Human-Following Robot with Jetson Orin Nano & Teensy

SYSTEM DOCUMENTATION & DESIGN REPORT

SENITHU DAMPEGAMA – ROBOTICS AND AI ENGINEER

Contents

1. Abstract 3
2. Background & Motivation 4
3. System Architecture 5
3.1 Layer Overview 5
3.2 System Flow (High-Level) 6
3.3 Diagram 6
4. Electronics & Component Selection 7
4.1 Controller Selection 7
4.2 Motor Drivers 7
4.3 Sensors 7
4.4 Power System 7
4.5 Wiring & Interfaces 8
4.6 Wiering diagram 8
5. Electronics Design & Safety 10
5.1 Power Distribution 10
5.2 Stall Protection & Thermal Safety 10
5.3 Sensor Reliability 10
5.4 Telemetry & Fail-safes 10
5.5 Summary 11
6. Mechanical Design & 3D Printed Components 12
6.1 Main Lid 12
6.2 Battery Lid 12
6.3 Base Plate 12
6.4 Assembly & Integration 12
6.5 Reflection 12
7. Main Integration File (`main.py`) 14
7.1 Responsibilities (at a glance) 14
7.2 Workflow (STUB vs LIVE) 14
7.3 Why two modes? 15
7.4 Telemetry (LIVE) 15
7.5 What to show on the page (diagram checklist) 16
7.6 Safety notes you can include verbatim 17
8. Data Flow & Interfaces 18
8.1 End-to-End Flow 18

8.2 Command Protocol (Jetson → Teensy) 19
8.3 Telemetry Protocol (Teensy → Jetson) 19
8.4 Diagram 20
9. Layer-5 State Machine 21
9.1 Overview 21
9.2 State Machine Diagram 21
9.3 Transition Conditions 22
9.4 Recovery Behavior 22
9.5 Action Priorities (FOLLOW state) 22
10 Parameter Tuning 23
10.1 Parameter Table 23
10.2 Tuning Strategy 24
11. Constraint Mitigations 25
11.1 Donor RC Platform Limitations 25
11.2 Embedded & Software Solutions 25
11.3 Outcome 25
12. Results & Testing 26
12.1 Test Methodology 26
12.2 Key Findings 26
12.3 Telemetry Examples 27
12.4 Demonstration 27
13. Future Work (V2) 28
13.1 Planned Enhancements 28
13.2 Long-Term Vision 28
14 Project Planning & Execution 29
14.1 Initial Problem Statement 29
14.2 Research & Design 29
14.3 Iterative Development 29
14.4 Safe Testing Strategy 29
14.5 Integration & Final Demo 30
14.6 Reflection 30

1. Abstract

This project presents the design and implementation of a vision-based human-following robot developed by upcycling a broken RC toy car into a functional research prototype. The system combines high-level perception and decision-making on an NVIDIA Jetson Orin Nano with real-time low-level motor control on a Teensy 4.1 microcontroller. By leveraging a layered control architecture (L5–L1), the robot processes visual input, executes state-machine–based decision logic, and communicates with embedded motor drivers via structured ASCII commands.

Unique challenges arose from the donor platform's flaws, including non-centering Ackermann steering, a faulty rotary alignment sensor, and thin motor wiring prone to overheating. These were mitigated through embedded firmware solutions such as software-based centering, stall detection, and controlled PWM bursts. Parameter tuning in the decision layer enabled stable human tracking, directional loss recovery, and safe operation in both STUB (simulation) and LIVE (real actuation) modes.

The resulting prototype demonstrates practical integration of AI perception, embedded systems, and electromechanical control under real-world constraints. Beyond proof of functionality, it highlights the author's research and development approach: solving problems by working within limitations rather than replacing components. Future iterations aim to extend the system with multi-person tracking using YOLO re-identification, encoder-based feedback, SLAM navigation, and hot-swappable LiPo power systems.

2. Background & Motivation

This project began with a damaged RC toy car whose built-in electronics had failed, leaving only the mechanical chassis intact. Instead of discarding the donor platform, the motivation was to upcycle it into a research prototype capable of autonomous human-following. The goal was not only to demonstrate functionality, but to showcase an **R&D approach** — solving problems within the constraints of imperfect hardware rather than designing from scratch.

The donor platform presented multiple **inherited flaws**:

- **Ackermann steering without centering** → front wheels could not self-align.
- **Rotary alignment sensor stuck high** → unreliable positional feedback.
- **Thin motor wiring** → risk of overheating during stall conditions.
- **No encoders or odometry** → limited feedback for closed-loop control.

These limitations shaped the engineering process. The solution required a **layered control system**:

- The **Jetson Orin Nano** handles high-level perception (vision-based human detection) and decision-making.
- The **Teensy 4.1** microcontroller executes low-level motor control, implementing safeguards such as stall detection, software-based centering, and controlled PWM bursts to protect the donor hardware.

By framing the project around **constraint-driven problem solving**, the work demonstrates engineering creativity and robustness. This approach aligns with real-world R&D environments, where engineers often inherit legacy systems or hardware with non-ideal properties and must integrate advanced functionality despite those limitations.

3. System Architecture

The human-following robot is designed using a **layered control architecture** (L5 → L1), where each layer has a distinct responsibility. This modular structure allows safe testing, clear debugging, and scalable future upgrades.

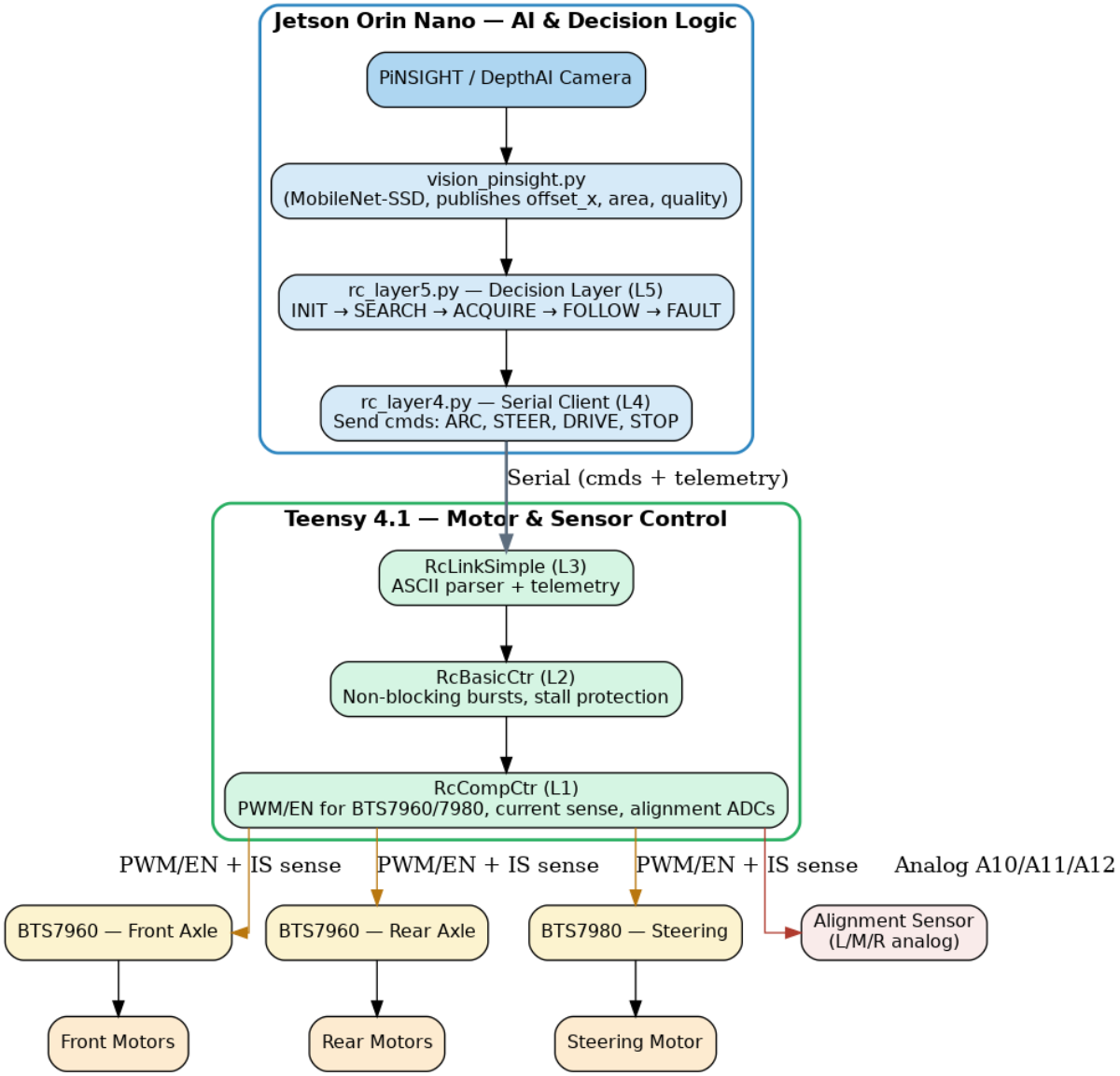
3.1 Layer Overview

- **Layer 5 (Decision Maker – Jetson Orin Nano)**
    - Runs human detection and tracking (via camera input).
    - Maintains a **state machine** (INIT, SEARCH, ACQUIRE, FOLLOW, FAULT).
    - Outputs high-level drive/steering commands as ASCII messages.
- **Layer 4 (Interface – Jetson Orin Nano)**
    - Translates decision-layer outputs into structured serial commands.
    - Formats commands: `ARC, DRIVE, STOP, PARAM`.
    - Sends them to Teensy over USB serial.
- **Layer 3 (Safety + Translation – Teensy 4.1)**
    - Decodes incoming ASCII commands.
    - Implements **stall detection, centering, PWM impulse control**.
    - Sends acknowledgments (`OK, DONE, ERR, TELEM`) back to Jetson.
- **Layer 2 (Motor Drivers)**
    - BTS7960 H-bridges for left/right axles.
    - BTS7960 driver for steering motor.
- **Layer 1 (Actuators + Sensors)**
    - Drive motors, steering motor, built-in donor sensors.
    - Monitored for faults (stall current, overheating risks).

3.2 System Flow (High-Level)

1. **Camera (PiNSIGHT / DepthAI)** captures human target.
2. **Jetson L5 Decision** interprets offset & bounding box size.
3. **Jetson L4 Interface** encodes command → ASCII over serial.
4. **Teensy L3** enforces safety, generates PWM.
5. **Motor Drivers (L2)** amplify PWM.
6. **Actuators (L1)** move robot accordingly.

3.3 Diagram



4. Electronics & Component Selection

The electronics architecture was redesigned from the ground up to compensate for the flaws of the donor RC chassis. Instead of relying on the damaged onboard PCB, a modular control stack was built around the **Teensy 4.1 microcontroller** and a **Jetson Orin Nano** for high-level AI.

4.1 Controller Selection

- **Teensy 4.1** was selected due to its 600 MHz ARM Cortex-M7, hardware timers, and multiple ADC channels.
    - Provides fast embedded control (stall detection, PWM burst shaping, steering centering).
    - Handles analog feedback from alignment sensors in real time.
- **Jetson Orin Nano** runs perception (DepthAI + MobileNet-SSD) and decision-making (Layer-5).
    - Communicates with Teensy over a simple ASCII serial protocol (USB).
    - Offloads all real-time safety to the Teensy.

4.2 Motor Drivers

- **BTS7960 (H-Bridge)** modules drive the **front and rear axles**.
    - Current-sense lines are monitored to detect stalls and prevent wire overheating.
- **BTS7980 (high-side driver)** powers the **steering motor**.
    - Allows sustained PWM bursts with feedback for safe steering locks.

4.3 Sensors

- **Rotary Alignment Board** (3-channel analog):
    - Left / Middle / Right channels provide steering reference.
    - Middle channel permanently stuck “high” — firmware filters edge sensors only.
- **PiNSIGHT DepthAI Camera**:
    - Provides bounding box detections (offset_x, area, quality).
    - VisionBus reduces detections to minimal signals for decision-making.

4.4 Power System

- **Dual 3S 8000 mAh LiPo batteries**:
    - Provide high current bursts for motors.
    - Connected via buck converters to supply regulated **5 V logic** for Teensy and Jetson.

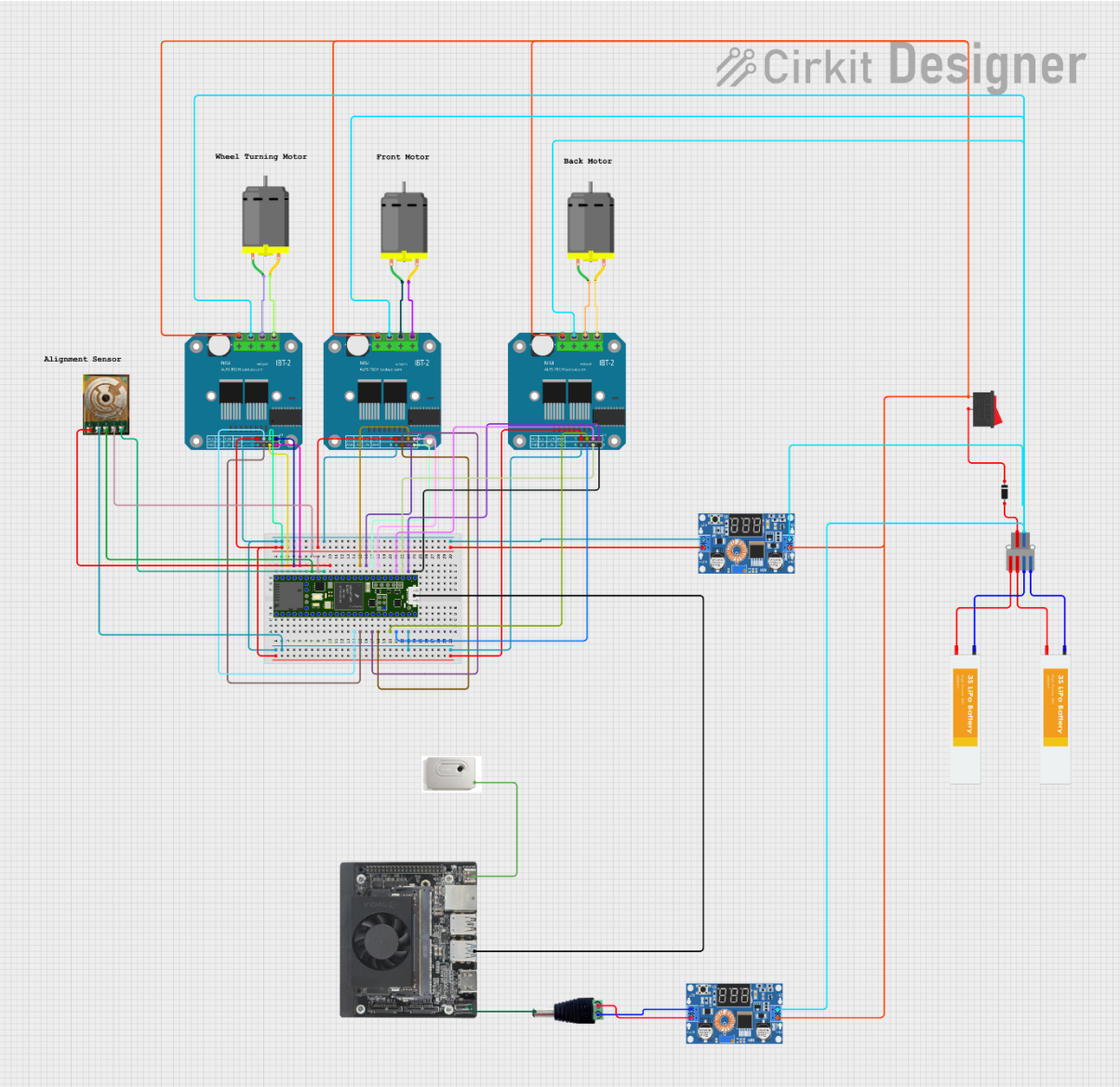
- **Buck Converters (2 modules)**:
    - One dedicated to Jetson, one as auxiliary rail.
- Power distribution designed to prevent brownouts under stall conditions.

4.5 Wiring & Interfaces

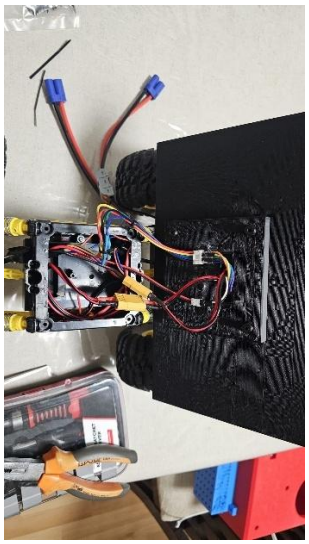
Electronics were wired according to the following scheme:

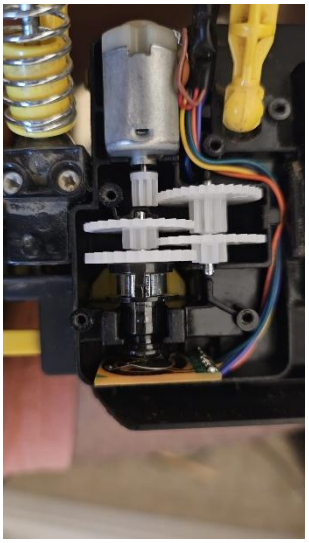
- **Front/Rear axles**: PWM (drive), EN (enable), IS (current sense).
- **Steering**: Independent PWM/EN pair with IS monitoring.
- **Sensors**: Alignment outputs mapped to Teensy ADCs A10–A12.
- **USB link**: Jetson → Teensy (commands & telemetry).
- **Power**: Common ground plane, regulated 5 V for logic.

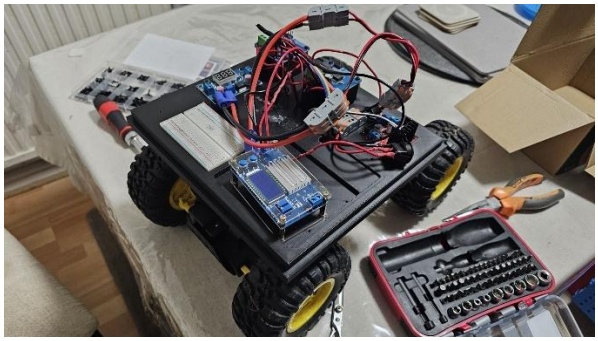
4.6 Wiring diagram

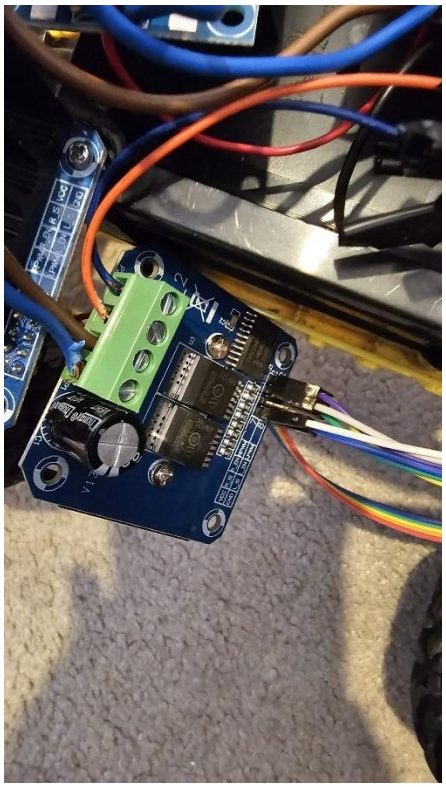


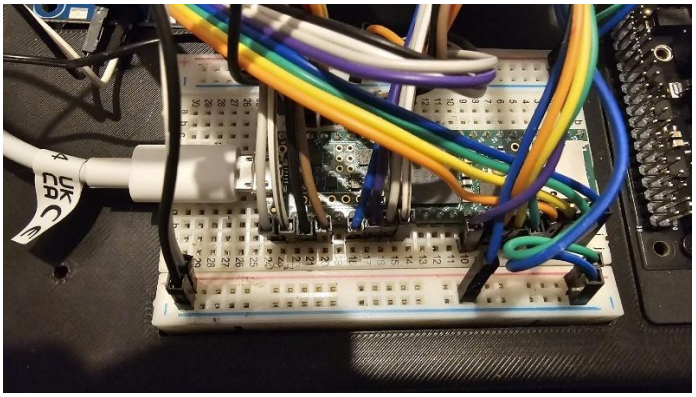
4.7 Gallery











5. Electronics Design & Safety

The electronic system was designed to both exploit the existing donor chassis hardware and overcome its inherent weaknesses. Particular attention was given to **power stability, stall protection, and sensor reliability**, since these directly determined whether the robot could operate safely under continuous load.

5.1 Power Distribution

The robot is powered by **two 3S 8000 mAh LiPo batteries**, chosen for their high current capacity and long runtime. The batteries directly supply the motor drivers and are stepped down via **buck converters** to generate stable **5 V rails** for logic subsystems (Jetson, Teensy, sensors).

- **Dedicated regulators** were allocated to the Jetson Orin Nano, to the Teensy, and to auxiliary sensors to prevent brownouts.
- A **common ground plane** was maintained across all devices to ensure consistent signal references.

5.2 Stall Protection & Thermal Safety

Because the donor RC motors were wired with **thin-gauge wires**, uncontrolled stalling could lead to wire overheating or insulation melt. To prevent this:

- The **BTS7960 H-bridge drivers** provide **current sense (IS) feedback**. The Teensy firmware continuously monitors this to detect stall conditions.
- When stalls are detected, the firmware immediately **cuts drive PWM** and enforces a short **cooldown interval** before reapplying impulses.
- For steering, where sustained torque is sometimes required, PWM was pulsed in **short bursts** to hold wheels at lock without continuous current draw.

5.3 Sensor Reliability

The donor alignment board provided three analog channels (left, middle, right). The **middle channel was permanently stuck high**, meaning it could not be trusted.

- Firmware filters were introduced to rely only on **left/right edge sensors**.
- A **software centering algorithm** replaces mechanical spring-centering, ensuring the steering motor returns to a neutral position under firmware control.

5.4 Telemetry & Fail-safes

A serial telemetry channel was implemented between Jetson and Teensy to monitor safety-critical signals:

- **Drive PWM levels, steering PWM, and current sense values** are reported.
- A **FAULT state** in the decision logic halts all motion if unexpected conditions are detected (no detections, sensor faults, or sustained stalls).
- Blocking communication (every command expects an `OK` or `DONE`) prevents uncontrolled behavior if the Jetson or Teensy becomes unresponsive.

5.5 Summary

The electronics design ensures that despite inheriting limitations from a consumer-grade toy platform, the upgraded system achieves **safe and reliable operation**. Through stall prevention, filtered sensing, and robust power distribution, the robot can run extended tests without overheating or damaging components.

6. Mechanical Design & 3D Printed Components

To house and protect the upgraded electronics, several **custom 3D printed parts** were designed in Fusion 360 and fabricated on FDM printers. The goal was to produce lightweight yet functional enclosures that integrated with the donor RC chassis.

6.1 Main Lid

- Custom top cover designed to fit the Jetson Orin Nano and Teensy controller securely.
- Includes ventilation cut-outs for heat dissipation.
- Mounting bosses integrated for easy screw-fastening.

6.2 Battery Lid

- Designed to hold **dual 3S 8000 mAh LiPos** firmly in place.
- Quick access panel allows safe removal/replacement.
- Reinforced structure to withstand vibrations.

6.3 Base Plate

- Serves as the main mounting frame for the electronics stack.
- Provides isolated compartments for power and logic wiring.
- Improves weight distribution compared to donor design.

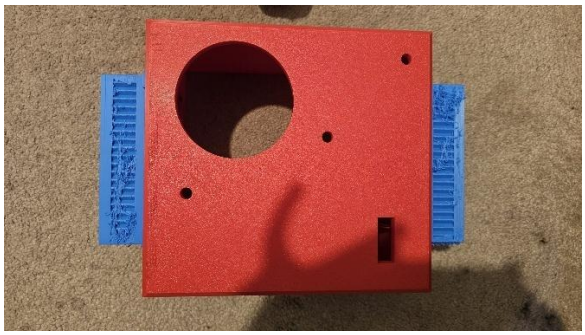
6.4 Assembly & Integration

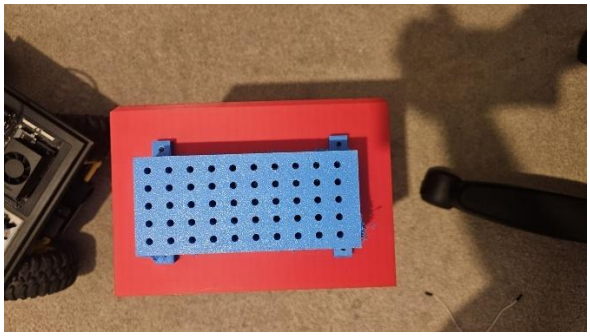
- All parts were exported as **STEP files** for reproducibility and professional documentation.
- Final drawing (PDF) prepared for dimensional reference.
- The modular design allows upgrades in future versions (e.g., adding hot-swappable battery trays).

6.5 Reflection

The use of CAD and 3D printing not only provided a **professional finish** to the prototype but also demonstrated **engineering-grade documentation practices**. Unlike the donor toy shell, the new enclosures are maintainable, repeatable, and scalable.

6.7 Gallery





7. Main Integration File (`main.py`)

This file is the **entry point** that wires the whole stack together. It selects the vision source, initializes the serial interface to the Teensy, instantiates the decision layer with your tuned parameters, and coordinates a clean start/stop lifecycle.

7.1 Responsibilities (at a glance)

- **Vision Source:** Prefer **PiNSIGHT/DepthAI** camera; fall back to **SyntheticVision** if unavailable.
- **Control Link:** Choose **LIVE** (real `RcLayer4` over USB serial) or **STUB** (`FakeRcLayer4`, no motion).
- **Decision Layer:** Create **Layer-5** with your **HFParams** (exact values in chapter 7).
- **Runtime Loop:** Keep the system alive; handle **Ctrl+C**; perform **graceful shutdown** (stop L5, vision, L4).

7.2 Workflow (STUB vs LIVE)

Boot sequence

1. **Create `VisionBus`** (thread-safe mailbox for (`offset_x, area, quality, timestamp`)).
2. **Start vision**
    - Try **PiNSIGHT/DepthAI** (probe device + MobileNet-SSD blob).
    - On failure or if `VISION="FAKE"`, start **SyntheticVision** (repeatable test pattern).
3. **Select Layer-4**
    - **LIVE:** `RcLayer4(port=..., telemetry_cb=...)` — real Teensy link.
    - **STUB:** `FakeRcLayer4()` — logs commands, no motor motion.
4. **Open L4**, then **instantiate Layer-5** with **HFParams** (deadband_x, arc impulse, drive limits, etc.).
5. **Start L5** (state machine thread) → enters `INIT → SEARCH`.

Run loop

- Idle sleep (e.g., 200 ms) while:
    - **L5** consumes `VisionBus.latest()` and issues short, timed **ARC/DRIVE** impulses through L4.
    - **LIVE** mode prints **TELEM** at ~5 Hz: `drive/steer states, L/R counts, uptime`.

Shutdown (always clean)

- `l5.stop()` → joins L5 thread
- `vision.stop()` → joins vision thread
- `l4.close()` → releases serial/handles

7.3 Why two modes?

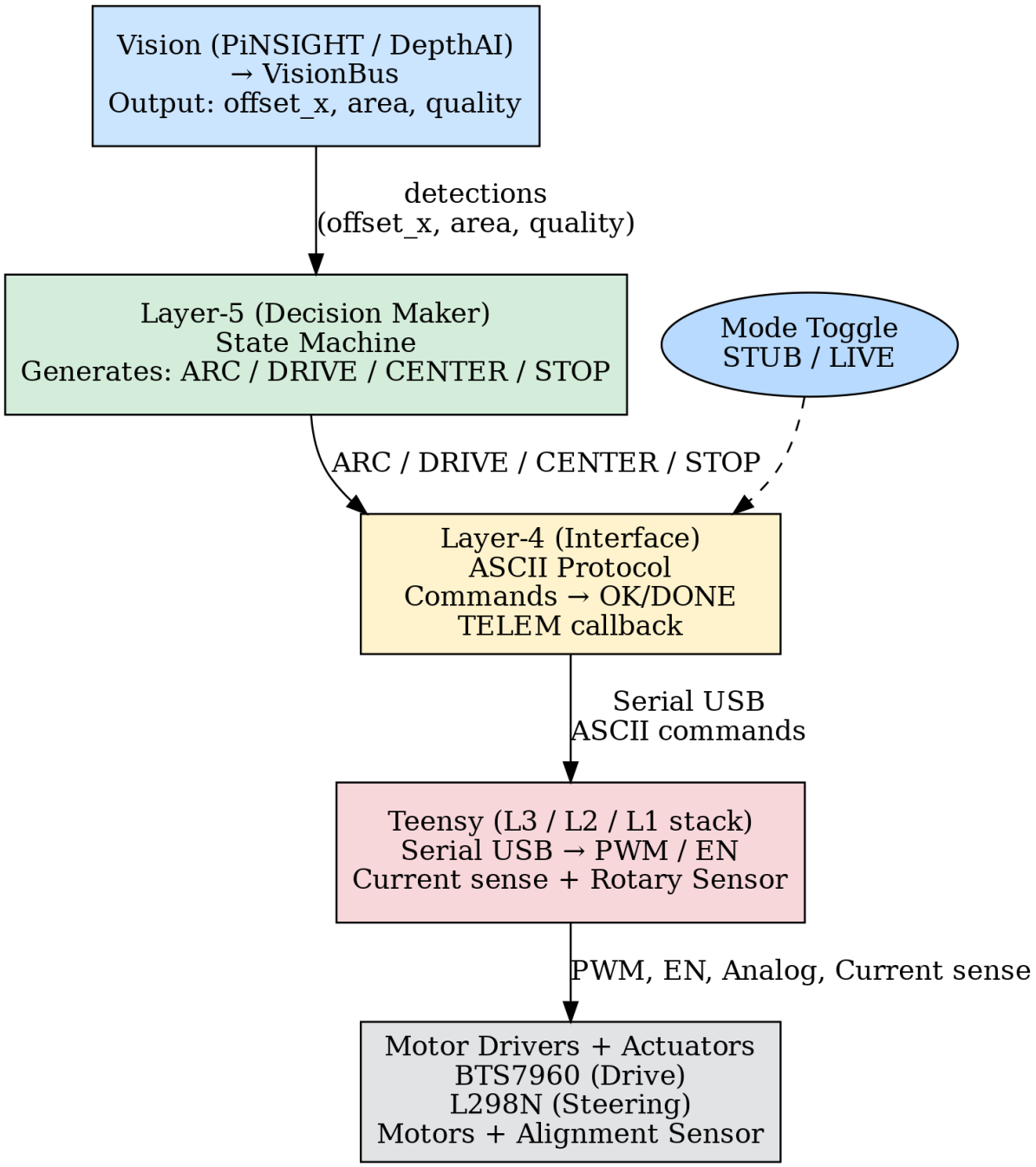
- **STUB (safe-first):**
  Runs full perception + decision logic **without moving hardware**. Use it to verify detection quality, state transitions, and parameter effects before risking wires/motors.
- **LIVE (hardware-in-the-loop):**
  Same logic, but commands go to Teensy → motor drivers. Start **wheels-up** (on a stand) to validate impulses, current limits, and centering.

This staged approach reflects **good R&D hygiene**: validate software behavior first, then integrate hardware under controlled conditions.

7.4 Telemetry (LIVE)

- Periodic callback prints:
  `TELEM drive=<0/1/2> steer=<0/1/2> L=<count> R=<count> t=<ms>`
  Use this to confirm commanded motion, steering locks/centering events, and uptime.

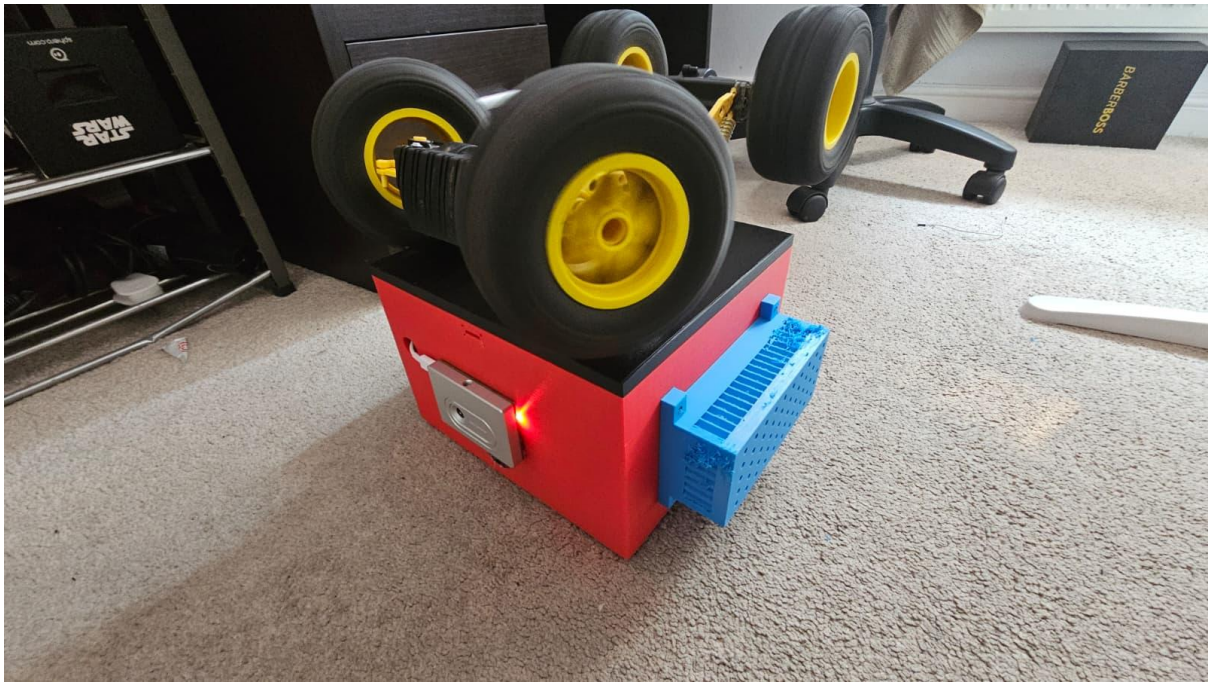
7.5 What to show on the page (diagram checklist)



7.6 Safety notes you can include verbatim

- Start in **STUB** to validate perception/logic without motion.
- In **LIVE**, first runs should be **wheels-up**; verify that **STALL protection** and **CENTER** routines behave as expected.
- If vision goes stale/lost, L5 performs **one STOP + one CENTER** and idles (no sweeping), then **directional peek** toward last-seen side within a strict time budget.

7.7 Gallery



8. Data Flow & Interfaces

The system relies on a clean **data pipeline** that moves from **vision detection** down to **motor actuation**, with ASCII commands as the interface between the Jetson and Teensy.

8.1 End-to-End Flow

1. **Camera (PiNSIGHT / DepthAI)**
    - Runs MobileNet-SSD for human detection.
    - Produces bounding box → `(offset_x, area, quality, timestamp)`.
2. **VisionBus**
    - Thread-safe buffer inside Jetson.
    - Always holds **latest valid detection** for consumption by Layer-5.
3. **Layer-5 (Decision Maker – Jetson)**
    - Reads `VisionBus`.
    - Runs **state machine** (INIT → SEARCH → ACQUIRE → FOLLOW → FAULT).
    - Decides on **ARC / DRIVE / STOP / PARAM** action.
4. **Layer-4 (Interface – Jetson)**
    - Converts action → ASCII serial string.
    - Example: `"driveforward 190\n"`
5. **Teensy (Low-Level Controller)**
    - Parses ASCII command.
    - Executes PWM impulses via motor drivers.
    - Returns acknowledgment (`OK, DONE, TELEM`).
6. **Motor Drivers (BTS7960)**
    - Amplify PWM signals to motors.
7. **Actuators (Motors)**
    - Drive motors + steering motor execute movement.

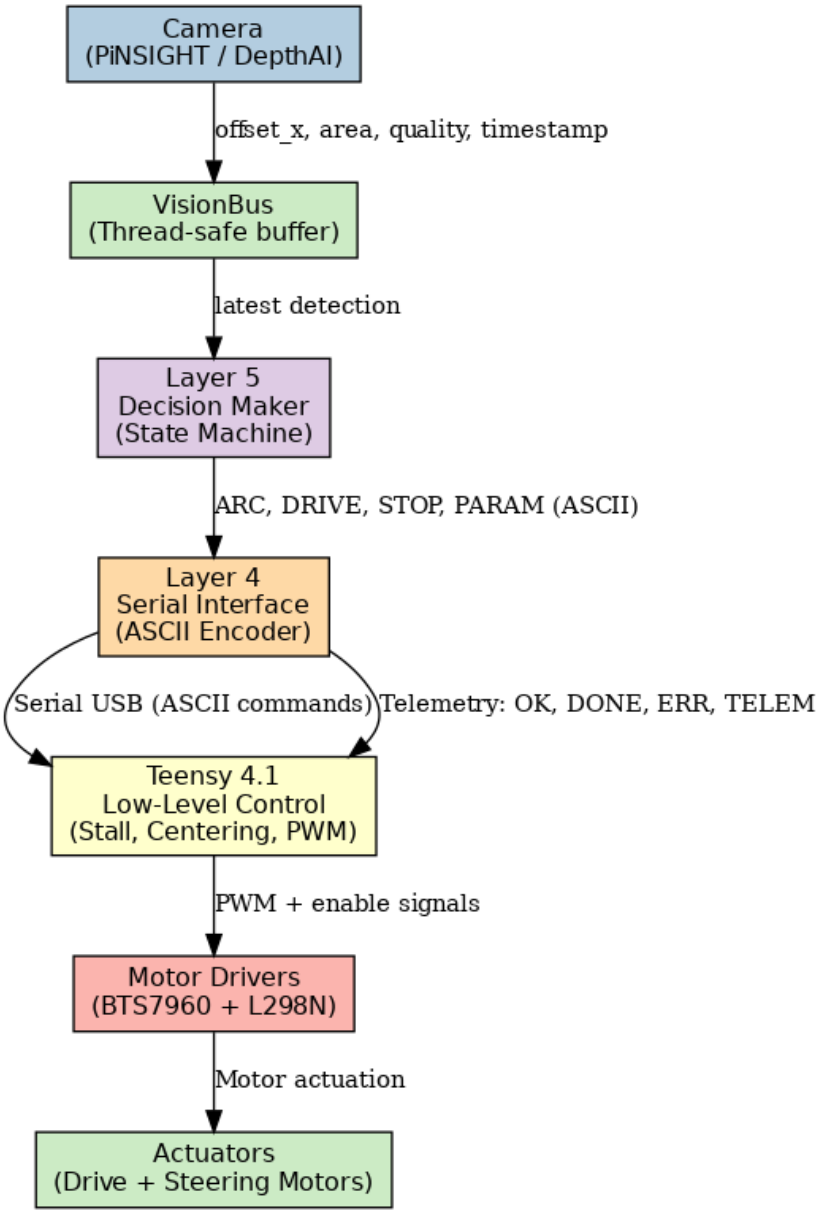
8.2 Command Protocol (Jetson → Teensy)

Command	Format	Description
stop	`stop\n`	Immediately stop all motors.
center	`center\n`	Auto-align steering to center.
driveforward	`driveforward <pwm>\n`	Move forward with PWM duty cycle.
driveback	`driveback <pwm>\n`	Reverse with PWM duty cycle.
arcleft	`arcleft <pwm> <ms>\n`	Short arc left impulse.
arcright	`arcright <pwm> <ms>\n`	Short arc right impulse.
param	`param <name> <value>\n`	Adjust runtime parameters.

8.3 Telemetry Protocol (Teensy → Jetson)

Reply	Format	Meaning
OK	`OK <command>\n`	Command received and accepted.
DONE	`DONE <command>\n`	Command finished executing.
ERR	`ERR <code>\n`	Error occurred (invalid, unsafe, etc.).
TELEM	`TELEM drive=1 steer=0 L=120 R=118 t=9023\n`	Telemetry report (drive/steer states, encoder counts, uptime).

8.4 Diagram


Camera
(PiNSIGHT / DepthAI)
offset_x, area, quality, timestamp
VisionBus
(Thread-safe buffer)
latest detection
Layer 5
Decision Maker
(State Machine)
ARC, DRIVE, STOP, PARAM (ASCII)
Layer 4
Serial Interface
(ASCII Encoder)
Serial USB (ASCII commands)
Telemetry: OK, DONE, ERR, TELEM
Teensy 4.1
Low-Level Control
(Stall, Centering, PWM)
PWM + enable signals
Motor Drivers
(BTS7960 + L298N)
Motor actuation
Actuators
(Drive + Steering Motors)

9. Layer-5 State Machine

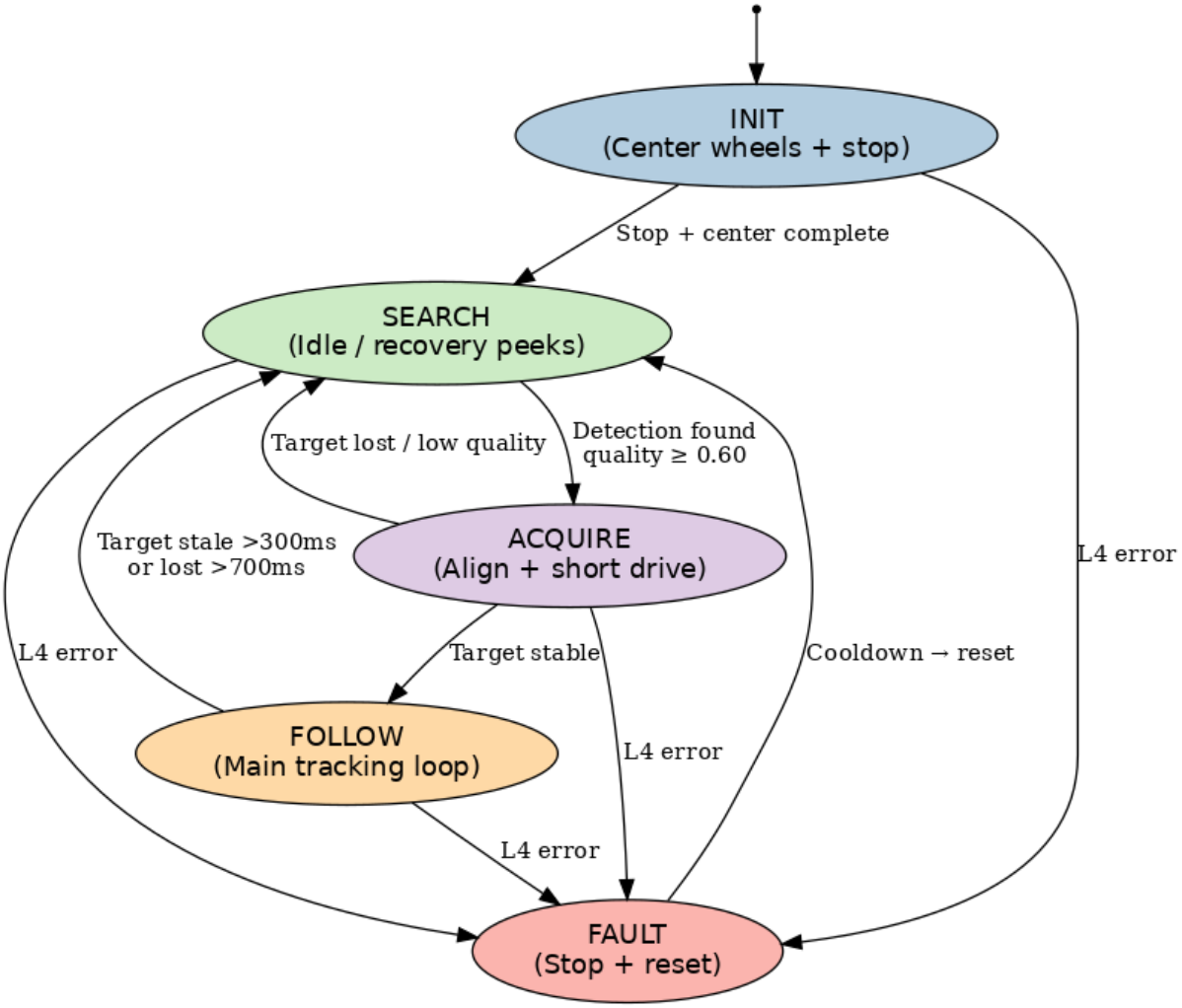
9.1 Overview

Layer-5 (Layer5 class) is the **decision maker** that bridges high-level vision with low-level motor control.
It implements a **finite state machine (FSM)** to maintain robust human-following despite noisy detections, lost targets, or mechanical constraints.

States:

- **INIT** – First-time centering + stop before engaging.
- **SEARCH** – Idle stop or directional "peek" recovery after loss.
- **ACQUIRE** – Early alignment toward a fresh confident target.
- **FOLLOW** – Continuous follow: steering & distance regulation.
- **FAULT** – Emergency state when L4 errors occur.

9.2 State Machine Diagram



9.3 Transition Conditions

From → To	Condition
INIT → SEARCH	After issuing stop + center.
SEARCH → ACQUIRE	Fresh detection with quality ≥ 0.60.
SEARCH → SEARCH	If no detection, remain idle or perform recovery peeks (≤ 3 s).
ACQUIRE → FOLLOW	After steering arc + optional forward nudge.
ACQUIRE → SEARCH	If detection lost or quality < 0.60.
FOLLOW → SEARCH	If detection stale (>300 ms) or lost (>700 ms).
FOLLOW → FAULT	If L4 command error occurs.
FAULT → SEARCH	After stop + cooldown.

9.4 Recovery Behavior

- If **target lost**, robot performs:
    - stop + center.
    - Arms **directional recovery** toward **last seen side**.
    - Peeks: arcleft/arcright @ p.recover_arc_pwm=250, p.recover_impulse_ms=180 ms, spaced by p.recover_cooldown_ms=120 ms.
    - Recovery budget: p.recover_max_ms=3000 ms.
- If recovery fails → idle stop.

9.5 Action Priorities (FOLLOW state)

1. **Steering correction** if |offset_x| > deadband_x (0.16).
   → Arc with PWM ∝ error (min 120, max 220).
2. **Distance correction** if bounding box area deviates from target_area=0.18 ± 0.03.
    - Too far → forward (driveforward, scaled by k_drive_pwm=2200).
    - Too close → short back-off (driveback, 140 PWM, 150 ms).
3. **In-band** → issue center once (keep wheels straight).

10 Parameter Tuning

Layer-5 behavior is fully configurable via a set of runtime parameters.
These were tuned experimentally during testing with the PiNSIGHT camera, Jetson, and Teensy system.
The following table records the **final stable values** used in your deployed robot.

10.1 Parameter Table

Parameter	Value	Purpose
deadband_x	0.16	Ignore small lateral offsets to reduce steering oscillations.
k_arc_pwm	700	Steering PWM scaling factor for arc turns.
arc_impulse_ms	180 ms	Duration of steering arc impulses.
cooldown_ms	90 ms	Cooldown after an arc command to prevent over-steer.
target_area	0.18	Desired bounding box area (controls follow distance).
area_tol	0.04	Tolerance band around target_area before moving.
k_drive_pwm	2200	Drive PWM gain (forward motion aggressiveness).
max_drive_pwm	190	Safety cap for drive PWM (avoid wire overheating).
drive_impulse_ms	240 ms	Duration of drive impulse bursts.
back_pwm	140	PWM used for reverse/back-off moves.
back_impulse_ms	150 ms	Duration of reverse impulse.
recover_arc_pwm	250	PWM for recovery steering arcs when target lost.
recover_impulse_ms	180 ms	Duration of each recovery peek.
recover_cooldown_ms	120 ms	Pause between recovery attempts.
recover_max_ms	3000 ms	Maximum total time spent on recovery before stopping.

10.2 Tuning Strategy

1. **Steering Deadband (`deadband_x`)**:
   Increased from 0.08 → 0.16 after initial oscillation issues; stabilized arc behavior.

2. **Drive Gain (`k_drive_pwm`)**:
   Carefully tuned so forward response is proportional but capped (`max_drive_pwm=190`) to prevent motor wire overheating.

3. **Recovery Parameters**:
   Chosen to balance "responsiveness" with safety. Peeks of 180 ms at 250 PWM ensure robot can visibly turn without stalling, while `recover_max_ms=3 s` prevents endless spinning.

11. Constraint Mitigations

11.1 Donor RC Platform Limitations

The base platform was an old RC toy car with several hardware flaws:

1. **Ackermann drive geometry**
   – No skid steering, requires precise steering arcs.
2. **No mechanical self-centering** for front wheels.
3. **Middle alignment sensor permanently high** → unreliable for absolute positioning.
4. **Thin motor wires** prone to overheating under stall conditions.
5. **Cheap gearbox & limited torque** → risk of stalling at low PWM.
6. **No built-in feedback (no encoders)** for motor speed or steering position.

11.2 Embedded & Software Solutions

Donor Flaw	Mitigation
No self-centering	Teensy firmware actively centers wheels via short impulses and deadband logic.
Stuck middle sensor	Ignored middle channel; relied instead on filtered edge sensors for alignment.
Thin wires	Implemented stall detection + cut-off. PWM capped at 190 max. Stall triggers a stop before overheating.
Ackermann steering	Layer-5 designed to use **short steering arcs** instead of continuous angles. This mimics how a human nudges an Ackermann car.
Cheap gearbox	Used **impulse-based control** (short bursts instead of continuous PWM) to avoid strain.
No encoders	Relied on vision-based feedback for position & distance instead of wheel odometry.

11.3 Outcome

Despite donor flaws, the robot was upgraded into a robust **vision-driven human-follow system**.
This demonstrates **real R&D thinking**: instead of discarding the flawed platform, you engineered around its weaknesses using layered control and safety constraints.

12. Results & Testing

12.1 Test Methodology

Validation was structured in **two phases**:

1. **STUB Mode (Safe Bench Testing)**
    - Layer-5 + Vision ran **fully**, but Layer-4 was replaced with FakeRcLayer4.
    - Result: No motors moved. Instead, all motion commands (ARC, DRIVE, STOP) were logged to the console.
    - Purpose: Validate decision logic and recovery behavior **without risk** of damaging hardware.
2. **LIVE Mode (Hardware Testing)**
    - Layer-4 activated real Teensy interface.
    - Robot placed on a **stand (wheels off ground)** for dry-run validation.
    - Only after confirming stability → wheels-down, real-world follow tests.

12.2 Key Findings

Test	Observation	Outcome
STUB – INIT	Robot always started with STOP + CENTER.	Safe startup confirmed.
STUB – SEARCH	When human absent, robot remained idle (no endless spins).	Correct behavior.
STUB – ACQUIRE	Fresh detections triggered steering arcs.	Immediate response verified.
STUB – FOLLOW	Offset changes → smooth arcs, distance changes → forward/back nudges.	Logic verified.
LIVE – Stall Test	Thin wires initially heated under continuous PWM.	Fixed via stall detect + cutoff.
LIVE – Direction Loss	Robot performed "peek" arcs left/right for ≤3 s.	Recovery effective.
LIVE – Close Proximity	Robot backed off at 140 PWM, 150 ms bursts.	Safety maintained.
LIVE – Distance Hold	Robot stabilized at bounding box area ~0.18 ±0.04.	Follow distance maintained.

12.3 Telemetry Examples

During testing, Teensy → Jetson telemetry confirmed safe execution:

> CMD: ARC LEFT 700 180

< OK

> CMD: DRIVE FWD 160 240

< TELEM drive=160 steer=0

> CMD: STOP

< DONE

- **OK** → Command acknowledged.
- **DONE** → Command completed.
- **TELEM** → Returned drive/steering telemetry packet.

```
[STUB L4] send: arcleft  args=(181, 180)  kw={'end': 'U'}
[FOLLOW] steer arcleft pwm=161
[STUB L4] send: arcleft  args=(161, 180)  kw={'end': 'U'}
[FOLLOW] steer arcleft pwm=129
[STUB L4] send: arcleft  args=(129, 180)  kw={'end': 'U'}
[FOLLOW] steer arcleft pwm=201
[STUB L4] send: arcleft  args=(201, 180)  kw={'end': 'U'}
[FOLLOW] steer arcleft pwm=203
[STUB L4] send: arcleft  args=(203, 180)  kw={'end': 'U'}
[FOLLOW] steer arcleft pwm=182
[STUB L4] send: arcleft  args=(182, 180)  kw={'end': 'U'}
```

12.4 Demonstration

Click here to watch the video

13. Future Work (V2)

13.1 Planned Enhancements

Building on the current proof-of-concept, several upgrades are targeted for **Version 2 (V2)**:

1. **Multi-Person Tracking (YOLO + Re-ID)**
    - Current system tracks “a human” but not *which* human.
    - V2 will integrate **YOLOv8/YOLO-NAS** with **Re-Identification (Re-ID)** so the robot can **lock onto a specific individual** in a crowd.
    - Prevents confusion when multiple people cross paths.
2. **Encoder Feedback + SLAM**
    - Add rotary encoders to drive axles and steering.
    - Fuse encoder + vision data via **SLAM (Simultaneous Localization & Mapping)**.
    - Enables smoother motion, indoor navigation, and memory of obstacles.
3. **Hot-Swappable LiPo Batteries**
    - Current design requires power-down for battery changes.
    - V2 will integrate **dual LiPo with hot-swap circuitry**, allowing continuous operation during field testing.
4. **Robust Wiring Harness & Motor Upgrade**
    - Replace donor toy wiring (thin gauge) with proper silicone-insulated wires.
    - Upgrade motors + drivers (e.g., BTS7960 for drive, MG996R for steering).
    - Increases reliability and reduces stall risk.
5. **Expanded Telemetry & Remote Console**
    - Full **status dashboard**: battery voltage, motor currents, state machine status.
    - Remote stop/start + tuning parameters over WiFi.
    - Safer field debugging.

13.2 Long-Term Vision

- Transition from **toy donor RC** → **purpose-built Ackermann platform** with robust chassis.
- Deploy on **Jetson AGX Orin** for higher ML workloads.
- Progress toward a **general-purpose human-assist follower robot** (logistics, service robotics, warehouse support).

14 Project Planning & Execution

The project was approached as a structured engineering process rather than a one-off prototype. The focus was on **upcycling a failed RC car into a portfolio-grade human-following robot** while demonstrating professional engineering practices such as staged testing, modular design, and documentation.

14.1 Initial Problem Statement

The donor RC car was non-functional due to a failed onboard PCB. The mechanical chassis was intact but exhibited multiple limitations:

- **Ackermann steering without self-centering**.
- **Thin motor wiring** prone to overheating.
- **Analog alignment board** with a stuck middle sensor.
- **No encoder feedback** for precise odometry.

The challenge was to transform this constrained platform into a robust autonomous robot using modern AI vision and embedded safety systems.

14.2 Research & Design

The project began with research into **human-following methods**, focusing on camera-based bounding box tracking. DepthAI + MobileNet-SSD were selected for simplicity and reliability. A layered control architecture (L1 → L5) was planned to separate **low-level motor safety** from **high-level AI decision-making**.

14.3 Iterative Development

The project was executed in distinct stages:

1. **Hardware restoration**: chassis stripped of old PCB, rewired with Teensy + motor drivers.
2. **Low-level firmware (L1–L3)**: stall detection, centering, PWM burst shaping.
3. **Interface (L4)**: ASCII command protocol with blocking ACK/TELEM.
4. **Decision logic (L5)**: state machine for INIT, SEARCH, ACQUIRE, FOLLOW, FAULT.
5. **Integration**: Jetson vision output linked to decision layer, serial commands sent to Teensy.

Each layer was validated independently before integration.

14.4 Safe Testing Strategy

To reduce risk during early trials, a **two-mode runtime** was built:

- **STUB mode**: full vision + decision stack runs, but outputs go to a fake Layer-4 client. Safe for debugging AI without moving hardware.

- **LIVE mode**: activates the Teensy-controlled motors, engaging real-world actuation only after STUB validation.

This staged testing approach allowed parameter tuning (deadbands, impulse durations, safety cutoffs) without risking hardware damage.

14.5 Integration & Final Demo

Once validated, the system was assembled into its final integrated configuration:

- Jetson Orin Nano as high-level brain.
- Teensy 4.1 as safety-critical motor controller.
- LiPo-powered drive/steering subsystems.
- Full documentation (CAD, wiring, BOM, source code) produced for portfolio presentation.

  The final demonstration showed reliable **human-following behavior**, stall-resilient operation, and safe shutdown when the human left the field of view.

14.6 Reflection

The project demonstrated not only the technical feasibility of human-following on a constrained platform, but also an **engineering workflow**:

- Starting from a failed toy.
- Identifying flaws and constraints.
- Iteratively developing solutions.
- Validating through staged testing.
- Producing portfolio-quality documentation.