EEEE1039 Applied Electrical and Electronic Engineering: Construction Project

2024/25 Session

Integrated Wireless Control and Robust Line Following for an Embedded Robotic Vehicle

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

UNIVERSITY OF NOTTINGHAM NINGBO CHINA

Name : Hongziheng Wang

Student ID : 20717880

Abstract

An Arduino-based mobile robotic platform was developed using a staged integration approach to achieve reliable operation through **interaction, sensing, and safe actuation**. Each functional block was implemented and verified with observable evidence before full system integration, resulting in a reproducible workflow suitable for both remote operation and closed-loop autonomy.

In **Session 3**, the focus was on establishing robust wireless control, measurement, and motor-drive safety foundations and then demonstrating them in integrated challenge tasks. Two short-range links were implemented and validated: an **nRF24L01+** packet-based transceiver channel and an **HC-06** Bluetooth UART bridge, both confirmed by repeatable serial behaviour and bidirectional testing. Motion sensing was introduced by interfacing an **MPU-6050** IMU via I²C and verifying physically consistent static tilt trends to support motion-aware logic. To ensure safe actuation, a **NAND-only unipolar PWM routing** stage was realised for the **L293D** driver; duty-cycle tests verified correct direction selection while preventing simultaneous PWM on both direction inputs. Building on these blocks, the platform completed the **remote/automatic control demonstrations**: **Task A** (autonomous run with ramp event handling and rotation, supported by runtime status feedback), **Task B1** (nRF24 joystick-based remote driving with stable command reception and safety handling), and **Task B3** (smartphone Bluetooth command control via HC-06 with defined command-action behaviour and watchdog-style stopping).

In **Session 4**, the validated platform was extended to closed-loop tape tracking, progressing from a simple baseline to a higher-resolution solution. **TCRT5000** reflective sensors were characterised over black/white surfaces to quantify discrimination margin versus distance and justify a practical mounting height for reliable tape detection. Closed-loop line following was then demonstrated using a **two-sensor PID** controller and subsequently upgraded to an **8-sensor array** with per-channel calibration, normalised 0–1000 response, weighted-average position estimation, and explicit line-loss handling. The final integrated controller combined the position estimate with bounded PID steering and practical tuning support, delivering stable tracking through straights and corners with defined safety behaviour under signal loss.

Chapter 1 Introduction

1.1 Project motivation and overall aim

A mobile robotic system must integrate **interaction**, **measurement**, and **safe actuation** to behave reliably on a real tape track. In this work, an Arduino-based vehicle platform was developed using a staged engineering workflow where each functional block was first implemented and verified before full integration. The overall aim was to achieve a system that supports **real-time wireless user operation** while also demonstrating **stable autonomous tape tracking** through closed-loop control.

1.2 Verified functional modules and integration roadmap

The project is organised into a set of verified modules that collectively enable remote control and line-following autonomy. Wireless interaction was implemented using an **nRF24L01+** packet link and an **HC-06 Bluetooth UART** bridge, enabling command transmission and observable verification via serial behaviour. Motion sensing was introduced by interfacing an **MPU-6050 IMU** via I²C and validating physically consistent static tilt trends to support subsequent event logic. For safe actuation, a **NAND-only unipolar PWM routing** stage was designed for the **L293D** so PWM is never applied to both direction inputs simultaneously. A handheld **HMI (joystick + LCD)** was assembled to provide stable input and runtime status feedback, improving test efficiency and integration traceability. Optical sensing was then characterised using a **TCRT5000** module to quantify black/white separability versus distance and justify a practical mounting height for tape detection. Building on these validated blocks, line following was implemented first using a **two-sensor PID** controller and then extended to an **8-sensor array** with calibration-aware normalisation and weighted-average position estimation, forming the basis of the final closed-loop tracking system with tuning and safety handling.

Chapter 2 Methods

2.1 Wireless communication (nRF24L01+) and Bluetooth link (HC-06)

Objective. This section aimed to implement and verify two short-range wireless links for the robot platform: (1) a 2.4 GHz nRF24L01and transceiver link between two Arduino Nano boards (TX/RX) for reliable packet-based communication, and (2) an HC-06 Bluetooth serial link for smartphone-to-Nano UART communication. Both links were validated using Serial Monitor outputs and bidirectional data tests. Representative full sketches and wiring photographs are provided in **Appendix A** (Arduino sketches) and **Appendix B** (Table & Photo).

2.1.1 nRF24L01+ hardware configuration

The nRF24L01+ module was powered from the Nano's **3.3 V** rail (**5 V was avoided to prevent damage**). The SPI interface used Nano's hardware SPI pins (SCK/MOSI/MISO), while **CE** and **CSN** were assigned as digital GPIO control pins (**Appendix B.3**).

2.1.2 nRF24L01+ software configuration (TX/RX sketches)

The RF24 library was used to configure the radio. Separate sketches were uploaded to two Nano boards(**Appendix B.1 & B.2**):

1. **Transmitter (TX)**: periodically sent a fixed string payload.
2. **Receiver (RX)**: continuously listened and printed received payloads to Serial Monitor.

Key radio parameters were configured consistently on both sides:

1. **RF channel**: set to **93** (to avoid interference; channels are mapped around 2.4 GHz).
2. **Power amplifier level**: RF24_PA_MAX to improve link margin.
3. **Data rate**: RF24_2MBPS for high throughput during testing.
4. **Address and pipe**: writing / reading pipe were configured using the same address string.

Validation method: On the receiver Nano, Serial Monitor was opened at **9600 baud**. Successful wireless operation was confirmed when the receiver printed the transmitted message repetitively and continuously.

2.1.3 HC-06 hardware configuration (UART + level shifting)

The HC-06 module was powered at **5 V**, and communication used a software serial port on the Nano. The implementation used **SoftwareSerial on pins D10 (RX)** and **D11 (TX)** as shown in the uploaded code. Therefore, the wiring was:

1. HC-06 **TXD → Nano D10 (RX)** (3.3 V logic into Nano input was acceptable).
2. Nano **D11 (TX) → HC-06 RXD** through a **voltage divider**, because the Nano output is 5 V but HC-06 RXD expects ~3.3 V logic.

A resistor divider was applied (**1.8 kΩ + 3.3 kΩ**) to reduce 5 V to ~3.3 V at the HC-06 RXD node. A simplified schematic of the HC-06 voltage divider is provided in Fig 1. (regard the nRF8001 as nRF24L01 considering their common pins)

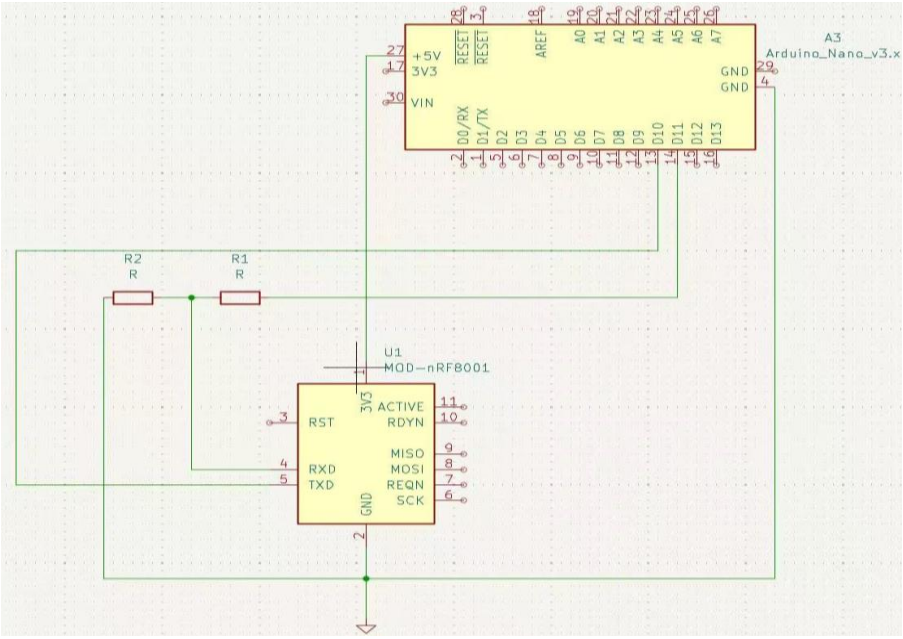


Figure 1. Voltage divider for HC-06 RXD level shifting

2.1.4 HC-06 software configuration and AT-command renaming

Two software steps were verified:

1. **Basic Bluetooth serial forwarding test:** A sketch initialized Serial for PC monitoring and SoftwareSerial at **9600 baud** for HC-06, then echoed received bytes to confirm the wireless UART link.

2. **Renaming the HC-06 module via AT commands:** The module was put into AT-command mode.

The command sequence followed the pattern [7]:

1. AT → expected response OK

2. AT+NAME=<custom name> → expected response similar to OKsetname.

This ensured the Bluetooth device could be uniquely identified during phone pairing.

Validation method: A smartphone Bluetooth terminal application was used. Successful operation was confirmed when text sent from the phone appeared on the Arduino Serial Monitor and vice versa.

2.2 MPU-6050 motion sensing (I²C)

Objective. To interface an **MPU-6050** inertial measurement unit (3-axis accelerometer + 3-axis gyroscope) with an Arduino Nano via the I²C bus, stream raw motion data reliably, and validate the static tilt estimation model by comparing measured acceleration components with the expected gravity projection relationship.

2.2.1 Hardware and I²C wiring.

An MPU-6050 module was connected to the Arduino Nano using the I²C interface. The wiring and configuration were:

- **Power:** VCC → 5 V, GND → GND (common ground required).
- **I²C lines:** SDA → A4, SCL → A5 (Arduino Nano hardware I²C pins).
- **I²C address:** the default address **0x68** was used by keeping **AD0 tied low (to GND)**.

Assembly considerations: the sensor module was mounted on a breadboard and connected with short jumper wires to reduce intermittent contacts and improve signal integrity [1].

2.2.2 Software libraries and data acquisition.

The sensor was accessed using **Jeff Rowberg's I2Cdevlib / MPU6050 library**. The sketch initialised the I²C bus, verified device connectivity, and continuously read:

- Accelerometer raw outputs: **ax, ay, az**
- Gyroscope raw outputs: **gx, gy, gz**

To express acceleration in physical units, the accelerometer was operated in the **±2 g full-scale range**, where the scale factor is 16384 LSB/g [1]. Therefore, each axis acceleration (in g) was computed as :

$$a_x(g) = \frac{a_x}{16384}, a_y(g) = \frac{a_y}{16384}, a_z(g) = \frac{a_z}{16384} \qquad \textit{(Eq.1)}$$

For static tilt estimation, the **pitch** angle was computed from accelerometer components using the gravity vector geometry:

$$pitch = arctan\ 2\left(-a_x, \sqrt{a_y^2 + a_z^2}\right) \cdot \frac{180}{\pi} \qquad \textit{(Eq.2)}$$

The sketch streamed a **tab-separated** output (**pitch (deg)** and one selected acceleration channel such as **az (g)**) to the Serial Monitor at a fixed sampling interval. The serial stream was copied into Excel for logging and plotting. The full Arduino sketch and the recorded raw dataset used for plotting are provided in **Appendix D.**

2.2.3 Measurement procedure for acceleration vs tilt.

The MPU-6050 board was slowly rotated by hand around one axis to generate a range of static orientations. For each orientation, multiple consecutive samples were observed; representative values were recorded to reduce the effect of noise. The collected dataset was then plotted to assess whether the acceleration component follows the expected gravity projection model (**Appendix D**).

2.3 Logic gates and PWM relevant to motor drive

Objective. To implement a safe and hardware-feasible motor direction/speed interface for an L293D H-bridge using **PWM speed control** and a single **direction bit (CTL)**, and to realize the required gating logic using **NAND-only (SN74HC00)**. The design goal is to guarantee **unipolar gating**: at any time, PWM is routed to **only one** of the two direction inputs (IN1/IN2), avoiding simultaneous switching/braking states [4].

2.3.1 Control requirement and gating logic (unipolar PWM routing)

For one L293D motor channel, direction is determined by **IN1/IN2**, while speed is determined by applying PWM to the active direction input [4]. A single digital signal **CTL** is used to select forward/backward:

- **CTL = 0 (forward):** PWM → IN1, IN2 held LOW.
- **CTL = 1 (backward):** PWM → IN2, IN1 held LOW.

Therefore, the required gated signals are:

- $Gate\_F = PWM \cdot \bar{CTL}$
- $Gate\_B = PWM \cdot CTL$

A compact truth table is given in **Appendix E** as **Table E.2**, showing that at most one gate can be HIGH for any (PWM, CTL) pair.

2.3.2 NAND-only implementation (SN74HC00)

Only 2-input NAND gates were allowed [4], NOT and AND were synthesised using NAND combinations:

- **Inverter (NOT):** $\bar{A} = NAND(A, A)$
- **AND using NAND + inverter:** $A \cdot B = NAND(NAND(A, B), NAND(A, B))$

Using the above identities, the two gated outputs were implemented as:

- Compute $\bar{CTL}$ using one NAND inverter.
- Generate $PWM \cdot \bar{CTL}$ and $PWM \cdot CTL$ using NAND-AND realizations.

This realization ensures the logic is reproducible with SN74HC00 and directly compatible with the L293D input requirements [4].

2.3.3 Hardware integration and wiring

Signal mapping (one motor channel) [4]:

- Arduino PWM pin → logic input PWM
- Arduino digital pin → logic input CTL
- Logic output $\boldsymbol{Gate\_F}$→ L293D IN1
- Logic output $\boldsymbol{Gate\_B}$→ L293D IN2
- **L293D EN** kept enabled so that speed is controlled by the gated PWM at **IN1/IN2**.

Power and grounding:

- SN74HC00 powered from +5 V logic, L293D logic supply VCC1 = +5 V [4].
- All grounds (Arduino, SN74HC00, L293D) were tied together to ensure a reference.
- A local 0.1 μF decoupling capacitor was placed close to the SN74HC00 supply pins to reduce switching noise on CMOS rails.

A schematic of the NAND gating + L293D channel wiring and a complete truth table are provided in **Appendix E.1.**

2.3.4 Verification procedure (logic correctness and motor response)

To verify the gating logic and motor response, the firmware performed:

1. **Static logic check:** with PWM applied, toggle **CTL** and confirm that motor direction changes while the inactive direction input remains LOW (no simultaneous drive).

2. **Dynamic PWM check:** sweep PWM duty cycle at a fixed direction to confirm smooth speed variation.

The Arduino verification sketch used to generate PWM/CTL and compute speed/distance from encoder pulses is included in **Appendix F**.

2.4 Remote HMI hardware (soldering & debugging)

Objective. To assemble and validate a handheld remote human–machine interface (HMI) that provides (1) real-time user input (joystick) (2) on-device status display (LCD1602) and (3) wireless communication module (nRF24L01), with a stable power subsystem. The focus of this session was hardware bring-up, soldering quality, and systematic debugging before software tasks.

2.4.1 Hardware assembly, wiring, and power integrity

To build a **remote HMI** capable of controlling the robot while displaying real-time status, we assembled a custom prototyping board populated with the following parts:

- **Arduino Nano.** Microcontroller providing ADC inputs, SPI for nRF24L01+, SoftwareSerial for HC-06 (if used later) and LCD digital control lines.
- **Joystick module.** Two 10 kΩ potentiometers supply analogue voltages proportional to X and Y tilt (VRx → A0, VRy → A1) and an optional push-button line (SW → digital D7) enables auxiliary functions.
- **LCD 1602 display.** Wired in 4-bit parallel mode with pins RS→D4, EN→D3, D4→D5, D5→D6, D6→D8, D7→D2, power pins to 5 V and GND, and backlight power through a 220 Ω resistor. A 10 kΩ trimmer between 0–5 V provided contrast control (VO pin).
- **nRF24L01.** Connected via SPI (MISO→D12, MOSI→D11, SCK→D13, CE→D9, CSN→D10) and powered from the regulated 3.3 V output on the prototyping board.
- **Power subsystem.** A MP1584 buck regulator converted a 2×18650 battery pack (~7.4 V) down to 5. Common grounds were maintained throughout the HMI and the robot to prevent spurious resets.

A representative schematic of the remote HMI wiring is provided as **Fig. 2**, while the top-view and underside solder-joint photos are included in **Appendix H** for build traceability.

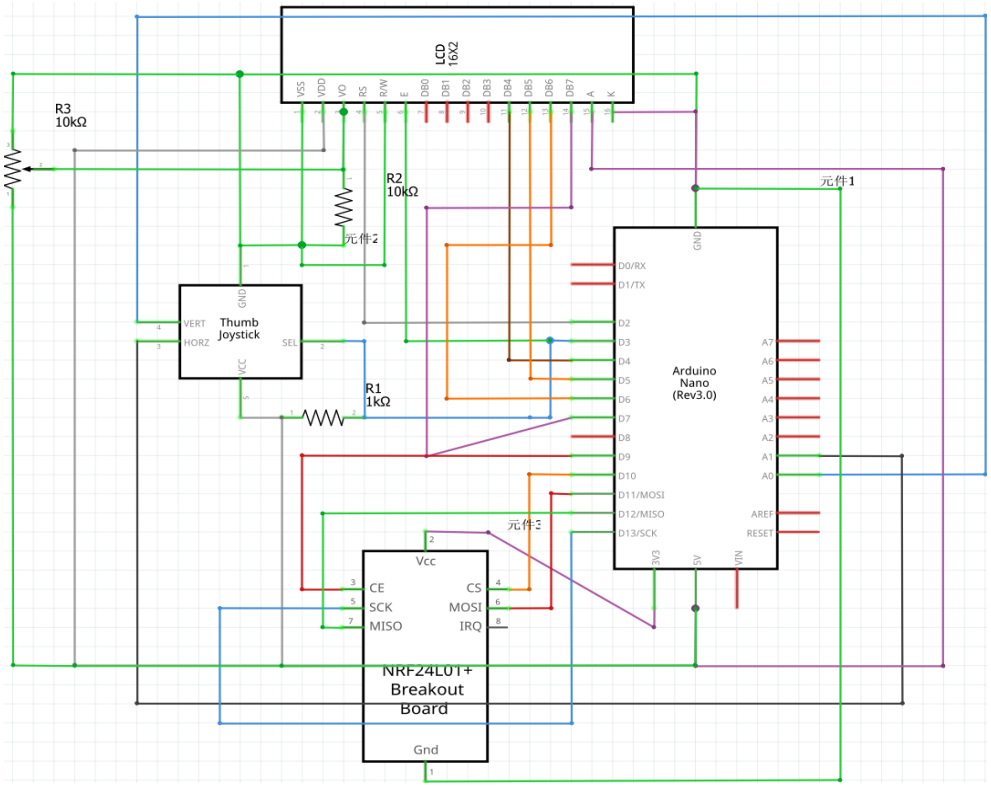


Figure 2. Remote HMI circuit schematic

2.4.2 Verification procedure

Hardware bring-up was performed in a fixed order (**power → display → analogue inputs → RF link**) to isolate faults efficiently:

1. **Power integrity (multimeter).** The MP1584 output was measured before connecting modules to confirm a stable **5 V rail** (≈ **5.02 V no-load**, **4.98 V under load**) and no abnormal current draw (≈ **60 mA idle**). This prevented intermittent resets during RF transmission and LCD updates.

2. **LCD bring-up (LiquidCrystal test).** A minimal sketch initialised the LCD in 4-bit mode and printed fixed labels to validate **RS/EN/D4–D7 wiring**. The contrast trimmer was adjusted to confirm correct **VO** connection and stable supply. The verification sketch is provided in **Appendix G**.

3. **Joystick ADC validation.** The same sketch sampled **A0/A1** and displayed raw ADC values. Centre readings were near mid-scale (~510–512), and full travel approached **0** and **1023** with smooth transitions, confirming correct analogue wiring and stable ADC reference.

4. **nRF24L01+ readiness and stability.** A minimal RF initialisation / send–receive test was run to confirm reliable module startup and packet transmission. If instability occurred, troubleshooting prioritised: (i) 3.3 V supply and common ground, (ii) shorter SPI wiring, and (iii) decoupling capacitors placed close to the module power pins.

Any detected issues (swapped wires or cold joints) were corrected, and the same sequence was re-run to ensure the remote HMI was reproducible before later challenge tasks.

2.5 Remote / automatic control challenge tasks

Objective. This section documents the implementation workflow of the challenge tasks that integrate (1) **autonomous line tracking + ramp event handling**, and (2) **remote/manual control** via nRF24 joystick and HC-06 Bluetooth. The scoring focus was on **robust closed-loop behaviour**, clear safety handling **(watchdog stop)**, and **traceable runtime feedback** (telemetry / status display). Full sketches are provided in **Appendix I**.

2.5.1 Task A – Automatic path tracking with ramp

Track and required behaviour. A tape-based closed-loop track included a ramp on a straight segment. The target sequence was: **line following → ramp climb → stop at top → 360° rotation → resume line following**.

Key control design (core scoring points).

1. **Ramp event detection (pitch-based state logic).** Ramp entry/exit was detected using **filtered pitch** from the MPU-6050. A short moving average (5-sample) was applied to suppress vibration spikes, and a threshold (**H > 10°**) was used to indicate uphill entry.

2. **Heading correction on ramp (differential compensation).** To mitigate drift on the ramp, a small differential correction was applied to left/right wheel commands based on a yaw-related error signal, improving straightness and reducing line loss on inclined surfaces.

3. **Top-platform rotation (closed-loop turning).** The 360° rotation routine used a closed-loop measure of turning progress (yaw accumulation / integrated change) instead of pure open-loop timing, improving repeatability under friction.

4. **Telemetry for traceability.** During Task A, key runtime information was transmitted to the handheld LCD via nRF24, enabling verification of ramp detection and rotation behavior. Relevant code is provided in **Appendix I.1.**

Success criteria (method verification). Task A was considered successful when the car completed the ramp segment without leaving the tape, executed stop + 360° turn on the top platform, required the line, and continued tracking while telemetry remained stable.

2.5.2 Task B1 – nRF24 joystick remote control

System split (TX/RX). Two sketches were used:

- **TX (handheld):** reads joystick axes (VRx/VRy) and optional button state, maps them into left/right wheel commands, and transmits a compact payload over nRF24.
- **RX (vehicle):** decodes the payload and drives motors through the I2C motor driver interface.

Safety watchdog. A receiver-side watchdog halted the vehicle if no valid packet was received within a short timeout (~200 ms), preventing runaway motion under RF dropouts.

Status feedback to LCD. To meet the "status display" requirement, the handheld processed return/ACK telemetry (RUN/STOP, speed/distance, tilt) and refreshed the LCD at a stable rate (no flicker, responsive updates). Relevant codes are included in **Appendix I.2 & I.3.**

2.5.3 Task B3 – Bluetooth remote control with HC-06

Hardware and interface. In Task B3, the vehicle radio link was replaced with an HC-06 Bluetooth UART module to accept commands from a smartphone serial terminal. The HC-06 used a SoftwareSerial port on the Nano. The wiring was: **VCC→5 V**, **GND→GND**, **TXD→D10** (RX) and **RXD→D11** via a voltage divider to avoid over-voltage from the 5 V Nano TX. The same I²C interface as previous tasks connected the Nano to the baseboard (pins SDA→A4, SCL→A5).

Command parser and drive state machine. The firmware implemented a single-character command parser. Each received command updated a **driveMode** state and a **speed** variable, then applied the corresponding motor command via driveAll(...).

Command set and user feedback. The control program taskB3code.ino shown in **Appendix I.4** implemented a simple command parser. A concise mapping between phone commands and actions is recommended as **Table 1**.

Table 1. HC-06 command mapping (command → action → feedback).

Command	Action	Feedback
F	Drive forward	1 kHz beep (120 ms)
B	Drive backward	800 Hz beep (120 ms)
L	Rotate left in place	600 Hz beep (120 ms)
R	Rotate right in place	1.2 kHz beep (120 ms)
S	Stop	400 Hz beep (120 ms)
1	Set speed to 40 %	900 Hz beep (80 ms)
2	Set speed to 60 %	1.1 kHz beep (80 ms)
3	Set speed to 80 %	1.3 kHz beep (80 ms)

Verification procedure.

- Pair HC-06 with the phone terminal and confirm stable text transmission.
- Send each command and confirm the corresponding vehicle motion
- Confirm watchdog stop when the phone stops sending commands or disconnects.

2.6 TCRT5000 reflective sensor characterization

Objective. Characterise the analogue output $V_{A0}$of a TCRT5000 reflective sensor module as a function of distance $d$for **white** and **black** surfaces, and derive the sensor sensitivity $\Delta V_{A0} = V_{A0,\text{black}} - V_{A0,\text{white}}$to determine a practical operating range for line-following [6].

2.6.1 Principle (what A0 represents).

The TCRT5000 contains an IR LED (emitter) and a phototransistor (receiver). The phototransistor behaves as a light-dependent resistance/conductance. On the module, the analogue pin **A0** is produced by a voltage divider, therefore:

- **Higher reflectivity (white)** → stronger received IR → phototransistor conducts more → **A0 voltage decreases** (pull-down effect) [5].
- **Lower reflectivity (black)** → weaker reflection → phototransistor conducts less → **A0 voltage increases** (approaching V_{CC}) [5].

2.6.2 Experimental setup.

The sensor was powered from a regulated **5 V** rail and placed above two controlled surfaces: **white paper** and a **black background/tape**. The sensor height $d$was adjusted and measured using a ruler while keeping the sensor approximately normal to the surface and positioned above a uniform region. Wiring and key measurement settings are summarised in **Table 2**.

Table 2. *TCRT5000 wiring and measurement settings*

Item	Connection	Notes
Supply	VCC=5V, GND common	for repeatable ADC/voltage
Analogue output	A0→Arduino analogue input	converted to voltage
Digital output (optional)	DO →Arduino digital input	as a threshold indicator
Distance range	d = 0.5 to 6.0cm, step 0.5cm	Same for white and black
Logging method	Serial stream to PC	sketch is in **Appendix L**

2.6.3 Procedure and data acquisition

1. Power the sensor and allow readings to stabilise (~5–10 s).
2. For each distance d, record the sensor output over **white** and then over **black** under similar ambient lighting.
3. For each d, record a stable reading and take a representative value.
4. The output stream was copied into a spreadsheet for tabulation and plotting. Where raw ADC values were logged, conversion to voltage used:

$$V_{A0} = ADC \cdot \frac{V_{CC}}{1023} \qquad (Eq.3)$$

5. Compute $\Delta V_{A0}(d)$to quantify surface separability and identify a practical operating range for line-following.

The Arduino logging sketch used for acquiring the analogue readings is in **Appendix L**.

2.7 Basic line following with two sensors

Objective. Achieve stable line following for two continuous cycles using two TCRT5000 analogue sensors and a discrete PID controller

2.7.1 Hardware setup and wiring

- **Sensors (TCRT5000 ×2):** mounted at the front underside of the chassis (left/right), close to the ground to maximise contrast between black tape and light floor.
- **Analogue inputs:** left sensor AO → A6, right sensor AO → A7. Digital DO outputs were not used for control.
- **Motor interface:** wheel-speed commands transmitted to the motor baseboard via I²C at address 42, using "sa" followed by left/right speed bytes (0–100) (Appendix J).

2.7.2 Sensor placement and quick calibration

To maximise contrast on the track (black tape on light floor), the sensors were aligned symmetrically about the tape centerline so that when centred **A6 ≈ A7**. Sensor height and onboard potentiometers were adjusted to avoid saturation (readings not pinned near 0/1023), ensuring a usable analogue error signal.

2.7.3 Control algorithm

A single sketch (Appendix J) executed a fixed-rate control loop (≈100 ms):

1. **Read sensors:** $L = \text{analogRead}(A6)$, $R = \text{analogRead}(A7)$
2. **Compute lateral error:** $e = L - R$
3. **Discrete PID:** $I \leftarrow \text{constrain}(I + e, -500, 500)$, $D \leftarrow e - e_{\text{prev}}$, and:

$$u = K_p e + K_i I + K_d D e \; Eq.4 \qquad (Eq.4)$$

4. **Differential speed command:** leftspeed = speed - u, rightspeed = speed + u
5. **Actuate motors:** send speeds via I²C using car_control1(leftspeed, rightspeed).

2.8 8-sensor array, weighted-average position, and calibration

Objective. Establish reliable I²C communication with the SF-8CHDTS 8-channel reflective sensor array, implement per-channel calibration and normalisation, and compute a continuous lateral position estimate (weighted average) suitable for closed-loop line following.

2.8.1 Hardware interface and data format (I²C)

The SF-8CHDTS sensor array was connected via the Arduino Wire (I²C) bus. The device address was verified as **0x09** using an I²C scanner (Appendix **K.1**). Each sensor update returns **16 bytes** (8 channels × 2 bytes), interpreted as one 16-bit raw value per channel:

$$raw_i = (hi_i \ll 8) \mid lo_i, i = 0 \ldots 7 \qquad (Eq.5)$$

A raw streaming test was then used to confirm stable 8-channel data and correct byte order before adding any processing (Appendix **K.2**).

2.8.2 Calibration strategy (white/black references)

Because absolute reflectance differs across channels and ambient conditions, a **per-channel calibration** was adopted using two reference arrays:

- $whiteRef[i]$: reading on background (white floor)
- $blackRef[i]$: reading on line (black tape)

The calibration routine stored these references and printed them for inspection (Appendix **K.3**). This step ensures that later scaling reflects **relative contrast** rather than absolute sensor variation.

2.8.3 Normalisation (raw → scaled)

Each channel was mapped into a unified scale **0…1000** (0 ≈ background, 1000 ≈ line), using:

$$s_i = clip\left(\frac{whiteRef[i]-raw_i}{whiteRef[i]-blackRef[i]} \cdot 1000,\ 0,\ 1000\right) \qquad \textit{(Eq.6)}$$

This normalisation makes channels comparable and prevents one "strong" channel dominating the position estimate simply due to hardware offset. (If the sensor polarity is reversed on a particular setup, the mapping branch in code handles that case **Appendix K.3**)

2.8.4 Weighted-average position estimation + line-loss detection

A continuous position estimate was computed using fixed symmetric weights:

$$w = [-3.5,\ -2.5,\ -1.5,\ -0.5,\ 0.5,\ 1.5,\ 2.5,\ 3.5], \qquad pos = \frac{\sum w_i s_i}{\sum s_i} \qquad \textit{(Eq.7)}$$

To detect line loss, the total intensity was monitored:

$$sumScaled = \sum s_i \qquad \textit{(Eq.8)}$$

If $sumScaled < SUM\_LOST\_TH$, the code outputs a sentinel to trigger recovery logic in the controller **(Appendix K.3**).

2.9 Real-time PID control + HMI tuni5g + challenge run

Objective. Integrate the 8-sensor position estimate into a real-time steering controller, provide an HMI for live tuning, and demonstrate stable closed-loop line following over repeated laps.

2.9.1 Real-time control loop (sense → estimate → control → actuate)

At each loop iteration:

1. Read 8-sensor raw data via I²C (readSF8)
2. Convert to scaled values (scaledSensor) and compute pos (computePosition)

3. Compute control error err = pos (center setpoint = 0)

4. Convert steering into differential wheel commands:

$$left = baseSpeed - turn, right = baseSpeed + turn \quad \textit{(Eq.9)}$$

5. Send motor commands to the motor baseboard over I²C using the existing drive function (**Appendix K.4**).

2.9.2 Discrete PID steering with limits (stability-focused)

A discrete PID controller was used to generate $turn$from err, with stability mechanisms:

- **Anti-windup**: integral term constrained within a safe bound to prevent runaway accumulation.
- **Output limiting**: clamp turn, left, and rightto valid motor ranges to avoid saturation-driven instability.
- **Consistent loop timing**: controller executed at a controlled update rate (timed loop), ensuring repeatable dynamics.

These measures are key for robust line following when the robot encounters sharp corners, speed changes, or brief sensor disturbances.

2.9.3 HMI for live tuning (LCD + rotary encoder)

A handheld HMI (LCD + rotary encoder) was used to adjust control parameters during operation without reflashing:

- Parameter selection cycled through K_p, K_i, K_d, and baseSpeed.
- Encoder rotation incremented/decremented the selected value in fixed steps.
- Values were clamped to predefined safe ranges to prevent unstable settings.

This interface reduced iteration time and enabled systematic tuning directly on the track.

2.9.4 Safety behaviour (line loss and I²C robustness)

Safety logic was included to prevent uncontrolled motion:

- **Line loss:** if $pos$is invalid / sentinel, the robot enters a recovery behaviour or stops depending on the implemented strategy.
- **I^2C failure handling:** if a sensor read fails, the firmware applies a conservative safe action (e.g., halt command) rather than continuing with stale data.

Chapter 3 Results and Discussions

3.1 nRF24L01+ and HC-06 communication verification

3.1.1 nRF24L01+ link test results

After uploading the RX sketch, the receiver Serial Monitor displayed a readiness message, indicating successful initialization of the RF24 module and reading pipe. When the TX sketch was executed, the receiver repeatedly printed the transmitted payload.

Observed outcome:

- Receiver printed initialization line ("nRF24L01 ready!").
- The message payload ("Happy Hacking!") appeared repeatedly at around 1 Hz.
- No manual reset was required once the radios were powered and configured, indicating stable SPI communication and correct CE/CSN wiring.

The complete TX/RX code is included in **Appendix A**, with wiring photos in **Appendix B**.

Discussion. This test verifies an end-to-end **packetized 2.4 GHz link** suitable for later tasks (telemetry, joystick control) and confirms that the chosen SPI pin mapping and radio configuration are reliable under continuous transmission.

3.1.2 HC-06 Bluetooth link test results

The HC-06 forwarding sketch successfully formed a transparent UART bridge between the Nano and a smartphone terminal (**Appendix A**): characters entered on the phone appeared unchanged on the Serial Monitor, and echoed output confirmed bidirectional serial forwarding.

In addition, the AT-mode renaming procedure behaved as expected (**Appendix A)**: AT commands returned valid acknowledgements and the updated device name appeared during subsequent scanning. This enabled unambiguous device selection in a shared lab environment.

Observed outcome:

- Phone → Nano: typed characters appeared on the PC Serial Monitor.
- Nano → phone: echoed strings were visible in the phone (bidirectional link).
- AT responses confirmed command reception and successful naming update.

Discussion: HC-06 provides a low-friction **smartphone HMI channel**. The renaming step reduces pairing errors and improves repeatability when multiple Bluetooth modules are present.

3.2 MPU-6050 sensing validity

3.2.1 Connectivity and data sanity checks

Once powered and addressed on the I²C bus, the MPU-6050 streamed stable raw accelerometer and gyroscope readings. The Serial Monitor output maintained the expected numeric ranges and update rate, confirming that (1) the I²C read routine, (2) axis decoding, and (3) unit conversion were functioning correctly.

For the ±2 g accelerometer range, the converted acceleration components were computed from the raw registers using the 16384 LSB/g scale factor, which produced physically consistent values during static tests (near ±1 g magnitude along the gravity-aligned axis) [1].

3.2.2 Static tilt relationship: acceleration projection trend

Fig. 3 shows a representative forward-and-return manual rotation. The measured a_z varies within approximately $\pm 1\ g$ while the pitch spans a wide angular range (tens of degrees), demonstrating that the accelerometer captures the gravity component along the sensor's z-axis as expected [2]. Overall, the trend is consistent with the gravity projection model,

$$a_z \approx g\cos(\theta) \qquad \textit{(Eq.10)}$$

where θ is the tilt angle. A small loop-like hysteresis is visible in the time-ordered trace, which is expected when the motion is not perfectly quasi-static. Despite this, the results confirm that the MPU-6050 provides usable orientation-related information for subsequent motion tasks [3].

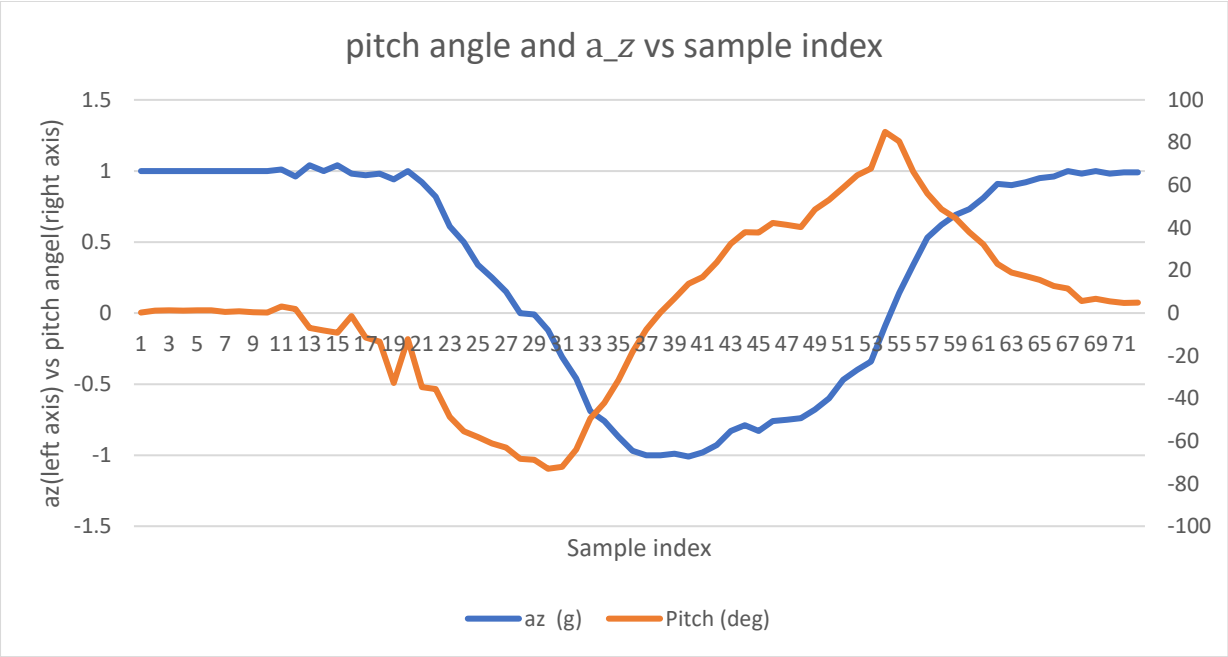


Figure 3. The pitch angle and the z-axis acceleration component a_z against sample index

3.3 PWM/logic verification

3.3.1 Functional verification of unipolar PWM routing

The NAND gating logic was validated by observing the motor driver behaviour under both direction states:

- **CTL = 0 (forward selected):** PWM was applied only to the forward input (IN1), while the reverse input (IN2) remained LOW.
- **CTL = 1 (reverse selected):** PWM was applied only to the reverse input (IN2), while IN1 remained LOW.

This confirms **unipolar gating**: at no time are both direction inputs PWM-driven simultaneously, preventing invalid switching states at the H-bridge input.

3.3.2 Direction and speed response versus PWM duty

With CTL held constant, increasing PWM duty produced a clear increase in encoder-derived wheel speed, demonstrating that the PWM information was preserved through the NAND network and correctly interpreted by the L293D channel. Switching CTL inverted the effective driven direction input and reversed the sign of motion accordingly, consistent with the intended mapping "PWM magnitude + CTL direction".

Discussion. These observations verify that the logic stage acts as a **safe interface** between higher-level controllers (PID/joystick/autonomy) and the motor driver: software can command speed via PWM while hardware enforces mutually exclusive direction actuation.

3.3.3 Timing considerations and practical robustness notes

The NAND propagation delay is in the nanosecond range, whereas Arduino PWM periods are in the millisecond range (sub-kHz). Therefore, the gate delay is negligible relative to the PWM period and does not materially distort duty ratio at the motor driver input.

One potential corner case is toggling CTL while PWM is HIGH, which could theoretically create a brief glitch due to purely combinational gating. In this implementation, direction changes were performed only when PWM was reduced to zero (i.e., switching between duty-sweep phases), making the verification unambiguous. For future integration, robustness can be further improved by enforcing a short "dead-time" in software (PWM→0, wait, toggle CTL, ramp PWM) or temporarily disabling the L293D enable during direction switching.

3.4 Remote HMI functionality

3.4.1 LCD + joystick readout verification

After assembly, the handheld HMI operated reliably and provided real-time feedback suitable for later remote-control integration:

- **LCD1602 display stability:** The LCD displayed static labels and continuously refreshed numeric values without visible flicker at the target update rate (~10 Hz), confirming correct 4-bit wiring and stable contrast setting.
- **Joystick ADC behaviour:** The joystick axes produced smooth and repeatable ADC variation. Typical centre readings were near mid-scale (~510–512), and full travel approached the expected ADC limits (near 0 and 1023), indicating correct analogue wiring and an adequate ADC reference.

Supporting evidence: LCD bring-up sketch and display test outputs are provided in **Appendix G**, and build/solder evidence is provided in **Appendix H**.

3.4.2 Power integrity and RF readiness (system-level robustness)

The HMI power subsystem remained stable during operation and prevented reset-related faults:

- **Regulated 5 V rail:** The measured output remained close to 5 V both at no-load and under typical operating load, indicating sufficient headroom for LCD updates and RF activity.
- **nRF24L01+ readiness:** The RF module initialized consistently (no repeated reboots/lockups observed during repeated tests), supporting that the wiring, common ground, and local decoupling were effective.

Discussion. These results show the handheld HMI was not only functionally correct (inputs + display), but also **robust enough for later integration** where RF bursts and LCD refresh can otherwise cause brown-outs or intermittent resets.

3.5 Session 3 – Topic 5: Challenge tasks (remote / automatic control)

3.5.1 Task A – Automatic path tracking with ramp

Two trials were performed with the ramp placed at different positions. In both runs, the car maintained autonomous tracking, correctly detected the ramp and top platform, **stopped for 2 s**, performed the required rotation, and re-entered line following.

Observed outcomes (representative):

- **Ramp event recognition:** The controller detected the ramp entry/exit consistently (pitch-threshold logic), triggering state transitions at the correct stage of motion rather than by timing alone.
- **Stop-and-hold at top platform:** The vehicle reliably paused for the required dwell time without drifting off the track.
- **Rotation completion:** The rotation routine completed and the vehicle subsequently re-acquired the track and continued line following.

Discussion. Using an IMU-derived event cue (pitch) reduced sensitivity to battery level and floor friction compared with purely time-based heuristics. Minor lateral deviation could still occur during the climb due to traction asymmetry; this is a mechanical/drive-train limitation rather than a sensing failure and would be further reduced by improved wheel matching or stronger closed-loop correction during high-load segments.

Relevant implementation and state-machine logic are documented in **Appendix I**.

3.5.2 Task B1 – nRF24 joystick remote control

The nRF24 joystick remote provided responsive and controllable manual driving:

- **Direction + proportional speed:** Forward/back joystick motion produced corresponding vehicle motion, and larger deflections produced higher speed
- **Steering by differential command:** Lateral stick movement produced smooth steering through left/right command imbalance rather than abrupt switching.
- **Link robustness and safety:** Communication remained stable over typical indoor range; when packets were lost or the link dropped, the receiver-side watchdog stopped the vehicle rather than allowing uncontrolled motion.

Discussion. The results demonstrate the remote-control stack achieved both **human usability (smooth response)** and **safety (watchdog stop)**, satisfying typical marking expectations for reliable control and fail-safe behaviour.

3.5.3 Task B3 – Bluetooth remote control with HC-06

Bluetooth control using a smartphone serial terminal provided a practical alternative to RF remote control:

- **Command execution:** Commands entered in the phone terminal were executed immediately, with consistent mapping from command to motion.
- **Fail-safe behaviour:** When the phone disconnected or commands stopped, the watchdog logic halted the vehicle, preventing runaway motion.

Discussion. While the Bluetooth range was shorter than the nRF24 solution and more line-of-sight dependent, it was highly convenient for quick testing because it only needs a phone. Watchdog behavior is the key safety feature makes this mode acceptable for unattended motion.

3.6 Optical sensor characterisation (TCRT5000)

3.6.1 Measured V_{A0} and sensitivity

The measured analogue voltage above white and black surfaces are listed in **Appendix M.1** The derived sensitivity ΔV_{A0} is included to quantify discrimination margin:

$$\Delta V_{A0}(d) = V_{A0,\,black} - V_{A0,\,white} \qquad (Eq.11)$$

The corresponding plots are shown in **Fig. 4** (raw voltages) and **Fig. 5** (margin vs distance).

- **Black surface response:** $V_{A0,black}$ increased rapidly with distance and then saturated close to the supply rail (~4.5 V), consistent with low reflectivity producing a weaker received IR signal and reduced phototransistor conduction.
- **White surface response:** $V_{A0,white}$ remained low at short distances and increased at larger distances, reducing contrast at far range.

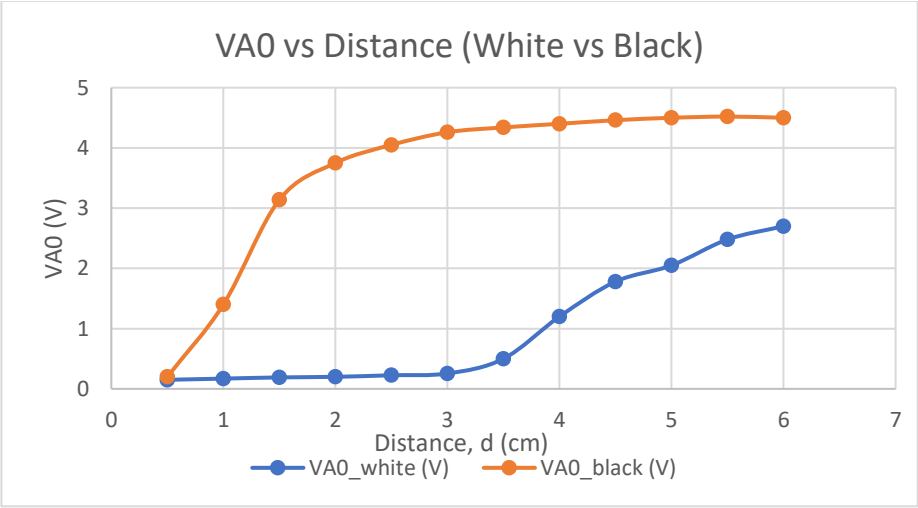


Figure 4. *Measured V_{A0} vs distance d for white and black surfaces.*

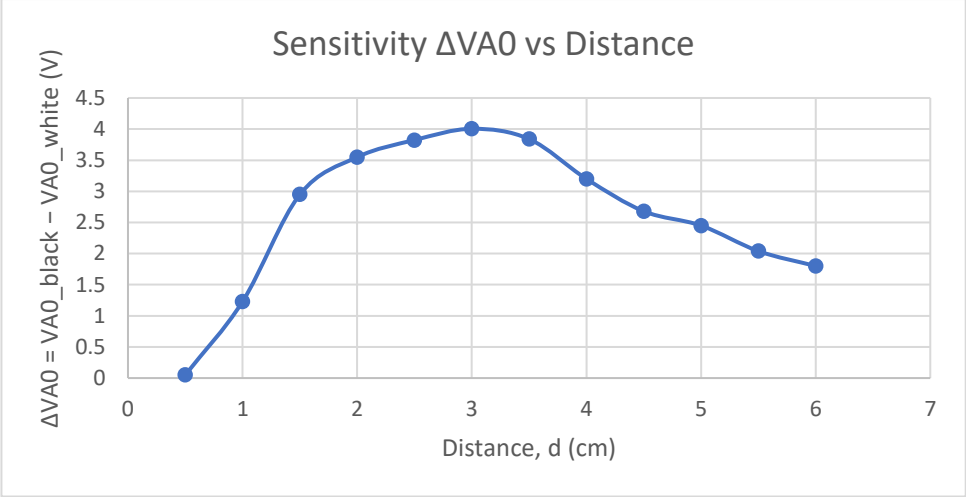


Figure 5. *Sensitivity* $\Delta V_{A0} = V_{A0,black} - V_{A0,white}$ *vs distance* d.

3.6.2 Key trends and recommended operating range

The discrimination margin ΔV_{A0} provides a direct measure of how reliably the sensor can separate black tape from a light background:

- The margin rises from near zero at very small gaps to a clear peak around $d \approx$ 3.0cm, where ΔV_{A0} is approximately 4.0 V (highest contrast).
- Beyond this region, ΔV_{A0} gradually decreases as the white reading rises and the black reading approaches saturation.

Recommended working band (for robust line following):

- A practical operating region is $d \approx$ 2.0–4.0 cm, where ΔV_{A0} remains high (≈3.2 V)
- The most tolerant "sweet spot" is $d \approx$ 2.5–3.5 cm, where the margin is near its maximum and the curve is relatively flat, making the system less sensitive to small height variations and vibration.

3.6.3 Discussion and implications for integration

Overall, the results confirm the TCRT5000 analogue output provides **strong separation** between black tape and white background across a usable height range, enabling both thresholding and continuous-control approaches. The margin-based selection of mounting height directly improves tracking stability by maximising contrast while reducing sensitivity to mechanical disturbances.

3.7 Two-sensor line following: baseline vs improved method

3.7.1 Demonstration outcome

Using the final integrated sketch (**Appendix J**), the vehicle achieved stable **two-sensor line following** and completed the required two continuous cycles on the tape track without leaving the line. Steering was smooth and continuous, indicating that the controller maintained the vehicle close to the tape centreline throughout straight segments and corners.

3.7.2 Telemetry-consistent control behaviour

Bluetooth serial output (**Appendix J**) was used to observe the internal control variables during motion, including the two analogue sensor readings (**A6**, **A7**), the computed PID output (**pid_output**), and the resulting wheel commands (**leftspeed**, **rightspeed**). The streamed data showed the expected closed-loop pattern:

- **Centred tracking:** when the robot was approximately centred, **A6 ≈ A7**, giving a small error ($e = A6 - A7 \approx 0$). In this state, **pid_output** remained near zero and **leftspeed ≈ rightspeed**, producing straight motion.
- **Corrective steering:** when the robot drifted laterally, the sensor mismatch increased (**A6 ≠ A7**), enlarging *e*. The PID output then produced a **differential speed** command (*leftspeed = speed − u*, *rightspeed = speed + u*), generating a corrective turn back toward the centreline.
- **Bounded actuation:** both wheel commands were clamped to **0–100**, keeping I²C motor commands within a valid range and preventing runaway steering when transient errors occurred.

Overall, the telemetry was consistent with a stable closed-loop differential drive controller operating as designed (**Appendix J**).

3.7.3 Discussion (stability mechanisms that mattered in practice)

Three implementation details were particularly important for reliable multi-lap operation:

1. **Integral anti-windup:** constraining the integral term to ±500 prevented accumulation during sustained cornering or temporary saturation, improving recovery when the error reduced again.

2. **Command limiting:** clamping wheel commands avoided invalid motor requests and reduced saturation-driven oscillation.

3. **Fixed-rate loop:** the ~100 ms update interval provided repeatable timing and consistent controller response across the run.

Limitation: with only two sensors, the controller can only infer left/right contrast difference rather than a richer "position profile"; therefore stability depends more strongly on maintaining sensor symmetry and consistent ground clearance (as controlled in Method 2.7.2).

3.8 8-sensor array bring-up, calibration and position output

3.8.1 I²C detection and raw streaming verification

The staged bring-up sequence confirmed correct sensor-array communication:

- The I²C scanner consistently detected the array at address 0x09 (**Appendix K.1**), indicating correct wiring and bus addressing.
- Continuous raw reads returned stable **16-byte frames** (8 channels × 2 bytes) without intermittent dropouts under normal operation (**Appendix K.2**), validating byte order and repeatability before any scaling or control logic was added.

3.8.2 Calibration and normalised response

After capturing per-channel references (**whiteRef[i]**, **blackRef[i]**) and applying the 0…1000 scaling (**Appendix K.3**), the array outputs became consistent across channels:

- When the tape was placed under a specific channel, that channel's scaled value rose toward **the high end of** the scale while background channels stayed low.
- Importantly, the response was not dominated by a single "strong" sensor, indicating that per-channel calibration effectively reduced **channel-to-channel bias** and made the array suitable for position estimation.

This demonstrates that the normalisation stage successfully converted "raw reflectance differences" into a comparable scale usable by downstream algorithms.

3.8.3 Weighted position output and line-loss detection

Using the scaled sensor values, the weighted-average estimator produced a continuous lateral position output **(Appendix K.3**):

- The computed **pos** changed **sign** and **magnitude** appropriately when the tape was moved left/right beneath the array:
  - $pos < 0$: tape toward left
  - $pos \approx 0$: tape centred
  - $pos > 0$: tape toward right
- The total intensity metric (**sumScaled**) provided a practical **line-loss indicator**. When the tape was absent, **sumScaled** dropped below the threshold and the firmware produced a sentinel position to trigger recovery behaviour (**Appendix K.3**).

3.8.4 Discussion (why this mattered for later closed-loop control)

Compared with two-sensor signals, the 8-sensor pipeline produced (1) a **continuous** position estimate rather than a simple left–right difference, and (2) an explicit **line-loss flag**. These two outputs are the key enablers for robust closed-loop line following at higher speed and through sharper corners, because the controller can act on a richer measurement and can switch into a defined recovery mode when the line is not detected.

3.9 Closed-loop line following with 8-sensor PID + HMI tuning

3.9.1 Closed-loop tracking behaviour

With the 8-sensor position estimate used as the control feedback, the robot maintained line tracking through straight segments and corners with smooth steering corrections (continuous differential speed modulation). The controller output remained bounded due to output limiting, preventing saturation-driven oscillations.

3.9.2 On-track tuning using HMI

The LCD + rotary encoder HMI enabled live adjustment of K_p, K_i, $K_d$and baseSpeed while the robot was running. This made it possible to converge to stable settings quickly and repeatably without firmware re-upload, and reduced trial-to-trial variability caused by manual code edits.

3.9.3 Robustness and safety behaviour observed

The integrated firmware included safety-oriented handling that contributed to reliable operation:

- **Line-loss handling:** when the line-loss condition was detected, the system entered the defined recovery behaviour rather than continuing with incorrect steering commands.
- **I^2C fault handling:** if the sensor read failed, a conservative safe action (halt/stop command) prevented uncontrolled motion under stale or missing sensor data.
- **Bounded control actions:** clamping of steering and wheel commands reduced the likelihood of saturation-driven instability, particularly during sharp turns or transient noise.

3.9.4 Discussion (system-level integration outcome)

Overall, the results confirm that the complete pipeline—**I²C sensing → calibration/normalisation → weighted position estimate → PID steering → differential drive actuation + HMI**—operated as a coherent closed-loop system. The key contribution of the 8-sensor method is that it provides both a continuous position measurement and a clear line-loss condition, enabling stable tracking and defined safety behaviour under disturbances (imperfect tape contrast or brief sensor noise).

Chapter 4 Conclusion

4.1 Key outcomes of module verification and system demonstration

A modular embedded robotics platform was successfully implemented and validated from communication and sensing to safe actuation and closed-loop autonomy. Reliable wireless interaction was achieved via an **nRF24L01+ transceiver link** and an **HC-06 Bluetooth UART bridge**, both verified by repeatable bidirectional serial behaviour suitable for remote testing and runtime feedback. The **MPU-6050 IMU** was interfaced via I²C and demonstrated physically consistent static tilt response, establishing confidence in the sensing pipeline used for higher-level logic. For actuation robustness, a **NAND-only logic network** realised **unipolar PWM routing** for the L293D driver; duty-cycle tests confirmed correct direction selection and preserved PWM magnitude while preventing unsafe simultaneous drive states. A handheld **HMI (joystick + LCD)** was assembled and debugged to provide stable user input and status display. The **TCRT5000 reflective sensor** was characterised over black/white surfaces, and the measured discrimination margin versus distance justified a practical working height range for reliable tape detection. Closed-loop line following was demonstrated first with a two-sensor PID controller and then with an **8-sensor array** achieving stable I²C streaming, per-channel calibration, normalised response, weighted-average position output, and explicit line-loss detection. The final integrated controller combined the 8-sensor position estimate with bounded PID steering and on-track parameter adjustment via the HMI, enabling stable tracking through straights and corners with defined safety behaviour under line loss or read failure.

4.2 Limitations and recommended improvements

Overall, the results show a complete and reproducible workflow from verified modules to an integrated robotic system capable of both wireless interaction and autonomous tape tracking. The main limitations are sensitivity to mechanical alignment (sensor height/angle and chassis vibration), and performance margin under sharper corners or brief signal interruptions. Future improvements include tighter mechanical mounting and shielding for consistent reflectance, more explicit and faster line re-acquisition logic, and refined timing/transition handling to increase achievable speed without sacrificing robustness.

References

[1] InvenSense Inc., “MPU-6000 and MPU-6050 Product Specification,” Rev. 3.4, Aug. 19, 2013.

[2] C. J. Fisher, “Using an Accelerometer for Inclination Sensing,” Analog Devices, Application Note AN-1057, Rev. 0, 2010.

[3] M. Pedley, “Tilt Sensing Using a Three-Axis Accelerometer,” Freescale Semiconductor, Application Note AN3461, Rev. 6, Mar. 2013.

[4] Texas Instruments, “SN74HC00 Quadruple 2-Input NAND Gates,” Data Sheet SCLS181H, Aug. 2021.

[5] Vishay Semiconductors, “TCRT5000, TCRT5000L Reflective Optical Sensor with Transistor Output,” Data Sheet, Rev. 1.7, Aug. 17, 2009.

[6] Vishay Semiconductors, “Application of Optical Sensors – Reflective,” Application Note, Doc. No. 81449, Rev. 1.0, Sep. 27, 2006.

[7] Guangzhou HC Information Technology Co., Ltd., “HC-06 Bluetooth Module Data Sheet,” Rev. 2.0, Apr. 6, 2011.

Chapter 5 Appendix A — Wireless link code (nRF24L01+ TX/RX and HC-06 test sketches)

This appendix lists the complete Arduino sketches used to do command on nRF24L01 and HC06, supporting reproducibility of Topic 1.

5.1 **Listing A.1** nRF24_TX.ino (transmitter)

```
#include <SPI.h>
#include "RF24.h"

RF24 rf24(9,10); // CE, CSN

const byte addr[] = "1Node";
const char msg[] = "Happy Hacking!";

void setup() {
  rf24.begin();
  Serial.begin(9600);
  rf24.setChannel(93);
  rf24.openWritingPipe(addr);
  rf24.setPALevel(RF24_PA_MAX);
  rf24.setDataRate(RF24_2MBPS);
  rf24.stopListening();
}

void loop() {
  rf24.write(&msg, sizeof(msg));
  delay(1000);
  Serial.println("oK!");
}
```

5.2 **Listing A.2** nRF24_RX.ino (receiver)

```
#include <SPI.h>
#include "RF24.h"

RF24 rf24(9,10); // CE, CSN

const byte addr[] = "1Node";
const byte pipe = 1;

void setup() {
  Serial.begin(9600);
  rf24.begin();
  rf24.setChannel(93);
  rf24.setPALevel(RF24_PA_MAX);
  rf24.setDataRate(RF24_2MBPS);
  rf24.openReadingPipe(pipe, addr);
  rf24.startListening();
  Serial.println("nRF24L01 ready!");
}
```

```
void loop() {
  if (rf24.available(&pipe)) {
    char msg[32] = "";
    rf24.read(&msg, sizeof(msg));
    Serial.println(msg);
  }
}
```

5.3 **Listing A.3** HC06_bridge_test.ino (Bluetooth serial bridge / echo test)

```
#include <SoftwareSerial.h>

SoftwareSerial hc06(2,3);

void setup(){
  //Initialize Serial Monitor
  Serial.begin(9600);
  Serial.println("ENTER AT Commands:");
  //Initialize Bluetooth Serial Port
  hc06.begin(9600);
}

void loop(){
  //Write data from HC06 to Serial Monitor
  if (hc06.available()){
    Serial.write(hc06.read());
  }

  //Write from Serial Monitor to HC06
  if (Serial.available()){
    hc06.write(Serial.read());
  }
}
```

5.4 **Listing A.4** HC06_AT_rename.ino (AT-command rename bridge)

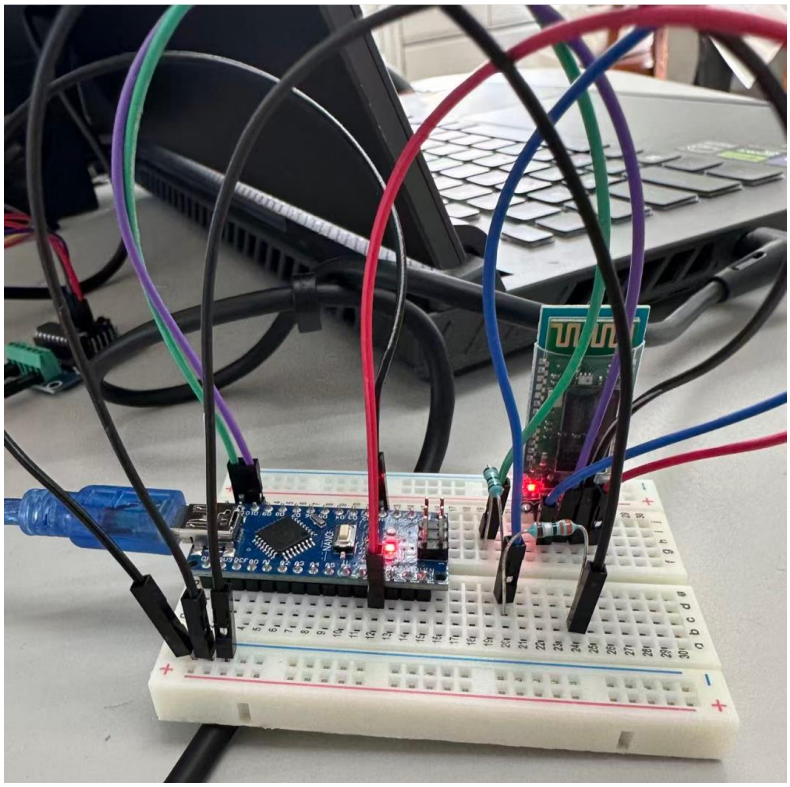
```
#include <SoftwareSerial.h>

// HC-06: TXD -> D10, RXD -> D11
SoftwareSerial BT(10, 11);  // RX, TX

void setup() {
  Serial.begin(9600);   // 电脑串口
  BT.begin(9600);       // 先假设 HC-06 AT 波特率是 9600

  Serial.println("HC-06 AT mode bridge ready.");
  Serial.println("Type AT commands here:");
}

void loop() {
  // PC -> HC-06
  if (Serial.available()) {
    char c = Serial.read();
    BT.write(c);        // 发给 HC-06
    Serial.write(c);    // 在屏幕上回显你输入的字符
```

```
  }

  // HC-06 -> PC
  if (BT.available()) {
    char c = BT.read();
    Serial.write(c);    // 把 HC-06 的回复显示出来
  }
}
```

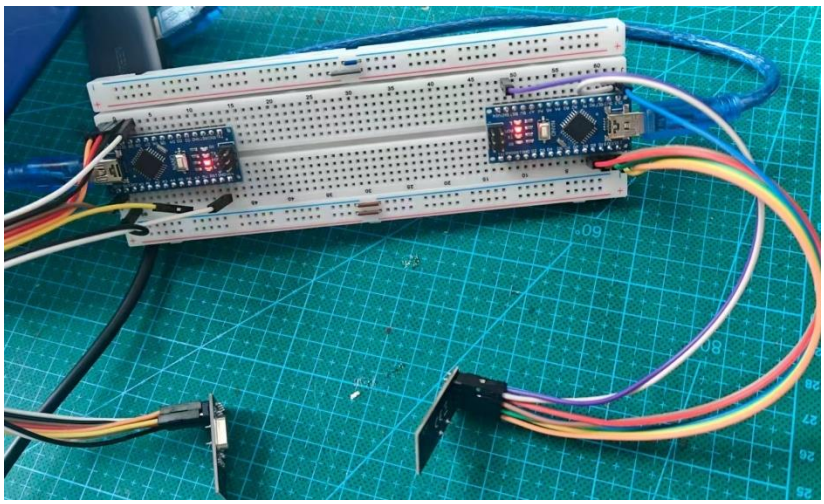
Chapter 6 Appendix B— Wiring photos and pin mapping tables (nRF24L01+ / HC-06)

This appendix provides photographic evidence of HC06 and nRF24L01 wiring for Topic 1.

6.1 **Fig. B.1** HC-06 module wiring and voltage divider implementation (photo of resistors and connections).



6.2 **Fig. B.2** nRF24L01 module wiring



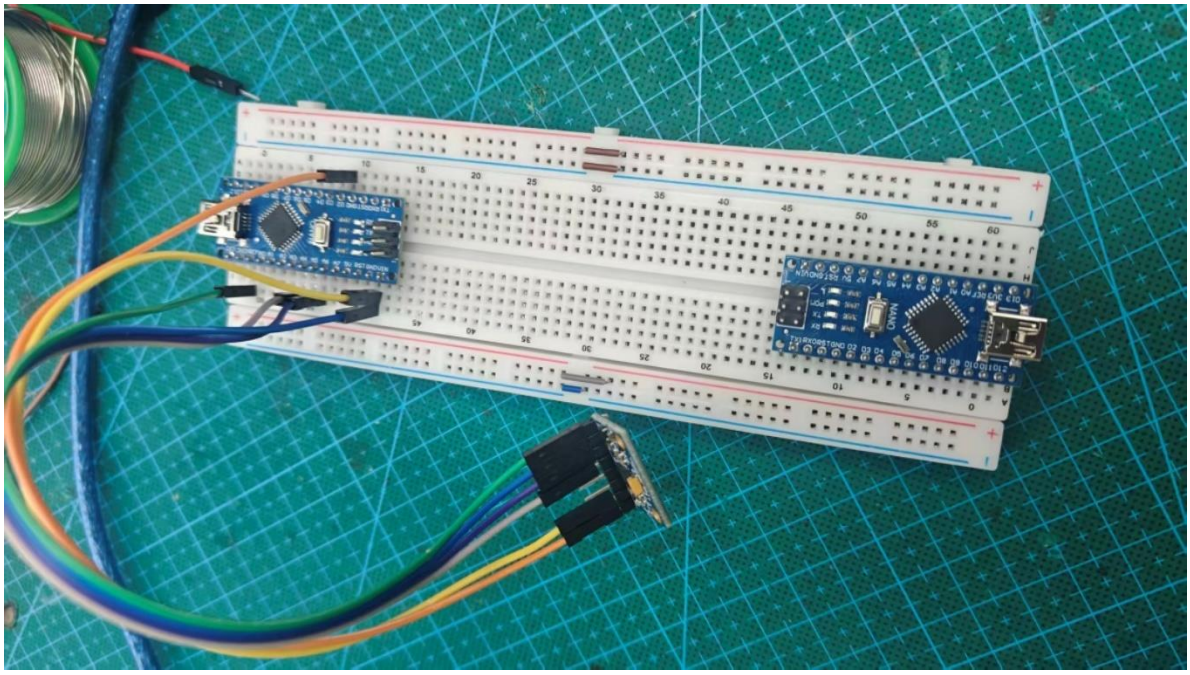
6.3 Table B.3 nRF24L01+ wiring (Arduino Nano)

nRF24L01	Nano Pin	Notes
VCC	3.3 V	5 V was avoided to prevent damage
GND	GND	Common ground required
CE	D9	Configured in code
CSN	D10	Configured in code
SCK	D13	Hardware SPI
MOSI	D11	Hardware SPI
MISO	D12	Hardware SPI

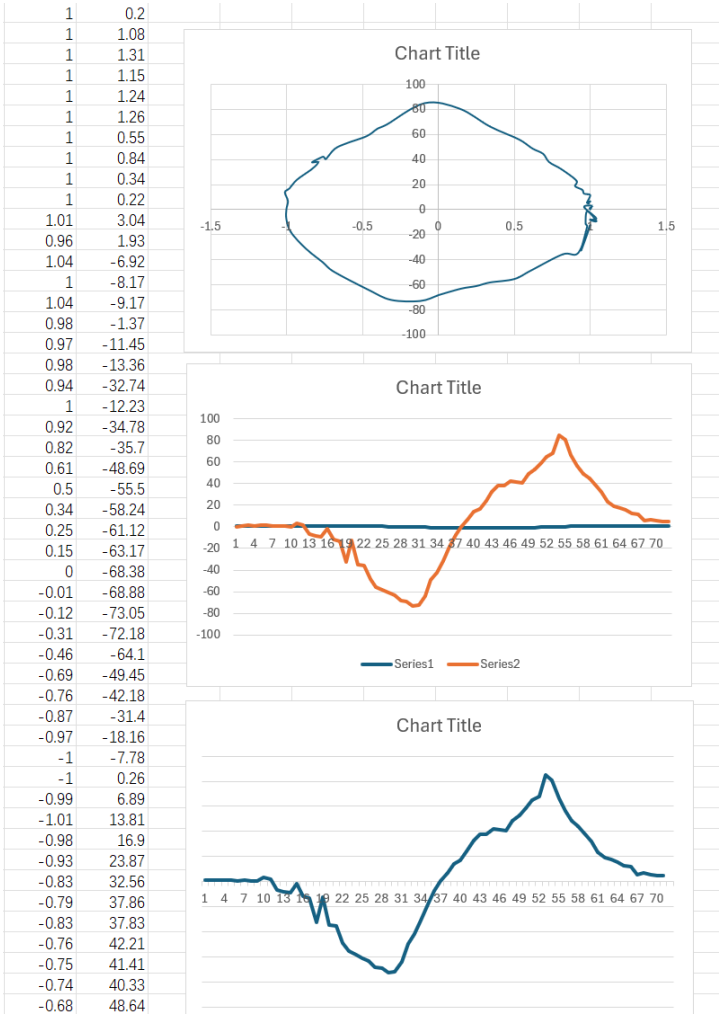
Chapter 7 Appendix C— MPU-6050 wiring and measurement evidence (photos / screenshots)

This appendix provides photographic evidence of the MPU-6050 I²C wiring and representative outputs used to verify data acquisition for Topic 2.

7.1 Fig. C.1 Breadboard wiring of the MPU-6050 to Arduino Nano via I²C.



7.2 **Fig. C.2** The screenshot of the recorded data of acceleration and tilt angle as well as their figures



Chapter 8 Appendix D— MPU-6050 code listings (raw read, tilt computation, DMP example)

This appendix lists the complete Arduino sketches used to acquire MPU-6050 data and compute tilt angle, supporting reproducibility of Topic 2.

8.1 Listing D1 basicreadexample.ino (I²C basic read)

```
#include <Wire.h>
#define uchar unsigned char
uchar t;
//void send  data(short a1,short b1,short c1,short d1,short e1,short f1);
uchar data[7];
void setup()
{
  Wire.begin();        // join i2c bus (address optional for master)
  Serial.begin(9600);  // start serial for output
  t = 0;
}
void loop()
{
  t = 0;
  Wire.requestFrom(0x1D, 7);    // request 7 bytes from slave device #0x1D
  while (Wire.available())   // slave may send less than requested
  {
    if (t < 7)
    {
      data[t] = Wire.read(); // receive a byte as character
      t++;
    }
    else
      Wire.read(); // dummy read
  }

  //Print the results (first byte is Status register and is ignored)
  Serial.print("X: ");
  Serial.print( (float)((data[1]<<8 | data[2])>>4)/1024 ); //Convert to bytes to a 12bit signed number, divide by 1024 to convert to g value
  Serial.print(" Y: ");
  Serial.print( (float)((data[3]<<8 | data[4])>>4)/1024 );
  Serial.print(" Z: ");
  Serial.println( (float)((data[5]<<8 | data[6])>>4)/1024 );
  delay(500);
}
```

8.2 Listing D2 MPU6050_raw.ino (tilt angle + acceleration output)

```
#include <SoftwareSerial.h>
#include "I2Cdev.h"
#include "MPU6050.h"
```

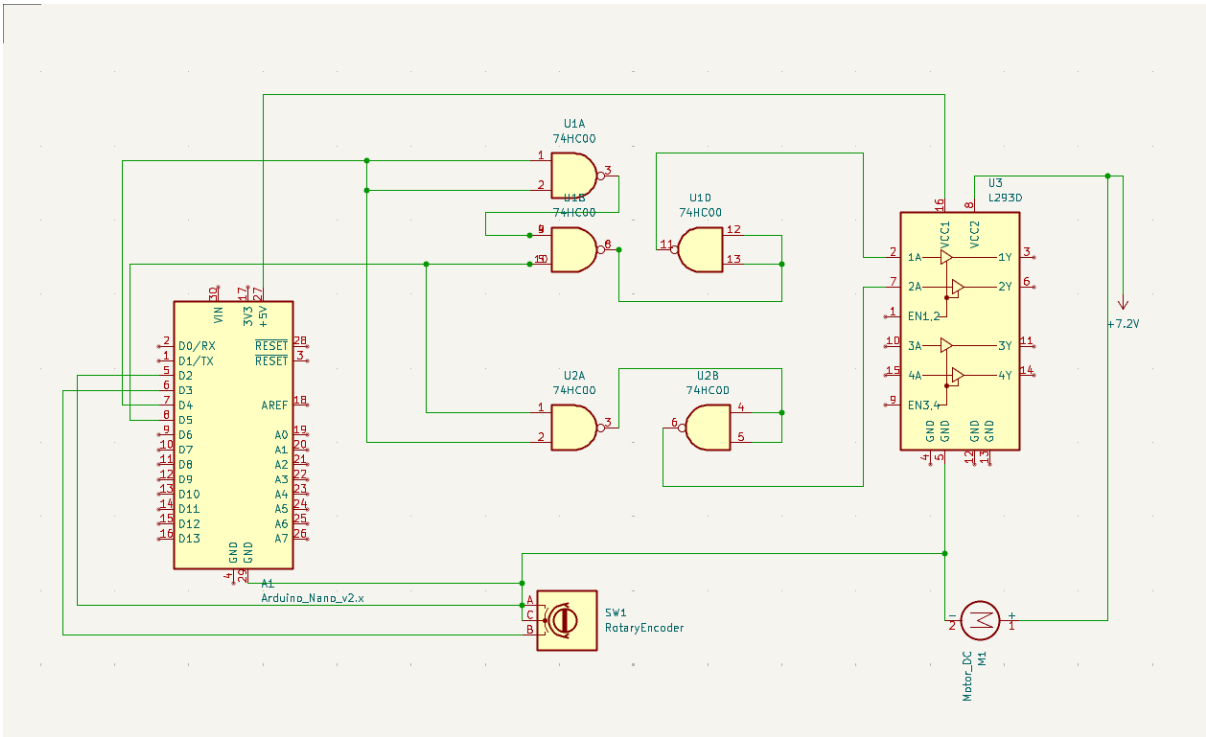
```
SoftwareSerial BT(10, 11); // RX, TX

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 accelgyro;
//MPU6050 accelgyro(0x69); // <-- use for AD0 high

int16_t ax, ay, az;
int16_t gx, gy, gz;

#define OUTPUT_READABLE_ACCELGYRO

#define LED_PIN 13
bool blinkState = false;

void setup() {
    BT.begin(9600);

    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    // initialize serial communication
    // (38400 chosen because it works as well at 8MHz as it does at 16MHz, but
    // it's really up to you depending on your project)
    Serial.begin(115200);

    // initialize device
    BT.println("Initializing I2C devices...");
    accelgyro.initialize();

    // verify connection
    Serial.println("Testing device connections...");
    Serial.println(accelgyro.testConnection() ? "MPU6050 connection successful" : "MPU6050 connection
failed");

    // configure Arduino LED pin for output
    pinMode(LED_PIN, OUTPUT);
}

void loop() {
    // read raw accel/gyro measurements from device
    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

    // these methods (and a few others) are also available
    //accelgyro.getAcceleration(&ax, &ay, &az);
```

```
    //accelgyro.getRotation(&gx, &gy, &gz);


    #ifdef OUTPUT_BINARY_ACCELGYRO
        Serial.write((uint8_t)(ax >> 8)); Serial.write((uint8_t)(ax & 0xFF));
        Serial.write((uint8_t)(ay >> 8)); Serial.write((uint8_t)(ay & 0xFF));
        Serial.write((uint8_t)(az >> 8)); Serial.write((uint8_t)(az & 0xFF));
        Serial.write((uint8_t)(gx >> 8)); Serial.write((uint8_t)(gx & 0xFF));
        Serial.write((uint8_t)(gy >> 8)); Serial.write((uint8_t)(gy & 0xFF));
        Serial.write((uint8_t)(gz >> 8)); Serial.write((uint8_t)(gz & 0xFF));
    #endif
    float ax_g = ax / 16384.0;  // assuming ±2g range
    float ay_g = ay / 16384.0;
    float az_g = az / 16384.0;

    // Tilt angle around X (pitch), in degrees:
    float pitch = atan2(-ax_g, sqrt(ay_g*ay_g + az_g*az_g)) * 180.0 / PI;
    BT.print(pitch);
    BT.print("\t");
    BT.println(az_g);
    delay(20);
    // blink LED to indicate activity
    blinkState = !blinkState;
    digitalWrite(LED_PIN, blinkState);
}
```

Chapter 9 Appendix E— NAND logic (SN74HC00) design: truth table and schematic

This appendix provides the complete circuit schematic and table for Topic 3 (NAND-only PWM gating for L293D motor drive with encoder feedback).

9.1 Figure E.1 schematic of the NAND gating + L293D channel wiring



Key signals: Arduino D5 outputs PWM; Arduino D4 outputs CTL (direction). The SN74HC00 network generates Gate_F = PWM·NOT CTL and Gate_B = PWM·CTL, which are routed to the L293D channel inputs so that only one input is PWM-driven at a time. L293D VCC1 is supplied by +5 V logic, VCC2 by the motor supply (~7.2 V), and all grounds are common.

9.2 Table. E.2 Truth table for unipolar gating

PWM	CTL	Gate_F = PWM·¬CTL	Gate_B = PWM·CTL
0	0	0	0
1	0	1	0
0	1	0	0
1	1	0	1

Chapter 10 Appendix F— Motor drive verification code (PWM/CTL generation and speed checks)

This appendix lists the complete Arduino sketch used in Topic 3 to generate PWM/CTL signals and compute encoder-based speed and distance during logic verification.

```
// 引脚定义
#define PWM_PIN 5     // D5，PWM 输出
#define CTL_PIN 4     // D4，方向控制
#define ENC_A 2       // D2，编码器 A
#define ENC_B 3       // D3，编码器 B

// 参数设置
const float PULSE_PER_REV = 20.0;      // 编码器每圈脉冲数
const float WHEEL_DIAMETER = 6.5;      // 轮子直径（cm）
const float WHEEL_CIRCUMFERENCE = WHEEL_DIAMETER * 3.1416; // 轮子周长（cm）

volatile long encoderCount = 0;
volatile bool encoderLastA = LOW;
unsigned long lastCalcTime = 0;
long lastEncoderCount = 0;
float speed_cm_s = 0.0;
float distance_cm = 0.0;
int lastPwm = 0;

void setup() {
  pinMode(PWM_PIN, OUTPUT);
  pinMode(CTL_PIN, OUTPUT);
  pinMode(ENC_A, INPUT_PULLUP);
  pinMode(ENC_B, INPUT_PULLUP);

  Serial.begin(115200);
  attachInterrupt(digitalPinToInterrupt(ENC_A), encoderISR, CHANGE);
  encoderLastA = digitalRead(ENC_A);

  Serial.println("NAND 门电机系统初始化完成");
  lastCalcTime = millis();
}

void loop() {
  // 前进: CTL=0
  digitalWrite(CTL_PIN, LOW);
  sweepPWM();

  delay(500);

  // 后退: CTL=1
  digitalWrite(CTL_PIN, HIGH);
  sweepPWM();

  delay(500);

  updateSpeedDistance(); // 保证无论如何 100ms 周期更新一次速度
```

```
}

// PWM 扫频：从 0 到 255 再到 0
void sweepPWM() {
  for (int pwm = 0; pwm <= 255; pwm += 5) {
    analogWrite(PWM  PIN, pwm);
    lastPwm = pwm; // 记录最新 PWM 值
    updateSpeedDistance();
    delay(10);
  }
  for (int pwm = 255; pwm >= 0; pwm -= 5) {
    analogWrite(PWM  PIN, pwm);
    lastPwm = pwm;
    updateSpeedDistance();
    delay(10);
  }
  analogWrite(PWM_PIN, 0);
  lastPwm = 0;
}

// 编码器中断
void encoderISR() {
  bool A = digitalRead(ENC_A);
  bool B = digitalRead(ENC_B);

  if (A != encoderLastA) {
    encoderCount += (B != A) ? 1 : -1;
    encoderLastA = A;
  }
}

// 速度距离计算+串口输出
void updateSpeedDistance() {
  unsigned long now = millis();
  unsigned long timeDelta = now - lastCalcTime;
  if (timeDelta >= 100) {
    long countDelta = encoderCount - lastEncoderCount;
    float revDelta = countDelta / PULSE_PER_REV;
    float distanceDelta = revDelta * WHEEL_CIRCUMFERENCE;
    distance_cm += distanceDelta;
    speed_cm_s = distanceDelta * (1000.0 / timeDelta);

    lastEncoderCount = encoderCount;
    lastCalcTime = now;
    Serial.print("CTL=");
    Serial.print(digitalRead(CTL_PIN));
    Serial.print(" | PWM=");
    Serial.print(lastPwm);
    Serial.print(" | 速度=");
    Serial.print(speed_cm_s, 2);
    Serial.print(" cm/s | 距离=");
    Serial.print(distance  cm, 2);
    Serial.print(" cm | 编码器=");
    Serial.println(encoderCount);
  }
}
```

Chapter 11 Appendix G— LCD1602 bring-up and display test sketch

This appendix includes the code for session3 topic 4 (Remote HMI test)

```
#include <LiquidCrystal.h>

// -------- LCD1602 引脚 --------
const int LCD_RS = 2;
const int LCD_EN = 3;
const int LCD_D4 = 4;
const int LCD_D5 = 5;
const int LCD_D6 = 6;
const int LCD_D7 = 7;

LiquidCrystal lcd(LCD_RS, LCD_EN, LCD_D4, LCD_D5, LCD_D6, LCD_D7);

// -------- Joystick 引脚 --------
const int JOY_X_PIN = A0;   // VRx
const int JOY_Y_PIN = A1;   // VRy
// 如果暂时不用按键，SW 可以先不接线
// const int JOY_SW_PIN = 8;

const long SERIAL_BAUD = 9600;
const unsigned long REFRESH_MS = 100;
unsigned long lastUpdate = 0;

void setup() {
  Serial.begin(SERIAL_BAUD);

  // 如果要用按键就打开
  // pinMode(JOY_SW_PIN, INPUT_PULLUP);

  // 初始化 LCD
  lcd.begin(16, 2);
  lcd.clear();

  // 先写好固定标签
  lcd.setCursor(0, 0);
  lcd.print("X:    Y:    ");
  lcd.setCursor(0, 1);
  lcd.print("Joystick LCD");
}

void loop() {
  unsigned long now = millis();
  if (now - lastUpdate < REFRESH_MS) {
    return;
  }
  lastUpdate = now;

  // 1. 读取摇杆
  int xRaw = analogRead(JOY_X_PIN);  // 0~1023
  int yRaw = analogRead(JOY_Y_PIN);  // 0~1023
```

```
  // 2. 串口输出，确认数值正常
  Serial.print("X = ");
  Serial.print(xRaw);
  Serial.print("   Y = ");
  Serial.println(yRaw);

  // 3. 在 LCD 上显示 X
  lcd.setCursor(2, 0);   // "X:" 后面
  lcd.print("    ");    // 先刷掉旧数字（4 个空格）
  lcd.setCursor(2, 0);
  lcd.print(xRaw);      // 再打印新的数值

  // 4. 在 LCD 上显示 Y
  lcd.setCursor(9, 0);   // "Y:" 后面
  lcd.print("    ");    // 刷掉旧数字
  lcd.setCursor(9, 0);
  lcd.print(yRaw);
}
```

Chapter 12 Appendix H— Remote HMI build evidence (top/bottom soldering photos)

Fig. H.1 Top view of the assembled remote HMI

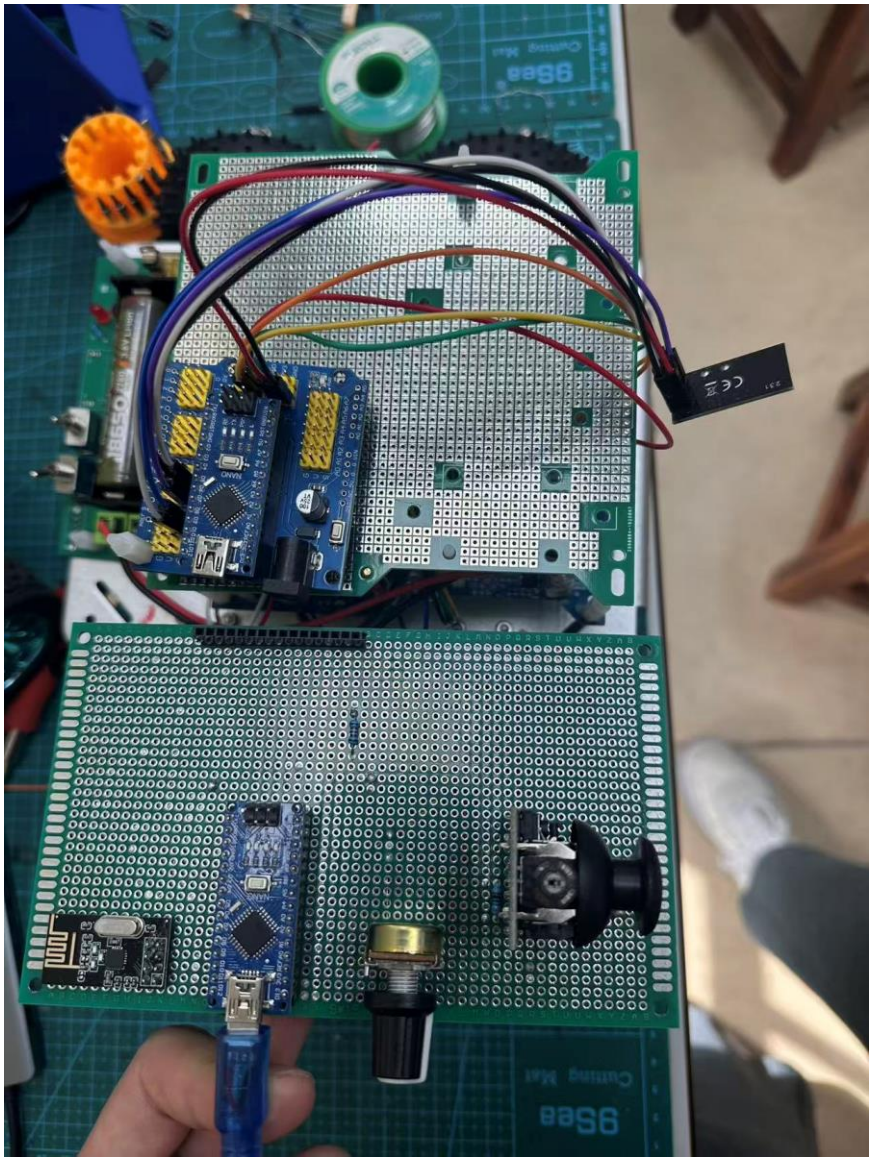
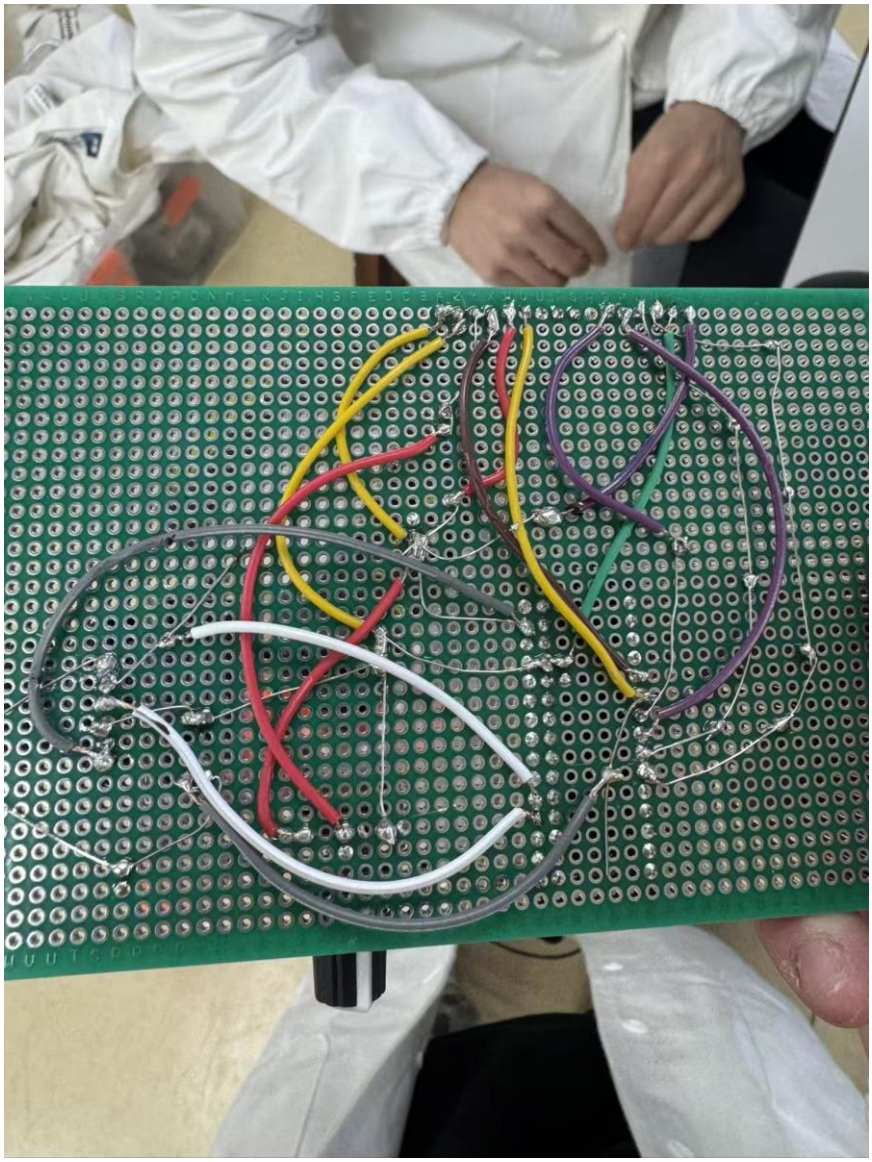


Fig. H.2 Underside of the remote HMI showing wiring and solder joints



Chapter 13 Appendix I— Session III challenge code (Task A state machine, Task B1/B3 remote control)

Appendix I summarizes the firmware code for Session III / Challenge tasks.

13.1 Listing I.1 Task A (Automatic path tracking with ramp) – code listing

```
float k = 0.45;// coefficient to control the offest
// larger k, small offset
// smaller k, large offset
int speed = 40;
int meter = 4;
unsigned int mval;// mval is converting to pitch
int state =  0;
//========parameter========

#include <SoftwareSerial.h>
SoftwareSerial BT(10, 11); // RX, TX


//========encoder data==========/

// each rotary for the wheel is 235 pitch

//diameter of the wheel is 21.7 cm

// 235 pitch per 0.217m
// 1 meter is 1083 pitch
#define encoder  shift 0x80000000UL

unsigned long e1_raw, e2_raw, e3_raw, e4_raw;

long cnt(unsigned long raw){
   return (long)raw - (long)encoder  shift;
}

unsigned int average;




float leftspeed;
float rightspeed;
unsigned int val;


//=================offest setup====================//

// Declare Variables
float G;
float G1 = 0;
float G2 = 0;
float G3 = 0;
```

```
float G4 = 0;
float G5 = 0;


float H;
float H1 = 0;
float H2 = 0;
float H3 = 0;
float H4 = 0;
float H5 = 0;


//=================offset setup====================//
#include "I2Cdev.h"
#include "MPU6050  6Axis  MotionApps20.h"

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
   #include "Wire.h"
#endif


MPU6050 mpu;
#define OUTPUT_READABLE_YAWPITCHROLL
#define INTERRUPT_PIN 2  // use pin 2 on Arduino Uno & most boards
#define LED  PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
bool blinkState = false;
// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer
// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]            gravity vector
float euler[3];         // [psi, theta, phi]    Euler angle container
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector
// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r', '\n' };



// ================================================================
// ===               zINTERRUPT DETECTION ROUTINE                 ===
// ================================================================

volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
   mpuInterrupt = true;
}
```

```
// ================================================================
// ===                      INITIAL SETUP                       ===
// ================================================================

void setup() {
     BT.begin(9600);

    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having compilation difficulties
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    // initialize serial communication
    // (115200 chosen because it is required for Teapot Demo output, but it's
    // really up to you depending on your project)
    Serial.begin(115200);


    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();
    pinMode(INTERRUPT_PIN, INPUT);

    // verify connection
    Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection
failed"));

    // wait for ready
    Serial.println(F("\nSend any character to begin DMP programming and demo: "));


        /*while (Serial.available() && Serial.read()); // empty buffer
        while (!Serial.available());                 // wait for data
        while (Serial.available() && Serial.read()); // empty buffer again
        */
    delay(2000);//wait for 2 seconds
    // load and configure the DMP
    Serial.println(F("Initializing DMP..."));
    devStatus = mpu.dmpInitialize();

    // supply your own gyro offsets here, scaled for min sensitivity
    mpu.setXGyroOffset(220);
    mpu.setYGyroOffset(76);
    mpu.setZGyroOffset(-85);
    mpu.setZAccelOffset(1788); // 1688 factory default for my test chip

    // make sure it worked (returns 0 if so)
    if (devStatus == 0) {
        // Calibration Time: generate offsets and calibrate our MPU6050
        mpu.CalibrateAccel(6);
        mpu.CalibrateGyro(6);
        mpu.PrintActiveOffsets();
        // turn on the DMP, now that it's ready
        Serial.println(F("Enabling DMP..."));
```

```
        mpu.setDMPEnabled(true);

        // enable Arduino interrupt detection
        Serial.print(F("Enabling interrupt detection (Arduino external interrupt "));
        Serial.print(digitalPinToInterrupt(INTERRUPT_PIN));
        Serial.println(F(")..."));
        attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();

        // set our DMP Ready flag so the main loop() function knows it's okay to use it
        Serial.println(F("DMP ready! Waiting for first interrupt..."));
        dmpReady = true;

        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();
    } else {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
    }

    // configure LED for output
    pinMode(LED  PIN, OUTPUT);
    mval = meter*1083;// set the preset distance
}
```

```
// ================================================================
// ===                MAIN PROGRAM LOOP                         ===
// ================================================================

void loop() {

    while(state !=4 )
    {
    updateIMU();
    updateEnc();
    calculate_offset();
    calculate_height();
    //car_control(leftspeed,rightspeed);
    car_control(leftspeed,rightspeed);
    BT.println("Straight");

    if(state == 3)
    {
      car_control(leftspeed,rightspeed);
      delay(500);
      car_contro0l(0,0);
```

```
      delay(500);
      car_control(0,50);
      BT.println("Rotate");
      delay(3170);
      car_control(50,50);
      BT.println("Straight");
      delay(2000);
      state = 4;
      BT.println("Stop");
      break;
    }
    }
    delay(2);
    car_control(0,0);


//==========stop==========
    average = (cnt(e1_raw)+cnt(e2_raw)+cnt(e3_raw)+cnt(e4_raw))/4;
    // Serial.println("average\t");
    val = -(average-65555);
    // Serial.println(val);


   /*if(val >= mval){// val is the data from the encoder, maval is the preset data.
    car_control(0,0);
}*/

//==========================
    delay(10);// 100Hz control frequency
}



void updateIMU(){

    // if programming failed, don't try to do anything
    if (!dmpReady) return;
    // read a packet from FIFO
    if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer)) { // Get the Latest packet
        #ifdef OUTPUT_READABLE_QUATERNION
            // display quaternion values in easy matrix form: w x y z
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            Serial.print("quat\t");
            Serial.print(q.w);
            Serial.print("\t");
            Serial.print(q.x);
            Serial.print("\t");
            Serial.print(q.y);
            Serial.print("\t");
            Serial.println(q.z);
        #endif

        #ifdef OUTPUT_READABLE_EULER
            // display Euler angles in degrees
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetEuler(euler, &q);
            Serial.print("euler\t");
```

```
    Serial.print(euler[0] * 180/M_PI);
    Serial.print("\t");
    Serial.print(euler[1] * 180/M_PI);
    Serial.print("\t");
    Serial.println(euler[2] * 180/M_PI);
#endif

#ifdef OUTPUT_READABLE_YAWPITCHROLL
    // display Euler angles in degrees
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
    Serial.print("ypr\t");
    Serial.print(ypr[0] * 180/M_PI);// this is yaw
    Serial.print("\t");
    // Serial.print(ypr[1] * 180/M_PI); // this is pitch
    // Serial.print("\t");
    // Serial.println(ypr[2] * 180/M_PI);// this is roll
#endif

#ifdef OUTPUT_READABLE_REALACCEL
    // display real acceleration, adjusted to remove gravity
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetAccel(&aa, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
    Serial.print("areal\t");
    Serial.print(aaReal.x);
    Serial.print("\t");
    Serial.print(aaReal.y);
    Serial.print("\t");
    Serial.println(aaReal.z);
#endif

#ifdef OUTPUT_READABLE_WORLDACCEL
    // display initial world-frame acceleration, adjusted to remove gravity
    // and rotated based on known orientation from quaternion
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetAccel(&aa, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
    mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
    Serial.print("aworld\t");
    Serial.print(aaWorld.x);
    Serial.print("\t");
    Serial.print(aaWorld.y);
    Serial.print("\t");
    Serial.println(aaWorld.z);
#endif

#ifdef OUTPUT_TEAPOT
    // display quaternion values in InvenSense Teapot demo format:
    teapotPacket[2] = fifoBuffer[0];
    teapotPacket[3] = fifoBuffer[1];
    teapotPacket[4] = fifoBuffer[4];
    teapotPacket[5] = fifoBuffer[5];
    teapotPacket[6] = fifoBuffer[8];
```

```
            teapotPacket[7] = fifoBuffer[9];
            teapotPacket[8] = fifoBuffer[12];
            teapotPacket[9] = fifoBuffer[13];
            Serial.write(teapotPacket, 14);
            teapotPacket[11]++; // packetCount, loops at 0xFF on purpose
        #endif

        // blink LED to indicate activity
        blinkState = !blinkState;
        digitalWrite(LED_PIN, blinkState);
    }
    delay(1);

}



//================car_control==============
void car_control(float a, float b){
//========methond A===========

  /*Wire.beginTransmission(42);  // begin to commuication with slave at 42 address
  Wire.write("baffff"); // set the direction of each whell, f representing forward, r representing backward
  // Motor1: right front
  Wire.write((int)rightspeed);    // speed
  // Motor2: right back
  Wire.write((int)leftspeed);
  // Motor3: left front
  Wire.write((int)rightspeed);
  // Motor4: left back
  Wire.write((int)leftspeed);
  Wire.endTransmission();*/
  //==========method B==========

  Wire.beginTransmission(42);  // 开始与地址 42 的从机通信
  Wire.write("sa");

  // Motor1: 右前
  Wire.write((int)b);    // 速度
  Wire.write(0);     // 方向（0=前进, 1=后退）
  // Motor2: 右后
  Wire.write((int)b);
  Wire.write(0);
  // Motor3: 左前
  Wire.write((int)a);
  Wire.write(0);
  // Motor4: 左后
  Wire.write((int)a);
  Wire.write(0);
  Wire.endTransmission();
  delay(1); // according to experience you need small delay 1ms after the end of each transmission

}


//================car_control==============
```

```
void car_control1(float a, float b){
//========methond A==========

  /*Wire.beginTransmission(42);  // begin to commuication with slave at 42 address
  Wire.write("baffff"); // set the direction of each whell, f representing forward, r representing backward
  // Motor1: right front
  Wire.write((int)rightspeed);    // speed
  // Motor2: right back
  Wire.write((int)leftspeed);
  // Motor3: left front
  Wire.write((int)rightspeed);
  // Motor4: left back
  Wire.write((int)leftspeed);
  Wire.endTransmission();*/
  //==========method B==========

  Wire.beginTransmission(42);  // 开始与地址 42 的从机通信
  Wire.write("sa");
  Wire.write("baffff");

  // Motor1: 右前
  Wire.write((int)b);    // 速度
  Wire.write(0);     // 方向（0=前进, 1=后退）
  // Motor2: 右后
  Wire.write((int)b);
  Wire.write(0);
  // Motor3: 左前
  Wire.write((int)a);
  Wire.write(0);
  // Motor4: 左后
  Wire.write((int)a);
  Wire.write(0);
  Wire.endTransmission();
  delay(1); // according to experience you need small delay 1ms after the end of each transmission
}


    void calculate  offset()
    {
    // Build 5 frame averager filter
    G5 = G4;
    G4 = G3;
    G3 = G2;
    G2 = G1;
    G1 = ypr[0] * 180/M_PI;
    // Average the last 5 readings
    G = (G1 + G2 + G3 + G4 + G5) / 5;
    leftspeed = speed - (G / k);
    rightspeed = speed + (G / k);
    Serial.println("left speed\t");
    Serial.println(leftspeed);
    Serial.println("right speed\t");
    Serial.println(rightspeed);
    delay(1);
    }
```

```
    void calculate_height()
    {
    // Build 5 frame averager filter
    H5 = H4;
    H4 = H3;
    H3 = H2;
    H2 = H1;
    H1 = ypr[1] * 180/M_PI;
    // Average the last 5 readings
    H = (H1 + H2 + H3 + H4 + H5) / 5;
    Serial.println("height\t");
    Serial.println(H);
    delay(1);
    if (state == 0 && H > 10) {
       state = 1;
       Serial.println(">>>> ENTER UPHILL");
       BT.println("UPHILL");

    }
    // uphill -> ground
    if (state == 1 && H < 10) {
       state = 3;
       Serial.println("<<<< BACK TO FLAT");
    }
    }


//=========encoder==========//
void updateEnc(){
    Wire.beginTransmission(42);
    Wire.write("i0");
    Wire.endTransmission();
    delay(1);

    Wire.requestFrom(42,8);
    e1_raw = (unsigned long)Wire.read();
    e1_raw += (unsigned long)Wire.read()<<8;
    e1_raw += (unsigned long)Wire.read()<<16;
    e1_raw += (unsigned long)Wire.read()<<24;

    e2_raw = (unsigned long)Wire.read();
    e2_raw += (unsigned long)Wire.read()<<8;
    e2_raw += (unsigned long)Wire.read()<<16;
    e2_raw += (unsigned long)Wire.read()<<24;


    Wire.beginTransmission(42);
    Wire.write("i5");
    Wire.endTransmission();
    delay(1);

    Wire.requestFrom(42,8);
    e3_raw = (unsigned long)Wire.read();
    e3_raw += (unsigned long)Wire.read()<<8;
    e3_raw += (unsigned long)Wire.read()<<16;
    e3_raw += (unsigned long)Wire.read()<<24;
```

```
    e4_raw = (unsigned long)Wire.read();
    e4_raw += (unsigned long)Wire.read()<<8;
    e4  raw += (unsigned long)Wire.read()<<16;
    e4  raw += (unsigned long)Wire.read()<<24;
}
```

13.2 Task B1 (nRF24L01+ joystick remote control)

13.2.1 Listing I.2 Remote (TX): Joystick → Nano → nRF24 (sends {L, R, buttons} at ~20 ms)

```
// ====== Task B1 TX: Joystick + LCD1602 + nRF24L01+ (Bi-directional) ======
#include <SPI.h>
#include <RF24.h>
#include <LiquidCrystal.h>

// ========== LCD1602 引脚 ==========
const int LCD_RS = 2;
const int LCD_EN = 3;
const int LCD  D4 = 4;
const int LCD  D5 = 5;
const int LCD_D6 = 6;
const int LCD_D7 = 7;
LiquidCrystal lcd(LCD_RS, LCD_EN, LCD_D4, LCD_D5, LCD_D6, LCD_D7);

// ========== Joystick 引脚 ==========
const int JOY_X_PIN = A0;   // VRx
const int JOY_Y_PIN = A1;   // VRy

// ========== nRF24L01+ ==========
// CE -> D9, CSN -> D10, SCK -> D13, MOSI -> D11, MISO -> D12
RF24 radio(9, 10);

// 和车端保持完全一致
const byte pipeAddr[6] = "GR18";    // Group18
const uint8_t GROUP   = 18;
const uint8_t CHANNEL = 129 - 2 * GROUP;  // => 93

// === 下行：手柄 -> 车，左右轮指令包 ===
struct PacketLR {
  int16_t L;     // 左轮 -255..+255
  int16_t R;     // 右轮 -255..+255
  uint8_t buttons;
};

// === 上行：车 -> 手柄，状态包（run/stop + speed + distance） ===
struct StatusPacket {
  uint8_t running;   // 1 = 车在运动, 0 = 停止
  uint8_t speed;     // 0..100 (平均速度)
  uint32_t distance; // 距离计数（单位可视为 "steps"）
};

const unsigned long REFRESH_MS = 50;
```

```
unsigned long lastUpdate = 0;

bool   haveStatus       = false;
StatusPacket lastStatus;

// 最近是否有 ACK（链路指示）
bool lastLinkOK = false;
unsigned long lastLinkOKTime = 0;
const unsigned long LINK_TIMEOUT_MS = 500;

int16  t clamp255(long v) {
  if (v > 255)  return 255;
  if (v < -255) return -255;
  return (int16_t)v;
}

// 辅助：固定 4 位宽打印整数（不够前面补空格）
void lcdPrint4(int v) {
  if (v < 0) {
    lcd.print('-');
    v = -v;
  } else {
    lcd.print(' ');
  }
  if (v < 1000) lcd.print(' ');
  if (v < 100)  lcd.print(' ');
  if (v < 10)   lcd.print(' ');
  lcd.print(v);
}

// 辅助：固定 3 位宽打印（速度）
void lcdPrint3(int v) {
  if (v < 100) lcd.print(' ');
  if (v < 10)  lcd.print(' ');
  lcd.print(v);
}

// 辅助：固定 5 位宽打印（距离）
void lcdPrint5(uint32_t v) {
  if (v < 10000) lcd.print(' ');
  if (v < 1000)  lcd.print(' ');
  if (v < 100)   lcd.print(' ');
  if (v < 10)    lcd.print(' ');
  lcd.print(v);
}

void setup() {
  Serial.begin(115200);

  // ---- LCD ----
  lcd.begin(16, 2);
  lcd.clear();
  lcd.setCursor(0, 0); lcd.print("X:    Y:    R");  // 最后一列 R/S
  lcd.setCursor(0, 1); lcd.print("V:   D:     ");   // 速度 + 距离

  // ---- nRF24 ----
```

```
  if (!radio.begin()) {
    Serial.println("ERROR: nRF24 not responding on TX!");
    while (1);
  }
  radio.setChannel(CHANNEL);
  radio.setPALevel(RF24_PA_LOW);
  radio.setDataRate(RF24_1MBPS);

  // 打开自动应答 + ACK Payload，用于双向通信
  radio.setAutoAck(true);
  radio.enableDynamicPayloads();
  radio.enableAckPayload();

  radio.openWritingPipe(pipeAddr);   // 手柄只发
  radio.stopListening();

  Serial.print("TX ready, channel = ");
  Serial.println(CHANNEL);
}

void loop() {
  unsigned long now = millis();
  if (now - lastUpdate < REFRESH_MS) return;
  lastUpdate = now;

  // 1. 读取摇杆输入
  int xRaw = analogRead(JOY_X_PIN);   // 0..1023
  int yRaw = analogRead(JOY_Y_PIN);

  int xCentered = xRaw - 512;
  int yCentered = 512 - yRaw;         // 上推为正

  const int deadZone = 40;
  if (abs(xCentered) < deadZone) xCentered = 0;
  if (abs(yCentered) < deadZone) yCentered = 0;

  // 2. 映射至油门 + 转向，再变成 L/R
  int16_t throttle = clamp255((long)yCentered * 255 / 512);
  int16_t steer    = clamp255((long)xCentered * 255 / 512);

  int16_t L = clamp255((long)throttle + (long)steer / 2);
  int16_t R = clamp255((long)throttle - (long)steer / 2);

  // 3. 下行：发给车端
  PacketLR pkt;
  pkt.L = L;
  pkt.R = R;
  pkt.buttons = 0;

  bool ok = radio.write(&pkt, sizeof(pkt));  // 有没有收到 ACK
  if (ok) {
    lastLinkOK = true;
    lastLinkOKTime = now;
  }

  // 4. 尝试读取 ACK Payload（车端返回的状态）
```

```
if (radio.isAckPayloadAvailable()) {
  uint8_t len = radio.getDynamicPayloadSize();
  if (len == sizeof(StatusPacket)) {
    radio.read(&lastStatus, len);
    haveStatus = true;
  } else {
    // 长度不匹配就丢掉
    uint8  t dummy[32];
    radio.read(dummy, len);
  }
}

// 5. 链路超时判定
if (lastLinkOK && (now - lastLinkOKTime > LINK_TIMEOUT_MS)) {
  lastLinkOK = false;
}

// 6. 串口调试输出
Serial.print("TX X="); Serial.print(xRaw);
Serial.print(" Y=");    Serial.print(yRaw);
Serial.print(" L=");    Serial.print(L);
Serial.print(" R=");    Serial.print(R);
if (haveStatus) {
  Serial.print(" | run=");
  Serial.print(lastStatus.running);
  Serial.print(" v=");
  Serial.print(lastStatus.speed);
  Serial.print(" d=");
  Serial.print(lastStatus.distance);
}
Serial.println(lastLinkOK ? " [Link]" : " [NoLink]");

// 7. 更新 LCD 显示
// 第一行：X, Y, run/stop
lcd.setCursor(2, 0);  // X:
lcdPrint4(xRaw);

lcd.setCursor(9, 0);  // Y:
lcdPrint4(yRaw);

lcd.setCursor(15, 0);
if (haveStatus && lastStatus.running) {
  lcd.print('R');   // Run
} else {
  lcd.print('S');   // Stop
}

// 第二行：V:xxx D:xxxxx （来自车端的状态）
lcd.setCursor(2, 1);
if (haveStatus) {
  lcdPrint3(lastStatus.speed);
} else {
  lcd.print("---");
}

lcd.setCursor(7, 1);
```

```
  if (haveStatus) {
    lcdPrint5(lastStatus.distance);
  } else {
    lcd.print("-----");
  }
}
```

13.2.2 Listing I.3 Car (RX): nRF24 → Nano → I2C baseboard (drives motors), watchdog stops if timeout

```
// ====== Task B1 RX: nRF24L01+ -> Nano -> Motor board (bi-directional) ======
#include <SPI.h>
#include <RF24.h>
#include <Wire.h>

// ---------- nRF24L01+ ----------
const uint8_t PIN_CE  = 9;
const uint8_t PIN_CSN = 10;
RF24 radio(PIN_CE, PIN_CSN);

// 与 TX 完全一致
const byte pipeAddr[6] = "GR18";
const uint8_t GROUP   = 18;
const uint8_t CHANNEL = 129 - 2 * GROUP;   // 93

// 手柄下行指令
struct PacketLR {
  int16_t L;      // -255..+255
  int16_t R;      // -255..+255
  uint8_t buttons;
};

// 车端上行状态
struct StatusPacket {
  uint8_t  running;   // 1 or 0
  uint8_t  speed;     // 0..100
  uint32_t distance;  // 距离计数
};

// ---------- I2C Motor control (from B3) ----------
constexpr uint8_t MOTOR_ADDR      = 0x2A;
constexpr int8_t  WHEEL_SENSE[4]  = {-1, 1, 1, -1};
constexpr int8_t  LINEAR_TRIM[4]  = {-2, 6, -2, 6};
constexpr char    CMD_SET_SPEED_DIR = 'b';
constexpr char    CMD_HALT_ALL      = 'h';
constexpr char    TARGET_ALL        = 'a';

// Link 指示 LED（Nano 板载 D13）
const int LINK_LED_PIN = 13;

static inline int16_t clamp100(int v) {
  if (v > 100)  return 100;
  if (v < -100) return -100;
  return v;
}
```

```
static inline void writeUint16LE(uint16_t x) {
  Wire.write((uint8_t)(x & 0xFF));
  Wire.write((uint8_t)(x >> 8));
}

void driveAll(int m1, int m2, int m3, int m4) {
  int req[4] = {m1, -m2, -m3, m4};
  char dir[4];
  uint16_t mag[4];

  for (uint8_t i = 0; i < 4; i++) {
    int v = req[i];
    v += (v >= 0) ? LINEAR_TRIM[i] : -LINEAR_TRIM[i];
    v = clamp100(v);
    int adj = clamp100(v * WHEEL_SENSE[i]);
    dir[i] = (adj >= 0) ? 'f' : 'r';
    mag[i] = (uint16_t)(adj >= 0 ? adj : -adj);
  }

  Wire.beginTransmission(MOTOR_ADDR);
  Wire.write(CMD_SET_SPEED_DIR);
  Wire.write(TARGET_ALL);
  // 顺序与 B3 一致：0,2,1,3
  Wire.write(dir[0]); Wire.write(dir[2]); Wire.write(dir[1]); Wire.write(dir[3]);
  writeUint16LE(mag[0]); writeUint16LE(mag[2]); writeUint16LE(mag[1]); writeUint16LE(mag[3]);
  Wire.endTransmission();
}

void stopAll() {
  Wire.beginTransmission(MOTOR_ADDR);
  Wire.write(CMD_HALT_ALL);
  Wire.write(TARGET_ALL);
  Wire.endTransmission();
}

// L/R [-255..255] -> 四轮 [-100..100] 并返回速度(0..100)
uint8_t applyLR_and_getSpeed(int16_t L, int16_t R) {
  int left  = clamp100((int)(L * 100L / 255L));
  int right = clamp100((int)(R * 100L / 255L));

  int m1 = left;
  int m3 = left;
  int m2 = right;
  int m4 = right;

  driveAll(m1, m2, m3, m4);

  uint8_t speed = (uint8_t)((abs(left) + abs(right)) / 2);
  return speed;
}

// 状态与看门狗
unsigned long lastPktTime = 0;
const unsigned long WATCHDOG_MS = 200;
uint32_t distanceCounter = 0;

void setup() {
```

```
  pinMode(LINK_LED_PIN, OUTPUT);
  digitalWrite(LINK_LED_PIN, LOW);

  Serial.begin(115200);
  delay(2000);
  Serial.println("Task B1 RX: motors + link LED + status feedback");

  Serial.println("Wire.begin()");
  Wire.begin();
  stopAll();
  Serial.println("Wire & motors init done.");

  Serial.print("radio.begin() on channel ");
  Serial.println(CHANNEL);
  bool ok = radio.begin();
  Serial.print("radio.begin() = ");
  Serial.println(ok ? "OK" : "FAIL");
  if (!ok) {
    Serial.println("ERROR: nRF24 not responding on RX! 进入死循环闪灯。");
    while (1) {
      digitalWrite(LINK_LED_PIN, !digitalRead(LINK_LED_PIN));
      delay(300);
    }
  }

  Serial.println("Config radio...");
  radio.setChannel(CHANNEL);
  radio.setPALevel(RF24_PA_LOW);
  radio.setDataRate(RF24_1MBPS);
  radio.setAutoAck(true);
  radio.enableDynamicPayloads();
  radio.enableAckPayload();
  radio.openReadingPipe(1, pipeAddr);
  radio.startListening();
  Serial.println("Radio config done.");

  lastPktTime = millis();
}

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

  if (radio.available()) {
    PacketLR pkt;
    while (radio.available()) {
      radio.read(&pkt, sizeof(pkt));  // 取最新包
    }
    lastPktTime = now;

    // 1. 用 L/R 控制电机
    uint8_t speed = applyLR_and_getSpeed(pkt.L, pkt.R);
    bool running  = (speed > 5);

    if (running) {
      distanceCounter += speed;   // 简单积分 demo
    }
```

```
  // 2. 点亮 link LED
  digitalWrite(LINK_LED_PIN, HIGH);

  // 3. 串口调试
  Serial.print("RX L="); Serial.print(pkt.L);
  Serial.print(" R="); Serial.print(pkt.R);
  Serial.print(" | run="); Serial.print(running);
  Serial.print(" v="); Serial.print(speed);
  Serial.print(" d="); Serial.println(distanceCounter);

  // 4. 准备 ACK payload（回传给手柄）
  StatusPacket st;
  st.running = running ? 1 : 0;
  st.speed = speed;
  st.distance = distanceCounter;
  radio.writeAckPayload(1, &st, sizeof(st));
 }

 // 看门狗：超时停车 + 熄灯
 if (now - lastPktTime > WATCHDOG_MS) {
  stopAll();
  digitalWrite(LINK_LED_PIN, LOW);
 }
}
```

13.3 Listing I.4 – Task B3 (HC-06 Bluetooth remote with cellphone)

```
#include <Wire.h>
#include <SoftwareSerial.h>

// ====== Pins ======
#define LED_PIN      13      // 状态灯（可以用板载 LED）
#define BUZZER_PIN   8       // 蜂鸣器输出

// HC-06 软串口引脚（不要和 I2C 冲突）
#define BT_RX_PIN    A2      // Arduino 接收 → 连接 HC-06 TXD
#define BT_TX_PIN    A3      // Arduino 发送 → 连接 HC-06 RXD
SoftwareSerial BT(BT_RX_PIN, BT_TX_PIN);

// ====== 小车参数（沿用 Session2） ======
int steerForward  = 5;       // 前进方向修正
int steerBackward = -5;      // 后退方向修正
int speedValue    = 50;      // 当前速度百分比 1~100

// ====== I2C 电机常量（与原代码保持一致） ======
constexpr uint8_t MOTOR_ADDR = 0x2A;
constexpr int8_t  WHEEL_SENSE[4] = {-1, 1, 1, -1};
constexpr int8_t  LINEAR_TRIM[4] = {-2, 6, -2, 6};
constexpr char    CMD_SET_SPEED_DIR = 'b';
constexpr char    CMD_HALT_ALL      = 'h';
constexpr char    TARGET_ALL        = 'a';

// ====== 驱动状态 ======
enum DriveMode {
```

```
  DRIVE_STOP,
  DRIVE_FWD,
  DRIVE_BACK,
  DRIVE_LEFT,
  DRIVE_RIGHT
};

DriveMode driveMode = DRIVE_STOP;
bool isRunning = false;

// 指令超时保护
unsigned long lastCmdTime = 0;
const unsigned long cmdTimeout = 1000;   // 1s 无指令自动停止

// ====== 蜂鸣器控制 ======
unsigned long buzzerStart = 0;
unsigned long buzzerDuration = 0;
bool buzzerActive = false;

void playBuzzer(int freq, int duration) {
  tone(BUZZER_PIN, freq, duration);
  buzzerStart = millis();
  buzzerDuration = duration;
  buzzerActive = true;
}

void updateBuzzer() {
  if (buzzerActive && millis() - buzzerStart >= buzzerDuration) {
    noTone(BUZZER_PIN);
    buzzerActive = false;
  }
}

// ====== 工具函数：-100~100 限幅 ======
static inline int16_t clamp100(int v) {
  if (v > 100)  return 100;
  if (v < -100) return -100;
  return v;
}

static inline void writeUint16LE(uint16_t x) {
  Wire.write((uint8_t)(x & 0xFF));
  Wire.write((uint8_t)(x >> 8));
}

// ====== I2C 驱动所有轮子（与你原来一样） ======
void driveAll(int m1, int m2, int m3, int m4) {
  int req[4] = {m1, -m2, -m3, m4};
  char dir[4];
  uint16_t mag[4];

  for (uint8_t i = 0; i < 4; i++) {
    int v = req[i];
    v += (v >= 0) ? LINEAR_TRIM[i] : -LINEAR_TRIM[i];
    v = clamp100(v);
    int adj = clamp100(v * WHEEL_SENSE[i]);
```

```
    dir[i] = (adj >= 0) ? 'f' : 'r';
    mag[i] = (uint16_t)(adj >= 0 ? adj : -adj);
  }

  Wire.beginTransmission(MOTOR_ADDR);
  Wire.write(CMD_SET_SPEED_DIR);
  Wire.write(TARGET_ALL);
  // 注意轮子顺序：0,2,1,3（和你原程序一致）
  Wire.write(dir[0]);
  Wire.write(dir[2]);
  Wire.write(dir[1]);
  Wire.write(dir[3]);
  writeUint16LE(mag[0]);
  writeUint16LE(mag[2]);
  writeUint16LE(mag[1]);
  writeUint16LE(mag[3]);
  Wire.endTransmission();
}

void stopAll() {
  Wire.beginTransmission(MOTOR_ADDR);
  Wire.write(CMD_HALT_ALL);
  Wire.write(TARGET_ALL);
  Wire.endTransmission();
}

// ====== 基本运动模式 ======
void driveStraight(bool forward) {
  int base   = speedValue;   // 1~100
  int offset = forward ? steerForward : steerBackward;

  int m1, m2, m3, m4;
  if (offset > 0) {
    m1 = base;  m3 = base;
    m2 = base - offset;  m4 = base - offset;
  } else if (offset < 0) {
    m1 = base + offset; m3 = base + offset;
    m2 = base;          m4 = base;
  } else {
    m1 = m2 = m3 = m4 = base;
  }

  if (!forward) {
    m1 = -m1; m2 = -m2; m3 = -m3; m4 = -m4;
  }

  driveAll(m1, m2, m3, m4);
}

// 原地左转
void driveSpinLeft() {
  int base = clamp100(speedValue);
  driveAll(-base, base, -base, base);
}

// 原地右转
void driveSpinRight() {
```

```
  int base = clamp100(speedValue);
  driveAll(base, -base, base, -base);
}

// 根据当前模式更新电机输出
void applyDriveMode() {
  switch (driveMode) {
    case DRIVE_STOP:
      stopAll();
      isRunning = false;
      digitalWrite(LED_PIN, LOW);
      break;
    case DRIVE_FWD:
      driveStraight(true);
      isRunning = true;
      digitalWrite(LED_PIN, HIGH);
      break;
    case DRIVE_BACK:
      driveStraight(false);
      isRunning = true;
      digitalWrite(LED_PIN, HIGH);
      break;
    case DRIVE_LEFT:
      driveSpinLeft();
      isRunning = true;
      digitalWrite(LED_PIN, HIGH);
      break;
    case DRIVE_RIGHT:
      driveSpinRight();
      isRunning = true;
      digitalWrite(LED_PIN, HIGH);
      break;
  }
}

// ======= 处理蓝牙收到的单字符指令 =======
void handleBT(char cmd) {
  switch (cmd) {
    case 'F':   // 前进
      driveMode = DRIVE_FWD;
      playBuzzer(1000, 120);
      break;

    case 'B':   // 后退
      driveMode = DRIVE_BACK;
      playBuzzer(800, 120);
      break;

    case 'L':   // 左转（原地）
      driveMode = DRIVE_LEFT;
      playBuzzer(600, 120);
      break;

    case 'R':   // 右转（原地）
      driveMode = DRIVE_RIGHT;
      playBuzzer(1200, 120);
```

```
      break;

    case 'S':   // 停车
      driveMode = DRIVE_STOP;
      playBuzzer(400, 120);
      break;

    // 三档速度：1 / 2 / 3
    case '1':
      speedValue = 40;
      playBuzzer(900, 80);
      break;
    case '2':
      speedValue = 60;
      playBuzzer(1100, 80);
      break;
    case '3':
      speedValue = 80;
      playBuzzer(1300, 80);
      break;

    default:
      // 其他字符忽略
      break;
  }

  lastCmdTime = millis();   // 更新最后指令时间
}

// ====== Setup ======
void setup() {
  Wire.begin();
  stopAll();

  pinMode(LED_PIN, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);

  Serial.begin(115200);   // PC 调试用
  BT.begin(9600);         // HC-06 默认 9600
  Serial.println("Task B-3 Bluetooth Remote ready.");
}

// ====== Loop ======
void loop() {
  // 1. 处理蓝牙指令
  if (BT.available()) {
    char c = BT.read();
    Serial.print("BT cmd: ");
    Serial.println(c);
    handleBT(c);
  }

  // 2. 超时自动停车（防止掉线后车失控）
  if (driveMode != DRIVE_STOP && (millis() - lastCmdTime > cmdTimeout)) {
    driveMode = DRIVE  STOP;
  }
```

```
  // 3. 根据当前模式驱动电机 & 更新蜂鸣器
  applyDriveMode();
  updateBuzzer();
}
```

Chapter 14 Appendix J— Two-sensor line following firmware (TCRT5000 ×2 + PID)

This Arduino sketch implements line following using two TCRT5000 analog signals on **A6 (left)** and **A7 (right)**. The control error is computed as left − right and fed into a discrete PID controller with anti-windup (integral constrained to ±500). The PID output modulates differential wheel speeds about a base speed (speed = 20, range 0–100) using leftSpeed = speed − u and rightSpeed = speed + u. Motor commands are transmitted via **I²C** (Wire) to the motor baseboard at address **42**, using a "sa" command header followed by right-side and left-side speed bytes for the four motors. A Bluetooth HMI is provided through SoftwareSerial BT(10,11) to adjust gains online using commands Pxx, Ixx, Dxx (values scaled by 1/100). The firmware also streams debug telemetry (PID output, left/right speeds, and raw sensor readings) via Bluetooth for tuning and verification.

Listing J.1.Two-sensor line-following with analog PID and Bluetooth tuning (topic2_code_PID.ino)

```
#include <SoftwareSerial.h>
#include <Wire.h>
// P 75 I6  D1
// ================== 蓝牙 ==================
SoftwareSerial BT(10, 11); // TX, RX

// ================== 速度设置 ==================
int speed = 20;      // 基础直行速度（0-100）

float leftspeed, rightspeed;

// ================== PID 参数 ==================
float Kp = 0.75;
float Ki = 0.06;
float Kd = 0.01;

float error = 0, lastError = 0, integral = 0, derivative = 0;
float pid_output = 0;

// ================== 蓝牙命令缓冲 ==================
String bt_cmd = "";

// ================== 初始化 ==================
void setup() {
  Serial.begin(9600);
  BT.begin(9600);

  pinMode(A6, INPUT);   // 左红外
  pinMode(A7, INPUT);   // 右红外

  Wire.begin();
}

// ================== 蓝牙读取 PID ==================
void readPIDfromBT() {
```

```
  while (BT.available()) {
    char c = BT.read();

    if (c == '\n' || c == '\r') {
      bt_cmd.trim();
      if (bt_cmd.length() >= 2) {
        char type = bt_cmd.charAt(0);
        float value = bt  cmd.substring(1).toFloat();
        bool updated = false;

        if (type == 'P') { Kp = value / 100.0; updated = true; }
        if (type == 'I') { Ki = value / 100.0; updated = true; }
        if (type == 'D') { Kd = value / 100.0; updated = true; }

        if (updated) {
          String out = "PID => P=" + String(Kp, 3) +
                       " I=" + String(Ki, 3) +
                       " D=" + String(Kd, 3);
          BT.println(out);
          Serial.println(out);
        }
      }
      bt_cmd = "";
    } else {
      bt_cmd += c;  // 累积命令
    }
  }
}

// ================= 主循环 =================
void loop() {
  // 1. 蓝牙非阻塞调 PID
  readPIDfromBT();

  // 2. 读取红外并归一化
  float left  = analogRead(A6);
  float right = analogRead(A7);

  // 3. 计算误差
  error = left - right;

  // 4. PID 计算
  integral += error;
  integral = constrain(integral, -500, 500); // 积分限幅
  derivative = error - lastError;
  pid_output = Kp * error + Ki * integral + Kd * derivative;
  lastError = error;

  // 5. PID → 左右轮速度
  leftspeed  = speed - pid  output;
  rightspeed = speed + pid  output;

  leftspeed  = constrain(leftspeed, 0, 100);
  rightspeed = constrain(rightspeed, 0, 100);

  // 6. 调用电机控制
```

```
  car_control1(leftspeed, rightspeed);
  BT.println("PID_output");
  BT.println(pid  output);
  BT.println("leftspeed");
  BT.println(leftspeed);
  BT.println("rightspeed");
  BT.println(rightspeed);
  BT.println("Leftread");
  BT.println(left);
  BT.println("Rightread");
  BT.println(right);

  delay(100);
}

// ================== 电机控制（原样保留） ==================
void car  control1(float a, float b) {
  Wire.beginTransmission(42);
  Wire.write("sa");

  // Motor1: 右前
  Wire.write((int)b);
  Wire.write(0);

  // Motor2: 右后
  Wire.write((int)b);
  Wire.write(0);

  // Motor3: 左前
  Wire.write((int)a);
  Wire.write(0);

  // Motor4: 左后
  Wire.write((int)a);
  Wire.write(0);

  Wire.endTransmission();
  delay(1);
}
```

Chapter 15 Appendix K— 8-sensor array firmware (SF-8CHIDTS): scanner, calibration, position, PID + HMI

This appendix includes the Source codes about Topic 3 ------ SF-8CHIDTS bring-up and 8-sensor PID line following

Appendix K.1 – Step1: I²C scanner (sensor address verification)

```
#include <Wire.h>

void setup() {
  Wire.begin();
  Serial.begin(9600);
  while (!Serial);   // 对部分板子安全
  Serial.println("\nI2C Scanner");
}

void loop() {
  byte error, address;
  int nDevices = 0;

  Serial.println("Scanning...");

  for (address = 1; address < 127; address++) {
    Wire.beginTransmission(address);
    error = Wire.endTransmission();

    if (error == 0) {
      Serial.print("I2C device found at address 0x");
      if (address < 16)
        Serial.print("0");
      Serial.print(address, HEX);
      Serial.println(" !");
      nDevices++;
    }
  }

  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    Serial.println("Done.\n");

  delay(2000);
}
```

Appendix K.2 – Step2: SF-8CHIDTS raw read test (8 channels streaming)

```
#include <Wire.h>

#define SENSOR_ADDR 0x09
#define NUM  SENSORS 8
```

```
uint16_t raw[NUM_SENSORS];

bool readSF8(uint16  t out[NUM  SENSORS]) {
  const uint8  t nBytes = 16;            // 8 sensors * 2 bytes
  uint8_t buf[nBytes];

  Wire.requestFrom(SENSOR_ADDR, nBytes);
  if (Wire.available() != nBytes) return false;

  for (int i = 0; i < nBytes; i++) {
    buf[i] = Wire.read();
  }

  // Combine bytes: assume [hi, lo] for each sensor
  for (int i = 0; i < NUM_SENSORS; i++) {
    uint8_t hi = buf[2*i];
    uint8_t lo = buf[2*i + 1];
    out[i] = ((uint16_t)hi << 8) | lo;
  }
  return true;
}

void setup() {
  Wire.begin();
  Serial.begin(115200);
  delay(200);
  Serial.println("SF-8CHIDTS raw read test");
}

void loop() {
  if (readSF8(raw)) {
    for (int i = 0; i < NUM  SENSORS; i++) {
      Serial.print(raw[i]);
      if (i < NUM_SENSORS - 1) Serial.print('\t');
    }
    Serial.println();
  } else {
    Serial.println("Read failed (bytes not available)");
  }
  delay(50); // 20 Hz for debug; later you will speed up
}
```

Appendix K.3 – Step3: Calibration + scaling + weighted position debug

```
#include <Wire.h>

#define SENSOR_ADDR 0x09
#define NUM  SENSORS 8
#define SCALE_MAX   1000

// ====== 丢线阈值（先给一个保守默认，后面你可按实际 sum 调）======
#define SUM_LOST_TH 200   // 若 sumScaled < 200 -> assume line lost

uint16_t raw[NUM_SENSORS];
uint16_t whiteRef[NUM_SENSORS];   // 白底参考（取 max）
```

```
uint16_t blackRef[NUM_SENSORS];  // 黑线参考（取 min）

float lastPos = 0.0;             // 用于丢线时保留上一次位置

// ====== 读 8 通道 raw（16 bytes）======
bool readSF8(uint16_t out[NUM_SENSORS]) {
  const uint8_t nBytes = 16;
  uint8_t buf[nBytes];

  Wire.requestFrom(SENSOR_ADDR, nBytes);
  if (Wire.available() != nBytes) return false;

  for (int i = 0; i < nBytes; i++) buf[i] = Wire.read();

  // 保持你 Step2 已验证过的字节顺序
  for (int i = 0; i < NUM_SENSORS; i++) {
    uint8_t hi = buf[2 * i];
    uint8_t lo = buf[2 * i + 1];
    out[i] = ((uint16_t)hi << 8) | lo;
  }
  return true;
}

// ====== 默认校准 ======
void setDefaultCalibration() {
  for (int i = 0; i < NUM_SENSORS; i++) {
    whiteRef[i] = 30000;  // 你实测白底量级
    blackRef[i] = 700;    // 你实测黑线量级
  }
}

// ====== 校准采集（2 秒）：白底取 max，黑线取 min ======
void captureCalibration(bool captureWhite, uint16_t ms = 2000) {
  unsigned long t0 = millis();

  uint16_t minv[NUM_SENSORS];
  uint16_t maxv[NUM_SENSORS];
  for (int i = 0; i < NUM_SENSORS; i++) {
    minv[i] = 65535;
    maxv[i] = 0;
  }

  while (millis() - t0 < ms) {
    if (!readSF8(raw)) continue;

    for (int i = 0; i < NUM_SENSORS; i++) {
      if (raw[i] < minv[i]) minv[i] = raw[i];
      if (raw[i] > maxv[i]) maxv[i] = raw[i];
    }
    delay(5);
  }

  if (captureWhite) {
    for (int i = 0; i < NUM_SENSORS; i++) whiteRef[i] = maxv[i];
    Serial.println("☑ White calibration captured (using MAX on white).");
  } else {
```

```
    for (int i = 0; i < NUM_SENSORS; i++) blackRef[i] = minv[i];
    Serial.println("☑ Black calibration captured (using MIN on black).");
  }
}

void printCalibration() {
  Serial.println("\nCalibration table:");
  Serial.println("ch\twhiteRef(max)\tblackRef(min)\tdiff(white-black)");
  for (int i = 0; i < NUM_SENSORS; i++) {
    Serial.print(i); Serial.print('\t');
    Serial.print(whiteRef[i]); Serial.print('\t');
    Serial.print(blackRef[i]); Serial.print('\t');
    Serial.println((int)whiteRef[i] - (int)blackRef[i]);
  }
  Serial.println();
}

// ====== raw -> scaled (黑线=1000, 白底=0)，自动兼容顺序 ======
int scaledSensor(uint16_t r, uint16_t w, uint16_t b) {
  if (w == b) return 0;

  long val;
  if (w > b) {
    // 白大黑小：scaled = (w - r)/(w - b) * 1000
    val = (long)(w - r) * SCALE  MAX / (long)(w - b);
  } else {
    // 白小黑大：scaled = (r - w)/(b - w) * 1000
    val = (long)(r - w) * SCALE_MAX / (long)(b - w);
  }

  if (val < 0) val = 0;
  if (val > SCALE_MAX) val = SCALE_MAX;
  return (int)val;
}

// ====== Step4：加权平均计算位置 ======
// 返回：pos（左负右正，中心≈0）
// 同时输出：sumScaled
float computePosition(const int scaled[NUM_SENSORS], long &sumScaledOut) {
  // 权重（先用索引权重，足够做循迹）
  const float w[NUM_SENSORS] = {-3.5, -2.5, -1.5, -0.5, 0.5, 1.5, 2.5, 3.5};

  long sum = 0;
  float weighted = 0.0;

  for (int i = 0; i < NUM_SENSORS; i++) {
    sum += scaled[i];
    weighted += w[i] * (float)scaled[i];
  }

  sumScaledOut = sum;

  // 丢线：sum 太小
  if (sum < SUM_LOST_TH) {
    return 9999.0f; // 用特殊值表示 lost
  }
```

```
  return weighted / (float)sum;
}

void setup() {
  Wire.begin();
  Serial.begin(115200);
  delay(200);

  setDefaultCalibration();

  Serial.println("=== Step4: Calibration + Scaled + Weighted Position ===");
  Serial.print("I2C addr = 0x"); Serial.println(SENSOR_ADDR, HEX);
  Serial.println("Commands (Serial Monitor 'No line ending' recommended):");
  Serial.println("  w -> capture WHITE (place all sensors on WHITE, wait 2s)");
  Serial.println("  b -> capture BLACK (place on BLACK line, wait 2s)");
  Serial.println("  p -> print calibration table");
  Serial.println();
}

void loop() {
  // ====== 串口指令 ======
  if (Serial.available()) {
    char c = Serial.read();
    if (c == 'w') {
      Serial.println("Put sensors on WHITE now... capturing 2s");
      captureCalibration(true, 2000);
    } else if (c == 'b') {
      Serial.println("Put sensors on BLACK line now... capturing 2s");
      Serial.println("(Tip: gently move to let different channels see the line)");
      captureCalibration(false, 2000);
    } else if (c == 'p') {
      printCalibration();
    }
  }

  // ====== 实时输出 raw + scaled + sum + pos ======
  if (readSF8(raw)) {
    int scaled[NUM_SENSORS];
    long sumScaled = 0;

    for (int i = 0; i < NUM_SENSORS; i++) {
      scaled[i] = scaledSensor(raw[i], whiteRef[i], blackRef[i]);
    }

    float pos = computePosition(scaled, sumScaled);
    if (pos != 9999.0f) lastPos = pos;  // 更新 lastPos

    // 打印 raw
    Serial.print("raw:\t");
    for (int i = 0; i < NUM_SENSORS; i++) {
      Serial.print(raw[i]);
      Serial.print(i == NUM_SENSORS - 1 ? "" : "\t");
    }

    // 打印 scaled
    Serial.print(" | scaled:\t");
```

```
    for (int i = 0; i < NUM_SENSORS; i++) {
      Serial.print(scaled[i]);
      Serial.print(i == NUM  SENSORS - 1 ? "" : "\t");
    }

    // 打印 sum + pos
    Serial.print(" | sum=");
    Serial.print(sumScaled);

    if (pos == 9999.0f) {
      Serial.print(" | pos=LOST");
      Serial.print(" (lastPos=");
      Serial.print(lastPos, 3);
      Serial.println(")");
    } else {
      Serial.print(" | pos=");
      Serial.println(pos, 3);
    }

  } else {
    Serial.println("Read failed (I2C)");
  }

  delay(50); // debug 用
}
```

Appendix K.4 – Final integrated code: 8-sensor PID line following + HMI (LCD+encoder)

```
#include <Wire.h>
#include <LiquidCrystal.h>
#include <math.h>

/* ==========================
   Part A: I2C Motor Driver
   ========================== */

constexpr uint8  t MOTOR  ADDR       = 0x2A;
constexpr int8_t  WHEEL_SENSE[4]    = {1, 1, 1, 1};
constexpr int8_t  LINEAR_TRIM[4]    = {-2, 6, -2, 6};
constexpr char    CMD_SET_SPEED_DIR = 'b';
constexpr char    CMD_HALT_ALL      = 'h';
constexpr char    TARGET  ALL       = 'a';

static inline int16_t clamp100(int v) {
  if (v > 100) return 100;
  if (v < -100) return -100;
  return v;
}

static inline void writeUint16LE(uint16_t x) {
  Wire.write((uint8  t)(x & 0xFF));
  Wire.write((uint8  t)(x >> 8));
}

void driveAll(int m1, int m2, int m3, int m4) {
```

```
  int req[4] = {m1, m2, m3, m4};
  char dir[4];
  uint16_t mag[4];

  for (uint8_t i = 0; i < 4; i++) {
    int v = req[i];
    v += (v >= 0) ? LINEAR_TRIM[i] : -LINEAR_TRIM[i];
    v = clamp100(v);

    int adj = clamp100(v * WHEEL_SENSE[i]);
    dir[i] = (adj >= 0) ? 'f' : 'r';
    mag[i] = (uint16_t)(adj >= 0 ? adj : -adj);
  }

  Wire.beginTransmission(MOTOR_ADDR);
  Wire.write(CMD_SET_SPEED_DIR);
  Wire.write(TARGET_ALL);
  Wire.write(dir[0]); Wire.write(dir[2]); Wire.write(dir[1]); Wire.write(dir[3]);
  writeUint16LE(mag[0]); writeUint16LE(mag[2]); writeUint16LE(mag[1]); writeUint16LE(mag[3]);
  Wire.endTransmission();
}

void haltAll() {
  Wire.beginTransmission(MOTOR_ADDR);
  Wire.write(CMD_HALT_ALL);
  Wire.write(TARGET_ALL);
  Wire.endTransmission();
}

/* ==========================
   Part B: SF-8CHIDTS sensor
   ========================== */

#define SENSOR_ADDR 0x09
#define NUM_SENSORS 8
#define SCALE_MAX   1000
#define SUM_LOST_TH 200

uint16_t raw[NUM_SENSORS];
uint16_t whiteRef[NUM_SENSORS];
uint16_t blackRef[NUM_SENSORS];
float lastPos = 0.0f;

bool readSF8(uint16_t out[NUM_SENSORS]) {
  const uint8_t nBytes = 16;
  uint8_t buf[nBytes];

  Wire.requestFrom(SENSOR_ADDR, nBytes);
  if (Wire.available() != nBytes) return false;

  for (int i = 0; i < nBytes; i++) buf[i] = Wire.read();

  for (int i = 0; i < NUM_SENSORS; i++) {
    uint8_t hi = buf[2 * i];
    uint8_t lo = buf[2 * i + 1];
    out[i] = ((uint16_t)hi << 8) | lo;
  }
```

```
  return true;
}

void setDefaultCalibration() {
  for (int i = 0; i < NUM_SENSORS; i++) {
    whiteRef[i] = 30000;
    blackRef[i] = 700;
  }
}

void captureCalibration(bool captureWhite, uint16_t ms = 2000) {
  unsigned long t0 = millis();

  uint16_t minv[NUM_SENSORS];
  uint16_t maxv[NUM_SENSORS];
  for (int i = 0; i < NUM_SENSORS; i++) { minv[i] = 65535; maxv[i] = 0; }

  while (millis() - t0 < ms) {
    if (!readSF8(raw)) continue;
    for (int i = 0; i < NUM_SENSORS; i++) {
      if (raw[i] < minv[i]) minv[i] = raw[i];
      if (raw[i] > maxv[i]) maxv[i] = raw[i];
    }
    delay(5);
  }

  if (captureWhite) {
    for (int i = 0; i < NUM_SENSORS; i++) whiteRef[i] = maxv[i];
    Serial.println("☑ White calibration captured (MAX on white).");
  } else {
    for (int i = 0; i < NUM  SENSORS; i++) blackRef[i] = minv[i];
    Serial.println("☑ Black calibration captured (MIN on black).");
  }
}

void printCalibration() {
  Serial.println("\nCalibration table:");
  Serial.println("ch\twhiteRef\tblackRef\tdiff");
  for (int i = 0; i < NUM_SENSORS; i++) {
    Serial.print(i); Serial.print('\t');
    Serial.print(whiteRef[i]); Serial.print('\t');
    Serial.print(blackRef[i]); Serial.print('\t');
    Serial.println((int)whiteRef[i] - (int)blackRef[i]);
  }
  Serial.println();
}

int scaledSensor(uint16_t r, uint16_t w, uint16_t b) {
  if (w == b) return 0;

  long val;
  if (w > b) val = (long)(w - r) * SCALE_MAX / (long)(w - b);
  else       val = (long)(r - w) * SCALE_MAX / (long)(b - w);

  if (val < 0) val = 0;
  if (val > SCALE_MAX) val = SCALE_MAX;
  return (int)val;
```

```
}

float computePosition(const int s[NUM_SENSORS], long &sumOut) {
  const float w[NUM_SENSORS] = {-3.5, -2.5, -1.5, -0.5, 0.5, 1.5, 2.5, 3.5};

  long sum = 0;
  float weighted = 0.0f;
  for (int i = 0; i < NUM_SENSORS; i++) {
    sum += s[i];
    weighted += w[i] * (float)s[i];
  }
  sumOut = sum;

  if (sum < SUM_LOST_TH) return 9999.0f;
  return weighted / (float)sum;
}

/* ==========================
   Part C: PID
   ========================== */

int baseSpeed = 25;

float Kp = 99.0f;
float Ki = 0.0f;
float Kd = 5.8f;

float I_term = 0.0f;
float I_max  = 30.0f;
float lastErr = 0.0f;
unsigned long lastT = 0;

float pidTurn(float err, float dt) {
  float P = Kp * err;

  I_term += err * dt;
  if (I_term > I_max) I_term = I_max;
  if (I_term < -I_max) I_term = -I_max;
  float I = Ki * I_term;

  float D = 0.0f;
  if (dt > 1e-4f) D = Kd * (err - lastErr) / dt;
  lastErr = err;

  return P + I + D;
}

/* ============================================================
   ===== HMI FINAL START  (按下切换 Kp/Ki/Kd，旋转调数值) =====
   Pins: LCD RS=D12 EN=D11 D4=D4 D5=D5 D6=D6 D7=D7
        Encoder A=D3  B=D10  Button(SW)=D9   (全部用 PULLUP)
   ============================================================ */

#define ENC_A_PIN    3
#define ENC_B_PIN    10
#define ENC_BTN_PIN  9
```

```
const int LCD_RS = 12, LCD_EN = 11, LCD_D4 = 4, LCD_D5 = 5, LCD_D6 = 6, LCD_D7 = 7;
LiquidCrystal lcd(LCD_RS, LCD_EN, LCD_D4, LCD_D5, LCD_D6, LCD_D7);

enum ParamSel { SEL_KP = 0, SEL_KI = 1, SEL_KD = 2, SEL_BASE = 3 };
ParamSel sel = SEL_KP;

uint8_t encLast = 0;          // 2-bit AB state
int encAccum = 0;

// 不同编码器“每格”跳变数不同，常见 2 或 4。你先用 4，不灵敏改 2
const int ENC_COUNTS_PER_DETENT = 4;

// 参数步长（每“格”旋转一次的改变量）
const float STEP_KP = 0.5f;
const float STEP_KI = 0.02f;
const float STEP_KD = 0.2f;
const int   STEP_BASE = 1;     // baseSpeed 每格 +1

// 按键去抖（非阻塞）
bool btnStable = HIGH;
bool btnLastStable = HIGH;
unsigned long btnChangeMs = 0;
const unsigned long BTN_DEBOUNCE_MS = 30;

// LCD 刷新频率（防闪）
unsigned long lcdMs = 0;
const unsigned long LCD_PERIOD_MS = 200;

// 限制数值，保证不会撑爆 16 列显示 & 运行安全
void clampParamsForLCD() {
  if (!isfinite(Kp)) Kp = 0;
  if (!isfinite(Ki)) Ki = 0;
  if (!isfinite(Kd)) Kd = 0;

  if (Kp < 0) Kp = 0;      if (Kp > 300.0f) Kp = 300.0f;   // "99.9"
  if (Ki < 0) Ki = 0;      if (Ki > 9.99f) Ki = 9.99f;   // "9.99"
  if (Kd < 0) Kd = 0;      if (Kd > 99.9f) Kd = 99.9f;

  baseSpeed = constrain(baseSpeed, 0, 100);              // 0~100
}

// 固定宽度渲染（避免 float 乱码/残影）
void lcdRender() {
  clampParamsForLCD();

  char kpStr[8], kiStr[8], kdStr[8];
  dtostrf(Kp, 5, 1, kpStr);   // "33.0" / " 6.5"
  dtostrf(Ki, 4, 2, kiStr);   // "0.00"
  dtostrf(Kd, 4, 1, kdStr);   // " 6.5"

  char line0[17], line1[17];

  // 行 0：  >P:xx.x I:x.xx   （严格 16 列）
  // 例：">P:33.0 I:0.00 "
  snprintf(line0, sizeof(line0), "%cP:%s I:%s ",
           (sel == SEL_KP) ? '>' : ' ',
```

```
        kpStr, kiStr);

  // 行 1： >D:xx.x B:xxx  （严格 16 列）
  // 例："">D: 6.5 B: 10  "
  snprintf(line1, sizeof(line1), "%cD:%s B:%3d ",
        (sel == SEL_KD) ? '>' : ' ',
        kdStr, baseSpeed);

  // 如果当前选中的是 BASE，就把 '>' 放到 B 前面（更直观）
  if (sel == SEL_BASE) {
   // 把 line1 的第 0 位改成空格，并在 B 前插入 >
   // line1 结构固定： [0] [1..] "D:.... B:xxx "
   // 我们把 "B:" 前的空格替换为 '>'
   // "xD:xxxx B:xxx "
   line1[0] = ' ';              // 取消 D 的 >
   // 找到 'B' 的位置，前一位改成 '>'
   for (int i = 0; i < 16; i++) {
    if (line1[i] == 'B') { if (i > 0) line1[i - 1] = '>'; break; }
   }
  }

  // 写 LCD 时建议关中断，减少电机干扰导致的乱码
  noInterrupts();
  lcd.setCursor(0, 0); lcd.print(line0);
  lcd.setCursor(0, 1); lcd.print(line1);
  interrupts();
}

// 查表解码：最稳
void updateEncoder() {
  uint8_t a = digitalRead(ENC_A_PIN);
  uint8_t b = digitalRead(ENC_B_PIN);
  uint8_t cur = (b << 1) | a;

  static const int8_t table[16] = {
    0, -1, +1,  0,
   +1,  0,  0, -1,
   -1,  0,  0, +1,
    0, +1, -1,  0
  };

  uint8_t idx = (encLast << 2) | cur;
  int8_t d = table[idx];
  encLast = cur;

  if (d == 0) return;
  encAccum += d;

  // 达到阈值算“一格”
  if (encAccum >= ENC_COUNTS_PER_DETENT) {
   encAccum = 0;

   if (sel == SEL_KP)   Kp += STEP_KP;
   if (sel == SEL_KI)   Ki += STEP_KI;
   if (sel == SEL_KD)   Kd += STEP_KD;
   if (sel == SEL_BASE) baseSpeed += STEP_BASE;
```

```
    clampParamsForLCD();
    lcdRender();
  }
  else if (encAccum <= -ENC_COUNTS_PER_DETENT) {
    encAccum = 0;

    if (sel == SEL_KP)   Kp -= STEP_KP;
    if (sel == SEL_KI)   Ki -= STEP_KI;
    if (sel == SEL_KD)   Kd -= STEP_KD;
    if (sel == SEL_BASE) baseSpeed -= STEP_BASE;

    clampParamsForLCD();
    lcdRender();
  }
}

// 按下切换 Kp/Ki/Kd/Base（非阻塞去抖）
void updateEncButton() {
  bool raw = digitalRead(ENC_BTN_PIN);

  if (raw != btnStable) {
    btnStable = raw;
    btnChangeMs = millis();
  }

  if (millis() - btnChangeMs > BTN_DEBOUNCE_MS) {
    if (btnLastStable == HIGH && btnStable == LOW) {
      sel = (ParamSel)((sel + 1) % 4);   // 0..3
      lcdRender();
    }
    btnLastStable = btnStable;
  }
}

void hmiInit() {
  pinMode(ENC_A_PIN, INPUT_PULLUP);
  pinMode(ENC_B_PIN, INPUT_PULLUP);
  pinMode(ENC_BTN_PIN, INPUT_PULLUP);

  encLast = ((digitalRead(ENC_B_PIN) << 1) | digitalRead(ENC_A_PIN));
  encAccum = 0;

  lcd.begin(16, 2);
  lcd.clear();
  clampParamsForLCD();
  lcdRender();
}

void hmiUpdate() {
  updateEncoder();
  updateEncButton();

  if (millis() - lcdMs > LCD_PERIOD_MS) {
    lcdMs = millis();
    lcdRender();
  }
```

```
}

/* ==========================
   ===== HMI FINAL END =====
   ========================== */

void setup() {
  Wire.begin();
  Serial.begin(115200);
  delay(200);

  setDefaultCalibration();
  lastT = millis();

  // ===== HMI INIT (新增) =====
  hmiInit();

  Serial.println("=== PID Line Following + HMI (Encoder+LCD) ===");
  Serial.println("Serial cmds: w/b/p calibration, s stop");
}

void loop() {
  // ===== HMI UPDATE (新增) =====
  hmiUpdate();

  // -------- Serial commands (校准) --------
  if (Serial.available()) {
    char c = Serial.read();
    if (c == 'w') captureCalibration(true, 2000);
    if (c == 'b') captureCalibration(false, 2000);
    if (c == 'p') printCalibration();
    if (c == 's') { haltAll(); Serial.println("STOP"); }
  }

  // -------- Read sensors --------
  if (!readSF8(raw)) {
    haltAll();
    delay(20);
    return;
  }

  int s[NUM_SENSORS];
  for (int i = 0; i < NUM_SENSORS; i++) s[i] = scaledSensor(raw[i], whiteRef[i], blackRef[i]);

  long sumScaled = 0;
  float pos = computePosition(s, sumScaled);

  // timing
  unsigned long now = millis();
  float dt = (now - lastT) / 1000.0f;
  if (dt <= 0) dt = 0.01f;
  lastT = now;

  int leftCmd = 0, rightCmd = 0;

  if (pos == 9999.0f) {
    int search = 25;
```

```
    if (lastPos < 0) { leftCmd = -search; rightCmd = +search; }
    else             { leftCmd = +search; rightCmd = -search; }
  } else {
    lastPos = pos;
    float err = pos;

    float turn = pidTurn(err, dt);

    // 误差越大越猛（保留你成功逻辑）
    float eabs = fabs(err);
    float turnGain = 1.0f;
    if (eabs > 1.2f) turnGain = 1.5f;
    if (eabs > 2.2f) turnGain = 2.2f;
    turn *= turnGain;

    turn = constrain(turn, -100, 100);

    leftCmd  = clamp100(baseSpeed - (int)turn);
    rightCmd = clamp100(baseSpeed + (int)turn);
  }

  driveAll(leftCmd, rightCmd, leftCmd, rightCmd);

  // 不要太大 delay，否则编码器会漏步；1ms OK
  delay(1);
}
```

Chapter 16 Appendix L — TCRT5000 (and sensor stream) data acquisition sketch (Arduino)

```
#include <MsTimer2.h>
#include <Wire.h>
//
//Connect Vcc and GND of TCRT5000 sensor module to 5V and GND of Arduino
//Connect D0 of TCRT5000 to A0 of Arduino
//Connect A0 of TCRT5000 to A1 of Arduino

//Connect 5V and GND of Line follower module to 5V and GND of Arduino
//Connect SDA and SCL of Line follower module to A4 and A5 of Arduino
//
#define CompIn (A0) // D0 output of TCRT5000 sensor module
#define Analog (A1) // A0 output of TCRT5000 sensor module
#define LED (13) // LED of Arduino board
#define FREQ  CTL 10



  // variable to store the value coming from the sensor
unsigned char dataRaw[16];
unsigned int sensorData[8], count  g1=0,sensorValue  1,sensorValue = 0;


void setup() {
  // declare the ledPin as an OUTPUT:
  pinMode(LED, OUTPUT);
  noInterrupts();
  Serial.begin(57600);   // connect to PC

  User_init_timer2();
  Wire.begin();
  interrupts();


}

void loop() {
  // read the value from the sensor:
  if (count_g1==1)
  {
    readSensorData();
    sensorValue = analogRead(Analog);
    sensorValue_1=map(sensorValue,0,1023,10,200);
    count_g1=0;
  }

  // turn the ledPin on
  digitalWrite(LED, HIGH);
  // stop the program for <sensorValue> milliseconds:
  delay(sensorValue  1);
  // turn the ledPin off:
  digitalWrite(LED, LOW);
```

```
  // stop the program for for <sensorValue> milliseconds:
  delay(sensorValue_1);
}
inline void User  init  timer2()
{
  MsTimer2::set(FREQ_CTL, Timer2ISR);
  MsTimer2::start();
}

void Timer2ISR()   // Timer2 interrupt service routine (ISR)
{
  static unsigned int counter_1 = 0;
  if (counter  1 < 100)                 // LEB blinking to make sure the baseboard is working well
  {
    counter_1++;
    count_g1=1;
  }
  else
  {
      Serial.println("---------------");
      Serial.print(sensorData[0]);
      Serial.print(" ");
      Serial.print(sensorData[1]);
      Serial.print(" ");
      Serial.print(sensorData[2]);
      Serial.print(" ");
      Serial.print(sensorData[3]);
      Serial.print(" ");
      Serial.print(sensorData[4]);
      Serial.print(" ");
      Serial.print(sensorData[5]);
      Serial.print(" ");
      Serial.print(sensorData[6]);
      Serial.print(" ");
      Serial.println(sensorData[7]);
      Serial.print(digitalRead(CompIn));
      Serial.print(" ");
      Serial.println(sensorValue);
    counter_1 = 0;
  }

}

void readSensorData(void)
{
  unsigned char n;            // Variable for counter value
  //unsigned char dataRaw[16];  // Array for raw data from module

  // Request data from the module and store into an array
  n = 0;  // Reset loop variable
  Wire.requestFrom(9, 16);  // Request 16 bytes from slave device #9 (IR Sensor)
  while(Wire.available())  // Loop until all the data has been read
  {
    if (n < 16)
    {
      dataRaw[n] = Wire.read();  // Read a byte and store in raw data array
      n++;
```

```
    }
    else
    {
      Wire.read();  // Discard any bytes over the 16 we need
      //n = 0;
    }
  }

  // Loop through and covert two 8 bit values to one 16 bit value
  // Raw data formatted as "MSBs 10 9 8 7 6 5 4 3", "x x x x x x 2 1 LSBs"
  for(n=0;n<8;n++)
  {
    sensorData[n] = dataRaw[n*2]<< 2;   // Shift the 8 MSBs up two places and store in array
    sensorData[n] += dataRaw[(n*2)+1];  // Add the remaining bottom 2 LSBs
  }
}
```

Chapter 17 Appendix M— Raw data tables and plots used in analysis

Table M.1 summarises the measured analogue output voltage V_{A0} versus distance d above **white** and **black** surfaces, together with the derived discrimination margin

Table M.1 *Measured V_{A0} versus distance for white/black surfaces and sensitivity ΔV_{A0}.*

Distance d (cm)	VA0_white (V)	VA0_black (V)	DeltaV (V)
0.5	0.15	0.2	0.05
1	0.17	1.4	1.23
1.5	0.19	3.14	2.95
2	0.2	3.75	3.55
2.5	0.226	4.05	3.824
3	0.255	4.26	4.005
3.5	0.5	4.34	3.84
4	1.2	4.4	3.2
4.5	1.78	4.46	2.68
5	2.05	4.5	2.45
5.5	2.48	4.52	2.04
6	2.7	4.5	1.8