Serial Commander

High Level Overview

Purpose

The Serial Commander is a terminal user interface (TUI) for sending commands to an STM32 CAN board over a serial connection. It provides a live split-screen view of outgoing commands and incoming serial data, with editable motor and PIO parameters.

Prerequisites

The STM32 firmware must be built with serial command reception enabled. The firmware must parse newline-terminated ASCII strings from UART/USB-CDC and handle the following command tokens:

- `speed <erpm>`
- `stop`
- `profile <erpm> <accel> <time_ms>`

Without this firmware support the TUI will open and display incoming data, but sent commands will have no effect aside form rebuilding and flashing the firmware

Core Responsibilities

The tool handles three concerns in one interface:

- **Command dispatch** — sends formatted serial strings (`speed`, `stop`, `profile`) to the STM32 on keypress
- **Live serial monitoring** — streams and colour-codes all incoming serial output in real time
- **PIO integration** — triggers PlatformIO build/flash operations without leaving the terminal

Architecture

Three concurrent components run during a session:

Main TUI Loop — Draws the interface at 100 ms intervals, handles keypresses, and dispatches commands or parameter edits. Runs on the main thread via `curses.wrapper`.

Serial Reader Thread — Daemon thread that continuously calls `ser.readline()` and appends decoded lines to a shared `rx_log` deque. Terminates automatically when the serial connection drops.

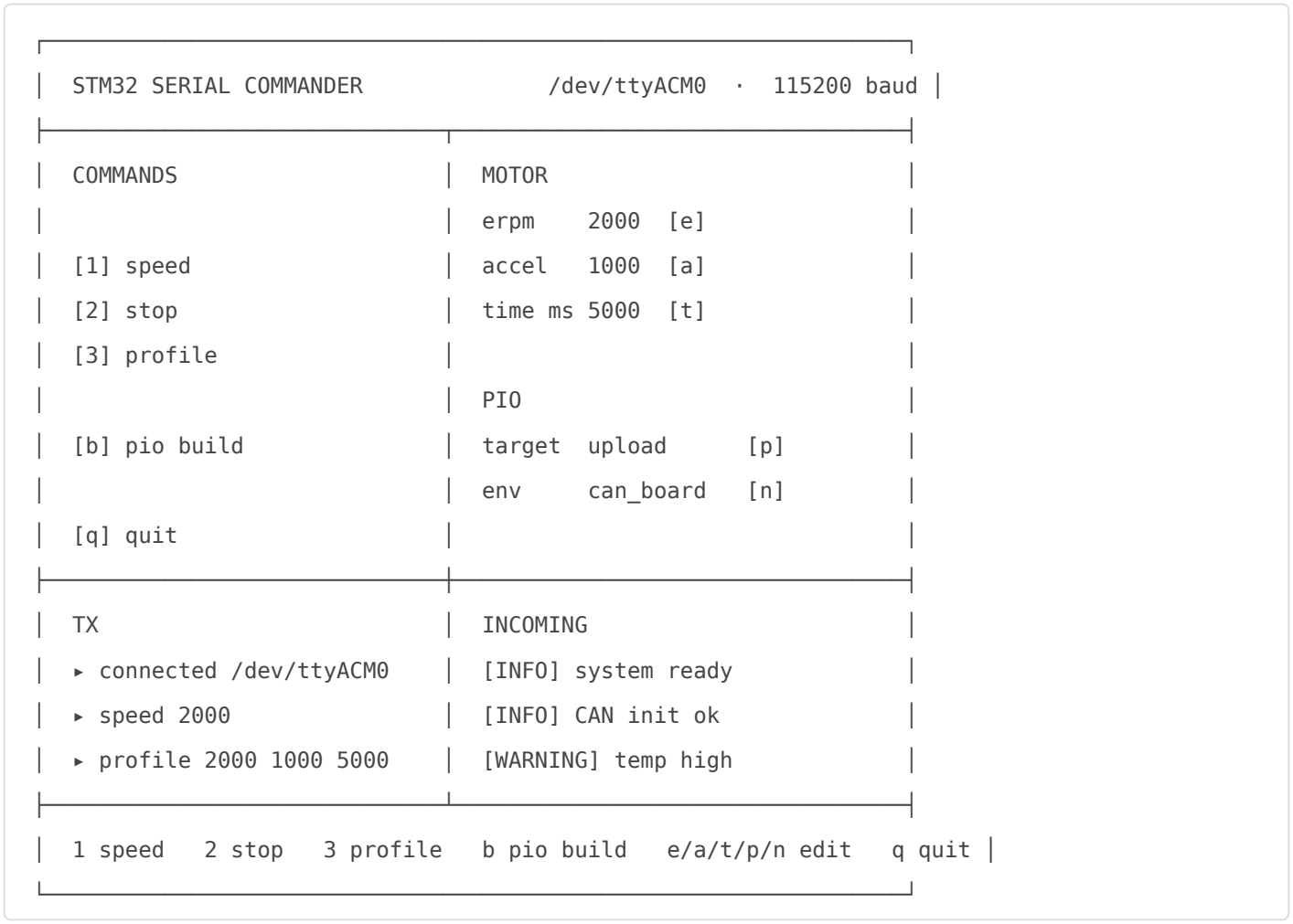
PIO Runner Thread — Spawned on demand when the user triggers a build. Runs `pio run -t <target> -e <env>` as a subprocess and streams stdout into `rx_log`. Does not block the TUI.

All three components share `rx_log` and `tx_log` as `collections.deque(maxlen=200)` — thread-safe for the single-producer/single-consumer append and iteration patterns used here.

Interface Layout

Layout

The TUI divides the terminal into four regions separated by line-drawing characters.



Regions

Top-left — COMMANDS: Available key bindings. `[q]` is visually separated from the command group.

Top-right — MOTOR / PIO: Editable parameters with current values. Each shows its edit key in brackets. Values highlight in yellow when active.

Bottom-left — TX: Scrolling log of sent commands, prefixed with `▸`. Most recent entries appear at the bottom.

Bottom-right — INCOMING: Scrolling serial output, colour-coded by severity:

Prefix	Colour
`[ERROR]`	Red
`[WARNING]` / `[WARN]`	Yellow
`[INFO]`	Cyan
(other)	White

Status bar: Shows edit mode prompt when a field is being edited; otherwise shows the full key reference.

Terminal Requirements

Minimum ~80×24 terminal. Colour support recommended — falls back to `curses.COLOR_MAGENTA` if the terminal cannot redefine colours (the TUI uses a custom purple `#5a2273`).

Commands and Keybindings

Serial Commands

Key	Command sent	Description
`1`	`speed <erpm>`	Sets motor speed to current erpm value
`2`	`stop`	Sends stop command
`3`	`profile <erpm> <max_accel> <time_ms>`	Sends full motion profile
`b`	*(PIO subprocess)*	Runs `pio run -t <target> -e <env>` in background
`q`	*(none)*	Quits the TUI and closes the serial port

Parameter Edit Keys

Key	Field	Validation
`e`	erpm	Must be integer (negative allowed)
`a`	accel	Must be integer (negative allowed)
`t`	time_ms	Must be integer (negative allowed)
`p`	pio target	Any non-empty string
`n`	pio env	Any non-empty string

Edit Mode Behaviour

Pressing an edit key switches the status bar to an edit prompt and opens an inline text field on the parameter row. **Enter** confirms; **ESC** restores the previous value. Invalid integer inputs are rejected and logged to the TX log with a `!!` prefix — the parameter retains its prior value.

PIO Build

Pressing `b` spawns a background thread running:

```
pio run -t <pio_target> -e <pio_env>
```

Output streams line-by-line into the INCOMING panel. `[INFO] PIO: done (exit 0)` appears on success. `[ERROR] PIO: 'pio' not found in PATH` appears if PlatformIO is not installed.

--- Page 4 — Usage

Output streams line-by-line into the INCOMING panel. `[INFO] PIO: done (exit 0)` appears on success. `[ERROR] PIO: 'pio' not found in PATH` appears if PlatformIO is not installed.

Usage

Dependencies

```
pip install pyserial
```

PlatformIO (`pio`) must be in `PATH` for build/flash functionality.

Running

Auto-detect STM32 device (VID `0x0483`):

```
python3 serial_commander.py
```

Specify port and baud rate manually:

```
python3 serial_commander.py --port /dev/ttyACM0 --baud 115200
```

If multiple STM32 devices are connected, the script presents a numbered selection menu before opening the TUI.

Default Parameter Values

Parameter	Default
erpm	2000
accel	1000
time_ms	5000
pio target	upload
pio env	can_board

Typical Workflow

1. Connect STM32 CAN board via USB.
2. Run `python3 serial_commander.py` — port is detected automatically.

3. Adjust `erpm`, `accel`, `time_ms` with `e`/`a`/`t` as needed.
4. Send `speed` (`1`) or `profile` (`3`) and observe STM32 response in INCOMING.
5. Use `b` to build/flash directly if firmware changes are needed.
6. Press `q` to quit — serial port closes cleanly.