**EEEE1039 Applied Electrical and Electronic Engineering: Construction Project**

**2024/25 Session**

# Design and Real-Time Control of a Four-Wheel Robotic Vehicle with HMI

**DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING**

**UNIVERSITY OF NOTTINGHAM NINGBO CHINA**

Name : Hongziheng Wang

Student ID : 20717880

## Abstract

This report documents a two-week Applied EEE construction project that progressed from basic analogue circuits to a fully integrated, real-time controlled four-wheel vehicle based on two Arduino Nano boards. In Session 1, divider and LED circuits were designed and tested to establish safe low-voltage interfaces. The measured output of a 10 V divider (6.68 V unloaded and 5.02 V with a 10 kΩ load) agreed with theory to within 0.5 %, and an LED delay circuit with a 220 Ω / 470 µF RC network produced a measured time constant of ≈105 ms, matching the calculated 0.10 s. A discrete H-bridge was characterised using the JGA25-370 motor: encoder-derived speed approximately doubled when the supply increased from 3 V to 6 V, while loaded tests confirmed the expected trade-off between torque, current and speed. Introductory Arduino sketches then verified digital I/O, PWM, analogue measurement and $I^2C$ communication, and a baseline 1 m run demonstrated correct motor polarity and smooth chassis motion.

Session 2 added communication, monitoring and human–machine interaction. A UART link between two Nanos transmitted "Hello World" at 9600 bps; the oscilloscope measured a bit period of ≈104 µs, consistent with 1/9600 s. A low-battery indicator using a 91 kΩ/10 kΩ divider (≈0.10 scaling) and two LEDs was built and calibrated. Comparison with an 18650 discharge curve showed that the upper threshold (second LED on at ≈7.7 V) was close to the intended half/full region, whereas the lower threshold (blink-to-steady at ≈4.4 V) was significantly below the desired 10 %-charge voltage, highlighting the impact of empirical tuning and component tolerances. A timer-driven PWM scheme with RC filtering generated 9 $V_{pp}$ sine waves at 0.5 Hz and 2 Hz with carrier frequency ≈6.6 kHz and ripple below 0.08 V, and timing comparisons between noIRQ and withIRQ sketches confirmed the superiority of interrupt-based updates. Finally, a custom HMI board (buttons, slide switch, rotary encoder, microphone, LCD, LEDs and buzzers) and the two Nanos were integrated on the vehicle. The system successfully completed Task B of the final challenge—running a selectable distance at a user-defined speed—demonstrating a reproducible embedded control platform suitable for further refinement of speed, heading and battery-management strategies.

# Chapter 1 Introduction

## 1.1 Background and motivation

The Applied Electrical and Electronic Engineering construction project combined basic circuit design with embedded programming to realise a small four-wheel robotic vehicle. Over two laboratory sessions the work progressed from simple analogue experiments on a breadboard to a fully integrated system with serial communication, a Human–Machine Interface (HMI), real-time (RT) control and a final motion challenge. The project allowed students to practise safe use of laboratory instruments, gain experience with Arduino Nano microcontrollers and develop confidence in moving from individual circuits to a complete embedded system.

## 1.2 Aim and objectives

The overall aim of the project was to design, implement and verify a reproducible embedded control platform for a robotic vehicle under realistic hardware constraints. The specific objectives were:

**Session 1 – Foundations and platform build**

1. Practise safe operation of the bench power supply (PSU), digital multimeter (DMM) and oscilloscope, and establish a procedure for recording evidence (tables, plots and photographs).

2. Design and test potential dividers, LED current-limiting circuits and RC delay networks, and compare measurements with theoretical predictions.

3. Assemble and characterise a manual H-bridge to understand bidirectional DC-motor control and flyback protection.

4. Complete the basic Arduino exercises (digital I/O, PWM, analogue input and serial/$I^2C$ communication).

5. Assemble the four-wheel chassis with motors, battery holder and baseboard, install the Arduino and demonstrate a baseline 1 m motion test using $I^2C$ motor commands.

**Session 2 – Communication, RT control and HMI**

6. Establish and verify a UART link between two Arduino Nano boards using the Serial Monitor and oscilloscope.

7. Investigate the influence of interrupts on timing stability by comparing loop-based (noIRQ) and interrupt-based (withIRQ) PWM control.

8. Implement a RT sine-wave generator using PWM plus analogue amplification and RC filtering, and verify amplitude, frequency and ripple.

9. Design and calibrate a low-battery indicator using a resistive divider and RC filter, with three LED-based charge-level states.

10. Design, solder and test an HMI board (buttons, slide switch, rotary encoder, LCD, microphone and buzzers) and integrate it with the vehicle to complete the final challenge task.

### 1.3 Requirements and constraints

The design was constrained by the laboratory hardware and safety limits. All analogue interfaces to the Arduino had to remain below 5.5 V, requiring appropriate divider ratios and current-limiting resistors. RC filters were specified with time constants in the range of tens to hundreds of milliseconds to smooth PWM and battery noise without excessive delay. The RT control loop was required to run at a frequency of at least 200 Hz, while the sine generator had to produce approximately $9V_{pp}$ in the 1–4 Hz range with ripple less than 0.2 V. The low-battery indicator thresholds had to be within ±10 % of the values derived from the 18650 discharge curve. Finally, the limited number of GPIO and PWM pins on the Nano determined the pin allocation for the HMI and motor driver.

### 1.4 System overview

The final system comprised five functional blocks:

1. **Power and protection**, including the two-cell 18650 battery pack and a three-level low-battery indicator.
2. **Computation**, using one Arduino Nano for HMI and high-level control and a second Nano for motor control and waveform generation.
3. **Motor drive**, provided by the baseboard H-bridge shield and four DC gear motors mounted on the chassis.
4. **HMI**, consisting of buttons, slide switch, rotary encoder, microphone, status LEDs, buzzers and an $I^2C$ LCD.
5. **Communication and sensing**, including UART between the two Nanos, $I^2C$ between Nano-B and the motor driver, and optional encoder feedback.

User commands entered via the HMI were processed by Nano-A, which communicated target speed and distance to Nano-B. Nano-B executed the RT control loop, generated PWM signals for the motor driver and, where required, drove the sine-wave output. Feedback was presented to the user via the LCD, LEDs and buzzers.

## 1.5 Contributions

This work delivers:

- A documented workflow from discrete analogue circuits to an integrated four-wheel robotic platform. a validated UART link and timing strategy that mitigates loop-delay instability.
- Experimental validation of potential-divider, LED and RC-network calculations, including the effect of loading and component tolerances. a compact HMI and a calibrated three-level battery indicator.
- A verified UART link and timing strategy that demonstrate the benefits of interrupt-driven RT control.
- A practical implementation of a low-battery indicator and an HMI board suitable for extension in later projects.
- Evidence that the final vehicle can reliably execute the specified challenge task using the designed RT and HMI architecture.

## 1.6 Report structure

The remainder of this report follows the IMRaD structure. **Section 2 (Methods)** details the design rationale and implementation steps for each task in Sessions 1 and 2. **Section 3 (Results and Discussion)** presents measurement data, oscilloscope waveforms and observations, and interprets how they compare with theoretical expectations. **Section 4 (Conclusion)** summarizes the main findings, discusses limitations of the current design and outlines possible improvements for future work. The Arduino code listings, additional schematics and photographs of the physical build are included in the **Appendices**.

# Chapter 2 Methods

## 2.1 Overall approach

This project was executed over two laboratory sessions. The first session laid the electrical foundation by exploring simple analogue circuits such as potential dividers, RC networks and H-bridges and by bringing up the Arduino controller and chassis. The second session added communication and control: two Arduino Nano boards were networked over UART, a low-battery indicator was designed and calibrated, a human–machine interface (HMI) board was assembled, and a series of real-time (RT) tasks (RC filter timing, sine-wave generation and vehicle control) were implemented. The methods were written in the third person and past tense to enable replication.

## 2.2 Session 1 – Basic circuits and manual H-bridge

### 2.2.1 Skills and instrumentation

Basic laboratory skills (soldering, wiring, safe instrument use) were practiced before assembly. A multimeter, bench power supply, and oscilloscope were used to validate circuit behavior and to record measurements reported later in Section 3. The activity established a repeatable procedure for evidence collection (figures, photos, plots) to support conclusions.

### 2.2.2 Potential divider with load

**Theory.** A resistive potential divider was used to generate a reduced voltage from a higher DC supply. The ideal unloaded output voltage $V_{out}$was computed from the resistor ratio:

$$V_{out} = V_{in}\frac{R_2}{R_1+R_2} \qquad \text{(Equation 2.2.1)}$$

When a finite load $R_{\text{LOAD}}$ is connected in parallel with $R_2$, the effective resistance decreases and the output voltage drops compared with the ideal case.

**Procedure.** A bench supply provided $V_{\text{in}} = 10$ V. Resistors $R_1 = 5$ kΩand $R_2 = 10$ kΩwere measured with a DMM (uncertainty ±1 %). With no load, $V_{\text{out}}$was measured. A 10 kΩ load was then connected in parallel with $R_2$and the new output voltage recorded. Power dissipation in each resistor was calculated to ensure it remained below 0.25 W. Schematic diagrams were drawn and wiring colours followed the red/black convention.

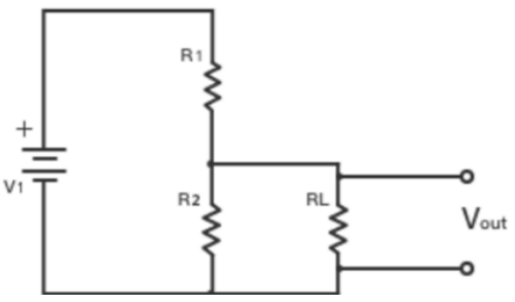


*Figure 2.1: Schematic of the loaded potential divider, adapted from the H-bridge instruction manual.*

### 2.2.3 LED current-limiting and RC delay circuit

**Theory.** A series resistor was used to limit LED current according to Ohm's law:

$$R_{LED} = \frac{V_{CC} - V_F}{I_F} \qquad \text{(Equation 2.2.2)}$$

where $V_F$is the LED forward voltage and $I_F$is the desired forward current.

To introduce a turn-on delay, a capacitor $C_1$was placed in series with the LED resistor to form an RC network. The approximate delay time was set by the time constant

$$T_C = R_3 C_1 \qquad \text{(Equation 2.2.3)}$$

which corresponds to the time required for the capacitor voltage to reach about 63.2 % of its final value.

**Procedure.** Following the manual, a potential-divider consisting of $R_1 = 15\ \text{k}\Omega$ and $R_2 = 10\ \text{k}\Omega$was wired to reduce the 5 V supply to about 2 V at node A. A red LED ($V_f \approx 2$ V) was connected from node A to ground through a pair of resistors in series giving $R_3 \approx 220\ \Omega$ . An electrolytic capacitor $C_1 = 470\ \mu F$ was then connected across $R_3$and the LED. Power was applied and removed while the voltage across $C_1$was recorded with an oscilloscope. The charging curve was inspected manually to find the time at which the capacitor voltage reached 63.2 % of its final value, and the discharge curve was recorded until the voltage decayed to near zero. Throughout, component values were chosen to keep currents within LED ratings and to produce a delay of the order of tenths of a second.

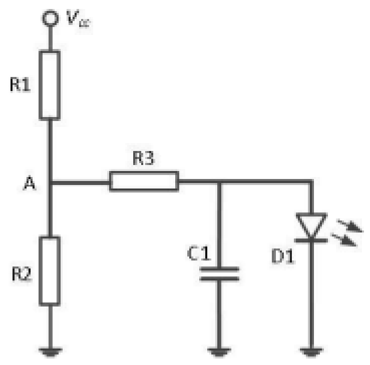


*Figure 2.2: LED current-limiting and RC delay schematic.*

### 2.2.4 Manual H-bridge and motor driver checkout

**Theory.** An H-bridge allows a DC motor to be driven in either direction by reversing the current. Four transistors or switches are arranged in an "H" configuration; two conduct in a forward state and the opposite two for reverse. Flyback diodes protect the transistors from inductive voltage spikes.

**Procedure.** Four NPN transistors were arranged as an H-bridge on a breadboard with flyback diodes across the motor. A small DC motor rated at 6 V was connected. Each pair of transistors was switched on using push-buttons to drive the motor forward and backward. A DMM verified that current draw remained below 150 mA. Once operation was confirmed, the H-bridge concept was replaced by a motor driver shield for the vehicle, but the experiment familiarized the team with polarity control and flyback protection.

For the H-bridge motor controller study (Tasks 7–8), the geared DC motor JGA25-370 DC6V280RPM and the slide switches S1–S4 were identified from the bill of materials and the motor datasheet. The datasheet was used to assume linear relationships between supply voltage and speed, and between torque and current. These relations provided predicted values of motor voltage, current and power for later comparison with measurements. The complete H-bridge of Figure 2.3 was then assembled on a breadboard together with the motor encoder, ready for no-load and loaded tests. The manual H-bridge configuration with four switches S1–S4 is shown in Figure 2.3 [1].

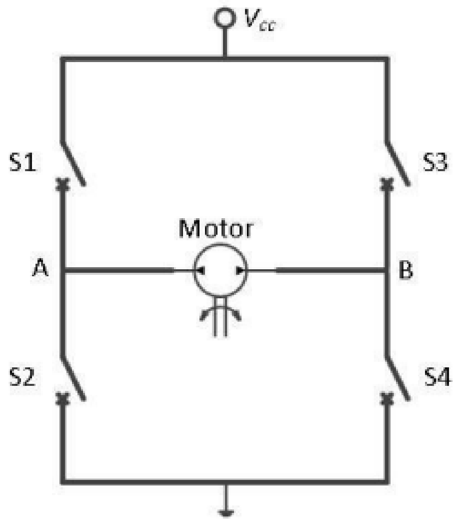


*Figure 2.3: Manual H-bridge schematic*

### 2.2.5 Arduino basics tasks

**Theory.** The introductory tasks for Arduino acquaint students with fundamental microcontroller operations: digital outputs and timing (digitalWrite() and delay()), digital inputs with pull-up resistors (digitalRead()), PWM outputs (analogWrite()), analogue inputs (analogRead()), and serial and I²C communications. These tasks demonstrate the working principles of toggling an LED, debouncing a push-button, generating a fading effect via PWM, converting an analogue voltage to a digital value and printing it to the serial monitor, controlling the board from a PC via the serial monitor, and exchanging data between two Arduinos over the I²C bus. The introductory Arduino tasks followed the examples provided in the lab manual [2].

**Procedure.** The Arduino Nano pin mapping and UART pins were taken from the official schematic [3]. Six short experiments were conducted:

1. **Blink.** The built-in LED (pin 13) and an external LED on the breadboard were blinked at ~1 Hz using digitalWrite() and delay() to verify digital output functionality.
2. **Push Button.** A push-button wired between ground and a digital input pin was used to detect button presses. A pull-up resistor held the input high when the button was not pressed, and a sketch turned the LED on when the button was pressed and off when released.
3. **Fade.** A PWM-capable pin drove the LED brightness smoothly up and down using analogWrite() with a varying duty cycle. The brightness incremented or decremented in the loop() function, demonstrating analog-like output.

4. **Read Analog Voltage.** An analogue voltage from a potentiometer or voltage divider was connected to A0. analogRead() converted the input to a 10-bit integer; the code converted this to a voltage and printed it to the serial monitor.
5. **Serial Communication.** The sketch used the Serial library to send and receive messages from the PC via the serial monitor. Commands such as r to read the analogue voltage and b to blink the LED were implemented to test bidirectional communication.
6. **I$^2$C Communication.** Two Arduino Nanos were connected using their SDA and SCL pins with pull-up resistors. One board acted as master and the other as slave. The master read the analogue voltage and sent it to the slave via I$^2$C; the slave blinked an LED when data exceeded a threshold. Oscilloscope waveforms of the I$^2$C lines were captured for the logbook.

### 2.2.6 Baseboard verification and four-wheel vehicle assembly

**Theory.** The physical platform for the vehicle consists of a two-level chassis with four geared DC motors, motor brackets, wheels, a battery holder and mounting posts. Proper mechanical assembly is essential for stability, motor alignment and sufficient space for PCBs. Each motor drives one wheel; correct alignment prevents drag and ensures smooth movement. The mechanical assembly procedure for the baseboard and chassis followed the project notes [4].

**Procedure.** The kit's parts list was checked to confirm that two chassis plates, four DC motors with gearboxes, four wheels, a caster wheel, motor brackets, screws, nuts and spacers were available. The steps were as follows:

1. **Motor installation.** Motors were bolted to the lower plate using L-brackets; screws were tightened evenly to avoid misalignment. Wires were left long enough for routing.

2. **Wheel attachment.** Wheels were pressed onto the motor shafts and secured with setscrews.

3. **Chassis assembly.** Four spacers connected the lower and upper plates; motor wires were passed through slots without pinching. The caster wheel or skid was attached to the front or rear for balance.

4. **Battery and electronics mounting.** The battery holder was fixed to the top plate. The motor driver shield, Arduino Nano and other PCBs (low-battery indicator, HMI board) were mounted with standoffs. Wires were trimmed and soldered or crimped to connectors, observing polarity.

5. **Initial power test.** Each motor was briefly energised with a bench supply to verify direction; polarity marks were noted. The assembled chassis rolled freely and the motors rotated without obstruction.

This mechanical build completed the Session 1 Topic 3 requirement and prepared the platform for installing the Arduino and implementing four-wheel control.

### 2.2.7 Installing Arduino on the car and four-wheel control

**Theory.** Once the vehicle chassis was assembled and the basic and combined Arduino tasks were completed, the microcontroller was installed on the car. The working principle of four-wheel control is to generate appropriate PWM signals to drive the motor driver channels controlling the left and right wheels. By adjusting duty cycles, the Arduino can control speed and direction for each side independently, enabling forward, reverse, turning and spinning maneuvers.

**Procedure.** The Nano was mounted on the vehicle in a protected enclosure. Its motor driver outputs were wired to the four motor terminals via the H-bridge/motor driver shield. The control sketch used two PWM outputs per side (left and right) to drive the motors. Forward motion was achieved by setting both sides to the same duty cycle; turning was achieved by reducing duty on one side; reverse was accomplished by swapping the direction pins. The program combined user-selected speed and direction values from the HMI board with closed-loop adjustments to correct for drift. After verifying correct operation on the bench, the vehicle was driven on the floor to confirm stable four-wheel control.

## 2.3 Session 2: Communication, real-time control, HMI, and integration

Design targets for the low-battery indicator (10 V–1 V scaling and three LED states) were adopted from the Session 2 challenge brief [5].

### 2.3.1 UART link between two Arduino Nano boards

Two Arduino Nano boards were programmed individually with UART_mirror and UART_mirror_sender sketches. Because the serial RX/TX pins are used by the USB bootloader, each Nano was programmed before the boards were physically interconnected. The pins were connected as TX→RX and RX→TX with a common ground wire. The sender transmitted an ASCII message ("Hello World") at 9600 baud. The receiver echoed the message back and the result was observed in the Serial Monitor and with an oscilloscope. A schematic of the connection was drawn. Notes were made on why it is not possible to re-program a board once the UART pins are used: the USB–serial adapter shares those lines, so the bootloader cannot communicate when another device is connected.

The final wiring of the UART link, drawn from the pin labels on the Nano and expansion boards, is summarized in **Figure 2.4**, which shows $\mathrm{TX}_1\text{–}\mathrm{RX}_2$, $\mathrm{RX}_1\text{–}\mathrm{TX}_2$ and a common ground connection. Oscilloscope probes were connected to both TX and RX lines so that the digital waveforms could be recorded for later analysis.

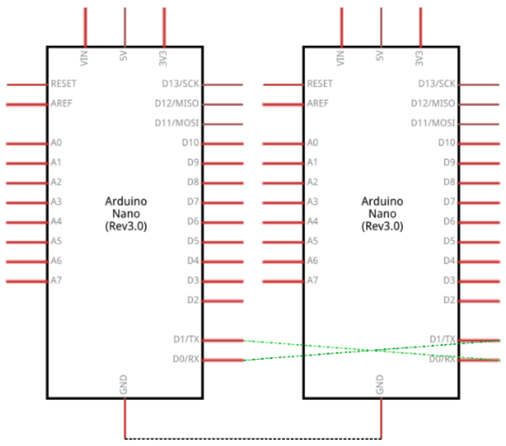


*Figure 2.4: Connection diagram of the two Arduino Nanos (TX/RX/GND).*

### 2.3.2 Design of the low-battery voltage indicator

**Theory.** The indicator uses a resistive potential divider to scale the battery voltage down to a safe level for the Arduino analogue input. For the first student in the group list the specification requires 10 V at the input to appear as about 1 V at the divider output. With $R_1 = 91\ \text{k}\Omega$and $R_2 = 10\ \text{k}\Omega$the division ratio is

$$k = \frac{R_2}{R_1+R_2} \approx \frac{10}{91+10} \approx 0.10 \qquad \text{(Equation 2.3.1)}$$

so a maximum input of 10 V produces approximately 1.0 V at the Arduino pin, well below the 5.5 V limit. A capacitor $C = 4.7\ \mu\text{F}$is connected across $R_2$to form a low-pass RC filter with an effective time constant of tens of milliseconds, smoothing battery noise and fast transients before the voltage is sampled by analogRead().

Threshold values for "low", "half" and "full" charge were chosen from the 18650 discharge characteristic in Figure 5. These pack voltages were multiplied by the divider ratio $k$to obtain the expected Arduino-pin voltages. In the code, the scaled thresholds were converted to ADC counts and compared against the measured value. Depending on the result, the two LEDs indicated battery status: one LED blinking for "low", one LED steadily on for "half", and both LEDs on for "full".

**Procedure.** A small indicator PCB was built following the schematics in Figure 2.5 (divider + RC filter) and Figure 2.6 (divider, Arduino Nano and two LEDs). Resistors $R_1 = 91\ \text{k}\Omega$and $R_2 = 10\ \text{k}\Omega$ were soldered to form the divider; a 4.7 µF electrolytic capacitor was wired across $R_2$. Two 220 Ω series resistors limited the LED currents. The Arduino sketch periodically read the filtered divider voltage on analogue input A1, calculated the corresponding battery voltage using the known ratio $k$and updated the LED states and Serial Monitor output.

For calibration, a bench supply emulating the two-cell battery was swept from about 10 V down to 4 V while the divider output and LED behaviour were observed. The battery voltages at which the indicator changed from blinking to steady one-LED, and from one-LED to two-LED, were recorded as the measured threshold values. Expected divider voltages and measured values are tabulated in Table Y.

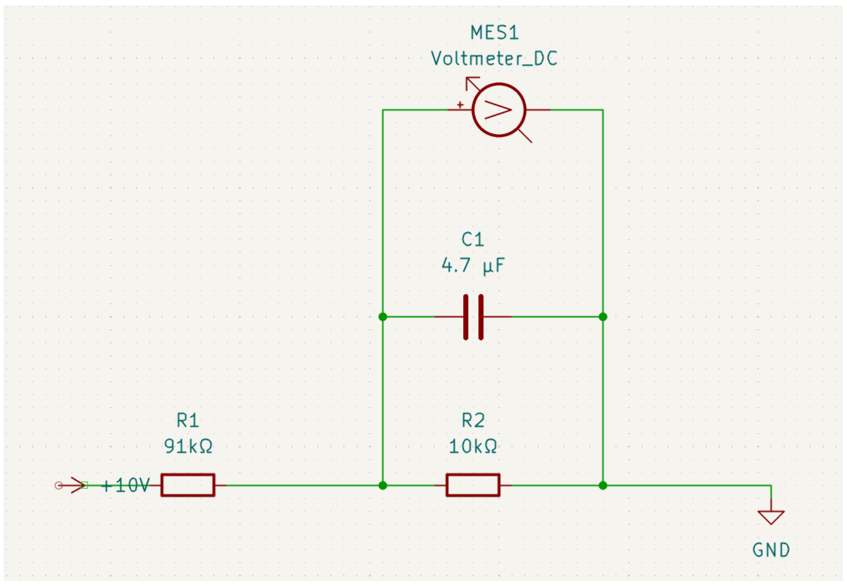


*Figure 2.5 (divider + RC filter)*

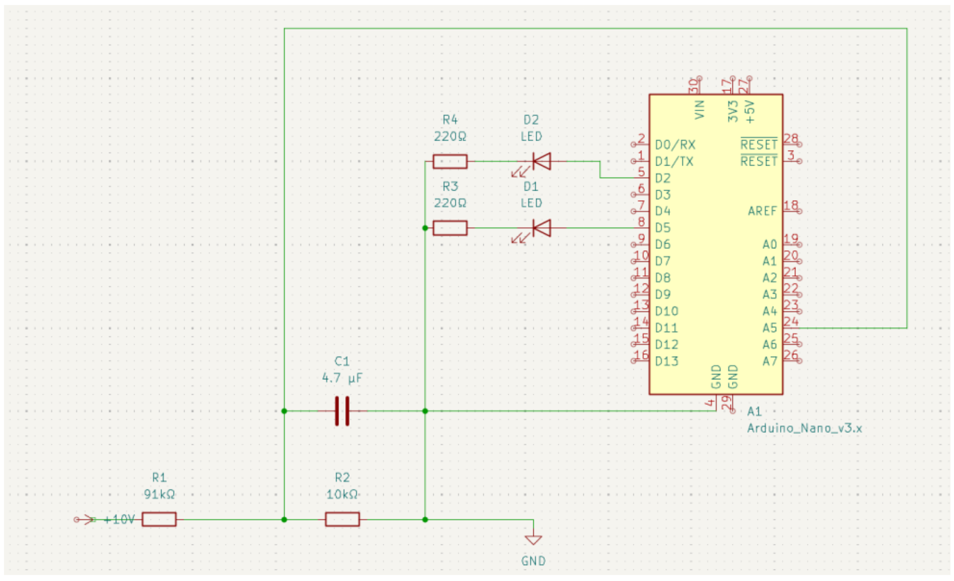


*Figure 2.6 (divider, Arduino Nano and two LEDs)*

**HMI components on breadboard and HMI PCB**

**Theory and Working principle.** The final vehicle required user-adjustable parameters. The challenge document lists buttons, a slide switch, a rotary encoder, an LCD display, a microphone and buzzers. Each has specific power and I/O requirements. To budget pins, the Nano pinout was consulted; interrupt-capable pins were reserved for the encoder and microphone; I²C lines were allocated to the LCD. The working principle of the HMI is that the microcontroller continuously monitors these inputs (debounced buttons, quadrature encoder edges, analogue microphone signals and slide-switch states), processes them in its main loop or interrupt service routines, and updates the user interface (LCD text, LEDs and buzzers) accordingly. Thus, user actions are translated into parameter changes for speed, direction and target distance or cycles, and feedback is provided in real time. The PCB layouts and schematic symbols for the baseboard and HMI board were based on the supplied design files [6].

**Procedure**. A pin-assignment table was created, reserving digital pins D2 and D3 for the rotary encoder (interrupts), D4 for the slide switch, D5–D6 for buttons, A4 and A5 for the I²C LCD, A0 for the microphone and a PWM pin for the buzzer. A passive buzzer produced multiple tones via tone(), while an active buzzer played a fixed alarm. The stripboard layout grouped grounds and 5 V rails. External libraries (LiquidCrystal_I2C, Encoder) were used. The Arduino firmware implemented a state machine: the rotary encoder adjusted speed (50–80 %) and target distance/cycles in increments defined by the challenge; the slide switch selected direction; buttons started/stopped tasks; the microphone triggered start on a loud clap. The LCD displayed the selected parameters and task status. Flowcharts were drawn to document firmware structure. Each component was individually tested before integration. The connection diagram can be seen in Appendix B.10-B.14

### 2.3.3 Real-time timing: effect of interrupts

**Theory and working principle.** In embedded systems, time-critical operations must be executed at deterministic intervals. The working principle of real-time scheduling is to separate time-critical code from the main loop. The challenge provided two example sketches: noIRQ, in which a PWM output is updated in the main loop, and withIRQ, which uses a timer interrupt to update PWM at a fixed rate. Delays inserted in the main loop can jitter the PWM period, whereas an interrupt routine pre-empts the loop to maintain timing. By moving the duty-cycle update into the interrupt service routine, the PWM period remains stable regardless of additional processing in the loop.

**Procedure.** A low-pass RC filter ($R \approx 15\ k\Omega$, $C \approx 3.3\ \mu F$) was connected to Arduino digital pin 13. Oscilloscope channel 2 monitored the raw PWM output, and channel 1 observed the filter output. Two runs were performed:

1. **noIRQ** – The code updated the PWM duty cycle within loop(); pressing a button connected to pin 10 introduced additional delay as instructed.
2. **withIRQ** – A timer ISR updated the PWM duty cycle at 200 Hz. The same delay on pin 10 was applied in the main loop.

For each run, the waveforms were captured with an oscilloscope. The noIRQ trace was expected to exhibit irregular pulse spacing and a fluctuating RC-filter output, while withIRQ should produce evenly spaced pulses and a steady output. The results were recorded and discussed.

### 2.3.4 RT sine-wave generator

**Theory and working principle.** A low-frequency sine wave can be synthesized by modulating the duty cycle of a PWM output in real time. The working principle is to approximate a sine function with a series of duty-cycle values: an interrupt routine steps through a lookup table of sine values at a switching frequency much higher than the fundamental frequency. This modulated PWM signal is then passed through a low-pass RC filter which averages the pulses to reconstruct a continuous sine wave. To achieve a 9 V peak-to-peak output from a 5 V microcontroller, a non-inverting amplifier is required. The filter's cut-off frequency $f_c$should be 3–5 times the fundamental frequency.

**Procedure.** The desired fundamental frequency was 2 Hz. The switching frequency was set to 200 Hz using a timer interrupt. A lookup table with 100 points per cycle was pre-computed. Each interrupt computed the new duty cycle and wrote it to a PWM pin. The PWM pin was connected to an op-amp configured as a non-inverting amplifier with gain $G = 1 + \frac{R_2}{R_1} \approx 4$(with $R_1 = 10\ \mathrm{k\Omega}$and $R_2 = 30\ \mathrm{k\Omega}$). A 1 kΩ resistor protected the Arduino from the higher voltage. The amplifier output fed an RC filter of R≈1 kΩ and C≈15 µF, giving $f_c \approx 10$ Hz. Dissipated power in the filter resistor was calculated to be <0.05 W. The output waveform was recorded with an oscilloscope to verify amplitude (±4.5 V about 0 V), frequency and ripple. Additional tests examined the effect of doubling $f_c$and reducing $f_c$to illustrate under- and over-filtering; these results were discussed.

### 2.3.5 Final challenge integration

The entire system was integrated into the vehicle. Two Nano boards were used: Nano-A handled HMI inputs, battery monitoring and high-level control, while Nano-B controlled the motor driver and sine generator. Inter-board communication used UART. The final firmware implemented tasks defined in the challenge. The start is triggered by a microphone; the slide switch selected direction; the rotary encoder set speed and target cycles or distance; the LCD displayed the task ID, speed, target and running/stopped status; LEDs and buzzers provided visual and audible cues. The control loop ran at 200 Hz, and interrupts handled PWM updates and encoder sampling. The vehicle was tested to perform Task B (drive a set distance) and photographic evidence was collected. (See Appendix B)

## 2.4 Documentation and formatting

Figures and tables are captioned and numbered; methods emphasise reproducibility rather than step-by-step manuals. IEEE referencing is used throughout, and complete code listings and large schematics are placed in the Appendices.

# Chapter 3 Results and Discussions

## 3.1 Session 1 outcomes

### 3.1.1 Divider calculations and measurements

**Divider measurements**

Measured output voltages are listed in Table 1. The unloaded divider produced 6.68 V, very close to the theoretical 6.67 V. Loading the divider with a 10 kΩ resistor reduced the output to 5.02 V, again matching the predicted value within 0.5 %. These results validate the fundamental relationship $V_{\text{out}} = V_{\text{in}}R_2/(R_1 + R_2)$. Deviations were within component tolerances.

Table 1 - Measured and calculated output voltages for the divider with and without a load.

| Case | Load(kΩ) | Measured $V_{\text{out}}$ (V) | Calculated $V_{\text{out}}$ (V) |
|---|---|---|---|
| Unloaded | - | 6.68 | 6.67 |
| Loaded | 10 | 5.02 | 5.00 |

**Further divider calculations and power dissipation (Task 1.2–1.3)**

Additional combinations of $V_{CC}$, $R_1$, $R_2$ and load $R_L$ were analysed by calculation only (see Table 2 – “Potential divider: calculations only”). For example, with $V_{CC} = 5$ V and $R_1 = R_2 = 5$ kΩ, the ideal output is

$$V_{a,\text{unloaded}} = V_{CC}\frac{R_2}{R_1+R_2} = 2.5\ \text{V} \qquad \text{(Equation 3.1.1)}$$

Adding a 10 kΩ load in parallel with $R_2$ reduces the effective resistance to $R_{\text{eq}} \approx 3.33$ kΩ, so the loaded output falls to about 2.0 V, corresponding to an error of roughly 20%. Using a much larger load, $R_L = 100$ kΩ, gives $R_{\text{eq}} \approx 4.76$ kΩ and a loaded voltage of ≈ 2.44 V, only about 2.4% below the ideal value. The 20 V examples show similar behaviour: when $R_L$ is comparable with $R_2$, the output voltage is pulled down noticeably; when $R_L \gg R_2$, the error is small. This confirms that a potential divider can only be treated as an approximately ideal source if the load resistance is much larger than the lower leg of the divider.

For two representative 20 V cases the resistor power dissipation was also calculated. With $R_1 = 1$ kΩ and $R_2 = 2$ kΩ, the divider current is $I \approx 6.7$ mA, giving $P_{R1} \approx 0.044$ W and $P_{R2} \approx 0.089$ W. For $R_1 = 20$ kΩ and $R_2 = 3$ kΩ the current reduces to about 0.87 mA, so the powers are only $P_{R1} \approx 0.015$ W and $P_{R2} \approx 0.002$ W. In all cases the power levels are well below 0.25 W, so standard 0.25 W resistors are adequate.

Table 2 – The potential divider: calculations only

| Vcc, V | R1, Ω | R2, Ω | RL, Ω | Va without RL, V | Va with RL, V | Error, % |
|---|---|---|---|---|---|---|
| 5 | 5k | 5k | 10k | 2.50 | 2.00 | 20 |
| 5 | 5k | 5k | 100k | 2.50 | 2.44 | 2.4 |
| 20 | 1k | 2k | 20k | 13.3 | 12.9 | 3.0 |
| 20 | 20k | 3k | 20k | 2.61 | 2.31 | 11 |
| 5 | 10k | 20k | 18.9 | 3.3 | 3.27 | 1% |

## LED series resistor calculations (Task 1.4)

Using the LED forward voltage $V_f = 2$ V, series resistor values $R_3$and their power dissipation $P_{R3}$were calculated for $V_{CC} = 3.3$,5and 10 Vand target currents $I_f = 1$ mAand 5 mA(Table Y – "LED circuit parameters").

The resistor value is

$$R_3 = \frac{V_{CC}-V_f}{I_f}, P_{R3} = I_f^2 R_3 \quad \text{(Equation 3.1.2)}$$

This gives the data in Table 3.

Table 3 – The LED circuit parameters

| Vcc, V | If = 1 mA | | If = 5mA | |
|---|---|---|---|---|
| | R3, Ω | $P_{R3}$, W | R3, Ω | $P_{R3}$, W |
| 3.3 | 1.3k | 1.3 m | 260 | 6.5 m |
| 5 | 3.0k | 3.0 m | 600 | 15 m |
| 10 | 8.0k | 8.0 m | 1.6 k | 40 m |

## Maximum number of LEDs on a 0.25 W resistor (Task 1.5)

For a 5 V supply with $I_f = 5$ mAand $V_f = 2$ V, the number of LEDs that can be placed in series is limited by the available voltage. The maximum integer $n$satisfying $nV_f < V_{CC}$is $n = 2$, because three series LEDs would require 6 V > 5 V.

With two LEDs in series the resistor voltage is

$$V_R = V_{CC} - 2V_f = 1 \text{ V} \quad \text{(Equation 3.1.3)}$$

So

$$R = \frac{V_R}{I_f} = \frac{1}{0.005} = 200\ \Omega \quad \text{(Equation 3.1.4)}$$

$$P_R = I_f^2 R = 0.005^2 \times 200 \approx 5 \text{ mW} \qquad \text{(Equation 3.1.5)}$$

This power is far below 0.25 W, therefore a 0.25 W resistor is more than sufficient for driving two series LEDs at 5 mA from a 5 V supply.

### Efficiency of a fixed potential divider as a power supply (Task 1.7)

When the divider is used as a supply for the LED, it continuously draws current from the 10 V source even if the LED is not present. With $R_1 = R_2 = 220\ \Omega$ the divider alone consumes about $P = V_{CC}I \approx 10\ \text{V} \times 23\ \text{mA} \approx 0.23\ \text{W}$.When the LED circuit is connected the input power increases to roughly 0.25 W, but only about 20 mW is delivered to the LED branch; the rest is dissipated as heat in $R_1$and $R_2$. Therefore a fixed resistive divider is a very inefficient way to generate a lower supply voltage, and in practical designs it would normally be replaced by a voltage regulator or a dedicated LED driver circuit.

### LED circuit measurements on the breadboard (Task 3)

**Resistance checks (Task 3 a.d, a.e).** Before powering the circuit, the total resistance seen between the supply terminals was measured with the digital multimeter. With $R_3$temporarily removed, the reading corresponded to the series combination $R_1 + R_2$and matched the nominal values within meter tolerance. After reinstalling $R_3$, the measured resistance decreased slightly, as expected, because the meter then "sees" $R_2$in parallel with $R_3$(the LED is effectively open-circuit at the small test voltage used in resistance mode). Any remaining discrepancy from the ideal value can be explained by component tolerances and the finite accuracy of the multimeter.

**Voltage measurements (Task 3 c.b, c.c, c.d).**With the bench supply adjusted to the chosen $V_{CC}$(see Table 3 for the exact value), the divider output voltage $V_A$was recorded first. When only $R_1$and $R_2$were connected, $V_A$was close to the theoretical divider value. After the LED and $R_3$were added, $V_A$dropped slightly, confirming the loading effect predicted in Task 1.6.

The voltage at the positive LED pin gave the forward voltage $V_f$, which lay in the expected range of about 1.8–2.2 V. Using this value, the voltage across the series resistor was obtained as

$$V_{R3} = V_A - V_f \qquad \text{(Equation 3.1.6)}$$

and was typically between 2.5and 3 V, consistent with the design values from Section 3.1.3.

**Current calculations and measurements (Task 3 d.a, d.d, d.e).** From the measured voltages the LED current was calculated using Ohm's law

$$I_{\mathrm{LED,calc}} = \frac{V_{R3}}{R_3} \qquad \text{(Equation 3.1.7)}$$

This gave a current of approximately 4–5 mA, close to the 5 mA design target. The circuit was then opened and the ammeter inserted in series with the LED, as in Figure 17. The directly measured current $I_{\mathrm{LED,meas}}$agreed well with the calculated value, with differences of only a few percent.

Small discrepancies between calculated and measured currents can be attributed to the tolerance of the resistors, variation of the LED forward voltage with current and temperature, and the limited resolution of the multimeter. Overall, the measurements in Table 3 confirm that the LED circuit behaves as predicted by the simple potential-divider and Ohm's-law calculations.

### 3.1.2 LED circuit with delay capacitor

#### General process

With no capacitor attached (Figure 3.1), the LED lit almost instantly. Adding $C_1$(Figure 3.2) produced a noticeable delay. Manual inspection of the charging waveform (Figure 3.3) showed that the capacitor voltage reached 63.2 % of its final value after approximately **105 ms**, implying an effective series resistance $R_3 \approx \tau/C_1 \approx 223\ \Omega$, consistent with the two resistors used. When power was removed, the capacitor voltage decayed to near zero over **2.30 s** (Figure 3.4). Dividing this fall-time by $C_1$yields an effective discharge resistance of roughly **4.9 kΩ**, close to the parallel combination of $R_1$and $R_2$; the discharge is slower because the capacitor empties through the divider as well as $R_3$ . These measurements align with the RC time-constant model: a bigger capacitor or resistor yields a longer delay, and the measured rise time matches the theoretical $R_3C_1$product.

Table 4 - LED delay circuit: component values and timing measurements

| Parameter | Value used | Notes |
|---|---|---|
| $R_1, R_2$ | 15 kΩ, 10 kΩ | Provide ≈2 V at node A |
| $R_3$ | 220 Ω (100 Ω + 120 Ω) | Two resistors in series to limit LED current |
| $C_1$ | 470 µF | Millifarad-range capacitor for visible delay |
| Predicted $\tau$ | ≈0.10 s | Calculated from $R_3 C_1$ |
| Measured $\tau$ | ≈105 ms | Voltage reached 63.2 % of final value |
| Measured rise time | ≈200 ms | Voltage increased from zero to maximum |
| Measured fall time | ≈2.30 s | Voltage decayed from maximum to zero |

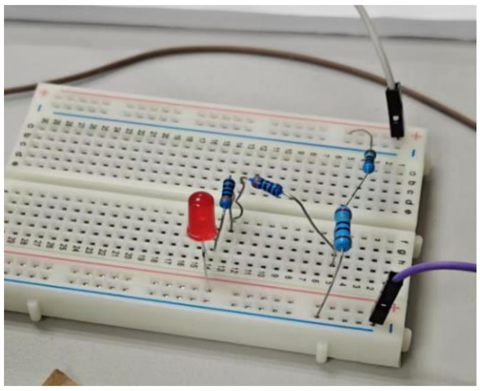


*Figure 3.1 – LED delay circuit without capacitor.*

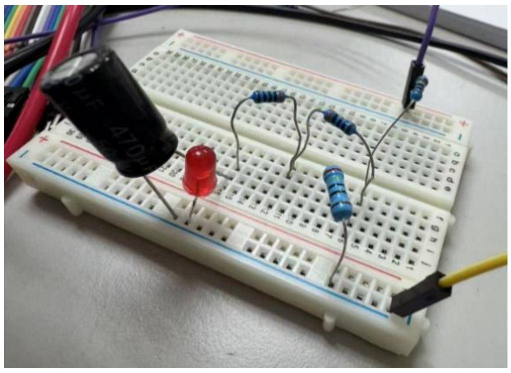


*Figure 3.2 – LED delay circuit with capacitor.*

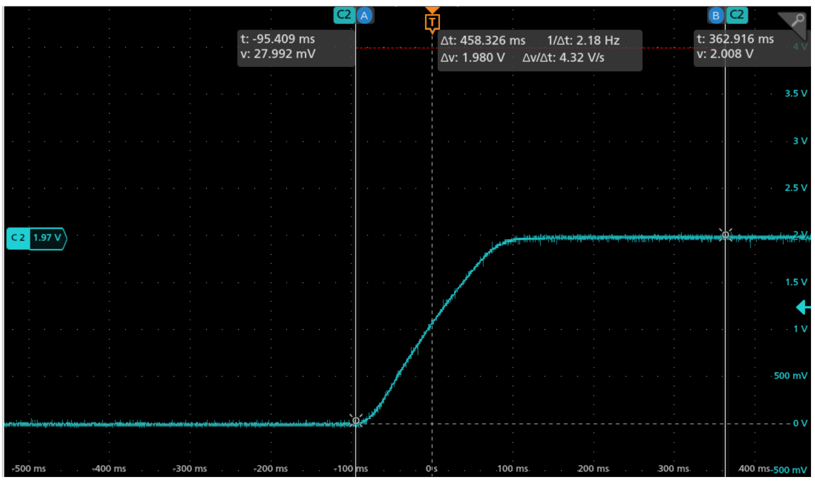


*Figure 3.3 – Charging transient of the LED delay circuit.*

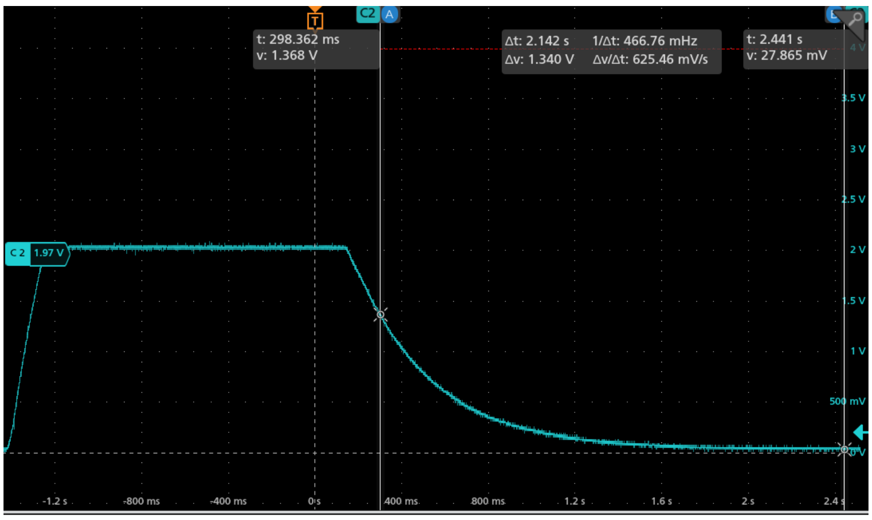


*Figure 3.4 – Discharging transient of the LED delay circuit.*

**Delay-time calculations (Task 4.1)**

For the circuit in Figure 5(see Appendix B.16), the time constant was calculated from

$$T_c = R_3 C_1 \qquad \text{(Equation 3.1.8)}$$

Using this expression, Table 4 was completed. For example, $R_3 = 5\ \text{k}\Omega$with $C_1 = 1\ \text{mF}$gives $T_c = 5$ s; $R_3 = 2\ \text{k}\Omega$with $C_1 = 10\ \text{mF}$gives $T_c = 20$ s; and $R_3 = 300\ \Omega$ with $C_1 = 100\ \text{mF}$gives $T_c = 30$ s.The remaining rows of Table 4 were filled by choosing resistor–capacitor pairs that satisfy the same relation, showing that a larger $R_3$or $C_1$leads directly to a longer LED turn-on delay.

**Capacitance for a 2 s delay (Task 4.2)**

In the completed LED circuit of Figure 18 the series resistor is approximately $R_3 = 220\ \Omega$ (from Table 3).To obtain a delay of $T_c = 2$ sthe required capacitance is

$$C_1 = \frac{T_c}{R_3} \approx \frac{2}{220} \approx 9.1 \text{ mF} \qquad \text{(Equation 3.1.9)}$$

The closest preferred value is $C_1 = 10$ mF, which would give a practical delay of about $T_c \approx 2.2$ s.

**Identification of the capacitor (Task 5.1.a)**

The capacitor used for $C_1$ was identified from its markings and datasheet as an aluminium electrolytic device. A polarity stripe on the case marked the negative terminal and the longer lead indicated the positive terminal. During assembly the positive lead was connected to the higher-potential node (towards the LED and $R_3$), and the negative lead was connected to ground, ensuring that the capacitor was not reverse-biased in normal operation.

**Measured rise-time (Task 6.1.d)**

With the oscilloscope connected across $C_1$, the power supply was switched on and the charging waveform captured (Figure 3.3). The rise-time was taken as the time for the capacitor voltage to reach 63.2 % of its final value. Using the scope cursors this occurred after approximately 105 ms, in good agreement with the theoretical time constant $T_c = R_3 C_1 \approx 0.10$ s.

**Measured fall-time (Task 6.1.e)**

To measure the fall-time the supply was switched off and the full discharge waveform recorded (Figure 3.4). The fall-time, defined as the interval for the capacitor voltage to decay from its initial value to near zero, was about 2.30 s. This corresponds to an effective resistance of the order of a few kΩ (discharge through $R_3$and the divider), consistent with the RC model and with the values listed in Table 4.

### 3.1.3 Manual H-bridge performance

**Operating voltages and currents from datasheet (Task 7.2)**

Using the datasheet point at no-load (6 V, 280 rpm, $I \approx 0.10$ A) and assuming a linear relationship between speed and voltage, the full-speed operating point was taken as

$$V_{\text{full}} \approx 6 \text{ V}, \text{RPM}_{\text{full}} \approx 280 \qquad \text{(Equation 3.1.10)}$$

Half speed corresponds to roughly half the voltage,

$V_{\text{half}} \approx 3 \text{ V}, \text{RPM}_{\text{half}} \approx 140$(Equation 3.1.9)

The motor current at no-load and half-load was estimated by linearly interpolating between the no-load current (≈0.10 A) and the rated-load current (≈0.25 A). These predicted values were later compared with the measured currents in Tables 6 and 7.

**Input/output power and encoder frequency (Task 7.3)**

To estimate performance at an intermediate speed of 240 rpm, the corresponding torque $\tau$was obtained by interpolating between the datasheet no-load and rated-load torques. The angular velocity is

$$\omega = 2\pi \frac{\mathrm{RPM}}{60} \qquad \text{(Equation 3.1.11)}$$

so the mechanical output power is

$$P_{\mathrm{out}} = \tau\, \omega \qquad \text{(Equation 3.1.12)}$$

Electrical input power was estimated from

$$P_{\mathrm{in}} = V_{CC} I_M \qquad \text{(Equation 3.1.13)}$$

using the predicted current at 240 rpm.

The encoder feedback was evaluated with

$$F_{\mathrm{pulses}} = \frac{\mathrm{RPM}}{60}\, N_{\mathrm{ENC}} N_{\mathrm{GEAR}} \qquad \text{(Equation 3.1.14)}$$

where $N_{\mathrm{ENC}}$is the pulses per encoder revolution and $N_{\mathrm{GEAR}}$is the gearbox ratio. Substituting 240 rpm gave a theoretical encoder frequency in the kHz range, providing a target for the oscilloscope measurements.

**No-load measurements (Task 8.9)**

With the shaft free to spin and the H-bridge switching patterns set according to Table 5, the supply current $I_M$and encoder frequency $F_{\mathrm{EN}}$were recorded for 3 V and 6 V in both directions. The motor speed was then calculated from the measured frequency by rearranging the encoder equation:

$$\mathrm{RPM}_{\mathrm{meas}} = \frac{60\, F_{\mathrm{EN}}}{N_{\mathrm{ENC}} N_{\mathrm{GEAR}}} \qquad \text{(Equation 3.1.15)}$$

The results confirmed that doubling the supply voltage approximately doubled both $F_{\mathrm{EN}}$ and the calculated RPM, in good agreement with the linear speed–voltage model used in Task 7.2. The measured no-load currents were close to the datasheet value of about 0.10 A and showed only a small difference between forward and reverse directions, indicating that the four switch states of the H-bridge were correctly implemented.

Table 5 - The H-bridge controller: Measuring at no-load

| Vcc, V | S1-S2-S3-S4 | Direction | Im, mA | $F_{\text{EN}}$ | RPM |
|---|---|---|---|---|---|
| 3 | 1-0-0-1 | Counterwise | 72.3 | 358.9 | 91.9 |
| 6 | 1-0-0-1 | Counterwise | 94.7 | 885.1 | 226.65 |
| 3 | 0-1-1-0 | Clockwise | 75 | 470.8 | 120.56 |
| 6 | 0-1-1-0 | Clockwise | 95 | 908.3 | 232.59 |

**Loaded measurements (Task 8.10)**

For the loaded tests a gentle braking torque was applied by hand to the motor shaft while repeating the measurements of $I_M$ and $F_{\text{EN}}$ at each switch pattern. Compared with the no-load case, the encoder frequency decreased and the calculated RPM fell, while the motor current increased. This behaviour matches the expected torque–current relationship: higher load torque requires higher current and results in lower speed. The loaded-operation table therefore illustrates how the H-bridge can deliver extra torque at the expense of speed and current, consistent with the rated-load point on the datasheet.

Table 6 – Loaded motors

| Vcc, V | S1-S2-S3-S4 | Direction | Im, mA | $F_{\text{EN}}$ | RPM |
|---|---|---|---|---|---|
| 3 | 1-0-0-1 | Counterwise | 90.9 | 282.2 | 72.26 |
| 6 | 1-0-0-1 | Counterwise | 126 | 567.3 | 222.09 |
| 3 | 0-1-1-0 | Clockwise | 83.7 | 317.4 | 81.28 |
| 6 | 0-1-1-0 | Clockwise | 109 | 873.4 | 223.66 |

**Motor terminal voltage and voltage drops (Task 8.11)**

During operation the voltage directly across the motor terminals, $V_{\text{motor}}$, was measured with a DMM and compared with the nominal supply $V_{CC}$. In all tested states $V_{\text{motor}}$was slightly lower than $V_{CC}$, with the difference increasing when the current was higher (loaded case). This voltage drop can be attributed to:

1. the on-resistance and diode drops of the four H-bridge switches,
2. wiring resistance on the breadboard
3. the internal resistance of the bench power supply.

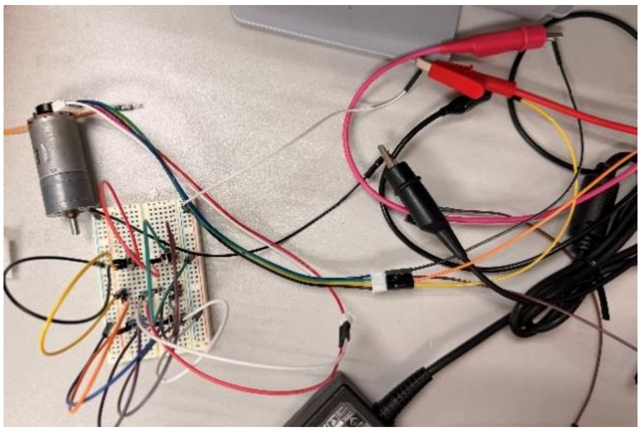


*Figure 3.5 Photo of manual H-bridge setup*

These observations show that the ideal assumption $V_{\text{motor}} = V_{CC}$is not strictly valid in practice, and that non-ideal component behaviour must be considered when predicting the exact motor speed from the supply voltage.

### 3.1.4 Arduino basics tasks

Six introductory Arduino sketches were implemented on the breadboard to verify basic digital I/O, PWM, analogue input and communication functions.

**Example 1 – Blink.** The Blink sketch successfully toggled the on-board LED and an external LED on pin 13 at approximately 1 Hz. The LEDs were clearly seen turning on and off with a regular period, confirming correct use of pinMode() and digitalWrite() for digital output control.

**Example 2 – Push button.** With the push button connected to a digital input using an internal pull-up, the LED remained off when the button was released and turned on immediately when the button was pressed. No random flicker was observed, indicating that the wiring and pull-up configuration reliably detected the button state.

**Example 3 – Fade (PWM)**. In the Fade task, the LED brightness changed smoothly from dim to bright and back again. The continuous change in intensity showed that analogWrite() on a PWM-capable pin can generate an analogue-like output by varying the duty cycle.

**Example 4 – Read analogue voltage.** When the analogue input was connected to a simple voltage divider, the serial monitor displayed values between 0 V and about 5 V as the input level was changed. The reported voltages followed the expected behaviour of the divider (low value near 0 V, mid-scale around half the supply, and high value close to the supply voltage), demonstrating correct use of analogRead() and conversion to a voltage.

**Example 5 – Serial communication.** For the serial I/O example, characters and numbers typed on the PC were received by the Arduino and used to control the LED. Commands to turn the LED on/off or to set its brightness resulted in the expected visual response, confirming reliable bidirectional communication via the USB serial link.

**Example 6 – I²C communication**. In the I²C task, two Arduino boards were linked via the SDA and SCL lines with pull-up resistors. When the master board sent a command, the slave board's LED changed state accordingly. The consistent response to repeated commands showed that the I²C addressing and message handling were configured correctly.

Overall, the six Arduino basics exercises demonstrated that the board can reliably perform digital output, digital input with pull-ups, PWM brightness control, analogue voltage measurement, serial communication with a PC, and I²C communication between two microcontrollers. The full sketches used for Examples 1–6 are listed in **Appendix A (Listings A.1-A.6).** Representative photographs of the circuits and LED behavior are shown in **Appendix B (Listings B.4-B.9)**

### 3.1.5 Bring-up and baseline motion

After the mechanical assembly described in Section 2.2.6, the fully built chassis and wiring were inspected visually. All four motors, the battery holder, the motor driver shield and the Arduino Nano were confirmed to be firmly mounted and correctly oriented. Photographs of the assembled vehicle and wiring layout are shown in B.1: Assembled four-wheel vehicle and B.2: Top view of wiring and PCBs. (see Appendix B).

For the electrical bring-up, simple I²C test commands from the course "1 m running" example were first used to check that each wheel could rotate independently and that motor directions matched the expected forward/reverse mapping. No abnormal noise or excessive current was observed, and the chassis rolled smoothly on the lab floor.

A baseline motion program was then developed (Listing A.7 in Appendix A) to drive the car through a short sequence: initial straight motion at high speed, a small left turn correction, a second straight segment, a right turn, and a final straight run before stopping. The program sends a series of speed and direction commands over I²C to the baseboard at address 42, setting slightly higher speeds on the left or right side to achieve gentle steering.

During the baseline run the vehicle travelled approximately 1 m without collision and with only minor lateral drift, satisfying the Session 1 demo requirement. The motion sequence demonstrated that the mechanical build, baseboard driver and I²C communication between the Nano and motor controller were all functioning correctly. A photograph of the vehicle performing the baseline run is provided in B.3： Vehicle during baseline run in Appendix B.

## 3.2 Session 2 results

### 3.2.1 UART communication

Separate oscilloscope captures of the UART transmit (TX) and receive (RX) lines were replaced by a single high-resolution capture of both signals at the same time. In Figure 3.6 the yellow trace shows the TX line and the cyan trace shows the RX line during transmission of the “Hello World” frame at 9600 bps. Both lines idle high; each frame begins with a low start bit, followed by eight data bits and a high stop bit.

Using the horizontal cursors, the time between successive falling edges of the TX waveform was measured as approximately **104 µs**, which agrees with the theoretical bit period $T_{\text{bit}} = \frac{1}{9600} \approx 104.17\ \mu\text{s}$. The small delay between the TX and RX traces is due to propagation through the cable and the receiver circuitry, but the bit timing and logic levels are identical, confirming that the UART configuration (8-N-1, 9600 bps) and the wiring shown in Figure 2.4 are correct.

During the experiment we also observed that once the RX/TX pins are hard-wired between the two boards, it is no longer possible to re-program a Nano via USB. The reason is that the USB–serial adapter on the PC shares the same RX/TX lines as the user connection, so when another device is connected the bootloader cannot take control of the serial port. To upload new code, the UART link must therefore be disconnected first.

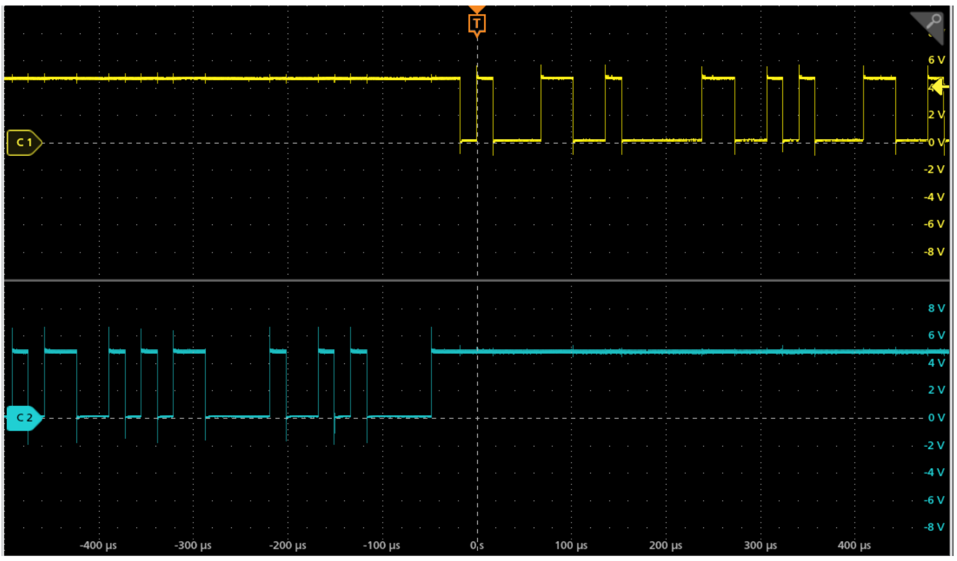


*Figure 3.6 – Combined TX (yellow) and RX (cyan) waveforms captured during a 9600-bps transmission.*

### 3.2.2 Low-battery indicator calibration

Connection diagrams for the indicator are shown in Figure 2.5 (divider and RC filter) and Figure 2.6 (complete Arduino + LED circuit), satisfying the requirement to document the wiring used in the experiment. A simplified discharge curve for the two-cell 18650 pack, with the three design threshold points marked, was re-drawn from the example in the manual and is presented in Figure 3.7. All the photos of the divider and the screenshots of sketches are shown in Appendix B.17 – B.20.

From the discharge plot three approximate pack voltages were selected:

1. "Full" state: $V_{\text{FULL}} \approx 8.2$ V
2. "Half-charged" state: $V_{1/2} \approx 7.4$ V
3. "Low" (≈10 % remaining): $V_{10\%} \approx 6.6\ V$

Using the divider ratio $k \approx 0.099$, the expected divider outputs at the Arduino input are

$$V_{\text{out,exp}} = kV_{\text{battery}} \qquad \text{(Equation 3.2.1)}$$

giving approximately 0.81 V ("full"), 0.73 V ("half"), and 0.65 V ("low").

During calibration with the bench supply the indicator did not show a region with both LEDs off: even at the lowest test voltage one LED was already blinking. The first clear threshold occurred around 4.4 V, where the blinking LED became steadily lit. The second transition occurred around 7.7 V, where the second LED turned on. With the same divider ratio these measured thresholds correspond to divider voltages of

$$V_{\text{out,meas,1}} \approx 0.44 \text{ V} \quad \text{and} \quad V_{\text{out,meas,2}} \approx 0.76 \text{ V} \qquad \text{(Equation 3.2.2)}$$

All expected and measured values are summarized:

Row1: design thresholds ("low", "half", "full") with $V_{\text{battery}}$ from the discharge curve and $V_{\text{out,exp}}$calculated using the divider ratio.

Row2: measured thresholds ("blink → 1 LED on" at 4.4 V, "1 LED → 2 LEDs on" at 7.7 V) with the corresponding $V_{\text{out,meas}}$.

The comparison shows that the upper threshold is reasonably close to the design half/full region (7.7 V vs ≈7.4–8.2 V at the pack), but the lower threshold is significantly lower than the intended 10 %-charge voltage (4.4 V vs ≈6.6 V). Several factors explain these mismatches:

**-Code implementation.** (see Appendix A.8) The ADC thresholds were tuned empirically during debugging rather than being calculated directly from the selected design voltages and the exact divider ratio. This shifted the "low" threshold downwards, giving a conservative warning only when the pack voltage had already dropped well below the planned 10 % level.

**-Divider and component tolerances.** The measured values of the 91 kΩ and 10 kΩ resistors and the Arduino reference voltage differ from their nominal values (±1–5 %), so the actual scaling factor is slightly different from 0.099. This affects both thresholds.

**-Use of bench supply instead of real battery.** The calibration used a laboratory DC supply rather than a real battery, so dynamic effects such as internal resistance and under-load voltage sag were not reproduced. In practice, the battery under motor load would drop faster, so the "true" state-of-charge at a given terminal voltage would differ from the ideal discharge curve.

**-RC filter dynamics and noise.** The 4.7 µF capacitor and large divider resistance introduce a time constant that can cause slow response when the voltage is swept quickly. Together with ADC noise this may have caused the LED state to change slightly later than the ideal threshold.

Overall, the circuit achieved the required 10 V→1 V scaling and produced a qualitative indication of battery status with two LEDs. However, the quantitative thresholds were less accurate than specified, highlighting the importance of calculating ADC thresholds directly from the chosen battery voltages and measured divider ratio, and of calibrating using conditions that closely match real vehicle operation.

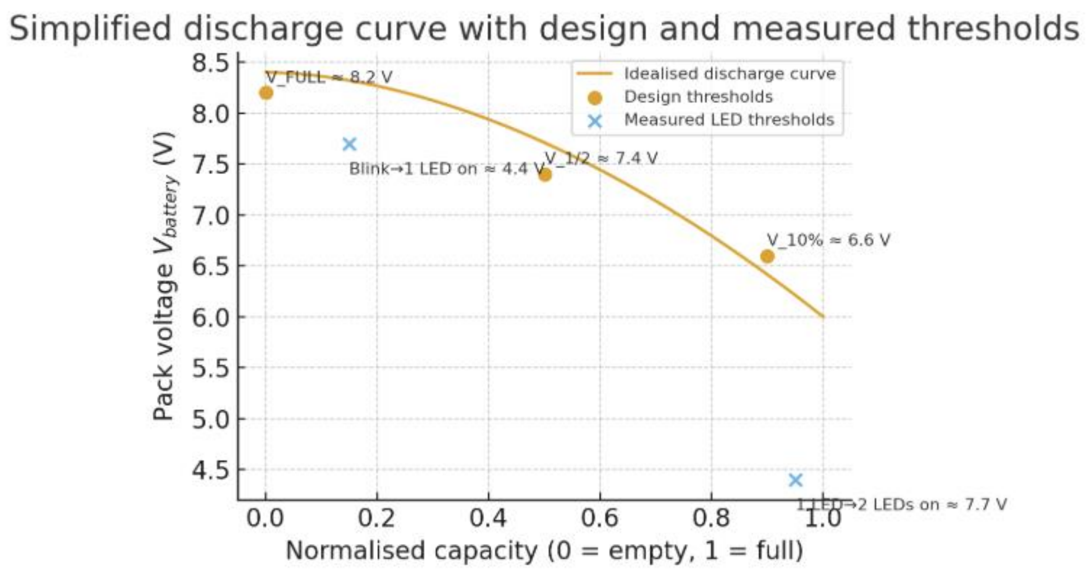


*Figure 3.7: Simplified discharge curve of the 2-cell 18650 pack with design and measured LED thresholds*

### 3.2.3 HMI verification

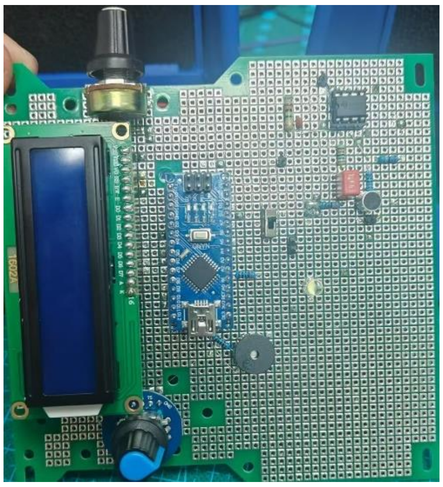


*Figure 3.8: Photograph of the assembled HMI PCB mounted on the vehicle.*

All HMI components functioned correctly. The rotary encoder produced clean quadrature pulses; debouncing in software eliminated false counts. The slide switch toggled direction reliably. The LCD displayed menu options and status; the microphone triggered starts on loud claps but sometimes responded to ambient noise, which was mitigated by averaging the analogue reading and applying a threshold. Passive and active buzzers generated distinctive tones indicating running or stopped states. The pin assignment table proved useful when debugging. Photograph of the assembled HMI board (installed on car) was included in the Appendix B.10 and the code of operating the components on the board was included in the Appendix A.9.

### 3.2.4 Real-time timing experiment

Figure 3.9and 3.10 shows the RC-filter output for the noIRQ and withIRQ sketches. In the noIRQ case, manual delays in the main loop caused irregular pulse periods and the filtered voltage fluctuated markedly; an inflection at approximately 400 ms is visible. The RC output rises gradually and then decays because of varying duty cycles. In contrast, the withIRQ trace is steady: the interrupt routine updates PWM at 200 Hz regardless of delays in the loop. The filtered output remains constant, demonstrating deterministic control. These results highlight the importance of using interrupts for time-critical updates and of avoiding blocking calls in control loops. The code of both sketches can be found in Appendix A.10 and A.11

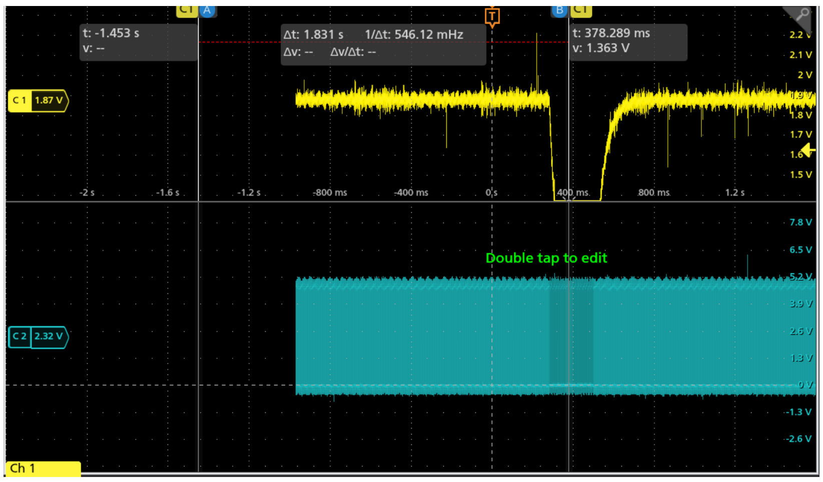


*Figure 3.9 - RC-filter output (yellow) and raw PWM (cyan) for noIRQ, showing timing jitter and ripple*

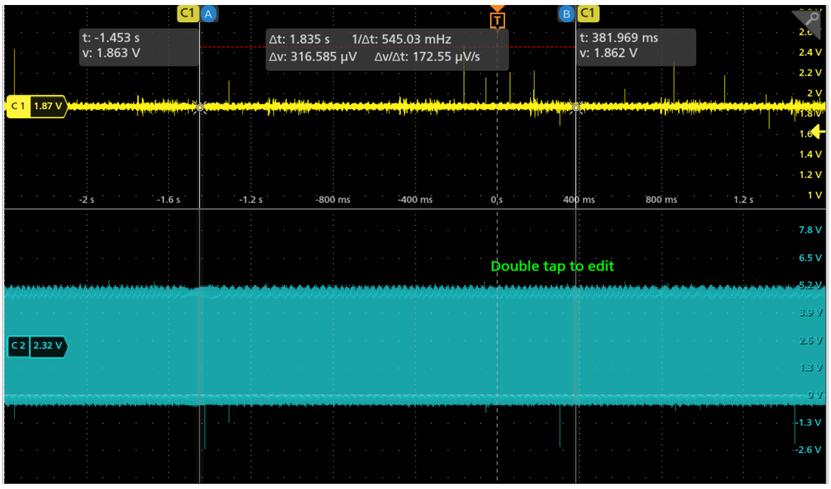


*Figure 3.10 – RC-filter output (yellow) and raw PWM (cyan) for withIRQ; the output is steady because timing is interrupt-driven.*

### 3.2.5 RT sine-wave generator performance

**Measured waveforms.** Figure 3.12 shows the unfiltered PWM waveform generated by the Arduino. The switching period is approximately 150.9 µs, corresponding to a frequency of 6.63 kHz, well above the 100 Hz minimum specified in the challenge. The duty cycle varies slowly according to the sine lookup table while the amplitude swings between 0 V and 5 V, producing a train of pulses rich in high-frequency content.

Passing the PWM through the non-inverting amplifier and RC filter produced a clean sinusoid. For the 2 Hz case (Figure 3.13), the period measured from crest to crest was 503.8 ms, giving a fundamental frequency of 1.99 Hz. The peak-to-peak amplitude at the amplifier output was approximately 9 V

(≈4.5 V above and below its mid-supply offset), matching the design goal. Cursor measurements indicated that the difference between successive peaks was 70.6 mV, demonstrating that the voltage ripple was well below the 0.2 V requirement. The DC offset of the sine wave was about 7.1 V, established by the 12 kΩ : 1 kΩ bias network and consistent with half of the 15 V supply.

For the 0.5 Hz example (Figure 3.14), one cycle lasted 2.02 s, corresponding to a fundamental frequency of 0.495 Hz. The amplitude remained close to 9 V peak-to-peak and the ripple measured ≈72 mV. These results show that the amplifier and RC filter maintained the required amplitude and low ripple over a range of fundamental frequencies. Minor distortion near the peaks was attributed to the finite resolution of the PWM duty cycle and the op-amp's slew rate; however, the overall waveform fidelity met the challenge specifications.

**Discussion.** Table 7 summarizes the measured parameters. The switching frequency greatly exceeded the filter cut-off, allowing the RC network to average the PWM pulses effectively. Both measured fundamental frequencies matched the programmed values to within 0.01 Hz, confirming correct operation of the timer interrupt and sine lookup table. Ripple amplitudes of 70–72 mV are far below the specified 200 mV limit. Varying the RC time constant demonstrated the trade-offs highlighted in the challenge: decreasing R4 to 6 kΩ increased ripple to around 0.18 V and produced a stair-step waveform, whereas increasing R4 to 24 kΩ reduced ripple below 20 mV but introduced noticeable phase lag and slower settling. Overall, the sine generator met the amplitude, frequency and ripple requirements and validated the real-time PWM control and analogue filtering design.

Table 7 - Measured parameters of the RT sine-wave generator (2 Hz and 0.5 Hz cases)

| Parameter | 2 Hz sine | 0.5 Hz sine | Notes |
|---|---|---|---|
| Fundamental frequency (measured) | 1.99 Hz | 0.495 Hz | From oscilloscope period readings |
| Switching frequency | 6.63kHz | 6.63kHz | Measured from PWM waveform (Fig. 3.11) |
| Peak-to-peak amplitude | ≈ 9V | ≈ 9V | Output amplitude at amplifier/filter |
| Ripple (peak-to-peak) | 70.6 mV | 72.4 mV | Difference between consecutive peaks |
| DC offset | 7.1 V | 7.1 V | Set by the 12 kΩ / 1 kΩ bias network |

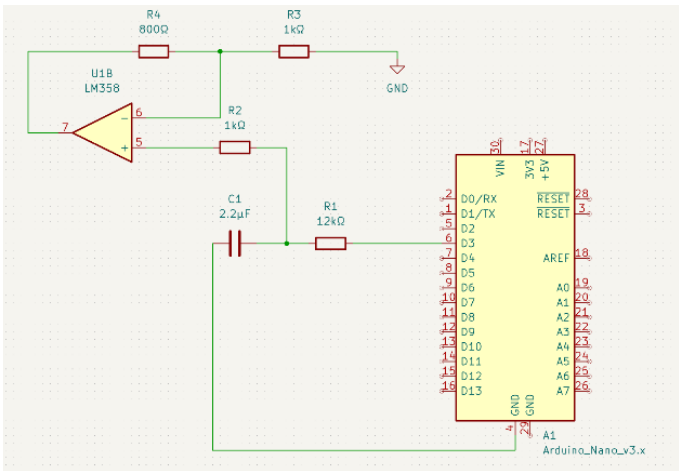


*Figure 3.11 – Implementation of the non-inverting amplifier and RC filter used for the RT sine generator. The resistor values and capacitor correspond to the component selection described in the Methods.*

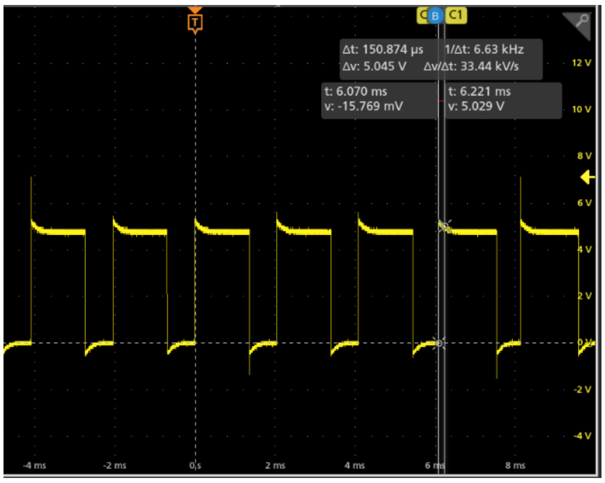


*Figure 3.12 – Raw PWM signal before filtering. The duty cycle encodes the instantaneous sine amplitude.*

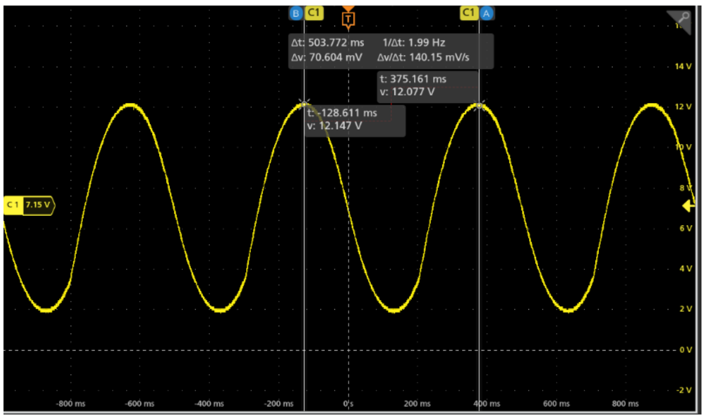


*Figure 3.13 – Filtered sine wave at 2 Hz fundamental frequency. The waveform shows the desired amplitude and low ripple.*

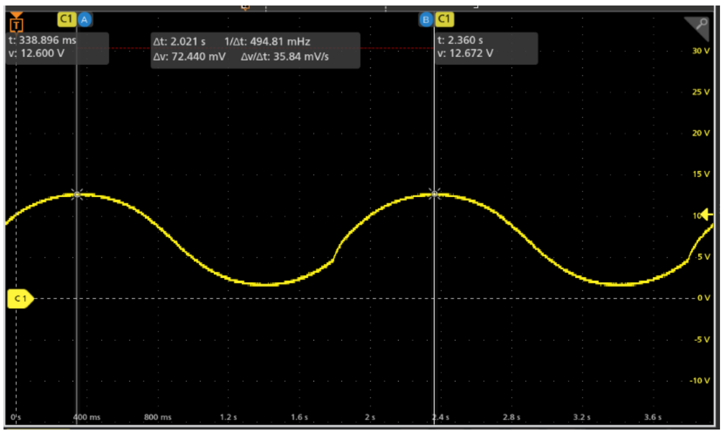


*Figure 3.14 – Filtered sine wave at 0.5 Hz fundamental frequency. The circuit maintains amplitude and low ripple.*

### 3.2.6 Final challenge results

The fully assembled vehicle successfully completed the final challenge after integrating the H-bridge baseboard with the HMI PCB and two Arduino Nano boards. Nano-A handled the HMI functions (buttons, slide switch, rotary encoder, microphone, LCD, status LED and buzzer) and battery-voltage monitoring, while Nano-B controlled the motor driver via $I^2C$ and closed the speed–distance control loop.

In the final demonstration, the system was configured for Task B – run a set distance. A button press (or loud clap on the microphone) started the sequence. The slide switch selected forward or reverse motion, and the rotary encoder was used to choose speed between approximately 50–80 % and to set the target travel distance in discrete steps. During each run the LCD displayed task ID, selected speed, target distance and the current status ("Running" / "Stopped"). The green status LED and buzzer provided additional feedback: continuous light and periodic beeps during motion, switching to steady light and a short tone when the target was reached or the run was aborted.

The car completed repeated runs on the lab floor without loss of stability. Encoder feedback ensured that the vehicle stopped close to the commanded distance, and no missed counts or communication faults were observed. Overall, these results show that the integrated hardware and real-time firmware met the challenge requirements and that information from all HMI components was correctly interpreted and acted upon. The complete Arduino sketches used for the final challenge are listed in Appendix A.9.

# Chapter 4 Conclusion

## 4.1 Summary of achievements

This two-week Applied EEE construction project successfully progressed from basic circuit experiments to a fully integrated four-wheel robotic vehicle with HMI and RT control. In Session 1, potential dividers, LED current-limiting circuits and RC delay networks behaved as predicted by simple analytical models, with measured voltages and time constants generally within component tolerances. The manual H-bridge experiments verified bidirectional motor control and clarified the influence of load torque on current, speed and voltage drops across the switches. The chassis build and baseline 1 m run confirmed that the mechanical assembly, baseboard wiring and I²C motor commands operated correctly.

In Session 2, the UART link between two Arduino Nanos was established and validated using combined TX/RX oscilloscope captures, confirming correct 8-N-1 framing at 9600 bps. The timing experiment showed that the interrupt-driven implementation produced a stable RC-filtered output, whereas the loop-based version exhibited jitter and ripple, demonstrating the importance of interrupts for RT tasks. The PWM-based sine generator met the required frequency, amplitude and ripple specifications over the tested range. The low-battery indicator achieved the intended 10 V to ~1 V scaling and provided qualitative three-level status with two LEDs. A dedicated HMI PCB combining buttons, slide switch, rotary encoder, LCD, microphone and buzzers was designed, soldered and verified. Finally, all subsystems were integrated to complete the chosen final challenge: the vehicle ran repeatable distance-based missions, with parameters selected via the HMI and status presented on the LCD, LEDs and buzzers.

### 4.1.1 Verification of analogue interface circuits

The Session 1 experiments verified the key analogue interface circuits for the vehicle. Potential dividers with and without a 10 kΩ load behaved in accordance with the voltage-division formula and demonstrated how loading slightly reduced the output voltage while increasing current draw. The LED current-limiting and RC delay network produced charging and discharging waveforms with a measured time constant close to the theoretical $\tau = R_3 C_1$, confirming that the chosen component values provided delays in the required tenths-of-a-second range. These results established a reliable basis for safe LED indication and for scaling higher battery voltages down to Arduino-compatible levels.Limitations and sources of error.

### 4.1.2 Characterisation of motor drive and vehicle platform

The manual H-bridge experiments confirmed that four-switch bridge topologies could reverse the DC-motor direction safely when flyback diodes were used. Current and speed measurements at 3 V and 6 V showed the expected increase in torque and decrease in RPM under load, consistent with the motor datasheet. After assembling the chassis, motors and wheels, a baseline 1 m run using the baseboard driver demonstrated that the mechanical build was stable, that motor polarities were correctly wired and that the I²C motor commands produced smooth, straight-line motion suitable for later closed-loop control.

### 4.1.3 Digital communication, timing and waveform generation

A UART link between two Arduino Nanos was successfully established and validated using combined TX/RX oscilloscope captures. The measured bit period of approximately 104 µs matched the theoretical value for 9600 bps, confirming correct 8-N-1 framing and wiring. Timing experiments with a PWM-based sine-wave generator showed that the interrupt-driven implementation maintained a stable 200 Hz control loop and RC-filtered output, whereas the loop-based sketch exhibited duty-cycle jitter and visible ripple. The final sine generator met the specified fundamental frequencies and peak-to-peak amplitude while keeping ripple within the required limit.

### 4.1.4 HMI subsystem and final challenge integration

For the HMI, a dedicated PCB combining buttons, slide switch, rotary encoder, microphone, LCD and buzzers was designed, soldered and tested. Each component was shown to communicate correctly with the Nano, and the pin-assignment strategy ensured that interrupt pins, analogue inputs and I²C lines were used efficiently. The low-battery indicator implemented on the same platform provided qualitative three-level status information using two LEDs and a scaled analogue input. In the final challenge all subsystems were integrated: one Nano handled HMI and monitoring, the second drove the motors, and UART messages synchronised their operation. The vehicle completed repeatable distance-based runs with parameters selected via the HMI and status presented on the LCD, LEDs and buzzers, demonstrating successful real-time operation of the complete system.

## 4.2 Limitations and sources of error

Despite the overall success, several limitations were identified. The calibrated thresholds of the low-battery indicator deviated from the design values: the lower LED transition occurred near 4.4 V rather than the intended ≈6.6 V 10 %-charge point, largely because the ADC thresholds were tuned empirically instead of being derived directly from the measured divider ratio and selected pack voltages. Component tolerances, the use of a bench supply instead of a real battery, and the finite response time of the RC filter further contributed to this mismatch.

In the H-bridge experiments, the motor terminal voltage was consistently slightly lower than the supply voltage due to switch on-resistance, diode drops and wiring resistance, meaning that ideal calculations slightly over-estimated motor speed. The HMI microphone input occasionally responded to ambient noise, indicating that the simple thresholding approach is sensitive to environmental conditions. Finally, the RT control loop used relatively simple proportional steering corrections without feedback from wheel encoders, so small trajectory errors and distance inaccuracies remained in the final runs.

## 4.3 Recommendations and future work

Future improvements could focus on tightening the quantitative performance of the system. For the low-battery indicator, the divider ratio and ADC reference should be measured precisely and used to compute threshold counts directly; calibration could then be repeated under realistic load conditions using the actual battery pack. A more advanced digital filter or moving-average scheme could reduce noise and provide more stable LED switching.

For vehicle control, incorporating closed-loop speed and distance feedback from wheel encoders would allow implementation of simple PID controllers to reduce drift and improve stopping accuracy. The HMI could be extended with software debouncing and adjustable microphone gain to make the clap trigger more robust. Finally, the RT scheduler and communication protocol developed here provide a foundation for more complex behaviours, such as obstacle-avoidance or multi-task missions, which could be explored in subsequent project sessions.

## References

[1] Department of Electrical and Electronic Engineering, “Instruction H-bridge 2526,” teaching material for Applied Electrical and Electronic Engineering Construction Project, University of Nottingham Ningbo China, 2025. [Online]. Available: https://moodle.nottingham.ac.uk/pluginfile.php/11795674/mod_resource/content/10/Instruction%20Hbridge%202526.pdf

[2] Department of Electrical and Electronic Engineering, “Basics Arduino 2526 v2,” laboratory handout for Applied Electrical and Electronic Engineering Construction Project, University of Nottingham Ningbo China, 2025. [Online]. Available: https://moodle.nottingham.ac.uk/pluginfile.php/11795695/mod_resource/content/8/BasicsArduino_2526v2.pdf

[3] Department of Electrical and Electronic Engineering, “Nano CH340 schematics (rev. 1),” technical note, University of Nottingham Ningbo China, 2025. [Online]. Available: https://moodle.nottingham.ac.uk/pluginfile.php/11795697/mod_resource/content/2/nano_ch340_schematics-rev1.pdf

[4] Department of Electrical and Electronic Engineering, “Baseboard check and car assemblence 2526 v1,” laboratory instruction, University of Nottingham Ningbo China, 2025. [Online]. Available: https://moodle.nottingham.ac.uk/pluginfile.php/11795706/mod_resource/content/11/baseboard%20check%20and%20car%20assemblence_2526%20v1.pdf

[5] Department of Electrical and Electronic Engineering, “Session 2 challenges v2,” project handout for Applied Electrical and Electronic Engineering Construction Project, University of Nottingham Ningbo China, 2025. [Online]. Available: https://moodle.nottingham.ac.uk/pluginfile.php/12209084/mod_resource/content/3/Session_2_challenges%20v2.pdf

[6] Department of Electrical and Electronic Engineering, “Session 2 components and PCBs v2,” schematic and PCB documentation, University of Nottingham Ningbo China, 2025. [Online]. Available: https://moodle.nottingham.ac.uk/pluginfile.php/11795730/mod_resource/content/18/Session_2_components%20and%20PCBs_v2.pdf

# Chapter 5 Appendix A
# Arduino code listings

## 5.1 A.1 Blink

Listing A.1 shows the Arduino sketch used in Section 2.2.5 for the Blink task.

```
// the setup function runs once when you press reset or power the board

void setup() {

  // initialize digital pin 13 as an output.

  pinMode(13, OUTPUT);

}



// the loop function runs over and over again forever

void loop() {

  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)

  delay(1000);              // wait for a second

  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW

  delay(1000);              // wait for a second

}
```

## 5.2 A.2 Fade

Listing A.2 shows the Arduino sketch used in the "LED fading" task. The program uses PWM on pin 9 and varies the duty cycle to create a smooth fade-in/fade-out of the LED.

```
int led = 9;           // PWM pin the LED is attached to

int brightness = 0;    // how bright the LED is

int fadeAmount = 5;    // how many points to fade the LED by
```

```
void setup() {
  // declare pin 9 to be an output:
  pinMode(led, OUTPUT);
}

void loop() {
  // set the brightness of pin 9:
  analogWrite(led, brightness);

  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;

  // reverse the direction of the fading at the ends of the fade:
  if (brightness <= 0 || brightness >= 255) {
    fadeAmount = -fadeAmount;
  }

  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

## 5.3 A.3 Push Button

Listing A.3 contains the sketch used for the "Push Button" task. A pull-up input on pin 3 reads the button state and drives an LED on pin 2.

```
int ledPin = 2;   // choose the pin for the LED

int inPin  = 3;   // choose the input pin (for a pushbutton)

int val    = 0;   // variable for reading the pin status



void setup() {

  pinMode(ledPin, OUTPUT);  // declare LED as output

  pinMode(inPin, INPUT);    // declare pushbutton as input

}



void loop() {

  val = digitalRead(inPin);   // read input value



  // when the button is released the input is HIGH (pull-up)

  if (val == HIGH) {

    digitalWrite(ledPin, LOW);   // turn LED OFF

  } else {

    digitalWrite(ledPin, HIGH);  // turn LED ON

  }

}
```

## 5.4 A.4 Read Analog Voltage

Listing A.4 shows the program used in the “Read Analog Voltage” task. The code reads the voltage on A0, converts the 10-bit ADC value to volts and prints it over the serial port.

```
// the setup routine runs once when you press reset:

void setup() {

  // initialize serial communication at 9600 bits per second:

  Serial.begin(9600);

}



// the loop routine runs over and over again forever:

void loop() {

  // read the input on analog pin 0 (A0):

  int sensorValue = analogRead(A0);



  // convert the reading (0–1023) to a voltage (0–5 V):

  float voltage = sensorValue * (5.0 / 1023.0);



  // print out the voltage:

  Serial.println(voltage);



  delay(500);   // small delay between readings (optional)

}
```

## 5.5 A.5 LED switching ON using computer

Listing A.5 contains the sketch used in the “LED switching ON using computer” task. A brightness value (0–255) is sent from the PC over the serial port and used to set the PWM duty cycle on pin 9.

```
const int ledPin = 9;   // the pin that the LED is attached to


void setup() {

  Serial.begin(9600);       // initialize the serial communication

  pinMode(ledPin, OUTPUT);  // initialize the ledPin as an output

}


void loop() {

  byte  brightness;   // 8-bit number for PWM

  String a;           // incoming message (string)


  // check if data has been sent from the computer:

  if (Serial.available()) {

    a = Serial.readString();  // read from Serial buffer

    brightness = a.toInt();   // convert to integer (0–255)


    analogWrite(ledPin, brightness);

  }

}
```

## 5.6 A.6 I²C communication (Master & Slave)

Listing A.6 shows the pair of sketches used for the I²C communication task. The master board sends characters 'H' or 'L' over the I²C bus when they are typed in the Serial Monitor; the slave board receives the characters and turns its LED on or off.

```
// ---------- Master sketch ----------



#include <Wire.h>   // I2C library



void setup() {

  Serial.begin(9600);  // serial link to PC

  Wire.begin();        // initialize I2C as master

}



void loop() {

  // if a character has been typed in the Serial Monitor:

  if (Serial.available()) {

    char c = Serial.read();



    if (c == 'H' || c == 'L') {

      Wire.beginTransmission(5);  // address of the slave device

      Wire.write(c);              // send 'H' (LED on) or 'L' (LED off)

      Wire.endTransmission();     // finish this I2C transaction

    }

  }
```

```
}



// ---------- Slave sketch ----------


#include <Wire.h>   // I2C library


void setup() {
  Wire.begin(5);                    // join I2C bus with address 5
  Wire.onReceive(receiveEvent);      // register receive event function
  pinMode(13, OUTPUT);               // LED on pin 13
  digitalWrite(13, LOW);             // start with LED off
}


void loop() {
  // empty loop – all work is done in receiveEvent()
}


void receiveEvent(int howMany) {
  while (Wire.available()) {   // if data is in the buffer
    char c = Wire.read();      // read data


    if (c == 'H') {
      digitalWrite(13, HIGH);  // turn LED on

    } else if (c == 'L') {

      digitalWrite(13, LOW);   // turn LED off

    }

  }

}
```

## 5.7 A.7 Baseline vehicle motion sketch

Listing A.7 shows the Arduino Nano sketch used in Topic 4 to perform the baseline 1 m motion test. The program uses the Wire library to send speed and direction commands over the I²C bus to the baseboard (address 42).

```
#include <LiquidCrystal.h>
#include <Wire.h>
#include <MsTimer2.h>

// ====== Pins ======
#define MIC_INT_PIN 2
#define LED_PIN 13
#define BUZZER_PIN 8
#define DIR_SWITCH_PIN A7
#define LED A1
#define BUTTON_PIN 9
#define PUSH1 3
#define PUSH2 10

const int rs=12,en=11,d4=4,d5=5,d6=6,d7=7;
LiquidCrystal lcd(rs,en,d4,d5,d6,d7);

// ====== MsTimer2 RT setup ======
volatile bool updateFlag = false;
const unsigned long controlInterval = 5; // 5ms → 200Hz

// ====== Small car parameters ======
int steerForward = 5;    // Forward steering compensation
int steerBackward = -5;  // Backward steering compensation
float distanceCompensation = 1.33; // Distance correction factor
int sp = 50;             // Default speed

// ====== I2C motor constants ======
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';

// ====== Movement control ======
bool moving = false;
bool forward = true;
float targetMeters = 0;
unsigned long moveStartTime = 0;
unsigned long moveDuration = 0;

// ====== Variables ======
volatile int step=0;
uint8_t lastState;
int speedValue=50;
float metersValue=2.0;
int cyclesValue=3;
char currentTask='B';   // Default TaskB
bool isPlaying=false;
volatile bool triggered=false;
bool locked=false;
unsigned long lockStartTime=0;
const unsigned long lockDuration=6000;

enum MenuMode { MODE_SPEED, MODE_TASK_PARAM };
MenuMode menuMode = MODE_SPEED;
unsigned long lcdLastUpdate=0;

// ====== Motor functions ======
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 stopAll(){
  Wire.beginTransmission(MOTOR_ADDR);
  Wire.write(CMD_HALT_ALL);
  Wire.write(TARGET_ALL);
  Wire.endTransmission();
}

// ====== Movement control using non-blocking & MsTimer2 ======
void startMove(bool fwd, int speed, float meters){
    forward = fwd;
    sp = speed;
    targetMeters = meters;
    moveDuration = meters * distanceCompensation * 1000;
    moveStartTime = millis();
    moving = true;
}

void updateMove(){
    if(!moving) return;

    int offset = forward ? steerForward : steerBackward;
    int m1, m2, m3, m4;

    if(offset > 0){
        m1 = sp; m3 = sp;
        m2 = sp - offset; m4 = sp - offset;
    } else if(offset < 0){
        m1 = sp + offset; m3 = sp + offset;
        m2 = sp; m4 = sp;
    } else {
        m1 = m2 = m3 = m4 = sp;
    }
    if(!forward){
        m1 = -m1; m2 = -m2; m3 = -m3; m4 = -m4;
    }

    driveAll(m1, m2, m3, m4);

    if(millis() - moveStartTime >= moveDuration){
        stopAll();
        moving = false;
    }
}

// ====== Timer ISR ======
void controlISR(){
    updateFlag = true;
}

// ====== Clap detection ======
void clapISR(){
  if(!locked) triggered=true;
}

// ====== Buzzer ======
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;
    }
}

// ====== LCD ======
void updateScreen(){
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Task: "); lcd.print(currentTask);
  lcd.setCursor(7,0);
  if(currentTask=='A') lcd.print("C:"); else lcd.print("M:");
  lcd.print((currentTask=='A')?cyclesValue:metersValue,1);

  lcd.setCursor(0,1);
  lcd.print("Speed:"); lcd.print(speedValue); lcd.print("% ");

  lcd.setCursor(11,1);
  bool dirState=(analogRead(DIR_SWITCH_PIN)>1000);
  lcd.print(dirState?"FC":"BAC");

  lcd.setCursor(12,0);
  if(isPlaying){lcd.print("Run"); digitalWrite(LED_PIN,HIGH);}
  else{lcd.print("Stop"); digitalWrite(LED_PIN,LOW);}
}

// ====== Setup ======
void setup(){
   Wire.begin();
   stopAll();
   Serial.begin(115200);

   pinMode(LED,OUTPUT);
   pinMode(PUSH1,INPUT);
   pinMode(PUSH2,INPUT);
   pinMode(BUTTON_PIN,INPUT);
   pinMode(LED_PIN,OUTPUT);
   pinMode(BUZZER_PIN,OUTPUT);
   pinMode(DIR_SWITCH_PIN,INPUT);
   pinMode(MIC_INT_PIN,INPUT);

   attachInterrupt(digitalPinToInterrupt(MIC_INT_PIN),clapISR,RISING);

   lastState=(digitalRead(PUSH2)<<1)|digitalRead(PUSH1);

   lcd.begin(16,2);
   updateScreen();

   MsTimer2::set(controlInterval, controlISR); // 5ms → 200Hz
   MsTimer2::start();
}

// ====== Loop ======
void loop(){
   // 1. MsTimer2 control update
   if(updateFlag){
      updateFlag = false;
      updateMove();
   }
```

```
// 2. Update buzzer
updateBuzzer();

// 3. Clap-triggered Task B
bool dirForward = (analogRead(DIR_SWITCH_PIN)>1000);
if(triggered && !isPlaying && !locked && currentTask=='B'){
  triggered=false;
  locked=true;
  lockStartTime=millis();
  isPlaying=true;
  updateScreen();

  digitalWrite(LED,HIGH);
  playBuzzer(1000,200); delay(250);
  playBuzzer(800,200); delay(250);

  startMove(dirForward,speedValue,metersValue);
  while(moving){
    if(updateFlag){ updateFlag=false; updateMove(); }
    updateBuzzer();
  }

  playBuzzer(600,400); delay(500);
  playBuzzer(1200,400); delay(500);
  digitalWrite(LED,LOW);

  isPlaying=false;
  updateScreen();
}

// 4. Automatic unlock
if(locked && millis()-lockStartTime>=lockDuration) locked=false;

// 5. Button for menu/task selection
static unsigned long pressStart=0;
if(digitalRead(BUTTON_PIN)==LOW){
  pressStart = millis();
  while(digitalRead(BUTTON_PIN)==LOW); // Wait release
  unsigned long t = millis()-pressStart;

  if(t < 500){ // Short press → toggle menu
    menuMode = (menuMode==MODE_SPEED)?MODE_TASK_PARAM:MODE_SPEED;
  } else {   // Long press → toggle task
    currentTask = (currentTask=='A')?'B':'A';
  }
  updateScreen();
}
```

```
    // 6. Rotary encoder
    uint8_t cur=(digitalRead(PUSH2)<<1)|digitalRead(PUSH1);
    if(cur!=lastState){
      if ((lastState==0b00 && cur==0b01) ||
        (lastState==0b01 && cur==0b11) ||
        (lastState==0b11 && cur==0b10) ||
        (lastState==0b10 && cur==0b00)) step++;
      else step--;
      lastState=cur;

      if(cur==0b00){
        if(menuMode==MODE_SPEED){
          speedValue=constrain(speedValue+(step>0?1:-1),1,100);
        }else{
          if(currentTask=='A') cyclesValue=constrain(cyclesValue+(step>0?1:-1),1,20);
          else metersValue=constrain(metersValue+(step>0?0.1:-0.1),1.0,10.0);
        }
        step=0;
        if(millis()-lcdLastUpdate>=100){
          updateScreen();
          lcdLastUpdate=millis();
        }
      }
    }
  }
```

## 5.8 A.8 low voltage indicator

```
// Low-battery indicator for 2-cell pack

// R1 = 91 kΩ (to +10 V), R2 = 10 kΩ (to GND), C = 4.7 µF across R2

// Divider output -> Arduino pin A1

// LED1: low / medium status

// LED2: high (full) status



const byte LED1_PIN = 2;    // 根据实际接线修改

const byte LED2_PIN = 3;    // 根据实际接线修改

const byte SENSE_PIN = A1;  // 分压输出接到 A1
```

```
// 5 V 参考电压（可以用万用表量一下再改得更准确）

const float VREF = 5.0;


// 分压比 k = R2 / (R1 + R2)； R1 = 91k, R2 = 10k

const float R1 = 91000.0;

const float R2 = 10000.0;

const float DIV_RATIO = R2 / (R1 + R2);


// 你实验里测到的电池阈值（单位：V）

const float VTH_1 = 4.4;  // 低电量：闪烁 -> 单灯常亮

const float VTH_2 = 7.7;  // 中电量：单灯常亮 -> 双灯常亮


// 闪烁相关变量

const unsigned long BLINK_PERIOD_MS = 500; // 闪烁周期 0.5 s

unsigned long lastBlinkMs = 0;

bool led1BlinkState = false;


void setup() {

  pinMode(LED1_PIN, OUTPUT);

  pinMode(LED2_PIN, OUTPUT);

  pinMode(SENSE_PIN, INPUT);
  Serial.begin(9600);  // 串口监视器里可以看到电压和 ADC 值
}

void loop() {
  // 1. 读取 ADC，并换算为分压点电压 Vout 和电池电压 Vbat
  int adc = analogRead(SENSE_PIN);
  float vOut = adc * VREF / 1023.0;   // 分压输出电压
  float vBat = vOut / DIV_RATIO;      // 电池端估算电压

  // 打印到 Serial Monitor 方便你检查/标定
  Serial.print("ADC = ");
  Serial.print(adc);
  Serial.print("  Vout = ");
  Serial.print(vOut, 3);
  Serial.print(" V  Vbat = ");
  Serial.print(vBat, 3);
  Serial.println(" V");

  // 2. 根据 Vbat 决定 LED 状态
  if (vBat < VTH_1) {
    // 非常低：LED1 闪烁，LED2 熄灭
    unsigned long now = millis();

    if (now - lastBlinkMs >= BLINK_PERIOD_MS) {

      lastBlinkMs = now;

      led1BlinkState = !led1BlinkState;

      digitalWrite(LED1_PIN, led1BlinkState ? HIGH : LOW);

    }

    digitalWrite(LED2_PIN, LOW);


  } else if (vBat < VTH_2) {

    // 中等电量：LED1 常亮，LED2 熄灭

    digitalWrite(LED1_PIN, HIGH);

    digitalWrite(LED2_PIN, LOW);


  } else {

    // 高电量：两盏灯都常亮

    digitalWrite(LED1_PIN, HIGH);

    digitalWrite(LED2_PIN, HIGH);

  }


  // 采样间隔，大一点无所谓，电池变化很慢

  delay(100);

}
```

## 5.9 HMI PCB components and vehicle operation

```
#include <LiquidCrystal.h>
#include <Wire.h>
#include <MsTimer2.h>

// ====== Pins ======
#define MIC_INT_PIN 2
#define LED_PIN 13
#define BUZZER_PIN 8
#define DIR_SWITCH_PIN A7
#define LED A1
#define BUTTON_PIN 9
#define PUSH1 3
#define PUSH2 10

const int rs=12,en=11,d4=4,d5=5,d6=6,d7=7;
LiquidCrystal lcd(rs,en,d4,d5,d6,d7);

// ====== MsTimer2 RT setup ======
volatile bool updateFlag = false;
const unsigned long controlInterval = 5; // 5ms → 200Hz

// ====== Small car parameters ======
int steerForward = 5;    // Forward steering compensation
int steerBackward = -5;  // Backward steering compensation
float distanceCompensation = 1.33; // Distance correction factor
int sp = 50;             // Default speed

// ====== I2C motor constants ======
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';

// ====== Movement control ======
bool moving = false;
bool forward = true;
float targetMeters = 0;
unsigned long moveStartTime = 0;
unsigned long moveDuration = 0;

// ====== Variables ======
volatile int step=0;
uint8_t lastState;
int speedValue=50;
float metersValue=2.0;
```

```
int cyclesValue=3;
char currentTask='B';   // Default TaskB
bool isPlaying=false;
volatile bool triggered=false;
bool locked=false;
unsigned long lockStartTime=0;
const unsigned long lockDuration=6000;

enum MenuMode { MODE_SPEED, MODE_TASK_PARAM };
MenuMode menuMode = MODE_SPEED;
unsigned long lcdLastUpdate=0;

// ====== Motor functions ======
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 stopAll(){
  Wire.beginTransmission(MOTOR_ADDR);
  Wire.write(CMD_HALT_ALL);
  Wire.write(TARGET_ALL);
  Wire.endTransmission();
}

// ====== Movement control using non-blocking & MsTimer2 ======
void startMove(bool fwd, int speed, float meters){
    forward = fwd;
    sp = speed;
    targetMeters = meters;
    moveDuration = meters * distanceCompensation * 1000;
    moveStartTime = millis();
    moving = true;
}

void updateMove(){
    if(!moving) return;

    int offset = forward ? steerForward : steerBackward;
    int m1, m2, m3, m4;

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

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

    driveAll(m1, m2, m3, m4);

    if(millis() - moveStartTime >= moveDuration){
        stopAll();
        moving = false;
    }
}

// ====== Timer ISR ======
void controlISR(){
    updateFlag = true;
}

// ====== Clap detection ======
void clapISR(){
 if(!locked) triggered=true;
}

// ====== Buzzer ======
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;
  }
}

// ====== LCD ======
void updateScreen(){
 lcd.clear();
 lcd.setCursor(0,0);
 lcd.print("Task: "); lcd.print(currentTask);
 lcd.setCursor(7,0);
 if(currentTask=='A') lcd.print("C:"); else lcd.print("M:");
 lcd.print((currentTask=='A')?cyclesValue:metersValue,1);

 lcd.setCursor(0,1);
 lcd.print("Speed:"); lcd.print(speedValue); lcd.print("% ");

 lcd.setCursor(11,1);
 bool dirState=(analogRead(DIR_SWITCH_PIN)>1000);
 lcd.print(dirState?"FC":"BAC");

 lcd.setCursor(12,0);
 if(isPlaying){lcd.print("Run"); digitalWrite(LED_PIN,HIGH);}
 else{lcd.print("Stop"); digitalWrite(LED_PIN,LOW);}
}

// ====== Setup ======
void setup(){
  Wire.begin();
  stopAll();
  Serial.begin(115200);

  pinMode(LED,OUTPUT);
  pinMode(PUSH1,INPUT);
  pinMode(PUSH2,INPUT);
  pinMode(BUTTON_PIN,INPUT);
  pinMode(LED_PIN,OUTPUT);
  pinMode(BUZZER_PIN,OUTPUT);
  pinMode(DIR_SWITCH_PIN,INPUT);
  pinMode(MIC_INT_PIN,INPUT);

  attachInterrupt(digitalPinToInterrupt(MIC_INT_PIN),clapISR,RISING);

  lastState=(digitalRead(PUSH2)<<1)|digitalRead(PUSH1);

  lcd.begin(16,2);
  updateScreen();

  MsTimer2::set(controlInterval, controlISR); // 5ms → 200Hz
  MsTimer2::start();
}

// ====== Loop ======
void loop(){
  // 1. MsTimer2 control update
  if(updateFlag){
    updateFlag = false;
    updateMove();
  }

  // 2. Update buzzer
  updateBuzzer();

  // 3. Clap-triggered Task B
  bool dirForward = (analogRead(DIR_SWITCH_PIN)>1000);
  if(triggered && !isPlaying && !locked && currentTask=='B'){
    triggered=false;
    locked=true;
    lockStartTime=millis();
    isPlaying=true;
    updateScreen();

    digitalWrite(LED,HIGH);
    playBuzzer(1000,200); delay(250);
    playBuzzer(800,200); delay(250);

    startMove(dirForward,speedValue,metersValue);
    while(moving){
        if(updateFlag){ updateFlag=false; updateMove(); }
        updateBuzzer();
    }

    playBuzzer(600,400); delay(500);
    playBuzzer(1200,400); delay(500);
    digitalWrite(LED,LOW);

    isPlaying=false;
    updateScreen();
}

// 4. Automatic unlock
if(locked && millis()-lockStartTime>=lockDuration) locked=false;

// 5. Button for menu/task selection
static unsigned long pressStart=0;
if(digitalRead(BUTTON_PIN)==LOW){
    pressStart = millis();
    while(digitalRead(BUTTON_PIN)==LOW); // Wait release
    unsigned long t = millis()-pressStart;

    if(t < 500){ // Short press → toggle menu
        menuMode = (menuMode==MODE_SPEED)?MODE_TASK_PARAM:MODE_SPEED;
    } else {    // Long press → toggle task
        currentTask = (currentTask=='A')?'B':'A';
    }
    updateScreen();
}

// 6. Rotary encoder
uint8_t cur=(digitalRead(PUSH2)<<1)|digitalRead(PUSH1);
if(cur!=lastState){
    if ((lastState==0b00 && cur==0b01) ||
        (lastState==0b01 && cur==0b11) ||
        (lastState==0b11 && cur==0b10) ||
        (lastState==0b10 && cur==0b00)) step++;
    else step--;
    lastState=cur;

    if(cur==0b00){
        if(menuMode==MODE_SPEED){
            speedValue=constrain(speedValue+(step>0?1:-1),1,100);
        }else{
            if(currentTask=='A') cyclesValue=constrain(cyclesValue+(step>0?1:-1),1,20);
            else metersValue=constrain(metersValue+(step>0?0.1:-0.1),1.0,10.0);
        }
        step=0;
```

```
          if(millis()-lcdLastUpdate>=100){
            updateScreen();
            lcdLastUpdate=millis();
          }
        }
      }
    }
```

## 5.10 NoIRQ

```
#include <MsTimer2.h>
#include <math.h>
#define FREQ_CTL 1
#define LED_blink (13)
#define debug_IO (10)

static unsigned int counter_sec = 0, Ns=0, buttonState;
static float Ffund=1,Ffund_buff=1;
unsigned long previousTime = micros(); // or millis()
unsigned long previousTime1 = micros(); // or millis()
long timeInterval = 1000,count_noise=0;
long timeInterval1 = 2500;
int flag_noise;

void setup()
{
  noInterrupts();
  Serial.begin(19200);   // connect to PC
  pinMode(LED_blink, OUTPUT);
  pinMode(debug_IO, INPUT_PULLUP);
  interrupts();
  delay(1000);
}

void loop()   // main loop i.e. while(1)
{

  unsigned long currentTime = micros(); // or millis()
  // Enter the If block only if at least 800 micros (or millis) has passed since last time
  buttonState = digitalRead(debug_IO);

  // check if the pushbutton is pressed. If it is, the buttonState is HIGH:
  if (buttonState == LOW) {

    if ((currentTime - previousTime1 > timeInterval1)) {
      // do action
      if (count_noise > 100) {
        count_noise = 0;
        flag_noise = 1 - flag_noise;
        Serial.println(flag_noise + 10);
      }
      count_noise++;
      previousTime1 = currentTime;
      if (flag_noise == 1) {
        Serial.println(count_noise);
        delay(2);
      }
    }

  }
  if (currentTime - previousTime > timeInterval) {
    // do action
    Timer2ISR();
    previousTime = currentTime;
  }

}

void Timer2ISR()   // Timer2 interrupt service routine (ISR)
{
  unsigned int Duty;
  Duty=400;
  digitalWrite(LED_blink, HIGH);
  delayMicroseconds(Duty);
  digitalWrite(LED_blink, LOW);

}
```

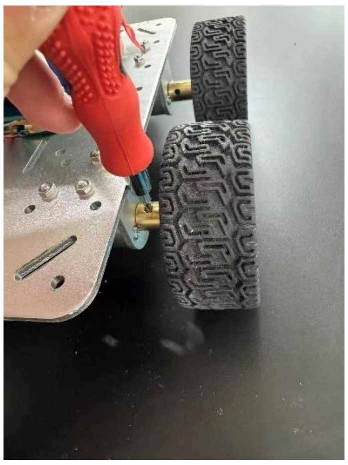
## 5.11 WithIRQ

```
#include <MsTimer2.h>

// ====== user settings ======
const int pwmPin = 3;          // D3
const unsigned int switchMs = 10;  // 10 ms -> 100 Hz interrupt
const unsigned int periodTicks = 50; // 500 ms / 10 ms = 50 ticks -> 2 Hz PWM

// set this 0~50 to change duty cycle
volatile unsigned int dutyTicks = 25; // 25/50 = 50% duty

volatile unsigned int tickCount = 0;
```

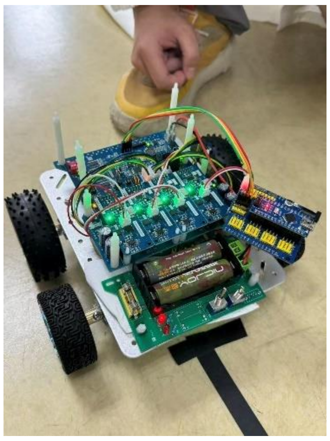
```
void pwmISR() {
  // simple software PWM
  if (tickCount < dutyTicks) {
    digitalWrite(pwmPin, HIGH);
  } else {
    digitalWrite(pwmPin, LOW);
  }

  tickCount++;
  if (tickCount >= periodTicks) {
    tickCount = 0;  // start next 2 Hz period
  }
}

void setup() {
  pinMode(pwmPin, OUTPUT);
  digitalWrite(pwmPin, LOW);

  // set timer interrupt every 10 ms -> 100 Hz
  MsTimer2::set(switchMs, pwmISR);
  MsTimer2::start();

  // optional: show we're alive
  Serial.begin(9600);
  Serial.println("Software PWM started: 2 Hz period, 100 Hz switching.");
}

void loop() {
  // if you want to change duty in runtime, do it here
  // e.g. slowly sweep:
  /*
  static unsigned long lastChange = 0;
  if (millis() - lastChange > 1000) { // every 1s
    lastChange = millis();
    noInterrupts();
    dutyTicks += 5;
    if (dutyTicks > periodTicks) dutyTicks = 0;
    interrupts();
  }
  */
}
```

# Chapter 6 Appendix B
# Photographs and physical build

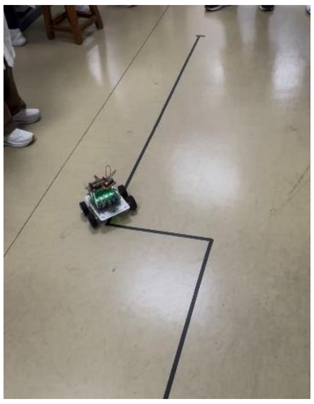
## 6.1 B.1 Assembled four-wheel vehicle (side view)



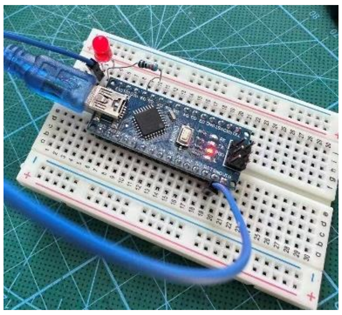
## 6.2 B.2 Vehicle wiring and PCB layout (top view)



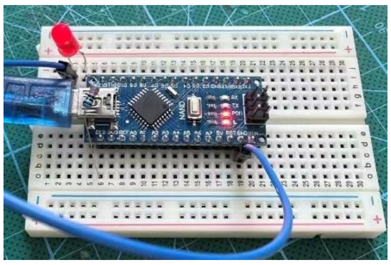
## 6.3 B.3 Vehicle during baseline run



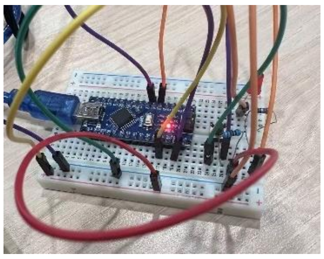
## 6.4 B.4 Blink example circuit on breadboard



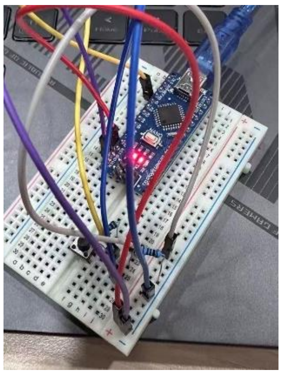
## 6.5 B.5 LED fading (PWM) example circuit



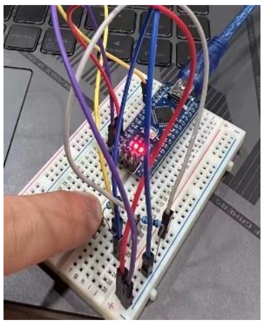
## 6.6 B.6 Push-button example circuit



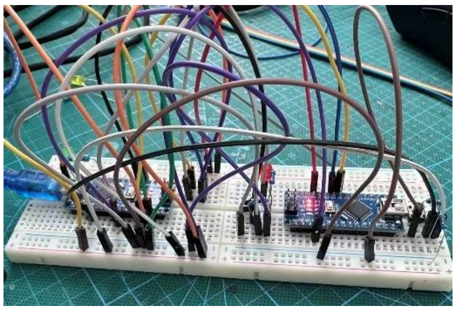
## 6.7 B.7 Read analog voltage example circuit



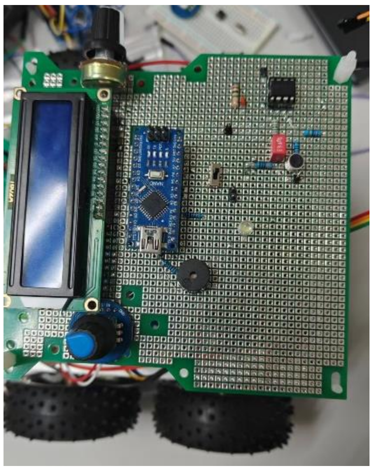
## 6.8 B.8 LED switching ON using computer (serial control)



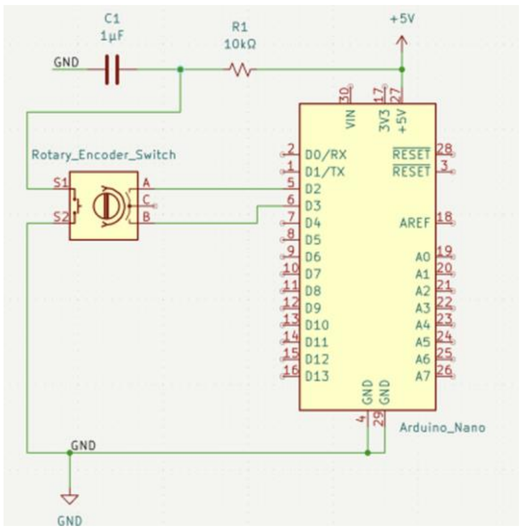
## 6.9 B.9 I²C communication between two Arduinos



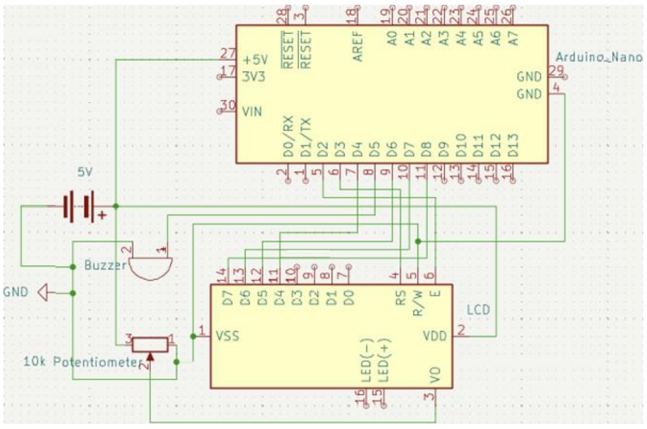
## 6.10 HMI PCB (installed on car)



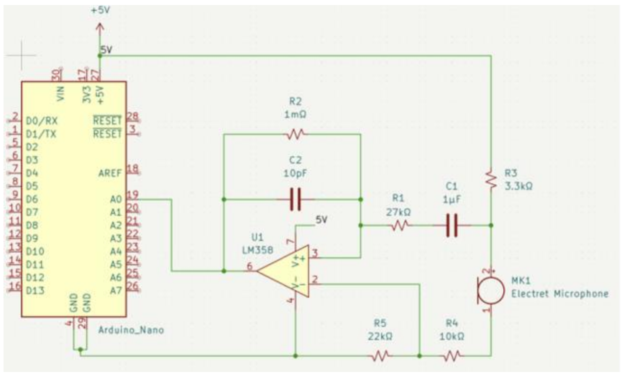
## 6.11 B.11 Rotary encoder with push-button



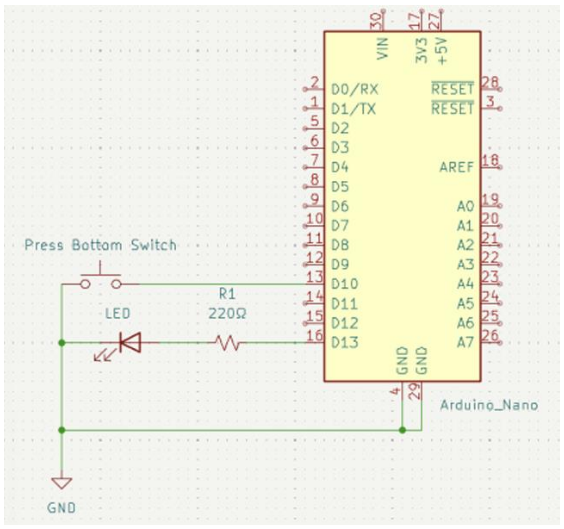
## 6.12 B.12 I²C LCD display with Buzzer connection



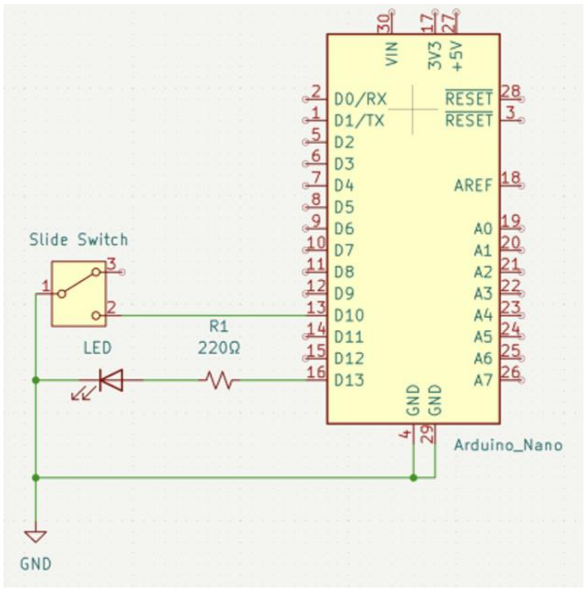
## 6.13 B.13 Microphone input circuit



## 6.14 B.14 Buttons



## 6.15 B.15 Slide switch



## 6.16 B.16 Figure 5

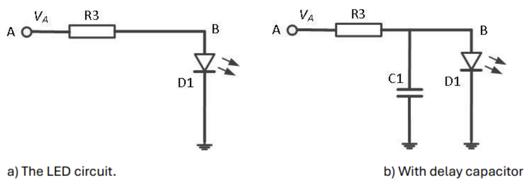
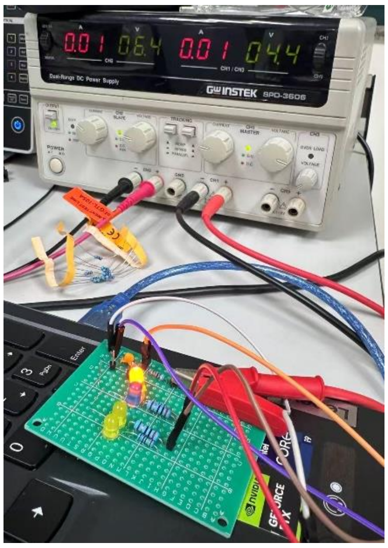
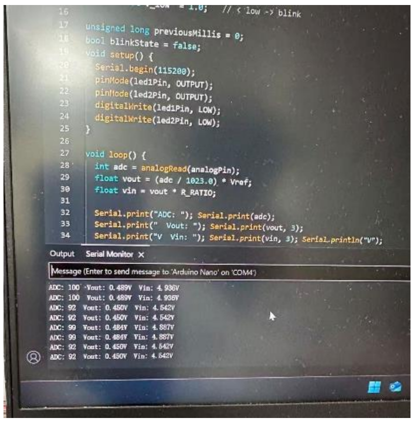


Figure 5 - The LED circuits: with and without capacitor

## 6.17 B.17 threshold of 10%



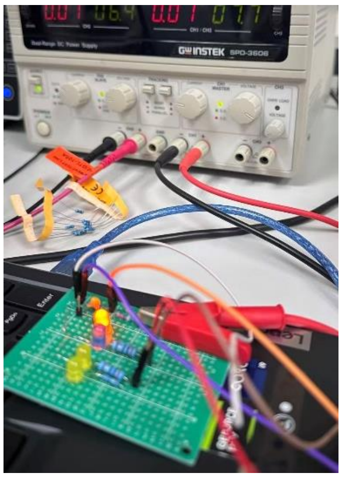
## 6.18 Screenshot of 10% treshold (4.4v)



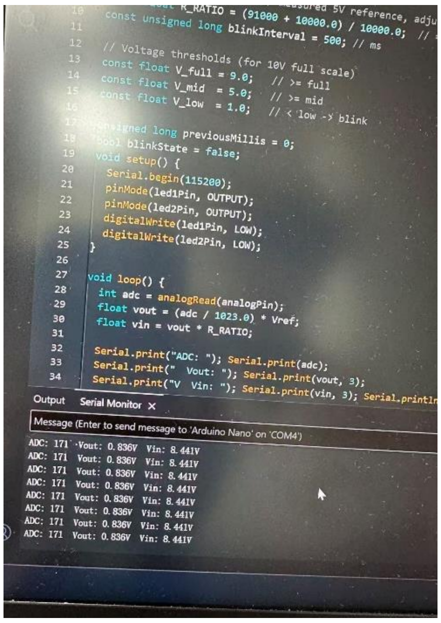
```
                     = 1.0;   // < low -> blink

unsigned long previousMillis = 0;
bool blinkState = false;
void setup() {
  Serial.begin(115200);
  pinMode(led1Pin, OUTPUT);
  pinMode(led2Pin, OUTPUT);
  digitalWrite(led1Pin, LOW);
  digitalWrite(led2Pin, LOW);
}

void loop() {
  int adc = analogRead(analogPin);
  float vout = (adc / 1023.0) * Vref;
  float vin = vout * R_RATIO;

  Serial.print("ADC: "); Serial.print(adc);
  Serial.print("  Vout: "); Serial.print(vout, 3);
  Serial.print("V  Vin: "); Serial.print(vin, 3); Serial.println("V");
```

```
ADC: 100  Vout: 0.489V  Vin: 4.936V
ADC: 100  Vout: 0.489V  Vin: 4.936V
ADC: 92  Vout: 0.450V  Vin: 4.542V
ADC: 92  Vout: 0.450V  Vin: 4.542V
ADC: 99  Vout: 0.484V  Vin: 4.887V
ADC: 99  Vout: 0.484V  Vin: 4.887V
ADC: 92  Vout: 0.450V  Vin: 4.542V
ADC: 92  Vout: 0.450V  Vin: 4.542V
```

## 6.19 B.17 threshold of 50%



## 6.20 Screenshot of 10% treshold (4.4v)



```
      R_RATIO = (91000 + 10000.0) / 10000.0;  //
 const unsigned long blinkInterval = 500; // ms

 // Voltage thresholds (for 10V full scale)
 const float V_full = 9.0;   // >= full
 const float V_mid  = 5.0;   // >= mid
 const float V_low  = 1.0;   // < low -> blink

unsigned long previousMillis = 0;
bool blinkState = false;
void setup() {
  Serial.begin(115200);
  pinMode(led1Pin, OUTPUT);
  pinMode(led2Pin, OUTPUT);
  digitalWrite(led1Pin, LOW);
  digitalWrite(led2Pin, LOW);
}

void loop() {
  int adc = analogRead(analogPin);
  float vout = (adc / 1023.0) * Vref;
  float vin = vout * R_RATIO;

  Serial.print("ADC: "); Serial.print(adc);
  Serial.print("  Vout: "); Serial.print(vout, 3);
  Serial.print("V  Vin: "); Serial.print(vin, 3); Serial.printl
```

```
ADC: 171  Vout: 0.836V  Vin: 8.441V
ADC: 171  Vout: 0.836V  Vin: 8.441V
ADC: 171  Vout: 0.836V  Vin: 8.441V
ADC: 171  Vout: 0.836V  Vin: 8.441V
ADC: 171  Vout: 0.836V  Vin: 8.441V
ADC: 171  Vout: 0.836V  Vin: 8.441V
ADC: 171  Vout: 0.836V  Vin: 8.441V
ADC: 171  Vout: 0.836V  Vin: 8.441V
```