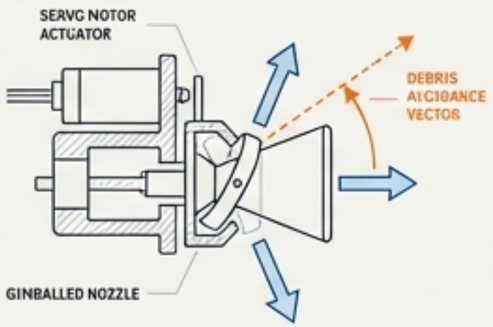
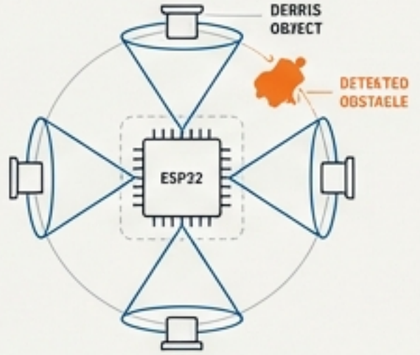
AI-Powered Swizzle Nozzle

Prototyping Autonomous Spacecraft Debris Avoidance



THRUST VECTORING ACTUATION (ESP32 CONTROLLED)

A complete system specification for an ESP32-based prototype that simulates spacecraft thrust vectoring. The system uses servo-controlled 'swizzle nozzles' and ultrasonic sensors for 360° debris detection, featuring real-time web monitoring and architecture designed for future AI integration.



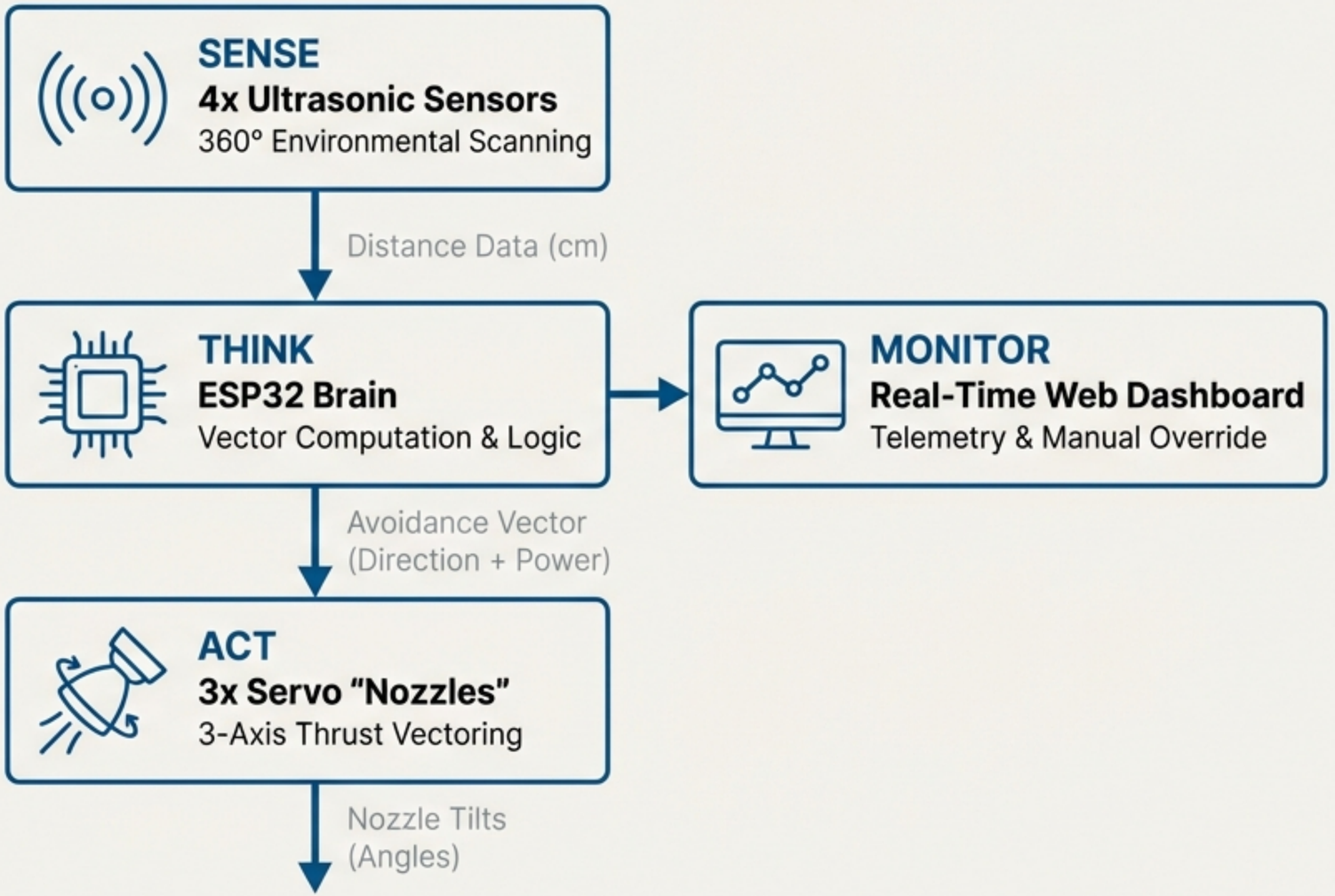
360° ULTRASONIC SENSOR ARRAY (REAL-TIME MAPPING)

The System at a Glance: Sense, Think, Act

Purpose

- **Primary Goal**: Simulate spacecraft thruster vectoring to avoid debris detected via ultrasonic sensors.
- **Educational Platform**: A hands-on tool for learning embedded systems, sensor fusion, control algorithms, and web-based telemetry.

Core Operational Loop

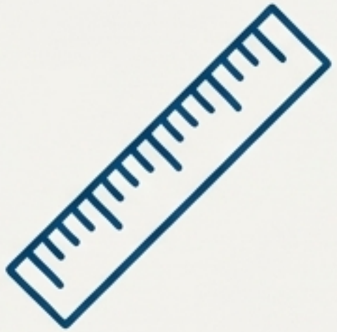


Engineered for High Performance

<150 ms

Response Time

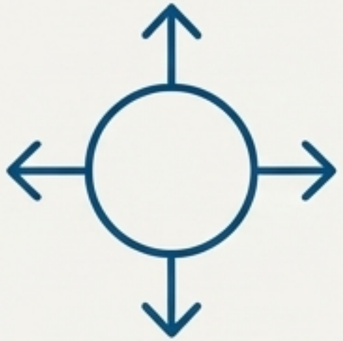
Total latency from initial obstacle detection to servo command execution.



±2 cm

Detection Accuracy

Sensor accuracy within a 10-300 cm range, ensuring precise threat assessment.



360° Coverage

Situational Awareness

Four ultrasonic sensors (Front, Rear, Left, Right) provide a complete environmental scan at a 10Hz refresh rate.

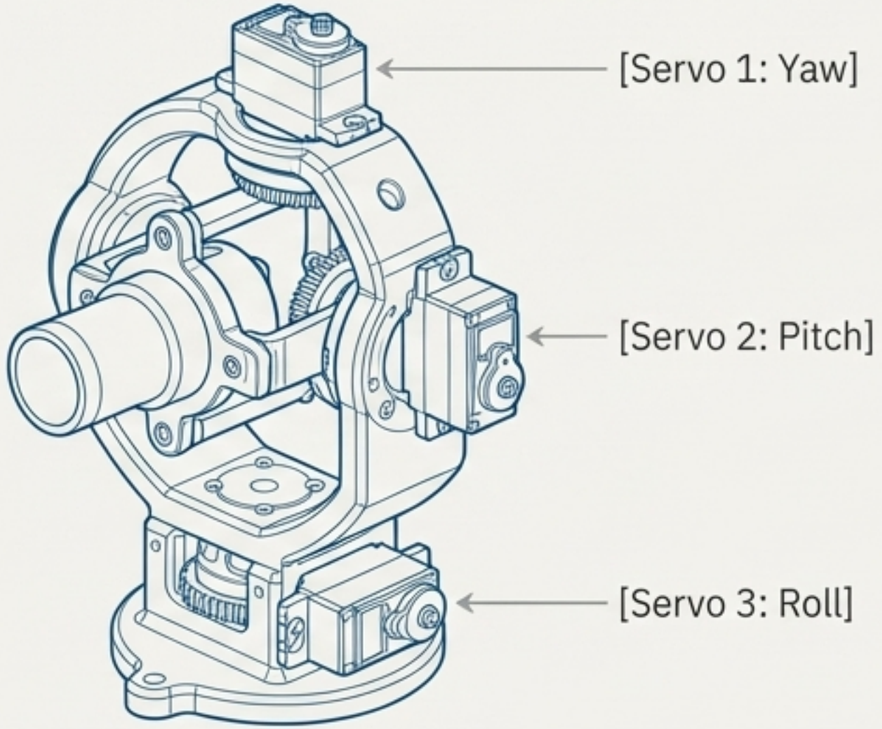
5 Hz Updates

Real-Time Telemetry

Live sensor data, vector calculations, and servo angles are broadcast to the web dashboard every 200ms via WebSocket.

The Physical Assembly: Mechanism and Sensors

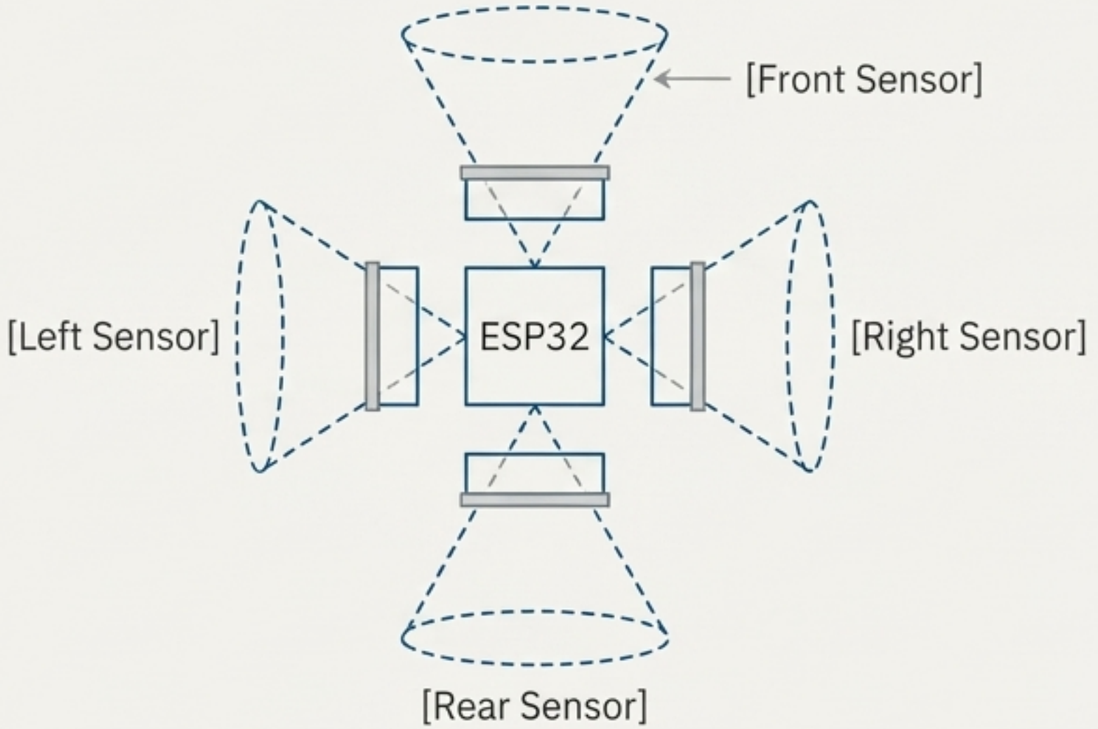
3-Axis Swizzle Nozzle Mechanism



Key Specifications:

- **Servos:** MG90S Micro Servos
- **Range of Motion:** 60° to 120° (±30° from neutral)
- **Construction:** 3D-printed PLA/ABS gimbal frame for light weight (<200g total).

360° Sensor Array

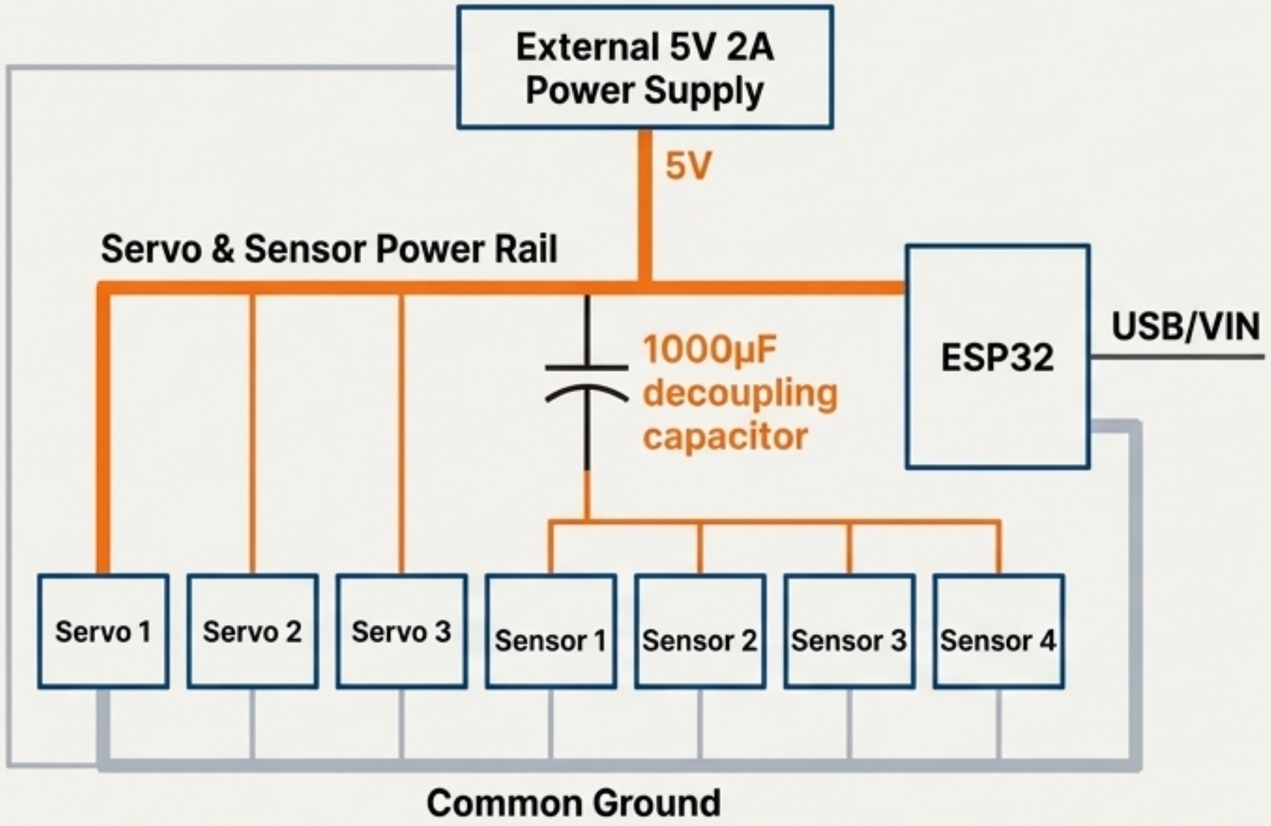


Key Specifications:

- **Sensors:** HC-SR04 Ultrasonic Sensors
- **Detection Range:** 2cm to 400cm
- **Mounting:** All sensors at the same elevation with >5cm spacing to prevent interference.

The Nervous System: A Robust Power & Wiring Architecture

Power System Design



Critical Power Considerations

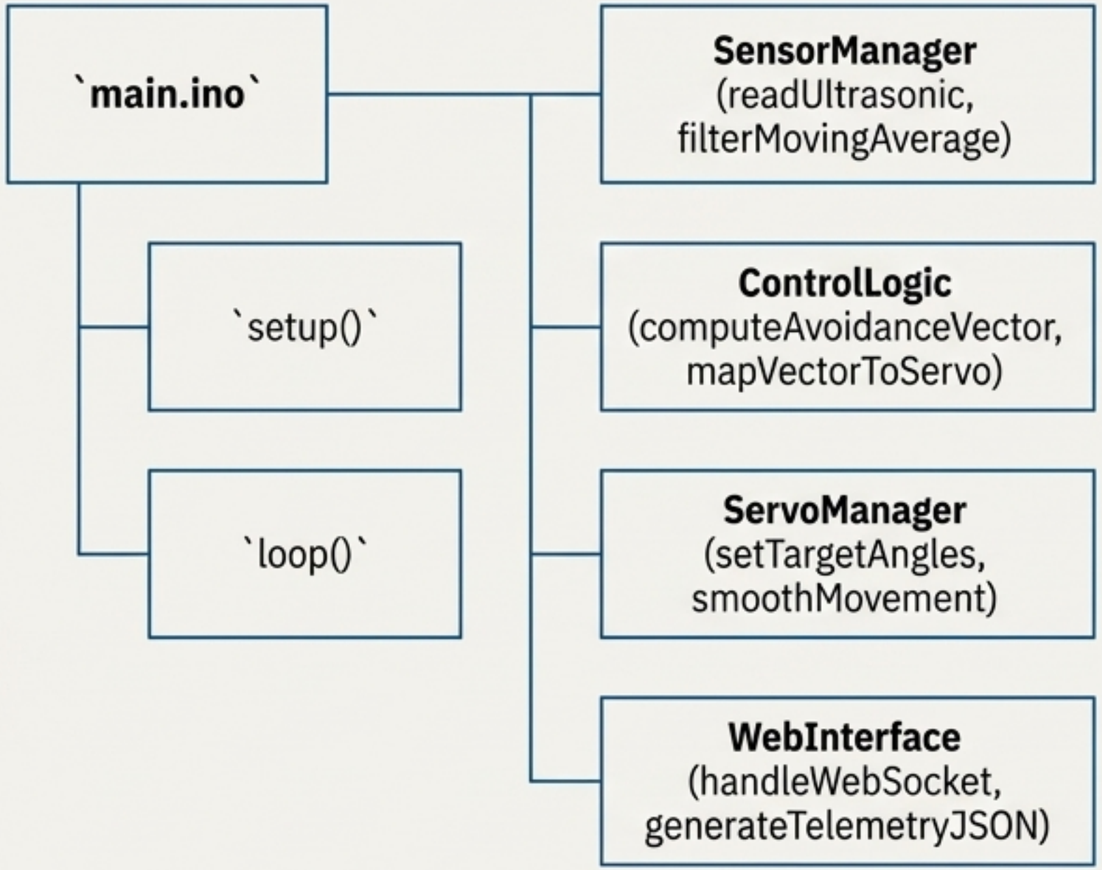
Separate Servo Power: NEVER power servos from the ESP32's 5V pin. An external 5V 2A+ supply is mandatory to handle peak currents up to 600mA per servo.

Common Ground: The ESP32 GND *must* be connected to the servo power supply GND to ensure stable signal reference and prevent erratic behavior.

Decoupling Capacitors: A large 1000µF capacitor across the servo power rail stabilizes voltage during current surges, preventing ESP32 brownouts. 100µF capacitors are placed on each sensor's power line.

The Software Brain: A Responsive, Non-Blocking Architecture

Modular Code Structure



The Non-Blocking `loop()`

```
// The system uses millis()-based timing to maintain responsiveness.
// No blocking delay() calls.

unsigned long lastSensorRead = 0;
const int SENSOR_INTERVAL = 100; // ms

void loop() {
  unsigned long currentTime = millis();

  // Read sensors on a fixed interval
  if (currentTime - lastSensorRead >= SENSOR_INTERVAL) {
    lastSensorRead = currentTime;
    readAllSensors();
    computeAvoidanceVector();
  }

  // Other tasks run continuously without blocking
  smoothServoMovement();
  handleWebSocketMessages();
}
```

Benefit: Sensors, servos, and the web server operate in parallel, ensuring consistent timing and a sensor-to-servo latency of under 150ms.

The Core Algorithm: Translating Distance into Direction

Inverse Distance Weighting

1. **Calculate Threat Weight:** For each sensor detecting an obstacle under a **50cm** threshold, a weight is calculated: **weight = (threshold - distance) / threshold**. Closer objects receive a higher weight.
2. **Compute Vector Components:** The final vector is a sum of weighted directions: **vector.x = weight[LEFT] - weight[RIGHT]**; **vector.y = weight[REAR] - weight[FRONT]**.
3. **Map to Servos:** The normalized vector (x, y) is mapped directly to Yaw and Pitch servo angles, while its magnitude maps to the Roll (thrust) servo.

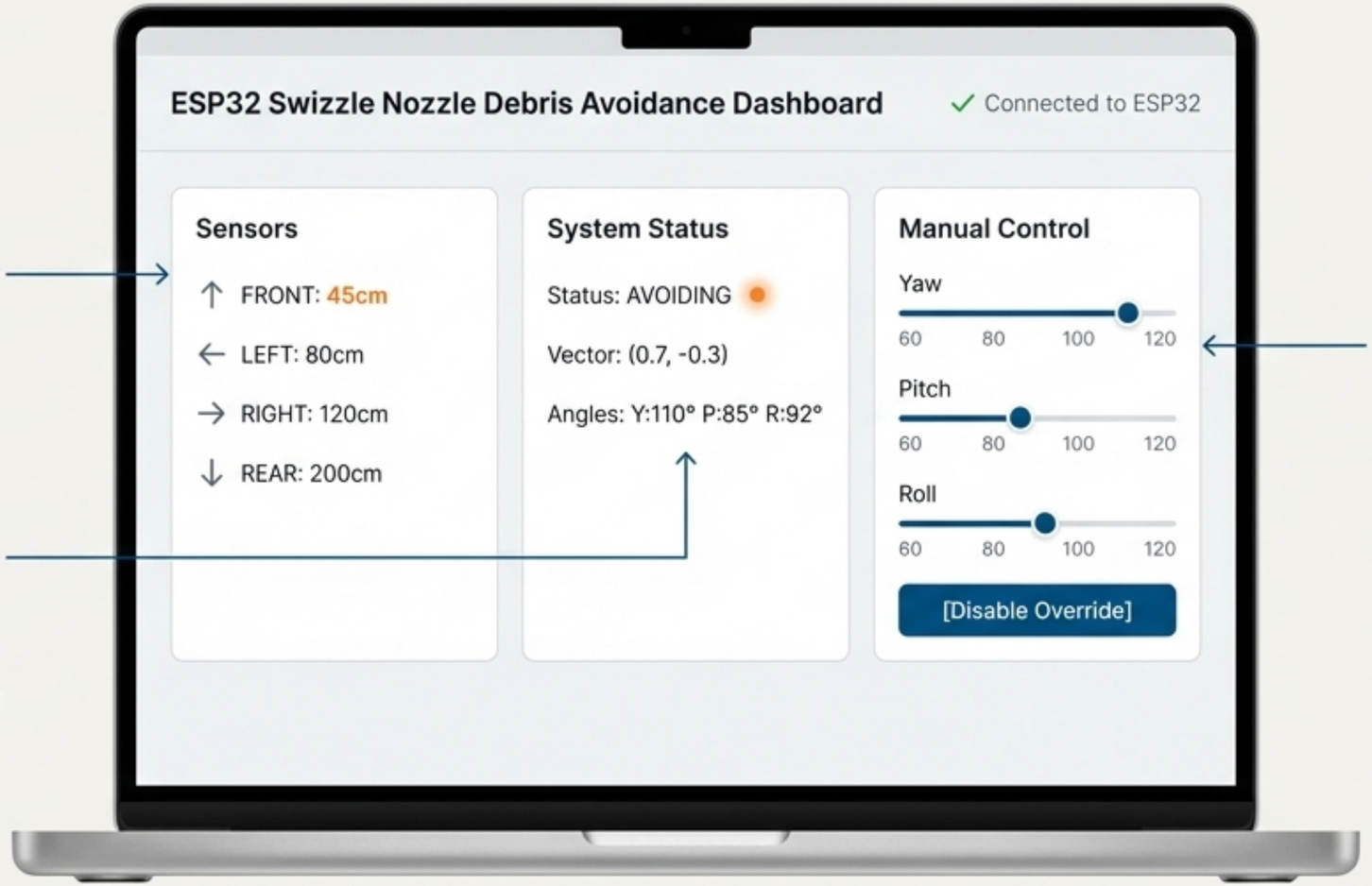
Example Scenarios

Scenario	Detected Distances	Resulting Vector	Servo Action
Front Obstacle	F=20cm	**(0, -1)**	Pitch Down (Thrust Backwards)
Left Obstacle	L=15cm	**(1, 0)**	Yaw Right (Thrust Away)
Front-Left	F=25cm, L=25cm	**(0.707, -0.707)**	Diagonal Back-Right
Surrounded	All < 30cm	**(0, 0)**	Neutral (Conflicting Threats)

The Human Interface: A Real-Time Control Dashboard

1. Live Telemetry:
UI updates 5 times per second using a low-latency WebSocket connection.

3. Responsive Design:
The layout adapts automatically for viewing on both desktop and mobile devices.



2. Full Control:
An "Enable Override" button allows for direct manual positioning of all three servos via sliders.

More Than a Prototype: AI-Ready by Design

The system's hardware and software were architected from the ground up to serve as a platform for developing and testing advanced AI control algorithms.

Pluggable AI Interface

The architecture supports receiving external servo commands via WiFi or Serial, allowing the onboard rule-based algorithm to be swapped with an offboard AI controller without hardware changes.

ML-Compatible Telemetry

System data (distances, vector, servo angles, status, timestamp) is packaged packaged into clean JSON format every 200ms, creating a data stream ideal for training machine learning pipelines.

Fail-Safe Operation

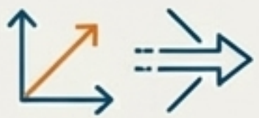
If the connection to an external AI is lost or provides invalid commands, the system is designed to seamlessly fall back to its robust, onboard vector; fall back to robust, onboard vector computation algorithm.

The Evolution Roadmap: From Rules to Reinforcement Learning

Stage 1: Rule-Based Logic (Current System)

Method: Direct vector computation from sensor data.

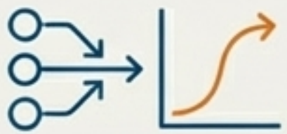
Capability: Fast, deterministic, and reliable obstacle avoidance. Serves as the performance baseline.



Stage 2: Sensor Fusion & Prediction

Method: Kalman Filters and LSTM Neural Networks.

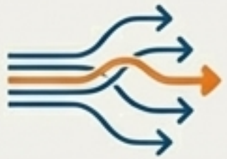
Capability: Moves from purely reactive to predictive. Smooths noisy sensor data and anticipates debris trajectories based on past movements.



Stage 3: Optimized Path Planning

Method: Monte Carlo or Genetic Algorithms.

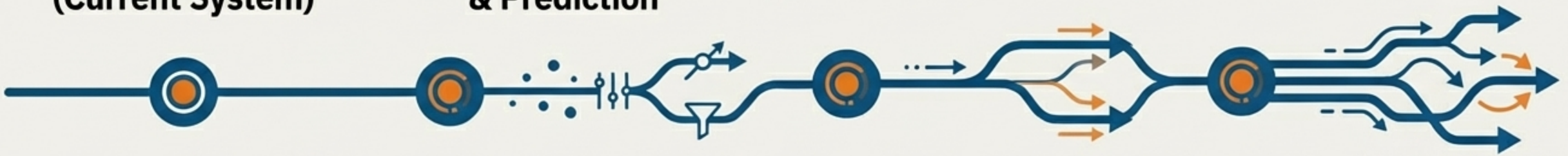
Capability: Simulates and evaluates hundreds of potential avoidance maneuvers to select the one that is safest and most efficient.



Stage 4: Reinforcement Learning (The Goal)

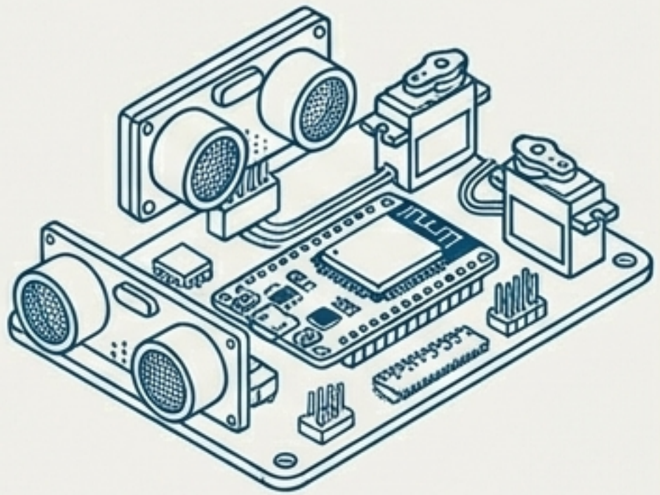
Method: Proximal Policy Optimization (PPO) in a simulated environment.

Capability: The system learns from experience. An AI agent is rewarded for successful avoidance and penalized for collisions, discovering novel control strategies that surpass human-programmed logic.



The AI Data Pipeline: Closing the Learning Loop

[ESP32 Prototype]



JSON Telemetry via WiFi (every 200ms)

```
{"distances": [45, 80, …],
  "servos": ...
}
```

Optimized Servo Commands

```
{"action": "override_on",
  "yaw": 105,
  "pitch": 88, ...
}
```

[Laptop / AI Controller]

AI Command Validation

All incoming commands from the external AI are validated on the ESP32 before execution. Commands that violate angle limits (60°-120°) or exceed maximum rate-of-change are rejected, ensuring the hardware remains safe.

About the Creator

Sanchit (‘Dark’)

A 16-year-old 11th-grade student and space technology enthusiast. With hands-on experience in Arduino, Processing, Processing, 3D modeling, and machine learning, he combines academic studies with ambitious prototype development to explore autonomous spacecraft navigation.

> **Building the future of space exploration, one prototype at a time.**

Project Context: This work is part of the ExoNova AI project.
License: Open-source hardware and software (MIT License).