ECE Senior Design Final Report

Remote-Controlled and Remote-Monitored Cat Toy

Project Sponsor: Dr. Tony Pinar
Sponsor Contact: ajpinar@mtu.edu

Faculty Advisor: Behzad Akbari
Faculty Advisor Email: behzadak@mtu.edu

Team Members

Vincent Maguire	Electrical Engineering	Team Lead, Electronic Specifications, PCB Design
Wyatt Gahm	Computer Engineering	Software Design, WebRTC Communication
Jason Bowers	Electrical Engineering	System Integration, Schematic Design
Wes Campbell	Computer Engineering	Object Detection, Simulation Design
Elliot McDonald	Electrical Engineering	System Integration, Enclosure Design

Department of Electrical and Computer Engineering
Michigan Technological University
June 4, 2026

Executive Summary

Project Overview

The Remote-Controlled and Remote-Monitor Cat Toy project addresses a common problem for pet owners: cats often experience boredom when left home alone, yet owners still want to have meaningful interaction with them. Our solution is a robotic mouse that can be driven remotely, with controls sent over the internet and that can also be configured to run autonomous play routines when no one is available. A low-latency video stream allows owners to see and control the toy in real time, keeping the cat engaged and reassuring the owner.

The system we designed has a simple framework that can be broken into three distinct sections, the entertaining robot ("the Mouse"), a central computer that processes camera information, hosts the web interface, and forwards controls to the Mouse ("the Hub"), and ("the User Web Interface") that the pet owner can use to control the mouse and monitor their pet.

This system improves upon existing smart home devices, such as the Ring camera, by eliminating the need for a third party company to access your data. Furthermore, extensibility and modification possibilities make it a good platform for integrating other aspects of a smart home into a system that the user can monitor while operating. In addition to being mutually entertaining, this design serves as a template and exploration for other secure smart devices.

Hardware Architecture

We designed the Mouse to be approximately the size, shape, and speed of a mouse, making it engaging for pets. The system incorporates two motors and corresponding motor drivers that provide propulsion and steering. A microcontroller directs motor behavior based on user inputs and executing commands from the Hub.

Wireless control commands are transmitted over Bluetooth to an onboard modem. The microcontroller interprets these commands and translates them into motion instructions for the motors. In addition to user control, the device includes the capability for a LiDAR sensor that continuously communicates with the microcontroller. This sensor allows the system to recognize when it is approaching a wall or other object and adjust its movement.

Remote Monitoring

A central component of our system architecture is a low-latency video monitoring module that enables pet owners to observe, in real time, their cat's interactions with the entertainment device. To accomplish the task, a camera sends video data over the internet to the pet owner via the Hub, the Hub also uses an object recognition program to map the room. This map is used to navigate the mouse away from the cat in its Autonomous Algorithm.

Acknowledgments

Thank-you Dr. Akbari for your guidance throughout the semester. Thank-you Dr. Pinar for your feedback and for sponsoring the project. Thank-you Chuck Sannes and Mark Sloat for technical assistance.

Contents

1 Introduction and Project Background

Many pet owners enjoy interacting with their animals but often have responsibilities that prevent them from being home consistently. This is especially true for cat owners, as cats benefit from regular stimulation, movement, and unpredictability in their play environment. Typically, owners provide this engagement directly through interactive toys or play sessions. However, when the owner is away, the cat may not receive the level of entertainment and enrichment required for healthy behavior.

To address this issue, our project aims to create a system that enables pet owners to interact with their cats remotely. This requires the ability to stream video to the owner with very low latency, allowing real-time interaction without the delay that would otherwise make remote play impractical. Additionally, the system must include a physical toy for the cat to engage with. For this purpose, we have developed the Mouse to be controlled through the same interface used for video streaming.

Beyond manual control, the system should also provide autonomous entertainment when the owner is unavailable. To support this, the robot will incorporate onboard sensing and basic object-recognition capabilities. These features allow the device to navigate safely within the room, avoid obstacles, and perform simple autonomous behaviors that encourage the cat to play even without direct human input. These features allow the device to navigate safely within the room, avoid obstacles, and perform simple autonomous behaviors that encourage the cat to play even without direct human input. A high-level project sketch and corresponding system block diagrams are shown in Appendices A and B, respectively.

2 Design Problem Analysis

A number of challenges exist for real time video feeds and small robotics. The constraints of embedded devices in this scope include battery operation, low latency control, motor driving, and packaging size. For real time video feeds, network challenges exist in the transport and link layer that can cause delays for the user [1, 2].

We are handling the video streaming challenges by writing platform-agnostic code and using the bandwidth and processing power of powerful single board computer, or SBC, to accomplish delay times that will make the user experience as smooth as possible, by utilizing H.264 video encoding hardware support to decreasing the latency of the streaming pipeline.

Operating a robotic device around pets introduces safety concerns such as electrocution risks, pinch points, and potentially harmful noise. To address these hazards, our design incorporates enclosed moving parts and safer wheel mechanisms to minimize exposure and ensure pet-safe operation. Additionally, the gearbox is situated inside the dust cover to minimize the risk of outside objects being sucked in or pinched.

The website that is used to control the service also needs to be secure. By having a webcam

exposed to the broader internet, we create a privacy issue. We have identified a need for authentication and encrypted communications using SSL. Other considerations must be taken when making a firewall exception on a home router. We are selecting a less commonly used port for traffic to eliminate chances of harmful interference from the broader web.

Our plan to use wireless control has the consequence that additional measures must be taken to ensure reliability. The Mouse cannot move when not directed to, and should implement error checking/correction for stable wireless control. The protocol design will encompass these and other safety considerations to make it safe for animals.

Several constraints exist with live streaming a camera over the internet. Latency is introduced in every hop at the link layer, limiting range in severe cases. Network configurations like NATs can interfere with connection establishment. Video data must also be time-encoded and encrypted for reliable playback and security. See references section for a detailed summary of the WebRTC protocol suite [2], and the decision matrix in the appendix. In addition, the system latency budget is summarized in Appendix H, and detailed protocol and language decision matrices are provided in Appendix M.

This project presents several key technical challenges. First, achieving low-latency video transmission requires careful selection of protocols and codecs, as well as using hardware that supports and can run that codec quickly. Second, the embedded system must balance processing capability and power consumption, motivating our choice of an NVIDIA Jetson Xavier NX Development Kit over lower-performance SBCs such as the Raspberry Pi for real-time video and object detection. Third, the motor and drive system must remain compact while providing enough torque and responsiveness for quick maneuvers. Finally, integrating sensors such as LiDAR and the camera with a safe remote-control command structure requires robust communication, collision-avoidance logic, and explicit safety constraints on how and when the robot can move around a pet.

A small mouse robot introduces unique design considerations regarding the compactness and speed requirements. First, the cat entertainer robot must be able to turn quickly and move quickly to provide real challenges to a feline. Motors and battery configuration must support a small form factor. The minimum speed is defined to be 1 m/s and the maximum size if 10 cm in length, close to real mouse capabilities. A two motor differential drive system addresses this problem in two ways: Dual motor control in both directions allows the robot to turn in place and halves the force required for forward movement. The power requirements of the motors made a high energy density battery necessary to achieve multiple hour runtime with an estimated motor stall current of 1 Ampere.

A top down approach for monitoring was chosen because the vantage of the mouse limits the user's awareness of the environment and cat movements. Anticipating cat movements and reacting to it's position is simplified for both the user and autonomous algorithms under this configuration.

3 Design Problem Solution

The proposed solution to this problem is to create a custom robot, "the mouse", that will interact with an animal while the owner is away from home. This mouse is controlled by "the hub", a single-board computer with internet and BLE. The hub then interacts with the user through "the user web interface", a website where the user will be able to view a live video feed of their room and control the mouse.

3.1 System Architecture

System Description

The Mouse system consists of three primary subsystems: the user control interface, the "Hub" processing unit, and the onboard robotic platform. The user interacts with the system through a Joystick UI element, which sends coordinate data to the Hub at a regular frequency. The Hub is stationed on a high vantage point like a bookshelf, and processes these inputs and transmits motion commands over Bluetooth Low Energy (BLE) to the Mouse's ESP32 microcontroller.

Onboard the robot, the ESP32 interprets the received commands and drives the dual-motor system through PWM-enabled motor driver H bridge ICs. A LiDAR sensor port is used to continuously report distance measurements to the ESP32, allowing the microcontroller to override external commands and reverse the motors when a collision risk is detected. A camera mounted on the robot streams real-time video back to the Hub using WebRTC, completing a closed-loop system where controls flow from the user to the robot, and live footage flows back to the operator. A graphical system architecture block diagram is provided in Appendix B

3.2 Hardware Design

3.2.1 PCB Development

The custom PCB for the entertainer device is developed to integrate all essential electronics into a compact chassis approximating the size of a mouse. The board consolidates sensing, power regulation, motor control, and communication into a single platform that interfaces with the 'Hub' or SBC base station over Bluetooth Low Energy (BLE). The PCB is designed with lowest wiring complexity and enclosure awareness, with Micro-USB and push button placed around outside edge of board. This design incorporates all the electronic subsystems into a clean form factor, achieving a robust integrated motion controller.

The PCB integrates the microcontroller, motor drivers, LiDAR, power regulation circuits, and programming interfaces. Attention is given to grounding strategy and separation between digital and motor-current domains to reduce conducted noise. Power planes are added to ensure low-impedance distribution to the motors, and the microcontroller traces were kept short to reduce signal integrity issues. Mounting points and connector locations were chosen based on the mechanical constraints of the mouse-shaped enclosure. The final schematics

for the motor drivers, DC-DC converters, and microcontroller interfaces are shown in Appendix D.

In laying out the PCB, several key considerations were taken into account. The most important was minimizing RF interference, so the motor drivers and high-current switching components were placed as far as possible from the ESP-32 package and its antenna region. Care was also taken in the mechanical design of the board, ensuring reliable mounting by incorporating SMT standoffs. The resulting PCB layouts are shown in Appendix E.

3.2.2 Power Distribution

Power for the system originates from an onboard Li-ion battery, regulated down through a staged power tree [8, 9]. A buck converter and LDO work together to step the power down to a stable 3.3V for the microcontroller and and sensor, while a separate motor power rail delivers higher current to the dual motor drivers. The power system also includes a simple BMS to charge the lithium pouch battery. The BMS system is designed to charge the single cell to a set voltage then cut off the current after the battery reaches the selected charge level.

3.2.3 Sensors

The camera mounted on the Hub side of the system is used for high-level video streaming and optional object detection. Research into low-latency video protocols led to the adoption of WebRTC, and YOLO for object detection. The camera is connected through the USB3 interface. The camera choice is situational, with some being more accurate or having better dynamic range for improved object detection and tracking. The field of view of the camera encompasses the play area in the ideal case. A high vantage point is ideal for object detection and manual usability when obstacles are present.

3.2.4 Motor Controllers

Two brushed DC motors are controlled through dedicated H-bridge motor driver ICs [8]. These drivers were selected for their current capability, ease of connection, and availability in surface-mount packages suitable for the compact PCB. The microcontroller generates PWM signals to command the motors by changing the registers.

3.2.5 Hub Hardware

The Jetson Development Kit was chosen as the single board computer (SBC) for the Hub side of the cat entertainer [5]. It was chosen due to it's video processing time, using WebRTC with low latency. Additionally, using the Jetson in our design allows us to take advantage of AI-optimized hardware for tasks such as object detection. The Hub also packetizes and forwards control signals to the entertainer robot over Bluetooth [5]. The Bluetooth subsystem on board the development kit supports a UART protocol that prevents corruption and lost packets. The Ubuntu operating system distributions support python and the necessary packages for streaming. The Jetson we used was the NVIDIA Jetson Xavier NX Development

kit with "Jetpack" linux image version 5.4, which was graciously supplied by our sponsor Dr. Pinar.

3.3 Software Design

The software system is responsible for three core functions: serving a secure web interface to the user, streaming low-latency video from the ceiling-mounted camera, and routing control commands from the user to the Mouse over Bluetooth. All software runs on the Hub and is implemented in Python. The Python (version ≥ 3.8) implementation provides portability, meaning the software can be used on other SBCs or regular computers. Libraries like OpenCV are optimized for specific hardware automatically.

3.3.1 Technology Selection

A language decision matrix was used to evaluate Python, C++, and JavaScript across criteria including real-time performance, computer vision library support, ease of development, WebRTC and networking support, community libraries, and maintainability. Python scored highest with a weighted total of 8.7, outperforming C++ (7.2) and JavaScript (6.5). Its strong ecosystem of libraries — particularly for computer vision (OpenCV, YOLO) and WebRTC (`aiortc`) — combined with ease of development made it the clear choice for the Hub software.

3.3.2 Networking and WebRTC

Networking is accomplished via WebRTC, a standardized protocol stack supported natively in all modern browsers. A protocol comparison matrix evaluated WebSocket, RTSP/RTP, MQTT, and WebRTC across latency, browser compatibility, ease of implementation, scalability, bidirectional communication, and reliability. WebRTC tied for the highest weighted total score of 4.0, with top marks for latency, browser compatibility, and bidirectional communication support.

WebRTC provides several properties essential to this system:

- **Encrypted transport:** Video is delivered over SRTP (Secure Real-Time Transport Protocol), and the control data channel uses DTLS for transport-layer security. This ensures that neither the video feed nor control commands are transmitted in plaintext.
- **NAT traversal:** ICE (Interactive Connectivity Establishment), STUN, and TURN protocols handle connection establishment across different network configurations — including NATed home routers — without requiring manual firewall configuration.
- **Data channels:** In addition to the video track, WebRTC data channels carry joystick control coordinates from the browser to the Hub. These are forwarded by the Hub as Bluetooth packets to the Mouse.
- **Browser compatibility:** No plugins or applications are required on the user's side. The interface loads as a standard webpage.

- **Security:** WebRTC protocols are used commonly, so they are already stress tested against various attack techniques
- **Support:** Browsers and native libraries support the WebRTC suite, and are tested frequently.

The Python library `aiortc` implements the full WebRTC stack and integrates with OpenCV for camera capture and frame encoding. The Hub hardware should be optimized for video encoding pipelines and matrix multiplication, which reduces CPU load and encoding latency in conjunction with object detection. The Hub pipeline waits for a client connection, serves the login-protected webpage, completes the WebRTC signaling handshake, then begins streaming encrypted video while keeping the data channel open for incoming control messages.

3.3.3 Latency

Video latency is a critical factor for the feasibility of internet-based remote control. Because the user must react to what they see, the total round-trip time — from sending a control input to observing the result on screen — determines whether real-time play is practical. A target of under 100 ms RTT was established to keep the interaction feel responsive.

RTT was measured across several network scenarios using ICMP ping tests and data channel RTT statistics. Results showed consistent performance across cases:

Table 1: Measured RTT by Network Scenario

Scenario	**RTT Range (ms)**	**Rating**
Video — same router	21–45	Good
Video — different network (∼2 mi)	54–81	Moderate
Video — Cloudflare Tunnel	73–103	Moderate
Controls — same router	12–23	Good
Controls — different network (∼2 mi)	24–36	Good
Controls — Cloudflare Tunnel	34–51	Good
Overall average RTT	**53.07**	

The control data channel consistently achieves lower RTT than the video stream, meaning the Mouse responds before the user sees the updated frame — a favorable asymmetry for interactive control. The Bluetooth (BLE) round trip from the Hub to the Mouse adds approximately 3–4 ms on average, which is negligible relative to the network component. The small packet sizes for control packets reduce processing delay in the network stack.

3.3.4 Web Server and Security

The Hub runs a Python web server that handles login authentication and WebRTC session signaling. Access to the camera feed and control interface requires a password before any WebRTC connection is established. This prevents unauthorized access to the video stream.

Broader security considerations include:

- The server runs exclusively on the Hub — there is no cloud intermediary. Video and control data do not pass through any third-party servers.
- All video and data channel traffic is end-to-end encrypted via SRTP and DTLS respectively.
- Remote access is facilitated through a Cloudflare Tunnel or a commercial VPN, which avoids the need to expose the Hub's IP address directly or open inbound ports on the home router.
- Signed certificates are used to encrypt the web communications, with OpenSSL python packages

3.3.5 User Web Interface

The user web interface is a browser-based application served by the Hub. It presents the live camera feed alongside a joystick control element and connection status indicators. The interface is fully responsive and supports both desktop and mobile browsers without any additional software installation.

In manual mode, the joystick sends X and Y coordinate data over the WebRTC data channel at a fixed frequency. The Hub receives these values and forwards them as single-byte BLE packets to the Mouse's microcontroller, where they are decoded into left and right motor commands. In autonomous mode, the joystick is replaced by the Hub's pathfinding output, and the user can monitor the autonomous play session through the live feed. A drive mode toggle allows switching between manual and autonomous control from the interface.

Representative code for the video streaming server, BLE motor controller interface, and web interface is included in Appendices Q, R, S, and T.

3.3.6 Embedded Code

The entertainer robot microcontroller will be programmed to the following psuedocode.

Initialization Routine

- Initialize the motor drivers so the microcontroller can command speed and direction for both motors.
- Initialize the Bluetooth modem or wireless link so control packets from the Hub can be received.
- Set the starting state of the robot to "stopped".

Main Loop

The robot repeatedly performs the following steps indefinitely:

1. **Check Connection Status**
    - Check if the bluetooth modem reports connection loss

- Backup system
  - Set a countdown hardware timer every time a packet is received
  - The timer interrupt will indicate loss of connection
- If no packet is received or connection stopped:
  - Decelerate and prepare for new connection

2. **Read Control Input**
   - Check if a new Bluetooth control packet has arrived.
   - If a packet is received:
     - Interpret each nibble of the byte as a direction bit & 3 bit integer
     - Update the robot's current command state accordingly.
3. **Check for Collision (LiDAR Override)**
   - Query the LiDAR for the distance to the nearest obstacle.
   - If the distance is below the collision threshold:
     - Override any Hub control commands.
     - Initiate the collision-avoidance routine:
       * Drive both motors in reverse for a short, predefined interval.
       * After reversing, stop both motors.
       * Skip normal movement logic for this cycle.
4. **Execute Movement Command (Only If No Collision)** Based on the most recent valid Bluetooth command:

$$\underbrace{1010}_{\text{Left Nibble}} \qquad \underbrace{0110}_{\text{Right Nibble}}$$

$$1010 \rightarrow \text{Backwards at: } 2/7 = 29\% \text{ power}$$

This protocol is found in Appendix O.

3.4 Schematic Development

In order to develop a working schematic for the Mouse, our design approach began with reverse–engineering an existing commercially available mouse–shaped cat toy (see Appendix C). By examining the internal electronics, motor layout, and mechanical linkage system, we gained a clear understanding of how low-cost remote-controlled cat toys implement steering, power distribution, and simple signal decoding. This analysis provided a baseline architecture from which we could design a significantly more capable, microcontroller-driven system.

From this reverse-engineering effort, we identified several critical schematic considerations required for our custom design:

- **PWM-Controllable Motor Driver ICs:** To achieve smooth speed control and turning behavior, the motors must be driven using pulse-width modulation. This requires selecting a motor driver IC capable of handling our expected stall current, supporting direction control, and ensuring thermal protection. Our schematic includes dual H-bridge drivers that interface directly with the ESP32's PWM-capable GPIO pins.

- **Programming Interface via POGO Pins:** For compactness and ease of assembly, we opted not to include a traditional USB connector on the robot PCB. Instead, we're using a dedicated programming interface using POGO pins and the standard ESP32 flashing sequence. This requires a small RC filter and supporting circuitry to ensure proper boot mode selection and signal integrity during firmware upload.

- **ESP32 Strapping Pins:** The ESP32 requires a specific set of boot strapping pin states to correctly enter normal execution mode or flashing mode [3]. These include pull-up/pull-down resistors on GPIO0, EN, and other configuration pins. Our schematic ensures that the strapping pin requirements are met while still allowing the microcontroller to use remaining GPIOs for motor control, LiDAR data, and Bluetooth communication.

- **Minimum Required ESP32 Connections:** Aside from strapping pins, several essential supporting circuits must be included for reliable operation—such as decoupling capacitors near the power rails, a stable 3.3V regulator, antenna clearance requirements, and protection components for input signals. These elements were incorporated following Espressif's official hardware design guidelines [4].

Through this schematic development process, we created a robust electrical architecture that supports motor control, wireless communication, and safe firmware programming. The full annotated schematic is included in Appendix D.

4 Enclosure Design

The enclosure for the Mouse needs to meet the following criteria: It must be self righting, it must not expose the battery to the cat, and it must allow room for motors and the PCB. A first revision shows the minimum size and profile requirements, while also testing the tolerances and accuracy of the 3D printing process. The early design was made obsolete when the PCB was separated from the battery holder. The new design includes a battery holder, carve-outs for additional ToF sensor, fully enclosed motors gearboxes and dustcovers, and mounting spots for a front wheel. The enclosure should be constructed from a safe and non-brittle material, which is important to protect pets from plastic fumes and sharp fragments. Aside from the ideal injection molded technique, 3D printing of the design is to be done with PET/PETG plastics. The FDA authorizes PET under 21 CFR for chemical safety. However, the prototype design features a PLA enclosure due to time and resource constraints in the available manufacturing technologies on campus. See Appendix N for CAD model images.

5 Project Management

5.1 Team Workflow and Meeting Strategy

Our team utilized a variable length meeting on Tuesday and Thursday to discuss important tasks, decisions, and work together on tasks that required it. Additionally, Our Thursday meeting featured our advisor, Dr. Behzad Akbari, discussing progress and other details. Some fundamental meeting activities are:

- Address action items for all team roles
- Discuss important decisions and project direction
- Update Gantt chart and budget
- Discuss objectives between meeting times
- Specifications refined & communicated
- Work on present objectives and refine goals

We used Google Drive to share files and documents, which is useful for collaboration on various documents and diagrams. We organized our goals, which each specific member could make progress on, in a Gantt chart. In addition, Vincent delegated and dispatched tasks (e.g., fixing a software bug or modifying a design) via communication channels. Email communication with Dr. Akbari was frequently used to clarify specifications or communicate import information. These messages often went to Vince, who was a point of communication with our advisor as well as the peer review team.

5.2 Task Breakdown and Scheduling

The first weeks of the fall semester were used to define our objectives and specifications for our robot. We then moved on to establish the video streaming protocol and finalizing important decisions for the mouse going forward. These decisions included motor type, shell design, controller, and sensors. Tasks were determined as follows: enclosure design, networking, software, and hardware.

The team needed to work together to produce progress presentations and other project documentation to ensure that all group contributions were integrated into the final materials.

Table 2: Task Breakdown by Area

Area	**Members**	**Tasks**
Enclosure design	Wyatt, Elliot	CAD modeling, fabrication, iteration, durability info
Software	Wes, Wyatt	WebRTC setup, data channels, debugging, latency testing
Hardware	Vincent, Jason	Schematic design, PCB layout, fabrication

For example, during the fall semester the hardware design team was responsible for designing the schematic and determining which parts and technologies to use. Ongoing decisions were make (consider the decision matricies in Appendix M) so the design was constantly updated.

In the first semester the software team developed a proof of concept for video streaming, debugged the Hub computer, demonstrated object detection, and created a basic web interface.

In the second half of the project the CAD modeling team made several configurations of the various components (see Appendix N) to solidify the final layout and attachment methods, among other things.
Refer to the Gantt chart in Appendix L.

5.3 Budget and Resources

The team was given a budget of $500 to create the robot and monitoring system. At the time of project completion, $295.48 of the budget was used. The main purchases include: an existing remote controlled cat toy for market research, a microcontroller with bluetooth capability, small DC motors to drive the wheels, a webcam for room imaging, and custom PCBs for the robot. A Jetson Xavier NX Development Kit is being used for the Hub, but is on loan from the ECE department and is not included in the current budget. The cost of components stayed $204.52 under the $500 budget. A detailed bill of materials is provided in Appendix J.

5.4 Individual Team Member Contributions

Team members focus on specific areas of the design in parallel. This allowed for independent work to be done and discussion of efforts in meetings. The work is divided into 3 primary groups: Software, Hardware, and Networking. Software members were responsible for developing the code and computer hardware (The base station / Camera) for the system, as well as troubleshooting integration. Hardware work involved selecting parts/components, designing, and assembling the Mouse robot. Networking involved troubleshooting and writing code for secure and encrypted video delivery over the network, managing latency, and integrating with hardware.

A specific example of work that was completed by the hardware team was the deconstruction of an existing circuit board for research purposes. Elliot and Vince de-soldered and labeled every component of the circuit board from an RC car. Later on, Vince and Jason mapped the traces on the board for important components. This research gave the hardware team a good starting point for our custom board layout. More information is found in Appendix C.

The hardware team also spent a lot of time revising our enclosure design throughout the building of the project. Elliot designed a very early version of our enclosure where the motors were above the PCB to limit the enclosure width. This design was turned down after Vince tested and found the wheels would have to be really large in order to reach the ground. Because of this, the enclosure design was quickly revised to have the motors be in line with the PCB. This design was deemed to have desirable qualities, specifically the fact

that it was tip resistant due to it being quite bottom heavy. The next revision in the design process was making the enclosure closed and look like a mouse. This second early revision was tackled by Elliot and Wyatt where they made a base for the hardware and a detachable top cover. This revision's desirable qualities were its mouse-like look as well as its ability to cover the electronics while still keeping them accessible. The final revision made to the enclosure design was putting the two early revisions together. Wyatt combined the in line motors, stacked PCB and battery design with the mouse-like detachable cover design, to create our final design. The final qualities were a tip-resistant, mouse-like enclosure, that had covered, but still accessible electronics. Early revisions and the final CAD model can be found in Appendix N

Table 3: Team Member Contributions

Team Member	**Major**	**Contributions**
Wyatt Gahm	Computer Engineering	Video transport code, integration, networking, enclosure design
Vincent Maguire	Electrical Engineering	Reverse engineering process, schematic design, PCB and firmware
Wes Campbell	Computer Engineering	Object detection, bluetooth communication
Jason Bowers	Electrical Engineering	Schematic design, PCB assembly
Elliot McDonald	Electrical Engineering	Enclosure design, enclosure and PCB assembly

6 Military Applications

The pathfinding and object evasion software used in the remote controlled cat toy can also have applications in the real world. In a military setting, autonomous ground robots equipped with this software, and supported by overhead surveillance from aerial cameras or persistent surveillance systems, have potential in reconnaissance roles. These robots could act as forward scouts, moving ahead of human units to gather visual or acoustic intelligence. Using environmental mapping and navigation, the robot will be able to traverse complex terrain and retreat if threats are detected.

6.1 Perimeter Security

A valuable application is in perimeter security around bases worldwide. Airborne or stationary cameras are able to monitor wide areas and communicate with the scout robots, directing them to investigate anomalies when they are detected. If a threat is identified, the scout would then be able to utilize the pathfinding capabilities to reposition and maintain observation while remaining safe and undetected.

6.2 Retreat and Survivability

The retreat to a safe zone functionality is paramount for survivability of the robot. When onboard sensors or overhead surveillance detect enemy proximity through motion or object

detection, the robot will be able to plan an escape route that minimizes exposure and maximizes safety. Rather than following the previously traveled route in reverse, the pathfinding software allows the scout to determine and travel along a route that minimizes risk potential.

7 Recommendations for Future Work

Additional algorithms and routines for autonomous play could be introduced, permitting the entertainer robot to engage in behaviors that are more natural to a mouse. Routines could be optimized to be more evasive or less predictable.

The Mouse could be equipped with multiple additional i2c devices:

- Gyroscope for improved tracking accuracy
- ToF sensor & housing for collision checks
- Accelerometer for measuring specific accelerations and orientations

This then introduces more firmware needs and considerations.

Because the object detection software, YOLO, was trained using a planar view, it is not completely accurate when used in a top down view. To improve object detection accuracy, more AI training in a top-down view is essential.

The front, fixed wheel is another flaw in our design. Because of it's inability to move as we turn, the robot will not turn until the turning force causes the front wheel to skid. This behavior makes it almost impossible to turn accurately in manual mode. This problem also exists in autonomous mode and makes its very hard for the mouse to follow a path. The solution for this would be to change the front wheel to small castor wheel. This would allow the wheel to move with the turn as opposed to skidding, providing accurate and predictable turning.

8 References

8.1 WebRTC

WebRTC is suite of protocols that serve to provide reliable, low latency communication across networks. Protocols are implemented to improve device connections and stability. This is important because various restrictive network configurations exist, so the underlying UDP transport streams require discoverability enhancements to navigate firewalls. Guaranteed support or implementation of protocols that enable data streaming and reliable transfer are also listed below.

- ICE - (interactive connectivity establishment) for candidate gathering and NAT traversal
- STUN - NAT traversal and relaying between networks
- TURN - fallback option for more intensive discovery pathways
- SCTP - Integration of this protocol using secure datagram level transport security is guaranteed
- RTP and RTPC - Secure and reliable video streaming protocol, well suited for live applications
- GCC - Google congestion control optimizes video quality to ensure low latency by mitigating queueing delays in the network layer from congestion

Stream: Internet Engineering Task Force (IETF)
RFC: 8835
Category: Standards Track
Published: January 2021
ISSN: 2070-1721
Author: H. Alvestrand
Google

RFC 8835
Transports for WebRTC

Abstract

This document describes the data transport protocols used by Web Real-Time Communication (WebRTC), including the protocols used for interaction with intermediate boxes such as firewalls, relays, and NAT boxes.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8835.

Figure 1: RFC 8835 Cover Page

8.2 ESP-32

- Industrial Automation
- Health Care
- Consumer Electronics
- Smart Agriculture
- POS Machines
- Service Robot
- Audio Devices
- Generic Low-power IoT Sensor Hubs
- Generic Low-power IoT Data Loggers
- Cameras for Video Streaming
- Speech Recognition
- Image Recognition
- SDIO Wi-Fi + Bluetooth Networking Card
- Touch and Proximity Sensing

For more information about ESP32, please refer to ESP32 Series Datasheet.

Note: Unless otherwise specified, "ESP32" used in this document refers to the series of chips, instead of a specific chip variant.

1.3 Schematic Checklist

1.3.1 Overview

The integrated circuitry of ESP32 requires only 20 electrical components (resistors, capacitors, and inductors) and a crystal, as well as an SPI flash. The high integration of ESP32 allows for simple peripheral circuit design. This chapter details the schematic design of ESP32.

The following figure shows a reference schematic design of ESP32. It can be used as the basis of your schematic design.

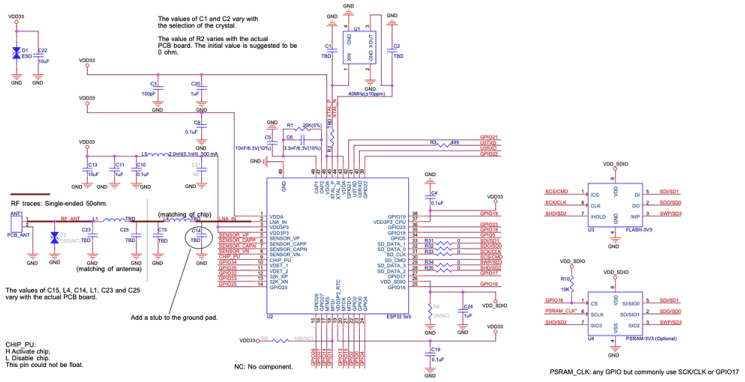


Fig. 1: ESP32 Reference Schematic

Note that Figure *ESP32 Reference Schematic* shows the connection method for quad 3.3 V external flash/PSRAM.

Figure 2: ESP-32 Reference Design [4]

Appendices

A High Level Project Concept

Figure 3: Project concept

B System Architecture Block Diagrams

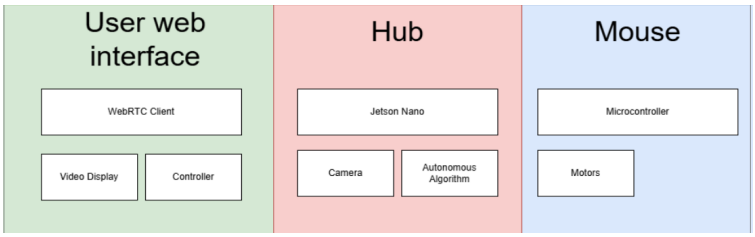
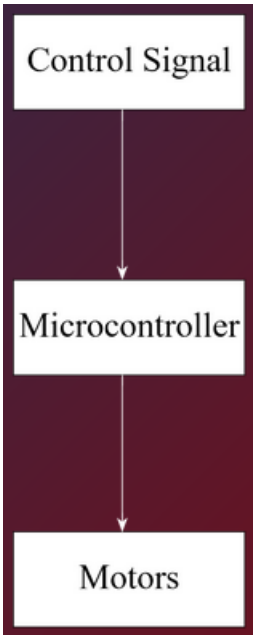
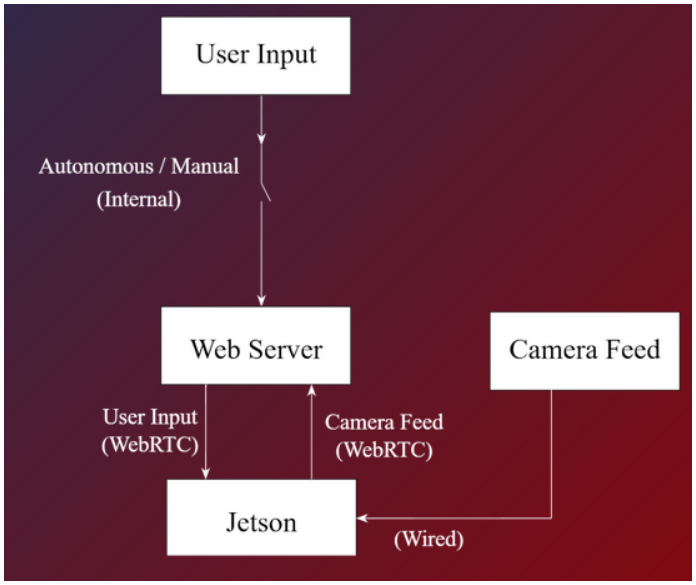


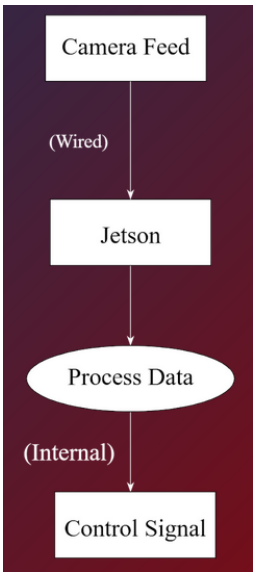
Figure 4: System block diagram



(a) Mouse subsystem block diagram showing onboard control, motor drivers, and power distribution.



(b) User interface block diagram showing control input interface and interaction with the Hub.



(c) Hub subsystem block diagram illustrating processing, communication, and video streaming architecture.

Figure 5: Sub-block diagrams for the each infrastructure sub-system, including the Mouse, the Hub, and User Interface.

C Reverse Engineering Process

Figure 6: Radio-controlled toy we chose to reverse engineer after doing market research. Toy sourced from Amazon.com

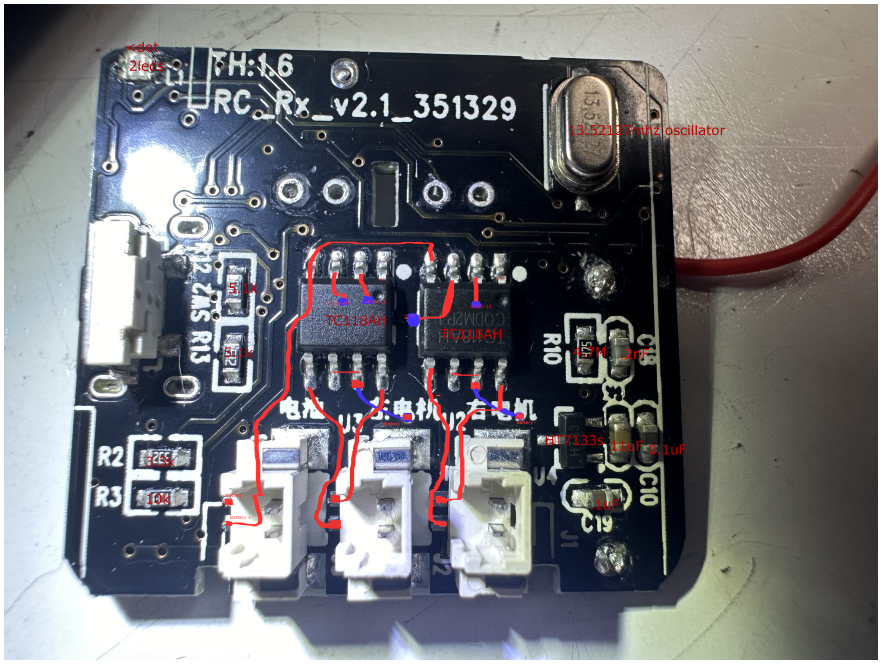


Figure 7: Top of reverse engineered mouse toy

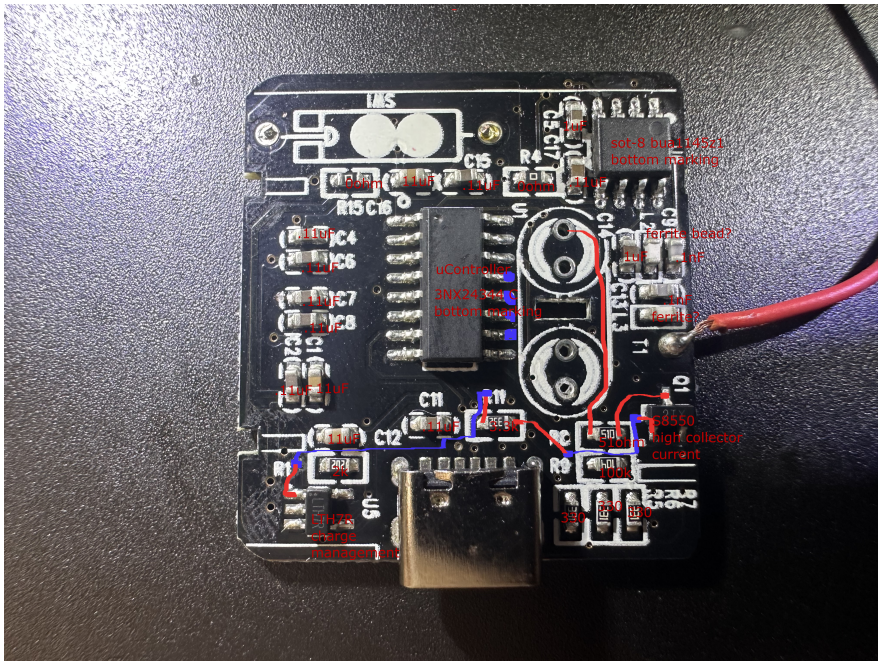


Figure 8: Bottom of reverse engineered mouse toy

All passives were de-soldered and measured, and ICs were identified. Some board traces were traced back in order to gain understanding of the existing hardware. In completing this exercise, we learned about how the commercialized product was designed. We used the same motor-driving infrastructure as the commercialized product, sourcing a surface mount H-Bridge driver like what was used on the commercialized product. Additionally, we decided to re-purpose the tires from this toy, but needed to print new wheels to fit them onto our motors, as the motors on the Mouse have a different keyway.

D Schematic Design

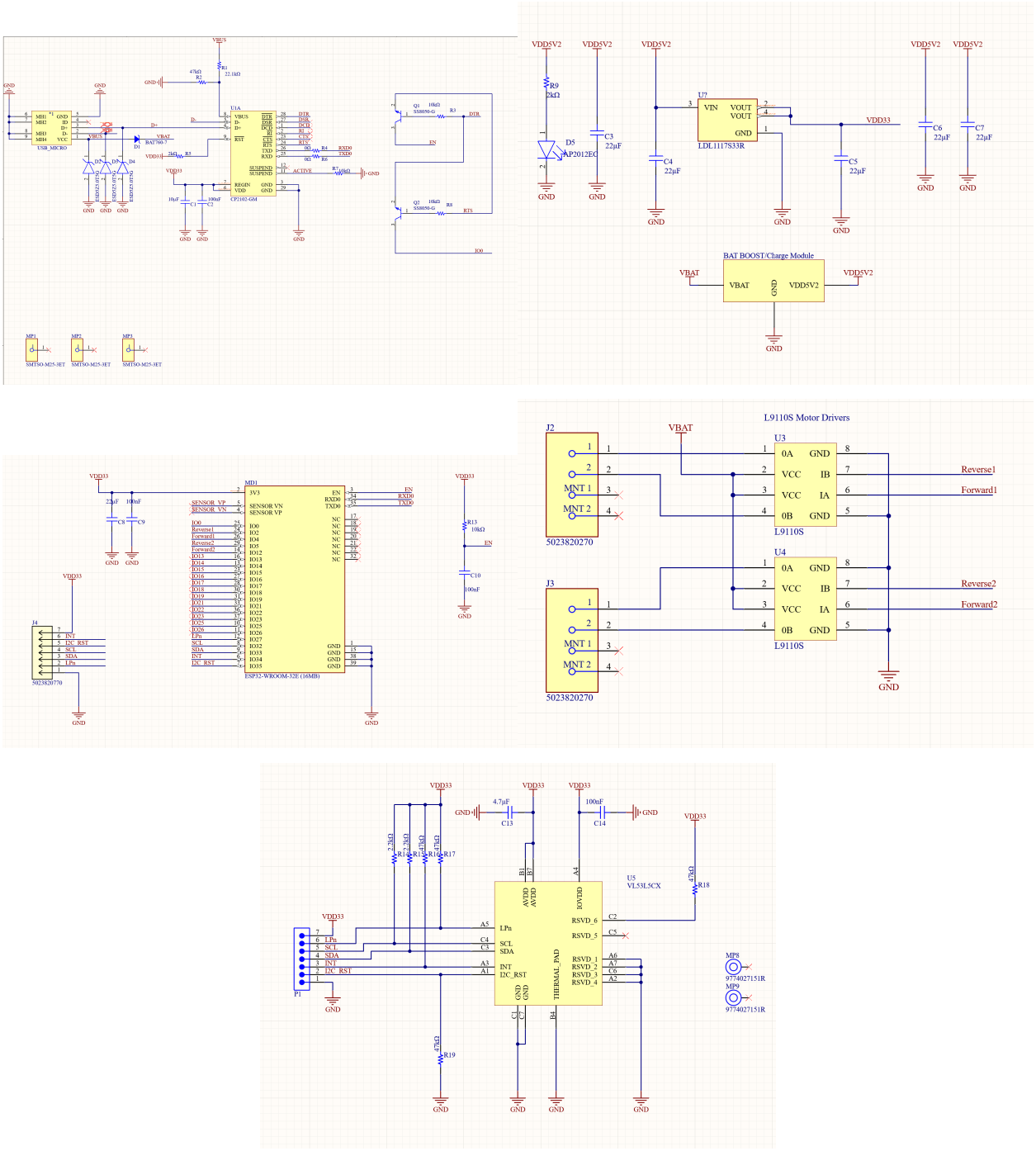


Figure 9: Schematic design views including the motor driver, DC-DC converter, microcontroller, and supporting circuitry.

E PCB Design

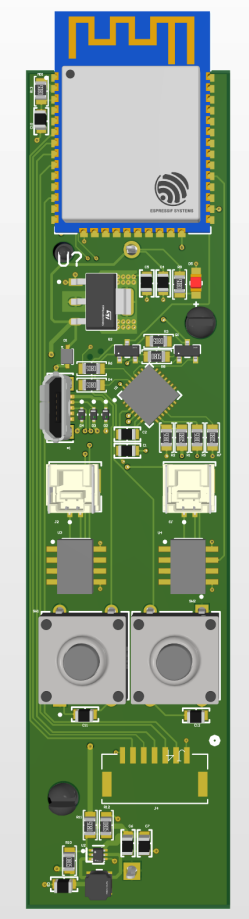


Figure 10: Main Board layout

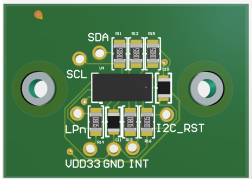


Figure 11: Lidar Tof board layout

F Patent Search Results

F.1 Patent: Remotely Operable User Controlled Pet Entertainment Device

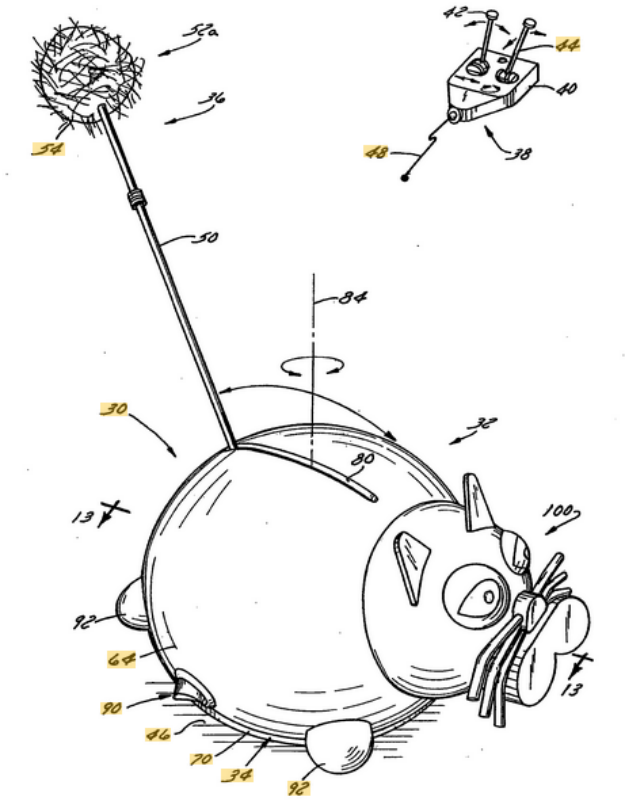


Figure 12: Patent Illustration

The claims of this patent extend to the ability of the device to exercise a pet, remote controlled operation, and two degrees of freedom in locomotion. This 2009 patent describes a specific application of remote controlled moving toys to "exercise" pets. Commercialization of the mouse robot we are developing would still be possible because our device is not inherently powered by human remote control all the time, nor does it serve the main purpose of providing exercise.

F.2 Patent: Scalable Web Real-Time Communications (WebRTC) media engines, and related methods, systems and computer-readable media

This patent extends to the usage of WebRTC and how we as consumers are allowed to use it.

G Bibliography

References

[1] J. Postel, "Transmission Control Protocol," RFC 793, Sep. 1981. [Online]. Available: https://www.rfc-editor.org/rfc/rfc793

[2] H. T. Alvestrand, "Transports for WebRTC," RFC 8835, Internet Engineering Task Force, Jan. 2021. doi:10.17487/RFC8835 [Online] Available: https://www.rfc-editor.org/rfc/rfc8835.pdf

[3] Espressif Systems, "ESP32 Series Datasheet," [Online]. Available: https://www.espressif.com/.

[4] Espressif Systems, "ESP32 Hardware Design Guidelines," [Online]. Available: https://www.espressif.com/.

[5] NVIDIA Corporation, "Jetson Xavier NX Development Kit Developer Kit Product Design Guide," [Online]. Available: https://developer.nvidia.com/.

[6] J. Redmon *et al.*, "You Only Look Once: Unified, Real-Time Object Detection," 2016. Available: https://arxiv.org/abs/1506.02640

[7] STMicroelectronics, "ToF VL53L3CX Datasheet," [Online]. Available: https://www.st.com/resource/en/datasheet/vl53l3cx.pdf.

[8] Elecrow, "Motor Driver L9110 Datasheet," [Online]. Available: https://www.elecrow.com/download/datasheet-l9110.pdf?srsltid=AfmBOorWia94kJF0YNokvRS9J8prObz2tf0zGGNsYhv0EbirlOzqEH-Q.

[9] ATNSINC, "Buck/Boost Step-Up Power Module," [Online]. Available: https://www.amazon.com/dp/B0BYMNCY65?ref=ppx_yo2ov_dt_b_fed_asin_title&th=1

[10] Pololu, "DC Gearmotor 2364 Datasheet," [Online] Available: https://www.pololu.com/product/2364

H Latency Data

Component	Target (ms)	Measured (ms)
Apartment to MTU (VPN)	50	45
Apartment to other house	30	26
BLE control round trip	4	3.2
Motor actuation latency	5	TBD
Total end-to-end	150	TBD

Table 4: Latency budget

I Relevant Standards

Table 5: WebRTC Protocol Suite References

RFC	Title	Organization	Date
8825	Overview: Real-Time Protocols for Browser-Based Applications	IETF	Jan. 2021
8829	JavaScript Session Establishment Protocol (JSEP)	IETF	Jan. 2021
8827	WebRTC Security Architecture	IETF	Jan. 2021
3550	RTP: A Transport Protocol for Real-Time Applications	IETF	Jul. 2003
3711	The Secure Real-time Transport Protocol (SRTP)	IETF	Mar. 2004
5764	DTLS Extension to Establish Keys for SRTP	IETF	May 2010
8834	Media Transport and Use of RTP in WebRTC	IETF	Jan. 2021

J Bill of Materials

Supplier	Manufacturer	Manufacturer Pa	Description	Quantity 1	Unit Price 1	Extended Price 1
Digikey	Molex	5023820770	CONN RCPT 7PC	1	0.80000	$0.80
Digikey	EVVO	AMS1117-3.3	IC REG LINEAR 3	1	0.27000	$0.27
Digikey	Diodes Incorpor	BAT760-7	DIODE SCHOTTK	1	0.47000	$0.47
Digikey	KYOCERA AVX	08056D106KAT2	CAP CER 10UF 6	1	0.08000	$0.08
Digikey	KEMET	C0805C104K5RA	CAP CER 0.1UF 5	10	0.01600	$0.16
Digikey	Samsung Electrc	CL21A226MAYN	CAP CER 22UF 2	3	0.04300	$0.13
Digikey	KEMET	C0805C104K5RA	CAP CER 0.1UF 5	1	0.08000	$0.08
Digikey	Samsung Electrc	CL21A475KAQN	CAP CER 4.7UF 2	1	0.10000	$0.10
Digikey	Panasonic Electr	ERA-6AEB473V	RES SMD 47K OH	5	0.06100	$0.31
Digikey	Stackpole Electr	RMCF0805FT22I	RES 22.1K OHM	1	0.10000	$0.10
Digikey	YAGEO	RC0805FR-072K	RES 2.2K OHM 1	3	0.01000	$0.03
Digikey	YAGEO	RC0805JR-070RI	RES 0 OHM JUM	3	0.01500	$0.05
Digikey	Stackpole Electr	RMCF0805FT10I	RES 10K OHM 1%	5	0.00800	$0.04
Digikey	UMW	L9110S	SOP-8 MOTOR D	2	0.59000	$1.18
Digikey	Molex	1051330001	CONN RCPT USB	1	1.24000	$1.24
Digikey	STMicroelectron	VL53L5CXV0GC/	SENSOR OPTICA	1	8.53000	$8.53
Digikey	onsemi	ESD5Z5.0T5G	TVS DIODE 5VW	3	0.09900	$0.30
Digikey	Alps Alpine	SKQBARA010	SWITCH TACTILE	2	1.05000	$2.10
Digikey	Molex	5023820270	CONN RCPT 2PC	2	0.58000	$1.16
Digikey	Keystone Electrc	1042P	BATTERY HOLDE	1	5.77000	$5.77
Digikey	Comchip Technc	SS8050-G	TRANS NPN 25V	2	0.14800	$0.30
Digikey	Kingbright	AP2012EC	LED RED CLEAR	1	0.23000	$0.23
Digikey	Molex	5023800700	CONN PLUG HSG	1	0.27000	$0.27
Digikey	Molex	5023800200	CONN PLUG HSG	2	0.16000	$0.32
Digikey	Würth Elektronil	9774027151R	ROUND STANDC	2	1.01000	$2.02
Digikey	Würth Elektronil	9774030360R	ROUND STANDC	2	1.76000	$3.52
Digikey	Adafruit Industri	4654	MINIBOOST 5V	1	3.95000	$3.95
Digikey	ZEUS Battery Pr	PCIFR18650-15C	BATTERY LITHIU	1	5.85000	$5.85
Digikey	Si Labs	CP2102N-A01-GQ	USB UART BRIDG	1	1.38000	$1.38
OSHPARK	OSHPARK	xx	TOF Board	1	2.50000	$2.50
OSHPARK	OSHPARK	xx	Main Board	1	23.8700	$23.50
NVIDIA	NVIDIA	xx	NVidia Jetson Or	1	249.990	$249.99
Amazon	Generic	xx	PETG 300G	1	8.00000	$8.00
Pololu	Pololu	2364	Pololu DC Gearn	2	23.9500	$47.90
			Total			
			BOM COST			$372.61

Figure 13: Bill of Materials for Project

K Budget

Supplier	Manufacturer	Manufacturer Part Number	Description	Quantity	Unit Price 1	Extended Price 1	Purchased yet?	Included in project budget?	
Digikey	Molex	5023820770	CONN RCPT 7POS 0.049 TIN SMD	3	0.80000	$2.40	Y	Y	
Digikey	EVVO	AMS1117-3.3	IC REG LINEAR 3.3V 1A SOT-223-3L	3	0.27000	$0.81	Y	Y	
Digikey	Diodes Incorporated	BAT760-7	DIODE SCHOTTKY 30V 1A SOD323	3	0.47000	$1.41	Y	Y	
Digikey	KYOCERA AVX	08056D106KAT2A	CAP CER 10UF 6.3V X5R 0805	3	0.08000	$0.24	Y	Y	
Digikey	KEMET	C0805C104K5RACTU	CAP CER 0.1UF 50V X7R 0805	30	0.01600	$0.48	Y	Y	
Digikey	Samsung Electro-Mechanics	CL21A226MAYNNNE	CAP CER 22UF 25V X5R 0805	10	0.04300	$0.43	Y	Y	
Digikey	KEMET	C0805C104K5RACTU	CAP CER 0.1UF 50V X7R 0805	3	0.08000	$0.24	Y	Y	
Digikey	Samsung Electro-Mechanics	CL21A475KAQNNNE	CAP CER 4.7UF 25V X5R 0805	3	0.10000	$0.30	Y	Y	
Digikey	Panasonic Electronic Components	ERA-6AEB473V	RES SMD 47K OHM 0.1% 1/8W 0805	20	0.06100	$1.22	Y	Y	
Digikey	Stackpole Electronics Inc	RMCF0805FT22K1	RES 22.1K OHM 1% 1/8W 0805	3	0.10000	$0.30	Y	Y	
Digikey	YAGEO	RC0805FR-072K2L	RES 2.2K OHM 1% 1/8W 0805	10	0.01000	$0.10	Y	Y	
Digikey	YAGEO	RC0805JR-0709RL	RES 0 OHM JUMPER 1/8W 0805	10	0.01500	$0.15	Y	Y	
Digikey	Stackpole Electronics Inc	RMCF0805FT10K0	RES 10K OHM 1% 1/8W 0805	25	0.00800	$0.20	Y	Y	
Digikey	UMW	L9110S	SOP-8 MOTOR DRIVER ICS ROHS	6	0.59000	$3.54	Y	Y	
Digikey	Molex	1051330001	CONN RCPT USB2.0 MICRO B SMD	3	1.24000	$3.72	Y	Y	
Digikey	STMicroelectronics	VL53L5CXV0GC/1	SENSOR OPTICAL 20-10000CM I2C	3	8.53000	$25.59	Y	Y	
Digikey	onsemi	ESD5Z5.0T5G	TVS DIODE 5VWM 18.6VC SOD523	10	0.09900	$0.99	Y	Y	
Digikey	Alps Alpine	SKQBASA010	SWITCH TACTILE SPST-NO 0.05A 12V	6	1.01000	$6.03	Y	Y	
Digikey	Molex	5023820270	CONN RCPT 2POS 0.049 TIN SMD	6	0.58000	$3.48	Y	Y	
Digikey	Keystone Electronics	1042P	BATTERY HOLDER 18650 SMD	2	5.77000	$11.54	Y	Y	
Digikey	Comchip Technology	SS8050-G	TRANS NPN 25V 1.5A SOT-23-3	10	0.14800	$1.48	Y	Y	
Digikey	Kingbright	AP2012EC	LED RED CLEAR SMD	3	0.23000	$0.69	Y	Y	
Digikey	Molex	5023800700	CONN PLUG HSG 7POS 1.25MM	3	0.27000	$0.81	Y	Y	
Digikey	Molex	5023800200	CONN PLUG HSG 2POS 1.25MM	6	0.16000	$0.96	Y	Y	
Digikey	Würth Elektronik	9774027151R	ROUND STANDOFF M2.5 STEEL 2.70MM	2	1.01000	$2.02	Y	Y	
Digikey	Würth Elektronik	9774030360R	ROUND STANDOFF M3X0.5 STEEL 3MM	2	1.76000	$3.52	Y	Y	
Digikey	Adafruit Industries LLC	4654	MINIBOOST 5V @ 1A - TPS61023	2	3.95000	$7.90	Y	Y	
Digikey	ZEUS Battery Products	PCIFR18650-1500	BATTERY LITHIUM 3.2V 1.5AH 18650	2	5.85000	$11.70	Y	Y	
Digikey	Si Labs	CP2102N-A01-GQFN28R	USB UART BRIDGE	1	1.38000	$1.38	N	N	We are harvesting this component from the board
Digikey	Jauch Quartz	LI18650JL5 HB 3.35AH PROT	BATT LITH-ION 3.6V 3.25AH 18650	1	15.4800	$15.48	Y	Y	
Digikey	Yageo	RC0603JR-10100KL	RES 100K OHM 5% 1/10W 0603	20	0.00400	$0.08	Y	Y	
Digikey	Taiyo	NR3015T1R0N	FIXED IND 1UH 2.1A 36 MOHM SMD	5	0.21000	$1.05	Y	Y	
Digikey	Yageo	RC0603FR-07732KL	RES 732K OHM 1% 1/10W 0603	10	0.00600	$0.06	Y	Y	
Digikey	Murata	GRM188R61A226ME15D	CAP CER 22UF 10V X5R 0603	10	0.07800	$0.78	Y	Y	
OSHPARK	OSHPARK	xx	TOF Board	3	7.50000	$7.50	Y	Y	
OSHPARK	OSHPARK	xx	Main Board	3	71.6000	$71.60	Y	Y	
Amazon	Generic	xx	Generic Mouse toy	1	15.0000	$15.00	Y	Y	
NVIDIA	NVIDIA	xx	NVidia Jetson Orin Nano	1	249.990	$249.99	N	N	department's
Amazon	Generic	xx	Wide FOV webcam	1	5.00000	$5.00	Y	Y	
Amazon	Nordic	B0DP6MVDZQ	MDBT50Q-CX Nordic nRF52840 Dongle Developmen	1	9.80000	$9.80	Y	Y	
Amazon	Hosyond	761325160694	Hosyond 2Pack ESP-WROOM-32 ESP32 Developmen	1	12.9900	$12.99	Y	Y	
Amazon	Ruilezyo	B0FF1RS5SN	Ruilezyo DC 19V 3.42A Charger Compatible with JBL	1	7.99000	$7.99	Y	Y	
Amazon	Generic	xx	PLA	1	8.00000	$8.00	Y	Y	
Pololu	Pololu	2364	Pololu DC Gearmotor	2	23.9500	$47.90	Y	Y	
			Total						
			What we have spent			$295.48			

Figure 14: Budget for Project

L Gantt Chart

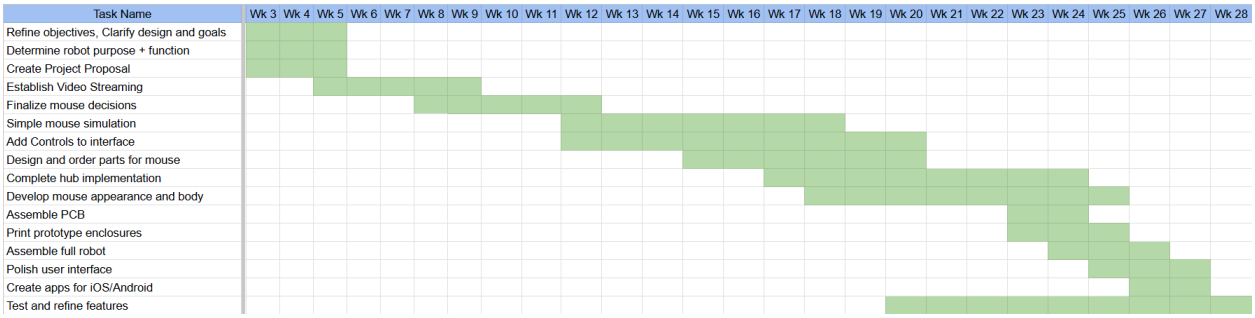


Figure 15: Gantt Chart for Semesters 1 and 2

M Engineering Decision Matrices

Table 6: Language Evaluation Criteria

Criteria	**Weight**	**Python**	**C++**	**JavaScript**
Performance (Real-time control)	0.25	7	9	6
Ease of Development	0.20	9	5	5
Computer Vision Support	0.20	10	9	4
WebRTC / Networking Support	0.15	8	5	10
Community / Libraries	0.10	10	9	9
Maintainability / Flexibility	0.10	9	6	8
Total	**1.00**	**8.7**	**7.2**	**6.5**

Table 7: Protocol Comparison

Criteria	**WebRTC**	**RTP**	**UDP**	**Custom Implementation**
Simplicity	9	7	4	1
Ubiquity	10	9	9	1
Security	9	6	4	2
Performance	9	9	9	7
Support	9	8	9	1
Total	**46**	**39**	**35**	**12**

Table 8: Camera Location

Criteria	**Weight**	**Birdseye View**	**Planar View**	**Mouse POV**
Latency	0.3	9	9	2
Path-finding Complexity	0.1	9	4	3
Battery Efficiency	0.1	10	10	2
Computation Load	0.1	9	4	2
Object Recognition	0.2	4	7	6
Area of Play	0.1	3	4	8
User Enjoyment	0.1	3	5	9
Total	**1**	**6.9**	**6.8**	**4.2**

Table 9: Enclosure Material

Criteria	**Weight**	**PLA**	**TPU**	**PETG**	**Nylon**	**ABA**	**ABS**
Durability	0.5	6	8	9	8	7	6
Non-Toxicity	0.3	9	9	9	7	1	1
Ease	0.2	9	5	9	2	5	5
Total	**1**	**7.5**	**7.7**	**9.0**	**6.5**	**4.8**	**4.3**

Table 10: Motor Selection

Criteria	**Weight**	**BLDC**	**DC**	**Stepper**
Speed Range	0.143	5	4	3
Low Speed Torque	0.2	4	3	5
Control Capability	0.143	2	5	3
Sensing Needs	0.143	3	4	5
Size	0.143	4	3	2
Weight	0.143	4	3	2
Cost	0.143	3	4	4
Total	**1**	**1.743**	**1.846**	**1.717**

N CAD Model Designs

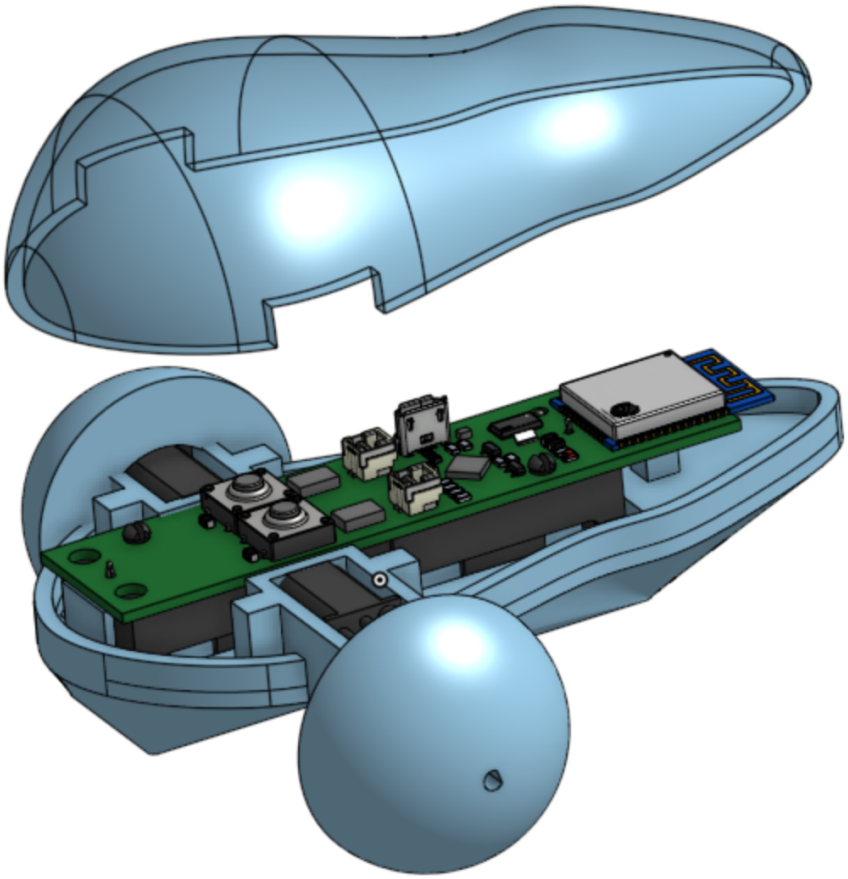


Figure 16: CAD Model View 1

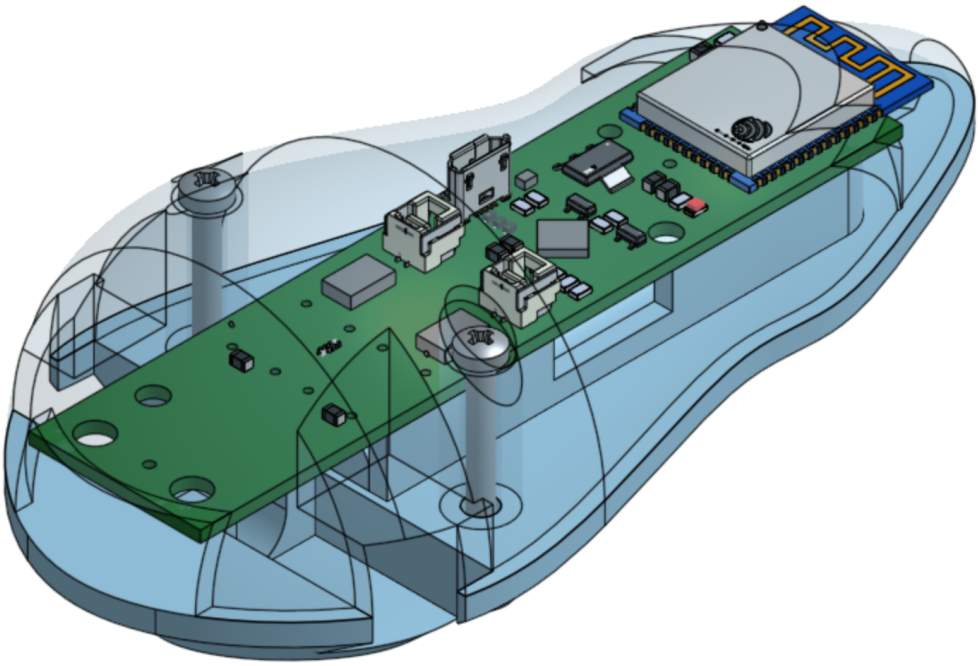


Figure 17: CAD Model View 2

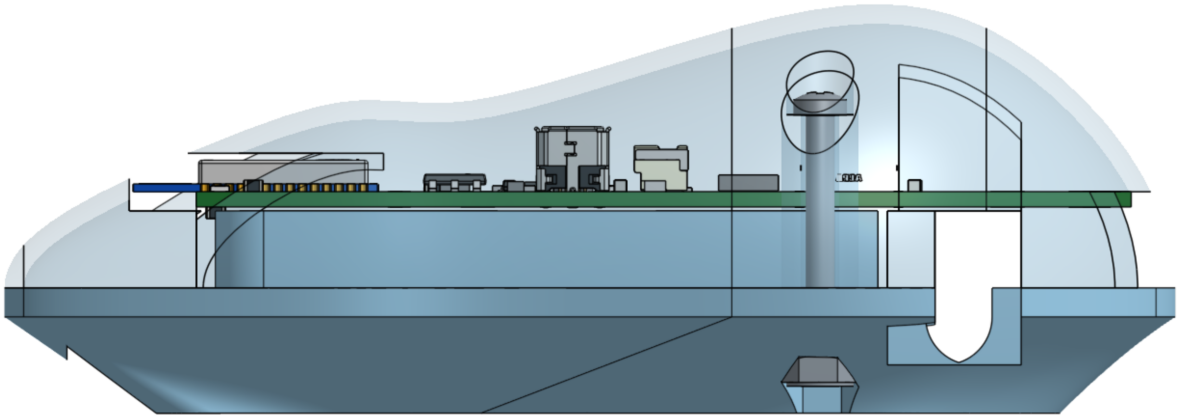


Figure 18: CAD Model View 3

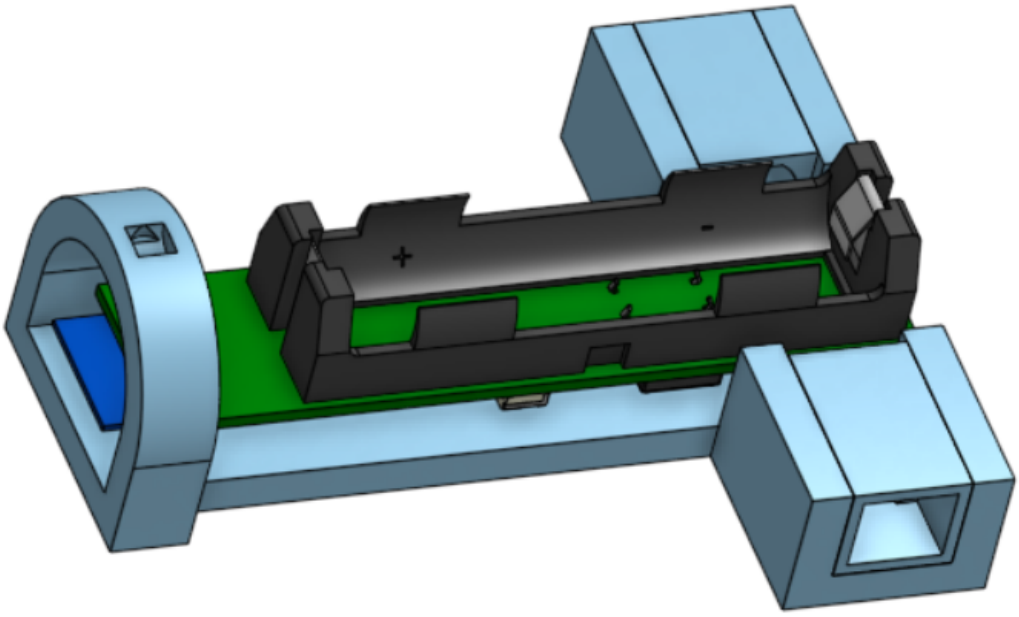


Figure 19: Early CAD Model Revision 1

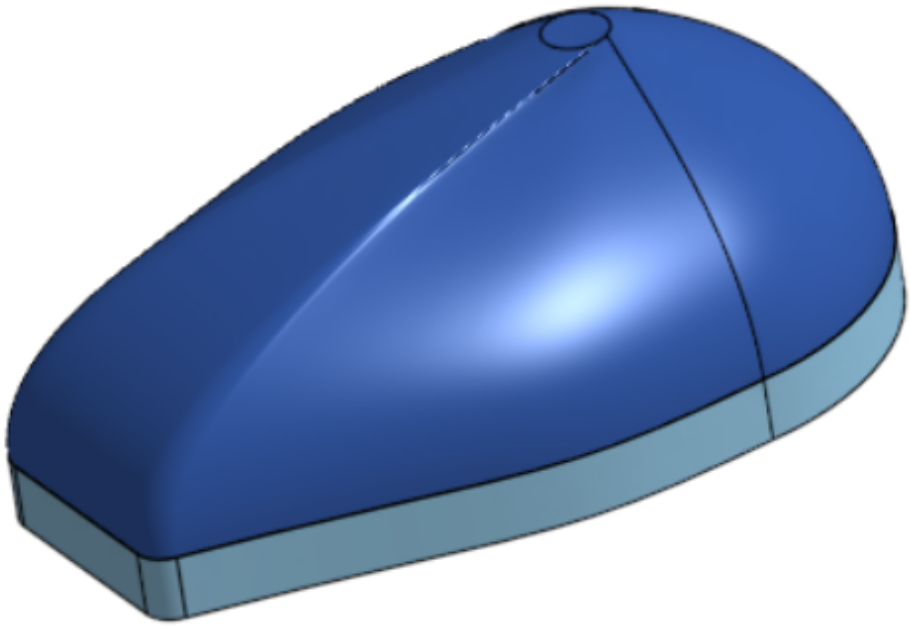


Figure 20: Early CAD Model Revision 2

O Bluetooth Motor Control Protocol

UART over BLE Command motor control BYT	0x	0	0	Right motor off
		0	1	Right motor slowest setting FWD
		0	2	
		0	3	
		0	4	
		0	5	
		0	6	
		0	7	Right motor fastest setting FWD
		0	8	Right motor off
		0	9	Right motor slowest setting REV
		0	A	
		0	B	
		0	C	
		0	D	
		0	E	
		0	F	Right motor fastest setting REV
		0	0	Left motor off
		1	0	Left motor slowest setting FWD
		2	0	
		3	0	
		4	0	
		5	0	
		6	0	
		7	0	Left motor fastest setting FWD
		8	0	Left motor off
		9	0	Left motor slowest setting REV
		A	0	
		B	0	
		C	0	
		D	0	
		E	0	
		F	0	Left motor fastest setting REV

Figure 21: BLE Motor Control Protocol

P Power Supply Retrofit

Type-C 5V 3A Boost Converter Step-Up Power Module IP5310 Mobile Power Bank Accessories with Switch LED Indicator(Type-C and USB-A 3A)

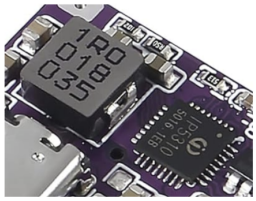
Brand: ATNSINC

4.4 ★★★★★ (18) | Search this page

Amazon's Choice

$11.99

Price history



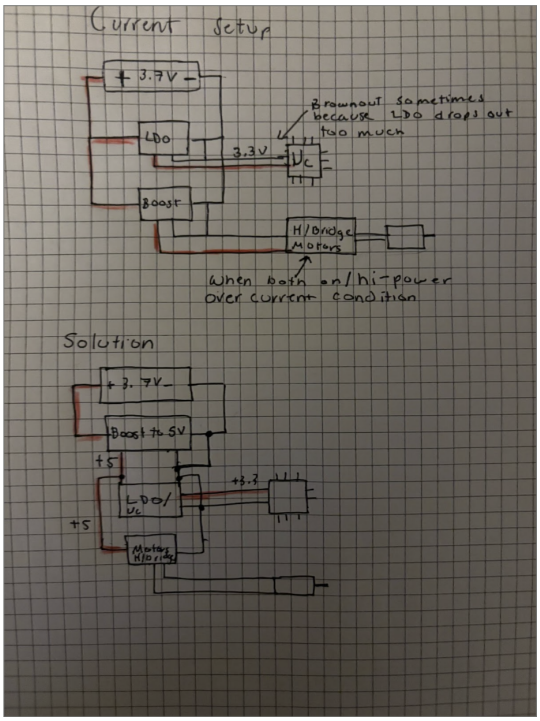


Figure 22: Power Supply Retrofit Part and Schematic

Q Video Streaming and Control Server

```
import asyncio
import json
import logging
import numpy as np
import cv2
import av
import ssl
from aiohttp import web
from aiortc import RTCPeerConnection, RTCSessionDescription, VideoStreamTrack
import socket
import os
import secrets
import time
import math
import threading
from collections import defaultdict, deque
import heapq
from cryptography import x509
from cryptography.x509.oid import NameOID
from cryptography.hazmat.primitives import serialization, hashes
from cryptography.hazmat.primitives.asymmetric import rsa
import datetime
from Mouse import Mouse

print(socket.gethostname())
logging.basicConfig(level=logging.INFO)
logging.getLogger("aioice").setLevel(logging.WARNING)
logging.getLogger("asyncio").setLevel(logging.CRITICAL)

#  CONFIG
PASSWORD         = "Cat123"
SESSION_DURATION = 60 * 60 * 8
MAX_LOGIN_ATTEMPTS = 5

_sessions        = {}
_login_attempts  = defaultdict(list)
_active_viewers  = {}
_log_subscribers = []
_mouse           = None

def server_log(msg: str, level: str = "info"):
    print(f"[{level.upper()}] {msg}")
    payload = json.dumps({"msg": msg, "level": level})
    dead = []
    for q in _log_subscribers:
        try:
            q.put_nowait(payload)
        except Exception:
            dead.append(q)
    for q in dead:
        _log_subscribers.remove(q)
```

```
def sse_only(msg: str, level: str = "info"):
    """Push to browser SSE subscribers without printing to terminal."""
    payload = json.dumps({"msg": msg, "level": level})
    dead = []
    for q in _log_subscribers:
        try:
            q.put_nowait(payload)
        except Exception:
            dead.append(q)
    for q in dead:
        _log_subscribers.remove(q)

#  YOLO + PATHFINDER CONFIG
MODEL_PATH     = "mouse_model.pt"
ROBOT_CLASS    = "mouse"
INFER_SIZE     = 320
ROBOT_CONF     = 0.35        # lowered  catches more detections, shape filter handles
↪  false positives
THREAT_CONF    = 0.45        # general model threshold
INFER_INTERVAL = 0.10
USE_HALF       = True

FRAME_W, FRAME_H = 640, 480

PLAY_X1, PLAY_Y1 = 80,  60
PLAY_X2, PLAY_Y2 = 560, 420
BOUNDARY_MARGIN  = 40
EDGE_REPULSION   = 2.5

# Robot detection filtering | reject detections that don't look like the mouse
ROBOT_MIN_AREA         = 400    # px² | sock on floor is usually larger or smaller
ROBOT_MAX_AREA         = 8000   # ignore huge blobs
ROBOT_MAX_ASPECT       = 3.5    # width/height or height/width | mouse is roughly square
ROBOT_CONFIRM_FRAMES   = 1      # single frame confirm  shape filter handles false
↪  positives

MOUSE_SPEED            = 4.0
AVOID_RADIUS           = 110
NUM_SAMPLE_DIRECTIONS  = 12
SAMPLE_DISTANCE        = 30
CURRENT_DIR_BIAS       = 2.0
HEAT_THRESHOLD         = 50
COOL_SEEKING_STRENGTH  = 2.0
EMERGENCY_ESCAPE_FORCE = 15.0
STUCK_THRESHOLD        = 8.0    # px/s in virtual space
STUCK_TIME_LIMIT       = 2.5
TEMPORAL_SMOOTHING     = 5
MIN_OBJECT_SIZE        = 300
GHOST_FRAMES           = 10

THREAT_OBJECTS   = {'person', 'cat', 'dog', 'bird'}
OBSTACLE_OBJECTS = {'chair','couch','sofa','table','bed','bench',
                    'dining table','potted plant','tv','laptop'}
```

```
ENABLE_SAFE_ZONE   = True
SAFE_ZONE_CENTER_X = FRAME_W // 4
SAFE_ZONE_CENTER_Y = FRAME_H // 2
SAFE_ZONE_RADIUS   = 60
TRIGGER_OBJECTS    = {'cat', 'dog', 'person'}

DRIVE_SCALE    = 1.0
DRIVE_DEADZONE = 0.08

#  AUTO-MODE STATE MACHINE
# States: idle → navigate → flee
# Navigation uses position-feedback only | no heading sensor needed.
# The robot always keeps moving forward; steering is derived from how the
# detected position actually moved between frames (which way IS forward).

PROBE_DURATION     = 0   # unused | probe removed
# These are set by calibration; nav_drive reads them at runtime
AUTO_FWD_LEVEL     = 3    # motor level that reliably moves forward (1-7)
AUTO_TURN_LEVEL    = 3    # level used for spin turns (same wheel fwd/rev)
AUTO_PX_PER_SEC    = None # px/second at AUTO_FWD_LEVEL (set by calibration)
AUTO_DEG_PER_SEC   = None # degrees/second of spin at AUTO_TURN_LEVEL (set by
↪  calibration)

EDGE_SLOW_MARGIN   = 80
EDGE_SLOW_MIN      = 0.25

NAV_WAYPOINT_R     = 40
NAV_MIN_MOVE_PX    = 5    # px | minimum movement to count as "actually moved"
NAV_MOVE_THRESHOLD = 8    # px | minimum to confirm forward motion during calibration
NAV_TURN_GAIN      = 1.0

#  CALIBRATION STATE
_cal_data          = {}    # raw measurements from last calibration run
_cal_running       = False

#  LOAD YOLO
# Mouse model on GPU (fast, custom). Threat model on CPU to avoid dual-GPU OOM on Jetson.
ROBOT_DEVICE  = 0      # GPU
THREAT_DEVICE = "cpu"  # CPU | avoids NvMapMemAllocInternalTagged error 12

server_log(f"Loading mouse model (GPU): {MODEL_PATH}", "info")
from ultralytics import YOLO
import torch, os as _os
yolo_model = YOLO(MODEL_PATH)
_dummy = np.zeros((INFER_SIZE, INFER_SIZE, 3), dtype=np.uint8)
# Suppress Jetson NvMap driver noise during warmup
_dn = open(_os.devnull, 'w'); _old_se = _os.dup(2); _os.dup2(_dn.fileno(), 2)
yolo_model.predict(_dummy, imgsz=INFER_SIZE, conf=ROBOT_CONF,
                   device=ROBOT_DEVICE, half=USE_HALF, verbose=False)
_os.dup2(_old_se, 2); _os.close(_old_se); _dn.close()
server_log(f"Mouse model ready | classes: {yolo_model.names}", "success")

THREAT_MODEL_PATH = "yolov8n.pt"
server_log(f"Loading threat model (CPU): {THREAT_MODEL_PATH}", "info")
```

```
threat_model = YOLO(THREAT_MODEL_PATH)
threat_model.predict(_dummy, imgsz=INFER_SIZE, conf=THREAT_CONF,
                     device=THREAT_DEVICE, half=False, verbose=False)
server_log("Threat model ready", "success")

#  SHARED FRAME STATE
# Pathfinder writes annotated frames here; camera track reads them
_frame_lock       = threading.Lock()
_latest_frame     = None          # raw camera frame
_annotated_frame  = None          # frame with overlays drawn on it

# Pathfinder state
_pf_lock             = threading.Lock()
_virt_pos            = np.array([FRAME_W//2, FRAME_H//2], dtype=np.float32)
_virt_vel            = np.zeros(2, dtype=np.float32)
_last_robot_pos      = None
_last_robot_pos_time = 0.0
DETECTION_EXPIRY     = 2.0
_low_speed_timer     = 0.0
_persistent_obstacles= {}
_obstacle_history    = deque(maxlen=TEMPORAL_SMOOTHING)

# Position history for heading estimation (ring buffer of confirmed positions)
_pos_history         = deque(maxlen=8)

# Detection confirmation buffer
_robot_confirm_buf   = deque(maxlen=ROBOT_CONFIRM_FRAMES)

# Click-to-navigate target
_nav_target          = None

# Inference state
_infer_lock      = threading.Lock()
_latest_preds    = []
_last_infer_time = 0.0
_infer_running   = False

# Manual drive override (from browser joystick)
_manual_drive    = False   # True = browser joystick controls, False = pathfinder

# Auto-mode state machine
_auto_state      = "idle"   # "idle" | "navigate" | "flee"
_auto_state_time = 0.0       # time when current state was entered

#  PATHFINDER HELPERS
def get_rect_repulsion(pos):
    rx, ry = 0.0, 0.0
    dl = pos[0]-PLAY_X1; dr = PLAY_X2-pos[0]
    dt = pos[1]-PLAY_Y1; db = PLAY_Y2-pos[1]
    if dl < BOUNDARY_MARGIN: rx +=  EDGE_REPULSION*((BOUNDARY_MARGIN-dl)/BOUNDARY_MARGIN)
    if dr < BOUNDARY_MARGIN: rx -= EDGE_REPULSION*((BOUNDARY_MARGIN-dr)/BOUNDARY_MARGIN)
    if dt < BOUNDARY_MARGIN: ry +=  EDGE_REPULSION*((BOUNDARY_MARGIN-dt)/BOUNDARY_MARGIN)
    if db < BOUNDARY_MARGIN: ry -= EDGE_REPULSION*((BOUNDARY_MARGIN-db)/BOUNDARY_MARGIN)
    return np.array([rx, ry], dtype=np.float32)
```

```
def clamp_to_rect(pos):
    pos[0] = np.clip(pos[0], PLAY_X1+2, PLAY_X2-2)
    pos[1] = np.clip(pos[1], PLAY_Y1+2, PLAY_Y2-2)
    return pos

def calculate_heat(pos, obstacles):
    heat = 0.0
    for obs in obstacles:
        box = obs['box']
        cx = (box[0]+box[2])/2; cy = (box[1]+box[3])/2
        if obs['is_target']:
            d = math.hypot(pos[0]-cx, pos[1]-cy)
            if d < AVOID_RADIUS:
                heat += ((AVOID_RADIUS-d)/AVOID_RADIUS)**2 * 10.0
        else:
            dx = max(box[0]-pos[0], 0, pos[0]-box[2])
            dy = max(box[1]-pos[1], 0, pos[1]-box[3])
            d  = math.hypot(dx, dy)
            if d < HEAT_THRESHOLD:
                heat += ((HEAT_THRESHOLD-d)/HEAT_THRESHOLD)*3.0
    px = (pos[0]-(PLAY_X1+PLAY_X2)/2)/((PLAY_X2-PLAY_X1)/2)
    py = (pos[1]-(PLAY_Y1+PLAY_Y2)/2)/((PLAY_Y2-PLAY_Y1)/2)
    nd = math.hypot(px, py)
    if nd > 0.7:
        heat += ((nd-0.7)/0.3)**2 * 5.0
    return heat

def find_path(start_pos, goal_pos, obstacles):
    start_pt = tuple(start_pos.astype(int))
    goal_pt  = tuple(goal_pos.astype(int))
    waypoints = [start_pt]
    path_vec  = goal_pos - start_pos
    path_len  = np.linalg.norm(path_vec)
    if path_len < 50:
        return [start_pt, goal_pt]
    path_dir = path_vec / path_len
    perp_dir = np.array([-path_dir[1], path_dir[0]])
    for step in range(1, 5):
        t    = step/5
        base = start_pos + path_vec*t
        for offset in [-60,-30,0,30,60]:
            wp    = base + perp_dir*offset
            wp[0] = np.clip(wp[0], PLAY_X1+10, PLAY_X2-10)
            wp[1] = np.clip(wp[1], PLAY_Y1+10, PLAY_Y2-10)
            waypoints.append((int(wp[0]), int(wp[1])))
    waypoints.append(goal_pt)
    def h(a,b): return math.hypot(a[0]-b[0], a[1]-b[1])
    def cost(a,b):
        dc = h(a,b)
        ht = sum(calculate_heat(
            np.array([a[0]*(1-t)+b[0]*t, a[1]*(1-t)+b[1]*t], dtype=np.float32),
            obstacles) for t in np.linspace(0,1,5))/5
        return dc*(1+ht*1.5)
```

```
    open_set=[(0,start_pt)]; came_from={}
    g={w:float('inf') for w in waypoints}; g[start_pt]=0
    f={w:float('inf') for w in waypoints}; f[start_pt]=h(start_pt,goal_pt)
    while open_set:
        _,cur=heapq.heappop(open_set)
        if cur==goal_pt:
            path=[]
            while cur in came_from:
                path.append(cur); cur=came_from[cur]
            path.append(start_pt); path.reverse()
            return path
        for nb in waypoints:
            if nb==cur or h(cur,nb)>path_len*0.5: continue
            tg=g[cur]+cost(cur,nb)
            if tg<g[nb]:
                came_from[nb]=cur; g[nb]=tg; f[nb]=tg+h(nb,goal_pt)
                if nb not in [x[1] for x in open_set]:
                    heapq.heappush(open_set,(f[nb],nb))
    return [start_pt, goal_pt]

def smooth_obstacles(current):
    global _persistent_obstacles
    _obstacle_history.append(current)
    matched = set()
    for obs in current:
        box=obs['box']; cx=(box[0]+box[2])/2; cy=(box[1]+box[3])/2
        best_id, best_d = None, float('inf')
        for oid,info in _persistent_obstacles.items():
            eb=info['data']['box']
            ecx=(eb[0]+eb[2])/2; ecy=(eb[1]+eb[3])/2
            d=math.hypot(cx-ecx,cy-ecy)
            if info['data']['name']==obs['name'] and d<80 and d<best_d:
                best_d=d; best_id=oid
        oid=best_id or f"{obs['name']}_{int(cx/50)}_{int(cy/50)}_{id(obs)}"
        _persistent_obstacles[oid]={'data':obs,'frames_missing':0,'is_ghost':False}
        matched.add(oid)
    for oid in list(_persistent_obstacles):
        if oid not in matched:
            _persistent_obstacles[oid]['frames_missing']+=1
            _persistent_obstacles[oid]['is_ghost']=True
            if _persistent_obstacles[oid]['frames_missing']>GHOST_FRAMES:
                del _persistent_obstacles[oid]
    result=[]
    for info in _persistent_obstacles.values():
        d=info['data'].copy(); d['is_ghost']=info['is_ghost']
        result.append(d)
    return result

def _get_heading():
    """
    Return current heading in radians from _pos_history, or None if unreliable.
    Heading is the direction the robot actually traveled (oldest→newest confirmed pos).
    Only trusted when net displacement is straight enough.
    """
```

```
    if len(_pos_history) < 4:
        return None
    oldest   = _pos_history[0]
    newest   = _pos_history[-1]
    net_move = newest - oldest
    net_dist = float(np.linalg.norm(net_move))
    total_path = sum(
        float(np.linalg.norm(_pos_history[i] - _pos_history[i-1]))
        for i in range(1, len(_pos_history))
    )
    straightness = net_dist / (total_path + 1e-6)
    if net_dist >= NAV_MIN_MOVE_PX * 2 and straightness > 0.35:
        return math.atan2(float(net_move[1]), float(net_move[0]))
    return None


def _seed_heading(timeout=2.0):
    """
    Try to establish heading without moving the robot.
    First waits up to 1s watching _pos_history | if the robot drifted
    even a little when placed down, that's enough.
    Only if we still have nothing after waiting does it issue ONE tiny
    0.15s nudge at AUTO_FWD_LEVEL, then waits again.
    Returns heading in radians, or None.
    """
    # Phase 1: passive wait | no driving
    t0 = time.time()
    while time.time() - t0 < 1.0:
        if _manual_drive: return None
        if time.time() - _last_robot_pos_time > DETECTION_EXPIRY: return None
        hdg = _get_heading()
        if hdg is not None:
            return hdg
        time.sleep(0.05)

    # Phase 2: one tiny nudge only
    server_log("Heading unknown | tiny nudge to establish direction", "info")
    t_nudge = time.time()
    while time.time() - t_nudge < 0.15:
        if _manual_drive: return None
        _mouse.drive_raw(AUTO_FWD_LEVEL, AUTO_FWD_LEVEL)
        _mouse.fallback.start()
        time.sleep(0.05)
    _mouse.drive_raw(0, 0)
    time.sleep(0.3)

    # Phase 3: wait for history from the nudge
    t1 = time.time()
    while time.time() - t1 < 1.0:
        if _manual_drive: return None
        hdg = _get_heading()
        if hdg is not None:
            return hdg
        time.sleep(0.05)
```

```
    return None


def execute_nav_segment(goal_px, label="WP"):
    """
    Blocking navigator for a single waypoint. Called from the nav worker thread.

    Algorithm:
      1. Get current heading (seed it with a short forward burst if needed).
      2. Compute turn angle to face goal.
      3. If calibrated: spin for angle/deg_per_sec seconds, checking heading each 0.1s.
         If not calibrated: proportional inner-wheel arc toward goal.
      4. Drive forward for dist/px_per_sec seconds (or until camera confirms arrival).
      5. Re-check position; if close enough return True, else caller re-plans.

    Returns True if waypoint reached, False if aborted (manual mode / lost detection).
    """
    global _pos_history

    ALIGN_TOLERANCE_DEG = 12   # degrees | consider aligned within this
    DRIVE_CHECK_INTERVAL = 0.15 # seconds between position re-checks while driving
    MAX_ALIGN_ATTEMPTS  = 3    # re-align passes before giving up and driving anyway

    if _mouse is None:
        return False

    #  Step 1: get position
    pos = _last_robot_pos if _last_robot_pos is not None else _virt_pos
    goal_px = np.array(goal_px, dtype=np.float32)
    diff = goal_px - pos
    dist = float(np.linalg.norm(diff))

    if dist < NAV_WAYPOINT_R:
        return True   # already there

    #  Step 2: get or seed heading
    heading = _get_heading()
    if heading is None:
        heading = _seed_heading(timeout=2.5)
        if _manual_drive: return False
        if heading is None:
            server_log(f"{label}: could not establish heading | driving blind", "warn")

    #  Step 3: orient toward goal
    if heading is not None:
        goal_angle = math.atan2(float(diff[1]), float(diff[0]))
        err = goal_angle - heading
        while err >  math.pi: err -= 2*math.pi
        while err < -math.pi: err += 2*math.pi
        err_deg = math.degrees(err)

        if abs(err_deg) > ALIGN_TOLERANCE_DEG:
            if AUTO_DEG_PER_SEC and AUTO_DEG_PER_SEC > 0:
```

```
            # Predictive timed spin
            turn_time = abs(err_deg) / AUTO_DEG_PER_SEC
            server_log(f"{label}: turn {err_deg:+.1f}  spin {turn_time:.2f}s",
            ↪  "info")

            # Spin in the correct direction
            if err_deg > 0:
                left_spin, right_spin = AUTO_TURN_LEVEL, -AUTO_TURN_LEVEL  # CCW
            else:
                left_spin, right_spin = -AUTO_TURN_LEVEL, AUTO_TURN_LEVEL  # CW

            t_spin = time.time()
            while time.time() - t_spin < turn_time:
                if _manual_drive: return False
                _mouse.drive_raw(left_spin, right_spin)
                _mouse.fallback.start()
                time.sleep(0.05)
            _mouse.drive_raw(0, 0)
            time.sleep(0.25)   # settle + let camera update
            _pos_history.clear()  # old history no longer reflects new heading

            # Verify and fine-correct (up to MAX_ALIGN_ATTEMPTS passes)
            for attempt in range(MAX_ALIGN_ATTEMPTS):
                # Seed heading in new direction
                new_heading = _seed_heading(timeout=2.0)
                if _manual_drive: return False
                if new_heading is None:
                    break

                pos = _last_robot_pos if _last_robot_pos is not None else _virt_pos
                diff = goal_px - pos
                goal_angle = math.atan2(float(diff[1]), float(diff[0]))
                residual = goal_angle - new_heading
                while residual >  math.pi: residual -= 2*math.pi
                while residual < -math.pi: residual += 2*math.pi
                residual_deg = math.degrees(residual)

                if abs(residual_deg) <= ALIGN_TOLERANCE_DEG:
                    server_log(f"{label}: aligned  (residual {residual_deg:+.1f})",
                    ↪  "info")
                    break

                # Correction spin
                corr_time = abs(residual_deg) / AUTO_DEG_PER_SEC
                server_log(f"{label}: correction {residual_deg:+.1f}
                ↪  {corr_time:.2f}s", "info")
                if residual_deg > 0:
                    cl, cr = AUTO_TURN_LEVEL, -AUTO_TURN_LEVEL
                else:
                    cl, cr = -AUTO_TURN_LEVEL, AUTO_TURN_LEVEL
                t_corr = time.time()
                while time.time() - t_corr < corr_time:
                    if _manual_drive: return False
                    _mouse.drive_raw(cl, cr)
```

```
                    _mouse.fallback.start()
                    time.sleep(0.05)
                _mouse.drive_raw(0, 0)
                time.sleep(0.2)
                _pos_history.clear()
        else:
            # No calibration | do a rough proportional arc for 0.5s
            server_log(f"{label}: no turn calibration | arc toward goal", "info")
            if err_deg > 0:
                _mouse.drive_raw(1, AUTO_FWD_LEVEL)
            else:
                _mouse.drive_raw(AUTO_FWD_LEVEL, 1)
            time.sleep(0.5)
            _mouse.drive_raw(0, 0)
            time.sleep(0.2)

#  Step 4: drive forward toward goal
pos = _last_robot_pos if _last_robot_pos is not None else _virt_pos
diff = goal_px - pos
dist = float(np.linalg.norm(diff))

if dist < NAV_WAYPOINT_R:
    return True

if AUTO_PX_PER_SEC and AUTO_PX_PER_SEC > 0:
    drive_time = dist / AUTO_PX_PER_SEC
    # Cap to avoid overshooting badly | we'll re-evaluate after
    drive_time = min(drive_time, 4.0)
    server_log(f"{label}: drive {dist:.0f}px → {drive_time:.2f}s fwd", "info")

    t_drive = time.time()
    while time.time() - t_drive < drive_time:
        if _manual_drive: return False
        _mouse.drive_raw(AUTO_FWD_LEVEL, AUTO_FWD_LEVEL)
        _mouse.fallback.start()

        # Re-check position mid-drive
        if time.time() - t_drive > DRIVE_CHECK_INTERVAL:
            cur = _last_robot_pos if _last_robot_pos is not None else _virt_pos
            remaining = float(np.linalg.norm(goal_px - cur))
            if remaining < NAV_WAYPOINT_R:
                break   # arrived early
        time.sleep(0.05)
    _mouse.drive_raw(0, 0)
    time.sleep(0.2)
    _pos_history.clear()   # fresh heading after this leg
else:
    # No speed calibration | drive for a fixed short burst
    _mouse.drive_raw(AUTO_FWD_LEVEL, AUTO_FWD_LEVEL)
    _mouse.fallback.start()
    time.sleep(0.6)
    _mouse.drive_raw(0, 0)
    time.sleep(0.2)
    _pos_history.clear()
```

```
    #  Step 5: check arrival
    time.sleep(0.1)   # camera settle
    pos = _last_robot_pos if _last_robot_pos is not None else _virt_pos
    dist_final = float(np.linalg.norm(goal_px - pos))
    reached = dist_final < NAV_WAYPOINT_R * 1.5  # slightly generous on final check
    server_log(f"{label}: arrived at dist {dist_final:.0f}px {'' if reached else '(close
    ↪  enough  continue)'}", "info")
    return reached


# nav worker thread state
_nav_thread       = None
_nav_thread_goal  = None
_nav_thread_done  = False
_nav_thread_result = False


def _nav_worker(waypoints, on_complete_state):
    """
    Worker thread: execute each waypoint in sequence using execute_nav_segment.
    Updates _auto_state when finished.
    """
    global _auto_state, _auto_state_time, _nav_thread_done, _nav_thread_result
    idx = 0
    while idx < len(waypoints):
        if _manual_drive:
            break
        wp = np.array(waypoints[idx], dtype=np.float32)
        label = f"WP{idx+1}/{len(waypoints)}"
        reached = execute_nav_segment(wp, label)
        if _manual_drive:
            break
        if reached or True:   # always advance | re-planning happens at loop level
            idx += 1

    _mouse.drive_raw(0, 0)
    _nav_thread_done = True
    _nav_thread_result = not _manual_drive
    _auto_state      = on_complete_state
    _auto_state_time = time.time()
    server_log(f"Nav complete  {on_complete_state}", "info" if not _manual_drive else
    ↪  "warn")

def edge_limit_axes(joy_x, joy_y, pos):
    """
    Per-axis edge enforcement for manual driving.

    Two behaviours depending on whether the robot is inside or outside the
    play area on each axis:

    INSIDE (within EDGE_SLOW_MARGIN of an edge):
      - Scale the component pushing toward that edge down to EDGE_SLOW_MIN.
      - Components pushing away from the edge are unaffected.
```

```
    OUTSIDE (past the play-area boundary on an axis):
      - Hard-zero any component that would push further out.
      - Allow full speed back toward the play area so the driver can recover.

    This means controls never invert | the worst case is the joystick feels
    sluggish near an edge, or does nothing when already outside.
    """
    x, y = joy_x, joy_y

    px, py = pos[0], pos[1]

    #  X axis
    left_dist  = px - PLAY_X1   # positive = inside left edge
    right_dist = PLAY_X2 - px  # positive = inside right edge

    if left_dist < 0:
        # Already outside left wall | block leftward motion (joy_x < 0)
        if x < 0:
            x = 0.0
    elif left_dist < EDGE_SLOW_MARGIN:
        # Approaching left wall | scale leftward component
        if x < 0:
            t = left_dist / EDGE_SLOW_MARGIN
            x *= EDGE_SLOW_MIN + (1.0 - EDGE_SLOW_MIN) * t

    if right_dist < 0:
        # Already outside right wall | block rightward motion (joy_x > 0)
        if x > 0:
            x = 0.0
    elif right_dist < EDGE_SLOW_MARGIN:
        # Approaching right wall | scale rightward component
        if x > 0:
            t = right_dist / EDGE_SLOW_MARGIN
            x *= EDGE_SLOW_MIN + (1.0 - EDGE_SLOW_MIN) * t

    #  Y axis
    # joy_y > 0 = forward = camera-up = decreasing pixel-Y, so:
    #   joy_y > 0 pushes toward PLAY_Y1 (top)
    #   joy_y < 0 pushes toward PLAY_Y2 (bottom)
    top_dist    = py - PLAY_Y1
    bottom_dist = PLAY_Y2 - py

    if top_dist < 0:
        # Outside top | block forward motion (joy_y > 0)
        if y > 0:
            y = 0.0
    elif top_dist < EDGE_SLOW_MARGIN:
        if y > 0:
            t = top_dist / EDGE_SLOW_MARGIN
            y *= EDGE_SLOW_MIN + (1.0 - EDGE_SLOW_MIN) * t

    if bottom_dist < 0:
        # Outside bottom | block backward motion (joy_y < 0)
```

```
        if y < 0:
            y = 0.0
    elif bottom_dist < EDGE_SLOW_MARGIN:
        if y < 0:
            t = bottom_dist / EDGE_SLOW_MARGIN
            y *= EDGE_SLOW_MIN + (1.0 - EDGE_SLOW_MIN) * t

    return x, y


def manual_drive_with_edge_limit(joy_x, joy_y, robot_pixel_pos):
    """Apply per-axis edge clamping then drive.
    Uses detected robot position when available, falls back to virtual position."""
    if _mouse is None:
        return
    pos = robot_pixel_pos if robot_pixel_pos is not None else _virt_pos
    joy_x, joy_y = edge_limit_axes(joy_x, joy_y, pos)
    _mouse.drive(joy_x, joy_y)

#  CALIBRATION
def _get_robot_pos(timeout=3.0, after_time=None):
    """Wait for a fresh robot detection. after_time ensures we get a NEW position after a
    ↪  move."""
    t0 = time.time()
    while time.time() - t0 < timeout:
        if (_last_robot_pos is not None and
                _last_robot_pos_time > (after_time if after_time else 0)):
            return _last_robot_pos.copy()
        time.sleep(0.05)
    return None

def _drive_and_wait(left, right, duration):
    """Send drive_raw command repeatedly for duration seconds, then stop."""
    t0 = time.time()
    while time.time() - t0 < duration:
        _mouse.drive_raw(left, right)
        time.sleep(0.05)
    _mouse.drive_raw(0, 0)
    time.sleep(0.25)

def run_calibration():
    global _cal_data, _cal_running, _manual_drive
    global AUTO_FWD_LEVEL, AUTO_TURN_LEVEL, AUTO_PX_PER_SEC, AUTO_DEG_PER_SEC

    if _mouse is None:
        server_log("Calibration failed | mouse not connected", "error")
        _cal_running = False
        return

    _manual_drive = True
    server_log("Cal phase 1: finding minimum speed", "info")
    found_level = None
    found_px_s  = None
```

```
    try:
        for level in range(2, 8):
            server_log(f"Testing level {level}...", "info")
            start_pos = _get_robot_pos(timeout=3.0)
            if start_pos is None:
                server_log("Robot not detected | aborting", "error")
                return

            t_move = time.time()
            _drive_and_wait(level, level, 0.4)
            end_pos = _get_robot_pos(timeout=2.0, after_time=t_move)

            if end_pos is None:
                server_log(f"Level {level}: lost detection", "warn")
                continue

            dist = float(np.linalg.norm(end_pos - start_pos))
            server_log(f"Level {level}: moved {dist:.1f}px", "info")

            if dist >= NAV_MOVE_THRESHOLD:
                found_px_s  = dist / 0.4
                found_level = level
                server_log(f"Min level={level} ({found_px_s:.1f}px/s)", "success")
                break
            else:
                server_log(f"Level {level}: stalled", "info")
                time.sleep(0.3)

        if found_level is None:
            server_log("Calibration failed | no level moved the robot", "error")
            return

        AUTO_FWD_LEVEL  = found_level
        AUTO_TURN_LEVEL = found_level
        AUTO_PX_PER_SEC = found_px_s

        # Phase 2: heading before spin (two short forward moves)
        server_log("Cal phase 2: turn rate", "info")
        time.sleep(0.3)

        t1 = time.time()
        _drive_and_wait(found_level, found_level, 0.35)
        p1 = _get_robot_pos(timeout=2.0, after_time=t1)
        t2 = time.time()
        _drive_and_wait(found_level, found_level, 0.35)
        p2 = _get_robot_pos(timeout=2.0, after_time=t2)

        if p1 is None or p2 is None:
            server_log("Phase 2: detection failed before spin | skipping", "warn")
            _cal_data = {"fwd_level": found_level, "px_per_s": round(found_px_s,1),
            ↪  "deg_per_s": None}
            server_log(f"Calibration complete  level={found_level}, {found_px_s:.1f}px/s
            ↪  fwd, turn=unknown", "success")
            sse_only('cal_done_manual', "info")
```

```
            return

        pre_angle = math.degrees(math.atan2(float(p2[1]-p1[1]), float(p2[0]-p1[0])))
        server_log(f"Pre-spin heading: {pre_angle:.1f}deg", "info")

        SPIN_DURATION = 0.8
        t_spin = time.time()
        while time.time() - t_spin < SPIN_DURATION:
            _mouse.drive_raw(found_level, -found_level)
            time.sleep(0.05)
        _mouse.drive_raw(0, 0)
        time.sleep(0.35)

        t3 = time.time()
        _drive_and_wait(found_level, found_level, 0.35)
        p3 = _get_robot_pos(timeout=2.0, after_time=t3)
        t4 = time.time()
        _drive_and_wait(found_level, found_level, 0.35)
        p4 = _get_robot_pos(timeout=2.0, after_time=t4)

        deg_per_sec = None
        if p3 is not None and p4 is not None:
            post_angle = math.degrees(math.atan2(float(p4[1]-p3[1]), float(p4[0]-p3[0])))
            delta = post_angle - pre_angle
            while delta >  180: delta -= 360
            while delta < -180: delta += 360
            deg_per_sec = abs(delta) / SPIN_DURATION
            AUTO_DEG_PER_SEC = deg_per_sec
            server_log(f"Turn rate={deg_per_sec:.1f}deg/s ({delta:.1f}deg in
            ↪  {SPIN_DURATION}s)", "success")
        else:
            server_log("Phase 2: lost detection after spin", "warn")

        _cal_data = {
            "fwd_level": found_level,
            "px_per_s":  round(found_px_s, 1),
            "deg_per_s": round(deg_per_sec, 1) if deg_per_sec else None,
        }
        server_log(
            f"Calibration complete | level={found_level}, {found_px_s:.1f}px/s fwd"
            + (f", {deg_per_sec:.1f}deg/s turn" if deg_per_sec else ", turn unknown"),
            "success"
        )
        sse_only('cal_done_manual', "info")

    finally:
        _cal_running  = False
        _manual_drive = False

#  INFERENCE THREAD
def _infer_thread_fn(frame_copy):
    global _latest_preds, _infer_running
    try:
        preds = []
```

```
        #  Mouse model | detect the robot with strict filtering
        results = yolo_model.predict(
            frame_copy, imgsz=INFER_SIZE, conf=ROBOT_CONF,
            device=ROBOT_DEVICE, half=USE_HALF, verbose=False)[0]
        if results.boxes is not None:
            for box in results.boxes:
                x1,y1,x2,y2 = box.xyxy[0].cpu().numpy()
                conf = float(box.conf[0])
                cls  = int(box.cls[0])
                name = results.names[cls]
                w, h_ = x2-x1, y2-y1
                area  = w * h_
                # Shape filter: reject detections that are too big, too small,
                # or too elongated (socks, cables, hands all fail this)
                if area < ROBOT_MIN_AREA or area > ROBOT_MAX_AREA:
                    continue
                aspect = max(w, h_) / (min(w, h_) + 1e-6)
                if aspect > ROBOT_MAX_ASPECT:
                    continue
                preds.append({
                    "class": name, "confidence": conf,
                    "x": (x1+x2)/2, "y": (y1+y2)/2,
                    "width": w, "height": h_,
                    "x1":x1,"y1":y1,"x2":x2,"y2":y2
                })

        #  General model | threats and obstacles only
        t_results = threat_model.predict(
            frame_copy, imgsz=INFER_SIZE, conf=THREAT_CONF,
            device=THREAT_DEVICE, half=False, verbose=False)[0]
        if t_results.boxes is not None:
            for box in t_results.boxes:
                x1,y1,x2,y2 = box.xyxy[0].cpu().numpy()
                conf = float(box.conf[0])
                cls  = int(box.cls[0])
                name = t_results.names[cls]
                if name not in THREAT_OBJECTS and name not in OBSTACLE_OBJECTS:
                    continue
                w, h_ = x2-x1, y2-y1
                preds.append({
                    "class": name, "confidence": conf,
                    "x": (x1+x2)/2, "y": (y1+y2)/2,
                    "width": w, "height": h_,
                    "x1":x1,"y1":y1,"x2":x2,"y2":y2
                })

        with _infer_lock:
            _latest_preds = preds
    except Exception as e:
        print(f"[infer] {e}")
    finally:
        _infer_running = False
```

```
def maybe_infer(frame):
    global _last_infer_time, _infer_running
    now = time.time()
    if _infer_running or (now-_last_infer_time < INFER_INTERVAL):
        return
    _last_infer_time = now; _infer_running = True
    threading.Thread(target=_infer_thread_fn, args=(frame.copy(),), daemon=True).start()

def get_latest_preds():
    with _infer_lock:
        return list(_latest_preds)

#  PATHFINDER LOOP (runs in background thread)
def pathfinder_loop():
    global _virt_pos, _virt_vel, _last_robot_pos, _last_robot_pos_time, _low_speed_timer
    global _auto_state, _auto_state_time
    global _pos_history, _robot_confirm_buf
    global _nav_target
    global _nav_thread, _nav_thread_done
    prev_time = time.time()

    cap = cv2.VideoCapture(0)
    cap.set(cv2.CAP_PROP_FRAME_WIDTH,  FRAME_W)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, FRAME_H)
    cap.set(cv2.CAP_PROP_FPS, 30)
    for _ in range(5):
        cap.read()
    server_log("Pathfinder camera ready", "success")

    _auto_state      = "idle"
    _auto_state_time = time.time()
    _nav_path        = []
    _nav_path_idx    = 0

    while True:
        ret, frame = cap.read()
        if not ret:
            time.sleep(0.01); continue

        with _frame_lock:
            global _latest_frame
            _latest_frame = frame.copy()

        now = time.time()
        dt  = now - prev_time
        prev_time = now

        maybe_infer(frame)
        predictions = get_latest_preds()

        #  Parse predictions
        raw_robot_pos     = None
        robot_detected    = False
        current_obstacles = []
```

```
    for pred in predictions:
        cls = pred["class"]
        cx, cy = pred["x"], pred["y"]
        x1,y1,x2,y2 = pred["x1"],pred["y1"],pred["x2"],pred["y2"]
        w, h_ = pred["width"], pred["height"]
        area  = w * h_

        if cls == ROBOT_CLASS:
            raw_robot_pos = np.array([cx, cy], dtype=np.float32)
            continue
        if area < MIN_OBJECT_SIZE:
            continue
        is_threat = cls in THREAT_OBJECTS
        current_obstacles.append({
            'box': [x1,y1,x2,y2], 'name': cls,
            'is_target': is_threat, 'confidence': pred["confidence"]
        })

    #  Robot position: require N consecutive detections
    _robot_confirm_buf.append(raw_robot_pos)
    confirmed = [p for p in _robot_confirm_buf if p is not None]
    if len(confirmed) >= ROBOT_CONFIRM_FRAMES:
        confirmed_pos = np.mean(confirmed, axis=0).astype(np.float32)
        robot_detected = True

        # Only append to history when position actually changed enough
        # (avoids polluting history while standing still)
        if not _pos_history or float(np.linalg.norm(confirmed_pos -
        ↪  _pos_history[-1])) >= NAV_MIN_MOVE_PX:
            _pos_history.append(confirmed_pos.copy())

        _virt_pos[:] = 0.7 * confirmed_pos + 0.3 * _virt_pos
        _last_robot_pos      = confirmed_pos.copy()
        _last_robot_pos_time = now

    smoothed = smooth_obstacles(current_obstacles)

    #  Threat detection
    threat_detected = any(
        obs['is_target'] and obs['name'] in TRIGGER_OBJECTS
        for obs in smoothed
    )

    #  AUTO STATE MACHINE
    planned_path    = None
    status_override = None

    if not _manual_drive:
        state_elapsed = now - _auto_state_time

        # Threat always escalates to flee
        if threat_detected and _auto_state not in ("flee",):
            _auto_state      = "flee"
```

```
            _auto_state_time = now
            server_log("Threat detected | fleeing!", "warn")
            _nav_path = []

        if _auto_state == "idle":
            if _mouse is not None:
                _mouse.drive_raw(0, 0)
            if _nav_target is not None:
                goal = np.array(_nav_target, dtype=np.float32)
                path = find_path(_virt_pos, goal, smoothed)
                waypoints = path[1:] if path and len(path) > 1 else
                ↪  [tuple(goal.astype(int))]
                _nav_path     = path or []
                _nav_path_idx = 1
                _nav_thread_done = False
                _nav_thread = threading.Thread(
                    target=_nav_worker, args=(waypoints, "idle"), daemon=True)
                _nav_thread.start()
                _auto_state      = "navigate"
                _auto_state_time = now
                server_log(f"Navigating to ({int(goal[0])},{int(goal[1])}) via
                ↪  {len(waypoints)} waypoints", "info")
                _nav_target = None
            else:
                status_override = "IDLE"

        elif _auto_state == "navigate":
            # Nav worker thread is running | just show the path and wait
            if _nav_thread_done or (_nav_thread and not _nav_thread.is_alive()):
                # Thread finished | state was already set by _nav_worker
                _nav_thread = None
                _nav_thread_done = False
            else:
                status_override = "NAV"
                planned_path = _nav_path

            # New click target: cancel current nav and start fresh
            if _nav_target is not None:
                if _nav_thread and _nav_thread.is_alive():
                    # Signal abort via _manual_drive trick | set briefly then clear
                    # Actually just let the thread finish its current segment
                    pass
                goal = np.array(_nav_target, dtype=np.float32)
                path = find_path(_virt_pos, goal, smoothed)
                waypoints = path[1:] if path and len(path) > 1 else
                ↪  [tuple(goal.astype(int))]
                _nav_path     = path or []
                _nav_path_idx = 1
                _nav_thread_done = False
                _nav_thread = threading.Thread(
                    target=_nav_worker, args=(waypoints, "idle"), daemon=True)
                _nav_thread.start()
                _nav_target = None
                server_log("New nav target | replanning", "info")
```

```
        elif _auto_state == "flee":
            safe_zone_pos = np.array([SAFE_ZONE_CENTER_X, SAFE_ZONE_CENTER_Y],
            ↪  dtype=np.float32)
            in_safe_zone  = np.linalg.norm(_virt_pos - safe_zone_pos) <
            ↪  SAFE_ZONE_RADIUS

            if not threat_detected or in_safe_zone:
                # Stop nav thread if running
                if _nav_thread and _nav_thread.is_alive():
                    pass  # thread will exit on next _manual_drive check or finish
                if _mouse is not None:
                    _mouse.drive_raw(0, 0)
                _auto_state = "idle"
                _auto_state_time = now
                _nav_path = []
                _pos_history.clear()
                if in_safe_zone and threat_detected:
                    server_log("Reached safe zone", "success")
                else:
                    server_log("Threat cleared | returning to idle", "info")
            elif not (_nav_thread and _nav_thread.is_alive()) and not
            ↪  _nav_thread_done:
                # Launch flee nav thread if not already running
                path = find_path(_virt_pos, safe_zone_pos, smoothed) or []
                waypoints = path[1:] if len(path) > 1 else
                ↪  [tuple(safe_zone_pos.astype(int))]
                _nav_path     = path
                _nav_path_idx = 1
                _nav_thread_done = False
                _nav_thread = threading.Thread(
                    target=_nav_worker, args=(waypoints, "idle"), daemon=True)
                _nav_thread.start()
                server_log(f"Fleeing via {len(waypoints)} waypoints", "warn")
                planned_path = _nav_path
                status_override = "FLEE"
            else:
                planned_path = _nav_path
                status_override = "FLEE"
    else:
        _auto_state      = "idle"
        _auto_state_time = now
        _nav_path        = []
        _nav_path_idx    = 0
        _pos_history.clear()
        _nav_thread_done = False

    #  DRAW OVERLAYS
    annotated = frame.copy()
    cv2.rectangle(annotated, (PLAY_X1,PLAY_Y1), (PLAY_X2,PLAY_Y2), (100,100,100), 2)

    if _manual_drive:
        x1s = PLAY_X1 + EDGE_SLOW_MARGIN
        y1s = PLAY_Y1 + EDGE_SLOW_MARGIN
```

```
            x2s = PLAY_X2 - EDGE_SLOW_MARGIN
            y2s = PLAY_Y2 - EDGE_SLOW_MARGIN
            cv2.rectangle(annotated, (x1s,y1s), (x2s,y2s), (60,120,60), 1)
            if _last_robot_pos is not None:
                rpx, rpy = _last_robot_pos[0], _last_robot_pos[1]
                if rpx < PLAY_X1 or rpx > PLAY_X2 or rpy < PLAY_Y1 or rpy > PLAY_Y2:
                    cv2.rectangle(annotated, (PLAY_X1,PLAY_Y1), (PLAY_X2,PLAY_Y2),
                    ↪  (0,0,255), 4)
                    cv2.putText(annotated, "OUT OF BOUNDS", (PLAY_X1+10, PLAY_Y1+30),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,0,255), 2)

        if ENABLE_SAFE_ZONE:
            in_safe_zone = np.linalg.norm(
                _virt_pos - np.array([SAFE_ZONE_CENTER_X, SAFE_ZONE_CENTER_Y],
                ↪  dtype=np.float32)
            ) < SAFE_ZONE_RADIUS
            szc = (0,255,0) if in_safe_zone else (100,255,100)
            cv2.circle(annotated, (SAFE_ZONE_CENTER_X, SAFE_ZONE_CENTER_Y),
            ↪  SAFE_ZONE_RADIUS, szc, 2)
            cv2.putText(annotated, "SAFE", (SAFE_ZONE_CENTER_X-20, SAFE_ZONE_CENTER_Y),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, szc, 2)

        if planned_path and len(planned_path) > 1:
            pts = [(int(p[0]), int(p[1])) for p in planned_path]
            for i in range(len(pts)-1):
                cv2.line(annotated, pts[i], pts[i+1], (255,0,255), 2)
            if 0 < _nav_path_idx < len(pts):
                cv2.circle(annotated, pts[_nav_path_idx], 8, (255,255,0), -1)

        for obs in smoothed:
            x1o,y1o,x2o,y2o = [int(v) for v in obs['box']]
            col = (0,0,255) if obs['is_target'] else ((80,80,80) if obs.get('is_ghost')
            ↪  else (0,200,255))
            cv2.rectangle(annotated, (x1o,y1o), (x2o,y2o), col, 2)
            cv2.putText(annotated, obs['name'], (x1o, y1o-5),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.4, col, 1)

        if _last_robot_pos is not None:
            rx, ry = int(_last_robot_pos[0]), int(_last_robot_pos[1])
            cv2.drawMarker(annotated, (rx,ry), (0,255,0), cv2.MARKER_CROSS, 20, 2)
            cv2.putText(annotated, "MOUSE", (rx+12, ry-8),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0,255,0), 1)
            if len(_pos_history) >= 2:
                move = _pos_history[-1] - _pos_history[0]
                if float(np.linalg.norm(move)) >= NAV_MIN_MOVE_PX:
                    hx = int(rx + 28 * move[0] / (np.linalg.norm(move)+1e-6))
                    hy = int(ry + 28 * move[1] / (np.linalg.norm(move)+1e-6))
                    cv2.arrowedLine(annotated, (rx,ry), (hx,hy), (0,255,128), 2,
                    ↪  tipLength=0.4)

        dot_col = (128,128,128) if _manual_drive else (0,255,0)
        cv2.circle(annotated, tuple(_virt_pos.astype(int)), 8, dot_col, -1)
        cv2.circle(annotated, tuple(_virt_pos.astype(int)), 8, (255,255,255), 2)
```

```
        status = "MANUAL" if _manual_drive else (status_override or _auto_state.upper())
        cv2.putText(annotated, f"Mode:{status}", (10,20),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0,255,0), 2)
        cv2.putText(annotated, f"Mouse:{'FOUND' if robot_detected else 'lost'}", (10,45),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,255,0) if robot_detected else
                    ↪  (0,80,255), 1)
        if len(_pos_history) >= 2:
            move = _pos_history[-1] - _pos_history[0]
            md = float(np.linalg.norm(move))
            if md >= NAV_MIN_MOVE_PX:
                hdg_deg = math.degrees(math.atan2(float(move[1]), float(move[0])))
                cv2.putText(annotated, f"Hdg:{hdg_deg:.0f}", (10,68),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.45, (180,180,0), 1)

        with _frame_lock:
            global _annotated_frame
            _annotated_frame = annotated

#  CAMERA TRACK (streams annotated frames via WebRTC)
class CameraVideoTrack(VideoStreamTrack):
    def __init__(self):
        super().__init__()

    async def recv(self):
        pts, time_base = await self.next_timestamp()
        with _frame_lock:
            frame = _annotated_frame if _annotated_frame is not None else _latest_frame
        if frame is None:
            frame = np.zeros((FRAME_H, FRAME_W, 3), dtype=np.uint8)
        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        video_frame = av.VideoFrame.from_ndarray(frame_rgb, format="rgb24")
        video_frame.pts = pts
        video_frame.time_base = time_base
        return video_frame

#  AUTH
def make_session_token():
    return secrets.token_urlsafe(32)

def is_valid_token(token):
    if not token or token not in _sessions: return False
    if time.time() > _sessions[token]:
        del _sessions[token]; return False
    return True

def get_token_from_request(request):
    token = request.cookies.get("mb_token")
    if not token: token = request.headers.get("X-Token")
    return token

def require_auth(handler):
    async def wrapper(request):
        token = get_token_from_request(request)
        if not is_valid_token(token):
```

```
            if request.method == "GET": raise web.HTTPFound("/login")
            return web.json_response({"error":"Unauthorized"}, status=401)
        return await handler(request)
    return wrapper

#  ROUTES
async def login_page(request):
    return web.FileResponse("public/login.html")

async def login_post(request):
    ip = request.remote; now = time.time()
    _login_attempts[ip] = [t for t in _login_attempts[ip] if now-t<60]
    if len(_login_attempts[ip]) >= MAX_LOGIN_ATTEMPTS:
        remaining = int(60-(now-_login_attempts[ip][0]))
        return web.json_response({"error":f"Too many attempts. Try again in
        ↪  {remaining}s"},status=429)
    _login_attempts[ip].append(now)
    try: data = await request.json()
    except Exception: return web.json_response({"error":"Invalid request"},status=400)
    if data.get("password") != PASSWORD:
        return web.json_response({"error":"Incorrect password"},status=401)
    token = make_session_token()
    _sessions[token] = time.time()+SESSION_DURATION
    response = web.json_response({"ok":True})
    response.set_cookie("mb_token",token,max_age=SESSION_DURATION,httponly=True,samesite
    ↪  ="Strict")
    return response

@require_auth
async def index(request):
    return web.FileResponse("public/index.html")

@require_auth
async def offer(request):
    token = get_token_from_request(request); ip = request.remote
    try: params = await request.json()
    except Exception:
        body = await request.text()
        print("Failed to parse JSON:", body); raise
    if "sdp" not in params or "type" not in params:
        return web.json_response({"error":"Invalid offer"},status=400)
    offer_desc = RTCSessionDescription(sdp=params["sdp"],type=params["type"])
    pc = RTCPeerConnection()
    _active_viewers[token] =
    ↪  {"ip":ip,"connected_at":datetime.datetime.now().strftime("%H:%M:%S"),"pc":pc}
    pc.addTrack(CameraVideoTrack())

    @pc.on("datachannel")
    async def on_datachannel(channel):
        @channel.on("message")
        async def on_message(message):
            global _manual_drive
            try:
                data = json.loads(message)
```

```
                # Ping/pong for control RTT measurement
                if "ping" in data:
                    channel.send(json.dumps({"pong": data["ping"]}))
                    return
                # Click-to-navigate: {"goto": {"x": 320, "y": 240}}
                if "goto" in data:
                    global _nav_target
                    gx = int(data["goto"].get("x", FRAME_W//2))
                    gy = int(data["goto"].get("y", FRAME_H//2))
                    # Clamp to play area
                    gx = max(PLAY_X1+10, min(PLAY_X2-10, gx))
                    gy = max(PLAY_Y1+10, min(PLAY_Y2-10, gy))
                    _nav_target = (gx, gy)
                    server_log(f"Nav target set: ({gx},{gy})", "info")
                    return
                # Mode switch: {"mode": "manual"} or {"mode": "auto"}
                if "mode" in data:
                    _manual_drive = (data["mode"] == "manual")
                    server_log(f"Drive mode switched to {'MANUAL' if _manual_drive else
                    ↪  'AUTO'}", "info")
                    return
                # Manual joystick: {"x": 0.5, "y": 0.3}
                x = data.get("x",0); y = data.get("y",0)
                if _mouse is not None and _manual_drive:
                    if x == 0 and y == 0:
                        _mouse.stop()          # bypass slew limiter for instant stop
                        _mouse.fallback.started = False
                    else:
                        manual_drive_with_edge_limit(x, y, _last_robot_pos)
                        _mouse.fallback.start()
                elif not _manual_drive:
                    pass
                else:
                    server_log("Control ignored - mouse not connected","warn")
            except json.JSONDecodeError: print("Invalid JSON:", message)
            except Exception as e: server_log(f"Control error: {e}","error")

    @pc.on("connectionstatechange")
    async def on_state_change():
        s = pc.connectionState
        lvl = "success" if s=="connected" else "warn" if s=="disconnected" else "error"
        ↪  if s=="failed" else "info"
        server_log(f"WebRTC {s} | {ip}", lvl)
        if s in ("failed","closed","disconnected"):
            _active_viewers.pop(token,None); await pc.close()

    await pc.setRemoteDescription(offer_desc)
    answer = await pc.createAnswer()
    await pc.setLocalDescription(answer)
    return web.json_response({"sdp":pc.localDescription.sdp,"type":pc.localDescription.t
    ↪  ype})

@require_auth
async def viewers_status(request):
```

```
    now_token = get_token_from_request(request)
    viewers = []
    for tok,info in _active_viewers.items():
        if info["pc"].connectionState == "connected":
            viewers.append({"ip":info["ip"],"connected_at":info["connected_at"],"is_you"
            ↪  :tok==now_token})
    return web.json_response({"viewers":viewers})

@require_auth
async def log_stream(request):
    q = asyncio.Queue(maxsize=100)
    _log_subscribers.append(q)
    resp = web.StreamResponse(headers={
        "Content-Type":"text/event-stream","Cache-Control":"no-cache","X-Accel-Buffering
        ↪  ":"no"})
    await resp.prepare(request)
    try:
        while True:
            try:
                payload = await asyncio.wait_for(q.get(),timeout=25)
                await resp.write(f"data: {payload}\n\n".encode())
            except asyncio.TimeoutError:
                await resp.write(b": keepalive\n\n")
    except Exception: pass
    finally:
        if q in _log_subscribers: _log_subscribers.remove(q)
    return resp

async def logout(request):
    token = get_token_from_request(request)
    _sessions.pop(token,None); _active_viewers.pop(token,None)
    response = web.HTTPFound("/login"); response.del_cookie("mb_token")
    return response

#  CERTS
def generate_certs():
    if os.path.exists("certs/rootCA.crt") and os.path.exists("certs/rootCA.key"):
        print("Certificates already exist"); return
    print("Generating self-signed certificates...")
    os.makedirs("certs",exist_ok=True)
    key = rsa.generate_private_key(public_exponent=65537,key_size=2048)
    subject = issuer = x509.Name([x509.NameAttribute(NameOID.COMMON_NAME,u"localhost")])
    cert = (x509.CertificateBuilder().subject_name(subject).issuer_name(issuer)
            .public_key(key.public_key()).serial_number(x509.random_serial_number())
            .not_valid_before(datetime.datetime.utcnow())
            .not_valid_after(datetime.datetime.utcnow()+datetime.timedelta(days=365))
            .add_extension(x509.SubjectAlternativeName([x509.DNSName(u"localhost"),
                x509.DNSName(u"127.0.0.1")]),critical=False).sign(key,hashes.SHA256()))
    with open("certs/rootCA.key","wb") as f:
        f.write(key.private_bytes(serialization.Encoding.PEM,
            serialization.PrivateFormat.TraditionalOpenSSL,serialization.NoEncryption()))
    with open("certs/rootCA.crt","wb") as f:
        f.write(cert.public_bytes(serialization.Encoding.PEM))
    print("Certificates generated!")
```

```
#  WATCHDOG
async def watchdog_task():
    while True:
        await asyncio.sleep(0.1)
        if _mouse and _mouse.fallback.should_turn_off_mouse():
            _mouse.write(b'\x00')
            _mouse.fallback.started = False

#  MAIN
@require_auth
async def ble_reconnect(request):
    """Reconnect BLE without restarting the server."""
    if _mouse is None:
        return web.json_response({"error": "Mouse not initialised"}, status=500)
    if _mouse.client and _mouse.client.is_connected:
        return web.json_response({"ok": True, "msg": "Already connected"})
    server_log("BLE reconnect requested via UI", "info")
    threading.Thread(target=_mouse.connect, daemon=True).start()
    return web.json_response({"ok": True, "msg": "Reconnecting..."})

@require_auth
async def calibrate(request):
    global _cal_running
    if _cal_running:
        return web.json_response({"error": "Calibration already running"}, status=409)
    if _last_robot_pos is None:
        return web.json_response({"error": "Robot not detected  make sure mouse is
        ↪  visible"}, status=400)
    _cal_running = True
    threading.Thread(target=run_calibration, daemon=True).start()
    return web.json_response({"ok": True, "msg": "Calibration started"})

@require_auth
async def cal_status(request):
    return web.json_response({
        "running":       _cal_running,
        "data":          _cal_data,
        "fwd_level":     AUTO_FWD_LEVEL,
        "px_per_s":      AUTO_PX_PER_SEC,
        "deg_per_s":     AUTO_DEG_PER_SEC,
    })

def main():
    global _mouse
    generate_certs()

    server_log("Initializing BLE mouse controller...", "info")
    _mouse = Mouse()
    _mouse.connect()

    # Start pathfinder in background thread
    pf_thread = threading.Thread(target=pathfinder_loop, daemon=True)
    pf_thread.start()
```

```
    server_log("Pathfinder started", "success")

    app = web.Application()
    app.router.add_get("/login",      login_page)
    app.router.add_post("/login",     login_post)
    app.router.add_get("/logout",     logout)
    app.router.add_get("/",           index)
    app.router.add_post("/offer",     offer)
    app.router.add_get("/viewers",    viewers_status)
    app.router.add_get("/log-stream", log_stream)
    app.router.add_post("/calibrate", calibrate)
    app.router.add_get("/cal-status", cal_status)
    app.router.add_post("/ble-reconnect", ble_reconnect)
    app.router.add_static("/static/", path="public", name="static")

    class _SilentPaths(logging.Filter):
        MUTE = ("/viewers","/log-stream","/cal-status")
        def filter(self, record):
            return not any(p in record.getMessage() for p in self.MUTE)
    logging.getLogger("aiohttp.access").addFilter(_SilentPaths())

    ssl_ctx = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
    ssl_ctx.load_cert_chain("certs/rootCA.crt","certs/rootCA.key")

    async def start_watchdog(app):
        app['watchdog'] = asyncio.create_task(watchdog_task())
    async def cleanup_watchdog(app):
        app['watchdog'].cancel()
        try: await app['watchdog']
        except asyncio.CancelledError: pass

    app.on_startup.append(start_watchdog)
    app.on_cleanup.append(cleanup_watchdog)

    try:
        web.run_app(app, host="0.0.0.0", port=8000, ssl_context=ssl_ctx)
    finally:
        server_log("Shutting down...", "info")
        if _mouse:
            _mouse.drive_raw(0, 0)
            _mouse.disconnect()

if __name__ == "__main__":
    main()
```

R BLE Motor Controller Interface

```
import asyncio
import threading
import time
import math
from collections import deque
```

```
from bleak import BleakClient, BleakScanner

# Import server_log so BLE events show up in the browser log.
# This works because Mouse.py runs in the same process as main.py.
try:
    from main import server_log, sse_only
except ImportError:
    def server_log(msg, level="info"):
        print(f"[{level.upper()}] {msg}")
    def sse_only(msg, level="info"):
        pass  # no SSE available standalone

USE_ACK = False
OFF_STATE = b'\x00'

BLE_RTT_WINDOW = 20   # samples kept for rolling average

class SlewLimiter2D:
    def __init__(self, max_rate):
        self.max_rate = max_rate
        self.x = 0.0
        self.y = 0.0
        self.last_time = time.time()

    def update(self, target_x, target_y):
        now = time.time()
        dt = now - self.last_time
        self.last_time = now

        dx = target_x - self.x
        dy = target_y - self.y
        dist = math.sqrt(dx*dx + dy*dy)
        max_step = self.max_rate * dt

        if dist > max_step and dist > 0:
            scale = max_step / dist
            dx *= scale
            dy *= scale

        self.x += dx
        self.y += dy
        return self.x, self.y


class MotionFallback:
    def __init__(self, frequency, threshold):
        self.last_time = time.time()
        self.f = frequency
        self.t = threshold
        self.started = False

    def start(self):
        self.started = True
        self.last_time = time.time()
```

```
    def should_turn_off_mouse(self):
        if self.started:
            if time.time() > self.last_time + self.t:
                server_log("Control data halted | falling back to off state", "warn")
                return True
            return False


class Mouse:
    def __init__(self):
        self.max_roc = 2.0   # units/sec | high enough that steering feels instant
        self.device_name = "ToyESP32"
        self.watchdog_frequency = 5
        self.uart_uuid = "6e400002-b5a3-f393-e0a9-e50e24dcca9e"
        self.fallback_timeout = 2

        self.limiter = SlewLimiter2D(self.max_roc)
        self.fallback = MotionFallback(self.watchdog_frequency, self.fallback_timeout)

        self.loop = asyncio.new_event_loop()
        self.thread = threading.Thread(target=self._run_loop, daemon=True)
        self.thread.start()

        self.client = None
        self.address = None

        # RTT rolling average
        self._rtt_samples = deque(maxlen=BLE_RTT_WINDOW)
        self.is_ble_connected = False
        self._connecting = False   # True while connection handshake is in progress

    def _run_loop(self):
        asyncio.set_event_loop(self.loop)
        self.loop.run_forever()

    # ----------- ASYNC -----------

    async def _find_device(self):
        server_log(f"Scanning for '{self.device_name}'...", "info")
        devices = await BleakScanner.discover()
        for d in devices:
            if d.name and self.device_name in d.name:
                return d.address
        return None

    async def _connect_async(self, retry=True):
        if self.client and self.client.is_connected:
            server_log("BLE already connected", "info")
            return True

        self._connecting = True
        addr = await self._find_device()
        if addr is None:
```

```
            server_log(f"'{self.device_name}' not found | BLE disconnected", "error")
            self.is_ble_connected = False
            self._connecting = False
            await self._send_disconnect_packet()
            if retry:
                await self._connect_async(False)
            return False

        self.address = addr
        server_log(f"Found {self.device_name} at {addr} | connecting...", "info")
        self.client = BleakClient(self.address)
        await self.client.connect()
        # Service discovery happens inside connect() | safe to write after this
        await self.client.write_gatt_char(self.uart_uuid, OFF_STATE)
        self.is_ble_connected = True
        self._connecting = False
        server_log(f"BLE connected to {self.device_name}", "success")
        return True

    async def _send_disconnect_packet(self):
        self._connecting = False
        try:
            if self.client and self.client.is_connected:
                await self.client.write_gatt_char(self.uart_uuid, OFF_STATE)
                await self.client.disconnect()
                self.is_ble_connected = False
                server_log("BLE disconnected cleanly", "info")
        except Exception as e:
            server_log(f"BLE disconnect error: {e}", "error")

    async def _write_async(self, data: bytes):
        t0 = time.time()
        await self.client.write_gatt_char(self.uart_uuid, data, response=USE_ACK)
        rtt_ms = round((time.time() - t0) * 1000)
        self._rtt_samples.append(rtt_ms)
        avg_ms = round(sum(self._rtt_samples) / len(self._rtt_samples))
        # Import lazily so we always get the live sse_only after main is fully loaded
        try:
            import main as _main
            _main.sse_only(f"BLE RTT: {rtt_ms} ms / {avg_ms} ms avg", "info")
        except Exception:
            pass

    async def _disconnect_async(self):
        if self.client:
            await self.client.disconnect()

    # ----------- BLOCKING -----------

    def connect(self):
        future = asyncio.run_coroutine_threadsafe(
            self._connect_async(), self.loop
        )
        return future.result()
```

```
def write(self, data: bytes):
    # Block writes while connection handshake is in progress (service discovery)
    if self._connecting:
        return
    if self.client and self.client.is_connected:
        future = asyncio.run_coroutine_threadsafe(
            self._write_async(data), self.loop
        )
        try:
            future.result()
        except Exception as e:
            # Catch ALL exceptions including BleakError from service discovery race
            self.is_ble_connected = False
            self._connecting = False
            err_name = type(e).__name__
            server_log(f"BLE write error ({err_name}) | reconnecting...", "warn")
            threading.Thread(target=self.connect, daemon=True).start()
    else:
        if self.is_ble_connected:
            self.is_ble_connected = False
            server_log(f"BLE connection to '{self.device_name}' lost", "error")

def disconnect(self):
    future = asyncio.run_coroutine_threadsafe(
        self._disconnect_async(), self.loop
    )
    return future.result()

def stop(self):
    """Immediately stop both motors | bypasses slew limiter."""
    self.limiter.x = 0.0
    self.limiter.y = 0.0
    self.drive_raw(0, 0)

def drive_raw(self, left_level, right_level):
    """
    Send exact motor levels directly | no joystick math, no slew limiter.
    Used by auto navigation so speed is precisely controlled.

    left_level, right_level: integers in range:
      0       = off
      1-7     = FWD slowest→fastest
      -1 to -7 = REV slowest→fastest
    """
    def level_to_nibble(lvl):
        lvl = int(max(-7, min(7, lvl)))
        if lvl == 0:
            return 0x0
        elif lvl > 0:
            return lvl           # 1-7 FWD
        else:
            return 0x8 + abs(lvl)  # 9-F REV
```

```
        l_nib = level_to_nibble(left_level)
        r_nib = level_to_nibble(right_level)
        cmd = (l_nib << 4) | r_nib
        self.write(bytes([cmd]))

    def drive(self, joy_x, joy_y):
        """
        Manual joystick drive. Encodes [-1,+1] inputs to motor bytes.
        min_moving=1 so the full joystick range is available to the user |
        they can feel the difference between speeds naturally.
        """
        # slew limiter
        x, y = self.limiter.update(joy_x, joy_y)

        # Hardware turns CCW for positive x | invert so right stick = right turn
        x = -x

        # Straight-line deadzone
        STEER_DZ = 0.15
        if abs(x) < STEER_DZ:
            x = 0.0
        else:
            x = math.copysign((abs(x) - STEER_DZ) / (1.0 - STEER_DZ), x)

        # Forward bias when turning from near-stop
        if abs(x) > 0.1 and abs(y) < 0.2:
            y = y + abs(x) * 0.35

        # Arcade mix → individual wheel speeds
        l = y + x
        r = y - x
        m = max(abs(l), abs(r), 1.0)
        l /= m
        r /= m

        def to_nibble(v):
            """
            Map [-1,+1] to motor nibble. No minimum clamp | user gets full range.
            Values that round to 1 or 2 are left as-is; if they stall, the user
            can push the stick further.
            """
            if abs(v) < 0.04:
                return 0x0
            if v > 0:
                spd = max(1, min(7, int(v * 7 + 0.5)))
                return spd
            else:
                spd = max(1, min(7, int(abs(v) * 7 + 0.5)))
                return 0x8 + spd

        r_nib = to_nibble(r)
        l_nib = to_nibble(l)
        cmd = (l_nib << 4) | r_nib
        self.write(bytes([cmd]))
```

```
# Standalone demo | LED toggle test
if __name__ == "__main__":
    ble = Mouse()
    f = 15
    ble.connect()
    for i in range(300):
        time.sleep(1/f)
        ble.write(b'1' if i % 2 == 0 else b'0')
    ble.disconnect()
```

S Login Interface

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>MouseBot | Login</title>
  <link href="https://fonts.googleapis.com/css2?family=Playfair+Display:wght@700;900&fam
  ↪  ily=DM+Sans:wght@400;500;600&display=swap" rel="stylesheet" />
  <style>
    *, *::before, *::after { box-sizing: border-box; margin: 0; padding: 0; }
    :root {
      --gold: #f5c842; --dark: #0f0f0f; --card: #1a1a1a;
      --border: #2e2e2e; --text: #f0ede6; --muted: #888;
      --red: #f87171; --radius: 16px;
    }
    body {
      background: var(--dark); color: var(--text);
      font-family: 'DM Sans', sans-serif;
      min-height: 100vh; display: flex;
      align-items: center; justify-content: center;
    }
    .card {
      background: var(--card); border: 1px solid var(--border);
      border-radius: var(--radius); padding: 40px 36px;
      width: 100%; max-width: 380px;
      box-shadow: 0 24px 64px rgba(0,0,0,0.5);
    }
    .logo {
      font-family: 'Playfair Display', serif;
      font-size: 1.7rem; font-weight: 900;
      color: var(--gold); text-align: center;
      margin-bottom: 6px;
    }
    .logo span { color: var(--text); }
    .subtitle {
      text-align: center; color: var(--muted);
      font-size: 0.85rem; margin-bottom: 32px;
    }
```

```
    label {
      display: block; font-size: 0.78rem; font-weight: 600;
      text-transform: uppercase; letter-spacing: 1px;
      color: var(--muted); margin-bottom: 8px;
    }
    input[type=password] {
      width: 100%; background: #111; border: 1px solid var(--border);
      border-radius: 10px; padding: 12px 16px;
      color: var(--text); font-family: 'DM Sans', sans-serif;
      font-size: 1rem; outline: none;
      transition: border-color 0.2s;
    }
    input[type=password]:focus { border-color: var(--gold); }
    .btn {
      width: 100%; margin-top: 20px;
      background: var(--gold); color: #0f0f0f;
      border: none; border-radius: 10px;
      padding: 13px; font-family: 'DM Sans', sans-serif;
      font-size: 0.95rem; font-weight: 700;
      cursor: pointer; transition: all 0.2s;
    }
    .btn:hover { background: #fdd835; transform: translateY(-1px); }
    .btn:disabled { opacity: 0.5; cursor: not-allowed; transform: none; }
    .error {
      margin-top: 14px; padding: 10px 14px;
      background: rgba(248,113,113,0.1);
      border: 1px solid rgba(248,113,113,0.25);
      border-radius: 8px; color: var(--red);
      font-size: 0.85rem; display: none;
    }
  </style>
</head>
<body>
  <div class="card">
    <div class="logo">Mouse<span>Bot</span></div>
    <div class="subtitle">Enter password to access control panel</div>
    <label for="pw">Password</label>
    <input type="password" id="pw" placeholder="••••••••" autofocus />
    <button class="btn" id="btn" onclick="login()">Unlock</button>
    <div class="error" id="err"></div>
  </div>
  <script>
    document.getElementById('pw').addEventListener('keydown', e => {
      if (e.key === 'Enter') login();
    });

    async function login() {
      const pw = document.getElementById('pw').value;
      const btn = document.getElementById('btn');
      const err = document.getElementById('err');
      if (!pw) return;

      btn.disabled = true; btn.textContent = 'Checking...';
      err.style.display = 'none';
```

```
    try {
      const res = await fetch('/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ password: pw })
      });
      const data = await res.json();
      if (res.ok) {
        window.location.href = '/';
      } else {
        err.textContent = data.error || 'Login failed';
        err.style.display = 'block';
        btn.disabled = false; btn.textContent = 'Unlock';
        document.getElementById('pw').value = '';
        document.getElementById('pw').focus();
      }
    } catch(e) {
      err.textContent = 'Could not reach server';
      err.style.display = 'block';
      btn.disabled = false; btn.textContent = 'Unlock';
    }
  }
  </script>
</body>
</html>
```

T Main Control Interface

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Mouse Robot | Control Panel</title>
  <link href="https://fonts.googleapis.com/css2?family=Playfair+Display:wght@700;900&fam
  ↪  ily=DM+Sans:wght@400;500;600&display=swap" rel="stylesheet" />
  <style>
    *, *::before, *::after { box-sizing: border-box; margin: 0; padding: 0; }

    :root {
      --gold:   #f5c842;
      --dark:   #0f0f0f;
      --mid:    #1c1c1c;
      --card:   #242424;
      --border: #2e2e2e;
      --text:   #f0ede6;
      --muted:  #888;
      --radius: 16px;
      --green:  #4ade80;
      --red:    #f87171;
    }

    body {
      background: var(--dark);
      color: var(--text);
      font-family: 'DM Sans', sans-serif;
      min-height: 100vh;
      overflow-x: hidden;
    }

    /* NAV */
    nav {
      display: flex;
      justify-content: space-between;
      align-items: center;
      padding: 20px 40px;
      background: rgba(15,15,15,0.85);
      backdrop-filter: blur(12px);
      border-bottom: 1px solid var(--border);
      position: sticky;
      top: 0;
      z-index: 100;
    }
    .logo {
      font-family: 'Playfair Display', serif;
      font-size: 1.4rem;
      font-weight: 900;
      color: var(--gold);
      letter-spacing: -0.5px;
    }
```

```
.logo span { color: var(--text); }

.status-pill {
  display: flex;
  align-items: center;
  gap: 8px;
  background: var(--card);
  border: 1px solid var(--border);
  border-radius: 99px;
  padding: 6px 16px;
  font-size: 0.82rem;
  font-weight: 600;
  color: var(--muted);
  transition: all 0.3s;
}
.status-pill.connected  { border-color: rgba(74,222,128,0.3);  color: var(--green); }
.status-pill.connecting { border-color: rgba(245,200,66,0.3);  color: var(--gold);  }
.status-pill.failed     { border-color: rgba(248,113,113,0.3); color: var(--red);   }
.status-dot {
  width: 7px; height: 7px;
  border-radius: 50%;
  background: var(--muted);
  transition: background 0.3s;
}
.status-pill.connected  .status-dot { background: var(--green); box-shadow: 0 0 6px
↪  var(--green); }
.status-pill.connecting .status-dot { background: var(--gold);  animation: pulse 1s
↪  infinite; }
.status-pill.failed     .status-dot { background: var(--red); }
@keyframes pulse { 0%,100% { opacity:1; } 50% { opacity:0.3; } }

.connect-btn {
  background: var(--gold);
  color: var(--dark);
  border: none;
  border-radius: 8px;
  padding: 9px 22px;
  font-family: 'DM Sans', sans-serif;
  font-weight: 600;
  font-size: 0.9rem;
  cursor: pointer;
  transition: all 0.2s;
}
.connect-btn:hover { background: #ffd700; transform: translateY(-1px); box-shadow: 0
↪  4px 20px rgba(245,200,66,0.35); }
.connect-btn:disabled { opacity: 0.5; cursor: not-allowed; transform: none;
↪  box-shadow: none; }

/* LAYOUT */
main {
  max-width: 1200px;
  margin: 0 auto;
  padding: 32px 40px;
  display: grid;
```

```
    grid-template-columns: 1fr 380px;
    gap: 24px;
    align-items: start;
  }

  /* VIDEO CARD */
  .video-card {
    background: var(--card);
    border: 1px solid var(--border);
    border-radius: var(--radius);
    overflow: hidden;
    animation: fadeUp 0.4s ease both;
  }
  .card-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 16px 20px;
    border-bottom: 1px solid var(--border);
  }
  .card-title {
    font-size: 0.78rem;
    font-weight: 600;
    letter-spacing: 1.2px;
    text-transform: uppercase;
    color: var(--muted);
  }
  .badge {
    background: rgba(245,200,66,0.1);
    border: 1px solid rgba(245,200,66,0.2);
    color: var(--gold);
    font-size: 0.73rem;
    font-weight: 600;
    padding: 3px 10px;
    border-radius: 99px;
  }
  .video-wrapper {
    position: relative;
    aspect-ratio: 16/9;
    background: #080808;
  }
  #remoteVideo {
    width: 100%; height: 100%;
    object-fit: cover;
    display: block;
  }
  .video-overlay {
    position: absolute; inset: 0;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    gap: 12px;
    background: rgba(8,8,8,0.75);
```

```
  transition: opacity 0.3s;
}
.video-overlay.hidden { opacity: 0; pointer-events: none; }
.overlay-icon {
  width: 52px; height: 52px;
  border: 1.5px solid var(--border);
  border-radius: 50%;
  display: flex; align-items: center; justify-content: center;
  font-size: 1.4rem;
}
.overlay-text { color: var(--muted); font-size: 0.88rem; }

/* SIDE PANEL */
.side-panel {
  display: flex;
  flex-direction: column;
  gap: 20px;
  animation: fadeUp 0.4s ease 0.1s both;
}
.panel-card {
  background: var(--card);
  border: 1px solid var(--border);
  border-radius: var(--radius);
  overflow: hidden;
}

/* JOYSTICK */
.joystick-wrap {
  padding: 28px 20px;
  display: flex;
  flex-direction: column;
  align-items: center;
  gap: 18px;
}
.joystick-zone {
  position: relative;
  width: 200px; height: 200px;
  border-radius: 50%;
  background: radial-gradient(circle at 50% 50%, #1c1c1c 0%, #0e0e0e 100%);
  border: 2px solid var(--border);
  cursor: grab;
  touch-action: none;
  user-select: none;
}
.joystick-zone::before {
  content: '';
  position: absolute;
  inset: 18px;
  border-radius: 50%;
  border: 1px solid #282828;
}
.joystick-zone::after {
  content: '';
  position: absolute;
```

```
    top: 50%; left: 10%; right: 10%;
    height: 1px;
    background: #252525;
  }
  .crosshair-v {
    position: absolute;
    left: 50%; top: 10%; bottom: 10%;
    width: 1px;
    background: #252525;
    transform: translateX(-50%);
  }
  .joystick-knob {
    position: absolute;
    width: 62px; height: 62px;
    border-radius: 50%;
    background: radial-gradient(circle at 38% 32%, #383838, #181818);
    border: 2px solid rgba(245,200,66,0.3);
    box-shadow: 0 4px 18px rgba(0,0,0,0.7), inset 0 1px 0 rgba(255,255,255,0.04);
    top: 50%; left: 50%;
    transform: translate(-50%, -50%);
    pointer-events: none;
    z-index: 2;
    transition: border-color 0.2s;
  }
  .joystick-zone.active .joystick-knob {
    border-color: var(--gold);
    box-shadow: 0 4px 20px rgba(245,200,66,0.18), inset 0 1px 0 rgba(255,255,255,0.04);
  }
  .joystick-zone.active { cursor: grabbing; }
  .coord-row {
    display: flex; gap: 10px;
  }
  .coord-chip {
    background: var(--mid);
    border: 1px solid var(--border);
    border-radius: 8px;
    padding: 6px 14px;
    font-size: 0.8rem;
    font-weight: 600;
    color: var(--muted);
    min-width: 84px;
    text-align: center;
    font-variant-numeric: tabular-nums;
  }
  .coord-chip b { color: var(--text); font-weight: 600; }

  /* STATS */
  .stats-grid {
    display: grid;
    grid-template-columns: 1fr 1fr;
    gap: 1px;
    background: var(--border);
  }
  .stat-cell {
```

```
  background: var(--card);
  padding: 15px 18px;
}
.stat-label {
  font-size: 0.72rem;
  font-weight: 600;
  letter-spacing: 1px;
  text-transform: uppercase;
  color: var(--muted);
  margin-bottom: 5px;
}
.stat-val {
  font-size: 1rem;
  font-weight: 600;
  color: var(--text);
  font-variant-numeric: tabular-nums;
}
.stat-val.good { color: var(--green); }
.stat-val.warn { color: var(--gold); }
.stat-val.bad  { color: var(--red); }

/* LOG */
.log-box {
  padding: 12px 18px;
  max-height: 130px;
  overflow-y: auto;
  display: flex;
  flex-direction: column;
  gap: 3px;
}
.log-box::-webkit-scrollbar { width: 3px; }
.log-box::-webkit-scrollbar-thumb { background: var(--border); border-radius: 2px; }
.log-row {
  display: flex; gap: 10px;
  font-size: 0.76rem;
}
.log-t { color: #484848; flex-shrink: 0; font-variant-numeric: tabular-nums; }
.log-row.info    .log-m { color: var(--muted); }
.log-row.success .log-m { color: var(--green); }
.log-row.warn    .log-m { color: var(--gold); }
.log-row.error   .log-m { color: var(--red); }

@keyframes fadeUp {
  from { opacity:0; transform:translateY(14px); }
  to   { opacity:1; transform:translateY(0); }
}

/* MODE TOGGLE */
.mode-toggle-wrap {
  display: flex;
  align-items: center;
  gap: 12px;
  padding: 14px 20px;
  border-top: 1px solid var(--border);
```

```
}
.mode-label {
  font-size: 0.78rem;
  font-weight: 600;
  letter-spacing: 1px;
  text-transform: uppercase;
  color: var(--muted);
  flex: 1;
}
.mode-badge {
  font-size: 0.75rem;
  font-weight: 700;
  padding: 3px 10px;
  border-radius: 99px;
  letter-spacing: 0.5px;
  transition: all 0.2s;
}
.mode-badge.manual { background: rgba(245,200,66,0.15); color: var(--gold); border:
↪  1px solid rgba(245,200,66,0.3); }
.mode-badge.auto   { background: rgba(74,222,128,0.12); color: var(--green); border:
↪  1px solid rgba(74,222,128,0.3); }
/* pill switch */
.switch {
  position: relative;
  width: 52px; height: 28px;
  flex-shrink: 0;
}
.switch input { opacity: 0; width: 0; height: 0; }
.slider {
  position: absolute; inset: 0;
  background: #2a2a2a;
  border: 1px solid var(--border);
  border-radius: 99px;
  cursor: pointer;
  transition: background 0.25s, border-color 0.25s;
}
.slider::before {
  content: '';
  position: absolute;
  width: 20px; height: 20px;
  left: 3px; top: 3px;
  background: var(--muted);
  border-radius: 50%;
  transition: transform 0.25s, background 0.25s;
}
.switch input:checked + .slider { background: rgba(74,222,128,0.2); border-color:
↪  rgba(74,222,128,0.4); }
.switch input:checked + .slider::before { transform: translateX(24px); background:
↪  var(--green); }
.switch input:not(:checked) + .slider { background: rgba(245,200,66,0.15);
↪  border-color: rgba(245,200,66,0.3); }
.switch input:not(:checked) + .slider::before { background: var(--gold); }

@media (max-width: 860px) {
```

```
      main { grid-template-columns: 1fr; padding: 16px; }
      nav  { padding: 14px 18px; }
    }
  </style>
</head>
<body>

<nav>
  <div class="logo">Mouse<span>Bot</span></div>
  <div class="status-pill" id="statusPill">
    <div class="status-dot"></div>
    <span id="statusText">Disconnected</span>
  </div>
  <div style="display:flex; gap:10px; align-items:center;">
    <button class="connect-btn" id="bleBtn" onclick="bleReconnect()"
    ↪  style="background:var(--card); border:1px solid var(--border);
    ↪  color:var(--muted);" title="Reconnect BLE without restarting">BLE</button>
    <button class="connect-btn" id="connectBtn">Connect</button>
    <a href="/logout" style="font-size:0.8rem; color:var(--muted); text-decoration:none;
    ↪  padding: 8px 14px; border:1px solid var(--border); border-radius:8px;
    ↪  transition:color 0.2s;" onmouseover="this.style.color='var(--text)'"
    ↪  onmouseout="this.style.color='var(--muted)'">Logout</a>
  </div>
</nav>

<!-- VIEWER WARNING BANNER -->
<div id="viewerBanner" style="display:none; background:rgba(248,113,113,0.08);
↪  border-bottom:1px solid rgba(248,113,113,0.2); padding:10px 40px; font-size:0.85rem;
↪  color:#f87171; display:none; align-items:center; gap:10px;">
  <span></span>
  <span id="viewerBannerText"></span>
</div>

<main>

  <!-- VIDEO -->
  <div class="video-card">
    <div class="card-header">
      <span class="card-title">Live Feed</span>
      <span class="badge" id="videoHint">Connect to stream</span>
    </div>
    <div class="video-wrapper" id="videoWrapper">
      <video id="remoteVideo" autoplay playsinline muted disablepictureinpicture></video>
      <div class="video-overlay" id="videoOverlay">
        <div class="overlay-icon"></div>
        <div class="overlay-text">Waiting for stream...</div>
      </div>
      <!-- Click-to-nav target indicator -->
      <div id="navDot" style="display:none; position:absolute; width:18px; height:18px;
           border-radius:50%; border:2px solid #f5c842; background:rgba(245,200,66,0.25);
           transform:translate(-50%,-50%); pointer-events:none; transition:opacity
           ↪  0.5s;"></div>
    </div>
  </div>
```

```
<!-- SIDE -->
<div class="side-panel">

  <!-- JOYSTICK -->
  <div class="panel-card">
    <div class="card-header">
      <span class="card-title">Movement Control</span>
    </div>
    <div class="joystick-wrap">
      <div class="joystick-zone" id="joystickZone">
        <div class="crosshair-v"></div>
        <div class="joystick-knob" id="joystickKnob"></div>
      </div>
      <div class="coord-row">
        <div class="coord-chip">X: <b id="cX">0.00</b></div>
        <div class="coord-chip">Y: <b id="cY">0.00</b></div>
      </div>
    </div>
    <!-- MODE TOGGLE -->
    <div class="mode-toggle-wrap">
      <span class="mode-label">Drive Mode</span>
      <span class="mode-badge manual" id="modeBadge">MANUAL</span>
      <label class="switch" title="Toggle Manual / Auto">
        <input type="checkbox" id="modeToggle" />
        <span class="slider"></span>
      </label>
    </div>
  </div>

  <!-- STATS -->
  <div class="panel-card">
    <div class="card-header">
      <span class="card-title">Connection Stats</span>
    </div>
    <div class="stats-grid">
      <div class="stat-cell"><div class="stat-label">State</div><div class="stat-val"
      ↪  id="sState"></div></div>
      <div class="stat-cell"><div class="stat-label">Data Ch.</div><div
      ↪  class="stat-val" id="sData"></div></div>
      <div class="stat-cell"><div class="stat-label">Video RTT</div><div
      ↪  class="stat-val" id="sRtt"></div></div>
      <div class="stat-cell"><div class="stat-label">Video Avg</div><div
      ↪  class="stat-val" id="sRttAvg"></div></div>
      <div class="stat-cell"><div class="stat-label">Control RTT</div><div
      ↪  class="stat-val" id="sCRtt"></div></div>
      <div class="stat-cell"><div class="stat-label">Control Avg</div><div
      ↪  class="stat-val" id="sCRttAvg"></div></div>
      <div class="stat-cell"><div class="stat-label">BLE RTT</div><div class="stat-val"
      ↪  id="sBleRtt"></div></div>
      <div class="stat-cell"><div class="stat-label">BLE Avg</div><div class="stat-val"
      ↪  id="sBleAvg"></div></div>
    </div>
  </div>
```

```
    <!-- CALIBRATION -->
    <div class="panel-card">
      <div class="card-header">
        <span class="card-title">Speed Calibration</span>
        <span class="badge" id="calBadge">Not run</span>
      </div>
      <div style="padding:14px 20px; display:flex; flex-direction:column; gap:10px;">
        <div style="font-size:0.78rem; color:var(--muted); line-height:1.5;">
          Place mouse at bottom of play area facing center. Tap Calibrate  the robot
          ↪  finds its minimum moving speed, then measures turn rate. Results update
          ↪  automatically via the event log.
        </div>
        <div id="calResults" style="display:none; font-size:0.78rem;
        ↪  font-variant-numeric:tabular-nums; line-height:1.8;"></div>
        <button class="btn" id="calBtn" onclick="startCal()"
        ↪  style="margin-top:4px;">Calibrate</button>
      </div>
    </div>

    <!-- LOG -->
    <div class="panel-card">
      <div class="card-header">
        <span class="card-title">Event Log</span>
      </div>
      <div class="log-box" id="logBox"></div>
    </div>

  </div>
</main>

<script>
  // LOG
  function log(msg, type = 'info') {
    const box = document.getElementById('logBox');
    const now = new Date();
    const t = `${String(now.getHours()).padStart(2,'0')}:${String(now.getMinutes()).padS
    ↪  tart(2,'0')}:${String(now.getSeconds()).padStart(2,'0')}`;
    const row = document.createElement('div');
    row.className = `log-row ${type}`;
    row.innerHTML = `<span class="log-t">${t}</span><span class="log-m">${msg}</span>`;
    box.appendChild(row);
    box.scrollTop = box.scrollHeight;
  }

  // STATUS
  function setStatus(cls, label) {
    document.getElementById('statusPill').className = `status-pill ${cls}`;
    document.getElementById('statusText').textContent = label;
  }

  // WEBRTC
  let pc = null, dc = null, statsTimer = null;
```

```
async function connect() {
  const btn = document.getElementById('connectBtn');
  btn.disabled = true; btn.textContent = 'Connecting...';
  setStatus('connecting', 'Connecting');
  log('Initiating connection...');

  // Low-latency RTCPeerConnection config
  pc = new RTCPeerConnection({
    iceServers: [],  // local-only, skip STUN round-trips
    bundlePolicy: 'max-bundle',
    rtcpMuxPolicy: 'require',
  });

  dc = pc.createDataChannel('control', { ordered: false, maxRetransmits: 0 });
  dc.onopen  = () => {
    log('Data channel open', 'success');
    setStat('sData','Open','good');
    setupPingPong();
    // Sync current UI mode state to server immediately
    sendMode(document.getElementById('modeToggle').checked);
  };
  dc.onclose = () => { log('Data channel closed', 'warn');
  ↪  setStat('sData','Closed','warn'); };

  pc.ontrack = (e) => {
    log('Video track received', 'success');
    const video = document.getElementById('remoteVideo');
    video.srcObject = e.streams[0];
    // Minimize jitter buffer | reduces latency at cost of occasional glitch
    if (video.srcObject && 'jitterBufferTarget' in video) {
      video.jitterBufferTarget = 0;
    }
    document.getElementById('videoOverlay').classList.add('hidden');
    setStat('sVideo','Live','good');
  };

  pc.onconnectionstatechange = () => {
    const s = pc.connectionState;
    log(`Connection: ${s}`, s==='connected'?'success':s==='failed'?'error':'info');
    setStat('sState', s.charAt(0).toUpperCase()+s.slice(1));
    if (s === 'connected') {
      setStatus('connected','Connected');
      btn.textContent = 'Disconnect'; btn.disabled = false;
      btn.onclick = disconnect;
      statsTimer = setInterval(pollStats, 2000);
    } else if (s === 'failed' || s === 'disconnected') {
      setStatus('failed','Disconnected');
      btn.textContent = 'Reconnect'; btn.disabled = false;
      btn.onclick = connect;
      clearInterval(statsTimer);
    }
  };

  const offer = await pc.createOffer({ offerToReceiveVideo: true });
```

```
  // SDP tweaks for low latency:
  // - x-google-min/max-bitrate: keeps encoder from buffering for quality
  // - level-asymmetry-allowed: lets browser pick lowest-latency H.264 level
  let sdp = offer.sdp;
  sdp = sdp.replace(/(m=video.*\r\n)/,
    '$1a=fmtp:96
    ↪  level-asymmetry-allowed=1;packetization-mode=1;profile-level-id=42e01f\r\n');
  sdp = sdp.replace('useinbandfec=1', 'useinbandfec=1; stereo=0;
  ↪  maxplaybackrate=16000');

  await pc.setLocalDescription({ type: offer.type, sdp });

  try {
    const res = await fetch('/offer', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ sdp: pc.localDescription.sdp, type:
      ↪  pc.localDescription.type })
    });
    const answer = await res.json();

    // Force low-latency bitrate in answer SDP
    answer.sdp = answer.sdp.replace(
      /a=fmtp:(\d+) (.*profile-level-id.*)/g,
      'a=fmtp:$1 $2\r\na=rtcp-fb:$1 transport-cc\r\nb=AS:2000'
    );

    await pc.setRemoteDescription(answer);
    log('Answer received | low-latency mode active', 'success');
  } catch(err) {
    log(`Server error: ${err.message}`, 'error');
    setStatus('failed','Error');
    btn.textContent = 'Retry'; btn.disabled = false; btn.onclick = connect;
  }
}

function disconnect() {
  if (pc) { pc.close(); pc = null; }
  clearInterval(statsTimer);
  setStatus('','Disconnected');
  document.getElementById('videoOverlay').classList.remove('hidden');
  ['sState','sVideo','sData','sRtt','sRttAvg','sCRtt','sCRttAvg','sBleRtt','sBleAvg'].
  ↪  forEach(id => {
    const el = document.getElementById(id);
    el.textContent = '|'; el.className = 'stat-val';
  });
  const btn = document.getElementById('connectBtn');
  btn.textContent = 'Connect'; btn.disabled = false; btn.onclick = connect;
  log('Disconnected', 'warn');
}

// Rolling average helper | keeps last N samples
function RollingAvg(n) {
```

```
  const buf = [];
  return {
    push(v) { buf.push(v); if (buf.length > n) buf.shift(); },
    avg()   { return buf.length ? Math.round(buf.reduce((a,b)=>a+b,0)/buf.length) :
    ↪  null; }
  };
}

const videoAvg   = RollingAvg(10);
const controlAvg = RollingAvg(10);

// Control RTT | send a timestamped ping over the data channel,
// server echoes it back, we measure the round trip.
let pendingPing = null;

function sendPing() {
  if (!dc || dc.readyState !== 'open') return;
  pendingPing = performance.now();
  dc.send(JSON.stringify({ ping: pendingPing }));
}

// Patch the data channel onmessage to catch pong replies
function setupPingPong() {
  if (!dc) return;
  dc.onmessage = (e) => {
    try {
      const msg = JSON.parse(e.data);
      if (msg.pong && pendingPing !== null) {
        const rtt = Math.round(performance.now() - pendingPing);
        pendingPing = null;
        controlAvg.push(rtt);
        const avg = controlAvg.avg();
        setStat('sCRtt',    `${rtt} ms`,  rtt  < 80 ? 'good' : rtt  < 200 ? 'warn' :
        ↪  'bad');
        setStat('sCRttAvg', `${avg} ms`,  avg  < 80 ? 'good' : avg  < 200 ? 'warn' :
        ↪  'bad');
      }
    } catch {}
  };
}

async function pollStats() {
  if (!pc) return;
  const stats = await pc.getStats();
  let rttMs = null;

  // Source 1: remote-inbound-rtp
  stats.forEach(s => {
    if (s.type === 'remote-inbound-rtp' && s.roundTripTime != null)
      rttMs = Math.round(s.roundTripTime * 1000);
  });
  // Source 2: succeeded candidate-pair
  if (rttMs === null) {
    stats.forEach(s => {
```

```
      if (s.type === 'candidate-pair' && s.state === 'succeeded' &&
      ↪  s.currentRoundTripTime != null)
        rttMs = Math.round(s.currentRoundTripTime * 1000);
    });
  }
  // Source 3: nominated candidate-pair
  if (rttMs === null) {
    stats.forEach(s => {
      if (s.type === 'candidate-pair' && s.nominated && s.currentRoundTripTime != null)
        rttMs = Math.round(s.currentRoundTripTime * 1000);
    });
  }

  if (rttMs !== null) {
    videoAvg.push(rttMs);
    const avg = videoAvg.avg();
    setStat('sRtt',    `${rttMs} ms`, rttMs < 80 ? 'good' : rttMs < 200 ? 'warn' :
    ↪  'bad');
    setStat('sRttAvg', `${avg} ms`,  avg   < 80 ? 'good' : avg   < 200 ? 'warn' :
    ↪  'bad');
  }

  // Send a control channel ping every stats poll cycle
  sendPing();
}

function setStat(id, val, cls = '') {
  const el = document.getElementById(id);
  el.textContent = val;
  el.className = `stat-val ${cls}`;
}

document.getElementById('connectBtn').onclick = connect;

//  JOYSTICK
const zone = document.getElementById('joystickZone');
const knob = document.getElementById('joystickKnob');
const MAX_R = 68;
let active = false, sendLoop = null;

function center() {
  const r = zone.getBoundingClientRect();
  return { x: r.left + r.width/2, y: r.top + r.height/2 };
}

function moveKnob(cx, cy) {
  const c = center();
  let dx = cx - c.x, dy = cy - c.y;
  const d = Math.hypot(dx, dy);
  if (d > MAX_R) { dx = dx/d*MAX_R; dy = dy/d*MAX_R; }
  knob.style.transform = `translate(calc(-50% + ${dx}px), calc(-50% + ${dy}px))`;
  const nx = +(dx/MAX_R).toFixed(2);
  const ny = +(-dy/MAX_R).toFixed(2);
  document.getElementById('cX').textContent = nx.toFixed(2);
```

```
  document.getElementById('cY').textContent = ny.toFixed(2);
  return { nx, ny };
}

function release() {
  active = false;
  zone.classList.remove('active');
  knob.style.transition = 'transform 0.15s ease';
  knob.style.transform = 'translate(-50%, -50%)';
  document.getElementById('cX').textContent = '0.00';
  document.getElementById('cY').textContent = '0.00';
  // Send stop twice to make sure it arrives | one may be dropped on lossy connection
  if (dc && dc.readyState === 'open') {
    dc.send(JSON.stringify({x:0, y:0, z:0}));
    setTimeout(() => { if (dc && dc.readyState === 'open')
    ↪  dc.send(JSON.stringify({x:0,y:0,z:0})); }, 60);
  }
  clearInterval(sendLoop); sendLoop = null;
  setTimeout(() => { knob.style.transition = ''; }, 150);
}

zone.addEventListener('pointerdown', (e) => {
  active = true;
  zone.classList.add('active');
  knob.style.transition = '';
  zone.setPointerCapture(e.pointerId);
  moveKnob(e.clientX, e.clientY);
  sendLoop = setInterval(() => {
    if (dc && dc.readyState === 'open') {
      const x = parseFloat(document.getElementById('cX').textContent);
      const y = parseFloat(document.getElementById('cY').textContent);
      dc.send(JSON.stringify({x: -x, y, z:0}));  // negate x: right stick = right turn
    }
  }, 50);
});

zone.addEventListener('pointermove', (e) => { if (active) moveKnob(e.clientX,
↪  e.clientY); });
zone.addEventListener('pointerup',     release);
zone.addEventListener('pointercancel', release);

//  VIEWER POLLING
async function pollViewers() {
  try {
    const res = await fetch('/viewers');
    if (res.status === 401) { window.location.href = '/login'; return; }
    const { viewers } = await res.json();
    const others = viewers.filter(v => !v.is_you);
    const banner = document.getElementById('viewerBanner');
    const bannerText = document.getElementById('viewerBannerText');
    if (others.length > 0) {
      const list = others.map(v => `${v.ip} (since ${v.connected_at})`).join(', ');
      bannerText.textContent = `${others.length} other device${others.length > 1 ? 's
      ↪  are' : ' is'} viewing the camera feed: ${list}`;
```

```
      banner.style.display = 'flex';
    } else {
      banner.style.display = 'none';
    }
  } catch(e) { /* server unreachable, ignore */ }
}

// Poll viewers every 5 seconds once page loads
setInterval(pollViewers, 5000);
pollViewers();

//  SERVER LOG STREAM (SSE)
let sseSource = null;
function startLogStream() {
  if (sseSource) sseSource.close();
  sseSource = new EventSource('/log-stream');
  sseSource.onmessage = (e) => {
    try {
      const { msg, level } = JSON.parse(e.data);

      // BLE RTT | update stats silently
      const bleMatch = msg.match(/BLE RTT: (\d+) ms \/ (\d+) ms avg/);
      if (bleMatch) {
        const rtt = parseInt(bleMatch[1]);
        const avg = parseInt(bleMatch[2]);
        setStat('sBleRtt', `${rtt} ms`, rtt < 20 ? 'good' : rtt < 50 ? 'warn' : 'bad');
        setStat('sBleAvg', `${avg} ms`, avg < 20 ? 'good' : avg < 50 ? 'warn' : 'bad');
        return;
      }

      // Calibration ended | sync toggle back to manual without sending a mode message
      if (msg === 'cal_done_manual') {
        modeToggle.checked = false;
        modeBadge.textContent = 'MANUAL';
        modeBadge.className = 'mode-badge manual';
        const jZone = document.getElementById('joystickZone');
        jZone.style.opacity = '1'; jZone.style.pointerEvents = '';
        return;
      }

      // Calibration complete | parse result from log message and update UI
      if (msg.startsWith('Calibration complete')) {
        const lvlM  = msg.match(/level=(\d+)/);
        const pxM   = msg.match(/([\d.]+)px\/s fwd/);
        const degM  = msg.match(/([\d.]+)°\/s turn/);
        _onCalComplete({
          fwd_level: lvlM  ? parseInt(lvlM[1])    : null,
          px_per_s:  pxM   ? parseFloat(pxM[1])   : null,
          deg_per_s: degM  ? parseFloat(degM[1])  : null,
        });
      }

      log('[jetson] ' + msg, level);
    } catch {}
```

```
  };
  sseSource.onerror = () => {
    setTimeout(() => {
      if (sseSource && sseSource.readyState === EventSource.CLOSED) startLogStream();
    }, 3000);
  };
}
startLogStream();

// ── MODE TOGGLE ──
const modeToggle = document.getElementById('modeToggle');
const modeBadge  = document.getElementById('modeBadge');

function sendMode(isAuto) {
  if (dc && dc.readyState === 'open') {
    dc.send(JSON.stringify({ mode: isAuto ? 'auto' : 'manual' }));
  }
  modeBadge.textContent = isAuto ? 'AUTO' : 'MANUAL';
  modeBadge.className = `mode-badge ${isAuto ? 'auto' : 'manual'}`;
  const jZone = document.getElementById('joystickZone');
  jZone.style.opacity       = isAuto ? '0.35' : '1';
  jZone.style.pointerEvents = isAuto ? 'none' : '';
  // Update video hint
  const hint = document.getElementById('videoHint');
  if (hint) hint.textContent = isAuto ? 'Tap video to navigate' : '640 × 480';
  log(`Drive mode  ${isAuto ? 'AUTO (tap video to navigate)' : 'MANUAL (joystick)'}`,
  ↪  'info');
}

modeToggle.addEventListener('change', () => sendMode(modeToggle.checked));

// ── CLICK-TO-NAVIGATE ──
// Camera resolution the server streams at
const CAM_W = 640, CAM_H = 480;

document.getElementById('remoteVideo').addEventListener('click', (e) => {
  if (!dc || dc.readyState !== 'open') return;
  if (!document.getElementById('modeToggle').checked) return; // manual mode only
  ↪  joystick

  const video = e.currentTarget;
  const rect  = video.getBoundingClientRect();

  // Click position relative to rendered video element
  const relX = (e.clientX - rect.left)  / rect.width;
  const relY = (e.clientY - rect.top)   / rect.height;

  // Map to camera pixel space
  const camX = Math.round(relX * CAM_W);
  const camY = Math.round(relY * CAM_H);

  dc.send(JSON.stringify({ goto: { x: camX, y: camY } }));
  log(`Nav → (${camX}, ${camY})`, 'info');
```

```
  // Show target dot
  const dot = document.getElementById('navDot');
  dot.style.left    = `${relX * 100}%`;
  dot.style.top     = `${relY * 100}%`;
  dot.style.display = 'block';
  dot.style.opacity = '1';
  setTimeout(() => { dot.style.opacity = '0'; }, 1500);
  setTimeout(() => { dot.style.display = 'none'; }, 2100);
});

//  BLE RECONNECT
async function bleReconnect() {
  const btn = document.getElementById('bleBtn');
  btn.textContent = 'BLE...'; btn.disabled = true;
  try {
    const res = await fetch('/ble-reconnect', { method: 'POST' });
    const data = await res.json();
    log(data.msg || 'BLE reconnect triggered', res.ok ? 'info' : 'error');
  } catch(e) {
    log('BLE reconnect request failed', 'error');
  }
  setTimeout(() => { btn.textContent = 'BLE'; btn.disabled = false; }, 3000);
}

//  CALIBRATION
async function startCal() {
  const btn   = document.getElementById('calBtn');
  const badge = document.getElementById('calBadge');
  btn.disabled = true; btn.textContent = 'Running...';
  badge.textContent = 'Running';
  badge.style.background = 'rgba(245,200,66,0.15)';
  badge.style.color = 'var(--gold)';
  document.getElementById('calResults').style.display = 'none';
  log('Calibration started | place mouse at bottom facing center', 'warn');
  try {
    const res  = await fetch('/calibrate', { method: 'POST' });
    const data = await res.json();
    if (!res.ok) {
      log(`Calibration error: ${data.error}`, 'error');
      btn.disabled = false; btn.textContent = 'Calibrate';
      badge.textContent = 'Failed'; badge.style.color = 'var(--red)';
    }
    // Completion is driven by SSE | see startLogStream handler below
  } catch(e) {
    log(`Calibration request failed: ${e.message}`, 'error');
    btn.disabled = false; btn.textContent = 'Calibrate';
  }
}

function _onCalComplete(data) {
  const btn   = document.getElementById('calBtn');
  const badge = document.getElementById('calBadge');
  const div   = document.getElementById('calResults');
  btn.disabled = false; btn.textContent = 'Re-calibrate';
```

```
    const ok = data.fwd_level && data.px_per_s;
    badge.textContent = ok ? 'Done' : 'Failed';
    badge.style.background = ok ? 'rgba(74,222,128,0.12)' : 'rgba(248,113,113,0.1)';
    badge.style.color = ok ? 'var(--green)' : 'var(--red)';
    if (ok) {
      const degStr = data.deg_per_s ? `${data.deg_per_s.toFixed(1)} /s` : 'not measured';
      div.innerHTML = `
        <div style="display:flex;justify-content:space-between;color:var(--text);">
          <span>Min moving level</span><span style="color:var(--green)">Level
          ↪  ${data.fwd_level}</span>
        </div>
        <div style="display:flex;justify-content:space-between;color:var(--text);">
          <span>Forward speed</span><span
          ↪  style="color:var(--green)">${data.px_per_s.toFixed(1)} px/s</span>
        </div>
        <div style="display:flex;justify-content:space-between;color:var(--text);">
          <span>Turn rate</span>
          <span style="color:${data.deg_per_s ? 'var(--green)' :
          ↪  'var(--muted)'}">${degStr}</span>
        </div>`;
      div.style.display = 'block';
    }
  }

  // INIT
  log('Panel ready | press Connect to start');
</script>
</body>
</html>
```

U Embedded Code Running on the Mouse

```
#include <NimBLEDevice.h>

static const int FWD1 = 4;
static const int REV1 = 2;
static const int FWD2 = 12;
static const int REV2 = 5;

static const char* SERVICE_UUID = "6e400001-b5a3-f393-e0a9-e50e24dcca9e";
static const char* CHAR_UUID    = "6e400002-b5a3-f393-e0a9-e50e24dcca9e";

// 0..15 nibble command stored for each motor
volatile uint8_t motor1Cmd = 0;
volatile uint8_t motor2Cmd = 0;

// software PWM counter
volatile uint8_t pwmCounter = 0;

// Decode nibble:
// 0x0 = stop
// 0x1..0x7 = forward speed 1..7
// 0x8 = stop
// 0x9..0xF = reverse speed 1..7
void decodeMotorNibble(uint8_t nibble, bool &forward, uint8_t &duty) {
  forward = true;
  duty = 0;

  if (nibble == 0x0 || nibble == 0x8) {
    duty = 0;
    return;
  }

  if (nibble >= 0x1 && nibble <= 0x7) {
    forward = true;
    duty = nibble;   // 1..7
    return;
  }

  if (nibble >= 0x9 && nibble <= 0xF) {
    forward = false;
    duty = nibble - 0x8;   // 1..7
    return;
  }
}

// Apply one motor command using software PWM.
// duty is 0..7, pwmCounter should wrap 0..7.
void driveMotor(int fwdPin, int revPin, uint8_t nibbleCmd) {
  bool forward;
  uint8_t duty;
  decodeMotorNibble(nibbleCmd, forward, duty);

  if (duty == 0) {
```

```
    digitalWrite(fwdPin, LOW);
    digitalWrite(revPin, LOW);
    return;
  }

  bool on = (pwmCounter < duty);

  if (forward) {
    digitalWrite(fwdPin, on ? HIGH : LOW);
    digitalWrite(revPin, LOW);
  } else {
    digitalWrite(fwdPin, LOW);
    digitalWrite(revPin, on ? HIGH : LOW);
  }
}

class CmdCallbacks : public NimBLECharacteristicCallbacks {
  void onWrite(NimBLECharacteristic* c, NimBLEConnInfo& connInfo) override {
    std::string v = c->getValue();
    if (v.empty()) return;

    // Expect exactly 1 byte from BLE
    uint8_t b = (uint8_t)v[0];

    motor1Cmd = (b >> 4) & 0x0F;   // upper nibble
    motor2Cmd = b & 0x0F;          // lower nibble

    Serial.printf("RX byte 0x%02X -> M1=0x%X M2=0x%X\n", b, motor1Cmd, motor2Cmd);
  }
};

class ServerCallbacks : public NimBLEServerCallbacks {
  void onDisconnect(NimBLEServer* s, NimBLEConnInfo& ci, int reason) override {
    NimBLEDevice::startAdvertising();
  }
};

void setup() {
  pinMode(FWD1, OUTPUT);
  pinMode(REV1, OUTPUT);
  pinMode(FWD2, OUTPUT);
  pinMode(REV2, OUTPUT);

  digitalWrite(FWD1, LOW);
  digitalWrite(REV1, LOW);
  digitalWrite(FWD2, LOW);
  digitalWrite(REV2, LOW);

  Serial.begin(115200);
  delay(200);

  NimBLEDevice::init("ToyESP32");
  NimBLEDevice::setPower(ESP_PWR_LVL_P9);
```

```
  NimBLEServer* server = NimBLEDevice::createServer();
  server->setCallbacks(new ServerCallbacks());

  NimBLEService* svc = server->createService(SERVICE_UUID);

  NimBLECharacteristic* cmd = svc->createCharacteristic(
    CHAR_UUID,
    NIMBLE_PROPERTY::WRITE | NIMBLE_PROPERTY::WRITE_NR
  );
  cmd->setCallbacks(new CmdCallbacks());
  cmd->setValue(std::string(1, '\x00'));

  svc->start();

  NimBLEAdvertising* adv = NimBLEDevice::getAdvertising();
  adv->addServiceUUID(SERVICE_UUID);
  adv->setName("ToyESP32");
  adv->start();

  Serial.println("BLE advertising: ToyESP32");
}

void loop() {
  // 3-bit PWM phase: 0..7
  pwmCounter = (pwmCounter + 1) & 0x07;

  driveMotor(FWD1, REV1, motor1Cmd);
  driveMotor(FWD2, REV2, motor2Cmd);

  delayMicroseconds(1200);
}
```