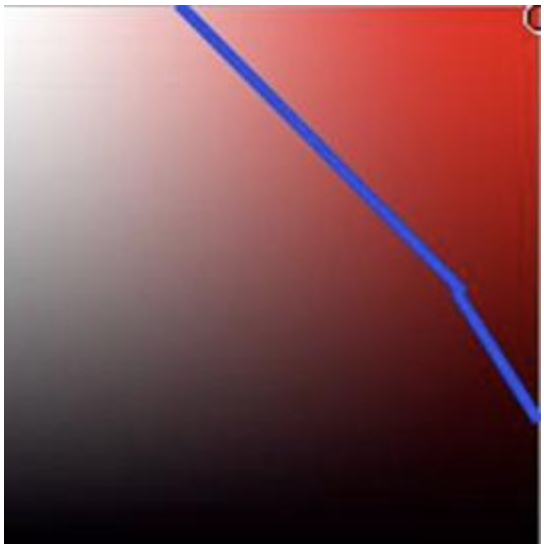

Figure 2. RGB Classification Graph for Red

E. 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 the HuskyLens and IMU, motion control algorithms, servo commands, and communication with the MCU.

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, an Integral (I) term that removes the steady state offset, a Derivative (D) term to damp oscillations, anti-windup limits stopping integral runaway, output clamping that stops motors from exceeding mechanical limits, and rate limits to ensure smooth transitions at high speeds.

The software will convert positional errors into servo angles by calculating the pixel offset where the target coordinate is the target, normalizing the offset by mapping the pixel errors on a scale based on the camera's resolution, and converting positional errors to angular errors using the camera's field of view. Each axis will have its own PID loop and the controller outputs will be mapped to pulse widths with a range of about 500 to 2500 microseconds.

F. Safety Measures and Fault Handling

The software includes several safety measures to ensure reliable operation. Bound checking prevents servo motors from exceeding their mechanical limits, with the pan servo operating between 0 and 120 degrees while the tilt operates between 0 and 90 degrees. Stall detection is included in case of failed updates to servo motors after repeating commands. Loss-of-target mode will return the mount to a neutral position if the HuskyLens loses sight of the target. Low-voltage protection is also implemented in case of voltage drops under heavy load.

VI. Testing

A. Sentry 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.

Blob Detection so far has been a success of up to 22 feet, which can be seen from the testing that has been performed. We have not expanded to other colors yet such as green and blue since we are first trying to get a base line test and work on communication between both PCBs in conjunction with the servos and motors. As seen from Figure 3, this image shows that the camera can take a picture and identify the red color in the room. During testing we have uploaded a model to the HuskyLens that can only detect a red ballon but as

for blob detection it will be able to detect on a slightly larger scale.

Figure 3. Picture showing blob detection can detect simple red objects in its field of view

Successful communication between the Arducam and the turret PCB has been achieved in simple data transfer. Our test was first making sure the Arducams could detect a red object, take a picture then print it to a laptop. This was achieved back in Senior Design one. However, communication to multiple PCBs and triggering GPIOs had not yet been tested. The second part of the test was seeing if sending an image to the turret PCB could trigger an LED to turn on which we could do. The next stage was to have the cameras detect an object, send this to the turret PCB however now the cameras will be constantly polling, and the LED had different status. The LED off means no object is detected, a blinking LED means one of the two cameras was detecting the object and the LED being constantly on meant both cameras were detecting the object. We managed to get a detection distance of up to ~22ft, which is well above our specification of 8ft we wanted.

B. PCB Testing

Currently all three boards; Regulators, access board and turret PCB have been made and are fully operational. This did require version two that we had to order.

A success we had was the connection between all hardware connected and powered. The power bank is able to successfully power the access box and regulators, the access box can distribute power to the turret PCB, and the turret PCB can operate the regulators as well as the HuskeyLens and servos all the same time. We are still working on integrating the solenoid as well as the laser, but we want to get the timing down and communication between the cameras to both boards and to the HuskeyLens figured out first before worrying about shooting the target.

Our first main issue was the connectors we had chosen. The practicality of plugging and unplugging so many times and in general how flimsy the wires were to the tiny plastic connectors did not mesh well with our project. This at first only caused a single ESP32 IC on the access board to be shorted. As well on the access board the boot and reboot buttons were not configured correctly. Since our design for the PCBs is modular in the sense that the PCB is mostly traces with all peripherals other then LEDs and a buzzer coming off the board through wires in some way. This was the main reason for wanting to order a new version of all the boards. We ended up ordering new version of the access boards and turret PCB. We decided not to touch the regulators since we wanted to introduce the least amount of risk as possible. The connector into the regulators will stay, we instead changed the incoming connector to the access board and turret instead so the regulator design could stay the same. Another IC blew during some initial connecting when we left power on and connected a connector that was halfway plugged it.

The 5V regulator was outputting at 5.2V to account for loss in wiring distance, however this was too high and caused the HuskeyLens to be shorted during testing as well as degradation of the Arducams we used as well. The 5V regulator R2 will be switched from 3.3k to 3k to account for this issue.

With the Arducam issue we came across, it happened in the second version of the access box. We noticed the cameras could write a value however reading was not always successful. The reason we found out was due to the degradation of the camera from over voltage. The max speed it can run according to the datasheet is 8Mhz, however after testing as much as we did it was down to 6.5Mhz to have multiple successful runs in a row. To account for this in the future, 100-ohm resistors were added to all SPI data lines going from the ESP32 to the connectors as well as pullup resistors on all lines as well.

C. HuskeyLens Testing

Communication between the HuskeyLens and servos are successful as well and being fine-tuned currently. We uploaded a database to train the HuskeyLens to specifically track red ballons as our targets. This allows for known pixel sizes, allowing deterministic behavior when the Arducams need to calculate the distance based off the picture and can send this data to the HuskeyLens to lock on. With this known database, the HuskeyLens can track its target in a relatively smooth way without the assistance of an IMU or PID controller yet. Both are being integrated to smooth out the transition even more as we plan to work on a moving target but first want to accomplish our goals first with a stationary target.

Initially, using the HuskeyLens Color detection model we had lots of issues. You can first assign color IDs when first training with the color tracking setting.

Different lighting conditions would cause massive inconsistencies in proper detection when doing this. Objects in the background such as a red cooler or a bag would get mistaken for the ballon regardless of distance. Trees with orange leaves in the background could be detected depending on lighting conditions and mistaken as red when we only want the Huskeylens to track red objects. This was the main reason for transferring to our own database so we can still have our original description of a stationary red object, without needing to order a new sensor.

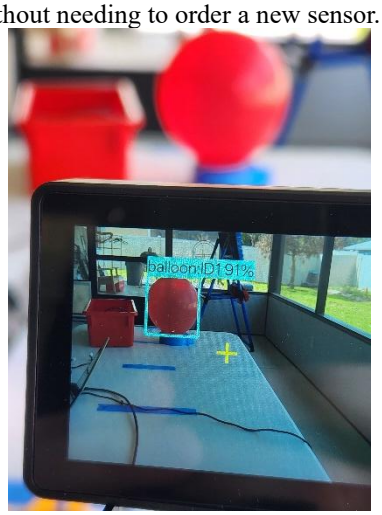

Figure 4. Showing what the HuskeyLens sees and even with two red object in its frame, it only puts a bounding box around the Ballon.

VII. 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 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.

For the vision portion of the project, the color-based blob detection proved to be quite challenging. Eventually 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.

PCB design had its many challenges but thankfully these changes were easy to implement. The connectors and addition of the SPI resistors were decided before ordering the second version of PCBs so thankfully these problems were found together instead of separately. Organization of the access box and turret happened as well. Version one of both were prototypes and made in haste to work enough to function. Once we knew connections were working, changing the layout so it is better for wiring and neater became easier.

The HuskeyLens being the main final detector has its faults, but we are working through all of them now. Adding our own database exponentially increases detection rate and adds a lot of maneuverability that we did not have before with just color detection. IMU integration along with its PID will be vital for the final demo as we need the turret to be steady before it can fire accurately.

Some steps that we plan on taking for the final demo is full tracking by the HuskeyLens. Currently, we can track but not in a real time sense, causing the object to go out of view faster than the turret moves. A stationary target right now the turret can track and aim however due to the tuning of the PID controller, it cannot aim completely at the center yet, rather it alters between the left and the right side of the bounding box of the HuskeyLens as seen from Figure 4. The integration of the Arducams communicating with the HuskeyLens is in works as of now as well. Both are being tuned, however, so progress still needs to be made. Integration of most peripherals such as status LEDs, buzzers, solenoid firing and laser still need to be done. We are leaving these for later as the main project is to be able to track and identify the enemy and be able to shoot. So, the first peripheral to be added will be the laser then the solenoid followed by LEDs and buzzer.

Acknowledgment

We would like to thank our advisors Dr Wei Sun, Dr. Qifeng Li, and Dr. Suboh Suboh for the help they have given to us throughout the project.

We would also like to thank Dr. Weeks with his help in approving and testing of our PCBs. We would be much further behind without his experience in this field.